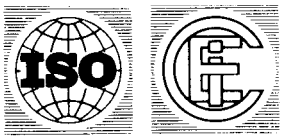
INTERNATIONAL STANDARD

ISO/IEC 6522

Second edition
1992-11-15

Information technology — Programming languages — PL/I general purpose subset

Technologies de l'information — Langages de programmation — Sous-ensemble PL/I pour usage général



Reference number
ISO/IEC 6522:1992(E)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Contents

1. Scope and Overviews **1**
1.1 Scope 1
1.2 Referenced Publications 1
1.3 Goals and Rationale 2
1.4 An Informal Guide to the PL/I Definition 3
1.4.1 A Summary of PL/I 3
1.4.2 The Form of the Definition 4
1.4.3 Summary of Chapter Structure 6
1.4.4 Introduction to the Metalanguage 10
1.4.4.1 Tree Concepts 10
1.4.4.2 Syntaxes 11
1.4.4.3 Algorithm Concepts 12
1.5 Relationships between an Implementation and this Definition 14
1.5.1 Flexibilities of Interpretation 15
1.5.1.1 Rejection of Programs 15
1.5.1.2 Quantitative Restrictions 15
1.5.1.3 Addressing Restrictions 15
1.5.1.4 Operating Environment 16
1.5.1.5 Character Sets 16
1.5.1.6 Expression Evaluation 16
1.5.1.7 Interrupts and Assignment 17
1.5.1.8 Input/Output 17
1.5.1.9 On-units 17
1.5.2 Implementation-defined Features 18
1.6 The Metalanguage 20
1.6.1 Trees 21
1.6.1.1 Tree Definitions 21
1.6.1.2 Node Objects 22
1.6.1.2.1 Unique-names 23
1.6.1.2.2 Types 23
1.6.1.3 Node Notation 23
1.6.1.4 Tree Notation 24
1.6.1.4.1 Enumerated-Trees 24
1.6.1.4.2 Forms 25
1.6.1.5 Tree Copies 25
1.6.2 Production Rules 25
1.6.2.1 Production Rules and Syntaxes 25
1.6.2.2 Complete and Partial Trees 26
1.6.2.3 Syntactic-expressions and Syntactic-units 28
1.6.2.4 Use of the Production Rules 28
1.6.2.4.1 Subnode-type Sequence with respect to a Syntax and Category-name 29
1.6.2.4.2 Subnode-type Sequence with respect to a Syntactic-expression 29
1.6.3 Operations 30
1.6.3.1 Nature of an Operation 30

© ISO/IEC 1992
All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

International Organization for Standardization
Case postale 56 • CH-1211 Genève 20 • Switzerland
Printed in Switzerland

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.6.3.2 Nondeterministic Operations 31
1.6.3.3 Format of an Operation 31
1.6.3.4 Instructions 32
1.6.3.5 Convert 34
1.6.3.6 Additional Notational Conventions 34
1.6.3.7 Arithmetic 34
1.6.3.7.1 Properties of Arithmetic in the Metalanguage 34
1.6.3.7.2 Properties of Arithmetic in PL/I 35
1.6.4 The Processor 36
1.6.5 Mechanization of the Metalanguage 37
1.7 Initialization of the Machine-state 39
1.7.1 The Machine-state 39
1.7.2 Initialization 39
1.7.3 The Top-level Operations 39
1.7.3.1 Define-program 39
1.7.3.2 Translation-phase 40
1.7.3.3 Interpretation-phase 40

2. Concrete Syntax 41
2.1 Introduction 41
2.2 The Intent of this Definition 41
2.2.1 Concrete and Abstract Syntaxes 41
2.3 Organization of the Concrete Syntax 41
2.4 The High-level Syntax of PL/I 42
2.4.1 Translation Unit 42
2.4.2 Package 42
2.4.3 Procedure 42
2.4.4 Unit 42
2.4.5 Executable Units 42
2.4.6 If Statement 43
2.4.7 Do Group 43
2.4.8 Select Group 43
2.4.9 Begin Block 43
2.4.10 On Unit 44
2.4.10.1 Condition Prefixes 44
2.4.10.2 Statement Name Prefixes 44
2.4.11 Data Declaration 44
2.4.11.1 Dimension Attribute and Dimension Suffix 44
2.4.11.2 Attributes 45
2.4.11.3 Data Attributes 45
2.4.11.4 Environment and Options 46
2.4.11.5 Initial 46
2.4.12 The Package Statement 46
2.4.13 The Procedure Statement 46
2.4.14 The Begin Statement 47
2.4.15 The Do Statement 47
2.4.16 The Select Statement 47
2.4.17 The End Statement 47
2.4.18 Flow of Control Statements 48
2.4.18.1 The Call and Return Statements 48
2.4.18.2 The Go To Statement 48
2.4.18.3 The Leave Statement 48
2.4.18.4 The Null Statement 48
2.4.18.5 The Revert and Signal Statements 48

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.4.18.6 The Stop Statement 48
2.4.19 Storage Control Statements 48
2.4.20 Input/Output Statements 49
2.4.20.1 The Open and Close Statements 49
2.4.20.2 Record I/O 49
2.4.20.3 Stream I/O 50
2.4.20.3.1 Stream Input Specification 50
2.4.20.3.2 Stream Output Specification 50
2.4.20.3.3 Format Specification Lists and the Format Statement 50
2.4.21 Expressions 51
2.4.22 Enquiry Functions 52
2.5 The Low-level Syntax of PL/I 57
2.5.1 PL/I Text 57
2.5.2 Comment 57
2.5.3 Identifier 57
2.5.4 Arithmetic Constant 57
2.5.5 String Constants and Pictures 58
2.5.6 Include 58
2.6 Character Sets 59
2.6.1 Language Character Set 60
2.6.1.1 Letters and Digits 60
2.6.1.2 Special Characters 61
2.6.2 Data Character Set 61
2.6.3 Secondary Case Characters in Numeric and Bit Values 61
2.7 Abbreviations 62

3. Abstract Syntax 63
3.1 Introduction 63
3.2 Abstract Syntax Rules 63
3.2.1 Program 63
3.2.2 Package 63
3.2.3 Procedure 63
3.2.4 Declaration 64
3.2.5 Variable 64
3.2.6 Data-description 64
3.2.7 Data-type 65
3.2.8 Named-constant 66
3.2.9 Entry-point 66
3.2.10 Executable-unit 66
3.2.11 Begin-block 67
3.2.12 Groups 67
3.2.13 Selections 67
3.2.14 On Statement 68
3.2.15 If Statement 68
3.2.16 Flow of Control Statements 68
3.2.17 Storage Statements 69
3.2.18 I/O Statements 69
3.2.19 Record I/O Statements 69
3.2.20 Stream I/O Statements 70
3.2.21 Expression 71
3.2.22 Types of Reference 72
3.2.23 Constant 73
3.2.24 Types of Value 73
3.2.25 Types of Picture 73

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4. The Translator **75**
4.1 Introduction 75
4.2 Translate 75
4.3 Forming the Translation Unit 76
4.3.1 Low-level-parse 77
4.3.2 High-level-parse 78
4.4 Validation of the Translation Unit 80
4.5 Completion of the External Unit 80
4.5.1 Reorganize 81
4.5.1.1 Complete-options 81
4.5.1.2 Complete-attribute-implications 81
4.5.1.3 Defactor-declarations 82
4.5.2 Construct-explicit-declarations 83
4.5.2.1 Declare-parameters 83
4.5.2.2 Declare-statement-names 84
4.5.2.3 Construct-statement-name-declarations 85
4.5.3 Complete-structure-declarations 87
4.5.3.1 Determine-structure 87
4.5.3.2 Convert-to-logical-levels 88
4.5.3.3 Propagate-alignment 88
4.5.3.4 Find-applicable-declaration 89
4.5.3.5 Find-fully-qualified-name 90
4.5.4 Construct-contextual-declarations 91
4.5.5 Complete-declarations 93
4.5.5.1 Test-attribute-consistency 94
4.5.5.2 Test-invalid-duplicates 96
4.5.5.3 Apply-defaults 96
4.5.5.4 Apply-attribute 98
4.5.5.5 Complete-constructed-entry-declaration 98
4.5.5.6 Test-name-in-description 101
4.5.6 Validate-concrete-declarations 101
4.5.6.1 Check-attribute-completeness-and-delete-attributes 102
4.6 Create-abstract-equivalent-tree 103
4.6.1 Creation of Blocks and Groups 104
4.6.1.1 Create-package 104
4.6.1.2 Create-procedure 105
4.6.1.3 Create-begin-block 106
4.6.1.4 Create-block 106
4.6.1.5 Replace-concrete-designators 107
4.6.1.6 Replace-concrete-precision-designators 107
4.6.1.7 Replace-concrete-named-constant-designators 108
4.6.1.8 Create-group 108
4.6.1.9 Create-selection 109
4.6.1.10 Create-executable-unit-list 110
4.6.1.11 Create-executable-unit 110
4.6.1.12 Create-entry-point 110
4.6.1.13 Create-statement-name 111
4.6.1.14 Create-condition-prefix-list 112
4.6.1.15 Create-condition 112
4.6.1.16 Test-descriptor-extent-expressions 113
4.6.2 Creation of Statements 113
4.6.2.1 Create-assignment-statement 113
4.6.2.2 Data-descriptions Proper for Assignment 114

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.2.3 Create-allocation ... 114
4.6.2.4 Create-format-statement ... 114
4.6.2.5 Create-remote-format ... 115
4.6.2.6 Create-format-iteration ... 115
4.6.2.7 Create-freeing ... 115
4.6.2.8 Create-if-statement ... 116
4.6.2.9 Create-balanced-unit ... 116
4.6.2.10 Create-leave-statement ... 117
4.6.2.11 Create-on-statement ... 118
4.6.2.12 Create-open-statement ... 119
4.6.3 Create-declaration ... 119
4.6.3.1 Create-named-constant ... 120
4.6.3.2 Create-variable ... 120
4.6.3.3 Create-bound-pair-list ... 121
4.6.3.4 Create-data-description ... 122
4.6.3.5 Create-data-type ... 124
4.6.3.6 Create-entry ... 125
4.6.3.7 Create-refer-option ... 126
4.6.3.8 Create-identifier ... 126
4.6.3.9 Create-initial ... 126
4.6.3.10 Create-array-init ... 127
4.6.3.11 Create-initial-element ... 127
4.6.3.12 Create-integer ... 128
4.6.3.13 Convert-named-literal ... 128
4.6.3.14 Check-arithmetic-type ... 130
4.6.3.15 Create-precision ... 130
4.6.3.16 Create-number-of-digits ... 130
4.6.3.17 Create-scale-factor ... 131
4.6.4 Create-expression ... 131
4.6.5 Create-reference ... 134
4.6.5.1 Collect-subscripts ... 137
4.6.5.2 Apply-subscripts ... 138
4.6.5.3 Create-value-reference ... 139
4.6.5.4 Trim-dd ... 139
4.6.5.5 Create-named-constant-reference ... 139
4.6.5.6 Create-argument-list ... 140
4.6.5.7 Create-builtin-function-reference ... 141
4.6.5.8 Create-pseudo-variable-reference ... 142
4.6.5.9 Create-entry-reference ... 143
4.6.5.10 Test-matching ... 144
4.6.6 Create-picture ... 144
4.6.6.1 Create-numeric-picture ... 145
4.6.7 Create-constant ... 147
4.6.8 Create-enquiry-reference ... 149
4.6.8.1 Create-arithmetic-enquiry-reference ... 151
4.7 Validation of the External Unit ... 152
4.7.1 Validate-declaration ... 152
4.7.2 Validate-automatic-declaration ... 152
4.7.3 Validate-based-declaration ... 153
4.7.4 Validate-defined-declaration ... 154
4.7.5 Validate-parameter-declaration ... 154
4.7.6 Validate-static-declaration ... 154
4.7.7 Validate-descriptor ... 155
4.7.8 Evaluate-restricted-expression ... 155
4.7.9 Apply-constraints ... 156

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.7.10 Test-constraints 157
4.7.11 Validate-leave-statements 158
4.8 Validate-program 159
4.8.1 Validate-external-declaration 159
4.8.2 Environment-names 160

5. The PL/I Interpreter 161
5.1 Introduction 161
5.2 The Interpretation-state 161
5.2.1 Directories 161
5.2.2 Block State 161
5.2.3 File Information 163
5.2.4 Storage and Values 164
5.2.5 Generations, Evaluated Data Descriptions, and Evaluated Targets 165
5.2.6 Dataset 165
5.3 Terminology and Definitions 166
5.3.1 Current 166
5.3.2 Block 166
5.4 The Interpret Operation and Interpretation State Initialization 166
5.4.1 Interpret 166
5.4.2 Initialize-interpretation-state 167
5.4.3 Build-file-directory-and-informations 167
5.4.4 Build-fdi 168
5.4.5 Allocate-static-storage-and-build-static-directory 168
5.4.6 Program-epilogue 169

6. Flow Of Control 171
6.1 Introduction 171
6.2 Program Activation and Termination 171
6.2.1 Program Termination 171
6.2.1.1 Execute-stop-statement 172
6.2.1.2 Stop-program 172
6.3 Block Activation and Termination 172
6.3.1 Activate-procedure 172
6.3.1.1 Install-arguments 173
6.3.2 Activate-begin-block 173
6.3.3 Prologue 174
6.3.4 Epilogue 175
6.4 Control within a Block 175
6.4.1 Normal-sequence 176
6.4.1.1 Advance-execution 176
6.4.2 Execute-executable-unit 176
6.4.3 Execute-begin-block 177
6.4.4 Execute-group 177
6.4.4.1 Establish-controlled-group 178
6.4.4.2 Initialize-spec-options 178
6.4.4.3 Test-spec 180
6.4.4.4 Advance-controlled-group 180
6.4.5 Execute-selection 182
6.4.5.1 Test-when-enablement 182
6.4.5.2 Test-when-matching 183
6.4.6 Execute-if-statement 184
6.4.6.1 Establish-truth-value 184
6.4.7 Execute-call-statement 185

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.7.1 Entry-references ... 185
6.4.7.1.1 Evaluate-entry-reference ... 185
6.4.7.1.2 Establish-argument ... 186
6.4.8 Execute-goto-statement ... 187
6.4.8.1 Local-goto ... 188
6.4.8.2 Trim-group-control ... 188
6.4.9 Execute-null-statement ... 189
6.4.10 Execute-return-statement ... 189
6.4.11 Execute-end-statement ... 190
6.4.12 Execute-leave-statement ... 191
6.5 Conditions and Interrupts ... 191
6.5.1 Conditions ... 192
6.5.1.1 Raise-condition ... 192
6.5.1.2 Test-enablement ... 192
6.5.1.3 Execute-signal-statement ... 193
6.5.1.4 Evaluate-named-io-condition ... 194
6.5.2 Interrupts ... 194
6.5.2.1 Execute-on-statement ... 194
6.5.2.2 Execute-revert-statement ... 195
6.5.3 Interrupt ... 196
6.5.4 System-action ... 197
6.5.4.1 Comment ... 197

7. Storage and Assignment ... 199
7.1 Introduction ... 199
7.2 The Generation ... 200
7.2.1 The Number of Elements in the Storage-index-list of a Generation ... 200
7.2.2 Correspondence between an Item-data-description and a Basic-value ... 201
7.2.3 Value of a Generation ... 202
7.2.4 Value of Storage Index ... 202
7.3 The Allocation of Storage ... 203
7.3.1 Execute-allocate-statement ... 203
7.3.2 Allocate-based-storage ... 204
7.3.3 Evaluate-in-option ... 206
7.3.4 Allocate ... 206
7.3.5 Evaluate-data-description-for-allocation ... 208
7.3.6 Find-directory-entry ... 208
7.3.7 Make-allocation-unit ... 209
7.3.8 Initialize-refer-options ... 210
7.3.9 Find-block-state-of-declaration ... 211
7.3.10 Construct-union-list ... 211
7.4 Initialization ... 213
7.4.1 Initialize-generation ... 213
7.4.2 Initialize-scalar-element ... 214
7.4.3 Initialize-array ... 214
7.5 The Freeing of Storage ... 216
7.5.1 Execute-free-statement ... 216
7.5.2 Free-based-storage ... 216
7.5.3 Deduce-in-option ... 218
7.5.4 Free ... 218
7.6 Assignment ... 219
7.6.1 The Assignment Statement ... 219
7.6.2 Target References ... 220
7.6.2.1 Evaluated Targets ... 221
7.6.3 The Assignment Operation ... 222

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.6.3.1 Promote-and-convert ... 222
7.6.3.2 The Set-storage Operation ... 223
7.6.3.3 Area-conversion ... 224
7.6.3.4 Test-union-matching ... 225
7.6.3.5 Set-union ... 225
7.6.4 Pseudo-variables ... 226
7.6.4.1 Exponent-pv ... 226
7.6.4.2 Onsource-pv ... 227
7.6.4.3 Pageno-pv ... 228
7.6.4.4 String-pv ... 228
7.6.4.5 Substr-pv ... 229
7.6.4.6 Unspec-pv ... 230
7.7 Variable-reference ... 231
7.7.1 Evaluate-variable-reference ... 231
7.7.1.1 Test Connectedness ... 232
7.7.1.2 Test String Class ... 233
7.7.2 Select-based-generation ... 234
7.7.3 Check-based-reference ... 234
7.7.4 Overlay-strings ... 236
7.7.5 Evaluate-data-description-for-reference ... 237
7.7.6 Select-qualified-reference ... 238
7.7.7 Select-subscripted-reference ... 239
7.7.8 Evaluate-defined-reference ... 241
7.7.9 Evaluate-simply-defined-reference ... 241
7.7.10 Evaluate-string-overlay-defined-reference ... 242
7.7.11 Check-simply-defined-reference ... 242
7.7.12 Extract-slice-of-array ... 243
7.8 Reference to Named Constant ... 244
7.8.1 Evaluate-named-constant-reference ... 244

8. Input/Output ... 247
8.1 Introduction ... 247
8.2 Datasets ... 247
8.2.1 Record Datasets ... 247
8.2.2 Stream Datasets ... 247
8.3 Files ... 248
8.3.1 Record Files ... 248
8.3.2 Stream Files ... 248
8.4 I/O Conditions ... 248
8.4.1 Raise-io-condition ... 249
8.5 Evaluate-file-option ... 249
8.6 File Opening and Closing ... 250
8.6.1 The Open Statement ... 250
8.6.1.1 Execute-open-statement ... 250
8.6.1.2 Execute-single-opening ... 250
8.6.1.3 Open ... 251
8.6.1.4 Evaluate-title-option ... 253
8.6.1.5 Evaluate-filename ... 253
8.6.2 The Close Statement ... 254
8.6.2.1 Execute-close-statement ... 254
8.6.2.2 Execute-single-closing ... 254
8.6.2.3 Close ... 254
8.7 The Record I/O Statements ... 255
8.7.1 The Read Statement ... 255

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.1.1 Execute-read-statement 255
8.7.1.2 Read 256
8.7.2 The Write Statement 258
8.7.2.1 Execute-write-statement 258
8.7.2.2 Write 259
8.7.3 The Rewrite Statement 259
8.7.3.1 Execute-rewrite-statement 259
8.7.3.2 Rewrite 260
8.7.4 The Delete Statement 261
8.7.4.1 Execute-delete-statement 261
8.7.4.2 Delete 262
8.7.5 Operations Applicable to Record I/O 263
8.7.5.1 Evaluate-from-option 263
8.7.5.2 Evaluate-into-option 263
8.7.5.3 Evaluate-pointer-set-option 263
8.7.5.4 Evaluate-key-option 264
8.7.5.5 Evaluate-keyfrom-option 264
8.7.5.6 Evaluate-keyto-option 264
8.7.5.7 Evaluate-sizeto-option 265
8.7.5.8 Construct-record 265
8.7.5.9 Insert-record 265
8.7.5.10 Position-file 266
8.7.5.11 Evaluate-size 267
8.7.5.12 Exit-from-io 268
8.7.5.13 Trim-io-control 268
8.8 The Stream I/O Statements 269
8.8.1 The Get Statement 269
8.8.1.1 Execute-get-statement 269
8.8.1.2 Execute-get-file 269
8.8.1.3 Execute-get-string 270
8.8.1.4 Get-list 270
8.8.1.4.1 Parse-list-input 272
8.8.1.4.2 Parsing Categories for List Directed Input 273
8.8.1.5 Get-edit 273
8.8.1.5.1 Execute-input-control-format 274
8.8.1.5.2 Execute-input-data-format 275
8.8.1.5.3 Validate-input-format 277
8.8.1.6 Input-stream-item 278
8.8.1.7 Basic-character-value 279
8.8.1.8 Basic-bit-value 279
8.8.1.9 Input-stream-item-for-edit 280
8.8.2 The Put Statement 280
8.8.2.1 Execute-put-statement 280
8.8.2.2 Execute-put-file 281
8.8.2.3 Execute-put-string 281
8.8.2.4 Put-list 282
8.8.2.5 Put-edit 284
8.8.2.5.1 Execute-output-control-format 284
8.8.2.5.2 Execute-output-data-format 285
8.8.2.5.3 Edit-numeric-output 286
8.8.2.6 Output-string 288
8.8.2.7 Output-string-item 288
8.8.2.8 Output-stream-item 289
8.8.2.9 Tab 289
8.8.2.9.1 Output-tab 290

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.2.10 Put-line 290
8.8.2.11 Put-page 291
8.8.3 Operations Applicable to Stream I/O 291
8.8.3.1 Skip 291
8.8.3.2 Evaluate-current-column 292
8.8.3.3 Evaluate-current-line 292
8.8.3.4 Establish-next-data-item 293
8.8.3.4.1 Expand-edd 295
8.8.3.4.2 Expand-generation 295
8.8.3.5 Establish-next-format-item 296
8.8.3.6 Evaluate-format-item 297
8.8.3.6.1 Evaluate-format-expression 297

9. Expressions and Conversion 299
9.1 Introduction 299
9.2 Aggregate Expressions 299
9.2.1 Scalar and Aggregate Types 299
9.2.1.1 Aggregate Type of a Data Description 299
9.2.1.2 Scalar Elements 300
9.2.1.3 Treatment of Scalars 300
9.2.1.4 Compatibility 300
9.2.1.5 Correspondence 300
9.2.1.5.1 Correspondence of Scalar Elements 300
9.2.1.5.2 Correspondence of Data Types 300
9.2.1.5.3 Correspondence of Union Data Descriptions 301
9.2.1.6 Generate-scalar-result 301
9.2.2 Integer Type 302
9.2.2.1 Evaluate-expression-to-integer 302
9.2.3 Derived Data Types 302
9.2.3.1 Derived Base and Scale 303
9.2.3.2 Converted Precision 303
9.2.3.3 Derived String Type 304
9.2.3.4 Further Definitions for Character and Bit Strings 304
9.2.4 Arithmetic Results 305
9.2.4.1 Conditions in Expressions 306
9.2.4.2 Operations Used in the Model of Floating-Point Arithmetic 306
9.2.4.2.1 Chopped-result 306
9.2.4.2.2 Composite-result 307
9.2.4.2.3 Rounded-result 308
9.2.4.2.4 Unconstrained-result 310
9.2.4.2.5 Test-model-number 310
9.2.4.2.6 Highest-positive-model-number 311
9.2.4.2.7 Lowest-positive-model-number 311
9.2.4.2.8 Lower-model-number 311
9.2.4.2.9 Higher-model-number 311
9.2.4.2.10 Model-base 312
9.2.4.2.11 Model-precision 312
9.2.4.2.12 Model-minimum-exponent 312
9.2.4.2.13 Model-maximum-exponent 312
9.2.5 Expressions 313
9.2.6 Value References 313
9.2.7 Constants 314
9.2.8 Parenthesized Expressions 314
9.2.9 Arguments 314

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.10 Enquiries ... 314
9.3 Prefix Operators ... 317
9.3.1 Prefix Expressions ... 317
9.3.2 Definition of the Prefix Operators ... 317
9.3.2.1 Prefix-minus ... 317
9.3.2.2 Prefix-not ... 318
9.3.2.3 Prefix-plus ... 318
9.4 Infix Operators ... 318
9.4.1 Infix Expressions ... 319
9.4.2 Definition of the Infix Operators ... 319
9.4.2.1 Infix-add ... 319
9.4.2.2 Infix-and ... 320
9.4.2.3 Infix-and-logical ... 320
9.4.2.4 Infix-cat ... 321
9.4.2.4.1 Concatenation of String Values ... 321
9.4.2.5 Infix-divide ... 321
9.4.2.6 Infix-eq ... 322
9.4.2.6.1 Compare ... 322
9.4.2.6.2 Non-trivial-basic-value-index ... 326
9.4.2.7 Infix-ge ... 326
9.4.2.8 Infix-gt ... 327
9.4.2.9 Infix-le ... 327
9.4.2.10 Infix-lt ... 328
9.4.2.11 Infix-multiply ... 328
9.4.2.12 Infix-ne ... 329
9.4.2.13 Infix-or ... 329
9.4.2.14 Infix-or-logical ... 330
9.4.2.15 Infix-power ... 330
9.4.2.16 Infix-subtract ... 331
9.4.2.17 Infix-xor ... 332
9.5 Built-in Functions ... 332
9.5.1 Built-in Function Reference ... 333
9.5.2 Special Terms Defined for Built-in Functions ... 333
9.5.2.1 Definition of N ... 333
9.5.2.2 The Arguments p and q ... 334
9.5.2.2.1 Constraints on p and q ... 334
9.5.2.2.2 Attributes of Result Determined by p and q ... 334
9.5.3 Operations Used in Built-in Function Definitions ... 334
9.5.3.1 Get-established-onvalue ... 334
9.5.4 Definition of the Built-in Functions ... 335
9.5.4.1 Abs-bif ... 335
9.5.4.2 Acos-bif ... 335
9.5.4.3 Add-bif ... 336
9.5.4.4 Addr-bif ... 336
9.5.4.5 Asin-bif ... 336
9.5.4.6 Atan-bif ... 337
9.5.4.7 Atand-bif ... 338
9.5.4.8 Atanh-bif ... 338
9.5.4.9 Binary-bif ... 339
9.5.4.10 Bit-bif ... 339
9.5.4.11 Bool-bif ... 339
9.5.4.12 Ceil-bif ... 340
9.5.4.13 Character-bif ... 341
9.5.4.14 Collate-bif ... 341
9.5.4.15 Copy-bif ... 342

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.16 Cos-bif 342
9.5.4.17 Cosd-bif 343
9.5.4.18 Cosh-bif 343
9.5.4.19 Currentsize-bif 344
9.5.4.20 Date-bif 344
9.5.4.21 Datetime-bif 345
9.5.4.22 Decimal-bif 345
9.5.4.23 Dimension-bif 345
9.5.4.24 Divide-bif 346
9.5.4.25 Edit-bif 347
9.5.4.26 Empty-bif 347
9.5.4.27 Every-bif 348
9.5.4.28 Exp-bif 348
9.5.4.29 Exponent-bif 349
9.5.4.30 Fileopen-bif 349
9.5.4.31 Fixed-bif 350
9.5.4.32 Float-bif 350
9.5.4.33 Floor-bif 350
9.5.4.34 Hbound-bif 351
9.5.4.35 High-bif 352
9.5.4.36 Identical-bif 352
9.5.4.37 Index-bif 353
9.5.4.38 Isochar-bif 354
9.5.4.39 Lbound-bif 355
9.5.4.40 Length-bif 356
9.5.4.41 Lineno-bif 356
9.5.4.42 Log-bif 357
9.5.4.43 Log10-bif 357
9.5.4.44 Log2-bif 357
9.5.4.45 Low-bif 358
9.5.4.46 Max-bif 358
9.5.4.47 Maxlength-bif 359
9.5.4.48 Min-bif 360
9.5.4.49 Mod-bif 361
9.5.4.50 Multiply-bif 362
9.5.4.51 Null-bif 362
9.5.4.52 Offset-bif 362
9.5.4.53 Oncode-bif 363
9.5.4.54 Onfile-bif 363
9.5.4.55 Onkey-bif 363
9.5.4.56 Onsource-bif 364
9.5.4.57 Pageno-bif 364
9.5.4.58 Pointer-bif 365
9.5.4.59 Prod-bif 365
9.5.4.60 Reverse-bif 366
9.5.4.61 Round-bif 366
9.5.4.62 Search-bif 367
9.5.4.63 Sign-bif 368
9.5.4.64 Sin-bif 368
9.5.4.65 Sind-bif 369
9.5.4.66 Sinh-bif 369
9.5.4.67 Some-bif 370
9.5.4.68 Sqrt-bif 370
9.5.4.69 String-bif 371

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.70 Substr-bif 371
9.5.4.71 Subtract-bif 372
9.5.4.72 Sum-bif 372
9.5.4.73 Tally-bif 373
9.5.4.74 Tan-bif 374
9.5.4.75 Tand-bif 375
9.5.4.76 Tanh-bif 375
9.5.4.77 Time-bif 375
9.5.4.78 Translate-bif 376
9.5.4.79 Trim-bif 377
9.5.4.80 Trunc-bif 378
9.5.4.81 Unspec-bif 379
9.5.4.82 Valid-bif 380
9.5.4.83 Verify-bif 380
9.6 Conversion 381
9.6.1 Conversion of Scalar Values 381
9.6.1.1 Informal Invocation of Convert 384
9.6.1.2 Convert-to-fixed 384
9.6.1.3 Convert-to-float 385
9.6.1.4 Convert-to-bit 386
9.6.1.5 Convert-to-character 387
9.6.1.6 Conversion to Float Decimal 388
9.6.1.7 Conversion from String or Picture to Arithmetic 389
9.6.1.8 Basic Numeric Value of a String 389
9.6.1.9 Evaluate-real-constant 390
9.6.1.10 Validate-offset 391
9.6.2 Numeric Pictures 391
9.6.2.1 Editing Numeric Pictures 392
9.6.2.2 Editing a Numeric Picture Field 393
9.6.2.3 Validity of a Numeric Pictured Value 397
9.6.2.4 Validity of a Field of a Numeric Pictured Value 398

Appendix A. Features New to This Revision Adopted from X3.53-1976 399
A.1 Lexical Constructs 399
A.2 Attributes and Declarations 399
A.3 Program Structure and Control 400
A.4 Storage Control 400
A.5 Input/Output 401
A.6 Expressions and Evaluation 401
A.7 Pictures 401
A.8 Built-in Functions and Pseudo-variables 401
A.9 Conditions and Exception Handling 403

Appendix B. Features New to This Revision 405
B.1 Lexical Constructs 405
B.2 Attributes and Declarations 405
B.3 Program Structure and Control 406
B.4 Storage Control 407
B.5 Input/Output 407
B.6 Expressions and Evaluation 408
B.7 Pictures 408
B.8 Built-in Functions and Pseudo-variables 409
B.9 Enquiries 410
B.10 Conditions and Exception Handling 411

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Appendix C. Clarifications and Restrictions from X3.74-1981 ..... **413**

Appendix D. Features of X3.53-1976 not Adopted in This Standard **415**
D.1 Program Structure 415
D.1.1 Syntactic Restrictions 415
D.1.2 Semantic Restrictions 415
D.1.3 Features Excluded 415
D.2 Program Control 416
D.2.1 Syntactic Restrictions 416
D.2.2 Semantic Restrictions 416
D.2.3 Features Excluded 416
D.3 Storage Control Statements 416
D.3.1 Syntactic Restrictions 416
D.3.2 Semantic Restrictions 416
D.3.3 Features Excluded 416
D.4 Input/Output 417
D.4.1 Syntactic Restrictions 417
D.4.2 Semantic Restrictions 417
D.4.3 Features Excluded 417
D.5 Attributes and Pictures 417
D.5.1 Syntactic Restrictions 417
D.5.2 Semantic Restrictions 418
D.5.3 Features Excluded 418
D.6 Built-in Functions and Pseudo-variables 418
D.6.1 Syntactic Restrictions 418
D.6.2 Semantic Restrictions 418
D.6.3 Features Excluded 419
D.7 Expressions 419
D.7.1 Syntactic Restrictions 419
D.7.2 Semantic Restrictions 419
D.7.3 Features Excluded 420
D.8 Change in Intermediate Results 420

Appendix E. Incompatible Changes from X3.74-1981 ..... **421**

Appendix F. Rationale for Major Design Decisions **423**

Index **427**

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Figures

1. Example of Syntax Brackets. . . . 4
2. The Prologue Operation. . . . 13
3. Examples of Syntax. . . . 27
4. An Example of an Operation Returning a Result. . . . 31
5. An Example of an Operation Containing a Step-list and a Case-set. . . . 32
6. Example Showing ⟨operation-list⟩s of a ⟨machine-state⟩. . . . 38
7. The Structure of a Simple ⟨translation-unit⟩. . . . 53
8. The Structure of an ⟨if-statement⟩. . . . 54
9. The Structure of an ⟨expression⟩. . . . 55
10. The Structure of a ⟨do-statement⟩. . . . 56
11. Examples of Abstract Text. . . . 74
12. Table of ⟨bit-values⟩ as a Function of ⟨symbols⟩ and ⟨radix-factors⟩ for CREATE-CONSTANT. . 149
13. An Example of ⟨generation⟩s and ⟨basic-value⟩s of Defined Variables. . . . 204
14. Table of Converted Precisions as a Function of Target and Source Attributes. . . . 304
15. An Example of Optional Arguments. . . . 333
16. Table of Scalar-results as a Function of ⟨bit-values⟩ for BOOL-BIF. . . . 340
17. An Example of the BOOL Built-in Function. . . . 340
18. An Example of the COPY Built-in Function. . . . 342
19. An Example of the DIMENSION Built-in Function. . . . 346
20. Examples of the INDEX Built-in Function. . . . 354
21. Some Examples of the MAXLENGTH Built-in Function. . . . 360
22. Some Examples of the TALLY Built-in Function. . . . 374
23. An Example of the TRANSLATE Built-in Function. . . . 377
24. An Example of the TRIM Built-in Function. . . . 378
25. Examples of the VERIFY Built-in Function. . . . 381
26. Table of Terminal Nodes of ⟨symbols⟩ as a Function of ⟨suppression-type⟩ for EDIT-NUMERIC-PICTURE-FIELD 397

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 6522 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

This second edition cancels and replaces the first edition (ISO 6522:1985), which has been technically revised.

Annexes A to F of this International Standard are for information only.

Terminology and conventions

The text of American National Standard ANSI X3.74-1987 has been approved for publication, without deviation, as an International Standard. Some terminology and certain conventions are not in accordance with the ISO/IEC Directives Part 3: "Drafting and presentation of International Standards"; attention is especially drawn to the following:

Wherever the word "standard" appears, referring to this International Standard, it should be read as "International Standard".

Cross-reference

American National Standard	Corresponding International Standard
ANSI X3.134.1	ISO/IEC 4873:1991, *Information technology — ISO 8-bit code for information interchange — Structure and rules for implementation.*
ANSI X3.4-1986	ISO/IEC 646:1991, *Information technology — ISO 7-bit coded character set for information interchange.*
ANSI X3.41-1974	ISO 2022:1986, *Information processing — ISO 7-bit and 8-bit coded character sets — Coded extension techniques.*
ANSI X3.53-1976	ISO 6160:1979, *Programming languages — PL/I.*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information technology — Programming languages PL/I general purpose subset

1. Scope and Overviews

1.1 Scope

This document defines the computer programming language PL/I General Purpose Subset, referred to in most of this definition simply as "PL/I". It is intended to serve as an authoritative reference rather than as a tutorial introduction.

The language defined by this document is different from those of previous PL/I standards, although substantially upward compatible, at the source program and semantic level. The differences are summarized in Appendix A. The text that follows defines standard PL/I in terms of an extended reference PL/I language. The extended reference PL/I language contains operations and portions of operations that are not accessed by PL/I programs that conform to this standard. Those features that are implied by the reference language but for which support is not required of conforming processors are retained for convenience of PL/I implementations containing extensions beyond this standard and for definitional compatibility with earlier PL/I standards (see Appendix A).

As explained in more detail in Section 1.5, this standard specifies the syntax and semantics of conforming PL/I programs. It makes no proscriptive statements about the syntax or semantics of non-conforming programs. It defines a conforming processor (or conforming implementation) only in terms of those conforming programs: a conforming processor is one that, subject to the flexibilities and requirements of Section 1.5, gives all conforming programs the interpretations specified by this definition.

The definition is accomplished by specifying a conceptual PL/I machine which translates and interprets intended PL/I programs. Section 1.4 provides a brief introduction to the statements and data types included in the language, to the structure and use of the document, and to the method of definition. The relationship between an actual implementation and the conceptual machine of this document is described in Section 1.5, and the detailed specification of the notation to be used follows in Section 1.6. The main body of the definition is then begun at Section 1.7, and is completed by Chapters 2 through 9.

1.2 Referenced Publications

This standard shall be used in conjunction with the following publications. When the following American National Standards are superseded by a revision approved by the American National Standards Institute, Inc., the revision shall apply.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- ANSI X3.134.1, Information Systems - 8-bit ASCII - Structure and Rules (under development)

  Note: For international use, ISO 4873-1984 or its successors should be substituted.

- ANSI X3.4-1986, Information Systems - Coded Character Set - 7-Bit American National Standard Code for Information Interchange

  Note: For international use, ISO 646-1983 or its successors should be substituted.

- ANSI X3.41-1974, Information Systems - Code Extension Techniques for Use with the 7-bit Coded Character Set of American National Standard Code for Information Interchange (ASCII)

  Note: For international use, ISO 2022-1986 or its successors should be substituted.

- ANSI X3.53-1976 (R 1987), Programming Language PL/I

  Note: Identical to ISO 6160-1979.

1.3 Goals and Rationale

The language defined by this document differs from the PL/I language of ANSI X3.53-1976 and ANSI X3.74-1981 by having a few additional features that have been determined to be necessary in the years since the publication of those documents. With the exception of those new features, it remains, like its predecessor, a proper subset of the full PL/I language specified in American National Standard Programming Language PL/I, ANSI X3.53-1976. The differences from those earlier standards are summarized in Appendix A and Appendix B. The language defined here is designed to have the following properties:

1. The subset is a general-purpose programming language that is well suited for use in commercial, scientific, and systems programming application areas.

2. The subset is small enough to achieve widespread implementation yet large enough to achieve widespread usage.

3. The subset is easier to learn and understand, substantially less expensive to implement, and utilizes computer resources more economically than full PL/I (ANSI X3.53-1976).

4. Programs written in the subset are more portable than programs written in the full language.

5. Programs written in the subset are less likely to contain programming errors than programs written in the full language because the subset is both simpler and less permissive than the full language.

Technology now permits the implementation of subsets of PL/I on a variety of medium and small hardware configurations with limited control program software, including most minicomputer and some microprocessor-based systems.

A major benefit of the language defined here is the reduction or elimination of incompatibilities among the various PL/I implementations on such machines.

The language defined here provides a high degree of portability of both programs and programmers among different computers and operating systems. It includes special features for making programs adaptable to differences in environment that go beyond the language itself and has been designed so that programs can be easily converted to and from non-standard PL/I dialects.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.4 An Informal Guide to the PL/I Definition

1.4.1 A Summary of PL/I

A PL/I program consists of a nonempty set of procedures and a possibly empty set of packages, each of which is written as a sequence of statements. The %INCLUDE construct may be used to include text from other sources during program translation.

All of the statement types are summarized here in groupings which are presented as a means of obtaining an overview of the language. The structure of the document reflects, to some extent, the organization of this list.

Structural:	PACKAGE PROCEDURE BEGIN DO END
Declarative:	DECLARE DEFINE CONSTANT FORMAT
Flow of Control:	CALL RETURN IF SELECT LEAVE GO TO Null Statement STOP
	ON REVERT SIGNAL — Interrupt Handling
Storage:	ALLOCATE FREE
	Assignment Statement
Input/Output:	OPEN CLOSE
	GET PUT — Stream I/O
	READ WRITE REWRITE DELETE — Record I/O

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Names may be declared to represent data of the following types, either as single values, or as aggregates in the form of arrays or structures:

```
Arithmetic
Arithmetic PICTURE
CHARACTER
BIT
AREA
ENTRY
FILE
LABEL
OFFSET
POINTER
```

Values may be computed by expressions written using a specific set of operators, built-in functions and enquiries, some of which may be applied to aggregates as well as to single values, together with user-defined procedures which, likewise, may operate on aggregate as well as single values, but may only return single values.

1.4.2 The Form of the Definition

The conceptual PL/I machine is a processor which has a set of operations acting on information stored in its memory. The operations are specified as algorithms, and may be viewed as the component parts of one single algorithm, "DEFINE-PROGRAM", which carries out the entire translation and interpretation process. The information in memory is all held in the form of tree structures.

The definition algorithm operates as follows.

Sequences of symbols which are intended to represent PL/I *packages* and *external procedures* (i.e. procedures not contained in any other procedure or package) are processed by a Translator. Packages and external procedures are collectively called *translation-units*. This processing involves systematically analyzing, transforming, and validating each translation unit. The analysis uses a grammar known as the *Concrete Syntax* to produce the *concrete form* of a translation unit as a tree structure. This is transformed to the *abstract form,* which is a tree satisfying the *Abstract Syntax,* designed to be more convenient for interpretation. Further validation is then carried out on the abstract form. The Translator finally performs some validation of the mutual consistency of the set of translation units which comprise the complete program.

The semantics of the program when applied to given initial *datasets* (i.e. collections of data) are then provided by an Interpreter. The datasets are part of the PL/I machine's memory, which is a tree satisfying the *Machine-state Syntax,* and it is the sequence of changes in the datasets which constitutes the defined semantics.

The denotation of the category names of each of the syntaxes includes the following brackets:

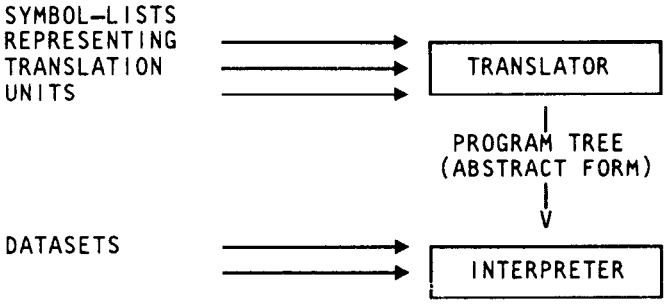
⊰...⊱	Concrete syntax
<...>	Abstract syntax
≪...≫	Machine state syntax

Figure 1. Example of Syntax Brackets.

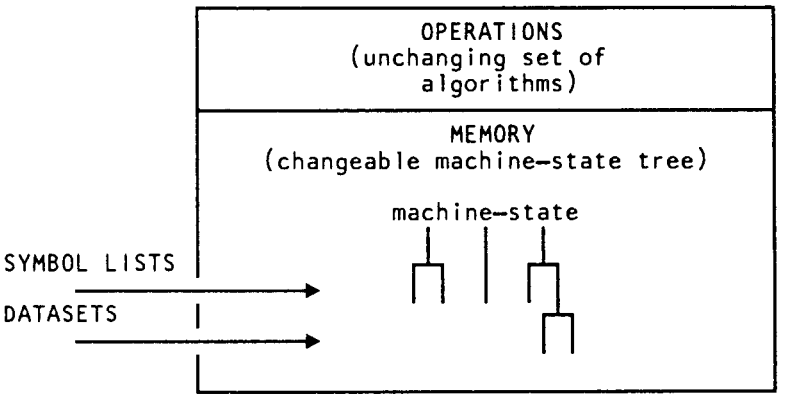
In addition to translating and interpreting all programs which are valid according to this definition, the PL/I machine detects as non-standard all those which violate a requirement involving the words "MUST" or "MUST NOT" in the algorithms performed during translation or interpretation. The implications for an actual implementation are described in Section 1.5.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The method of definition may be seen in outline as follows:



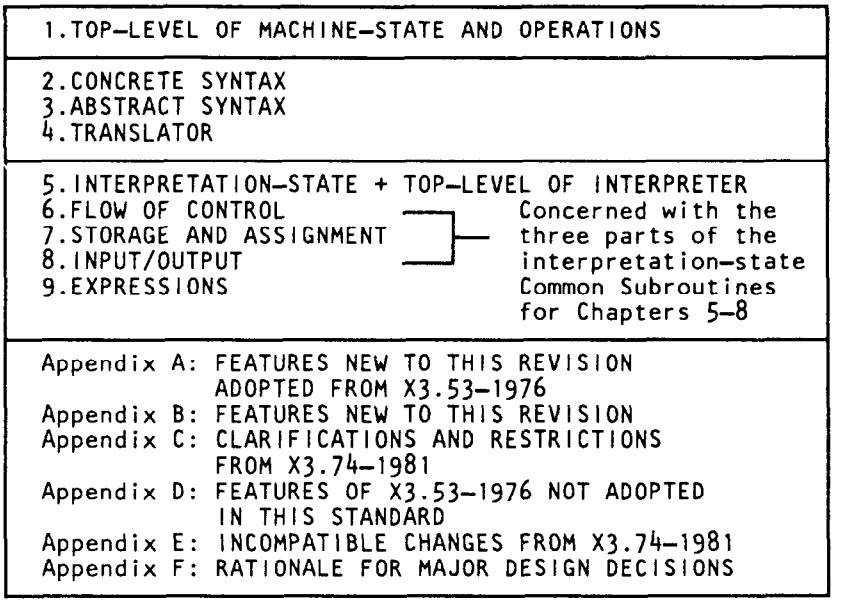
It may also be helpful to visualize the abstract machine on which both the Translator and the Interpreter are "executed":



The inputs from outside the machine occur at initialization of the Translator and Interpreter; the datasets may change during interpretation. However, there are no outputs defined since the datasets are treated for the purposes of this definition as being a part of the storage of the machine, i.e. as being "on-line" when needed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.4.3 Summary of Chapter Structure



The operations of Chapter 1 serve to drive the Translator and Interpreter.

The operations of the Translator are all contained in Chapter 4, and use the syntaxes of Chapters 2 and 3.

The operations of the Interpreter comprise all the operations in Chapters 5-9. After the initialization in Chapter 5, the relevant operations will be in Chapters 6, 7, or 8 depending on the type of statement being interpreted. All of these chapters invoke operations in Chapter 9 where necessary.

All readers are recommended to acquire a good understanding of Chapter 1 in its entirety. Thereafter, it is possible to read the definition as a systematic whole, or to use the document to locate answers to specific questions by combining an appreciation of the overall structure of the definition with judicious use of the index. To illustrate this latter usage, we consider each chapter in turn together with a sample question answerable from it.

Chapter 2 contains the definition of the Concrete Syntax. The Concrete Syntax consists of rules describing valid forms of PL/I constructs in concrete tree form. The syntax is permissive in the sense that some of the constructs permitted as being syntactically correct may later be found to be meaningless.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

QUESTION

Is the following statement correct?

GET LIST (A(I,J) DO I = 1 TO M,N);

ANSWER

The first possibility is that there may be an error according to the Concrete Syntax. The index entries for "get-statement" lead to the High-level Syntax, and study of rules CH115, CH116, CH123, CH124, and CH125 reveals that an extra pair of parentheses is required around the form ⟨input-target-commalist⟩ DO ⟨do-spec⟩. The correct form is:

GET LIST ((A(I,J) DO I = 1 TO M),N);

Chapter 3 contains the definition of the Abstract Syntax. Many parts of the Abstract Syntax description intentionally bear a strong resemblance to the corresponding parts of the Concrete Syntax. Names in the Abstract Syntax have been chosen to resemble those of corresponding parts of the Concrete Syntax in order to make obvious as far as possible the relationship between the syntaxes.

QUESTION

May the KEYTO option on a READ statement specify that the key be assigned to a substring of a variable?

ANSWER

The Concrete Syntax for a ⟨keyto-option⟩ shows merely that a ⟨reference⟩ MUST be specified. However, the Abstract Syntax shows the form of a program after the Translator has completed all declarations, and has thus been able to associate each reference with the appropriate declaration and make more subtle distinctions. The rule A123 for <keyto-option> shows "<variable-reference> (scalar AND character)". The parenthesized constraint "(scalar AND character)" shows that it MUST be a single target (not an array or structure) which is character-valued.

<pseudo-variable-reference>s are only allowed in a <target-reference>. Since the <keyto-option> does not accept a <target-reference>, a substring may not be used. Pseudo-variables are only allowed on the left hand side of an assignment statement.

Chapter 4 defines the Translator whereby each of the individual PL/I program portions (translation units) is translated from the submitted character string form to tree form and appended to the program tree. This process involves the parsing of each translation-unit using the Concrete Syntax to obtain a concrete tree, insertion of missing options and completion of attribute sets in that concrete tree, conversion from that concrete tree to an abstract tree, and, finally, validation of the whole program. Once formed, the abstract tree is not modified.

QUESTION

What file is implied in PUT LIST(X); ?

ANSWER

This seems at first sight as though it might be a semantic question about the <put-statement>. However, the Abstract Syntax shows that a <file-option>is required to be present in <put-file> (A130), and this means that if it was absent in the concrete form, it would have been supplied by the Translator if the statement was valid.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In fact, immediately after parsing the input, the Translator completes the concrete procedure in various ways, one of which is to insert the equivalent of FILE (SYSPRINT) into our ⟨put-statement⟩ (Step 2 of the operation COMPLETE-OPTIONS, 4.5.1.1).

The reason that it has to be handled early in the Translator is that it will lead to a contextual declaration of the name SYSPRINT if our statement is not within the scope of an existing declaration for SYSPRINT. It is necessary to complete all declarations prior to execution in order to resolve references correctly.

Chapter 5 contains the definitions of the Machine-state Syntax, the initialization of the machine-state tree, and the starting of the interpretation process.

QUESTION

May the first procedure executed in a program have arguments passed to it?

ANSWER

For the initialization of execution, we consult Sections 1.7.3.3 and 5.4.1.

In Step 1 of operation INTERPRETATION-PHASE (Section 1.7.3.3) an ⟨entry-value⟩ is obtained from outside the definition. It designates the first procedure in the <program> to be activated. The "MUST NOT" condition specifies that the document gives no meaning to a program whose <entry-point> designated by the ⟨entry-value⟩ contains a <parameter-name-list>. (See Section 1.6.3.4 for the definition of MUST.)

Additionally, in Step 2 of operation INTERPRET (5.4.1), an ⟨evaluated-entry-reference⟩ containing only an ⟨entry-value⟩ and not an ⟨established-argument-list⟩ (see production rule M22 in 5.2.2) is used to activate the first procedure.

Thus the passing of an argument-list to the first procedure would be an extension beyond the language defined in this document.

Chapter 6 describes the operations of the Interpreter affecting the flow of control through blocks, groups, and statements of the program. Normal flow of control consists of the execution of a list of executable units within a block. The definition also defines calling, parameter/argument matching and result returning, and abnormal flow of control caused by interrupts.

QUESTION

Is it permissible to REVERT a condition for which no ON statement has been executed?

ANSWER

This is a semantic question about interrupt handling. The operation EXECUTE-REVERT-STATEMENT (6.5.2.2) deletes a member from the current ⟨established-on-unit-list⟩ if an appropriate one exists, otherwise it performs NORMAL-SEQUENCE to pass on to the next statement. Since the action of EXECUTE-ON-STATEMENT is merely to append ⟨established-on-unit⟩s to such lists, it is clear that EXECUTE-REVERT-STATEMENT is indifferent to the absence of such actions. The answer to the question is that it is permissible and has no effect on the interrupt handling mechanism.

Chapter 7 defines the use of storage including the allocation, freeing and initialization of storage, and the referencing of variables and named constants. The assignment statement, aggregate assignment, and pseudo-variables, are also defined.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

QUESTION

Does DECLARE A(5) INITIAL (0); lead to all 5 elements of A being set to 0 when A is allocated?

ANSWER

The allocation and initialization of storage are treated in Chapter 7.

The operation INITIALIZE-ARRAY (7.4.3) iterates over Step 4 while $n \leq nt$. Since nt in this case is 1 (see Step 2) and n is initially 1 and is incremented in Step 4.7, the iteration is performed once only, thus initializing the first element A(1) only.

If it is desired to initialize the entire array with a scalar value, there is a mechanism to do this. The syntax is of the form DCL A(5) INITIAL((*)0); and the operation is carried out in Step 1 of INITIALIZE-ARRAY.

Chapter 8 deals with the transmission of data between external media and internal storage, including the opening and closing of files, stream and record transmission, and interrupts applicable to I/O operations.

QUESTION

Is DECLARE F RECORD; OPEN FILE(F) PRINT; valid?

ANSWER

Although most of the declarative structure is evident in the Abstract Syntax and checked by the Translator, this is not altogether true of file attributes since they may still be incomplete until the file is opened during execution. Therefore their valid combinations have to be tested also at this late stage.

The OPEN operation (Section 8.6.1.3) shows that <print> implies <stream> in Step 2, and that <stream> and <record> cause the result <fail> to be returned (Step 7) from the attempt to open the file. Thus if the <open-statement> is executed, it will lead to the raising of the UNDEFINEDFILE condition when an attempt is made to open the file (see Step 6 of EXECUTE-SINGLE-OPENING, Section 8.6.1.2).

Chapter 9 describes the evaluation of expressions, also conversions between data types, which can take place in expression evaluation, and the built-in functions of PL/I.

QUESTION

Is it possible to add two arrays?

ANSWER

The evaluation of expressions in Chapter 9 begins by considering the general treatment of aggregate operands. Investigation of the INFIX-ADD operation (9.4.2.1) and the ADD-BIF (9.5.4.3) indicates that the operands of the addition operator or the ADD built-in function must be scalars. Thus two arrays may not be added together, except by using an explicit loop to add each element.

Appendix A describes the features of this standard that extend it from its earlier version by adding language constructions present in ANSI X3.53-1976.

Appendix B describes the features of this standard that represent new language — features not present in either the earlier version of this standard ANSI X3.74-1981 or in ANSI X3.53-1976.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Appendix C describes clarifications and restrictions new to this version of the standard. These clarifications and restrictions are more likely to affect processors for the language than practical applications programs written in it.

Appendix D describes the features of ANSI X3.53-1976 that have not been incorporated in this standard and describes a difference in the semantics of the two standards.

Appendix E lists changes to the language described in ANSI X3.74-1981 that are incompatible, i.e., changes that will have to be made to programs conforming to that standard to make them operate correctly with this one.

Appendix F gives the rationale for the major decisions in the design of the language specified in this standard.

1.4.4 Introduction to the Metalanguage

An informal discussion of the main features of the metalanguage is now offered before proceeding to its more rigorous definition in Section 1.6.

1.4.4.1 Tree Concepts

The definition of PL/I deals with three classes of trees (concrete parse, abstract text, and machine-state), and a uniform concept of tree is employed throughout. This is of a tree which is a directed graph with a label (e.g. <procedure>) at each node, and where the subtrees of any node are ordered. (Although the ordering is often irrelevant, it is needed, e.g., in the concrete parse and in lists, and it was decided that the simple uniformity of concept outweighed the advantages of explicitly distinguishing instances where the order was significant.) Moreover, each node implicitly has a unique-name by which it can be "designated" when required. A copy of the tree has the same ordered structure and labelling, but new unique-names to designate its nodes. Equality of trees requires equality of all except the unique-names of nodes, and identity requires that these also match.

The explicit label of a node may be either a grammatical category-name, or some other type of value such as an integer or a designator (i.e. a copy of the unique-name implicitly associated with some node). Thus a single value may be handled as a degenerate tree with only a root-node labelled with this value, and data objects referenced in the metalanguage are uniformly regarded as trees.

The terminology applied to trees is developed from the starting-point of a tree consisting of a node (the *root-node*) and an ordered, possibly empty, set of *immediately contained* trees.

A tree, X, is said to be *contained* in a tree, Y, if X is immediately contained in Y, or if X is immediately contained in some tree contained in Y. X is *simply contained* in Y if and only if it is contained in Y and it is not contained in any tree, Z, contained in Y and having a root-node equal to that of X or of Y. For example, if we refer to an <expression> simply contained in a tree representing some statement in a program, we mean the complete <expression> tree and not some subexpression which might be a tree rooted in <expression> at some lower level within it. Since simple containment is a frequently required concept, the use of any form of possessive phrase not employing the verb "contain" or the noun "component", is taken to imply simple containment, e.g. "if y has an <expression>" (meaning a tree with root-node labelled <expression>), "the <expression> of y", or "its <expression>".

The terminology makes it possible to perform the three essential types of operation on trees, namely, to test them for the presence or type of their subtrees, to select subtrees from them, and to construct or alter them. However, construction can be laborious if phrased as "root-node <a> with two immediate subtrees, the first being <b> with immediate subtrees <c> and <d>, and the second being x" (where x is the name of some tree, meaning that a copy of it is to be made in constructing the new tree). This may therefore be abbreviated as

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<a>:
   <b>:
      <c>
      <d>;
   x;
```

The indentation is inessential, although helpful with large trees. One or more trailing semicolons at the end of such a constructed tree may optionally be replaced by a period.

1.4.4.2 Syntaxes

Proceeding from consideration of particular trees to classes of trees, we encounter grammars composed of sets of production-rules couched in a slightly extended Backus Naur Form. The interpretation we make of such rules is that they describe the structure of a tree belonging to the class described by the grammar — it is only in the traditional syntactic use of these rules that the sequence of terminal nodes of a tree acquires a special importance as being the sequence of characters representing an utterance in the language.

As an illustration, we will construct a tree that conforms to a rule of the form

<p> ::= <q> | {<r-list> | <s>}•<t>[<u>]

A grammatical rule in the metalanguage is rather like a statement in a language which it may be used to define, i.e. a metalanguage rule itself has a grammatical structure which has to be validated and correctly interpreted. We have chosen to escape from circularity or regress at this point by not giving a formal grammar for the metalanguage, but describing in prose how a rule is to be interpreted. So in the above illustration, the rule indicates that any node labelled <p> must have subtrees as described by the right-hand side of the rule.

In descending order of precedence there are three types of metalanguage operators:

1. for concatenation (indicated by juxtaposition in the rule),
2. for permutation ("•"), and
3. for a choice between alternatives (" | ").

Expressions within braces are regarded as single operands, and the first stage in using a rule to construct a tree is to partition the syntax expression into subexpressions separated by " | ", the operator of lowest precedence, and to choose one of these alternative subexpressions. Suppose we discard the alternative of having <q> alone, and next partition the other alternative by the "•" operator and decide to use the permutation which reverses these partitions to yield

<t>[<u>]{<r-list> | <s>}

The brackets indicate that their contents may be optionally omitted, which we choose to do here, and the braces enclose a syntax expression which is interpreted according to the method just described. Choosing the alternative <r-list> we would then have the tree

```
<p>:
   <t>
   <r-list>;
```

The underlining of t indicates that it always labels a terminal node in the grammar in question, i.e. it does not occur on the left-hand side of any production-rule. <r-list> indicates by convention that it is to have one or more immediate subtrees of type <r> beneath it. We then apply the rule for <r> separately to each of these. The construction is completed when each terminal node of the tree is either of a type which is always terminal or of a type which has produced an empty set of subtrees.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.4.4.3 Algorithm Concepts

The operations of the abstract machine are defined by algorithms, which may be compared with the logic or microcode supporting the operation codes of a computer. These operations inspect the ≪machine-state≫ or change the ≪machine-state≫, or both, i.e. the memory of the abstract machine, which contains all the information directly or indirectly affecting semantics, including some form of the <program> itself. (The metabrackets "≪" and "≫" are used in the PL/I definition of the machine-state nodes, except for those belonging to the grammar of the abstract program which are distinguished by "<" and ">" to help the reader.)

An operation of the machine is defined by a sequence of Steps, or a set of mutually exclusive Cases, numbered from 1 to n. The *i*th Step or Case may itself contain a sequence of Steps or a set of Cases numbered from i.1 to i.m, and so forth. Each Step or Case specifies actions to be performed, using various informal "statement types" in the metalanguage. A Case begins by stating the predicate that must be satisfied for it to be applicable.

As an example of an operation, consider the definition of the PROLOGUE operation in Figure 2 on page 13. This begins with an introductory paragraph which gives some guidance to the reader (and that is not part of this standard), but the definitive material is not reached until the "Operation:" heading.

Step 1 consists of an *attach* action, meaning that one tree is to be copied as a subtree of another in the position in which it may validly occur according to the syntactic rules governing the type of the latter. The word "current" has been defined precisely with respect to the particular ≪machine-state≫ used in defining PL/I - the "current linkage-part" is the linkage-part simply contained in the "block-state" corresponding to the block currently being executed. Step 2 indicates iteration with a *for each* specification applied to the first form of the *perform* action, which refers to one or more Steps in the same operation. A name such as d introduced after a comma is a variable, local to the operation, whose value (v, say) is a designator of the tree mentioned immediately preceding the "d". On subsequent uses of d, it is dereferenced, i.e. it means "the tree designated by v" except when it is redefined in a way similar to its original definition (as would happen here on the next iteration of "for each"), or by its occurrence after the word "let". The *let* statement is merely a more explicit syntax for this kind of definition of a local variable, and Step 2.1 contains an instance of this. If d contains <automatic>, the predicate of Case 2.1.1 is satisfied and Steps 2.1.1.1 to 2.1.1.3 will be performed in sequence.

The second form of *perform,* which is used to invoke another operation as a subroutine or function, occurs in Step 2.1.1.1. Here it is a function that is invoked "to obtain" a resulting value, which is then given a name. edd is passed as an argument to this operation, whose definition will name a parameter to correspond to it. Argument passing is by reference, i.e. a designator to the argument is passed and becomes the value of the parameter, which is then dereferenced on use, behaving just like a local variable.

The *append* action in Step 2.1.1.2 attaches a tree at the end of a list, and Step 2.1.1.3 exemplifies an *if* statement containing a subroutine call. Steps 3 and 4 consist of the other two tree actions which we use, *delete* and *replace*. When a tree is replaced by another, the first tree is deleted and a copy of the second tree is made with its root at the same position, and having the same implicit name, as had the root of the tree just deleted. Assignment of a value to a variable is permitted in the syntactic form of a *set* action, e.g. "Set tv to ≪true≫", but this is only an alternative form of replacement with the usual implied dereferencing of the variable name, meaning "replace the tree designated by the strict designator value of tv by ≪true≫".

Other statement types not illustrated here are *go to* (used sparingly), *terminate* an operation at some point other than the end of the last Step, and *return* a value from a function (which also terminates the operation). Values are returned by reference, so that if the result is a tree constructed within the function it MUST be copied back to the caller who will then receive a designator of the copy; values local to an operation are deleted when it terminates.

We have now reached a point at which it is necessary to indicate a mechanization of the metalanguage which will suffice to bolster intuition in its weaker moments. Step 4 provides the cue for this, since it constructs a tree

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.3 Prologue

This operation is invoked at the beginning of every block activation to establish the <automatic> and <defined> variables local to that block. The <automatic> variables are initialized if their <declaration>s specify initialization. Any <expression>s evaluated during the prologue, such as in <extent-expression>s or <expression>s in <initial>, are not allowed to reference other <automatic> or <defined> variables local to this block. The operation FIND-DIRECTORY-ENTRY will impose the restriction when it finds a reference to a variable declared in a block for which there exists a ⟨prologue-flag⟩. The ⟨prologue-flag⟩ is only present while the operation PROLOGUE is active.

Operation: PROLOGUE

Step 1. Attach a ⟨prologue-flag⟩ to the current ⟨linkage-part⟩.

Step 2. For each <declaration>,d, of the current block, that contains <automatic> or <defined>, perform Step 2.1

Step 2.1. Let id be the <identifier> immediately contained in d, and let dd be the <data-description> immediately contained in the <variable> of d. Perform EVALUATE-DATA-DESCRIPTION-FOR-ALLOCATION(dd) to obtain an ⟨evaluated-data-description⟩,edd.

Case 2.1.1. d contains <automatic>.

Step 2.1.1.1. Perform ALLOCATE(edd) to obtain a ⟨generation⟩,g.

Step 2.1.1.2. Append to the current ⟨automatic-directory-entry-list⟩ an ⟨automatic-directory-entry⟩: id g.

Step 2.1.1.3. If d contains <initial> then perform INITIALIZE-GENERATION (g,d).

Case 2.1.2. d contains <defined>.

Append to the current ⟨defined-directory-entry-list⟩ a ⟨defined-directory-entry⟩: id edd.

Step 3. Delete the ⟨prologue-flag⟩ of the current ⟨linkage-part⟩.

Step 4. Replace the current ⟨statement-control⟩ by a

⟨statement-control⟩:
⟨operation-list⟩:
⟨operation⟩ for ADVANCE-EXECUTION.

Figure 2. The Prologue Operation.

which is to be placed in the ⟨machine-state⟩ at exactly the point at which it tails off into informality. Of the ⟨operation-list⟩s in the ⟨machine-state⟩, just one is said to be "active" and has a conceptual processor associated with it which may execute operations. In particular, the PL/I machine has a current ⟨block-state⟩ which has a ⟨statement-control⟩ where one could mechanize the actions of the metalanguage involved in interpreting the statements of the current block.

The ⟨operation-list⟩ is to be conceived as a push-down stack, where each ⟨operation⟩ will have a subtree which is not formally defined, but contains such information as the name of the operation, a list of its

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

parameters and local variables with their current values, all trees constructed during execution of the operation or copied back to it from invoked functions, an indication of whereabouts in which Step or Case it is executing its algorithm, how far it has proceeded through a "for each" iteration, and so forth. When a "perform" invokes another operation, this pushes down the stack and becomes the active operation; when an operation terminates, its whole tree (including its local values) is deleted and the stack pops up, activity in the operation now at the head of the stack being resumed immediately after the point in its algorithm at which it was suspended.

To return to the analogy suggested at the beginning of this section, it is as though an abstract processor had a machine cycle which could cause the execution of microcoded operations with their own local memories per invocation, together with the sophistication of the stacking capability.

1.5 Relationships between an Implementation and this Definition

The inputs of the conceptual PL/I machine are one or more ⟨symbol-list⟩s representing PL/I translation units, an ⟨entry-value⟩, and a ⟨dataset-list⟩. This combination will be referred to as a *program-run.*

The standard definition of PL/I for a particular implementation is completed by defining (not necessarily in the style of this document) the IMPLEMENTATION-DEFINED features listed in Section 1.5.2, together with the representation of a program-run's elements (⟨symbol-list⟩s, ⟨entry-value⟩, and ⟨dataset-list⟩) in the implementation's operating environment. With this information available, the conceptual PL/I machine gives one or more interpretations to a program-run.

The main purpose of this document is to define the semantics of interpreting valid PL/I program-runs. These semantics are constituted solely by the sequence of changes in the ⟨dataset⟩s of the ⟨machine-state⟩, and an implementation is free to achieve them by any means. The operations and other parts of the ⟨machine-state⟩ which are the mechanisms used in this document to define the semantics need not be reflected directly in implementation.

An implementation's interpretation of a program-run conforms to this standard if and only if it conforms to one of the conceptual interpretations as follows:

1. If the conceptual interpretation rejects a program-run (via failure of a MUST test) or if it never completes the translation-phase, then any interpretation by the implementation conforms. In particular, an implementation may or may not reject a program-run at the same point as it is rejected by this standard, or at all.

2. Otherwise, the implementation's interpretation conforms if it makes the same sequence of changes to datasets as does the conceptual interpretation.

3. The implementation's interpretation also conforms if it deviates from (1) and (2) only as permitted by the flexibilities of interpretation specified in Section 1.5.1.

Note: This implies that an implementation may provide extensions beyond the language defined in this standard, but is still required to conform for a program not using those extensions just as if the extensions were not available.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.5.1 Flexibilities of Interpretation

Through use of the terms "optionally", "implementation-dependent", and "in any order", the operation definitions of the conceptual PL/I machine permit most of the flexibility necessary for efficient implementation of PL/I. However, there are some rules that, if given formally with a metalanguage of the kind used here, would require constructions so elaborate as to impede understanding seriously. These rules are given in this section using informal language and making direct reference to possible actions of an implementation.

Note: "implementation-dependent" and IMPLEMENTATION-DEFINED are not the same. IMPLEMENTATION-DEFINED features are given a specific definition, and requirements are imposed on them, in Section 1.5.2.

1.5.1.1 Rejection of Programs

If some part of a program is such that its interpretation would cause the program to be rejected, then an implementation may reject the program even if the conceptual interpretation does not reach the offending part.

1.5.1.2 Quantitative Restrictions

An implementation may make quantitative restrictions not contained in this standard. For example, restrictions may be made in the following contexts:

1. Where syntaxes allow iterative or recursive constructs of arbitrary size, an implementation may restrict the size of these constructs provided the construct is not deleted. In particular, a standard implementation may limit the maximum length of an ⊰identifier⊱, provided it is not less than 31 characters.

2. The quantity of information existing during translation or execution may be restricted.

3. The time permitted for interpretation of a program-run may be bounded.

1.5.1.3 Addressing Restrictions

An implementation may make restrictions on the use or addressability of "unaligned" objects (an UNALIGNED scalar or an aggregate which contains no ALIGNED scalars) in any of three contexts:

1. An argument to the ADDR built-in function.

   For example, an implementation might require that a ⊰reference⊱ to a variable that has the attribute BIT appearing in an argument to the ADDR built-in function have the attribute ALIGNED .

2. Values transmitted to or from ≪dataset≫s in RECORD I/O.

   For example, an implementation might require the ⊰reference⊱ in a FROM or INTO option, or the ≪generation≫ to be accessed from the ⊰reference⊱ in a SET option, to have the attribute ALIGNED.

3. A ≪generation≫ and a based <variable-reference> used to access the ≪generation≫ do not have identical <data-description>s and are not suitable for based string overlaying (before pruning of the <data-description> or the <variable-reference>, see Step 3.1.3 of CHECK-BASED-REFERENCE).

   For example, an implementation need not support the construction ADDR(S) –> T when S and T were declared as follows:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
DECLARE 1 S,
          2 A BIT UNALIGNED,
          2 B CHARACTER UNALIGNED,
          2 C FIXED BINARY,

        1 T BASED,
          2 A BIT UNALIGNED,
          2 B CHARACTER UNALIGNED;
```

1.5.1.4 Operating Environment

An implementation may make restrictions at the interface with its environment, for example, in the composition of external identifiers, external names, or titles.

An implementation may require appropriate extralingual information in order to execute a program in conformance with this standard.

1.5.1.5 Character Sets

This standard requires than an implementation support primary and secondary cases for letters in the character set (see Section 2.6). Typically, some permutation of the lowercase and uppercase letters in the implementation's character set are used for the primary and secondary case letters of the PL/I character set. However, this standard does not specify the mapping of the primary and secondary case letters (or any other characters in the PL/I character set) on to specific graphic characters nor does it specify their hardware representation (see Section 1.5.2, items 14 and 15). Various operations treat corresponding primary and secondary case letters as equivalent in particular contexts. In addition, a program intended for interchange must not depend on whether or not Step 6.1 of HIGH-LEVEL-PARSE is performed.

1.5.1.6 Expression Evaluation

The operations "EVALUATE-EXPRESSION", "EVALUATE-VARIABLE-REFERENCE", and "EVALUATE-TARGET-REFERENCE" define one or more strict orders of evaluation. An implementation may deviate from these strict orders in the following respects:

1. Not all of the interrupts raised by any of the strict evaluations need be raised by an implementation.

2. The order in which interrupts are raised by an implementation may differ from the order in which they are raised by strict evaluation.

3. An implementation may evaluate an <expression> (without an <and-logical> or <or-logical>), <value-reference>, <variable-reference>, or <target-reference> by evaluating its contained <expression>s, <value-reference>s, and <variable-reference>s in any order.

   Note: This does not permit the association of operators with their operands to be altered, since the association is fixed in the tree structure of an <expression> by the Translator.

4. If an implementation can determine the result produced by evaluating an <expression>, <variable-reference>, or <target-reference> without evaluating it, the implementation need not perform the evaluation. However, an implementation is required to evaluate any <procedure-function-reference>s which are evaluated in the strict order of evaluation.

   An implementation is permitted to ignore the possibility that an evaluation could raise one or more interrupts when determining whether it is required to perform the evaluation.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.5.1.7 Interrupts and Assignment

The operation "EXECUTE-ASSIGNMENT-STATEMENT" determines a strict order of evaluation. An implementation may deviate from one or more strict orders in the number and order in which it raises interrupts as described in Section 1.5.1.6, parts 1 and 2. When an implementation exceeds one or more of its IMPLEMENTATION-DEFINED limits, it may raise one or more of the following conditions: overflow, underflow, fixedoverflow, zerodivide, size, storage, area, error.

1.5.1.8 Input/Output

1. The <environment> attribute and option are provided for the specification of IMPLEMENTATION-DEFINED information concerned with the manipulation of the ⩽file-value⩾s and ⩽dataset⩾s. If present, an <environment> may affect the algorithms described in this standard by either causing (if necessary) the operation EVALUATE-EXPRESSION to be performed or affecting the correspondence between open ⩽file-value⩾s and ⩽dataset⩾s. An <environment> option to an OPEN statement, a CLOSE statement, or one of the input-output transmission statements may also affect the semantics of these statements, including the sequence in which values are transmitted, additional or alternate processing to be performed, or default positioning before or after operations. An <expression> appearing in an <environment> is evaluated at an IMPLEMENTATION-DEFINED point. If no <environment> is present, the algorithms work as given.

2. If there is more than one <single-opening> in an <open-statement>, an implementation may deviate from the strict order of execution of the RAISE-IO-CONDITION(<undefinedfile-condition>,fv) operation performed by the EXECUTE-SINGLE-OPENING operation. In particular, an implementation may defer the RAISE-IO-CONDITION operation(s) until all other operations in the statement have been performed. However, when the conditions are raised, it is required that they be raised in the same sequence implied by the strict order of execution.

3. During the execution of any input/output statement which contains a <file-option> an implementation may raise the <transmit-condition>. Continued execution of the program beyond the point where the condition is raised may be undefined depending on the use and validity of the data transmitted by the input/output statement. The <transmit-condition> is raised by performing RAISE-IO-CONDITION(<transmit-condition>,file-value,key) where file-value and key depend on the input/output statement.

4. A <get-string> MUST NOT immediately contain an <expression> that simply contains a <variable-reference> that identifies an ⩽allocation-unit⩾ that is referenced by the evaluation of the <input-specification> of the <get-string>.

5. A <put-string> MUST NOT immediately contain a <variable-reference> that identifies an ⩽allocation-unit⩾ that is referenced by the evaluation of the <output-specification> of the <put-string>.

1.5.1.9 On-units

In order to avoid defining certain "side-effects" of <on-unit>s, and to avoid overly defining the state of the machine upon entry to <on-unit>s, a program MUST satisfy the following constraints not explicitly enforced by the PL/I machine:

1. An <on-unit>, or any of its dynamic descendants, entered for the underflow condition MUST NOT allocate, free, or assign a value to any ⩽allocation-unit⩾ used in the block of interrupt, unless the <on-unit> terminates by executing a <goto-statement>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Let B be the current ⟨block-state⟩ at the time the interrupt was raised; i.e. the block of interrupt. An ⟨allocation-unit⟩ is *used* in block B if it is referenced by any operation of the PL/I machine while B is the current ⟨block-state⟩.

2. Let B be a ⟨block-state⟩; let E be an <executable-unit> executed while B is the current ⟨block-state⟩. During the execution of E, various ⟨generation⟩s are returned by invocations of EVALUATE-TARGET-REFERENCE. Let T be the set of ⟨allocation-unit⟩s designated by these ⟨generation⟩s, i.e., the targets of a given statement execution.

   During the execution of E whenever the INTERRUPT operation is invoked, all ⟨basic-value⟩s contained in any member of T are set ⟨undefined⟩. If control reaches Step 4 of the INTERRUPT operation, the original ⟨basic-value⟩s are restored, unless their containing ⟨allocation-unit⟩ was assigned one or more ⟨basic-value⟩s by the <on-unit> or its dynamic descendants. In the case of such assignment, the ⟨basic-value⟩s of the ⟨allocation-unit⟩ are set ⟨undefined⟩.

1.5.2 Implementation-defined Features

The PL/I language features listed below are termed IMPLEMENTATION-DEFINED : their specification is regarded as completing the definition of the language for a particular implementation. A brief description of each feature is given, with references in parentheses to the sections of this document where further details can be found.

1. The actual restrictions imposed, if any, under the provisions for restrictions on the size of constructs (item 1 of 1.5.1.2) or in addressing (items 1, 2 and 3 of 1.5.1.3).

2. Whether or not case mapping is performed on ⟨identifier⟩s (Step 6.1 of 4.3.2).

3. The mapping functions for <external-name>s and <identifier>s to <environment-name>s (4.8.2).

4. Determination of the ⟨dataset-list⟩ passed to the INTERPRET operation (1.7.3.3).

5. Determination of the ⟨entry-value⟩ passed to the INTERPRET operation (1.7.3.3).

6. ENVIRONMENT attribute and option syntax (2.4.11.4) and semantics (Chapter 8 and item 1 in 1.5.1.8).

7. OPTIONS attribute and option syntax (2.4.11.4) and semantics.

8. Consistency requirements for ENVIRONMENT and OPTIONS attributes in EXTERNAL declarations (4.8.1), and for OPTIONS attributes in ENTRY references (6.4.7.1.1).

9. The form of the ⟨text-name⟩ in the INCLUDE construct (2.5.6).

10. Interpretation of UNSPEC pseudo-variable (7.6.4.6).

11. Result of UNSPEC built-in function (9.5.4.81).

12. Length of string returned by TIME built-in function (9.5.4.77) and of the DATETIME built-in function (9.5.4.21).

13. Extralingual characters in data character set (2.5.5 and 2.6.2).

14. Collating sequence, hardware representations, graphic representations, and symbol names of an implementation's character set (2.6 and 9.5.4.14) and the hardware representations of the control character sequences returned by the ISOCHAR built-in function (9.5.4.38).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15. The external hardware representations of the characters specified by ⟨symbol⟩, i.e., the set of characters in which PL/I programs are written (2.6).

16. Representation of currency symbol and digit and sign symbols (9.6.2.2).

17. Default precisions of arithmetic data (4.5.5.3).

18. Maximum <number-of-digits> for each combination of <base> and <scale>(4.6.3.5, 9.2.3.2, 9.6.1.9, and elsewhere).

19. Precision of integer-type (9.2.2).

20. Accuracy and range of floating-point results of expressions and built-in functions (1.6.3.7.2, 9.2.4, and elsewhere).

21. Accuracy and range of ROUND built-in function with floating-point argument (9.5.4.61).

22. Results of numeric conversions (9.6.1.2, 9.6.1.3, and 9.6.1.6).

23. Number of digits in the exponent of a floating-point number (9.6.1.5).

24. Results of the MAXVAL, MINVAL, EPSILON, and RADIX arithmetic-enquiries (9.2.10).

25. Result of the EXPONENT built-in function and effect of the EXPONENT pseudo-variable (9.5.4.29 and 7.6.4.1).

26. Default AREA size (4.5.5.3).

27. Size of an ⟨area-value⟩ passed as a dummy argument (6.4.7.1.2).

28. Use of AREA size specification (7.3.4).

29. Information output when SNAP is specified in ON statement (6.5.3).

30. Form of comment output as standard system action (6.5.4.1).

31. Value returned by ONCODE built-in function (6.5.3).

32. Standard system action for STORAGE condition (6.5.4).

33. Standard system action for ERROR condition (6.5.4).

34. Circumstances in which TRANSMIT condition is raised (1.5.1.8).

35. Circumstances in which the KEY condition is raised (8.6.2.3, 8.7.2.2, and 8.7.5.10).

36. Circumstances under which records are written, or not written, when the RECORD condition is raised and normal return occurs, and the values of those records (8.7.3.2, and 8.7.5.9).

37. Raising of RECORD condition by WRITE and REWRITE statements (8.7.3.2 and 8.7.5.9).

38. Concrete representation of a ⟨dataset⟩ (8.2).

39. The "size" of a ⟨record⟩ (8.2.1).

40. Length of a key (8.2.1).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

41. Representation of stream dataset control items (8.2.2).

42. Determination of a ≤dataset≥ on file opening (8.6.1.3).

43. Length of file title (8.6.1.4).

44. Default LINESIZE for a STREAM OUTPUT file (8.6.1.3).

45. Default PAGESIZE for a PRINT file (8.6.1.3).

46. Default tab positions for a PRINT file (8.6.1.3).

47. Position of records in a KEYED SEQUENTIAL file (8.7.5.9).

48. Maximum <number-of-digits> used in editing relative to a <fixed-point-format> (8.8.2.5.3).

1.6 The Metalanguage

Following the introductory material in Section 1.4.4, this section now gives a more precise and careful definition of the metalanguage.

The definitive part of this document consists of:

a set of production-rules
a set of operations
constraints
attribute definitions and argument names
tables
definitions of terms

together with the section describing the relationship between an implementation and this mechanized definition.

The metalanguage in which the definition is expressed has three main notational parts, to be presented later in this section:

a notation for trees, the fundamental type of data in the metalanguage;
a notation for production-rules, which define classes of trees;
a notation for operations, which manipulate trees.

Other definitive material follows headings "Constraints:", "Attributes:", or "Arguments:".

At the point where a new term (other than a syntactic category) is defined, the term is italicized to indicate this; subsequent uses of the term are not italicized.

Tables are a part of the definition and are set off from the body of the text. Small tables may not have captions. Larger tables have a figure caption which identifies them as tables. Other figures are examples and are not part of the definition. Introductory paragraphs to syntaxes and operations, and explanatory notes (denoted by "*Note:*") are likewise without any definitive force.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.6.1 Trees

Trees are the sole type of data manipulated by the actions of the process defined by this document. All of the internal operations of the process use only trees, all of the inputs to the process from its environment are suitably constructed trees, and all interpretations of the semantics defined by this process are in terms of the tree manipulations performed by it. For uniformity even simple values, such as numbers or characters, are regarded within the process as single node trees.

In a strict mechanization of the process defined by this document, there could in fact be only a single tree used to hold the entire "state" of the process, and all of the trees discussed here would be subtrees of this single tree. Since, however, this document leaves certain informal "gaps" in its tree definitions, it is also possible to regard the process as one that operates on a set of independent trees, one for the "state" and others which are more local to particular phases of the definition.

The general abstract form of trees as employed in this document plus the technical terms used in discussing trees are defined in Section 1.6.1.1. These trees are then made more specific by discussing in Section 1.6.1.2 the basic nature of the objects used in composing tree nodes. In Section 1.6.1.3 and Section 1.6.1.4 the written notations used for individual nodes and then for whole trees are discussed. Section 1.6.1.5 deals with copies of trees.

Following this general section on trees and tree notations, Section 1.6.2 then discusses production-rules, which function in a declarative manner to specify the particular classes of trees used in this document.

1.6.1.1 Tree Definitions

A *node* is an ordered pair of objects, termed the *type* of the node, and the *unique-name* of the node.

A *tree* is a finite set of one or more nodes together with some structuring relationships among these nodes. These relationships are such that:

1. There is a specified node termed the *root-node* of the tree.

2. Excluding this root-node, the remaining nodes (termed the *subnodes* of the tree) are divided up into zero or more disjoint sets, each of which in turn forms a tree. These trees are termed the *immediate subtrees* of the defined tree.

3. There is a specified linear ordering among these immediate subtrees.

A tree, X, is said to be a *subtree* of a given tree, Y, if X is either:

1. an immediate subtree of Y, or
2. a subtree of an immediate subtree of Y.

A subtree, X, of a given tree, Y, is said to be a *simple subtree* of Y if there does not exist a tree, Z, such that:

1. Z is a subtree of Y,
2. X is a subtree of Z, and
3. the root-node of Z has the same type as either the root-node of X or the root-node of Y.

Terminology based on the word "contained" is used consistently as follows: subnodes and subtrees are said to be *contained* in the given tree, and immediate subtrees and simple subtrees are said to be *immediately contained* and *simply contained* respectively. Similarly for the word "component", i.e. subnodes and subtrees are said to be *components* of the given tree, immediate subtrees and simple subtrees are said to be *immediate components* and *simple components* respectively. The root-nodes of immediate subtrees and simple subtrees are said to be *immediate subnodes* and *simple subnodes* respectively.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The concept denoted by the term "simply contains" is used so frequently in the sequel that the words themselves are usually elided. Any relational statement between two nodes that implies containment or possession, without using explicitly any form of the words "contain" or "component", is to be interpreted as implying the simple containment relation. For example, "A simply contains B" may be expressed as "A with a B", "A has B", "the B in the A", or even just "B A". These abbreviated forms may also be compounded in a single sentence.

Two trees are said to be *equal* if they contain the same number of nodes and if:

1. when they each contain a single node, their respective types are the same, or
2. when they each contain more than one node, the respective types of their root-nodes are the same, they have the same number of immediate subtrees, and their respective immediate subtrees, taken pairwise, are equal.

A tree, X, is said to *immediately follow* a tree, Y, if they are both immediate subtrees of some tree, Z, and if X is next after Y in the linear ordering of the subtrees of Z.

A tree, X, is said to *immediately precede* a tree Y if Y immediately follows X.

A tree, X, is said to *follow* a tree, Y, if any of the following is true:

1. X immediately follows Y.
2. There is a tree Z such that X follows Z and Z follows Y.
3. There is a tree Z such that Z contains X and Z follows Y.
4. There is a tree Z such that Z contains Y and X follows Z.

A tree, X, is said to *precede* a tree, Y, if Y follows X.

Note: The above definitions do not define a linear ordering on all of the subnodes of a tree, just a partial ordering. In particular, any tree which contains another does not either precede or follow the contained tree.

The words *first* and *last* applied to any distinct set of trees have their usual sense of "have none that precede" and "have none that follow". Although these definitions are not such as to give a unique first or last tree for some sets of trees, "first" and "last" will be applied in this document only to sets such that there is a unique result.

Similarly, the *next* tree following a given tree is defined in the usual sense of "first that follows" and will also be used in contexts where it is unique.

A node, M, contained in a tree, X, is said to *correspond* to a node, N, contained in a tree, Y, if M and N occupy the same ordinal position in the ordered set of immediate subtrees of either:

1. the root-nodes of X and Y, or of
2. corresponding nodes of X and Y.

Note: Chapters 2 (see Section 2.6) and 9 (see Section 9.2.1.5) contain further definitions of "correspond" useful in certain special contexts.

1.6.1.2 Node Objects

The definition of node given in Section 1.6.1.1 leaves undefined the nature of the objects used for node types and node unique-names. The sets from which these objects are selected are limited as described in Section 1.6.1.2.1 and Section 1.6.1.2.2

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.6.1.2.1 Unique-names

The set of objects which may be employed as node unique-names plays two roles. All node unique-names are selected from this set, but in addition, some node types may be selected from this set. Any potentially infinite set of objects which are distinguishable from the other objects used as node types will suffice.

The unique-name component of a node is so called for two reasons. First, at no time do two distinct nodes have equal unique-name components. Second, no node is ever created with a unique-name component equal to that of any node which has ever been created. The unique-name component of a node does not change during the life of the node and thus serves to identify, or designate, the node (see Section 1.6.1.2.2).

1.6.1.2.2 Types

The set of objects which may be employed as node types is the union of the following disjoint sets:

1. The set of *category-names*. This is a finite set of the objects employed in production-rules. This set can be further subdivided into several logically coherent subsets, each with a definite notational convention.

2. The set of real numbers, or possibly some implementation-dependent subset of them which includes the integers. The integers are used throughout for such purposes as indices and ordinals. Real numbers (including possibly integers) are used as the values of arithmetic variables.

3. The set of unique-names, as defined in Section 1.6.1.2.1. A member of this set used as a node type is termed a *designator*. Designators are used explicitly for the purpose of uniquely picking out, or designating, nodes of a tree. A designator (or a tree containing a single designator) *designates* exactly that node which has the same object as unique-name.

   Note: This construction, together with the uniqueness rule in Section 1.6.1.2.1, means that it is possible to examine a designator and constructively determine if the potentially designated node has been in fact deleted.

As a general rule, the type of a node does not change during the life of the node. Modifications to the tree occur by removing old nodes and constructing new nodes with new types and new unique-names. The single exception to this is the replace instruction (see Section 1.6.3).

1.6.1.3 Node Notation

Throughout the metalanguage, the unique-names of nodes have only an implicit and essentially invisible function in guaranteeing unique designation and proper subtree distinction. The metalanguage discussion of nodes is always in terms of their types. Particular objects from the set of unique-names are never referred to directly, so no written notation for them is required.

The written notation used for real numbers is just ordinary decimal notation throughout. In the sequel, context is sufficient to distinguish numbers used as node types from numbers used for other purposes.

The written notation used for category-names varies, depending on the logical nature of the use of the particular category-name. These various notations and their meaning is as follows:

1. Named Categories. Names formed of lowercase letters, hyphens, and numbers, including surrounding brackets of the form "⟨" and "⟩", "<" and ">", or "≤" and "≥" are used as the denotation of some category-names. Optionally the name exclusive of the brackets may be underlined. In general, mnemonic English words or abbreviations are chosen to indicate the function of the category-name. In addition, the three types of brackets indicate whether the category-name functions primarily in the concrete, abstract, or interpretation phases of the definitional process respectively. The underlining, if present, indicates that the category-name occurs only as the type of a node that has no components.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2. PL/I Characters. The 83 characters of the PL/I language character set are used as category-names. Two denotations are used. In the great majority of situations, where no confusion is liable to arise, they are denoted by straight-forward individual character denotations. Uppercase or lowercase letters are used for primary and secondary case letters, respectively, while the quoted symbol (i.e. that which is inside the following quotes) "␣" is used for blank. In situations where confusion might arise, the concrete brackets are used around the straightforward denotations, e.g. ⟨,⟩ or ⟨ | ⟩. In some cases, primary and secondary case letters are treated as if they were equivalent, see Section 2.6.3.

3. PL/I Keywords. Certain category-names represent PL/I keywords, i.e. selected sequences of letters or numbers that have particular significance in PL/I. Two denotations are used. In the great majority of situations where no confusion is liable to arise, they are denoted by a straightforward concatenation of the denotations for the individual letters or digits that form these keywords in PL/I, written without intervening spaces on the page. Examples are the quoted symbols "ENTRY" and "FLOAT". In situations where confusion might arise, the concrete brackets are used around the straightforward denotations. For these keywords, while primary case letters are always used in the metalanguage, secondary case letters are equivalent in the actual keywords, see Section 2.7.

Nodes which have a type of either of the classes (2) or (3) above are said to possess a *concrete-representation,* which is a (non-tree) character string composed from any of the 83 PL/I language characters. For these nodes this concrete-representation is just the simple denotation of the node type, with a blank space substituted for ␣. Any possible subnodes of the category ⟨extralingual-character⟩ are assumed to have a concrete-representation, each of which is different from that of any PL/I character.

Any tree that satisfies the restriction that all of its nodes which contain no components have a type which possesses a concrete-representation, may also be said to possess a *concrete-representation.* This representation is just the character string formed by concatenating, in precedence order, the concrete-representations of these nodes.

1.6.1.4 Tree Notation

1.6.1.4.1 Enumerated-Trees

A particular tree may be completely specified by stating its root-node and describing each subnode in terms of immediate components down to the terminal nodes.

This may be expressed more concisely as an *enumerated-tree* in a notation which specifies:

1. a node type, for the root-node, optionally followed by a comma and a name by which this node can be designated (see Section 1.6.1.3),

   optionally followed by

2. a colon, the immediate components of the root-node (which may themselves be enumerated-trees, or may be names designating other trees which are to be copied) which may be enclosed in brackets ("[" and "]") denoting a component that is to be omitted if and only if its value is <absent>,

   and a semicolon.

For example, the tree which consists of a <named-constant> which immediately contains <file> and a <file-description-set>, the latter of which immediately contains <stream> and <input>, can be written as: <named-constant>,nc: <file><file-description-set>: <stream><input>;;. Indentation is often used as a visual aid, e.g.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<named-constant>,nc:
   <file>
   <file-description-set>:
      <stream>
      <input>;;.
```

Semicolons at the end of an enumerated-tree specification may be omitted immediately before a period.

Use of an enumerated-tree in the metalanguage indicates the creation of a local-tree (see Section 1.6.3.1) having the structure and node types indicated, with appropriate copies inserted of the trees to be copied (see (2) above), and with the designators corresponding to the names optionally used as in (1) above set to designate the nodes there created.

A frequently used abbreviation for a particular class of enumerated-trees is to enclose a potential PL/I concrete-representation in double quotes. This is an abbreviation for that tree rooted in ⟨symbol-list⟩ which has this enclosed string as its concrete-representation.

1.6.1.4.2 Forms

Patterns to be searched for in trees may be indicated in the metalanguage by a notation which is the same as that for enumerated-trees, except that the names of trees to be copied may not be included.

Use of such a notation in the metalanguage is always preceded by the word *form*. Its use indicates that a search for, or test of conformance to, the pattern is to be made, yielding true or false, and that the designators corresponding to the names used as in (1) of Section 1.6.1.4.1 are to be set to designate the nodes corresponding to them if and only if the search, or test, returns true. Use of brackets in a form indicates the bracketed components may be either present or absent.

1.6.1.5 Tree Copies

A *copy* of a given tree is constructed as follows:

1. Construct a tree which is equal to the given tree.

2. For each designator node X in the given tree which designates a node Y also in the given tree, change the constructed node which corresponds to X so that it designates the constructed node which corresponds to Y.

1.6.2 Production Rules

The trees actually employed in this document are a limited subset of all the possible trees that could be formed according to the definitions given in Section 1.6.1. Production rules serve as the declarative portion of the metalanguage and do so by specifying restrictions on the forms assumed by the trees used in the definitional process of this document.

1.6.2.1 Production Rules and Syntaxes

A *production-rule* is written with an optional label formed of capital letters and digits, and consists of a category-name, followed by the quoted symbol "::=" and then followed by either a syntactic-expression (see Section 1.6.2.3) or, in a few instances, an English language phrase. Such a production-rule is termed a *defining production-rule* for that category-name written before the "::=". Within this document, there is at most one defining production-rule for any given category-name.

The basic function of a production-rule is to define a set of possibilities for the number, type(s), and order of the immediate subnodes of a node whose type is the defined category-name. This is done by applying the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

definition of a *subnode-type sequence with respect to a syntactic-expression,* given in Section 1.6.2.4.2, to the syntactic-expression of the defining production-rule.

Production-rules are augmented in their function of specifying immediate subnodes by a notational convention used for creating lists of repetitive immediate subnodes. This convention applies to the bracketed category-names whose denotation, exclusive of the brackets, terminates in the quoted symbols "-list" or "-commalist", and it substitutes for the explicit appearance of a defining production-rule for such category-names (i.e. such category-names have no defining production-rule written in this document).

A *syntax* is any set of production-rules. For example, the set of all of the production-rules in this document is a syntax. Four (disjoint) subsets of the production-rules in this document have enough logical coherence that they have been given names, i.e. the High-level Concrete Syntax, the Low-level Concrete Syntax, the Abstract Syntax, and the Machine-state Syntax. In order to distinguish these syntaxes, the production-rules comprising them have been given numbered labels starting with the quoted symbols "CH", "CL", "A", and "M" respectively. The unlabeled production-rules of this document do not belong to any of these syntaxes.

If a defining production-rule for a category-name occurs in a syntax, then that category-name is said to be *non-terminal with respect to* that syntax. Any category-name that occurs somewhere within the syntactic-expressions of the production-rules of the syntax, but has no defining production-rule in the syntax, is said to be:

1. *non-terminal with respect to* that syntax if its denotation exclusive of the brackets, ends with "-list", "-commalist", or "-designator", and
2. *terminal with respect to* that syntax otherwise.

A category-name that is non-terminal with respect to the syntax composed of all the production-rules occurring in this document is said to be just *non-terminal;* similarly for *terminal.*

The Abstract Syntax additionally allows constraints to be specified for certain category-names. The constraint, written in parentheses after the relevant category-name, is applied by the Translator, or during the interpretation phase, but has no effect on the constitution of a tree specified by the syntax.

1.6.2.2 Complete and Partial Trees

Given a syntax and a category-name, a *complete tree with respect to the given syntax and category-name* is any tree which can be obtained by

1. starting from a node whose type is the given category-name;
2. marking that node "incomplete";
3. choosing a node marked "incomplete" and completing it as follows:
   a. attaching immediate subnodes to it in the number, type(s), and order specified by applying the definition of a *subnode-type sequence with respect to a syntax and category-name,* given in Section 1.6.2.4.1, to the given syntax and that category-name which is the type of the chosen node;
   b. marking as "incomplete" each attached immediate subnode that has a type which is a category-name; and
   c. removing the "incomplete" mark from the chosen node;
4. repeating (3) as long as some node marked "incomplete" remains in the tree being constructed.

A complete tree with respect to the syntax composed of all the production-rules occurring in this document is said to be just a *complete tree.*

A *partial tree* is any tree which is not a complete tree but which can be obtained by deleting some nodes from some complete tree.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The production-rule

<bound-pair> ::= <lower-bound> <upper-bound> | <asterisk>

defines two possibilities, which may be written as (a) or drawn as (b)

(a)
```
<bound-pair>:                    <bound-pair>:
  <lower-bound>        or          <asterisk>;
  <upper-bound>;
```

(b)
```
<bound-pair>                or   <bound-pair>
     |                                |
  ┌──┴──┐                            |
<lower-bound> <upper-bound>          <asterisk>
```

A node whose type is <identifier-list> may have any non-zero number of immediate subnodes of type <identifier>, i.e.

```
<identifier-list>:          or  <identifier-list>:          or  <identifier-list>:
  <identifier>;                   <identifier>                    <identifier>
                                  <identifier>;                   <identifier>
                                                                  <identifier>;
```

and so on.

A node whose type is ⟨subscript-commalist⟩ may have any non-zero number of ⟨subscript⟩ immediate subnodes, but with nodes whose type is the PL/I character ⟨,⟩ interspersed between adjacent ones, i.e.

```
⟨subscript-commalist⟩:      or  ⟨subscript-commalist⟩:      or  ⟨subscript-commalist⟩:
  ⟨subscript⟩;                    ⟨subscript⟩                     ⟨subscript⟩
                                  ⟨,⟩                             ⟨,⟩
                                  ⟨subscript⟩;                    ⟨subscript⟩
                                                                  ⟨,⟩
                                                                  ⟨subscript⟩;
```

and so on.

The production-rule

<repeat-option> ::= <expression> (scalar)

specifies that only <expression>s yielding scalar values (i.e. not aggregate values) are valid immediate subnodes of a <repeat-option>.

Figure 3. Examples of Syntax.

The trees utilized by the definition process of this document are only complete trees or partial trees. Other possible forms of trees are never utilized. Furthermore, it is the usual case that complete trees are utilized, or at least utilized at the interfaces between the various operations of the definition. Although there are a few

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

specific exceptions, it is a general rule that partial trees occur only in a very local context in the process of building up a complete tree. Several of the "instructions" (see Section 1.6.3.4) of the metalanguage are in fact designed to assist in the process of building complete trees.

Most of the complete trees used in this document are constructed explicitly by particular algorithms, or implicitly by displaying an enumerated tree. In a few cases (notably the PARSE, HIGH-LEVEL-PARSE, and LOW-LEVEL-PARSE operations of Chapter 4), only the existence of complete trees satisfying certain conditions is required, and no algorithm for constructing them is even mentioned.

1.6.2.3 Syntactic-expressions and Syntactic-units

Given a syntax, a *syntactic-expression* is defined to be either a single syntactic-unit, or several syntactic-units any of the adjacent pairs of which is possibly separated by a " | " or a "•". The symbols are called the or-symbol and the bullet respectively.

Given a syntax, a *syntactic-unit* is defined to be one of the following:

a single category-name,
a syntactic-expression enclosed in the brackets "[" and "]", or
a syntactic-expression enclosed in the braces "{" and "}".

1.6.2.4 Use of the Production Rules

In any complete tree with respect to a syntax s and a category-name, the immediate subnodes of each node n whose type is a category-name c must satisfy certain requirements. The requirements dictate the number, type(s) and order of the immediate subnodes. The requirements are determined by c and the production-rules of s, augmented by a few notational conventions. To obtain those requirements, the production-rules and notational conventions are interpreted according to the definitions in Sections 1.6.2.4.1 and 1.6.2.4.2. In some cases they uniquely prescribe the number, type(s) and order of the immediate subnodes of n, as illustrated by the fourth example shown in Figure 3 on page 27. In other cases, as illustrated by the first three examples shown there, they present more than one choice (possibly infinitely many choices) for the number, type(s) and order of the immediate subnodes of n.

In any complete tree, each node whose type is an object other that a category-name is constrained to have no immediate subnodes. This constraint follows from the definition of a complete tree.

During the construction (by any algorithm) of a complete tree with respect to a syntax s and a category-name, the following two questions arise repeatedly: Given, in the tree being constructed, a node n whose type is a category-name c, how many immediate subnodes may n have, and what may their types be? To answer these questions, one applies the definition in Section 1.6.2.4.1 to s and c to obtain one or more (possibly empty) sequences of node types, each termed a *subnode-type sequence with respect to the given syntax and category-name.* If nt[1],...,nt[nsn], nsn$\geq$0, is any such sequence of node types, then n may have nsn immediate subnodes of types nt[1],...,nt[nsn], in order.

The first three cases of the definition in Section 1.6.2.4.1 deal with the notational conventions augmenting the production-rules. Case 4.1 of the definition, which deals with production-rules containing a syntactic-expression, appeals to a second definition (shown in Section 1.6.2.4.2). In this case, the answers to the questions posed above are provided indirectly by the second definition. Thus, given a node n whose type is a category-name defined by a production-rule containing a syntactic-expression se, one applies the definition of Section 1.6.2.4.2 to se to obtain one or more (possibly empty) sequences of node types, each termed a *subnode-type sequence with respect to the given syntactic-expression.* If nt[1],...,nt[nsn], nsn$\geq$0, is any such sequence of node types, then n may have nsn immediate subnodes of types nt[1],...,nt[nsn], in order.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.6.2.4.1 Subnode-type Sequence with respect to a Syntax and Category-name

A subnode-type sequence with respect to a syntax s and a category-name c is defined as follows:

Case 1. The denotation of c, excluding any terminating bracket, ends with "-list".

A subnode-type sequence with respect to s and c consists of a sequence of any non-zero number of instances of the category-name whose denotation is obtained by deleting the "-list" from the denotation of c.

Case 2. The denotation of c, excluding any terminating bracket, ends with "-commalist".

A subnode-type sequence with respect to s and c consists of a sequence which:

1. contains any non-zero number, k, of instances of the category-name whose denotation is obtained by deleting the "-commalist" from the denotation of c, and which
2. contains k−1 instances of the category-name ⟨,⟩ and which
3. is arranged so that no two instances of the same category-name are adjacent.

Case 3. The denotation of c, excluding any terminating bracket, ends with "-designator".

Let cdn be the category-name whose denotation is obtained by deleting the "-designator" from the denotation of c. Let dn be any existing node whose type is cdn. Let u be the unique-name of dn. Then, a subnode-type sequence with respect to s and c is a sequence consisting of a single instance of u.

Case 4. c is defined by a production-rule in s.

Case 4.1. The defining production-rule contains a syntactic-expression se written following the "::=".

A subnode-type sequence with respect to s and c is a subnode-type sequence with respect to se (see Section 1.6.2.4.2).

Case 4.2. (Otherwise).

A subnode-type sequence with respect to s and c is implicit in the English language remark written following the "::=" in the defining production-rule (and may be IMPLEMENTATION-DEFINED).

Case 5. (Otherwise).

c is necessarily terminal with respect to s. A subnode-type sequence with respect to s and c is the empty sequence.

1.6.2.4.2 Subnode-type Sequence with respect to a Syntactic-expression

A subnode-type sequence with respect to a syntactic-expression se is defined as follows:

Case 1. se is a syntactic-unit.

Case 1.1. The syntactic-unit is a single category-name.

A subnode-type sequence with respect to se is a sequence consisting of a single instance of that category-name.

Case 1.2. The syntactic-unit is a syntactic-expression ise enclosed in the brackets "[" and "]".

A subnode-type sequence with respect to se is either a subnode-type sequence with respect to ise or the empty sequence.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1.3. The syntactic-unit is a syntactic-expression ise enclosed in the brackets "{" and "}".

A subnode-type sequence with respect to se is a subnode-type sequence with respect to ise.

Case 2. se is a sequence of two or more syntactic-units possibly separated by a " | " or a "•".

Case 2.1. An or-symbol occurs as at least one such separator.

Consider all or-symbols occurring thus in se to partition it into a sequence ise[1],...,ise[k], $k \geq 2$, of inner syntactic-expressions. A subnode-type sequence with respect to se is a subnode-type sequence with respect to any one of the ise[i], i=1,...,k, chosen arbitrarily.

Case 2.2. A bullet occurs as a separator and an or-symbol does not.

Consider all bullets occurring thus in se to partition it into a sequence ise[1],...,ise[k], $k \geq 2$, of inner syntactic-expressions. A subnode-type sequence with respect to se is a subnode-type sequence with respect to a syntactic-expression obtained from se by rearranging all the ise[i], i=1,...,k, in an arbitrary order and omitting these bullets.

Case 2.3. (Otherwise).

se is necessarily a sequence of syntactic-units optionally separated by blanks. Consider each such syntactic-unit as a syntactic-expression, thereby obtaining from se a sequence ise[1],...,ise[k], $k \geq 2$, of inner syntactic-expressions. A subnode-type sequence with respect to se is the concatenation, in order, of subnode-type sequences with respect to each of the ise[i], i=1,...,k. (The sequence formed by concatenating sequences consists of the elements of the first sequence, in order, followed by the elements of the second sequence, in order, and so on.)

1.6.3 Operations

The procedural part of the metalanguage provides for the writing of algorithms termed *operations.* These are expressed in a semi-formal programming language which uses the grammatical flexibility of ordinary English prose, while at the same time attaching precise meaning to certain words and phrases, in order that the flow of control and the tree manipulations in the operations be well defined. Completely formal notation is used to describe trees in accordance with their syntactic definitions.

1.6.3.1 Nature of an Operation

An operation, applied to zero or more *operands,* may be performed by the processor (see Section 1.6.4), with the effect of:

1. changing the ⟨machine-state⟩, or
2. changing an operand, or
3. returning a *result,* or
4. any combination of (1), (2), and (3).

Within an operation either *operand-names* or *local-variable-names* may be used for accomplishing this effect. These names serve as designators of trees, which may be either portions of the ⟨machine-state⟩, or *local-trees* created within this or another operation.

When an operation is performed, its operand-names are set to designate the operand trees it has been passed. It then has the following data available to it:

1. The whole ⟨machine-state⟩, which is directly accessible for inspection or change at any time.
2. The operands which it has been passed, which can be inspected or changed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3. Local-trees local to itself, which it can freely construct, inspect, or modify. These trees are deleted when the operation terminates.

The operation may also apply any operation defined in this document (including itself) to any operands which it may select from among the trees available to it.

Upon completion of its actions, an operation may return a result, which then becomes available to whichever operation applied the given operation to its operands. As with operands, this result may be selected from among the trees available to the operation. In the case that (a portion of) a local-tree local to this operation is selected, this tree is not deleted, but is copied to become a local-tree local to the applying operation. (See Section 1.6.3.4 on "perform".)

1.6.3.2 Nondeterministic Operations

The phrases "optionally" and "in any order" are used in some operation descriptions. They indicate that the processor is to make a choice each time that part of the operation is executed. In general, then, the conceptual PL/I machine defines a set of possible interpretations for a program.

1.6.3.3 Format of an Operation

The written description of an operation has a format consisting of a heading and a body.

The heading may have three parts:

1. There is always a specification of the form "Operation:" followed by the underlined name of the operation, optionally followed by a parenthesized list of the names used to refer to the operands passed to the operation, the names being separated by commas.

2. For an operation which has operands, the word "where" then precedes a description of the type(s) of tree to which each operand name may refer. Brackets around a node type indicate that the operand is optional, which is an abbreviation for stating that it may alternatively have the value ⟪absent⟫.

3. For an operation which returns a result, a final part of the heading is of the form "result:" followed by a description of the type(s) of tree which may be returned.

Operation: EVALUATE-IN-OPTION(al,vr)

Where: al is an <allocation>,
vr is a <variable-reference>.

Result: a ⟪generation⟫ or ⟪fail⟫.

Figure 4. An Example of an Operation Returning a Result.

The body of an operation consists of either a Step-list or a Case-set. Each Step or Case is a numbered section containing written instruction descriptions of arbitrary complexity, and may itself contain a Step-list or Case-set, numbered with an additional index position and indented to indicate this containment.

Steps are normally performed sequentially.

Each *Case* consists of a condition part followed by an executable part. Within each case-set, the set of condition parts is such that, at any execution point, exactly one of the condition parts is satisfied. (The

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

conventional condition part "(Otherwise)" is satisfied whenever none of the other condition parts of the case-set is satisfied.) The execution of a case-set consists of executing the executable part of the one case whose condition part is then satisfied.

The normal sequence for Steps and Cases may be modified by explicit instructions using such terms as "go to" or "perform" which are defined in Section 1.6.3.4. The terminology for selecting some actions, either in a defined left-to-right order or in an unspecified order, is "in left-to-right order" and "in any order" respectively. When selection is between two options, "in either order" may be used instead of "in any order". Optional selection as to whether an action will be carried out is indicated by words such as "optionally perform".

Operation: EVERY-BIF(rdd,x)

Step 1. Perform EVALUATE-EXPRESSION(x) to obtain an ⟨aggregate-value⟩,u.

Step 2. In any order, convert each scalar-element of u to ⟨bit⟩, to obtain v.

Step 3.

Case 3.1. Every scalar-element of v contains ⟨null-bit-string⟩.

Let r be ⟨zero-bit⟩.

Case 3.2. At least one scalar-element of v which does not contain ⟨null-bit-string⟩ has a ⟨bit-string-value⟩ with at least one ⟨bit-value⟩ containing ⟨zero-bit⟩.

Let r be ⟨zero-bit⟩.

Case 3.3. (Otherwise).

Let r be ⟨one-bit⟩.

Step 4. Return an ⟨aggregate-value⟩ containing a ⟨bit-string-value⟩ containing r.

Figure 5. An Example of an Operation Containing a Step-list and a Case-set.

1.6.3.4 Instructions

An *instruction* is a specification of some action involving the creation, destruction, inspection, or modification of some tree(s), or causing some departure to be made from the normal sequential flow of control through an operation.

Operand-names and local-variable-names strictly denote designators of trees rather than tree values themselves. However, apart from their use in a context where they acquire designator values, are passed as operands, or are returned as results, references to these names are always taken to be an abbreviation for references to the tree designated by the strict value.

The "let" instruction is used to indicate that the named variable is henceforth to reference the specified tree, which may be an existing tree or one newly created, e.g. by use of the enumerated-tree notation or by copying an existing tree. Use of a name for a tree following some description of the root-node and a comma, e.g. in the enumerated-tree notation, is equivalent to use of a "let" instruction. No change to any tree previously designated by the named variable or operand occurs as a result of the "let" instruction.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In contrast, the "replace" instruction is used to indicate that the tree referenced by the named variable or operand is to be replaced with the specified tree. The replacement occurs exactly at the root of the referenced tree, and the named variable or operand henceforth references the replacement.

The "attach" instruction, as in "attach b to c", indicates that b is to be attached to c as a component of c. If b can be an immediate component of c, then it is attached as an immediate component. Otherwise, there will be a unique way that a node of type b can be a simple component of c, and exactly the minimal necessary nodes which are both contained by c and also contain b are created so as to attach b as a simple component.

The "append" instruction, as in "append b to c", also specifies an attachment process. There are three cases.

Case 1. c is a b-list or a b-commalist.

In this case b is attached as the last immediate subtree of c. c may be any existing tree or, if it is uniquely specified, may be non-existent. This latter case causes the construction of the (single) node c. In order to make the specification of the potentially missing node c unique, the notation illustrated by "append b to c in d" can be employed to indicate that c may be missing and is to be constructed as a simple component of d.

Case 2. b is a ⪡,⪢ and c is a d-commalist for some d.

In this case b is attached to the last immediate subtree of c.

Note: This action is only performed just prior to appending a d to c, and only when c already exists and contains one or more subtrees.

Case 3. Otherwise.

In this case, "append b to c" is equivalent to "append b to the b-list in c" or "append b to the b-commalist in c", as required.

The "delete" instruction, as in "delete b" indicates that b is to be removed from its containing tree (and discarded). In addition, if b is a mandatory component of the tree which immediately contained b, say c, then the "delete" instruction is applied to c (i.e., c is discarded and the process continues with the tree which immediately contained c).

The "perform" instruction either calls another operation, possibly passing operands, possibly receiving a result, or calls for the execution of some Steps with an operation out of the normal sequence. In calling other operations, references to trees specified by the argument list are passed to the named operation as operands (any missing arguments are given the value ⪡absent⪢ in the called operation). If the operation returns a result, then the term "to obtain" is used to indicate the obtained result. When other steps in an operation are performed, control returns to the instruction following the "perform" instruction.

The "go to" instruction indicates that the normal sequence of control is broken and that the named Step is to be executed next.

The "terminate" instruction indicates that the execution of the current operation is to be terminated and control is to be returned to the calling operation. If control reaches the end of an operation that does not return a result, an implicit "terminate" instruction is assumed.

The "return" instruction indicates that the current operation is to be terminated with a reference to a specified tree as the result.

The "if" instruction indicates that if the specified condition is true then the instruction list following the "then" is executed and the instruction list (if any) following the "otherwise" is skipped. If the condition is false the instruction list following the "then" is skipped and, if there is one, the instruction list following the "otherwise" is executed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The "for each" instruction specifies actions that are to be carried out once for each member of a set of objects, in some sequence which may be in any order or in some specified order.

The "must" instruction specifies a test to be performed. If the test is not satisfied, the program (when combined with the particular initial entry-point and datasets if the INTERPRETATION-PHASE has begun) is rejected by this standard, and the conceptual PL/I machine stops. This is the only sense in which "must" and "must not" are used in operation descriptions.

1.6.3.5 Convert

CONVERT is an exception to the general rules for naming and performing (see Section 1.6.3.4). Use of this operation is generally specified in an informal style, e.g. "convert the value of the <expression>,x to integer-type". Details are given in 9.6.1.1.

1.6.3.6 Additional Notational Conventions

Common mathematical symbols are used with their usual meaning. In addition, the following notational conventions are used:

*	denotes multiplication;
/	denotes division;
b^{n-1}	denotes exponentiation;
Σ	denotes iterated addition (summation); the result is taken as zero if the iteration range is empty;
Π	denotes iterated multiplication (product); the result is taken as one if the iteration range is empty;
[ ]	denotes subscripting in the metalanguage as a notational convenience used with local names;
π	denotes the mathematical constant of that name;
e	denotes the mathematical constant of that name;
ceil(x)	denotes the smallest integer larger than or equal to x;
floor(x)	denotes the largest integer smaller than or equal to x;
min(x,y)	denotes the value of x if x<y, otherwise the value of y;
max(x,y)	denotes the value of x if x>y, otherwise the value of y;
log(x)	denotes the natural logarithm of x;
| x |	denotes the absolute value of x.

1.6.3.7 Arithmetic

1.6.3.7.1 Properties of Arithmetic in the Metalanguage

In the metalanguage, an arithmetic expression denotes the exact mathematical value.

In general the distinction between numbers and trees containing them is ignored. For example, if z is a tree of the form

```
⟨basic-value⟩:
   ⟨real-value⟩:
      ⟨real-number⟩,x;;
```

then the arithmetic expression "z+1" denotes the real number x+1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.6.3.7.2 Properties of Arithmetic in PL/I

Because digital computers are inherently limited by the finite range and precision with which they can represent real numbers, and because their hardware and software algorithms for operating on real numbers to produce other real numbers are sometimes imperfect, this standard does not define the result of evaluating an arithmetic <expression> to be an exact mathematical value, in general.

The situations in which an arithmetic result may be approximated by the PL/I machine, or in which exceptional conditions may occur, are defined in such operations as ARITHMETIC-RESULT and CONVERT, and in the operations performed by them.

In the case of an arithmetic <expression> that yields a FIXED result, the language features require, and typical hardware capabilities permit, exact results much of the time. Nevertheless, approximations are allowed in certain instances, such as in the result of the division operator and when numbers with fractional parts undergo base conversion.

In the case of an arithmetic <expression> that yields a FLOAT result, this standard specifies only that the result will be an approximation to the exact mathematical value, or that an OVERFLOW or UNDERFLOW condition may occur. Furthermore, for some arithmetic operations, enquiries, built-in functions, pseudo-variables, and conversions this approximate result is constrained to lie within a bounded range of possible values, and the situations where the OVERFLOW or UNDERFLOW conditions may occur are likewise constrained. These constraints are adapted from a model of floating-point arithmetic published by W. S. Brown in an article entitled "A Simple but Realistic Model of Floating-Point Computation", which appeared in the December 1981 issue of the *ACM Transactions on Mathematical Software*. The model associates four IMPLEMENTATION-DEFINED floating-point *model-parameters* with each <data-type> that has <float>. The constraints on floating-point results subject to the model are then determined by the values of the operands, the floating-point model-parameters corresponding to the <data-type>s of the operands, and the floating-point model-parameters corresponding to the <data-type> of the result. An implementation may produce any result that satisfies the constraints.

The four floating-point model-parameters are known as the *model-base,* the *model-precision,* the *model-minimum-exponent,* and the *model-maximum-exponent.* They are used in this standard to define underflow and overflow thresholds; the value of the EXPONENT built-in function and the effect of the EXPONENT pseudo-variable; the MAXVAL, MINVAL , EPSILON, and RADIX enquiries; and, through the concept of *model-numbers* (see Section 9.2.4.2.5), the accuracy of floating-point results.

To be in conformance with this standard, an implementation is required to satisfy the following constraints for each <data-type> that has <float>.

Let dt be a

```
<data-type>:
   <computational-type>:
      <arithmetic>:
         <base>,ba
         <scale>:
            <float>;
         <precision>,p.
```

Let b=2 or b=10 according as ba has <binary> or <decimal>. Let b', p', emin and emax be the model-base, model-precision, model-minimum-exponent, and model-maximum-exponent, respectively, corresponding to dt. Then the following relations are required to hold:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1. a. if b' > b: p' ≥ 1 + ceil((p − 1)/$\log_b$(b'));
   b. if b' = b: p' ≥ p;
   c. if b' < b: p' ≥ ceil(p/$\log_b$(b'));
2. emin ≤ 2 − 2*p';
3. emax ≥ 2*p' − 1;
4. 2*emin + emax ≤ 3 − p';
5. emin + 2*emax ≥ p' + 1.

where "$\log_b$" denotes the logarithm of the argument with respect to the base b.

On the accuracy and range of floating-point results, these constraints impose lower bounds that are both weak enough to be realistic and strong enough to be useful; they also require the range of floating-point numbers to be reasonably balanced.

The accuracy and range of floating-point results in excess of the minimum required by these conformance criteria are affected by such implementation choices as data representations, hardware capabilities, computational algorithms, and so forth. An implementation is never obliged, by this standard, to favor one such choice over another in order to maximize the accuracy or range of floating-point results. As a consequence of the way the flexibilities of the implementation are preserved in this standard, it is not possible for the various operations and definitions in the body of this standard to express the need for an implementation to maximize the accuracy and range of floating-point results, as reflected in the model-parameters, when doing so does *not* affect the choice of data representations, hardware capabilities, computational algorithms, and the like. Informally, an implementation is required to use the highest value of model-precision, the highest value of model-maximum-exponent, and the lowest value of model-minimum-exponent consistent with the implementation choices it has made prior to considering this requirement.

1.6.4 The Processor

The *processor* is the active agent capable of performing various actions on the ⟨machine-state⟩ tree. These actions are carried out as directed by the written algorithm which comprises the PL/I definition.

A portion of the ⟨machine-state⟩ tree, either the ⟨control-state⟩ or a ⟨statement-control⟩, holds the information which controls the processor. An ⟨operation⟩ is *known* by the processor if it is a simple component of the ⟨control-state⟩ or a ⟨statement-control⟩. There is at most one active operation. An ⟨operation⟩ is *active* if:

1. it is the last member of its immediately containing ⟨operation-list⟩, and
2. either:
   a. the ⟨operation⟩ is simply contained in an ⟨operation-list⟩ which is simply contained in a ⟨statement-control⟩ which is simply contained in a ⟨block-state⟩ that is the last member of its immediately containing ⟨block-state-list⟩, or
   b. there are no ⟨statement-control⟩s contained in the ⟨machine-state⟩, and the operation is simply contained in an ⟨operation-list⟩ which is simply contained in the ⟨control-state⟩.

(See Section 1.7.1 and Section 5.2 for the definitions of these category-names.)

The processor carries out actions as follows:

> Whenever there exists a known active ⟨operation⟩, the processor carries out the actions specified by the written description of the corresponding operation.
> Whenever there exists no known active ⟨operation⟩, the processor does nothing.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.6.5 Mechanization of the Metalanguage

A deeper understanding of the metalanguage may be obtained by considering how its formal mechanization may be carried out as an extension of the ⟨machine-state⟩. Each ⟨operation⟩ in an ⟨operation-list⟩ may be given a subtree containing the name of the operation, the names and designator values of its operands, the names and designator values of its local variables, the local-trees constructed during the performance of this operation, and control information indicating which Step or Case is currently being executed, which members of an iterative "for each" have still to be performed, and so on. When a "perform" instruction causes a new operation to be invoked, a new ⟨operation⟩ tree will be appended to the ⟨operation-list⟩ and activity in the present operation will be suspended. On return from an operation, any local-tree returned by it as a result is copied back as a subnode of the preceding ⟨operation⟩ tree, and the ⟨operation⟩ tree for the terminating operation is deleted together with all its local information. Activity in the invoking operation is then resumed from the point at which it was suspended.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

During the TRANSLATION-PHASE, the only ≺operation-list≻ is the one in the ≺control-state≻. During most of the INTERPRETATION-PHASE, the ≺operation≻s in the ≺control-state≻ will be dormant while there is an active ≺operation≻ in a ≺statement-control≻ as shown below.

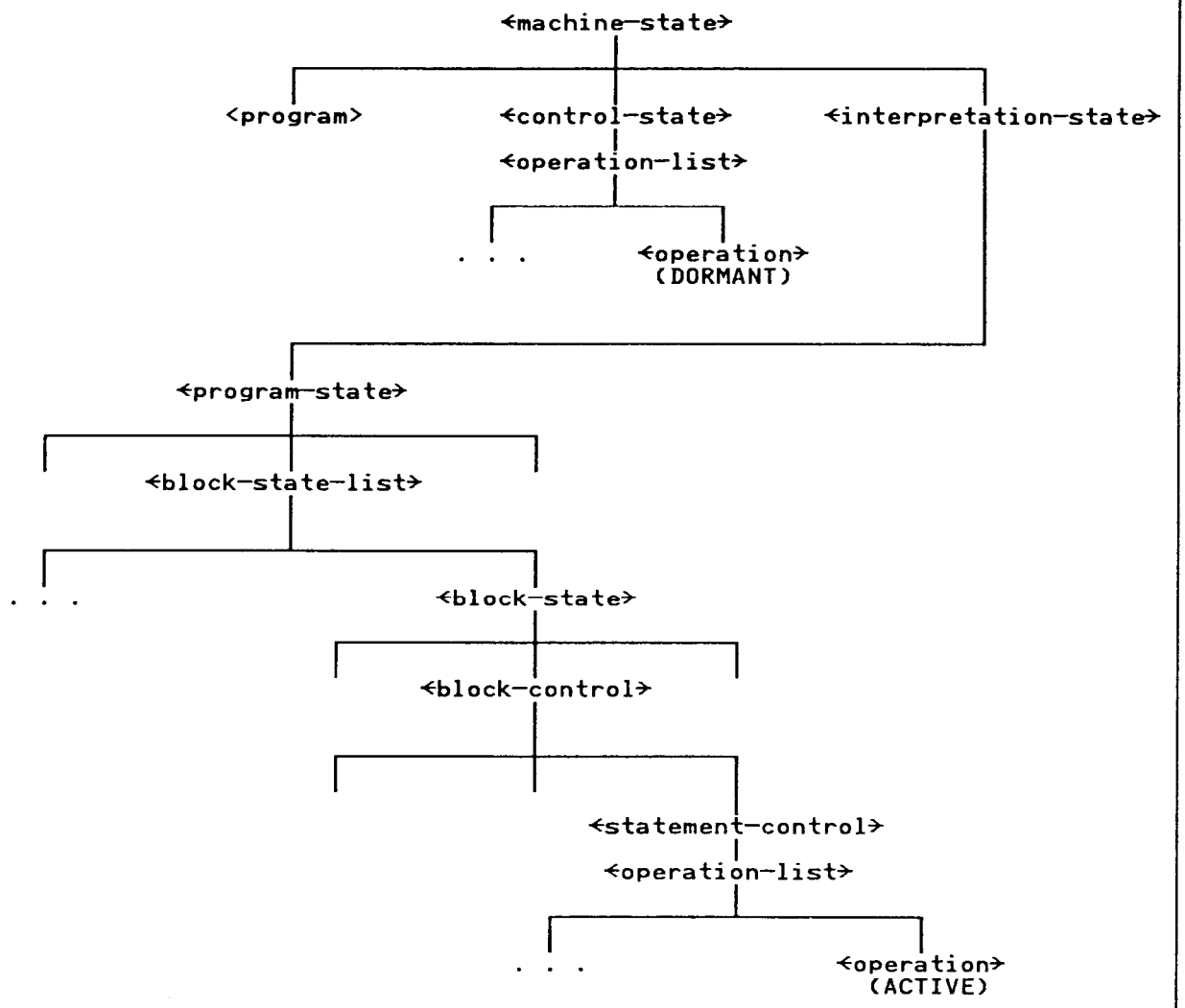


Figure 6. Example Showing ≺operation-list≻s of a ≺machine-state≻.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.7 Initialization of the Machine-state

1.7.1 The Machine-state

M1. ⟨machine-state⟩ ::= <program> ⟨control-state⟩[⟨translation-state⟩ | ⟨interpretation-state⟩]

M2. ⟨control-state⟩ ::= ⟨operation-list⟩

M3. ⟨translation-state⟩ ::= [{external-unit}]

M4. {external-unit} ::= [{declaration-commalist}] {{procedure} | {package}}

M5. ⟨operation⟩ ::= The exact structure of ⟨operation⟩ is left unformalized and unspecified. It has adequate structure and capacity to represent the carrying out of the actions of an operation. This includes designating the particular operation and the current position within it, holding the operands given to the operation, and holding the values of any variables used by the operation (see Section 1.6.5).

The definitions of {declaration}, {package}, and {procedure} are given in Chapter 2; the definition of <program> is given in Chapter 3; the definition of ⟨interpretation-state⟩ is given in Chapter 5.

1.7.2 Initialization

The PL/I definition process begins by creating an initial ⟨machine-state⟩ tree, consisting of:

⟨machine-state⟩:
  <program>
  ⟨control-state⟩:
    ⟨operation-list⟩:
      ⟨operation⟩ for DEFINE-PROGRAM.

The processor then performs the ⟨operation⟩ for DEFINE-PROGRAM. When DEFINE-PROGRAM stops, the definition algorithm is successfully completed.

1.7.3 The Top-level Operations

1.7.3.1 Define-program

Operation: DEFINE-PROGRAM

Step 1. Perform TRANSLATION-PHASE.

Step 2. Perform INTERPRETATION-PHASE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1.7.3.2 Translation-phase

Operation: TRANSLATION-PHASE

Step 1. Append ⋖translation-state⋗ to the ⋖machine-state⋗.

Step 2.

Step 2.1. Obtain, from a source outside this definition, a sequence of characters composing a putative PL/I external procedure, constructed in the form of a ⦃symbol-list⦄,sl.

Step 2.2. Perform TRANSLATE(sl) to obtain an <external-unit>,eu. Append eu to the <external-unit-list> in the <program>.

Step 2.3. Optionally go to Step 2.

Step 3. Perform VALIDATE-PROGRAM.

Step 4. Delete the ⋖translation-state⋗.

1.7.3.3 Interpretation-phase

Operation: INTERPRETATION-PHASE

Step 1. Obtain, from a source outside this definition, the following items:

1. A collection of information to be used for input/output, constructed in the form of a suitable ⋖dataset-list⋗,dl.

2. A designation, as the first to be activated, of one of the <entry-point>s of a <procedure> simple component of <program>, constructed in the form of a suitable ⋖entry-value⋗,ev. Such an <entry-point>,ep MUST satisfy the following constraints:

   a. ep MUST exist.
   b. ep MUST NOT be contained in a <package>.
   c. ep MUST NOT have <parameter-name-list> or <returns-descriptor> components.

Step 2. Perform INTERPRET(dl,ev). (See Section 5.4.1).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2. Concrete Syntax

2.1 Introduction

The Concrete Syntax of PL/I is specified mainly by means of production-rules using the notation defined in Chapter 1. The first such rule defines a ⟨translation-unit⟩. A ⟨translation-unit⟩ is either a ⟨package⟩ or ⟨procedure⟩. The second and third rules define ⟨package⟩s and ⟨procedure⟩s, and subsequent rules define the permitted forms of ⟨package⟩s or ⟨procedure⟩s and their components in increasingly fine detail, until every component is ultimately described in terms of sequences of characters of the language character set.

2.2 The Intent of this Definition

As the first stage of translation (Chapter 4), any given sequence of symbols is parsed to determine whether that sequence indeed represents a ⟨translation-unit⟩ valid according to a set of rules known in this document as the "Concrete Syntax".

2.2.1 Concrete and Abstract Syntaxes

This formal Concrete Syntax is permissive in the sense that some of the constructs permitted are not actually valid ⟨translation-unit⟩s. Thus, for example, the sequence of symbols "DECLARE X FLOAT FIXED;" is a syntactically correct construct that may be parsed as a ⟨declare-statement⟩. Errors of this sort will be detected later in the translation, because of a failure to satisfy the Abstract Syntax (Chapter 3).

In addition, some constructs in the Concrete Syntax (e.g. ⟨data-attribute⟩: STRUCTURE;) are definitional artifices and are not allowed in translation-units. See the operation VALIDATE-TRANSLATION-UNIT in Section 4.4.

2.3 Organization of the Concrete Syntax

The rules of the Concrete Syntax, which are context-free, are arranged in two levels, so that a context-dependent feature of this grammar, namely the presence of blanks and comments, may be resolved at the interface between the levels.

The two levels of syntax correspond to the two levels of the PARSE algorithm described in Chapter 4.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.4 The High-level Syntax of PL/I

2.4.1 Translation Unit

CH1. ⟨translation-unit⟩ ::= ⟨package⟩ | ⟨procedure⟩

2.4.2 Package

CH2. ⟨package⟩ ::= {⟨simple-statement-name⟩ • [⟨statement-condition-prefix⟩]}
⟨package-statement⟩ [⟨define-constant-statement-list⟩]
⟨unit-list⟩ ⟨ending⟩

Constraint: Each ⟨unit⟩ in the ⟨unit-list⟩ of a ⟨package⟩ MUST only have the following immediate subnodes:

⟨declare-statement⟩
⟨procedure⟩

Further, some ⟨unit⟩ in the ⟨unit-list⟩ of a ⟨package⟩ MUST immediately contain a ⟨procedure⟩.

2.4.3 Procedure

CH3. ⟨procedure⟩ ::= {⟨simple-statement-name⟩ • [⟨statement-condition-prefix⟩]}
⟨procedure-statement⟩ [⟨define-constant-statement-list⟩]
[⟨unit-list⟩] ⟨ending⟩

2.4.4 Unit

CH4. ⟨unit⟩ ::= ⟨declare-statement⟩ | ⟨simple-statement-name⟩ ⟨format-statement⟩ |
⟨procedure⟩ | ⟨executable-unit⟩

2.4.5 Executable Units

CH5. ⟨executable-unit⟩ ::= {[⟨statement-name⟩]
{⟨group⟩ | ⟨selection⟩ | ⟨on-statement⟩ | ⟨if-statement⟩ |
⟨executable-single-statement⟩}} |
⟨begin-block⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CH6. ⟨executable-single-statement⟩ ::= ⟨allocate-statement⟩ | ⟨assignment-statement⟩ |
⟨call-statement⟩ | ⟨close-statement⟩ |
⟨delete-statement⟩ | ⟨free-statement⟩ |
⟨get-statement⟩ | ⟨goto-statement⟩ |
⟨leave-statement⟩ | ⟨null-statement⟩ |
⟨open-statement⟩ | ⟨put-statement⟩ |
⟨read-statement⟩ | ⟨return-statement⟩ |
⟨revert-statement⟩ | ⟨rewrite-statement⟩ |
⟨signal-statement⟩ | ⟨stop-statement⟩ |
⟨write-statement⟩

2.4.6 If Statement

CH7. ⟨if-statement⟩ ::= ⟨if-clause⟩ {⟨executable-unit⟩ | ⟨balanced-unit⟩ ELSE ⟨executable-unit⟩}

CH8. ⟨balanced-unit⟩ ::= {[⟨statement-name⟩]
{⟨executable-single-statement⟩ | ⟨group⟩ | ⟨selection⟩ |
⟨on-statement⟩ |
⟨if-clause⟩ ⟨balanced-unit⟩ ELSE ⟨balanced-unit⟩}} |
⟨begin-block⟩

CH9. ⟨if-clause⟩ ::= IF ⟨expression⟩ THEN

2.4.7 Do Group

CH10. ⟨group⟩ ::= ⟨do-statement⟩ [⟨unit-list⟩] ⟨ending⟩

2.4.8 Select Group

CH11. ⟨selection⟩ ::= ⟨select-statement⟩ [⟨when-statement-list⟩] [⟨otherwise-statement⟩] ⟨ending⟩

CH12. ⟨when-statement⟩ ::= ⟨when-clause⟩ ⟨executable-unit⟩

CH13. ⟨otherwise-statement⟩ ::= ⟨otherwise-clause⟩ ⟨executable-unit⟩

CH14. ⟨when-clause⟩ ::= WHEN [ANY | ALL] (⟨expression-commalist⟩)

CH15. ⟨otherwise-clause⟩ ::= OTHERWISE

2.4.9 Begin Block

CH16. ⟨begin-block⟩ ::= {[⟨simple-statement-name⟩] • [⟨statement-condition-prefix⟩]}
⟨begin-statement⟩ [⟨define-constant-statement-list⟩] [⟨unit-list⟩]
⟨ending⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.4.10 On Unit

CH17. ⟨on-statement⟩ ::= ON ⟨condition-name-commalist⟩ [SNAP] {⟨on-unit⟩ | SYSTEM;}

CH18. ⟨on-unit⟩ ::= ⟨executable-single-statement⟩ | ⟨begin-block⟩

2.4.10.1 Condition Prefixes

CH19. ⟨statement-condition-prefix⟩ ::= (⟨condition-prefix-commalist⟩):

CH20. ⟨condition-prefix⟩ ::= ⟨computational-condition⟩ | ⟨disabled-computational-condition⟩

CH21. ⟨computational-condition⟩ ::= CONVERSION | FIXEDOVERFLOW | OVERFLOW | SIZE | UNDERFLOW | ZERODIVIDE

CH22. ⟨disabled-computational-condition⟩ ::= NOCONVERSION | NOFIXEDOVERFLOW | NOOVERFLOW | NOSIZE | NOUNDERFLOW | NOZERODIVIDE

2.4.10.2 Statement Name Prefixes

CH23. ⟨statement-name⟩ ::= ⟨identifier⟩ [(⟨signed-integer⟩)]:

CH24. ⟨simple-statement-name⟩ ::= ⟨identifier⟩:

CH25. ⟨signed-integer⟩ ::= [+ | −] ⟨integer⟩

2.4.11 Data Declaration

CH26. ⟨declare-statement⟩ ::= DECLARE ⟨declaration-commalist⟩;

CH27. ⟨declaration⟩ ::= [⟨level⟩] {⟨identifier⟩ | (⟨declaration-commalist⟩)} [⟨dimension-suffix⟩] [⟨attribute-list⟩]

CH28. ⟨level⟩ ::= ⟨integer⟩

CH29. ⟨define-constant-statement⟩ ::= DEFINE CONSTANT ⟨declaration-commalist⟩;

2.4.11.1 Dimension Attribute and Dimension Suffix

CH30. ⟨dimension-attribute⟩ ::= DIMENSION [⟨dimension-suffix⟩]

CH31. ⟨dimension-suffix⟩ ::= (⟨bound-pair-commalist⟩)

CH32. ⟨bound-pair⟩ ::= [⟨lower-bound⟩:] ⟨upper-bound⟩ | *

CH33. ⟨lower-bound⟩ ::= ⟨extent-expression⟩

CH34. ⟨upper-bound⟩ ::= ⟨extent-expression⟩

CH35. ⟨extent-expression⟩ ::= ⟨expression⟩ [⟨refer-option⟩]

CH36. ⟨refer-option⟩ ::= REFER (⟨unsubscripted-reference⟩)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.4.11.2 Attributes

CH37. ⟨attribute⟩ ::=

⟨data-attribute⟩	AUTOMATIC	
BUILTIN	BASED [(⟨reference⟩)]	
CONDITION	CONSTANT	
DIRECT	DEFINED (⟨reference⟩)	
⟨environment⟩	EXTERNAL [(⟨external-name⟩)]	
⟨initial⟩	INPUT	
INTERNAL	KEYED	
NONPRINT	⟨options⟩	
OUTPUT	PARAMETER	
PRINT	RECORD	
RESERVED	SEQUENTIAL	
STATIC	STREAM	
UPDATE	VALUE (⟨expression⟩)	
VARIABLE		

Note: The ⟨attribute⟩ CONSTANT may not be supplied in a ⟨translation-unit⟩. It is only used as a definitional artifice. See the operation VALIDATE-TRANSLATION-UNIT.

2.4.11.3 Data Attributes

CH38. ⟨data-attribute⟩ ::=

ALIGNED	AREA [(⟨area-size⟩)]	
BINARY [⟨precision⟩]	BIT [(⟨maximum-length⟩)]	
DECIMAL [⟨precision⟩]	CHARACTER [(⟨maximum-length⟩)]	
⟨dimension-attribute⟩	ENTRY [([⟨description-commalist⟩])]	
FILE	FIXED [⟨precision⟩]	
FORMAT	FLOAT [(⟨number-of-digits⟩)]	
LABEL	MEMBER	
NONVARYING	OFFSET [(⟨reference⟩)]	
PACKAGE	PICTURE [⟨picture⟩]	
POINTER	PRECISION [⟨precision⟩]	
⟨returns-descriptor⟩	STRUCTURE	
UNALIGNED	UNION	
VARYING		

Note: Some ⟨data-attribute⟩s (i.e. FORMAT, MEMBER, PACKAGE, PRECISION and STRUCTURE) may not be supplied in a ⟨translation-unit⟩. They are used only as a definitional artifice. See the operation VALIDATE-TRANSLATION-UNIT.

CH39. ⟨area-size⟩ ::= ⟨extent-expression⟩ | *

CH40. ⟨precision⟩ ::= (⟨number-of-digits⟩ [,⟨scale-factor⟩])

CH41. ⟨number-of-digits⟩ ::= ⟨integer⟩ | ⟨identifier⟩

CH42. ⟨scale-factor⟩ ::= ⟨integer⟩ | ⟨identifier⟩

CH43. ⟨maximum-length⟩ ::= ⟨extent-expression⟩ | *

CH44. ⟨description⟩ ::= [⟨level⟩] [⟨dimension-suffix⟩] [⟨data-attribute-list⟩]

Constraint: At least one subnode MUST be present.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CH45. ⟨picture⟩ ::= ⟨simple-character-string-constant⟩

2.4.11.4 Environment and Options

CH46. ⟨environment⟩ ::= ENVIRONMENT (⟨environment-specification⟩)

CH47. ⟨environment-specification⟩ ::= This category is IMPLEMENTATION-DEFINED but is partially constrained by the HIGH-LEVEL-PARSE operation in Chapter 4.

CH48. ⟨options⟩ ::= OPTIONS (⟨options-specification⟩)

CH49. ⟨options-specification⟩ ::= This category is IMPLEMENTATION-DEFINED but is partially constrained by the HIGH-LEVEL-PARSE operation in Chapter 4.

2.4.11.5 Initial

CH50. ⟨initial⟩ ::= INITIAL [(⟨initial-element-commalist⟩ | ⟨array-init⟩)]

CH51. ⟨array-init⟩ ::= (*) {⟨initial-constant-two⟩ | (⟨initial-constant-one⟩) | ⟨parenthesized-expression⟩}

CH52. ⟨initial-element⟩ ::= * | ⟨parenthesized-expression⟩ | (⟨iteration-factor⟩) {⟨initial-constant-two⟩ | * | (⟨initial-element-commalist⟩)} | ⟨initial-constant-one⟩

CH53. ⟨initial-constant-one⟩ ::= [⟨prefix-operator⟩] ⟨simple-string-constant⟩ | ⟨initial-constant-two⟩

CH54. ⟨initial-constant-two⟩ ::= [⟨prefix-operator⟩] {⟨reference⟩ | ⟨arithmetic-constant⟩} | [+ | −] ⟨real-constant⟩

CH55. ⟨iteration-factor⟩ ::= ⟨expression⟩

2.4.12 The Package Statement

CH56. ⟨package-statement⟩ ::= PACKAGE {⟨exports-option⟩ • [⟨options⟩]};

CH57. ⟨exports-option⟩ ::= EXPORTS ({⟨exported-entry-commalist⟩ | *})

CH58. ⟨exported-entry⟩ ::= ⟨identifier⟩ [EXTERNAL (⟨external-name⟩)]

2.4.13 The Procedure Statement

CH59. ⟨procedure-statement⟩ ::= PROCEDURE ⟨entry-information⟩;

CH60. ⟨entry-information⟩ ::= [(⟨parameter-name-commalist⟩)] {[⟨returns-descriptor⟩] • [⟨options⟩] • [EXTERNAL (⟨external-name⟩)] • [RECURSIVE | NONRECURSIVE]}

CH61. ⟨parameter-name⟩ ::= ⟨identifier⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CH62. ⟨returns-descriptor⟩ ::= RETURNS [(⟨description-commalist⟩)]

Constraint: The ⟨description-commalist⟩ in a ⟨returns-descriptor⟩ MUST contain exactly one ⟨description⟩,rd. rd MUST immediately contain a ⟨data-attribute-list⟩ and no other components.

CH63. ⟨external-name⟩ ::= ⟨expression⟩

2.4.14 The Begin Statement

CH64. ⟨begin-statement⟩ ::= BEGIN [⟨options⟩];

2.4.15 The Do Statement

CH65. ⟨do-statement⟩ ::= DO LOOP; | DO {[⟨while-option⟩] • [⟨until-option⟩]}; |
DO ⟨do-spec⟩;

CH66. ⟨do-spec⟩ ::= ⟨reference⟩ = ⟨spec⟩

CH67. ⟨spec⟩ ::= ⟨expression⟩ [⟨to-by⟩ | ⟨repeat-option⟩]
{[⟨while-option⟩] • [⟨until-option⟩]}

CH68. ⟨to-by⟩ ::= ⟨to-option⟩ [⟨by-option⟩] | ⟨by-option⟩ [⟨to-option⟩]

CH69. ⟨to-option⟩ ::= TO ⟨expression⟩

CH70. ⟨by-option⟩ ::= BY ⟨expression⟩

CH71. ⟨while-option⟩ ::= WHILE (⟨expression⟩)

CH72. ⟨until-option⟩ ::= UNTIL (⟨expression⟩)

CH73. ⟨repeat-option⟩ ::= REPEAT ⟨expression⟩

2.4.16 The Select Statement

CH74. ⟨select-statement⟩ ::= SELECT [(⟨expression⟩)];

2.4.17 The End Statement

CH75. ⟨ending⟩ ::= [⟨statement-name⟩] ⟨end-statement⟩

CH76. ⟨end-statement⟩ ::= END [⟨identifier⟩];

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.4.18 Flow of Control Statements

2.4.18.1 The Call and Return Statements

CH77. ⟨call-statement⟩ ::= CALL ⟨reference⟩;

CH78. ⟨return-statement⟩ ::= RETURN [(⟨expression⟩)];

2.4.18.2 The Go To Statement

CH79. ⟨goto-statement⟩ ::= {GOTO | GO TO} ⟨reference⟩;

2.4.18.3 The Leave Statement

CH80. ⟨leave-statement⟩ ::= LEAVE [⟨identifier⟩];

2.4.18.4 The Null Statement

CH81. ⟨null-statement⟩ ::= ;

2.4.18.5 The Revert and Signal Statements

CH82. ⟨revert-statement⟩ ::= REVERT ⟨condition-name-commalist⟩;

CH83. ⟨signal-statement⟩ ::= SIGNAL ⟨condition-name⟩;

CH84. ⟨condition-name⟩ ::= ⟨computational-condition⟩ | ⟨named-io-condition⟩ | ⟨programmer-named-condition⟩ | AREA | ERROR | FINISH | STORAGE

CH85. ⟨named-io-condition⟩ ::= ⟨io-condition⟩ (⟨reference⟩)

CH86. ⟨io-condition⟩ ::= BEGINPAGE | ENDFILE | ENDPAGE | KEY | RECORD | TRANSMIT | UNDEFINEDFILE

CH87. ⟨programmer-named-condition⟩ ::= CONDITION (⟨identifier⟩)

2.4.18.6 The Stop Statement

CH88. ⟨stop-statement⟩ ::= STOP;

2.4.19 Storage Control Statements

CH89. ⟨assignment-statement⟩ ::= ⟨reference⟩ = ⟨expression⟩;

CH90. ⟨allocate-statement⟩ ::= ALLOCATE ⟨allocation-commalist⟩;

CH91. ⟨allocation⟩ ::= ⟨identifier⟩ {[⟨set-option⟩] • [⟨in-option⟩]}

CH92. ⟨set-option⟩ ::= SET (⟨reference⟩)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CH93. ⟨in-option⟩ ::= IN (⟨reference⟩)

CH94. ⟨free-statement⟩ ::= FREE ⟨freeing-commalist⟩;

CH95. ⟨freeing⟩ ::= [⟨locator-qualifier⟩] ⟨identifier⟩ [⟨in-option⟩]

2.4.20 Input/Output Statements

2.4.20.1 The Open and Close Statements

CH96. ⟨open-statement⟩ ::= OPEN ⟨single-opening-commalist⟩;

CH97. ⟨single-opening⟩ ::= ⟨file-option⟩ • [⟨title-option⟩] •
[⟨linesize-option⟩] • [⟨pagesize-option⟩] •
[STREAM] • [RECORD] • [INPUT] • [OUTPUT] • [UPDATE] •
[SEQUENTIAL] • [DIRECT] • [PRINT] • [NONPRINT] • [KEYED] •
[⟨environment⟩]

CH98. ⟨file-option⟩ ::= FILE (⟨reference⟩)

CH99. ⟨title-option⟩ ::= TITLE (⟨expression⟩)

CH100. ⟨linesize-option⟩ ::= LINESIZE (⟨expression⟩)

CH101. ⟨pagesize-option⟩ ::= PAGESIZE (⟨expression⟩)

CH102. ⟨close-statement⟩ ::= CLOSE ⟨single-closing-commalist⟩;

CH103. ⟨single-closing⟩ ::= ⟨file-option⟩ • [⟨environment⟩]

2.4.20.2 Record I/O

CH104. ⟨delete-statement⟩ ::= DELETE {⟨file-option⟩ • [⟨key-option⟩] • [⟨environment⟩]};

CH105. ⟨pointer-set-option⟩ ::= SET (⟨reference⟩)

CH106. ⟨read-statement⟩ ::= READ
{⟨file-option⟩ • ⟨into-option⟩ •
[⟨environment⟩] • [⟨key-option⟩ | ⟨keyto-option⟩] |
⟨file-option⟩ • ⟨pointer-set-option⟩ • [⟨sizeto-option⟩] •
[⟨environment⟩] • [⟨key-option⟩ | ⟨keyto-option⟩] }

CH107. ⟨into-option⟩ ::= INTO (⟨reference⟩)

CH108. ⟨key-option⟩ ::= KEY (⟨expression⟩)

CH109. ⟨keyto-option⟩ ::= KEYTO (⟨reference⟩)

CH110. ⟨rewrite-statement⟩ ::= REWRITE
{⟨file-option⟩ • [⟨environment⟩] | ⟨file-option⟩ •
[⟨key-option⟩] • ⟨from-option⟩ • [⟨environment⟩]};

CH111. ⟨write-statement⟩ ::= WRITE
{⟨file-option⟩ • ⟨from-option⟩ • [⟨keyfrom-option⟩] •
[⟨environment⟩]};

CH112. ⟨from-option⟩ ::= FROM (⟨reference⟩)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CH113. ⟨keyfrom-option⟩ ::= KEYFROM (⟨expression⟩)

CH114. ⟨sizeto-option⟩ ::= SIZETO (⟨reference⟩)

2.4.20.3 Stream I/O

CH115. ⟨get-statement⟩ ::= GET {⟨get-file⟩ | ⟨get-string⟩};

CH116. ⟨get-file⟩ ::= [⟨file-option⟩] • [⟨skip-option⟩] • [⟨input-specification⟩]

CH117. ⟨skip-option⟩ ::= SKIP [(⟨expression⟩)]

CH118. ⟨get-string⟩ ::= STRING (⟨expression⟩) • ⟨input-specification⟩

CH119. ⟨put-statement⟩ ::= PUT {⟨put-file⟩ | ⟨put-string⟩};

CH120. ⟨put-file⟩ ::= [⟨file-option⟩] • [⟨skip-option⟩] • [⟨line-option⟩] • [PAGE] • [⟨output-specification⟩]

CH121. ⟨line-option⟩ ::= LINE (⟨expression⟩)

CH122. ⟨put-string⟩ ::= STRING (⟨reference⟩) • ⟨output-specification⟩

2.4.20.3.1 Stream Input Specification

CH123. ⟨input-specification⟩ ::= ⟨list-directed-input⟩ | ⟨edit-directed-input⟩

CH124. ⟨list-directed-input⟩ ::= LIST (⟨input-target-commalist⟩)

CH125. ⟨input-target⟩ ::= ⟨reference⟩ | (⟨input-target-commalist⟩ DO ⟨do-spec⟩)

CH126. ⟨edit-directed-input⟩ ::= EDIT ⟨edit-input-pair⟩

CH127. ⟨edit-input-pair⟩ ::= (⟨input-target-commalist⟩) (⟨format-specification-commalist⟩)

2.4.20.3.2 Stream Output Specification

CH128. ⟨output-specification⟩ ::= ⟨list-directed-output⟩ | ⟨edit-directed-output⟩

CH129. ⟨list-directed-output⟩ ::= LIST (⟨output-source-commalist⟩)

CH130. ⟨output-source⟩ ::= ⟨expression⟩ | (⟨output-source-commalist⟩ DO ⟨do-spec⟩)

CH131. ⟨edit-directed-output⟩ ::= EDIT ⟨edit-output-pair⟩

CH132. ⟨edit-output-pair⟩ ::= (⟨output-source-commalist⟩) (⟨format-specification-commalist⟩)

2.4.20.3.3 Format Specification Lists and the Format Statement

CH133. ⟨format-specification⟩ ::= ⟨format-item⟩ | ⟨format-iteration⟩

CH134. ⟨format-iteration⟩ ::= ⟨format-iteration-factor⟩ {⟨format-item⟩ | (⟨format-specification-commalist⟩)}

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CH135. ⟨format-iteration-factor⟩ ::= ⟨integer⟩ | (⟨expression⟩)

CH136. ⟨format-item⟩ ::= ⟨data-format⟩ | ⟨control-format⟩ | ⟨remote-format⟩

CH137. ⟨data-format⟩ ::= ⟨real-format⟩ | ⟨picture-format⟩ | ⟨string-format⟩ | ⟨l-format⟩

CH138. ⟨real-format⟩ ::= ⟨fixed-point-format⟩ | ⟨floating-point-format⟩

CH139. ⟨fixed-point-format⟩ ::= F (⟨expression⟩ [,⟨expression⟩])

CH140. ⟨floating-point-format⟩ ::= E (⟨expression⟩ [,⟨expression⟩])

CH141. ⟨picture-format⟩ ::= P ⟨picture⟩

CH142. ⟨string-format⟩ ::= ⟨character-format⟩ | ⟨bit-format⟩

CH143. ⟨character-format⟩ ::= A [(⟨expression⟩)]

CH144. ⟨bit-format⟩ ::= ⟨radix-factor⟩ [(⟨expression⟩)]

CH145. ⟨l-format⟩ ::= L

CH146. ⟨control-format⟩ ::= ⟨tab-format⟩ | ⟨line-format⟩ | ⟨space-format⟩ | ⟨skip-format⟩ | ⟨column-format⟩ | PAGE

CH147. ⟨tab-format⟩ ::= TAB [(⟨expression⟩)]

CH148. ⟨line-format⟩ ::= LINE (⟨expression⟩)

CH149. ⟨space-format⟩ ::= X (⟨expression⟩)

CH150. ⟨skip-format⟩ ::= SKIP [(⟨expression⟩)]

CH151. ⟨column-format⟩ ::= COLUMN (⟨expression⟩)

CH152. ⟨remote-format⟩ ::= R (⟨identifier⟩)

CH153. ⟨format-statement⟩ ::= FORMAT (⟨format-specification-commalist⟩);

2.4.21 Expressions

CH154. ⟨expression⟩ ::= ⟨expression-nine⟩ | ⟨expression⟩ ⟨| :⟩ ⟨expression-nine⟩

CH155. ⟨expression-nine⟩ ::= ⟨expression-eight⟩ | ⟨expression-nine⟩ &: ⟨expression-eight⟩

CH156. ⟨expression-eight⟩ ::= ⟨expression-seven⟩ | ⟨expression-eight⟩ {⟨ | ⟩ | ¬} ⟨expression-seven⟩

CH157. ⟨expression-seven⟩ ::= ⟨expression-six⟩ | ⟨expression-seven⟩ & ⟨expression-six⟩

CH158. ⟨expression-six⟩ ::= ⟨expression-five⟩ | ⟨expression-six⟩ ⟨comparison-operator⟩ ⟨expression-five⟩

CH159. ⟨comparison-operator⟩ ::= > | >= | = | < | <= | ¬> | ¬= | ¬<

CH160. ⟨expression-five⟩ ::= ⟨expression-four⟩ | ⟨expression-five⟩ ⟨| | ⟩ ⟨expression-four⟩

CH161. ⟨expression-four⟩ ::= ⟨expression-three⟩ | ⟨expression-four⟩ { + | − } ⟨expression-three⟩

CH162. ⟨expression-three⟩ ::= ⟨expression-two⟩ | ⟨expression-three⟩ { * | / } ⟨expression-two⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CH163. ⟨expression-two⟩ ::= ⟨primitive-expression⟩ | ⟨prefix-expression⟩ | ⟨parenthesized-expression⟩ | ⟨expression-one⟩

CH164. ⟨expression-one⟩ ::= {⟨primitive-expression⟩ | ⟨parenthesized-expression⟩} ** ⟨expression-two⟩

CH165. ⟨prefix-expression⟩ ::= ⟨prefix-operator⟩ ⟨expression-two⟩

CH166. ⟨prefix-operator⟩ ::= + | − | ¬

CH167. ⟨parenthesized-expression⟩ ::= (⟨expression⟩)

CH168. ⟨primitive-expression⟩ ::= ⟨reference⟩ | ⟨constant⟩ | ⟨enquiry-reference⟩

CH169. ⟨reference⟩ ::= [⟨locator-qualifier⟩] ⟨basic-reference⟩ [⟨arguments-list⟩]

CH170. ⟨locator-qualifier⟩ ::= ⟨reference⟩ −>

CH171. ⟨arguments⟩ ::= ([⟨subscript-commalist⟩])

CH172. ⟨basic-reference⟩ ::= [⟨structure-qualification-list⟩] ⟨identifier⟩

CH173. ⟨structure-qualification⟩ ::= ⟨identifier⟩ [⟨arguments⟩].

CH174. ⟨subscript⟩ ::= ⟨expression⟩

CH175. ⟨unsubscripted-reference⟩ ::= [⟨unsubscripted-reference⟩.] ⟨identifier⟩

CH176. ⟨constant⟩ ::= ⟨arithmetic-constant⟩ | ⟨string-constant⟩

CH177. ⟨string-constant⟩ ::= ⟨simple-string-constant⟩

CH178. ⟨simple-string-constant⟩ ::= ⟨simple-character-string-constant⟩ | ⟨simple-bit-string-constant⟩

2.4.22 Enquiry Functions

CH179. ⟨enquiry-reference⟩ ::= ⟨declaresize-enquiry-reference⟩ | ⟨epsilon-enquiry-reference⟩ | ⟨maxval-enquiry-reference⟩ | ⟨minval-enquiry-reference⟩ | ⟨null-enquiry-reference⟩ | ⟨radix-enquiry-reference⟩

CH180. ⟨enquiry-lbracket⟩ ::= (:

CH181. ⟨enquiry-rbracket⟩ ::= :)

CH182. ⟨declaresize-enquiry-reference⟩ ::= ⟨declaresize-enquiry-name⟩ ⟨enquiry-lbracket⟩ ⟨declaresize-arguments⟩ ⟨enquiry-rbracket⟩

CH183. ⟨declaresize-enquiry-name⟩ ::= DECLARESIZE

CH184. ⟨declaresize-arguments⟩ ::= ⟨identifier⟩

CH185. ⟨null-enquiry-reference⟩ ::= ⟨null-enquiry-name⟩ ⟨enquiry-lbracket⟩ ⟨null-arguments⟩ ⟨enquiry-rbracket⟩

CH186. ⟨null-enquiry-name⟩ ::= NULL

CH187. ⟨null-arguments⟩ ::= POINTER | OFFSET | ENTRY | LABEL | FILE

CH188. ⟨maxval-enquiry-reference⟩ ::= ⟨maxval-enquiry-name⟩ ⟨enquiry-lbracket⟩ ⟨maxval-arguments⟩ ⟨enquiry-rbracket⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CH189. ⟨maxval-enquiry-name⟩ ::= MAXVAL

CH190. ⟨maxval-arguments⟩ ::= ⟨arithmetic-enquiry-description⟩

CH191. ⟨minval-enquiry-reference⟩ ::= ⟨minval-enquiry-name⟩ ⟨enquiry-lbracket⟩ ⟨minval-arguments⟩ ⟨enquiry-rbracket⟩

CH192. ⟨minval-enquiry-name⟩ ::= MINVAL

CH193. ⟨minval-arguments⟩ ::= ⟨arithmetic-enquiry-description⟩

CH194. ⟨epsilon-enquiry-reference⟩ ::= ⟨epsilon-enquiry-name⟩ ⟨enquiry-lbracket⟩ ⟨epsilon-arguments⟩ ⟨enquiry-rbracket⟩

CH195. ⟨epsilon-enquiry-name⟩ ::= EPSILON

CH196. ⟨epsilon-arguments⟩ ::= ⟨arithmetic-enquiry-description⟩

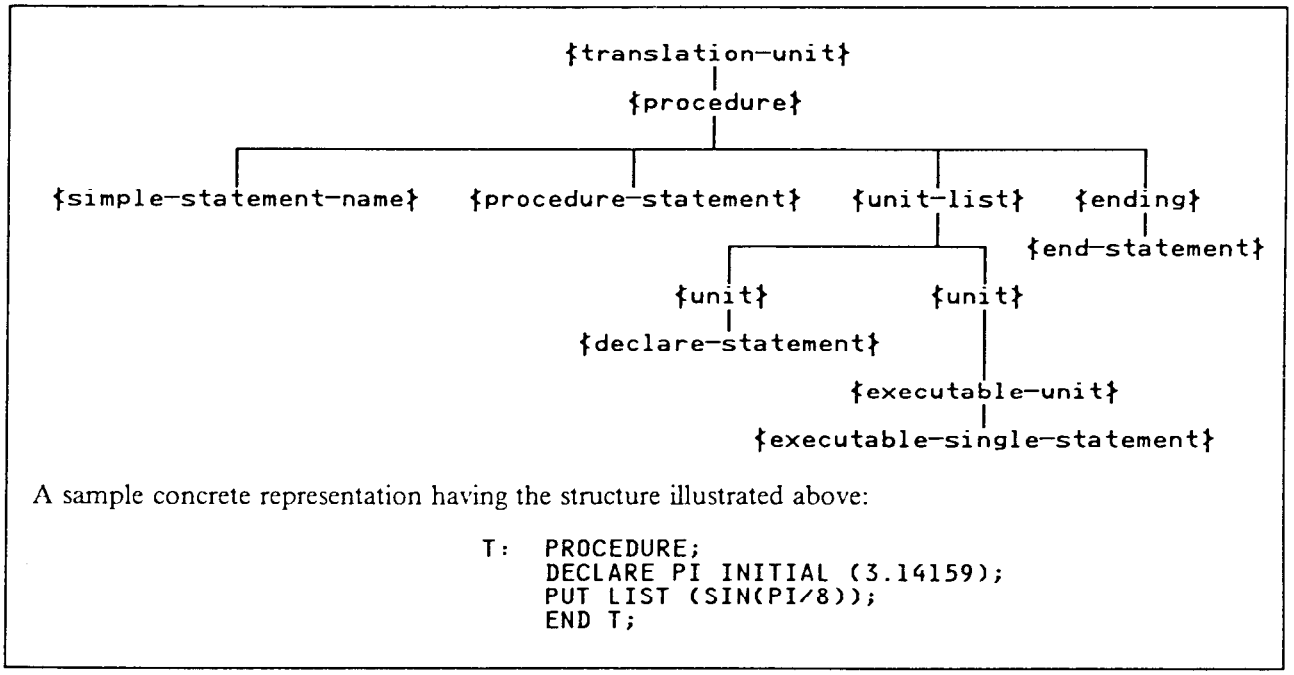
CH197. ⟨radix-enquiry-reference⟩ ::= ⟨radix-enquiry-name⟩ ⟨enquiry-lbracket⟩ ⟨radix-arguments⟩ ⟨enquiry-rbracket⟩

CH198. ⟨radix-enquiry-name⟩ ::= RADIX

CH199. ⟨radix-arguments⟩ ::= ⟨arithmetic-enquiry-description⟩

CH200. ⟨arithmetic-enquiry-description⟩ ::= ⟨description⟩

The following are examples of four high-level parses which illustrate the structure of a simple ⟨translation-unit⟩, the structure of a simple ⟨if-statement⟩, the structure of an ⟨expression⟩, and the structure of a ⟨do-statement⟩, respectively.

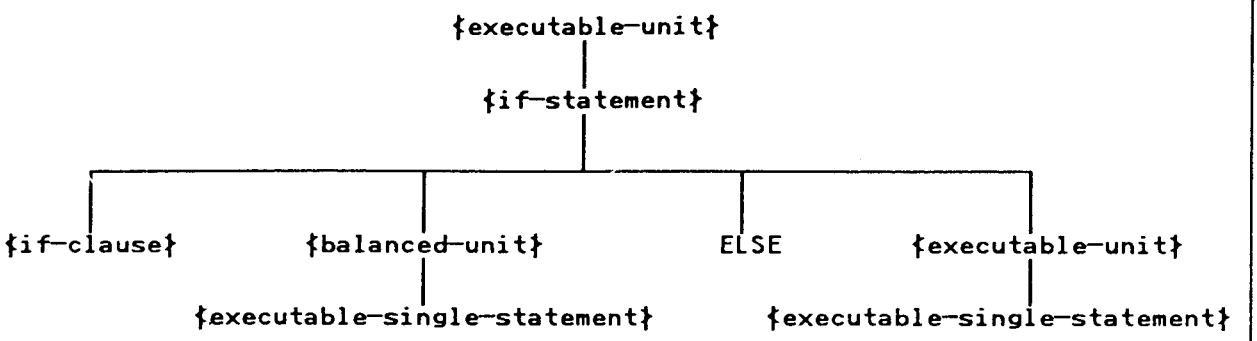


A sample concrete representation having the structure illustrated above:

```
T:  PROCEDURE;
    DECLARE PI INITIAL (3.14159);
    PUT LIST (SIN(PI/8));
    END T;
```

Figure 7. The Structure of a Simple ⟨translation-unit⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



A sample concrete representation having the structure illustrated above:

```
IF X>0 THEN PUT LIST (SIN(PI/8));
       ELSE PUT LIST (COS(PI/8));
```

Figure 8. The Structure of an ⟨if-statement⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

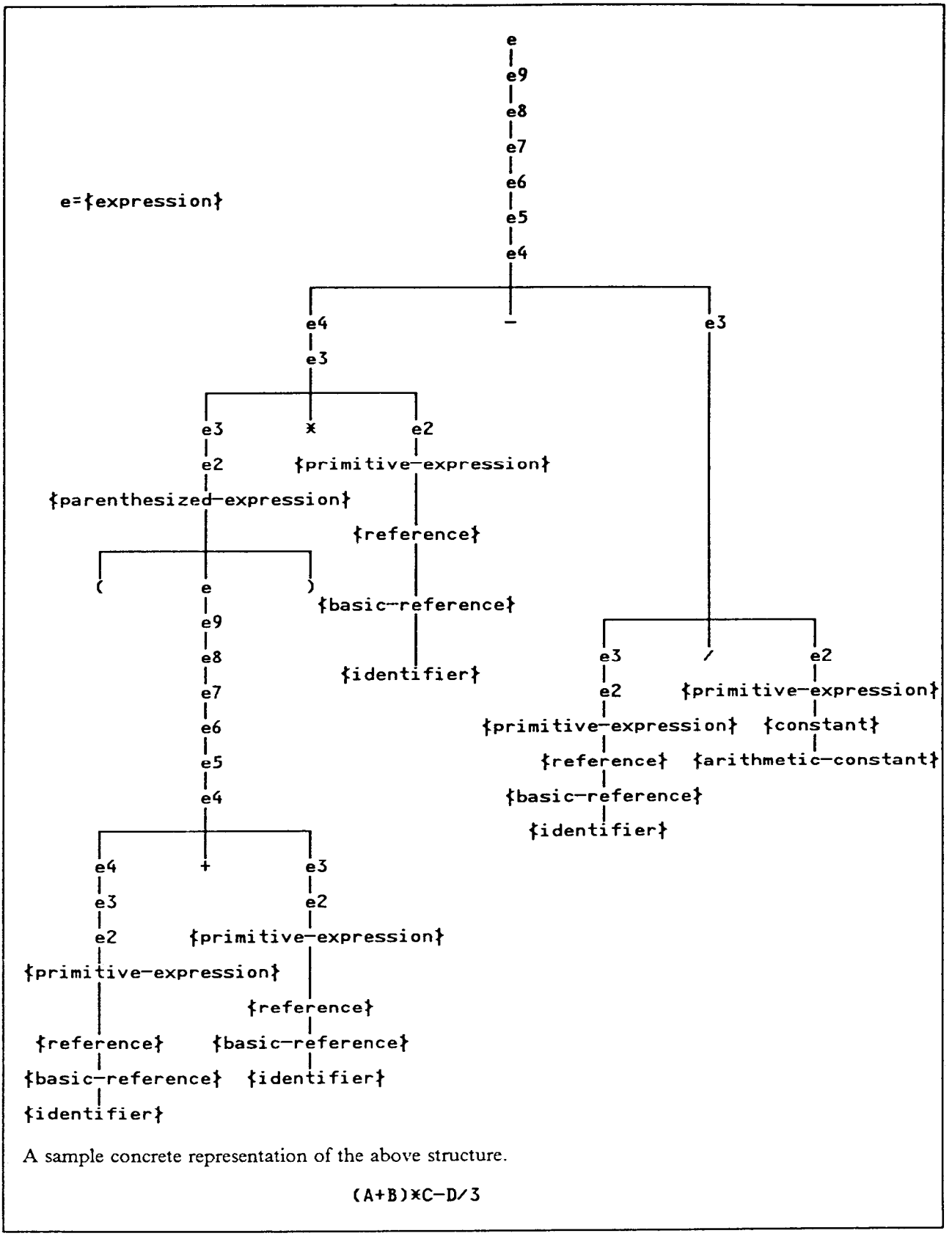
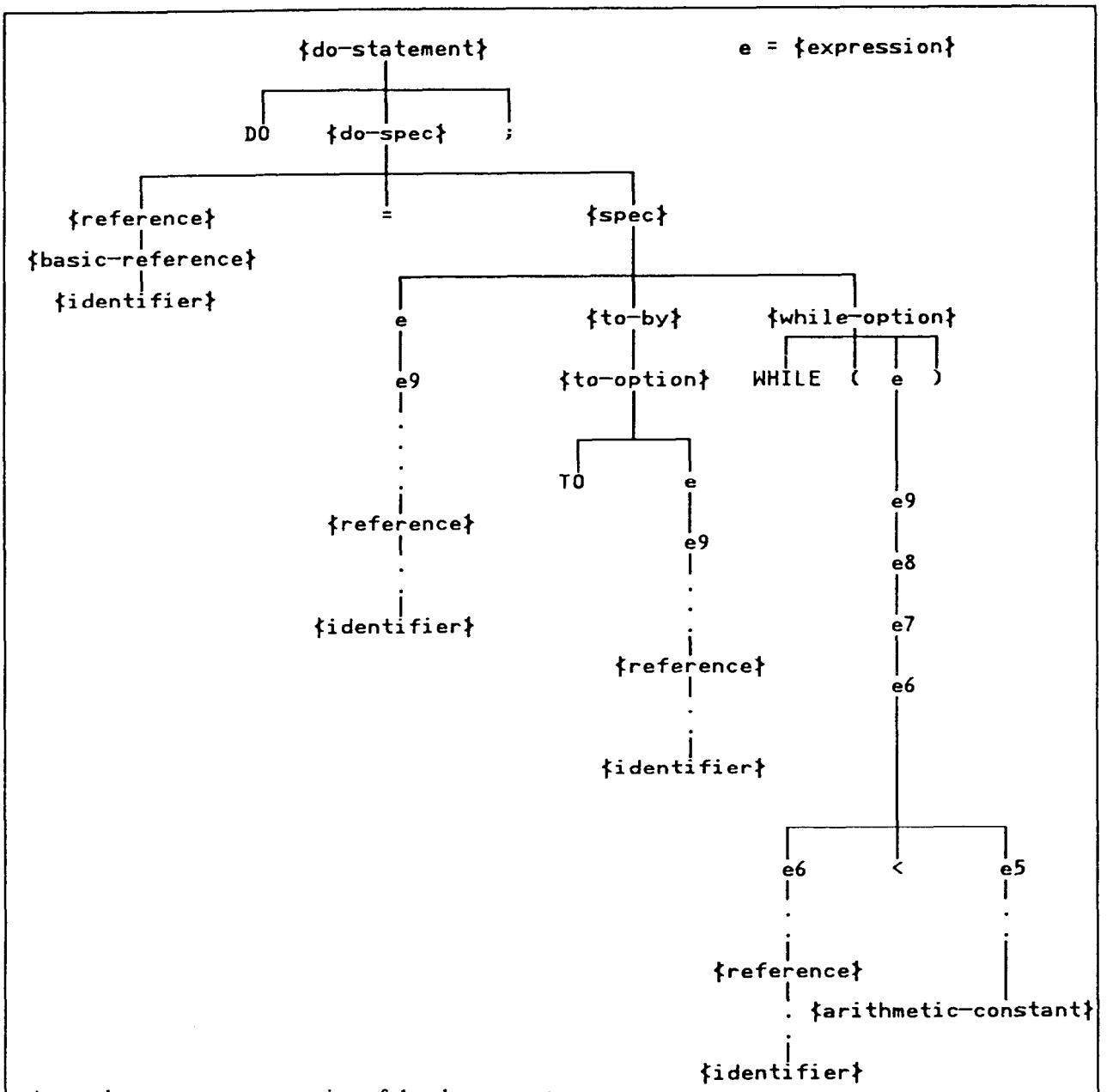


Figure 9. The Structure of an ⟨expression⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



A sample concrete representation of the above structure:

```
DO I=K TO J WHILE(A<0);
```

Figure 10. The Structure of a ⟨do-statement⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.5 The Low-level Syntax of PL/I

2.5.1 PL/I Text

CL1. ⟨pli-text⟩ ::= [⟨delimiter-list⟩] [⟨delimiter-pair-list⟩]

CL2. ⟨delimiter-pair⟩ ::= ⟨non-delimiter⟩ ⟨delimiter-list⟩

CL3. ⟨delimiter⟩ ::= + | − | * | / | ** | > | < | = | >= | <= | ¬> | ¬< | ¬ = | ¬ | & | ⟨|⟩ | &: | ⟨|:⟩ | ⟨||⟩ | (|) | (: | :) | . | , | ; | : | −> | ⊬ | ⟨comment⟩ | ⟨include⟩

CL4. ⟨non-delimiter⟩ ::= ⟨identifier⟩ | ⟨arithmetic-constant⟩ | ⟨simple-bit-string-constant⟩ | ⟨simple-character-string-constant⟩ | ⟨pic-spec⟩

2.5.2 Comment

CL5. ⟨comment⟩ ::= /* [⟨comment-body-list⟩] ⟨*-list⟩ /

CL6. ⟨comment-body⟩ ::= [⟨*-list⟩] ⟨comment-character⟩ | /

CL7. ⟨comment-character⟩ ::= ⟨letter⟩ | ⟨digit⟩ | . | ⊬ | ' | = | + | − | (|) | , | __ | $ | ; | : | & | ⟨|⟩ | ¬ | > | < | % | ⟨extralingual-character⟩

Note: Rules CL5, CL6 and CL7 effectively state that a comment begins with /* and ends with */ and that any characters may appear between these except the consecutive pair */.

2.5.3 Identifier

CL8. ⟨identifier⟩ ::= ⟨letter⟩ [⟨identifier-tail-list⟩]

CL9. ⟨identifier-tail⟩ ::= ⟨letter⟩ | ⟨digit⟩ | __

2.5.4 Arithmetic Constant

CL10. ⟨arithmetic-constant⟩ ::= ⟨real-constant⟩

CL11. ⟨real-constant⟩ ::= ⟨decimal-constant⟩

CL12. ⟨decimal-constant⟩ ::= ⟨decimal-number⟩ [E ⟨exponent⟩]

CL13. ⟨decimal-number⟩ ::= ⟨integer⟩[.[⟨digit-list⟩]] | . ⟨digit-list⟩

CL14. ⟨integer⟩ ::= ⟨digit-list⟩

CL15. ⟨exponent⟩ ::= [+ | −] ⟨integer⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.5.5 String Constants and Pictures

CL16. ⟨simple-bit-string-constant⟩ ::= '[⟨string-or-picture-symbol-list⟩]' ⟨radix-factor⟩

CL17. ⟨radix-factor⟩ ::= B | B1 | B2 | B3 | B4

CL18. ⟨pic-spec⟩ ::= ⟨pic-key⟩ ⟨simple-character-string-constant⟩

Note: The category ⟨pic-spec⟩ is used only by the operation LOW-LEVEL-PARSE. No ⟨pic-spec⟩ exist in the ⟨pli-text⟩ returned by LOW-LEVEL-PARSE (see Step 2).

CL19. ⟨pic-key⟩ ::= PICTURE | P

CL20. ⟨simple-character-string-constant⟩ ::= '[⟨string-or-picture-symbol-list⟩]'

CL21. ⟨string-or-picture-symbol⟩ ::= ⟨letter⟩ | ⟨digit⟩ | . | ∅ | '' | = | + | − | * | / | (|) | , | __ | $ | ; | : | & | ⟨ | ⟩ | ¬ | > | < | % | ⟨extralingual-character⟩

Note: A ⟨string-or-picture-symbol⟩ may be two consecutive quote characters ('') or any character other than '.

CL22. ⟨extralingual-character⟩ ::= The category ⟨extralingual-character⟩ in rule CL22 is IMPLEMENTATION-DEFINED, as specified in Section2.6.2 .

2.5.6 Include

CL23. ⟨include⟩ ::= %INCLUDE ⟨text-name⟩;

CL24. ⟨text-name⟩ ::= The category ⟨text-name⟩ in rule CL24 is IMPLEMENTATION-DEFINED.

If any of the following are contained in ⟨text-name⟩ then they MUST be contained in a ⟨simple-character-string-constant⟩ which is the only immediate subnode of ⟨text-name⟩:

the character ";"
the character sequence "/*" if it is not part of a ⟨comment⟩

The character "'" MUST NOT appear in a ⟨text-name⟩ except as a component of a ⟨simple-character-string-constant⟩ contained in a ⟨text-name⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.6 Character Sets

The character set used in the formation of PL/I text is a finite set of symbols, comprising 83 language characters, and zero or more ⟨extralingual-character⟩s which are distinct from these and from each other and are IMPLEMENTATION-DEFINED.

⟨symbol⟩ ::= ⟨letter⟩ | ⟨digit⟩ | . | ⊬ | ' | = | + | – | * | / | (|) | , | __ | $ | ; | : | & |
⟨ | ⟩ | ¬ | < | > | % | ⟨extralingual-character⟩

⟨letter⟩ ::= ⟨secondary-case-letter⟩ | ⟨primary-case-letter⟩

⟨secondary-case-letter⟩ ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

⟨primary-case-letter⟩ ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

A ⟨secondary-case-letter⟩ corresponds to the ⟨primary-case-letter⟩ with the equivalent position in their respective production rules.

⟨digit⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

This document does not specify internal or external hardware representations of the characters, nor does it define a collating sequence for them.

Note: Collating sequences and internal or external hardware representations may be the subjects of other standards. In particular, it is anticipated, but not required, by this standard, that the collating sequence and external representations in implementations and environments whose character representations conform to current versions of ISO 646 or ANSI X3.4-1986 will be in conformity with those standards.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.6.1 Language Character Set

The names of the symbols in the language character set, together with the graphic representations of them to be used in this document, are given in Section 2.6.1.1 and Section 2.6.1.2.

2.6.1.1 Letters and Digits

Name	Primary Case Graphic	Secondary Case Graphic	Name	Graphic
Letter A	A	a	Digit 0	0
Letter B	B	b	Digit 1	1
Letter C	C	c	Digit 2	2
Letter D	D	d	Digit 3	3
Letter E	E	e	Digit 4	4
Letter F	F	f	Digit 5	5
Letter G	G	g	Digit 6	6
Letter H	H	h	Digit 7	7
Letter I	I	i	Digit 8	8
Letter J	J	j	Digit 9	9
Letter K	K	k		
Letter L	L	l		
Letter M	M	m		
Letter N	N	n		
Letter O	O	o		
Letter P	P	p		
Letter Q	Q	q		
Letter R	R	r		
Letter S	S	s		
Letter T	T	t		
Letter U	U	u		
Letter V	V	v		
Letter W	W	w		
Letter X	X	x		
Letter Y	Y	y		
Letter Z	Z	z		

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.6.1.2 Special Characters

Name	Graphic	Name	Graphic
Plus	+	Period	.
Minus	–	Comma	,
Asterisk	*	Semicolon	;
Slash	/	Colon	:
Greater than	>	Blank	␣
Less than	<	Single quote	'
Equal	=	Left parenthesis	(
Not	¬	Right parenthesis	)
And	&	Break	
Or	|	Dollar	$
Percent	%		

2.6.2 Data Character Set

Data in stream-datasets or in character-string-values may be represented by characters from the language character set plus any other characters permitted by the particular implementation's ⟨extralingual-character⟩s.

2.6.3 Secondary Case Characters in Numeric and Bit Values

The ⟨secondary-case-letter⟩s corresponding to the ⟨primary-case-letter⟩s E in the definition of ⟨decimal-constant⟩ and B in the definition of ⟨radix-factor⟩ will be recognized as equivalents in every respect to the corresponding ⟨primary-case-letter⟩s.

The ⟨secondary-case-letter⟩s corresponding to the ⟨primary-case-letter⟩s A, B, C, D, E, and F within bit-string constants and data will be recognized as equivalents in every respect to the corresponding ⟨primary-case-letter⟩s.

The cases above occur in the operation CREATE-CONSTANT in the translator and in the operations referenced by the operation CONVERT. These operations control the recognition and conversion of valid numeric and bit values in the concrete syntax, in stream ⟨dataset⟩s, and in conversion from character.

The recognition of both ⟨primary-case-letter⟩s and ⟨secondary-case-letter⟩s in pictures is required by the operation VALIDATE-FIELD-OF-PICTURED-VALUE.

The effect of these rules is to require that both ⟨primary-case-letter⟩s and ⟨secondary-case-letter⟩s be supported in all contexts where the ⟨symbol-list⟩s can be supplied by sources external to this standard.

Only ⟨primary-case-letter⟩s are generated by the conversion operations in this standard (e.g. see Case 1.2.2 of CONVERT-TO-CHARACTER, Case 4.9.1.2 of EDIT-NUMERIC-PICTURE-FIELD, and Step 3.2.5 of EDIT-NUMERIC-OUTPUT).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.7 Abbreviations

Abbreviations are provided for certain keywords (see Section 4.3.2), enquiry-names, and builtin-function-names. The abbreviations will be recognized as synonymous in every respect with the full denotations, except that in the case of builtin-function-names the abbreviations have separate declarations (explicit or contextual) and name scopes. The abbreviations are shown to the right of the full denotations in the following list.

Keyword	Abbreviation	Keyword	Abbreviation
ALLOCATE	ALLOC	NOCONVERSION	NOCONV
AUTOMATIC	AUTO	NOFIXEDOVERFLOW	NOFOFL
BINARY	BIN	NONVARYING	NONVAR
CHARACTER	CHAR	NOOVERFLOW	NOOFL
COLUMN	COL	NOUNDERFLOW	NOUFL
CONDITION	COND	NOZERODIVIDE	NOZDIV
CONVERSION	CONV	OVERFLOW	OFL
DECIMAL	DEC	PARAMETER	PARM
DECLARE	DCL	PICTURE	PIC
DECLARESIZE	DCLSIZE	POINTER	PTR
DEFINED	DEF	PRECISION	PREC
DIMENSION	DIM	PROCEDURE	PROC
ENVIRONMENT	ENV	SEQUENTIAL	SEQL
EXTERNAL	EXT	UNALIGNED	UNAL
FIXEDOVERFLOW	FOFL	UNDERFLOW	UFL
IDENTICAL	IDENT	UNDEFINEDFILE	UNDF
INITIAL	INIT	VARYING	VAR
INTERNAL	INT	ZERODIVIDE	ZDIV

Keywords, built-in function names, picture specifications, and format items can be defined using ⟨primary-case-letter⟩s, the corresponding ⟨secondary-case-letter⟩s, or any combination of these two. These different specifications will be recognized as synonymous in every respect to ones with only ⟨primary-case-letter⟩s.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3. Abstract Syntax

3.1 Introduction

This chapter specifies the Abstract Syntax of PL/I, which is the syntax of programs in a tree representation which is convenient for the definition of semantics. The notation for this syntax is defined in Chapter 1, together with some examples of its use. Further examples are provided at the end of this chapter.

Many parts of the Abstract Syntax bear a strong resemblance to the corresponding parts of the Concrete Syntax, and the relationship between them is intended to be, or become, intuitively obvious. In other parts, notably in the treatment of <declaration>s, the Abstract Syntax exhibits a structuring and completeness of information which involves a more complex transformation. The detailed description of the transformation between the concrete and abstract forms of a program will be given in the next chapter.

There are also many instances of context-dependent constraints which have been inserted in parentheses in the Abstract Syntax. These are attached to categories such as <expression>, <target-reference>, <value-reference> and <declaration-designator>, where these are required to fall within the scope of appropriate <declaration>s so that they have the properties indicated. The Translator (Chapter 4) checks that these constraints are satisfied.

3.2 Abstract Syntax Rules

3.2.1 Program

A1. <program> ::= [<external-unit-list>]

A2. <external-unit> ::= [<declaration-list>] {<procedure> | <package>}

3.2.2 Package

A3. <package> ::= [<condition-prefix-list>] [<declaration-list>] <procedure-list>

3.2.3 Procedure

A4. <procedure> ::= [<declaration-list>] [<procedure-list>] [<format-statement-list>]
[<condition-prefix-list>] [<recursive>] <entry-point> <executable-unit-list>

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.2.4 Declaration

A5. <declaration> ::= <identifier> <scope> <declaration-type> ⟨declaration-designator⟩

A6. <scope> ::= <external> [<external-name>] | <internal>

A7. <external-name> ::= <expression> (scalar AND character)

A8. <declaration-type> ::= <variable> | <named-constant> | <builtin> | <condition>

3.2.5 Variable

A9. <variable> ::= <storage-type> <data-description>

A10. <storage-type> ::= <storage-class> | <defined> | <parameter>

A11. <storage-class> ::= <automatic> | <based> | <static> [<reserved>]

A12. <based> ::= [<value-reference> (scalar AND locator) | ⟨reference-designator⟩]

A13. <defined> ::= <base-item>

A14. <base-item> ::= <variable-reference> (NOT defined AND NOT based) | ⟨reference-designator⟩

3.2.6 Data-description

A15. <data-description> ::= <dimensioned-data-description> |<structure-data-description> |
<union-data-description> |<item-data-description>

A16. <dimensioned-data-description> ::= <element-data-description> <bound-pair-list>

A17. <element-data-description> ::= <structure-data-description> | <union-data-description> |
<item-data-description>

A18. <bound-pair> ::= <lower-bound> <upper-bound> | <asterisk>

A19. <lower-bound> ::= <extent-expression>

A20. <upper-bound> ::= <extent-expression>

A21. <extent-expression> ::= {<expression> (scalar AND integer) |
⟨expression-designator⟩ | ⟨integer-value⟩} [<refer-option>]

A22. <refer-option> ::= <identifier-list> | ⟨integer-value⟩

A23. <structure-data-description> ::= [<identifier-list>] <member-description-list>

A24. <union-data-description> ::= [<identifier-list>] <member-description-list>

A25. <member-description> ::= <data-description>

A26. <item-data-description> ::= [<alignment>] <data-type> [<initial>]

Constraint: An <item-data-description> MUST NOT have an <initial> component with an <iteration-factor>, or an <initial-element-list> with more than one <initial-element> immediate

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

component, unless the <item-data-description> is a (not necessarily immediate) component of a <dimensioned-data-description>. An <item-data-description> MUST NOT have an <initial> if it is a (not necessarily immediate) component of a <union-data-description>.

A27. <initial> ::= <initial-element-list> | ⟨initial-designator⟩ | <array-init>

A28. <array-init> ::= <parenthesized-expression> (scalar)

A29. <initial-element> ::= <asterisk> | <parenthesized-expression> (scalar) | <iteration-factor> <initial-element-list>

A30. <iteration-factor> ::= <expression> (scalar AND integer)

A31. <alignment> ::= <aligned> | <unaligned>

3.2.7 Data-type

A32. <data-type> ::= <computational-type> | <non-computational-type>

A33. <computational-type> ::= <arithmetic> | <string> | <pictured>

A34. <non-computational-type> ::= <area> | <entry> | <file> | <format> | <label> | <locator>

A35. <arithmetic> ::= <base> <scale> <precision>

A36. <base> ::= <binary> | <decimal>

A37. <scale> ::= <fixed> | <float>

A38. <precision> ::= <number-of-digits> [<scale-factor>] | ⟨precision-designator⟩

A39. <number-of-digits> ::= <integer>

Constraint: The <integer> MUST NOT be zero.

A40. <scale-factor> ::= <integer>

A41. <string> ::= <string-type> <maximum-length> [<varying> | <nonvarying>]

A42. <string-type> ::= <character> | <bit>

A43. <maximum-length> ::= <extent-expression> | <asterisk>

A44. <pictured> ::= <pictured-numeric>

A45. <locator> ::= <pointer> | <offset>

A46. <offset> ::= [<variable-reference> (scalar AND area) | ⟨reference-designator⟩]

A47. <entry> ::= [<parameter-descriptor-list>] [<returns-descriptor>] [<options>]

A48. <parameter-descriptor> ::= <data-description>

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A49. <returns-descriptor> ::= <data-description>

Note: The <data-description> in a <returns-descriptor> always immediately contains an <item-data-description>. See the constraint following CH62.

A50. <options> ::= This category is IMPLEMENTATION-DEFINED.

A51. <area> ::= <area-size>

A52. <area-size> ::= <extent-expression> | <asterisk>

3.2.8 Named-constant

A53. <named-constant> ::= <package-constant> |
{<entry> | <file> <file-description-set> | <format> |
<label> [<bound-pair>]} |
{<arithmetic> | <string>} ⟨basic-value⟩ | ⟨declaration-designator⟩

A54. <file-description-set> ::= [<stream>] [<record>] [<input>] [<output>] [<update>]
[<sequential>] [<direct>] [<print>] [<nonprint>] [<keyed>]
[<environment>]

A55. <environment> ::= This category is IMPLEMENTATION-DEFINED.

3.2.9 Entry-point

A56. <entry-point> ::= [<statement-name>] <entry-information>

A57. <statement-name> ::= <identifier> [<signed-integer>]

A58. <entry-information> ::= [<parameter-name-list>] [<returns-descriptor>] [<options>]

A59. <parameter-name> ::= <identifier>

3.2.10 Executable-unit

A60. <executable-unit> ::= [<statement-name>]
{<begin-block> | <group> | <selection> |
<allocate-statement> | <assignment-statement> | <call-statement> |
<close-statement> | <delete-statement> | <end-statement> |
<free-statement> | <get-statement> | <goto-statement> |
<if-statement> | <leave-statement> | <null-statement> |
<on-statement> | <open-statement> | <put-statement> |
<read-statement> | <return-statement> | <revert-statement> |
<rewrite-statement> | <signal-statement> | <stop-statement> |
<write-statement>}

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.2.11 Begin-block

A61. <begin-block> ::= [<declaration-list>] [<procedure-list>] [<format-statement-list>] [<condition-prefix-list>] [<options>] <executable-unit-list>

3.2.12 Groups

A62. <group> ::= <iterative-group> | <non-iterative-group>

A63. <iterative-group> ::= <controlled-group> | <conditional-group> | <looping-group>

A64. <controlled-group> ::= <do-spec> <executable-unit-list>

A65. <do-spec> ::= <variable-reference> (scalar AND pointer OR scalar AND integer) <spec>

A66. <spec> ::= <expression> (scalar AND pointer OR scalar AND arithmetic) [<to-by> | <repeat-option>] [<while-option>] [<until-option>]

A67. <to-by> ::= <to-option> [<by-option>] | <by-option>

A68. <repeat-option> ::= <expression> (scalar AND pointer OR scalar AND arithmetic)

A69. <by-option> ::= <expression> (scalar AND arithmetic)

A70. <to-option> ::= <expression> (scalar AND arithmetic)

A71. <while-option> ::= <expression> (scalar AND bit)

A72. <until-option> ::= <expression> (scalar AND bit)

A73. <conditional-group> ::= [<while-option>] [<until-option>] <executable-unit-list>

Note: A <conditional-group> contains at least one of <while-option> or <until-option> or both.

A74. <looping-group> ::= <executable-unit-list>

A75. <non-iterative-group> ::= <executable-unit-list>

3.2.13 Selections

A76. <selection> ::= [<expression> (scalar)] [<when-unit-list>] [<otherwise-unit>]

A77. <when-unit> ::= {<when-any-values> | <when-all-values>} <executable-unit>

A78. <otherwise-unit> ::= <executable-unit>

A79. <when-any-values> ::= <expression-list> (scalar)

A80. <when-all-values> ::= <expression-list> (scalar)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.2.14 On Statement

A81. <on-statement> ::= <condition-name-list> [<snap>] {<on-unit> | <system-action>}

A82. <on-unit> ::= <procedure>

3.2.15 If Statement

A83. <if-statement> ::= <test> <then-unit> [<else-unit>]

A84. <test> ::= <expression> (scalar AND bit)

A85. <then-unit> ::= <executable-unit>

A86. <else-unit> ::= <executable-unit>

3.2.16 Flow of Control Statements

A87. <call-statement> ::= <subroutine-reference>

A88. <leave-statement> ::= [<named-constant-reference> (scalar AND label)]

A89. <goto-statement> ::= <value-reference> (scalar AND label)

A90. <return-statement> ::= [<expression> (scalar)]

A91. <revert-statement> ::= <condition-name-list>

A92. <signal-statement> ::= <condition-name>

A93. <condition-name> ::= <computational-condition> | <named-io-condition> |
<programmer-named-condition> | <area-condition> |
<error-condition> | <finish-condition> |
<storage-condition>

A94. <condition-prefix> ::= <computational-condition> {<enabled> | <disabled>}

A95. <computational-condition> ::= <conversion-condition> | <fixedoverflow-condition> |
<overflow-condition> | <size-condition> |
<underflow-condition> | <zerodivide-condition>

A96. <named-io-condition> ::= <io-condition> <value-reference> (scalar AND file)

A97. <io-condition> ::= <beginpage-condition> | <endfile-condition> | <endpage-condition> |
<key-condition> | <record-condition> | <transmit-condition> |
<undefinedfile-condition>

A98. <programmer-named-condition> ::= <declaration-designator> (condition)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.2.17 Storage Statements

A99. <assignment-statement> ::= <target-reference> <expression>

A100. <allocate-statement> ::= <allocation-list>

A101. <allocation> ::= <declaration-designator> (based) [<set-option>] [<in-option>]

A102. <set-option> ::= <variable-reference> (scalar AND locator)

Constraint: The <data-description> immediate component of the <variable-reference> MUST NOT have <offset> without a <variable-reference> subnode.

A103. <in-option> ::= <variable-reference> (scalar AND area)

A104. <free-statement> ::= <freeing-list>

A105. <freeing> ::= [<locator-qualifier>] <declaration-designator> (based) [<in-option>]

3.2.18 I/O Statements

A106. <open-statement> ::= <single-opening-list>

A107. <single-opening> ::= <file-option> [<title-option>]
[<linesize-option>] [<pagesize-option>]
[<stream>] [<record>] [<input>] [<output>] [<update>]
[<sequential>] [<direct>] [<print>] [<nonprint>] [<keyed>]
[<environment>]

A108. <file-option> ::= <value-reference> (scalar AND file)

A109. <title-option> ::= <expression> (scalar AND computational-type)

A110. <linesize-option> ::= <expression> (scalar AND computational-type)

A111. <pagesize-option> ::= <expression> (scalar AND computational-type)

A112. <close-statement> ::= <single-closing-list>

A113. <single-closing> ::= <file-option> [<environment>]

3.2.19 Record I/O Statements

A114. <delete-statement> ::= <file-option> [<key-option>] [<environment>]

A115. <pointer-set-option> ::= <variable-reference> (scalar AND pointer)

A116. <read-statement> ::= <file-option>
{ <into-option> | <pointer-set-option> [<sizeto-option>] }
[<key-option> | <keyto-option>] [<environment>]

A117. <into-option> ::= <variable-reference>

A118. <rewrite-statement> ::= <file-option> [[<key-option>] <from-option>] [<environment>]

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A119. <write-statement> ::= <file-option> <from-option> [<keyfrom-option>] [<environment>]

A120. <from-option> ::= <variable-reference>

A121. <key-option> ::= <expression> (scalar AND computational-type)

A122. <keyfrom-option> ::= <expression> (scalar AND computational-type)

A123. <keyto-option> ::= <variable-reference> (scalar AND character)

A124. <sizeto-option> ::= <variable-reference> (scalar AND arithmetic)

3.2.20 Stream I/O Statements

A125. <get-statement> ::= <get-file> | <get-string>

A126. <get-file> ::= <file-option> [<skip-option>] [<input-specification>]

Constraint: At least one of the last two options MUST be present.

A127. <skip-option> ::= <expression> (scalar AND computational-type)

A128. <get-string> ::= <expression> (scalar AND character) <input-specification>

A129. <put-statement> ::= <put-file> | <put-string>

A130. <put-file> ::= <file-option> [<skip-option>] [<line-option>] [<page>] [<output-specification>]

Constraint: At least one of the last four options MUST be present and the <skip-option> MUST NOT be used together with a <line-option> or <page>.

A131. <line-option> ::= <expression> (scalar AND computational-type)

A132. <put-string> ::= <variable-reference> (scalar AND character) <output-specification>

A133. <input-specification> ::= <list-directed-input> | <edit-directed-input>

A134. <list-directed-input> ::= <input-target-list>

A135. <input-target> ::= <variable-reference> (computational-type AND NOT union) | <input-target-list> <do-spec>

A136. <edit-directed-input> ::= <edit-input-pair>

A137. <edit-input-pair> ::= <input-target-list> <format-specification-list>

A138. <output-specification> ::= <list-directed-output> | <edit-directed-output>

A139. <list-directed-output> ::= <output-source-list>

A140. <output-source> ::= <expression> (computational-type AND NOT union) | <output-source-list> <do-spec>

A141. <edit-directed-output> ::= <edit-output-pair>

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A142. <edit-output-pair> ::= <output-source-list> <format-specification-list>

A143. <format-specification> ::= <format-item> | <format-iteration>

A144. <format-iteration> ::= <format-iteration-factor> <format-specification-list>

A145. <format-iteration-factor> ::= <expression> (scalar AND integer)

A146. <format-item> ::= <data-format> | <control-format> | <remote-format>

A147. <data-format> ::= <real-format> | <picture-format> | <string-format> | <l-format>

A148. <real-format> ::= <fixed-point-format> | <floating-point-format>

A149. <fixed-point-format> ::= {<expression> (scalar AND integer) | ⩽integer-value⩾}
[{<expression> (scalar AND integer) | ⩽integer-value⩾}]

A150. <floating-point-format> ::= {<expression> (scalar AND integer) | ⩽integer-value⩾}
[{<expression> (scalar AND integer) | ⩽integer-value⩾}]

A151. <picture-format> ::= <pictured>

A152. <string-format> ::= <character-format> | <bit-format>

A153. <character-format> ::= [{<expression> (scalar AND integer) | ⩽integer-value⩾}]

A154. <bit-format> ::= <radix-factor> [{<expression> (scalar AND integer) | ⩽integer-value⩾}]

A155. <radix-factor> ::= 1 | 2 | 3 | 4

A156. <control-format> ::= <tab-format> | <line-format> | <space-format> | <skip-format> |
<column-format> | <page>

A157. <tab-format> ::= {<expression> (scalar AND integer) | ⩽integer-value⩾}

A158. <line-format> ::= {<expression> (scalar AND integer) | ⩽integer-value⩾}

A159. <space-format> ::= {<expression> (scalar AND integer) | ⩽integer-value⩾}

A160. <skip-format> ::= {<expression> (scalar AND integer) | ⩽integer-value⩾}

A161. <column-format> ::= {<expression> (scalar AND integer) | ⩽integer-value⩾}

A162. <remote-format> ::= <value-reference> (scalar AND format)

A163. <format-statement> ::= <statement-name> <format-specification-list>

3.2.21 Expression

A164. <expression> ::= {<value-reference> | <constant> | <infix-expression> |
<prefix-expression> | <parenthesized-expression>} <data-description>

A165. <infix-expression> ::= <expression> <infix-operator> <expression> <data-description>

A166. <infix-operator> ::= <or-logical> | <and-logical> | <or> | <xor> |
<and> | <gt> | <ge> | <eq> |
<le> | <lt> | <ne> | <cat> |
<add> | <subtract> | <multiply> | <divide> |
<power>

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A167. <prefix-expression> ::= <prefix-operator> <expression> <data-description>

A168. <prefix-operator> ::= <plus> | <minus> | <not>

A169. <parenthesized-expression> ::= <expression> <data-description>

3.2.22 Types of Reference

A170. <value-reference> ::= {<variable-reference> | <procedure-function-reference> | <builtin-function-reference> | <named-constant-reference> | <enquiry-reference>} <data-description>

A171. <variable-reference> ::= [<locator-qualifier>] <declaration-designator> (variable) [<identifier-list>] [<subscript-list>] <data-description>

A172. <locator-qualifier> ::= <value-reference> (scalar AND locator)

A173. <subscript> ::= <expression> (scalar AND integer) | <asterisk> | <integer-value>

A174. <procedure-function-reference> ::= <value-reference> [<argument-list>] <data-description>

A175. <argument> ::= <expression> [<dummy>] <data-description>

A176. <builtin-function-reference> ::= <builtin-function> [<argument-list>] <data-description>

A177. <builtin-function> ::=

<abs-bif>	<acos-bif>	<add-bif>	<addr-bif>
<asin-bif>	<atan-bif>	<atand-bif>	<atanh-bif>
<binary-bif>	<bit-bif>	<bool-bif>	<ceil-bif>
<character-bif>	<collate-bif>	<copy-bif>	<cos-bif>
<cosd-bif>	<cosh-bif>	<currentsize-bif>	<date-bif>
<datetime-bif>	<decimal-bif>	<dimension-bif>	<divide-bif>
<edit-bif>	<empty-bif>	<every-bif>	<exp-bif>
<exponent-bif>	<fileopen-bif>	<fixed-bif>	<float-bif>
<floor-bif>	<hbound-bif>	<high-bif>	<identical-bif>
<index-bif>	<isochar-bif>	<lbound-bif>	<length-bif>
<lineno-bif>	<log-bif>	<log10-bif>	<log2-bif>
<low-bif>	<max-bif>	<maxlength-bif>	<min-bif>
<mod-bif>	<multiply-bif>	<null-bif>	<offset-bif>
<oncode-bif>	<onfile-bif>	<onkey-bif>	<onsource-bif>
<pageno-bif>	<pointer-bif>	<prod-bif>	<reverse-bif>
<round-bif>	<search-bif>	<sign-bif>	<sin-bif>
<sind-bif>	<sinh-bif>	<some-bif>	<sqrt-bif>
<string-bif>	<substr-bif>	<subtract-bif>	<sum-bif>
<tally-bif>	<tan-bif>	<tand-bif>	<tanh-bif>
<time-bif>	<translate-bif>	<trim-bif>	<trunc-bif>
<unspec-bif>	<valid-bif>	<verify-bif>	

A178. <named-constant-reference> ::= <declaration-designator> (named-constant) [<subscript>] <data-description>

A179. <target-reference> ::= {<variable-reference> | <pseudo-variable-reference>} <data-description>

A180. <pseudo-variable-reference> ::= <pseudo-variable> [<argument-list>] <data-description>

A181. <pseudo-variable> ::= <exponent-pv> | <onsource-pv> | <pageno-pv> | <string-pv> | <substr-pv> | <unspec-pv>

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A182. <subroutine-reference> ::= <value-reference> [<argument-list>]

A183. <enquiry-reference> ::= {<declaresize-enquiry>
<data-description> <declaration-designator>} |
{<null-enquiry> <data-description>} |
{<maxval-enquiry> <data-type> <data-description>} |
{<minval-enquiry> <data-type> <data-description>} |
{<epsilon-enquiry> <data-type> <data-description>} |
{<radix-enquiry> <data-type> <data-description>}

3.2.23 Constant

A184. <constant> ::= ⟨basic-value⟩ <data-type>

3.2.24 Types of Value

A185. <identifier> ::= The category <identifier> is defined as a ⟨symbol-list⟩ corresponding to the sequence of characters in an ⟨identifier⟩. See rule CL8 in Chapter 2.

A186. <signed-integer> ::= { + | − } <integer>

A187. <integer> ::= The category <integer> is defined as a ⟨symbol-list⟩ corresponding to the sequence of characters in an ⟨integer⟩. See rule CL14 in Chapter 2.

3.2.25 Types of Picture

A188. <pictured-numeric> ::= <numeric-picture-specification> <arithmetic>

A189. <numeric-picture-specification> ::= ⟨picture-element-list⟩

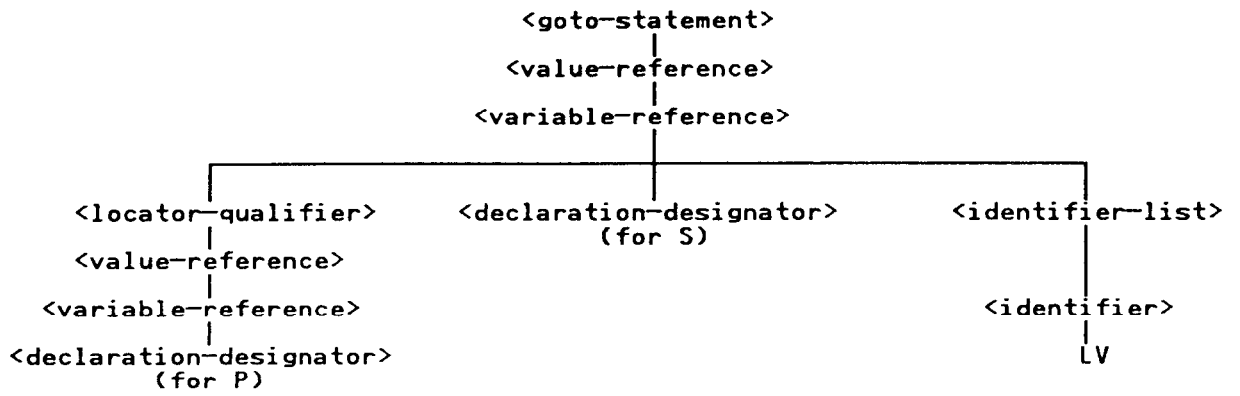
Note: The category ⟨picture-element⟩ is defined in Section 4.6.6, CREATE-PICTURE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The abstract text corresponding to the concrete representation

```
GO TO P -> S.LV;
```

is as follows:



Note: < value-reference > and < variable-reference > nodes immediately contain a < data-description > (not shown).

The abstract text corresponding to the concrete-representation

```
DECLARE Z EXTERNAL STATIC ALIGNED COMPLEX DECIMAL FLOAT(15);
```

is as follows:

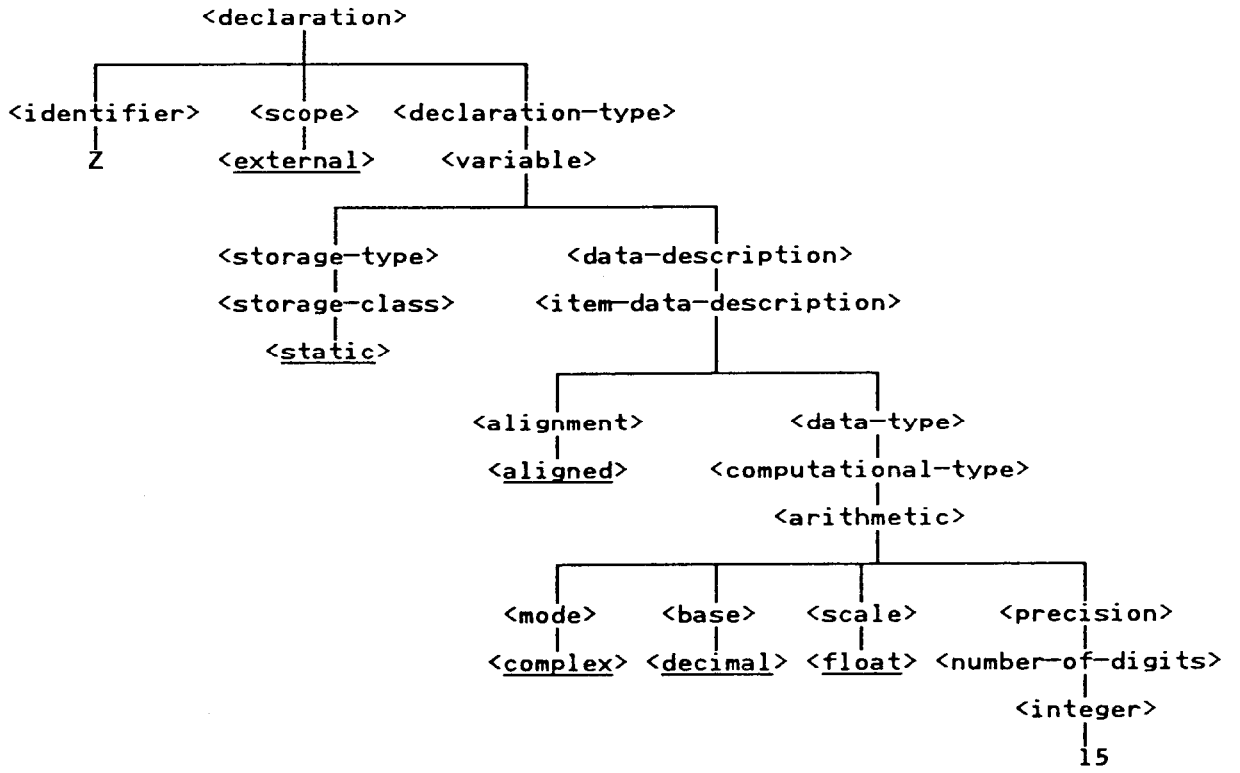


Figure 11. Examples of Abstract Text.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4. The Translator

4.1 Introduction

This Section defines an operation which translates a ⟨symbol-list⟩ into an <external-unit>. <external-unit>s represent the abstract form of an ⟨external-unit⟩. All the <external-unit>s that are part of a PL/I <program> are then combined. The <program> is used by the PL/I machine to determine the course of execution.

4.2 Translate

A *concrete-block* is a ⟨begin-block⟩, ⟨procedure⟩, ⟨package⟩, or ⟨external-unit⟩.

x is a *concrete-block-component* of y if y is a concrete-block, and y contains x but does not contain any other concrete-block which also contains x. In this case y *concrete-block-contains* x.

An *abstract-block* is a <begin-block>, <procedure>, <package>, or <external-unit>.

x is an *abstract-block-component* of y if y is an abstract-block, and y contains x but does not contain any other abstract-block which also contains x. In this case y *abstract-block-contains* x.

The informal term *block-component* is used for either concrete-block-component or abstract-block-component, and *block-contains* is used for either concrete-block-contains or abstract-block-contains where the context makes it obvious which formal term is required.

A ⟨declaration⟩ or ⟨description⟩,d *declaration-contains* a node n, if d contains n and d does not contain a ⟨description⟩ that also contains n.

A node, n, is a *declaration-component* of a node, d, if d declaration-contains n.

Operation: TRANSLATE(t)

Where: t is a ⟨symbol-list⟩.

Result: an <external-unit>.

Step 1. Perform PARSE(t,⟨translation-unit⟩) to obtain a ⟨translation-unit⟩,tu. Let p be the ⟨procedure⟩ or ⟨package⟩ immediately contained in tu.

Step 2. Perform VALIDATE-TRANSLATION-UNIT(tu).

Step 3. Attach an ⟨external-unit⟩,eu: p; to the ⟨translation-state⟩.

Step 4. Perform COMPLETE-EXTERNAL-UNIT.

Step 5. Let aeu be an <external-unit>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 6. For each ⟨declaration⟩,d which is a block-component of eu perform CREATE-DECLARATION(d) to obtain a <declaration>,ad, and append ad to the <declaration-list> in aeu.

Step 7. For each <declaration>,ad which is a block-component of aeu and which contains at least one ⟨precision-designator⟩, perform REPLACE-CONCRETE-PRECISION-DESIGNATORS(ad).

Step 8. For each <declaration>,ad which is a block-component of aeu and which contains at least one ⟨expression-designator⟩ or ⟨reference-designator⟩, perform REPLACE-CONCRETE-DESIGNATORS(ad).

Step 9.

Case 9.1. eu immediately contains a ⟨package⟩,pk.

Perform CREATE-PACKAGE(pk) to obtain a <package>,apk and attach apk to aeu.

Case 9.2. eu immediately contains a ⟨procedure⟩,p.

Perform CREATE-PROCEDURE(p) to obtain a <procedure>,ap and attach ap to aeu.

Step 10. Delete eu.

Step 11. Perform VALIDATE-EXTERNAL-UNIT(aeu).

Step 12. Return aeu.

4.3 Forming the Translation Unit

The PARSE operation is applied to a ⟨symbol-list⟩ to construct a complete tree with respect to the Concrete Syntax for a specified category-name. If this category-name is defined in the high-level syntax then some additional mapping at the interface between the low-level and high-level syntaxes is required.

Operation: PARSE(sl,n)

Where: sl is a ⟨symbol-list⟩,
n is a tree with a single node whose type is a category-name, tn, which is non-terminal with respect to the Concrete Syntax.

Result: a complete tree with respect to the Concrete Syntax and tn.

Case 1. tn is non-terminal with respect to the low-level syntax.

Perform LOW-LEVEL-PARSE(sl,n) to obtain nt. Return nt.

Case 2. tn is non-terminal with respect to the high-level syntax.

Step 2.1. Perform LOW-LEVEL-PARSE(sl,⟨pli-text⟩) to obtain a ⟨pli-text⟩,pt.

Step 2.2. Perform HIGH-LEVEL-PARSE(pt,n) to obtain nt.

Step 2.3. Return nt.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.3.1 Low-level-parse

Operation: LOW-LEVEL-PARSE(sl,n)

Where: sl is a ⟨symbol-list⟩,
n is a tree with a single node whose type is a category-name tn which is non-terminal with respect to the low-level syntax.

Result: a complete tree with respect to the low-level syntax and tn.

Step 1. There MUST exist one and only one tree, nt, which is a complete tree with respect to the low-level syntax and tn, such that the following conditions are true:

1. the concrete-representation of nt is exactly the same as the concrete-representation of sl, and
2. every occurrence of ⟨/*⟩ or ⟨*/⟩ in the concrete-representation of nt MUST be such that the ⟨/⟩ and ⟨*⟩ are nodes of a ⟨comment⟩ category or are contained in a ⟨non-delimiter⟩, and
3. of all possible trees satisfying conditions (1) and (2), nt is that one containing the least number of ⟨delimiter-pair⟩s and ⟨delimiter⟩s.

Step 2. Let dpl be the ⟨delimiter-pair-list⟩ in nt. Perform Steps 2.1 and 2.2 as long as there is a

```
⟨delimiter-pair⟩,dp:
   ⟨non-delimiter⟩:
      ⟨pic-spec⟩:
         ⟨pic-key⟩,pk
         ⟨simple-character-string-constant⟩,cs;
      ⟨delimiter⟩,dl;
```

in dpl.

Step 2.1. Let pkid be an ⟨identifier⟩ with same concrete representation as pk. Let dpl2 be a

```
⟨delimiter-pair-list⟩:
   ⟨delimiter-pair⟩:
      ⟨non-delimiter⟩:
         pkid;
      ⟨delimiter⟩:
         ⊬;;
   ⟨delimiter-pair⟩:
      ⟨non-delimiter⟩:
         cs;
      dl.
```

Step 2.2. Replace dp in dpl by the immediate components of dpl2 in the same order.

Note: The ⟨delimiter⟩: ⊬; in dpl2 is a definitional artifice. It is removed by Step 3 of HIGH-LEVEL-PARSE.

Step 3. Return nt.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.3.2 High-level-parse

A *keyword* is a category-name specified in the high-level syntax as a sequence of uppercase letters.

⟨delimiter-or-non-delimiter⟩ ::= ⟨delimiter⟩ | ⟨non-delimiter⟩

Operation: HIGH-LEVEL-PARSE(pt,cn)

Where: pt is a ⟨pli-text⟩,
cn is a tree with a single node whose type is a category-name tcn which is non-terminal with respect to the high-level syntax.

Result: a complete tree with respect to (a) the syntax composed of all the production-rules occurring in the high-level syntax and in the low-level syntax, and (b) tcn.

Step 1. Let t be a ⟨delimiter-or-non-delimiter-list⟩ which contains a copy of the ⟨delimiter⟩ and ⟨non-delimiter⟩ components of pt in the same order.

Step 2. Repeat Steps 2.1 through 2.5 as long as there is a ⟨delimiter-or-non-delimiter⟩,d: ⟨delimiter⟩: ⟨include⟩;; in t.

Step 2.1. Let s be ⟨symbol-list⟩: ⟨symbol⟩: ↯.

Step 2.2. Append to s any ⟨symbol⟩s obtained in an IMPLEMENTATION-DEFINED way from the ⟨text-name⟩ in d. Append ⟨symbol⟩: ↯; to s.

Step 2.3. Perform LOW-LEVEL-PARSE(s,⟨pli-text⟩) to obtain a ⟨pli-text⟩,tx.

Step 2.4. Let t1 be a ⟨delimiter-or-non-delimiter-list⟩ which contains a copy of the ⟨delimiter⟩ and ⟨non-delimiter⟩ components of tx in the same order.

Step 2.5. Replace d by the immediate components of t1 in the same order.

Step 3. Delete from t any ⟨delimiter⟩ containing a ↯ or a ⟨comment⟩. This MUST NOT cause t to be deleted.

Step 4. Let nt[i], i=1,...,n, be the ordered list of nodes which are the immediate components of the ⟨delimiter⟩s and ⟨non-delimiter⟩s in t.

Step 5. There MUST exist one and only one tree, ht, such that the following conditions are true:

1. ht is a complete tree with respect to the high-level syntax and tcn,

2. ht satisfies any additional constraints specified with that syntax, and

3. ht contains m terminal nodes nht[j], j=1,...,m and there is a one-to-one correspondence between the nt[i], i=1,...,n and nht[j], j=1,...,m taken in left-to-right order as specified by Case 5.1 through Case 5.5 except for the following instance:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If nht[j] is an ⟨environment-specification⟩ or an ⟨options-specification⟩ then it corresponds to k nodes nt[l], l=i,...,i+k−1 such that

a. no nt[l], l=i,...,i+k−1 is a ⟨;⟩;

b. among the nt[l], l=1,...,i+k−1, those nodes which are either ⟨(⟩ or ⟨)⟩ MUST match according to the usual rules for balancing parentheses.

Case 5.1. nht[j] is a keyword.

nt[i] MUST be an ⟨identifier⟩ containing the same terminals as the characters appearing either in the denotation of nht[j] or in the abbreviation for nht[j]. (See Section 2.7.)

Case 5.2. nht[j] is a non-bracketed category-name other than a keyword.

nht[j] and nt[i] MUST be equal.

Case 5.3. nht[j] is a ⟨radix-factor⟩.

nt[i] MUST be an ⟨identifier⟩ such that the ordered sequence, seq, of its terminals can be a denotation of an immediate component of a ⟨radix-factor⟩. Replace nht[j] by a ⟨radix-factor⟩: seq.

Case 5.4. nht[j] is a ⟨real-constant⟩ or an ⟨integer⟩.

nt[i] MUST be an ⟨arithmetic-constant⟩ containing just a node, lc, with the same type as nht[j]. (There may be intermediate nodes between nt[i] and lc, but no side branches.) Replace nht[j] by the tree with root-node lc.

Case 5.5. nht[j] is one of the following:

⟨identifier⟩
⟨arithmetic-constant⟩
⟨simple-bit-string-constant⟩
⟨simple-character-string-constant⟩

nt[i] and nht[j] MUST be of the same type. Replace nht[j] by nt[i].

Case 5.6. nht[j] is a ⟨letter⟩.

nt[i] MUST be an ⟨identifier⟩:⟨letter⟩,c. Replace nht[j] by c.

Step 6. In an IMPLEMENTATION-DEFINED manner, either perform Step 6.1 or go to Step 7.

Step 6.1. For each ⟨identifier⟩,id in ht replace each ⟨secondary-case-letter⟩,lc in id by the ⟨primary-case-letter⟩ which corresponds to lc.

Note: Programs intended for interchange among implementations MUST NOT depend on whether Step 6.1 is performed.

For example, the following program fragment may not be easily interchanged among implementations:

```
A:   PROCEDURE(B,b);
```

Step 1.1 of DECLARE-PARAMETERS will have a different effect if the ⟨identifier⟩s "B" and "b" are mapped to the same case or if they are left unmapped.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 7. For each tree in ht of the form

```
⟨ending⟩,end:
   [⟨statement-name⟩]
   ⟨end-statement⟩:
      END
      ⟨identifier⟩,idend;;,
```

in any order, perform Steps 7.1 and 7.2.

Step 7.1. Let b be the node in ht which immediately contains end.

Case 7.1.1. b is a ⟨group⟩ or ⟨selection⟩.

Let b1 be the ⟨executable-unit⟩ or ⟨balanced-unit⟩ in ht which immediately contains b.

Case 7.1.2. b is a ⟨begin-block⟩, ⟨procedure⟩ or ⟨package⟩.

Let b1 be b.

Step 7.2. b1 MUST immediately contain a ⟨statement-name⟩,sn or ⟨simple-statement-name⟩,sn. sn MUST contain an ⟨identifier⟩,idsn equal to idend. sn MUST NOT contain a ⟨signed-integer⟩.

Step 8. Return ht.

4.4 Validation of the Translation Unit

The concrete syntax is permissive in that some of the constructs permitted are used solely as a definitional artifice for historical reasons. This operation verifies that these constructs are not used in the translation-unit.

Operation: VALIDATE-TRANSLATION-UNIT(tu)

Where: tu is a ⟨translation-unit⟩.

Step 1. tu MUST NOT contain a ⟨data-attribute⟩ which immediately contains FORMAT, MEMBER, PACKAGE, PRECISION, or STRUCTURE.

Step 2. tu MUST NOT contain an ⟨attribute⟩ which immediately contains CONSTANT.

4.5 Completion of the External Unit

The ⟨external-unit⟩ is "completed" in the sense that all declarations are constructed or completed.

Operation: COMPLETE-EXTERNAL-UNIT

Step 1. Perform REORGANIZE.

Step 2. Perform CONSTRUCT-EXPLICIT-DECLARATIONS.

Step 3. Perform COMPLETE-STRUCTURE-DECLARATIONS.

Step 4. Perform CONSTRUCT-CONTEXTUAL-DECLARATIONS.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 5. Perform COMPLETE-DECLARATIONS.

Step 6. Perform VALIDATE-CONCRETE-DECLARATIONS.

4.5.1 Reorganize

The ⟨external-unit⟩ is reorganized in various ways to simplify and complete it.

Operation: REORGANIZE

Step 1. Perform COMPLETE-OPTIONS.

Step 2. Perform COMPLETE-ATTRIBUTE-IMPLICATIONS.

Step 3. For each ⟨declaration-commalist⟩,ds immediate component of a ⟨declare-statement⟩ or ⟨define-constant-statement⟩ component of the ⟨external-unit⟩ perform DEFACTOR-DECLARATIONS(ds) to obtain a ⟨declaration-commalist⟩,dds and replace ds by dds.

4.5.1.1 Complete-options

Various modifications are made to ⟨put-statement⟩s, ⟨get-statement⟩s, and ⟨format-statement⟩s to complete their options. These are performed before the creation of any contextual declarations (see Section 4.5.4, CONSTRUCT-CONTEXTUAL-DECLARATIONS).

Operation: COMPLETE-OPTIONS

Step 1. For each ⟨get-statement⟩,gs component of the ⟨external-unit⟩ if gs does not contain a ⟨file-option⟩ or a ⟨get-string⟩, then perform PARSE("FILE(SYSIN)",⟨file-option⟩) to obtain a ⟨file-option⟩,fo, and attach fo to gs.

Step 2. For each ⟨put-statement⟩,ps component of the ⟨external-unit⟩ if ps does not contain a ⟨file-option⟩ or a ⟨put-string⟩ then perform PARSE("FILE(SYSPRINT)",⟨file-option⟩) to obtain a ⟨file-option⟩,fo, and attach fo to ps.

Step 3. For each ⟨tab-format⟩,tf component of the ⟨external-unit⟩ if tf does not contain an ⟨expression⟩ then perform PARSE("TAB(1)", ⟨tab-format⟩) to obtain a ⟨tab-format⟩,ntf, and replace tf by ntf.

Step 4. For each ⟨skip-option⟩,sk component of the ⟨external-unit⟩ if sk does not contain an ⟨expression⟩ then perform PARSE("SKIP(1)",⟨skip-option⟩) to obtain a ⟨skip-option⟩,nsk, and replace sk by nsk.

Step 5. For each ⟨skip-format⟩,sf component of the ⟨external-unit⟩ if sf does not contain an ⟨expression⟩ then perform PARSE("SKIP(1)",⟨skip-format⟩) to obtain a ⟨skip-format⟩,nsf, and replace sf by nsf.

Step 6. For each ⟨radix-factor⟩,rf component of the ⟨external-unit⟩ if rf contains only B then replace rf by a ⟨radix-factor⟩: B1.

4.5.1.2 Complete-attribute-implications

The ⟨dimension-attribute⟩, ⟨precision⟩, and the ⟨data-attribute⟩: FIXED; can be implied without the use of the keywords DIMENSION, PRECISION, and FIXED. These implications are replaced by explicit declarations of these attributes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: COMPLETE-ATTRIBUTE-IMPLICATIONS

Step 1. For each ⟨dimension-suffix⟩,ds component of the ⟨external-unit⟩ such that ds is not a component of a ⟨dimension-attribute⟩ append a

```
⟨data-attribute⟩:
   ⟨dimension-attribute⟩:
      DIMENSION
      dsc;;
```

where dsc is a copy of ds, to the node immediately containing ds, and delete ds.

Step 2. For each ⟨attribute⟩: ⟨data-attribute⟩,atr; or ⟨data-attribute⟩,atr which is an immediate component of a list, l, component of the ⟨external-unit⟩ if atr simply contains a ⟨precision⟩,p but not PRECISION then append to l a

```
⟨data-attribute⟩:
   PRECISION
   p1;
```

where p1 is a copy of p, and delete p.

Step 3. For each ⟨data-attribute-list⟩,al or ⟨attribute-list⟩,al of the ⟨external-unit⟩ if ai simply contains a ⟨data-attribute⟩ with ⟨precision⟩,p and p contains a ⟨scale-factor⟩ then append to al a

```
⟨data-attribute⟩:
   FIXED.
```

4.5.1.3 Defactor-declarations

The syntax of ⟨declare-statement⟩ and of ⟨define-constant-statement⟩ allows ⟨identifier⟩s to be factored together to give them the same structuring or attributes. This factoring is unravelled to provide a single ⟨declaration⟩ for each ⟨identifier⟩.

Operation: DEFACTOR-DECLARATIONS(dc)

Where: dc is a ⟨declaration-commalist⟩.

Result: a ⟨declaration-commalist⟩.

Step 1. Let ds be a copy of dc.

Step 2. For each ⟨declaration⟩,d immediate component of ds, if d immediately contains a ⟨declaration-commalist⟩,dcl then perform DEFACTOR-DECLARATIONS(dcl) to obtain a ⟨declaration-commalist⟩,dcl1 and replace dcl by dcl1.

Step 3. For each ⟨declaration⟩,d immediate component of ds, if d immediately contains a ⟨declaration-commalist⟩,dl then perform Steps 3.1 through 3.3.

Step 3.1. If d immediately contains a ⟨level⟩,lv then there MUST NOT be a ⟨level⟩ declaration-contained in any of the ⟨declaration⟩s immediately contained in dl, and attach a copy of lv to each ⟨declaration⟩ immediate component of dl.

Step 3.2. If d immediately contains an ⟨attribute-list⟩,al then append a copy of each ⟨attribute⟩ of al to each ⟨declaration⟩ immediate component of dl.

Step 3.3. Replace d by the immediate components of dl in the same order.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4. Return ds.

4.5.2 Construct-explicit-declarations

The occurrence of an ⟨identifier⟩ in the ⟨external-unit⟩ as an immediate component of a ⟨declaration⟩ explicitly specifies that ⟨identifier⟩ as the name of some data item. Other contexts may also constitute explicit declarations of an ⟨identifier⟩, in which case a ⟨declaration⟩ is created for it.

⟨declared-statement-names⟩ ::= [⟨name-list⟩] {⟨package⟩ | ⟨procedure⟩ | ⟨begin-block⟩}.

⟨name⟩ ::= ⟨statement-name⟩ | ⟨simple-statement-name⟩ [⟨entry-information⟩]

Operation: CONSTRUCT-EXPLICIT-DECLARATIONS

Step 1. For each ⟨procedure⟩,p contained in the ⟨external-unit⟩,eu perform DECLARE-PARAMETERS(p).

Step 2.

Case 2.1. eu block-contains a ⟨package⟩,p.

Step 2.1.1. Perform DECLARE-STATEMENT-NAMES(p) to obtain a ⟨declared-statement-names⟩,dsn. Replace p by the ⟨package⟩ of dsn.

Step 2.1.2. Perform CONSTRUCT-STATEMENT-NAME-DECLARATIONS(nl,eattr) where eattr is an

⟨attribute⟩:
⟨data-attribute⟩:
PACKAGE;;

to obtain a ⟨unit⟩ containing a ⟨declaration-commalist⟩,dc.

Case 2.2. eu block-contains a ⟨procedure⟩,p.

Step 2.2.1. Perform DECLARE-STATEMENT-NAMES(p) to obtain a ⟨declared-statement-names⟩,dsn. Replace p by the ⟨procedure⟩ of dsn.

Step 2.2.2. Perform CONSTRUCT-STATEMENT-NAME-DECLARATIONS(nl,eattr) where eattr is an

⟨attribute⟩:
⟨data-attribute⟩:
ENTRY();;

to obtain a ⟨unit⟩ containing a ⟨declaration-commalist⟩,dc.

Step 3. Replace all occurrences of ⟨attribute⟩: INTERNAL; in dc by ⟨attribute⟩: EXTERNAL.

Step 4. Attach dc to eu.

4.5.2.1 Declare-parameters

Each ⟨identifier⟩ occurring in a ⟨parameter-name-commalist⟩ represents an explicit declaration of a parameter. Unless it is already declared, a declaration is introduced for it. If it has been declared erroneously then a conflict will occur when this declaration is being transformed to its abstract equivalent.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: DECLARE-PARAMETERS(p)

Where: p is a ⟨procedure⟩.

Step 1. For each ⟨parameter-name-commalist⟩,pl block-component of p perform Step 1.1.

Step 1.1. pl MUST be such that no two ⟨identifier⟩ components of pl are equal. For each ⟨identifier⟩,id component of pl perform Steps 1.1.1 through 1.1.3.

Step 1.1.1. Perform FIND-APPLICABLE-DECLARATION(id) to obtain d.

Step 1.1.2. d MUST NOT be ⟨absent⟩. d MUST be a concrete-block-component of p. d MUST NOT declaration-contain a ⟨level⟩ whose value is not 1.

Step 1.1.3. If d does not contain ⟨attribute⟩: PARAMETER; then attach ⟨attribute⟩: PARAMETER; to d.

4.5.2.2 Declare-statement-names

A ⟨statement-name⟩ or ⟨simple-statement-name⟩ may occur as a component of a ⟨unit⟩ which contains a ⟨format-statement⟩ or some other ⟨executable-unit⟩, or as a component of a ⟨package⟩ or ⟨procedure⟩. These contexts are used to determine the ⟨attribute⟩s to be attached to the explicit declarations for the ⟨identifier⟩ of the ⟨statement-name⟩ or ⟨simple-statement-name⟩.

Operation: DECLARE-STATEMENT-NAMES(p)

Where: p is a ⟨package⟩, ⟨procedure⟩, or ⟨begin-block⟩.

Result: a ⟨declared-statement-names⟩.

Step 1. Let pc be a copy of p and let lnl, pnl, fnl, and enl each be a ⟨name-list⟩ with no components.

Step 2. For each ⟨statement-name⟩,sn or ⟨simple-statement-name⟩,sn concrete-block-component of pc perform Step 2.1.

Step 2.1. One of the following cases MUST apply:

Case 2.1.1. sn is immediately contained in an ⟨executable-unit⟩,eu.

Append sn to lnl.

Case 2.1.2. sn is immediately contained in a ⟨unit⟩ that has a ⟨format-statement⟩ immediate component.

Append sn to fnl.

Case 2.1.3. sn is immediately contained in pc and pc is a ⟨procedure⟩.

Let ep be the ⟨procedure-statement⟩ in pc. ep has an ⟨entry-information⟩,ei. Append ⟨name⟩: sn ei; to enl.

Case 2.1.4. sn is immediately contained in pc and pc is a ⟨package⟩.

Append ⟨name⟩: sn; to enl.

Case 2.1.5. sn is immediately contained in pc and pc is a ⟨begin-block⟩.

No action.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 3. For each ⟨statement-name⟩,sn immediately contained in any ⟨begin-block⟩ concrete-block-component of pc, perform Step 3.1.

Step 3.1. Append sn to lnl.

Step 4. For each ⟨procedure⟩ or ⟨begin-block⟩, pn concrete-block-component of pc perform Step 4.1.

Step 4.1. Perform DECLARE-STATEMENT-NAMES(pn) to obtain a ⟨declared-statement-names⟩,dsn. Append the elements of the ⟨name-list⟩ of dsn to pnl. Replace pn by the ⟨procedure⟩ or ⟨begin-block⟩ of dsn.

Step 5. If lnl is not empty then perform CONSTRUCT-STATEMENT-NAME-DECLARATIONS(lnl,lattr), where lattr is an

```
⟨attribute⟩:
   ⟨data-attribute⟩:
      LABEL;;
```

to obtain a ⟨unit⟩,d, and append d to the ⟨unit-list⟩ of pc.

Step 6. If fnl is not empty then perform CONSTRUCT-STATEMENT-NAME-DECLARATIONS(fnl,fattr), where fattr is an

```
⟨attribute⟩:
   ⟨data-attribute⟩:
      FORMAT;;
```

to obtain a ⟨unit⟩,d and append d to the ⟨unit-list⟩ of pc.

Step 7. If pnl is not empty then perform CONSTRUCT-STATEMENT-NAME-DECLARATIONS(pnl,pattr), where pattr is an

```
⟨attribute⟩:
   ⟨data-attribute⟩:
      ENTRY();;
```

to obtain a ⟨unit⟩,d and append d to the ⟨unit-list⟩ of pc.

Step 8. Return the ⟨declared-statement-names⟩ consisting of enl and pc.

4.5.2.3 Construct-statement-name-declarations

This operation takes a ⟨name-list⟩ containing ⟨statement-name⟩s, ⟨simple-statement-name⟩s, and ⟨entry-information⟩s and constructs a ⟨declare-statement⟩ for them. The type of the ⟨statement-name⟩s is given by a supplied ⟨attribute⟩. A label ⟨statement-name⟩ may contain a ⟨signed-integer⟩, signifying that it is one element of an array of ⟨statement-name⟩s. The explicit declaration for this array is constructed with a ⟨dimension-attribute⟩ component.

Operation: CONSTRUCT-STATEMENT-NAME-DECLARATIONS(nl,att)

Where: nl is a ⟨name-list⟩,
att is an ⟨attribute⟩.

Result: a ⟨unit⟩.

Step 1. Let nlc be a copy of the ⟨name-list⟩,nl. Let un be a

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
⟨unit⟩:
   ⟨declare-statement⟩:
      DECLARE
      ⟨declaration-commalist⟩,dcl
      ⟨;⟩.
```

Step 2. While nlc contains any element, perform Steps 2.1 through 2.6.

Step 2.1. Let id be the ⟨identifier⟩ component of the first element of nlc.

Step 2.2. Let tnl be a ⟨name-list⟩ containing a copy of all elements of nlc whose ⟨statement-name⟩ or ⟨simple-statement-name⟩ immediately contains an ⟨identifier⟩ equal to id.

Step 2.3. Let d be a

```
⟨declaration⟩:
   id
   ⟨attribute-list⟩,al:
      ⟨attribute⟩:
         INTERNAL;
      ⟨attribute⟩:
         CONSTANT;
      att.
```

Step 2.4. If tnl contains more than one element or if tnl contains only one element and this element has a ⟨signed-integer⟩, then perform Steps 2.4.1 through 2.4.6.

Step 2.4.1. Let m be the number of elements of tnl. Each element of tnl MUST contain a ⟨signed-integer⟩. tnl MUST be such that no two ⟨signed-integer⟩ components of tnl represent the same numerical value.

Step 2.4.2. Let si[i] be the ⟨signed-integer⟩ of the *i*th element of tnl.

Step 2.4.3. Let ub and lb be, respectively, the largest and least value of si[1],...,si[m].

Step 2.4.4. Let bpl be a

```
⟨bound-pair⟩:
   ⟨lower-bound⟩:
      e1;
   ⟨:⟩
   ⟨upper-bound⟩:
      e2;;
```

where e1 and e2 are ⟨extent-expression⟩s representing lb and ub respectively.

Step 2.4.5. Let bpcl be a ⟨bound-pair-commalist⟩ containing bpl.

Step 2.4.6. Append an

```
⟨attribute⟩:
   ⟨data-attribute⟩:
      ⟨dimension-attribute⟩:
         DIMENSION
         ⟨dimension-suffix⟩:
            (
            bpcl
            );;;;
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

to al, the ⟨attribute-list⟩ of d.

Step 2.5. If dcl has a subcomponent, append ⟨,⟩ to dcl. Append d to dcl.

Step 2.6. Delete from nlc all those elements of which there is a copy in tnl.

Step 3. Return un.

4.5.3 Complete-structure-declarations

A structure is specified by a hierarchical set of names that refers to a group of individual items each of which may have a different data type. Conversely, an array is specified by a single name referring to a group of items all of the same data type. The component items of a structure may themselves be structures or arrays.

Operation: COMPLETE-STRUCTURE-DECLARATIONS

Step 1. For each ⟨declaration-commalist⟩ or ⟨description-commalist⟩,dl component of the ⟨external-unit⟩ perform DETERMINE-STRUCTURE(dl) to obtain a ⟨declaration-commalist⟩ or ⟨description-commalist⟩,dlm. Replace dl by dlm.

Step 2. For each ⟨declaration-commalist⟩ or ⟨description-commalist⟩, dl component of the ⟨external-unit⟩ that declaration-contains STRUCTURE or UNION, perform CONVERT-TO-LOGICAL-LEVELS(dl) to obtain dl1, a node of the same type as dl. Perform PROPAGATE-ALIGNMENT(dl1) to obtain dl2, a node of the same type as dl1. Replace dl by dl2.

4.5.3.1 Determine-structure

Operation: DETERMINE-STRUCTURE(cml)

Where: cml is a ⟨declaration-commalist⟩ or ⟨description-commalist⟩.

Result: a ⟨declaration-commalist⟩ or a ⟨description-commalist⟩.

Step 1. Let cmlc be a copy of cml. Let e[j] be the *j*th immediate component of cmlc that is not ⟨,⟩. Let n be the number of such components.

Step 2. For each e[j] that immediately contains a ⟨level⟩,lv, perform Steps 2.1 through 2.3.

Step 2.1. lv MUST NOT be 0.

Step 2.2. If j is less than n and e[j+1] immediately contains a ⟨level⟩ whose numeric value is greater than that of lv and e[j] does not declaration-contain UNION, then attach STRUCTURE to e[j].

Step 2.3. If the numeric value of lv is greater than one, attach MEMBER to e[j].

Step 3. For each e[j], all of the following MUST be false:

1. e[j] immediately contains ⟨level⟩ and does not declaration-contain UNION, STRUCTURE or MEMBER.

2. e[j] immediately contains ⟨level⟩ whose value is 1 and e[j] declaration-contains MEMBER.

3. e[j] declaration-contains UNION, STRUCTURE or MEMBER and does not immediately contain ⟨level⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4. e[j] declaration-contains UNION or STRUCTURE, and either j is equal to n or e[j+1] does not declaration-contain a ⟨level⟩ whose value is greater than the ⟨level⟩ declaration-component of e[j].

5. e[j] declaration-contains MEMBER and either j is equal to one or e[j−1] does not declaration-contain UNION, MEMBER or STRUCTURE.

6. e[j] declaration-contains both STRUCTURE and UNION.

Step 4. Return cmlc.

4.5.3.2 Convert-to-logical-levels

Operation: CONVERT-TO-LOGICAL-LEVELS(cml)

Where: cml is a ⟨declaration-commalist⟩ or ⟨description-commalist⟩.

Result: a ⟨declaration-commalist⟩ or ⟨description-commalist⟩.

Step 1. Let cmlc be a copy of cml. Let e[j] be the *j*th component of cmlc that is not ⟨,⟩ and let m be the number of such components. Let l[j] be the ⟨level⟩ declaration-contained in e[j], if it exists, and let n[j] be the numeric value of l[j].

Step 2.

Case 2.1. There exists an integer t less than m such that e[t] and e[t+1] each declaration-contains a ⟨level⟩, and n[t+1] is greater than n[t]+1.

Let j be the least such t.

Case 2.2. (Otherwise).

Return cmlc.

Step 3. Let j1 be the least integer greater than j such that either j1 is equal to m, or e[j1+1] does not declaration-contain MEMBER, or n[j1+1] is less than or equal to n[j]+1. For i=j+1,...,j1, perform Step 3.1.

Step 3.1. Replace l[i] by a ⟨level⟩ whose numeric value is n[i]−1.

Step 4. Go to Step 2.

4.5.3.3 Propagate-alignment

Operation: PROPAGATE-ALIGNMENT(cml)

Where: cml is a ⟨declaration-commalist⟩ or ⟨description-commalist⟩.

Result: a ⟨declaration-commalist⟩ or ⟨description-commalist⟩.

Step 1. Let cmlc be a copy of cml. Let e[j] be the *j*th component of cmlc that is not ⟨,⟩ and let m be the number of such components. Let l[j] be the ⟨level⟩ declaration-contained in e[j], if it exists, and let n[j] be the numeric value of l[j].

Step 2. For i=1,...,m, perform Steps 2.1 and 2.2.

Step 2.1. e[i] MUST NOT declaration-contain both ALIGNED and UNALIGNED.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.2.

Case 2.2.1. e[i] declaration-contains MEMBER, and e[i] declaration-contains neither ALIGNED nor UNALIGNED.

Let k be the greatest integer such that k is less than i and n[k] is equal to n[i]−1. If e[k] declaration-contains ALIGNED or UNALIGNED, then attach ALIGNED or UNALIGNED, respectively, to e[i].

Case 2.2.2. (Otherwise).

No action.

Step 3. For i=1,...,m, perform Steps 3.1 through 3.2.

Step 3.1. If e[i] contains STRUCTURE and ALIGNED, or STRUCTURE and UNALIGNED, delete ALIGNED or UNALIGNED, respectively, from e[i].

Step 3.2. If e[i] contains UNION and ALIGNED, or UNION and UNALIGNED, delete ALIGNED or UNALIGNED respectively from e[i].

Step 4. Return cmlc.

4.5.3.4 Find-applicable-declaration

Operation: FIND-APPLICABLE-DECLARATION(r,sb)

Where: r is an ⟨identifier⟩, ⟨unsubscripted-reference⟩, ⟨basic-reference⟩, or ⟨reference⟩,
sb is a ⟨begin-block⟩, ⟨procedure⟩, ⟨package⟩, or ⟨external-unit⟩, or is ⟪absent⟫.

Result: a ⟨declaration⟩ or ⟪absent⟫.

Step 1.

Case 1.1. r is an ⟨identifier⟩.

Let idl be an ⟨identifier-list⟩: r.

Case 1.2. r is an ⟨unsubscripted-reference⟩.

Append, in order, a copy of each ⟨identifier⟩ component of r to an ⟨identifier-list⟩,idl.

Case 1.3. r is a ⟨basic-reference⟩.

Append, in order, a copy of each ⟨identifier⟩ component of r which is not a component of an ⟨arguments⟩, in order, to an ⟨identifier-list⟩,idl.

Case 1.4. r is a ⟨reference⟩.

Let br be the ⟨basic-reference⟩ immediate component of r. Perform FIND-APPLICABLE-DECLARATION(br) to obtain d. Return d.

Step 2. If sb is ⟪absent⟫, let b be the concrete-block that block-contains r; otherwise, let b be sb. If r is contained in a ⟨statement-name⟩,sn or ⟨simple-statement-name⟩,sn then perform Step 2.1.

Step 2.1. If sn is immediately contained in a ⟨begin-block⟩, ⟨package⟩ or ⟨procedure⟩ then let b be the concrete-block which block-contains b.

Step 3. Let id be the rightmost ⟨identifier⟩ component of idl.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4.

Case 4.1. r is in a ⟨refer-option⟩.

Let dcl be the ⟨declaration-commalist⟩ which contains r and which is not contained in another ⟨declaration-commalist⟩. Let dl be a ⟨declaration-designator-list⟩ each component of which designates a ⟨declaration⟩ contained in dcl that contains an ⟨identifier⟩ that is equal to id.

Case 4.2. (Otherwise).

Let dl be a ⟨declaration-designator-list⟩ each component of which designates a ⟨declaration⟩ that is a concrete-block-component of b and that immediately contains an ⟨identifier⟩ that is equal to id.

Step 5. Delete from dl any component that designates a ⟨declaration⟩,d such that FIND-FULLY-QUALIFIED-NAME(d) returns an ⟨identifier-list⟩,didl such that idl does not contain an ordered sublist of the ⟨identifier⟩s contained in didl.

Step 6.

Case 6.1. dl is empty.

Case 6.1.1. r is in a ⟨refer-option⟩.

Return ≤absent≥.

Case 6.1.2. (Otherwise).

Let b be the concrete-block that block-contains b. If there is no such block then return ≤absent≥; otherwise, go to Step 4.

Case 6.2. dl contains a single component.

Let d be the ⟨declaration⟩ designated by the single component of dl. Return d.

Case 6.3. dl contains more than one component.

dl MUST contain exactly one component that designates a ⟨declaration⟩,d such that FIND-FULLY-QUALIFIED-NAME(d) returns an ⟨identifier-list⟩ equal to idl. Return d.

4.5.3.5 Find-fully-qualified-name

Each ⟨declaration⟩ has a fully qualified name associated with it. If it is an array or item declaration then this is a single ⟨identifier⟩. If the ⟨declaration⟩ is a member of a structure then it may have as many ⟨identifier⟩s as its logical level-number indicates its depth of embedding to be.

Operation: FIND-FULLY-QUALIFIED-NAME(d)

Where: d is a ⟨declaration⟩.

Result: an ⟨identifier-list⟩.

Case 1. d declaration-contains MEMBER.

Step 1.1. Let dcml be the ⟨declaration-commalist⟩ that immediately contains d. Let dl be the rightmost preceding ⟨declaration⟩ of dcml that declaration-contains a ⟨level⟩ whose numeric value is less than the numeric value of the ⟨level⟩ of d.

Step 1.2. Perform FIND-FULLY-QUALIFIED-NAME(dl) to obtain an ⟨identifier-list⟩,idl.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.3. Append the ⟨identifier⟩ immediate component of d to idl.

Step 1.4. Return idl.

Case 2. d does not declaration-contain MEMBER.

Step 2.1. Let id be the ⟨identifier⟩ immediate component of d.

Step 2.2. Return ⟨identifier-list⟩: id.

4.5.4 Construct-contextual-declarations

Certain contexts in a procedure specify the attributes of an identifier appearing in those contexts. If an identifier is not explicitly declared, but is used in such a context, then it is contextually declared and a declaration for it is introduced into the ⟨external-unit⟩. All the attributes implied by the context are added onto this generated declaration.

Operation: CONSTRUCT-CONTEXTUAL-DECLARATIONS

Step 1. Let eu be the ⟨external-unit⟩. Let u be a

```
⟨unit⟩:
   ⟨declare-statement⟩:
      DECLARE
      ⟨declaration-commalist⟩,dcml
      ⟨;⟩;;
```

where dcml contains no elements.

Step 2. For each ⟨identifier⟩,id in eu perform Steps 2.1 through 2.3.

Step 2.1. Let attrs be ⟪absent⟫.

Step 2.2.

Case 2.2.1. id is a component of a

```
⟨reference⟩,r:
   ⟨basic-reference⟩:
      ⟨identifier⟩,id;;
```

where r has no other components.

Step 2.2.1.1.

Case 2.2.1.1.1. r is an immediate component of a ⟨call-statement⟩.

Let attrs be an

```
⟨attribute-list⟩:
   ⟨attribute⟩:
      BUILTIN.
```

Note: This action and Case 3.3.2.3 of CREATE-REFERENCE are meant to illustrate the manner in which an implementation should support built-in subroutines (procedures which do not return values) as extensions to this standard. No built-in subroutines are defined by this standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.2.1.1.2. r is an immediate component of a ⟨file-option⟩ or ⟨named-io-condition⟩ and id contains in order precisely the ⟨symbol⟩s of the word SYSIN or the word SYSPRINT.

Let attrs be an

⟨attribute-list⟩:
⟨attribute⟩:
⟨data-attribute⟩:
FILE;;
⟨attribute⟩:
CONSTANT.

Case 2.2.1.1.3. (Otherwise).

No action.

Case 2.2.2. id is the only component of the ⟨basic-reference⟩ of a

⟨reference⟩:
⟨basic-reference⟩:
⟨identifier⟩,id;
⟨arguments-list⟩;

where the ⟨reference⟩ has no other immediate components.

Let attrs be an

⟨attribute-list⟩:
⟨attribute⟩:
BUILTIN.

Case 2.2.3. (Otherwise).

No action.

Step 2.3. If attrs is not ⟨absent⟩ then perform FIND-APPLICABLE-DECLARATION(id) to obtain d. If d is ⟨absent⟩ then perform Steps 2.3.1 through 2.3.3.

Step 2.3.1. Let d be a

⟨declaration⟩:
id
attrs.

Step 2.3.2. If an element of dcml immediately contains an ⟨identifier⟩ equal to id then the ⟨attribute-list⟩ of this ⟨declaration⟩ MUST equal attrs.

Step 2.3.3. If no element of dcml immediately contains an ⟨identifier⟩ equal to id then append d to dcml.

Step 3. If dcml contains any elements, append u to the ⟨unit-list⟩ of the ⟨procedure⟩ or ⟨package⟩ immediately contained in eu.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.5.5 Complete-declarations

Operation: COMPLETE-DECLARATIONS

Step 1. For each ⟨declaration⟩,d, or ⟨description⟩,d component of the ⟨external-unit⟩ perform Step 1.1.

Step 1.1. Perform TEST-ATTRIBUTE-CONSISTENCY(d) to obtain tv. tv MUST NOT be ≪false≫.

Step 2. For each ⟨declaration⟩,d component of the ⟨external-unit⟩ perform Steps 2.1 and 2.2.

Step 2.1.

Case 2.1.1. d was produced by CONSTRUCT-STATEMENT-NAME-DECLARATIONS, and d declaration-contains ENTRY.

Perform COMPLETE-CONSTRUCTED-ENTRY-DECLARATION(d).

Case 2.1.2. d is a component of a ⟨define-constant-statement⟩.

Step 2.1.2.1. If d declaration-contains neither ⟨attribute⟩: INTERNAL; nor ⟨attribute⟩: EXTERNAL; then append an ⟨attribute⟩: INTERNAL; to the ⟨attribute-list⟩ of d.

Step 2.1.2.2. If d declaration-contains a ⟨data-attribute⟩,da that contains BIT or CHARACTER without a ⟨maximum-length⟩, then attach a ⟨maximum-length⟩: ⟨*⟩; to da.

Note: d may not have a complete set of ⟨data-attribute⟩s (indeed, it may not even have any). Nevertheless, a complete <data-type> is obtained when d is translated to its abstract equivalent; the <data-type> will depend on the ⟨data-attribute⟩s (if any) in d and on the ⟨expression⟩ in the VALUE attribute in d. (See CREATE-NAMED-CONSTANT.)

Case 2.1.3. (Otherwise).

Perform APPLY-DEFAULTS(d).

Step 2.2. Perform CHECK-ATTRIBUTE-COMPLETENESS-AND-DELETE-ATTRIBUTES(d).

Step 3. For each ⟨arithmetic-enquiry-description⟩,aed component of the ⟨external-unit⟩ that is not contained in a ⟨declaration⟩, perform Steps 3.1 through 3.3.

Step 3.1. Let d be the ⟨description⟩ immediately contained in aed.

Step 3.2. Perform APPLY-DEFAULTS(d).

Step 3.3. Perform CHECK-ATTRIBUTE-COMPLETENESS-AND-DELETE-ATTRIBUTES(d).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.5.5.1 Test-attribute-consistency

Operation: TEST-ATTRIBUTE-CONSISTENCY(d,das)

Where: d is a ⟨declaration⟩ or ⟨description⟩,
das is an [⟨attribute⟩].

Result: ⟨true⟩ or ⟨false⟩.

Step 1. If das is ⟨absent⟩ then:

Case 1.1. d is a ⟨declaration⟩.

For each pair of ⟨attribute⟩s, a1 and a2, which are declaration-components of d, perform TEST-INVALID-DUPLICATES(a1,a2) to obtain tv. tv MUST NOT be ⟨true⟩.

Case 1.2. d is a ⟨description⟩.

For each pair of ⟨data-attribute⟩s, a1 and a2, which are declaration-components of d, perform TEST-INVALID-DUPLICATES(a1,a2) to obtain tv. tv MUST NOT be ⟨true⟩.

Step 2.

Case 2.1. das is ⟨absent⟩.

Let akl be an ⟨attribute-list⟩ consisting of a copy of each keyword declaration-component of d.

Case 2.2. das is not ⟨absent⟩.

Step 2.2.1. Let al be an ⟨attribute-list⟩ consisting of a copy of das, and each ⟨attribute⟩ or ⟨data-attribute⟩ declaration-component of d where d is a ⟨declaration⟩ or ⟨description⟩ respectively.

Step 2.2.2. For each pair of ⟨attribute⟩ components of al, a1 and a2, perform TEST-INVALID-DUPLICATES(a1,a2) to obtain tv. If tv is ⟨true⟩, return ⟨false⟩.

Step 2.2.3. Let akl be an ⟨attribute-list⟩ consisting of a copy of each keyword declaration-component of al.

Step 3. Replace all multiple occurrences of a given keyword in akl by a single occurrence of that keyword.

Step 4. If akl contains MEMBER and also contains EXTERNAL, return ⟨false⟩.

Step 5. If akl contains VALUE and d is a ⟨declaration⟩ contained in a ⟨declare-statement⟩, return ⟨false⟩.

Step 6. If akl does not contain VALUE and d is a ⟨declaration⟩ contained in a ⟨define-constant-statement⟩, return ⟨false⟩.

Step 7.

Case 7.1. d is a ⟨description⟩.

If the set of all keywords in akl is not a subset of the keywords that form the concrete-representation of some tree whose root-node is ⟨consistent-description⟩, return ⟨false⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 7.2. d is a partial ⟨declaration⟩ produced by the operation CREATE-CONSTANT.

If the set of all keywords in akl is not a subset of the keywords that form the concrete-representation of some tree whose root-node is ⟨consistent-literal-constant⟩, return ≪false≫.

Case 7.3. d is a ⟨declaration⟩.

If the set of all keywords in akl is not a subset of the keywords that form the concrete-representation of some tree whose root-node is ⟨consistent-declaration⟩, return ≪false≫.

Step 8. Return ≪true≫.

⟨consistent-declaration⟩ ::= ⟨scope⟩ ⟨declaration-type⟩

⟨scope⟩ ::= EXTERNAL | INTERNAL

⟨declaration-type⟩ ::= ⟨variable⟩ | ⟨named-constant⟩ | BUILTIN | CONDITION

⟨variable⟩ ::= VARIABLE ⟨storage-type⟩ [DIMENSION]
{⟨data-type⟩ [INITIAL] | STRUCTURE | UNION}

⟨storage-type⟩ ::= ⟨storage-class⟩ | DEFINED | PARAMETER | MEMBER

⟨storage-class⟩ ::= AUTOMATIC | BASED | STATIC [RESERVED]

⟨data-type⟩ ::= ⟨computational-type⟩ | ⟨non-computational-type⟩ ALIGNED

⟨computational-type⟩ ::= ⟨arithmetic⟩ ALIGNED | ⟨string⟩ | ⟨pictured⟩

⟨non-computational-type⟩ ::= AREA | ⟨entry⟩ | FILE | LABEL | ⟨locator⟩

⟨arithmetic⟩ ::= {FIXED | FLOAT}{ BINARY | DECIMAL} PRECISION

⟨string⟩ ::= {CHARACTER { VARYING | NONVARYING} | BIT NONVARYING} ⟨alignment⟩

⟨pictured⟩ ::= PICTURE UNALIGNED

⟨alignment⟩ ::= ALIGNED | UNALIGNED

⟨entry⟩ ::= ENTRY [RETURNS] [OPTIONS]

⟨locator⟩ ::= POINTER | OFFSET

⟨named-constant⟩ ::= CONSTANT
{PACKAGE | ⟨entry⟩ | FILE [⟨file-description-set⟩] |
FORMAT | [DIMENSION] LABEL |
VALUE {⟨arithmetic⟩ | CHARACTER | BIT }}

⟨file-description-set⟩ ::= [ENVIRONMENT] {⟨stream-set⟩ | ⟨record-set⟩}

⟨stream-set⟩ ::= STREAM {INPUT | OUTPUT [PRINT | NONPRINT]}

⟨record-set⟩ ::= RECORD {INPUT | OUTPUT | UPDATE}{ ⟨sequential-set⟩ | ⟨direct-set⟩}

⟨sequential-set⟩ ::= SEQUENTIAL [KEYED]

⟨direct-set⟩ ::= DIRECT KEYED

⟨consistent-literal-constant⟩ ::= CONSTANT {⟨arithmetic⟩ | BIT | CHARACTER }

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

⟨consistent-description⟩ ::= [DIMENSION] {⟨data-type⟩ | STRUCTURE | UNION} [MEMBER]

4.5.5.2 Test-invalid-duplicates

Operation: TEST-INVALID-DUPLICATES(a1,a2)

Where: a1 and a2 are ⟨attribute⟩s or ⟨data-attribute⟩s.

Result: ≪true≫ or ≪false≫.

Step 1. Let k1 and k2 be the leftmost keyword components of a1 and a2 respectively.

Step 2.

Case 2.1. k1 and k2 are equal.

Case 2.1.1. a1 and a2 have no terminal components other than k1 and k2.

Return ≪false≫.

Case 2.1.2. k1 (respectively, k2) is the only terminal component of a1 (respectively, a2) but k2 (respectively, k1) is not the only terminal component of a2 (respectively, a1).

Return ≪false≫.

Case 2.1.3. (Otherwise).

Return ≪true≫.

Case 2.2. k1 and k2 are not equal.

Return ≪false≫.

4.5.5.3 Apply-defaults

Operation: APPLY-DEFAULTS(d)

Where: d is a ⟨declaration⟩ or a ⟨description⟩.

Step 1. d MUST declaration-contain at least one of AREA, BINARY, BIT, BUILTIN, CHARACTER , CONDITION, CONSTANT, DECIMAL, ENTRY, FILE, FIXED, FLOAT, FORMAT , LABEL, OFFSET, PACKAGE, PICTURE, POINTER, or UNION.

Note: This requires that each non-structure declaration in a translation-unit contain at least one of the attributes in the list except CONSTANT, FORMAT, or PACKAGE.

Step 2. If d does not declaration-contain CONSTANT then perform Steps 2.1 through 2.5.

Step 2.1. If d does not declaration-contain PICTURE then perform APPLY-ATTRIBUTE(d,"FIXED") and perform APPLY-ATTRIBUTE(d,"BINARY").

Step 2.2. If d declaration-contains FIXED and BINARY then perform APPLY-ATTRIBUTE(d,"PRECISION(p1,0)"), where p1 is an IMPLEMENTATION-DEFINED integer.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.3. If d declaration-contains FIXED and DECIMAL then perform APPLY-ATTRIBUTE(d,"PRECISION(p2,0)"), where p2 is an IMPLEMENTATION-DEFINED integer.

Step 2.4. If d declaration-contains FLOAT and BINARY then perform APPLY-ATTRIBUTE(d,"PRECISION(p3)"), where p3 is an IMPLEMENTATION-DEFINED integer.

Step 2.5. If d declaration-contains FLOAT and DECIMAL then perform APPLY-ATTRIBUTE(d,"PRECISION(p4)"), where p4 is an IMPLEMENTATION-DEFINED integer.

Step 3. If d declaration-contains CHARACTER then perform APPLY-ATTRIBUTE(d,"CHARACTER(1)").

Step 4. If d declaration-contains BIT then perform APPLY-ATTRIBUTE(d,"BIT(1)").

Step 5. If d declaration-contains BIT or CHARACTER then perform APPLY-ATTRIBUTE(d,"NONVARYING").

Step 6. If d declaration-contains BIT or CHARACTER or PICTURE then perform APPLY-ATTRIBUTE(d,"UNALIGNED").

Step 7. If d declaration-contains AREA then perform APPLY-ATTRIBUTE(d,"AREA(s1)") where s1 is an IMPLEMENTATION-DEFINED integer.

Step 8. Perform APPLY-ATTRIBUTE(d,"ALIGNED").

Step 9. If d is a ⟨declaration⟩ then perform Steps 9.1 through 9.7 .

Step 9.1. If d declaration-contains ENTRY or FILE then perform APPLY-ATTRIBUTE(d,"CONSTANT").

Step 9.2. Perform APPLY-ATTRIBUTE(d,"VARIABLE").

Step 9.3. If

1. d declaration-contains CONDITION, or
2. d declaration-contains ENTRY and CONSTANT or FILE and CONSTANT, then

perform APPLY-ATTRIBUTE(d,"EXTERNAL").

Step 9.4. Perform APPLY-ATTRIBUTE(d,"INTERNAL").

Step 9.5. If declaration-contains EXTERNAL then perform APPLY-ATTRIBUTE(d,"STATIC").

Step 9.6. If d declaration-contains RESERVED then perform APPLY-ATTRIBUTE(d,"STATIC") and perform APPLY-ATTRIBUTE(d,"EXTERNAL").

Step 9.7. Let b be the block which block-contains d.

Case 9.7.1. b is a ⟨package⟩.

Perform APPLY-ATTRIBUTE(d,"STATIC").

Case 9.7.2. (Otherwise).

Perform APPLY-ATTRIBUTE(d,"AUTOMATIC").

Step 10. For each ⟨description⟩,d1 declaration-component of d, perform APPLY-DEFAULTS(d1).

Step 11. If there is a ⟨data-attribute⟩,att declaration-component of d, such that att declaration-contains RETURNS, but not as its sole terminal component, then delete all ⟨data-attribute⟩ declaration-components of d which declaration-contain RETURNS, except for att; otherwise,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

delete all but one of the ⟨data-attribute⟩ declaration-components of d which declaration-contain RETURNS.

Step 12. If d declaration-contains a ⟨returns-descriptor⟩,d1 which does not contain a ⟨description⟩, then perform Steps 12.1 through 12.3.

Step 12.1. Let desc be a ⟨description⟩ with no components.

Step 12.2. Perform APPLY-DEFAULTS(desc).

Step 12.3. If desc contains a subnode, then replace d1 by a

```
⟨returns-descriptor⟩:
   (
   ⟨description-commalist⟩:
      desc;
   ).
```

4.5.5.4 Apply-attribute

Operation: APPLY-ATTRIBUTE(d,ats)

Where: d is a ⟨declaration⟩ or a ⟨description⟩,
ats is a ⟨symbol-list⟩.

Step 1. Perform PARSE(ats,⟨attribute⟩) to obtain an ⟨attribute⟩,at.

Step 2. Perform TEST-ATTRIBUTE-CONSISTENCY(d,at) to obtain tv.

Step 3. If tv is ⟨true⟩ and d is a ⟨declaration⟩, then append to the ⟨attribute-list⟩ in d a copy of at.

Step 4. If tv is ⟨true⟩ and d is a ⟨description⟩, then append to the ⟨data-attribute-list⟩ in d a copy of the ⟨data-attribute⟩ immediately contained in at.

4.5.5.5 Complete-constructed-entry-declaration

Operation: COMPLETE-CONSTRUCTED-ENTRY-DECLARATION(d)

Where: d is a ⟨declaration⟩ containing ENTRY, produced by CONSTRUCT-STATEMENT-NAME-DECLARATIONS.

Step 1. Let e be the ⟨data-attribute⟩ declaration-component of d that immediately contains ENTRY.

Step 2. Let sn be the single occurrence of a

```
⟨simple-statement-name⟩:
   ⟨identifier⟩,id
   ⟨:⟩;
```

such that d is obtained by performing FIND-APPLICABLE-DECLARATION(id). Let n be 1.

Step 3. sn is immediately contained in a ⟨procedure⟩,p. Let es be the immediately contained ⟨procedure-statement⟩ of p.

Step 4.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 4.1. es contains a ⟨parameter-name-commalist⟩,pl.

Step 4.1.1. Let pn[j] be the ⟨identifier⟩ of the *j*th immediately contained ⟨parameter-name⟩ component of pl. Let m be the number of such components. For each j, j=1,...,m, taken in order, perform Steps 4.1.1.1 through 4.1.1.4.

Step 4.1.1.1. Perform FIND-APPLICABLE-DECLARATION(pn[j]) to obtain pd. pd MUST be a ⟨declaration⟩.

Step 4.1.1.2. Let dcml be the ⟨declaration-commalist⟩ that immediately contains pd. Let dcm[k] be the *k*th ⟨declaration⟩ component of dcml. Let there be kmx such components and let k be such that dcm[k] is identical to pd.

Step 4.1.1.3. Let knd be the minimum integer greater than or equal to k such that one of the following is true:

1. knd is equal to kmx;
2. dcm[knd+1] does not contain ⟨level⟩;
3. dcm[knd+1] contains a ⟨level⟩ whose ⟨integer⟩ has the value 1.

Step 4.1.1.4. For each inx, inx=k,...,knd, taken in order, perform Steps 4.1.1.4.1 through 4.1.1.4.5.

Step 4.1.1.4.1. Let pnd be a copy of dcm[inx]. Delete from pnd any occurrence of an

```
⟨attribute⟩:
   PARAMETER.
```

Step 4.1.1.4.2. For each ⟨reference⟩,nm contained in dcm[inx], and each ⟨identifier⟩,nm contained in dcm[inx] but not in a ⟨reference⟩ contained in dcm[inx], perform TEST-NAME-IN-DESCRIPTION(nm).

Step 4.1.1.4.3. If pnd contains a

```
⟨data-attribute⟩,da:
   ENTRY
   (
   ⟨description-commalist⟩
   );
```

then replace da with a

```
⟨data-attribute⟩:
   ENTRY.
```

Step 4.1.1.4.4. If pnd contains an

```
⟨attribute⟩,atr:
   ⟨data-attribute⟩:
      ⟨returns-descriptor⟩;;
```

then delete atr.

Step 4.1.1.4.5. Append, with intermediate ⟨,⟩s as required, to a ⟨description-commalist⟩,des, a ⟨description⟩ simply containing copies of the ⟨level⟩ and ⟨data-attribute⟩ simple components of pnd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 4.2. es does not contain a ⟨parameter-name-commalist⟩.

Let des be ⟪absent⟫.

Step 5. Let d1 be a copy of d.

Step 6. If es contains a ⟨returns-descriptor⟩, let rd be that ⟨returns-descriptor⟩ and append a copy of rd to the ⟨attribute-list⟩ of d1; otherwise, let rd be ⟪absent⟫.

Step 7. If es contains an ⟨options⟩ then let ops be that ⟨options⟩ and append a copy of ops to the ⟨attribute-list⟩ of d1; otherwise let ops be ⟪absent⟫.

Step 8.

Case 8.1. es contains an ⟨external-name⟩,en.

p MUST NOT be contained in a ⟨procedure⟩ or ⟨package⟩. Append an ⟨attribute⟩: EXTERNAL en; to the ⟨attribute-list⟩ of d1.

Case 8.2. (Otherwise).

Let en be ⟪absent⟫.

Step 9. Perform APPLY-DEFAULTS(d1).

Step 10. Perform CHECK-ATTRIBUTE-COMPLETENESS-AND-DELETE-ATTRIBUTES(d1).

Step 11. If d1 declaration-contains a ⟨returns-descriptor⟩,drd, then perform Steps 11.1 through 11.3.

Step 11.1. rd MUST NOT be ⟪absent⟫.

Step 11.2. Replace rd by drd.

Step 11.3. For each ⟨reference⟩,nm contained in rd, and each ⟨identifier⟩,nm contained in rd but not in a ⟨reference⟩ contained in rd, perform TEST-NAME-IN-DESCRIPTION(nm).

Step 12. If d1 declaration-contains an ⟨options⟩,dops, then perform Steps 12.1 through 12.3.

Step 12.1. ops MUST NOT be ⟪absent⟫.

Step 12.2. Replace ops by dops.

Step 12.3. For each ⟨reference⟩,nm contained in ops, and each ⟨identifier⟩,nm contained in ops but not in a ⟨reference⟩ contained in ops, perform TEST-NAME-IN-DESCRIPTION(nm).

Step 13. If d1 declaration-contains an ⟨external-name⟩,den then perform Steps 13.1 through 13.3.

Step 13.1. en MUST NOT be ⟪absent⟫.

Step 13.2. Replace en by den.

Step 13.3. For each ⟨reference⟩,nm contained in en and each ⟨identifier⟩,nm contained in en but not in a ⟨reference⟩ contained in en, perform TEST-NAME-IN-DESCRIPTION(nm).

Step 14. If des is not ⟪absent⟫, replace e with a

```
⟨data-attribute⟩:
  ENTRY
  (
  des
  ).
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 15. If rd is not ≤absent≥, append a copy of rd to the ⟨attribute-list⟩ of d.

Step 16. If ops is not ≤absent≥, append a copy of ops to the ⟨attribute-list⟩ of d.

Step 17. If en is not ≤absent≥, append an ⟨attribute⟩: EXTERNAL en; to the ⟨attribute-list⟩ of d.

4.5.5.6 Test-name-in-description

This operation tests whether a name in an attribute will have the same meaning if it is copied into an ENTRY declaration in the surrounding block.

Operation: TEST-NAME-IN-DESCRIPTION(nm)

Where: nm is a ⟨reference⟩ or an ⟨identifier⟩.

Step 1. Perform FIND-APPLICABLE-DECLARATION(nm) to obtain d, which MUST be a ⟨declaration⟩.

Step 2. If d is block-contained in the same block as nm, then perform Steps 2.1 through 2.2.

Step 2.1. Let b be the concrete-block that block-contains nm. Let sb be the concrete-block that block-contains b. Perform FIND-APPLICABLE-DECLARATION(nm,sb) to obtain dsb, which MUST be a ⟨declaration⟩.

Step 2.2. Both d and dsb MUST declaration-contain EXTERNAL or BUILTIN.

4.5.6 Validate-concrete-declarations

Various checks are applied to the ⟨external-unit⟩ to ensure that a valid set of declarations has been generated. Each declaration MUST be unique within its own scope. A check is also made to ensure that ⟨declaration⟩s of INTERNAL CONSTANT ENTRY, CONSTANT FORMAT, and CONSTANT LABEL were constructed by the operation CONSTRUCT-STATEMENT-NAME-DECLARATIONS.

Operation: VALIDATE-CONCRETE-DECLARATIONS

Step 1. The ⟨external-unit⟩ MUST NOT contain two ⟨declaration⟩s, d1 and d2, such that d1 and d2 are block-components of the same block, and FIND-FULLY-QUALIFIED-NAME(d1) yields the same result as FIND-FULLY-QUALIFIED-NAME(d2). (d1 and d2 are duplicate declarations).

Step 2. The ⟨external-unit⟩ MUST NOT contain a ⟨reference⟩,r, ⟨basic-reference⟩,r, or ⟨unsubscripted-reference⟩,r such that FIND-APPLICABLE-DECLARATION(r) yields ≤absent≥. (There is no ⟨declaration⟩ for r).

Step 3. The ⟨external-unit⟩ MUST NOT contain a ⟨declaration⟩,d that declaration-contains PACKAGE or LABEL and CONSTANT, or FORMAT and CONSTANT, or INTERNAL and ENTRY and CONSTANT, unless d was created by the operation CONSTRUCT-STATEMENT-NAME-DECLARATIONS.

Step 4. For each ⟨declaration⟩,d in the ⟨external-unit⟩ which is a block-component of a ⟨package⟩ perform Step 4.1.

Step 4.1. d MUST NOT declaration-contain DEFINED, PARAMETER, or AUTOMATIC.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.5.6.1 Check-attribute-completeness-and-delete-attributes

Operation: CHECK-ATTRIBUTE-COMPLETENESS-AND-DELETE-ATTRIBUTES(d)

Where: d is a ⟨declaration⟩ or a ⟨description⟩.

Step 1. For each distinct keyword, k which is a declaration-component of d perform Step 1.1.

Step 1.1. If there is an ⟨attribute⟩,att or a ⟨data-attribute⟩,att declaration-component of d, such that att declaration-contains k, but not as its sole terminal component, then delete all ⟨attribute⟩ or ⟨data-attribute⟩ declaration-components of d which declaration-contain k, except for att; otherwise, delete all but one of the ⟨attribute⟩ or ⟨data-attribute⟩ declaration-components of d which declaration-contain k.

Step 2. d MUST NOT declaration-contain an ⟨attribute⟩ or ⟨data-attribute⟩ with AREA, BIT, CHARACTER, DIMENSION, INITIAL, OPTIONS, PICTURE, PRECISION or RETURNS as its sole terminal node.

Step 3. If d declaration-contains EXTERNAL, it MUST NOT declaration-contain AUTOMATIC, BASED, DEFINED, PARAMETER, VALUE, or BUILTIN.

Step 4. If d declaration-contains RESERVED, it MUST also declaration-contain EXTERNAL.

Step 5. If d declaration-contains EXTERNAL and CONSTANT and ENTRY, it MUST NOT declaration-contain DIMENSION.

Step 6. If d is a ⟨declaration⟩ which declaration-contains RESERVED, BASED, DEFINED or PARAMETER (or declaration-contains MEMBER and the rightmost preceding ⟨declaration⟩ whose ⟨level⟩ has the value one declaration-contains RESERVED, BASED, DEFINED or PARAMETER), then d MUST NOT declaration-contain INITIAL.

Step 7. If d is a ⟨declaration⟩ which declaration-contains STATIC and either ENTRY or LABEL (or declaration-contains MEMBER and either ENTRY or LABEL, and the rightmost preceding ⟨declaration⟩ whose ⟨level⟩ has the value one declaration-contains STATIC), then d MUST NOT declaration-contain INITIAL.

Step 8. Let akl be an ⟨attribute-list⟩ consisting of copies of the keyword declaration-components of d.

Step 9. Delete from akl any keyword that can be contained in a tree whose root-node is ⟨file-description-set⟩.

Step 10.

Case 10.1. d is a ⟨description⟩.

There MUST exist a tree whose root-node is ⟨consistent-description⟩ and whose concrete-representation consists of the same set of keywords as are contained in akl.

Case 10.2. d is a ⟨declaration⟩.

If d is contained in a ⟨declare-statement⟩, then there MUST exist a tree whose root-node is ⟨consistent-declaration⟩ and whose concrete representation consists of the same set of keywords as are contained in akl.

Step 11. For each ⟨description⟩,d1 declaration-component of d, perform CHECK-ATTRIBUTE-COMPLETENESS-AND-DELETE-ATTRIBUTES(d1).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6 Create-abstract-equivalent-tree

Where the Concrete and Abstract Syntaxes are similar, a simple transformation generates a specific abstract tree from the given concrete one. Essentially this consists of ignoring those concrete terminals which represent the "punctuation" of the concrete program, and transforming each concrete node into its abstract equivalent. Where the syntaxes are not similar an operation specifies the translation.

Operation: CREATE-ABSTRACT-EQUIVALENT-TREE(ct)

Where: ct is a tree belonging to a category of the Concrete Syntax.

Result: a tree belonging to a related category of the Abstract Syntax, or ≤none≥.

Case 1. ct is an ⟨allocation⟩, ⟨array-init⟩, ⟨assignment-statement⟩, ⟨begin-block⟩, ⟨constant⟩, ⟨expression⟩, ⟨format-iteration⟩, ⟨freeing⟩, ⟨group⟩, ⟨identifier⟩, ⟨if-statement⟩, ⟨initial⟩, ⟨initial-element⟩, ⟨integer⟩, ⟨leave-statement⟩, ⟨on-statement⟩, ⟨open-statement⟩, ⟨picture⟩, ⟨remote-format⟩, or ⟨selection⟩.

Perform CREATE-F(ct) to obtain abt, where "⟨f⟩" is the name of the category of ct. Return abt.

Case 2. ct belongs to a terminal category.

Case 2.1. ct is a keyword and there is a terminal category whose name is of the form "<cn>" where cn is the lowercase equivalent of category name of ct (e.g., if ct is FIXED, cn is "fixed").

Return a <cn>.

Case 2.2. ct is an *.

Return an <asterisk>.

Case 2.3. (Otherwise).

Return ≤none≥.

Case 3. The category-name of ct is of the form "⟨cn-list⟩" or "⟨cn-commalist⟩", where "cn" is some name.

Let x be a <cn-list>. For each ⟨cn⟩,y, in ct, taken in order, perform CREATE-ABSTRACT-EQUIVALENT-TREE(y) to obtain z, and append z to x. Return x.

Case 4. ct is a ⟨radix-factor⟩.

Let d be the digit in the denotation of the immediate component of ct. Return <radix-factor>: d.

Case 5. ct is a ⟨condition-name⟩.

Perform CREATE-CONDITION(ct) to obtain abt. Return abt.

Case 6. (Otherwise).

Step 6.1. Let "cn" be the name such that "⟨cn⟩" is the category-name of ct. Let x be a <cn>. If <cn> is a terminal category, return x.

Step 6.2. For each immediate subnode, y, of ct, taken in order, perform Step 6.2.1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 6.2.1.

Case 6.2.1.1. y is a ⟨reference⟩.

Let ref be a <variable-reference>, or a <value-reference>, or a <target-reference>, or a <subroutine-reference>, choosing the alternative that permits ref to be attached to x as an immediate subnode. Perform CREATE-REFERENCE(y,ref) to obtain z. Attach z to x.

Case 6.2.1.2. y is an ⟨unsubscripted-reference⟩.

Perform CREATE-REFERENCE(y,<variable-reference>) to obtain z. Attach z to x.

Case 6.2.1.3. (Otherwise).

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(y) to obtain z. If z is not equal to ⟨none⟩, attach z to x.

Step 6.3. Return x.

4.6.1 Creation of Blocks and Groups

4.6.1.1 Create-package

Operation: CREATE-PACKAGE(cpk)

Where: cpk is a ⟨package⟩.

Result: a <package>.

Step 1. cpk immediately contains a ⟨package-statement⟩,cpks. Let ex be the ⟨exports-option⟩ of cpks. Let apk be a <package>: <procedure-list>.

Step 2.

Case 2.1. ex has an ⟨exported-entry-commalist⟩,exentcl.

exentcl MUST be such that no two ⟨identifier⟩s in exentcl are equal. For each ⟨exported-entry⟩,exent in exentcl, in any order, perform Steps 2.1.1 through 2.1.4.

Step 2.1.1. Let id be the ⟨identifier⟩ in exent. Perform FIND-APPLICABLE-DECLARATION(id) to obtain a ⟨declaration⟩,d, or ⟨absent⟩,d.

Step 2.1.2. d MUST NOT be ⟨absent⟩. d MUST have been produced by CONSTRUCT-STATEMENT-NAME-DECLARATIONS. d MUST declaration-contain ENTRY.

Step 2.1.3.

Case 2.1.3.1. exent contains an ⟨external-name⟩,en.

Let aext be an

```
⟨attribute⟩:
   EXTERNAL
   (
   en
   ).
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.1.3.2. (Otherwise).

Let aext be an

⟨attribute⟩:
EXTERNAL.

Step 2.1.4. d has an ⟨attribute⟩,aint: INTERNAL. Replace aint by aext.

Case 2.2. ex has ⟨*⟩.

For each ⟨declaration⟩,d such that

1. d is a block-component of cpk,
2. d was produced by CONSTRUCT-STATEMENT-NAME-DECLARATIONS, and
3. d declaration-contains ENTRY,

in any order, perform Step 2.2.1.

Step 2.2.1. d has an ⟨attribute⟩,aint: INTERNAL. Replace aint by an

⟨attribute⟩:
EXTERNAL.

Step 3. Perform CREATE-BLOCK(cpk,apk).

Step 4. If cpk immediately contains a ⟨statement-condition-prefix⟩,scp then perform CREATE-CONDITION-PREFIX-LIST(scp) to obtain a <condition-prefix-list>,cpl and attach cpl to apk.

Step 5. If cpks has an ⟨options⟩, attach an implementation-dependent tree, op, of type <options> to apk.

Step 6. Return apk.

4.6.1.2 Create-procedure

Operation: CREATE-PROCEDURE(cp)

Where: cp is a ⟨procedure⟩.

Result: a <procedure>.

Step 1. cp immediately contains a ⟨procedure-statement⟩,cps. Perform CREATE-ENTRY-POINT(cps) to obtain an <entry-point>,ep. Let ap be a

<procedure>:
ep.

Step 2. Perform CREATE-BLOCK(cp,ap).

Step 3. If cp immediately contains a ⟨statement-condition-prefix⟩,scp then perform CREATE-CONDITION-PREFIX-LIST(scp) to obtain a <condition-prefix-list>,cpl and attach cpl to ap.

Step 4. If cps simply contains RECURSIVE, then attach <recursive> to ap.

Step 5. Return ap.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.1.3 Create-begin-block

A <begin-block> is constructed in much the same way as a <procedure>, which it resembles except for the presence of any entry or return information.

Operation: CREATE-BEGIN-BLOCK(cbb)

Where: cbb is a ⟨begin-block⟩.

Result: a <begin-block>.

Step 1. Let abb be a <begin-block>. If the ⟨begin-statement⟩ in cbb contains an ⟨options⟩, attach an implementation-dependent tree of type <options> to abb.

Step 2. Perform CREATE-BLOCK(cbb,abb).

Step 3. If cbb immediately contains a ⟨statement-condition-prefix⟩,scp then perform CREATE-CONDITION-PREFIX-LIST(scp) to obtain a <condition-prefix-list>,cpl and attach cpl to abb.

Step 4. Return abb.

4.6.1.4 Create-block

Operation: CREATE-BLOCK(cb,ab)

Where: cb is a ⟨package⟩, ⟨procedure⟩, or ⟨begin-block⟩,
ab is a <package>, <procedure>, or <begin-block>.

Step 1. For each ⟨declaration⟩,d that is a block-component of cb and that does not declaration-contain MEMBER perform CREATE-DECLARATION(d) to obtain a <declaration>,ad, and append ad to the <declaration-list> immediately contained in ab.

Step 2. For each <named-constant>,nc component of ab such that nc immediately contains a ⟨declaration-designator⟩,dd perform REPLACE-CONCRETE-NAMED-CONSTANT-DESIGNATORS(nc).

Step 3. For each <declaration>,ad component of ab such that ad contains at least one ⟨precision-designator⟩, perform REPLACE-CONCRETE-PRECISION-DESIGNATORS(ad).

Step 4. For each <declaration>,ad component of ab such that ad contains at least one ⟨expression-designator⟩, ⟨reference-designator⟩, or ⟨initial-designator⟩ perform REPLACE-CONCRETE-DESIGNATORS(ad).

Step 5. For each ⟨procedure⟩,p that is a block-component of cb perform CREATE-PROCEDURE(p) to obtain a <procedure>,ap and append ap to the <procedure-list> in ab.

Step 6. For each ⟨format-statement⟩,fs that is a block-component of cb, perform CREATE-FORMAT-STATEMENT(fs) to obtain a <format-statement>,afs and append afs to the <format-statement-list> in ab.

Step 7. If cb immediately contains a ⟨unit-list⟩, let ul be that ⟨unit-list⟩. Otherwise let ul be ≤absent≥. If cb is a ⟨procedure⟩ or ⟨begin-block⟩ then perform CREATE-EXECUTABLE-UNIT-LIST(ul) to obtain an <executable-unit-list>,eul and attach eul to ab.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.1.5 Replace-concrete-designators

Operation: REPLACE-CONCRETE-DESIGNATORS(ad)

Where: ad is a <declaration> or a <data-description>.

Step 1. For each {expression-designator},ed component of ad, perform Steps 1.1 and 1.2.

Step 1.1. Let e be the {expression} designated by ed.

Step 1.2. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(e) to obtain an <expression>,ae and replace ed by ae.

Step 2. For each {reference-designator},rd component of ad, perform Steps 2.1 and 2.2.

Step 2.1. Let r be the {reference} designated by rd.

Case 2.1.1. rd is an immediate component of <based>.

Perform CREATE-REFERENCE(r,<value-reference>) to obtain a <value-reference>,vr.

Case 2.1.2. rd is an immediate component of <base-item> or <offset>.

Perform CREATE-REFERENCE(r,<variable-reference>) to obtain a <variable-reference>,vr.

Step 2.2. Replace rd by vr.

Step 3. For each {initial-designator},ides component of ad, perform Step 3.1.

Step 3.1. Let int be the {initial} designated by ides. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(int) to obtain an <initial>,i and replace int by i.

4.6.1.6 Replace-concrete-precision-designators

Operation: REPLACE-CONCRETE-PRECISION-DESIGNATORS(ad)

Where: ad is a <declaration> or a <data-description>

Step 1. For each {precision-designator},pd component of ad, perform Steps 1.1 through 1.3.

Step 1.1. Let p be the {precision} designated by pd, and let dt be the <data-type> that simply contains pd.

Step 1.2. Perform CREATE-PRECISION(p,dt) to obtain a <precision>,ap. Replace pd by ap.

Step 1.3. Perform CHECK-ARITHMETIC-TYPE(dt).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.1.7 Replace-concrete-named-constant-designators

Operation: REPLACE-CONCRETE-NAMED-CONSTANT-DESIGNATORS(nc)

Where: nc is a <named-constant> that immediately contains a ⟨declaration-designator⟩,dd.

Step 1. Let d be the ⟨declaration⟩ designated by dd. d declaration-contains an ⟨attribute⟩,at containing VALUE.

Step 2. Let ce be the ⟨expression⟩ in at. Perform CREATE-EXPRESSION(ce) to obtain an <expression>,ae.

Step 3. Perform EVALUATE-RESTRICTED-EXPRESSION(ae) to obtain c, which MUST be a <constant> having <computational-type>. Let bv and dt be the ⩽basic-value⩾ and <data-type>, respectively, in c.

Step 4. If d does not declaration-contain a ⟨data-attribute⟩, then let ct be the <arithmetic> or <string> immediate component of the <computational-type> of dt, and let nnc be a <named-constant>: ct bv. Otherwise perform CONVERT-NAMED-LITERAL(d,dt,bv) to obtain a <named-constant>,nnc. Replace nc by nnc.

4.6.1.8 Create-group

Operation: CREATE-GROUP(g)

Where: g is a ⟨group⟩.

Result: a <group>.

Step 1. Let ds be the ⟨do-statement⟩ in g. If g immediately contains a ⟨unit-list⟩, let ul be that ⟨unit-list⟩. Otherwise let ul be ⩽absent⩾. Perform CREATE-EXECUTABLE-UNIT-LIST(ul) to obtain an <executable-unit-list>,eul.

Case 1.1. ds immediately contains a ⟨do-spec⟩,dsp.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(dsp) to obtain a <do-spec>,adsp. Return a <group>: <iterative-group>: <controlled-group>: adsp eul.

Case 1.2. ds immediately contains a ⟨while-option⟩ or ⟨until-option⟩ or both.

Step 1.2.1. Let g be a <group>: <iterative-group>: <conditional-group>: eul.

Step 1.2.2. If ds immediately contains a ⟨while-option⟩,wo then perform CREATE-ABSTRACT-EQUIVALENT-TREE(wo) to obtain a <while-option>,awo and attach awo to g.

Step 1.2.3. If ds immediately contains a ⟨until-option⟩,uo then perform CREATE-ABSTRACT-EQUIVALENT-TREE(uo) to obtain a <until-option>,auo and attach auo to g.

Case 1.3. ds immediately contains LOOP.

Return a <group>: <iterative-group>: <looping-group>: eul.

Case 1.4. (Otherwise).

Return a <group>: <non-iterative-group>: eul.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.1.9 Create-selection

Operation: CREATE-SELECTION(cs)

Where: cs is a ⟨selection⟩.

Result: a <selection>.

Step 1. Let as be a <selection>. Let awul be a <when-unit-list>.

Step 2. If the ⟨select-statement⟩ in cs has an ⟨expression⟩,ce, perform CREATE-EXPRESSION(ce) to obtain an <expression>,ae and attach ae to as.

Step 3. If cs immediately contains a ⟨when-statement-list⟩,wsl, then for each ⟨when-statement⟩,ws in wsl, taken in order, perform Steps 3.1 through 3.6.

Step 3.1. Let awu be a <when-unit>.

Step 3.2.

Case 3.2.1. The ⟨when-clause⟩ in ws immediately contains ALL.

Let wv be a <when-all-values>.

Case 3.2.2. (Otherwise).

Let wv be a <when-any-values>.

Step 3.3. For each ⟨expression⟩,ce in the ⟨expression-commalist⟩ in the ⟨when-clause⟩ of ws, taken in order, perform Step 3.3.1.

Step 3.3.1. Perform CREATE-EXPRESSION(ce) to obtain an <expression>,ae. Append ae to wv.

Step 3.4. Attach wv to awu.

Step 3.5. Let ceu be the ⟨executable-unit⟩ in ws. Perform CREATE-EXECUTABLE-UNIT(ceu) to obtain an <executable-unit>,aeu. Attach aeu to awu.

Step 3.6. Append awu to awul.

Step 4. If awul has at least one component, attach awul to as.

Step 5. If cs contains an ⟨otherwise-statement⟩,oth, then perform Step 5.1.

Step 5.1. Let ceu be the ⟨executable-unit⟩ in oth. Perform CREATE-EXECUTABLE-UNIT(ceu) to obtain an <executable-unit>, aeu. Attach to as a <otherwise-unit>:aeu.

Step 6.

Case 6.1. as immediately contains an <expression>,se.

se and all the <expression>s in all the <when-any-values> or <when-all-values> in the <when-unit-list> of as MUST have <data-type>s which

1. all have <arithmetic>, or
2. all have <bit>, or
3. all have <character>, or
4. all have <non-computational-type>, with the immediate subnodes of the <non-computational-type>s all belonging to the same category other than <area>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 6.2. (Otherwise).

All the <expression>s in all the <when-any-values> or <when-all-values> in the <when-unit-list> of as MUST have <data-type>s which have <bit>.

Step 7. Return as.

4.6.1.10 Create-executable-unit-list

Operation: CREATE-EXECUTABLE-UNIT-LIST(ul)

Where: ul is a [⟨unit-list⟩].

Result: an <executable-unit-list>.

Step 1. Let eul be an <executable-unit-list>. If ul is not ⟨absent⟩ then for each ⟨unit⟩,u in ul, taken in order, perform Step 1.1.

Step 1.1. If u immediately contains an ⟨executable-unit⟩,eu, perform CREATE-EXECUTABLE-UNIT(eu) to obtain an <executable-unit>,aeu and append aeu to eul.

Step 2. Append an <executable-unit>: <end-statement>; to eul, and return eul.

4.6.1.11 Create-executable-unit

Operation: CREATE-EXECUTABLE-UNIT(eu)

Where: eu is ⟨executable-unit⟩.

Result: an <executable-unit>.

Step 1. If eu immediately contains an ⟨executable-single-statement⟩,es, let st be the immediate component of es; otherwise let st be the immediate component of eu that is not a ⟨statement-name⟩.

Step 2. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(st) to obtain ast. Let aeu be an <executable-unit>: ast.

Step 3. If eu immediately contains a ⟨statement-name⟩,sn or eu immediately contains a ⟨begin-block⟩ which immediately contains a ⟨statement-name⟩,sn then perform CREATE-STATEMENT-NAME(sn) to obtain a <statement-name>,asn and attach asn to aeu.

Step 4. Return aeu.

4.6.1.12 Create-entry-point

Operation: CREATE-ENTRY-POINT(es)

Where: es is a ⟨procedure-statement⟩.

Result: an <entry-point>.

Step 1. Let ep be an <entry-point>: <entry-information>,ei. Let sn be the ⟨simple-statement-name⟩ immediately contained in the ⟨procedure⟩ immediately containing es.

Step 2. Perform CREATE-STATEMENT-NAME(sn) to obtain a <statement-name>,asn and attach asn to ep.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 3. Let eid be the ⟨identifier⟩ in sn. Perform FIND-APPLICABLE-DECLARATION(eid) to obtain a ⟨declaration⟩,d. Let ed be the <declaration> whose ⟨declaration-designator⟩ designates d. Let e be the <entry> of ed. If e immediately contains a <parameter-descriptor-list>,pdl, then let npd be the number of <parameter-descriptor>s in pdl and let epdd[i], i=1,...,npd, be the <data-description> in the *i*th <parameter-descriptor> in pdl. If e immediately contains a <returns-descriptor>, then let erdd be the <data-description> in that <returns-descriptor>. If e immediately contains an <options>, then let edop be that <options>.

Step 4. If es has a ⟨parameter-name-commalist⟩,pnl, then perform Steps 4.1 and 4.2.

Step 4.1. For each i, i=1,...,npd, perform Step 4.1.1.

Step 4.1.1. Let pnid be the ⟨identifier⟩ in the *i*th ⟨parameter-name⟩ in pnl. Perform FIND-APPLICABLE-DECLARATION(pnid) to obtain a ⟨declaration⟩,pnd. Let apnd be the <declaration> whose ⟨declaration-designator⟩ designates pnd. Let pndd be the <data-description> in apnd. For each <extent-expression>,pexp in pndd, let epexp be corresponding <extent-expression> in epdd[i] and perform TEST-DESCRIPTOR-EXTENT-EXPRESSIONS(pexp,epexp).

Step 4.2. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(pnl) to obtain a <parameter-name-list>,apnl and attach apnl to ei.

Step 5. If es has a ⟨returns-descriptor⟩,rd, then perform Steps 5.1 and 5.2.

Step 5.1. Let drd be the only ⟨description⟩ in rd. Perform CREATE-DATA-DESCRIPTION(drd) to obtain a <data-description>,dd. Perform REPLACE-CONCRETE-PRECISION-DESIGNATORS(dd). Perform REPLACE-CONCRETE-DESIGNATORS(dd). Let rdd be a <returns-descriptor>: dd; and attach rdd to ei.

Step 5.2. For each <extent-expression>,rexp in dd, let erexp be the corresponding <extent-expression> in erdd and perform TEST-DESCRIPTOR-EXTENT-EXPRESSIONS(rexp,erexp).

Step 6. If es has an ⟨options⟩, attach an implementation-dependent tree, op, of type <options> to ei. For each <extent-expression>,oexp in op, let eoexp be the corresponding <extent-expression> in edop and perform TEST-DESCRIPTOR-EXTENT-EXPRESSIONS(oexp,eoexp).

Step 7. Return ep.

4.6.1.13 Create-statement-name

Operation: CREATE-STATEMENT-NAME(pl)

Where: pl is a ⟨statement-name⟩ or ⟨simple-statement-name⟩.

Result: a <statement-name>.

Step 1. Let id be the ⟨identifier⟩ in sn. Perform CREATE-IDENTIFIER(id) to obtain an <identifier>,ad. Let asn be a <statement-name>: ad. If sn has a ⟨signed-integer⟩,si perform Step 1.1.

Step 1.1. Let asi be a <signed-integer> whose concrete-representation is the same as that of si. Attach asi to asn.

Step 2. Return asn.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.1.14 Create-condition-prefix-list

Operation: CREATE-CONDITION-PREFIX-LIST(scp)

Where: scp is a ⟨statement-condition-prefix⟩.

Result: a <condition-prefix-list>.

Step 1. Let cpl be a <condition-prefix-list>.

Step 2. For each ⟨computational-condition⟩,cc in scp, perform Step 2.1.

Step 2.1. Perform CREATE-CONDITION(cc) to obtain a <computational-condition>,acc. Append a <condition-prefix>: acc <enabled>; to cpl.

Step 3. For each ⟨disabled-computational-condition⟩,dcc, in scp, perform Step 3.1.

Step 3.1. Perform CREATE-CONDITION(dcc) to obtain a <computational-condition>,acc. cpl MUST NOT contain <condition-prefix>: acc <enabled>. Append a <condition-prefix>: acc <disabled>; to cpl.

Step 4. Return cpl.

4.6.1.15 Create-condition

Operation: CREATE-CONDITION(c)

Where: c is a ⟨computational-condition⟩, ⟨disabled-computational-condition⟩, ⟨condition-name⟩, or ⟨io-condition⟩.

Result: a <computational-condition>, <io-condition>, or <condition-name>.

Case 1. c is a ⟨computational-condition⟩.

Return a <computational-condition>: <x-condition>; where "x" is the lowercase equivalent of the concrete-representation of c.

Case 2. c is a ⟨disabled-computational-condition⟩.

Return a <computational-condition>: <x-condition>; where x is a name such that "nox" is the lowercase equivalent of the concrete-representation of c.

Case 3. c is an ⟨io-condition⟩.

Return an <io-condition>: <x-condition>; where "x" is the lowercase equivalent of the concrete-representation of c.

Case 4. c is a ⟨condition-name⟩ that contains AREA, ERROR, FINISH, or STORAGE.

Return a <condition-name>: <x-condition>; where "x" is the lowercase equivalent of the concrete-representation of c.

Case 5. c is a ⟨condition-name⟩: ⟨computational-condition⟩,cc.

Perform CREATE-CONDITION(cc) to obtain a <computational-condition>,acc, and return a <condition-name>: acc.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 6. c is a ⟨condition-name⟩: ⟨named-io-condition⟩: ⟨io-condition⟩,ioc (⟨reference⟩,ref).

Perform CREATE-CONDITION(ioc) to obtain an <io-condition>,aioc, and perform CREATE-REFERENCE(ref,<value-reference>) to obtain a <value-reference>,vr. Return a <condition-name>: <named-io-condition>: aioc vr.

Case 7. c is a ⟨condition-name⟩: ⟨programmer-named-condition⟩.

Let id be the ⟨identifier⟩ in c. Perform FIND-APPLICABLE-DECLARATION(id) to obtain a ⟨declaration⟩,dd. Let des be a <declaration-designator> designating the <declaration> whose ⟨declaration-designator⟩ designates dd. Return a <condition-name>: <programmer-named-condition>: des.

4.6.1.16 Test-descriptor-extent-expressions

Operation: TEST-DESCRIPTOR-EXTENT-EXPRESSIONS(e1,e2)

Where: e1 and e2 are <extent-expression>s.

Step 1. Let ex1 be the <expression> in e1. Let ex2 be the <expression> in e2.

Step 2. Perform EVALUATE-RESTRICTED-EXPRESSION(ex1) and EVALUATE-RESTRICTED-EXPRESSION(ex2) to obtain v1 and v2 respectively. v1 and v2 MUST both be <constant>s having <computational-type>. Perform EVALUATE-EXPRESSION-TO-INTEGER(ex1) and EVALUATE-EXPRESSION-TO-INTEGER(ex2) to obtain ⟨integer-value⟩s i1 and i2 respectively. i1 and i2 MUST be equal.

4.6.2 Creation of Statements

4.6.2.1 Create-assignment-statement

Operation: CREATE-ASSIGNMENT-STATEMENT(ast)

Where: ast is an ⟨assignment-statement⟩.

Result: an <assignment-statement>.

Step 1. Let aast be an <assignment-statement>. Let r be the ⟨reference⟩ immediately contained in ast. Perform CREATE-REFERENCE(r,<target-reference>) to obtain a <target-reference>,tr, and attach tr to aast. Let e be the ⟨expression⟩ immediately contained in ast. Perform CREATE-EXPRESSION(e) to obtain an <expression>,ae, and attach ae to aast.

Step 2. Let aedd be the <data-description> of ae. Let trdd be the <data-description> of tr. One of the following conditions MUST exist:

1. aedd is proper for assignment to trdd, or
2. a. aedd immediately contains an <item-data-description>,aeidd, and
   b. trdd immediately contains a <dimensioned-data-description>: <element-data-description>: <item-data-description>,tridd, and
   c. aedd is proper for assignment to a <data-description> immediately contained in tridd.

Step 3. Return aast.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.2.2 Data-descriptions Proper for Assignment

The <data-description>,dds is *proper for assignment* to the <data-description>,ddt if and only if the following conditions exist:

1. the ⋖aggregate-type⋗ of dds is compatible (see Section 9.2.1.4) with the ⋖aggregate-type⋗ of ddt,
2. if the common ⋖aggregate-type⋗ of ddt and dds immediately contains ⋖scalar⋗ then the <data-type>s of dds and ddt:

   a. both have <computational-type>, or
   b. both have <non-computational-type>, with the immediate subnodes of the <non-computational-type>s belonging to the same category;

   otherwise, corresponding <data-type>s of dds and ddt are equal, ignoring subnodes of <area-size> and <maximum-length>.

4.6.2.3 Create-allocation

Operation: CREATE-ALLOCATION(al)

Where: al is an ⊰allocation⊱.

Result: an <allocation>.

Step 1. Let id be the ⊰identifier⊱ immediately contained in al. Perform FIND-APPLICABLE-DECLARATION(id) to obtain a ⊰declaration⊱,cdcl. cdcl MUST NOT declaration-contain MEMBER.

Step 2. Let adcl be the <declaration> whose ⊰declaration-designator⊱ designates cdcl. If adcl contains a <based>,b and al does not contain a ⊰set-option⊱, then b MUST immediately contain a <value-reference> that immediately contains a <variable-reference>.

Step 3. Let des be a <declaration-designator> designating adcl. Let aal be an <allocation>: des. If al contains a ⊰set-option⊱,sp, then perform CREATE-ABSTRACT-EQUIVALENT-TREE(sp) to obtain a <set-option>,asp, and attach asp to aal. If al contains an ⊰in-option⊱,io, then perform CREATE-ABSTRACT-EQUIVALENT-TREE(io) to obtain an <in-option>,aio, and attach aio to aal.

Step 4. Return aal.

4.6.2.4 Create-format-statement

Operation: CREATE-FORMAT-STATEMENT(fs)

Where: fs is a ⊰format-statement⊱.

Result: a <format-statement>.

Step 1. Let fsl be the ⊰format-specification-commalist⊱ component of fs. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(fsl) to obtain a <format-specification-list>,afsl.

Step 2. The ⊰unit⊱ that immediately contains fs immediately contains a ⊰simple-statement-name⊱,sn. Perform CREATE-STATEMENT-NAME(sn) to obtain a <statement-name>,asn. Let afs be a <format-statement>: asn afsl.

Step 3. Return afs.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.2.5 Create-remote-format

Operation: CREATE-REMOTE-FORMAT(rf)

Where: rf is a ⟨remote-format⟩.

Result: a <remote-format>.

Step 1. Let id be the ⟨identifier⟩ in rf. Perform CREATE-REFERENCE(id,<value-reference>) to obtain a <value-reference>,vr.

Step 2. Return a <remote-format>: vr.

4.6.2.6 Create-format-iteration

Operation: CREATE-FORMAT-ITERATION(fi)

Where: fi is a ⟨format-iteration⟩.

Result: a <format-iteration>.

Step 1. Let afi be a <format-iteration>.

Step 2. Let ff be the ⟨format-iteration-factor⟩ immediately contained in fi. If ff immediately contains an ⟨expression⟩, let e be that ⟨expression⟩; otherwise let e be an ⟨expression⟩ to which the immediately contained ⟨integer⟩ of ff has been attached. Perform CREATE-EXPRESSION(e) to obtain an <expression>,ae and attach a <format-iteration-factor>: ae; to afi.

Step 3.

Case 3.1. fi immediately contains a ⟨format-specification-commalist⟩,fsc.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(fsc) to obtain a <format-specification-list>,fl, and attach fl to afi.

Case 3.2. fi immediately contains a ⟨format-item⟩,ft.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(ft) to obtain a <format-item>,ft and attach a <format-specification-list>: <format-specification>: ft;; to afi.

Step 4. Return afi.

4.6.2.7 Create-freeing

Operation: CREATE-FREEING(fr)

Where: fr is a ⟨freeing⟩.

Result: a <freeing>.

Step 1. Let id be the ⟨identifier⟩ immediately contained in fr. Perform FIND-APPLICABLE-DECLARATION(id) to obtain a ⟨declaration⟩,cdcl. cdcl MUST NOT declaration-contain MEMBER.

Step 2. Let adcl be the <declaration> whose ⟨declaration-designator⟩ designates cdcl.

Step 3. Let des be a <declaration-designator> designating adcl. Let afr be a <freeing>: des. If fr immediately contains a ⟨locator-qualifier⟩ with ⟨reference⟩,r, perform

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CREATE-REFERENCE(r,<value-reference>) to obtain a <value-reference>,vr and attach a <locator-qualifier>: vr; to afr. If afr contains an ⟨in-option⟩,io, perform CREATE-ABSTRACT-EQUIVALENT-TREE(io) to obtain an <in-option>,aio, and attach aio to afr. Return afr.

4.6.2.8 Create-if-statement

Operation: CREATE-IF-STATEMENT(ifs)

Where: ifs is an ⟨if-statement⟩.

Result: an <if-statement>.

Step 1. ifs immediately contains an ⟨if-clause⟩ which contains an ⟨expression⟩,e. Perform CREATE-EXPRESSION(e) to obtain an <expression>,ae.

Step 2.

Case 2.1. ifs immediately contains an ⟨executable-unit⟩,eu, but not an ELSE.

Perform CREATE-EXECUTABLE-UNIT(eu) to obtain an <executable-unit>,aeu. Return an

<if-statement>:
  <test>: ae;
  <then-unit>: aeu.

Case 2.2. ifs immediately contains a ⟨balanced-unit⟩,bu, an ELSE, and an ⟨executable-unit⟩,eu.

Perform CREATE-BALANCED-UNIT(bu) to obtain an <executable-unit>,eu1. Perform CREATE-EXECUTABLE-UNIT(eu) to obtain an <executable-unit>,eu2. Return an

<if-statement>:
  <test>: ae;
  <then-unit>: eu1;
  <else-unit>: eu2.

4.6.2.9 Create-balanced-unit

Operation: CREATE-BALANCED-UNIT(bu)

Where: bu is a ⟨balanced-unit⟩.

Result: an <executable-unit>.

Step 1.

Case 1.1. bu immediately contains an ⟨executable-single-statement⟩,ess.

Let s be the immediate component of ess. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(s) to obtain stat.

Case 1.2. bu immediately contains a ⟨group⟩, ⟨selection⟩, ⟨begin-block⟩, or ⟨on-statement⟩.

Let s be that ⟨group⟩, ⟨selection⟩, ⟨begin-block⟩, or ⟨on-statement⟩. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(s) to obtain stat.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1.3. (Otherwise).

Let e be the ⟨expression⟩ immediately contained in the ⟨if-clause⟩ immediately contained in bu. Perform CREATE-EXPRESSION(e) to obtain an <expression>,ae. Let bu1 and bu2 be, in order, the ⟨balanced-unit⟩s immediately contained in bu. Perform CREATE-BALANCED-UNIT(bu1) to obtain eu1 and CREATE-BALANCED-UNIT(bu2) to obtain eu2. Let stat be an

<if-statement>:
  <test>: ae;
  <then-unit>: eu1;
  <else-unit>: eu2.

Step 2. Let eu be an <executable-unit>: stat. If bu immediately contains a ⟨statement-name⟩,sn or bu immediately contains a ⟨begin-block⟩ which immediately contains a ⟨statement-name⟩,sn, perform Step 2.1.

Step 2.1. Perform CREATE-STATEMENT-NAME(sn) to obtain a <statement-name>,asn and attach asn to eu.

Step 3. Return eu.

4.6.2.10 Create-leave-statement

Operation: CREATE-LEAVE-STATEMENT(ls)

Where: ls is a ⟨leave-statement⟩.

Result: a <leave-statement>.

Case 1. ls contains an ⟨identifier⟩,id.

Step 1.1. Perform FIND-APPLICABLE-DECLARATION(id) to obtain a ⟨declaration⟩,cdcl. Let adcl be the <declaration> whose ⟨declaration-designator⟩ designates cdcl. adcl MUST contain <named-constant>.

Step 1.2. Perform CREATE-NAMED-CONSTANT-REFERENCE(adcl) to obtain a <named-constant-reference>,ncr.

Step 1.3. Return a

<leave-statement>:
  ncr.

Case 2. ls does not contain an ⟨identifier⟩.

Return a

<leave-statement>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.2.11 Create-on-statement

Operation: CREATE-ON-STATEMENT(os)

Where: os is an ⟨on-statement⟩.

Result: an <on-statement>.

Step 1.

Case 1.1. os immediately contains SYSTEM.

Let p be a <system-action>.

Case 1.2. os immediate contains an ⟨on-unit⟩,ou.

Step 1.2.1.

Case 1.2.1.1. ou immediately contains an ⟨executable-single-statement⟩,ess.

Let s be the immediate component of ess. s MUST NOT be a ⟨leave-statement⟩. ess MUST NOT immediately contain a ⟨statement-name⟩.

Case 1.2.1.2. ou immediately contains a ⟨begin-block⟩.

Let s be that ⟨begin-block⟩. s MUST NOT immediately contain a ⟨statement-name⟩.

Step 1.2.2. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(s) to obtain sa. For each <return-statement>,rs contained in sa, sa MUST contain a <procedure> that contains rs. Let ep be an <entry-point> whose only subnode is <entry-information>. Let eul be an

```
<executable-unit-list>:
   <executable-unit>:
      sa;
   <executable-unit>:
      <end-statement>.
```

Step 1.2.3. Let p be an <on-unit>: <procedure>: ep eul.

Step 2. Let aos be an <on-statement>: p. If os immediately contains SNAP attach <snap> to aos. For each ⟨condition-name⟩,cn in the ⟨condition-name-commalist⟩ in os, perform CREATE-CONDITION(cn) to obtain a <condition-name>,acn, and append acn to the <condition-name-list> in aos. Return aos.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.2.12 Create-open-statement

Operation: CREATE-OPEN-STATEMENT(os)

Where: os is an ⟨open-statement⟩.

Result: an <open-statement>.

Step 1. Let soc be the ⟨single-opening-commalist⟩ of os. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(soc) to create a <single-opening-list>,asoc.

Step 2. For each <single-opening>,so in asoc, perform Steps 2.1 through 2.4.

Step 2.1. If so contains <stream> or <print>, then so MUST NOT contain <record>, <direct>, <sequential>, <update>, or <keyed>.

Step 2.2. If so contains <print>, then so MUST NOT contain <input>.

Step 2.3. If so contains <direct>, then so MUST NOT contain <sequential>.

Step 2.4. so MUST contain at most one of <input>, <output>, or <update>.

Step 3. Return an <open-statement>:asoc.

4.6.3 Create-declaration

Operation: CREATE-DECLARATION(d)

Where: d is a ⟨declaration⟩.

Result: a <declaration>.

Step 1. Let cid be the ⟨identifier⟩ of d. Perform CREATE-IDENTIFIER(cid) to obtain an <identifier>,id.

Step 2.

Case 2.1. d contains INTERNAL.

Let sc be <scope>: <internal>.

Case 2.2. d contains EXTERNAL.

Let sc be <scope>: <external>.

Step 3.

Case 3.1. d contains VARIABLE.

Perform CREATE-VARIABLE(d) to obtain a <variable>,v. Let dt be <declaration-type>: v.

Case 3.2. d contains CONSTANT or VALUE.

Perform CREATE-NAMED-CONSTANT(d) to obtain a <named-constant>,nc. Let dt be <declaration-type>: nc.

Case 3.3. d contains BUILTIN.

Let dt be a <declaration-type>: <builtin>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3.4. d contains CONDITION.

Let dt be a <declaration-type>: <condition>.

Step 4. Let dd be a {declaration-designator} that designates d.

Step 5. Return <declaration>: id sc dt dd.

4.6.3.1 Create-named-constant

Operation: CREATE-NAMED-CONSTANT(d)

Where: d is a {declaration}.

Result: a <named-constant>.

Step 1. Let nc be a <named-constant>.

Step 2. If d declaration-contains a {dimension-attribute},da then perform CREATE-BOUND-PAIR-LIST(da) to obtain a <bound-pair-list>,bpl containing one <bound-pair>,bp and attach bp to nc.

Step 3.

Case 3.1. d declaration-contains ENTRY.

Perform CREATE-ENTRY(d) to obtain an <entry>,ae. Attach ae to nc.

Case 3.2. d declaration-contains FILE.

Attach <file> to nc. Let fd be a <file-description-set>. For each {attribute} component of d which declaration-contains STREAM, RECORD, INPUT, OUTPUT, UPDATE , SEQUENTIAL, DIRECT, PRINT, NONPRINT, KEYED, or ENVIRONMENT attach <stream>, <record>, <input>, <output>, <update>, <sequential>, <direct>, <print>, <nonprint>, <keyed>, or <environment>, respectively, to fd. Attach fd to nc. If <environment> was attached then perform some IMPLEMENTATION-DEFINED action.

Case 3.3. d declaration-contains FORMAT or LABEL.

Attach <format> or <label> respectively, to nc.

Case 3.4. d declaration-contains an {attribute} containing VALUE.

Let dd be a {declaration-designator} designating d. Let nc be a <named-constant>: dd.

Case 3.5. d declaration-contains PACKAGE.

Attach <package-constant> to nc.

Step 4. Return nc.

4.6.3.2 Create-variable

For each distinct variable a <declaration> which contains <variable> is constructed and completed according to the declared attributes of the item. A <variable> may be referred to by a <value-reference>, a <target-reference>, or a <subroutine-reference> (see Section 4.6.5).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: CREATE-VARIABLE(d)

Where: d is a ⟨declaration⟩.

Result: a <variable>.

Step 1.

Case 1.1. d contains AUTOMATIC.

Let st be a <storage-type>: <storage-class>: <automatic>.

Case 1.2. d contains STATIC.

Let st be a <storage-type>: <storage-class>: <static>. If d also contains RESERVED, attach <reserved> to st.

Case 1.3. d contains an ⟨attribute⟩,atr that immediately contains BASED.

Let st be a <storage-type>: <storage-class>: <based>. If atr also simply contains a ⟨reference⟩,r then attach a ⟨reference-designator⟩: designator of r; to st.

Note: The translation of r will be completed after the processing of all ⟨declare-statement⟩s.

Case 1.4. d contains DEFINED (⟨reference⟩,r).

Let st be

<storage-type>:
<defined>:
<base-item>:
⟨reference-designator⟩:
designator of r.

Case 1.5. d contains ⟨attribute⟩: PARAMETER.

Let st be a <storage-type>: <parameter>.

Step 2. Perform CREATE-DATA-DESCRIPTION(d) to obtain a <data-description>,dd.

Step 3. Return a <variable>: st dd.

4.6.3.3 Create-bound-pair-list

Operation: CREATE-BOUND-PAIR-LIST(da)

Where: da is a ⟨dimension-attribute⟩.

Result: a <bound-pair-list>.

Step 1. Let bpl be the ⟨bound-pair-commalist⟩ component of da.

Step 2. For each ⟨bound-pair⟩,bp of bpl such that bp contains an ⟨upper-bound⟩ and does not contain a ⟨lower-bound⟩, attach a ⟨lower-bound⟩: un; to bp, where un is a copy of an ⟨extent-expression⟩ whose concrete representation is the character ⟨1⟩.

Step 3. Let abpl be a <bound-pair-list>. For each ⟨bound-pair⟩,bp component of bpl in left-to-right order perform Step 3.1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3.1.

Case 3.1.1. bp contains *.

Append a <bound-pair>: <asterisk>; to abpl.

Case 3.1.2. (Otherwise).

Step 3.1.2.1. Let abp be a

<bound-pair>:
<lower-bound>,alb
<upper-bound>,aup.

Attach to alb an ⟨expression-designator⟩ designating the ⟨expression⟩ simply contained in the ⟨lower-bound⟩ component of bp. Attach to aup an ⟨expression-designator⟩ designating the ⟨expression⟩ simply contained in the ⟨upper-bound⟩ component of bp.

Step 3.1.2.2. If the ⟨lower-bound⟩ component of bp contains a ⟨refer-option⟩,ro then perform CREATE-REFER-OPTION(ro) to obtain a <refer-option>,aro and attach aro to alb.

Step 3.1.2.3. If the ⟨upper-bound⟩ component of bp contains a ⟨refer-option⟩,ro perform CREATE-REFER-OPTION(ro) to obtain a <refer-option>,aro and attach aro to aup.

Step 3.1.2.4. Append abp to abpl.

Step 4. Return abpl.

4.6.3.4 Create-data-description

The <data-description> component of a <declaration> specifies the aggregate properties of a variable or the descriptor of a variable, and also the data properties associated with elements of the variable. This distinction between structures, arrays, and scalars is made at a high level, and the individual element data properties are attached to each scalar.

Operation: CREATE-DATA-DESCRIPTION(d)

Where: d is a ⟨declaration⟩ or a ⟨description⟩.

Result: a <data-description>.

Step 1. Let dd be a <data-description>.

Step 2. If d declaration-contains a ⟨dimension-attribute⟩,da then perform Steps 2.1 and 2.2.

Step 2.1. Perform CREATE-BOUND-PAIR-LIST(da) to obtain a <bound-pair-list>,abpl.

Step 2.2. Attach <dimensioned-data-description>: <element-data-description> abpl; to dd.

Step 3. If d declaration-contains STRUCTURE or UNION then perform Steps 3.1. through 3.4.

Step 3.1. Let dl be the node that immediately contains d. Let e[j] be the *j*th immediate component of dl that is not a ⟨,⟩. Let n be the number of such components and let k be such that e[k] is identical to d. Let i be k+1. Let ld be the numeric value of the ⟨level⟩ of d.

Step 3.2. Let idl be an <identifier-list> and let mdl be a <member-description-list>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 3.3. While i<n perform Steps 3.3.1 through 3.3.4.

Step 3.3.1. If e[i] does not declaration-contain MEMBER then go to Step 3.4.

Step 3.3.2. Let le be the numeric value of the ⟨level⟩ of e[i]. If le<ld then go to Step 3.4.

Step 3.3.3. If le=ld+1 then perform Steps 3.3.3.1 through 3.3.3.3.

Step 3.3.3.1. Perform CREATE-DATA-DESCRIPTION(e[i]) to obtain a <data-description>,add.

Step 3.3.3.2. Append a <member-description>: add; to mdl.

Step 3.3.3.3. If d is a ⟨declaration⟩ then let cid be the ⟨identifier⟩ of e[i], perform CREATE-IDENTIFIER(cid) to obtain an <identifier>,id, and append id to idl.

Step 3.3.4. Let i be i+1.

Step 3.4.

Case 3.4.1. d declaration-contains STRUCTURE.

If idl has no components, then let sdd be a <structure-data-description>: mdl; otherwise let sdd be a <structure-data-description>: idl mdl. Attach sdd to dd.

Case 3.4.2. d declaration-contains UNION.

If idl has no components then, let udd be a <union-data-description>: mdl; otherwise let udd be a <union-data-description>: idl mdl. Attach udd to dd.

Step 4. If d does not declaration-contain STRUCTURE or UNION then perform Steps 4.1 through 4.4.

Step 4.1. One of the following cases MUST apply:

Case 4.1.1. d declaration-contains ALIGNED.

Let al be <alignment>: <aligned>.

Case 4.1.2. d declaration-contains UNALIGNED.

Let al be <alignment>: <unaligned>.

Step 4.2. Perform CREATE-DATA-TYPE(d) to obtain a <data-type>,dt. Let idd be an <item-data-description>: al dt.

Step 4.3. If d declaration-contains ⟨initial⟩,int then let intd be an ⟨initial-designator⟩ that designates int and attach intd to idd.

Step 4.4. Attach idd to dd.

Step 5. Return dd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.3.5 Create-data-type

Operation: CREATE-DATA-TYPE(d)

Where: d is a ⟨declaration⟩ or a ⟨description⟩.

Result: a <data-type>.

One and only one of the following cases MUST apply:

Case 1. d declaration-contains AREA(⟨area-size⟩,asz).

Step 1.1. Let ar be an <area>. If asz immediately contains an ⟨*⟩ then let ed be an <asterisk>; otherwise let ed be an ⟨expression-designator⟩ that designates the ⟨expression⟩ simple component of asz. Attach ed to ar.

Step 1.2. If asz contains a ⟨refer-option⟩,ro, perform CREATE-REFER-OPTION(ro) to obtain a <refer-option>,aro and attach aro to ar.

Step 1.3. Return <data-type>: <non-computational-type>: ar.

Case 2. d declaration-contains ENTRY.

Perform CREATE-ENTRY(d) to obtain an <entry>,e and return a <data-type>: <non-computational-type>: e.

Case 3. d declaration-contains FILE.

Return a <data-type>: <non-computational-type>: <file>.

Case 4. d declaration-contains FORMAT or LABEL.

Attach <format> or <label> respectively to <data-type>,dt: <non-computational-type>. Return dt.

Case 5. d declaration-contains POINTER or OFFSET[(⟨reference⟩,r)].

Let dt be a <data-type>: <non-computational-type>: <locator>,loc. Attach <pointer> or <offset>,ofs, respectively to loc. If r exists then attach a ⟨reference-designator⟩ that designates r to ofs.

Case 6. d declaration-contains PICTURE ⟨picture⟩,pa.

Perform CREATE-PICTURE(pa) to obtain a <pictured>,p and return a <data-type>: <computational-type>: p.

Case 7. d declaration-contains a ⟨data-attribute⟩,sa which declaration-contains CHARACTER or BIT.

Step 7.1. If sa contains CHARACTER then let st be a <string-type>: <character>. Otherwise, let st be a <string-type>: <bit>.

Step 7.2. Let ml be the ⟨maximum-length⟩ component of sa. Let aml be a <maximum-length>. If ml immediately contains an ⟨*⟩ then let ed be an <asterisk>; otherwise let ed be an ⟨expression-designator⟩ that designates the ⟨expression⟩ simple component of ml. Attach ed to aml.

Step 7.3. fml contains a ⟨refer-option⟩,ro then perform CREATE-REFER-OPTION(re) to obtain a <refer-option>,aro and attach aro to aml.

Step 7.4. If d declaration-contains VARYING let v be <varying>; otherwise let v be <nonvarying>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 7.5. Return a <data-type>: <computational-type>: <string>: st aml v.

Case 8. (Otherwise).

Step 8.1. Let dt be a

<data-type>:
<computational-type>:
<arithmetic>:
<base>,b
<scale>,s
<precision>,p.

Step 8.2. If d declaration-contains BINARY then attach <binary> to b; otherwise, attach <decimal>.

Step 8.3. If d declaration-contains FIXED then attach <fixed> to s; otherwise, attach <float>.

Step 8.4. Let cp be the ⟨precision⟩ declaration-component of d. Let pd be a ⟨precision-designator⟩ designating cp. Attach pd to p.

Step 8.5. Return dt.

4.6.3.6 Create-entry

Operation: CREATE-ENTRY(d)

Where: d is a ⟨declaration⟩ or a ⟨description⟩ that declaration-contains an ENTRY(⟨description-commalist⟩,dlo).

Result: an <entry>.

Step 1. Let ent be an <entry> with no subnodes.

Step 2. If dlo exists, then for each ⟨description⟩,pd in dlo that does not declaration-contain MEMBER, perform Steps 2.1 and 2.2.

Step 2.1. Perform CREATE-DATA-DESCRIPTION(pd) to obtain a <data-description>,dd.

Step 2.2. Append a <parameter-descriptor>: dd; to the <parameter-descriptor-list> in ent.

Step 3. If d declaration-contains RETURNS (⟨description-commalist⟩,dc), perform Steps 3.1 through 3.3.

Step 3.1. Let rd be the only ⟨description⟩ immediate component of dc.

Step 3.2. Perform CREATE-DATA-DESCRIPTION(rd) to obtain a <data-description>,dd.

Step 3.3. Attach a <returns-descriptor>: dd; to ent.

Step 4. If d contains OPTIONS, attach <options> with some IMPLEMENTATION-DEFINED subnodes, to ent.

Step 5. Return ent.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.3.7 Create-refer-option

Operation: CREATE-REFER-OPTION(cro)

Where: cro is a ⟨refer-option⟩.

Result: a <refer-option>.

Step 1. Let ur be the ⟨unsubscripted-reference⟩ of cro. Perform FIND-APPLICABLE-DECLARATION(ur) to obtain d, which MUST NOT be ⟨absent⟩.

Step 2. Perform FIND-FULLY-QUALIFIED-NAME(d) to obtain an ⟨identifier-list⟩,idl.

Step 3. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(idl) to obtain an <identifier-list>,aidl.

Step 4. Return a <refer-option>: aidl.

4.6.3.8 Create-identifier

Operation: CREATE-IDENTIFIER(id)

Where: id is an ⟨identifier⟩.

Result: an <identifier>.

Step 1. Return an <identifier> whose concrete-representation is the same as that of id.

4.6.3.9 Create-initial

Operation: CREATE-INITIAL(init)

Where: init is an ⟨initial⟩.

Result: an <initial>.

Step 1. Let ainit be an <initial>.

Case 1.1. init has an ⟨initial-element-commalist⟩,iecl.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(iecl) to obtain an <initial-element-list>,aiel. Attach aiel to ainit.

Case 1.2. init has an ⟨array-init⟩,art.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(art) to obtain an <array-init>,aar. Attach aar to ainit.

Step 2. Return ainit.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.3.10 Create-array-init

Operation: CREATE-ARRAY-INIT(arni)

Where: arni is an ⟨array-init⟩

Result: an <array-init>

Step 1. Let aarni be an <array-init>.

Case 1.1. arni immediately contains an ⟨initial-constant-two⟩,ict.

Let e be the ⟨expression⟩ whose concrete-representation is the same as the concrete-representation of ict. Perform CREATE-EXPRESSION(e) to obtain an <expression>,ae. Attach a <parenthesized-expression>: ae; to aarni.

Case 1.2. arni immediately contains an ⟨initial-constant-one⟩,ico.

Let e be the ⟨expression⟩ whose concrete-representation is the same as the concrete-representation of ico.

Perform CREATE-EXPRESSION(e) to obtain an <expression>,ae. Attach a <parenthesized-expression>: ae; to aarni.

Case 1.3. arni immediately contains a ⟨parenthesized-expression⟩,cpe.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(cpe) to obtain a <parenthesized-expression>,pe and attach pe to arni.

Step 2. Return aarni.

4.6.3.11 Create-initial-element

Operation: CREATE-INITIAL-ELEMENT(ine)

Where: ine is an ⟨initial-element⟩.

Result: an <initial-element>.

Step 1. Let aine be an <initial-element>.

Case 1.1. ine immediately contains * as its only component.

Attach an <asterisk> to aine.

Case 1.2. ine immediately contains a ⟨parenthesized-expression⟩,cpe.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(cpe) to obtain a <parenthesized-expression>,pe and attach pe to aine.

Case 1.3. ine immediately contains an ⟨initial-constant-one⟩,ico.

Let e be the ⟨expression⟩ whose concrete-representation is the same as the concrete-representation of ico.

Perform CREATE-EXPRESSION(e) to obtain an <expression>,ae, and attach a <parenthesized-expression>: ae; to aine.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1.4. ine immediately contains an ⟨iteration-factor⟩,itf.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(itf) to obtain aif and attach aif to aine.

Case 1.4.1. ine immediately contains an *.

Attach an <initial-element-list>: <initial-element>: <asterisk>;; to aine.

Case 1.4.2. ine immediately contains an ⟨initial-constant-two⟩,ict.

Let e be the ⟨expression⟩ whose concrete-representation is the same as the concrete-representation of ict. Perform CREATE-EXPRESSION(e) to obtain an <expression>,ae. Attach an <initial-element-list>: <initial-element>: <parenthesized-expression>: ae;;; to aine.

Case 1.4.3. ine immediately contains an ⟨initial-element-commalist⟩,iec.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(iec) to obtain aiec, and attach aiec to aine.

Step 2. Return aine.

4.6.3.12 Create-integer

Operation: CREATE-INTEGER(int)

Where: int is an ⟨integer⟩.

Result: an <integer>.

Step 1. Return an <integer> whose concrete-representation contains the ⟨digit⟩s of int in the same order as they appear in int.

4.6.3.13 Convert-named-literal

Operation: CONVERT-NAMED-LITERAL(d,dt,bv)

Where: d is a ⟨declaration⟩,
dt is a <data-type>,
bv is a ⟨basic-value⟩.

Result: a <named-constant>.

Step 1.

Case 1.1. d declaration-contains a ⟨data-attribute⟩,da which immediately contains BIT or CHARACTER.

Step 1.1.1. Let ct be a

<string>:
<string-type>,st
<maximum-length>,ml
<nonvarying>.

Attach <bit> or <character>, respectively, to st.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.1.2. If the ⟨maximum-length⟩ of da immediately contains ⟨*⟩, then attach <asterisk> to ml. Otherwise, perform Steps 1.1.2.1 through 1.1.2.5.

Step 1.1.2.1. Let ee be the ⟨extent-expression⟩ in the ⟨maximum-length⟩ of da. ee MUST NOT contain a ⟨refer-option⟩. Let mice be the ⟨expression⟩ in ee.

Step 1.1.2.2. Perform CREATE-EXPRESSION(mice) to obtain an <expression>,mlae.

Step 1.1.2.3. Perform EVALUATE-RESTRICTED-EXPRESSION(mlae) to obtain mlc, which MUST be a <constant>.

Step 1.1.2.4. Perform EVALUATE-EXPRESSION-TO-INTEGER(mlae) to obtain an ⟨integer-value⟩,iv.

Step 1.1.2.5. Attach an <extent-expression>:iv; to ml.

Case 1.2. d declaration-contains a ⟨data-attribute⟩,da which immediately contains BINARY, DECIMAL, FIXED, FLOAT, or PRECISION.

Step 1.2.1. Let ct be an

<arithmetic>:
  <base>,b
  <scale>,s.

Step 1.2.2. If da declaration-contains BINARY or DECIMAL, then attach <binary> or <decimal>, respectively, to b. Otherwise, replace b by a copy of the derived <base> of dt.

Step 1.2.3. If da declaration-contains FIXED or FLOAT, then attach <fixed> or <float>, respectively, to s. Otherwise, replace s by a copy of the derived <scale> of dt.

Step 1.2.4. If s contains <fixed> and the derived <scale> of dt contains <float>, then da MUST declaration-contain a ⟨precision⟩.

Step 1.2.5. If da declaration-contains a ⟨precision⟩,cp, then perform CREATE-PRECISION(cp,dt) to obtain a <precision>,ap. Otherwise, let ap be the converted <precision> of dt for the <base> and <scale> of ct. Attach ap to ct.

Step 1.2.6. Perform CHECK-ARITHMETIC-TYPE(ct).

Step 2. Let tt be a <data-type>: <computational-type>: ct. Perform CONVERT(tt,dt,bv) to obtain cv.

Step 3. Let nc be a

<named-constant>:
  ct
  ⟨basic-value⟩:
    cv.

Return nc.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.3.14 Check-arithmetic-type

Operation: CHECK-ARITHMETIC-TYPE(dt)

Where: dt is a <data-type>.

Step 1. Let s be the <scale> of dt, and let p be the <precision> of dt.

Step 2. If s contains <float>, then p MUST NOT contain a <scale-factor>.

Step 3. If s contains <fixed> and p does not contain a <scale-factor>, then attach a <scale-factor>: 0; to p.

Step 4. The <number-of-digits> in p MUST NOT be greater than the maximum <number-of-digits> allowed for the <base> and <scale> of dt.

Step 5. If p has a <scale-factor>,sf and dt contains <binary>, then sf MUST have 0.

4.6.3.15 Create-precision

Operation: CREATE-PRECISION(cp,dt)

Where: cp is a ⟨precision⟩,
dt is a <data-type>.

Result: a <precision>.

Step 1. Let ap be a <precision>.

Step 2. Let nd be the ⟨number-of-digits⟩ in cp. Perform CREATE-NUMBER-OF-DIGITS(nd) to obtain an <integer>,ind, and attach a <number-of-digits>:ind; to ap.

Step 3. If cp contains a ⟨scale-factor⟩,sf, then perform CREATE-SCALE-FACTOR(sf) to obtain an <integer>,isf, and attach a <scale-factor>:isf; to ap.

Step 4. If dt contains <fixed> and ap does not contain a <scale-factor>, then attach a <scale-factor>: <integer>: 0;; to ap.

Step 5. Return ap.

4.6.3.16 Create-number-of-digits

Operation: CREATE-NUMBER-OF-DIGITS(nd)

Where: nd is a ⟨number-of-digits⟩.

Result: an <integer>.

Step 1.

Case 1.1. nd contains an ⟨integer⟩,cint.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(cint) to obtain an <integer>,int.

Case 1.2. nd contains an ⟨identifier⟩,id.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.2.1. Perform CREATE-REFERENCE(id,<value-reference>) to obtain a <value-reference>,vr. Let dd be the <data-description> immediately contained in vr. Let ae be an <expression>: vr dd.

Step 1.2.2. Perform EVALUATE-RESTRICTED-EXPRESSION(ae) to obtain c, which MUST be a <constant> having <computational-type>.

Step 1.2.3. Perform EVALUATE-EXPRESSION-TO-INTEGER(ae) to obtain an ⩽integer-value⩾,iv. iv MUST be greater than zero.

Step 1.2.4. Let int be an <integer> representing the same integer as iv.

Step 2. Return int.

4.6.3.17 Create-scale-factor

Operation: CREATE-SCALE-FACTOR(sf)

Where: sf is a ⩽scale-factor⩾.

Result: an <integer>.

Case 1. sf contains an ⩽integer⩾,csint.

Perform CREATE-ABSTRACT-EQUIVALENT-TREE(csint) to obtain an <integer>,sint. Return sint.

Case 2. sf contains an ⩽identifier⩾,id.

Step 2.1. Perform CREATE-REFERENCE(id,<value-reference>) to obtain a <value-reference>,vr. Let dd be the <data-description> immediately contained in vr. Let ae be an <expression>: vr dd.

Step 2.2. Perform EVALUATE-RESTRICTED-EXPRESSION(ae) to obtain c, which MUST be a <constant> having <computational-type>.

Step 2.3. Perform EVALUATE-EXPRESSION-TO-INTEGER(ae) to obtain an ⩽integer-value⩾,iv.

Step 2.4. iv MUST NOT be negative.

Step 2.5. Let int be an <integer> representing the same integer as iv.

Step 2.6. Return int.

4.6.4 Create-expression

Operation: CREATE-EXPRESSION(e)

Where: e is an ⩽expression⩾, ⩽expression-nine⩾, ⩽expression-eight⩾, ⩽expression-seven⩾, ⩽expression-six⩾, ⩽expression-five⩾, ⩽expression-four⩾, ⩽expression-three⩾, ⩽expression-two⩾, ⩽expression-one⩾, ⩽primitive-expression⩾, ⩽prefix-expression⩾, or ⩽parenthesized-expression⩾.

Result: an <expression>.

Case 1. e is an ⩽expression⩾, ⩽expression-seven⩾, ⩽expression-six⩾, ⩽expression-five⩾, ⩽expression-four⩾, ⩽expression-three⩾, or an ⩽expression-two⩾ and e has only one component, ec.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Perform CREATE-EXPRESSION(ec) to obtain an <expression>,aec. Return aec.

Case 2. e is an ⟨expression⟩, ⟨expression-seven⟩, ⟨expression-six⟩, ⟨expression-five⟩, ⟨expression-four⟩, ⟨expression-three⟩, or an ⟨expression-one⟩ and e has three components, e1, op, and e2.

Step 2.1. Perform CREATE-EXPRESSION(e1) to obtain an <expression>,ae1. Perform CREATE-EXPRESSION(e2,) to obtain an <expression>,ae2.

Step 2.2. If op is or has |, ¬ , &, |:, &:, >, >=, =, <=, < , ¬>, ¬=, ¬< , | |, +, −, *, /, ** then let aop be <or>, <xor>, <and>, <or-logical>, <and-logical>, <gt>, <ge>, <eq>, <le>, <lt>, <le>, <ne>, <ge>, <cat>, <add>, <subtract>, <multiply>, <divide>, or <power> respectively.

Step 2.3. Let dd1 and dd2 be the <data-description>s immediately contained in ae1 and ae2, respectively.

The associated ⟨aggregate-type⟩s of dd1 and dd2 MUST be compatible. Individual <data-type>s in dd1 and dd2 and corresponding <data-type>s in dd1 and dd2 MUST satisfy any constraints specified in the "Constraint" paragraphs of the section of Chapter 9 for the <infix-operator>,aop.

Let dd be a <data-description> whose associated ⟨aggregate-type⟩ is the common ⟨aggregate-type⟩ of dd1 and dd2 and whose <data-type>s are defined as "scalar-result-types" in the "Attributes" paragraphs of the section of Chapter 9 for the <infix-operator>,aop in terms of the corresponding <data-type>s in dd1 and dd2.

Step 2.4. Return an

```
<expression>:
   <infix-expression>:
      ae1
      <infix-operator>:
         aop;
      ae2
      dd;
   dd.
```

Case 3. e is a ⟨primitive-expression⟩.

Let ec be the immediate component of e.

Case 3.1. ec is a ⟨reference⟩.

Perform CREATE-REFERENCE(ec,<value-reference>) to obtain a <value-reference>,vr. Let dd be the <data-description> immediately contained in vr. Return an <expression>: vr dd.

Case 3.2. ec is a ⟨constant⟩.

Perform CREATE-CONSTANT(ec) to obtain a <constant>,c. Let dt be the <data-type> immediately contained in c. Return an

```
<expression>:
   c
   <data-description>:
      <item-data-description>: dt.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3.3. ec is an ⊰enquiry-reference⊱.

Step 3.3.1. Let ecr be the immediate component of ec. Let id be the first immediate component of ecr.

Note: This component (id) will be a ⊰NAME-enquiry-name⊱.

Step 3.3.2. If there are four immediate subnodes of ecr, let al be a copy of the third subnode. Otherwise, let al be ≤absent≥.

Note: As discussed in Section 2.4.22, the ⊰enquiry-reference⊱ immediate subnode (there is only one) will be a specific enquiry name. It, in turn, will have one of the following two forms at this point:

⊰NAME-enquiry-name⊱
⊰enquiry-lbracket⊱
⊰enquiry-rbracket⊱

or

⊰NAME-enquiry-name⊱
⊰enquiry-lbracket⊱
a specific enquiry argument node, with one or more subnodes.
⊰enquiry-rbracket⊱

Step 3.3.3. Perform CREATE-ENQUIRY-REFERENCE(id,al) to obtain an <expression>,el.

Step 3.3.4. Return el.

Case 4. e is a ⊰prefix-expression⊱: op e1.

Step 4.1. Perform CREATE-EXPRESSION(e1) to obtain ae1.

Step 4.2. If op has ¬, +, − then let aop be <not>, <plus>, or <minus> respectively.

Step 4.3. Let dd1 be the <data-description> immediately contained in ae1.

The <data-type>s in dd1 MUST satisfy any constraints in the "Constraint" paragraphs of the section of Chapter 9 for the <prefix-operator>,aop.

Let dd be a <data-description> whose associated ≤aggregate-type≥ is the same as that of dd1 and whose <data-type>s are defined as "scalar-result-types" in the "Attributes" paragraphs of the section of Chapter 9 for the <prefix-operator>,aop in terms of the corresponding <data-type> in dd1.

Step 4.4. Return an

<expression>:
  <prefix-expression>:
    <prefix-operator>: aop;
    ae1
    dd;
  dd.

Case 5. e is a ⊰parenthesized-expression⊱: (⊰expression⊱,e1).

Step 5.1. Perform CREATE-EXPRESSION(e1) to obtain an <expression>,ae. Let dd be the <data-description> immediate component of ae.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 5.2. dd MUST immediately contain an <item-data-description>. Return an

```
<expression>:
  <parenthesized-expression>:
    ae
    dd;
  dd.
```

4.6.5 Create-reference

Operation: CREATE-REFERENCE(cr,targ)

Where: cr is a ⟨reference⟩, an ⟨unsubscripted-reference⟩, or an ⟨identifier⟩,
targ is a <variable-reference>, <value-reference>, <subroutine-reference>, or <target-reference>.

Result: a tree of the same type as targ.

Step 1. If cr immediately contains an ⟨arguments-list⟩,ca, let al be a copy of ca. Otherwise let al be an ⟨arguments-list⟩. Perform FIND-APPLICABLE-DECLARATION(cr) to obtain a ⟨declaration⟩,cd.

Case 1.1. cd declaration-contains MEMBER.

Let rcd be the rightmost preceding ⟨declaration⟩ that does not declaration-contain MEMBER. Let dcl be the <declaration> whose ⟨declaration-designator⟩ designates rcd.

Case 1.2. (Otherwise).

Let dcl be the <declaration> whose ⟨declaration-designator⟩ designates cd. It MUST NOT contain <condition>.

Step 2. If cr is an ⟨identifier⟩, dcl MUST be a <declaration> having a <named-constant> containing <arithmetic> or <string>.

Step 3.

Case 3.1. dcl is a <declaration> that has <variable>.

Step 3.1.1. Let des be a <declaration-designator> designating dcl. Let dd be a copy of the <data-description> in dcl. Let ref be a <variable-reference>: des dd. dd MUST NOT contain a ⟨precision-designator⟩.

Step 3.1.2. Perform FIND-FULLY-QUALIFIED-NAME(cd) to obtain an ⟨identifier-list⟩,il. Perform CREATE-ABSTRACT-EQUIVALENT-TREE(il) to obtain an <identifier-list>,idl. Delete the first <identifier> in idl. If idl still contains any <identifier>s, attach idl to ref, and for each <identifier>,id, in idl, taken in order, perform Steps 3.1.2.1 through 3.1.2.3.

Step 3.1.2.1. Let dd be the <data-description> immediately contained in ref. It will have a <structure-data-description>,sdd or <union-data-description>,sdd. If dd has a <dimensioned-data-description>, let bpl be the <bound-pair-list> in dd; otherwise let bpl be a <bound-pair-list>. Let il be the <identifier-list> in sdd, and let mdl be the <member-description-list> in sdd. Let i be the integer such that the *i*th <identifier> in il equals id, and then let mdd be the <data-description> immediately contained in the *i*th <member-description> of mdl.

Step 3.1.2.2. If mdd has a <dimensioned-data-description>, append copies of the <bound-pair>s in mdd to bpl, and let tdd be the <union-data-description>,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

<item-data-description> or <structure-data-description> immediately contained in the <element-data-description> of mdd. Otherwise, let tdd be the <union-data-description>, <item-data-description> or <structure-data-description> immediately contained in mdd.

Step 3.1.2.3. If bpl contains <bound-pair>s, replace dd by a

```
<data-description>:
   <dimensioned-data-description>:
      <element-data-description>:
         tdd;
      bpl.
```

Otherwise replace dd by a <data-description>: tdd.

Step 3.1.3. If cr is a ⟨reference⟩, then let br be the ⟨basic-reference⟩ immediately contained in cr, and perform COLLECT-SUBSCRIPTS(br) to obtain a ⟨subscript-commalist⟩,sl. Otherwise let sl be a ⟨subscript-commalist⟩.

Case 3.1.3.1. The <data-description> immediately contained in ref does not have a <dimensioned-data-description>.

In this case sl MUST NOT contain any ⟨subscript⟩s.

Case 3.1.3.2. The <data-description>,dd, immediate component of ref has a <dimensioned-data-description>.

Perform APPLY-SUBSCRIPTS(cr,ref,sl,al).

Case 3.2. dcl is a <declaration> that has <named-constant>.

Perform CREATE-NAMED-CONSTANT-REFERENCE(dcl) to obtain a <named-constant-reference>,ref. Let dd be the <data-description> immediately contained in ref. dd MUST NOT contain a ⟨precision-designator⟩. If dd has a <dimensioned-data-description>, let sl be a ⟨subscript-commalist⟩, and perform APPLY-SUBSCRIPTS(cr,ref,sl,al).

Case 3.3. dcl is a <declaration> that has <builtin>.

Step 3.3.1.

Case 3.3.1.1. al has an ⟨arguments⟩.

In this case, al MUST have only one ⟨arguments⟩. Let ar be the ⟨arguments⟩ in ca. Perform CREATE-ARGUMENT-LIST(ar) to obtain args. Delete the ⟨arguments⟩ from al.

Case 3.3.1.2. al does not have an ⟨arguments⟩.

Let args be ≤absent≥.

Step 3.3.2.

Case 3.3.2.1. targ is a <value-reference>.

Perform CREATE-BUILTIN-FUNCTION-REFERENCE(dcl,args) to obtain a <builtin-function-reference>,ref.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3.3.2.2. targ is a <target-reference>.

Perform CREATE-PSEUDO-VARIABLE-REFERENCE(dcl,args) to obtain a <pseudo-variable-reference>,ref.

Case 3.3.2.3. targ is a <subroutine-reference>.

This case MUST NOT occur.

Note: This action and Case 2.2.1.1.1 of CONSTRUCT-CONTEXTUAL-DECLARATIONS are meant to illustrate the manner in which an implementation should support built-in subroutines (procedures which do not return values) as extensions to this standard. No built-in subroutines are defined by this standard.

Case 3.3.2.4. targ is a <variable-reference>.

This case MUST NOT occur.

Step 4.

Case 4.1. cr immediately contains a ⟨locator-qualifier⟩,lq.

Let r be the ⟨reference⟩ immediately contained in lq. Perform CREATE-REFERENCE(r,<value-reference>) to obtain a <value-reference>,vr. All the following conditions MUST hold:

1. ref MUST be a <variable-reference>;
2. The <declaration> designated by the <declaration-designator> immediately contained in ref MUST have <based>;
3. The <data-description> immediately contained in vr MUST immediately contain an <item-data-description> whose <data-type> MUST have <locator>, and if the <data-type> has an <offset>,os, then os MUST have a <variable-reference> or a ⟨reference-designator⟩.

Attach a <locator-qualifier>: vr; to ref.

Case 4.2. cr does not immediately contain a ⟨locator-qualifier⟩ and dcl is a <declaration> that has <variable>.

If dcl contains a <based>,b, then b MUST immediately contain a <value-reference> or a ⟨reference-designator⟩.

Case 4.3. (Otherwise).

No action.

Step 5. If al contains an ⟨arguments⟩, perform Steps 5.1 to 5.4.

Step 5.1. ref MUST NOT be a <subroutine-reference>. Perform CREATE-VALUE-REFERENCE(ref) to obtain evr. The <data-description> immediate component of evr MUST immediately contain an <item-data-description> whose <data-type> MUST have <entry>.

Step 5.2. Let ar be the first ⟨arguments⟩ in al. Perform CREATE-ARGUMENT-LIST(ar) to obtain args. Delete ar from al.

Step 5.3. Perform CREATE-ENTRY-REFERENCE(evr,args) to obtain ref.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 5.4. Go to Step 5.

Step 6.

Case 6.1. targ is a <variable-reference>.

In this case ref MUST be a <variable-reference>. Return ref.

Case 6.2. targ is a <value-reference>.

ref MUST NOT be a <subroutine-reference>. Perform CREATE-VALUE-REFERENCE(ref) to obtain a <value-reference>,vr. Return vr.

Case 6.3. targ is a <subroutine-reference>.

If ref is a <subroutine-reference>, return ref. Otherwise perform CREATE-VALUE-REFERENCE(ref) to obtain a <value-reference>,vr whose immediately contained <data-description> MUST immediately contain an <item-data-description> whose <data-type> MUST have <entry>. Perform CREATE-ENTRY-REFERENCE(vr) to obtain sr, which MUST be a <subroutine-reference>. Return sr.

Case 6.4. targ is a <target-reference>.

In this case ref MUST be a <variable-reference> or <pseudo-variable-reference>. Let dd be a copy of the <data-description> immediately contained in ref. Return a <target-reference>: ref dd.

4.6.5.1 Collect-subscripts

Operation: COLLECT-SUBSCRIPTS(br)

Where: br is a ⟨basic-reference⟩.

Result: a ⟨subscript-commalist⟩ which may have no components.

Step 1. Let scl be a ⟨subscript-commalist⟩ with no components.

Step 2. For each ⟨structure-qualification⟩,sq in br, taken in order, such that sq contains an ⟨arguments⟩, perform Step 2.1.

Step 2.1. Let args be the ⟨arguments⟩ of sq. args MUST immediately contain a ⟨subscript-commalist⟩,scla. Append the ⟨subscript⟩s of scla to scl in order, appending ⟨,⟩ as required.

Step 3. Return scl.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.5.2 Apply-subscripts

Operation: APPLY-SUBSCRIPTS(cr,ref,sl,al)

Where: cr is a ⟨reference⟩,
ref is a <variable-reference> or a <named-constant-reference>,
sl is a ⟨subscript-commalist⟩,
al is an ⟨arguments-list⟩.

Step 1. Let dd be the <data-description> immediately contained in ref. Let m be the number of <bound-pair>s in dd, and let n be the number of ⟨subscript⟩s in sl (n may be 0). If n < m, and if al has a first immediate component, args, that has a ⟨subscript-commalist⟩, then perform Step 1.1.

Step 1.1. Append the ⟨subscript⟩s in args to sl, appending ⟨,⟩ as required. Delete args from al.

Step 2. Let n be the number of ⟨subscript⟩s in sl.

Case 2.1. n=0.

Case 2.1.1. ref is a <variable-reference>.

Attach a <subscript-list> containing m occurrences of <asterisk> to ref.

Case 2.1.2. ref is a <named-constant-reference>.

Attach a <subscript>:<asterisk>; to ref.

Case 2.2. n≠0.

Step 2.2.1.

Case 2.2.1.1. ref is a <variable-reference>.

In this case n MUST equal m. Attach a <subscript-list>,sl2 to ref. For i=1,...,n, perform Step 2.2.1.1.1.

Step 2.2.1.1.1. Let s be the *i*th ⟨subscript⟩ in sl. s immediately contains an ⟨expression⟩,e. Perform CREATE-EXPRESSION(e) to obtain an <expression>,e2 and append a <subscript>: e2; to sl2.

Case 2.2.1.2. ref is a <named-constant-reference>.

n must equal 1. m is 1. Let s be the ⟨subscript⟩ in sl. s immediately contains an ⟨expression⟩,e. Perform CREATE-EXPRESSION(e) to obtain an <expression>,e2. Attach a <subscript>: e2 to ref.

Step 2.2.2. Let dd2 be the immediate component of the <element-data-description> of dd. Replace dd by a <data-description>: dd2.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.5.3 Create-value-reference

Operation: CREATE-VALUE-REFERENCE(ref)

Where: ref is a <variable-reference>, <procedure-function-reference>, <builtin-function-reference>, or <named-constant-reference>.

Result: a <value-reference>.

Step 1. Let dd be a copy of the <data-description> immediately contained in ref.

Step 2. Perform TRIM-DD(dd).

Note: The trimming is performed so as to remove attributes that pertain to stored data only, and to make the treatment of extents consistent among different kinds of <value-reference>s. All bounds and extents are replaced by asterisks so that the <value-reference> will satisfy the comparisons made in determining whether its <data-description> is proper for assignment.

Return a <value-reference>: ref dd.

4.6.5.4 Trim-dd

Operation: TRIM-DD(dd)

Where: dd is a <data-description>.

Step 1. Delete from dd any occurrences of the following categories that are not contained in an <entry>:

<alignment>,
<initial>,
<identifier-list>,
<options>,
<varying>,
<nonvarying>.

Replace by an <asterisk> the immediate component of each <maximum-length>, <area-size> or <bound-pair> that is not a component of an <entry>.

4.6.5.5 Create-named-constant-reference

Operation: CREATE-NAMED-CONSTANT-REFERENCE(dcl)

Where: dcl is a <declaration>.

Result: a <named-constant-reference>.

Step 1. Let ct be a copy of the leftmost immediate component of the <named-constant> component of dcl. ct MUST NOT be a <package-constant>. ct MUST NOT be a ⟨declaration-designator⟩.

Note: A consequence of this requirement (and the actions of Steps 1 and 2 of CREATE-BLOCK) is that, if the definition of a named computational constant, c2, references a named computational constant, c1, the following MUST be true:

1. c1 MUST be defined in a block which contains the block of c2 and not redefined in the block of c2, or

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2. c1 MUST be defined in the same block as c2, and the definition of c2 MUST be after the definition of c1.

If ct is an <arithmetic> or a <string>, then let dt be an

```
<item-data-description>:
   <data-type>:
      <computational-type>:
         ct.
```

Otherwise, let dt be an

```
<item-data-description>:
   <data-type>:
      <non-computational-type>:
         ct.
```

Step 2.

Case 2.1. dcl contains a <bound-pair>,bp.

Let bpl be a <bound-pair-list>: <bound-pair>: <asterisk>. Let dd be a

```
<dimensioned-data-description>:
   <element-data-description>:
      dt;
   bpl.
```

Case 2.2. (Otherwise).

Let dd be dt.

Step 3. Let ddg be a <declaration-designator> designating dcl. Return a

```
<named-constant-reference>:
   ddg
   <data-description>:
      dd.
```

4.6.5.6 Create-argument-list

Operation: CREATE-ARGUMENT-LIST(al)

Where: al is an ⟨arguments⟩.

Result: an <argument-list> or ⟨absent⟩.

Step 1. If al does not contain a ⟨subscript-commalist⟩ then return ⟨absent⟩; otherwise let scl be the ⟨subscript-commalist⟩ immediately contained in al.

Step 2. Let n be the number of ⟨subscript⟩ immediate components of scl.

Step 3. Let xal be an <argument-list> and let ec[i], i=1,...,n be the ⟨expression⟩ simple components of scl taken in left-to-right order.

Step 4. For each element, ec[i], i=1,...,n, perform Steps 4.1 and 4.2.

Step 4.1. Perform CREATE-EXPRESSION(ec[i]) to obtain an <expression>, eac[i].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4.2. Let rdd be the <data-description> immediately contained in eac[i]. Append to xal an

<argument>:
eac[i] rdd.

Step 5. Return xal.

4.6.5.7 Create-builtin-function-reference

A *builtin-function-name* is a sequence of uppercase letters and digits such that the corresponding sequence of lowercase letters and digits followed by "-bif" is the category-name of a subnode of <builtin-function>.

Operation: CREATE-BUILTIN-FUNCTION-REFERENCE(ad,al)

Where: ad is a <declaration>,
al is an [<argument-list>].

Result: a <builtin-function-reference>.

Step 1. Let id be the <identifier> contained in ad. There MUST be a <builtin-function>,bf, whose name or abbreviation (as listed in Section 2.7) corresponds to the concrete-representation of id. Let bfr be a <builtin-function-reference>: bf.

Step 2. The number of <argument>s in al MUST be as shown in the "Arguments" section of the description of bf (see Chapter 9).

Step 3.

Case 3.1. al is not ⟪absent⟫.

Step 3.1.1. For each <argument>,ar in al which corresponds to an argument named "p", "q", or "n" in the "Arguments" section of the description of bf (see Chapter 9), and which is not of the form <argument>: <expression>: <constant>;, perform Steps 3.1.1.1 through 3.1.1.4.

Step 3.1.1.1. ar MUST be of the form

<argument>:
<expression>,e:
<value-reference>:
<named-constant-reference>.

Step 3.1.1.2. Perform EVALUATE-RESTRICTED-EXPRESSION(e) to obtain c, which MUST be a <constant> having <fixed> and a <scale-factor> with zero.

Step 3.1.1.3. Perform EVALUATE-EXPRESSION-TO-INTEGER(e) to obtain an ⟪integer-value⟫,iv.

Step 3.1.1.4. Let dt be a <data-type> which is integer-type (Section 9.2.2). Let rv be a ⟪real-value⟫ with the same component as iv. Let c be a

<constant>:
⟪basic-value⟫:
rv;
dt.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Replace ar by an

```
<argument>:
   <expression>:
      c
      dt;
   dt;
```

Step 3.1.2. All <data-description>s immediately contained in the <argument>s in al MUST satisfy the constraints given in the "Constraint" section of the same builtin-function description.

Step 3.1.3. Append al to bfr.

Case 3.2. (Otherwise).

No action.

Step 4. Construct a <data-description>,rdd, as specified in the "Attributes" section of the same <builtin-function> description. Append rdd to bfr.

Step 5. Return bfr.

4.6.5.8 Create-pseudo-variable-reference

Operation: CREATE-PSEUDO-VARIABLE-REFERENCE(ad,al)

Where: ad is a <declaration>,
al is an [<argument-list>].

Result: a <pseudo-variable-reference>.

Step 1. Let id be the <identifier> contained in ad. There MUST be a <pseudo-variable>,pv whose name corresponds to the concrete-representation of id. Let pvr be a <pseudo-variable-reference>: pv.

Step 2. The number of <argument>s in al MUST be as shown in the "Arguments" section of the description of pv (see Section 7.6.4).

Step 3.

Case 3.1. al is not <absent>.

Step 3.1.1. All <data-description>s immediately contained in the <argument>s in al MUST satisfy the constraints given in the "Constraint" section of the description of pv.

Step 3.1.2. Attach al to pvr.

Case 3.2. (Otherwise).

No action.

Step 4. Construct a <data-description>,rdd, as specified in the "Attributes" section of the same <pseudo-variable> description. Attach rdd to pvr.

Step 5. Return pvr.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.6.5.9 Create-entry-reference

Operation: CREATE-ENTRY-REFERENCE(vr,al)

Where: vr is an <value-reference> whose <data-type> has <entry>,
al is an [<argument-list>].

Result: a <procedure-function-reference> or a <subroutine-reference>.

Step 1. Let dd be the <data-description> immediately contained in vr.

Step 2.

Case 2.1. dd simply contains a <parameter-descriptor-list>,pdl.

al MUST NOT be ⩽absent⩾. The number of elements in pdl MUST be equal to the number of elements in al.

Case 2.2. (Otherwise).

al MUST be ⩽absent⩾. Go to Step 4.

Step 3. For each <argument>,arg, immediate component of al, perform Steps 3.1 through 3.4.

Step 3.1. Let pdd be the <data-description> immediate component of the <parameter-descriptor> corresponding to arg.

Step 3.2. arg MUST NOT contain a <named-constant-reference> which has a <dimensioned-data-description> which has a <data-type> with <label>.

Step 3.3.

Case 3.3.1. arg immediately contains <expression>: <value-reference>: <variable-reference>,var.

Perform TEST-MATCHING(var,pdd) to obtain tv. If tv is ⩽false⩾, then attach <dummy> to arg.

Case 3.3.2. (Otherwise).

Attach <dummy> to arg.

Step 3.4.

Case 3.4.1. <dummy> was attached to arg in Step 3.3.

Let rdd be the <data-description> immediately contained in arg. rdd MUST immediately contain an <item-data-description>. rdd MUST be proper for assignment to pdd (see Section 4.6.2.2).

Case 3.4.2. (Otherwise).

No action.

Step 4.

Case 4.1. dd simply contains a <returns-descriptor>,rd.

Let rde be a copy of the <data-description> immediately contained in rd. Return <procedure-function-reference>: vr [al] rde.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 4.2. (Otherwise).

Return <subroutine-reference>: vr [al].

4.6.5.10 Test-matching

Operation: TEST-MATCHING(var,pd)

Where: var is a <variable-reference>,
pd is a <data-description> immediate component of a <parameter-descriptor>.

Result: ≪true≫ or ≪false≫.

Step 1. Let dcl be the <declaration> designated by the <declaration-designator> in var.

Step 2. Let dd be a copy of the <data-description> immediately contained in var. Let pdd be a copy of pd.

Step 3. If any of the following subtrees exist as a component of dd or pdd then delete every occurrence.

<initial>
<variable-reference> as component of an <offset>
<parameter-descriptor-list>
<returns-descriptor>
<options>
<identifier-list> as a component of a <structure-data-description> or
<union-data-description>.

Step 4. If pdd and dd are not equal, disregarding comparison of the subnodes of any <maximum-length>, <bound-pair>, or <area-size>, then return ≪false≫.

Step 5. For each <extent-expression>,e1, in pdd, perform Step 5.1.

Step 5.1. If there does not exist a corresponding <extent-expression>,e2, in dd, return ≪false≫. If e2 contains a <refer-option>, return ≪false≫. Let ex1 and ex2 be the <expression>s in e1 and e2, respectively. Perform EVALUATE-RESTRICTED-EXPRESSION(ex1) and EVALUATE-RESTRICTED-EXPRESSION(ex2) to obtain v1 and v2. If v1 or v2 is not a <constant> having <computational-type>, return ≪false≫. Otherwise, perform EVALUATE-EXPRESSION-TO-INTEGER(ex1) and EVALUATE-EXPRESSION-TO-INTEGER(ex2) to obtain ≪integer-value≫s i1 and i2 respectively. If i1 and i2 are not equal, return ≪false≫.

Step 6. Return ≪true≫.

4.6.6 Create-picture

A ⟨picture⟩ may occur as a component of a ⟨declaration⟩ or a ⟨format-item⟩. In both cases it is translated to a <pictured>. Elements of a ⟨picture⟩ may be repeated by the specification of a ⟨repetition-factor⟩ which is expanded into a sequence of elements first. Then the ⟨picture⟩ is translated to a <pictured-numeric>.

This operation is also used by the EDIT-BIF to translate the picture specification supplied as its second argument.

The content of a ⟨picture⟩ is governed by the following syntax:

⟨picture-content⟩ ::= ⟨picture-item-list⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

⟨picture-item⟩ ::= [⟨repetition-factor⟩] ⟨picture-element⟩

⟨repetition-factor⟩ ::= (⟨integer⟩)

⟨picture-element⟩ ::= S | Z | $ | 9 | + | − | (|) | * | ⟨>⟩ | V | ⟨insertion-string⟩ |
⟨insertion-character⟩ | ⟨credit⟩ | ⟨debit⟩

⟨insertion-character⟩ ::= B | / | . | ,

⟨credit⟩ ::= CR

⟨debit⟩ ::= DB

⟨insertion-string⟩ ::= ⟨<⟩ ⟨insertion-symbol-list⟩ ⟨>⟩

⟨insertion-symbol⟩ ::= ⟨letter⟩ | ⟨digit⟩ | . | ₿ | '' | = | + | − | * | / | (|) | , | __ | $ | ; |
: | & | ⟨|⟩ | ¬ | ⟨<<⟩ | ⟨<>⟩ | % | ⟨extralingual-character⟩

Operation: CREATE-PICTURE(p)

Where: p is a ⟨picture⟩.

Result: a <pictured>.

Step 1. There MUST be a ⟨string-or-picture-symbol-list⟩,spsl in p. Let S be the set of all ⟨picture-content⟩ trees whose concrete-representation is the same as that of spsl. There MUST be one and only one tree, t, in S such that no other tree in S contains more ⟨repetition-factor⟩s than t.

Note: This resolves the syntactic ambiguity between uses of ⟨(⟩ and ⟨)⟩ in ⟨repetition-factor⟩s and ⟨picture-element⟩s in favor of ⟨repetition-factor⟩s.

Step 2. For each component of t which is a

⟨picture-item⟩,c:
⟨repetition-factor⟩,rf
⟨picture-element⟩,pe;

let iv be the decimal value of the ⟨integer⟩ in rf. iv MUST NOT be equal to zero. pe MUST NOT contain an ⟨insertion-string⟩. Replace c by iv occurrences of a ⟨picture-item⟩: pe.

Step 3. Perform CREATE-NUMERIC-PICTURE(t) to obtain a <pictured-numeric>,pn. Return <pictured>: pn.

4.6.6.1 Create-numeric-picture

The *picture-validation* syntax is as follows:

⟨numeric-picture-specification⟩ ::= ⟨non-drifting-field⟩ | ⟨drifting-field⟩

⟨non-drifting-field⟩ ::= [$] ⟨digits⟩ { ⟨credit⟩ | ⟨debit⟩ } |
{ ⟨lparen⟩ • [$] } ⟨digits⟩ ⟨rparen⟩ |
⟨lparen⟩ { [$] • ⟨digits⟩ } ⟨rparen⟩ |
[⟨sign⟩] ⟨digits⟩ [$] |
[$] ⟨digits⟩ [⟨sign⟩]

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

⟨digits⟩ ::= ⟨scaled-digits-field⟩ |
⟨Z-list⟩ [⟨scaled-digits-field⟩] |
[⟨Z-list⟩] V ⟨Z-list⟩ |
⟨*-list⟩ [⟨scaled-digits-field⟩] |
[⟨*-list⟩] V ⟨*-list⟩

⟨scaled-digits-field⟩ ::= ⟨9-list⟩ [V ⟨9-list⟩] | V ⟨9-list⟩

⟨sign⟩ ::= S | + | −

⟨drifting-field⟩ ::= { ⟨drifting-sign-field⟩ • [$] } |
{ ⟨drifting-dollar-field⟩ • [⟨sign⟩] } |
⟨drifting-dollar-field⟩ { ⟨credit⟩ | ⟨debit⟩} |
⟨lparen⟩ ⟨drifting-dollar-field⟩ ⟨rparen⟩ |
{ ⟨drifting-lparen-field⟩ • [$] } ⟨rparen⟩ |
{ ⟨drifting-insert-field⟩ • [$] } |
{ ⟨drifting-insert-field⟩ • [⟨sign⟩] } |
⟨drifting-insert-field⟩ {⟨credit⟩ | ⟨debit⟩} |
⟨lparen⟩ ⟨drifting-insert-field⟩ ⟨rparen⟩

⟨drifting-sign-field⟩ ::= ⟨signs⟩ [⟨scaled-digits-field⟩] |
S [⟨S-list⟩] V ⟨S-list⟩ |
+ [⟨plus-list⟩] V ⟨plus-list⟩ |
− [⟨minus-list⟩] V ⟨minus-list⟩

⟨lparen⟩ ::= (

⟨rparen⟩ ::=)

⟨drifting-lparen-field⟩ ::= ⟨lparen⟩ ⟨lparen-list⟩ [⟨scaled-digits-field⟩] |
⟨lparen⟩ [⟨lparen-list⟩] V ⟨lparen-list⟩

⟨plus⟩ ::= +

⟨minus⟩ ::= −

⟨signs⟩ ::= S ⟨S-list⟩ | ⟨plus⟩ ⟨plus-list⟩ | ⟨minus⟩ ⟨minus-list⟩

⟨drifting-dollar-field⟩ ::= $ ⟨$-list⟩ [⟨scaled-digits-field⟩] | $ [⟨$-list⟩] V ⟨$-list⟩

⟨drifting-insert-field⟩ ::= ⟨insertion-string⟩ ⟨rangle-list⟩ [⟨scaled-digits-field⟩] |
⟨insertion-string⟩ [⟨rangle-list⟩] V ⟨rangle-list⟩

⟨rangle⟩ ::= ⟨>⟩

A *digit-position* is any occurrence of a 9 or Z or *, or any occurrence of S, +, −, (, >, or $ in an ⟨S-list⟩, ⟨plus-list⟩, ⟨minus-list⟩, ⟨lparen-list⟩, ⟨rangle-list⟩, or ⟨$-list⟩.

Operation: CREATE-NUMERIC-PICTURE(p)

Where: p is of the form ⟨picture-content⟩: ⟨picture-item-list⟩,pil.

Result: a <pictured-numeric>.

Step 1. Let pilw be a copy of pil. Delete from pilw any ⟨picture-item⟩ which contains an ⟨insertion-character⟩. Delete from pilw any ⟨picture-item⟩ which contains an ⟨insertion-string⟩ except for the last (if any) ⟨picture-item⟩ containing an ⟨insertion-string⟩ which precedes the first (if any) ⟨rangle-list⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 2. It MUST be possible to construct a ⟨numeric-picture-specification⟩,nps according to the picture-validation syntax (above), such that the concrete-representation of nps is the same as that of pilw.

Step 3. Let pn be a partial tree

<pictured-numeric>:
<numeric-picture-specification>,anps
<arithmetic>:
<base>:
<decimal>;
<scale>:
<fixed>;
<precision>,prec:
<number-of-digits>: n;;;;

where n is the number of digit-positions in nps. n MUST NOT be greater than the maximum <number-of-digits> for <base>: <decimal>; and <scale>: <fixed>.

Step 4. Attach to anps a ⟨picture-element-list⟩ with the same concrete-representation as pil.

Step 5. Let m be the number of digit-positions after the V in nps if V appears, and 0 otherwise. Attach a <scale-factor>: m; to prec.

Step 6. Return pn.

4.6.7 Create-constant

Operation: CREATE-CONSTANT(c)

Where: c is a ⟨constant⟩.

Result: a <constant>.

Case 1. c contains a ⟨simple-character-string-constant⟩.

Step 1.1.

Case 1.1.1. c contains no ⟨string-or-picture-symbol-list⟩.

Let csv be a ⟨character-string-value⟩: ⟨null-character-string⟩.

Case 1.1.2. (Otherwise).

Let csv be a ⟨character-string-value⟩ whose ⟨symbol⟩s (in order) have the same concrete-representations as the ⟨string-or-picture-symbol⟩s in c, except that each ⟨string-or-picture-symbol⟩: ⟨''⟩; in c becomes a ⟨symbol⟩: ⟨'⟩; in csv.

Step 1.2. Return a <constant>: ⟨basic-value⟩: csv; dt; where dt is a <data-type> containing <character>, <nonvarying>, and <maximum-length>: <asterisk>.

Case 2. c contains a ⟨simple-bit-string-constant⟩.

Step 2.1.

Case 2.1.1. c contains no ⟨string-or-picture-symbol-list⟩.

Let bsv be a ⟨bit-string-value⟩: ⟨null-bit-string⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.1.2. (Otherwise).

Let m be 1,2,3,4 according to whether ⟨radix-factor⟩ in c has B1, B2, B3, or B4. Let s[i], i=1,...,k, be the ⟨string-or-picture-symbol⟩s in c.

Each s[i] MUST have an entry in Figure 12 on page 149 which is valid for the value of m. Let bsv be a ≪bit-string-value≫ containing m*k ≪bit-value≫s, such that ≪bit-value≫s (i*m+1-m) through (i*m) are obtained from Figure 12 on page 149 as a function of m and s[i], i=1,...,k.

Step 2.2. Return a <constant>: ≪basic-value≫: bsv; dt; where dt is a <data-type> containing <bit>, <nonvarying>, and <maximum-length>: <asterisk>.

Case 3. c contains an ⟨arithmetic-constant⟩,ac.

Step 3.1. Perform EVALUATE-REAL-CONSTANT(rc), where rc is the ⟨real-constant⟩ in ac, to obtain a ≪value-and-type≫: ≪real-value≫,v <data-type>,t.

Step 3.2. Let ds be a partial ⟨declaration⟩ containing:

CONSTANT
if c contains E, then FLOAT

(ds is partial in that it contains no ⟨identifier⟩).

Step 3.3. Perform APPLY-DEFAULTS(ds).

Step 3.4. Let dt be a partial <data-type>: <computational-type>: <arithmetic>;; and complete it as follows:

Step 3.4.1. For each of the following which is contained in ds, append the abstract-equivalent to dt: BINARY, DECIMAL, FIXED, FLOAT. If dt is still without <base> or <scale> (or both), copy the <base> or <scale> (or both) from t.

Step 3.4.2. Let cp be the converted <precision> of t for the <base> and <scale> of dt.

Step 3.4.3.

Case 3.4.3.1. ds contains a ⟨precision⟩,p.

Perform CREATE-PRECISION(p,dt) to obtain a <precision>,ap. Attach ap to dt.

Case 3.4.3.2. (Otherwise).

Attach cp to dt.

Step 3.4.4. Perform CHECK-ARITHMETIC-TYPE(dt).

Step 3.5. Perform CONVERT(dt,t,v) to obtain a ≪real-value≫,rv. Let bv be a ≪basic-value≫: rv.

Step 3.6. Return a <constant>: bv dt.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

si	m=1	m=2	m=3	m=4
0	0	00	000	0000
1	1	01	001	0001
2	-	10	010	0010
3	-	11	011	0011
4	-	-	100	0100
5	-	-	101	0101
6	-	-	110	0110
7	-	-	111	0111
8	-	-	-	1000
9	-	-	-	1001
A	-	-	-	1010
B	-	-	-	1011
C	-	-	-	1100
D	-	-	-	1101
E	-	-	-	1110
F	-	-	-	1111
Other	-	-	-	-

Figure 12. Table of ≪bit-values≫ as a Function of ⟨symbols⟩ and ⟨radix-factors⟩ for CREATE-CONSTANT.

Where si is the contents of *i*th ⟨symbol⟩ or ⟨string-or-picture-symbol⟩, the m=n columns indicate the contents of ≪bit-value≫s (i*m+1−m) through (i*m). The - indicates that the corresponding ⟨symbol⟩ or ⟨string-or-picture-symbol⟩ is invalid for this value of m, 0 indicates ≪zero-bit≫, 1 indicates ≪one-bit≫.

4.6.8 Create-enquiry-reference

Operation: CREATE-ENQUIRY-REFERENCE(rfn,rfa)

Where: rfn is an ⟨XX-enquiry-name⟩,
rfa is an ⟨XX-arguments⟩ or ≪absent≫, and XX is a declaresize, epsilon, maxval, minval, null, or radix.

Result: an <expression>.

Step 1.

Case 1.1. rfn is an ⟨declaresize-enquiry-name⟩.

Step 1.1.1. Let id be the ⟨identifier⟩ immediately contained in rfa. Perform FIND-APPLICABLE-DECLARATION(id) to obtain cdcl, which MUST be a ⟨declaration⟩. cdcl MUST NOT declaration-contain MEMBER.

Step 1.1.2. Let adcl be the <declaration> whose ⟨declaration-designator⟩ designates cdcl. adcl MUST contain <based>.

Step 1.1.3. Let dd be a <data-description>: <item-data-description>,idd. Attach a <data-type> which is integer-type to idd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.1.4. Let des be a <declaration-designator> designating adcl. Let e1 be an

```
<expression>:
   <value-reference>:
      <enquiry-reference>:
         <declaresize-enquiry>
         dd
         des;
      dd;
   dd.
```

Case 1.2. rfn is an ⟨epsilon-enquiry-name⟩.

Perform CREATE-ARITHMETIC-ENQUIRY-REFERENCE(<epsilon-enquiry>,rfa) to obtain an <expression>,e1.

Case 1.3. rfn is a ⟨maxval-enquiry-name⟩.

Perform CREATE-ARITHMETIC-ENQUIRY-REFERENCE(<maxval-enquiry>,rfa) to obtain an <expression>,e1.

Case 1.4. rfn is a ⟨minval-enquiry-name⟩.

Perform CREATE-ARITHMETIC-ENQUIRY-REFERENCE(<minval-enquiry>,rfa) to obtain an <expression>,e1.

Case 1.5. rfn is a ⟨null-enquiry-name⟩.

Step 1.5.1. Let dd be a

```
<data-description>:
   <item-data-description>:
      <data-type>:
         <non-computational-type>,nct.
```

Case 1.5.1.1. rfa contains POINTER.

Attach a <locator>: <pointer>; to nct. Let e1 be an

```
<expression>:
   <value-reference>:
      <builtin-function-reference>:
         <builtin-function>:
            <null-bif>;
         dd;
      dd;
   dd.
```

Note: This form is written this way to guarantee exactly the same value as the NULL() built-in function.

Case 1.5.1.2. rfa contains OFFSET, ENTRY, LABEL or FILE.

Step 1.5.1.2.1.

Case 1.5.1.2.1.1. rfa contains OFFSET.

Attach a <locator>: <offset>; to nct.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1.5.1.2.1.2. rfa contains ENTRY.

Attach an <entry> to nct.

Case 1.5.1.2.1.3. rfa contains LABEL.

Attach a <label> to nct.

Case 1.5.1.2.1.4. rfa contains FILE.

Attach a <file> to nct.

Step 1.5.1.2.2. Let e1 be an

```
<expression>:
   <value-reference>:
      <enquiry-reference>:
         <null-enquiry>
         dd;
      dd;
   dd.
```

Case 1.6. rfn is a ⟨radix-enquiry-name⟩.

Perform CREATE-ARITHMETIC-ENQUIRY-REFERENCE(<radix-enquiry>,rfa) to obtain an <expression>,e1.

Step 2. Return e1.

4.6.8.1 Create-arithmetic-enquiry-reference

Operation: CREATE-ARITHMETIC-ENQUIRY-REFERENCE(rtn,rfa)

Where: rtn is an ⟨XX-enquiry⟩
rfa is an ⟨XX-arguments⟩, and XX is one of maxval, minval, epsilon, or radix.

Result: an <expression>.

Step 1. Let cd be the ⟨description⟩ simply contained in rfa.

Step 2. Perform CREATE-DATA-DESCRIPTION(cd) to obtain a ⟨description⟩,ad. ad MUST immediately contain an <item-data-description>,idd.

Step 3. Let dt be the <data-type> in idd. dt MUST immediately contain an <arithmetic> which has <float>. dt MUST NOT contain a <pictured>.

Step 4. Let dd be a <data-description>: <item-data-description>: dt.

Step 5. Return an

```
<expression>:
   <value-reference>:
      <enquiry-reference>:
         rtn
         dt
         dd;
      dd;
   dd.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.7 Validation of the External Unit

An <external-unit> has been constructed so far by the Translator, corresponding to the specified ⟨external-unit⟩. This <external-unit> is now examined by some final tests before being returned as the result of the TRANSLATE operation. These tests validate each declaration and apply constraints.

Operation: VALIDATE-EXTERNAL-UNIT(aeu)

Where: aeu is an <external-unit>.

Step 1. Perform APPLY-CONSTRAINTS(aeu).

Step 2. For each <declaration>,ad component of aeu perform VALIDATE-DECLARATION(ad).

Step 3. For each <parameter-descriptor>,d and for each <returns-descriptor>,d contained in aeu perform VALIDATE-DESCRIPTOR(d).

Step 4. aeu MUST satisfy all the Constraints appearing under the heading "Constraint:" in the Abstract Syntax.

Step 5. Each <do-spec> MUST satisfy the Constraints specified in Section 6.4.4.

Step 6. Perform VALIDATE-LEAVE-STATEMENTS(aeu).

4.7.1 Validate-declaration

When all the <declaration>s have been completed, and their contained <expression>s and <value-reference>s properly completed, some additional validation is required to ensure that such <declaration>s do represent realistic data entities.

Operation: VALIDATE-DECLARATION(ad)

Where: ad is a <declaration>.

Step 1. If ad contains <automatic>, <based>. <defined>, <parameter>, or <static> then perform VALIDATE-AUTOMATIC-DECLARATION(ad), VALIDATE-BASED-DECLARATION(ad), VALIDATE-DEFINED-DECLARATION(ad), VALIDATE-PARAMETER-DECLARATION(ad), or VALIDATE-STATIC-DECLARATION(ad), respectively.

Step 2. If ad contains a <named-constant>,nc but nc does not immediately contain <arithmetic> or <string>, then perform VALIDATE-STATIC-DECLARATION(ad).

Step 3. If ad contains any <data-description>,dd which simply contains <initial>, then for each such dd perform Step 3.1.

Step 3.1. The <data-description> immediate component of each <parenthesized-expression> immediate component of an <initial-element> contained in dd MUST be proper for assignment to dd.

4.7.2 Validate-automatic-declaration

Automatic declarations MUST satisfy constraints which enable them to be allocated and possibly initialized at the time that the block to which they belong is being activated.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Operation: VALIDATE-AUTOMATIC-DECLARATION(ad)

Where: ad is a <declaration>.

Step 1. ad MUST NOT contain an <area-size>: <asterisk>;, a <maximum-length>: <asterisk>;, or a <bound-pair>: <asterisk>;, except as subnodes of <entry>.

Step 2. Each <extent-expression> in ad MUST NOT contain a <refer-option>.

Step 3. For each <variable-reference>: <declaration-designator>,vdd; contained in ad, which is not contained in an <offset>, perform Step 3.1.

Step 3.1. vdd MUST NOT designate a <declaration>,d containing <automatic> or <defined> if d is a block-component of the same block as ad.

Step 4. For each <item-data-description>,idd in ad that is contained in a <dimensional-data-description> also contained in ad, perform Step 4.1.

Step 4.1. If idd immediately contains an <initial>,init then init MUST immediately contain an <array-init>.

4.7.3 Validate-based-declaration

Based declarations may contain a <structure-data-description> whose last <member-description> refers to another (previous) member of the same structure by means of the <refer-option>.

Operation: VALIDATE-BASED-DECLARATION(bd)

Where: bd is a <declaration>.

Step 1. Each <area-size>, <maximum-length>, or <bound-pair> in bd MUST NOT contain <asterisk>, except as subnodes of <entry>. bd MUST NOT contain more than one <refer-option>.

Step 2. If bd has a <refer-option>,ro, perform Steps 2.1 through 2.5 .

Step 2.1. ro MUST be simply contained in a <member-description>,ms.

Step 2.2. Let id[1],...,id[n] be the components of the <identifier-list> in ro. bd MUST simply contain a <structure-data-description>,sd[1]: idl[1] mdl[1]. For i=1,...,(n−2), in order, perform Step 2.2.1.

Step 2.2.1. idl[i] has an <identifier>,idc[i] equal to id[i+1]. The <member-description> in mdl[i] corresponding to idc[i] in idl[i] MUST simply contain a <structure-data-description>,sd[i+1]: idl[i+1] mdl[i+1].

Step 2.3. idl[n−1] has an <identifier>,idc[n−1] equal to id[n]. The <member-description>,mo in mdl[n−1] corresponding to idc[n−1] in idl[n−1] MUST immediately contain a <data-description>: <item-data-description>,idd. idd MUST have a <data-type> with <fixed> and <binary>. mo MUST NOT be contained in a <dimensioned-data-description> or <union-data-description>.

Step 2.4. mo MUST precede ms in bd.

Step 2.5. There MUST NOT exist an <item-data-description> in sd[1] which follows ms.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.7.4 Validate-defined-declaration

Operation: VALIDATE-DEFINED-DECLARATION(dd)

Where: dd is a <declaration>.

Step 1. dd MUST NOT contain an <area-size>: <asterisk>;, a <maximum-length>: <asterisk>;, or a <bound-pair>: <asterisk>;, except as subnodes of <entry>.

Step 2. Each <extent-expression> in dd MUST NOT contain a <refer-option>.

Step 3. For each <variable-reference>: <declaration-designator>,vdd; contained in dd, which is not contained in a <base-item> or <offset>, perform Step 3.1.

Step 3.1. vdd MUST NOT designate a <declaration>,d containing <automatic> or <defined> if d is a block-component of the same block as dd.

4.7.5 Validate-parameter-declaration

Operation: VALIDATE-PARAMETER-DECLARATION(pd)

Where: pd is a <declaration>.

Step 1. Each <extent-expression> in pd MUST NOT contain a <refer-option>.

Step 2. For each <extent-expression>: <expression>,e; in pd perform Step 2.1.

Step 2.1. Perform EVALUATE-RESTRICTED-EXPRESSION(e) to obtain c. c MUST be a <constant>. Replace the first immediate component of e by c.

4.7.6 Validate-static-declaration

Operation: VALIDATE-STATIC-DECLARATION(sd)

Where: sd is a <declaration>.

Step 1. sd MUST NOT contain an <area-size>: <asterisk>;, a <maximum-length>: <asterisk>;, or a <bound-pair>: <asterisk>;, except as subnodes of <entry>.

Step 2. Each <extent-expression> in sd MUST NOT contain a <refer-option>.

Step 3. For each <variable-reference>: <declaration-designator>,vdd; contained in sd, which is not contained in an <offset>, perform Step 3.1.

Step 3.1. vdd MUST NOT designate a <declaration>,d containing <automatic> or <defined> if d is a block-component of the same block as sd.

Step 4. For each <expression>,e simple component of sd which is not a component of an <offset> perform EVALUATE-RESTRICTED-EXPRESSION(e) to obtain c, which MUST be a <constant> or a <value-reference>. If c is a <value-reference> then it must satisfy the following constraints:

1. c is contained in an <initial-element> or <array-init>, and
2. c immediately contains a <builtin-function-reference> with <empty-bif> or <null-bif>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.7.7 Validate-descriptor

Operation: VALIDATE-DESCRIPTOR(d)

Where: d is a <parameter-descriptor> or a <returns-descriptor>.

Step 1. For each <extent-expression>: <expression>,e; in d, perform Step 1.1.

Step 1.1. Perform EVALUATE-RESTRICTED-EXPRESSION(e) to obtain c. c MUST be a <constant>. Replace the first immediate component of e by c.

Step 2. If d is a <parameter-descriptor>, any <entry> simply contained in d MUST NOT have any subnodes.

Step 3. d MUST NOT contain any <refer-option>s.

4.7.8 Evaluate-restricted-expression

Operation: EVALUATE-RESTRICTED-EXPRESSION(e)

Where: e is an <expression>.

Result: a <constant> or a <value-reference> or ⩽fail⩾.

Step 1. For each <expression>,ex simply contained in e, perform Steps 1.1 and 1.2.

Step 1.1. Perform EVALUATE-RESTRICTED-EXPRESSION(ex) to obtain r. If r is ⩽fail⩾ then return ⩽fail⩾.

Step 1.2. Replace the first immediate component of ex by r.

Step 2.

Case 2.1. e immediately contains a <value-reference>,vr.

Case 2.1.1. vr immediately contains a <named-constant-reference>,ncr.

Step 2.1.1.1. Let d be the <declaration> designated by the <declaration-designator> of ncr. If the <named-constant> of d does not immediately contain <arithmetic> or <string>, then return vr.

Step 2.1.1.2. Perform EVALUATE-NAMED-CONSTANT-REFERENCE(ncr) to obtain an ⩽aggregate-value⩾,av.

Case 2.1.2. vr immediately contains a <builtin-function-reference>,bfr.

Step 2.1.2.1. If bfr simply contains <empty-bif> or <null-bif>, then return vr.

Step 2.1.2.2. If bfr simply contains <date-bif>, <oncode-bif>, <onfile-bif>, <onkey-bif>, <onsource-bif>, <time-bif>, or <datetime-bif>, then return ⩽fail⩾.

Step 2.1.2.3. If bfr simply contains an <argument-list>,al, then return ⩽fail⩾ if any <expression> simply contained in al immediately contains a <value-reference>.

Step 2.1.2.4. Perform EVALUATE-BUILTIN-FUNCTION-REFERENCE(bfr) to obtain an ⩽aggregate-value⩾,av.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.1.3. vr immediately contains a <enquiry-reference>,rfr.

rfr simply contains <epsilon-enquiry>, <maxval-enquiry>, <minval-enquiry>, <null-enquiry>, or <radix-enquiry>. Perform EVALUATE-ENQUIRY-REFERENCE(rfr) to obtain an ⋖aggregate-value⋗,av.

Case 2.1.4. (Otherwise).

Return ⋖fail⋗.

Case 2.2. (Otherwise).

Perform EVALUATE-EXPRESSION(e) to obtain an ⋖aggregate-value⋗,av.

Step 3. av MUST immediately contain an ⋖aggregate-type⋗ which immediately contains ⋖scalar⋗. Let v be the ⋖basic-value⋗ in av.

Step 4. Let dt be the <data-type> in the <data-description> immediate component of e.

Step 5. Return a <constant>: v dt.

4.7.9 Apply-constraints

In certain contexts of an <external-unit> only restricted forms of the specified category are permitted. The restrictions are shown by a constraint-expression enclosed in parentheses. The definition of a constraint-expression is as follows.

constraint-expression ::= multiple-constraint | constraint-expression OR multiple-constraint

multiple-constraint ::= constraint | multiple-constraint AND constraint

constraint ::= NOT constraint | area | arithmetic | based | bit | character | condition | named-constant | defined | computational-type | file | fixed | format | integer | label | locator | pointer | scalar | union | variable

Operation: APPLY-CONSTRAINTS (aeu)

Where: aeu is an <external-unit>.

Step 1. For each subnode, c in aeu corresponding to a category-name in the Abstract Syntax with an attached constraint-expression,ce, if c is an <expression-list>, let c be each component of the list in turn, and perform Steps 1.1 and 1.2; otherwise, perform Steps 1.1 and 1.2.

Step 1.1.

Case 1.1.1. c is a <variable-reference> and ce contains a constraint containing "defined" or "based".

Let d be the <declaration> designated by the <declaration-designator> immediately contained in c.

Case 1.1.2. c is a <declaration-designator>.

Let d be the <declaration> designated by c.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1.1.3. c is a <variable-reference> (other than as in Case 1.1.1), a <value-reference>, an <expression>, a <parenthesized-expression>, a <target-reference>, or a <named-constant-reference>.

Let d be the <data-description> immediately contained in c.

Step 1.2. Perform TEST-CONSTRAINTS(d,ce) to obtain r. r MUST be ⩽true⩾.

4.7.10 Test-constraints

Operation: TEST-CONSTRAINTS(d,ce)

Where: d is a <declaration> or a <data-description>,
ce is a constraint-expression, a multiple-constraint, or a constraint.

Result: ⩽true⩾ or ⩽false⩾.

Case 1. ce is of the form constraint-expression: constraint-expression,cce OR multiple-constraint,mc.

Perform TEST-CONSTRAINTS(d,cce) to obtain r1; perform TEST-CONSTRAINTS(d,mc) to obtain r2. If r1 is ⩽true⩾, return ⩽true⩾; otherwise return r2.

Case 2. ce is of the form multiple-constraint: multiple-constraint,mc AND constraint,ct.

Perform TEST-CONSTRAINTS(d,mc) to obtain r1; perform TEST-CONSTRAINTS(d,ct) to obtain r2. If r1 is ⩽false⩾, return ⩽false⩾; otherwise return r2.

Case 3. ce is of the form constraint: NOT constraint,ct.

Perform TEST-CONSTRAINTS(d,ct) to obtain r. If r is ⩽false⩾, return ⩽true⩾; otherwise return ⩽false⩾.

Case 4. ce is a constraint other than in Case 3.

Step 4.1. If d is a <declaration> then let dd be the <data-description> in d. Otherwise, let dd be d. Let cat be the subnode of ce.

Step 4.2.

Case 4.2.1. cat is scalar.

If d is a <declaration> and does not contain <variable> then return ⩽false⩾. If dd immediately contains an <item-data-description> then return ⩽true⩾. Otherwise return ⩽false⩾.

Case 4.2.2. cat is union.

If dd contains a <union-data-description> not contained in an <expression> or <data-type> in dd then return ⩽true⩾. Otherwise return ⩽false⩾.

Case 4.2.3. cat is integer.

If dd contains a <data-type>, which is not contained in a <expression> or <data-type> in dd, with <fixed> and a <scale-factor> with 0 then return ⩽true⩾. Otherwise return ⩽false⩾.

Case 4.2.4. cat is arithmetic, area, bit, character, computational-type, file, fixed, format, label, locator, or pointer.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If dd contains a category-name equal to <cat> which is not contained in an <expression> or <data-type> in dd then return ≤true≥. Otherwise return ≤false≥.

Case 4.2.5. cat is based, condition, named-constant, defined, or variable.

If d is a <data-description> then return ≤false≥. If d contains a category-name equal to <cat> then return ≤true≥. Otherwise return ≤false≥.

Case 5. (Otherwise).

Let cs be the immediate component of ce. Perform TEST-CONSTRAINTS(d,cs) to obtain r. Return r.

4.7.11 Validate-leave-statements

This operation checks for invalid uses of LEAVE statements. In particular, it verifies that a LEAVE statement does not attempt to exit from a <begin-block>, that the LEAVE statement is contained in a <group>. It also verifies that the optional label reference in the LEAVE statement refers to a label on a <group> which contains the LEAVE statement, and that the reference is not subscripted.

Operation: VALIDATE-LEAVE-STATEMENTS(aeu)

Where: aeu is an <external-unit>.

Step 1. For each <leave-statement>,ls contained in aeu, perform Steps 1.1 and 1.2 .

Step 1.1. Let b be the abstract-block which block-contains ls.

Step 1.2.

Case 1.2.1. ls has a <named-constant-reference>,lv.

Step 1.2.1.1. Let d be the <declaration> designated by the <declaration-designator> in lv. b MUST block-contain d.

Step 1.2.1.2. The <named-constant> in d MUST NOT have a <bound-pair>.

Step 1.2.1.3. Let eu be the <executable-unit> block-contained in b which has a <statement-name> containing an <identifier> equal to the <identifier> in d.

Step 1.2.1.4. eu MUST immediately contain a <group>,g, which contains ls.

Case 1.2.2. ls does not have a <named-constant-reference>.

ls MUST be contained in a <group> which is contained in b.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.8 Validate-program

Operation: VALIDATE-PROGRAM

Step 1. Let n be the number of distinct <identifier> components of the <program> which are immediate components of a <declaration> which contains <external>. Let id[i], i=1,...,n be the distinct <identifier>s.

Step 2. For i=1,...,n, in any order, perform Steps 2.1 through 2.3 .

Step 2.1. Let adl be a <declaration-list> containing copies of all <declaration>s which contain <external> and immediately contain an <identifier> equal to id[i].

Step 2.2. Perform VALIDATE-EXTERNAL-DECLARATION(adl).

Step 2.3.

Case 2.3.1. All of the <declaration>s in adl contain an <external-name>.

Let en be the <external-name> in the first <declaration> of adl (note that all the <external-name>s of adl are equal).

Perform MAP-EXTERNAL-NAME-TO-ENVIRONMENT-NAME(en) to obtain an <environment-name>,envm[i].

Case 2.3.2. None of the <declaration>s in adl contain an <external-name>.

Perform MAP-IDENTIFIER-TO-ENVIRONMENT-NAME(id[i]) to obtain an <environment-name>,envm[i].

Step 3. All of the <environment-name>s, envm[i], i=1,...,n, MUST be distinct (i.e. not equal).

4.8.1 Validate-external-declaration

Operation: VALIDATE-EXTERNAL-DECLARATION(adl)

Where: adl is a <declaration-list>.

Step 1. Delete all <identifier-list> components of adl which are immediate components of <union-data-description> or <structure-data-description>.

Step 2. Delete all <variable-reference>s which are immediate components of an <offset>.

Step 3. For each <declaration>,d which has <storage-class>: <static>; and for each <item-data-description> component of d which contains an <initial>, change any <initial-element> which contains an <iteration-factor> and an <initial-element-list> into the equivalent number of <initial-element>s.

Step 4. For each <extent-expression>,ee in adl containing a <constant>,c, perform Step 4.1.

Step 4.1. Let e be the <expression> of ee. Perform EVALUATE-EXPRESSION-TO-INTEGER(e) to obtain an <integer-value>,iv. Replace e by iv.

Step 5. For each <initial-element>,ie in adl containing a <constant>,c, perform Step 5.1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 5.1. Let tdt be the <data-type> immediately contained in the <item-data-description> containing ie. Let cdt be the <data-type> of c, and let cbv be the ⩽basic-value⩾ of c. Perform CONVERT(tdt,cdt,cbv) to obtain a ⩽basic-value⩾,bv. Replace c by <constant>: bv tdt.

Step 6. In an IMPLEMENTATION-DEFINED fashion, compare <declaration>s of adl which have <environment> components with those which do not have corresponding <environment> components, and compare corresponding <environment> components of the <declaration>s of adl. Delete all <environment> components of adl.

Step 7. In an IMPLEMENTATION-DEFINED fashion, compare <declaration>s of adl which have <options> components with those which do not have corresponding <options> components, and compare corresponding <options> components of the <declaration>s of adl. Delete all <options> components of adl.

Step 8. Let adlc be a copy of adl. Delete each <declaration> component of adlc that contains <reserved>. adlc MUST NOT be empty. All <declaration> components of adlc MUST be equal.

Step 9. Delete from adl every <initial> component and every <reserved> component. All <declaration> components of adl MUST be equal.

4.8.2 Environment-names

<environment-name> ::= The category <environment-name> is IMPLEMENTATION-DEFINED and represents a name used in the operating environment to designate PL/I EXTERNAL items.

Due to the IMPLEMENTATION-DEFINED freedoms in the following operations, an implementation is free to make any restrictions in the composition of external <identifier>s and the composition and meaning of <external-name>s.

Operation: MAP-EXTERNAL-NAME-TO-ENVIRONMENT-NAME(en)

Where: en is an <external-name>.

Result: an <environment-name>

Step 1. en MUST be acceptable to the implementation.

Step 2. In an IMPLEMENTATION-DEFINED manner, create an <environment-name>,envm depending on en.

Step 3. Return envm.

Operation: MAP-IDENTIFIER-TO-ENVIRONMENT-NAME(id)

Where: id is an <identifier>.

Result: an <environment-name>.

Step 1. In an IMPLEMENTATION-DEFINED manner, create an <environment-name>,envm depending on id.

Step 2. Return envm.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5. The PL/I Interpreter

5.1 Introduction

This chapter gives the interpretation-state part of the Machine-state Syntax and also introduces the interpretation phase of the definition. Section 5.2 defines the interpretation-state. Section 5.3 defines some terminology used in the subsequent chapters. Section 5.4 gives the operation INTERPRET and some operations called from it to initialize and terminate the interpretation phase. The subsequent chapters complete the definition of the interpretation phase.

5.2 The Interpretation-state

M6. <interpretation-state> ::= <program-state> <allocated-storage> <dataset-list>

M7. <program-state> ::= <program-directory> [<block-state-list>] [<file-information-list>]

5.2.1 Directories

M8. <program-directory> ::= <static-directory> <file-directory>

M9. <static-directory> ::= [<static-directory-entry-list>]

M10. <static-directory-entry> ::= { <external> | <declaration-designator> }
<identifier> <generation>

M11. <file-directory> ::= [<file-directory-entry-list>]

M12. <file-directory-entry> ::= { <external> | <declaration-designator> }
<identifier> <file-information-designator>

5.2.2 Block State

M13. <block-state> ::= <block-directory> <block-control> <linkage-part> [<block-environment>]
[<established-on-unit-list>] [<condition-bif-value-list>]

M14. <block-directory> ::= <automatic-directory> <defined-directory> [<parameter-directory>]

M15. <automatic-directory> ::= [<automatic-directory-entry-list>]

M16. <automatic-directory-entry> ::= <identifier> <generation>

M17. <defined-directory> ::= [<defined-directory-entry-list>]

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

M18. ⟨defined-directory-entry⟩ ::= <identifier> ⟨evaluated-data-description⟩

M19. ⟨parameter-directory⟩ ::= [⟨parameter-directory-entry-list⟩]

M20. ⟨parameter-directory-entry⟩ ::= <identifier> {⟨undefined⟩ | ⟨established-argument⟩}

M21. ⟨established-argument⟩ ::= ⟨generation⟩ { ⟨dummy⟩ | ⟨not-dummy⟩ }

M22. ⟨evaluated-entry-reference⟩ ::= ⟨entry-value⟩ [⟨established-argument-list⟩]

M23. ⟨block-control⟩ ::= <executable-unit-designator> ⟨group-control⟩
⟨statement-control⟩ [⟨string-io-control⟩]
[⟨data-item-control-list⟩] [⟨format-control-list⟩]
[⟨current-scalar-item-list⟩] [⟨remote-block-state⟩]
[⟨current-file-value⟩] [<when-unit-designator>]

M24. ⟨remote-block-state⟩ ::= ⟨block-state-designator⟩

M25. ⟨current-file-value⟩ ::= ⟨file-value⟩

M26. ⟨group-control⟩ ::= [⟨controlled-group-state-list⟩]

M27. ⟨controlled-group-state⟩ ::= <spec-designator> ⟨cv-target⟩ ⟨cv-type⟩
[⟨by-value⟩ ⟨converted-by-type⟩]
[⟨to-value⟩ ⟨converted-to-type⟩]

M28. ⟨cv-target⟩ ::= ⟨evaluated-target⟩

M29. ⟨cv-type⟩ ::= <data-type>

M30. ⟨by-value⟩ ::= ⟨real-value⟩

M31. ⟨converted-by-type⟩ ::= <data-type>

M32. ⟨to-value⟩ ::= ⟨real-value⟩

M33. ⟨converted-to-type⟩ ::= <data-type>

M34. ⟨statement-control⟩ ::= ⟨operation-list⟩

M35. ⟨string-io-control⟩ ::= [⟨alpha⟩] ⟨character-string-value⟩ [⟨string-limit⟩] [⟨first-comma⟩]

M36. ⟨string-limit⟩ ::= ⟨integer-value⟩

M37. ⟨first-comma⟩ ::= ⟨on⟩ | ⟨off⟩

M38. ⟨data-item-control⟩ ::= ⟨data-list-indicator⟩ ⟨data-item-indicator⟩

M39. ⟨data-list-indicator⟩ ::= ⟨designator⟩

M40. ⟨data-item-indicator⟩ ::= ⟨designator⟩ | ⟨undefined⟩

M41. ⟨format-control⟩ ::= <format-specification-list-designator>
⟨format-list-index⟩[⟨format-iteration-value⟩] [⟨format-iteration-index⟩]
[<format-statement-designator>] [⟨remote-block-state⟩]

M42. ⟨format-list-index⟩ ::= ⟨integer-value⟩

M43. ⟨format-iteration-value⟩ ::= ⟨integer-value⟩

M44. ⟨format-iteration-index⟩ ::= ⟨integer-value⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

M45. ⟨current-scalar-item⟩ ::= ⟨basic-value⟩ <data-type> | ⟨evaluated-target⟩

M46. ⟨linkage-part⟩ ::= [<entry-point-designator>] [⟨returned-value⟩ | ⟨returned-onsource-value⟩] [⟨prologue-flag⟩]

M47. ⟨returned-value⟩ ::= ⟨aggregate-value⟩

M48. ⟨returned-onsource-value⟩ ::= ⟨character-string-value⟩

M49. ⟨block-environment⟩ ::= ⟨block-state-designator⟩

M50. ⟨established-on-unit⟩ ::= ⟨evaluated-condition⟩ {⟨entry-value⟩ | <system-action>} [<snap>]

M51. ⟨evaluated-condition⟩ ::= <computational-condition> | <area-condition> | ⟨evaluated-io-condition⟩ | <error-condition> | <programmer-named-condition> | <finish-condition> | <storage-condition>

M52. ⟨evaluated-io-condition⟩ ::= <io-condition> ⟨file-value⟩

M53. ⟨condition-bif-value⟩ ::= ⟨oncode-value⟩ | ⟨onfile-value⟩ | ⟨onkey-value⟩ | ⟨onsource-value⟩

M54. ⟨oncode-value⟩ ::= ⟨integer-value⟩

M55. ⟨onfile-value⟩ ::= ⟨character-string-value⟩

M56. ⟨onkey-value⟩ ::= ⟨character-string-value⟩

M57. ⟨onsource-value⟩ ::= ⟨character-string-value⟩

5.2.3 File Information

M58. ⟨file-information⟩ ::= ⟨open-state⟩ ⟨filename⟩ [⟨evaluated-environment⟩] <file-description-set> [⟨file-opening⟩]

M59. ⟨open-state⟩ ::= ⟨open⟩ | ⟨closed⟩

M60. ⟨filename⟩ ::= ⟨character-string-value⟩

M61. ⟨file-opening⟩ ::= ⟨dataset-designator⟩ ⟨complete-file-description⟩ ⟨current-position⟩ [⟨delete-flag⟩] [⟨page-top-flag⟩] [⟨allocated-buffer⟩] [⟨page-number⟩] [⟨first-comma⟩]

M62. ⟨complete-file-description⟩ ::= ⟨evaluated-file-description-list⟩

M63. ⟨evaluated-file-description⟩ ::= <stream> | <record> | <input> | <output> | <update> | <sequential> | <direct> | <print> | <nonprint> | <keyed> | ⟨evaluated-environment⟩ | ⟨evaluated-tab-option⟩ | ⟨evaluated-title⟩ | ⟨evaluated-linesize⟩ | ⟨evaluated-pagesize⟩

M64. ⟨evaluated-tab-option⟩ ::= ⟨integer-value-list⟩

M65. ⟨evaluated-title⟩ ::= ⟨character-string-value⟩

M66. ⟨evaluated-linesize⟩ ::= ⟨integer-value⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

M67. ⟨evaluated-pagesize⟩ ::= ⟨integer-value⟩

M68. ⟨evaluated-environment⟩ ::= This category is IMPLEMENTATION-DEFINED.

M69. ⟨current-position⟩ ::= ⟨designator⟩ | ⟨undefined⟩

M70. ⟨allocated-buffer⟩ ::= ⟨generation⟩

M71. ⟨page-number⟩ ::= ⟨integer-value⟩

5.2.4 Storage and Values

M72. ⟨allocated-storage⟩ ::= [⟨allocation-unit-list⟩]

M73. ⟨allocation-unit⟩ ::= ⟨basic-value-list⟩ [⟨union-list⟩] ⟨evaluated-data-description⟩

M74. ⟨basic-value⟩ ::= ⟨real-value⟩ | ⟨character-string-value⟩ | ⟨bit-string-value⟩ | ⟨entry-value⟩ | ⟨label-value⟩ | ⟨format-value⟩ | ⟨file-value⟩ | ⟨pointer-value⟩ | ⟨offset-value⟩ | ⟨area-value⟩ | ⟨undefined⟩

M75. ⟨real-value⟩ ::= ⟨real-number⟩

M76. ⟨real-number⟩ ::= The members of the set of real numbers are the alternative choices as immediate and terminal components of ⟨real-number⟩.

M77. ⟨integer-value⟩ ::= The members of the set of integers are the alternative choices as immediate and terminal components of ⟨integer-value⟩.

M78. ⟨character-string-value⟩ ::= ⟨character-value-list⟩ | ⟨null-character-string⟩

M79. ⟨character-value⟩ ::= {⟨symbol⟩} | ⟨undefined⟩

M80. ⟨bit-string-value⟩ ::= ⟨bit-value-list⟩ | ⟨null-bit-string⟩

M81. ⟨bit-value⟩ ::= ⟨zero-bit⟩ | ⟨one-bit⟩ | ⟨undefined⟩

M82. ⟨entry-value⟩ ::= { <entry-point-designator> [⟨block-state-designator⟩] } | ⟨null-entry⟩

M83. ⟨label-value⟩ ::= { <executable-unit-designator> ⟨block-state-designator⟩ } | ⟨null-label⟩

M84. ⟨format-value⟩ ::= <format-statement-designator> ⟨block-state-designator⟩

M85. ⟨file-value⟩ ::= ⟨file-information-designator⟩ | ⟨null-file⟩

M86. ⟨pointer-value⟩ ::= ⟨generation⟩ | ⟨null⟩

M87. ⟨offset-value⟩ ::= ⟨evaluated-data-description⟩ ⟨area-action-list⟩ ⟨storage-index-list⟩ | ⟨null⟩

M88. ⟨area-value⟩ ::= ⟨area-action-list⟩ | ⟨empty⟩

M89. ⟨area-action⟩ ::= ⟨area-allocation⟩ | ⟨area-freeing⟩ | ⟨area-size⟩

M90. ⟨area-allocation⟩ ::= ⟨evaluated-data-description⟩ [⟨allocation-unit⟩]

M91. ⟨area-freeing⟩ ::= ⟨integer-value⟩

M92. ⟨area-size⟩ ::= ⟨integer-value⟩

M93. ⟨union⟩ ::= ⟨union-alternative-list⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

M94. ⟨union-alternative⟩ ::= ⟨basic-value-index-list⟩

M95. ⟨aggregate-value⟩ ::= ⟨aggregate-type⟩ ⟨basic-value-list⟩

M96. ⟨aggregate-type⟩ ::= ⟨dimensioned-aggregate-type⟩ | ⟨structure-aggregate-type⟩ | ⟨union-aggregate-type⟩ | ⟨scalar⟩

M97. ⟨dimensioned-aggregate-type⟩ ::= ⟨element-aggregate-type⟩ <bound-pair-list>

M98. ⟨element-aggregate-type⟩ ::= ⟨structure-aggregate-type⟩ | ⟨scalar⟩

M99. ⟨structure-aggregate-type⟩ ::= ⟨member-aggregate-type-list⟩

M100. ⟨member-aggregate-type⟩ ::= ⟨aggregate-type⟩

M101. ⟨union-aggregate-type⟩ ::= ⟨member-aggregate-type-list⟩

5.2.5 Generations, Evaluated Data Descriptions, and Evaluated Targets

M102. ⟨generation⟩ ::= ⟨evaluated-data-description⟩ ⟨allocation-unit-designator⟩ ⟨storage-index-list⟩

M103. ⟨evaluated-data-description⟩ ::= <data-description>

Note: <extent-expression> components of ⟨evaluated-data-description⟩: <data-description> contain only ⟨integer-value⟩s. (See Section 7.2)

M104. ⟨storage-index⟩ ::= ⟨basic-value-index⟩ [⟨position-index⟩]

M105. ⟨basic-value-index⟩ ::= ⟨integer-value⟩

M106. ⟨position-index⟩ ::= ⟨integer-value⟩

M107. ⟨evaluated-target⟩ ::= ⟨generation⟩ | ⟨evaluated-pseudo-variable-reference⟩

M108. ⟨evaluated-pseudo-variable-reference⟩ ::= <pseudo-variable> [⟨generation⟩ | ⟨aggregate-value⟩] [⟨aggregate-value⟩] [⟨aggregate-value⟩]

5.2.6 Dataset

M109. ⟨dataset⟩ ::= ⟨dataset-name⟩ {⟨record-dataset⟩ | ⟨stream-dataset⟩}

M110. ⟨dataset-name⟩ ::= ⟨character-string-value⟩

M111. ⟨record-dataset⟩ ::= ⟨sequential-dataset⟩ | ⟨keyed-dataset⟩ | ⟨keyed-sequential-dataset⟩

M112. ⟨sequential-dataset⟩ ::= ⟨alpha⟩ [⟨record-list⟩] ⟨omega⟩

M113. ⟨keyed-dataset⟩ ::= [⟨keyed-record-list⟩]

M114. ⟨keyed-sequential-dataset⟩ ::= ⟨alpha⟩ [⟨keyed-record-list⟩] ⟨omega⟩

M115. ⟨keyed-record⟩ ::= ⟨record⟩ ⟨key⟩

M116. ⟨record⟩ ::= ⟨evaluated-data-description⟩ ⟨basic-value-list⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

M117. ⟨key⟩ ::= ⟨character-string-value⟩

M118. ⟨stream-dataset⟩ ::= ⟨alpha⟩ [⟨stream-item-list⟩] ⟨omega⟩

M119. ⟨stream-item⟩ ::= {⟨symbol⟩} | ⟨linemark⟩ | ⟨pagemark⟩ | ⟨carriage-return⟩

5.3 Terminology and Definitions

The following terms are employed at various places throughout the operations which comprise the interpretation phase.

5.3.1 Current

1. The last ⟨block-state⟩ member (if any) of the ⟨block-state-list⟩ is termed the *current* ⟨block-state⟩.

2. Excepting only components of its ⟨controlled-group-state-list⟩ simple component (if any), any component of the current ⟨block-state⟩ is termed *current.* For example, the <executable-unit-designator> simple component of the current ⟨block-state⟩ is termed the current <executable-unit-designator>.

3. The last ⟨controlled-group-state⟩ member (if any) of the current ⟨controlled-group-state-list⟩ is termed the *current* ⟨controlled-group-state⟩.

4. Any component of the current ⟨controlled-group-state⟩ is termed *current.* For example, the ⟨by-value⟩ component of the current ⟨controlled-group-state⟩ is termed the current ⟨by-value⟩.

5. The corresponding block (see Section 5.3.2) of the current ⟨block-state⟩ is termed the *current block* .

6. Any simple component of the current block is also termed *current.* For example, the <end-statement> immediate component of the current block is termed the current <end-statement>.

5.3.2 Block

The term *block* is used to refer to a <begin-block> or a <procedure>. Each ⟨block-state⟩ present in the ⟨machine-state⟩ is created to be associated with some particular block. That block is termed the *corresponding block* of the ⟨block-state⟩ and it may be located since it is that block which has as block-component the <executable-unit> designated by the <executable-unit-designator> simple component of the ⟨block-state⟩ in question.

5.4 The Interpret Operation and Interpretation State Initialization

5.4.1 Interpret

First, the ⟨interpretation-state⟩ is initialized, and then the <program> is executed.

Operation: INTERPRET(dl,ev)

Where: dl is a ⟨dataset-list⟩,
ev is an ⟨entry-value⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 1. Perform INITIALIZE-INTERPRETATION-STATE(dl).

Step 2. Let eer be ⟨evaluated-entry-reference⟩: ev. Perform ACTIVATE-PROCEDURE(eer).

Step 3. Perform PROGRAM-EPILOGUE.

5.4.2 Initialize-interpretation-state

INITIALIZE-INTERPRETATION-STATE constructs the initial configuration of the ⟨interpretation-state⟩, including certain portions that are a function of the <program> to be interpreted.

Operation: INITIALIZE-INTERPRETATION-STATE(dl)

Where: dl is a ⟨dataset-list⟩.

Step 1. Append to the ⟨machine-state⟩ the tree

⟨interpretation-state⟩:
⟨program-state⟩:
⟨program-directory⟩:
⟨static-directory⟩
⟨file-directory⟩;;
⟨allocated-storage⟩
dl.

Step 2. Perform BUILD-FILE-DIRECTORY-AND-INFORMATIONS.

Step 3. Perform ALLOCATE-STATIC-STORAGE-AND-BUILD-STATIC-DIRECTORY.

5.4.3 Build-file-directory-and-informations

Operation: BUILD-FILE-DIRECTORY-AND-INFORMATIONS

Step 1. For each <declaration>,d in the <program>, which has <named-constant> with <file>, selected in any order:

Case 1.1. d has <external>, and there exists in the ⟨file-directory⟩ a ⟨file-directory-entry⟩ with both <external> and an <identifier> that is equal to the <identifier> in d.

No action.

Case 1.2. (Otherwise).

Perform BUILD-FDI(d).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.4.4 Build-fdi

Operation: BUILD-FDI(d)

Where: d is a <declaration>,

Step 1. Let fn be a ⟪filename⟫ whose component ⟪symbol⟫s are those of the <identifier> in d, and are taken in the same order. Let fd be the <file-description-set> in d. Let info be a

⟪file-information⟫:
⟪open-state⟫:
⟪closed⟫;
fn
fd.

Step 2. If fd has an <environment>,env, then evaluate env in an IMPLEMENTATION-DEFINED manner to obtain an ⟪evaluated-environment⟫,eenv, and attach eenv to info.

Step 3. Append info to the ⟪file-information-list⟫. Let fid be a ⟪file-information-designator⟫ which designates this appended ⟪file-information⟫ node.

Step 4. If d has <external>, then let sc be <external>; otherwise let sc be a <declaration-designator> designating d. Let id be the <identifier> in d. Let fde be

⟪file-directory-entry⟫:
sc
id
fid.

Step 5. Append fde to the ⟪file-directory-entry-list⟫.

5.4.5 Allocate-static-storage-and-build-static-directory

Operation: ALLOCATE-STATIC-STORAGE-AND-BUILD-STATIC-DIRECTORY

Step 1. For each <declaration>,d in the <program>, selected in any order:

Case 1.1. d has <static>.

Case 1.1.1. d has <external> but does not contain <reserved>, and there exists in the ⟪static-directory⟫ a ⟪static-directory-entry⟫ with both <external> and an <identifier> that is equal to the <identifier> in d.

No action.

Case 1.1.2. d has <reserved>.

No action.

Case 1.1.3. (Otherwise).

Step 1.1.3.1. Let dd be the <data-description> in d. Perform EVALUATE-DATA-DESCRIPTION-FOR-ALLOCATION(dd) to obtain an ⟪evaluated-data-description⟫,edd.

Step 1.1.3.2. Perform ALLOCATE(edd) to obtain a ⟪generation⟫,g.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.1.3.3. If d has <initial>, then perform INITIALIZE-GENERATION(g,d).

Step 1.1.3.4. If d has <external>, then let sc be <external>; otherwise let sc be a <declaration-designator> designating d. Let id be the <identifier> in d.

Step 1.1.3.5. Append, to the ≪static-directory-entry-list≫, a ≪static-directory-entry≫: sc id g.

Case 1.2. (Otherwise).

No action.

5.4.6 Program-epilogue

Operation: PROGRAM-EPILOGUE

Step 1. For each ≪file-information≫,fi containing ≪open≫ perform CLOSE(fv), where fv is a ≪file-value≫ designating fi.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6. Flow Of Control

6.1 Introduction

The definition of the control mechanism of the PL/I Interpreter, introduced in Chapter 5, is completed in this chapter. The definition treats in order the three levels of control, pertaining to the program, the block, and the operations within the block. This is followed by the definition of the control of interrupt operations.

Within the execution of the program, there may be several blocks active at any time, but execution proceeds sequentially only within the most recently activated block while the execution of the other blocks is temporarily suspended.

6.2 Program Activation and Termination

The activation of a program is described in Section 5.4, the initialization of the ⟨program-state⟩ being followed by the performance of the ACTIVATE-PROCEDURE operation. This causes the first ⟨block-state⟩ to be created.

A ⟨program-state⟩ in general contains a list of ⟨block-state⟩s, one for each block activated within it and not yet terminated. When the execution of a program is terminated, all the contained block activations are also terminated. The deletion of the last remaining ⟨block-state⟩ results in control returning to Step 3 of the INTERPRET operation and performance of the PROGRAM-EPILOGUE operation (see Section 5.4.6).

6.2.1 Program Termination

A program may be terminated:

1. "abnormally", by execution of a <stop-statement>, or

2. "normally", by execution of an <end-statement> or <return-statement>, in circumstances which lead to the EPILOGUE operation being performed in the original ⟨block-state⟩. Since the <end-statement> and <return-statement> can also be used for other purposes, their execution will be described in Section 6.4.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.1.1 Execute-stop-statement

Operation: EXECUTE-STOP-STATEMENT(ss)

Where: ss is a <stop-statement>.

Step 1. Perform RAISE-CONDITION(<finish-condition>).

Step 2. Perform STOP-PROGRAM.

6.2.1.2 Stop-program

Operation: STOP-PROGRAM

Step 1. For each ⋖block-state⋗ contained in the ⋖program-state⋗ except the current ⋖block-state⋗, replace the ⋖statement-control⋗ by

⋖statement-control⋗:
⋖operation-list⋗:
⋖operation⋗ for EPILOGUE.

Step 2. Perform EPILOGUE.

6.3 Block Activation and Termination

Block activation is described by defining first those actions which are different for <procedure>s and <begin-block>s. The operations PROLOGUE and EPILOGUE are the same for both kinds of block.

6.3.1 Activate-procedure

A <procedure> may be activated by execution of a <call-statement>, or by evaluation of a <value-reference> which is a <procedure-function-reference>. Also an <on-unit> has a <procedure> which may be activated on the occurrence of an interrupt.

This operation completes when EPILOGUE (see Section 6.3.4) is executed and eliminates the ⋖block-state⋗ and its contained operations.

Operation: ACTIVATE-PROCEDURE(eer,cbifs)

Where: eer is an ⋖evaluated-entry-reference⋗,
cbifs is a [⋖condition-bif-value-list⋗].

Step 1. Let epd be the <entry-point-designator> of eer, designating an <entry-point>,ep, which is a simple component of a <procedure>,p.

Step 2. If there exists in the ⋖block-state-list⋗ a ⋖block-state⋗ whose corresponding block is p, then p MUST simply contain <recursive>, unless p is the immediate component of an <on-unit>.

Step 3. Let eud be a designator of the first <executable-unit> in the <executable-unit-list> of p. If eer contains a ⋖block-state-designator⋗,bsd, let ble be ⋖block-environment⋗: bsd. Otherwise ble is ⋖absent⋗. If bsd is present, it MUST designate an existing ⋖block-state⋗.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4. Let bs be a

⩽block-state⩾:
⩽block-directory⩾:
⩽automatic-directory⩾
⩽defined-directory⩾
⩽parameter-directory⩾;
⩽block-control⩾:
<executable-unit-designator>:
eud;
⩽group-control⩾
⩽statement-control⩾:
⩽operation-list⩾:
⩽operation⩾ for INSTALL-ARGUMENTS(eer);;;
⩽linkage-part⩾:
epd.

If ble is a ⩽block-environment⩾ then attach ble to bs. If cbifs is a ⩽condition-bif-value-list⩾ then attach cbifs to bs.

Step 5. Append bs to the ⩽block-state-list⩾.

6.3.1.1 Install-arguments

Operation: INSTALL-ARGUMENTS(eer)

Where: eer is an ⩽evaluated-entry-reference⩾.

Step 1. If the <parameter-name-list>,pnl of the <entry-point> designated by the <entry-point-designator> of eer exists, then perform Step 1.1.

Step 1.1. Let eal be the ⩽established-argument-list⩾ of eer. Attach to the current ⩽parameter-directory⩾ a ⩽parameter-directory-entry-list⩾,pdel, with the same number of immediate components as pnl and whose *i*th immediate component is

⩽parameter-directory-entry⩾:
<identifier> of the *i*th immediate component of pnl
⩽established-argument⩾, the *i*th immediate component of eal.

Step 2. For each <identifier>,id immediately contained in a <declaration> containing <parameter> in the current block, and not contained in pnl, append to pdel a

⩽parameter-directory-entry⩾:
id
⩽undefined⩾.

Step 3. Perform PROLOGUE.

6.3.2 Activate-begin-block

A <begin-block> is activated when the operation EXECUTE-EXECUTABLE-UNIT is applied to the <executable-unit> immediately containing it.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: ACTIVATE-BEGIN-BLOCK

Step 1. Let eud be a designator of the first <executable-unit> of the

```
<executable-unit>:
   <begin-block>:
      <executable-unit-list>;;
```

designated by the current <executable-unit-designator>. Let bsd be a ⟪block-state-designator⟫ designating the current ⟪block-state⟫. Append to the ⟪block-state-list⟫ a

```
⟪block-state⟫:
   ⟪block-directory⟫:
      ⟪automatic-directory⟫
      ⟪defined-directory⟫;
   ⟪block-control⟫:
      <executable-unit-designator>: eud;
      ⟪group-control⟫
      ⟪statement-control⟫:
         ⟪operation-list⟫:
            ⟪operation⟫ for PROLOGUE;;;
   ⟪linkage-part⟫
   ⟪block-environment⟫:
      bsd.
```

6.3.3 Prologue

This operation is invoked at the beginning of every block activation to establish the <automatic> and <defined> variables local to that block. The <automatic> variables are initialized if their <declaration>s specify initialization. Any <expression>s evaluated during the prologue, such as in <extent-expression>s or <expression>s in <initial>, are not allowed to reference other <automatic> or <defined> variables local to this block. The operation FIND-DIRECTORY-ENTRY will impose the restriction when it finds a reference to a variable declared in a block for which there exists a ⟪prologue-flag⟫. The ⟪prologue-flag⟫ is only present while the PROLOGUE operation is active.

Operation: PROLOGUE

Step 1. Attach a ⟪prologue-flag⟫ to the current ⟪linkage-part⟫.

Step 2. For each <declaration>,d, of the current block, that contains <automatic> or <defined>, perform Step 2.1.

Step 2.1. Let id be the <identifier> immediately contained in d, and let dd be the <data-description> immediately contained in the <variable> of d. Perform EVALUATE-DATA-DESCRIPTION-FOR-ALLOCATION(dd) to obtain an ⟪evaluated-data-description⟫,edd.

Case 2.1.1. d contains <automatic>.

Step 2.1.1.1. Perform ALLOCATE(edd) to obtain a ⟪generation⟫,g.

Step 2.1.1.2. Append to the current ⟪automatic-directory-entry-list⟫ an ⟪automatic-directory-entry⟫: id g.

Step 2.1.1.3. If d contains <initial> then perform INITIALIZE-GENERATION(g,d).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Case 2.1.2. d contains <defined>.

Append to the current ⩽defined-directory-entry-list⩾ a ⩽defined-directory-entry⩾: id edd.

Step 3. Delete the ⩽prologue-flag⩾ of the current ⩽linkage-part⩾.

Step 4. Replace the current ⩽statement-control⩾ by a

⩽statement-control⩾:
⩽operation-list⩾:
⩽operation⩾ for ADVANCE-EXECUTION.

6.3.4 Epilogue

This operation is used to terminate the execution of a block and may be invoked from executing an <end-statement>, a <return-statement>, a <stop-statement>, or a <goto-statement> which causes a transfer of control out of a block. This operation, which deletes the ⩽block-state⩾, normally causes a return to ACTIVATE-PROCEDURE or ACTIVATE-BEGIN-BLOCK, in the previous ⩽block-state⩾.

Operation: EPILOGUE

Step 1. For each current ⩽parameter-directory-entry⩾,pde, which contains ⩽dummy⩾, let g be the ⩽generation⩾ in pde and perform FREE(g).

Step 2. For each current ⩽automatic-directory-entry⩾,ade, let h be the ⩽generation⩾ in ade and perform FREE(h).

Step 3. If there is a current ⩽onsource-value⩾, then attach its immediate subtree to the ⩽returned-onsource-value⩾ of the preceding ⩽block-state⩾ of the ⩽block-state-list⩾.

Step 4. Delete the current ⩽block-state⩾.

6.4 Control within a Block

The operation NORMAL-SEQUENCE sets the current <executable-unit-designator> to designate the next <executable-unit>. This is normally the last action of each execution of an <executable-unit>. The special cases of execution of a <goto-statement> and <end-statement> may set the <executable-unit-designator> independently, while the <return-statement> and <stop-statement> have no further need for it. The <executable-unit-designator> is initialized in each activation of a <procedure> or <begin-block>, and may be reset on entering a <group>, <selection> or <if-statement>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.1 Normal-sequence

Operation: NORMAL-SEQUENCE

Step 1. Let eu be the <executable-unit> designated by the current <executable-unit-designator>. Let eul be the <executable-unit-list> which contains eu, but does not contain any other <executable-unit-list> which also contains eu.

Step 2. Let eu2 be that <executable-unit> which immediately follows eu as an immediate component of eul.

Step 3. Set the current <executable-unit-designator> to designate eu2.

6.4.1.1 Advance-execution

This operation is the "driver" which initiates execution of each <executable-unit> as selected by the current <executable-unit-designator>.

Operation: ADVANCE-EXECUTION

Step 1. Perform EXECUTE-EXECUTABLE-UNIT.

Step 2. Go to Step 1.

6.4.2 Execute-executable-unit

The current <executable-unit-designator> designates the <executable-unit> to be executed. Execution of an <executable-unit> consists of performing the appropriate "execute" operation. That operation normally terminates with the current <executable-unit-designator> designating some other <executable-unit> in the <program>. Return of control to ADVANCE-EXECUTION then causes EXECUTE-EXECUTABLE-UNIT to be applied again.

Operation: EXECUTE-EXECUTABLE-UNIT

Step 1. Let f be the rightmost immediate component of the <executable-unit> designated by the current <executable-unit-designator>.

Step 2. Perform EXECUTE-XXX(f), where "xxx" is replaced by the sequence of symbols forming the name of the type of f.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.3 Execute-begin-block

Operation: EXECUTE-BEGIN-BLOCK(b)

Where: b is a <begin-block>.

Step 1. Perform ACTIVATE-BEGIN-BLOCK.

Step 2. Perform NORMAL-SEQUENCE.

6.4.4 Execute-group

Constraint: In a <do-spec>,dsp, let tr be the <variable-reference> component. With respect to the <spec> in dsp, let

e be the <expression> immediately contained in the <spec>,
b be the <expression> in the <by-option>,
te be the <expression> in the <to-option>,
r be the <expression> in the <repeat-option>,

if such options are present. The following constraints MUST hold:

1. If te or b is present, then tr MUST have <fixed>.
2. If tr has <fixed>, then te, e, b and r (if present) MUST have <arithmetic>.
3. If tr has <pointer> then e and r (if present) MUST also have <pointer>.

Operation: EXECUTE-GROUP(g)

Where: g is a <group>.

Step 1. Let feu be the first <executable-unit> simply contained in g.

Case 1.1. g has a <conditional-group> without a <while-option>, a <non-iterative-group>, or a <looping-group>.

Set the current <executable-unit-designator> to designate feu.

Case 1.2. g has a <conditional-group> with a <while-option>.

Let exp be the <expression> of the <while-option> of g. Perform ESTABLISH-TRUTH-VALUE(exp) to obtain t. If t is ≪true≫, then set the current <executable-unit-designator> to designate feu; otherwise, perform NORMAL-SEQUENCE.

Case 1.3. g has a <controlled-group>.

Let dsp be the <do-spec> of g. Perform ESTABLISH-CONTROLLED-GROUP(dsp) to obtain t. If t is ≪true≫, then set the current <executable-unit-designator> to designate feu; otherwise, perform NORMAL-SEQUENCE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.4.1 Establish-controlled-group

This operation is used to set up an iteration in the cases of a <controlled-group>, <list-directed-input>, <list-directed-output>, <edit-directed-input>, and <edit-directed-output>.

If the controlling <do-spec> is such as to indicate iteration, then an appropriate ⋖controlled-group-state⋗ is established and ⋖true⋗ is returned. If the controlling <do-spec> indicates no iteration, then no ⋖controlled-group-state⋗ is established and ⋖false⋗ is returned.

Operation: ESTABLISH-CONTROLLED-GROUP(dsp)

Where: dsp is a <do-spec>.

Result: ⋖true⋗ or ⋖false⋗.

Step 1. Let tr be the <variable-reference> of dsp, and dt be the <data-type> of tr. Perform EVALUATE-TARGET-REFERENCE(tr) to obtain an ⋖evaluated-target⋗,et.

Step 2. Let sp be the <spec> of dsp. Append to the current ⋖controlled-group-state-list⋗, the tree

⋖controlled-group-state⋗:
<spec-designator>: a designator designating sp;
⋖cv-target⋗: et;
⋖cv-type⋗: dt.

Step 3. Perform INITIALIZE-SPEC-OPTIONS to obtain tv. If tv is ⋖true⋗, then return ⋖true⋗; otherwise delete the current ⋖controlled-group-state⋗ and return ⋖false⋗.

6.4.4.2 Initialize-spec-options

This operations attempts to establish the first ⋖basic-value⋗ generated by a <spec>. It returns ⋖true⋗ if such a ⋖basic-value⋗ exists, and ⋖false⋗ otherwise. It also establishes the ⋖converted-to-type⋗ and ⋖converted-by-type⋗ needed by ADVANCE-CONTROLLED-GROUP.

Operation: INITIALIZE-SPEC-OPTIONS

Result: ⋖true⋗ or ⋖false⋗.

Step 1. Let sp be the <spec> designated by the current <spec-designator>. Let e be the <expression> immediate component of sp.

Step 2. Perform Steps 2.1 through 2.3 in any order.

Step 2.1. Perform EVALUATE-EXPRESSION(e) to obtain an ⋖aggregate-value⋗,av.

Step 2.2. If sp contains a <to-option>,t: <expression>,et; then perform Steps 2.2.1 through 2.2.3.

Step 2.2.1. Let etdt be the <data-type> of et. Let dt be a

<data-type>:
<computational-type>:
<arithmetic>:
the derived common <base> of etdt and the current ⋖cv-type⋗
the derived common <scale> of etdt and the current ⋖cv-type⋗
the converted <precision> of etdt.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.2.2. Perform EVALUATE-EXPRESSION(et) to obtain an ⟪aggregate-value⟫,x. Let y be the ⟪basic-value⟫ in x. Perform CONVERT(dt,etdt,y) to obtain a ⟪basic-value⟫: ⟪real-value⟫,z. Attach a ⟪to-value⟫: z; to the current ⟪controlled-group-state⟫.

Step 2.2.3. Let cp be the converted <precision> of the current ⟪cv-type⟫, where dt is used as the target <data-type> for determining cp. Replace the <precision> tree in dt by cp. Attach a ⟪converted-to-type⟫: dt; to the current ⟪controlled-group-state⟫. (The ⟪converted-to-type⟫ will be used later as the target <data-type> when the value of the ⟪cv-target⟫ is converted for comparison with the ⟪to-value⟫.)

Step 2.3. If sp contains a <by-option> or a <to-option> then perform Steps 2.3.1 through 2.3.5.

Step 2.3.1.

Case 2.3.1.1. sp contains a <by-option>: <expression>,eb.

Perform EVALUATE-EXPRESSION(eb) to obtain an ⟪aggregate-value⟫,x. Let ebv be the ⟪basic-value⟫ in x. Let ebdt be the <data-type>of eb.

Case 2.3.1.2. sp contains a <to-option> but not a <by-option>.

Let ebv be a ⟪basic-value⟫: ⟪real-value⟫: 1. Let ebdt be a <data-type> which is integer-type, except that its <base> has <decimal> and its <number-of-digits> has 1.

Step 2.3.2. Let dt be a

<data-type>:
<computational-type>:
<arithmetic>:
derived common <base> of ebdt and current ⟪cv-type⟫
derived common <scale> of ebdt and current ⟪cv-type⟫
converted <precision> of ebdt.

Perform CONVERT(dt,ebdt,ebv) to obtain a ⟪basic-value⟫,x. Let y be the ⟪real-value⟫ immediately contained in x. Attach a ⟪by-value⟫: y; to the current ⟪controlled-group-state⟫.

Step 2.3.3. Let cp be the converted <precision> of the current ⟪cv-type⟫, where dt is used as the target <data-type> for determining cp. Let p be the <number-of-digits> of cp, and let r be the <number-of-digits> of dt.

Step 2.3.4.

Case 2.3.4.1. dt has <float>.

Change the value of the <number-of-digits> of dt to max(p,r).

Case 2.3.4.2. dt has <fixed>.

Let q be the <scale-factor> of cp, and let s be the <scale-factor> of dt. Let m=min(N,max(p−q,r−s)+max(q,s)+1), where N is the maximum <number-of-digits> allowed for <fixed> with the <base> of dt. Let n=max(q,s). If dt has <binary> then n MUST be 0. Change the value of the <number-of-digits> in dt to m, and change the value of the <scale-factor> in dt to n.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.3.5. Attach a ⟨converted-by-type⟩: dt; to the current ⟨controlled-group-state⟩. (The ⟨converted-by-type⟩ will be used later as the result <data-type> for the addition of the ⟨by-value⟩ and the value of the ⟨cv-target⟩.)

Step 3. Let cvt be the immediate component of the current ⟨cv-target⟩. Let dd be the <data-description> immediate component of e. Perform ASSIGN(cvt,av,dd).

Step 4. Perform TEST-SPEC to obtain tv. Return tv.

6.4.4.3 Test-spec

This operation is used to test whether the <spec> in the controlling <do-spec> indicates continuation (⟨true⟩ returned) or termination (⟨false⟩ returned).

Operation: TEST-SPEC

Result: ⟨true⟩ or ⟨false⟩.

Step 1. If the immediate component of the current ⟨controlled-group-state⟩ contains a ⟨to-value⟩,y, perform Steps 1.1 through 1.3.

Step 1.1. Let cvt be the immediate component of the current ⟨cv-target⟩. Perform VALUE-OF-EVALUATED-TARGET(cvt) to obtain an ⟨aggregate-value⟩ containing a ⟨basic-value⟩,x. Let xt be the current ⟨cv-type⟩.

Step 1.2. Let dt be the current ⟨converted-to-type⟩. Perform CONVERT(dt,xt,x) to obtain a ⟨basic-value⟩,cx.

Step 1.3. Let bv be the current ⟨by-value⟩. If bv $\geq$ 0 and cx > y, or if bv < 0 and cx < y, return ⟨false⟩.

Step 2. If the <spec> designated by the current <spec-designator> contains a <while-option>: <expression>,e; then perform Step 2.1.

Step 2.1. Perform ESTABLISH-TRUTH-VALUE(e) to obtain tv. Return tv.

Step 3. Return ⟨true⟩.

6.4.4.4 Advance-controlled-group

This operation is used to advance an iteration set up by ESTABLISH-CONTROLLED-GROUP. If the controlling <do-spec> indicates that another iteration should be performed, then an appropriate ⟨basic-value⟩ is generated, the appropriate <executable-unit-designator> or ⟨data-item-indicator⟩ is set to continue, and ⟨true⟩ is returned. Otherwise, ⟨false⟩ is returned and the iteration is terminated.

Operation: ADVANCE-CONTROLLED-GROUP

Result: ⟨true⟩ or ⟨false⟩.

Step 1. Let sp be the <spec> designated by the current <spec-designator>. Let evt be the ⟨evaluated-target⟩ of the current ⟨cv-target⟩.

Step 2. If sp contains an <until-option>: <expression>,e; then perform Step 2.1 .

Step 2.1. Perform ESTABLISH-TRUTH-VALUE(e) to obtain tv. If tv is ⟨true⟩, return ⟨false⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 3.

Case 3.1. sp contains a <repeat-option>: <expression>,re.

Let dd be the <data-description> immediate component of re. Perform EVALUATE-EXPRESSION(re) to obtain an ⋖aggregate-value⋗,av. Perform ASSIGN(evt,av,dd).

Case 3.2. sp contains a <by-option> or a <to-option>.

Step 3.2.1. Let bt be the <data-type> in the current ⋖converted-by-type⋗. Let cvt be the <data-type> in the current ⋖cv-type⋗. Let cvt2 be a <data-type> that is the same as bt except for its <precision>, which is the converted <precision> of cvt, with bt being the target <data-type> for determining the converted <precision>.

Step 3.2.2. Let et be the ⋖evaluated-target⋗ in the current ⋖cv-target⋗. Perform VALUE-OF-EVALUATED-TARGET(et) to obtain an ⋖aggregate-value⋗,x1. Let x2 be the ⋖basic-value⋗ in x1. Perform CONVERT(cvt2,cvt,x2) to obtain a ⋖basic-value⋗,x3. Let x be the ⋖real-value⋗ in x3.

Step 3.2.3. Let y be the ⋖real-value⋗ in the current ⋖by-value⋗. Perform ARITHMETIC-RESULT(x+y,bt) to obtain a ⋖real-value⋗,z.

Step 3.2.4. Let rtd be a

<data-description>:
  <item-data-description>:
    <data-type>:
      bt.

Let w be an

⋖aggregate-value⋗:
  ⋖aggregate-type⋗:
    ⋖scalar⋗;
  ⋖basic-value-list⋗:
    ⋖basic-value⋗:
      z.

Perform ASSIGN(et,w,rtd).

Case 3.3. sp does not contain a <repeat-option>, a <to-option>, or a <by-option>.

Go to Step 5.

Step 4. Perform TEST-SPEC to obtain tv. If tv is ⋖true⋗, return ⋖true⋗.

Step 5. Delete the current ⋖controlled-group-state⋗. Return ⋖false⋗.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.5 Execute-selection

Operation: EXECUTE-SELECTION(s)

Where: s is a <selection>.

Step 1. If s immediately contains an <expression>,te, then perform Step 1.1 .

Step 1.1. Let tdt be the <data-type> in the <data-description> of te. Perform EVALUATE-EXPRESSION(te) to obtain an ⟪aggregate-value⟫,av. Let tv be the ⟪basic-value⟫ in av.

Step 2. For each <when-unit>,wu in s, if any, taken in order, perform Steps 2.1 through 2.4.

Step 2.1. Attach a <when-unit-designator>, which designates wu, to the current ⟪block-control⟫.

Step 2.2.

Case 2.2.1. s immediately contains an <expression>.

Perform TEST-WHEN-MATCHING(tv,tdt,wu) to obtain wm.

Case 2.2.2. s does not immediately contain an <expression>.

Perform TEST-WHEN-ENABLEMENT(wu) to obtain wm.

Step 2.3. Delete the current <when-unit-designator>.

Step 2.4. If wm is ⟪true⟫ then set the current <executable-unit-designator> to designate the <executable-unit> in wu and terminate this operation.

Step 3.

Case 3.1. s contains an <otherwise-unit>,oth.

Set the current <executable-unit-designator> to designate the <executable-unit> in oth.

Case 3.2. s does not contain an <otherwise-unit>.

Perform RAISE-CONDITION(<error-condition>).

6.4.5.1 Test-when-enablement

Operation: TEST-WHEN-ENABLEMENT(wu)

Where: wu is a <when-unit>.

Result: a ⟪true⟫ or ⟪false⟫.

Step 1. For each <expression>,e in the <when-any-values> or <when-all-values> immediately contained in wu, taken in order, perform Steps 1.1 through 1.2.

Step 1.1. Perform ESTABLISH-TRUTH-VALUE(e) to obtain tv.

Step 1.2.

Case 1.2.1. tv is ⟪true⟫ and wu immediately contains a <when-any-values>.

Return ⟪true⟫.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Case 1.2.2. tv is ≪false≫ and wu immediately contains a <when-all-values>.

Return ≪false≫.

Case 1.2.3. (Otherwise).

No action.

Step 2.

Case 2.1. wu immediately contains a <when-any-values>.

Return ≪false≫.

Case 2.2. wu immediately contains a <when-all-values>.

Return ≪true≫.

6.4.5.2 Test-when-matching

Operation: TEST-WHEN-MATCHING(tv,tdt,wu)

Where: tv is a ≪basic-value≫,
tdt is a <data-type>,
wu is a <when-unit>.

Result: a ≪true≫ or ≪false≫.

Step 1. For each <expression>,we in the <when-any-values> or <when-all-values> immediately contained in wu, taken in order, perform Steps 1.1 through 1.2.

Step 1.1. Let wdt be the <data-type> in the <data-description> of we. Perform EVALUATE-EXPRESSION(we) to obtain an ≪aggregate-value≫,av. Let wv be the ≪basic-value≫ in av.

Step 1.2. Perform COMPARE(tv,wv,tdt,wdt) to obtain a ≪comparison-result≫,cr.

Step 1.3.

Case 1.3.1. cr contains ≪equal≫ and wu immediately contains a <when-any-values>.

Return ≪true≫.

Case 1.3.2. cr contains ≪not-equal≫ and wu immediately contains a <when-all-values>.

Return ≪false≫.

Case 1.3.3. (Otherwise).

No action.

Step 2.

Case 2.1. wu immediately contains a <when-any-values>.

Return ≪false≫.

Case 2.2. wu immediately contains a <when-all-values>.

Return ≪true≫.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.6 Execute-if-statement

Operation: EXECUTE-IF-STATEMENT(ifs)

Where: ifs is an <if-statement>.

Step 1. Let e be the <expression> immediate component of the <test> of ifs. Perform ESTABLISH-TRUTH-VALUE(e) to obtain tv.

Step 2.

Case 2.1. tv is ⟪true⟫.

Replace the immediate component of the current <executable-unit-designator> by a designator of the <executable-unit> of the <then-unit> of ifs.

Case 2.2. tv is ⟪false⟫ and ifs simply contains an <else-unit>,eu.

Replace the immediate component of the current <executable-unit-designator> by a designator of the <executable-unit> of eu.

Case 2.3. tv is ⟪false⟫ and ifs does not simply contain an <else-unit>.

Perform NORMAL-SEQUENCE.

6.4.6.1 Establish-truth-value

Operation: ESTABLISH-TRUTH-VALUE(exp)

Where: exp is an <expression>.

Result: ⟪true⟫ or ⟪false⟫.

Step 1. Perform EVALUATE-EXPRESSION(exp) to obtain an ⟪aggregate-value⟫,av: ⟪basic-value-list⟫: ⟪basic-value⟫,sv. Let sdt be the <data-type> of exp. sdt has <bit>.

Step 2. sv MUST contain exactly one ⟪bit-value⟫,bv.

Step 3. If bv contains a ⟪one-bit⟫; then return ⟪true⟫; otherwise return ⟪false⟫.

6.4.7 Execute-call-statement

Operation: EXECUTE-CALL-STATEMENT(cs)

Where: cs is a <call-statement>.

Step 1. Let sr be the <subroutine-reference> component of cs. Perform EVALUATE-ENTRY-REFERENCE(sr) to obtain an ⟪evaluated-entry-reference⟫,eer.

Step 2. Perform ACTIVATE-PROCEDURE(eer).

Step 3. Perform NORMAL-SEQUENCE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.4.7.1 Entry-references

An *entry-reference* is either a <subroutine-reference> or a <procedure-function-reference>. (<builtin-function-reference>s are described in Section 9.5.)

The main difference between a <subroutine-reference> and a <procedure-function-reference> is that normal termination of a <procedure> in the <subroutine-reference> case is by a <return-statement> not containing an <expression>, or by an <end-statement>, whereas in the <procedure-function-reference> case it is by a <return-statement> containing an <expression>.

Evaluation of an entry-reference normally takes place just before activation of a <procedure>.

6.4.7.1.1 Evaluate-entry-reference

Operation: EVALUATE-ENTRY-REFERENCE(er)

Where: er is a <subroutine-reference> or a <procedure-function-reference>.

Result: an ⟪evaluated-entry-reference⟫.

Step 1. Let vr be the <value-reference> immediate component of er. Perform EVALUATE-VALUE-REFERENCE(vr) to obtain an ⟪aggregate-value⟫,ag. Let ev be the ⟪entry-value⟫ in ag. ev MUST NOT contain ⟪null-entry⟫.

Step 2. Let ep be the <entry-point> designated by the <entry-point-designator> in ev. Let id be the <identifier> in the <statement-name> in ep. Let p be the <procedure> that block-contains ep. Let b be the abstract-block that block-contains p. Let dl be the <declaration-list> immediately contained in b. Let d be the <declaration> in dl whose <identifier> equals id. Let de be a copy of the <entry> immediately contained in the <named-constant> of d, and let vre be a copy of the <entry> of the <data-description> of vr.

Delete from de and vre all <variable-reference>s which are immediate components of <offset>. In an IMPLEMENTATION-DEFINED fashion, compare corresponding <options> components of de and vre and, if either de or vre has <options> components and the other does not have corresponding <options> components, compare de and vre. Delete all <options> components of de and vre.

For each <extent-expression>,ee in de or vre containing a <constant>,c, let e be an <expression>: c; and perform EVALUATE-EXPRESSION-TO-INTEGER(e) to obtain an ⟪integer-value⟫,iv and replace ee by <extent-expression>: iv. de and vre MUST now be equal. (This checks that the entry point to be invoked agrees with the declaration of the entry value reference, vr.)

Step 3.

Case 3.1. er does not contain an <argument-list>.

Return an ⟪evaluated-entry-reference⟫: ev.

Case 3.2. er contains an <argument-list>,al.

Let n be the number of <argument>s in al. Let eal be an ⟪established-argument-list⟫ with n ⟪established-argument⟫ immediate components. For i=1,...,n, taken in any order, perform Step 3.2.1.

Step 3.2.1. Let arg be the *i*th <argument> in al. Let dd be the <data-description> in the *i*th <parameter-descriptor> in the <parameter-descriptor-list> in the <data-type> of vr.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Perform ESTABLISH-ARGUMENT(arg,dd) to obtain an ⟪established-argument⟫,x.
Replace the *i*th ⟪established-argument⟫ of eal by x.

Step 4. Return an ⟪evaluated-entry-reference⟫: ev eal.

6.4.7.1.2 Establish-argument

Operation: ESTABLISH-ARGUMENT(arg,dd)

Where: arg is an <argument>,
dd is a <data-description>.

Result: an ⟪established-argument⟫.

Case 1. arg does not immediately contain <dummy>.

Let vr be the <variable-reference> simple component of arg. Perform EVALUATE-VARIABLE-REFERENCE(vr) to obtain a ⟪generation⟫,g. Perform TEST-CONNECTEDNESS(g) to obtain carg, which MUST be ⟪true⟫. Return an ⟪established-argument⟫: g ⟪not-dummy⟫.

Case 2. arg immediately contains <dummy>.

Note: The ⟪aggregate-type⟫ of the <expression> in arg immediately contains ⟪scalar⟫ (see Case 3.4.1 of CREATE-ENTRY-REFERENCE).

Step 2.1. Let e be the <expression> in arg. Perform EVALUATE-EXPRESSION(e) to obtain an ⟪aggregate-value⟫,av.

Step 2.2. Let avdd be the <data-description> immediate component of e. Perform PROMOTE-AND-CONVERT(dd,av,avdd) to obtain an ⟪aggregate-value⟫,av2.

Step 2.3. Let cdd be a copy of dd. If cdd contains an <area-size>,x, let y be an ⟪integer-value⟫ determined by an IMPLEMENTATION-DEFINED algorithm, and replace x by an

```
<area-size>:
   <extent-expression>:
      y.
```

If cdd contains a <maximum-length>,ml:<asterisk>; replace ml by

```
<maximum-length>:
   <extent-expression>:
      ⟪integer-value⟫:
         m;;;
```

where m is the length of the ⟪character-string-value⟫ or ⟪bit-string-value⟫ in av2.

Step 2.4. Perform EVALUATE-DATA-DESCRIPTION-FOR-ALLOCATION(cdd) to obtain an ⟪evaluated-data-description⟫,edd. Perform ALLOCATE(edd) to obtain a ⟪generation⟫,g.

Step 2.5. Let bvl be the ⟪basic-value-list⟫ in av2. Perform SET-STORAGE(g,bvl).

Step 2.6. Return an ⟪established-argument⟫: g ⟪dummy⟫.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.8 Execute-goto-statement

Operation: EXECUTE-GOTO-STATEMENT(gs)

Where: gs is a <goto-statement>.

Step 1. Let vr be the <value-reference> of gs. Perform EVALUATE-VALUE-REFERENCE(vr) to obtain an ⋖aggregate-value⋗,ag. ag MUST contain the form

⋖label-value⋗,lv:
<executable-unit-designator>,tp
⋖block-state-designator⋗,bsn.

Step 2. The ⋖block-state-list⋗ MUST contain a ⋖block-state⋗,bs, designated by bsn. (Its corresponding block contains the <executable-unit> designated by tp.)

Case 2.1. bs is the current ⋖block-state⋗.

Perform LOCAL-GOTO(lv).

Case 2.2. bs is not the current ⋖block-state⋗.

Step 2.2.1. The ⋖statement-control⋗ component of bs MUST NOT contain an ⋖operation⋗ for EXECUTE-ALLOCATE-STATEMENT or PROLOGUE.

Step 2.2.2. Replace the ⋖statement-control⋗ component of bs by:

⋖statement-control⋗:
⋖operation-list⋗:
⋖operation⋗ for ADVANCE-EXECUTION
⋖operation⋗ for TRIM-IO-CONTROL
⋖operation⋗ for LOCAL-GOTO(lv).

Step 2.2.3. For each ⋖block-state⋗,b that occurs after bs and before the current ⋖block-state⋗ in the ⋖block-state-list⋗ (of the ⋖interpretation-state⋗) perform Steps 2.2.3.1 and 2.2.3.2.

Step 2.2.3.1. The ⋖statement-control⋗ component of b MUST NOT contain an ⋖operation⋗ for EXECUTE-ALLOCATE-STATEMENT or PROLOGUE.

Step 2.2.3.2. Replace the ⋖statement-control⋗ of b by

⋖statement-control⋗:
⋖operation-list⋗:
⋖operation⋗ for EPILOGUE.

Step 2.2.4. Replace the current ⋖statement-control⋗ by

⋖statement-control⋗:
⋖operation-list⋗:
⋖operation⋗ for EPILOGUE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.8.1 Local-goto

Operation: LOCAL-GOTO(lv)

Where: lv is a ⋖label-value⋗.

Step 1. Let tp be the <executable-unit-designator> in lv. Let eu be the <executable-unit> designated by tp. If there is an <iterative-group>,g, <when-unit>,g, or <otherwise-unit>,g that contains eu but does not contain an <iterative-group>, <when-unit>, <otherwise-unit> or <begin-block> that contains eu, then the current <executable-unit-designator> MUST designate an <executable-unit> that is contained in g.

Step 2. Perform TRIM-GROUP-CONTROL(tp).

Step 3. Replace the current <executable-unit-designator> by tp.

6.4.8.2 Trim-group-control

Operation: TRIM-GROUP-CONTROL(eud)

Where: eud is an <executable-unit-designator>.

Step 1. Let eu be the <executable-unit> designated by eud. Let b be the <begin-block> or <procedure> that block-contains eu.

Step 2.

Case 2.1. b contains a <controlled-group> that contains eu.

Let n be the number of <controlled-group>s that contain eu and are contained in b. If the current ⋖controlled-group-state-list⋗ contains more than n ⋖controlled-group-state⋗s, delete those after the *n*th ⋖controlled-group-state⋗.

Case 2.2. (Otherwise).

If there is a current ⋖controlled-group-state-list⋗, delete it.

Step 3. If there is a current <when-unit-designator>, delete it.

6.4.9 Execute-null-statement

Operation: EXECUTE-NULL-STATEMENT(ns)

Where: ns is a <null-statement>.

Step 1. Perform NORMAL-SEQUENCE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.4.10 Execute-return-statement

Operation: EXECUTE-RETURN-STATEMENT(rs)

Where: rs is a <return-statement>.

Step 1. Let p be the last ⋖block-state⋗ of the ⋖block-state-list⋗ whose corresponding block is a <procedure>.

Case 1.1. rs does not contain an <expression>.

There MUST NOT be a <returns-descriptor> in the <entry-point> designated by the <entry-point-designator> of p. If p is the first ⋖block-state⋗ in the ⋖block-state-list⋗ then perform RAISE-CONDITION(<finish-condition>).

Case 1.2. rs contains an <expression>,e.

There MUST be a <returns-descriptor>,rd in the <entry-point> designated by the <entry-point-designator> of p. Let rdd be the <data-description> immediately contained in rd. Let dd be the <data-description> immediate component of e. dd MUST be proper for assignment to rdd (see Section 4.6.2.2).

Step 1.2.1. Perform EVALUATE-EXPRESSION(e) to obtain ev.

Step 1.2.2. Perform PROMOTE-AND-CONVERT(rdd,ev,dd) to obtain an ⋖aggregate-value⋗,av. Attach ⋖returned-value⋗: av; to the ⋖linkage-part⋗ of the ⋖block-state⋗ immediately preceding p in the ⋖block-state-list⋗.

Step 2. For each ⋖block-state⋗ (if any) which is or which follows p in the ⋖block-state-list⋗ except the current ⋖block-state⋗, replace its ⋖statement-control⋗ by a

⋖statement-control⋗:
⋖operation-list⋗:
⋖operation⋗ for EPILOGUE.

Step 3. Perform EPILOGUE.

6.4.11 Execute-end-statement

Operation: EXECUTE-END-STATEMENT(es)

Where: es is an <end-statement>.

Step 1. Let eul be that <executable-unit-list> which contains es but does not contain any other <executable-unit-list> which also contains es.

Let n be the node which immediately contains eul.

Case 1.1. n is a <procedure>.

Step 1.1.1. The <entry-point> designated by the <entry-point-designator> component of the current ⋖linkage-part⋗ MUST NOT contain a <returns-descriptor>.

Step 1.1.2. If the ⋖block-state-list⋗ contains only one ⋖block-state⋗ then perform RAISE-CONDITION(<finish-condition>).

Step 1.1.3. Perform EPILOGUE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1.2. n is a <begin-block>.

Perform EPILOGUE.

Case 1.3. n is a <non-iterative-group>.

Step 1.3.1. Set the current <executable-unit-designator> to designate the <executable-unit> which simply contains n.

Step 1.3.2. Perform NORMAL-SEQUENCE.

Case 1.4. n is a <conditional-group>, without an <until-option>, or a <looping-group>.

Set the current <executable-unit-designator> to designate the <executable-unit> which simply contains n.

Case 1.5. n is a <conditional-group> with an <until-option>.

Let exp be the <expression> of the <until-option> of n. Perform ESTABLISH-TRUTH-VALUE(exp) to obtain tv. Set the current <executable-unit-designator> to designate the <executable-unit> which simply contains n.

Case 1.5.1. tv is ⩽true⩾.

Perform NORMAL-SEQUENCE.

Case 1.5.2. tv is ⩽false⩾.

No action.

Case 1.6. n is a <controlled-group>.

Set the current <executable-unit-designator> to designate the <executable-unit> that simply contains n. Perform ADVANCE-CONTROLLED-GROUP to obtain t.

Case 1.6.1. t is ⩽true⩾.

Set the current <executable-unit-designator> to designate the first <executable-unit> of eul.

Case 1.6.2. t is ⩽false⩾.

Perform NORMAL-SEQUENCE.

6.4.12 Execute-leave-statement

This operation determines the outermost <group> to be terminated by a LEAVE statement. The ⩽controlled-group-state⩾s for any <controlled-group>s being terminated are removed. The current <executable-unit-designator> is then set to cause execution of the statement immediately after the outermost <group> that is being terminated.

Operation: EXECUTE-LEAVE-STATEMENT(ls)

Where: ls is a <leave-statement>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.

Case 1.1. ls has a <named-constant-reference>,ncr.

Step 1.1.1. Perform EVALUATE-NAMED-CONSTANT-REFERENCE(ncr) to obtain an ⩽aggregate-value⩾,ag. ag will contain a

⩽label-value⩾,lv:
<executable-unit-designator>,tp
⩽block-state-designator⩾,bsn.

Note: bsn will designate the current ⩽block-state⩾ because of the constraint in Section 4.7.11, VALIDATE-LEAVE-STATEMENTS. lv is a label on the outermost <group> being terminated.

Case 1.2. ls does not have a <named-constant-reference>.

Step 1.2.1. Let g be the <group> which simply contains ls.

Step 1.2.2. Let tp be an <executable-unit-designator> which designates the <executable-unit> which immediately contains g.

Step 2. Perform TRIM-GROUP-CONTROL(tp).

Step 3. Set the current <executable-unit-designator> to tp.

Step 4. Perform NORMAL-SEQUENCE.

6.5 Conditions and Interrupts

There are two distinct concepts of "condition" and "interrupt". When a "condition" occurs, e.g. RAISE-CONDITION(<overflow-condition>) is performed, it may lead to an "interrupt", i.e. invocation of the INTERRUPT operation.

The circumstances in which the various "conditions" occur are defined throughout Chapters 6 to 9 at the appropriate points, wherever the operation RAISE-CONDITION is to be performed. This section defines how the occurrence of a "condition" may also be signalled explicitly, and how the operations RAISE-CONDITION, INTERRUPT, and SYSTEM-ACTION are performed.

6.5.1 Conditions

6.5.1.1 Raise-condition

A condition may be "raised" either implicitly from circumstances defined elsewhere, or explicitly by the execution of a <signal-statement>. In either case, the operation TEST-ENABLEMENT is used to determine whether the "condition" is enabled and hence to determine whether the operation INTERRUPT is to be performed.

Operation: RAISE-CONDITION(c,cbifs)

Where: c is an ⩽evaluated-io-condition⩾, a <programmer-named-condition>, or a terminal node of <condition-name> apart from the <named-io-condition>s or <programmer-named-condition>s,
cbifs is a [⩽condition-bif-value-list⩾].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 1. There MUST exist at least one ⟨block-state⟩.

Step 2. If c is one of the terminal nodes of a <computational-condition>, then perform TEST-ENABLEMENT(c) to obtain r, which MUST NOT be ⟨disabled⟩.

Step 3. Let cc be a copy of cbifs. If c is a <conversion-condition> and the current ⟨block-control⟩ contains a ⟨current-file-value⟩: ⟨file-value⟩,fv;, and cc does not contain an ⟨onfile-value⟩, then attach to cc an ⟨onfile-value⟩: fn; where fn is the ⟨character-string-value⟩ in the ⟨filename⟩ in fi, where fi is the ⟨file-information⟩ designated by fv.

Step 4. Perform INTERRUPT(c,cc).

6.5.1.2 Test-enablement

Operation: TEST-ENABLEMENT(c)

Where: c is one of the terminal nodes of <computational-condition>.

Result: ⟨enabled⟩ or ⟨disabled⟩.

Step 1. Let eu be the <executable-unit> designated by the current <executable-unit-designator>.

Case 1.1. The current ⟨linkage-part⟩ does not contain a ⟨prologue-flag⟩ and the current ⟨block-control⟩ does not contain a ⟨remote-block-state⟩.

Case 1.1.1. The current ⟨block-control⟩ contains a <when-unit-designator>,wud.

Let tp be the <when-unit> designated by wud.

Case 1.1.2. (Otherwise).

Let tp be eu.

Case 1.2. The current ⟨linkage-part⟩ does not contain a ⟨prologue-flag⟩ and the current ⟨block-control⟩ contains a ⟨remote-block-state⟩,rbs.

Let fc be the last ⟨format-control⟩ of the current ⟨format-control-list⟩ that contains a ⟨remote-block-state⟩. (This ⟨remote-block-state⟩ equals rbs.) Let tp be the <format-statement> designated by the <format-statement-designator> of fc.

Case 1.3. The current ⟨linkage-part⟩ contains a ⟨prologue-flag⟩.

Let tp be the <procedure> or <begin-block> of which eu is a block-component.

Step 2. If tp immediately contains a <condition-prefix-list>,cpl, and if cpl contains a <condition-prefix>,cp, containing a <computational-condition> equal to <computational-condition>: c;, then return the second component of cp.

Step 3.

Case 3.1. There exists a <begin-block>,b, <procedure>,b, or <package>,b which has tp as a block-component.

Let tp be b and go to Step 2.

Case 3.2. (Otherwise).

If c is <size-condition>, then return ⟨disabled⟩; otherwise return ⟨enabled⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.5.1.3 Execute-signal-statement

Operation: EXECUTE-SIGNAL-STATEMENT(ss)

Where: ss is a <signal-statement>.

Step 1. Let cn be the <condition-name> immediate component of ss. Let cns be the immediate subnode of cn. If cns is a <named-io-condition> or <programmer-named-condition>, then let c be cns; otherwise, let c be the terminal subnode of cns. If c is not one of the terminal nodes of <computational-condition>, then let r be ⩽enabled⩾. Otherwise perform TEST-ENABLEMENT(c) to obtain r.

Case 1.1. r is ⩽disabled⩾.

Perform NORMAL-SEQUENCE and terminate this operation.

Case 1.2. r is ⩽enabled⩾.

Case 1.2.1. c is <conversion-condition>.

Let cbifs be a

⩽condition-bif-value-list⩾:
⩽condition-bif-value⩾:
⩽onsource-value⩾:
⩽character-string-value⩾:
⩽null-character-string⩾.

Case 1.2.2. c contains <key-condition>.

Let cbifs be a

⩽condition-bif-value-list⩾:
⩽condition-bif-value⩾:
⩽onkey-value⩾:
⩽character-string-value⩾:
⩽null-character-string⩾.

Case 1.2.3. (Otherwise).

Let cbifs be ⩽absent⩾.

Step 2. If c is a <named-io-condition> then perform EVALUATE-NAMED-IO-CONDITION(c) to obtain ec; otherwise let ec be c.

Step 3. Perform INTERRUPT(ec,cbifs).

Step 4. Perform NORMAL-SEQUENCE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.5.1.4 Evaluate-named-io-condition

Operation: EVALUATE-NAMED-IO-CONDITION(nioc)

Where: nioc is a <named-io-condition>.

Result: an ⩽evaluated-io-condition⩾.

Step 1. Let vr be the <value-reference> immediate component of nioc. Perform EVALUATE-FILE-OPTION(vr) to obtain a ⩽file-value⩾,f.

Step 2. Let ioc be the <io-condition> component of nioc. Return ⩽evaluated-io-condition⩾: ioc f.

6.5.2 Interrupts

The <on-statement> and <revert-statement> may be used to influence the action taken on the occurrence of an interrupt operation. First these statements are described, and then the operation INTERRUPT itself is defined.

6.5.2.1 Execute-on-statement

Operation: EXECUTE-ON-STATEMENT(os)

Where: os is an <on-statement>.

Step 1. For each <condition-name>,cn in the <condition-name-list> component of os taken in left-to-right order perform Steps 1.1 through 1.5 .

Step 1.1.

Case 1.1.1. cn has <named-io-condition>,nic.

Perform EVALUATE-NAMED-IO-CONDITION(nic) to obtain an ⩽evaluated-io-condition⩾,eic. Let ec be an ⩽evaluated-condition⩾: eic.

Case 1.1.2. cn does not have <named-io-condition>.

Let cn1 be the immediate subtree of cn. Let ec be an ⩽evaluated-condition⩾: cn1.

Step 1.2. If the current ⩽block-state⩾ contains an ⩽established-on-unit-list⩾,eoul then if eoul contains an ⩽established-on-unit⩾,eou containing ec, then delete eou.

Step 1.3.

Case 1.3.1. os contains an <on-unit>,ou.

Let epd be an <entry-point-designator> designating the <entry-point> of ou. Let es be an

⩽entry-value⩾:
epd
⩽block-state-designator⩾ designating the current ⩽block-state⩾.

Case 1.3.2. (Otherwise).

Let es be <system-action>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.4. Let neou be

⟨established-on-unit⟩:
ec
es.

Step 1.5. If os contains <snap>, attach <snap> to neou. Append neou to the current ⟨established-on-unit-list⟩.

Step 2. Perform NORMAL-SEQUENCE.

6.5.2.2 Execute-revert-statement

Operation: EXECUTE-REVERT-STATEMENT(rs)

Where: rs is a <revert-statement>.

Step 1. Let cnl be the <condition-name-list> immediate component of rs. For each <condition-name>,c in cnl taken in left-to-right order perform Steps 1.1 and 1.2 .

Step 1.1.

Case 1.1.1. c has <named-io-condition>,nic.

Perform EVALUATE-NAMED-IO-CONDITION(nic) to obtain an ⟨evaluated-io-condition⟩,eioc. Let ec be an ⟨evaluated-condition⟩: eioc.

Case 1.1.2. c does not have a <named-io-condition>.

Let ec be an ⟨evaluated-condition⟩: the immediate component of c.

Step 1.2. If the current ⟨block-state⟩ contains an ⟨established-on-unit-list⟩,eoul then if eoul contains an ⟨established-on-unit⟩, eou containing ec, then delete eou.

Step 2. Perform NORMAL-SEQUENCE.

6.5.3 Interrupt

Operation: INTERRUPT(c,cbifs)

Where: c is an ⟨evaluated-io-condition⟩, a <programmer-named-condition>, or a terminal node of <condition-name> apart from the <named-io-condition>s or <programmer-named-condition>s,
cbifs is a [⟨condition-bif-value-list⟩].

Step 1. Let bs be the current ⟨block-state⟩.

Step 2. Append to cbifs each of the following components whose immediate subnode it does not already possess (or let cbifs be a ⟨condition-bif-value-list⟩ with these components if cbifs is ⟨absent⟩):

⟨condition-bif-value⟩:
⟨oncode-value⟩:
⟨integer-value⟩: an IMPLEMENTATION-DEFINED integer;;;

and if c is an ⟨evaluated-io-condition⟩ and cbifs does not contain an ⟨onfile-value⟩:

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

≤condition-bif-value≥:
≤onfile-value≥:
the ≤character-string-value≥ in the ≤filename≥ of the ≤file-information≥ designated by the ≤file-value≥ of c.

Step 3.

Case 3.1. bs contains an ≤established-on-unit-list≥,eoul and eoul contains an ≤established-on-unit≥,eou, which contains an ≤evaluated-condition≥ with a subtree equal to c, or c is a <programmer-named-condition> such that the following conditions are true:

1. c designates a <declaration>,ad containing <external>, and

2. eoul contains an ≤established-on-unit≥,eou which contains a <programmer-named-condition> designating a <declaration> equal to ad.

Step 3.1.1. If eou contains <snap>, then output IMPLEMENTATION-DEFINED information (e.g. a list of names of currently active blocks) by an implementation-dependent means.

Step 3.1.2.

Case 3.1.2.1. eou contains an ≤entry-value≥,ev.

Let eer be an ≤evaluated-entry-reference≥: ev. Perform ACTIVATE-PROCEDURE(eer,cbifs).

Case 3.1.2.2. eou contains <system-action>.

Perform SYSTEM-ACTION(c,cbifs).

Case 3.2. (Otherwise).

Case 3.2.1. bs is not the first ≤block-state≥ of the ≤block-state-list≥,bsl.

Let bs be the immediately preceding ≤block-state≥ of bsl and go to Step 3 .

Case 3.2.2. (Otherwise).

Perform SYSTEM-ACTION(c,cbifs).

Step 4. c MUST NOT be <error-condition>, <fixedoverflow-condition>, <overflow-condition>, <size-condition>, or <zerodivide-condition>.

6.5.4 System-action

For every ≤evaluated-condition≥, there is the possibility, as defined in Section 6.5.3, that an interrupt operation for it will lead to the operation SYSTEM-ACTION.

Operation: SYSTEM-ACTION(c,cbifs)

Where: c is an ≤evaluated-io-condition≥, a <programmer-named-condition>, or a terminal node of a <condition-name> apart from the <named-io-condition>s or <programmer-named-condition>s,
cbifs is a ≤condition-bif-value-list≥.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1. c is <beginpage-condition> or <finish-condition>.

Terminate this operation.

Case 2. c is <programmer-named-condition> or <underflow-condition>.

Perform COMMENT.

Case 3. c contains <endpage-condition>.

Let fv be the ≤file-value≥ component of c. Perform PUT-PAGE(fv) (see Section 8.8.2.11).

Case 4. c is <error-condition> or <storage-condition>.

The action is IMPLEMENTATION-DEFINED.

Case 5. (Otherwise).

Step 5.1. Perform COMMENT.

Step 5.2. Perform RAISE-CONDITION(<error-condition>,cbifs).

6.5.4.1 Comment

Operation: COMMENT

Output IMPLEMENTATION-DEFINED information by an implementation-dependent means.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7. Storage and Assignment

7.1 Introduction

This chapter defines all the operations of the PL/I Machine that change the ⟨allocated-storage⟩ of the ⟨machine-state⟩. The main Sections are:

7.2 The Generation
7.3 The Allocation of Storage
7.4 Initialization
7.5 The Freeing of Storage
7.6 Assignment
7.7 Variable-reference
7.8 Reference to Named Constant

The ⟨allocated-storage⟩ consists of an ⟨allocation-unit-list⟩. Elements are appended to this list by the ALLOCATE operation; this may be invoked either during the execution of an <allocate-statement> or <read-statement>, or directly by any of the operations ESTABLISH-ARGUMENT, PROLOGUE or ALLOCATE-STATIC-STORAGE-AND-BUILD-STATIC-DIRECTORY. An ⟨allocation-unit⟩ is deleted from the list by the FREE operation; this may be invoked either during the execution of a <free-statement> or <read-statement>, or directly by the EPILOGUE operation.

An ⟨area-value⟩ contains a set of ⟨allocation-unit⟩s as subnodes of its ⟨area-action-list⟩. ⟨allocation-unit⟩s are added to this list as subnodes of ⟨area-allocation⟩s by the ALLOCATE operation during execution of an <allocate-statement>. They are deleted by the FREE-BASED-STORAGE operation during execution of a <free-statement>.

Elements of the ⟨basic-value-list⟩ of an ⟨allocation-unit⟩ are changed by the SET-STORAGE operation; this may be invoked during the execution of the <assignment-statement>, the <group>, the <read-statement>, the <get-statement>, or during initialization.

If the <data-description> used to allocate an ⟨allocation-unit⟩ contains any <union-data-description>s, then the ⟨allocation-unit⟩ will contain a ⟨union-list⟩. Each ⟨union⟩ corresponds to a single union in storage, so that for an array of unions there will be a ⟨union⟩ for each element of the array.

Each ⟨union⟩, in turn, contains a list of ⟨union-alternative⟩s, one for each member of the union. Each ⟨union-alternative⟩ contains a list of ⟨basic-value-index⟩s that specify those ⟨basic-value⟩s that belong to that ⟨union-alternative⟩. Whenever an assignment is made to a ⟨basic-value⟩ that belongs to a ⟨union-alternative⟩, the ⟨basic-value⟩s designated by all other ⟨union-alternative⟩s of the containing ⟨union⟩ are set to ⟨undefined⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2 The Generation

The evaluation of a <variable-reference> yields a ⟪generation⟫; a ⟪pointer-value⟫ is also a ⟪generation⟫. A ⟪generation⟫ describes some or all of the elements of the ⟪basic-value-list⟫ of the ⟪allocation-unit⟫ designated by the ⟪allocation-unit-designator⟫ of the ⟪generation⟫. The elements of the ⟪storage-index-list⟫ component of the ⟪generation⟫ specify which elements of the ⟪basic-value-list⟫ of the ⟪allocation-unit⟫ are being described. Each such element is a *scalar-element.*

The ⟪evaluated-data-description⟫ component of a ⟪generation⟫ contains a <data-description> where each <extent-expression> contains only an ⟪integer-value⟫.

7.2.1 The Number of Elements in the Storage-index-list of a Generation

Let the ⟪evaluated-data-description⟫ of the ⟪generation⟫,g, have the immediate <data-description> component dd. Because dd is contained in an ⟪evaluated-data-description⟫, each <bound-pair> of dd has two <extent-expression>s which contain only ⟪integer-value⟫s; that is, no <bound-pair> contains an <asterisk> or <expression> with a <variable-reference>. The number of elements of the ⟪storage-index-list⟫ of g is the number of scalar-elements corresponding to dd. This number is determined by the operation SCALAR-ELEMENTS-OF-DATA-DESCRIPTION.

Operation: SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd)

Where: dd is a <data-description>.

Result: an integer.

Case 1. dd is of the form <data-description>: <item-data-description>.

Return the integer 1.

Case 2. dd is of the form <data-description>: <structure-data-description>: <member-description-list>,mdd, or dd is of the form <data-description>: <union-data-description>: <member-description-list>,mdd.

For each tree of the form <member-description>: <data-description>,dd1[i]; in mdd, i=1,...,m, perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd1[i]) to obtain an integer, n[i]. Return the integer

$$\sum_{i=1}^{m} n[i].$$

Case 3. dd is of the form

<data-description>:
  <dimensioned-data-description>:
    <element-data-description>,edd
    <bound-pair-list>,bpl.

Let ic be the immediate component of edd. Let dd1 be a <data-description>: ic. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd1) to obtain the integer n. For each tree of the form <bound-pair>,bp[i] in bpl, i=1,...,m, let ub[i] and lb[i] be the ⟪integer-value⟫ components of the <upper-bound> and <lower-bound> respectively of bp[i]. Return the integer

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$n * \prod_{i=1}^{m} (ub[i] - lb[i] + 1).$$

7.2.2 Correspondence between an Item-data-description and a Basic-value

There is a correspondence between an element of the ⟪storage-index-list⟫ of a ⟪generation⟫ and a ⟪basic-value⟫ in the ⟪allocated-storage⟫. There is also a correspondence between a ⟪storage-index⟫ and an <item-data-description> of the ⟪evaluated-data-description⟫ of the ⟪generation⟫. In general, this is a many-to-one correspondence defined by the operation FIND-ITEM-DATA-DESCRIPTION which finds the <item-data-description> of a <data-description> that corresponds to a given element of the ⟪storage-index-list⟫.

Operation: FIND-ITEM-DATA-DESCRIPTION(dd,ord)

Where: dd is a <data-description>,
ord is the ordinal of an element of a ⟪storage-index-list⟫.

Result: an <item-data-description>.

Case 1. The immediate component of dd is an <item-data-description>.

Return this <item-data-description>.

Case 2. The immediate component of dd is a <structure-data-description> or <union-data-description>.

Let sdd be this <structure-data-description> or <union-data-description>. Let mdl be the <member-description-list> of sdd. Let n[i] be the number of scalar-elements corresponding to the *i*th element of mdl, obtained by performing SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd[i]) where dd[i] is the <data-description> of the *i*th element of mdl. Let sm[0] be 0. Let sm[i] be the sum of the n[j] for the first i elements of mdl. Let j be such that $sm[j-1] \leq ord \leq sm[j]$. Perform FIND-ITEM-DATA-DESCRIPTION(dd[j],k), where $k = ord - sm[j-1]$, to obtain an <item-data-description>,idd. Return idd.

Case 3. The immediate component of dd is a <dimensioned-data-description>.

Let ddd be this <dimensioned-data-description>.

Case 3.1. The <element-data-description> of ddd has an <item-data-description> as the immediate component.

Return a copy of this <item-data-description>.

Case 3.2. (Otherwise).

Let dd1 be a <data-description> immediately containing a copy of the <structure-data-description> or <union-data-description> of ddd. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd1) to obtain an integer n. Perform FIND-ITEM-DATA-DESCRIPTION(dd1,k+1), where k is the value of the remainder obtained when ord−1 is divided by n, to obtain an <item-data-description>,idd. Return idd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2.3 Value of a Generation

Operation: VALUE-OF-GENERATION(g)

Where: g is a ⟨generation⟩.

Result: an ⟨aggregate-value⟩.

Step 1. The ⟨allocation-unit⟩ designated by the ⟨allocation-unit-designator⟩ of g MUST be contained in ⟨allocated-storage⟩.

Step 2. Let av be an ⟨aggregate-value⟩.

Step 3. Let dd be the <data-description> of g, with associated ⟨aggregate-type⟩,agt. Attach agt to av.

Step 4. Let n be the number of elements in the ⟨storage-index-list⟩ of g, and let v be the ⟨basic-value-list⟩ of the ⟨allocation-unit⟩ designated by the ⟨allocation-unit-designator⟩ of g.

Step 5. For i=1,...,n, perform Steps 5.1 through 5.3.

Step 5.1. Perform FIND-ITEM-DATA-DESCRIPTION(dd,i) to obtain an <item-data-description>,id. Let d be the <data-type> of id.

Step 5.2. Let p be the *i*th ⟨storage-index⟩ immediately contained in g.

Step 5.3. Perform VALUE-OF-STORAGE-INDEX(p,d,v) to obtain a ⟨basic-value⟩,bv. Append bv to the ⟨basic-value-list⟩ of av.

Step 6. Return av.

7.2.4 Value of Storage Index

Operation: VALUE-OF-STORAGE-INDEX(p,d,v)

Where: p is a ⟨storage-index⟩,
d is a <data-type>,
v is a ⟨basic-value-list⟩.

Result: a ⟨basic-value⟩.

Step 1. Let i be the ⟨basic-value-index⟩ of p.

Step 2. Let v[i] be the immediate component of the *i*th ⟨basic-value⟩ of v.

Case 2.1. d does not have <string>: <nonvarying>; or <pictured>.

Let e be a copy of v[i].

Case 2.2. d has <string>: <nonvarying>; or <pictured>.

Let N be the <maximum-length> component of d (if d has <string>) or the associated character-string length of d (if d has <pictured>).

Case 2.2.1. N = 0.

Let e be the ⟨null-character-string⟩ (if d has <character> or <pictured>) or the ⟨null-bit-string⟩ (if d has <bit>).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.2.2. $N \neq 0$.

Let M be the ⟨position-index⟩ component of p. Then there are integers j and k such that:

1. $i \leq j \leq k$ (i=j=k is possible),

2. v[i], v[i+1], ..., v[k] are either all ⟨bit-string-value⟩s or all ⟨character-string-value⟩s,

3. $\sum_{s=i}^{j-1} L[s] < M \leq \sum_{s=i}^{j} L[s]$, and

4. $M + N - 1 \leq \sum_{s=i}^{k} L[s]$.

Here L[x] is the length of v[x]. Let e be a ⟨character-string-value⟩ (if v[i] is a ⟨character-string-value⟩) or a ⟨bit-string-value⟩ (if v[i] is a ⟨bit-string-value⟩). The length of e is N. The N components of e are the same N successive components of v[j], v[j+1], ..., v[k] beginning at component number (M−L[i]−L[i+1]−...−L[j−1]) in v[j].

Step 3. Return a ⟨basic-value⟩ containing a copy of e.

7.3 The Allocation of Storage

7.3.1 Execute-allocate-statement

This operation causes the construction of an ⟨allocation-unit⟩ corresponding to the <declaration> designated by the <declaration-designator>. Under certain circumstances, the <storage-condition> or the <area-condition> may be raised.

Operation: EXECUTE-ALLOCATE-STATEMENT(ast)

Where: ast is an <allocate-statement>.

Step 1. For each <allocation>,al, in the <allocation-list> of ast, chosen in left-to-right order, perform Steps 1.1 and 1.2 .

Step 1.1. Let d be the <declaration> designated by the <declaration-designator> of al.

Step 1.2. Perform ALLOCATE-BASED-STORAGE(al) to obtain the ⟨generation⟩,g.

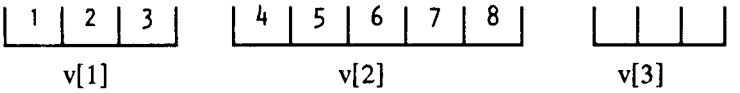
Step 2. Perform NORMAL-SEQUENCE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

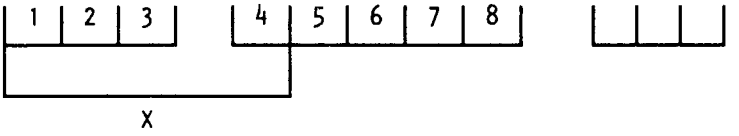
```
DECLARE 1 A AUTOMATIC UNALIGNED,
          2 B CHARACTER(3),
          2 C CHARACTER(5),
          2 D CHARACTER(3),
        1 S DEFINED (A) UNALIGNED,
          2 X CHARACTER(4),
          2 Y CHARACTER(7);

A.B = '123';
A.C = '45678';
```

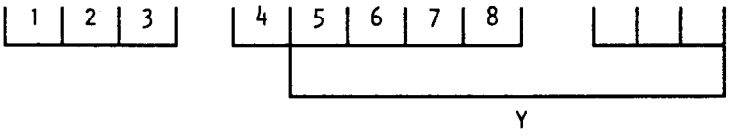
After the assignments, the ≪generation≫ corresponding to A looks like



Here v[i] is the *i*th component of the ≪basic-value-list≫, a digit represents the corresponding character, and an empty slot represents ≪undefined≫. Evaluation of a <variable-reference> X yields a ≪scalar≫ ≪generation≫ accessing the ≪generation≫ of A as follows.



Hence a <value-reference> X yields '1234'. Evaluation of a <variable-reference> Y yields a ≪scalar≫ ≪generation≫ accessing the ≪generation≫ of A as follows.



The value of the ≪generation≫ corresponding to Y has ≪undefined≫ as its fifth, sixth, and seventh components. Therefore, evaluation of a <value-reference> Y at this point would be in error.

Figure 13. An Example of ≪generation≫s and ≪basic-value≫s of Defined Variables.

7.3.2 Allocate-based-storage

This operation causes the allocation of storage for a <based> variable and the assignment of the resulting ≪generation≫ to a <locator> variable.

Operation: ALLOCATE-BASED-STORAGE(al)

Where: al is an <allocation>.

Result: a ≪generation≫.

Step 1. Let d be the <declaration> designated by the <declaration-designator> of al.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.

Case 2.1. al has no <set-option> component.

The <based> component of d MUST have the component <value-reference> that immediately contains a <variable-reference>. Let vrs be this <variable-reference>.

Case 2.2. al has a <set-option> component.

Let vrs be the <variable-reference> of the <set-option> of al.

Step 3. Perform EVALUATE-VARIABLE-REFERENCE(vrs) to obtain a ⟨generation⟩,gs.

Step 4. Perform EVALUATE-IN-OPTION(al,vrs). If the allocation is to be made in an area, a ⟨generation⟩ with an <area> component will be obtained; let this be gi; otherwise the value ⟨fail⟩ will be obtained.

Step 5. Make a copy, dd, of the <data-description> immediately contained in the <variable> of d. Perform EVALUATE-DATA-DESCRIPTION-FOR-ALLOCATION(dd) to obtain an ⟨evaluated-data-description⟩,edd.

Step 6.

Case 6.1. A ⟨generation⟩,gi, was obtained in Step 4.

Make an allocation for dd in the area ⟨generation⟩,gi, by performing ALLOCATE(edd,gi). The result obtained will be either the ⟨generation⟩,g, or the value ⟨fail⟩. In the latter case, perform RAISE-CONDITION(<area-condition>); on normal return go to Step 4.

Case 6.2. ⟨fail⟩ was obtained in Step 4.

Make an allocation for dd in the ⟨allocated-storage⟩ by performing ALLOCATE(edd) to obtain the ⟨generation⟩,g.

Step 7. Perform Steps 7.1 and 7.2 in either order.

Step 7.1. Let av be an

⟨aggregate-value⟩:
⟨aggregate-type⟩:
⟨scalar⟩;
⟨basic-value-list⟩:
⟨basic-value⟩:
⟨pointer-value⟩:
g.

Let ddp be of the form <data-description>: <item-data-description>: <pointer>. Let egs be ⟨evaluated-target⟩: gs. Perform ASSIGN(egs,av,ddp).

Step 7.2. Perform INITIALIZE-REFER-OPTIONS(g) to carry out the initializations of g specified by any. <refer-option> in the ⟨evaluated-data-description⟩ of g.

Step 8. Return g.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.3.3 Evaluate-in-option

If an allocation is to be made in an area, this operation yields the area ⟨generation⟩ in which the allocation is to be made.

Operation: EVALUATE-IN-OPTION(al,vr)

Where: al is an <allocation>,
vr is a <variable-reference>.

Result: a ⟨generation⟩ or ⟨fail⟩.

Step 1. Let ds be the <declaration> designated by the <declaration-designator> component of vr. ds is the <declaration> for the locator that will be used to identify the ⟨allocation-unit⟩ that will result from the <allocation>,al.

Step 2. If the component <locator> of ds has <offset> and this has the subnode <variable-reference>, then let this <variable-reference> be vro.

Step 3.

Case 3.1. al has an <in-option> component.

Let the <variable-reference> of the <in-option> be vri.

Case 3.2. al has no <in-option> component.

Case 3.2.1. ds has an <offset> component.

vro MUST have been created in Step 2. That is, the area base for the offset MUST have been specified. Let vri be the same as vro.

Case 3.2.2. ds has a <pointer> component.

Return the value ⟨fail⟩, since the ⟨allocation-unit⟩ for al is to be constructed in the ⟨allocated-storage⟩.

Step 4. Perform EVALUATE-VARIABLE-REFERENCE(vri) to obtain the ⟨generation⟩,gia.

Step 5. If vro has been created in Step 2, then perform EVALUATE-VARIABLE-REFERENCE(vro) to obtain the ⟨generation⟩,gib. gia MUST be the same as gib.

Step 6. Return gia.

7.3.4 Allocate

This operation either adds an ⟨allocation-unit⟩ to ⟨allocated-storage⟩ or adds an ⟨area-allocation⟩ to the area ⟨generation⟩. The newly allocated storage is described by an ⟨evaluated-data-description⟩. The result of the operation is the ⟨generation⟩ that identifies the newly allocated storage.

Operation: ALLOCATE(edd,g)

Where: edd is an ⟨evaluated-data-description⟩,
g is a [⟨generation⟩].

Result: a ⟨generation⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1. If g is a ⟨generation⟩, perform VALUE-OF-GENERATION(g) to obtain an ⟨aggregate-value⟩,avg. Let bv be the ⟨basic-value⟩ in avg. bv MUST NOT simply contain ⟨undefined⟩. Let av be the ⟨area-value⟩ in bv.

Step 2.

Case 2.1. g is ⟨absent⟩.

Optionally go to Step 3. Otherwise perform RAISE-CONDITION(⟨storage-condition⟩) and go to Step 2.

Note: Normally, the STORAGE condition is raised by an implementation only to indicate lack of ability to locate sufficient space to perform the allocation.

Case 2.2. g is a ⟨generation⟩.

Optionally return ⟨fail⟩.

Step 3. Perform MAKE-ALLOCATION-UNIT(edd) to obtain an ⟨allocation-unit⟩,au.

Step 4.

Case 4.1. g is ⟨absent⟩.

Append au to ⟨allocated-storage⟩.

Case 4.2. g is a ⟨generation⟩.

Let aa be an ⟨area-action⟩: ⟨area-allocation⟩: edd. Attach au to aa. Append aa to the ⟨area-action-list⟩ in av.

Step 5. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd), to obtain n, where dd is the <data-description> of edd. Let i be 1. Construct a ⟨storage-index-list⟩,sil, by performing Steps 5.1 through 5.3 n times.

Step 5.1. Perform FIND-ITEM-DATA-DESCRIPTION(dd,i) to obtain an <item-data-description>,idd.

Step 5.2.

Case 5.2.1. idd contains either a <pictured> component or a <string> component that has <nonvarying>.

Construct a ⟨storage-index⟩,si, that contains a ⟨basic-value-index⟩ that has an ⟨integer-value⟩ equal to i and also a ⟨position-index⟩ that has an ⟨integer-value⟩ equal to 1.

Case 5.2.2. (Otherwise).

Construct a ⟨storage-index⟩,si, that contains a ⟨basic-value-index⟩ that has an ⟨integer-value⟩ equal to i.

Step 5.3. Append si to sil and set i to i+1.

Step 6. Return the ⟨generation⟩ constructed from an ⟨allocation-unit-designator⟩ designating au, edd, and sil.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.3.5 Evaluate-data-description-for-allocation

This operation forms an ⟨evaluated-data-description⟩ from a <data-description> by evaluating each <extent-expression>. When the <data-description> is a <parameter-descriptor> or <returns-descriptor> it may contain <asterisk>s instead of <extent-expression>s. These <asterisk>s are not replaced by ⟨integer-value⟩s in the ⟨evaluated-data-description⟩ that is returned.

Operation: EVALUATE-DATA-DESCRIPTION-FOR-ALLOCATION(dd)

Where: dd is a <data-description>.

Result: an ⟨evaluated-data-description⟩.

Step 1. Let cdd be a copy of the <data-description>,dd. For each <extent-expression> in cdd that immediately contains an <expression>, perform Step 1.1, with the <extent-expression>s chosen in any order.

Step 1.1. Let the chosen <extent-expression> be ee. Evaluate ee by performing EVALUATE-EXPRESSION-TO-INTEGER(e) where e is the <expression> of ee, to obtain the value i. The <expression> of the <extent-expression> is replaced by an ⟨integer-value⟩ with value i.

Step 2. In each <bound-pair> component of cdd which does not contain <asterisk>, the value of the ⟨integer-value⟩ of the <upper-bound> MUST be greater than or equal to the value of the ⟨integer-value⟩ of the <lower-bound>.

Step 3. In each <maximum-length> component of cdd which does not contain <asterisk>, the value of the ⟨integer-value⟩ MUST be greater than or equal to zero.

Step 4. In each <area-size> component of cdd which does not contain <asterisk>, the value of the ⟨integer-value⟩ MUST be greater than or equal to zero.

Step 5. If cdd contains a <refer-option>,ro, then perform Step 5.1

Step 5.1. Let mo be the <member-description> in cdd that ro references as in Step 2.1 of VALIDATE-BASED-DECLARATION. Let n be an ordinal such that the result of FIND-ITEM-DATA-DESCRIPTION(cdd,n) is equal to the <item-data-description> in mo. Replace the <identifier-list> component of ro by an ⟨integer-value⟩ equal to n.

Step 6. Delete all <identifier-list>s which are immediate components of any <structure-data-description> or <union-data-description> in cdd.

Step 7. Return the ⟨evaluated-data-description⟩: cdd.

7.3.6 Find-directory-entry

This operation searches the appropriate ⟨machine-state⟩ directory for an entry corresponding to a declaration.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: FIND-DIRECTORY-ENTRY(dp)

Where: dp is a <declaration-designator> designating a <declaration>,d.

Result:

a ⩽static-directory-entry⩾, or
an ⩽automatic-directory-entry⩾, or
a ⩽parameter-directory-entry⩾, or
a ⩽defined-directory-entry⩾.

Step 1. Let st be the <storage-type> of d.

Step 2.

Case 2.1. st contains <automatic>, <parameter>, or <defined>.

Perform FIND-BLOCK-STATE-OF-DECLARATION(dp) to obtain a ⩽block-state⩾,bs. If st contains <automatic> or <defined> then bs MUST NOT contain a ⩽prologue-flag⩾ in its ⩽linkage-part⩾. Find the directory entry, e, in the ⩽automatic-directory⩾, ⩽parameter-directory⩾, or ⩽defined-directory⩾ of bs, as appropriate, such that the <identifier> of e is equal to the <identifier> of d.

Case 2.2. st contains <static>.

Case 2.2.1. The <scope> of d has an <external> component.

Search the ⩽static-directory⩾ for the directory entry e whose <identifier> is equal to the <identifier> of d and that has an <external> component.

Case 2.2.2. The scope of d has an <internal> component.

Let e be the ⩽static-directory-entry⩾ which contains a <declaration-designator> equal to dp.

Step 3. Return e.

7.3.7 Make-allocation-unit

This operation forms an ⩽allocation-unit⩾ corresponding to an ⩽evaluated-data-description⩾.

Operation: MAKE-ALLOCATION-UNIT(edd)

Where: edd is an ⩽evaluated-data-description⩾.

Result: an ⩽allocation-unit⩾.

Step 1. Let dd be the simply contained <data-description> component of edd. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd) to obtain n, the number of scalar-elements that correspond to edd. Construct a ⩽basic-value-list⩾,svl, by performing Steps 1.1 through 1.2 for i=1,...,n.

Step 1.1. Perform FIND-ITEM-DATA-DESCRIPTION(dd,i) to obtain the <item-data-description>,idd, that corresponds to the *i*th scalar-element of edd.

Step 1.2.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1.2.1. idd contains <pictured>.

Let m be the associated character-string length of the <pictured> component of idd. Append to svl a ⟨character-value-list⟩ consisting of m elements, each of which is ⟨undefined⟩.

Case 1.2.2. idd contains <string> containing <nonvarying>.

Let m be the ⟨integer-value⟩ contained by the <maximum-length> component of <string>.

Case 1.2.2.1. The <string> of idd contains <character>.

If m is zero, append a ⟨character-string-value⟩ consisting of the ⟨null-character-string⟩ to svl; otherwise append a ⟨character-string-value⟩ consisting of a ⟨character-value-list⟩ with m elements containing ⟨undefined⟩.

Case 1.2.2.2. The <string> of idd contains <bit>.

If m is zero, append a ⟨bit-string-value⟩ consisting of the ⟨null-bit-string⟩ to svl; otherwise append a ⟨bit-string-value⟩ consisting of a ⟨bit-value-list⟩ with m elements containing ⟨undefined⟩.

Case 1.2.3. idd contains an <area> component, and idd is not simply contained in a <union-data-description>.

Let as be the ⟨integer-value⟩ in the <area-size> of idd. Append to svl an

⟨area-value⟩:
⟨area-action-list⟩:
⟨area-action⟩:
⟨area-size⟩:
as.

Case 1.2.4. (Otherwise).

Append an ⟨undefined⟩ element to svl.

Step 2. Perform CONSTRUCT-UNION-LIST(edd,1) to obtain a ⟨union-list⟩,ual or ⟨absent⟩,ual. Let edd1 be a copy of edd.

Step 3. Return an ⟨allocation-unit⟩: svl [ual] edd1.

7.3.8 Initialize-refer-options

This operation initializes the object of each <refer-option> in a ⟨generation⟩.

Operation: INITIALIZE-REFER-OPTIONS(g)

Where: g is a ⟨generation⟩ whose ⟨aggregate-type⟩ immediately contains ⟨structure-aggregate-type⟩.

Step 1. Let edd be the ⟨evaluated-data-description⟩ of g and let dd be the <data-description> in edd. If edd has a <refer-option>,ro then perform Steps 1.1 and 1.2.

Step 1.1. The <refer-option>,ro, has an ⟨integer-value⟩ index, i, constructed by EVALUATE-DATA-DESCRIPTION-FOR-ALLOCATION; this identifies the element of g that is the object of the <refer-option>,ro. Let ord be an ordinal equal to i. Perform FIND-ITEM-DATA-DESCRIPTION(dd,ord) to obtain an <item-data-description>,idd, of the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

element. Let eddr be an ⟨evaluated-data-description⟩ with idd as immediate component. Construct the ⟨generation⟩,gr, from eddr, a copy of the ⟨allocation-unit-designator⟩ of g and the ⟨storage-index-list⟩ containing a single element, a copy of the *i*th element of the ⟨storage-index-list⟩ of g.

Step 1.2. ro is a component of an <extent-expression> that has an ⟨integer-value⟩,iv. Let ddi be a <data-description> with integer-type (see Section 9.2.2). Let av be of the form

⟨aggregate-value⟩:
⟨aggregate-type⟩:
⟨scalar⟩;
⟨basic-value-list⟩:
⟨basic-value⟩,iv.

Let et be ⟨evaluated-target⟩: gr. Perform ASSIGN(et,av,ddi) to assign the value of the <extent-expression> to the object of the <refer-option>.

7.3.9 Find-block-state-of-declaration

Operation: FIND-BLOCK-STATE-OF-DECLARATION(dp)

Where: dp is a <declaration-designator> designating the <declaration>,d.

Result: a ⟨block-state⟩.

Step 1. Let bs be the current ⟨block-state⟩. If the ⟨block-control⟩ of bs contains the form ⟨remote-block-state⟩: ⟨block-state-designator⟩,bsd, then let bs be the ⟨block-state⟩ designated by bsd.

Step 2. Let bb be the corresponding block of bs (see Section 5.3.2). If bb contains d, then return bs.

Step 3. bs MUST have a ⟨block-environment⟩: ⟨block-state-designator⟩,bsd. Let bs be the ⟨block-state⟩ designated by bsd. Go to Step 2.

7.3.10 Construct-union-list

CONSTRUCT-UNION-LIST constructs a ⟨union-list⟩ from an ⟨evaluated-data-description⟩. The ⟨union-list⟩ has a ⟨union⟩ for each <union-data-description> in the given ⟨evaluated-data-description⟩. Each ⟨union⟩, in turn, has a ⟨union-alternative⟩ for each member of the <union-data-description>.

Operation: CONSTRUCT-UNION-LIST(edd,i)

Where: edd is an ⟨evaluated-data-description⟩ or a <data-description>,
i is an integer (the ordinal of the first ⟨storage-index⟩ corresponding to edd).

Result: a ⟨union-list⟩ or ⟨absent⟩.

Step 1. If edd is an ⟨evaluated-data-description⟩ then let dd be the <data-description> in edd; otherwise, let dd be edd.

Step 2.

Case 2.1. dd immediately contains an <item-data-description>.

Return ⟨absent⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.2. dd immediately contains a <union-data-description>,udd.

Step 2.2.1. Let n be the number of components of the <member-description-list> of udd, and let md[1],...,md[n] be the <member-description>s of udd. Let ual be a ⟨union-alternative-list⟩ with no subnodes. Let uasl be a ⟨union-list⟩ with no subnodes. (ual collects the immediate alternatives of udd, while uasl collects sets of alternatives for any unions contained in udd by recursively descending through udd.) Let i1 be i. Perform Steps 2.2.1.1 through 2.2.1.4 for t=1,...,n.

Step 2.2.1.1. Let dd1 be the <data-description> of md[t]. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd1) to obtain an integer, z. Let bvil be a ⟨basic-value-index-list⟩ with no components. Perform Step 2.2.1.1.1 for j=i1,...,i1+z−1.

Step 2.2.1.1.1. Append a ⟨basic-value-index⟩: ⟨integer-value⟩: j;; to bvil.

Step 2.2.1.2. Append a ⟨union-alternative⟩: bvil; to ual.

Step 2.2.1.3. Perform CONSTRUCT-UNION-LIST(dd1,i1) to obtain a ⟨union-list⟩,uasl1 or ⟨absent⟩,uasl1. If uasl1 is not ⟨absent⟩, append the immediate components of uasl1, in order, to uasl.

Step 2.2.1.4. Let i1 be i1+z.

Step 2.2.2. Append ual to uasl. If uasl has no components, return ⟨absent⟩; otherwise return uasl.

Case 2.3. dd immediately contains a <structure-data-description>,sdd.

Step 2.3.1. Let n be the number of <member-description>s of sdd, and let md[1],...,md[n] be those <member-description>s. Let i1 be 1. Let uasl be a ⟨union-list⟩ with no components. Perform Steps 2.3.1.1 through 2.3.1.2 for t=1,...,n.

Step 2.3.1.1. Let dd1 be the <data-description> of md[t]. Perform CONSTRUCT-UNION-LIST(dd1,i1) to obtain a ⟨union-list⟩,uasl1 or ⟨absent⟩,uasl1. If uasl1 is not ⟨absent⟩, append the immediate components of uasl1, in order, to uasl.

Step 2.3.1.2. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd1) to obtain an integer, z. Let i1 be i1+z.

Step 2.3.2. If uasl has no components, return ⟨absent⟩; otherwise return uasl.

Case 2.4. dd immediately contains a <dimensioned-data-description>,ddd.

Step 2.4.1. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd) to obtain an integer, nt. Let dd1 be the <data-description> of the <element-data-description> of ddd. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd1) to obtain an integer, n. If n is 0, let m be 0; otherwise let m be nt/n. (m is the number of elements in the array.) Let i1 be i. Let uasl be a ⟨union-list⟩ with no components. Perform Steps 2.4.1.1 through 2.4.1.2 for t=1,...,m.

Step 2.4.1.1. Perform CONSTRUCT-UNION-LIST(dd1,i1) to obtain a ⟨union-list⟩,uasl1 or ⟨absent⟩,uasl1. If uasl1 is not ⟨absent⟩, append the immediate components of uasl1, in order, to uasl.

Step 2.4.1.2. Let i1 be i1+n.

Step 2.4.2. If uasl has no components, return ⟨absent⟩; otherwise return uasl.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4 Initialization

7.4.1 Initialize-generation

This operation initializes a ≤generation≥ according to the specification contained in a <declaration>.

Operation: INITIALIZE-GENERATION(g,d)

Where: g is a ≤generation≥,
d is a <declaration>.

Case 1. The immediate component of the <data-description> immediately contained in the <variable> of d is an <item-data-description>,idd.

Perform INITIALIZE-SCALAR-ELEMENT(g,idd).

Case 2. The immediate component of the <data-description> immediately contained in the <variable> of d is a <dimensioned-data-description> whose <element-data-description> is an <item-data-description>.

Perform INITIALIZE-ARRAY(g,d).

Case 3. The immediate component of the <data-description>,dd immediately contained in the <variable> of d is either a <structure-data-description>, or a <dimensioned-data-description> whose <element-data-description> is a <structure-data-description>.

Let sdd be the simply contained <structure-data-description> in dd. For each <member-description> immediately contained in the <member-description-list> of sdd, chosen in any order, that has an <initial> component, not necessarily immediate, perform Steps 3.1 through 3.4.

Step 3.1. Let the chosen <member-description>,md be the *i*th immediate component of the <member-description-list>.

Step 3.2. Perform SELECT-QUALIFIED-REFERENCE(g,il,d), where il is an <identifier-list> consisting of the single <identifier>,id, that is a copy of the *i*th immediate component of the <identifier-list> of sdd, to obtain a modified ≤generation≥,gl.

Step 3.3. Let dc be a copy of the <declaration>,d, and let sddc be the copy of sdd contained in dc.

Case 3.3.1. The immediate component of dd is <dimensioned-data-description> and the <data-description> of md has a <dimensioned-data-description>,ddd.

Append to the <bound-pair-list> of dc the immediate components of the <bound-pair-list> of ddd. Replace sddc by the immediate subtree of the <element-data-description> of ddd; this will always be a <structure-data-description> or an <item-data-description>.

Case 3.3.2. (Otherwise).

Replace sddc by the immediate subtree of the <data-description> of md.

Step 3.4. Perform INITIALIZE-GENERATION(gl,dc).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4.2 Initialize-scalar-element

This operation initializes a generation consisting of a single element.

Operation: INITIALIZE-SCALAR-ELEMENT(g,idd)

Where: g is a ⋖generation⋗,
idd is an <item-data-description>.

Step 1. If idd does not contain an <initial-element> then terminate this operation.

Step 2. If the <initial-element> component of idd has an <asterisk> then terminate this operation.

Step 3. Perform EVALUATE-EXPRESSION(e), where e is the <expression> simply contained in the <initial-element> of idd, to obtain an ⋖aggregate-value⋗,v.

Step 4. Create an ⋖evaluated-target⋗,et, and attach g to it.

Step 5. Perform ASSIGN(et,v,ddl), where ddl is the <data-description> immediately contained in e.

7.4.3 Initialize-array

This operation initializes a ⋖generation⋗ that has an array of ⋖basic-value⋗s.

⋖evaluated-initial-element⋗ ::= <asterisk> | <parenthesized-expression> (scalar) |
{<iteration-factor> | ⋖evaluated-iteration-factor⋗}
⋖evaluated-mitial-element-list⋗ |
⋖evaluated-initial-item⋗ | ⋖absent⋗

⋖evaluated-iteration-factor⋗ ::= ⋖integer-value⋗

⋖evaluated-initial-item⋗ ::= ⋖basic-value⋗ <data-description>

Operation: INITIALIZE-ARRAY(g,d)

Where: g is a ⋖generation⋗,
d is a <declaration>.

Step 1. Let edd be a copy of the ⋖evaluated-data-description⋗ of g in which the simply contained <dimensioned-data-description> has been replaced by the subtree of its <element-data-description>. If d has an <array-init>,ai then perform Steps 1.1 through 1.3 .

Step 1.1. Let e be the <parenthesized-expression> contained in ai. Perform EVALUATE-EXPRESSION(e) to obtain an ⋖aggregate-value⋗,v.

Step 1.2. Let et be an ⋖evaluated-target⋗: g.

Step 1.3. Perform ASSIGN(et,v,dd), where dd is the <data-description> immediately contained in e. Terminate this operation.

Step 2. If d does not have an <initial-element-list> then terminate this operation. Let iel be the <initial-element-list> of d. Let m and n be 1. Let mt be the number of elements in the ⋖storage-index-list⋗ of g. Let nt be the number of elements in the <initial-element-list>,iel.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3. Construct an ⟪evaluated-initial-element-list⟫,eiel, by making a copy of iel and replacing each <initial-element-list> node by an ⟪evaluated-initial-element-list⟫ and each <initial-element> node by an ⟪evaluated-initial-element⟫ in the copy.

Step 4. Perform Steps 4.1 through 4.7 while m≤mt and n≤nt.

Step 4.1. Let the *n*th element of eiel be eie[n].

Step 4.2. If eie[n] immediately contains an <iteration-factor>,itf, then perform EVALUATE-EXPRESSION-TO-INTEGER(e), where e is the <expression> of itf, to obtain the ⟪integer-value⟫,v, and replace itf by an ⟪evaluated-iteration-factor⟫ containing v.

Step 4.3. If eie[n] immediately contains an ⟪evaluated-initial-element-list⟫,eiel1, then for each <expression>,e, contained in eiel1, chosen in any order, perform Step 4.3.1.

Step 4.3.1.

Case 4.3.1.1. e is immediately contained in an <iteration-factor>,itf1.

Optionally perform EVALUATE-EXPRESSION-TO-INTEGER(e) to obtain an ⟪integer-value⟫,i1 and replace itf1 by an ⟪evaluated-iteration-factor⟫ containing i1.

Case 4.3.1.2. e is immediately contained in a <parenthesized-expression>,pe.

Optionally perform EVALUATE-EXPRESSION(e) to obtain an ⟪aggregate-value⟫ having a ⟪basic-value⟫,bv, and replace pe by an ⟪evaluated-initial-item⟫ comprising of bv and a copy of the <data-description> immediate component of pe.

Step 4.4. If eie[n] immediately contains an ⟪evaluated-iteration-factor⟫,eif, then perform Steps 4.4.1 through 4.4.3.

Step 4.4.1. Let eiel2 be the ⟪evaluated-initial-element-list⟫ of eie[n]. Let i2 be the ⟪integer-value⟫ of eif.

Step 4.4.2.

Case 4.4.2.1. i2≤0.

Replace eie[n] by ⟪absent⟫.

Case 4.4.2.2. (Otherwise).

Let k be the number of elements in eiel2. Replace ei e[n] by the i2*k elements formed from i2 replications of the sequence of elements of eiel2. Let nt be nt+i2*k−1.

Step 4.4.3. Go to Step 4.1.

Step 4.5. If eie[n] is neither an <asterisk> nor an ⟪absent⟫ then perform Steps 4.5.1 and 4.5.2.

Step 4.5.1. If eie[n] is a <parenthesized-expression>,pe1, then perform EVALUATE-EXPRESSION(e), where e is the <expression> of pe1, to obtain an ⟪aggregate-value⟫ having the ⟪basic-value⟫,bv1, and replace pe1 by an ⟪evaluated-initial-item⟫ comprising of bv1 and a copy of the <data-description> immediate component of pe1.

Step 4.5.2. eie[n] is an ⟪evaluated-initial-item⟫. Let v be of the form

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

⩽aggregate-value⩾:
⩽aggregate-type⩾:
⩽scalar⩾;
⩽basic-value-list⩾:
bv;;

where bv is the ⩽basic-value⩾ of eie[n]. Let dd be the <data-description> immediate component of eie[n]. Let si be a copy of the *m*th element of the ⩽storage-index-list⩾ of g. Perform ASSIGN(⩽evaluated-target⩾:g1;,v,dd), where g1 is the ⩽generation⩾ comprising of edd constructed in Step 1, a copy of the ⩽allocation-unit-designator⩾ of g, and a ⩽storage-index-list⩾ consisting of the single element si.

Step 4.6. If eie[n] is not ⩽absent⩾ then let m be m+1.

Step 4.7. Let n be n+1.

7.5 The Freeing of Storage

The <free-statement> causes storage allocated for specified <based> variables to be freed.

7.5.1 Execute-free-statement

Operation: EXECUTE-FREE-STATEMENT(fs)

Where: fs is a <free-statement>.

Step 1. For each <freeing>,fr, in the <freeing-list> of fs, chosen in left-to-right order, perform Steps 1.1 and 1.2.

Step 1.1. Let d be the <declaration> designated by the <declaration-designator> component of fr.

Step 1.2. Perform FREE-BASED-STORAGE(fr).

Step 2. Perform NORMAL-SEQUENCE.

7.5.2 Free-based-storage

This operation frees a <based> variable specified in a <freeing>.

Operation: FREE-BASED-STORAGE(fr)

Where: fr is a <freeing>.

Step 1. fr can consist of three components:

a <declaration-designator>,dp
a <locator-qualifier>
and an <in-option>.

Of these, only dp always exists.

Step 2. Let dd be the <data-description> of the <declaration> designated by dp. If fr contains a <locator-qualifier>,lq then let vr be <variable-reference>: lq dp dd. Otherwise let vr be

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

<variable-reference>: dp dd. Let n be the number of <bound-pair>s in dd. If n is not equal to zero, attach a <subscript-list> containing n occurrences of <asterisk> to vr.

Step 3. Perform EVALUATE-VARIABLE-REFERENCE(vr) to obtain a ⟨generation⟩,gf to be freed.

Step 4. Let au be the ⟨allocation-unit⟩ designated by the ⟨allocation-unit-designator⟩ of gf. Let eddg be the ⟨evaluated-data-description⟩ in gf. Let edda be the ⟨evaluated-data-description⟩ in au.

Step 5. Perform TEST-STRING-CLASS(eddg) to obtain scg. Perform TEST-STRING-CLASS(edda) to obtain sca.

Case 5.1. dd has no <refer-option>, and sca is equal to scg, and sca is <bit> or <character>.

Step 5.1.1. Let la be the number of ⟨character-value⟩s or ⟨bit-value⟩s in au.

Step 5.1.2. Let ddg be the <data-description> in eddg. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(ddg) to obtain ng. Let lg be 0. For i=1,...,ng, in order, perform Steps 5.1.2.1 and 5.1.2.2 .

Step 5.1.2.1. Perform FIND-ITEM-DATA-DESCRIPTION(ddg,i) to obtain an <item-data-description>,idd.

Case 5.1.2.1.1. idd contains <string>.

Let li be the value of the <maximum-length> of idd.

Case 5.1.2.1.2. (Otherwise).

Let li be the associated character-string length of the <data-type> of idd.

Step 5.1.2.2. Let lg = lg+li.

Step 5.1.3. la MUST be equal to lg.

Case 5.2. (Otherwise).

Step 5.2.1. Let edda1 be a copy of edda. Let eddg1 be a copy of eddg.

Step 5.2.2. Delete from edda1 and eddg1 all occurrences of <initial>, and all <variable-reference> components of <offset>.

Step 5.2.3. edda1 and eddg1 MUST be equal.

Step 6. Perform DEDUCE-IN-OPTION(fr). If an area containing gf can be inferred from fr, a ⟨generation⟩,ga, will be obtained; otherwise ⟨fail⟩ will be obtained.

Step 7.

Case 7.1. ga exists.

Step 7.1.1. Let av be the ⟨area-value⟩ referred to by ga. av has an ⟨area-allocation⟩,aa which contains au.

Step 7.1.2. Delete au from aa. Let fi be the ordinal position of aa in the ⟨area-action-list⟩ of av. Append an ⟨area-action⟩: ⟨area-freeing⟩: ⟨integer-value⟩: fi;;; to the ⟨area-action-list⟩ of av.

Case 7.2. DEDUCE-IN-OPTION returned ⟨fail⟩ in Step 6.

au MUST be an immediate component of the ⟨allocation-unit-list⟩ of ⟨allocated-storage⟩. The ⟨generation⟩ gf MUST NOT:

1. be equal to a component of the ⟨static-directory⟩,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2. for any ⟨block-state⟩, be equal to a component of
   a. the ⟨automatic-directory⟩
   b. the ⟨parameter-directory⟩ as the component of an ⟨established-argument⟩ containing ⟨dummy⟩
3. for any ⟨file-information⟩, be equal to a component of the ⟨allocated-buffer⟩.

Perform FREE(gf).

7.5.3 Deduce-in-option

This operation infers, if possible, an area ⟨generation⟩ in which a freeing is to be applied.

Operation: DEDUCE-IN-OPTION(fr)

Where: fr is a <freeing>.

Result: a ⟨generation⟩ or ⟨fail⟩.

Case 1. fr contains an <in-option>.

Let io be the <in-option> of fr. Let vr be the <variable-reference> component of io. Perform EVALUATE-VARIABLE-REFERENCE(vr) to obtain the ⟨generation⟩,ga. Return ga.

Case 2. fr contains no <in-option>.

Step 2.1.

Case 2.1.1. There is a <locator-qualifier> as an immediate subnode of fr.

Let vr be the <value-reference> contained in the <locator-qualifier>.

Case 2.1.2. There is no <locator-qualifier> as an immediate subnode of fr.

The <declaration-designator> of fr designates a <declaration>,d. The <based> component of d MUST have a <value-reference>. Let vr be this <value-reference>.

Step 2.2.

Case 2.2.1. vr contains a <variable-reference> with a <declaration-designator> that designates a <declaration> whose <data-type> contains <offset>.

The <declaration-designator> of vr designates a <declaration>,dvr. The <offset> component of dvr MUST have a <variable-reference> component, vra. Perform EVALUATE-VARIABLE-REFERENCE(vra) to obtain the ⟨generation⟩,ga. Return ga.

Case 2.2.2. (Otherwise).

Return the value ⟨fail⟩.

7.5.4 Free

This operation frees an ⟨allocation-unit⟩.

Operation: FREE(g)

Where: g is a ⟨generation⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1. Let au be the ⟨allocation-unit⟩ designated by the ⟨allocation-unit-designator⟩ in g.

Step 2. Delete the ⟨allocation-unit⟩,au, from the ⟨allocation-unit-list⟩ of ⟨allocated-storage⟩.

7.6 Assignment

Assignment involves changing ⟨basic-value⟩ components of storage (the common case), components in the ⟨file-information-list⟩ (in the case of the <pageno-pv>), or components in the ⟨condition-bif-value⟩s of the current ⟨block-state⟩ (in the case of the <onsource-pv>). The components to be changed are determined by evaluating a <target-reference> (Section 7.6.2), to obtain an ⟨evaluated-target⟩ (Section 7.6.2.1). The actual assignment is effected by the operation ASSIGN (Section 7.6.3), which, in general, will involve conversion of ⟨basic-value⟩s and promotion of ⟨aggregate-value⟩s. The full generality of assignment is available through the <assignment-statement> (Section 7.6.1), but other constructions cause invocation of the operation ASSIGN.

7.6.1 The Assignment Statement

Operation: EXECUTE-ASSIGNMENT-STATEMENT(ast)

Where: ast is an <assignment-statement>.

Step 1.

Case 1.1. The <target-reference>,tr of ast has a <data-description> which immediately contains an <item-data-description>.

In either order, perform Steps 1.1.1 and 1.1.2.

Step 1.1.1. Perform EVALUATE-TARGET-REFERENCE(tr) to obtain an ⟨evaluated-target⟩,et.

Step 1.1.2. Let e be the <expression> of ast. Perform EVALUATE-EXPRESSION(e) to obtain an ⟨aggregate-value⟩,v. Let d be the <data-description> of e.

Case 1.2. The <target-reference>,tr of ast has a <data-description> which does not immediately contain an <item-data-description>.

Let e be the <expression> of ast.

Case 1.2.1. The <data-description> of e immediately contains an <item-data-description>.

In either order, perform Steps 1.2.1.1 and 1.2.1.2.

Step 1.2.1.1. Perform EVALUATE-TARGET-REFERENCE(tr) to obtain an ⟨evaluated-target⟩,et. et contains a ⟨generation⟩,g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⟨true⟩.

Step 1.2.1.2. Perform EVALUATE-EXPRESSION(e) to obtain an ⟨aggregate-value⟩,v. Let d be the <data-description> of e.

Case 1.2.2. The <data-description> of e does not immediately contain an <item-data-description>.

e MUST be of the form <expression>: <value-reference>: <variable-reference>,vr.

In either order, perform Steps 1.2.2.1 and 1.2.2.2.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.2.2.1. Perform EVALUATE-TARGET-REFERENCE(tr) to obtain an ⩽evaluated-target⩾,et. et contains a ⩽generation⩾,g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⩽true⩾.

Step 1.2.2.2. Perform EVALUATE-VARIABLE-REFERENCE(vr) to obtain a ⩽generation⩾,g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⩽true⩾. Perform VALUE-OF-GENERATION(g) to obtain an ⩽aggregate-value⩾,v. Let d be the <data-description> of the ⩽evaluated-data-description⩾ of g.

Step 2. Perform ASSIGN(et,v,d).

Step 3. Perform NORMAL-SEQUENCE.

7.6.2 Target References

Attributes: The result <data-description> of a <target-reference> that immediately contains a <variable-reference> is the same as the result <data-description> of the <variable-reference>. For a <target-reference> that has a <pseudo-variable-reference>, the result <data-description> is given in the description of the <pseudo-variable> (Section 7.6.4).

Operation: EVALUATE-TARGET-REFERENCE(tr)

Where: tr is a <target-reference> or <variable-reference>.

Result: an ⩽evaluated-target⩾.

Case 1. tr is a <target-reference> which immediately contains a <variable-reference>,vr.

Perform EVALUATE-VARIABLE-REFERENCE(vr) to obtain a ⩽generation⩾,g. Return an ⩽evaluated-target⩾: g.

Case 2. tr is a <target-reference> which immediately contains a <pseudo-variable-reference>,pvr.

Perform EVALUATE-F(x[1],...,x[n]), where f is the immediate component of the <pseudo-variable> contained by pvr, and x[1],...,x[n] are the <argument>s, if any, in the <argument-list>. Let g be the result of EVALUATE-F(x[1],...,x[n]). (It will be either a ⩽generation⩾ or an ⩽evaluated-pseudo-variable-reference⩾.) Return an ⩽evaluated-target⩾: g.

Case 3. tr is a <variable-reference>.

Perform EVALUATE-VARIABLE-REFERENCE(tr) to obtain a ⩽generation⩾,g. Return an ⩽evaluated-target⩾: g.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.6.2.1 Evaluated Targets

Operation: VALUE-OF-EVALUATED-TARGET(t)

Where: t is an ⟨evaluated-target⟩.

Result: an ⟨aggregate-value⟩.

Case 1. t immediately contains a ⟨generation⟩,g.

Perform VALUE-OF-GENERATION(g) to obtain an ⟨aggregate-value⟩,av, which MUST NOT contain ⟨undefined⟩. Return av.

Case 2. t contains <exponent-pv>.

Let g be the ⟨generation⟩ in t, and perform VALUE-OF-GENERATION(g) to obtain an ⟨aggregate-value⟩,x, which MUST NOT contain ⟨undefined⟩. Let sr be the scalar-result of performing Steps 1 and 2 of EXPONENT-BIF (see Section 9.5.4.29), taking:

the scalar <data-type> of x to be the <data-type> in the ⟨generation⟩ of t;
the scalar-result of x to be the ⟨basic-value⟩ in x.

Return an ⟨aggregate-value⟩ containing sr.

Case 3. t contains <onsource-pv>.

Perform ONSOURCE-BIF (see Section 9.5.4.56), to obtain an ⟨aggregate-value⟩,av. Return av.

Case 4. t contains <pageno-pv>.

The ⟨file-value⟩,fv in t MUST obey the constraints of PAGENO-BIF, Step 1 (see Section 9.5.4.57). Perform Steps 2 and 3 of PAGENO-BIF.

Case 5. t contains <substr-pv>.

Perform VALUE-OF-GENERATION(g) to obtain an ⟨aggregate-value⟩,sa, where g is the ⟨generation⟩ in t. sa MUST NOT contain ⟨undefined⟩. Let st be the first ⟨aggregate-value⟩ in t; let le be the second ⟨aggregate-value⟩ in t, if present.

Let sr be the scalar-result of performing Steps 1 to 4 of SUBSTR-BIF (see Section 9.5.4.70), taking:

the scalar-value of sa (or of st or of le) to be the single ⟨basic-value⟩ in sa (or in st or in le);
the scalar-result-type to be the <data-type> in the ⟨generation⟩ in t.

Return an ⟨aggregate-value⟩ containing sr.

Case 6. t contains <unspec-pv>.

Let g be the ⟨generation⟩ in t. Perform Step 3 of UNSPEC-BIF (see Section 9.5.4.81).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.6.3 The Assignment Operation

Operation: ASSIGN(et,sv,sd)

Where: et is an ⩽evaluated-target⩾,
sv is an ⩽aggregate-value⩾,
sd is a <data-description>.

Step 1. For each <union-data-description>,udds, in sd, perform Step 1.1 .

Step 1.1. For each <union-data-description>,uddt, in et, that corresponds to udds, perform TEST-UNION-MATCHING(uddt,udds) to obtain tv. tv MUST be ⩽true⩾.

Step 2.

Case 2.1. et immediately contains a ⩽generation⩾,g.

Step 2.1.1. Let edd be the ⩽evaluated-data-description⩾ of g. Perform PROMOTE-AND-CONVERT(edd,sv,sd), to obtain an ⩽aggregate-value⩾,av.

Step 2.1.2. Perform SET-STORAGE(g,svl), where svl is the ⩽basic-value-list⩾ of av.

Case 2.2. et immediately contains an ⩽evaluated-pseudo-variable-reference⩾,epvr.

Let f be the component of the <pseudo-variable> in epvr. Thus f is the name of the <pseudo-variable>. Perform ASSIGN-F(epvr,sv,sd).

7.6.3.1 Promote-and-convert

Let x and y be ⩽aggregate-type⩾s. Then x is *promotable* to y if x and y are compatible or x is a scalar and y is an array of scalars. However where one has a <bound-pair> which has <asterisk>, the other may have a <bound-pair> whose <expression> components contain ⩽integer-value⩾s.

Operation: PROMOTE-AND-CONVERT(td,sv,sd)

Where: td is a <data-description> or ⩽evaluated-data-description⩾,
sv is an ⩽aggregate-value⩾,
sd is a <data-description>.

Result: an ⩽aggregate-value⩾.

Step 1. Let ats be the ⩽aggregate-type⩾ of sv.

Case 1.1. td is an ⩽evaluated-data-description⩾.

Let tdd be td.

Case 1.2. td is not an ⩽evaluated-data-description⩾.

Perform EVALUATE-DATA-DESCRIPTION-FOR-ALLOCATION(td) to obtain an ⩽evaluated-data-description⩾,tdd. For each tree of the form <bound-pair>,bp: <asterisk>; contained in tdd that is not a component of an <entry>, perform Step 1.2.1.

Step 1.2.1. If a <bound-pair>,cbp corresponding to bp exists in ats, then replace bp by cbp. Otherwise, replace bp by a <bound-pair> whose <lower-bound> and <upper-bound> each have an ⩽integer-value⩾ of 1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(tdd) to obtain an integer, n. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(sd) to obtain an integer, m.

Step 3.

Case 3.1. m = 1.

Step 3.1.1. Let x be the scalar-element of sv. Let xdt be the <data-type> of the <item-data-description> in sd. Let ydt be the <data-type> of the <item-data-description> in tdd.

Step 3.1.2.

Case 3.1.2.1. Both xdt and ydt contain <computational-type>.

Perform CONVERT(ydt,xdt,x) to obtain y.

Case 3.1.2.2. Both xdt and ydt contain <area>.

Perform either Step 3.1.2.2.1 or 3.1.2.2.2.

Step 3.1.2.2.1. Perform RAISE-CONDITION(<area-condition>). Let y be a ⟨basic-value⟩ containing ⟨undefined⟩.

Step 3.1.2.2.2. Perform AREA-CONVERSION(ydt,x) to obtain an ⟨area-value⟩,y.

Case 3.1.2.3. (Otherwise).

Let y be a copy of x.

Step 3.1.3. Let av be an ⟨aggregate-value⟩ whose ⟨aggregate-type⟩ equals the ⟨aggregate-type⟩ associated with the <data-description> of tdd, and whose ⟨basic-value-list⟩,svl, contains n ⟨basic-value⟩ nodes, each of which contains a copy of y.

Step 3.1.4. Return av.

Case 3.2. m > 1.

Each <bound-pair-list>, <area-size>, or <maximum-length> in sd MUST equal the corresponding <bound-pair-list>, <area-size>, or <maximum-length> in tdd. Return a copy of sv.

7.6.3.2 The Set-storage Operation

This operation sets the elements of a ⟨basic-value-list⟩ in ⟨allocated-storage⟩ described by a ⟨generation⟩ to have the values contained in a ⟨basic-value-list⟩.

Operation: SET-STORAGE(g,ssvl)

Where: g is a ⟨generation⟩,
ssvl is a ⟨basic-value-list⟩.

Step 1. Let au be the ⟨allocation-unit⟩ designated by the ⟨allocation-unit-designator⟩ of g. au MUST be contained in ⟨allocated-storage⟩. Let tsvl be the ⟨basic-value-list⟩ of au.

Step 2. If au contains a ⟨union-list⟩,uasl, then for each ⟨union⟩,uas in uasl perform Step 2.1.

Step 2.1. If uas contains a ⟨basic-value-index⟩ that is equal to any ⟨basic-value-index⟩ in g, then perform SET-UNION(g,uas).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3. Let dd be the <data-description> of the ⩽evaluated-data-description⩾ of g and let sil be the ⩽storage-index-list⩾ of g.

Step 4. For each of the elements of sil, taken in any order, perform Steps 4.1 through 4.3 .

Step 4.1. Let si be the chosen element and let i be its ordinal in sil. Let j be a copy of the value of the ⩽integer-value⩾ of the ⩽basic-value-index⩾ of si.

Step 4.2. Let ssv be the *i*th element of the ⩽basic-value-list⩾,ssvl.

Step 4.3.

Case 4.3.1. si contains a ⩽position-index⩾.

Let k be a copy of the value of the ⩽integer-value⩾ of the ⩽position-index⩾ of si. ssv will contain either a ⩽character-string-value⩾ or a ⩽bit-string-value⩾. Perform FIND-ITEM-DATA-DESCRIPTION(dd,i) to obtain the <item-data-description>,idd, of the target scalar-element.

If idd contains <pictured> then let n be its associated string-length; otherwise let n be the value of the <maximum-length> in idd. Let m be 1. Perform Steps 4.3.1.1 through 4.3.1.4 n times.

Step 4.3.1.1. Let tl be the ⩽character-value-list⩾ or ⩽bit-value-list⩾ of the *j*th element of tsvl.

Step 4.3.1.2. Replace the *k*th element of tl by a copy of the *m*th element of ssv.

Step 4.3.1.3. Let k be k+1. If k is now greater than the number of elements in tl, then let k be 1 and let j be j+1. Repeat Step 4.3.1.3.1 while the *j*th element of tsvl contains ⩽null-character-string⩾ or ⩽null-bit-string⩾.

Step 4.3.1.3.1. Set j to j+1.

Step 4.3.1.4. Let m be m+1.

Case 4.3.2. si does not contain a ⩽position-index⩾.

Replace the *j*th element of tsvl by a copy of ssv.

7.6.3.3 Area-conversion

Operation: AREA-CONVERSION(tdt,av)

Where: tdt is a <data-type> containing <area>,
av is an ⩽area-value⩾.

Result: an ⩽area-value⩾.

Step 1. Let ts be the ⩽integer-value⩾ in the <area-size> of tdt.

Step 2.

Case 2.1. av contains ⩽empty⩾.

Return an

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

⟨area-value⟩:
⟨area-action-list⟩:
⟨area-action⟩:
⟨area-size⟩:
ts.

Case 2.2. av does not contain ⟨empty⟩.

The first ⟨area-action⟩ in the ⟨area-action-list⟩ of av contains an ⟨area-size⟩: ⟨integer-value⟩.as.

Case 2.2.1. as is equal to ts.

Return a copy of av.

Case 2.2.2. as is not equal to ts.

Let tv be a copy of av. Append to the ⟨area-action-list⟩ of tv an ⟨area-action⟩: ⟨area-size⟩: ts. Return tv.

7.6.3.4 Test-union-matching

TEST-UNION-MATCHING insures that two <union-data-description>s are identical, disregarding the comparison of irrelevant subnodes.

Operation: TEST-UNION-MATCHING(udd1,udd2)

Where: udd1 and udd2 are both <union-data-description>s.

Result: ⟨true⟩ or ⟨false⟩.

Step 1. Let uddc1 and uddc2 be copies of udd1 and udd2 respectively. Delete each occurrence of the following subtrees of uddc1 and uddc2:

<variable-reference> as a component of an <offset>
<parameter-descriptor-list>
<returns-descriptor>
<options>
<identifier-list> as a component of a <structure-data-description> or a <union-data-description>

Step 2. uddc1 and uddc2 MUST be equal.

7.6.3.5 Set-union

This operation determines which alternative of a union is being set by a SET-STORAGE operation, and sets all others undefined.

Operation: SET-UNION(g,uas)

Where: g is a ⟨generation⟩,
uas is a ⟨union⟩.

Step 1. Let au be the ⟨allocation-unit⟩ designated by the ⟨allocation-unit-designator⟩ of g. Let edd be the ⟨evaluated-data-description⟩ of au. Let bvl be the ⟨basic-value-list⟩ of au.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2. If every ⟨basic-value-index⟩ of uas is in g, then terminate this operation. (An assignment is being made to the entire union rather than to a single member.)

Step 3. Let ua be the ⟨union-alternative⟩ of uas such that one or more of the ⟨basic-value-index⟩s of ua is in g. By construction, uas exists and is unique.

Step 4. For each ⟨union-alternative⟩,ua1 of uas other than ua, perform Step 4.1.

Step 4.1. For each ⟨basic-value-index⟩,bvix of ua1, perform Step 4.1.1.

Step 4.1.1. Let i be the integer of the ⟨basic-value-index⟩ of bvix. Let bv be the *i*th ⟨basic-value⟩ of bvl. Perform FIND-ITEM-DATA-DESCRIPTION(edd,i) to obtain an <item-data-description>,idd.

Case 4.1.1.1. idd has <string> which in turn has <nonvarying>, or idd has <pictured>.

Replace the immediate subnode of each ⟨bit-value⟩ or ⟨character-value⟩ of bv (as appropriate) by ⟨undefined⟩.

Case 4.1.1.2. (Otherwise).

Replace the immediate subnode of bv by ⟨undefined⟩.

7.6.4 Pseudo-variables

This Section presents the definitions of the <pseudo-variable>s in alphabetical order. For each <pseudo-variable>:

- The Arguments Section indicates the number of arguments to the <pseudo-variable> and supplies names by which the arguments are referenced in the Attributes and Constraint Sections.
- The Constraint Section specifies constraints on the arguments.
- The Attributes Section defines the <data-description> of the <pseudo-variable-reference> (and of the <target-reference> containing it) in terms of the <data-description>s of the arguments.

One or two operations are defined for each <pseudo-variable>. This first operation, EVALUATE-F, where f is the name of the <pseudo-variable>, is used by the operation EVALUATE-TARGET-REFERENCE. This operation returns a ⟨generation⟩ or an ⟨evaluated-pseudo-variable-reference⟩.

The second operation, ASSIGN-F, where f is the name of the <pseudo-variable>, is defined only for those <pseudo-variable>s whose EVALUATE-F operation yields an ⟨evaluated-pseudo-variable-reference⟩.

7.6.4.1 Exponent-pv

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. x MUST have the form <argument>: <expression>: <value-reference>: <variable-reference>.

The <data-type> of x MUST have <arithmetic> (but not <pictured-numeric>) with <scale>: <float>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> has integer-type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: EVALUATE-EXPONENT-PV(x)

Step 1. Let y be the <variable-reference> in x. Perform EVALUATE-VARIABLE-REFERENCE(y) to obtain a ⩽generation⩾,g. Return an ⩽evaluated-pseudo-variable-reference⩾: <pseudo-variable>: <exponent-pv>; g.

Operation: ASSIGN-EXPONENT-PV(t,sv,sd)

Where: t is an ⩽evaluated-pseudo-variable-reference⩾ as returned by EVALUATE-EXPONENT-PV, i.e.,

⩽evaluated-pseudo-variable-reference⩾:
<pseudo-variable>:
<exponent-pv>;
⩽generation⩾,g:
⩽evaluated-data-description⩾,edd
⩽allocation-unit-designator⩾,aud
⩽storage-index-list⩾,sil;;,

sv is an ⩽aggregate-value⩾,
sd is a <data-description>.

Step 1. Let dr be the same as edd except that the <data-type> has integer-type. Perform PROMOTE-AND-CONVERT(dr,sv,sd) to obtain an ⩽aggregate-value⩾,av.

Step 2. Let dt be the <data-type> in edd.

Step 3. Perform MODEL-MAXIMUM-EXPONENT(dt) to obtain an ⩽integer-value⩾,emax. Perform MODEL-MINIMUM-EXPONENT(dt) to obtain an ⩽integer-value⩾,emin. Perform MODEL-BASE(dt) to obtain an ⩽integer-value⩾,b.

Step 4. Let n be the ⩽real-number⩾ of the scalar-element in av. n MUST be such that emin $\leq$ n $\leq$ emax.

Step 5. Perform VALUE-OF-GENERATION(g) to obtain an ⩽aggregate-value⩾ containing a single ⩽basic-value⩾,bv, which MUST NOT contain ⩽undefined⩾. Let v be the ⩽real-number⩾ in bv.

Step 6.

Case 6.1. v = 0.

Let w be a ⩽real-number⩾: 0.

Case 6.2. (Otherwise).

Let e be the unique integer such that $b^{e-1} \leq |v| < b^e$. Let w be a ⩽real-number⩾: $v*b^{n-e}$.

Step 7. Let bvl be a ⩽basic-value-list⩾: ⩽basic-value⩾: ⩽real-value⩾: w. Perform SET-STORAGE(g,bvl).

7.6.4.2 Onsource-pv

Arguments: (none)

Attributes: The result has ⩽aggregate-type⩾: ⩽scalar⩾. The result <data-type> has <character>.

Operation: EVALUATE-ONSOURCE-PV

Step 1. Return an ⩽evaluated-pseudo-variable-reference⩾: <pseudo-variable>: <onsource-pv>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: ASSIGN-ONSOURCE-PV(t,sv,sd)

Where: t is an ⟨evaluated-pseudo-variable-reference⟩,
sv is an ⟨aggregate-value⟩,
sd is a <data-description>.

Step 1. Perform GET-ESTABLISHED-ONVALUE(⟨onsource-value⟩) to obtain str. str MUST NOT be ⟨fail⟩. Let n be its length. Convert sv to <character> of length n, using the <data-type> in sd as the source <data-type> for this conversion, to obtain cv.

Step 2. Replace str by cv.

7.6.4.3 Pageno-pv

Arguments: fn

Constraint: fn MUST have ⟨aggregate-type⟩: ⟨scalar⟩. The <data-type> of fn MUST have <file>.

Attributes: The result has ⟨aggregate-type⟩: ⟨scalar⟩. The result <data-type> has integer-type.

Operation: EVALUATE-PAGENO-PV(fn)

Step 1. Perform EVALUATE-EXPRESSION(fn) to obtain fv. fv MUST NOT contain ⟨null-file⟩. Return a

⟨evaluated-pseudo-variable-reference⟩:
<pseudo-variable>:
<pageno-pv>;
fv.

Operation: ASSIGN-PAGENO-PV(t,sv,sd)

Where: t is an ⟨evaluated-pseudo-variable-reference⟩,
sv is an ⟨aggregate-value⟩,
sd is a <data-description>.

Step 1. Let f be the ⟨file-information⟩ designated by the ⟨file-value⟩ in t. f MUST have ⟨open⟩. The ⟨complete-file-description⟩ of f MUST have <print>.

Step 2. Let bv be the ⟨basic-value⟩ in sv. Let tdt be a <data-type> which is integer-type (Section 9.2.2). Perform CONVERT(tdt,sd,bv) to obtain a ⟨real-value⟩,rv. rv MUST be non-negative.

Step 3. Set the ⟨page-number⟩ component in f to contain an ⟨integer-value⟩ with the same component as rv.

7.6.4.4 String-pv

Arguments: x

Constraint: x MUST have the form <argument>: <expression>: <value-reference>: y;;; where y is a <variable-reference>. The <data-description> of x MUST NOT have a <union-data-description>. Let ad be the <declaration> designated by the <declaration-designator> in x. Further, the result of performing TEST-STRING-CLASS(ad) MUST be either <bit> or <character>.

Attributes: The result has ⟨aggregate-type⟩: ⟨scalar⟩. The result <data-type> has the derived common <string-type> of the <data-type>s of x, and <nonvarying>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: EVALUATE-STRING-PV(x)

Step 1. Let y be the <variable-reference> in x. Perform EVALUATE-VARIABLE-REFERENCE(y) to obtain a ⟨generation⟩,gy. Perform TEST-CONNECTEDNESS(gy) to obtain tv, which MUST be ⟨true⟩. Let the ⟨evaluated-data-description⟩, ⟨allocation-unit-designator⟩, and ⟨storage-index-list⟩ components of gy be d, aud and s, respectively.

Step 2. If the <data-type> of the *i*th scalar-element in the ⟨generation⟩ has <string>, let k[i] be the value of the corresponding <maximum-length> component in d; otherwise let k[i] be the associated character-string length of the <pictured> in d.

Step 3. Return a ⟨generation⟩,g, with components as follows. The ⟨evaluated-data-description⟩ of g is described under Attributes, above, with <maximum-length> the sum of all the k[i]. The ⟨allocation-unit-designator⟩ of g is a copy of aud. The ⟨storage-index-list⟩ of g contains a copy of the first component of s.

7.6.4.5 Substr-pv

Arguments: t,st[,le]

Constraint: The ⟨aggregate-type⟩s of t, st and le MUST immediately contain ⟨scalar⟩. The <data-type>s of st and le MUST have <fixed> (but not <pictured-numeric>) with <scale-factor>: 0. t MUST have the form <argument>: <expression>: <value-reference>: <variable-reference>.

The <data-type> of t MUST have <string>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> is the same as the <data-type> of t.

Operation: EVALUATE-SUBSTR-PV(t,st,le)

Step 1. Let ty be the <variable-reference> in t. In any order, perform EVALUATE-VARIABLE-REFERENCE(ty) to obtain a ⟨generation⟩,g, perform EVALUATE-EXPRESSION(st) and EVALUATE-EXPRESSION(le), if le occurs, to obtain ⟨aggregate-value⟩s, x and y.

Step 2. Return an ⟨evaluated-pseudo-variable-reference⟩: <pseudo-variable>: <substr-pv>; g x [y].

Operation: ASSIGN-SUBSTR-PV(t,sv,sd)

Where: t is an ⟨evaluated-pseudo-variable-reference⟩ as returned by EVALUATE-SUBSTR-PV, i.e.

```
⟨evaluated-pseudo-variable-reference⟩:
   <pseudo-variable>:
      <substr-pv>;
   ⟨generation⟩,g:
      ⟨evaluated-data-description⟩,edd
      ⟨allocation-unit-designator⟩,aud
      ⟨storage-index-list⟩,sil;
   ⟨aggregate-value⟩,x1
   [⟨aggregate-value⟩,y1];,
```

sv is an ⟨aggregate-value⟩,
sd is a <data-description>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1. Perform VALUE-OF-GENERATION(g) to obtain an ⟨aggregate-value⟩,av. If sd has <varying> then av MUST NOT contain ⟨undefined⟩. Let k be the length of the ⟨character-string-value⟩ or ⟨bit-string-value⟩ in av (cf. Section 9.2.3.4).

Step 2. Let i be the scalar-element in x1. If y1 exists, let j be the scalar-element in y1; otherwise let j=k−i+1.

Step 3.

Case 3.1. $0 \leq i-1 \leq j+i-1 \leq k$.

Step 3.1.1. Let st be the <string-type> contained in g. Let sgl be a

```
⟨generation⟩:
   ⟨evaluated-data-description⟩:
      <data-description>:
         <item-data-description>:
            <data-type>:
               <computational-type>:
                  <string>:
                     st
                     <maximum-length>:
                        <extent-expression>:
                           ⟨integer-value⟩:
                              j;;;
                     <nonvarying>;;;;;;
aud
sil.
```

Append to p a ⟨position-index⟩ of i.

Step 3.1.2. Perform CONVERT(dtl,ssv,ssd) to obtain ssvl, where dtl is the <data-type> of sgl, ssv is the scalar-element in sv, and ssd is the <data-type> in sd.

Step 3.1.3. Perform SET-STORAGE(sgl,ssvl).

Case 3.2. (Otherwise).

This case MUST NOT occur.

7.6.4.6 Unspec-pv

The UNSPEC pseudo-variable is normally used to store a bit string representing the internal representation of a variable into the generation of storage associated with that variable. Its use makes a PL/I program dependent on the particular implementation and the representations it uses, which are not specified by this standard.

Arguments: x

Constraint: x MUST have the form <argument>: <expression>: <value-reference>: <variable-reference>.

Attributes: The result has ⟨aggregate-type⟩: ⟨scalar⟩. The result <data-type> has <bit>.

Operation: EVALUATE-UNSPEC-PV(x)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1. Let y be the <variable-reference> in x. Perform EVALUATE-VARIABLE-REFERENCE(y) to obtain a ⩽generation⩾,g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⩽true⩾. Return an ⩽evaluated-pseudo-variable-reference⩾: <pseudo-variable>: <unspec-pv>; g.

Operation: ASSIGN-UNSPEC-PV(t,sv,sd)

Where: t is an ⩽evaluated-pseudo-variable-reference⩾,
sv is an ⩽aggregate-value⩾,
sd is a <data-description>.

Step 1. sv MUST have ⩽aggregate-type⩾: ⩽scalar⩾. Let g be the ⩽generation⩾ in t. Convert sv to <bit> of length n, where n depends on g in an IMPLEMENTATION-DEFINED fashion.

Step 2. Let nse be the number-of-scalar-elements of the <data-description> in g.

Step 3. In an IMPLEMENTATION-DEFINED fashion, construct from the converted value of sv, a ⩽basic-value-list⩾,bvl containing nse ⩽basic-value⩾s which depend on the properties of g. Any or all of these ⩽basic-value⩾s may contain ⩽undefined⩾.

Step 4. Perform SET-STORAGE(g,bvl).

7.7 Variable-reference

7.7.1 Evaluate-variable-reference

The result of this operation is the ⩽generation⩾ referenced in the <variable-reference>.

Operation: EVALUATE-VARIABLE-REFERENCE(vr)

Where: vr is a <variable-reference>.

Result: a ⩽generation⩾.

Step 1. Let d be the <declaration> designated by the <declaration-designator>,dp, of vr.

Step 2.

Case 2.1. The <storage-type> of d has <based>.

Perform SELECT-BASED-GENERATION(vr) to obtain the ⩽generation⩾,g.

Case 2.2. The <storage-type> of d has <defined>.

Perform EVALUATE-DEFINED-REFERENCE(vr) to obtain the ⩽generation⩾,g. Go to Step 5.

Case 2.3. The <storage-type> of d has <parameter>.

Perform FIND-BLOCK-STATE-OF-DECLARATION(dp) to obtain the ⩽block-state⩾,bs, of d. Find the ⩽parameter-directory-entry⩾,pde, in the ⩽parameter-directory⩾ of bs whose <identifier> component is the same as the <identifier> immediate component of d. pde MUST NOT immediately contain the ⩽undefined⩾ component. Let g be a copy of the ⩽generation⩾ of pde. The ⩽allocation-unit⩾ designated by the ⩽allocation-unit-designator⩾ of g MUST be contained in the ⩽allocation-unit-list⩾ of ⩽allocated-storage⩾, possibly as a component of an ⩽area-value⩾.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.4. The <storage-type> of d has <automatic> or <static>.

Perform FIND-DIRECTORY-ENTRY(dp) to obtain a directory entry. Let g be a copy of the ⋖generation⋗ component of the directory entry.

Step 3. If vr immediately contains an <identifier-list>,il, then perform SELECT-QUALIFIED-REFERENCE(g,il,d) to obtain a ⋖generation⋗,g.

Step 4. If vr has a <subscript-list>, then perform SELECT-SUBSCRIPTED-REFERENCE(g,sl), where sl is the <subscript-list>, to obtain a ⋖generation⋗,g.

Step 5. In the ⋖evaluated-data-description⋗ of g, replace each <offset> component by a copy of the corresponding <offset> component of the <data-description> immediate component of the <variable-reference>,vr.

Note: This is a definitional artifice to make the proper target <offset>s available for conversion in the operation ASSIGN.

Step 6. Return g.

7.7.1.1 Test Connectedness

Operation: TEST-CONNECTEDNESS(g)

Where: g is a ⋖generation⋗

Result: ⋖true⋗ or ⋖false⋗

Step 1. Let g be

⋖generation⋗:
⋖evaluated-data-description⋗,edd:
<data-description>,dd;
⋖allocation-unit-designator⋗,aud
⋖storage-index-list⋗,sil.

Let m be the number of elements in sil.

Step 2. Denote by bv[i] the *i*th element of the ⋖basic-value-list⋗ of the ⋖allocation-unit⋗ designated by aud.

Step 3. Perform TEST-STRING-CLASS(dd) to obtain sc.

Case 3.1. sc is <bit> or <character>.

Step 3.1.1. Denote by s[k] the *k*th ⋖storage-index⋗ of sil. Denote by p[k] the *k*th ⋖position-index⋗ of sil.

Step 3.1.2. For k = 1,...,m−1 perform Steps 3.1.2.1 through 3.1.2.2 .

Step 3.1.2.1. Perform FIND-ITEM-DATA-DESCRIPTION(dd,k) to obtain an <item-data-description>,idd.

Step 3.1.2.2. If idd contains <pictured>, let ml[k] be the associated character-string length of the <data-type> of idd. Otherwise, let ml[k] the ⋖integer-value⋗ contained in the <maximum-length> of idd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Note: The (s[k], p[k]) pairs indicate the positions of the successive scalar elements of g. The ml[k] indicate the number of characters or bits that should be between (s[k],p[k]) and (s[k+1],p[k+1]) for g to be connected. We check this condition by creating the virtual position (s[k],p[k]+ml[k]) and then moving through successive ⟪basic-value⟫s, each time subtracting from the virtual position the number of characters or bits in the basic value. When the virtual position lies within the actual position, we are done.

Step 3.1.3. For k = 1,...,m−1 perform Steps 3.1.3.1 through 3.1.3.2.

Step 3.1.3.1. Let sx = s[k], px = p[k] + ml[k]. Let sw be ⟪false⟫. Perform Step 3.1.3.1.1 until sw is ⟪true⟫.

Step 3.1.3.1.1. Let nel be the number of elements in the ⟪character-value-list⟫ or ⟪bit-value-list⟫ of bv[sx], or 0 if bv[sx] is a ⟪null-character-string⟫ or ⟪null-bit-string⟫. If px is less than or equal to nel, let sw be ⟪true⟫; otherwise let px = px−nel (px is necessarily greater than 0 in this case) and let sx = sx+1 (thus advancing to the next scalar element).

Step 3.1.3.2. If sx is not equal to s[k+1], or if px is not equal to p[k+1], return ⟪false⟫. (g is not connected.)

Step 3.1.4. Return ⟪true⟫. (All scalar elements of g have met the test, so g is connected.)

Case 3.2. (Otherwise.)

Denote by s[k] the *k*th element of sil. If the s[k], k=1,...,m, are consecutive integers then return ⟪true⟫; otherwise, return ⟪false⟫. (Since g could not have been defined through string overlaying, any ⟪position-index⟫s that are present MUST necessarily be 1.)

7.7.1.2 Test String Class

Operation: TEST-STRING-CLASS(dd)

Where: dd is a <data-description> or an ⟪evaluated-data-description⟫.

Result: <bit> or <character> or ⟪none⟫.

Step 1. Let iddl be an <item-data-description-list> with no subnodes.

Step 2. For each <item-data-description>,idd contained in dd, perform Step 2.1.

Step 2.1. If there does not exist a <data-type>,dte or <expression>,dte such that dd contains dte and dte contains idd, append idd to iddl.

Note: This test excludes <item-data-description>s contained in <offset>s, <entry>s, <extent-expression>s, and similar nodes.

Step 3. If any <alignment> immediately contained in an immediate component of iddl has <aligned>, return ⟪none⟫.

Step 4.

Case 4.1. Each <data-type> immediately contained in an immediate component of iddl contains <bit> and <nonvarying>.

Return <bit>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 4.2. Each <data-type> immediately contained in an immediate component of iddl contains either

1. <character> and <nonvarying>, or
2. <pictured>.

Return <character>.

Case 4.3. (Otherwise).

Return ⟨none⟩.

7.7.2 Select-based-generation

The result of this operation is the ⟨generation⟩ corresponding to the <based> variable referenced in a <variable-reference> before any name-qualification or subscripting has been performed.

Operation: SELECT-BASED-GENERATION(vr)

Where: vr is a <variable-reference>.

Result: a ⟨generation⟩.

Step 1.

Case 1.1. vr has a <locator-qualifier>,lq.

Let valr be the <value-reference> of lq.

Case 1.2. vr has no <locator-qualifier>.

The <based> component of the <declaration> designated by the <declaration-designator> of vr has a <value-reference>. Let this be valr.

Step 2. Perform EVALUATE-VALUE-REFERENCE(valr) to obtain an ⟨aggregate-value⟩ with a ⟨basic-value⟩,v.

Step 3. Let dt be the <data-type> of valr. If dt has <offset>, perform CONVERT(<pointer>,dt,v) to obtain a ⟨pointer-value⟩,v.

Step 4. v MUST NOT contain ⟨null⟩. Let g be the ⟨generation⟩ in v.

Step 5. The ⟨allocation-unit⟩ designated by the ⟨allocation-unit-designator⟩ component of g MUST be contained in the ⟨allocation-unit-list⟩ of ⟨allocated-storage⟩, possibly as a component of an ⟨area-value⟩.

Step 6. Perform CHECK-BASED-REFERENCE(g,vr) to obtain a ⟨generation⟩,g1.

Step 7. Return g1.

7.7.3 Check-based-reference

This operation checks that the attributes of the variable being referenced agree with those of the ⟨generation⟩ being referenced.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Operation: CHECK-BASED-REFERENCE(g,vr)

Where: g is a ⩽generation⩾,
vr is a <variable-reference>.

Result: a ⩽generation⩾.

Step 1. Let d be the <declaration> designated by the <declaration-designator> of vr. Let dd be the <data-description> immediately contained in the <variable> of d.

Step 2. Let g comprise the ⩽allocation-unit-designator⩾,aud, the ⩽evaluated-data-description⩾,edd, and the ⩽storage-index-list⩾,sil.

Step 3.

Case 3.1. The immediate component of dd is a <structure-data-description>,sdd.

Step 3.1.1. sdd contains an <identifier-list>,mil, and a <member-description-list>,mdl.

Step 3.1.2. If vr contains an <identifier-list>,il, then let m be the position of the element of mil that is identical to the first element of il. Otherwise, let m be the number of elements in mil.

Step 3.1.3. Let dd1 be

<data-description>:
<structure-data-description>:
<identifier-list>,mil1
<member-description-list>,mdl1;;

where mil1 contains a copy of the first m elements of mil, and mdl1 contains a copy of the first m elements of mdl.

Case 3.2. (Otherwise).

Let dd1 be a copy of dd.

Step 4. Perform EVALUATE-DATA-DESCRIPTION-FOR-REFERENCE(dd1,g) to obtain an ⩽evaluated-data-description⩾,edd1.

Step 5. Perform TEST-STRING-CLASS(edd) to obtain sc. Perform TEST-STRING-CLASS(edd1) to obtain sc1.

Case 5.1. All of the following are true:

1. dd1 has no <refer-option> node,
2. sc1 is equal to sc,
3. sc1 is either <bit> or <character>,
4. edd and dd1 do not have <union-data-description> nodes.

Perform OVERLAY-STRINGS(edd1,g) to obtain a ⩽storage-index-list⩾,sil1.

Case 5.2. (Otherwise).

Let cedd be a copy of edd. Delete from cedd and edd2, a copy of edd1, all occurrences of <initial>, and all <variable-reference> components of <offset>. Each subnode of edd2 MUST be equal to the corresponding subnode of cedd except that if edd2 is an

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

≤evaluated-data-description≥:
<data-description>:
<structure-data-description>:
<identifier-list>
<member-description-list>,mdl2;;;

then the <member-description-list> in cedd corresponding to mdl2 may have more components than mdl2, and the excess components are ignored.

Let dd1e be the <data-description> of edd1. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd1e) to obtain n, the number of scalar-elements corresponding to dd1e. Let sil1 be a ≤storage-index-list≥ containing a copy of the first n elements of sil.

Step 6. Return a ≤generation≥: aud eddl sil1.

7.7.4 Overlay-strings

This operation constructs a ≤storage-index-list≥ that reflects the fact that in a string-overlay defined reference or a reference to a based variable, the resulting ≤generation≥ may describe strings that start inside a ≤character-string-value≥ or ≤bit-string-value≥.

Operation: OVERLAY-STRINGS(edd,g)

Where: edd is an ≤evaluated-data-description≥,
g is a ≤generation≥.

Result: a ≤storage-index-list≥.

Step 1. Let dd be the <data-description> simply contained in edd. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd) to obtain the number of scalar-elements, n, described by edd. Perform Step 1.1 for i=1,...,n.

Step 1.1. Perform FIND-ITEM-DATA-DESCRIPTION(dd,i) to obtain the <item-data-description>,idd of the *i*th scalar-element described by dd. If idd contains <string> then let m[i] be the value of the <maximum-length> of idd; otherwise, let m[i] be the associated character-string length of the <data-type> of idd.

Step 2. Let ddg be the <data-description> simply contained in g. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(ddg) to obtain the number of scalar-elements corresponding to g, ng. Perform Step 2.1 for i=1,...,ng.

Step 2.1. Perform FIND-ITEM-DATA-DESCRIPTION(ddg,i) to obtain the <item-data-description>,idd, of the *i*th scalar element corresponding to g. If idd contains <string> then let mg[i] be the value of the <maximum-length> of idd; otherwise, let mg[i] be the associated character-string length of the <data-type> of idd.

Step 3. Let indxc be 1. Let ml be the value of the sum mg[1]+...+mg[ng]. Let sil be a ≤storage-index-list≥ with no elements. Let i be 1. Perform Steps 3.1 through 3.7 n times.

Step 3.1. Find the maximum integer j such that j<ng and such that the value of the sum mg[1]+...+mg[j] is less than indxc. Let the value of this sum be slg.

Note: j and slg may be zero.

Step 3.2. Let j be j+1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3.3. Let bvi be the ⩽basic-value-index⩾ of the *j*th element of the ⩽storage-index-list⩾ of g, and let p be the ⩽position-index⩾ of the same element.

Step 3.4. Let px be the ⩽integer-value⩾ containing the value indxc−slg−1+p.

Step 3.5. Append

```
⩽storage-index⩾:
   bvi
   ⩽position-index⩾:
      px;;
```

to sil.

Step 3.6. Let indxc be indxc+m[i]. The value of indxc MUST NOT be greater than ml+1.

Step 3.7. Let i be i+1.

Step 4. Return sil.

7.7.5 Evaluate-data-description-for-reference

This operation takes a <data-description> and a ⩽generation⩾ and constructs an ⩽evaluated-data-description⩾, using the ⩽generation⩾ to evaluate any <refer-option>.

Operation: EVALUATE-DATA-DESCRIPTION-FOR-REFERENCE(dd,g)

Where: dd is a <data-description>,
g is a ⩽generation⩾.

Result: an ⩽evaluated-data-description⩾.

Step 1. Let cdd be a copy of dd. For each <extent-expression>,ee, of cdd, taken in left-to-right order, perform Step 1.1.

Step 1.1.

Case 1.1.1. ee has a <refer-option>,ro.

Step 1.1.1.1. Let idl be the <identifier-list> of ro. Let n be the number of elements in idl. Let cdd1 be a copy of cdd and let ddp designate the <data-description>,cdd1. Let i = 1. Perform Steps 1.1.1.1.1 through 1.1.1.1.3 n times.

Step 1.1.1.1.1. Let id be the *i*th element of idl. Let sdd be the <structure-data-description> that is the immediate component of the <data-description> designated by ddp. Let idl1 be the <identifier-list> of sdd. Let the *m*th element of idl1 be identical with id.

Step 1.1.1.1.2. Let mdl1 be the <member-description-list> of sdd. Delete all elements following the *m*th element in both idl1 and mdl1. Set ddp to designate the <data-description> immediately contained in the *m*th element of mdl1.

Step 1.1.1.1.3. Set i to i+1.

Step 1.1.1.2. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(cdd1) to obtain j, the number of scalar elements corresponding to cdd1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.1.1.3. Let p be the *j*th ⟪storage-index⟫ in the ⟪storage-index-list⟫ of g, let au be the ⟪allocation-unit⟫ designated by the ⟪allocation-unit-designator⟫ of g, and let bvl be the ⟪basic-value-list⟫ of au. Let dt be the <data-type> of the <data-description> designated by ddp. Perform VALUE-OF-STORAGE-INDEX(p,dt,bvl) to obtain a ⟪basic-value⟫,bv. Convert bv to integer-type to obtain an ⟪integer-value⟫,iv, using dt as the source type for the conversion. Replace the <expression> of ee by iv.

Case 1.1.2. (Otherwise).

Perform EVALUATE-EXPRESSION-TO-INTEGER(e), where e is the <expression> of ee, to obtain the ⟪integer-value⟫,j. Replace e by j.

Step 2. Delete all <identifier-list>s which are immediate components of any <structure-data-description> or <union-data-description> in cdd.

Step 3. Return the ⟪evaluated-data-description⟫ containing cdd, as modified, as its immediate component.

7.7.6 Select-qualified-reference

This operation selects the part of a ⟪generation⟫ that corresponds to an <identifier-list>.

Operation: SELECT-QUALIFIED-REFERENCE(g,idl,d)

Where: g is a ⟪generation⟫,
idl is an <identifier-list>,
d is a <declaration>.

Result: a ⟪generation⟫.

Step 1. Let tdd be the immediate component of the <data-description> immediately contained in the <variable> of d. Let csil be a copy of the ⟪storage-index-list⟫ of g and cedd be a copy of the ⟪evaluated-data-description⟫ of g. Let redd and tedd both be the immediate component of the <data-description> of cedd.

Step 2. Let ecount be 1. Let m be the number of elements in idl. For j=1,...,m, perform Steps 2.1 through 2.9.

Step 2.1. If tdd is a <dimensioned-data-description> then perform Steps 2.1.1 and 2.1.2 .

Step 2.1.1. Let bpl be the <bound-pair-list> of tedd. Let ub[i] and lb[i] be the ⟪integer-value⟫ components of the <upper-bound> and the <lower-bound> of the *i*th <bound-pair> of bpl respectively. Let ecount be the value of the product

$$\prod_{i=1}^{k}(ub[i]-lb[i]+1)$$

where k is the number of elements of bpl.

Step 2.1.2. Let tdd and tedd be the immediate components of the <element-data-description>s of tdd and redd respectively.

Step 2.2. tdd and tedd will both be a <structure-data-description> or <union-data-description>. Let mdl and mil be, respectively, the <member-description-list> and the <identifier-list> of tdd. Let emdl be the <member-description-list> of tedd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 2.3. Let the *n*th element of mil be the one that is identical with the *j*th element of idl.

Step 2.4. Let the number of elements of mdl be nm. Let dd[k] be the <data-description> of the *k*th element of emdl. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd[k]) to obtain the number of scalar-elements nse[k] corresponding to dd[k] for $1 \leq k \leq nm$.

Step 2.5. Let i1 be the value of the sum nse[1]+...+nse[n−1], i2 be nse[n], and i3 be the value of the sum nse[n+1]+...+nse[nm].

Step 2.6. Let indx be 1+(i1+i2+i3)*(ecount−1). Perform Steps 2.6.1 through 2.6.3 ecount times.

Step 2.6.1. Delete i3 successive elements of csil starting with the element whose index is indx+i1+i2.

Step 2.6.2. Delete i1 successive elements of csil starting with the element whose index is indx.

Step 2.6.3. Set indx to indx−(i1+i2+i3).

Step 2.7. Let tdd be the immediate component of the <data-description> of the *n*th element of mdl and let tedd be the immediate component of the <data-description> of the *n*th element of emdl.

Step 2.8.

Case 2.8.1. tdd is a <dimensioned-data-description> and redd is a <dimensioned-data-description>.

Append the elements of the <bound-pair-list> of tedd to the <bound-pair-list> of redd. Replace the <element-data-description> of redd by the <element-data-description> of tedd.

Case 2.8.2. redd is a <dimensioned-data-description> and tdd is not a <dimensioned-data-description>.

Replace the <element-data-description> of redd by tedd.

Case 2.8.3. (Otherwise).

Replace redd by tedd.

Step 2.9. Let tedd be redd.

Step 3. Let retedd be an ⩽evaluated-data-description⩾: <data-description>: redd.

Step 4. Return the ⩽generation⩾ comprising a copy of the ⩽allocation-unit-designator⩾ of g, csil, and retedd.

7.7.7 Select-subscripted-reference

This operation selects the part of a ⩽generation⩾ that corresponds to a given <subscript-list>.

Operation: SELECT-SUBSCRIPTED-REFERENCE(g,sbl)

Where: g is a ⩽generation⩾,
sbl is a <subscript-list>.

Result: a ⩽generation⩾.

Step 1. Let cg be a copy of g. cg will have the form

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

⩽generation⩾:
⩽evaluated-data-description⩾:
<data-description>:
<dimensioned-data-description>:
<element-data-description>,eldd
<bound-pair-list>,bpl;;;
⩽allocation-unit-designator⩾
⩽storage-index-list⩾,sil.

Step 2. Let cbpl be a copy of bpl.

Step 3. Let dd be a <data-description> that immediately contains a copy of the immediate subtree of eldd. Perform SCALAR-ELEMENTS-OF-DATA-DESCRIPTION(dd) to obtain the integer n.

Step 4. Let m be the number of elements of sbl. Let sb[i] be the *i*th element of sbl. Let cbp[i] be the *i*th element of cbpl and let ub[i] and lb[i] be, respectively, the ⩽integer-value⩾ contained in the <upper-bound> and <lower-bound> of cbp[i]. For i=1,...,m, where the values are chosen in any order, perform Steps 4.1 and 4.2 .

Step 4.1.

Case 4.1.1. sb[i] has <asterisk>.

Let ubp be a copy of cbp[i].

Case 4.1.2. (Otherwise).

Step 4.1.2.1. If sb[i] immediately contains an <expression>,e then perform EVALUATE-EXPRESSION-TO-INTEGER(e) to obtain the ⩽integer-value⩾,v. Otherwise let v be the ⩽integer-value⩾ contained in sb[i].

Step 4.1.2.2. The inequality lb[i]≤v≤ub[i] MUST be true. Let ubp be

<bound-pair>:
<upper-bound>:
<extent-expression>:
v;;
<lower-bound>:
<extent-expression>:
v.

Step 4.2. Replace cbp[i] by ubp.

Step 5. Perform EXTRACT-SLICE-OF-ARRAY(bpl,cbpl,n,sil) to obtain a ⩽storage-index-list⩾,nsil. Replace sil by nsil.

Step 6. Let nbpl be a <bound-pair-list> with no components. For i=1,...,m, if sb[i] has <asterisk> then append a copy of the *i*th component of bpl to nbpl. If nbpl has no components then replace the <data-description> of cg by dd; otherwise replace bpl by nbpl.

Step 7. Return cg.

7.7.8 Evaluate-defined-reference

This operation evaluates a <variable-reference> for a <variable> whose <storage-type> is <defined> and yields a ⩽generation⩾.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: EVALUATE-DEFINED-REFERENCE(vr)

Where: vr is a <variable-reference>.

Result: a ⩽generation⩾.

Step 1. Let dp be the <declaration-designator> of the <variable-reference>,vr, let d be the <declaration> designated by dp, and let dc be the <defined> component of d. Perform FIND-BLOCK-STATE-OF-DECLARATION(dp) to obtain the ⩽block-state⩾,bs.

Step 2. Let dde be the ⩽defined-directory-entry⩾ in bs whose <identifier> is equal to that of d. Let edd be the ⩽evaluated-data-description⩾ of dde.

Step 3. d has a <base-item> with a <variable-reference>,bvr1. Let bvr2 be a copy of bvr1.

Step 4. Perform EVALUATE-VARIABLE-REFERENCE(bvr2) to obtain the ⩽generation⩾,g.

Step 5. Perform CHECK-SIMPLY-DEFINED-REFERENCE(vr). If the value obtained is ⩽true⩾, then perform EVALUATE-SIMPLY-DEFINED-REFERENCE(vr,edd,g) to obtain the ⩽generation⩾,gd. Otherwise perform EVALUATE-STRING-OVERLAY-DEFINED-REFERENCE(vr,edd,g) to obtain the ⩽generation⩾,gd.

Step 6. Return gd.

7.7.9 Evaluate-simply-defined-reference

This operation takes a simply-defined <variable-reference>, the ⩽evaluated-data-description⩾ associated with the variable and the ⩽generation⩾ being referenced and constructs the ⩽generation⩾ that is the result of the reference.

Operation: EVALUATE-SIMPLY-DEFINED-REFERENCE(vr,edd,g)

Where: vr is a <variable-reference>,
edd is an ⩽evaluated-data-description⩾,
g is a ⩽generation⩾.

Result: a ⩽generation⩾.

Step 1. Let eddg be the ⩽evaluated-data-description⩾ of g.

Step 2. For every <extent-expression>,e contained in edd that is not contained in a <parameter-descriptor> or <returns-descriptor>, let v be the ⩽integer-value⩾ contained in e, and perform Step 2.1.

Step 2.1. v MUST be equal to the corresponding ⩽integer-value⩾ of eddg.

Step 3. Let sil be the ⩽storage-index-list⩾ of g. Let aud be the ⩽allocation-unit-designator⩾ of g. Let g1 be a ⩽generation⩾: aud edd sil.

Step 4. If vr has an <identifier-list>,idl, then perform SELECT-QUALIFIED-REFERENCE(g1,idl,d), where d is the <declaration> designated by the <declaration-designator> of vr, to obtain a ⩽generation⩾. Replace g1 with this ⩽generation⩾.

Step 5. If vr has a <subscript-list>,sbl, perform SELECT-SUBSCRIPTED-REFERENCE(g1,sbl) to obtain a ⩽generation⩾. Replace g1 with this ⩽generation⩾.

Step 6. Return g1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7.10 Evaluate-string-overlay-defined-reference

This operation takes a <variable-reference> for a string-overlay-defined <variable>, the ⩽evaluated-data-description⩾ associated with the <variable>, and the base ⩽generation⩾ being referenced and constructs the ⩽generation⩾ that is the result of the reference.

Operation: EVALUATE-STRING-OVERLAY-DEFINED-REFERENCE(vr,edd,g)

Where: vr is a <variable-reference>,
edd is an ⩽evaluated-data-description⩾,
g is a ⩽generation⩾.

Result: a ⩽generation⩾.

Step 1. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⩽true⩾. g MUST NOT have a <union-data-description>. edd MUST NOT have a <union-data-description>. Let edd1 be the ⩽evaluated-data-description⩾ immediately contained in g. Perform TEST-STRING-CLASS(edd) to obtain sc. Perform TEST-STRING-CLASS(edd1) to obtain sc1. sc MUST be either <bit> or <character>, and MUST be equal to sc1.

Step 2. Let d be the <declaration> designated by the <declaration-designator> of vr.

Step 3. Perform OVERLAY-STRINGS(edd,g) to obtain a ⩽storage-index-list⩾,nsil. Let g1 be the ⩽generation⩾ comprising the ⩽allocation-unit-designator⩾ of g, a copy of edd, and nsil.

Step 4. If vr immediately contains an <identifier-list>,idl, then perform SELECT-QUALIFIED-REFERENCE(g1,idl,d) to obtain a ⩽generation⩾,g1.

Step 5. If vr immediately contains a <subscript-list>,sbl, then perform SELECT-SUBSCRIPTED-REFERENCE(g1,sbl) to obtain a ⩽generation⩾,g1.

Step 6. Return g1.

7.7.11 Check-simply-defined-reference

This operation checks that the relationship between the <declaration> of a <variable-reference> to a variable that is <defined> and the <declaration> referenced in the <base-item> is suitable for evaluation as a simply-defined reference.

Operation: CHECK-SIMPLY-DEFINED-REFERENCE(vr)

Where: vr is a <variable-reference>.

Result: ⩽true⩾ or ⩽false⩾.

Step 1. Let d be the <declaration> designated by the <declaration-designator> of vr. Let dd be a copy of the <data-description> immediately contained in the <variable> of d. Replace each <bound-pair> in dd by <bound-pair>: <asterisk>;, each <maximum-length> by <maximum-length>: <asterisk>;, and each <area-size> by <area-size>: <asterisk>. Let pd be

<parameter-descriptor>:
dd.

Step 2. Let bvr be the <variable-reference> of the <defined> component of dd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3. Perform TEST-MATCHING(bvr,pd). If the value obtained is ⟨true⟩ then return ⟨true⟩; otherwise return ⟨false⟩.

7.7.12 Extract-slice-of-array

This operation selects the part of a ⟨storage-index-list⟩ or ⟨basic-value-list⟩ according to a ⟨bound-pair-list⟩.

Operation: EXTRACT-SLICE-OF-ARRAY(obpl,nbpl,n,sil)

Where: obpl is a ⟨bound-pair-list⟩,
nbpl is a ⟨bound-pair-list⟩,
n is an integer,
sil is a ⟨storage-index-list⟩, or ⟨basic-value-list⟩.

Result: a ⟨storage-index-list⟩ or ⟨basic-value-list⟩.

Step 1. For each ⟨bound-pair⟩,obp[i], i=1,...,m, in obpl let oub[i] and olb[i] be the ⟨integer-value⟩ components of the ⟨upper-bound⟩ and ⟨lower-bound⟩, respectively, of obp[i].

Step 2. For each ⟨bound-pair⟩,nbp[i], i=1,...,m, in nbpl let nub[i] and nlb[i] be the ⟨integer-value⟩ components of the ⟨upper-bound⟩ and ⟨lower-bound⟩, respectively, of nbp[i].

Step 3. Let n2 be

$$n * \prod_{i=1}^{m} (nub[i]-nlb[i] + 1).$$

If sil is a ⟨storage-index-list⟩ then let nsil be a ⟨storage-index-list⟩ with n2 ⟨storage-index⟩ elements. Otherwise, let nsil be a ⟨basic-value-list⟩ with n2 elements of the form ⟨basic-value⟩: ⟨undefined⟩.

Step 4. For each set of integers s[i], i=1,...,m, such that nlb[i]≤s[i]≤nub[i], perform Steps 4.1 and 4.2.

Step 4.1. Let

$$k = 1 + n * (s[m]-olb[m] + \sum_{i=1}^{m-1} ((s[i]-olb[i]) * \prod_{j=i+1}^{m} (oub[j]-olb[j] + 1)))$$

and let

$$k2 = 1 + n * (s[m]-nlb[m] + \sum_{i=1}^{m-1} ((s[i]-nlb[i]) * \prod_{j=i+1}^{m} (nub[j]-nlb[j] + 1))).$$

Step 4.2. Replace nsil[k2+j] by sil[k+j] for j=0,...,n−1.

Step 5. Return nsil.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.8 Reference to Named Constant

7.8.1 Evaluate-named-constant-reference

This operation obtains the value of a <named-constant-reference>.

Operation: EVALUATE-NAMED-CONSTANT-REFERENCE(ncr)

Where: ncr is a <named-constant-reference>.

Result: an ≤aggregate-value≥.

Step 1. Let d be the <declaration> designated by the <declaration-designator> of ncr and let nc be a copy of the <named-constant> component of d.

Step 2.

Case 2.1. nc contains a <bound-pair>,bp.

Step 2.1.1. For each <expression>,e of bp, chosen in any order, perform EVALUATE-EXPRESSION-TO-INTEGER(e) to obtain the ≤integer-value≥,i and replace e by i.

Step 2.1.2. Let ub and lb be the ≤integer-value≥ components of the <upper-bound> and <lower-bound>, respectively, of bp.

Step 2.1.3. Let n be the integer

ub-lb+1

and let svl be a ≤basic-value-list≥ consisting of n ≤undefined≥ elements.

Case 2.2. (Otherwise).

Step 2.2.1. Let n be 1, and let svl be a ≤basic-value-list≥ consisting of one ≤undefined≥ element.

Step 3.

Case 3.1. nc has an <entry> component and d has <external>.

Let ep be the <entry-point> that is immediately contained in a <procedure> of the <external-unit-list> and that has a <statement-name> with an <identifier> that is equal to the <identifier> of d. ep MUST exist and be unique. Let epd be an <entry-point-designator> designating ep. Let ev be an ≤entry-value≥: epd;, and replace the ≤undefined≥ component of svl by ev.

Case 3.2. nc has an <entry> component and d has <internal>.

Step 3.2.1. Perform FIND-BLOCK-STATE-OF-DECLARATION(d) to obtain the ≤block-state≥,bs. Let pl be the <procedure-list> of the <begin-block> or <procedure> that simply contains d.

Step 3.2.2. Let ep be the <entry-point> that is simply contained in pl and whose <statement-name> has an <identifier> that is equal to the <identifier> of d.

Step 3.2.3. Let epd be an <entry-point-designator> designating ep and let bsd be a ≤block-state-designator≥ designating bs. Let ev be an ≤entry-value≥: epd bsd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3.2.4. Replace the ⟪undefined⟫ component of svl by ev.

Case 3.3. nc has a <file> component.

Step 3.3.1. Search the ⟪file-directory⟫ for a ⟪file-directory-entry⟫,e, whose <identifier> is identical with the <identifier> of d and which has the component <external> if d has the component <external>; otherwise it has a <declaration-designator> component that designates d.

Step 3.3.2. Let fid be the ⟪file-information-designator⟫ in e and let bv be a ⟪basic-value⟫: ⟪file-value⟫: fid.

Step 3.3.3. Replace the ⟪undefined⟫ component of svl by bv.

Case 3.4. nc has a <format> component.

Step 3.4.1. Perform FIND-BLOCK-STATE-OF-DECLARATION(d) to obtain the ⟪block-state⟫,bs. Let fsl be the <format-statement-list> component of the <begin-block> or <procedure> that simply contains d.

Step 3.4.2. Let fs be the <format-statement> element of fsl that has a <statement-name> whose <identifier> is equal to the <identifier> of d.

Step 3.4.3. Let fsd be a <format-statement-designator> designating fs and let bsd be a ⟪block-state-designator⟫ designating bs. Let fv be a ⟪format-value⟫: fsd bsd.

Step 3.4.4. Replace the ⟪undefined⟫ component of svl by fv.

Case 3.5. nc has a <label> component.

Step 3.5.1. Perform FIND-BLOCK-STATE-OF-DECLARATION(d) to obtain the ⟪block-state⟫,bs. Let eul be the <executable-unit-list> of the <begin-block> or <procedure> that simply contains d.

Step 3.5.2. For each <executable-unit>,eu, component of eul that has a <statement-name> whose <identifier> is equal to the <identifier> of d, perform Steps 3.5.2.1 and 3.5.2.2.

Step 3.5.2.1. Let eud be an <executable-unit-designator> designating eu, and let bsd be a ⟪block-state-designator⟫ designating bs. Let lv be a ⟪label-value⟫: eud bsd.

Step 3.5.2.2.

Case 3.5.2.2.1. The <statement-name> of eu has a <signed-integer>,s.

Let sn be the integer

$$1+s-lb$$

Replace the ⟪undefined⟫ component of the *sn*th element of svl by lv.

Case 3.5.2.2.2. (Otherwise).

Replace the ⟪undefined⟫ component of svl by lv.

Case 3.6. nc has an <arithmetic> immediate component or a <string> immediate component.

Replace the ⟪undefined⟫ component of svl by a copy of the ⟪basic-value⟫ of nc.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4.

Case 4.1. nc contains <bound-pair>,bp.

Let agt be

⩽aggregate-type⩾:
⩽dimensioned-aggregate-type⩾:
⩽element-aggregate-type⩾:
⩽scalar⩾;
<bound-pair-list>:
bp.

Case 4.2. (Otherwise).

Let agt be ⩽aggregate-type⩾: ⩽scalar⩾.

Step 5. Let av be ⩽aggregate-value⩾: agt svl.

Step 6. If ncr has a <subscript> then perform Steps 6.1 through 6.5.

Step 6.1. Let lb and ub be as determined in Step 2. Let bpl2 be a <bound-pair-list> with one element of the form

<bound-pair>:
<upper-bound>,ub2
<lower-bound>,lb2;

Step 6.2.

Case 6.2.1. The <subscript-list> in ncr has an <asterisk>.

Let lb2 and ub2 be, respectively, lb and ub.

Case 6.2.2. The <subscript-list> in ncr has an <expression>,exp.

Perform EVALUATE-EXPRESSION-TO-INTEGER(exp) to obtain the ⩽integer-value⩾,v and let lb2 and ub2 be the integer in v. The inequality lb≤v≤ub MUST be true.

Step 6.3. Perform EXTRACT-SLICE-OF-ARRAY(bpl,bpl2,1,svl) to obtain a ⩽basic-value-list⩾,bvl2.

Step 6.4. Replace the ⩽basic-value-list⩾ in av by a copy of bvl2.

Step 6.5.

Case 6.5.1. The <subscript-list> in ncr has <expression>.

Replace the ⩽aggregate-type⩾ of av by ⩽aggregate-type⩾: ⩽scalar⩾.

Case 6.5.2. (Otherwise).

No action.

Step 7. The ⩽aggregate-value⩾ av MUST NOT contain ⩽undefined⩾. Return av.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8. Input/Output

8.1 Introduction

This Chapter describes the abstract structure of a ⟨dataset⟩ and the transmission of data between a ⟨dataset⟩ and the ⟨allocated-storage⟩ of the ⟨machine-state⟩ directed by PL/I programs as introduced in Chapter 5. The main Sections are concerned with the following:

8.2 Datasets and the interface between them and the program
8.3 Files
8.4 Conditions applicable to I/O operations
8.5 Evaluation of a <file-option>
8.6 File opening and closing
8.7 Statements performing record transmission
8.8 Statements performing stream transmission, and a description of how data may be organized in the data stream

8.2 Datasets

A ⟨dataset⟩ is an abstract model of a physical dataset. Its properties and structure are those which are necessary for a correct interpretation of a PL/I program. The concrete-representation of a ⟨dataset⟩ is IMPLEMENTATION-DEFINED. ⟨alpha⟩ and ⟨omega⟩ are end-markers for ⟨dataset⟩s that have a sequence.

8.2.1 Record Datasets

A ⟨record-dataset⟩ may contain discrete ⟨record⟩s; ⟨record-dataset⟩s without any ⟨record⟩s are permitted.

The "size" (see Section 8.7.5.11) is an IMPLEMENTATION-DEFINED function of the ⟨evaluated-data-description⟩ of a ⟨record⟩. It is checked whenever the ⟨record⟩ is transmitted, and under IMPLEMENTATION-DEFINED circumstances may cause the raising of the <record-condition>.

⟨key⟩s are a means of identifying particular ⟨record⟩s. Within a ⟨dataset⟩, ⟨key⟩s are unique.

8.2.2 Stream Datasets

A ⟨stream-item⟩ is either a ⟨symbol⟩ or a control item which indicates (in an IMPLEMENTATION-DEFINED way) a line or page break or that the following ⟨symbol⟩s are to be sent to the same line as the preceding ones. The ⟨stream-item⟩s, ⟨pagemark⟩ and ⟨carriage-return⟩ are not allowed in a ⟨stream-item-list⟩ associated with a file open for stream input and may only appear in a ⟨stream-item-list⟩ associated with a file open for print stream output.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3 Files

Whenever a PL/I program requires access to a ≤dataset≥, it does so by naming a <file-option>. This <file-option> is evaluated yielding a ≤file-value≥,fv, which designates a ≤file-information≥,fi. In order to access a ≤dataset≥ successfully, fi MUST contain ≤open≥, in which case it also contains a ≤file-opening≥ with a ≤dataset-designator≥,dsd. dsd designates a ≤dataset≥, which is thus accessed by naming the original <file-option>.

8.3.1 Record Files

The ≤current-position≥ of a ≤file-opening≥ containing <record> contains either the ≤designator≥ of a component of a ≤record-dataset≥ or ≤undefined≥. The designated node may be a ≤record≥, a ≤keyed-record≥, ≤alpha≥, or ≤omega≥. One of the actions in executing a statement may be to update the ≤current-position≥.

The ≤delete-flag≥, which may be present in a ≤file-opening≥ when <record> appears, is an indication that certain actions MUST NOT be performed. (See, for example, the operation DELETE.)

The ≤allocated-buffer≥ in a ≤file-opening≥ contains a ≤generation≥ allocated by a <read-statement> with the <pointer-set-option>.

8.3.2 Stream Files

The ≤current-position≥ of a ≤file-opening≥ containing <stream> contains either the ≤designator≥ of a component of a ≤stream-dataset≥ or ≤undefined≥. The designated node may be a ≤stream-item≥, ≤alpha≥, or ≤omega≥. One of the actions in executing a statement may be to update the ≤current-position≥.

The ≤page-number≥ component of a ≤file-opening≥ is applicable only when the ≤file-opening≥ also contains <stream> and <print>.

The ≤first-comma≥ component of a ≤file-opening≥ is applicable only when the ≤file-opening≥ also contains <stream> and <input>.

8.4 I/O Conditions

In describing the execution of each input/output statement, the circumstances under which any <io-condition>, except <transmit-condition>, may be raised, are indicated.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.4.1 Raise-io-condition

Operation: RAISE-IO-CONDITION(cond,fv,str)

Where: cond is <endfile-condition>, <endpage-condition>, <beginpage-condition>, <key-condition>, <record-condition>, <transmit-condition>, <undefinedfile-condition>, or <conversion-condition>,
fv is a [≪file-value≫],
str is a [≪character-string-value≫].

Step 1. If fv is a ≪file-value≫ then let fi be the ≪file-information≫ designated by fv.

Step 2.

Case 2.1. cond is not <conversion-condition>.

If fv is ≪absent≫ then perform RAISE-CONDITION(<error-condition>). Let eioc be an

≪evaluated-io-condition≫:
<io-condition>:
cond;
fv.

Case 2.2. cond is <conversion-condition>.

Let eioc be <conversion-condition>.

Step 3.

Case 3.1. cond is <conversion-condition>.

Let cbifs be a ≪condition-bif-value-list≫ simply containing ≪onsource-value≫: str. If fv is a ≪file-value≫ then let fn be the ≪character-string-value≫ in the ≪filename≫ in fi and attach an ≪onfile-value≫: fn; to cbifs.

Case 3.2. cond is <transmit-condition>, <record-condition> or <key-condition>, and str is present.

Let cbifs be a ≪condition-bif-value-list≫: ≪condition-bif-value≫: ≪onkey-value≫: str.

Case 3.3. (Otherwise).

Let cbifs be ≪absent≫.

Step 4. Perform RAISE-CONDITION(eioc,cbifs).

Step 5. If cond is <key-condition> or <endfile-condition> then perform EXIT-FROM-IO(fv).

8.5 Evaluate-file-option

The evaluation of a <file-option> may be performed during input/output statements. This operation is also used to evaluate options of <on-statement>s, <signal-statement>s, and <revert-statement>s containing <named-io-condition>s.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: EVALUATE-FILE-OPTION(vr)

Where: vr is a <value-reference>.

Result: a ⟪file-value⟫.

Step 1. Perform EVALUATE-VALUE-REFERENCE(vr) to obtain an ⟪aggregate-value⟫,av. The ⟪file-value⟫ in av MUST NOT contain ⟪null-file⟫. Return the ⟪file-value⟫ in av.

8.6 File Opening and Closing

Opening a file causes a ⟪file-opening⟫ to be attached to the ⟪file-information⟫ and a ⟪dataset⟫ to be associated with the file. Closing a file removes the ⟪file-opening⟫ and dissociates the ⟪dataset⟫ from the file.

8.6.1 The Open Statement

8.6.1.1 Execute-open-statement

Operation: EXECUTE-OPEN-STATEMENT(os)

Where: os is an <open-statement>.

Step 1. For each <single-opening>,sgo in os, in order, perform EXECUTE-SINGLE-OPENING(sgo).

Step 2. Perform NORMAL-SEQUENCE.

8.6.1.2 Execute-single-opening

Operation: EXECUTE-SINGLE-OPENING(sgo)

Where: sgo is a <single-opening>.

Step 1. Let efdl be an ⟪evaluated-file-description-list⟫ with no components.

Step 2. Perform Steps 2.1 to 2.5 in any order, with the constraint that Step 2.1 be performed before Step 2.5.

Step 2.1. Let fo be the <value-reference> in the <file-option> in sgo. Perform EVALUATE-FILE-OPTION(fo) to obtain a ⟪file-value⟫,fv. Let fi be the ⟪file-information⟫ designated by fv.

Step 2.2. If sgo contains a <title-option>,tto, then perform EVALUATE-TITLE-OPTION(tto) to obtain an ⟪evaluated-title⟫,et, and append ⟪evaluated-file-description⟫: et; to efdl.

Step 2.3. If sgo contains a <linesize-option>, then let lzo be its <expression>, perform EVALUATE-EXPRESSION-TO-INTEGER(lzo) to obtain an ⟪integer-value⟫,lz, which MUST be greater than zero, and append ⟪evaluated-file-description⟫: ⟪evaluated-linesize⟫: lz;; to efdl.

Step 2.4. If sgo contains a <pagesize-option>, then let pzo be its <expression>, perform EVALUATE-EXPRESSION-TO-INTEGER(pzo) to obtain an ⟪integer-value⟫,pz, which MUST be greater than zero, and append ⟪evaluated-file-description⟫: ⟪evaluated-pagesize⟫: pz;; to efdl.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.5. If sgo contains an <environment>,envo, evaluate envo in an IMPLEMENTATION-DEFINED manner to obtain an ≪evaluated-environment≫,eenvo; otherwise, let eenvo be ≪absent≫. If fi contains an ≪evaluated-environment≫, let eenvf be that ≪evaluated-environment≫; otherwise, let eenvf be ≪absent≫. In an IMPLEMENTATION-DEFINED manner, combine eenvo and eenvf to obtain a merged ≪evaluated-environment≫,eenvm, which may be ≪absent≫. If eenvm is not ≪absent≫, append eenvm to efdl.

Step 3. If fi contains ≪open≫ then terminate this operation.

Step 4. For each immediate component, tn, of sgo which is a terminal node, append ≪evaluated-file-description≫: tn; to efdl.

Step 5. Perform OPEN(fv,efdl) to obtain res.

Step 6. If res is ≪fail≫ then perform RAISE-IO-CONDITION(<undefinedfile-condition>,fv).

8.6.1.3 Open

Operation: OPEN(fv,efdl)

Where: fv is a ≪file-value≫,
efdl is an ≪evaluated-file-description-list≫.

Result: ≪succeed≫ or ≪fail≫.

Step 1. Let fi be the ≪file-information≫ designated by fv. For each terminal node, tn, of the <file-description-set> of fi, append ≪evaluated-file-description≫: tn; to efdl.

Step 2. Augment efdl with implied attributes as follows: for each terminal node in efdl which occurs under "Attribute" in the table below, append to efdl trees containing the corresponding "Implied Attributes" categories.

Attribute	Implied Attributes	
<direct>	<record>	<keyed>
<keyed>	<record>	
<print>	<stream>	<output>
<nonprint>	<stream>	<output>
<sequential>	<record>	
<update>	<record>	

Step 3. Augment efdl with default attributes as follows: when efdl does not contain any of the "Alternative Attributes" in a line of the table below, append to efdl a tree containing the corresponding "Default" category.

Alternative Attributes	Default
<stream> | <record>	<stream>
<input> | <output> | <update>	<input>

Step 4. If efdl contains <record> but does not contain either <sequential> or <direct>, append <sequential> to efdl.

Step 5. If the ≪filename≫ in fi contains in order precisely the ≪symbol≫s of the word SYSPRINT, and the ≪file-directory-entry≫ whose ≪file-information-designator≫ designates fi has <external>, and efdl contains <stream> and <output> but not <nonprint>, then append ≪evaluated-file-description≫: <print>; to efdl.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 6. If efdl contains <stream> and <output> but neither <nonprint> nor <print> then append ⟨evaluated-file-description⟩: <nonprint>; to efdl. If efdl contains both <print> and <nonprint> then return ⟨fail⟩.

Step 7. If both <stream> and <record>, or any two of <input>, <output>, and <update>, or both <direct> and <sequential> are contained in efdl, then return ⟨fail⟩.

Step 8. If efdl contains an ⟨evaluated-linesize⟩ then it MUST contain <stream> and <output>. If efdl contains <stream> and <output> but not an ⟨evaluated-linesize⟩ then append ⟨evaluated-file-description⟩: ⟨evaluated-linesize⟩: an IMPLEMENTATION-DEFINED ⟨integer-value⟩;; to efdl.

If efdl contains an ⟨evaluated-pagesize⟩ then it MUST contain <print>. If efdl contains <print> but not an ⟨evaluated-pagesize⟩ then append ⟨evaluated-file-description⟩: ⟨evaluated-pagesize⟩: an IMPLEMENTATION-DEFINED ⟨integer-value⟩;; to efdl.

If efdl contains <print> then append ⟨evaluated-file-description⟩: ⟨evaluated-tab-option⟩: an IMPLEMENTATION-DEFINED ⟨integer-value-list⟩;; to efdl.

If efdl does not contain an ⟨evaluated-title⟩ then let fn be the ⟨filename⟩ in fi and perform EVALUATE-FILENAME(fn) to obtain an ⟨evaluated-title⟩,t, and append ⟨evaluated-file-description⟩: t; to efdl.

Step 9. Attempt to find, among the ⟨dataset⟩s of the ⟨machine-state⟩ (if any), a ⟨dataset⟩,ds, whose ⟨dataset-name⟩ has a ⟨character-string-value⟩ that matches the ⟨character-string-value⟩ of the ⟨evaluated-title⟩ in efdl in an IMPLEMENTATION-DEFINED manner. If the attempt fails, or if efdl is unacceptable to the implementation, then return ⟨fail⟩.

If efdl contains <record> then ds MUST contain a ⟨record-dataset⟩.

If efdl contains <stream> then ds MUST contain a ⟨stream-dataset⟩.

If efdl contains <sequential> and <keyed> then ds MUST contain a ⟨keyed-sequential-dataset⟩.

If efdl contains <sequential> and <record> but not <keyed> then ds MUST contain a ⟨sequential-dataset⟩.

If efdl contains <direct> then ds MUST contain a ⟨keyed-dataset⟩.

If efdl contains <stream> and <input> then ds MUST NOT contain any ⟨pagemark⟩ or ⟨carriage-return⟩.

If efdl contains <record> and <keyed> then ds MUST NOT contain two distinct ⟨keyed-record⟩s whose ⟨key⟩s are equal.

Step 10. Attach to fi a

⟨file-opening⟩,fo:
⟨dataset-designator⟩
⟨complete-file-description⟩:
efdl.

The ⟨dataset-designator⟩ designates ds, which has been associated with fv in Step 9. If efdl contains <record> then attach a ⟨delete-flag⟩ to fo. If efdl contains <print> then attach a ⟨page-number⟩ with 1 to fo. If efdl contains <stream> and <input> then attach ⟨first-comma⟩: ⟨on⟩; to fo.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 11.

Case 11.1. efdl contains <direct>.

Attach to fo a ⟨current-position⟩: ⟨undefined⟩.

Case 11.2. efdl contains <output> and not <direct>, and ds contains a ⟨record-list⟩,rl, a ⟨keyed-record-list⟩,rl, or a ⟨stream-item-list⟩,rl.

Attach to fo a ⟨current-position⟩ designating the last immediate component of rl.

Case 11.3. (Otherwise).

Attach to fo a ⟨current-position⟩ designating the ⟨alpha⟩ in ds.

Step 12. Set the ⟨open-state⟩ in fi to contain ⟨open⟩.

Step 13. Return ⟨succeed⟩.

8.6.1.4 Evaluate-title-option

Operation: EVALUATE-TITLE-OPTION(t)

Where: t is a <title-option>.

Result: an ⟨evaluated-title⟩.

Step 1. Let e be the <expression> in t. Perform EVALUATE-EXPRESSION(e) to obtain an ⟨aggregate-value⟩,av. Let dd be the <data-description> immediately contained in e.

Step 2. Let sdt be the <data-type> in dd and let tdt be a <data-type> containing <character>, <nonvarying> and an IMPLEMENTATION-DEFINED <maximum-length>. Let bv be the ⟨basic-value⟩ in av. Perform CONVERT(tdt,sdt,bv) to obtain a ⟨character-string-value⟩,csv.

Step 3. Return ⟨evaluated-title⟩: csv.

8.6.1.5 Evaluate-filename

Operation: EVALUATE-FILENAME(fn)

Where: fn is a ⟨filename⟩.

Result: an ⟨evaluated-title⟩.

Step 1. Let s be the ⟨character-string-value⟩ in fn and let m be the number of ⟨character-value⟩s in s. Let n be the IMPLEMENTATION-DEFINED maximum length of the ⟨evaluated-title⟩.

Step 2.

Case 2.1. $m = 0$.

Let csv be ⟨character-string-value⟩: ⟨null-character-string⟩.

Case 2.2. $m \geq n > 0$.

Let csv be ⟨character-string-value⟩: ⟨character-value-list⟩; where the length of ⟨character-value-list⟩ is n and where the *i*th ⟨character-value⟩ is the same as the *i*th ⟨character-value⟩ in s, i=1,...,n.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.3. m < n.

Let csv be ⟨character-string-value⟩: ⟨character-value-list⟩; where the length of ⟨character-value-list⟩ is n and where the *i*th ⟨character-value⟩ is the same as the *i*th ⟨character-value⟩ in s for $1 \leq i \leq m$, and where the remaining ⟨character-value⟩s have ⌀s.

Step 3. Return ⟨evaluated-title⟩: csv.

8.6.2 The Close Statement

8.6.2.1 Execute-close-statement

Operation: EXECUTE-CLOSE-STATEMENT(cs)

Where: cs is a <close-statement>.

Step 1. For each <single-closing>,sc in cs, in order, perform EXECUTE-SINGLE-CLOSING(sc).

Step 2. Perform NORMAL-SEQUENCE.

8.6.2.2 Execute-single-closing

Operation: EXECUTE-SINGLE-CLOSING(sc)

Where: sc is a <single-closing>.

Step 1. Let fo be the <value-reference> in the <file-option> in sc. Perform EVALUATE-FILE-OPTION(fo) to obtain a ⟨file-value⟩,fv.

Step 2.

Case 2.1. sc contains an <environment>,envo.

Step 2.1.1. Evaluate envo in an IMPLEMENTATION-DEFINED manner to obtain an ⟨evaluated-environment⟩,eenvo.

Step 2.1.2. If the ⟨file-information⟩ designated by fv contains ⟨open⟩ then perform CLOSE(fv,eenvo).

Case 2.2. sc does not contain an <environment>.

If the ⟨file-information⟩ designated by fv contains ⟨open⟩ then perform CLOSE(fv).

8.6.2.3 Close

Operation: CLOSE(fv,eenvo)

Where: fv is a ⟨file-value⟩,
eenvo is an [⟨evaluated-environment⟩].

Step 1. Let fi be the ⟨file-information⟩ designated by fv, and let ds be the ⟨dataset⟩ designated by the ⟨dataset-designator⟩ in fi.

Step 2. If abuf is present then perform FREE(g) and delete abuf.

Step 3. Delete the ⟨file-opening⟩ in fi, and set the ⟨open-state⟩ in fi to contain ⟨closed⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7 The Record I/O Statements

The record I/O statements perform data transmission to and from ⟨record-dataset⟩s. Several of the record I/O statements use common operations. These are described in Section 8.7.5. Several local variables are used in 8.7 in a consistent manner:

abuf	for ⟨allocated-buffer⟩
d	for <declaration>
ds	for ⟨record-dataset⟩
edd	for ⟨evaluated-data-description⟩
eio	for ⟨evaluated-into-option⟩
eko	for ⟨evaluated-keyto-option⟩
epso	for ⟨evaluated-pointer-set-option⟩
efdl	for ⟨evaluated-file-description-list⟩
eso	for ⟨evaluated-sizeto-option⟩
fi	for ⟨file-information⟩
fn	for ⟨filename⟩
fv	for ⟨file-value⟩
g	for ⟨generation⟩
int	for ⟨integer-value⟩
k	for ⟨key⟩
kr	for ⟨keyed-record⟩ or ⟨record⟩
pos	for ⟨current-position⟩
r	for ⟨record⟩

8.7.1 The Read Statement

The <read-statement> causes a ⟨record⟩ to be transmitted from a ⟨record-dataset⟩ to a target ⟨generation⟩ or an ⟨allocated-buffer⟩.

8.7.1.1 Execute-read-statement

⟨evaluated-read-statement⟩ ::= ⟨file-value⟩ {⟨evaluated-into-option⟩ |
⟨evaluated-pointer-set-option⟩ [⟨evaluated-sizeto-option⟩]}
[⟨key⟩ | ⟨evaluated-keyto-option⟩] [⟨evaluated-environment⟩]

Operation: EXECUTE-READ-STATEMENT(rs)

Where: rs is a <read-statement>.

Step 1. Let ers be an ⟨evaluated-read-statement⟩ without subnodes.

Step 2. Perform Steps 2.1 through 2.7 in any order.

Step 2.1. Let f be the immediate component of the <file-option> in rs. Perform EVALUATE-FILE-OPTION(f) to obtain a ⟨file-value⟩,fv. Attach fv to ers.

Step 2.2. If rs contains an <into-option>,ito, perform EVALUATE-INTO-OPTION(ito) to obtain an ⟨evaluated-into-option⟩,eito and attach eito to ers.

Step 2.3. If rs contains a <pointer-set-option>,pso, perform EVALUATE-POINTER-SET-OPTION(pso) to obtain an ⟨evaluated-pointer-set-option⟩,epso and attach epso to ers.

Step 2.4. If rs contains a <sizeto-option>,so, perform EVALUATE-SIZETO-OPTION(so) to obtain an ⟨evaluated-sizeto-option⟩,eso and attach eso to ers.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.5. If rs contains a <key-option>,ko, perform EVALUATE-KEY-OPTION(ko) to obtain a ⟪key⟫,k and attach k to ers.

Step 2.6. If rs contains a <keyto-option>,kto, perform EVALUATE-KEYTO-OPTION(kto) to obtain an ⟪evaluated-keyto-option⟫,ekto and attach ekto to ers.

Step 2.7. If rs contains an <environment>,envo, evaluate envo in an IMPLEMENTATION-DEFINED manner to obtain an ⟪evaluated-environment⟫,eenvo and attach eenvo to ers.

Step 3. If the ⟪file-information⟫,fi, designated by fv contains ⟪open⟫ then go to Step 5 .

Step 4.

Step 4.1. Let efdl be an ⟪evaluated-file-description-list⟫ containing <record>.

Step 4.2. If fi does not contain <update> then attach ⟪evaluated-file-description⟫: <input>; to efdl.

Step 4.3. Perform OPEN(fv,efdl) to obtain rf. If rf is ⟪fail⟫ then perform RAISE-IO-CONDITION(<undefinedfile-condition>,fv). If, on normal return, fi contains ⟪closed⟫ then perform RAISE-CONDITION(<error-condition>).

Step 5. fi MUST contain:

<record>;
<input> or <update>;
if ers does not contain a ⟪key⟫, then <sequential>;
if ers contains a ⟪key⟫ or an ⟪evaluated-keyto-option⟫, then <keyed>.

Step 6. Perform READ(ers).

Step 7. Perform NORMAL-SEQUENCE.

8.7.1.2 Read

Operation: READ(ers)

Where: ers is an ⟪evaluated-read-statement⟫.

Step 1. Let fi be the ⟪file-information⟫ designated by the ⟪file-value⟫,fv, in ers. fi MUST contain ⟪open⟫. If ers contains a ⟪key⟫, then let k be this ⟪key⟫ and let k1 be the immediate component of k. Otherwise let k and k1 be ⟪absent⟫.

Step 2. If fi contains an ⟪allocated-buffer⟫,abuf, then perform FREE(g), where g is the ⟪generation⟫ in abuf, and delete abuf from fi.

Step 3. Perform POSITION-FILE(ers).

Step 4. If ers does not contain an ⟪evaluated-keyto-option⟫,eko, then go to Step 5. Otherwise let ns be the number of {symbol} subnodes of the ⟪character-string-value⟫,csvk in the ⟪key⟫ in the ⟪keyed-record⟫ designated by the ⟪current-position⟫ in fi. Let dd be a <data-description> containing <character>, <maximum-length> containing ns, and <nonvarying>. Let avk be an ⟪aggregate-value⟫ containing csvk. Let ekoet be the ⟪evaluated-target⟫ in eko.

Step 5.

Case 5.1. ers contains an ⟪evaluated-into-option⟫,eio.

Let r be the ⟪record⟫ or ⟪keyed-record⟫ designated by the ⟪current-position⟫ in fi.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 5.1.1. Let edd1 be the ⟨evaluated-data-description⟩ in r and edd2 be the ⟨evaluated-data-description⟩ in eio. Perform EVALUATE-SIZE(edd1) to obtain an ⟨integer-value⟩,int1, and perform EVALUATE-SIZE(edd2) to obtain an ⟨integer-value⟩,int2.

Step 5.1.2. Let g be the ⟨generation⟩ in eio.

Case 5.1.2.1. int1 is equal to int2.

Perform Steps 5.1.2.1.1 and 5.1.2.1.2 in either order.

Step 5.1.2.1.1. edd1 and edd2 MUST be equal. Perform SET-STORAGE(g,v), where v is the ⟨basic-value-list⟩ in r.

Step 5.1.2.1.2. If ers contains an ⟨evaluated-keyto-option⟩ then perform ASSIGN(ekoet,avk,dd).

Case 5.1.2.2. (Otherwise).

Step 5.1.2.2.1. Let nsi be the number of ⟨storage-index⟩s in g. Let undef be a ⟨basic-value-list⟩ containing ⟨basic-value⟩: ⟨undefined⟩; nsi times. Perform Steps 5.1.2.2.1.1 and 5.1.2.2.1.2 in either order.

Step 5.1.2.2.1.1. Perform SET-STORAGE(g,undef).

Step 5.1.2.2.1.2. If ers contains an ⟨evaluated-keyto-option⟩ then perform ASSIGN(ekoet,avk,dd).

Step 5.1.2.2.2. Perform RAISE-IO-CONDITION(<record-condition>,fv,k1).

Case 5.2. ers has an ⟨evaluated-pointer-set-option⟩,epso.

Perform Steps 5.2.1 through 5.2.6 in any order such that Step 5.2.1 precedes Step 5.2.2, Step 5.2.3, Step 5.2.4, and Step 5.2.5.

Step 5.2.1. Let r be the ⟨record⟩ or ⟨keyed-record⟩ designated by the ⟨current-position⟩ in fi. Let edd be the ⟨evaluated-data-description⟩ in r. Perform ALLOCATE(edd) to obtain a ⟨generation⟩,g.

Step 5.2.2. Let abuf be an ⟨allocated-buffer⟩: g. Attach abuf to the ⟨file-opening⟩ in fi.

Step 5.2.3. Let v be the ⟨basic-value-list⟩ in r. Perform SET-STORAGE(g,v).

Step 5.2.4. Let pdd be a <data-description> simply containing <pointer> and no other terminal subnodes. Let epsoet be an ⟨evaluated-target⟩ containing the ⟨generation⟩ contained in epso. Let agv be an ⟨aggregate-value⟩ containing the ⟨pointer-value⟩: g. Perform ASSIGN(epsoet,agv,pdd).

Step 5.2.5. If ers contains an ⟨evaluated-sizeto-option⟩,eso, then let avs be an ⟨aggregate-value⟩ containing an IMPLEMENTATION-DEFINED ⟨real-value⟩, depending of edd. avs should be the same value which would be returned if CURRENTSIZE-BIF were invoked with a <variable-reference> identifying the ⟨generation⟩ g. Let sdd be a <data-description> simply containing a <data-type>, which is integer-type, and no other terminal subnodes. Let esoet be the ⟨evaluated-target⟩ in eso. Perform ASSIGN(esoet,avs,sdd).

Step 5.2.6. If ers contains an ⟨evaluated-keyto-option⟩ then perform ASSIGN(ekoet,avk,dd).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 5.3. (Otherwise).

If ers contains an ⩽evaluated-keyto-option⩾ then perform ASSIGN(ekoet,avk,dd).

8.7.2 The Write Statement

The <write-statement> causes a ⩽record⩾ or ⩽keyed-record⩾ to be transmitted from a ⩽generation⩾ to a ⩽record-dataset⩾.

8.7.2.1 Execute-write-statement

⩽evaluated-write-statement⩾ ::= ⩽file-value⩾ ⩽evaluated-from-option⩾ [⩽evaluated-keyfrom-option⩾] [⩽evaluated-environment⩾]

Operation: EXECUTE-WRITE-STATEMENT(ws)

Where: ws is a <write-statement>.

Step 1. Let ews be an ⩽evaluated-write-statement⩾ without subnodes.

Step 2. Perform Steps 2.1 through 2.4 in any order.

Step 2.1. Let f be the immediate component of the <file-option> in ws. Perform EVALUATE-FILE-OPTION(f) to obtain a ⩽file-value⩾,fv. Attach fv to ews.

Step 2.2. Let fr be the <from-option> in ws. Perform EVALUATE-FROM-OPTION(fr) to obtain an ⩽evaluated-from-option⩾,efo and attach efo to ews.

Step 2.3. If ws contains a <keyfrom-option>,ko, then perform EVALUATE-KEYFROM-OPTION(ko) to obtain an ⩽evaluated-keyfrom-option⩾,ekfo and attach ekfo to ews.

Step 2.4. If ws contains an <environment>,envo, evaluate envo in an IMPLEMENTATION-DEFINED manner to obtain an ⩽evaluated-environment⩾,eenvo and attach eenvo to ews.

Step 3. If the ⩽file-information⩾,fi, designated by fv contains ⩽open⩾ then go to Step 5.

Step 4.

Step 4.1. Let efdl be an ⩽evaluated-file-description-list⩾ containing <record>.

Step 4.2. If fi does not contain <update> then attach ⩽evaluated-file-description⩾: <output>; to efdl.

Step 4.3. Perform OPEN(fv,efdl) to obtain sf. If sf is ⩽fail⩾ then perform RAISE-IO-CONDITION(<undefinedfile-condition>,fv). If on normal return fi contains ⩽closed⩾ then perform RAISE-CONDITION(<error-condition>).

Step 5. fi MUST contain <record>.

Case 5.1. fi contains <output>.

If and only if ews contains an ⩽evaluated-keyfrom-option⩾ then fi MUST contain <keyed>.

Case 5.2. fi contains <update>.

fi MUST contain <keyed> and ews MUST contain an ⩽evaluated-keyfrom-option⩾.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 5.3. (Otherwise).

This case MUST NOT occur.

Step 6. Perform WRITE(ews).

Step 7. Perform NORMAL-SEQUENCE.

8.7.2.2 Write

Operation: WRITE(ews)

Where: ews is an ⟨evaluated-write-statement⟩.

Step 1. Let fi be the ⟨file-information⟩ designated by the ⟨file-value⟩,fv, in ews. fi MUST contain ⟨open⟩. If ews contains an ⟨evaluated-keyfrom-option⟩: ⟨character-string-value⟩,csv; then let k be a ⟨key⟩: csv. Otherwise let k be ⟨absent⟩. Let ds be the ⟨record-dataset⟩ designated by the ⟨dataset-designator⟩ in fi.

Step 2. If fi contains an ⟨allocated-buffer⟩,abuf, containing a ⟨generation⟩,g, then perform FREE(g) and delete abuf from fi.

Step 3. Let g be the ⟨generation⟩ in the ⟨evaluated-from-option⟩ in ews. Perform CONSTRUCT-RECORD(g,k) to obtain kr.

Step 4. If fi contains <keyed> then if k is equal to any ⟨key⟩ in ds or if k is unacceptable to the implementation then perform RAISE-IO-CONDITION(<key-condition>,fv,csv).

Step 5. Perform INSERT-RECORD(kr,fv) to obtain a ⟨designator⟩,pos.

Step 6. Replace the immediate component of the ⟨current-position⟩ in fi by pos.

8.7.3 The Rewrite Statement

The <rewrite-statement> causes replacement of an existing ⟨record⟩ or ⟨keyed-record⟩ in a ⟨record-dataset⟩.

8.7.3.1 Execute-rewrite-statement

⟨evaluated-rewrite-statement⟩ ::= ⟨file-value⟩ [[⟨key⟩] ⟨evaluated-from-option⟩]
[⟨evaluated-environment⟩]

Operation: EXECUTE-REWRITE-STATEMENT(rws)

Where: rws is a <rewrite-statement>.

Step 1. Let erws be an ⟨evaluated-rewrite-statement⟩ without subnodes.

Step 2. Perform Steps 2.1 through 2.4 in any order.

Step 2.1. Let f be the immediate component of the <file-option> in rws. Perform EVALUATE-FILE-OPTION(f) to obtain a ⟨file-value⟩,fv. Attach fv to erws.

Step 2.2. If rws contains a <from-option>,fr, then perform EVALUATE-FROM-OPTION(fr) to obtain an ⟨evaluated-from-option⟩,efo and attach efo to erws.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.3. If rws contains a <key-option>,ko, then perform EVALUATE-KEY-OPTION(ko) to obtain a ⩽key⩾,k and attach k to erws.

Step 2.4. If rws contains a <environment>,envo, evaluate envo in an IMPLEMENTATION-DEFINED manner to obtain an ⩽evaluated-environment⩾,eenvo and attach eenvo to erws.

Step 3. If the ⩽file-information⩾,fi, designated by fv contains ⩽open⩾ then go to step 5.

Step 4.

Step 4.1. Let efdl be an ⩽evaluated-file-description-list⩾ containing <record> and <update>.

Step 4.2. Perform OPEN(fv,efdl) to obtain sf. If sf is ⩽fail⩾ then perform RAISE-IO-CONDITION(<undefinedfile-condition>,fv). If on normal return fi contains ⩽closed⩾ then perform RAISE-CONDITION(<error-condition>).

Step 5. fi MUST contain:

<record>;
<update>;
if erws does not contain a ⩽key⩾, then <sequential>;
if erws contains a ⩽key⩾, then <keyed>.

Step 6. Perform REWRITE(erws).

Step 7. Perform NORMAL-SEQUENCE.

8.7.3.2 Rewrite

Operation: REWRITE(erws)

Where: erws is an ⩽evaluated-rewrite-statement⩾.

Step 1. Let fi be the ⩽file-information⩾ designated by the ⩽file-value⩾,fv in erws. fi MUST contain ⩽open⩾. If erws contains a ⩽key⩾, then let k be this ⩽key⩾; otherwise let k be ⩽absent⩾. If k is a ⩽key⩾ then let csv be its immediate component; otherwise let csv be ⩽absent⩾.

Step 2.

Case 2.1. k is ⩽absent⩾.

fi MUST NOT contain a ⩽delete-flag⩾ and the ⩽current-position⩾ in fi MUST not contain ⩽undefined⩾. The ⩽current-position⩾ in fi MUST NOT designate an ⩽alpha⩾ or ⩽omega⩾. If the ⩽current-position⩾ in fi designates a ⩽keyed-record⩾,kr then let k be a copy of the ⩽key⩾ in kr and let csv be the immediate component of k.

Case 2.2. k is a ⩽key⩾.

Perform POSITION-FILE(erws).

Step 3.

Case 3.1. erws contains an ⩽evaluated-from-option⩾,efo.

Let g be the ⩽generation⩾ in efo.

Case 3.2. (Otherwise).

fi MUST contain an ⩽allocated-buffer⩾,abuf. Let g be the ⩽generation⩾ in abuf.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4. Perform CONSTRUCT-RECORD(g,k) to obtain r.

Step 5. Let edd1 be the ⟨evaluated-data-description⟩ in g and edd2 the ⟨evaluated-data-description⟩ in the ⟨record⟩ designated by the ⟨current-position⟩ in fi. Perform EVALUATE-SIZE(edd1) to obtain an ⟨integer-value⟩,int1 and EVALUATE-SIZE(edd2) to obtain an ⟨integer-value⟩,int2.

Step 6. Let rd be the ⟨record⟩ or ⟨keyed-record⟩ designated by the ⟨current-position⟩ in fi.

Case 6.1. int1 and int2 are equal.

Replace rd by r.

Case 6.2. int1 and int2 are not equal.

Step 6.2.1. In either order, perform Steps 6.2.1.1 and 6.2.1.2.

Step 6.2.1.1. Perform RAISE-IO-CONDITION(<record-condition>,fv,csv).

Step 6.2.1.2. If rd is a ⟨record⟩, replace rd by an IMPLEMENTATION-DEFINED ⟨record⟩; otherwise replace rd by an IMPLEMENTATION-DEFINED ⟨keyed-record⟩ with an equal ⟨key⟩.

Note: An implementation is expected to perform the steps in the order given if possible.

Step 6.2.2. Optionally perform EXIT-FROM-IO(fv).

Step 7. If fi contains an ⟨allocated-buffer⟩,abuf, then let g be the ⟨generation⟩ in abuf, perform FREE(g), and delete abuf from fi.

8.7.4 The Delete Statement

The <delete-statement> deletes a ⟨record⟩ or ⟨keyed-record⟩ from a ⟨record-dataset⟩.

8.7.4.1 Execute-delete-statement

⟨evaluated-delete-statement⟩ ::= ⟨file-value⟩ [⟨key⟩] [⟨evaluated-environment⟩]

Operation: EXECUTE-DELETE-STATEMENT(dls)

Where: dls is a <delete-statement>.

Step 1. Let eds be an ⟨evaluated-delete-statement⟩ without subnodes.

Step 2. Perform Steps 2.1 through 2.3 in any order.

Step 2.1. Let f be the immediate component of the <file-option> in dls. Perform EVALUATE-FILE-OPTION(f) to obtain a ⟨file-value⟩,fv. Attach fv to eds.

Step 2.2. If dls contains a <key-option>,ko, then perform EVALUATE-KEY-OPTION(ko) to obtain a ⟨key⟩,k and attach k to eds.

Step 2.3. If dls contains an <environment>,envo, then evaluate envo in an IMPLEMENTATION-DEFINED manner to obtain an ⟨evaluated-environment⟩,eenvo and attach eenvo to eds.

Step 3. If the ⟨file-information⟩,fi, designated by fv contains ⟨open⟩ then go to Step 5.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4.

Step 4.1. Let efdl be an ⟨evaluated-file-description-list⟩ containing <update> and <record>.

Step 4.2. Perform OPEN(fv,efdl) to obtain sf. If sf is ⟨fail⟩ then perform RAISE-IO-CONDITION(<undefinedfile-condition>,fv). If on normal return fi contains ⟨closed⟩ then perform RAISE-CONDITION(<error-condition>).

Step 5. fi MUST contain:

<record>;
<update>;
if eds contains a ⟨key⟩, then <keyed>;
if eds does not contain a ⟨key⟩, then <sequential>.

Step 6. Perform DELETE(eds).

Step 7. Perform NORMAL-SEQUENCE.

8.7.4.2 Delete

Operation: DELETE(eds)

Where: eds is an ⟨evaluated-delete-statement⟩.

Step 1. Let fi be the ⟨file-information⟩ designated by the ⟨file-value⟩,fv, in eds. fi MUST contain ⟨open⟩. If eds contains a ⟨key⟩, then let k be this ⟨key⟩ and let csv be the immediate component of k; otherwise let k and csv be ⟨absent⟩.

Step 2.

Case 2.1. k is ⟨absent⟩.

fi MUST NOT contain a ⟨delete-flag⟩. The ⟨current-position⟩ in fi MUST NOT contain ⟨undefined⟩. The ⟨current-position⟩ in fi MUST NOT designate an ⟨alpha⟩ or ⟨omega⟩.

Case 2.2. k is a ⟨key⟩.

Perform POSITION-FILE(eds).

Step 3. Let ds be the immediate component of the ⟨record-dataset⟩ designated by the ⟨dataset-designator⟩ in fi. Let kr be the node designated by the ⟨current-position⟩ in fi (kr is a ⟨record⟩ or a ⟨keyed-record⟩). Delete kr from ds.

Step 4.

Case 4.1. fi contains <direct>.

Replace the immediate subnode of the ⟨current-position⟩ in fi by ⟨undefined⟩.

Case 4.2. fi contains <sequential>.

Replace the immediate subnode of the ⟨current-position⟩ in fi by a ⟨designator⟩ designating the predecessor of kr in ds (this may be ⟨alpha⟩, a ⟨record⟩ or a ⟨keyed-record⟩).

Step 5. Attach a ⟨delete-flag⟩ to the ⟨file-opening⟩ in fi.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 6. If fi contains an ⟪allocated-buffer⟫,abuf, then let g be the ⟪generation⟫ in abuf, perform FREE(g), and delete abuf from fi.

8.7.5 Operations Applicable to Record I/O

8.7.5.1 Evaluate-from-option

⟪evaluated-from-option⟫ ::= ⟪generation⟫

Operation: EVALUATE-FROM-OPTION(fr)

Where: fr is a <from-option>.

Result: an ⟪evaluated-from-option⟫.

Step 1. Let v be the <variable-reference> immediately contained in fr. Perform EVALUATE-VARIABLE-REFERENCE(v) to obtain a ⟪generation⟫,g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⟪true⟫.

Step 2. Return an ⟪evaluated-from-option⟫: g.

8.7.5.2 Evaluate-into-option

⟪evaluated-into-option⟫ ::= ⟪generation⟫

Operation: EVALUATE-INTO-OPTION(ito)

Where: ito is an <into-option>.

Result: an ⟪evaluated-into-option⟫.

Step 1. Let v be the <variable-reference> immediately contained in ito. Perform EVALUATE-VARIABLE-REFERENCE(v) to obtain a ⟪generation⟫,g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⟪true⟫.

Step 2. Return an ⟪evaluated-into-option⟫: g.

8.7.5.3 Evaluate-pointer-set-option

⟪evaluated-pointer-set-option⟫ ::= ⟪generation⟫

Operation: EVALUATE-POINTER-SET-OPTION(pso)

Where: pso is a <pointer-set-option>.

Result: an ⟪evaluated-pointer-set-option⟫.

Step 1. Let v be the <variable-reference> immediately contained in pso. Perform EVALUATE-VARIABLE-REFERENCE(v) to obtain a ⟪generation⟫,g.

Step 2. Return an ⟪evaluated-pointer-set-option⟫: g.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.5.4 Evaluate-key-option

Operation: EVALUATE-KEY-OPTION(ko)

Where: ko is a <key-option>.

Result: a ⩽key⩾.

Step 1. Let e be the <expression> immediately contained in ko. Perform EVALUATE-EXPRESSION(e) to obtain res, containing a ⩽basic-value⩾,bv.

Step 2. Let tt be a <data-type> containing <character>, <nonvarying>, and <maximum-length>: <asterisk>. Let rt be the <data-type> of e. Perform CONVERT(tt,rt,bv) to obtain a ⩽character-string-value⩾,chs.

Step 3. Return a ⩽key⩾: chs.

8.7.5.5 Evaluate-keyfrom-option

⩽evaluated-keyfrom-option⩾ ::= ⩽character-string-value⩾

Operation: EVALUATE-KEYFROM-OPTION(kfo)

Where: kfo is a <keyfrom-option>.

Result: an ⩽evaluated-keyfrom-option⩾.

Step 1. Let e be the <expression> immediately contained in kfo. Perform EVALUATE-EXPRESSION(e) to obtain res, containing a ⩽basic-value⩾,bv.

Step 2. Let tt be a <data-type> containing <character>, <nonvarying>, and <maximum-length>: <asterisk>. Let rt be the <data-type> of e. Perform CONVERT(tt,rt,bv) to obtain a ⩽character-string-value⩾,chs.

Step 3. Return an ⩽evaluated-keyfrom-option⩾: chs.

8.7.5.6 Evaluate-keyto-option

⩽evaluated-keyto-option⩾ ::= ⩽evaluated-target⩾

Operation: EVALUATE-KEYTO-OPTION(kto)

Where: kto is a <keyto-option>.

Result: an ⩽evaluated-keyto-option⩾.

Step 1. Let ktotr be the <variable-reference> in kto. Perform EVALUATE-TARGET-REFERENCE(ktotr) to obtain an ⩽evaluated-target⩾,et.

Step 2. Return an ⩽evaluated-keyto-option⩾: et.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.5.7 Evaluate-sizeto-option

⟨evaluated-sizeto-option⟩ ::= ⟨evaluated-target⟩

Operation: EVALUATE-SIZETO-OPTION(so)

Where: so is a <sizeto-option>.

Result: an ⟨evaluated-sizeto-option⟩.

Step 1. Let sotr be the <variable-reference> in so. Perform EVALUATE-TARGET-REFERENCE(sotr) to obtain an ⟨evaluated-target⟩,et.

Step 2. Return an ⟨evaluated-sizeto-option⟩: et.

8.7.5.8 Construct-record

Operation: CONSTRUCT-RECORD(g,k)

Where: g is a connected ⟨generation⟩,
k is a [⟨key⟩].

Result: a ⟨record⟩ or a ⟨keyed-record⟩.

Step 1. Let agv be an ⟨aggregate-value⟩ which is the value of g. (See Section 7.2.3) Let edd be the ⟨evaluated-data-description⟩ in g. Let bvl be the ⟨basic-value-list⟩ in agv. Let r be a ⟨record⟩: edd bvl.

Step 2.

Case 2.1. k is ⟨absent⟩.

Return r.

Case 2.2. k is a ⟨key⟩.

Return ⟨keyed-record⟩: r k.

8.7.5.9 Insert-record

Operation: INSERT-RECORD(kr,fv)

Where: kr is a ⟨record⟩ or a ⟨keyed-record⟩,
fv is a ⟨file-value⟩.

Result: a ⟨designator⟩.

Step 1. If kr is a ⟨keyed-record⟩ then let k be the ⟨key⟩ in kr and let csv be the immediate component of k; otherwise let k and csv be ⟨absent⟩. Let fi be the ⟨file-information⟩ designated by fv. Let rl be the ⟨record-list⟩ or ⟨keyed-record-list⟩ in the ⟨dataset⟩ designated by the ⟨dataset-designator⟩ in fi. Let edd be the ⟨evaluated-data-description⟩ in kr. Perform EVALUATE-SIZE(edd) to obtain an ⟨integer-value⟩,int. If int is not acceptable to the implementation then perform Step 1.1.

Step 1.1. If kr is a ⟨record⟩, let kr be a new IMPLEMENTATION-DEFINED ⟨record⟩; otherwise let kr be an IMPLEMENTATION-DEFINED ⟨keyed-record⟩ with an equal ⟨key⟩. Optionally perform

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2. Perform RAISE-IO-CONDITION(<record-condition>,fv,csv). Optionally perform EXIT-FROM-IO(fv).

Step 2.

Case 2.1. fi contains <direct>.

Attach kr to rl in a position chosen in an IMPLEMENTATION-DEFINED way.

Case 2.2. fi contains <keyed> and <sequential>.

Attach kr to rl in an IMPLEMENTATION-DEFINED position which may depend on the ⟨key⟩ in kr, on the ⟨record⟩ in kr and the ⟨current-position⟩ in fi.

Case 2.3. fi contains <sequential> and not <keyed>.

Append kr to rl in the position immediately following the immediate component of rl designated by the ⟨current-position⟩ in fi.

Step 3. Return a ⟨designator⟩ designating kr in rl.

8.7.5.10 Position-file

Operation: POSITION-FILE(evst)

Where: evst is an ⟨evaluated-read-statement⟩, an ⟨evaluated-rewrite-statement⟩ or an ⟨evaluated-delete-statement⟩.

Step 1. If evst contains a ⟨key⟩ then let k be this ⟨key⟩ and let csv be its immediate component; otherwise let k and csv both be ⟨absent⟩. Let fv be the ⟨file-value⟩ in evst, let fi be the ⟨file-information⟩ designated by fv, let ds be the ⟨record-dataset⟩ in the ⟨dataset⟩ designated by the ⟨dataset-designator⟩ in fi, and let pos be the ⟨current-position⟩ in fi.

Step 2.

Step 2.1. If k is a ⟨key⟩ and k is unacceptable to the implementation then perform RAISE-IO-CONDITION(<key-condition>,fv,csv).

Step 2.2.

Case 2.2.1. k is ⟨absent⟩.

Go to Step 3.

Case 2.2.2. k is a ⟨key⟩ and k is not equal to any ⟨key⟩ in ds.

Step 2.2.2.1. Replace the immediate component of pos by ⟨undefined⟩.

Step 2.2.2.2. Perform RAISE-IO-CONDITION(<key-condition>,fv,csv).

Case 2.2.3. k is equal to a ⟨key⟩ in a ⟨keyed-record⟩,kr, in ds.

Step 2.2.3.1. If fi contains a ⟨delete-flag⟩,dfl, then delete dfl from fi.

Step 2.2.3.2. Replace the immediate component of pos by a ⟨designator⟩ designating kr.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3.

Case 3.1. evst is an ⟨evaluated-read-statement⟩ not containing a ⟨key⟩.

Let int be an ⟨integer-value⟩ with value 1.

Case 3.2. (Otherwise).

Terminate this operation.

Step 4. pos MUST contain a ⟨designator⟩,rdes.

Step 5.

Case 5.1. rdes designates ⟨omega⟩.

Perform RAISE-IO-CONDITION(⟨endfile-condition⟩,fv)

Case 5.2. rdes designates ⟨alpha⟩.

Replace rdes by a ⟨designator⟩ designating the first ⟨keyed-record⟩ (or ⟨record⟩) in ds if there is one; otherwise, replace rdes by a ⟨designator⟩ designating the ⟨omega⟩ in ds.

Case 5.3. rdes designates the last element of the ⟨keyed-record-list⟩ (or ⟨record-list⟩) in ds.

Replace rdes by a ⟨designator⟩ designating the ⟨omega⟩ in ds.

Case 5.4. (Otherwise).

Replace rdes by a ⟨designator⟩ designating the next ⟨keyed-record⟩ (or ⟨record⟩) in ds.

Step 6. If rdes designates ⟨omega⟩ then perform RAISE-IO-CONDITION(⟨endfile-condition⟩,fv).

Step 7. If fi contains a ⟨delete-flag⟩ then delete it from fi.

8.7.5.11 Evaluate-size

Operation: EVALUATE-SIZE(edd)

Where: edd is an ⟨evaluated-data-description⟩.

Result: an ⟨integer-value⟩.

Step 1. Return an IMPLEMENTATION-DEFINED ⟨integer-value⟩, depending on edd.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.7.5.12 Exit-from-io

Operation: EXIT-FROM-IO(fv)

Where: fv is a ⟨file-value⟩.

Step 1. Let fi be the ⟨file-information⟩ designated by fv. If fi contains an ⟨allocated-buffer⟩,abuf, containing a ⟨generation⟩,g, then perform FREE(g) and delete abuf from fi. If fi contains a ⟨page-top-flag⟩ then delete the ⟨page-top-flag⟩ from fi.

Step 2. Perform TRIM-IO-CONTROL.

Step 3. Let eud be the current ⟨executable-unit-designator⟩. Perform TRIM-GROUP-CONTROL(eud).

Step 4. Replace the immediate component of the current ⟨statement-control⟩ by

⟨operation-list⟩:
⟨operation⟩ for ADVANCE-EXECUTION
⟨operation⟩ for NORMAL-SEQUENCE.

8.7.5.13 Trim-io-control

Operation: TRIM-IO-CONTROL

Step 1. Let bc be the current ⟨block-control⟩.

Step 2. If bc contains a ⟨data-item-control-list⟩,dicl, then delete dicl from bc.

Step 3. If bc contains a ⟨current-scalar-item-list⟩,csil, then delete csil from bc.

Step 4. If bc contains a ⟨string-io-control⟩,sioc, then delete sioc from bc.

Step 5. If bc contains a ⟨format-control-list⟩,fcl, then delete fcl from bc.

Step 6. If bc contains a ⟨remote-block-state⟩,rbs, then delete rbs from bc.

Step 7. If bc contains a ⟨current-file-value⟩,cfv, then delete cfv from bc.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8 The Stream I/O Statements

8.8.1 The Get Statement

8.8.1.1 Execute-get-statement

Operation: EXECUTE-GET-STATEMENT(gs)

Where: gs is a <get-statement>.

Step 1.

Case 1.1. gs has a <get-file>,gf.

Perform EXECUTE-GET-FILE(gf).

Case 1.2. gs has a <get-string>,gstr.

Perform EXECUTE-GET-STRING(gstr).

Step 2. Perform TRIM-IO-CONTROL.

Step 3. Perform NORMAL-SEQUENCE.

8.8.1.2 Execute-get-file

Operation: EXECUTE-GET-FILE(gf)

Where: gf is a <get-file>.

Step 1. Perform Steps 1.1 and 1.2 in either order.

Step 1.1. Let fo be the <value-reference> in the <file-option> in gf. Perform EVALUATE-FILE-OPTION(fo) to obtain a ⩽file-value⩾,fv.

Step 1.2. If gf contains a <skip-option>,sko, then let e be the <expression> in sko, and perform EVALUATE-EXPRESSION-TO-INTEGER(e) to obtain an ⩽integer-value⩾,sk.

Step 2. Let fi be the ⩽file-information⩾ designated by fv. If fi contains ⩽open⩾ then go to Step 4.

Step 3. Let efdl be an ⩽evaluated-file-description-list⩾ containing <stream> and <input>. Perform OPEN(fv,efdl) to obtain rf. If rf is ⩽fail⩾ then perform RAISE-IO-CONDITION(<undefinedfile-condition>,fv). If on normal return fi contains ⩽closed⩾ then perform RAISE-CONDITION(<error-condition>).

Step 4. fi MUST contain <stream> and <input>.

Step 5. If gf has a <skip-option> then perform SKIP(sk,fv).

Step 6. If gf has an <input-specification> then:

Case 6.1. gf has a <list-directed-input>,ldi.

Perform GET-LIST(ldi,fv).

Case 6.2. gf has an <edit-directed-input>,edi.

Perform GET-EDIT(edi,fv).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 7. If gf has a <list-directed-input> then terminate this operation. Otherwise set the ⟨first-comma⟩ in fi to contain ⟨on⟩.

8.8.1.3 Execute-get-string

Operation: EXECUTE-GET-STRING(gstr)

Where: gstr is a <get-string>.

Step 1. Let e be the <expression> immediately contained in gstr. Perform EVALUATE-EXPRESSION(e), to obtain an ⟨aggregate-value⟩,av. Let sdt be the <data-type> in the <data-description> immediate component of e. Let sv be the ⟨basic-value⟩ in av. Let tdt be a <data-type> simply containing <character>, <nonvarying>, and <maximum-length>: <asterisk>. Perform CONVERT(tdt,sdt,sv) to obtain a ⟨character-string-value⟩,csv. Attach to the current ⟨block-control⟩ a ⟨string-io-control⟩: ⟨alpha⟩ csv ⟨first-comma⟩: ⟨on⟩.

Step 2.

Case 2.1. gstr has a <list-directed-input>,ldi.

Perform GET-LIST(ldi).

Case 2.2. gstr has an <edit-directed-input>,edi.

Perform GET-EDIT(edi).

8.8.1.4 Get-list

Operation: GET-LIST(ldi,fv)

Where: ldi is a <list-directed-input>,
fv is a [⟨file-value⟩].

Step 1. Attach to the current ⟨block-control⟩ a

⟨data-item-control-list⟩:
⟨data-item-control⟩:
⟨data-list-indicator⟩ designating the <input-target-list> of ldi
⟨data-item-indicator⟩:
⟨undefined⟩.

Step 2. Perform ESTABLISH-NEXT-DATA-ITEM to obtain a ⟨current-scalar-item⟩,ndi or ⟨none⟩,ndi. If ndi is ⟨none⟩ then terminate this operation.

Step 3. Perform PARSE-LIST-INPUT(fv) to obtain a ⟨character-string-value⟩,csv.

Step 4. If fv is a ⟨file-value⟩ then attach to the current ⟨block-control⟩ a ⟨current-file-value⟩: fv.

Case 4.1. csv contains just one terminal which is a ⟨null-character-string⟩.

Go to Step 6.

Case 4.2. The terminal nodes of csv can be parsed as "⟨non-blank-comma-quote⟩ [⟨non-blank-comma-list⟩]".

Let v be csv. Let st be <character>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 4.3. The terminal nodes of csv can be parsed as "⟨simple-character-string-constant⟩".

Perform BASIC-CHARACTER-VALUE(csv) to obtain a ⟨character-string-value⟩,v. Let st be <character>.

Case 4.4. The terminal nodes of csv can be parsed as "⟨simple-bit-string-constant⟩".

Perform BASIC-BIT-VALUE(csv) to obtain a ⟨bit-string-value⟩,v. Let st be <bit>.

Case 4.5. (Otherwise).

Perform RAISE-IO-CONDITION(<conversion-condition>,fv,csv). On normal return let csv be the immediate component of the current ⟨returned-onsource-value⟩; csv MUST NOT contain only blanks; go to Step 4.

Step 5. Let et be the ⟨evaluated-target⟩ in ndi. Let agv be

⟨aggregate-value⟩:
⟨aggregate-type⟩:
⟨scalar⟩;
⟨basic-value-list⟩:
⟨basic-value⟩:
v.

Let dd be

<data-description>:
<item-data-description>:
<data-type>:
<computational-type>:
<string>:
<string-type>: st;
<maximum-length>:
<asterisk>;
<nonvarying>.

Perform ASSIGN(et,agv,dd).

Step 6. If fv is a ⟨file-value⟩ then delete the current ⟨current-file-value⟩. Go to Step 2.

8.8.1.4.1 Parse-list-input

Operation: PARSE-LIST-INPUT(fv)

Where: fv is a [⟨file-value⟩].

Result: a ⟨character-string-value⟩.

Step 1.

Case 1.1. fv is a ⟨file-value⟩.

Let fi be the ⟨file-information⟩ designated by fv, and let ds be the ⟨dataset⟩ designated by the ⟨dataset-designator⟩ in fi. Let cp be the ⟨current-position⟩ in fi. If ds contains a ⟨stream-item-list⟩ then let lsi designate the last ⟨stream-item⟩ in this list; otherwise let lsi be ⟨absent⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1.1.1. cp = lsi and the ⟨first-comma⟩,fc in fi contains ⟨on⟩.

Set fc to contain ⟨off⟩. Return ⟨character-string-value⟩: ⟨null-character-string⟩.

Case 1.1.2. There is no ⟨stream-item-list⟩ in ds, or cp = lsi and Case 1.1.1 does not apply.

Perform RAISE-IO-CONDITION(⟨endfile-condition⟩,fv).

Case 1.1.3. (Otherwise).

Let sl be a ⟨stream-item-list⟩ containing the ⟨stream-item⟩s in the ⟨stream-item-list⟩ in ds, following the ⟨stream-item⟩ designated by cp.

Case 1.2. fv is ⟨absent⟩.

Let sl be the ⟨character-string-value⟩ in the current ⟨string-io-control⟩. If sl contains no ⟨symbol⟩s and the ⟨first-comma⟩,fc in the current ⟨string-io-control⟩ contains ⟨on⟩ then set fc to contain ⟨off⟩ and return ⟨character-string-value⟩: ⟨null-character-string⟩. Otherwise, if sl contains no ⟨symbol⟩s then perform RAISE-CONDITION(⟨error-condition⟩).

Step 2.

Case 2.1. sl can be parsed as "[⟨leading-delimiter-list⟩] , [⟨stream-item-list⟩]".

Let ld be that sequence in sl which satisfies "[⟨leading-delimiter-list⟩] ," in this parse, and let its number of terminal nodes be ln. Perform INPUT-STREAM-ITEM(fv) ln times. If the current ⟨first-comma⟩ exists and contains ⟨on⟩ or the ⟨first-comma⟩ in the ⟨file-information⟩,fi exists and contains ⟨on⟩ then return ⟨character-string-value⟩: ⟨null-character-string⟩. Otherwise set the current ⟨first-comma⟩ (respectively, the ⟨first-comma⟩ in fi) to contain ⟨on⟩, perform PARSE-LIST-INPUT(fv) to obtain csv, and return csv.

Case 2.2. sl can be parsed as "⟨leading-delimiter-list⟩".

If fv is a ⟨file-value⟩ and the ⟨first-comma⟩,fc contains ⟨on⟩ and cp designates ⟨alpha⟩, or fv is ⟨absent⟩ and the ⟨first-comma⟩,fc in the current ⟨string-io-control⟩,sioc contains ⟨on⟩ and sioc immediately contains ⟨alpha⟩, then set fc to ⟨off⟩. Let ln be the number of terminal nodes in sl. Perform INPUT-STREAM-ITEM(fv) ln times. Go to Step 1.

Case 2.3. sl can be parsed as "[⟨leading-delimiter-list⟩] ' [⟨string-symbol-or-linemark-list⟩]".

Let ln be the number of terminal nodes in sl. Perform INPUT-STREAM-ITEM(fv) ln times. Perform RAISE-CONDITION(⟨error-condition⟩).

Case 2.4. sl can be parsed as "[⟨leading-delimiter-list⟩] ⟨putative-list-constant⟩ [{∅ | ,} [⟨stream-item-list⟩]]"

Let ln be the number of terminal nodes in sl preceding that part which satisfies "[⟨stream-item-list⟩]" in this parse. Perform INPUT-STREAM-ITEM(fv) ln times. Let csv be a ⟨character-string-value⟩ whose terminal nodes are those of the part of sl which satisfies "⟨putative-list-constant⟩ [∅ | ,]" except those terminals that are ⟨linemark⟩s. Let fc be the current ⟨first-comma⟩, if that exists; otherwise let fc be the ⟨first-comma⟩ in fi. If the last ⟨symbol⟩ of csv is a ⟨,⟩ then set fc to contain ⟨on⟩; otherwise set fc to contain ⟨off⟩. Delete from csv the last ⟨symbol⟩, if that ⟨symbol⟩ is a ∅ or a ⟨,⟩. Return csv.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.1.4.2 Parsing Categories for List Directed Input

Some of the categories used in parsing input streams for list directed input are categories of the Concrete Syntax or the Machine-state Syntax. Others are defined as follows:

⟨leading-delimiter⟩ ::= ⊬ | ⟨linemark⟩

⟨putative-list-constant⟩ ::= '[⟨string-symbol-or-linemark-list⟩]'
[⟨non-blank-comma-quote⟩ [⟨non-blank-comma-list⟩]] |
{⟨data-symbol⟩ | ⟨;⟩ | = } [⟨non-blank-comma-list⟩]

⟨data-symbol⟩ ::= ⟨letter⟩ | ⟨digit⟩ | _ | + | − | (|) | . | : | & | ⟨|⟩ | ¬ | > | < | / |
* | $ | % | ⟨extralingual-character⟩

Note: The subnodes of ⟨data-symbol⟩ are those of ⟨symbol⟩ except for ⊬, ⟨,⟩, ⟨;⟩, ⟨'⟩ and ⟨=⟩.

⟨non-blank-comma⟩ ::= ⟨non-blank-comma-quote⟩ | '

⟨non-blank-comma-quote⟩ ::= ⟨data-symbol⟩ | ; | = | ⟨linemark⟩

⟨string-symbol-or-linemark⟩ ::= ⟨string-or-picture-symbol⟩ | ⟨linemark⟩ | '⟨linemark-list⟩'

8.8.1.5 Get-edit

Operation: GET-EDIT(edi,fv)

Where: edi is an <edit-directed-input>,
fv is a [⟨file-value⟩].

Step 1. Attach to the current ⟨block-control⟩ a

⟨data-item-control-list⟩:
⟨data-item-control⟩:
⟨data-list-indicator⟩ designating the
<input-target-list> of edi
⟨data-item-indicator⟩:
⟨undefined⟩.

Step 2. Attach to the current ⟨block-control⟩ a

⟨format-control-list⟩:
⟨format-control⟩:
<format-specification-list-designator> designating the
<format-specification-list> of edi
⟨format-list-index⟩:
⟨integer-value⟩:
0.

Step 3. Perform ESTABLISH-NEXT-DATA-ITEM to obtain a ⟨current-scalar-item⟩,ndi:
⟨evaluated-target⟩,et; or ⟨none⟩,ndi. If ndi is ⟨none⟩ then terminate this operation.

Step 4. Perform ESTABLISH-NEXT-FORMAT-ITEM to obtain a <format-item>,efi.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 5.

Case 5.1. efi contains a <control-format>,ecf.

Perform EXECUTE-INPUT-CONTROL-FORMAT(ecf,fv). Go to Step 4.

Case 5.2. efi contains a <data-format>,edf.

Perform EXECUTE-INPUT-DATA-FORMAT(et,edf,fv). Go to Step 3.

8.8.1.5.1 Execute-input-control-format

Operation: EXECUTE-INPUT-CONTROL-FORMAT(ecf,fv)

Where: ecf is a <control-format>,
fv is a [≪file-value≫].

Case 1. fv is a ≪file-value≫.

Step 1.1. Let fi be the ≪file-information≫ designated by fv. ecf MUST NOT have a <page>, <line-format>, or <tab-format>. Let ds be the ≪dataset≫ designated by the ≪dataset-designator≫ in fi.

Step 1.2.

Case 1.2.1. ecf has a <space-format>: ≪integer-value≫,w.

w MUST NOT be negative. If w=0 then terminate this operation; otherwise perform INPUT-STREAM-ITEM-FOR-EDIT(fv,i), for i=1,...,w.

Case 1.2.2. ecf has a <skip-format>: ≪integer-value≫,w.

Perform SKIP(w,fv).

Case 1.2.3. ecf has a <column-format>: ≪integer-value≫,w.

Step 1.2.3.1. w MUST NOT be negative. Perform EVALUATE-CURRENT-COLUMN(fv) to obtain cc. If w = 0 then let w be 1. Let n be the number of ≪symbol≫s in ds which follow the ≪stream-item≫ designated by the ≪current-position≫ in fi, up to the next ≪stream-item≫ which has a ≪linemark≫ or ≪omega≫.

Step 1.2.3.2.

Case 1.2.3.2.1. cc ≤ (w−1) ≤ cc+n and n > 0.

Perform INPUT-STREAM-ITEM(fv) (w−cc−1) times.

Case 1.2.3.2.2. (Otherwise).

Perform SKIP(≪integer-value≫:1;,fv). Let n1 be the number of ≪symbol≫s in ds which follow the ≪stream-item≫ designated by the ≪current-position≫ in fi, up to the next ≪stream-item≫ which has a ≪linemark≫ or ≪omega≫. If n1 ≥ (w−1) > 0 then perform INPUT-STREAM-ITEM(fv) (w−1) times.

Case 2. fv is ≪absent≫.

ecf MUST have a <space-format>: ≪integer-value≫,w. w MUST NOT be negative. If w=0 then terminate this operation; otherwise perform INPUT-STREAM-ITEM-FOR-EDIT(fv,i), for i=1,...,w.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.1.5.2 Execute-input-data-format

Operation: EXECUTE-INPUT-DATA-FORMAT(et,edf,fv)

Where: et is an ⟨evaluated-target⟩,
edf is a <data-format>,
fv is a [⟨file-value⟩].

Step 1.

Case 1.1. edf immediately contains an <l-format>.

Step 1.1.1. fv MUST be a ⟨file-value⟩. Let sil be a ⟨stream-item-list⟩.

Step 1.1.2. Perform INPUT-STREAM-ITEM(fv) to obtain si. If si is neither ⟨omega⟩ nor ⟨stream-item⟩: ⟨linemark⟩; then append si to sil and go to Step 1.1.2.

Step 1.1.3. If sil does not have subnodes then let csv be a ⟨character-string-value⟩: ⟨null-character-string⟩;, otherwise let csv be a ⟨character-string-value⟩ containing, in order, the ⟨symbol⟩s in sil.

Step 1.1.4. Let agv be an

⟨aggregate-value⟩:
⟨aggregate-type⟩:
⟨scalar⟩;
⟨basic-value-list⟩:
⟨basic-value⟩:
csv.

Let dd be a <data-description> simply containing a <data-type>, which has <string>: <character> <maximum-length>: <asterisk>;;, and no other subnodes.

Step 1.1.5. Attach to the current ⟨block-control⟩ a ⟨current-file-value⟩: fv. Perform ASSIGN(et,agv,dd). Delete the current ⟨current-file-value⟩. Terminate this operation.

Case 1.2. edf immediately contains a <real-format> or a <string-format>.

Let w be the first ⟨integer-value⟩ in edf; w MUST be present.

Case 1.3. edf immediately contains a <picture-format>,pf.

Let w be the associated character-string length of pf. (See Section 9.6.2)

Step 2. w MUST NOT be negative. If w=0 then let v be a ⟨basic-value⟩ : ⟨character-string-value⟩ : ⟨null-character-string⟩;; and go to Step 5. Otherwise perform INPUT-STREAM-ITEM-FOR-EDIT(fv,i), for i=1,...,w, to obtain w ⟨stream-item⟩s, si[i] each containing a ⟨symbol⟩. If fv is a ⟨file-value⟩ then attach to the current ⟨block-control⟩ a ⟨current-file-value⟩: fv. Let csv be a ⟨character-string-value⟩ containing, in order, the w ⟨character-value⟩s cv[i], i=1,...,w, where cv[i] contains the ⟨symbol⟩ si[i], for i=1,...,w.

Step 3. If edf immediately contains a <real-format>,f or a <string-format>,f, then perform Steps 3.1 through 3.3.

Step 3.1. Let df be the immediate component of f.

Step 3.2. Perform VALIDATE-INPUT-FORMAT(df,csv) to obtain a ⟨value-and-type⟩,v, or a ⟨character-string-value⟩,v, or ⟨invalid⟩,v. If v is ⟨invalid⟩ then perform

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

RAISE-IO-CONDITION (<conversion-condition>,fv,csv); on normal return let csv be the immediate component of the current ⟨returned-onsource-value⟩ and go to Step 3.2.

Step 3.3. Go to Step 5.

Step 4. If edf immediately contains a <picture-format>,pf, then perform Step 4.1.

Step 4.1. Perform VALIDATE-INPUT-FORMAT(pf,csv) to obtain a ⟨value-and-type⟩,v, or a ⟨character-string-value⟩,v, or ⟨invalid⟩,v. If v is ⟨invalid⟩ then perform RAISE-IO-CONDITION (<conversion-condition>,fv,csv); on normal return let csv be the immediate component of the current ⟨returned-onsource-value⟩ and go to Step 4.1.

Step 5.

Case 5.1. v is a ⟨character-string-value⟩ or a ⟨bit-string-value⟩.

Let edd be an ⟨evaluated-data-description⟩ containing an <item-data-description> of <character> or <bit> (respectively) and <maximum-length>: <asterisk>. Let bv be a ⟨basic-value⟩: v.

Case 5.2. v is a ⟨value-and-type⟩: val adt.

Let edd be an ⟨evaluated-data-description⟩ containing an <item-data-description> containing adt. Let bv be a ⟨basic-value⟩: val.

Step 6. Let agv be an

⟨aggregate-value⟩:
⟨aggregate-type⟩:
⟨scalar⟩;
⟨basic-value-list⟩:
bv.

Let dd be the <data-description> immediate component of edd. Perform ASSIGN(et,agv,dd). If fv is a ⟨file-value⟩ then delete the current ⟨current-file-value⟩.

8.8.1.5.3 Validate-input-format

Operation: VALIDATE-INPUT-FORMAT(df,csv)

Where: df is an immediate component of a <real-format> or a <string-format>, or is a <picture-format>,
csv is a ⟨character-string-value⟩.

Result: ⟨invalid⟩ or ⟨character-string-value⟩ or ⟨bit-string-value⟩ or ⟨value-and-type⟩.

Case 1. df is a <picture-format>.

Perform VALIDATE-NUMERIC-PICTURED-VALUE(df,csv) to obtain a ⟨picture-validity⟩,pv. If pv has ⟨picture-invalid⟩ then return ⟨invalid⟩. If pv has ⟨picture-valid⟩: val; then return a ⟨value-and-type⟩: val aadt; where aadt is the associated arithmetic data-type of df.

Case 2. df is a <character-format>.

Return csv.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3. df is a <bit-format>.

Case 3.1. csv conforms to the syntax "[⟨ϐ-list⟩] [bs-list] [⟨ϐ-list⟩]" where bs is one of the ⟨symbol⟩s in the first column of Figure 12 on page 149 corresponding to the <radix-factor> in df.

Let bc be that part of csv which satisfies "[bs-list]". Let bcx be the ⩽character-string-value⩾ obtained by concatenating bc with a ⟨'⟩ on the left, and, on the right, with a 'B1, 'B2, 'B3 or 'B4, according to the <radix-factor> of df. Perform BASIC-BIT-VALUE(bcx) to obtain a ⩽bit-string-value⩾,bsv. Return bsv.

Case 3.2. (Otherwise).

Return ⩽invalid⩾.

Case 4. df is a <fixed-point-format>.

Step 4.1. Let w and d be respectively the ⩽integer-value⩾s in df. If d is ⩽absent⩾ then 0 is assumed. d MUST NOT be negative.

Step 4.2.

Case 4.2.1. csv conforms to the syntax "⟨ϐ-list⟩" or is a ⩽character-string-value⩾: ⩽null-character-string⩾.

Return ⩽value-and-type⩾: ⩽real-value⩾: 0; adt; where adt is a <data-type> containing <fixed>, <decimal>, <number-of-digits>: 1; and <scale-factor>: 0.

Case 4.2.2. csv conforms to the syntax "[⟨ϐ-list⟩] [+ | −] ⟨decimal-number⟩ [⟨ϐ-list⟩]".

Perform BASIC-NUMERIC-VALUE(csv) to obtain a ⩽value-and-type⩾: v dt. If csv contains a ⟨.⟩ then let s=0; otherwise, let s=−d. d MUST NOT be greater than the <scale-factor> in dt. Return a ⩽value-and-type⩾: ⩽real-value⩾: $(v*10^{s})$; adt; where adt is a <data-type> equal to dt except that <scale-factor> is decremented by s.

Case 4.2.3. (Otherwise).

Return ⩽invalid⩾.

Case 5. df is a <floating-point-format>.

Step 5.1. Let w and d be respectively the ⩽integer-value⩾s in df. If d is ⩽absent⩾ then let d be 0. d MUST NOT be negative.

Step 5.2.

Case 5.2.1. csv conforms to the syntax "⟨ϐ-list⟩" or is a ⩽character-string-value⩾: ⩽null-character-string⩾.

Return ⩽value-and-type⩾: ⩽real-value⩾: 0; adt; where adt is a <data-type> containing <fixed>, <decimal>, <number-of-digits>: 1;, and <scale-factor>: 0.

Case 5.2.2. csv conforms to the syntax "[⟨ϐ-list⟩] [+ | −] ⟨decimal-number⟩ [{[E]{+ | −} | E} ⟨integer⟩] [⟨ϐ-list⟩]".

If csv contains a sign, but not an E, after a substring conforming to ⟨decimal-number⟩, then replace that substring with the concatenation of that substring with E. Perform BASIC-NUMERIC-VALUE(csv) to obtain a ⩽value-and-type⩾: v dt. If csv contains a ⟨.⟩ then let d'=0; otherwise let d'=d. Return a ⩽value-and-type⩾: ⩽real-value⩾: $(v*10^{-d'})$; adt; where adt is a <data-type> equal to dt except that its <scale-factor> (if any) is incremented by d'.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 5.2.3. (Otherwise).

Return ⟨invalid⟩.

8.8.1.6 Input-stream-item

Operation: INPUT-STREAM-ITEM(fv)

Where: fv is a [⟨file-value⟩].

Result: a ⟨stream-item⟩ or ⟨omega⟩.

Step 1.

Case 1.1. fv is a ⟨file-value⟩.

Step 1.1.1. Let fi be the ⟨file-information⟩ designated by fv; let cp be the ⟨current-position⟩ in fi, and let ds be the ⟨dataset⟩ designated by the ⟨dataset-designator⟩ in fi.

Step 1.1.2.

Case 1.1.2.1. cp designates ⟨alpha⟩.

Replace the immediate component of cp by a ⟨designator⟩ to the first ⟨stream-item⟩ in ds, or to the ⟨omega⟩ in ds if there are no ⟨stream-item⟩s in ds.

Case 1.1.2.2. cp designates a ⟨stream-item⟩.

Replace the immediate component of cp by a ⟨designator⟩ to the next ⟨stream-item⟩ in ds, or to the ⟨omega⟩ in ds if there are no further ⟨stream-item⟩s in ds.

Case 1.1.2.3. cp designates ⟨omega⟩.

Go to Step 1.1.3.

Step 1.1.3. Let si be the node designated by cp.

Case 1.2. fv is ⟨absent⟩.

Let sioc be the current ⟨string-io-control⟩. Let csv be the ⟨character-string-value⟩ in sioc. If csv contains a ⟨null-character-string⟩ then perform RAISE-CONDITION(⟨error-condition⟩). Otherwise let si be a ⟨stream-item⟩: v;, where v is the first ⟨symbol⟩ in csv. If csv contains only one ⟨symbol⟩ then replace csv by ⟨character-string-value⟩: ⟨null-character-string⟩. Otherwise delete the first ⟨character-value⟩ in csv. If sioc immediately contains ⟨alpha⟩ then delete ⟨alpha⟩.

Step 2. Return si.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.1.7 Basic-character-value

Operation: BASIC-CHARACTER-VALUE(csvin)

Where: csvin is a ⟨character-string-value⟩.

Result: a ⟨character-string-value⟩.

Step 1. The ⟨symbol⟩s of csvin MUST conform to the syntax of ⟨simple-character-string-constant⟩.

Step 2. Let csvo be a copy of csvin. Remove the first and last ⟨character-value⟩s of csvo. If no ⟨character-value⟩s remain then return ⟨character-string-value⟩: ⟨null-character-string⟩.

Replace all adjacent pairs of ⟨character-value⟩: ⟨symbol⟩: ⟨'⟩; originally in csvo with a single ⟨character-value⟩: ⟨symbol⟩: ⟨'⟩.

Step 3. Return csvo.

8.8.1.8 Basic-bit-value

Operation: BASIC-BIT-VALUE(csvin)

Where: csvin is a ⟨character-string-value⟩.

Result: a ⟨bit-string-value⟩.

Step 1. csvin MUST conform to the syntax of ⟨simple-bit-string-constant⟩.

Step 2. Let m be 1, 1, 2, 3, or 4 according as the ⟨radix-factor⟩ in csvin is B, B1, B2, B3, or B4. Remove the portion of csvin corresponding to the syntax for ⟨radix-factor⟩ and remove the first and last of the remaining ⟨character-value⟩s. Let n be the number of ⟨character-value⟩s remaining, if any. If n=0 then return ⟨bit-string-value⟩: ⟨null-bit-string⟩.

Step 3. Let sy[i], for i=1,...,n, be the ⟨symbol⟩s, in order, of the remaining ⟨character-value⟩s in csvin. Each sy[i] MUST have an entry in Figure 12 on page 149 which is valid for the value of m. Let bsv be a ⟨bit-string-value⟩, containing m*n ⟨bit-value⟩s, such that ⟨bit-value⟩s (i*m+1−m) through (i*m) are obtained from Figure 12 on page 149 as a function of m and sy[i], for i=1,...,n.

Step 4. Return bsv.

8.8.1.9 Input-stream-item-for-edit

Operation: INPUT-STREAM-ITEM-FOR-EDIT(fv,i)

Where: fv is a [⟨file-value⟩],
i is an ⟨integer-value⟩.

Result: a ⟨stream-item⟩.

Step 1. Perform INPUT-STREAM-ITEM(fv) to obtain a ⟨stream-item⟩,si or an ⟨omega⟩,si.

Step 2.

Case 2.1. si is ⟨omega⟩.

If i=1 then perform RAISE-IO-CONDITION(⟨endfile-condition⟩,fv); otherwise perform RAISE-CONDITION(⟨error-condition⟩).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.2. (Otherwise).

If si contains ⟪linemark⟫ then go to Step 1. Otherwise return si.

8.8.2 The Put Statement

8.8.2.1 Execute-put-statement

Operation: EXECUTE-PUT-STATEMENT(ps)

Where: ps is a <put-statement>.

Step 1.

Case 1.1. ps has a <put-file>,pf.

Perform EXECUTE-PUT-FILE(pf).

Case 1.2. ps has a <put-string>,pstr.

Perform EXECUTE-PUT-STRING(pstr).

Step 2. Perform TRIM-IO-CONTROL.

Step 3. Perform NORMAL-SEQUENCE.

8.8.2.2 Execute-put-file

Operation: EXECUTE-PUT-FILE(pf)

Where: pf is a <put-file>.

Step 1. Perform Steps 1.1, 1.2, and 1.3 in any order.

Step 1.1. Let fo be the <value-reference> of the <file-option> in pf. Perform EVALUATE-FILE-OPTION(fo), to obtain a ⟪file-value⟫,fv.

Step 1.2. If pf has a <skip-option>,sko, then let e be the <expression> in sko, and perform EVALUATE-EXPRESSION-TO-INTEGER(e), to obtain an ⟪integer-value⟫,sk.

Step 1.3. If pf has a <line-option>,lopt, then let el be the <expression> in lopt, and perform EVALUATE-EXPRESSION-TO-INTEGER(el), to obtain an ⟪integer-value⟫,l.

Step 2. Let fi be the ⟪file-information⟫ designated by fv. If fi contains ⟪open⟫ then go to Step 4; otherwise go to Step 3.

Step 3. Let efdl be an ⟪evaluated-file-description-list⟫ containing <stream> and <output>. Perform OPEN(fv,efdl) to obtain a result rf. If rf is ⟪fail⟫ then perform RAISE-IO-CONDITION(<undefinedfile-condition>,fv). If on normal return fi contains ⟪closed⟫ then perform RAISE-CONDITION(<error-condition>).

Step 4. fi MUST contain <stream> and <output>.

Step 5. If pf immediately contains <page> then perform PUT-PAGE(fv).

Step 6. If pf has a <line-option> then perform PUT-LINE(l,fv).

Step 7. If pf has a <skip-option> then perform SKIP(sk,fv).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 8. If pf has an <output-specification> then:

Case 8.1. pf has a <list-directed-output>,ldo.

Perform PUT-LIST(ldo,fv).

Case 8.2. pf has an <edit-directed-output>,edo.

Perform PUT-EDIT(edo,fv).

8.8.2.3 Execute-put-string

Operation: EXECUTE-PUT-STRING(pstr)

Where: pstr is a <put-string>.

Step 1. Let tr be the <variable-reference> in pstr. Perform EVALUATE-TARGET-REFERENCE(tr) to obtain an ⟨evaluated-target⟩,et. Let n be the <maximum-length> contained in et.

Step 2. Attach to the current ⟨block-control⟩ a

⟨string-io-control⟩:
⟨character-string-value⟩:
⟨null-character-string⟩;
⟨string-limit⟩:
⟨integer-value⟩:
n.

Step 3.

Case 3.1. pstr has a <list-directed-output>,ldo.

Perform PUT-LIST(ldo).

Case 3.2. pstr has an <edit-directed-output>,edo.

Perform PUT-EDIT(edo).

Step 4. Let os be the ⟨character-string-value⟩ of the current ⟨string-io-control⟩. Let m be the number of {symbol}s in os. Perform ASSIGN(et,agv,dd) where agv is an ⟨aggregate-value⟩ containing os, and dd is a <data-description> containing <character>, <nonvarying>, and <maximum-length> with m.

Step 5. Delete the ⟨string-io-control⟩ from the current ⟨block-control⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.2.4 Put-list

Operation: PUT-LIST(ldo,fv)

Where: ldo is a <list-directed-output>,
fv is a [⟨file-value⟩].

Step 1. Attach to the current ⟨block-control⟩ a

⟨data-item-control-list⟩:
⟨data-item-control⟩:
⟨data-list-indicator⟩ designating the
<output-source-list> of ldo
⟨data-item-indicator⟩:
⟨undefined⟩.

Step 2. If fv is a ⟨file-value⟩ then let fi be the ⟨file-information⟩ designated by fv.

Step 3. Let t be an ⟨integer-value⟩.

Case 3.1. fv is a ⟨file-value⟩ and fi contains <print>.

Step 3.1.1. Perform EVALUATE-CURRENT-COLUMN(fv) to obtain cc.

Step 3.1.2. If the ⟨evaluated-tab-option⟩ of fi contains an ⟨integer-value⟩,iv, such that cc=iv−1, then let t=0; otherwise let t=1.

Case 3.2. (Otherwise).

Let t=0.

Step 4. Perform ESTABLISH-NEXT-DATA-ITEM to obtain a ⟨current-scalar-item⟩,ndi or ⟨none⟩,ndi. If ndi is ⟨none⟩ then perform OUTPUT-STRING-ITEM(⟨stream-item⟩:⟨symbol⟩:⊮;;,fv) and terminate this operation.

Step 5. Let sdt be the <data-type> in ndi, and let tdt be a <data-type> simply containing <character>, <nonvarying>, and <maximum-length>: <asterisk>. Perform CONVERT(tdt,sdt,bv), where bv is the ⟨basic-value⟩ in ndi, to obtain a ⟨character-string-value⟩,csv.

Step 6.

Case 6.1. sdt contains <bit>.

Let cv be a ⟨stream-item-list⟩ containing, in order, the ⟨symbol⟩s

⟨'⟩
the ⟨symbol⟩s of csv, if any,
⟨'⟩
B

Case 6.2. sdt contains <character>, and either fv is ⟨absent⟩ or fv is a ⟨file-value⟩ and fi does not contain <print>.

Let cv be a ⟨stream-item-list⟩ containing, in order, the ⟨symbol⟩s

⟨'⟩
the ⟨symbol⟩s of csv, if any, but with each ⟨symbol⟩: ⟨'⟩; replaced by two occurrences of ⟨symbol⟩: ⟨'⟩;
⟨'⟩

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 6.3. (Otherwise).

Let cv be a ⟨stream-item-list⟩ containing, in order, the ⟨symbol⟩s of csv.

Step 7.

Case 7.1. fv is a ⟨file-value⟩ and fi contains <print>.

Perform TAB(t,fv).

Case 7.2. (Otherwise).

If t=1 then perform OUTPUT-STRING(⟨stream-item-list⟩: ⟨stream-item⟩: ⟨symbol⟩: ∅;;;,fv).

Step 8. Let t=1.

Step 9.

Case 9.1. fv is a ⟨file-value⟩.

Perform EVALUATE-CURRENT-COLUMN(fv) to obtain ccol. Let lsz be the ⟨evaluated-linesize⟩ in fi. Let lcv be the number of ⟨symbol⟩s in cv. If lcv > (lsz−ccol) and ccol ≠ 0 then perform SKIP(⟨integer-value⟩:1;,fv). Perform OUTPUT-STRING(cv,fv).

Case 9.2. fv is ⟨absent⟩.

Perform OUTPUT-STRING(cv,fv).

Step 10. Go to Step 4.

8.8.2.5 Put-edit

Operation: PUT-EDIT(edo,fv)

Where: edo is an <edit-directed-output>,
fv is a [⟨file-value⟩].

Step 1. Attach to the current ⟨block-control⟩ a

⟨data-item-control-list⟩:
⟨data-item-control⟩:
⟨data-list-indicator⟩ designating the
<output-source-list> of edo
⟨data-item-indicator⟩:
⟨undefined⟩.

Step 2. Attach to the current ⟨block-control⟩ a

⟨format-control-list⟩:
⟨format-control⟩:
<format-specification-list-designator> designating the
<format-specification-list> of edo;
⟨format-list-index⟩:
⟨integer-value⟩:
0.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3. Perform ESTABLISH-NEXT-DATA-ITEM to obtain a ⟪current-scalar-item⟫,ndi, or ⟪none⟫,ndi. If ndi is ⟪none⟫ then terminate this operation.

Step 4. Perform ESTABLISH-NEXT-FORMAT-ITEM to obtain a <format-item>,efi.

Step 5.

Case 5.1. efi contains a <control-format>,ecf.

Perform EXECUTE-OUTPUT-CONTROL-FORMAT(ecf,fv). Go to Step 4.

Case 5.2. efi contains a <data-format>,edf.

Perform EXECUTE-OUTPUT-DATA-FORMAT(ndi,edf,fv). Go to Step 3.

8.8.2.5.1 Execute-output-control-format

Operation: EXECUTE-OUTPUT-CONTROL-FORMAT(ecf,fv)

Where: ecf is a <control-format>,
fv is a [⟪file-value⟫].

Step 1. If fv is a ⟪file-value⟫ then let fi be the ⟪file-information⟫ designated by fv. If fv is ⟪absent⟫ then ecf MUST have a <space-format>.

Step 2.

Case 2.1. ecf has a <space-format>: ⟪integer-value⟫,w.

w MUST NOT be negative. If w=0 then terminate this operation. Perform OUTPUT-STRING(str,fv) where str is a ⟪stream-item-list⟫ whose w terminal nodes are all ∅s.

Case 2.2. ecf has a <skip-format>: ⟪integer-value⟫,w.

Perform SKIP(w,fv).

Case 2.3. ecf has a <line-format>: ⟪integer-value⟫,w.

Perform PUT-LINE(w,fv).

Case 2.4. ecf has <page>.

Perform PUT-PAGE(fv).

Case 2.5. ecf has a <tab-format>: ⟪integer-value⟫,w.

w MUST NOT be negative. Perform TAB(w,fv).

Case 2.6. ecf has a <column-format>: ⟪integer-value⟫,w.

Step 2.6.1. Perform EVALUATE-CURRENT-COLUMN(fv) to obtain cc. w MUST NOT be negative. If w = 0 or w exceeds the ⟪evaluated-linesize⟫ in fi then let w be 1.

Step 2.6.2.

Case 2.6.2.1. cc < w−1.

Perform OUTPUT-STRING(str,fv), where str is a ⟪stream-item-list⟫ containing (w−cc−1) terminal nodes, each of which is a ∅.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.6.2.2. cc = w−1.

Terminate this operation.

Case 2.6.2.3. cc > w−1.

Perform SKIP(≤integer-value≥:1;,fv). If w > 1 then perform OUTPUT-STRING(str,fv), where str is a ≤stream-item-list≥ containing (w−1) terminal nodes, each of which is a ∅.

8.8.2.5.2 Execute-output-data-format

Operation: EXECUTE-OUTPUT-DATA-FORMAT(csi,edf,fv)

Where: csi is a ≤current-scalar-item≥,
edf is a <data-format>,
fv is a [≤file-value≥].

Case 1. edf contains a <bit-format>,ebf.

Let m be the value of the <radix-factor> in ebf. Perform CONVERT(tdt,sdt,bv) to obtain a ≤bit-string-value≥,bsv, where tdt is a <data-type> containing <bit>, <nonvarying>, and <maximum-length>: <asterisk>;, sdt is the <data-type> of csi, and bv is the ≤basic-value≥ immediate component of csi. Let n be the number of immediate components of bsv. Let nn=n. If n is not a multiple of m, then let nn be the next higher multiple of m and insert (nn−n) ≤bit-value≥: ≤zero-bit≥;s at the end of the ≤bit-value-list≥ of bsv. Let n=nn. Let k=n/m. If ebf contains an ≤integer-value≥,w, and n > w*m then let k=w. If ebf has no ≤integer-value≥ then let w be k. If w=0 then terminate this operation. Let csv be a ≤stream-item-list≥ containing w ≮symbol≯s as follows:

> The first k contain, in order, the ≮symbol≯s in Figure 12 on page 149 corresponding to successive groups of m ≤bit-value≥s in bsv.
> The remaining w−k contain ∅s.

Perform OUTPUT-STRING(csv,fv).

Case 2. edf contains a <character-format>,ecf.

Let int be the ≤integer-value≥ in ecf (or <asterisk> if there is no ≤integer-value≥ in ecf). Let tdt be a <data-type> containing <character> and <maximum-length>: <extent-expression>: int. Let sdt be the <data-type> and bv the ≤basic-value≥ of csi. Perform CONVERT(tdt,sdt,bv) to obtain a ≤character-string-value≥,csv. If csv contains a ≤null-character-string≥ then terminate this operation; otherwise let sil be a ≤stream-item-list≥ containing, in order, the ≮symbol≯s of csv. Perform OUTPUT-STRING(sil,fv).

Case 3. edf contains an <l-format>.

Step 3.1. fv MUST be a ≤file-value≥. Let tdt be a <data-type> containing <character> and <maximum-length>: <asterisk>;. Let sdt be the <data-type> and bv the ≤basic-value≥ of csi. Perform CONVERT(tdt,sdt,bv) to obtain a ≤character-string-value≥,csv. If csv does not contain a ≤null-character-string≥ then perform Step 3.1.1.

Step 3.1.1. Let sil be a ≤stream-item-list≥ containing, in order, the ≮symbol≯s of csv. Perform OUTPUT-STRING(sil,fv).

Step 3.2. Perform SKIP(≤integer-value≥:1;,fv).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 4. edf immediately contains a <picture-format> with a <pictured> node, pic.

Let sdt be the <data-type> and bv the ⟪basic-value⟫ of csi. Perform CONVERT(pdt,sdt,bv), where pdt is a <data-type> containing pic, to obtain a ⟪character-string-value⟫,csv. Let sil be a ⟪stream-item-list⟫ containing, in order, the ⟨symbol⟩s of csv. Perform OUTPUT-STRING(sil,fv).

Case 5. (Otherwise).

Let fpf be the <fixed-point-format> or <floating-point-format> in edf. Perform EDIT-NUMERIC-OUTPUT(bv,dt,fpf) to obtain a ⟪character-string-value⟫,csv, where bv is the ⟪basic-value⟫ in csi and dt is the <data-type> in csi. Let sil be a ⟪stream-item-list⟫ containing, in order, the ⟨symbol⟩s of csv. If sil contains one or more ⟨symbol⟩s then perform OUTPUT-STRING(sil,fv).

8.8.2.5.3 Edit-numeric-output

Operation: EDIT-NUMERIC-OUTPUT(bv,sdt,fpf)

Where: bv is a ⟪basic-value⟫,
sdt is a <data-type>,
fpf is a <fixed-point-format> or a <floating-point-format>.

Result: a ⟪character-string-value⟫.

Step 1.

Case 1.1. bv has a ⟪real-value⟫,cv.

Let cdt be sdt.

Case 1.2. bv has a ⟪character-string-value⟫ or a ⟪bit-string-value⟫.

Perform CONVERT-TO-ARITHMETIC(sdt,v) to obtain a ⟪value-and-type⟫ with first two components cv and dt, where v is the immediate component of bv. Let cdt be a <data-type> containing dt.

Step 2. Let w and d be (respectively) the ⟪integer-value⟫s in fpf, if they exist.

Step 3.

Case 3.1. fpf is a <fixed-point-format>.

Step 3.1.1. If the second ⟪integer-value⟫ is missing from fpf, let d=0. d MUST NOT be negative.

Step 3.1.2. Let tdt be a <data-type> containing <fixed>, <decimal>, <scale-factor>:d. N, used below, is IMPLEMENTATION-DEFINED.

Case 3.1.2.1. d = 0 and cv ≥ 0.

w MUST satisfy w>0. Let n=w, p=min(n,N), i=w−p, and j=p−1. Let pic be '(i)B(j)Z9'.

Case 3.1.2.2. d = 0 and cv < 0.

w MUST satisfy w>1. Let n=w−1, p=min(n,N), i=w−p−1, and j=p. Let pic be '(i)B(j)−9'.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3.1.2.3. $d > 0$ and $cv \geq 0$.

w MUST satisfy $w>d+1$. Let $n=w-1$, $p=\min(n,N)$, $i=w-p-1$, and $j=p-d-1$. j MUST satisfy $j \geq 0$. Let pic be '(i)B(j)Z9V.(d)9'.

Case 3.1.2.4. $d > 0$ and $cv < 0$.

w MUST satisfy $w>d+2$. Let $n=w-2$, $p=\min(n,N)$, $i=w-p-2$, and $j=p-d$. j MUST satisfy $j \geq 1$. Let pic be '(i)B(j)−9V.(d)9'.

Step 3.1.3. Attach <number-of-digits>: p; to tdt. Remove from pic any repetition factor equal to 0 and the character following it. Let pdt be the <data-type> corresponding to the ⟨picture⟩ with concrete-representation pic, defined above. (For a fuller treatment, see Section 4.6.6). Let x be the ⟨real-value⟩ calculated as $x=cv+0.5*\text{sign}(cv)*10^{-d}$. Perform CONVERT(tdt,cdt,x) to obtain cv3 and then perform EDIT-NUMERIC-PICTURE(cv3,pdt) to obtain a ⟨character-string-value⟩,csv. Return csv.

Case 3.2. fpf is a <floating-point-format>.

Step 3.2.1. If the second ⟨integer-value⟩ is missing from fpf, let s be the converted <number-of-digits> of sdt for a target <data-type> of <float>, <decimal>, and let $d=s-1$; otherwise let $s=d+1$. d MUST NOT be less than 0 and s MUST NOT be less than d.

Step 3.2.2. Let tdt be a <data-type> which has <float>, <decimal>, <number-of-digits>: s. Perform CONVERT-TO-FLOAT-DECIMAL(tdt,cdt,cv) to obtain a ⟨real-value⟩,cv3.

Step 3.2.3. There exists a unique representation of cv3 in the form $vm*10^{ve}$, where ve is an integer and either $vm=ve=0$ or $10^{s-d-1} \leq |vm| < 10^{s-d}$.

Step 3.2.4. If $vm \geq 0$, let $i=0$; otherwise, let $i=1$. Let $j=\max(s-d,1)$. If $d=0$, let $k=0$; otherwise let $k=1$. Let n be the IMPLEMENTATION-DEFINED size of the exponent field for <floating-point-format>s. Let $m=w-i-j-k-d-2-n$. (Informally, i indicates the need for a sign position in the mantissa field; j is the number of digit positions before the decimal point; k indicates the need for a decimal point in the mantissa field; m is the excess of the field-size over the requirements of the specification for the particular value). If $m<0$ or $|ve| \geq 10^n$, perform RAISE-CONDITION(<size-condition>).

Step 3.2.5. Let picx be the ⟨picture⟩ with concrete-representation '(m)B(i)−(j)9V(k).(d)9'. Remove from picx any repetition factor whose value is zero and the character following it. Let picm be the <data-type> corresponding to picx. Perform EDIT-NUMERIC-PICTURE(rvm,picm) to obtain a ⟨character-string-value⟩,cvm, where rvm is a ⟨real-value⟩ containing vm. Let pice be the <data-type> corresponding to the ⟨picture⟩ with concrete-representation '<E>S(n)9'. Perform EDIT-NUMERIC-PICTURE(rve,pice) to obtain a ⟨character-string-value⟩,cve, where rve is a ⟨real-value⟩ containing ve. Perform CONCATENATE(cum,cve) to obtain cvs. Return cvs.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.2.6 Output-string

Operation: OUTPUT-STRING(sil,fv)

Where: sil is a ⟨stream-item-list⟩,
fv is a [⟨file-value⟩].

Step 1. For each ⟨stream-item⟩,si, in sil in order, perform OUTPUT-STRING-ITEM(si,fv).

8.8.2.7 Output-string-item

Operation: OUTPUT-STRING-ITEM(si,fv)

Where: si is a ⟨stream-item⟩,
fv is a [⟨file-value⟩].

Step 1. If fv is a ⟨file-value⟩ then let fi be the ⟨file-information⟩ designated by fv.

Step 2. If fv is a ⟨file-value⟩ then perform Steps 2.1 and 2.2.

Step 2.1. Perform EVALUATE-CURRENT-COLUMN(fv) to obtain an ⟨integer-value⟩,cc, and perform EVALUATE-CURRENT-LINE(fv) to obtain an ⟨integer-value⟩,cl. Let lsz be the ⟨evaluated-linesize⟩ in fi, and psz be the ⟨evaluated-pagesize⟩ in fi, where they exist.

Step 2.2.

Case 2.2.1. fi contains <print>, cc = 0, cl = 1, and fi does not contain a ⟨page-top-flag⟩.

Attach a ⟨page-top-flag⟩ to fi, perform RAISE-IO-CONDITION(<beginpage-condition>,fv), and on normal return go to Step 2.1.

Case 2.2.2. fi contains <print>, cc = lsz, and cl = psz.

Perform OUTPUT-STREAM-ITEM(⟨stream-item⟩:⟨linemark⟩;,fv). Perform RAISE-IO-CONDITION(<endpage-condition>,fv), and on normal return go to Step 2.1.

Case 2.2.3. cc = lsz, but Case 2.2.2 does not apply.

Perform OUTPUT-STREAM-ITEM(⟨stream-item⟩:⟨linemark⟩;,fv).

Case 2.2.4. (Otherwise).

Go to Step 3.

Step 3. Perform OUTPUT-STREAM-ITEM(si,fv).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.2.8 Output-stream-item

Operation: OUTPUT-STREAM-ITEM(si,fv)

Where: si is a ⟨stream-item⟩,
fv is a [⟨file-value⟩].

Case 1. fv is a ⟨file-value⟩.

Let fi be the ⟨file-information⟩ designated by fv. fi MUST contain <stream>. Let ds be the ⟨dataset⟩ designated by the ⟨dataset-designator⟩ in fi, and let cp be the ⟨current-position⟩ in fi.

If si has ⟨pagemark⟩ or ⟨carriage-return⟩ then fi MUST contain <print>. Append si to the ⟨stream-item-list⟩ in ds. Replace the immediate component of cp by a ⟨designator⟩ to the last ⟨stream-item⟩ in ds. If fi contains <print> and si is a ⟨pagemark⟩ then add 1 to the ⟨integer-value⟩ in the ⟨page-number⟩ in fi. If fi contains a ⟨page-top-flag⟩ then delete the ⟨page-top-flag⟩ from fi.

Case 2. fv is ⟨absent⟩.

si MUST contain a ⟨symbol⟩. Let csv be the ⟨character-string-value⟩ in the current ⟨string-io-control⟩ and let m be the ⟨integer-value⟩ of the ⟨string-limit⟩ of the current ⟨string-io-control⟩. If csv contains m ⟨symbol⟩s or if m=0 then perform RAISE-CONDITION(<error-condition>).

Let s be the ⟨symbol⟩ in si. If csv contains ⟨null-character-string⟩ then replace the immediate component of csv with ⟨character-value-list⟩: ⟨character-value⟩: s. Otherwise, append ⟨character-value⟩: s; to the ⟨character-value-list⟩ of csv.

8.8.2.9 Tab

Operation: TAB(w,fv)

Where: w is an ⟨integer-value⟩,
fv is a ⟨file-value⟩.

Step 1. Let eto be the ⟨evaluated-tab-option⟩ in the ⟨file-information⟩,fi designated by fv. w MUST be non-negative. If w=0 then terminate this operation.

Step 2. Perform EVALUATE-CURRENT-COLUMN(fv) to obtain an ⟨integer-value⟩,cc.

Step 3.

Case 3.1. There are at least w ⟨integer-value⟩s, t in eto satisfying cc < t−1 < lsz, where lsz is the ⟨evaluated-linesize⟩ in fi.

Perform OUTPUT-TAB(fv) w times.

Case 3.2. (Otherwise).

Step 3.2.1. Perform SKIP(⟨integer-value⟩:1;,fv).

Step 3.2.2. If the first ⟨integer-value⟩ in eto is not 1 then perform OUTPUT-TAB(fv).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.2.9.1 Output-tab

Operation: OUTPUT-TAB(fv)

Where: fv is a ⟨file-value⟩.

Step 1. Perform EVALUATE-CURRENT-COLUMN(fv) to obtain an ⟨integer-value⟩,cc.

Step 2. Let tv be the smallest ⟨integer-value⟩ in the ⟨evaluated-tab-option⟩ of the ⟨file-information⟩ designated by fv, which is greater than cc+1.

Step 3. Let bs be a ⟨stream-item-list⟩, the (tv−cc−1) ⟨stream-item⟩s of which all contain ⁄s. Perform OUTPUT-STRING(bs,fv).

8.8.2.10 Put-line

Operation: PUT-LINE(w,fv)

Where: w is an ⟨integer-value⟩,
fv is a ⟨file-value⟩.

Step 1. Perform EVALUATE-CURRENT-LINE(fv) to obtain an ⟨integer-value⟩,cl, and perform EVALUATE-CURRENT-COLUMN(fv) to obtain an ⟨integer-value⟩,cc. Let fi be the ⟨file-information⟩ designated by fv, and let ep be the ⟨evaluated-pagesize⟩ in fi. fi MUST contain <print>.

Step 2.

Case 2.1. cc = 0, cl = 1, and fi does not contain a ⟨page-top-flag⟩.

Attach a ⟨page-top-flag⟩ to fi, perform RAISE-IO-CONDITION(<beginpage-condition>,fv), and on normal return go to Step 1.

Case 2.2. cc = 0, cl = w, and Case 2.1 does not apply.

Terminate this operation.

Case 2.3. cl < w ≤ ep.

Perform SKIP(⟨integer-value⟩:(w−cl);,fv).

Case 2.4. ep ≥ cl ≥ w and neither Case 2.1 nor Case 2.2 applies, or w > ep ≥ cl.

Perform OUTPUT-STREAM-ITEM(⟨stream-item⟩:⟨linemark⟩;,fv) (ep−cl+1) times. Perform RAISE-IO-CONDITION(<endpage-condition>,fv).

Case 2.5. (Otherwise).

Perform PUT-PAGE(fv).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.2.11 Put-page

Operation: PUT-PAGE(fv)

Where: fv is a ⟨file-value⟩.

Step 1. Perform OUTPUT-STREAM-ITEM(⟨stream-item⟩: ⟨pagemark⟩;,fv).

8.8.3 Operations Applicable to Stream I/O

8.8.3.1 Skip

Operation: SKIP(w,fv)

Where: w is an ⟨integer-value⟩,
fv is a ⟨file-value⟩.

Step 1. Let fi be the ⟨file-information⟩ designated by fv.

Case 1.1. fi contains <print>.

w MUST be greater than or equal to 0.

Case 1.2. fi does not contain <print>.

w MUST be greater than 0.

Step 2.

Case 2.1. The value of w is zero.

Perform OUTPUT-STREAM-ITEM(⟨stream-item⟩: ⟨carriage-return⟩;,fv).

Case 2.2. fi contains <input>.

Perform repeatedly INPUT-STREAM-ITEM(fv) until either w ⟨linemark⟩s have been returned or an ⟨omega⟩ is returned. In the latter case perform RAISE-IO-CONDITION(<endfile-condition>,fv).

Case 2.3. fi contains <output> but not <print>.

Perform OUTPUT-STREAM-ITEM(⟨stream-item⟩:⟨linemark⟩;,fv) w times.

Case 2.4. fi contains <print>, and w ≠ 0.

Step 2.4.1. Perform EVALUATE-CURRENT-LINE(fv) to obtain an ⟨integer-value⟩,cl, and perform EVALUATE-CURRENT-COLUMN(fv) to obtain an ⟨integer-value⟩,cc. Let ep be the ⟨evaluated-pagesize⟩ in fi.

Step 2.4.2.

Case 2.4.2.1. cc = 0, cl = 1, and fi does not contain a ⟨page-top-flag⟩.

Attach a ⟨page-top-flag⟩ to fi, perform RAISE-IO-CONDITION(<beginpage-condition>,fv), and on normal return go to Step 2.4.1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.4.2.2. ep ≥ cl+w and Case 2.4.2.1 does not apply, or cl > ep.

Perform OUTPUT-STREAM-ITEM(≤stream-item≥:≤linemark≥;,fv) w times.

Case 2.4.2.3. (Otherwise).

Perform OUTPUT-STREAM-ITEM(≤stream-item≥:≤linemark≥;,fv) (ep−cl+1) times. Perform RAISE-IO-CONDITION(<endpage-condition>,fv).

8.8.3.2 Evaluate-current-column

The current column of a line of a stream file is defined as the number of ≮symbol≯s preceding the current position in the line. It is zero at the start of a line. The <expression> in a <column-format>, however, is 1 at the start of a line.

Operation: EVALUATE-CURRENT-COLUMN(fv)

Where: fv is a ≤file-value≥.

Result: an ≤integer-value≥.

Step 1. Let fi be the ≤file-information≥ designated by fv, and ds the ≤dataset≥ designated by the ≤dataset-designator≥ in fi. Let cp be the node in ds designated by the ≤current-position≥ in fi.

Step 2.

Case 2.1. cp contains an ≤alpha≥, a ≤linemark≥, a ≤pagemark≥ , or a ≤carriage-return≥.

Return ≤integer-value≥: 0.

Case 2.2. cp contains a ≮symbol≯ or an ≤omega≥.

Let p be that ≤alpha≥, ≤linemark≥, ≤pagemark≥ , or ≤carriage-return≥ in ds preceding cp and such that any ≤stream-item≥s in ds between p and cp contain ≮symbol≯s. Let n be the number of ≮symbol≯s between p and cp, inclusive. Return ≤integer-value≥: n.

8.8.3.3 Evaluate-current-line

Operation: EVALUATE-CURRENT-LINE(fv)

Where: fv is a ≤file-value≥.

Result: an ≤integer-value≥.

Step 1. Let fi be the ≤file-information≥ designated by fv, and ds the ≤dataset≥ designated by the ≤dataset-designator≥ in fi. Let cp be the node in ds designated by the ≤current-position≥ in fi.

Step 2.

Case 2.1. cp contains an ≤alpha≥ or a ≤pagemark≥.

Return ≤integer-value≥: 1.

Case 2.2. (Otherwise).

Let p be that ≤alpha≥ or ≤pagemark≥ in ds preceding cp such that all ≤stream-item≥s in ds between p and cp contain ≮symbol≯s, ≤linemark≥s, or ≤carriage-return≥s. Let n be the number of ≤linemark≥s between p and cp, inclusive. Return ≤integer-value≥: n+1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.3.4 Establish-next-data-item

Operation: ESTABLISH-NEXT-DATA-ITEM

Result: a ⟨current-scalar-item⟩ or ⟨none⟩.

Step 1. In the current ⟨block-control⟩ let cdld be the last ⟨data-list-indicator⟩, and let cdid be the last ⟨data-item-indicator⟩.

Step 2.

Case 2.1. cdid is ⟨undefined⟩.

Set cdid to designate the first component of the list designated by cdld.

Case 2.2. cdid designates an immediate component of the list designated by cdld, but not the last component.

Set cdid to designate the next immediate component of the list designated by cdld.

Case 2.3. cdid designates the last immediate component of the list designated by cdld.

If cdid designates a ⟨current-scalar-item⟩ then delete the current ⟨current-scalar-item-list⟩ and go to Step 4. If the current ⟨data-item-control-list⟩ contains precisely one ⟨data-item-control⟩ then return ⟨none⟩. Otherwise perform ADVANCE-CONTROLLED-GROUP to obtain t. If t is ⟨false⟩ then go to Step 4. Otherwise, let cdid designate the first immediate component of the list designated by cdld.

Step 3.

Case 3.1. The node designated by cdid immediately contains an <expression>,e.

Perform EVALUATE-EXPRESSION(e) to obtain an ⟨aggregate-value⟩,av, having n ⟨basic-value⟩s. Let csil be a ⟨current-scalar-item-list⟩ having n elements whose *i*th element is a ⟨current-scalar-item⟩ containing the *i*th ⟨basic-value⟩ in av, bv[i], and the <data-type> corresponding to bv[i] in the <data-description> immediately contained in e. Attach csil to the current ⟨block-control⟩. Append to the current ⟨block-control⟩ a

⟨data-item-control⟩:
⟨data-list-indicator⟩ designating csil
⟨data-item-indicator⟩ designating the first element of csil.

Return the first element of csil.

Case 3.2. The node designated by cdid immediately contains a <variable-reference>,tr, that is contained in an <input-target>.

Perform EVALUATE-TARGET-REFERENCE(tr) to obtain an ⟨evaluated-target⟩,et.

Step 3.2.1. Let g be the ⟨generation⟩ immediately contained in et. Perform EXPAND-GENERATION(g) to obtain a ⟨generation-list⟩,ggl, having n elements. Let csil be a ⟨current-scalar-item-list⟩ having n elements whose *i*th element is ⟨current-scalar-item⟩: ⟨evaluated-target⟩: gg[i];;, where gg[i] is the *i*th element of ggl.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3.2.2. Attach csil to the current ⟨block-state⟩. Append to the ⟨data-item-control-list⟩ in the current ⟨block-control⟩ a

⟨data-item-control⟩:
⟨data-list-indicator⟩ designating csil
⟨data-item-indicator⟩ designating the first element of csil.

Return the first element of csil.

Case 3.3. The node designated by cdid immediately contains a <variable-reference>,vr, that is contained in an <output-source>.

Step 3.3.1. Perform EVALUATE-VARIABLE-REFERENCE(vr) to obtain a ⟨generation⟩,g. Let edd be the ⟨evaluated-data-description⟩ in g. Perform EXPAND-EDD(edd) to obtain an ⟨evaluated-data-description-list⟩,eddl having n elements. Let csil be a ⟨current-scalar-item-list⟩ having n elements whose *i*th element contains:

the *j*th ⟨basic-value⟩ in the ⟨allocation-unit⟩ designated by the ⟨allocation-unit-designator⟩ in g, where j is the *i*th ⟨storage-index⟩ in g;
the <data-type> from the *i*th ⟨evaluated-data-description⟩ of eddl.

Step 3.3.2. Attach csil to the current ⟨block-control⟩. Append to the current ⟨data-item-control-list⟩ a

⟨data-item-control⟩:
⟨data-list-indicator⟩ designating csil
⟨data-item-indicator⟩ designating the first element of csil.

Return the first element of csil.

Case 3.4. The node designated by cdid immediately contains a tree of the form <do-spec>,dsp.

Let dld1 be the first immediate component of the node designated by cdid. Let did1 be the first immediate component of dld1. Append to the current ⟨data-item-control-list⟩ a

⟨data-item-control⟩:
⟨data-list-indicator⟩:
⟨designator⟩ designating dld1;
⟨data-item-indicator⟩:
⟨designator⟩ designating did1.

Let cdid be the current ⟨data-item-indicator⟩. Perform ESTABLISH-CONTROLLED-GROUP(dsp) to obtain t. If t is ⟨true⟩ then go to Step 3.

Case 3.5. The node designated by cdid is a ⟨current-scalar-item⟩,csi.

Return csi.

Step 4. If the current ⟨data-item-control-list⟩ contains more than one ⟨data-item-control⟩ then delete the last of them and go to Step 1. Otherwise return ⟨none⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.3.4.1 Expand-edd

Operation: EXPAND-EDD(edd)

Where: edd is an ⟨evaluated-data-description⟩.

Result: an ⟨evaluated-data-description-list⟩.

Note: edd will not contain a <union-data-description> because of constraints in A135, <input-target>, and A140, <output-source>.

Case 1. edd immediately contains a <data-description> immediately containing a <dimensioned-data-description>,ddd.

Let bpl be the <bound-pair-list> in ddd, let lb[i] and ub[i] be respectively the values of the *i*th <lower-bound> and <upper-bound> in bpl and let ne be the number of elements in bpl. Let n be the integer

$$\prod_{i=1}^{ne}(ub[i]-lb[i]+1).$$

Let edd1 be an ⟨evaluated-data-description⟩: <data-description>: the immediate subtree of the <element-data-description> in ddd. Perform EXPAND-EDD(edd1) to obtain an ⟨evaluated-data-description-list⟩,eddl. Let eddln be an ⟨evaluated-data-description-list⟩ with, in order, n replications of the subtrees of eddl. Return eddln.

Case 2. edd immediately contains a <data-description> immediately containing a <structure-data-description>,sdd.

For each <data-description>,dd[i] in sdd perform EXPAND-EDD(⟨evaluated-data-description⟩: dd[i]) to obtain an ⟨evaluated-data-description-list⟩,eddl[i]. Let eddln be an ⟨evaluated-data-description-list⟩ with, in order, the subtrees of these eddl[i]s. Return eddln.

Case 3. edd immediately contains a <data-description> immediately containing an <item-data-description>.

Return ⟨evaluated-data-description-list⟩: edd.

8.8.3.4.2 Expand-generation

Operation: EXPAND-GENERATION(g)

Where: g is a ⟨generation⟩.

Result: a ⟨generation-list⟩.

Step 1. Let aud be the ⟨allocation-unit-designator⟩ in g. Let edd be the ⟨evaluated-data-description⟩ in g. Let sil be the ⟨storage-index-list⟩ in g. Let n be the number of elements in sil. Perform EXPAND-EDD(edd) to obtain an ⟨evaluated-data-description-list⟩,eddl. Let gg be a ⟨generation-list⟩ without subnodes. For i=1,...,n, let si[i] be a copy of the *i*th element of sil, let edd[i] be a copy of the *i*th element of eddl, and attach to gg a ⟨generation⟩: edd[i] au d ⟨storage-index-list⟩: si[i].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.3.5 Establish-next-format-item

Operation: ESTABLISH-NEXT-FORMAT-ITEM

Result: a <format-item>.

Step 1. Let fc be the last ⩽format-control⩾ in the current ⩽format-control-list⩾. In fc, let flp be the <format-specification-list-designator>, let fli be the immediate subnode of the ⩽format-list-index⩾, and, where they exist, let fiv be the immediate subnode of the ⩽format-iteration-value⩾, and let fii be the immediate subnode of the ⩽format-iteration-index⩾. Let fsl be the <format-specification-list> designated by flp, and let n be the number of immediate components of fsl.

Step 2.

Case 2.1. fli is less than n.

Add 1 to fli, giving a value, m.

Case 2.1.1. The *m*th element of fsl is a <format-item>,fi.

Let cfi be a copy of fi. Perform EVALUATE-FORMAT-ITEM(cfi) to obtain a <format-item>,efi. Return efi.

Case 2.1.2. The *m*th element of fsl is a <format-iteration>,fi.

Let e be the <expression> in the <format-iteration-factor> immediately contained in fi. Perform EVALUATE-FORMAT-EXPRESSION(e) to obtain an ⩽integer-value⩾,intg. intg MUST NOT be negative. If intg > 0 then append to the current ⩽format-control-list⩾ a

```
⩽format-control⩾:
   <format-specification-list-designator> designating
   the <format-specification-list> in the mth
   element of the list designated by flp
   ⩽format-list-index⩾:
      ⩽integer-value⩾:
         0;;
   ⩽format-iteration-value⩾:
      intg;
   ⩽format-iteration-index⩾:
      ⩽integer-value⩾:
         1.
```

Go to Step 1.

Case 2.2. fli equals n, and fc is the only ⩽format-control⩾ in the current ⩽format-control-list⩾.

Set fli to 0 and go to Step 1.

Case 2.3. fli equals n, and there are at least two ⩽format-control⩾s in the current ⩽format-control-list⩾.

Case 2.3.1. fii is less than fiv.

Add 1 to fii, set fli to 0 and go to Step 1.

Case 2.3.2. fii equals fiv.

Delete fc, and go to Step 1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.8.3.6 Evaluate-format-item

When a <format-item> contains a <data-format> or a <control-format>, all <expression>s contained in it are evaluated, and a tree having the same structure as the <format-item> (but with ⟪integer-value⟫s replacing the <expression>s) is returned. When a <format-item> contains a <remote-format>, this <remote-format> specifies a <format-statement>; designators of the <format-specification-list> of the statement, and of the <format-statement> itself, are placed in the current ⟪format-control-list⟫. (The latter designator allows the correct environment to be established for the evaluation of <expression>s.) The operation ESTABLISH-NEXT-FORMAT-ITEM is then re-invoked to scan the <format-specification-list>.

Operation: EVALUATE-FORMAT-ITEM(fi)

Where: fi is a <format-item>.

Result: a <format-item>.

Case 1. fi contains a <control-format> or a <data-format>.

Let cfi be a copy of fi. For each <expression>,e, in cfi, chosen in any order, perform EVALUATE-FORMAT-EXPRESSION(e) to obtain an ⟪integer-value⟫,iv and replace e by iv. Return cfi.

Case 2. fi contains a <remote-format>,rf.

Step 2.1. If there is in the current ⟪format-control-list⟫ a ⟪remote-block-state⟫, then let rbs be the last such ⟪remote-block-state⟫. Attach a copy of rbs to the current ⟪block-control⟫.

Step 2.2. Let vr be the <value-reference> of rf. Perform EVALUATE-VALUE-REFERENCE(vr) to obtain an ⟪aggregate-value⟫ containing a single ⟪format-value⟫,fv.

Step 2.3. If there exists a ⟪remote-block-state⟫,rbs in the current ⟪block-control⟫ then delete rbs. fv contains a ⟪block-state-designator⟫,bsd and a <format-statement-designator>,fsd. The current ⟪format-control-list⟫ MUST NOT contain a ⟪format-control⟫ with a <format-statement-designator> equal to fsd. The current ⟪block-state-list⟫ MUST contain a ⟪block-state⟫ designated by bsd.

Step 2.4. Let fsl designate the <format-specification-list> in the node designated by fsd. Append to the current ⟪format-control-list⟫ a

⟪format-control⟫:
<format-specification-list-designator>: fsl;
⟪format-list-index⟫:
⟪integer-value⟫:
0;;
fsd
⟪remote-block-state⟫:
bsd.

Step 2.5. Perform ESTABLISH-NEXT-FORMAT-ITEM to obtain a <format-item>,efi. Return efi.

8.8.3.6.1 Evaluate-format-expression

An <expression> which appears in a <format-specification-list> is evaluated and converted to an ⟪integer-value⟫. However, when such an <expression> occurs in a <format-statement> (and is accessed via a <remote-format>), the necessary environment MUST be set up for its evaluation. This is achieved by temporarily placing a ⟪remote-block-state⟫ in the current ⟪block-control⟫.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: EVALUATE-FORMAT-EXPRESSION(e)

Where: e is an <expression>.

Result: an ⟨integer-value⟩.

Case 1. There exists in the current ⟨format-control-list⟩ a ⟨remote-block-state⟩.

Let rbs be the last such ⟨remote-block-state⟩. Attach a copy of rbs to the current ⟨block-control⟩. Perform EVALUATE-EXPRESSION-TO-INTEGER(e) to obtain an ⟨integer-value⟩,intg. Delete the ⟨remote-block-state⟩ from the current ⟨block-control⟩. Return intg.

Case 2. (Otherwise).

Perform EVALUATE-EXPRESSION-TO-INTEGER(e) to obtain an ⟨integer-value⟩,intg. Return intg.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9. Expressions and Conversion

9.1 Introduction

This chapter defines operations that are involved in the evaluation of expressions. Included are definitions for built-in functions and the rules for data conversion. The main Sections are:

9.2 Aggregate Expressions
9.3 Prefix Operators
9.4 Infix Operators
9.5 Built-in functions
9.6 Conversion

9.2 Aggregate Expressions

9.2.1 Scalar and Aggregate Types

9.2.1.1 Aggregate Type of a Data Description

An ⟪aggregate-type⟫ is an alternative way of specifying the part of a <data-description> which describes the structuring of elements into an aggregate. For each <data-description> there is an *associated* ⟪aggregate-type⟫ determined as follows:

Ignoring all <identifier-list> immediate components of <structure-data-description>s and all components of <item-data-description>s, the shapes of the trees for the <data-description> and the ⟪aggregate-type⟫ are identical, and categories correspond as follows:

<data-description>	⟪aggregate-type⟫
<dimensioned-data-description>	⟪dimensioned-aggregate-type⟫
<element-data-description>	⟪element-aggregate-type⟫
<bound-pair-list> and all its components	<bound-pair-list> and equal components if ⟪integer-value⟫s otherwise <asterisk> components
<structure-data-description>	⟪structure-aggregate-type⟫
<union-data-description>	⟪union-aggregate-type⟫
<member-description>	⟪member-aggregate-type⟫
<item-data-description>	⟪scalar⟫

In many contexts, properties of <data-description>s are stated in terms of properties of their associated ⟪aggregate-type⟫s without explicitly building the corresponding tree or even using the term "associated".

For example: "The result has ⟪aggregate-type⟫: ⟪scalar⟫;" is equivalent to "the result <data-description> immediately contains an <item-data-description>".

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.1.2 Scalar Elements

The *scalar-elements* of an ⟨aggregate-value⟩ are the components of its ⟨basic-value-list⟩; the *scalar-elements* of a ⟨generation⟩ are the components of its ⟨storage-index-list⟩.

The *number-of-scalar-elements* is defined for a <data-description> by the operation SCALAR-ELEMENTS-OF-DATA-DESCRIPTION (Section 7.2.1), and applies equally to a <data-description>, a ⟨generation⟩ containing that <data-description>, the associated ⟨aggregate-type⟩ of that <data-description>, and an ⟨aggregate-value⟩ containing that ⟨aggregate-type⟩.

9.2.1.3 Treatment of Scalars

To facilitate the systematic treatment of expression evaluation and assignment, operations such as EVALUATE-EXPRESSION and EVALUATE-BUILTIN-FUNCTION-REFERENCE are defined so that they return or accept ⟨aggregate-value⟩s in all cases. There are, of course, many contexts in which a value's ⟨aggregate-type⟩ is known, a priori, to immediately contain ⟨scalar⟩, but these are treated as degenerate cases of the general mechanism.

9.2.1.4 Compatibility

Compatibility between ⟨aggregate-type⟩s is defined below. In addition the *common* ⟨aggregate-type⟩ of a set of *compatible* ⟨aggregate-type⟩s is also defined.

Let t[1],...,t[n] be a set of ⟨aggregate-type⟩s. If n=1, the set is compatible, and its common ⟨aggregate-type⟩ is t[1]. If n>2, the set is compatible if and only if t[2],...,t[n] are compatible and t[1] is compatible with the common ⟨aggregate-type⟩,ct, of t[2],...,t[n]. In this case the common ⟨aggregate-type⟩ of t[1],...,t[n] equals the common ⟨aggregate-type⟩ of t[1] and ct.

Finally, if n=2, the set is compatible if and only if t[1] and t[2] are equal except that where a <bound-pair>: <asterisk>; occurs in one, the other may have a <bound-pair> containing a <lower-bound> and <upper-bound>. The common ⟨aggregate-type⟩ of t[1] and t[2] is t[1] except that where a <bound-pair> in t[1] has <asterisk>, the corresponding <bound-pair> of t[2] is used.

9.2.1.5 Correspondence

The following two sections extend the notion of node correspondence (see Section 1.6.1.1) to apply to scalar-elements of ⟨aggregate-value⟩s and ⟨generation⟩s whose (associated) ⟨aggregate-type⟩s are compatible, and <data-type>s and <union-data-description>s in <data-description>s whose associated ⟨aggregate-type⟩s are compatible. The extended notion of correspondence is used extensively in defining aggregate operations and in defining the result <data-description>s of <expression>s.

9.2.1.5.1 Correspondence of Scalar Elements

Let atx and aty be compatible ⟨aggregate-type⟩s, with number-of-scalar-elements nx and ny, respectively (by construction, nx equals ny). Let x[1],...,x[nx] and y[1],...,y[ny] be sequences of scalar-elements. (For example, x[1],...,x[nx] might be the scalar-elements of an ⟨aggregate-value⟩ whose ⟨aggregate-type⟩ is atx). Then x[i] corresponds to y[i] for i=1,...,nx.

9.2.1.5.2 Correspondence of Data Types

Let x be an ⟨aggregate-value⟩ or a ⟨generation⟩, and let dd be a <data-description> whose associated ⟨aggregate-type⟩ equals the ⟨aggregate-type⟩ of x. Let x[1], x[2],... be the scalar-elements of x. Then the <data-type>s in dd correspond to the x[1], x[2],... as follows:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 1. dd immediately contains an <item-data-description>.

The single <data-type> simply contained in dd corresponds to the single scalar-element x[1].

Case 2. dd immediately contains a <structure-data-description> or <union-data-description>.

Let the <member-description>s simply contained in dd be md[1],...,md[p]. Let n[i], $1 \leq i \leq p$, be the number-of-scalar-elements of md[i]. Let s[i] be 0 when i=1. Let s[i] be the sum of n[j], $1 \leq j < i$ for $2 \leq i \leq p$. The <data-type>s in md[i] correspond to the scalar-elements x[s[i]+j], $1 \leq j \leq n[i]$, for $1 \leq i \leq p$, the correspondence being determined by the ⩽aggregate-type⩾ of md[i].

Case 3. dd immediately contains a <dimensioned-data-description>.

Let ed be the <element-data-description> simply contained in dd, and let n be the number-of-scalar-elements of the ⩽aggregate-type⩾ of ed. Then for i=0,1,... the <data-type>s in ed correspond to the scalar-elements x[n*i+j], $1 \leq j \leq n$, the correspondence being determined by the ⩽aggregate-type⩾ of ed.

Now the correspondence is easily extended to cover cases where the ⩽aggregate-type⩾s of dd and x are merely compatible. Suppose that all <bound-pair>s in dd contain ⩽integer-value⩾s, and let y be any ⩽aggregate-value⩾ whose ⩽aggregate-type⩾ is the same as that of dd. Then a <data-type> in dd corresponds to a scalar-element in x if and only if they both correspond to some scalar-element in y. If some <bound-pair>s in dd have <asterisk>s, consider instead a <data-description> that is equal to dd except for <bound-pair>s and whose associated ⩽aggregate-type⩾ is the same as that of x.

Finally the correspondence can be established between <data-type>s of two <data-description>s, dd1 and dd2, as follows. Let x be an ⩽aggregate-value⩾ whose ⩽aggregate-type⩾ equals the common ⩽aggregate-type⩾ of dd1 and dd2 except that all <bound-pair>s have <lower-bound> equal to 1 and <upper-bound> equal to 1. Then a <data-type> in dd1 corresponds to a <data-type> in dd2 if and only if both correspond to some scalar-element in x.

9.2.1.5.3 Correspondence of Union Data Descriptions

Let dd1 and dd2 be two compatible <data-description>s. Let ud1 and ud2 be two <union-data-description>s in dd1 and dd2 respectively. Then ud1 and ud2 correspond if the first <data-type> of ud1 corresponds to the first <data-type> of ud2. (Correspondence of the first <data-type>s guarantees correspondence of the remaining <data-type>s of ud1 and ud2).

9.2.1.6 Generate-scalar-result

Most operations that return a scalar ⩽aggregate-value⩾ work in the same general way:

The operands are evaluated. The scalar-element of the result is determined by performing a sequence of steps involving the scalar-elements of the operand values.

The various scalar operations differ only in the details of this sequence of scalar steps. To simplify the description of these scalar operations a special macro operation, GENERATE-SCALAR-RESULT, is used.

Macro Operation: GENERATE-SCALAR-RESULT

A reference to this macro operation always occurs as the first statement in the body of an operation, f. The remainder of the body will be a sequence of Steps numbered 1,...,nstep, where $nstep \geq 1$. To perform f, substitute these Steps into the following macro body immediately after Step 3, renumber them and their corresponding Steps and Cases as Step 3.1, Step 3.2, ..., Step 3.nstep, and, in Step 3, replace "nstep" by its actual value. Renumber references to these Steps and Cases correspondingly. Then interpret the expanded macro body as an operation body in the normal way.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Macro Body:

Step 1. For each operand, op[j+1], j=1,...,(number of operands)−1, taken in any order, perform EVALUATE-EXPRESSION(op[j+1]) to obtain an ⟨aggregate-value⟩,val[j].

Step 2. Let *scalar-result-type* be that <data-type> of rdd. For each operand, op[j+1], let the *scalar* <data-type> of op[j+1] be the <data-type> of op[j+1], and let the *scalar-value* of op[j+1] be the scalar-element in val[j].

Step 3. Perform Steps 3.1 through 3.nstep to obtain *scalar-result*. Let aval equal scalar-result.

[Replace this line with the Steps following the macro reference, renumbered as above.]

Step 4. Return an ⟨aggregate-value⟩ whose ⟨aggregate-type⟩ immediately contains ⟨scalar⟩, and whose ⟨basic-value-list⟩ contains aval.

Note:

1. An operation referencing GENERATE-SCALAR-RESULT always has a first operand, rdd, which is a <data-description>. Its other operands will be <expression>s or <argument>s. To refer to the scalar-elements of the operands, the result, and their respective <data-type>s, it uses the four special terms defined in Steps 2 and 3, namely: scalar-value, scalar <data-type>, scalar-result, and scalar-result-type. In the expanded macro body, scalar-value, scalar-result, and scalar-result-type are local variables, together with nse, aval, and val[j+1].

2. The concepts of scalar-value, scalar <data-type>, scalar-result, and scalar-result-type are also used in an extended sense in the section on "Attributes" and "Constraints", where they refer to the parts of a <data-description> being constructed or checked by the Translator, rather than to parts of rdd.

9.2.2 Integer Type

The term *integer-type* means a <data-type> which has <arithmetic> containing <fixed> <binary> <number-of-digits>: n; and <scale-factor>: 0. n is IMPLEMENTATION-DEFINED and depends on the particular context in which the term is used.

9.2.2.1 Evaluate-expression-to-integer

Operation: EVALUATE-EXPRESSION-TO-INTEGER(e)

Where: e is an <expression>.

Result: an ⟨integer-value⟩.

Step 1. Perform EVALUATE-EXPRESSION(e) to obtain an ⟨aggregate-value⟩,av. Let bv be the ⟨basic-value⟩ in av. Let sdt be the <data-type> of e.

Step 2. Let tdt be a <data-type> which is integer-type (Section 9.2.2). Perform CONVERT(tdt,sdt,bv) to obtain a ⟨real-value⟩,rv.

Step 3. Return an ⟨integer-value⟩ with the same component as rv.

9.2.3 Derived Data Types

In general, a scalar <data-type> of an expression or a target <data-type> for a conversion operation is derived from other <data-type>s. Special terminology is introduced for the most common cases.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.3.1 Derived Base and Scale

Let S be a set of one or more <data-type>s which have <computational-type>.

The *derived* <scale> of S has <float> if any element of S has <arithmetic> (including <arithmetic> in <pictured-numeric>) with <scale>: <float>. Otherwise the *derived* <scale> has <fixed>.

The *derived* <base> of S has <binary> if any element of S has

1. <arithmetic> with <base>: <binary>; or
2. <string> with <string-type>: <bit>;

and either

1. the derived <scale> is <float> or
2. no element of S has <arithmetic> (including <arithmetic> in <pictured-numeric>) with <base>: <decimal>; and a <scale-factor>: q;, where q is greater than zero.

Otherwise the *derived* <base> has <decimal>.

When the set S contains at least two <data-type>s, the terminology *derived common* <base> and *derived common* <scale>, will generally be used.

9.2.3.2 Converted Precision

Given a source <data-type> which has <computational-type>, a target <base>, and a target <scale>, the source <data-type> has associated with it a *converted* <precision>, with a *converted* <number-of-digits> and a *converted* <scale-factor>. For a source <data-type>which has <arithmetic> (including <arithmetic> in <pictured-numeric>), the converted <precision>, converted <number-of-digits> and converted <scale-factor> are defined in Figure 14.

The converted <scale-factor> MUST be zero if the target <base> has <binary> and greater than or equal to zero if the <base> has <decimal>.

For a source <data-type> which has <string> with <string-type>: <bit>; the converted <precision> is the same as for a source <data-type> which has <arithmetic> with <base>: <binary>;, <scale>: <fixed>;, <scale-factor>: 0; and <number-of-digits>: N;, where N is the maximum <number-of-digits> allowed for <binary> and <fixed>.

For a source <data-type> which has <string> with <string-type>: <character>;, the converted <precision> is the same as for a source <data-type>which has <arithmetic> with <base>: <decimal>;, <scale>: <fixed>;, <scale-factor>: 0; and <number-of-digits>: N;, where N is the maximum <number-of-digits> allowed for <decimal>and <fixed>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Source <base> and <scale>	Target <base> and <scale>	Converted Precision
Binary Fixed	Binary Fixed	p'=p q'=q
	Decimal Fixed	p'=min(ceil(p/$\log_2(10)$)+1,N) q'=ceil(q/$\log_2(10)$)
	Binary Float	p'=min(p,N)
	Decimal Float	p'=min(ceil(p/$\log_2(10)$),N)
Decimal Fixed	Binary Fixed	p'=min(ceil(p*$\log_2(10)$)+1,N) q'=ceil(q*$\log_2(10)$)
	Decimal Fixed	p'=p q'=q
	Binary Float	p'=min(ceil(p*$\log_2(10)$),N)
	Decimal Float	p'=min(p,N)
Binary Float	Binary Fixed	
	Decimal Fixed	
	Binary Float	p'=p
	Decimal Float	p'=min(ceil(p/$\log_2(10)$),N)
Decimal Float	Binary Fixed	
	Decimal Fixed	
	Binary Float	p'=min(ceil(p*$\log_2(10)$),N)
	Decimal Float	p'=p

Figure 14. Table of Converted Precisions as a Function of Target and Source Attributes.

Each table entry shows the <number-of-digits>, p', of the converted <precision>, and the <scale-factor>, q', of the converted <precision> as functions of the <number-of-digits>, p, and <scale-factor>, q, of the source <precision>. Those table entries left blank are for cases which never arise in the language definition. N is the maximum <number-of-digits> allowed for the target <base> and <scale>.

In many cases, evaluation of an expression requires conversion of the values of its subexpressions to target <data-type>s which have <arithmetic> and whose <base> and <scale> are the same, but whose <precision>s are determined individually using the rules for converted <precision>.

9.2.3.3 Derived String Type

Let S be a set of one or more <data-type>s which have <computational-type>.

The *derived* <string-type> of S has <bit> if all the types in S have <string> with <string-type>: <bit>. Otherwise the *derived* <string-type> has <character>.

When the set S contains at least two <data-type>s, the terminology *derived common* <string-type> will generally be used.

9.2.3.4 Further Definitions for Character and Bit Strings

In the definitions of operations whose result <data-type> is either <character> or <bit>, the following definitions are used:

1. The *length* of a ⟪character-string-value⟫ or of a ⟪bit-string-value⟫ is zero if it contains a ⟪null-character-string⟫ or a ⟪null-bit-string⟫; otherwise it is equal to the number of components in its ⟪character-value-list⟫ or ⟪bit-value-list⟫.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2. A *null-string* is a ⟨character-string-value⟩ containing ⟨null-character-string⟩ when the result <string-type> has <character> or a ⟨bit-string-value⟩ containing a ⟨null-bit-string⟩ when the result <string-type> has <bit>.

3. A *string* is a ⟨character-string-value⟩ when the result <string-type> has <character>, or a ⟨bit-string-value⟩ when the result <string-type> has <bit>.

4. A *substring,* c, of a string, st, is a string of the same type as st. It either contains a null-string, or it contains a contiguous set of ⟨character-value⟩s or ⟨bit-value⟩s from st in the same order as in st.

9.2.4 Arithmetic Results

In the definitions of the evaluation for many <infix-expression>s and <builtin-function-reference>s the final step in determining a scalar-result is to take an arithmetic value computed in some natural way (e.g. by addition of two numbers) and adjust it in a way that represents the special properties of arithmetic in PL/I. This adjustment is made by the operation ARITHMETIC-RESULT, which is defined below. This operation may cause certain conditions to be raised. Note, however, there are other points in a <program> execution where the same conditions can be raised.

Operation: ARITHMETIC-RESULT(v,rt)

Where: v is a ⟨real-value⟩,
rt is a <data-type> that has <arithmetic>.

Result: a ⟨real-value⟩.

Let b be the numerical-base of rt, m be the <number-of-digits> of rt, n be the <scale-factor> of rt (if rt has <fixed>), and N be the maximum <number-of-digits> allowed for the <base> and <scale> of rt.

Case 1. rt has <fixed>.

Determine a ⟨real-value⟩,v' as follows:

$v' = (b^{-n})*\text{floor}(v*b^{n})$, if $v \geq 0$,
$v' = (b^{-n})*\text{ceil}(v*b^{n})$, if $v < 0$.

Case 1.1. This operation was invoked by ADD-BIF, DIVIDE-BIF, MULTIPLY-BIF, or SUBTRACT-BIF.

Case 1.1.1. $|v'| \geq b^{m-n}$ and $|v'| < b^{N-n}$.

Perform RAISE-CONDITION(<size-condition>).

Case 1.1.2. $|v'| \geq b^{N-n}$.

Optionally perform RAISE-CONDITION(<fixedoverflow-condition>); otherwise, perform RAISE-CONDITION(<size-condition>).

Case 1.1.3. (Otherwise).

Return v'.

Case 1.2. (Otherwise).

If $|v'| \geq b^{N-n}$, then optionally perform RAISE-CONDITION(<fixedoverflow-condition>); otherwise, return v'.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2. rt has <float>.

Step 2.1.

Case 2.1.1. This operation was invoked by PREFIX-MINUS, INFIX-ADD, INFIX-SUBTRACT, INFIX-MULTIPLY, ABS-BIF, ADD-BIF, CEIL-BIF , FLOOR-BIF, MULTIPLY-BIF, SUBTRACT-BIF, TRUNC-BIF, or ADVANCE-CONTROLLED-GROUP.

Perform CHOPPED-RESULT(v,rt) to obtain a ⟨real-value⟩,v'.

Case 2.1.2. This operation was invoked by INFIX-DIVIDE, or DIVIDE-BIF.

Perform COMPOSITE-RESULT(v,rt) to obtain a ⟨real-value⟩,v'.

Case 2.1.3. This operation was invoked by ROUND-BIF.

Perform ROUNDED-RESULT(v,rt) to obtain a ⟨real-value⟩,v'.

Case 2.1.4. (Otherwise).

Perform UNCONSTRAINED-RESULT(v,rt) to obtain a ⟨real-value⟩,v'.

Step 2.2. Return v'.

9.2.4.1 Conditions in Expressions

Certain properties are common to a number of <expression>s that yield arithmetic results. If a condition may occur at the point where a result is normally obtained, then the operation ARITHMETIC-RESULT is invoked. If a condition may occur during the course of evaluation of a mathematical built-in function, then the operation CONDITIONS-IN-ARITHMETIC-EXPRESSION is invoked.

Operation: CONDITIONS-IN-ARITHMETIC-EXPRESSION(v)

Where: v is a <data-type>.

Step 1. Optionally perform RAISE-CONDITION(<error-condition>).

9.2.4.2 Operations Used in the Model of Floating-Point Arithmetic

9.2.4.2.1 Chopped-result

Operation: CHOPPED-RESULT(x,dt)

Where: x is a ⟨real-value⟩,
dt is a <data-type> that has <float>.

Result: a ⟨real-value⟩.

Step 1. Let vx be the ⟨real-number⟩ in x.

Step 2. Peform HIGHEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⟨real-number⟩,lambda. Perform LOWEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⟨real-number⟩,sigma.

Step 3.

Case 3.1. | vx | > lambda.

Perform either Step 3.1.1 or Step 3.1.2.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 3.1.1. Perform RAISE-CONDITION(<overflow-condition>).

Step 3.1.2. Let r be an implementation-dependent ⩽real-number⩾ that approximates vx.

Case 3.2. | vx | < sigma and vx ≠ 0.

Perform either Step 3.2.1 or Step 3.2.2.

Step 3.2.1. Perform RAISE-CONDITION(<underflow-condition>). Let r be a ⩽real-number⩾: 0.

Step 3.2.2. Let r be an implementation-dependent ⩽real-number⩾ with the same sign as vx such that 0 ≤ | r | ≤ sigma.

Note: Normally, an implementation would be expected to either raise underflow or return a value less than sigma but strictly greater than zero. However, since this standard deliberately does not require an implementation or its underlying hardware to detect underflow, a zero value is permitted in this step.

Case 3.3. (Otherwise).

Step 3.3.1. Perform TEST-MODEL-NUMBER(vx,dt) to obtain tv.

Case 3.3.1.1. tv is ⩽true⩾.

Let r = vx.

Case 3.3.1.2. tv is ⩽false⩾.

Perform LOWER-MODEL-NUMBER(vx,dt) to obtain a ⩽real-number⩾,lb. Perform HIGHER-MODEL-NUMBER(vx,dt) to obtain a ⩽real-number⩾,ub. Let r be an implementation-dependent ⩽real-number⩾ such that lb ≤ r ≤ ub.

Step 4. Return a ⩽real-value⩾: r.

9.2.4.2.2 Composite-result

Operation: COMPOSITE-RESULT(x,dt)

Where: x is a ⩽real-value⩾,
dt is a <data-type> that has <float>.

Result: a ⩽real-value⩾.

Step 1. Let vx be the ⩽real-number⩾ in x.

Step 2. Peform HIGHEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⩽real-number⩾,lambda. Perform LOWER-MODEL-NUMBER(lambda,dt) to obtain a ⩽real-number⩾,lambda'. Perform LOWEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⩽real-number⩾,sigma. Perform HIGHER-MODEL-NUMBER(sigma,dt) to obtain a ⩽real-number⩾,sigma'.

Step 3.

Case 3.1. | vx | > lambda'.

Perform either Step 3.1.1 or Step 3.1.2.

Step 3.1.1. Perform RAISE-CONDITION(<overflow-condition>).

Step 3.1.2. Let r be an implementation-dependent ⩽real-number⩾ that approximates vx.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3.2. | vx | < sigma' and vx ≠ 0.

Perform either Step 3.2.1 or Step 3.2.2.

Step 3.2.1. Perform RAISE-CONDITION(<underflow-condition>). Let r be a ⟨real-number⟩: 0.

Step 3.2.2. If | vx | ≤ sigma, let ub = sigma'; otherwise, perform HIGHER-MODEL-NUMBER(sigma',dt) to obtain a ⟨real-number⟩,ub. Let r be an implementation-dependent ⟨real-number⟩ with the same sign as vx such that 0 ≤ | r | ≤ ub.

Note: See note after Step 3.2.2 of Section 9.2.4.2.1, Chopped-result.

Case 3.3. (Otherwise).

Step 3.3.1. Perform TEST-MODEL-NUMBER(vx,dt) to obtain tv.

Case 3.3.1.1. tv is ⟨true⟩.

Let lb1 and ub1 both be vx.

Case 3.3.1.2. tv is ⟨false⟩.

Perform LOWER-MODEL-NUMBER(vx,dt) to obtain a ⟨real-number⟩,lb1. Perform HIGHER-MODEL-NUMBER(vx,dt) to obtain a ⟨real-number⟩,ub1.

Step 3.3.2.

Case 3.3.2.1. lb1 = ub1 = 0.

Let r be a ⟨real-number⟩: 0.

Case 3.3.2.2. (Otherwise).

Perform LOWER-MODEL-NUMBER(lb1,dt) to obtain a ⟨real-number⟩,lb2. Perform HIGHER-MODEL-NUMBER(ub1,dt) to obtain a ⟨real-number⟩,ub2. Let r be an implementation-dependent ⟨real-number⟩ such that lb2 ≤ r ≤ ub2.

Step 4. Return a ⟨real-value⟩: r.

9.2.4.2.3 Rounded-result

Operation: ROUNDED-RESULT(x,dt)

Where: x is a ⟨real-value⟩,
dt is a <data-type> that has <float>.

Result: a ⟨real-value⟩.

Step 1. Let vx be the ⟨real-number⟩ in x.

Step 2. Peform HIGHEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⟨real-number⟩,lambda. Perform LOWEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⟨real-number⟩,sigma.

Step 3.

Case 3.1. | vx | > lambda.

Perform either Step 3.1.1 or Step 3.1.2.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3.1.1. Perform RAISE-CONDITION(<overflow-condition>).

Step 3.1.2. Let r be an implementation-dependent ⟨real-number⟩ that approximates vx.

Case 3.2. | vx | < sigma and vx ≠ 0.

Perform either Step 3.2.1 or Step 3.2.2.

Step 3.2.1. Perform RAISE-CONDITION(<underflow-condition>). Let r be a ⟨real-number⟩: 0.

Step 3.2.2. Let r be an implementation-dependent ⟨real-number⟩ with the same sign as vx such that 0 ≤ | r | ≤ sigma.

Note: See note after Step 3.2.2 of Section 9.2.4.2.1, Chopped-result.

Case 3.3. (Otherwise).

Step 3.3.1. Perform TEST-MODEL-NUMBER(vx,dt) to obtain tv.

Case 3.3.1.1. tv is ⟨true⟩.

Let r = vx.

Case 3.3.1.2. tv is ⟨false⟩.

Step 3.3.1.2.1. Perform LOWER-MODEL-NUMBER(vx,dt) to obtain a ⟨real-number⟩,lb.
Perform HIGHER-MODEL-NUMBER(vx,dt) to obtain a ⟨real-number⟩,ub.
Let mp = (lb+ub)/2.

Step 3.3.1.2.2.

Case 3.3.1.2.2.1. vx < mp.

Let ub = mp.

Case 3.3.1.2.2.2. vx > mp.

Let lb = mp.

Case 3.3.1.2.2.3. vx = mp.

No action.

Step 3.3.1.2.3. Let r be an implementation-dependent ⟨real-number⟩ such that lb ≤ r ≤ ub.

Step 4. Return a ⟨real-value⟩: r.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.4.2.4 Unconstrained-result

Operation: UNCONSTRAINED-RESULT(x,dt)

Where: x is a ⟨real-value⟩,
dt is a <data-type> that has <float>.

Result: a ⟨real-value⟩.

Step 1. Let vx be the ⟨real-number⟩ in x.

Step 2. Perform either Step 2.1 or Step 2.2 or Step 2.3.

Step 2.1. Perform RAISE-CONDITION(<overflow-condition>).

Step 2.2. Perform RAISE-CONDITION(<underflow-condition>). Let r be a ⟨real-number⟩: 0.

Step 2.3. Let r be an implementation-dependent ⟨real-number⟩ that approximates vx. (This approximation may be the subject of other standards).

Step 3. Return a ⟨real-value⟩: r.

9.2.4.2.5 Test-model-number

Operation: TEST-MODEL-NUMBER(v,dt)

Where: v is a ⟨real-number⟩,
dt is a <data-type> that has <float>.

Result: ⟨true⟩ or ⟨false⟩.

Case 1. v = 0.

Return ⟨true⟩.

Case 2. (Otherwise).

Step 2.1. Perform MODEL-BASE(dt) to obtain an ⟨integer-value⟩,b. Perform MODEL-PRECISION(dt) to obtain an ⟨integer-value⟩,p. Perform MODEL-MINIMUM-EXPONENT(dt) to obtain an ⟨integer-value⟩,emin. Perform MODEL-MAXIMUM-EXPONENT(dt) to obtain an ⟨integer-value⟩,emax.

Step 2.2. If there exist integers c[1],...,c[p] and e such that $v = f*b^e$, where

1. $f = \pm \sum_{i=1}^{p} c[i] * b^{-i}$,

2. $1 \leq c[1] \leq b-1$,

3. $0 \leq c[i] \leq b-1$ for i=2,...,p, and

4. $emin \leq e \leq emax$,

then return ⟨true⟩; otherwise, return ⟨false⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.2.4.2.6 Highest-positive-model-number

Operation: HIGHEST-POSITIVE-MODEL-NUMBER(dt)

Where: dt is a <data-type> that has <float>.

Result: a ⩽real-number⩾.

Step 1. Perform MODEL-BASE(dt) to obtain an ⩽integer-value⩾,b. Perform MODEL-PRECISION(dt) to obtain an ⩽integer-value⩾,p. Perform MODEL-MAXIMUM-EXPONENT(dt) to obtain an ⩽integer-value⩾,emax.

Step 2. Return a ⩽real-number⩾: $(1-b^{-p})*b^{emax}$.

9.2.4.2.7 Lowest-positive-model-number

Operation: LOWEST-POSITIVE-MODEL-NUMBER(dt)

Where: dt is a <data-type> that has <float>.

Result: a ⩽real-number⩾.

Step 1. Perform MODEL-BASE(dt) to obtain an ⩽integer-value⩾,b. Perform MODEL-MINIMUM-EXPONENT(dt) to obtain an ⩽integer-value⩾,emin.

Step 2. Return a ⩽real-number⩾: $b^{-1}*b^{emin}$.

9.2.4.2.8 Lower-model-number

Operation: LOWER-MODEL-NUMBER(v,dt)

Where: v is a ⩽real-number⩾,
dt is a <data-type> that has <float>.

Result: a ⩽real-number⩾.

Step 1. Let r be the largest ⩽real-number⩾ less than v such that TEST-MODEL-NUMBER(r,dt) returns ⩽true⩾.

Step 2. Return r.

9.2.4.2.9 Higher-model-number

Operation: HIGHER-MODEL-NUMBER(v,dt)

Where: v is a ⩽real-number⩾,
dt is a <data-type> that has <float>.

Result: a ⩽real-number⩾.

Step 1. Let r be the smallest ⩽real-number⩾ greater than v such that TEST-MODEL-NUMBER(r,dt) returns ⩽true⩾.

Step 2. Return r.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.4.2.10 Model-base

Operation: MODEL-BASE(dt)

Where: dt is a <data-type> that has <float>.

Result: an ⩽integer-value⩾.

Step 1. Let v be an IMPLEMENTATION-DEFINED model-base corresponding to dt (see Section 1.6.3.7.2). v MUST NOT depend on the <precision> in dt.

Step 2. Return an ⩽integer-value⩾: v.

9.2.4.2.11 Model-precision

Operation: MODEL-PRECISION(dt)

Where: dt is a <data-type> that has <float>.

Result: an ⩽integer-value⩾.

Step 1. Let v be an IMPLEMENTATION-DEFINED model-precision corresponding to dt (see Section 1.6.3.7.2).

Step 2. Return an ⩽integer-value⩾: v.

9.2.4.2.12 Model-minimum-exponent

Operation: MODEL-MINIMUM-EXPONENT(dt)

Where: dt is a <data-type> that has <float>.

Result: an ⩽integer-value⩾.

Step 1. Let v be an IMPLEMENTATION-DEFINED model-minimum-exponent corresponding to dt (see Section 1.6.3.7.2).

Step 2. Return an ⩽integer-value⩾: v.

9.2.4.2.13 Model-maximum-exponent

Operation: MODEL-MAXIMUM-EXPONENT(dt)

Where: dt is a <data-type> that has <float>.

Result: an ⩽integer-value⩾.

Step 1. Let v be an IMPLEMENTATION-DEFINED model-maximum-exponent corresponding to dt (see Section 1.6.3.7.2).

Step 2. Return an ⩽integer-value⩾: v.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.5 Expressions

Operation: EVALUATE-EXPRESSION(e)

Where: e is an <expression> or <argument>.

Result: an ⟪aggregate-value⟫.

Step 1. If e is an <argument>, let x be the first immediate component of the <expression> immediately contained in e; otherwise, let x be the first immediate component of e. Let f be the type of x (e.g. prefix-expression). Perform EVALUATE-F(x) to obtain v. Return v.

9.2.6 Value References

Operation: EVALUATE-VALUE-REFERENCE(vr)

Where: vr is a <value-reference>.

Result: an ⟪aggregate-value⟫.

Case 1. vr immediately contains a <builtin-function-reference>,x.

Perform EVALUATE-BUILTIN-FUNCTION-REFERENCE(x) to obtain v. Return v.

Case 2. vr immediately contains a <named-constant-reference>,x.

Perform EVALUATE-NAMED-CONSTANT-REFERENCE(x) to obtain v. Return v.

Case 3. vr immediately contains a <variable-reference>,x.

Perform EVALUATE-VARIABLE-REFERENCE(x) to obtain a ⟪generation⟫,g. Perform VALUE-OF-GENERATION(g) to obtain an ⟪aggregate-value⟫,v. v MUST NOT contain ⟪undefined⟫. Return v.

Case 4. vr immediately contains a <procedure-function-reference>,x.

Step 4.1. Perform EVALUATE-ENTRY-REFERENCE(x) to obtain an ⟪evaluated-entry-reference⟫,eer.

Step 4.2. Perform ACTIVATE-PROCEDURE(eer).

Step 4.3. Let v be a copy of the ⟪aggregate-value⟫ in the ⟪returned-value⟫ of the current ⟪block-state⟫. Return v.

Case 5. vr immediately contains an <enquiry-reference>,x.

Perform EVALUATE-ENQUIRY-REFERENCE(x) to obtain an ⟪aggregate-value⟫,v. Return v.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2.7 Constants

Operation: EVALUATE-CONSTANT(c)

Where: c is a <constant>.

Result: an ⟪aggregate-value⟫.

Step 1. Return an ⟪aggregate-value⟫ containing the ⟪basic-value⟫ in c.

9.2.8 Parenthesized Expressions

Operation: EVALUATE-PARENTHESIZED-EXPRESSION(e)

Where: e is a <parenthesized-expression>.

Result: an ⟪aggregate-value⟫.

Step 1. Let x be the <expression> immediately contained in e. Perform EVALUATE-EXPRESSION(x) to obtain v. Return v.

9.2.9 Arguments

Note: There is no operation EVALUATE-ARGUMENT. See EVALUATE-EXPRESSION in Section 9.2.5.

9.2.10 Enquiries

Operation: EVALUATE-ENQUIRY-REFERENCE(r)

Where: r is a <enquiry-reference>.

Result: an ⟪aggregate-value⟫.

Step 1. Let rtn be the first immediate subnode of r.

Step 2.

Case 2.1. rtn is a <declaresize-enquiry>.

Note: The DECLARESIZE enquiry is normally used to obtain the amount of storage, in IMPLEMENTATION-DEFINED units, that a variable would occupy if allocated at the time the function is invoked. The declaration must have the BASED storage class. Use of DECLARESIZE does not require that a generation of storage be associated with the variable (or, more specifically, the declaration) being referenced. The value returned by DECLARESIZE is often used in implementing specialized storage management schemes and is expected to be consistent with the value that would be returned by CURRENTSIZE if allocation actually took place and the function were then immediately invoked. Since the value returned depends on both storage mappings and the units in which storage sizes are expressed, neither of which is specified by this standard, a program using this enquiry may become dependent on a particular implementation.

Step 2.1.1. Let d be the <declaration> designated by the <declaration-designator> immediately contained in r.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.1.2. Let dd be the <data-description> immediately contained in the <variable> of d. Perform EVALUATE-DATA-DESCRIPTION-FOR-ALLOCATION(dd) to obtain an ⩽evaluated-data-description⩾,edd.

Step 2.1.3. Let s be an IMPLEMENTATION-DEFINED ⩽real-value⩾, depending on edd.

Step 2.1.4. Return an

⩽aggregate-value⩾:
⩽aggregate-type⩾:
⩽scalar⩾;
⩽basic-value-list⩾:
⩽basic-value⩾:
s.

Case 2.2. rtn is <epsilon-enquiry>.

Step 2.2.1. Let dt be the <data-type> immediately contained in r. dt MUST contain <float>.

Step 2.2.2. Perform MODEL-BASE(dt) to obtain an ⩽integer-value⩾,b. Perform MODEL-PRECISION(dt) to obtain an ⩽integer-value⩾,p. Let eps be a ⩽real-number⩾: b^{1-p}.

Step 2.2.3. Return an

⩽aggregate-value⩾:
⩽aggregate-type⩾:
⩽scalar⩾;
⩽basic-value-list⩾:
⩽basic-value⩾:
⩽real-value⩾:
eps.

Case 2.3. rtn is <maxval-enquiry>.

Step 2.3.1. Let dt be the <data-type> immediately contained in r. dt MUST contain <float>.

Step 2.3.2. Perform HIGHEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⩽real-value⩾,lambda.

Step 2.3.3. Return an

⩽aggregate-value⩾:
⩽aggregate-type⩾:
⩽scalar⩾;
⩽basic-value-list⩾:
⩽basic-value⩾:
⩽real-value⩾:
lambda.

Case 2.4. rtn is <minval-enquiry>.

Step 2.4.1. Let dt be the <data-type> immediately contained in r. dt MUST contain <float>.

Step 2.4.2. Perform LOWEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⩽real-value⩾,sigma.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.4.3. Return an

⟨aggregate-value⟩:
⟨aggregate-type⟩:
⟨scalar⟩;
⟨basic-value-list⟩:
⟨basic-value⟩:
⟨real-value⟩:
sigma.

Case 2.5. rtn is a <null-enquiry>.

Step 2.5.1. Let dd be the <data-description> in r.

Case 2.5.1.1. dd has <offset>.

Let nr be an ⟨offset-value⟩: ⟨null⟩.

Case 2.5.1.2. dd has <entry>.

Let nr be an ⟨entry-value⟩: ⟨null-entry⟩.

Case 2.5.1.3. dd has <label>.

Let nr be a ⟨label-value⟩: ⟨null-label⟩.

Case 2.5.1.4. dd has <file>.

Let nr be a ⟨file-value⟩: ⟨null-file⟩.

Step 2.5.2. Return an

⟨aggregate-value⟩:
⟨aggregate-type⟩:
⟨scalar⟩;
⟨basic-value-list⟩:
⟨basic-value⟩:
nr.

Case 2.6. rtn is <radix-enquiry>.

Step 2.6.1. Let dt be the <data-type> immediately contained in r. dt MUST contain <float>.

Step 2.6.2. Perform MODEL-BASE(dt) to obtain an ⟨integer-value⟩,b. Let rdx be a ⟨real-number⟩: b.

Step 2.6.3. Return an

⟨aggregate-value⟩:
⟨aggregate-type⟩:
⟨scalar⟩;
⟨basic-value-list⟩:
⟨basic-value⟩:
⟨real-value⟩:
rdx.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3 Prefix Operators

This section describes the prefix operators available in PL/I. Section 9.3.1 gives the general rules for evaluating a <prefix-expression>, and Section 9.3.2 presents the details for each <prefix-operator> in alphabetical order.

9.3.1 Prefix Expressions

Operation: EVALUATE-PREFIX-EXPRESSION(e)

Where: e is a <prefix-expression>.

Result: an ⩽aggregate-value⩾.

Step 1. Let x be the <expression> immediately contained in e. Let rdd be the <data-description> immediately contained in e. Perform PREFIX-F(rdd,x) to obtain v, where f is the name of the <prefix-operator> immediately contained in e. Return v.

9.3.2 Definition of the Prefix Operators

The descriptions of the <prefix-operator>s are given in the following sections in alphabetical order.

In this section, a heading "Attributes" appears in the description of each <prefix-operator>. Construction of the result <data-description> for the <prefix-expression> is described under this heading. The operation CREATE-EXPRESSION in Section 4.6.4 uses the "Attributes" section to construct the result <data-description> for a <prefix-expression>. EVALUATE-PREFIX-EXPRESSION supplies that <data-description>as the first argument (rdd) when invoking the operation which evaluates the <prefix-expression> (See Section 9.3.1).

Under Constraints and Attributes, x denotes the <expression> immediately contained in the <prefix-expression>. Note that most of the operations use the macro operation GENERATE-SCALAR-RESULT.

9.3.2.1 Prefix-minus

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾.

The scalar-result-type has <arithmetic> with the derived <base> and <scale>and the converted <precision> of the <data-type> of x.

Operation: PREFIX-MINUS(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain x'.

Step 2. Perform ARITHMETIC-RESULT(−x',scalar-result-type), to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.2.2 Prefix-not

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <bit>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

The scalar-result-type has <bit>.

Operation: PREFIX-NOT(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to <bit> to obtain y.

Step 2. If y contains the ⟨null-bit-string⟩, then the scalar-result is ⟨bit-string-value⟩: ⟨null-bit-string⟩. Otherwise the scalar-result is a ⟨bit-string-value⟩ whose length is the same as the length of y. The *i*th ⟨bit-value⟩ of the result has ⟨zero-bit⟩ if the *i*th ⟨bit-value⟩ of y has ⟨one-bit⟩; it has ⟨one-bit⟩ if the *i*th ⟨bit-value⟩ of y has ⟨zero-bit⟩.

9.3.2.3 Prefix-plus

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

The scalar-result-type has <arithmetic> with the derived <base> and <scale> and the converted <precision> of the <data-type> of x.

Operation: PREFIX-PLUS(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain the scalar-result.

9.4 Infix Operators

This Section describes the <infix-operator>s available in PL/I. Section 9.4.1 gives the general rules for evaluating an <infix-expression>, and Section 9.4.2 presents the details for each <infix-operator> in alphabetical order.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4.1 Infix Expressions

Operation: EVALUATE-INFIX-EXPRESSION(e)

Where: e is an <infix-expression>.

Result: an ⩽aggregate-value⩾.

Step 1. Let x and y be, in order, the <expression>s immediately contained in e. Let rdd be the <data-description> immediately contained in e. Perform INFIX-F(rdd,x,y) to obtain v, where f is the name of the <infix-operator> immediately contained in e. Return v.

9.4.2 Definition of the Infix Operators

The descriptions of the <infix-operator>s are given in the following sections in alphabetical order.

In this section, a heading "Attributes" appears in the description of each <infix-operator>. Construction of the result <data-description> for the <infix-expression> is described under this heading. The operation CREATE-EXPRESSION in Section 4.6.4 uses the "Attributes" section to construct the result <data-description> for an <infix-expression>. EVALUATE-INFIX-EXPRESSION supplies that <data-description> as the first argument (rdd) when invoking the operation which evaluates the <infix-expression> (See Section 9.4.1).

Under Constraints and Attributes, x and y denote, in order, the <expression>s immediately contained in the <infix-expression>. Further, for the <infix-operator>s <add>, <subtract>, <multiply>, <divide>, and <power>, certain local variables are used with special meanings as follows. The local variables m, p, and r denote the <number-of-digits> components of, respectively, the scalar-result-type, the converted <precision> of the <data-type> of x, and the converted <precision> of the <data-type> of y. The local variables n, q, and s denote the <scale-factor> components of, respectively, the scalar-result-type, the converted <precision> of the <data-type> of x, and the converted <precision> of the <data-type> of y. Unless otherwise stated, the target <base> and <scale> for determining the converted <precision> are the <base> and <scale> of the scalar-result-type. The letter N denotes the maximum <number-of-digits> allowed for data of the <base> and <scale> of the scalar-result-type. The local variable b denotes the numerical-base of the scalar-result-type.

Note that all operations, except INFIX-AND-LOGICAL and INFIX-OR-LOGICAL, use the macro operation GENERATE-SCALAR-RESULT (Section 9.2.1.6).

9.4.2.1 Infix-add

Constraint: The ⩽aggregate-type⩾s of x and y MUST immediately contain ⩽scalar⩾. The <data-type>s of x and y MUST have <arithmetic>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾.

The scalar-result-type has <arithmetic> with the derived common <base> and <scale> of the <data-type>s of x and y. If the result <scale> has <float>, then m = max(p,r). If the result <scale> has <fixed>, then

$$m = \min(N,\max(p-q,r-s)+\max(q,s)+1)$$
$$n = \max(q,s).$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: INFIX-ADD(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-values of x and y to target scalar <data-type>s which have <arithmetic> with the derived common <base> and <scale> of the scalar <data-type>s of x and y. The target <precision> for the conversion of the scalar-value of x (respectively, y) is the converted <precision> of the scalar <data-type> of x (respectively, y). Let x' and y' be the converted values.

Step 2. Perform ARITHMETIC-RESULT(x'+y',scalar-result-type), to obtain the scalar-result.

9.4.2.2 Infix-and

Constraint: The ⟪aggregate-type⟫s of x and y MUST immediately contain ⟪scalar⟫. The <data-type>s of x and y MUST have <bit>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫.

The scalar-result-type has <bit>.

Operation: INFIX-AND(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Let tta be a <data-type> having <bit>, <nonvarying>, and <maximum-length>: <asterisk>. Perform CONVERT(tta,xdt,xval) where xdt is the scalar <data-type> of x and xval is the scalar-value of x to obtain the ⟪bit-value⟫,xbv. Perform CONVERT(tta,ydt,yval) where ydt is the scalar <data-type> of y and yval is the scalar-value of y to obtain the ⟪bit-value⟫,ybv. Let n be the maximum of the lengths of xbv and ybv.

Step 2. Let ttn be a <data-type> having <bit>, <nonvarying>, and <maximum-length>: n. If the length of xbv is less than n then perform CONVERT(ttn,tta,xbv) to obtain x'; otherwise let x' = xbv. If the length of ybv is less than n then perform CONVERT(ttn,tta,ybv) to obtain y'; otherwise let y' = ybv.

Step 3. If n = 0, the scalar-result is a ⟪bit-string-value⟫: ⟪null-bit-string⟫. Otherwise the scalar-result is a ⟪bit-string-value⟫ of length n. The *i*th ⟪bit-value⟫ of the result is ⟪one-bit⟫ if the *i*th ⟪bit-value⟫s of both x' and y' are ⟪one-bit⟫; otherwise the *i*th ⟪bit-value⟫ of the result is ⟪zero-bit⟫.

9.4.2.3 Infix-and-logical

Constraint: The ⟪aggregate-type⟫s of x and y MUST immediately contain ⟪scalar⟫. The <data-type>s of x and y MUST have <bit>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫.

The result <data-type> has <bit>.

Operation: INFIX-AND-LOGICAL(rdd,x,y)

Step 1. Perform ESTABLISH-TRUTH-VALUE(x) to obtain tv.

Step 2. If tv is ⟪true⟫ then perform ESTABLISH-TRUTH-VALUE(y) to obtain tv.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3.

Case 3.1. tv is ⟨true⟩.

Return an ⟨aggregate-value⟩ containing a ⟨bit-string-value⟩: ⟨one-bit⟩.

Case 3.2. tv is ⟨false⟩.

Return an ⟨aggregate-value⟩ containing a ⟨bit-string-value⟩: ⟨zero-bit⟩.

9.4.2.4 Infix-cat

Constraint: The ⟨aggregate-type⟩s of x and y MUST immediately contain ⟨scalar⟩. The <data-type>s of x and y MUST have <computational-type>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

The scalar-result-type has the derived common <string-type> of the <data-type>s of x and y.

Operation: INFIX-CAT(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-values of x and y to the derived common <string-type> of the scalar <data-type>s of x and y. Let the converted values be x' and y'.

Step 2. Perform CONCATENATE(x',y') to obtain the scalar-result.

9.4.2.4.1 Concatenation of String Values

Operation: CONCATENATE(s1,s2)

Where: s1 and s2 have the same <string-type>.

Result: a string of the same <string-type> as s1 and s2.

Case 1. s1 and s2 are both null-strings.

Return a null-string.

Case 2. (Otherwise).

Return a string containing the ⟨character-value⟩s or ⟨bit-value⟩s of s1, if any, in order, followed by the ⟨character-value⟩s or ⟨bit-value⟩s of s2, if any, in order.

9.4.2.5 Infix-divide

Constraint: The ⟨aggregate-type⟩s of x and y MUST immediately contain ⟨scalar⟩. The <data-type>s of x and y MUST have <arithmetic>. The <data-type>s of x and y MUST NOT both have <binary> and <fixed>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

The scalar-result-type has <arithmetic> with the derived common <base> and <scale> of the <data-type>s of x and y. If the result <scale> has <float>, then m = max(p,r). If the result <scale> has <fixed>, then:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

m = N,
n = N−p+q−s.

If the result <base> has <binary> and <float> then n MUST be zero; otherwise, n MUST be greater than or equal to zero.

Operation: INFIX-DIVIDE(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-values of x and y to target scalar <data-type>s which have <arithmetic> with the derived common <base> and <scale> of the scalar <data-type>s of x and y. The target <precision> for the conversion of the scalar-value of x (respectively, y) is the converted <precision> of the scalar <data-type> of x (respectively, y). Let x' and y' be the converted values.

Step 2. If y' = 0, perform RAISE-CONDITION(<zerodivide-condition>); otherwise perform ARITHMETIC-RESULT(x'/y',scalar-result-type), to obtain the scalar-result.

9.4.2.6 Infix-eq

Constraint: The ⟪aggregate-type⟫s of x and y MUST immediately contain ⟪scalar⟫. The <data-type>s of x and y MUST:

1. both have <arithmetic>, or
2. both have <bit>, or
3. both have <character>, or
4. both have <non-computational-type>, with the immediate subnodes of the <non-computational-type>s belonging to the same category other than <area>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫.

The scalar-result-type has <bit>.

Operation: INFIX-EQ(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform COMPARE(x',y',t[1],t[2]) to obtain comp, where x' is the scalar-value of x, y' is the scalar-value of y, t[1] is the scalar <data-type> of x, and t[2] is the scalar <data-type> of y.

Step 2. If comp is ⟪equal⟫, the scalar-result is a ⟪bit-string-value⟫ containing ⟪one-bit⟫. Otherwise the scalar-result is a ⟪bit-string-value⟫ containing ⟪zero-bit⟫.

9.4.2.6.1 Compare

⟪comparison-result⟫ ::= ⟪equal⟫ | ⟪not-equal⟫

⟪not-equal⟫ ::= [⟪less-than⟫ | ⟪greater-than⟫]

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: COMPARE(v[1],v[2],t[1],t[2])

Where: v[1] is a ⟨basic-value⟩,
v[2] is a ⟨basic-value⟩,
t[1] is a scalar <data-type>,
t[2] is a scalar <data-type>.

Result: a subtree of ⟨comparison-result⟩.

Case 1. At least one of t[1] and t[2] has <arithmetic> or <pictured-numeric>.

Convert v[1] and v[2] to target scalar <data-type>s which have <arithmetic> with the derived common <base> and <scale> of t[1] and t[2]. The <precision> of the target type for v[i] is the converted <precision> of t[i]. Let v[1]' and v[2]' be the converted values.

Case 1.1. v[1]' = v[2]'.

Return ⟨equal⟩.

Case 1.2. v[1]' < v[2]'.

Return ⟨not-equal⟩: ⟨less-than⟩.

Case 1.3. v[1]' > v[2]'.

Return ⟨not-equal⟩: ⟨greater-than⟩.

Case 2. Each of t[1] and t[2] has <string>.

Step 2.1. Let st be the derived common <string-type> of t[1] and t[2].

Step 2.2. Let tta be a <data-type> having <string-type>: st, <nonvarying>, and <maximum-length>: <asterisk>. Perform CONVERT(tta,t[1],v[1]) to obtain vs[1]. Perform CONVERT(tta,t[2],v[2]) to obtain vs[2]. Let n be the maximum of the lengths of vs[1] and vs[2].

Step 2.3. Let ttn be a <data-type> having <string-type>: st, <nonvarying>, and <maximum-length>: n. If the length of vs[1] is less than n then perform CONVERT(ttn,tta,vs[1]) to obtain v[1]'; otherwise let v[1]' = vs[1]. If the length of vs[2] is less than n then perform CONVERT(ttn,tta,vs[2]) to obtain v[2]'; otherwise let v[2]' = vs[2].

Step 2.4.

Case 2.4.1. v[1]' = v[2]'.

Return ⟨equal⟩.

Case 2.4.2. v[1]' and v[2]' are ⟨bit-string-value⟩s, and i is the smallest integer such that the *i*th ⟨bit-value⟩s of v[1]' and v[2]' differ.

If the *i*th ⟨bit-value⟩ of v[1]' is ⟨zero-bit⟩ (while the *i*th ⟨bit-value⟩ of v[2]' is ⟨one-bit⟩), return ⟨not-equal⟩: ⟨less-than⟩. Otherwise return ⟨not-equal⟩: ⟨greater-than⟩.

Case 2.4.3. v[1]' and v[2]' are ⟨character-string-value⟩s, and i is the smallest integer such that the *i*th ⟨character-value⟩ of v[1]' and v[2]' differ.

If the ⟨symbol⟩ in the *i*th ⟨character-value⟩ of v[1]' precedes the ⟨symbol⟩ in the *i*th ⟨character-value⟩ of v[2]' in the result of performing COLLATE-BIF, return ⟨not-equal⟩: ⟨less-than⟩. Otherwise return ⟨not-equal⟩: ⟨greater-than⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3. t[1] and t[2] both have <non-computational-type>.

Let v[1]' = v[1], and v[2]' = v[2].

Case 3.1. v[1]' and v[2]' do not contain ⋖pointer-value⋗s or ⋖offset-value⋗s.

If v[1]' or v[2]' contains a ⋖block-state-designator⋗,bsd, then bsd MUST designate a ⋖block-state⋗ in the ⋖block-state-list⋗. If v[1]' and v[2]' are identical, return ⋖equal⋗; otherwise return ⋖not-equal⋗ with no subnode.

Case 3.2. v[1]' and v[2]' contain ⋖pointer-value⋗s.

Step 3.2.1. Any ⋖allocation-unit-designator⋗ contained in v[1]' or v[2]' MUST designate an ⋖allocation-unit⋗ that exists.

Step 3.2.2.

Case 3.2.2.1. v[1]' and v[2]' are equal.

Return ⋖equal⋗.

Case 3.2.2.2. Either v[1]' or v[2]', but not both, has ⋖null⋗.

Return ⋖not-equal⋗.

Case 3.2.2.3. (Otherwise).

Let edd1 and edd2 be the ⋖evaluated-data-description⋗ of v[1]' and v[2]' respectively. Let aud1 and aud2 be the ⋖allocation-unit-designator⋗ of v[1]' and v[2]' respectively. Let bvl1 and bvl2 be the ⋖basic-value-list⋗ of the ⋖allocation-unit⋗ designated by aud1 and aud2 respectively. Let si1 and si2 be the first component of the ⋖storage-index-list⋗ of v[1]' and v[2]' respectively.

Case 3.2.2.3.1.

1. aud1 is not equal to aud2, and
2. edd1 and edd2 both contain a <data-type> that has neither a <maximum-length> with 0 nor an <area-size> with 0.

Return ⋖not-equal⋗.

Case 3.2.2.3.2.

1. aud1 is equal to aud2, and
2. the results of performing NON-TRIVIAL-BASIC-VALUE-INDEX(si1,bvl1) and NON-TRIVIAL-BASIC-VALUE-INDEX(si2,bvl2) are not equal.

Return ⋖not-equal⋗.

Case 3.2.2.3.3. (Otherwise).

In an implementation-dependent manner, return either ⋖equal⋗ or ⋖not-equal⋗.

Case 3.3. v[1]' and v[2]' contain ⋖offset-value⋗s.

Case 3.3.1. v[1]' and v[2]' are equal.

Return ⋖equal⋗.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3.3.2. v[1]' or v[2]', but not both, has ⟪null⟫.

Return ⟪not-equal⟫.

Case 3.3.3. (Otherwise).

Step 3.3.3.1.

Case 3.3.3.1.1. The ⟪area-action-list⟫ in v[1]' contains fewer immediate components than the ⟪area-action-list⟫ in v[2]'.

Let aal1 be the ⟪area-action-list⟫ in v[1]'. Let aal2 be the ⟪area-action-list⟫ in v[2]'.

Case 3.3.3.1.2. (Otherwise).

Let aal1 be the ⟪area-action-list⟫ in v[2]'. Let aal2 be the ⟪area-action-list⟫ in v[1]'.

Note: If aal1 and aal2 have different numbers of immediate components, aal1 will have fewer.

Step 3.3.3.2. Let n1 be the number of immediate components of aal1. Let n2 be the number of immediate components of aal2. Let aa1[i], i=1,...,n1, be the immediate components of aal1, in order. Let aa2[i], i=1,...,n2, be the immediate components of aal2, in order.

Step 3.3.3.3. Let laa be the last ⟪area-allocation⟫ in aal1. Let ledd be the ⟪evaluated-data-description⟫ in laa.

Step 3.3.3.4.

Case 3.3.3.4.1.

1. n1 < n2, and
2. aa1[i] is equal to aa2[i], i=1,...,n1, and
3. aal2 does not contain an

```
⟪area-freeing⟫:
   ⟪integer-value⟫:
      n1;;,
```

and

4. ledd has a <data-type> which contains neither a <maximum-length> with 0 nor an <area-size> with 0.

Return ⟪not-equal⟫.

Case 3.3.3.4.2. (Otherwise).

In an implementation-dependent manner, return either ⟪equal⟫ or ⟪not-equal⟫.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4.2.6.2 Non-trivial-basic-value-index

Operation: NON-TRIVIAL-BASIC-VALUE-INDEX(si,bvl)

Where: si is a ⟨storage-index⟩,
bvl is a ⟨basic-value-list⟩.

Result: an ⟨integer-value⟩.

Step 1. Let bvi be the ⟨basic-value-index⟩ in si. Let p be 0.

Step 2. For i=1,...,bvi−1, in any order, perform Steps 2.1 through 2.2.

Step 2.1. Let v be the immediate component of the *i*th ⟨basic-value⟩ of bvl.

Case 2.1.1. v is a ⟨character-string-value⟩.

Let n be the number of ⟨character-value⟩s in v, possibly 0.

Case 2.1.2. v is a ⟨bit-string-value⟩.

Let n be the number of ⟨bit-value⟩s in v, possibly 0.

Case 2.1.3. v is an ⟨area-value⟩.

The first ⟨area-action⟩ in the ⟨area-action-list⟩ of v contains an ⟨area-size⟩,as. If as has 0 then let n be 0; otherwise, let n be 1.

Case 2.1.4. (Otherwise).

Let n be 1.

Step 2.2. Let p be p+n.

Step 3. If si contains a ⟨position-index⟩,pi, then let p be p+pi.

Step 4. Return an ⟨integer-value⟩: p.

9.4.2.7 Infix-ge

Constraint: The ⟨aggregate-type⟩s of x and y MUST immediately contain ⟨scalar⟩. The <data-type>s of x and y MUST:

1. both have <arithmetic>, or
2. both have <bit>, or
3. both have <character>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

The scalar-result-type has <bit>.

Operation: INFIX-GE(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform COMPARE(x',y',t[1],t[2]) to obtain comp, where x' is the scalar-value of x, y' is the scalar-value of y, t[1] is the scalar <data-type> of x, and t[2] is the scalar <data-type> of y.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 2. If comp is ≤not-equal≥: ≤greater-than≥; or ≤equal≥, the scalar-result is a ≤bit-string-value≥ containing ≤one-bit≥; otherwise the scalar-result is a ≤bit-string-value≥ containing ≤zero-bit≥.

9.4.2.8 Infix-gt

Constraint: The ≤aggregate-type≥s of x and y MUST immediately contain ≤scalar≥. The <data-type>s of x and y MUST:

1. both have <arithmetic>, or
2. both have <bit>, or
3. both have <character>.

Attributes: The result ≤aggregate-type≥ immediately contains ≤scalar≥.

The scalar-result-type has <bit>.

Operation: INFIX-GT(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform COMPARE(x',y',t[1],t[2]) to obtain comp, where x' is the scalar-value of x, y' is the scalar-value of y, t[1] is the scalar <data-type> of x, and t[2] is the scalar <data-type> of y.

Step 2. If comp is ≤not-equal≥: ≤greater-than≥; then the scalar-result is a ≤bit-string-value≥ containing ≤one-bit≥; otherwise the scalar-result is a ≤bit-string-value≥ containing ≤zero-bit≥.

9.4.2.9 Infix-le

Constraint: The ≤aggregate-type≥s of x and y MUST immediately contain ≤scalar≥. The <data-type>s of x and y MUST:

1. both have <arithmetic>, or
2. both have <bit>, or
3. both have <character>.

Attributes: The result ≤aggregate-type≥ immediately contains ≤scalar≥.

The scalar-result-type has <bit>.

Operation: INFIX-LE(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform COMPARE(x',y',t[1],t[2]) to obtain comp, where x' is the scalar-value of x, y' is the scalar-value of y, t[1] is the scalar <data-type> of x, and t[2] is the scalar <data-type> of y.

Step 2. If comp is ≤not-equal≥: ≤less-than≥; or ≤equal≥ then the scalar-result is a ≤bit-string-value≥ containing ≤one-bit≥; otherwise the scalar-result is a ≤bit-string-value≥ containing ≤zero-bit≥.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4.2.10 Infix-lt

Constraint: The ⟨aggregate-type⟩s of x and y MUST immediately contain ⟨scalar⟩. The <data-type>s of x and y MUST:

1. both have <arithmetic>, or
2. both have <bit>, or
3. both have <character>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

The scalar-result-type has <bit>.

Operation: INFIX-LT(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform COMPARE(x',y',t[1],t[2]) to obtain comp, where x' is the scalar-value of x, y' is the scalar-value of y, t[1] is the scalar <data-type> of x, and t[2] is the scalar <data-type> of y.

Step 2. If comp is ⟨not-equal⟩: ⟨less-than⟩; then the scalar-result is a ⟨bit-string-value⟩ containing ⟨one-bit⟩; otherwise the scalar-result is a ⟨bit-string-value⟩ containing ⟨zero-bit⟩.

9.4.2.11 Infix-multiply

Constraint: The ⟨aggregate-type⟩s of x and y MUST immediately contain ⟨scalar⟩. The <data-type>s of x and y MUST have <arithmetic>.

Attributes: The scalar-result-type has <arithmetic> with the derived common <base> and <scale> of the <data-type>s of x and y. If the result <scale> has <float>, then m = max(p,r). If the result <scale> has <fixed>, then

$$m = \min(N, p+r+1)$$
$$n = q+s.$$

The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

Operation: INFIX-MULTIPLY(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-values of x and y to target scalar <data-type>s which have <arithmetic> with the derived common <base> and <scale> of the scalar <data-type>s of x and y. The target <precision> for the conversion of the scalar-value of x (respectively, y) is the converted <precision> of the scalar <data-type> of x (respectively, y). Let x' and y' be the converted values.

Step 2. Perform ARITHMETIC-RESULT(x'*y',scalar-result-type), to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4.2.12 Infix-ne

Constraint: The ⟨aggregate-type⟩s of x and y MUST immediately contain ⟨scalar⟩. The <data-type>s of x and y MUST:

1. both have <arithmetic>, or
2. both have <bit>, or
3. both have <character>, or
4. both have <non-computational-type>, with the immediate subnodes of the <non-computational-type>s belonging to the same category other than <area>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

The scalar-result-type has <bit>.

Operation: INFIX-NE(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform COMPARE(x',y',t[1],t[2]) to obtain comp, where x' is the scalar-value of x, y' is the scalar-value of y, t[1] is the scalar <data-type> of x, and t[2] is the scalar <data-type> of y.

Step 2. If comp is ⟨not-equal⟩, the scalar-result is a ⟨bit-string-value⟩ containing ⟨one-bit⟩; otherwise the scalar-result is a ⟨bit-string-value⟩ containing ⟨zero-bit⟩ .

9.4.2.13 Infix-or

Constraint: The ⟨aggregate-type⟩s of x and y MUST immediately contain ⟨scalar⟩. The <data-type>s of x and y MUST have <bit>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

The scalar-result-type has <bit>.

Operation: INFIX-OR(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Let tta be a <data-type> having <bit>, <nonvarying>, and <maximum-length>: <asterisk>. Perform CONVERT(tta,xdt,xval) where xdt is the scalar <data-type> of x and xval is the scalar-value of x to obtain the ⟨bit-value⟩,xbv. Perform CONVERT(tta,ydt,yval) where ydt is the scalar <data-type> of y and yval is the scalar-value of y to obtain the ⟨bit-value⟩,ybv. Let n be the maximum of the lengths of xbv and ybv.

Step 2. Let ttn be a <data-type> having <bit>, <nonvarying>, and <maximum-length>: n. If the length of xbv is less than n then perform CONVERT(ttn,tta,xbv) to obtain x'; otherwise let x' = xbv. If the length of ybv is less than n then perform CONVERT(ttn,tta,ybv) to obtain y'; otherwise let y' = ybv.

Step 3. If n = 0, the scalar-result is a ⟨bit-string-value⟩: ⟨null-bit-string⟩. Otherwise the scalar-result is a ⟨bit-string-value⟩ of length n. The *i*th ⟨bit-value⟩ of the result is ⟨zero-bit⟩ if the *i*th ⟨bit-value⟩s of both x' and y' are ⟨zero-bit⟩; otherwise the *i*th ⟨bit-value⟩ of the result is ⟨one-bit⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4.2.14 Infix-or-logical

Constraint: The ⟪aggregate-type⟫s of x and y MUST immediately contain ⟪scalar⟫. The <data-type>s of x and y MUST have <bit>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫.

The result <data-type> has <bit>.

Operation: INFIX-OR-LOGICAL(rdd,x,y)

Step 1. Perform ESTABLISH-TRUTH-VALUE(x) to obtain tv.

Step 2. If tv is ⟪false⟫ then perform ESTABLISH-TRUTH-VALUE(y) to obtain tv.

Step 3.

Case 3.1. tv is ⟪true⟫.

Return an ⟪aggregate-value⟫ containing a ⟪bit-string-value⟫: ⟪one-bit⟫.

Case 3.2. tv is ⟪false⟫.

Return an ⟪aggregate-value⟫ containing a ⟪bit-string-value⟫: ⟪zero-bit⟫.

9.4.2.15 Infix-power

Constraint: The ⟪aggregate-type⟫s of x and y MUST immediately contain ⟪scalar⟫. The <data-type>s of x and y MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫.

The scalar-result-type has <arithmetic>. The result <base>, <scale>, and <precision> are determined from the <data-type>s of x and y as follows.

Case 1. The derived <scale> of x has <fixed>; y is a <constant> with <scale-factor>: 0; and whose value, y', is positive; $(p+1)*y'-1$ does not exceed the maximum <number-of-digits> for the derived <base> and <scale> of x.

The result <scale> has <fixed>, the result <base> has the derived <base> of x; and

$$m = (p+1)*y'-1,$$
$$n = q*y'.$$

Case 2. The derived <scale> of y has <fixed>, s=0, but Case 1 does not hold.

The result <scale> has <float>, the result <base> has the derived <base> of x, and m=p.

Case 3. (Otherwise).

The result <scale> has <float>; the result <base> has the derived common <base> of x and y, and m = max(p,r).

Operation: INFIX-POWER(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1. Determine values u and v as follows:

Case 1.1. Conditions the same as for Attributes, Case 1.

Convert the scalar-value of x to a target scalar <data-type> having <arithmetic> with the derived <base> and <scale>of the scalar <data-type> of x. The target <precision> for the conversion is the converted <precision> of the scalar <data-type> of x. Let u be the converted scalar-value of x, and let v be the scalar-value of y.

Case 1.2. Conditions the same as for Attributes, Case 2.

Convert the scalar-value of x to the scalar-result-type, and let u be the converted value. Let v be the scalar-value of y.

Case 1.3. Conditions the same as for Attributes, Case 3.

Convert the scalar-values of x and y to target scalar <data-type>s which have <arithmetic> with <scale>: <float>;, which have the derived common <base> of the scalar <data-type>s of x and y, and whose <precision>s are the converted <precision>s of the scalar <data-type>s of x and y, respectively. Let u and v be the converted scalar-values of x and y, respectively.

Step 2. Determine a value z as follows:

Case 2.1. $u < 0$.

If the conditions of Attributes, Case 1 or Case 2, hold, then $z=u^v$; otherwise, perform RAISE-CONDITION(<error-condition>).

Case 2.2. $u = 0$.

If $v \leq 0$ then perform RAISE-CONDITION(<error-condition>); otherwise let $z = 0$.

Case 2.3. $u > 0$.

Then:

$z = 1/(u^{-v})$, if $v < 0$,
$z = 1$, if $v = 0$,
$z = u^v$, if $v > 0$.

Step 3. Perform ARITHMETIC-RESULT(z,scalar-result-type) to obtain the scalar-result.

9.4.2.16 Infix-subtract

Constraint: The ⋖aggregate-type⋗s of x and y MUST immediately contain ⋖scalar⋗. The <data-type>s of x and y MUST have <arithmetic>.

Attributes: The result ⋖aggregate-type⋗ immediately contains ⋖scalar⋗.

The scalar-result-type has <arithmetic> with the derived common <base> and <scale> of the <data-type>s of x and y. If the result <scale> has <float>, then m = max(p,r). If the result <scale> has <fixed>, then

$m = \min(N,\max(p-q,r-s)+\max(q,s)+1)$
$n = \max(q,s)$.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: INFIX-SUBTRACT(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-values of x and y to target scalar <data-type>s which have <arithmetic> with the derived common <base>, and <scale> of the scalar <data-type>s of x and y. The target <precision> for the conversion of the scalar-value of x (respectively, y) is the converted <precision> of the scalar <data-type> of x (respectively, y). Let x' and y' be the converted values.

Step 2. Perform ARITHMETIC-RESULT(x'-y',scalar-result-type), to obtain the scalar-result.

9.4.2.17 Infix-xor

Constraint: The ⟨aggregate-type⟩s of x and y MUST immediately contain ⟨scalar⟩. The <data-type>s of x and y MUST have <bit>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩.

The scalar-result-type has <bit>.

Operation: INFIX-XOR(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Let tta be a <data-type> having <bit>, <nonvarying>, and <maximum-length>: <asterisk>. Perform CONVERT(tta,xdt,xval) where xdt is the scalar <data-type> of x and xval is the scalar-value of x to obtain the ⟨bit-value⟩,xbv. Perform CONVERT(tta,ydt,yval) where ydt is the scalar <data-type> of y and yval is the scalar-value of y to obtain the ⟨bit-value⟩,ybv. Let n be the maximum of the lengths of xbv and ybv.

Step 2. Let ttn be a <data-type> having <bit>, <nonvarying>, and <maximum-length>: n. If the length of xbv is less than n then perform CONVERT(ttn,tta,xbv) to obtain x'; otherwise let x' = xbv. If the length of ybv is less than n then perform CONVERT(ttn,tta,ybv) to obtain y'; otherwise let y' = ybv.

Step 3. If n = 0, the scalar-result is a ⟨bit-string-value⟩: ⟨null-bit-string⟩. Otherwise the scalar-result is a ⟨bit-string-value⟩ of length n. The *i*th ⟨bit-value⟩ of the result is ⟨zero-bit⟩ if the *i*th ⟨bit-value⟩s of x' and y' are the same; otherwise the *i*th ⟨bit-value⟩ of the result is ⟨one-bit⟩.

9.5 Built-in Functions

This section describes the <builtin-function>s available in PL/I. Section 9.5.1 gives the general rules for evaluating a <builtin-function-reference>, and Section 9.5.4 presents the details for each <builtin-function> in alphabetical order.

In this section, a heading "Attributes" appears in the description of each <builtin-function>. Construction of the result <data-description> for the <builtin-function> is described under this heading. The operation CREATE-BUILTIN-FUNCTION-REFERENCE in Section 4.6.5.7 uses the "Attributes" section to construct the result <data-description> for a <builtin-function-reference>. EVALUATE-BUILTIN-FUNCTION-REFERENCE supplies that <data-description> as the first argument (rdd) when invoking the operation which evaluates the <builtin-function> (See Section 9.5.1).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In this section an additional heading "Arguments" appears in the description of each <builtin-function>. The number of <argument>s required by the <builtin-function> and the <argument> names used in the description are given under this heading. The <argument> names are separated by commas and the names of optional <argument>s are enclosed in brackets.

Arguments: x,y[,p[,q]]

The letters x, y, p, and q will be used to refer to the <argument>s in the description of the <builtin-function>. The <argument>s p and q are optional, i.e. neither need be specified; but if q is specified, then p MUST also be specified.

Figure 15. An Example of Optional Arguments.

9.5.1 Built-in Function Reference

Operation: EVALUATE-BUILTIN-FUNCTION-REFERENCE(bfr)

Where: bfr is a <builtin-function-reference>.

Result: an ⩽aggregate-value⩾.

Step 1. Let bif be the <builtin-function> immediately contained in bfr. Let x[1], x[2],...,x[n] be the <argument>s, if any, simply contained in bfr.

Step 2.

Case 2.1. bif is <collate-bif>, <date-bif>, <datetime-bif>, <empty-bif>, <null-bif>, <oncode-bif>, <onfile-bif>, <onkey-bif>, <onsource-bif>, or <time-bif>.

Perform f, where f is the operation whose name is the same as that of bif, to obtain an ⩽aggregate-value⩾,v. Return v.

Case 2.2. (Otherwise).

Let rdd be the <data-description> immediately contained in bfr. Perform f(rdd,x[1],x[2],...,x[n]), where f is the operation whose name is the same as that of bif, to obtain an ⩽aggregate-value⩾,v. Return v.

9.5.2 Special Terms Defined for Built-in Functions

9.5.2.1 Definition of N

Under the heading "Attributes", N is used in describing the <precision> of the result <data-type>. It denotes the maximum <number-of-digits> allowed by the implementation for the result <base> and <scale>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.2.2 The Arguments p and q

9.5.2.2.1 Constraints on p and q

When p and q are used under the heading "Arguments", certain special rules apply. Both p and q MUST be of the form <argument>: <expression>: <constant>: <data-type>,dt;;;, where dt has <fixed> and has <scale-factor> equal to zero. (Note that the <constant> may have been produced by the evaluation of an argument <expression> as a restricted expression in CREATE-BUILTIN-FUNCTION-REFERENCE, Section 4.6.5.7). The value of p MUST be greater than zero and less than or equal to N, i.e. the maximum <number-of-digits> for the result <base> and <scale>. q MUST NOT occur unless the scalar-result-type has <fixed>. If the scalar-result-type has <binary>, q MUST be zero otherwise, q MUST be greater than or equal to zero.

9.5.2.2.2 Attributes of Result Determined by p and q

Case 1. p and q are both specified.

The <number-of-digits> of the result-type is the value of p. The <scale-factor> of the result-type is the value of q.

Case 2. p is specified, q is not specified.

The <number-of-digits> of the result-type is the value of p. If the <scale> of the result-type has <fixed>, then the <scale-factor> of the result-type has 0.

Case 3. Neither p nor q is specified.

The <number-of-digits> and <scale-factor> of the result-type are the converted <number-of-digits> and <scale-factor> of the first argument (x) of the built-in function.

9.5.3 Operations Used in Built-in Function Definitions

9.5.3.1 Get-established-onvalue

Operation: GET-ESTABLISHED-ONVALUE(tp)

Where: tp is one of ⩽oncode-value⩾, ⩽onfile-value⩾, ⩽onkey-value⩾, or ⩽onsource-value⩾.

Result: a ⩽character-string-value⩾, an ⩽integer-value⩾, or ⩽fail⩾.

Step 1. Let bs be the current ⩽block-state⩾.

Step 2.

Case 2.1. bs contains a ⩽condition-bif-value⩾,cbv, with an immediate component of the type indicated by tp.

Let r be the ⩽integer-value⩾ or ⩽character-string-value⩾ component of cbv, if that component exists; otherwise return ⩽fail⩾. Return r.

Case 2.2. (Otherwise).

Case 2.2.1. There is a ⩽block-state⩾ immediately preceding bs in the ⩽block-state-list⩾.

Let bs be this ⩽block-state⩾. Go to Step 2.

Case 2.2.2. (Otherwise).

Return ⩽fail⩾.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4 Definition of the Built-in Functions

The descriptions of the <builtin-function>s are given in the following sections in alphabetical order.

9.5.4.1 Abs-bif

Arguments: x

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The scalar-result-type has <arithmetic>. The <base> of the scalar-result-type is the derived <base> of the <data-type> of x. The <scale> and <precision> of the scalar-result-type are the derived <scale> and converted <precision> of the <data-type> of x.

Operation: ABS-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type. Let y= | z | where z is the converted value.

Step 2. Perform ARITHMETIC-RESULT(y,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.2 Acos-bif

Arguments: x

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: ACOS-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain y.

Step 2. The value of y MUST be between -1 and 1, inclusive. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let w be the arc cosine of y, in radians, such that $0 \leq w \leq \pi$.

Step 3. Perform ARITHMETIC-RESULT(w,rt), where rt is the scalar-result-type, to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.3 Add-bif

Arguments: x,y,p[,q]

Constraint: The ⟪aggregate-type⟫s of x and y MUST immediately contain ⟪scalar⟫. The <data-type>s of x and y MUST have <arithmetic>. Constraints on p and q are described in Section 9.5.2.2.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <base> and <scale> of the scalar-result-type are the derived common <base> and <scale> of the <data-type>s of x and y. The <precision> of the scalar-result-type is determined as defined in Section 9.5.2.2.

Operation: ADD-BIF(rdd,x,y,p[,q])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 and 1.2 in either order.

Step 1.1. Convert the scalar-value of x to the derived common <base> and <scale> of the scalar <data-type>s of x and y, and the converted <precision> of the <data-type> of x.

Step 1.2. Convert the scalar-value of y to the derived common <base> and <scale> of the scalar <data-type>s of x and y, and the converted <precision> of the <data-type> of y.

Step 2. Let z be the sum of the converted scalar-value of x and the converted scalar-value of y.

Step 3. Perform ARITHMETIC-RESULT(z,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.4 Addr-bif

Arguments: x

Constraint: x MUST be of the form <argument>: <expression>: <value-reference>: <variable-reference>,y.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The result <data-type> has <pointer>.

Operation: ADDR-BIF(rdd,x)

Step 1. Let dp be the <declaration-designator> immediately contained in y. Perform EVALUATE-VARIABLE-REFERENCE(y) to obtain a ⟪generation⟫,g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⟪true⟫. Return an ⟪aggregate-value⟫ containing ⟪pointer-value⟫: g.

9.5.4.5 Asin-bif

Arguments: x

Constraint: The ⟪aggregate-type⟫ of x MUST immediately contain ⟪scalar⟫. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: ASIN-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain y.

Step 2. The value of y MUST be between -1 and 1, inclusive. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let w be the arc sine of y, in radians, such that $-\pi/2 \leq w \leq \pi/2$.

Step 3. Perform ARITHMETIC-RESULT(w,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.6 Atan-bif

Arguments: y[,x]

Constraint: The ⟪aggregate-type⟫s of y and x MUST immediately contain ⟪scalar⟫. The <data-type>s of y and x MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The scalar-result-type has <scale>: <float>. The <base> of the scalar-result-type is the derived common <base> of the <data-type>s in y and x. The <precision> of the scalar-result-type is the greater of the converted <precision>s of the <data-type>s of y and x.

Operation: ATAN-BIF(rdd,y[,x])

Perform GENERATE-SCALAR-RESULT.

Step 1.

Case 1.1. x is not ⟪absent⟫.

In either order, convert the scalar-value of y to the scalar-result-type to obtain s and convert the scalar-value of x to the scalar-result-type to obtain r. The values of r and s MUST NOT both be 0.

Case 1.2. x is ⟪absent⟫.

Convert the scalar-value of y to the scalar-result-type to obtain s.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type.

Case 2.1. x is not ⟪absent⟫.

Let v be the value, in radians, of arctangent(s/r) such that v satisfies all of the following constraints:

if $s > 0$ then $0 < v < \pi$,
if $s = 0$ then $v = 0$ or $v = \pi$,
if $s < 0$ then $-\pi < v < 0$,
if $r > 0$ then $|v| < \pi/2$,
if $r = 0$ then $|v| = \pi/2$,
if $r < 0$ then $|v| > \pi/2$.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2.2. x is ⟨absent⟩.

Let v be the arc tangent of s, such that

$$-\pi/2 < v < \pi/2.$$

Step 3. Perform ARITHMETIC-RESULT(v,rt) to obtain the scalar-result.

9.5.4.7 Atand-bif

Arguments: y[,x]

Constraint: The ⟨aggregate-type⟩s of y and x MUST immediately contain ⟨scalar⟩. The <data-type>s of y and x MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The <scale> of the scalar-result-type has <float>. The <base> of the scalar-result-type is the derived common <base> of the <data-type>s of y and x. The <precision> of the scalar-result-type is the greater of the converted <precision>s of <data-type>s of y and x.

Operation: ATAND-BIF(rdd,y[,x])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform ATAN-BIF(y,x) to obtain a value, v.

Step 2. Let w be the value of v multiplied by $180/\pi$.

Step 3. Perform ARITHMETIC-RESULT(w,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.8 Atanh-bif

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: ATANH-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain y. y MUST be less than 1 in absolute value.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let w be the arc-hyperbolic tangent of y.

Step 3. Perform ARITHMETIC-RESULT(w,rt), where rt is the scalar-result-type, to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.9 Binary-bif

Arguments: x[,p[,q]]

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <computational-type>. Constraints on p and q are described in Section 9.5.2.2. If the <data-type> of x has <fixed>, <decimal> and a <scale-factor> which does not contain 0, then p MUST be specified.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The <base> of the scalar-result-type has <binary>. The <scale> of the scalar-result-type is the derived <scale> of the <data-type> of x. The <precision> of the scalar-result-type is determined as defined in Section 9.5.2.2.

Operation: BINARY-BIF(rdd,x[,p[,q]])

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain the scalar-result.

9.5.4.10 Bit-bif

Arguments: x[,le]

Constraint: The ⩽aggregate-type⩾s of x and le MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <computational-type>. The <data-type> of le MUST have <fixed> (but not <pictured-numeric>) with <scale-factor>: 0.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The scalar-result-type has <bit>.

Operation: BIT-BIF(rdd,x[,le])

Perform GENERATE-SCALAR-RESULT.

Step 1.

Case 1.1. le is ⩽absent⩾.

Convert the scalar-value of x to <bit> to obtain the scalar-result.

Case 1.2. le is not ⩽absent⩾.

Convert le to integer-type. The result, n, MUST NOT be negative. Convert the scalar-value of x to <bit> of length n to obtain the scalar-result.

9.5.4.11 Bool-bif

Arguments: x,y,ca

Constraint: The ⩽aggregate-type⩾s of x, y and ca MUST immediately contain ⩽scalar⩾. The <data-type>s of x, y and ca MUST have <bit>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The scalar-result-type has <bit>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: BOOL-BIF(rdd,x,y,ca)

Perform GENERATE-SCALAR-RESULT.

Step 1. In either order, convert the scalar-value of x to <bit> to obtain ra and convert the scalar-value of y to <bit> to obtain rb. Let le be the greater of the lengths of ra and rb. In either order, convert ra to <bit> of length le to obtain sa, and convert rb to <bit> of length le to obtain sb.

Step 2. Convert ca to <bit> of length 4 to obtain d. Let the ⩽bit-value⩾s within d be named d[1], d[2], d[3], d[4], from left-to-right, respectively.

Step 3.

Case 3.1. le = 0.

The scalar-result is a null-string.

Case 3.2. le > 0.

The *i*th ⩽bit-value⩾ of the scalar-result is set to one of the values d[1], d[2], d[3], d[4] depending on the *i*th ⩽bit-value⩾s of sa and sb as shown in Figure 16.

sa	sb	scalar-result
0	0	d[1]
0	1	d[2]
1	0	d[3]
1	1	d[4]

Figure 16. Table of Scalar-results as a Function of ⩽bit-values⩾ for BOOL-BIF.

In this example BOOL is used to perform an exclusive-or operation on the first two arguments. The value '1001'B is the result of the evaluation of:

```
BOOL('1100'B,'0101'B,'0110'B)
```

Figure 17. An Example of the BOOL Built-in Function.

9.5.4.12 Ceil-bif

Arguments: x

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The scalar-result-type has <arithmetic> with the derived <base> and <scale> of the <data-type> of x.

Case 1. The scalar-result-type has <scale>: <float>.

The <precision> of the scalar-result-type is the converted <precision> of the <data-type> of x.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2. The scalar-result-type has <scale>: <fixed>.

Let r denote the <number-of-digits> and s denote the <scale-factor> of the converted value of x. The <precision> of the scalar-result-type has:

<number-of-digits>: min(N,max(r−s+1,1));
<scale-factor>: 0.

Operation: CEIL-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to its derived <base>, <scale>, and converted <precision>.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the smallest integer that is greater than or equal to the converted scalar-value of x.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.13 Character-bif

Arguments: sa[,le]

Constraint: The ⟨aggregate-type⟩s of sa and le MUST immediately contain ⟨scalar⟩. The <data-type> of le MUST have <fixed> (but not <pictured-numeric>) with <scale-factor>: 0. The <data-type> of sa MUST have <computational-type>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The scalar-result-type has <character>.

Operation: CHARACTER-BIF(rdd,sa[,le])

Perform GENERATE-SCALAR-RESULT.

Case 1. le is ⟨absent⟩.

Convert the scalar-value of sa to <character> to obtain the scalar-result.

Case 2. le is not ⟨absent⟩.

Convert le to integer-type. The result, n, MUST NOT be negative. Convert the scalar-value of sa to <character> of length n to obtain the scalar-result.

9.5.4.14 Collate-bif

Arguments: (none)

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> has <character>.

Operation: COLLATE-BIF

Step 1. Let v be a ⟨character-string-value⟩ containing all the terminal nodes of the category ⟨symbol⟩ just once in an IMPLEMENTATION-DEFINED order known as the *collating sequence.* (See Section 9.4.2.6.1 for use of this sequence in the comparison of ⟨character-string-value⟩s).

Step 2. Return an ⟨aggregate-value⟩ containing v.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.15 Copy-bif

Arguments: sa,le

Constraint: The ⟪aggregate-type⟫s of sa and le MUST immediately contain ⟪scalar⟫. The <data-type>s of sa and le MUST have <computational-type>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The scalar-result-type has the derived <string-type> of the <data-type> of sa.

Operation: COPY-BIF(rdd,sa,le)

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 and 1.2 in either order.

Step 1.1. Convert le to integer-type. The result, n, MUST NOT be negative.

Step 1.2. Convert the scalar-value of sa to the scalar-result-type, to obtain sv.

Step 2.

Case 2.1. $n > 0$.

Let v be a null-string. Perform Step 2.1.1 n times. The scalar-result is v.

Step 2.1.1. Perform CONCATENATE(v,sv) to obtain v.

Case 2.2. $n = 0$.

The scalar-result is a null-string.

An invocation of

```
COPY('12',3)
```

will return the ⟪character-string-value⟫ '121212'.

Figure 18. An Example of the COPY Built-in Function.

9.5.4.16 Cos-bif

Arguments: x

Constraint: The ⟪aggregate-type⟫ of x MUST immediately contain ⟪scalar⟫. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type have the derived <base> and converted <precision> of the corresponding <data-type> of x.

Operation: COS-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type. The value of x is assumed to be given in radians.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the cosine of the converted scalar-value of x.

Step 3. Perform ARITHMETIC-RESULT(v,rt), to obtain the scalar-result.

9.5.4.17 Cosd-bif

Arguments: x

Constraint: The ⟪aggregate-type⟫ of x MUST immediately contain ⟪scalar⟫. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <scale> of the scalar-result-type has <float>. The <base> of the scalar-result-type is the derived <base> of the <data-type> of x. The <precision> of the scalar-result-type is the converted <precision> of the <data-type> of x.

Operation: COSD-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type. The value of x is assumed to be given in degrees.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the cosine of the converted scalar-value of x.

Step 3. Perform ARITHMETIC-RESULT(v,rt), to obtain the scalar-result.

9.5.4.18 Cosh-bif

Arguments: x

Constraint: The ⟪aggregate-type⟫ of x MUST immediately contain ⟪scalar⟫. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: COSH-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain w.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the hyperbolic cosine of w.

Step 3. Perform ARITHMETIC-RESULT(v,rt) to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.19 Currentsize-bif

The CURRENTSIZE built-in function is normally used to obtain the amount of storage, in IMPLEMENTATION-DEFINED units, that a generation of storage associated with a variable occupies. Unlike the DECLARESIZE enquiry, it is an error to invoke CURRENTSIZE when there is no generation of storage. The value returned by CURRENTSIZE is often used in implementing specialized storage management schemes and is expected to be consistent with the value that DECLARESIZE would have returned if it was invoked immediately before the allocation. Since the value returned depends on both storage mappings and the units in which storage sizes are expressed, neither of which is specified by this standard, a program using this built-in function may become dependent on a particular implementation.

Arguments: x

Constraint: x MUST be of the form

<argument>:
<expression>:
<value-reference>:
<variable-reference>,y.

y MUST NOT contain an <identifier-list>. If y has a <subscript-list>,sl then each <subscript> in sl MUST have <asterisk>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> is integer-type.

Operation: CURRENTSIZE-BIF(rdd,x)

Step 1. Perform EVALUATE-VARIABLE-REFERENCE(y) to obtain a ⟨generation⟩,g. Let edd be the ⟨evaluated-data-description⟩ in g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⟨true⟩.

Step 2. Return an ⟨aggregate-value⟩ containing an IMPLEMENTATION-DEFINED ⟨real-value⟩, depending on edd.

9.5.4.20 Date-bif

Arguments: (none)

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> has <character>.

Operation: DATE-BIF

The result has length 6. It represents the date in the form "yymmdd", where each letter represents one ⟨character-value⟩ in the result and each ⟨character-value⟩ is a digit.

yy, mm, and dd are respectively in the ranges
00:99, 01:12, and 01:31, representing
year, month, and day.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.21 Datetime-bif

Arguments: (none)

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The result <data-type> has <character>.

Operation: DATETIME-BIF

Step 1. Let v be a ⟪character-string-value⟫ of length 14+t, where t is an IMPLEMENTATION-DEFINED non-negative integer.

v represents the date and the time-of-day on a 24-hour scale, in the form CCYYMMDDhhmmss[d...], where each letter represents one ⟪character-value⟫ and each contained ⟨symbol⟩ is a ⟨digit⟩.

CC, YY, MM and DD are in the ranges
00:99, 00:99, 01:12, and 01:31, representing
centuries, years, months, and days, respectively.

hh, mm, and ss are in the ranges
00:23, 00:59, and 00:59, representing
hours, minutes, and seconds, respectively.

d... represents decimal fractions of a second.

Step 2. Return an ⟪aggregate-value⟫ containing v.

9.5.4.22 Decimal-bif

Arguments: x[,p[,q]]

Constraint: The ⟪aggregate-type⟫ of x MUST immediately contain ⟪scalar⟫. The <data-type> of x MUST have <computational-type>. Constraints on p and q are described in Section 9.5.2.2.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <base> of the scalar-result-type has <decimal>. The <scale> of the scalar-result-type is the derived <scale> of the <data-type> of x. The <precision> of the scalar-result-type is determined as defined in Section 9.5.2.2.

Operation: DECIMAL-BIF(rdd,x[,p[,q]])

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the the scalar-value of x to the scalar-result-type to obtain the scalar-result.

9.5.4.23 Dimension-bif

Arguments: x[,n]

Constraint: x MUST have the form <argument>: <expression>: <value-reference>: <variable-reference>,y: <data-description>: <dimensioned-data-description>. Each <subscript> in the <subscript-list> of y MUST have <asterisk>.

The ⟪aggregate-type⟫ of n MUST immediately contain ⟪scalar⟫. n MUST be of the form <argument>: <expression>: <constant>: <data-type>,dt;;;, where dt has <fixed> with

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

<scale-factor>:0. (Note that the <constant> may have been produced by the evaluation of an argument <expression> as a restricted expression in CREATE-BUILTIN-FUNCTION-REFERENCE, Section 4.6.5.7).

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The result <data-type> is integer-type.

Operation: DIMENSION-BIF(rdd,x[,n])

Step 1. Let y be the <variable-reference> in x.

Step 2.

Case 2.1. n is ⟪absent⟫.

The number of <bound-pair>s simply contained in the <data-description> immediate component of y MUST be exactly 1. Let j = 1.

Case 2.2. (Otherwise).

The value of n MUST be positive and not greater than the number of <bound-pair>s simply contained in the <data-description> immediate component of y.

Step 3. Perform EVALUATE-VARIABLE-REFERENCE(y) to obtain a ⟪generation⟫,g. Let k and i be the <lower-bound> and <upper-bound> respectively of the *j*th <bound-pair> in the <bound-pair-list> of g's ⟪evaluated-data-description⟫. Return an ⟪aggregate-value⟫ containing ⟪real-value⟫: (i−k+1).

```
DECLARE A(10,2:6) BASED(APTR);
ALLOCATE A;
N = DIMENSION(A,2);
```

The value of N is 5 after the execution of the statements shown.

Figure 19. An Example of the DIMENSION Built-in Function.

9.5.4.24 Divide-bif

Arguments: x,y,p[,q]

Constraint: The ⟪aggregate-type⟫s of x and y MUST immediately contain ⟪scalar⟫. The <data-type>s of x and y MUST have <arithmetic>. Constraints for p and q are described in Section 9.5.2.2.

If q is supplied, then the scalar-result-type MUST have <decimal>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <base> and <scale> of the scalar-result-type are the derived common <base> and <scale> of the <data-type>s in x and y. The <precision> of the scalar-result-type is determined as defined in Section 9.5.2.2.

Operation: DIVIDE-BIF(rdd,x,y,p[,q])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 and 1.2 in either order.

Step 1.1. Convert the scalar-value of x to the derived common <base> and <scale>of the <data-type>s of x and y, and the converted <precision> of the <data-type> of x.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.2. Convert the scalar-value of y to the derived common <base> and <scale>of the <data-type>s of x and y, and the converted <precision> of the <data-type> of y.

Step 2. If the converted scalar-value of y is zero, perform RAISE-CONDITION(<zerodivide-condition>).

Step 3. Let v be the converted scalar-value of x divided by the converted scalar-value of y.

Step 4. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.25 Edit-bif

Arguments: x,ps

Constraint: The ⟪aggregate-type⟫s of x and ps MUST immediately contain ⟪scalar⟫. The <data-type> of x MUST have <computational-type>. The <data-type> of ps MUST have <string-type>: <character>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The scalar-result-type has <character>.

Operation: EDIT-BIF(rdd,x,ps)

Perform GENERATE-SCALAR-RESULT.

Step 1. Let psl be a ⟨symbol-list⟩ which contains the following components in order:

1. a ⟨symbol⟩: ⟨'⟩;,
2. copies of the ⟨symbol⟩ components in the scalar-value of ps, in the same order as they appear in the scalar-value of ps, except that each ⟨symbol⟩: ⟨'⟩; is replaced by two occurences of ⟨symbol⟩: ⟨'⟩,
3. a ⟨symbol⟩: ⟨'⟩.

Step 2. Perform PARSE(psl,⟨picture⟩) to obtain a ⟨picture⟩,cp.

Step 3. Perform CREATE-PICTURE(cp) to obtain a <pictured>,ap.

Step 4. Let apdt be a <data-type>: <computational-type>: ap. Let xdt be the scalar <data-type> of x. Let xv be the scalar-value of x.

Step 5. Perform CONVERT(apdt,xdt,xv) to obtain the scalar-result.

9.5.4.26 Empty-bif

Arguments: (none)

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The result <data-type> has <area>.

Operation: EMPTY-BIF

Step 1. Return the ⟪area-value⟫: ⟪empty⟫.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.27 Every-bif

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <bit>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> has <bit>.

Operation: EVERY-BIF(rdd,x)

Step 1. Perform EVALUATE-EXPRESSION(x) to obtain an ⟨aggregate-value⟩,u.

Step 2. In any order, convert each scalar-element of u to <bit>, to obtain v.

Step 3.

Case 3.1. Every scalar-element of v contains ⟨null-bit-string⟩.

Let r be ⟨zero-bit⟩.

Case 3.2. At least one scalar-element of v which does not contain ⟨null-bit-string⟩ has a ⟨bit-string-value⟩ with at least one ⟨bit-value⟩ containing ⟨zero-bit⟩.

Let r be ⟨zero-bit⟩.

Case 3.3. (Otherwise).

Let r be ⟨one-bit⟩.

Step 4. Return an ⟨aggregate-value⟩ containing a ⟨bit-string-value⟩ containing r.

9.5.4.28 Exp-bif

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: EXP-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain w.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be e^{w}, where e is the base of the natural logarithm system.

Step 3. Perform ARITHMETIC-RESULT(v,rt) to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.29 Exponent-bif

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <arithmetic> (but not <pictured-numeric>) with <scale>: <float>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The scalar-result-type is integer-type.

Operation: EXPONENT-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Let dt be the scalar <data-type> of x. Perform HIGHEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⟨real-number⟩,lambda. Perform LOWEST-POSITIVE-MODEL-NUMBER(dt) to obtain a ⟨real-number⟩,sigma. Perform MODEL-BASE(dt) to obtain an ⟨integer-value⟩,b.

Step 2. Let v be the scalar-value of x.

Case 2.1. $v = 0$.

The scalar-result is 0.

Case 2.2. (Otherwise).

v MUST be such that $sigma \leq |v| \leq lambda$. The scalar-result is the unique integer e such that $b^{e-1} \leq |v| < b^{e}$.

9.5.4.30 Fileopen-bif

Arguments: fn

Constraint: fn MUST have ⟨aggregate-type⟩: ⟨scalar⟩. The <data-type> of fn MUST have <file>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> has <bit>.

Operation: FILEOPEN-BIF(rdd,fn)

Step 1. Perform EVALUATE-EXPRESSION(fn) to obtain a ⟨file-value⟩,fv. fv MUST NOT contain ⟨null-file⟩.

Step 2.

Case 2.1. The ⟨file-information⟩ designated by fv contains ⟨open⟩.

Let r be ⟨one-bit⟩.

Case 2.2. (Otherwise).

Let r be ⟨zero-bit⟩.

Step 3. Return an ⟨aggregate-value⟩ containing a ⟨bit-string-value⟩ containing r.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.31 Fixed-bif

Arguments: x,p[,q]

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <computational-type>. Constraints on p and q are given in Section 9.5.2.2.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The <scale> of the scalar-result-type has <fixed>. The <base> of the scalar-result-type is the derived <base> of the <data-type> of x. The <precision> of the scalar-result-type is determined as defined in Section 9.5.2.2.

Operation: FIXED-BIF(rdd,x,p[,q])

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain the scalar-result.

9.5.4.32 Float-bif

Arguments: x,p

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <computational-type>. Constraints on p are described in Section 9.5.2.2.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The <scale> of the scalar-result-type has <float>. The <base> of the scalar-result-type is the derived <base> of the <data-type> of x. The <precision> of the scalar-result-type is determined as defined in Section 9.5.2.2.

Operation: FLOAT-BIF(rdd,x,p)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain the scalar-result.

9.5.4.33 Floor-bif

Arguments: x

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The scalar-result-type has <arithmetic>. The scalar-result-type has the derived <base> and <scale> of the <data-type> of x.

Case 1. The scalar-result-type has <scale>: <float>.

The <precision> of the scalar-result-type is the converted <precision> of the <data-type> of x.

Case 2. The scalar-result-type has <scale>: <fixed>.

Let r denote the <number-of-digits> and s denote the <scale-factor> of the converted scalar-value of x. The <precision> of the scalar-result-type is given by:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

<number-of-digits>: min(N,max(r−s+1,1));
<scale-factor>: 0.

Operation: FLOOR-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to its derived <base>, <scale>, and converted <precision>.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the greatest integer less than or equal to the converted scalar-value of x.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.34 Hbound-bif

Arguments: x[,n]

Constraint: The ⟪aggregate-type⟫ of n MUST immediately contain ⟪scalar⟫. n MUST be of the form <argument>: <expression>: <constant>: <data-type>,dt;;;, where dt has <fixed> with <scale-factor>:0. (Note that the <constant> may have been produced by the evaluation of an argument <expression> as a restricted expression in CREATE-BUILTIN-FUNCTION-REFERENCE, Section 4.6.5.7). x MUST have the form <argument>: <expression>: <value-reference>: <variable-reference>,y: <data-description>: <dimensioned-data-description>. Each <subscript> in the <subscript-list> of y MUST have <asterisk>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The result <data-type> is integer-type.

Operation: HBOUND-BIF(rdd,x[,n])

Step 1. Let y be the <variable-reference> simply contained in x.

Step 2.

Case 2.1. n is ⟪absent⟫.

The number of <bound-pair>s simply contained in the <data-description> immediate component of y MUST be exactly 1. Let j = 1.

Case 2.2. (Otherwise).

Perform EVALUATE-EXPRESSION-TO-INTEGER(n) to obtain j. The value of j MUST be positive and not greater than the number of <bound-pair>s simply contained in the <data-description> immediate component of y.

Step 3. Perform EVALUATE-VARIABLE-REFERENCE(y) to obtain a ⟪generation⟫,g. Let k be the <upper-bound> of the *j*th <bound-pair> in the <bound-pair-list> of g's ⟪evaluated-data-description⟫. Return an ⟪aggregate-value⟫ containing ⟪real-value⟫: k.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.35 High-bif

Arguments: le

Constraint: The <data-type> of le MUST have <computational-type>. le MUST have ⩽aggregate-type⩾: ⩽scalar⩾.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The result <data-type> has <character>.

Operation: HIGH-BIF(rdd,le)

Step 1. Perform EVALUATE-EXPRESSION-TO-INTEGER(le) to obtain n. The value of n MUST NOT be negative.

Step 2.

Case 2.1. $n \neq 0$.

Let v be a ⩽character-string-value⩾ of length n containing n occurrences of the last ⩽character-value⩾ in the result of performing COLLATE-BIF.

Case 2.2. $n = 0$.

Let v be a ⩽character-string-value⩾: ⩽null-character-string⩾.

Step 3. Return an ⩽aggregate-value⩾ containing v.

9.5.4.36 Identical-bif

Arguments: x,y

Constraint: x and y MUST both be of the form

<argument>:
<expression>:
<value-reference>:
<variable-reference>.

The <data-description> of x and y MUST NOT contain a <union-data-description>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The result <data-type> has <bit>.

Operation: IDENTICAL-BIF(rdd,x,y)

Step 1. Let xr be the <variable-reference> in x. Let yr be the <variable-reference> in y.

Step 2. Perform Steps 2.1 and 2.2 in any order.

Step 2.1. Perform EVALUATE-VARIABLE-REFERENCE(xr) to obtain a ⩽generation⩾,gx. Let eddx be the ⩽evaluated-data-description⩾ in gx. Perform TEST-CONNECTEDNESS(gx) to obtain txr, which MUST be ⩽true⩾.

Step 2.2. Perform EVALUATE-VARIABLE-REFERENCE(yr) to obtain a ⩽generation⩾,gy. Let eddy be the ⩽evaluated-data-description⩾ in gy. Perform TEST-CONNECTEDNESS(gy) to obtain tyr, which MUST be ⩽true⩾.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3. eddx and eddy MUST be equal, disregarding the ⟨integer-value⟩ subnodes of <extent-expression>s.

Step 4. Let r be a ⟨zero-bit⟩.

Step 5. For each pair of corresponding <extent-expression>s, eex and eey, contained in eddx and eddy, perform Step 5.1.

Step 5.1. If eex is not equal to eey, then go to Step 10.

Step 6. Perform VALUE-OF-GENERATION(gx) to obtain an ⟨aggregate-value⟩,ax. ax MUST NOT contain ⟨undefined⟩.

Step 7. Perform VALUE-OF-GENERATION(gy) to obtain an ⟨aggregate-value⟩,ay. ay MUST NOT contain ⟨undefined⟩.

Step 8. For each pair of corresponding ⟨basic-value⟩s, bx and by, contained in ax and ay, perform Steps 8.1 and 8.2.

Step 8.1. Let dd be the <data-description> immediately contained in eddx. Let dt be the <data-type> in dd that corresponds to the ⟨basic-value⟩ bx in ax.

Step 8.2.

Case 8.2.1. dt has <pictured> or <string>.

If bx is not equal to by, then go to Step 10.

Case 8.2.2. (Otherwise).

Perform COMPARE(bx,by,dt,dt) to obtain a ⟨comparison-result⟩,cr. If cr contains ⟨not-equal⟩, then go to Step 10.

Step 9. Let r be a ⟨one-bit⟩.

Step 10. Return an ⟨aggregate-value⟩ containing a ⟨bit-string-value⟩ containing r.

9.5.4.37 Index-bif

Arguments: sa,ca[,pa]

Constraint: The ⟨aggregate-type⟩s of sa, ca and pa MUST immediately contain ⟨scalar⟩. The <data-type> of pa MUST have <fixed> (but not <pictured-numeric>) with <scale-factor>: 0. The <data-type>s of sa and ca MUST both have <bit> or both have <character>

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The scalar-result-type is integer-type.

Operation: INDEX-BIF(rdd,sa,ca[,pa])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 through 1.3 in any order.

Step 1.1. Convert the scalar-value of sa to the derived common <string-type> of sa and ca to obtain sb.

Step 1.2. Convert the scalar-value of ca to the derived common <string-type> of sa and ca to obtain cb.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1.3.

Case 1.3.1. pa is ⟨absent⟩.

Let pb be 1.

Case 1.3.2. pa is not ⟨absent⟩.

Convert the scalar-value of pa to integer-type to obtain pb.

Step 2. Let sl be the length of sb. Let cl be the length of cb. sl and pb MUST be such that $1 \leq pb \leq sl+1$.

Step 3.

Case 3.1. sl = 0 or
cl = 0 or
cl+pb > sl+1.

The scalar-result is 0.

Case 3.2. (Otherwise).

Case 3.2.1. There exists some j which satisfies the following conditions:

1. for i=1,...,cl, the *i*th ⟨bit-value⟩ or ⟨character-value⟩ of cb is equal to the *(i−1+j)*th ⟨bit-value⟩ or ⟨character-value⟩ of sb.
2. $j \geq pb$.

The scalar-result is the smallest such j.

Case 3.2.2. (Otherwise).

The scalar-result is 0.

The integer 3 is the result of evaluating:

```
INDEX('ABCDABCD','CD')
```

The integer 5 is the result of evaluating:

```
INDEX('ABABAB','AB',4)
```

Figure 20. Examples of the INDEX Built-in Function.

9.5.4.38 Isochar-bif

The ISOCHAR built-in function is intended to provide standard PL/I programs with a conventional way to access control and other special characters independent of the external character set(s) of an implementation. The row and column positions of the ISO character tables are used to identify a specific character and the associated semantics. The implementation is expected to translate those semantics to the appropriate corresponding characters for that implementation. This arrangement is intended to permit, when needed, both the eight-bit to seven-bit transformations specified in ANSI X3.41-1974 and ISO 2022-1986, or the eight-bit codings of ANSI X3.134.1 (under development), ISO 2022-1986, or ISO 4873-1984 or completely different character sets appropriate to a specific implementation.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The arguments to the function are defined in terms of the control characters terminology (C0 and C1 sets) of ANSI X3.134.1 (under development) and ISO 2022-1986. Applicability to graphics characters would be a permitted extension, but programs taking advantage of such features would, of course, be inappropriate for interchange since the "semantics" of graphic character positions are not unambiguously defined.

Arguments: c,r

Constraint: c and r MUST be of the form <argument>: <expression>: <constant>: <data-type>,dt;;; where dt has <fixed> with <scale-factor>:0. (Note that the <constant> may have been produced by the evaluation of an argument <expression> as a restricted expression in create-builtin-function-reference, Section 4.6.5.7). The values of c and r MUST meet one of the following sets of constraints:

1. c is 0, 1, 8 or 9 and $0 \leq r \leq 15$.
2. c is 7 and r is 15.
3. c is 15 and r is 15.

Attributes: The result ⋖aggregate-type⋗ immediately contains ⋖scalar⋗. The result <data-type> has <character>.

Operation: ISOCHAR-BIF(rdd,c,r)

Step 1. Let irv be a standard character set table such as those of ANSI X3.134.1 (under development), ISO 2022-1986, or ISO 4873-1984.

Note: For the purposes for which they are used in the operation below, these tables are identical.

Step 2. Let vc be the value of c and let vr be the value of r.

Step 3. Let absc be the control character located at column vc, row vr of irv.

Step 4. Let idc be an IMPLEMENTATION-DEFINED ⋖character-string-value⋗ whose value corresponds to the semantics associated with the control character absc in other applicable standards.

Step 5. Return an ⋖aggregate-value⋗ containing idc.

9.5.4.39 Lbound-bif

Arguments: x[,n]

Constraint: The ⋖aggregate-type⋗ of n MUST immediately contain ⋖scalar⋗. n MUST be of the form <argument>: <expression>: <constant>: <data-type>,dt;;;, where dt has <fixed> with <scale-factor>:0. (Note that the <constant> may have been produced by the evaluation of an argument <expression> as a restricted expression in CREATE-BUILTIN-FUNCTION-REFERENCE, Section 4.6.5.7). x MUST have the form <argument>: <expression>: <value-reference>: <variable-reference>,y: <data-description>: <dimensioned-data-description>. Each <subscript> in the <subscript-list> of y MUST have <asterisk>.

Attributes: The result ⋖aggregate-type⋗ immediately contains ⋖scalar⋗. The result <data-type> is integer-type.

Operation: LBOUND-BIF(rdd,x[,n])

Step 1. Let y be the <variable-reference> simply contained in x.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.

Case 2.1. n is ⟨absent⟩.

The number of <bound-pair>s simply contained in the <data-description> immediate component of y MUST be exactly 1. Let j = 1.

Case 2.2. (Otherwise).

Perform EVALUATE-EXPRESSION-TO-INTEGER(n) to obtain j. The value of j MUST be positive and not greater than the number of <bound-pair>s simply contained in the <data-description> immediate component of y.

Step 3. Perform EVALUATE-VARIABLE-REFERENCE(y) to obtain a ⟨generation⟩,g. Let k be the <lower-bound> of the *j*th <bound-pair> in the <bound-pair-list> of g's ⟨evaluated-data-description⟩. Return an ⟨aggregate-value⟩ containing ⟨real-value⟩: k.

9.5.4.40 Length-bif

Arguments: sa

Constraint: The ⟨aggregate-type⟩ of sa MUST immediately contain ⟨scalar⟩. The <data-type> of sa MUST have <string>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The scalar-result-type is integer-type.

Operation: LENGTH-BIF(rdd,sa)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of sa to the derived <string-type> of sa to obtain sb.

Step 2. The scalar-result is the length of sb.

9.5.4.41 Lineno-bif

Arguments: fn

Constraint: fn MUST have ⟨aggregate-type⟩: ⟨scalar⟩. The <data-type> of fn MUST have <file>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> is integer-type.

Operation: LINENO-BIF(rdd,fn)

Step 1. Perform EVALUATE-EXPRESSION(fn) to obtain a ⟨file-value⟩,fv. fv MUST NOT contain a ⟨null-file⟩. The ⟨file-information⟩,fi, designated by fv MUST contain ⟨open⟩. The ⟨evaluated-file-description⟩ of fi MUST contain <print>.

Step 2. Perform EVALUATE-CURRENT-LINE(fv) to obtain an ⟨integer-value⟩,iv. Return an ⟨aggregate-value⟩ containing iv.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.42 Log-bif

Arguments: x

Constraint: The ≤aggregate-type≥ of x MUST immediately contain ≤scalar≥. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ≤aggregate-type≥ immediately contains ≤scalar≥. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: LOG-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain u. u MUST be greater than 0.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the natural logarithm of u.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.43 Log10-bif

Arguments: x

Constraint: The ≤aggregate-type≥ of x MUST immediately contain ≤scalar≥. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ≤aggregate-type≥ immediately contains ≤scalar≥. The scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: LOG10-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain u. The value of u MUST be greater than 0.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the common logarithm, i.e. base 10, of u.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.44 Log2-bif

Arguments: x

Constraint: The ≤aggregate-type≥ of x MUST immediately contain ≤scalar≥. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ≤aggregate-type≥ immediately contains ≤scalar≥. The scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: LOG2-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain u. The value of u MUST be greater than 0.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the base 2 logarithm of u.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.45 Low-bif

Arguments: le

Constraint: The <data-type> of le MUST have <computational-type>. le MUST have ⩽aggregate-type⩾: ⩽scalar⩾.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The result <data-type> has <character>.

Operation: LOW-BIF(rdd,le)

Step 1. Perform EVALUATE-EXPRESSION-TO-INTEGER(le) to obtain n. The value of n MUST NOT be negative.

Step 2.

Case 2.1. $n \neq 0$.

Let v be a ⩽character-string-value⩾ of length n containing n occurrences of the first ⩽character-value⩾ in the result of performing COLLATE-BIF.

Case 2.2. $n = 0$.

Let v be a ⩽character-string-value⩾: ⩽null-character-string⩾.

Step 3. Return an ⩽aggregate-value⩾ containing v.

9.5.4.46 Max-bif

Arguments: x1,x2

Constraint: The ⩽aggregate-type⩾s of x1 and x2 MUST immediately contain ⩽scalar⩾. The <data-type>s of x1 and x2 MUST have <arithmetic>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The <base> and <scale> of the scalar-result-type are the derived common <base> and <scale> of the <data-type>s of the <argument>s.

Case 1. The derived common <scale> of the <data-type>s of x1 and x2 has <float>.

The <precision> of the scalar-result-type is max(p1,p2),where p1 and p2 are the converted <number-of-digits> of the <data-type>s of x1 and x2.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 2. The derived common <scale> of the <data-type>s of x1 and x2 has <fixed>.

Let (p1,q1) and (p2,q2) be the converted <number-of-digits> and <scale-factor> of the <data-type>s of x1 and x2 respectively. Then the <precision> of the scalar-result-type is given by:

<number-of-digits>: min(N,max(p1−q1,p2−q2) + max(q1,q2));
<scale-factor>: max(q1,q2).

Operation: MAX-BIF(rdd,x1,x2)

Perform GENERATE-SCALAR-RESULT.

Step 1. In any order, convert the scalar-values of x1 and x2 to the scalar-result-type. Let the converted scalar-values be u1 and u2.

Step 2. The scalar-result is max(u1,u2).

9.5.4.47 Maxlength-bif

Arguments: sa

Constraint: The ⩽aggregate-type⩾ of sa MUST immediately contain ⩽scalar⩾. sa MUST be of the form <argument>: <expression>: <value-reference>: <variable-reference>,sr.

The <data-type> of sr MUST have <string-type>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The result <data-type> is integer-type.

Operation: MAXLENGTH-BIF(rdd,sa)

Step 1. Perform EVALUATE-VARIABLE-REFERENCE(sr) to obtain a ⩽generation⩾,sg.

Step 2. Let sdd be the <data-description> in the ⩽evaluated-data-description⩾ in sg.

Step 3. Let d be the <data-type> in sdd.

Step 4. Let ml be the component of the ⩽integer-value⩾ in the <maximum-length> of d.

Step 5. Let len be a ⩽basic-value⩾: ⩽real-value⩾: ⩽real-number⩾: ml.

Step 6. Return an ⩽aggregate-value⩾ whose ⩽aggregate-type⩾ immediately contains ⩽scalar⩾, and whose ⩽basic-value-list⩾ contains len.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Consider the following PL/I code fragment.

```
DECLARE   X      CHAR(50) VARYING;

          CALL E(X);

E:        PROCEDURE(Y);

DECLARE   Y      CHAR(*) VARYING,
          B      BIT(30);

          M = MAXLENGTH(Y);
          N = MAXLENGTH(B);

          END E;
```

During execution of the above statements, M is assigned a value of 50, and N is assigned a value of 30.

The following invocations of the MAXLENGTH built-in function are invalid because the argument of MAXLENGTH MUST be a <variable-reference>.

```
MAXLENGTH('ABC')
MAXLENGTH(X||Y)
```

Figure 21. Some Examples of the MAXLENGTH Built-in Function.

9.5.4.48 Min-bif

Arguments: x1,x2

Constraint: The ⟨aggregate-type⟩s of x1 and x2 MUST immediately contain ⟨scalar⟩. The <data-type>s of x1 and x2 MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The <base> and <scale> of the scalar-result-type are the derived common <base> and <scale> of the <data-type>s of the <argument>s.

Case 1. The derived common <scale> of <data-type>s of x1 and x2 has <float>.

The <precision> of the scalar-result-type is max(p1,p2), where p1 and p2 are the converted <number-of-digits> of the <data-type>s of x1 and x2.

Case 2. The derived common <scale> of <data-type>s of x1 and x2 has <fixed>.

Let (p1,q1) and (p2,q2) be the converted <number-of-digits> and <scale-factor> of the <data-type>s of x1 and x2 respectively. Then the <precision> of the scalar-result-type is given by:

<number-of-digits>: min(N,max(p1−q1,p2−q2) + max(q1,q2));
<scale-factor>: max(q1,q2).

Operation: MIN-BIF(rdd,x1,x2)

Perform GENERATE-SCALAR-RESULT.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1. In any order, convert the scalar-values of x1 and x2 to the scalar-result-type. Let the converted scalar-values be u1 and u2.

Step 2. The scalar-result is min(u1,u2).

9.5.4.49 Mod-bif

Arguments: x,y

Constraint: The ⟪aggregate-type⟫s of x and y MUST immediately contain ⟪scalar⟫. The <data-type>s of x and y MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <base> and <scale> of the scalar-result-type are the derived common <base> and <scale> of the <data-type>s of x and y.

Case 1. The <scale> of the scalar-result-type has <float>.

The <precision> of the scalar-result-type is the greater of the converted <precision>s of the <data-type>s of x and y.

Case 2. The <scale> of the scalar-result-type has <fixed>.

Let (p[1],q[1]) and (p[2],q[2]) be the respective converted <number-of-digits> and <scale-factor> of the <data-type>s of x and y. Then the <precision> of the scalar-result-type is given by:

<number-of-digits>: min(N,p[2]−q[2]+max(q[1],q[2]));
<scale-factor>: max(q[1],q[2]).

Operation: MOD-BIF(rdd,x,y)

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 and 1.2 in either order.

Step 1.1. Convert the scalar-value of x to the derived common <base> and <scale> of the <data-type>s of x and y, and the converted <precision> of the <data-type> of x.

Step 1.2. Convert the scalar-value of y to the derived common <base> and <scale> of the <data-type>s of x and y, and the converted <precision> of the <data-type> of y.

Step 2. Let the converted scalar-value of x be u and the converted scalar-value of y be w. The value of w MUST NOT be 0.

Step 3. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be given by:

v=u−w*floor(u/w).

Step 4. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.50 Multiply-bif

Arguments: x,y,p[,q]

Constraint: The ⩽aggregate-type⩾s of x and y MUST immediately contain ⩽scalar⩾. The <data-type>s of x and y MUST have <arithmetic>. Constraints on p and q are given in Section 9.5.2.2.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The <base> and <scale>of the scalar-result-type are the derived common <base> and <scale> of the <data-type>s of x and y. The <precision> of the scalar-result-type is determined as defined in Section 9.5.2.2.

Operation: MULTIPLY-BIF(rdd,x,y,p[,q])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 and 1.2 in either order.

Step 1.1. Convert the scalar-value of x to the derived common <base> and <scale> of the <data-type>s of x and y, and the converted <precision> of the <data-type> of x.

Step 1.2. Convert the scalar-value of y to the derived common <base> and <scale> of the <data-type>s of x and y, and the converted <precision> of the <data-type> of y.

Step 2. Let v be the product of the converted scalar-value of x and the converted scalar-value of y.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.51 Null-bif

Arguments: (none)

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The result <data-type> has <pointer>.

Operation: NULL-BIF

Step 1. Return an ⩽aggregate-value⩾ containing ⩽pointer-value⩾: ⩽null⩾.

9.5.4.52 Offset-bif

Arguments: pt,ar

Constraint: The ⩽aggregate-type⩾s of pt and ar MUST immediately contain ⩽scalar⩾. The <data-type> of pt MUST have <pointer>. The <data-type> of ar MUST have <area>. ar MUST have the form <argument>: <expression>: <value-reference>: <variable-reference>. pt MUST be of the form

<argument>:
<expression>:
<value-reference>:
<builtin-function-reference>:
<builtin-function>:
<addr-bif>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The scalar-result-type has <offset> with no subnode.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: OFFSET-BIF(rdd,pt,ar)

Perform GENERATE-SCALAR-RESULT.

Step 1. Let odt be a <data-type> containing <offset>: vr; where vr is the <variable-reference> in ar.

Step 2. Convert the scalar-value of pt to <data-type>,odt to obtain the scalar-result.

9.5.4.53 Oncode-bif

Arguments: (none)

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The result <data-type> is integer-type.

Operation: ONCODE-BIF

Step 1. Perform GET-ESTABLISHED-ONVALUE(⟪oncode-value⟫) to obtain j.

Case 1.1. j is ⟪fail⟫.

Let v be 0.

Case 1.2. (Otherwise).

Let v be j, an IMPLEMENTATION-DEFINED value.

Step 2. Return an ⟪aggregate-value⟫ containing ⟪real-value⟫: v.

9.5.4.54 Onfile-bif

Arguments: (none)

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The result <data-type> has <character>.

Operation: ONFILE-BIF

Step 1. Perform GET-ESTABLISHED-ONVALUE(⟪onfile-value⟫) to obtain sa.

Case 1.1. sa is ⟪fail⟫.

Let v be a ⟪character-string-value⟫: ⟪null-character-string⟫.

Case 1.2. (Otherwise).

Let v be sa.

Step 2. Return an ⟪aggregate-value⟫ containing v.

9.5.4.55 Onkey-bif

Arguments: (none)

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The result <data-type> has <character>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: ONKEY-BIF

Step 1. Perform GET-ESTABLISHED-ONVALUE(≤onkey-value≥) to obtain sa.

Case 1.1. sa is ≤fail≥.

Let v be a ≤character-string-value≥: ≤null-character-string≥.

Case 1.2. (Otherwise).

Let v be sa.

Step 2. Return an ≤aggregate-value≥ containing v.

9.5.4.56 Onsource-bif

Arguments: (none)

Attributes: The result ≤aggregate-type≥ immediately contains ≤scalar≥. The result <data-type> has <character>.

Operation: ONSOURCE-BIF

Step 1. Perform GET-ESTABLISHED-ONVALUE(≤onsource-value≥) to obtain sa.

Case 1.1. sa is ≤fail≥.

Let v be a ≤character-string-value≥: ≤null-character-string≥.

Case 1.2. (Otherwise).

Let v be sa.

Step 2. Return an ≤aggregate-value≥ containing v.

9.5.4.57 Pageno-bif

Arguments: fn

Constraint: fn MUST have ≤aggregate-type≥: ≤scalar≥. The <data-type> of fn MUST have <file>.

Attributes: The result ≤aggregate-type≥ immediately contains ≤scalar≥. The result <data-type> is integer-type.

Operation: PAGENO-BIF(rdd,fn)

Step 1. Perform EVALUATE-EXPRESSION(fn) to obtain a ≤file-value≥,fv. fv MUST NOT contain a ≤null-file≥. The ≤file-information≥,fi, designated by fv MUST contain ≤open≥. The ≤evaluated-file-description≥ of fi MUST contain <print>.

Step 2. Let v be the ≤integer-value≥ in ≤page-number≥ in fi.

Step 3. Return an ≤aggregate-value≥ containing ≤real-value≥: v.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.58 Pointer-bif

Arguments: ofe,ar

Constraint: The ⟪aggregate-type⟫s of ofe and ar MUST immediately contain ⟪scalar⟫. The <data-type> of ofe MUST have <offset>. The <data-type> of ar MUST have <area>. ar MUST have the form <argument>: <expression>: <value-reference>: <variable-reference>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The scalar-result-type has <pointer>.

Operation: POINTER-BIF(rdd,ofe,ar)

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 and 1.2 in either order.

Step 1.1. Perform EVALUATE-EXPRESSION(ofe) to obtain ofs.

Step 1.2. Let odt be a <data-type> containing <offset>: vr; where vr is the <variable-reference> in ar.

Step 2. Perform CONVERT(pdt,odt,ofs) to obtain the scalar-result, where pdt is a <data-type> containing <pointer>.

9.5.4.59 Prod-bif

Arguments: x

Constraint: x MUST be of the form <argument>: <expression>: <data-description>: <dimensioned-data-description>: <element-data-description>: <item-data-description>. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <base> of the result <data-type> is the derived <base> of the <data-type> of x.

Case 1. The derived <scale> of the <data-type> of x has <fixed> and the converted <scale-factor> has 0.

The result <data-type> has <fixed> and <number-of-digits>: N. The result <data-type> has <scale-factor>: 0.

Case 2. (Otherwise).

The result <data-type> has <float> and its <number-of-digits> is the converted <number-of-digits> of the <data-type> of x.

Operation: PROD-BIF(rdd,x)

Step 1. Perform EVALUATE-EXPRESSION(x) to obtain an ⟪aggregate-value⟫,v.

Step 2. Convert the scalar-elements of v to the result <data-type> in any order. Let w be the product of the converted values.

Step 3. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the <data-type> component of rdd.

Step 4. Perform ARITHMETIC-RESULT(w,rt), where rt is the <data-type> of rdd, to obtain z. Return an ⟪aggregate-value⟫ containing z.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.60 Reverse-bif

Arguments: sa

Constraint: The ⩽aggregate-type⩾ of sa MUST immediately contain ⩽scalar⩾. The <data-type> of sa MUST have <string>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The scalar-result-type has the derived <string-type> of the <data-type> of sa.

Operation: REVERSE-BIF(rdd,sa)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of sa to the scalar-result-type to obtain sb.

Step 2.

Case 2.1. sb is a null-string.

The scalar-result is a null-string.

Case 2.2. sb is not a null-string.

Let m be the length of sb. The scalar-result is a string of length m whose *i*th ⩽bit-value⩾ or ⩽character-value⩾ is the *(m−i+1)*th ⩽bit-value⩾ or ⩽character-value⩾ in sb.

9.5.4.61 Round-bif

Arguments: x,n

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <decimal> or <float>. n MUST be of the form <argument>: <expression>: <constant>: <data-type>,dt;;;, where dt has <fixed> with <scale-factor>:0. (Note that the <constant> may have been produced by the evaluation of an argument <expression> as a restricted expression in CREATE-BUILTIN-FUNCTION-REFERENCE, Section 4.6.5.7). If the derived <scale> of the <data-type> of x has <float>, the value of n MUST be greater than zero; otherwise, the value of n MUST be greater than or equal to zero.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The <base> and <scale> of the scalar-result-type are the derived <base> and <scale> of the <data-type> of x.

Case 1. The derived <scale> of the <data-type> of x has <fixed>.

Let r and s be the converted <number-of-digits> and <scale-factor> of the corresponding <data-type> of x.

The <precision> of the scalar-result-type is given by:

<number-of-digits>: max(1,min(r−s+1+n,N));
<scale-factor>: n.

Case 2. The derived <scale> of the <data-type> of x has <float>.

The <number-of-digits> in the scalar-result-type is given by min(n,N).

Operation: ROUND-BIF(rdd,x,n)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to its derived <base>, <scale>, and converted <precision>. Let v be the converted value.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let b=2 or b=10 according as the scalar-result-type has <binary> or <decimal>.

Step 3.

Case 3.1. The <scale> of the <data-type> of v has <fixed>.

Let $v'=sign(v)*(b^{-n})*floor(|v|*(b^{n})+1/2)$. Let w be a ⩽real-value⩾,v'.

Case 3.2. The <scale> of the <data-type> of v has <float>.

Let w be v. (The result in this case is determined entirely by ROUNDED-RESULT, which is invoked by ARITHMETIC-RESULT).

Step 4. Perform ARITHMETIC-RESULT(w,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.62 Search-bif

Arguments: sa,ca[,pa]

Constraint: The ⩽aggregate-type⩾s of sa, ca and pa MUST immediately contain ⩽scalar⩾. The <data-type> of pa MUST have <fixed> (but not <pictured-numeric>) with <scale-factor>: 0. The <data-type>s of sa and ca MUST have <character>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The scalar-result-type is integer-type.

Operation: SEARCH-BIF(rdd,sa,ca[,pa])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 through 1.3 in any order.

Step 1.1. Convert the scalar-value of sa to <character> to obtain sb.

Step 1.2. Convert the scalar-value of ca to <character> to obtain cb.

Step 1.3.

Case 1.3.1. pa is ⩽absent⩾.

Let pb be 1.

Case 1.3.2. pa is not ⩽absent⩾.

Convert the scalar-value of pa to integer-type to obtain pb.

Step 2. Let m be the length of sb. m and pb MUST be such that $1 \leq pb \leq m+1$.

Step 3.

Case 3.1. $m-pb+1 = 0$.

The scalar-result is 0.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3.2. (Otherwise).

Let sbc[i], i=1,...,m, be the *i*th ⟨character-value⟩ of sb.

Case 3.2.1. Each sbc[i], i=pb,...,m, does not occur in cb.

The scalar-result is 0.

Case 3.2.2. (Otherwise).

Let j be the smallest value of i for which sbc[i] occurs in cb and i ≥ pb.

The scalar-result is j.

9.5.4.63 Sign-bif

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The scalar-result-type is integer-type.

Operation: SIGN-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to its derived <scale>, <base>, and converted <precision> to obtain v.

Step 2.

Case 2.1. v>0.

The scalar-result is ⟨real-value⟩: 1.

Case 2.2. v=0.

The scalar-result is ⟨real-value⟩: 0.

Case 2.3. v<0.

The scalar-result is ⟨real-value⟩: −1.

9.5.4.64 Sin-bif

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contain ⟨scalar⟩. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: SIN-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type. The scalar-value of x is assumed to be given in radians.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the sine of the converted scalar-value of x.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.65 Sind-bif

Arguments: x

Constraint: The ⟪aggregate-type⟫ of x MUST immediately contain ⟪scalar⟫. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: SIND-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type. The value of x is assumed to be given in degrees.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the sine of the converted scalar-value of x.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.66 Sinh-bif

Arguments: x

Constraint: The ⟪aggregate-type⟫ of x MUST immediately contain ⟪scalar⟫. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contain ⟪scalar⟫. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: SINH-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain w.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the hyperbolic sine of w.

Step 3. Perform ARITHMETIC-RESULT(v,rt) to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.67 Some-bif

Arguments: x

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <bit>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The result <data-type> has <bit>.

Operation: SOME-BIF(rdd,x)

Step 1. Perform EVALUATE-EXPRESSION(x) to obtain an ⩽aggregate-value⩾,u.

Step 2. In any order, convert each scalar-element of u to <bit>.

Step 3.

Case 3.1. At least one ⩽bit-value⩾ in some converted scalar-element has ⩽one-bit⩾.

Let r be ⩽one-bit⩾.

Case 3.2. (Otherwise).

Let r be ⩽zero-bit⩾.

Step 4. Return an ⩽aggregate-value⩾ containing a ⩽bit-string-value⩾ containing r.

9.5.4.68 Sqrt-bif

Arguments: x

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: SQRT-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain y.

Step 2. The value of y MUST NOT be negative. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let w be the positive square root of y.

Step 3. Perform ARITHMETIC-RESULT(w,rt), where rt is the scalar-result-type, to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.69 String-bif

Arguments: sa

Constraint: All <data-type>s of sa MUST have <computational-type>. sa MUST be of the form <argument>: <expression>: <value-reference>: <variable-reference>,sv. The <data-description> of sa MUST NOT contain a <union-data-description>. Let sd be the <declaration> designated by the <declaration-designator> in sv. The result of performing TEST-STRING-CLASS(sd) must be either <bit> or <character>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The result <data-type> has the derived common <string-type> of the <data-type>s of sa.

Operation: STRING-BIF(rdd,sa)

Step 1. Perform EVALUATE-VARIABLE-REFERENCE(sd) to obtain a ⟪generation⟫,g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⟪true⟫. Perform VALUE-OF-GENERATION(g) to obtain an ⟪aggregate-value⟫,v.

Step 2. In any order, convert all the scalar-elements in v to the derived common <string-type> of all the <data-type>s of sa. Let the converted values be cv[i] in the order of the scalar-elements of v.

Step 3. Let r be a null-string. For i=1 to number-of-scalar-elements of v, perform Step 3.1 .

Step 3.1. Perform CONCATENATE(r,cv[i]) to obtain r.

Step 4. Return an ⟪aggregate-value⟫ containing r.

9.5.4.70 Substr-bif

Arguments: sa,st[,le]

Constraint: The ⟪aggregate-type⟫s of sa, st and le MUST immediately contain ⟪scalar⟫. The <data-type>s of st and le MUST have <fixed> (but not <pictured-numeric>) with <scale-factor>: 0. The <data-type> of sa MUST have <string>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The scalar-result-type has <string> with the derived <string-type> of the <data-type> of sa.

Operation: SUBSTR-BIF(rdd,sa,st[,le])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 to 1.3 in any order.

Step 1.1. Convert the scalar-value of sa to the scalar-result-type to obtain sb. Let the length of sb be k.

Step 1.2. Convert the scalar-value of st to integer-type to obtain i.

Step 1.3. If le is not ⟪absent⟫, convert its scalar-value to integer-type to obtain j.

Step 2. If le is ⟪absent⟫, let j=k−i+1.

Step 3. j and k MUST be such that:

$$1 \le i \le k + 1$$
$$0 \le j \le k - i + 1.$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4.

Case 4.1. j = 0.

The scalar-result is a null-string.

Case 4.2. j > 0.

The scalar-result is a string of length j whose *n*th ⟪bit-value⟫ or ⟪character-value⟫ ($1 \leq n \leq j$) is the *(i+n−1)*th ⟪bit-value⟫ or ⟪character-value⟫ in the string sb, n=1,...,j.

9.5.4.71 Subtract-bif

Arguments: x,y,p[,q]

Constraint: The ⟪aggregate-type⟫s of x and y MUST immediately contain ⟪scalar⟫. The <data-type>s of x and y MUST have <arithmetic>. Constraints on p and q are described in Section 9.5.2.2.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <base> and <scale>of the scalar-result-type are the derived common <base> and <scale> of the <data-type>s of x and y. The <precision> of the scalar-result-type is determined as defined in Section 9.5.2.2.

Operation: SUBTRACT-BIF(rdd,x,y,p[,q])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 and 1.2 in either order.

Step 1.1. Convert the scalar-value of x to the derived common <base> and <scale> of the <data-type>s of x and y, and the converted <precision> of the <data-type> of x.

Step 1.2. Convert the scalar-value of y to the derived common <base> and <scale> of the <data-type>s of x and y, and the converted <precision> of the <data-type> of y.

Step 2. Let v be the difference between the converted scalar-value of x and the converted scalar-value of y.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.72 Sum-bif

Arguments: x

Constraint: x MUST be of the form <argument>: <expression>: <data-description>: <dimensioned-data-description>: <element-data-description>: <item-data-description>. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟪aggregate-type⟫ immediately contains ⟪scalar⟫. The <base> and <scale> of the result <data-type> are the derived <base> and <scale> of the <data-type> of x.

Case 1. The derived <scale> of the <data-type> of x has <fixed>.

Let r be the converted <scale-factor> of the <data-type> of x. Then the <precision> of the result <data-type> has <number-of-digits>: N; and <scale-factor>: r.

Case 2. The derived <scale> of the <data-type> of x has <float>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The <precision> of the result <data-type> has the converted <precision> of the <data-type> of x.

Operation: SUM-BIF(rdd,x)

Step 1. Perform EVALUATE-EXPRESSION(x) to obtain an ⩽aggregate-value⩾,v.

Step 2. In any order, convert the scalar-elements of v to the result <data-type>. Let w be the sum of the converted values.

Step 3. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the <data-type> component of rdd.

Step 4. Perform ARITHMETIC-RESULT(w,rt), where rt is the <data-type> component of rdd, to obtain z. Return an ⩽aggregate-value⩾ containing z.

9.5.4.73 Tally-bif

Arguments: sa,ca

Constraint: The ⩽aggregate-type⩾s of sa and ca MUST immediately contain ⩽scalar⩾. The <data-type>s of sa and ca MUST both have <bit> or both have <character>.

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The scalar-result-type is integer-type.

Operation: TALLY-BIF(rdd,sa,ca)

Perform GENERATE-SCALAR-RESULT.

Step 1. In either order, convert the scalar-values of sa and ca to their derived common <string-type>. Let sb be the converted scalar-value of sa, and cb be the converted scalar-value of ca.

Step 2.

Case 2.1. The length of either sb or cb is 0.

The scalar-result is 0.

Case 2.2. (Otherwise).

Case 2.2.1. cb is not a substring of sb.

The scalar-result is 0.

Case 2.2.2. cb is a substring of sb.

Let i denote the number of distinct values of j such that j denotes the position of the first ⩽bit-value⩾ or ⩽character-value⩾ of an occurrence of cb in sb.

The scalar-result is i.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The result of evaluating:

```
TALLY('0100101010'B,'1'B)
```

is the integer: 4

The result of evaluating:

```
TALLY('ABC XYZ ABC','ABC')
```

is the integer: 2

The result of evaluating:

```
TALLY('AAAAA','AA')
```

is the integer: 4

The result of evaluating:

```
TALLY('','A')
```

is the integer: 0

Figure 22. Some Examples of the TALLY Built-in Function.

9.5.4.74 Tan-bif

Arguments: x

Constraint: The ⩽aggregate-type⩾ of x MUST immediately contain ⩽scalar⩾. The <data-type> of x MUST have <arithmetic>

Attributes: The result ⩽aggregate-type⩾ immediately contains ⩽scalar⩾. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: TAN-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type. Its value is assumed to be given in radians, and MUST NOT be an odd multiple of $\pi/2$.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the tangent of the converted scalar-value of x.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.75 Tand-bif

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: TAND-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type. The converted scalar-value is assumed to be given in degrees, and MUST NOT be an odd multiple of 90 degrees.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the tangent of the converted scalar-value of x.

Step 3. Perform ARITHMETIC-RESULT(v,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.76 Tanh-bif

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contain ⟨scalar⟩. The <scale> of the scalar-result-type has <float>. The <base> and <precision> of the scalar-result-type are the derived <base> and converted <precision> of the <data-type> of x.

Operation: TANH-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain w.

Step 2. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type. Let v be the hyperbolic tangent of w.

Step 3. Perform ARITHMETIC-RESULT(v,rt) to obtain the scalar-result.

9.5.4.77 Time-bif

Arguments: (none)

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> has <character>.

Operation: TIME-BIF

Step 1. Let v be a ⟨character-string-value⟩ of length 6+t, where t is an IMPLEMENTATION-DEFINED non-negative integer. v represents the time-of-day on a 24-hour scale, in the form hhmmss[d...], where each letter represents one ⟨character-value⟩ and each contained ⟨symbol⟩ is a ⟨digit⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

hh,mm,ss are respectively in the ranges 00:23, 00:59, and 00:59, representing hours, minutes, and seconds. d... represents decimal fractions of a second.

Step 2. Return an ⟨aggregate-value⟩ containing v.

9.5.4.78 Translate-bif

Arguments: sa,ra[,pa]

Constraint: The ⟨aggregate-type⟩s of sa, ra and pa MUST immediately contain ⟨scalar⟩. The <data-type>s of sa, ra and pa MUST have <character>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The scalar-result-type has <character>.

Operation: TRANSLATE-BIF(rdd,sa,ra[,pa])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 and 1.2 in either order.

Step 1.1. Convert the scalar-value of sa to <character> to obtain sb. Let m be the length of sb.

Step 1.2.

Case 1.2.1. pa is not ⟨absent⟩.

Convert the scalar-value of pa to <character> to obtain pb.

Case 1.2.2. pa is ⟨absent⟩.

Perform COLLATE-BIF to obtain pb.

Step 2. Convert the scalar-value of ra to <character>. If the length of this converted scalar-value is less than the length of pb, then convert it to a <character> string of the length of pb. Let rb denote the converted scalar-value of ra.

Step 3.

Case 3.1. The string sb is a null-string.

The scalar-result is a null-string.

Case 3.2. The string sb is not a null-string.

For each i, $1 \leq i \leq m$, let s[i] be the *i*th ⟨character-value⟩ in sb and let t[i] be the *i*th ⟨character-value⟩ in the scalar-result. The string pb is searched for the leftmost occurrence of s[i]. If s[i] is not found in pb, then let t[i] be s[i]. If s[i] occurs in pb, let j be the ordinal of the occurrence in pb and let t[i] be the *j*th ⟨character-value⟩ in the string rb. The scalar-result is t.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The result of evaluating:

```
TRANSLATE( '1 2', '0', ' ' )
```

is the ≤character-string-value≥ '102'.

Figure 23. An Example of the TRANSLATE Built-in Function.

9.5.4.79 Trim-bif

Arguments: sa[,la,ra]

Constraint: The ≤aggregate-type≥s of sa, la and ra MUST immediately contain ≤scalar≥. The <data-type>s of sa, la and ra MUST have <character>.

Attributes: The result ≤aggregate-type≥ immediately contains ≤scalar≥. The scalar-result-type has <character>.

Operation: TRIM-BIF(rdd,sa[,la,ra])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 thru 1.3 in any order.

Step 1.1. Convert the scalar-value of sa to <character> to obtain sb. Let m be the length of sb.

Step 1.2.

Case 1.2.1. la is not ≤absent≥.

Convert the scalar-value of la to <character> to obtain lb.

Case 1.2.2. la is ≤absent≥.

Let lb be a ≤character-string-value≥: ≤character-value≥: ∅.

Step 1.3.

Case 1.3.1. ra is not ≤absent≥.

Convert the scalar-value of ra to <character> to obtain rb.

Case 1.3.2. ra is ≤absent≥.

Let rb be a ≤character-string-value≥: ≤character-value≥: ∅.

Step 2.

Case 2.1. The string sb is a null-string.

The scalar-result is a null-string.

Case 2.2. The string sb is not a null-string.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2.2.1.

Case 2.2.1.1. All ⟨character-value⟩s of sb occur in lb.

Let li be m+1.

Case 2.2.1.2. (Otherwise).

Let li be such that the *li*th ⟨character-value⟩ of sb is the leftmost ⟨character-value⟩ of sb that does not occur in lb.

Step 2.2.2.

Case 2.2.2.1. All ⟨character-value⟩s of sb occur in rb.

Let ri be 0.

Case 2.2.2.2. (Otherwise).

Let ri be such that the *ri*th ⟨character-value⟩ of sb is the rightmost ⟨character-value⟩ of sb that does not occur in rb.

Step 2.2.3.

Case 2.2.3.1. li ≤ ri.

The scalar-result is a ⟨character-string-value⟩ of length ri−li+1 whose *n*th ⟨character-value⟩ is the *(li+n−1)*th ⟨character-value⟩ in the string sb, for n = 1,...,ri−li+1.

Case 2.2.3.2. li > ri

The scalar-result is a null-string.

The result of evaluating:

```
TRIM('  0123. ', ' 0', ' .')
```

is the ⟨character-string-value⟩ '123'.

Figure 24. An Example of the TRIM Built-in Function.

9.5.4.80 Trunc-bif

Arguments: x

Constraint: The ⟨aggregate-type⟩ of x MUST immediately contain ⟨scalar⟩. The <data-type> of x MUST have <arithmetic>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The <base> and <scale> of the scalar-result-type are the derived <base> and <scale> of the <data-type> of x.

Case 1. The derived <scale> of the <data-type> of x has <fixed>.

Let r be the converted <number-of-digits> and s be the converted <scale-factor> of the <data-type> of x. Then the <precision> of the scalar-result-type is given by:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

<number-of-digits>: min(N,max(r−s+1,1));
<scale-factor>: 0.

Case 2. The derived <scale> of the <data-type> of x has <float>.

The <precision> of the scalar-result-type is the converted <precision> of the <data-type> of x.

Operation: TRUNC-BIF(rdd,x)

Perform GENERATE-SCALAR-RESULT.

Step 1. Convert the scalar-value of x to the scalar-result-type to obtain v. Perform CONDITIONS-IN-ARITHMETIC-EXPRESSION(rt), where rt is the scalar-result-type.

Step 2.

Case 2.1. $v < 0$.

Let w=ceil(v).

Case 2.2. $v = 0$.

Let w=0.

Case 2.3. $v > 0$.

Let w=floor(v).

Step 3. Perform ARITHMETIC-RESULT(w,rt), where rt is the scalar-result-type, to obtain the scalar-result.

9.5.4.81 Unspec-bif

The UNSPEC built-in function is normally used to obtain the internal representation of a variable as a bit string. Its use makes a PL/I program dependent on the particular implementation and the representations it uses, which are not specified by this standard.

Arguments: x

Constraint: x MUST be of the form <argument>: <expression>: <value-reference>: <variable-reference>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The result <data-type> has <bit>.

Operation: UNSPEC-BIF(rdd,x)

Step 1. Let y be the <variable-reference> simply contained in x.

Step 2. Perform EVALUATE-VARIABLE-REFERENCE(y) to obtain a ⟨generation⟩,g. Perform TEST-CONNECTEDNESS(g) to obtain tv, which MUST be ⟨true⟩.

Step 3. Return an ⟨aggregate-value⟩ containing a ⟨bit-string-value⟩,v, which depends on the properties of g in an IMPLEMENTATION-DEFINED manner. This value may be ⟨undefined⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.4.82 Valid-bif

Arguments: sa

Constraint: The ⟨aggregate-type⟩ of sa MUST immediately contain ⟨scalar⟩. The <data-type> of sa MUST have <pictured>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The scalar-result-type has <bit>.

Operation: VALID-BIF(rdd,sa)

Perform GENERATE-SCALAR-RESULT.

Step 1. Let sv be the scalar-value of sa; let dt be the scalar <data-type> corresponding to sa. Perform VALIDATE-NUMERIC-PICTURED-VALUE(dt,sv) to obtain a ⟨picture-validity⟩,pv.

Step 2.

Case 2.1. pv has ⟨picture-valid⟩.

The scalar-result is a ⟨bit-string-value⟩: ⟨bit-value-list⟩: ⟨bit-value⟩: ⟨one-bit⟩.

Case 2.2. pv has ⟨picture-invalid⟩.

The scalar-result is a ⟨bit-string-value⟩: ⟨bit-value-list⟩: ⟨bit-value⟩: ⟨zero-bit⟩.

9.5.4.83 Verify-bif

Arguments: sa,ca[,pa]

Constraint: The ⟨aggregate-type⟩s of sa, ca and pa MUST immediately contain ⟨scalar⟩. The <data-type> of pa MUST have <fixed> (but not <pictured-numeric>) with <scale-factor>: 0. The <data-type>s of sa and ca MUST have <character>.

Attributes: The result ⟨aggregate-type⟩ immediately contains ⟨scalar⟩. The scalar-result-type is integer-type.

Operation: VERIFY-BIF(rdd,sa,ca[,pa])

Perform GENERATE-SCALAR-RESULT.

Step 1. Perform Steps 1.1 through 1.3 in any order.

Step 1.1. Convert the scalar-value of sa to <character> to obtain sb.

Step 1.2. Convert the scalar-value of ca to <character> to obtain cb.

Step 1.3.

Case 1.3.1. pa is ⟨absent⟩.

Let pb be 1.

Case 1.3.2. pa is not ⟨absent⟩.

Convert the scalar-value of pa to integer-type to obtain pb.

Step 2. Let m be the length of sb. m and pb MUST be such that $1 \leq pb \leq m+1$.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3.

Case 3.1. m−pb+1 = 0.

The scalar-result is 0.

Case 3.2. (Otherwise).

Let sbc[i], i=1,...,m, be the *i*th ≪character-value≫ of sb.

Case 3.2.1. Each sbc[i], i=pb,...,m, occurs in cb.

The scalar-result is 0.

Case 3.2.2. (Otherwise).

Let j be the smallest value of i for which sbc[i] does not occur in cb and i ≥ pb.

The scalar-result is j.

```
VERIFY('297B', '0123456789')
```

will return the value 4, i.e., the position of the first non-numeric character in the string '297B'

```
VERIFY('A=123,B=456', '0123456789',3)
```

will return the value 6, i.e., the position of the first non-numeric character (',') in the string 'A=123,B=456' after the first '=' character.

Figure 25. Examples of the VERIFY Built-in Function.

9.6 Conversion

9.6.1 Conversion of Scalar Values

The operation CONVERT is used to convert a basic-value, sv, into a new basic-value consistent with the target <data-type>,tt. The operand, st is a <data-type> associated with sv; for example, it may be the <data-type> of an <expression> whose evaluation yielded sv.

One of the following three relationships always holds for the operands:

1. The <data-type>s, tt and st, have <computational-type> and sv is a ≪real-value≫, ≪character-string-value≫, or ≪bit-string-value≫ that does not contain ≪undefined≫.

2. The <data-type>, tt has <offset>, st has <pointer>, and sv is a ≪pointer-value≫.

3. The <data-type>, tt has <pointer>, st has <offset>, and sv is an ≪offset-value≫.

In (2) and (3), the <offset> contains a <variable-reference>.

Note that CONVERT may be invoked informally (see Section 9.6.1.1).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: CONVERT(tt,st,sval)

Where: tt is a <data-type>,
st is a <data-type>,
sval is

a ⟨real-value⟩,sv,
or a ⟨character-string-value⟩,sv,
or a ⟨bit-string-value⟩,sv,
or a ⟨pointer-value⟩,sv,
or an ⟨offset-value⟩,sv,
or a ⟨basic-value⟩ containing an immediate component, sv, of one of the above categories,
or an ⟨aggregate-value⟩ containing a ⟨basic-value⟩ with such an immediate component, sv.

Result: a ⟨real-value⟩, or a ⟨character-string-value⟩, or a ⟨bit-string-value⟩, or a ⟨pointer-value⟩, or an ⟨offset-value⟩.

Case 1. tt has <fixed> but not <pictured-numeric>, and sv is a ⟨real-value⟩.

Perform CONVERT-TO-FIXED(tt,st,sv) to obtain cv. Return cv.

Case 2. tt has <float> but not <pictured-numeric>, and sv is a ⟨real-value⟩.

Perform CONVERT-TO-FLOAT(tt,st,sv) to obtain cv. Return cv.

Case 3. tt is <arithmetic> but not <pictured-numeric>, and sv is a ⟨character-string-value⟩ or a ⟨bit-string-value⟩.

Step 3.1. Perform CONVERT-TO-ARITHMETIC(st,sv) to obtain a ⟨value-and-type⟩: rv rdt.

Step 3.2. Perform CONVERT(tt,rdt,rv) to obtain v. Return v.

Case 4. tt is <pictured-numeric> and sv is a ⟨real-value⟩, ⟨bit-string-value⟩, or ⟨character-string-value⟩.

Step 4.1. If tt and st are identical, optionally return sv.

Step 4.2. Let att be the associated arithmetic data-type (see Section 9.6.2) of tt.

Step 4.3.

Case 4.3.1. att has <fixed>.

Perform CONVERT(att,st,sv) to obtain v.

Case 4.3.2. att has <float>.

Step 4.3.2.1.

Case 4.3.2.1.1. sv is a ⟨character-string-value⟩ or a ⟨bit-string-value⟩.

Perform CONVERT-TO-ARITHMETIC(st,sv) to obtain a ⟨value-and-type⟩ whose first two components are cv and ct.

Case 4.3.2.1.2. sv is a ⟨real-value⟩.

Let cv be sv; let ct be st.

Step 4.3.2.2. Perform CONVERT-TO-FLOAT-DECIMAL(att,ct,cv) to obtain v.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4.4. Perform EDIT-NUMERIC-PICTURE(v,tt) to obtain a ⟨character-string-value⟩,csv.

Return csv.

Case 5. tt has <string>, and sv is a ⟨real-value⟩, ⟨bit-string-value⟩, or ⟨character-string-value⟩.

Step 5.1. If tt has <bit>, perform CONVERT-TO-BIT(st,sv) to obtain s; otherwise, perform CONVERT-TO-CHARACTER(st,sv) to obtain s.

Step 5.2. Let n be the <maximum-length> component of tt. If n has <asterisk>, return s.

Step 5.3. Let m be the length of s.

Step 5.4.

Case 5.4.1. $n = 0$.

Return a null-string.

Case 5.4.2. $m \geq n > 0$.

Return a string whose length is n and whose *i*th ⟨bit-value⟩ or ⟨character-value⟩ is the same as the *i*th ⟨bit-value⟩ or ⟨character-value⟩ in s, i=1,...,n.

Case 5.4.3. $m < n$.

If tt has <varying>, then return s; otherwise return a string whose length is n, whose *i*th ⟨bit-value⟩ or ⟨character-value⟩ is the same as the *i*th ⟨bit-value⟩ or ⟨character-value⟩ in s for $1 \leq i \leq m$, and whose remaining ⟨bit-value⟩s or ⟨character-value⟩s have, respectively, ⟨zero-bit⟩s or ∅s.

Case 6. tt has <pointer>, st has <offset>: <variable-reference>,vr; and sv is an ⟨offset-value⟩.

Step 6.1. Perform Steps 6.1.1 and 6.1.2 in either order.

Step 6.1.1. If sv contains ⟨null⟩, then return ⟨pointer-value⟩: ⟨null⟩.

Step 6.1.2. Perform EVALUATE-VARIABLE-REFERENCE(vr) to obtain a ⟨generation⟩,g.

Step 6.2. Let edd, aal, and sil be, respectively, the ⟨evaluated-data-description⟩, ⟨area-action-list⟩, and ⟨storage-index-list⟩ components of sv. Perform VALUE-OF-GENERATION(g) to obtain an ⟨aggregate-value⟩ which has an ⟨area-value⟩,av.

Step 6.3. Perform VALIDATE-OFFSET(sv,av).

Step 6.4. Let aa be the ⟨area-allocation⟩ in the ⟨area-action-list⟩ of av that corresponds to the last ⟨area-allocation⟩ in aal.

Step 6.5. Let aud be an ⟨allocation-unit-designator⟩ that designates the ⟨allocation-unit⟩ in aa. Return a ⟨pointer-value⟩: ⟨generation⟩: edd aud sil.

Case 7. tt has <offset>: <variable-reference>,vr;, st has <pointer> and sv is a ⟨pointer-value⟩.

Step 7.1. Perform Steps 7.1.1 and 7.1.2 in either order.

Step 7.1.1. If sv contains ⟨null⟩, then return ⟨offset-value⟩: ⟨null⟩.

Step 7.1.2. Perform EVALUATE-VARIABLE-REFERENCE(vr) to obtain a ⟨generation⟩,g.

Step 7.2. Let edd, aud, and sil be, respectively, the ⟨evaluated-data-description⟩, ⟨allocation-unit-designator⟩ and ⟨storage-index-list⟩ components of sv. Perform VALUE-OF-GENERATION(g) to obtain an ⟨aggregate-value⟩ which has an ⟨area-value⟩,av.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 7.3. Let aal1 be a copy of the ⩽area-action-list⩾,aal in av. aud must designate an ⩽allocation-unit⩾ in an ⩽area-action⩾,aa of aal.

Step 7.4. Delete all ⩽area-action⩾s in aal1 which follow the ⩽area-action⩾ in aal1 corresponding to aa in aal. Delete all ⩽allocation-unit⩾s from aal1.

Step 7.5. Return an ⩽offset-value⩾: edd aal1 sil.

9.6.1.1 Informal Invocation of Convert

The operation CONVERT may be invoked informally by text of the form "convert source-value to target-type", where "source-value" is text describing the third operand, sv, and "target-type" is text describing the first operand, tt. The result returned by CONVERT may be referred to as "the converted value".

The second operand, st, may be specified explicitly in following text that describes the *source-type* for the conversion. Alternatively, if the source-type is not explicitly specified, and sv was obtained by performing EVALUATE-EXPRESSION(x), the source-type is by implication the scalar <data-type> of x that corresponds to the ⩽basic-value⩾,sv.

The description of the operand tt need not be complete. The following conventions are applied to complete the description of the operand tt:

1. When tt has <string>, its <maximum-length> has <asterisk> unless the contrary is explicitly specified by a phrase such as "of specified length n", and it has <varying> unless <nonvarying> is explicitly specified.

2. When <base> and <scale> components are specified, then tt has <arithmetic> and not <pictured-numeric> unless <pictured-numeric> is explicitly specified.

9.6.1.2 Convert-to-fixed

Operation: CONVERT-TO-FIXED(tt,st,sv)

Where: tt is a <data-type> that has <fixed>,
st is a <data-type>,
sv is a ⩽real-value⩾.

Result: a ⩽real-value⩾.

Step 1. Let b=2 or b=10, according as tt has <binary> or <decimal>. Let p and q be the <number-of-digits> and <scale-factor> of tt.

Step 2. Let cv = (b^{-q})*sign(sv)*floor((b^{q})* | sv |).

Step 3.

Case 3.1. st has <fixed> and <scale-factor>: 0;, or st has <fixed> and the same <base> as tt.

Let v = cv.

Case 3.2. (Otherwise).

Let v be an implementation-dependent approximation to cv such that $v=w*b^{-q}$ for some integer w.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Step 4.

Case 4.1. $|v| \geq b^{p-q}$ and $|v| < b^{N-q}$, where N is the maximum <number-of-digits> allowed for the <base> and <scale> of tt.

Perform RAISE-CONDITION(<size-condition>).

Case 4.2. $|v| \geq b^{N-q}$.

Optionally perform RAISE-CONDITION(<fixedoverflow-condition>); otherwise, perform RAISE-CONDITION(<size-condition>).

Case 4.3. (Otherwise).

Return a ⟨real-value⟩: v.

9.6.1.3 Convert-to-float

Operation: CONVERT-TO-FLOAT(tt,st,sv)

Where: tt is a <data-type> that has <float>,
st is a <data-type>,
sv is a ⟨real-value⟩.

Result: a ⟨real-value⟩.

Step 1.

Case 1.1. st has <float> (but not <pictured-numeric>) and the same <base> as tt, and sv is not a component of a ⟨value-and-type⟩ produced by EVALUATE-REAL-CONSTANT, BASIC-NUMERIC-VALUE, or CONVERT-TO-ARITHMETIC.

Perform CHOPPED-RESULT(sv,tt) to obtain a ⟨real-value⟩,r.

Case 1.2. (Otherwise).

Step 1.2.1. Let b=2 or b=10, according as tt has <binary> or <decimal>. Let p be the <number-of-digits> of tt.

Step 1.2.2.

Case 1.2.2.1. sv is an integer such that $|sv| < b^p$.

Let r be sv.

Case 1.2.2.2. (Otherwise).

Perform UNCONSTRAINED-RESULT(sv,tt) to obtain a ⟨real-value⟩,r.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.6.1.4 Convert-to-bit

Operation: CONVERT-TO-BIT(st,sv)

Where: st is a <data-type>,
sv is a ⩽real-value⩾, ⩽bit-string-value⩾, or ⩽character-string-value⩾.

Result: a ⩽bit-string-value⩾.

Case 1. st has <arithmetic> (including <arithmetic> in <pictured-numeric>).

Step 1.1.

Case 1.1.1. sv is a ⩽real-value⩾.

Let v= | sv | . Let ct be st.

Case 1.1.2. st has <pictured-numeric>, and sv is a ⩽character-string-value⩾.

Perform VALIDATE-NUMERIC-PICTURED-VALUE(st,sv) to obtain ⩽picture-validity⩾,w. w MUST have ⩽picture-valid⩾. The ⩽picture-valid⩾ component of w then immediately contains a ⩽real-value⩾,x. Let v = | x | . Let ct be the associated arithmetic data-type of st.

Step 1.2. Let r be the <number-of-digits> of ct, and let s be the <scale-factor> of ct. Let N be the maximum <precision> for <fixed> and <binary>. According to the <base> and <scale> of ct, determine p as follows:

<binary>	<fixed>	:	$p = \min(N,\max(r-s,0))$
<decimal>	<fixed>	:	$p = \min(N,\max(\text{ceil}(\log_2(10)*(r-s)),0))$
<binary>	<float>	:	$p = \min(N,r)$
<decimal>	<float>	:	$p = \min(N,\text{ceil}(\log_2(10)*r))$.

Step 1.3. If p=0, return a ⩽bit-string-value⩾: ⩽null-bit-string⩾. Otherwise let tt be a <data-type> that has <fixed>, and <binary> with <number-of-digits>: p; and <scale-factor>: 0. Perform CONVERT(tt,ct,v) to obtain a non-negative integer, n.

Step 1.4. If $n \geq 2^p$, then perform RAISE-CONDITION(<size-condition>); otherwise n can be exactly represented as the sum of $c[i]*2^{p-i}$ for i=1,...,p where each c[i] is 0 or 1.

Return a ⩽bit-string-value⩾ of length p, whose *i*th ⩽bit-value⩾ has a ⩽zero-bit⩾ or ⩽one-bit⩾ according as c[i] is 0 or 1.

Case 2. st has <bit>, and sv is a ⩽bit-string-value⩾.

Return sv.

Case 3. st has <character>, and sv is a ⩽character-string-value⩾.

Step 3.1. If sv contains a ⩽character-value⩾ that has neither ⩽0⩾ nor ⩽1⩾, go to Step 3.3 .

Step 3.2. If sv has a ⩽null-character-string⩾, return the ⩽bit-string-value⩾: ⩽null-bit-string⩾. Otherwise return a ⩽bit-string-value⩾, whose length equals the length of sv, and whose *i*th ⩽bit-value⩾ has a ⩽zero-bit⩾ or ⩽one-bit⩾ according as the *i*th ⩽character-value⩾ in sv has ⩽0⩾ or ⩽1⩾.

Step 3.3. Let cbifs be a ⩽condition-bif-value-list⩾ containing ⩽onsource-value⩾: sv. Perform RAISE-CONDITION(<conversion-condition>,cbifs). Let sv be the immediate component of the current ⩽returned-onsource-value⩾, and go to Step 3.1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.6.1.5 Convert-to-character

In this operation the notation '(n)⟨picture-element⟩' (or some other lowercase letter in place of n) is used to indicate n ($n \geq 0$) consecutive occurrences of that ⟨picture-element⟩.

Operation: CONVERT-TO-CHARACTER(st,sv)

Where: st is a <data-type>,
sv is a ⟨real-value⟩, ⟨bit-string-value⟩, or ⟨character-string-value⟩.

Result: a ⟨character-string-value⟩.

Case 1. st has <fixed> but not <pictured-numeric>.

Step 1.1. If st has <binary>, let dst be a <data-type> which has <fixed> and <decimal> with the converted <precision> of st for that target; otherwise, let dst be st. Perform CONVERT(dst,st,sv) to obtain cv.

Let p and q be the <number-of-digits> and <scale-factor> of dst.

Step 1.2.

Case 1.2.1. $p \geq q \geq 0$.

Let pic be the <data-type> corresponding to the ⟨picture⟩ with concrete-representation as follows:

if q=0:	'(2)B(p)−9'
if p=q>0:	'−9V.(q)9'
if p>q>0:	'B(n)−9V.(q)9' where n=p−q.

Perform EDIT-NUMERIC-PICTURE(cv,pic) to obtain a ⟨character-string-value⟩,v.

Return v.

Case 1.2.2. $q > p$.

Let pic1 be the <data-type> corresponding to the ⟨picture⟩ with concrete-representation '(p)−9'.

Perform EDIT-NUMERIC-PICTURE(x,pic1) to obtain a ⟨character-string-value⟩,v1, where x is a ⟨real-value⟩: $cv*10^q$.

Let k be the unique integer such that $10^{k-1} \leq |q| < 10^k$. Let pic2 be the <data-type> corresponding to the ⟨picture⟩ with concrete-representation '<F>S(k)9'. Perform EDIT-NUMERIC-PICTURE(−q,pic2) to obtain a ⟨character-string-value⟩,v2.

Perform CONCATENATE(v1,v2) to obtain s2.

Return s2.

Case 2. st has <float> but not <pictured-numeric>.

Step 2.1. If st has <binary>, let dst be a <data-type> which has <float> and <decimal> with the converted <precision> of st for that target; otherwise, let dst be st. Perform CONVERT-TO-FLOAT-DECIMAL(dst,st,sv) to obtain cv.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Let p be the <number-of-digits> of dst.

Step 2.2. There is a unique representation of cv as $cm * 10^{cx}$, where cx is a signed integer such that:

$$1 \leq |cm| < 10 \quad \text{if } cv \neq 0$$
$$\text{or } cm = cx = 0 \quad \text{if } cv = 0$$

Step 2.3. Let pic1 be the <data-type> corresponding to the ⟨picture⟩ with concrete-representation '-9V.(n)9', where n=p−1. Perform EDIT-NUMERIC-PICTURE(cm,pic1) to obtain a ⟨character-string-value⟩,v1.

Step 2.4. Let k be the IMPLEMENTATION-DEFINED size of the exponent field for <floating-point-format>s. Let pic2 be the <data-type> corresponding to the ⟨picture⟩ with concrete-representation '<E>S(k)9'. Perform EDIT-NUMERIC-PICTURE(cx,pic2) to obtain a ⟨character-string-value⟩,v2.

Step 2.5. Perform CONCATENATE(v1,v2) to obtain v. Return v.

Case 3. st has <bit> and sv is a ⟨bit-string-value⟩.

If sv has the ⟨null-bit-string⟩, return ⟨character-string-value⟩: ⟨null-character-string⟩. Otherwise return a ⟨character-string-value⟩ whose length equals the length of sv, and whose *i*th ⟨character-value⟩ has ⟨0⟩ or ⟨1⟩ according to whether the *i*th ⟨bit-value⟩ of sv has ⟨zero-bit⟩ or ⟨one-bit⟩.

Case 4. st has <character>, or <pictured-numeric>, and sv is a ⟨character-string-value⟩.

Return sv.

9.6.1.6 Conversion to Float Decimal

This operation returns a ⟨real-value⟩ which is exactly representable in floating notation in a given number (p) of significant digits. It is used in the conversion of numeric values to character representations.

Operation: CONVERT-TO-FLOAT-DECIMAL(tt,st,sv)

Where: tt is a <data-type> which has <float> and <decimal>,
st is a <data-type> which has <arithmetic>,
sv is a ⟨real-value⟩.

Result: a ⟨real-value⟩.

Step 1. If sv is 0, return a ⟨real-value⟩: 0.

Step 2. Let p be the <number-of-digits> in tt. Let e be the unique integer such that $10^{e-1} \leq |sv| < 10^e$.

Let $cv = 10^{e-p} * sign(sv) * floor(10^{p-e} * |sv| + 0.5)$.

Step 3.

Case 3.1. st has <fixed> and <decimal>.

Return a ⟨real-value⟩: cv.

Case 3.2. (Otherwise).

Return a ⟨real-value⟩ containing an implementation-dependent approximation to cv, having the form $m * 10^n$, where m and n are integers and $|m| < 10^p$.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.6.1.7 Conversion from String or Picture to Arithmetic

Operation: CONVERT-TO-ARITHMETIC(st,sv)

Where: st is a <data-type>,
sv is a ⟨character-string-value⟩ or a ⟨bit-string-value⟩.

Result: a ⟨value-and-type⟩ (see Section 9.6.1.8).

Case 1. st has <pictured-numeric>.

Step 1.1. Perform VALIDATE-NUMERIC-PICTURED-VALUE(st,sv) to obtain a ⟨picture-validity⟩,pv, which MUST have ⟨picture-valid⟩.

Step 1.2. Let rv be the ⟨real-value⟩ in pv. Let ast be the associated arithmetic data-type of st. Return a ⟨value-and-type⟩: rv ast.

Case 2. st has <bit>.

Step 2.1.

Case 2.1.1. sv has ⟨null-bit-string⟩.

Let cv=0 and n=1.

Case 2.1.2. (Otherwise).

Let n be the length of sv. Let x[i] be 0 or 1 according as the *i*th ⟨bit-value⟩ of sv is ⟨zero-bit⟩ or ⟨one-bit⟩; let cv be the sum of $x[i]*2^{n-i}$, for $1 \le i \le n$.

Step 2.2. Let ct be a <data-type> containing <fixed>, <binary>, <number-of-digits>: n; and <scale-factor>: 0.

Return a ⟨value-and-type⟩: ⟨real-value⟩: cv; ct.

Case 3. st has <character>.

Step 3.1.

Case 3.1.1. sv conforms to the syntax for ⟨numeric-string⟩ (see Section 9.6.1.8).

Perform BASIC-NUMERIC-VALUE(sv) to obtain a ⟨value-and-type⟩,vt. Return vt.

Case 3.1.2. (Otherwise).

Let cbifs be a ⟨condition-bif-value-list⟩ containing ⟨onsource-value⟩: sv.

Perform RAISE-CONDITION(<conversion-condition>,cbifs). Let sv be the immediate component of the current ⟨returned-onsource-value⟩, and go to Step 3.1.

9.6.1.8 Basic Numeric Value of a String

⟨value-and-type⟩ ::= ⟨real-value⟩ <data-type>

⟨numeric-string⟩ ::= ⟨blanks⟩ | ⟨blanks⟩ [+ | −] {⟨arithmetic-constant⟩} ⟨blanks⟩

⟨blanks⟩ ::= [{⟨blank⟩-list}]

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Operation: BASIC-NUMERIC-VALUE(str)

Where: str is a ⟨character-string-value⟩ whose terminal components are permitted terminal components of ⟨numeric-string⟩.

Result: a ⟨value-and-type⟩.

Step 1. Let ns be the ⟨numeric-string⟩ whose components are the elements of str, taken in order.

Step 2.

Case 2.1. ns contains only ⟨blanks⟩.

Return a ⟨value-and-type⟩ with components ⟨real-value⟩: 0; and <data-type> containing <arithmetic> with <fixed>, <decimal>, <number-of-digits>: 1;, and <scale-factor>: 0.

Case 2.2. ns immediately contains ⟨arithmetic-constant⟩: ⟨real-constant⟩,rc.

Perform EVALUATE-REAL-CONSTANT(rc) to obtain a ⟨value-and-type⟩,vt. If ns immediately contains ⟨−⟩, return a ⟨value-and-type⟩ equal to vt except that the sign of the ⟨real-value⟩ is negative; otherwise, return vt.

9.6.1.9 Evaluate-real-constant

Operation: EVALUATE-REAL-CONSTANT(rc)

Where: rc is a ⟨real-constant⟩.

Result: a ⟨value-and-type⟩ (see Section 9.6.1.8).

Step 1. Let v be the ⟨real-value⟩ obtained by interpreting the ⟨decimal-number⟩ in rc as a decimal constant.

Let b = 10.

Step 2. Let ar be a <data-type> containing <arithmetic> containing <base>: <decimal>; and <scale>and <precision> as defined below:

Step 2.1. <number-of-digits> is the total number of ⟨digit⟩s in ⟨decimal-number⟩.

Step 2.2.

Case 2.2.1. rc contains ⟨E⟩.

If ⟨exponent⟩ exists in rc, let x be the ⟨integer-value⟩ obtained by interpreting the components of ⟨exponent⟩ as a decimal constant; otherwise, let x=0. <scale> has <float>.

Case 2.2.2. (Otherwise).

<scale> has <fixed>. <scale-factor> is q, where q is the number of ⟨digit⟩s following ⟨.⟩ (if any) in ⟨decimal-number⟩. Let x=0.

Step 3.

Case 3.1. ar has <fixed>.

The <number-of-digits> in ar MUST NOT exceed the maximum <number-of-digits> allowed for the <base> of ar and <scale>: <fixed>.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 3.2. ar has <float>.

Let N be the maximum <number-of-digits> allowed for the <base> of ar and <scale>: <float>, and let n be the value of the <integer> in the <number-of-digits> of ar. If n exceeds N, replace the <integer> in the <number-of-digits> of ar by N.

Note: An implementation may ignore digits beyond the *N*th since later conversions in CREATE-CONSTANT and PROMOTE-AND-CONVERT always approximate the result to N digits.

Step 4. Return a ⟪value-and-type⟫: ⟪real-value⟫: $v*b^x$; ar.

9.6.1.10 Validate-offset

Operation: VALIDATE-OFFSET(ov,av)

Where: ov is an ⟪offset-value⟫,
av is an ⟪area-value⟫.

Step 1. Let oal be a copy of the ⟪area-action-list⟫ in ov. Let aal be a copy of the ⟪area-action-list⟫ in av.

Step 2. Delete all ⟪allocation-unit⟫s from aal. Let no be the number of ⟪area-action⟫s in oal. Let na be the number of ⟪area-action⟫s in aal.

Step 3. no MUST be less than or equal to na.

Step 4. For i=1,...,no, the *i*th ⟪area-action⟫ in oal MUST be equal to the *i*th ⟪area-action⟫ in aal.

Step 5. The ⟪area-allocation⟫ in av that corresponds to the last ⟪area-allocation⟫ in ov must have an ⟪allocation-unit⟫.

9.6.2 Numeric Pictures

Informally, <pictured-numeric> is a way of holding a numeric value in a ⟪character-string-value⟫. For a <pictured-numeric> in a <data-type>, the ⟪character-string-value⟫ will be a value in the machine-state and represents a real numeric value; for a <pictured-numeric> in a <format-item> the value will be a string of characters transmitted to or from a ⟪stream-dataset⟫, an <expression> in a <get-string>, or a <variable-reference>in a <put-string>, and represents a real numeric value.

The <numeric-picture-specification> in <pictured-numeric> specifies:

1. The <precision> and <scale>of the decimal numeric values which may be held.

   Together with <base>: <decimal>; and a <scale>: <fixed>;, these make up the <arithmetic> subnode of <pictured-numeric> established in Section 4.6.6.1. The <data-type> which contains <arithmetic> with the same subnodes is known as the *associated arithmetic data-type* of the <pictured-numeric> <data-type> or <format-item>.

2. The constant length of the ⟪character-string-value⟫ representations of numeric values.

   The *associated character-string length* of a <numeric-picture-specification>,np or ⟨picture-element-list⟩,np is equal to nv+ni+ncrdb, where:

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

nv is the number of ⟨picture-element⟩s in np which do not immediately contain ⟨V⟩ or ⟨insertion-string⟩,
ni is the number of ⟨insertion-symbol⟩s in np,
ncrdb is the number of ⟨credit⟩s or ⟨debit⟩s in np (i.e. 0 or 1).

The *associated character-string length* of a <data-type> (or <format-item>) containing <pictured-numeric>is equal to that of its <numeric-picture-specification>.

3. Exactly how the ⟨real-value⟩ is to be edited into or extracted from the ⟨character-string-value⟩.

Section 9.6.2.1 defines the editing of the ⟨real-value⟩ into the ⟨character-string-value⟩ under the control of the <pictured-numeric> specification; Section 9.6.2.2 is a sub-operation of Section 9.6.2.1 which edits a single field (see below) of a picture.

Section 9.6.2.3 defines the reverse process: checking the validity of a ⟨character-string-value⟩ against a <pictured-numeric> specification, and extracting the associated ⟨real-value⟩ from a valid ⟨character-string-value⟩; Section 9.6.2.4 is a sub-operation of Section 9.6.2.3 which checks the validity of (and extracts the value from) a single field (see below) of a picture.

9.6.2.1 Editing Numeric Pictures

Operation: EDIT-NUMERIC-PICTURE(vc,pic)

Where: vc is a ⟨real-value⟩,
pic is a <data-type> containing <pictured-numeric> with associated arithmetic data-type, adt.

Result: a ⟨character-string-value⟩.

Step 1. Let p and q be the <number-of-digits> and <scale-factor> of adt. There is a unique representation of vc in terms of integers d[j] :

$$\text{sgn} * \sum_{j=1}^{p} (d[j] * 10^{p-q-j})$$

where:

$0 \le d[j] \le 9, j=1,...,p$

$\text{sgn} = +1$ if $vc \ge 0$
$\text{sgn} = -1$ if $vc < 0$.

Step 2. Perform EDIT-NUMERIC-PICTURE-FIELD(spec,d,sgn) to obtain a ⟨character-string-value⟩,s, where:

spec is the ⟨picture-element-list⟩ of pic,
d is the ⟨character-string-value⟩ of length p whose *j*th ⟨character-value⟩has the ⟨symbol⟩ which is the ⟨digit⟩ that represents d[j].

Step 3. Return s.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.6.2.2 Editing a Numeric Picture Field

⟨pic-status⟩ ::= ⟨suppression⟩ [⟨suppression-type⟩]

⟨suppression⟩ ::= ⟨on⟩ | ⟨off⟩

⟨suppression-type⟩ ::= [⟨$⟩ | ⟨S⟩ | ⟨+⟩ | ⟨−⟩ | ⟨Z⟩ | ⟨*⟩ | ⟨(⟩ | ⟨insertion-string⟩]

Operation: EDIT-NUMERIC-PICTURE-FIELD(pic,d,sgn)

Where: pic is a ⟨picture-element-list⟩ containing ns ⟨picture-element⟩s,
d is a ⟨character-string-value⟩ containing p ⟨symbol⟩s, where p is the number of digit-positions in pic and each ⟨symbol⟩ corresponds to a digit,
sgn is the value +1 or −1.

Result: a ⟨character-string-value⟩ of length nc, where nc is the associated character-string length of pic.

Step 1. If sgn = −1, perform RAISE-CONDITION(⟨size-condition⟩) unless pic contains at least one ⟨picture-element⟩ which immediately contains ⟨S⟩, ⟨+⟩, ⟨−⟩, ⟨(⟩, ⟨credit⟩, or ⟨debit⟩.

Step 2. If pic contains no ⟨picture-element⟩ which immediately contains ⟨9⟩, and if all ⟨symbol⟩s in d have ⟨0⟩, then return a ⟨character-string-value⟩ of length nc all of whose ⟨character-value⟩s contain ⟨*⟩ or ∅ according to whether pic contains an ⟨*⟩ or not.

Step 3. Let s[i], i=1,...,ns, be the ⟨picture-element⟩s in pic.

Let d[j], j=1,...,p, be the ⟨character-value⟩s in d.

Let c[k], k=1,...,nc, be ⟨character-value⟩s; the remainder of this operation completes the trees c[k] as a function of the s[i] and d[j].

Let pstat be a ⟨pic-status⟩: ⟨suppression⟩,sup: ⟨off⟩.

Let each of i, j, k be initially 1.

Step 4. Select the appropriate Case depending on s[i].

Case 4.1. s[i] immediately contains ⟨9⟩.

Step 4.1.1.

Case 4.1.1.1. sup is ⟨suppression⟩: ⟨on⟩.

Perform Step 7.

Case 4.1.1.2. sup is ⟨suppression⟩: ⟨off⟩.

No action.

Step 4.1.2. Attach to c[k] the immediate component of d[j].

Step 4.1.3. i=i+1; j=j+1; k=k+1.

Case 4.2. s[i] immediately contains ⟨Z⟩ or ⟨*⟩.

Step 4.2.1. If pstat does not have ⟨suppression-type⟩, replace sup by ⟨suppression⟩: ⟨on⟩; and append ⟨suppression-type⟩: pc; to pstat, where pc is the component of s[i].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 4.2.2.

Case 4.2.2.1. sup is ⟨suppression⟩: ⟨on⟩; and d[j] has ⟨0⟩.

If s[i] has ⟨Z⟩, attach ∅ to c[k];
if s[i] has ⟨*⟩, attach ⟨*⟩ to c[k].

Case 4.2.2.2. sup is ⟨suppression⟩: ⟨off⟩; or d[j] does not have ⟨0⟩.

Attach the immediate component of d[j] to c[k]. Replace sup by ⟨suppression⟩: ⟨off⟩.

Step 4.2.3. i=i+1; j=j+1; k=k+1.

Case 4.3. s[i] immediately contains ⟨$⟩, ⟨S⟩, ⟨+⟩, ⟨−⟩, or ⟨(⟩, and there is no other n such that s[n] = s[i].

Step 4.3.1. Attach to c[k] the ⟨symbol⟩ obtained from Figure 26 on page 397 as a function of s[i] and (possibly) sgn.

Step 4.3.2. i=i+1; k=k+1.

Case 4.4.

1. s[i] immediately contains ⟨$⟩, ⟨S⟩, ⟨+⟩, ⟨−⟩, or ⟨(⟩, and s[n] = s[i] for some n greater than i, but for no n less than i, or,

2. s[i] immediately contains an ⟨insertion-string⟩ and some s[n] (i<n≤ns) immediately contains ⟨>⟩.

Step 4.4.1. Replace sup by ⟨suppression⟩: ⟨on⟩; append ⟨suppression-type⟩: pc; to pstat, where pc is the component of s[i].

Step 4.4.2. Attach ∅ to c[k].

Step 4.4.3. i=i+1; k=k+1.

Case 4.5.

1. s[i] immediately contains ⟨$⟩, ⟨S⟩, ⟨+⟩, ⟨−⟩, or ⟨(⟩, and s[n] = s[i] for some n less than i, or,

2. s[i] immediately contains ⟨>⟩.

Step 4.5.1.

Case 4.5.1.1. sup is ⟨suppression⟩: ⟨on⟩; and d[j] has ⟨0⟩.

Attach ∅ to c[k].

Case 4.5.1.2. sup is ⟨suppression⟩: ⟨on⟩; and d[j] does not have ⟨0⟩.

Perform Step 7.

Case 4.5.1.3. sup is ⟨suppression⟩: ⟨off⟩.

Attach the immediate component of d[j] to c[k].

Step 4.5.2. i=i+1; j=j+1; k=k+1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case 4.6. s[i] immediately contains ⟨V⟩.

Step 4.6.1. Perform Step 4.1.1.

Step 4.6.2. If pstat does not have ⟨suppression-type⟩, append ⟨suppression-type⟩ (with no subnode) to pstat.

Note: This handles the case of pictures with all digit-positions suppressible and after ⟨V⟩ (such as 'VZZZ'), for non-zero values. If ⟨suppression-type⟩ were not to be appended to pstat, Case 4.2 would suppress ⟨0⟩ characters in digit-positions after ⟨V⟩, which is not permitted.

Step 4.6.3. i=i+1.

Case 4.7. s[i] immediately contains ⟨insertion-character⟩.

Step 4.7.1.

Case 4.7.1.1. sup is ⟨suppression⟩: ⟨off⟩.

If s[i] has ⟨B⟩, attach ∅ to c[k]; otherwise attach to c[k] the terminal component of s[i].

Case 4.7.1.2. sup is ⟨suppression⟩: ⟨on⟩.

If ⟨suppression-type⟩ has ⟨*⟩, attach ⟨*⟩ to c[k], otherwise, attach ∅ to c[k].

Step 4.7.2. i=i+1; k=k+1.

Case 4.8. s[i] immediately contains ⟨insertion-string⟩,is and no s[n] ($1 \leq n \leq ns$) contains ⟨⟩.

Perform Step 8.

Case 4.9. s[i] immediately contains ⟨credit⟩ or ⟨debit⟩.

Step 4.9.1.

Case 4.9.1.1. sgn = +1.

Attach ∅ to each of c[k] and c[k+1].

Case 4.9.1.2. sgn = −1.

Attach to c[k] and c[k+1] ⟨C⟩ and ⟨R⟩ (respectively) if s[i] has ⟨credit⟩, or ⟨D⟩ and ⟨B⟩ (respectively) if s[i] has ⟨debit⟩.

Step 4.9.2. i=i+1; k=k+2.

Case 4.10. s[i] immediately contains ⟨)⟩.

Step 4.10.1.

Case 4.10.1.1. sgn = +1.

Attach ∅ to c[k].

Case 4.10.1.2. sgn = −1.

Attach ⟨)⟩ to c[k].

Step 4.10.2. i=i+1; k=k+1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 5. If i≤ns then go to Step 4.

Step 6. Return a ⟨character-string-value⟩ containing c[k], k=1,...,nc, in order.

Step 7. Replace sup by ⟨suppression⟩: ⟨off⟩.

Case 7.1. ⟨suppression-type⟩ immediately contains ⟨$⟩, ⟨S⟩, ⟨+⟩, ⟨−⟩, or ⟨(⟩.

Replace the ⟨symbol⟩ contained in c[k−1] by the ⟨symbol⟩ obtained from Figure 26 on page 397 as a function of ⟨suppression-type⟩ and (possibly) sgn.

Case 7.2. ⟨suppression-type⟩ immediately contains ⟨insertion-string⟩,is.

Let n be the number of immediate components in is. Let k = k − n. For m=0,...,n−1, delete any subnodes of c[k+m]. Perform Step 8.

Case 7.3. (Otherwise).

No action.

Step 8. For each ⟨insertion-symbol⟩,sy in is, in order, perform Steps 8.1 and 8.2.

Step 8.1.

Case 8.1.1. sup is ⟨suppression⟩: ⟨off⟩.

Case 8.1.1.1. sy contains ⟨''⟩.

Attach to c[k] a ⟨symbol⟩: ⟨'⟩.

Case 8.1.1.2. sy contains ⟨<<⟩.

Attach to c[k] a ⟨symbol⟩: ⟨<⟩.

Case 8.1.1.3. sy contains ⟨<>⟩.

Attach to c[k] a ⟨symbol⟩: ⟨>⟩.

Case 8.1.1.4. (Otherwise).

Attach to c[k] the terminal component of sy.

Case 8.1.2. sup is ⟨suppression⟩: ⟨on⟩.

Attach ∅ to c[k].

Step 8.2. k = k+1.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

s[i] or ⟨suppression-type⟩	sgn	⟨symbol⟩
⟨S⟩	+1	⟨+⟩
	−1	⟨−⟩
⟨+⟩	+1	⟨+⟩
	−1	∅
⟨−⟩	+1	∅
	−1	⟨−⟩
⟨(⟩	+1	∅
	−1	⟨(⟩
⟨$⟩	+1	*
	−1	*

Figure 26. Table of Terminal Nodes of ⟨symbols⟩ as a Function of ⟨suppression-type⟩ for EDIT-NUMERIC-PICTURE-FIELD.

The positions indicated by * represent a ⟨symbol⟩ which is IMPLEMENTATION-DEFINED. This ⟨symbol⟩ represents a currency symbol.

9.6.2.3 Validity of a Numeric Pictured Value

⟨picture-validity⟩ ::= ⟨picture-valid⟩ | ⟨picture-invalid⟩

⟨picture-valid⟩ ::= [⟨real-value⟩]

The subnode of ⟨picture-valid⟩ is the associated numeric value of v with respect to pic.

This operation is invoked by the operation VALID-BIF, by the operation CONVERT for <pictured-numeric> source-type, and by the operation VALIDATE-INPUT-FORMAT for a <picture-format>.

Operation: VALIDATE-NUMERIC-PICTURED-VALUE(pic,v)

Where: pic is a <data-type> or <picture-format> containing <pictured-numeric>,
v is a ⟨character-string-value⟩ of length equal to the associated character-string length of pic.

Result: a ⟨picture-validity⟩.

Step 1. Perform VALIDATE-FIELD-OF-PICTURED-VALUE(plst,v) to obtain a ⟨picture-validity⟩,pv, where plst is the ⟨picture-element-list⟩ in pic.

Step 2.

Case 2.1. pv has ⟨picture-valid⟩.

Let rv be the ⟨real-value⟩ in pv. Return a ⟨picture-validity⟩: ⟨picture-valid⟩: ⟨real-value⟩: $rv*10^{-q}$;;;, where q is the <scale-factor> in the associated arithmetic data-type of pic.

Case 2.2. pv has ⟨picture-invalid⟩.

Return pv.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.6.2.4 Validity of a Field of a Numeric Pictured Value

Operation: VALIDATE-FIELD-OF-PICTURED-VALUE(plst,v)

Where: plst is a ⟨picture-element-list⟩,
v is a ⟨character-string-value⟩ of length equal to the associated character-string length of plst.

Result: a ⟨picture-validity⟩ (Section 9.6.2.3).

Case 1. w is one of the values obtainable by normal return from performing EDIT-NUMERIC-PICTURE-FIELD(plst,d,sgn) where d is the ⟨character-string-value⟩ containing as many digits as there are digit-positions in plst, each digit independently takes each value 0 through 9 in turn, and sgn takes values +1 and −1 in turn, and w only differs from v in the case of ⟨letter⟩s in corresponding positions.

Let vd and vsign be the unique values of d and sgn which edited to w. Let vx be the ⟨real-value⟩ which is the integer containing the same digits as vd.

Return a ⟨picture-validity⟩: ⟨picture-valid⟩: ⟨real-value⟩: vx*vsign.

Case 2. (Otherwise).

Return a ⟨picture-validity⟩: ⟨picture-invalid⟩.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Appendix A. Features New to This Revision Adopted from X3.53-1976

(This Appendix is not part of AMERICAN NATIONAL STANDARD X3.74-1987, but is included for information only.)

This Appendix describes the features of American National Standard Programming Language PL/I, ANSI X3.53-1976, which have been provided in this standard and were not provided in ANSI X3.74-1981.

Except as noted, these features have been provided compatibly with the corresponding features in ANSI X3.53-1976.

In certain cases, features are restricted to conform to the goals and language of this standard.

A.1 Lexical Constructs

1. The /** /** **/ restriction removed

   /* may be embedded in the body of a comment.

A.2 Attributes and Declarations

1. REFER at end of structure

   A single REFER option at the end of any BASED structure is permitted by this standard.

2. AREA and OFFSET data types

   The AREA and OFFSET data types have been provided. Implicit conversion between OFFSET and POINTER is not permitted except when an OFFSET value is used as a locator qualifier.

3. AUTOMATIC INITIAL

   Variables with the AUTOMATIC storage class may have the INITIAL attribute. INITIAL value lists may consist of scalar expressions and asterisks. Dimensioned AUTOMATIC variables may have a "*" iteration factor (described in item 1 of Section B.2) in any INITIAL attribute.

4. Asterisks in INITIAL value lists

   Asterisks may be used in INITIAL value lists to denote uninitialized variables.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5. Restricted expressions in extent declarations

   Restricted expressions (scalar arithmetic expressions that can be evaluated at translation time) are permitted in declaration of extents in STATIC variables, parameters, and returns-descriptors, as well as INITIAL attributes of STATIC variables.

6. DIMENSION, PARAMETER, NONVARYING, ALIGNED, and UNALIGNED keywords

   These keywords are now recognized and checked for consistency. Only string types may be UNALIGNED and PICTUREs may not be ALIGNED.

7. Contextual declaration of SYSIN/ SYSPRINT

   The standard stream input and output files, SYSIN and SYSPRINT, need not be explicitly declared.

8. Declaration of programmer-named conditions.

   Programmer-named conditions may be declared.

   See "Conditions and Exception Handling", below.

A.3 Program Structure and Control

1. RETURNS(CHARACTER(*))

   RETURNS (CHARACTER(*)) means that the length of the returned character string is dynamically determined by the called procedure.

2. RETURNS(BIT(*))

   RETURNS (BIT(*)) means that the length of the returned bit string is dynamically determined by the called procedure.

3. OPTIONS on BEGIN

   A BEGIN statement may have an OPTIONS clause, specifying implementation defined options for the block.

4. Allow labelled statements after THEN and ELSE

A.4 Storage Control

1. SET optional for ALLOCATE

   An ALLOCATE statement may allocate storage without specifying a SET pointer option.

2. IN option in ALLOCATE and FREE

   An ALLOCATE or FREE statement may have an associated IN option to specify the AREA in which to allocate or free the requested storage.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3. ALLOCATE and FREE of multiple items

   Additional syntax to allow multiple objects to be allocated or freed by a single statement.

4. Allow assignment to target overlapping source

   The target of a string assignment may overlap the source.

A.5 Input/Output

1. Expressions in I/O formats

   Expressions may be used in FORMAT lists for EDIT input and output.

2. Recursive I/O

   Allow, for example, a condition handler to do I/O to a file on which I/O was active when a condition was signalled, or a function invoked by a format expression to do I/O to the file for which the format was being evaluated, with well-defined results.

3. ENVIRONMENT option on record I/O statements

   The ENVIRONMENT option is permitted on CLOSE, DELETE, READ, WRITE, and REWRITE. (This is in addition to the OPEN statement, as permitted by ANSI X3.74-1981).

4. OPEN and CLOSE of multiple files

   Additional syntax to allow multiple files to be opened or closed by a single statement.

5. FROM optional for REWRITE

   The REWRITE statement may be used without a FROM option.

A.6 Expressions and Evaluation

None.

A.7 Pictures

None.

A.8 Built-in Functions and Pseudo-variables

1. EMPTY built-in function

   EMPTY returns an area containing no allocations.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2. ONSOURCE built-in function

   ONSOURCE returns the character-string source value that was being converted when a CONVERSION condition was signalled.

   (Note that the CONVERSION condition is also a new addition to this standard).

3. ONSOURCE pseudo-variable

   This pseudo-variable allows a copy of the character-string source value that caused a CONVERSION condition to be modified so that the conversion may be re-tried.

4. OFFSET and POINTER built-in functions

   OFFSET and POINTER provide for explicit conversion between offsets and pointers.

   This standard restricts the first argument of the OFFSET built-in function to an invocation of the ADDR built-in function.

5. PROD built-in function

   PROD produces the multiplicative product of all elements of an array.

6. SUM built-in function

   SUM produces the sum of all elements of an array.

7. ADD, MULTIPLY, SUBTRACT built-in functions

   These functions provide arithmetic operations with programmer-specified precision and scale.

8. EVERY and SOME built-in functions

   EVERY returns the logical "and" of all elements of a bit string.

   SOME returns the logical "or" of all elements of a bit string.

9. REVERSE built-in function

   REVERSE returns a string that contains the bits or characters of its string argument in reverse order.

10. HIGH and LOW built-in functions

    HIGH and LOW return a string of the last and first characters in an implementation's collating-sequence.

11. ROUND built-in function with FLOAT argument

    ROUND now accepts floating-point arguments.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.9 Conditions and Exception Handling

1. Permit normal return from UNDERFLOW

   "Normal" termination of an UNDERFLOW condition handler is now defined. The computation is continued, using zero as the result of the underflowing operation.

2. Additional conditions

   CONVERSION, SIZE, RECORD, TRANSMIT, FINISH, AREA, and STORAGE conditions have been provided.

3. Programmer-named conditions

   Programmer-named conditions are provided. Programmer-named conditions may be used in SIGNAL, ON, and REVERT statements.

   Note that, since this standard does not allow contextual declarations of any identifiers but SYSIN, SYSPRINT, and built-in functions; programmer-named conditions must be declared as CONDITIONs.

4. Condition-name-commalists

   A list of condition names, separated by commas, may be specified in ON and REVERT statements.

5. Condition prefixes

   Condition prefixes may be applied to PROCEDURE and BEGIN statements, and assert enablement and disablement of individual conditions in that block or procedure.

   Note that a disabled condition is an assertion by the program that the condition will not be signalled, and that a program that causes a disabled condition to be signalled is not a standard-conforming program.

6. SNAP

   This ON statement option results in implementation defined output when the condition handler is entered.

7. ON SYSTEM

   ON condition-name-commalist SYSTEM causes the SYSTEM condition-handler to be invoked for the specified conditions. The behavior of the SYSTEM condition-handler is defined in the standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Appendix B. Features New to This Revision

(This Appendix is not part of AMERICAN NATIONAL STANDARD X3.74-1987, but is included for information only.)

This Appendix describes new language features provided by this standard that are not provided in either ANSI X3.53-1976 or ANSI X3.74-1981.

B.1 Lexical Constructs

1. Lowercase characters

   Lowercase characters are permitted in the language character set, in keywords, in picture specifications, in format items, in constants, in input data, in declared names, and in all other alphanumeric constructs.

   In declared names, the correspondence of lowercase characters and uppercase characters is IMPLEMENTATION-DEFINED. That is, programs intended for interchange must not depend upon whether an implementation considers lowercase and uppercase characters as distinct in declared names.

   In stream output of bit and numeric values, and in conversion of numeric values to character strings, lowercase characters are not generated.

2. Blank after P not required

   P'string' is a legal picture constant. (Previously, only P'string' was permitted.)

3. Oversized floating-point literals

   Floating-point literals are permitted, in program text and stream input, which exceed the capability of the implementation to represent them. Insignificant digits in excess of the implementation's representational capability are ignored.

B.2 Attributes and Declarations

1. The asterisk INITIAL iteration factor

   Asterisk in the INITIAL iteration factor means "initialize the entire aggregate to one (specified) scalar value".

2. The RESERVED attribute

   RESERVED implies STATIC EXTERNAL, but can never cause storage to be allocated or initialized. Declaring a name with the RESERVED attribute implies that the PL/I program must contain at least one declaration of the same name with STATIC EXTERNAL and without RESERVED.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3. The UNION attribute

   The UNION attribute is used to declare a set of MEMBERs, of which it is only valid to access the MEMBER which last had a value assigned to it. This standard thus permits (but does not require) the MEMBERs of a UNION to be allocated to overlapping storage.

4. Named computational constants

   The DEFINE CONSTANT statement provides for definition of constant identifiers (named computational constants), and permits use of such named constants in lexically subsequent declarations and expressions within the defining package, procedure, or block.

   DEFINE CONSTANT statements must appear before any other statements in the containing package, procedure, or block, and the definition of a named computation constant must lexically precede any uses in the defining package, procedure, or block.

5. Extend EXTERNAL attribute

   An optional ('string') may be associated with the EXTERNAL attribute, e.g.

```
DECLARE GKS_OPEN_WORKSTATION ENTRY EXTERNAL ('GKS002');
```

   The effect of this feature is to define, in an implementation defined fashion, the environment name (GKS002 in this case) to be used for the declared object, and the PL/I language name (GKS__OPEN__WORKSTATION in this case) to be used within the declaring package, procedure, or block.

6. EXTERNAL on PROCEDURE statements and ENTRY declarations

   The EXTERNAL attribute may be applied to PROCEDURE statements and ENTRY declarations, so that they can have implementation defined environment names.

B.3 Program Structure and Control

1. The NONRECURSIVE keyword

   The NONRECURSIVE keyword is an assertion that a procedure is not recursive. (This is symmetric with the RECURSIVE keyword, and although NONRECURSIVE is the default if RECURSIVE is not specified, the NONRECURSIVE keyword permits explicit documentation of the properties of nonrecursive procedures.

2. SELECT groups

   SELECT groups, containing WHEN and OTHERWISE clauses, provide case-driven selection of executable code units.

3. DO LOOP

   DO LOOP specifies indefinite iteration.

4. The LEAVE statement

   The LEAVE statement permits exit from the immediately containing DO group, or a specific labelled DO group.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5. The UNTIL clause

   The UNTIL clause provides for repetitive execution of a group until a condition is satisfied. The condition is tested after each execution of the group.

6. Packages

   Packages may be constructed, using the PACKAGE and END statements. A package is a block which may immediately contain only declarations and procedures. The package forms a name scope which is shared by all declarations and procedures contained in the package. By use of the EXPORTS option on the PACKAGE statement, procedures immediately contained in the package can be made visible outside of the name scope formed by the package. Packages thus permit multiple, externally visible procedures to execute within and share the declarative context formed by the package.

   Condition prefixes specified on a PACKAGE statement apply to all procedures contained in the package (unless overridden).

   A package must only be declared at the top level of a program. That is, a package cannot be contained in any other package or procedure. The declarations immediately contained in a package must consist only of constants, and BASED and STATIC variables; AUTOMATIC and DEFINED variables must not be immediately contained in a package, and a package cannot have parameters.

   Only top level procedures (procedures immediately contained in a package) can be exported from a package, and any procedures not explicitly exported are local to the package. All top level procedures contained in a package can be exported by use of the EXPORTS(*) construct.

B.4 Storage Control

None.

B.5 Input/Output

1. L format item

   L format, used in GET EDIT and PUT EDIT format lists, inputs the rest of the current line and skips past the linemark, or outputs a string followed by a linemark.

2. SIZETO option for varying length records

   The SIZETO option may be used with the READ statement to determine the actual length of the transmitted record.

3. The BEGINPAGE condition

   See "Conditions and Exception Handling", below.

4. WRITE to KEYED SEQUENTIAL UPDATE file

   WRITE is allowed to files opened for KEYED SEQUENTIAL UPDATE.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.6 Expressions and Evaluation

1. Short-circuit "and/or" operators

   Short-circuit "and" and "or" operators (spelled &: and |:) provide boolean expressions in which the right operand must not be evaluated if the value of the left operand is sufficient to determine the value of the expression. This permits explicit control of the order in which the operands are evaluated, overriding the usual rule that permits operands to be evaluated in either order.

2. Exclusive-or operator

   The exclusive-or operator is binary ¬.

3. Restricted Expressions

   "Restricted expressions" are a subset of PL/I language expressions, designed to be evaluated at translation time, and allowed in DECLARE and DEFINE CONSTANT statements for the purpose of defining named computational constants and in declaration of extents of STATIC variables, parameters, and returns descriptors, as well as INITIAL attributes of STATIC variables. (In the specification of precision and scale in declarations and built-in functions, only a limited form of restricted expressions — namely, a literal or a reference to a named constant — is permitted.)

   The form of allowable restricted expressions has been extended to be very general. All operators, all enquiry functions except DECLARESIZE, and all built-in functions except DATE, TIME, DATETIME, ONCODE, ONFILE, ONKEY, and ONSOURCE are allowed in restricted expressions. String constants, numeric constants, and named constants may all be operands.

B.7 Pictures

1. Generalized picture insertion/float

   This feature provides generalized specification of insertion and floating characters in pictures. If a picture specification includes (in a manner valid with respect to the syntax of pictures) the quoted characters "<" and ">", then the characters between "<" and ">" will be inserted at that point in the picture value.

   Additional ">" characters following the ">" specify a floating field, and the characters between "<" and ">" will float right to the left of the most significant digit position.

   One expected use of this capability is support of international currency symbols.

2. Parentheses in pictures

   This feature permits a PICTURE pattern to begin with an arbitrary number of left parentheses and to end with a single right parenthesis. The left parentheses are floating characters. If the picture value is negative, it is represented enclosed by a single left and right parenthesis pair.

   This provides a straightforward way to parenthesize negative values on output.

3. EDIT built-in function

   See "Built-in Functions and Pseudo-variables" below.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.8 Built-in Functions and Pseudo-variables

1. TRIM built-in function

   TRIM removes a sequence of characters, belonging to a programmer-specified set, from the beginning and/or end of a string.

2. SEARCH built-in function

   SEARCH returns the location in a target string of the first occurrence of any character contained by a search string. See also item 14, below.

3. CURRENTSIZE built-in function

   CURRENTSIZE returns the amount of storage, in IMPLEMENTATION-DEFINED units, that a generation of storage associated with a variable occupies.

4. EXPONENT built-in function and pseudo-variable

   The EXPONENT built-in function and pseudo-variable permit floating-point data to be scaled efficiently and without loss of accuracy, by providing access to and manipulation of the exponent field of floating-point values. This built-in function and pseudo-variable provide the effect of multiplication and division, by a power of the implementation radix, without the loss of low-order significant digits that an actual multiply or divide operation may cause.

   The effect of EXPONENT is implementation-dependent, and does not require knowledge of exponent bias or representation of zero to correctly utilize.

5. FILEOPEN built-in function

   FILEOPEN takes a file value and returns a boolean value indicating whether or not the file is open.

6. ISOCHAR built-in function

   ISOCHAR provides a standard way to access control characters and other special characters independent of the external character set(s) of an implementation.

   The arguments to ISOCHAR specify the row and column positions, in the ISO character tables, which identify a specific character and its associated semantics. Implementations will translate those semantics to the appropriate, corresponding character(s) for that implementation.

7. MAXLENGTH built-in function

   MAXLENGTH takes a varying string and returns the maximum declared length of the varying string (in contrast to LENGTH, which returns the current length).

8. TALLY built-in function

   TALLY takes two bit or character string arguments, and returns the number of occurrences of the second argument in the first argument.

9. DATETIME built-in function

   DATETIME returns the date and time in an extended format that will continue to work correctly after December 31, 1999.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10. EDIT built-in function

    EDIT takes a numeric or string first argument and a character string second argument; the second argument must be a valid picture pattern. The first argument is converted to a character string according to the picture represented by the second argument. This conversion yields the same character string that would result from assigning the first argument to a picture variable declared with the picture specified by the second argument.

11. IDENTICAL built-in function

    IDENTICAL allows comparison of arbitrary objects. If two objects have identical data descriptions, then if they have identical values, then IDENTICAL returns boolean true, otherwise it returns false. When comparing the values of two aggregates, any fill occupying storage between members of each aggregate are ignored.

12. Permit UNSPEC with aggregate argument

    UNSPEC accepts an array, structure, or union argument.

13. HBOUND/LBOUND/DIMENSION built-in function defaults

    If the second argument, the dimension, is not supplied to HBOUND, LBOUND, or DIMENSION, it defaults to 1.

14. Extend INDEX/VERIFY/SEARCH built-in functions

    INDEX, VERIFY, and SEARCH take an optional third argument: the location within the argument string at which to begin processing.

B.9 Enquiries

A new class of functions, denoted as "Enquiries", has been defined.

These functions are distinguished from built-in functions by their syntax (which utilizes "(:" and ":)", for example "DECLARESIZE(:XYZ:)") and by their arguments (which may be keywords or names without values, whereas the built-in functions require values for arguments.)

A special subclass of these functions is denoted as the "Arithmetic Enquiries". These functions provide information regarding the floating-point environment and model-numbers of an implementation.

1. DECLARESIZE enquiry

   The DECLARESIZE enquiry function returns the amount of storage, in IMPLEMENTATION-DEFINED units, that a BASED variable would occupy if it were allocated when DECLARESIZE is invoked. Use of DECLARESIZE does not require that there actually be a generation of storage associated with the referenced variable when DECLARESIZE is invoked.

2. NULL enquiry

   NULL takes one argument, which must be one of the keywords POINTER, OFFSET, ENTRY, LABEL, or FILE, and returns a null value of the appropriate type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3. EPSILON arithmetic enquiry

   EPSILON returns the interval separating successive values representable by a floating-point data type given as its argument.

4. MAXVAL arithmetic enquiry

   MAXVAL returns the maximum value representable by a floating-point data type given as its argument.

5. MINVAL arithmetic enquiry

   MINVAL returns the minimum value representable by a floating-point data type given as its argument.

6. RADIX arithmetic enquiry

   RADIX returns the actual implementation base corresponding to a floating-point data type given as its argument.

B.10 Conditions and Exception Handling

BEGINPAGE condition

BEGINPAGE is signalled at the beginning of every page of a PRINT file.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Appendix C. Clarifications and Restrictions from X3.74-1981

(This Appendix is not part of AMERICAN NATIONAL STANDARD X3.74-1987, but is included for information only.)

This Appendix identifies several clarifications and restrictions from the syntax and semantics permitted, or apparently permitted, by the earlier version of this standard, i.e., by the combination of ANSI X3.74-1981 and ANSI X3.53-1976.

None of these changes are modifications to add or reduce functionality, or to substitute one functionality for another. None are expected to have any effect on a correct, properly functioning, and well and conservatively written program that conforms to ANSI X3.74-1981. This is not to suggest that specific test cases cannot be written that will identify the differences, only that production programs are exceedingly unlikely to notice them. Rather than changes in function, these changes are motivated by one or more of the following considerations:

(a) In this revision, some of the conformance rules that determine what a user can expect a processor to do have been tightened somewhat. Specific actions, or bounds on values, are now required where the previous definitions permitted greater implementation flexibility. These changes should lead to improved predictability, and hence portability, across implementations. In each case, the values now required, or restrictions imposed, force action upon an implementation that was among the several actions it could have taken under the earlier definition. Consequently, a program that was prepared to deal with the range of behaviors permitted by the previous definitions would correctly respond to all behaviors now permitted.

(b) In a small number of cases, the previous definitions were ambiguous or did not support the expected or required behavior in some particular circumstances, typically under a boundary condition. Where these situations have been detected, unambiguous results and behaviors have now been specified. These additional specifications are all consistent with existing practice in PL/I implementations.

(c) In some cases, the language definition has been modified to permit an implementation to detect an error at translation time. The general approach taken in ANSI X3.53-1976 was to permit such early detection as much as possible. Translation time detection typically permits an implementation to give more information to the user at a more convenient point in program development. Explicit language appears in that standard and this one to specify flexibility for translation time detection. These changes will not change the behavior of any functioning production program into a non-functioning one or vice versa.

(d) In order to permit the maximum number of implementations to conform to the standard, some restrictions have been imposed on alignment, overlaying, and constructions arising out of the extensions discussed in Appendix A and Appendix B.

A list of these changes follows. Each entry is followed by one or more letter codes to indicate which of the rules above applies.

1. The erroneous construction OPEN FILE(x) INPUT OUTPUT is now prohibited by a "must not" qualification. The definition previously required the UNDEFINEDFILE condition to be raised, requiring interpretation-time action. (c)

2. The MOD built-in function has been clarified to not permit a second argument with a value of zero. (b)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3. The values to be returned in unusual or boundary conditions of a few constructions, notably ATAN at (0,x), EVERY with a null string, and GET LIST with blank input, have been specified. (a), (b)

4. The form of the name that follows %INCLUDE has been specified in a fashion consistent with the practice in all known implementations. (a)

5. An implementation is now required to document the arguments to, and behavior of, the ENVIRONMENT option. (a)

6. Implementations are permitted to impose restrictions on the use of unaligned objects and their addresses. (d)

7. If a mathematical built-in function encounters computational or boundary difficulties, it is now required to raise the ERROR condition. Previously, any of a series of conditions could be raised, forcing the programmer to either rely on ERROR or to be prepared to handle all of those conditions, possibly requiring knowledge of the underlying algorithms. (a), (b)

8. Constraints have been placed on the results of computation of REAL values. These constraints require values that were previously permitted, and permit a greater degree of predictability and portability. They do not require any known program to be changed. (a)

9. In conjunction with the changes mentioned in (8), additional constraints have been placed on the circumstances in which an implementation may raise various computational conditions. Under the previous definitions, very broad flexibility was given to implementations in raising conditions at, in principle, arbitrary times and much of this flexibility has been eliminated. A conforming program that functions correctly under the previous definition will function identically under this one; newer programs need not be prepared for as many cases and possibilities. (a)

10. Modifications have been made to the definition's various models of allocation and related relationships to clarify their intent and make them more obviously consistent with common practice in PL/I implementations. (a), (b)

11. The construction SKIP(0) in a PUT statement for a non-print file has been disallowed entirely in conforming programs. This construction has never had a consistent and unambiguous interpretation, nor has there ever been a reason to use it. (a), (b)

12. The number of different types of expressions in DO-statements have been altered to promote portability. DO...REPEAT has been restricted somewhat (string values are now prohibited) and TO and BY have been expanded slightly. The restriction on REPEAT will not affect any program that does not depend on rather unusual conversions formerly permitted; the alterations to TO and BY are extensions that will not affect existing programs. (a)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Appendix D. Features of X3.53-1976 not Adopted in This Standard

(This Appendix is not part of AMERICAN NATIONAL STANDARD X3.74-1987, but is included for information only.)

This Appendix describes the features of American National Standard Programming Language PL/I, ANSI X3.53-1976, which have not been provided in this standard, and changes specific to moving programs written for that standard to this one.

Most of the information required to convert a program conforming to ANSI X3.53-1976 to operate with a processor for the language specified by this standard is available by reading this Appendix in conjunction with Appendix C. However, the reader is cautioned that text in this Appendix applies to ANSI X3.53-1976, and in general does not reflect new features described in Appendix B.

D.1 Program Structure

D.1.1 Syntactic Restrictions

1. A statement must not have multiple label prefixes.
2. A DECLARE statement must not have a label prefix.
3. A label prefix must not have more than one subscript. Label prefixes appearing on PROCEDURE or FORMAT statements must not be subscripted.
4. Condition prefixes must not appear except on BEGIN and PROCEDURE statements.
5. END statements must not cause multiple group or block closure. Any closure label must match the label prefix on the corresponding DO, SELECT, BEGIN, or PROCEDURE statement.
6. Arithmetic constants must not contain the default-suppressing characters B, F, I, or P.

D.1.2 Semantic Restrictions

All functions must yield scalar values.

D.1.3 Features Excluded

1. The DEFAULT statement.
2. The ENTRY statement.
3. Binary constants and replicated string constants.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.2 Program Control

D.2.1 Syntactic Restrictions

DO statements must not contain commalists.

D.2.2 Semantic Restrictions

1. The control variable of a DO statement must be of an arithmetic (including pictured) or pointer type. The values of the TO, BY, and REPEAT options must be arithmetic (including pictured) or pointer values.

2. The expression in an IF statement or a WHILE clause or an UNTIL clause must have a bit-string value of length one.

3. An array whose storage is unconnected must not be passed as an argument.

4. A RETURN statement must not be used within an on-unit.

D.2.3 Features Excluded

Debugging conditions: NAME, STRINGRANGE, STRINGSIZE, SUBSCRIPTRANGE.

D.3 Storage Control Statements

D.3.1 Syntactic Restrictions

An assignment statement may have only one target.

D.3.2 Semantic Restrictions

None.

D.3.3 Features Excluded

BY NAME assignment.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.4 Input/Output

D.4.1 Syntactic Restrictions

1. GET and PUT statements may contain only one I/O list and at most one format list.
2. The E and F format items must not contain a third value.
3. The INTO and FROM options of the record I/O statements must not reference bit-aligned variables.
4. File constants must not be dimensioned.

D.4.2 Semantic Restrictions

None.

D.4.3 Features Excluded

1. The LOCATE statement.
2. The TAB option of the OPEN statement.
3. The DATA option of the GET and PUT statements.
4. The COPY option of the GET statement.
5. The IGNORE option of the READ statement.
6. The C format item.

D.5 Attributes and Pictures

D.5.1 Syntactic Restrictions

1. Pictures are restricted to a large subset of fixed-point decimal pictures.
2. The following attributes are supported, but are not keywords in the language: CONSTANT, MEMBER, PRECISION, REAL, and STRUCTURE.
3. Only string types may be UNALIGNED and PICTUREs may not be ALIGNED.
4. The INITIAL attribute for STATIC variables must not be a LABEL CONSTANT, ENTRY CONSTANT, or FILE CONSTANT.
5. The INITIAL attribute for a dimensioned AUTOMATIC item must have a "*" iteration factor.
6. A negative scale factor must not be specified. A nonzero scale factor may be specified only for FIXED DECIMAL variables.
7. Format constants are supported only in the form of statement names on FORMAT statements.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.5.2 Semantic Restrictions

1. The data type of each name used in a program, except SYSIN and SYSPRINT when used in a FILE option or named I/O condition, must be specified either in a DECLARE or a DEFINE CONSTANT statement or else by use of the name as a label prefix or a built-in function. The data type must be given for each parameter in an ENTRY attribute, as well as for the return value in the RETURNS option of the PROCEDURE statement, and for the return value in the RETURNS attribute.

2. The reference given in a DEFINED attribute must identify a variable of identical type and extents as the defined variable, or else both the defined variable and the base variable must be suitable for string overlay defining.

3. If a declaration contains a REFER option, then it can have only one, which must be located at the end of the structure, and the target of which must contain FIXED and BINARY.

D.5.3 Features Excluded

1. Attributes: COMPLEX, CONTROLLED, GENERIC, LIKE, LOCAL, and POSITION.

2. Picture characters: A E I K R T X Y.

3. BIT VARYING.

4. iSUB defining.

5. FORMAT variables.

D.6 Built-in Functions and Pseudo-variables

D.6.1 Syntactic Restrictions

1. The MAX and MIN built-in functions must have exactly two arguments.

2. Pseudo-variables may appear only on the left side of an assignment statement.

3. All arguments specifying precision, scale, or number-of-digits to built-in functions must be either an integer or a reference to a named constant.

D.6.2 Semantic Restrictions

1. Various built-in functions have restrictions on their arguments either to prevent the creation of an unsupported data type or to permit the compile-time evaluation of the function.

2. Built-in functions must not yield aggregate results.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.6.3 Features Excluded

1. Built-in functions: AFTER, ALLOCATION, BEFORE, COMPLEX, CONJG, DECAT, DOT, ERF, ERFC, IMAG, ONCHAR, ONFIELD, ONLOC, PRECISION, and REAL.

2. Pseudo-variables: IMAG, ONCHAR, and REAL.

D.7 Expressions

D.7.1 Syntactic Restrictions

None.

D.7.2 Semantic Restrictions

1. All operators must yield scalar values.

2. No expression can yield a fixed-point decimal result with a negative scale factor or a scaled fixed-point binary result.

3. Subscripts and extents must evaluate to integer values.

4. Arithmetic operators and built-in functions must have arithmetic or pictured operands.

5. Bit-string operators and built-in functions must have bit-string operands.

6. Character-string operators and built-in functions must have character-string operands.

7. The operands of the relational operators must both be arithmetic (including pictured) values, both be character string values, both be bit string values, or must both have the same data type.

8. Expressions in format iteration factors and format list items must yield integer values.

9. A pointer value must not be assigned to an OFFSET variable, passed to an OFFSET PARAMETER, or returned from a function with the RETURNS(OFFSET) attribute. The first argument of the OFFSET built-in function must be an invocation of the ADDR built-in function. It may be used in this restricted form to convert pointer values to offset values.

10. An offset value must not be assigned to a POINTER variable, passed to a POINTER PARAMETER, or returned from a function with the RETURNS(POINTER) attribute. (The POINTER built-in function will convert offset values to pointer values).

11. The only allowed case of aggregate promotion is an assignment statement of the form "reference = expression", where reference denotes a connected array of scalars and expression is a scalar expression.

12. In an assignment statement, "reference1 = reference2" where both references are aggregates, the data descriptions of the two references must be identical and both references must denote connected storage.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.7.3 Features Excluded

None.

D.8 Change in Intermediate Results

When the derived scale of an arithmetic result is FIXED, the derived base has been changed from BINARY to DECIMAL when fractional (i.e., scale factor greater than zero) decimal values are combined with integral (i.e., scale factor equal to zero) binary values. This change provides improved consistency and is unlikely to affect actual results.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Appendix E. Incompatible Changes from X3.74-1981

(This Appendix is not part of AMERICAN NATIONAL STANDARD X3.74-1987, but is included for information only.)

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Appendix F. Rationale for Major Design Decisions

(This Appendix is not part of AMERICAN NATIONAL STANDARD X3.74-1987, but is included for information only.)

There are many costs attendant upon each feature included in this standard: costs to design, document, learn, remember, compile, execute, and maintain the feature. These costs were weighed against the liabilities of not including the feature: popularity of usage, difficulty of programming with alternative constructs, inefficiency of alternative constructs, and loss of readability due to the use of alternative constructs.

The content of this standard derives from American National Standard Programming Language PL/I (ANSI X3.53-1976), especially those portions included in ANSI X3.74-1981, and a series of proposals and suggestions for "new" language features. "New", in this case, refers to any feature not specified in ANSI X3.53-1976.

Of the features of ANSI X3.53-1976, some were easy to exclude because they were common sources of programming errors, or because the cost to implement or use them was high in relation to their utility. Other features were more difficult to evaluate, particularly those whose cost and utility were high.

Many entirely new features (relative to ANSI X3.53-1976) were considered for inclusion in this standard. A large fraction of them were proposed, at least informally, during the final review before adoption of ANSI X3.53-1976 and deferred to see how, or if, they would be implemented and used.

Proposals were viewed favorably if they met one or more of the following criteria:

1. The functionality had already been incorporated into existing implementations and had been found to be useful and successful.

2. Similar functionality was provided in several existing implementations using different mechanisms and agreement could be reached that the functionality was important and that a single mechanism was, after experience, most appropriate.

3. There was clear user demand for the proposed modifications and they were obvious and compatible extensions to well-established facilities.

By contrast, the following criteria caused a proposal to be looked on with disfavor:

1. The proposal would have introduced a new incompatibility, either with ANSI X3.53-1976 or with ANSI X3.74-1981, or with important non-conforming implementations of PL/I. It was an important goal that the revised standard specify a language that, insofar as possible, would permit programs written in it to be portable among all sensible implementations of PL/I.

2. The proposal would have introduced an alternate mechanism for performing some function where the functionality already existed in the language or would have introduced little new functionality at high implementation cost or with a very high degree of complexity.

3. The proposed feature could not have been adequately described in implementation independent ways without unreasonably constraining hardware, software, or operating systems.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4. The proposal represented a very large change or a change that would alter the basic character of PL/I into that of another language.

5. The proposed feature would have been completely new to PL/I and would have represented standardization prior to any implementation experience.

Several proposals, of course, met both positive and negative criteria and, in these cases, the committee weighed both sets of factors in its decisions, paying special attention to existing practice.

The major decisions to exclude features of ANSI X3.53-1976 made during the development of this standard and the rationale for each decision are discussed here so that the reader will have some understanding of why various features were included in, or excluded from, this standard.

1. Because the number of distinct data types in a language has a multiplicative effect on the number of code patterns and library routines needed to support execution of programs, the number of distinct scalar data types is limited to approximately one half of those provided by ANSI X3.53-1976, by excluding little-used data types. Over half of the excluded data types are forms of COMPLEX data and little-used forms of PICTURE data.

2. Although array-valued expressions were recognized to be of high utility in scientific applications, their cost was considered to be excessive for inclusion in this standard. Structure-valued expressions and mixed-aggregate-type expressions were considered to be much less useful than were array-valued expressions. These expressions have also been found to cause difficulties in program interchange.

3. Data-directed stream I/O is excluded in order to reduce the amount of execution-time code required to support this standard. This restriction also simplify compilers and consequently reduce the cost to implement and use them.

4. DEFAULT, ENTRY, and LOCATE statements were excluded because their frequency of usage did not justify their cost either in terms of compiler complexity or additional language complexity. The DEFAULT statement has also been found to inhibit, rather than enhance, program clarity and interchange. The addition of the PACKAGE construction to the language further reduces the requirement for ENTRY.

5. Explicit declaration of all user-defined names either by means of a DECLARE statement or by means of a label prefix is required in order to reduce programming errors and to improve the readability of programs. This requirement also reduces the complexity of compilers.

6. Implicit conversion from one data type to another is limited to those contexts in which such conversions are likely to produce the desired result. This limitation eliminates a very common source of programming errors.

7. The conditions that can be explicitly designated in this standard are limited. This reduces implementation complexity while retaining most of the power of the exception handling provided by ANSI X3.53-1976.

8. The DO statement was designed to make it easy to learn and easy to use without introducing programming errors.

9. All storage classes except CONTROLLED are included, and variable extents are permitted where permitted by ANSI X3.53-1976. CONTROLLED is excluded because it is largely redundant with BASED. Variable extents are permitted because no other language feature provides a suitable alternative.

10. REFER options are included in a single, simplified form. While REFER is needed for some important programming applications, its full generality implies implementation cost and user confusion that outweighs the advantage of that generality.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11. The DEFINED attribute is included for simple and string overlay defining. For simple defining, all extents are required to match. POSITION, iSUBs, and *-subscripts in the reference to the base variable are excluded because of excessive complexity for both users and implementers.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Index

Special Characters

‹'› 273, 277, 279, 282, 283, 347, 396
‹;› 79, 86, 91, 273
‹+› 393, 394, 396, 397
‹.› 277, 390
‹<› 145, 396
‹<<› 145, 396
‹<>› 145, 396
‹(› 79, 145, 393, 394, 396, 397
‹|› 51, 57, 58, 59, 145, 273
‹||› 24, 51, 57
‹|:› 51, 57
‹$› 393, 394, 396, 397
‹$-list› 146
‹*› 77, 93, 105, 124, 129, 393, 394, 395
‹*-list› 57, 146
‹*/› 77
‹)› 79, 145, 395
‹/› 77
‹/*› 77
‹,› 24, 27, 29, 33, 87, 88, 99, 122, 137, 138, 272, 273
‹>› 145, 146, 394, 395, 396
‹:› 86, 98
‹=› 273
‹-› 390, 393, 394, 396, 397

A

A 51, 61
abs-bif 335, 306
<abs-bif> 72
<absent>
24, 31, 33, 84, 89, 90, 91, 92, 94, 100, 101, 104, 106, 108, 110, 126, 133, 135, 140, 141, 142, 143, 149, 172, 193, 195, 207, 210, 211, 212, 214, 215, 216, 249, 251, 256, 259, 260, 262, 265, 266, 271, 272, 274, 277, 278, 282, 283
284, 289, 337, 339, 341, 346, 351, 354, 356, 367, 371, 376, 377, 380
abstract form 4
abstract syntax 4
abstract-block 75
abstract-block-component 75
abstract-block-contains 75
acos-bif 335
<acos-bif> 72
activate-begin-block 174, 175, 177
activate-procedure 172, 167, 171, 175, 184, 196, 313
active [operation] 36
ADD 9, 402
<add> 71, 132, 319
add-bif 336, 9, 305, 306
<add-bif> 72
ADDR 15, 402, 419
addr-bif 336
<addr-bif> 72, 362
advance-controlled-group 180, 178, 190, 293, 306
advance-execution 176, 13, 175, 176, 187, 268
AFTER 419
aggregate 299
<aggregate-type> 165
114, 132, 133, 156, 165, 181, 186, 202, 205, 210, 211, 216, 222, 223, 226, 227, 228, 229, 230, 231, 246, 271, 275, 276, 299, 300, 301, 302, 315, 316, 317, 318, 319, 320, 321, 322, 326, 327, 328, 329, 330, 331, 332, 335, 336, 337, 338, 339, 340
341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380
<aggregate-value> 165
32, 155, 156, 163, 165, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 189, 191, 202, 205, 207, 211, 214, 215, 219, 220, 221, 222, 223, 227, 228, 229, 231, 234, 244, 246, 250, 253, 256, 257, 265, 270, 271, 275, 276, 281, 293, 297, 300, 301
302, 313, 314, 315, 316, 317, 319, 321, 330, 333, 336, 341, 344, 345, 346, 348, 349, 351, 352, 353, 355, 356, 358, 359, 362, 363, 364, 365, 370, 371, 373, 376, 379, 382, 383
ALIGNED 15, 45, 74, 88, 89, 95, 97, 123, 400, 417
<aligned> 65, 74, 123, 233
<alignment> 65, 64, 65, 74, 123, 139, 233
‹alignment› 95, 95
ALL 43, 109
ALLOC 62
ALLOCATE 206, 3, 13, 48, 62, 168, 174, 186, 199, 205, 257, 400, 401
allocate-based-storage 204, 203
<allocate-statement> 69, 66, 69, 199, 203
‹allocate-statement› 48, 43, 48
allocate-static-storage-and-build-static-directory 168, 167, 199
<allocated-buffer> 164, 163, 164, 218, 248, 255, 256, 257, 259, 260, 261, 263, 268
<allocated-storage> 164, 161, 164, 167, 199, 201, 202, 205, 206, 207, 217, 219, 223, 231, 234, 247
ALLOCATION 419
<allocation> 69, 31, 69, 114, 203, 204, 206
‹allocation› 48, 48, 103, 114
‹allocation-commalist› 48
<allocation-list> 69, 203
<allocation-unit> 164, 17, 18, 164, 199, 200, 202, 203, 206, 207, 209, 210, 217, 218, 219, 223, 225, 231, 232, 234, 238, 294, 324, 383, 384, 391
<allocation-unit-designator> 165, 200, 202, 207, 211, 216, 217, 219, 223, 225, 227, 229, 231, 232, 234, 235, 238, 239, 240, 241, 242, 294, 295, 324, 383
<allocation-unit-list> 164, 199, 217, 219, 231, 234
<alpha> 162, 165, 166, 247, 248, 253, 260, 262, 267, 270, 272, 278, 292

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

AND 156
<and> 71, 132
<and-logical> 16, 71, 132
ANY 43
append [instruction] 33
apply-attribute 98, 96, 97
apply-constraints 156, 152
apply-defaults 96, 93, 97, 98, 100, 148
apply-subscripts 138, 135
AREA 4, 19, 45, 48, 95, 96, 97, 102, 112, 124, 399, 400, 403
<area> 66, 65, 66, 109, 124, 205, 210, 223, 224, 322, 329, 347, 362, 365
≪area-action≫ 164, 164, 207, 210, 217, 225, 326, 384, 391
≪area-action-list≫ 164, 199, 207, 210, 217, 225, 325, 326, 383, 384, 391
≪area-allocation≫ 164, 164, 199, 206, 207, 217, 325, 383, 391
<area-condition> 68, 163, 203, 205, 223
area-conversion 224, 223
≪area-freeing≫ 164, 164, 217, 325
<area-size> 66, 66, 114, 139, 144, 153, 154, 186, 208, 210, 223, 224, 242, 324, 325
≪area-size≫ 45, 45, 124
≪area-size≫ 164, 164, 210, 225, 326
≪area-value≫ 164, 19, 164, 199, 207, 210, 217, 223, 224, 225, 231, 234, 326, 347, 383, 391
<argument> 72, 72, 141, 142, 143, 185, 186, 220, 226, 228, 229, 230, 302, 313, 333, 334, 336, 344, 345, 351, 352, 355, 358, 359, 360, 362, 365, 366, 371, 372, 379
<argument-list> 72, 73, 140, 141, 142, 143, 155, 185, 220
≪arguments≫ 52, 52, 89, 135, 136, 137, 140
≪arguments-list≫ 52, 92, 134, 138
<arithmetic> 65
35, 65, 66, 73, 108, 109, 125, 129, 134, 140, 147, 148, 151, 152, 155, 177, 178, 179, 226, 245, 302, 303, 304, 305, 317, 318, 319, 320, 321, 322, 323, 326, 327, 328, 330, 331, 332, 335, 336, 337, 338, 340, 342, 343, 346, 348, 349, 350, 357
357, 358, 360, 361, 365, 368, 369, 370, 372, 374, 375, 378, 382, 384, 386, 388, 390, 391
≪arithmetic≫ 95, 95
≪arithmetic-constant≫ 57, 46, 52, 55, 56, 57, 79, 148, 389, 390
≪arithmetic-enquiry-description≫ 53, 53, 93
arithmetic-result 305, 35, 181, 305, 306, 317, 320, 322, 328, 331, 332, 335, 336, 337, 338, 341, 343, 347, 348, 351, 357, 358, 361, 362, 365, 367, 369, 370, 372, 373, 374, 375, 379
<array-init> 65, 65, 126, 127, 153, 154, 214
≪array-init≫ 46, 46, 103, 126, 127
asin-bif 337
≪asin-bif≫ 72
assign 222, 180, 181, 205, 211, 214, 216, 219, 220, 232, 257, 258, 271, 275, 276, 281
assign-exponent-pv 227
See also assign-f
assign-f 226, 222
See also assign-exponent-pv
See also assign-onsource-pv
See also assign-pageno-pv
See also assign-substr-pv
See also assign-unspec-pv
assign-onsource-pv 228
See also assign-f
assign-pageno-pv 228
See also assign-f
assign-substr-pv 229
See also assign-f
assign-unspec-pv 231
See also assign-f
<assignment-statement> 69, 66, 69, 113, 199, 219
≪assignment-statement≫ 48, 43, 48, 103, 113
associated 299
associated arithmetic data-type 391
associated character-string length 391
<asterisk>
27, 64, 65, 66, 72, 103, 122, 124, 127, 128, 138, 139, 140, 147, 148, 153, 154, 186, 200, 208, 214, 215, 217, 222, 240, 242, 246, 264, 270, 271, 275, 276, 282, 285, 299, 300, 301, 320, 323, 329, 332, 344, 345, 351, 355, 383, 384
ATAN 414
atan-bif 337, 338
<atan-bif> 72
atand-bif 338
<atand-bif> 72
atanh-bif 338
<atanh-bif> 72
attach [instruction] 33
≪attribute≫ 45, 45, 80, 82, 83, 84, 85, 86, 91, 92, 93, 94, 96, 98, 99, 100, 101, 102, 104, 105, 108, 120, 121
attribute-consistency [syntax] 95
≪attribute-list≫ 44, 82, 86, 87, 91, 92, 93, 94, 98, 100, 101, 102
AUTO 62
AUTOMATIC 45, 62, 95, 97, 101, 102, 121, 399, 407, 417
<automatic> 12, 13, 64, 121, 152, 153, 154, 174, 209, 232
≪automatic-directory≫ 161, 161, 173, 174, 209, 218
≪automatic-directory-entry≫ 161, 13, 161, 174, 175, 209
≪automatic-directory-entry-list≫ 13, 161, 174

B

B 61, 81, 415
≪B≫ 395
≪B-list≫ 277, 389
≪balanced-unit≫ 43, 43, 54, 80, 116, 117
<base> 65
19, 35, 65, 74, 125, 129, 130, 147, 148, 178, 179, 303, 304, 305, 317, 318, 319, 320, 321, 322, 323, 328, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 345, 346, 348, 350, 351, 357, 358, 360, 361, 362, 365

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

367, 368, 369, 370, 372, 374, 375, 378, 384, 385, 386, 390, 391
<base-item> 64, 64, 107, 121, 154, 241, 242
BASED 16, 45, 95, 102, 121, 314, 399, 407, 410, 424
<based> 64, 64, 107, 114, 121, 136, 149, 152, 204, 205, 216, 218, 231, 234
basic-bit-value 279, 271, 277
basic-character-value 279, 271
basic-numeric-value 390, 277, 385, 389
{basic-reference} 52, 52, 55, 56, 89, 91, 92, 101, 135, 137
≪basic-value≫ 164
18, 34, 66, 73, 108, 128, 129, 141, 147, 148, 156, 160, 163, 164, 178, 179, 180, 181, 182, 183, 184, 199, 201, 202, 203, 205, 207, 211, 214, 215, 216, 219, 221, 223, 226, 227, 228, 231, 233, 234, 238, 243, 245, 253, 257, 264, 270, 271, 275 276, 282, 285, 286, 293, 294, 302, 314, 315, 316, 323, 326, 353, 359, 382, 384
≪basic-value-index≫ 165, 165, 199, 202, 207, 212, 223, 224, 226, 237, 326
≪basic-value-index-list≫ 165, 212
≪basic-value-list≫ 164, 165, 181, 184, 186, 199, 200, 202, 204, 205, 209, 211, 216, 222, 223, 224, 225, 227, 231, 232, 238, 243, 244, 246, 257, 265, 271, 275, 276, 300, 302, 315, 316, 324, 326, 359
BEFORE 419
BEGIN 3, 47, 400, 403, 415
<begin-block> 67, 66, 67, 75, 106, 158, 166, 172, 173, 174, 175, 177, 188, 190, 192, 244, 245
{begin-block} 43, 42, 43, 44, 75, 80, 83, 84, 85, 89, 103, 106, 110, 116, 117, 118
{begin-statement} 47, 43, 47, 106
BEGINPAGE 48, 407, 411
<beginpage-condition> 68, 197, 249, 288, 290, 291
BIN 62
BINARY 16, 45, 62, 95, 96, 97, 125, 129, 148, 418, 420
<binary> 35, 65, 125, 129, 130, 153, 179, 302, 303, 321, 322, 334, 339, 367, 384, 385, 386, 387, 389
binary-bif 339
<binary-bif> 72
BIT 4, 15, 16, 45, 93, 95, 96, 97, 102, 124, 128, 400, 418
<bit> 32, 65, 109, 110, 124, 128, 148, 184, 202, 210, 217, 228, 230, 231, 232, 233, 235, 242, 271, 276, 282, 285, 303, 304, 305, 318, 320, 322, 326, 327, 328, 329, 330, 332, 339, 340, 348, 349, 352, 353, 370, 371, 373, 379, 380, 383, 386, 388, 389
bit-bif 339
<bit-bif> 72
<bit-format> 71, 71, 277, 285
{bit-format} 51, 51
≪bit-string-value≫ 164, 32, 147, 148, 164, 186, 203, 210, 224, 230, 236, 271, 276, 277, 279, 285, 286, 304, 305, 318, 320, 321, 322, 323, 326, 327, 328, 329, 330, 332, 348, 349, 353, 370, 379, 380, 381, 382, 383, 386, 387, 388, 389
≪bit-value≫ 164, 32, 148, 149, 164, 184, 217, 226, 279, 285, 305, 318, 320, 321, 323, 326, 329, 332, 340, 348, 354, 366, 370, 372, 373, 380, 383, 386, 388, 389
≪bit-value-list≫ 164, 210, 224, 233, 285, 304, 380
≪blanks≫ 389, 389, 390
block 166
block-component 75
block-contains 75
≪block-control≫ 162, 38, 161, 162, 173, 174, 182, 192, 211, 268, 270, 273, 275, 281, 282, 283, 293, 294, 297, 298
≪block-directory≫ 161, 161, 173, 174
≪block-environment≫ 163, 161, 163, 172, 173, 174, 211
≪block-state≫ 161, 13, 18, 36, 38, 161, 166, 171, 172, 173, 174, 175, 187, 189, 191, 192, 194, 195, 196, 209, 211, 218, 219, 231, 241, 244, 245, 294, 297, 313, 324, 334
≪block-state-designator≫ 162, 163, 164, 172, 174, 187, 191, 194, 211, 244, 245, 297, 324
≪block-state-list≫ 36, 38, 161, 166, 172, 173, 174, 175, 187, 189, 196, 297, 324, 334
bool-bif 340
<bool-bif> 72
<bound-pair> 64, 27, 27, 64, 66, 120, 122, 134, 135, 138, 139, 140, 144, 153, 154, 158, 200, 208, 217, 222, 238, 240, 242, 243, 244, 246, 300, 301, 346, 351, 356
{bound-pair} 44, 44, 86, 121
{bound-pair-commalist} 44, 86, 121
<bound-pair-list> 64, 120, 121, 122, 134, 140, 165, 200, 213, 223, 238, 239, 240, 243, 246, 295, 299, 346, 351, 356
brackets 30
build-fdi 168, 167
build-file-directory-and-informations 167, 167
BUILTIN 45, 91, 92, 95, 96, 101, 102, 119
<builtin> 64, 119, 135
<builtin-function> 72, 72, 141, 142, 150, 332, 333, 335, 362
<builtin-function-reference> 72, 72, 135, 139, 141, 150, 154, 155, 185, 305, 313, 332, 333, 362
bullet 30
BY 47, 414, 416
<by-option> 67, 67, 177, 179, 181
{by-option} 47, 47
≪by-value≫ 162, 162, 166, 179, 180, 181

C

C 61, 417
{C} 395
CALL 3, 48
<call-statement> 68, 66, 68, 172, 184
{call-statement} 48, 43, 48, 91
≪carriage-return≫ 166, 247, 252, 289, 291, 292
case 31
<cat> 71, 132, 158
category-names 23
ceil-bif 341, 306
<ceil-bif> 72
CHAR 62
CHARACTER 4, 16, 45, 62, 93, 95, 96, 97, 102, 124, 128, 400
<character>

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

65, 109, 124, 128, 147, 202, 210, 217, 227, 228, 232, 233, 234, 235, 242, 253, 256, 264, 270, 271, 275, 276, 281, 282, 285, 303, 304, 305, 322, 326, 327, 329, 341, 344, 347, 352, 353, 355, 358, 363, 364, 367, 371, 373, 375, 376, 377, 380, 386 388, 389
character-bif 341
<character-bif> 72
<character-format> 71, 71, 276, 285
{character-format} 51, 51
≪character-string-value≫ 164 147, 162, 163, 164, 165, 186, 192, 193, 196, 203, 210, 224, 230, 236, 249, 252, 253, 254, 256, 259, 264, 270, 271, 272, 275, 276, 277, 278, 279, 281, 282, 285, 286, 287, 289, 304, 305, 323, 326, 334, 341, 342, 345, 352, 355, 358, 363, 364, 375 377, 378, 381, 382, 383, 386, 387, 388, 389, 391, 392, 393, 396, 397, 398
≪character-value≫ 164, 164, 217, 226, 253, 254, 275, 278, 279, 289, 305, 321, 323, 326, 344, 345, 352, 354, 358, 366, 368, 372, 373, 375, 376, 377, 378, 381, 383, 386, 388, 392, 393
≪character-value-list≫ 164, 210, 224, 233, 253, 254, 289, 304
check-arithmetic-type 130, 107, 129, 148
check-attribute-completeness-and-delete-attributes 102, 93, 100, 102
check-based-reference 235, 15, 234
check-simply-defined-reference 242, 241
chopped-result 306, 306, 385
CLOSE 254, 3, 17, 49, 169, 254, 401
<close-statement> 69, 66, 69, 254
{close-statement} 49, 43, 49
≪closed≫ 163, 168, 254, 256, 258, 260, 262, 269, 280
<cn> 103
{cn} 103
{cn-commalist} 103
<cn-list> 103
{cn-list} 103
COL 62
collate-bif 341, 323, 352, 358, 376
<collate-bif> 72, 333
collect-subscripts 137, 135
COLUMN 51, 62
<column-format> 71, 71, 274, 284, 292
{column-format} 51, 51
comment 197, 197
{comment} 57, 57, 58, 77, 78
{comment-body} 57, 57
{comment-body-list} 57
{comment-character} 57, 57
common aggregate type 300
compare 323, 183, 322, 326, 327, 328, 329, 353
{comparison-operator} 51, 51
≪comparison-result≫ 322, 183, 322, 323, 353
compatible 300
complete tree 26
complete-attribute-implications 82, 81
complete-constructed-entry-declaration 98, 93
complete-declarations 93, 81
complete-external-unit 80, 75
≪complete-file-description≫ 163, 163, 228, 252
complete-options 81, 8, 81
complete-structure-declarations 87, 80
COMPLEX 418, 419, 424
component 21, 10
composite-result 307, 306
<computational-condition> 68, 68, 112, 163, 192, 193
{computational-condition} 44, 44, 48, 112
<computational-type> 65, 35, 65, 74, 108, 113, 114, 124, 125, 129, 131, 140, 144, 148, 178, 179, 223, 230, 271, 303, 304, 321, 339, 341, 342, 345, 347, 350, 352, 358, 371, 381
{computational-type} 95, 95
concatenate 321, 287, 321, 342, 371, 387, 388
concrete form 4
concrete syntax 4
concrete-block 75
concrete-block-component 75
concrete-block-contains 75
concrete-representation 24
COND 62
CONDITION 45, 48, 62, 95, 96, 97, 120, 403
<condition> 64, 120, 134
≪condition-bif-value≫ 163, 163, 193, 195, 196, 219, 249, 334
≪condition-bif-value-list≫ 161, 172, 173, 191, 193, 195, 196, 249, 386, 389
<condition-name> 68, 68, 112, 113, 118, 191, 193, 194, 195, 196
{condition-name} 48, 48, 103, 112, 113, 118
{condition-name-commalist} 44, 48, 118
<condition-name-list> 68, 118, 194, 195
<condition-prefix> 68, 68, 112, 192
{condition-prefix} 44, 44
{condition-prefix-commalist} 44
<condition-prefix-list> 63, 67, 105, 106, 112, 192
<conditional-group> 67, 67, 108, 177, 190
conditions-in-arithmetic-expression 306, 306, 335, 337, 338, 341, 343, 348, 351, 357, 358, 361, 365, 367, 369, 370, 373, 374, 375, 379
CONJG 419
{consistent-declaration} 95, 95, 102
{consistent-description} 96, 94, 96, 102
{consistent-literal-constant} 95, 95
CONSTANT 3, 44, 45, 80, 86, 92, 95, 96, 97, 101, 102, 119, 148, 406, 408, 417, 418
<constant> 73, 71, 73, 108, 113, 129, 131, 132, 141, 144, 147, 148, 154, 155, 156, 159, 160, 185, 314, 330, 334, 345, 346, 351, 355, 366
{constant} 52, 52, 55, 103, 132, 147
constraint 156, 156, 26, 63
constraint-expression 156, 156
construct-contextual-declarations 91, 80, 81, 136
construct-explicit-declarations 83, 80
construct-record 265, 259, 261
construct-statement-name-declarations 85, 83, 85, 93, 98, 101, 104, 105
construct-union-list 211, 210, 211, 212
contain 21, 10
<control-format> 71, 71, 274, 284, 297
{control-format} 51, 51

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

≪control-state≫ 39, 36, 38, 39
CONTROLLED 418, 424
<controlled-group> 67, 67, 108, 177, 178, 188, 190
≪controlled-group-state≫ 162, 162, 166, 178, 179, 180, 181, 188, 190
≪controlled-group-state-list≫ 162, 166, 178, 188
CONV 62
CONVERSION 44, 62, 402, 403
<conversion-condition> 68, 192, 193, 249, 271, 276, 386, 389
convert 382, 34, 35, 61, 129, 148, 160, 179, 180, 181, 223, 228, 230, 234, 253, 264, 270, 282, 285, 286, 287, 302, 320, 323, 329, 332, 347, 365, 381, 382, 384, 386, 387, 397
convert-named-literal 128, 108
convert-to-arithmetic 389, 286, 382, 385
convert-to-bit 386, 383
convert-to-character 387, 61, 383
convert-to-fixed 384, 382
convert-to-float 385, 382
convert-to-float-decimal 388, 287, 382, 387
convert-to-logical-levels 88, 87
converted number-of-digits 303
converted precision 303
converted scale-factor 303
≪converted-by-type≫ 162, 162, 178, 180, 181
≪converted-to-type≫ 162, 162, 178, 179, 180
COPY 417
copy [of a tree] 25
copy-bif 342
<copy-bif> 72
correspondence [of data-types] 300
correspondence [of letters] 59
correspondence [of nodes] 22
correspondence [of scalar-elements] 300
correspondence [of union-data-descriptions] 301
corresponding block 166
COS 54
cos-bif 342
<cos-bif> 72
cosd-bif 343
<cosd-bif> 72
cosh-bif 343
<cosh-bif> 72
create-abstract-equivalent-tree 103, 104, 107, 108, 110, 111, 114, 115, 116, 118, 119, 126, 127, 128, 130, 131, 134
create-allocation 114
    See also create-f
create-argument-list 140, 135, 136
create-arithmetic-enquiry-reference 151, 150, 151
create-array-init 127
    See also create-f
create-assignment-statement 113
    See also create-f
create-balanced-unit 116, 116, 117
create-begin-block 106
    See also create-f
create-block 106, 105, 106, 139
create-bound-pair-list 121, 120, 122
create-builtin-function-reference 141, 135, 332, 334, 346, 351, 355, 366
create-condition 112, 103, 112, 113, 118
create-condition-prefix-list 112, 105, 106
create-constant 147, 61, 95, 132, 391
    See also create-f
create-data-description 122, 111, 121, 123, 125, 151
create-data-type 124, 123
create-declaration 119, 76, 106
create-enquiry-reference 149, 133
create-entry 125, 120, 124
create-entry-point 110, 105
create-entry-reference 143, 136, 137, 186
create-executable-unit 110, 109, 110, 116
create-executable-unit-list 110, 106, 108
create-expression 131, 108, 109, 113, 115, 116, 117, 127, 128, 129, 132, 133, 138, 140, 317, 319
    See also create-f
create-f 103
    See also create-allocation
    See also create-array-init
    See also create-assignment-statement
    See also create-begin-block
    See also create-constant
    See also create-expression
    See also create-format-iteration
    See also create-freeing
    See also create-group
    See also create-identifier
    See also create-if-statement
    See also create-initial
    See also create-initial-element
    See also create-integer
    See also create-leave-statement
    See also create-on-statement
    See also create-open-statement
    See also create-picture
    See also create-remote-format
    See also create-selection
create-format-iteration 115
    See also create-f
create-format-statement 114, 106
create-freeing 115
    See also create-f
create-group 108
    See also create-f
create-identifier 126, 111, 119, 123
    See also create-f
create-if-statement 116
    See also create-f
create-initial 126
    See also create-f
create-initial-element 127
    See also create-f
create-integer 128
    See also create-f
create-leave-statement 117
    See also create-f
create-named-constant 120, 93, 119
create-named-constant-reference 139, 117, 135
create-number-of-digits 130, 130
create-numeric-picture 146, 145
create-on-statement 118

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

See also create-f
create-open-statement 119
See also create-f
create-package 104, 76
create-picture 145, 73, 124, 347
See also create-f
create-precision 130, 107, 129, 148
create-procedure 105, 76, 106
create-pseudo-variable-reference 142, 136
create-refer-option 126, 122, 124
create-reference 134, 91, 104, 107, 113, 115, 131, 132, 136
create-remote-format 115
See also create-f
create-scale-factor 131, 130
create-selection 109
See also create-f
create-statement-name 111, 110, 114, 117
create-value-reference 139, 136, 137
create-variable 121, 119
⟨credit⟩ 145, 145, 146, 392, 393, 395
current 166
current block 166
⟨current-file-value⟩ 162, 162, 192, 268, 270, 271, 275, 276
⟨current-position⟩ 164, 163, 164, 248, 253, 255, 256, 257, 259, 260, 261, 262, 266, 271, 274, 278, 289, 292
⟨current-scalar-item⟩ 163, 163, 270, 273, 282, 284, 285, 293, 294
⟨current-scalar-item-list⟩ 162, 268, 293, 294
CURRENTSIZE 314, 344, 409
currentsize-bif 344, 257
<currentsize-bif> 72
⟨cv-target⟩ 162, 162, 178, 179, 180, 181
⟨cv-type⟩ 162, 162, 178, 179, 180, 181

D

D 61
⟨D⟩ 395
DATA 417
⟨data-attribute⟩ 45, 41, 45, 80, 81, 82, 83, 85, 86, 92, 93, 94, 96, 97, 98, 99, 100, 102, 108, 124, 128, 129
⟨data-attribute-list⟩ 45, 47, 82, 98
<data-description> 64
13, 15, 64, 65, 66, 69, 71, 72, 73, 107, 111, 113, 114, 121, 122, 123, 125, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 142, 143, 144, 149, 150, 151, 152, 153, 156, 157, 158, 165, 168, 174, 180, 181, 182, 183, 185, 186, 189
199, 200, 201, 202, 205, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 219, 220, 222, 223, 226, 227, 228, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 240, 242, 253, 256, 257, 270, 271, 275, 276, 281, 293, 295, 299, 300, 301, 302
315, 316, 317, 319, 332, 333, 345, 346, 351, 352, 353, 355, 356, 359, 365, 371, 372
<data-format> 71, 71, 274, 275, 284, 285, 297
⟨data-format⟩ 51, 51
⟨data-item-control⟩ 162, 162, 270, 273, 282, 283, 293, 294
⟨data-item-control-list⟩ 162, 268, 270, 273, 282, 283, 293, 294
⟨data-item-indicator⟩ 162, 162, 180, 270, 273, 282, 283, 293, 294
⟨data-list-indicator⟩ 162, 162, 270, 273, 282, 283, 293, 294
⟨data-symbol⟩ 273, 273
<data-type> 65
35, 64, 65, 73, 93, 107, 108, 109, 110, 114, 123, 124, 125, 128, 129, 130, 132, 133, 136, 137, 140, 141, 143, 147, 148, 149, 150, 151, 153, 156, 157, 160, 162, 178, 179, 181, 182, 183, 184, 185, 202, 217, 218, 221, 223, 224, 226, 227, 228
229, 230, 232, 233, 234, 236, 238, 253, 257, 264, 270, 271, 275, 277, 282, 285, 286, 287, 293, 294, 300, 301, 302, 303, 304, 305, 306, 307, 308, 310, 311, 312, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331
332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381
384, 385, 386, 387, 388, 389, 390, 391, 392, 397
⟨data-type⟩ 95, 95, 96
⟨dataset⟩ 165, 14, 15, 17, 19, 20, 61, 165, 247, 248, 250, 252, 254, 265, 266, 271, 274, 278, 289, 292
⟨dataset-designator⟩ 163, 248, 252, 254, 259, 262, 265, 266, 271, 274, 278, 289, 292
⟨dataset-list⟩ 14, 18, 40, 161, 166, 167
⟨dataset-name⟩ 165, 165, 252
datasets 4
DATE 408
date-bif 344
<date-bif> 72, 155, 333
DATETIME 18, 408, 409
datetime-bif 345
<datetime-bif> 72, 155, 333
DCL 62
DCLSIZE 62
⟨debit⟩ 145, 145, 146, 392, 393, 395
DEC 62
DECAT 419
DECIMAL 45, 62, 74, 95, 96, 97, 129, 148, 417, 420
<decimal> 35, 65, 74, 125, 129, 147, 179, 277, 286, 287, 303, 339, 345, 346, 366, 367, 384, 385, 386, 387, 388, 390, 391
decimal-bif 345
<decimal-bif> 72
⟨decimal-constant⟩ 57, 57, 61
⟨decimal-number⟩ 57, 57, 277, 390
<declaration> 64
13, 63, 64, 74, 76, 106, 107, 111, 113, 114, 115, 117, 119, 120, 122, 134, 135, 136, 139, 141, 142, 144, 149, 152, 153, 154, 155, 156, 157, 158, 159, 160, 167, 168, 173, 174, 185, 196, 203, 204, 206, 209, 211, 213, 214, 216, 218

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

228, 231, 234, 235, 238, 241, 242, 244, 255, 314, 371
⟨declaration⟩ 44, 39, 44, 75, 76, 82, 83, 86, 89, 90, 92, 93, 94, 95, 96, 97, 98, 99, 101, 102, 104, 105, 106, 108, 111, 113, 114, 115, 117, 119, 120, 121, 122, 123, 124, 125, 128, 134, 144, 148, 149
⟨declaration-commalist⟩ 39, 44, 81, 82, 83, 86, 87, 88, 90, 91, 99
declaration-component 75
declaration-contains 75
<declaration-designator> 63, 68, 69, 72, 73, 74, 113, 114, 115, 134, 136, 140, 144, 150, 153, 154, 155, 156, 158, 161, 168, 169, 203, 204, 206, 209, 211, 216, 218, 228, 231, 234, 235, 241, 242, 244, 245, 314, 336, 371
⟨declaration-designator⟩ 64, 66, 106, 108, 111, 113, 114, 115, 117, 120, 134, 139, 149
⟨declaration-designator-list⟩ 90
<declaration-list> 63, 67, 76, 106, 159, 185
<declaration-type> 64, 64, 74, 119, 120
⟨declaration-type⟩ 95, 95
DECLARE 3, 9, 16, 41, 44, 53, 62, 74, 86, 91, 406, 408, 415, 418, 424
declare-parameters 84, 79, 83
⟨declare-statement⟩ 44, 41, 42, 44, 53, 81, 82, 85, 86, 91, 94, 102, 121
declare-statement-names 84, 83, 85
⟨declared-statement-names⟩ 83, 83, 84, 85
DECLARESIZE 52, 62, 314, 344, 408, 410
⟨declaresize-arguments⟩ 52, 52
See also ⟨XX-arguments⟩
<declaresize-enquiry> 73, 150, 314
See also ⟨XX-enquiry⟩
⟨declaresize-enquiry-name⟩ 52, 52, 149
See also ⟨XX-enquiry-name⟩
⟨declaresize-enquiry-reference⟩ 52, 52
deduce-in-option 218, 217
DEF 62
defactor-declarations 82, 81, 82
DEFAULT 415, 424
DEFINE 3, 44, 406, 408, 418
⟨define-constant-statement⟩ 44, 44, 81, 82, 93, 94
⟨define-constant-statement-list⟩ 42, 43
define-program 39, 4, 39
DEFINED 45, 62, 95, 101, 102, 121, 407, 418, 425
<defined> 64, 13, 64, 121, 152, 153, 154, 174, 175, 209, 231, 240, 241, 242
≪defined-directory≫ 161, 161, 173, 174, 209
≪defined-directory-entry≫ 162, 13, 162, 175, 209, 241
≪defined-directory-entry-list≫ 13, 161, 175
defining production-rule 25
DELETE 262, 3, 49, 248, 262, 401
delete [instruction] 33
≪delete-flag≫ 163, 248, 252, 260, 262, 266, 267
<delete-statement> 69, 66, 69, 261
⟨delete-statement⟩ 49, 43, 49
⟨delimiter⟩ 57, 57, 77, 78
⟨delimiter-list⟩ 57
⟨delimiter-or-non-delimiter⟩ 78, 78
⟨delimiter-or-non-delimiter-list⟩ 78
⟨delimiter-pair⟩ 57, 57, 77
⟨delimiter-pair-list⟩ 57, 77
denotation [of category-name] 23
denotation [of PL/I characters] 24
denotation [of PL/I keyword] 24
derived base 303
derived common base 303
derived common scale 303
derived common string-type 304
derived scale 303
derived string-type 304
⟨description⟩ 45, 45, 47, 53, 75, 93, 94, 96, 97, 98, 99, 102, 111, 122, 124, 125, 151
⟨description-commalist⟩ 45, 47, 87, 88, 98, 99, 125
designates 23
designator 23
≪designator≫ 162, 164, 248, 259, 262, 265, 266, 267, 278, 289, 294
determine-structure 87, 87
⟨digit⟩ 59, 57, 58, 59, 128, 145, 273, 345, 375, 390, 392
⟨digit-list⟩ 57
digit-position 146
⟨digits⟩ 146, 145, 146
DIM 62
DIMENSION 44, 62, 81, 82, 86, 95, 96, 102, 400, 410
⟨dimension-attribute⟩ 44, 44, 45, 81, 82, 85, 86, 120, 121, 122
dimension-bif 346
<dimension-bif> 72
⟨dimension-suffix⟩ 44, 44, 45, 82, 86
<dimensional-data-description> 153
≪dimensioned-aggregate-type≫ 165, 165, 246, 299
<dimensioned-data-description> 64, 64, 65, 113, 122, 134, 135, 140, 143, 153, 200, 201, 212, 213, 214, 238, 239, 240, 295, 299, 301, 345, 351, 355, 365, 372
DIRECT 45, 49, 95, 120
<direct> 66, 69, 119, 120, 163, 251, 252, 253, 262, 266
⟨direct-set⟩ 95, 95
<disabled> 68, 112
≪disabled≫ 192, 193
⟨disabled-computational-condition⟩ 44, 44, 112
<divide> 71, 132, 319
divide-bif 346, 305, 306
<divide-bif> 72
DO 3, 7, 47, 50, 56, 406, 414, 415, 416, 424
<do-spec> 67, 67, 70, 108, 152, 177, 178, 180, 294
⟨do-spec⟩ 47, 7, 47, 50, 56, 108
⟨do-statement⟩ 47, 43, 47, 53, 56, 108
DOT 419
⟨drifting-dollar-field⟩ 146, 146
⟨drifting-field⟩ 146, 145, 146
⟨drifting-insert-field⟩ 146, 146
⟨drifting-lparen-field⟩ 146, 146
⟨drifting-sign-field⟩ 146, 146
<dummy> 72, 143, 186
≪dummy≫ 162, 175, 186, 218

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

E

E 51, 57, 61, 417
⟨E⟩ 390
EDIT 50, 401, 407, 408, 410
edit-bif 347, 144
⟨edit-bif⟩ 72
<edit-directed-input> 70, 70, 178, 269, 270, 273
⟨edit-directed-input⟩ 50, 50
<edit-directed-output> 70, 70, 178, 281, 283
⟨edit-directed-output⟩ 50, 50
<edit-input-pair> 70, 70
⟨edit-input-pair⟩ 50, 50
edit-numeric-output 286, 61, 286
edit-numeric-picture 392, 287, 383, 387, 388
edit-numeric-picture-field 393, 61, 392, 398
<edit-output-pair> 71, 70, 71
⟨edit-output-pair⟩ 50, 50
≪element-aggregate-type≫ 165, 165, 246, 299
<element-data-description> 64, 64, 113, 122, 135, 138, 140, 200, 201, 212, 213, 214, 238, 239, 240, 295, 299, 301, 365, 372
ELSE 43, 54, 116, 400
<else-unit> 68, 68, 116, 117, 184
EMPTY 401
≪empty≫ 164, 224, 225, 347
empty-bif 347
<empty-bif> 72, 154, 155, 333
<enabled> 68, 112
≪enabled≫ 192, 193
END 3, 47, 53, 80, 407, 415
<end-statement> 66, 110, 118, 166, 171, 175, 185, 189
⟨end-statement⟩ 47, 47, 53, 80
ENDFILE 48
<endfile-condition> 68, 249, 267, 272, 279, 291
⟨ending⟩ 47, 42, 43, 47, 53, 80
ENDPAGE 48
<endpage-condition> 68, 197, 249, 288, 290, 292
⟨enquiry-lbracket⟩ 52, 52, 53, 133
⟨enquiry-rbracket⟩ 52, 52, 53, 133
<enquiry-reference> 73, 72, 73, 150, 151, 156, 313, 314
⟨enquiry-reference⟩ 52, 52, 133
ENTRY 4, 18, 24, 45, 52, 83, 85, 93, 95, 96, 97, 98, 99, 100, 101, 102, 104, 105, 120, 124, 125, 150, 151, 406, 410, 415, 417, 418, 424
<entry> 65, 65, 66, 111, 120, 124, 125, 136, 137, 139, 143, 151, 153, 154, 155, 185, 222, 233, 244, 316
⟨entry⟩ 95, 95
<entry-information> 66, 66, 110, 118
⟨entry-information⟩ 46, 46, 83, 84, 85
<entry-point> 66, 8, 40, 63, 66, 105, 110, 118, 172, 173, 185, 189, 194, 244
<entry-point-designator> 163, 164, 172, 173, 185, 189, 194, 244
entry-reference 185
≪entry-value≫ 164, 8, 14, 18, 40, 162, 163, 164, 166, 185, 194, 196, 244, 316
enumerated-tree 24
ENV 62
ENVIRONMENT 18, 46, 62, 95, 120, 401, 414
<environment> 66, 17, 66, 69, 70, 120, 160, 168, 251, 254, 256, 258, 260, 261
⟨environment⟩ 46, 45, 46, 49
<environment-name> 160, 18, 159, 160
⟨environment-specification⟩ 46, 46, 79
epilogue 175, 171, 172, 187, 189, 190, 199
EPSILON 19, 35, 53, 411
⟨epsilon-arguments⟩ 53, 53
See also ⟨XX-arguments⟩
<epsilon-enquiry> 73, 150, 156, 315
See also ⟨XX-enquiry⟩
⟨epsilon-enquiry-name⟩ 53, 53, 150
See also ⟨XX-enquiry-name⟩
⟨epsilon-enquiry-reference⟩ 53, 52, 53
<eq> 71, 132
equal [trees] 22
≪equal≫ 183, 322, 323, 324, 325, 327
ERF 419
ERFC 419
ERROR 19, 48, 112, 414
<error-condition> 68, 163, 182, 196, 197, 249, 256, 258, 260, 262, 269, 272, 278, 279, 280, 289, 306, 331
establish-argument 186, 186, 199
establish-controlled-group 178, 177, 180, 294
establish-next-data-item 293, 270, 273, 282, 284
establish-next-format-item 296, 273, 284, 297
establish-truth-value 184, 177, 180, 182, 184, 190, 320, 330
≪established-argument≫ 162, 162, 173, 185, 186, 218
≪established-argument-list≫ 8, 162, 173, 185
≪established-on-unit≫ 163, 8, 163, 194, 195, 196
≪established-on-unit-list≫ 8, 161, 194, 195, 196
evaluate-argument 314
evaluate-builtin-function-reference 333, 155, 300, 313, 332
evaluate-constant 314
See also evaluate-f
evaluate-current-column 292, 274, 282, 283, 284, 288, 289, 290, 291
evaluate-current-line 292, 288, 290, 291, 356
evaluate-data-description-for-allocation 208, 13, 168, 174, 186, 205, 210, 222, 315
evaluate-data-description-for-reference 237, 235
evaluate-defined-reference 241, 231
evaluate-enquiry-reference 314, 156, 313
evaluate-entry-reference 185, 184, 313
evaluate-exponent-pv 227
See also evaluate-f
evaluate-expression 313, 16, 17, 32, 156, 178, 179, 181, 182, 183, 184, 186, 189, 214, 215, 219, 228, 229, 253, 264, 270, 293, 300, 302, 314, 348, 349, 356, 364, 365, 370, 373, 384
evaluate-expression-to-integer 302, 113, 129, 131, 141, 144, 159, 185, 208, 215, 238, 240, 244, 246, 250, 269, 280, 298, 351, 352, 356, 358
evaluate-f 226, 226, 220, 313
See also evaluate-constant
See also evaluate-exponent-pv
See also evaluate-infix-expression

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

See also evaluate-onsource-pv
See also evaluate-pageno-pv
See also evaluate-parenthesized-expression
See also evaluate-prefix-expression
See also evaluate-string-pv
See also evaluate-substr-pv
See also evaluate-unspec-pv
See also evaluate-value-reference
evaluate-file-option 250, 194, 250, 254, 255, 258, 259, 261, 269, 280
evaluate-filename 253, 252
evaluate-format-expression 298, 296, 297
evaluate-format-item 297, 296
evaluate-from-option 263, 258, 259
evaluate-in-option 206, 31, 205
evaluate-infix-expression 319
See also evaluate-f
evaluate-into-option 263, 255
evaluate-key-option 264, 256, 260, 261
evaluate-keyfrom-option 264, 258
evaluate-keyto-option 264, 256
evaluate-named-constant-reference 244, 155, 191, 313
evaluate-named-io-condition 194, 193, 194, 195
evaluate-onsource-pv 227
See also evaluate-f
evaluate-pageno-pv 228
See also evaluate-f
evaluate-parenthesized-expression 314
See also evaluate-f
evaluate-pointer-set-option 263, 255
evaluate-prefix-expression 317
See also evaluate-f
evaluate-real-constant 390, 148, 385, 390
evaluate-restricted-expression 155, 108, 113, 129, 131, 141, 144, 154, 155
evaluate-simply-defined-reference 241, 241
evaluate-size 267, 257, 261, 265
evaluate-sizeto-option 265, 255
evaluate-string-overlay-defined-reference 242, 241
evaluate-string-pv 229
See also evaluate-f
evaluate-substr-pv 229
See also evaluate-f
evaluate-target-reference 220, 18, 16, 178, 219, 220, 226, 264, 265, 281, 293
evaluate-title-option 253, 250
evaluate-unspec-pv 230
See also evaluate-f
evaluate-value-reference 313, 185, 187, 234, 250, 297
See also evaluate-f
evaluate-variable-reference 231, 16, 186, 205, 206, 217, 218, 220, 227, 229, 231, 241, 263, 294, 313, 336, 344, 346, 351, 352, 356, 359, 371, 379, 383
<evaluated-condition> 163, 163, 194, 195, 196
<evaluated-data-description> 165
13, 162, 164, 165, 168, 174, 186, 200, 201, 205, 206, 208, 209, 210, 211, 214, 217, 220, 222, 224, 225, 227, 229, 230, 232, 233, 235, 236, 237, 238, 239, 241, 242, 247, 255, 257, 261, 265, 267, 276, 294, 295, 315, 324, 325, 344, 346
351, 352, 356, 359, 383
<evaluated-data-description-list> 294, 295
<evaluated-delete-statement> 261, 261, 262, 266
<evaluated-entry-reference> 162, 8, 162, 167, 172, 173, 184, 185, 186, 196, 313
<evaluated-environment> 164, 163, 164, 168, 251, 254, 255, 256, 258, 259, 260, 261
<evaluated-file-description> 163, 163, 250, 251, 252, 256, 258, 356, 364
<evaluated-file-description-list> 163, 250, 251, 255, 256, 258, 260, 262, 269, 280
<evaluated-from-option> 263, 258, 259, 260, 263
<evaluated-initial-element> 214, 214, 215
<evaluated-initial-element-list> 214, 215
<evaluated-initial-item> 214, 214, 215
<evaluated-into-option> 263, 255, 256, 263
<evaluated-io-condition> 163, 163, 191, 194, 195, 196, 249
<evaluated-iteration-factor> 214, 214, 215
<evaluated-keyfrom-option> 264, 258, 259, 264
<evaluated-keyto-option> 264, 255, 256, 257, 258, 264
<evaluated-linesize> 163, 163, 250, 252, 283, 284, 288, 289
<evaluated-pagesize> 164, 163, 164, 250, 252, 288, 290, 291
<evaluated-pointer-set-option> 263, 255, 257, 263
<evaluated-pseudo-variable-reference> 165, 165, 220, 222, 226, 227, 228, 229, 231
<evaluated-read-statement> 255, 255, 256, 266, 267
<evaluated-rewrite-statement> 259, 259, 260, 266
<evaluated-sizeto-option> 265, 255, 257, 265
<evaluated-tab-option> 163, 163, 252, 282, 289, 290
<evaluated-target> 165, 162, 163, 165, 178, 180, 181, 205, 211, 214, 216, 219, 220, 221, 222, 256, 257, 264, 265, 271, 273, 275, 281, 293
<evaluated-title> 163, 163, 250, 252, 253, 254
<evaluated-write-statement> 258, 258, 259
EVERY 402, 414
every-bif 348, 32
<every-bif> 72
<executable-single-statement> 43, 42, 43, 44, 53, 54, 110, 116, 118
<executable-unit> 66, 18, 66, 67, 68, 109, 110, 116, 117, 118, 158, 166, 172, 173, 174, 175, 176, 177, 182, 184, 187, 188, 190, 191, 192, 245
<executable-unit> 42, 42, 43, 53, 54, 80, 84, 109, 110, 116
<executable-unit-designator> 162, 164, 166, 173, 174, 175, 176, 177, 180, 182, 184, 187, 188, 190, 191, 192, 245, 268
<executable-unit-list> 63, 67, 106, 108, 110, 118, 172, 174, 176, 189, 245
execute-allocate-statement 203, 187
See also execute-xxx
execute-assignment-statement 219, 17
See also execute-xxx
execute-begin-block 177
See also execute-xxx
execute-call-statement 184
See also execute-xxx
execute-close-statement 254
See also execute-xxx
execute-delete-statement 261

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

See also execute-xxx
execute-end-statement 189
See also execute-xxx
execute-executable-unit 176, 173, 176
execute-free-statement 216
See also execute-xxx
execute-get-file 269, 269
execute-get-statement 269
See also execute-xxx
execute-get-string 270, 269
execute-goto-statement 187
See also execute-xxx
execute-group 177
See also execute-xxx
execute-if-statement 184
See also execute-xxx
execute-input-control-format 274, 274
execute-input-data-format 275, 274
execute-leave-statement 190
See also execute-xxx
execute-null-statement 188
See also execute-xxx
execute-on-statement 194, 8
See also execute-xxx
execute-open-statement 250
See also execute-xxx
execute-output-control-format 284, 284
execute-output-data-format 285, 284
execute-put-file 280, 280
execute-put-statement 280
See also execute-xxx
execute-put-string 281, 280
execute-read-statement 255
See also execute-xxx
execute-return-statement 189
See also execute-xxx
execute-revert-statement 195, 8
See also execute-xxx
execute-rewrite-statement 259
See also execute-xxx
execute-selection 182
See also execute-xxx
execute-signal-statement 193
See also execute-xxx
execute-single-closing 254, 254
execute-single-opening 250, 9, 17, 250
execute-stop-statement 172
See also execute-xxx
execute-write-statement 258
See also execute-xxx
execute-xxx 176
See also execute-allocate-statement
See also execute-assignment-statement
See also execute-begin-block
See also execute-call-statement
See also execute-close-statement
See also execute-delete-statement
See also execute-end-statement
See also execute-free-statement
See also execute-get-statement
See also execute-goto-statement
See also execute-group
See also execute-if-statement
See also execute-leave-statement
See also execute-null-statement
See also execute-on-statement
See also execute-open-statement
See also execute-put-statement
See also execute-read-statement
See also execute-return-statement
See also execute-revert-statement
See also execute-rewrite-statement
See also execute-selection
See also execute-signal-statement
See also execute-stop-statement
See also execute-write-statement
exit-from-io 268, 249, 261, 266
exp-bif 348
<exp-bif> 72
expand-edd 295, 294, 295
expand-generation 295, 293
EXPONENT 19, 35, 409
{exponent} 57, 57, 390
exponent-bif 349, 221
<exponent-bif> 72
<exponent-pv> 72, 221, 227
{exported-entry} 46, 46, 104
{exported-entry-commalist} 46, 104
EXPORTS 46, 407
{exports-option} 46, 46, 104
<expression> 71
10, 13, 16, 17, 27, 34, 35, 63, 64, 65, 67, 68, 69, 70, 71, 72, 107, 108, 109, 110, 113, 115, 116, 117, 127, 128, 129, 131, 132, 133, 134, 138, 140, 141, 142, 143, 144, 149, 150, 151, 152, 154, 155, 157, 159, 174, 177, 178
179, 180, 181, 182, 183, 184, 185, 186, 189, 190, 200, 208, 214, 215, 219, 222, 226, 228, 229, 230, 233, 238, 240, 244, 246, 250, 253, 264, 269, 270, 280, 292, 293, 296, 297, 300, 302, 306, 313, 314, 317, 319, 334, 336, 344, 345, 351, 352, 355 359, 362, 365, 366, 371, 372, 379, 381, 391
{expression} 51, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 55, 56, 81, 93, 103, 107, 108, 109, 113, 115, 116, 117, 122, 124, 127, 128, 129, 131, 132, 133, 138, 140
{expression-commalist} 43, 109
{expression-designator} 64, 76, 106, 107, 122, 124
{expression-eight} 51, 51, 131
{expression-five} 51, 51, 131, 132
{expression-four} 51, 51, 131, 132
<expression-list> 67, 156
{expression-nine} 51, 51, 131
{expression-one} 52, 52, 131, 132
{expression-seven} 51, 51, 131, 132
{expression-six} 51, 51, 131, 132
{expression-three} 51, 51, 131, 132
{expression-two} 52, 51, 52, 131
EXT 62
<extent-expression> 64, 13, 64, 65, 66, 111, 113, 129, 144, 153, 154, 155, 159, 165, 174, 185, 186, 200, 208, 211, 230, 233, 237, 240, 241, 285, 353
{extent-expression} 44, 44, 45, 86, 121, 129

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

EXTERNAL 18, 45, 46, 62, 74, 83, 93, 94, 95, 97, 100, 101, 102, 104, 105, 119, 160, 405, 406
external procedures 4
<external> 64, 74, 119, 159, 161, 167, 168, 169, 196, 209, 244, 245, 251
<external-name> 64, 18, 64, 159, 160
{external-name} 47, 45, 46, 47, 100, 104
<external-unit> 63, 40, 63, 75, 152, 156, 158
{external-unit} 39, 39, 75, 80, 81, 82, 83, 87, 89, 91, 93, 101, 152
<external-unit-list> 40, 63, 244
extract-slice-of-array 243, 240, 246
{extralingual-character} 58, 24, 57, 58, 59, 61, 145, 273

F

F 51, 61, 415, 417
{f} 103
≪fail≫ 9, 31, 155, 156, 205, 206, 207, 217, 218, 228, 251, 252, 256, 258, 260, 262, 269, 280, 334, 363, 364
≪false≫ 93, 94, 95, 96, 143, 144, 157, 158, 178, 180, 181, 182, 183, 184, 190, 225, 232, 233, 242, 243, 293, 307, 308, 309, 310, 321, 330
FILE 4, 8, 9, 45, 49, 52, 81, 92, 95, 96, 97, 120, 124, 150, 151, 410, 413, 417, 418
<file> 24, 25, 65, 66, 120, 124, 151, 167, 228, 245, 316, 349, 356, 364
<file-description-set> 66, 24, 25, 66, 120, 163, 168, 251
{file-description-set} 95, 95, 102
≪file-directory≫ 161, 161, 167, 245
≪file-directory-entry≫ 161, 161, 167, 168, 245, 251
≪file-directory-entry-list≫ 161, 168
≪file-information≫ 163, 163, 168, 169, 192, 196, 218, 228, 248, 249, 250, 251, 254, 255, 256, 258, 259, 260, 261, 262, 265, 266, 268, 269, 271, 272, 274, 278, 280, 282, 284, 288, 289, 290, 291, 292, 349, 356, 364
≪file-information-designator≫ 161, 164, 168, 245, 251
≪file-information-list≫ 161, 168, 219
≪file-opening≫ 163, 163, 248, 250, 252, 254, 257, 262
<file-option> 69, 7, 17, 69, 70, 247, 248, 249, 250, 254, 255, 258, 259, 261, 269, 280
{file-option} 49, 49, 50, 81, 92
≪file-value≫ 164
17, 162, 163, 164, 169, 192, 194, 196, 197, 221, 228, 245, 248, 249, 250, 251, 254, 255, 256, 258, 259, 260, 261, 262, 265, 266, 268, 269, 270, 271, 272, 273, 274, 275, 276, 278, 279, 280, 282, 283, 284, 285, 288, 289, 290, 291, 292, 316, 349 356, 364
≪filename≫ 163, 163, 168, 192, 196, 249, 251, 252, 253, 255
FILEOPEN 409
fileopen-bif 349
<fileopen-bif> 72
find-applicable-declaration 89, 84, 89, 92, 98, 99, 101, 104, 111, 113, 114, 115, 117, 126, 134, 149
find-block-state-of-declaration 211, 209, 231, 241, 244, 245
find-directory-entry 209, 13, 174, 232
find-fully-qualified-name 90, 90, 101, 126, 134
find-item-data-description 201, 201, 202, 207, 208, 209, 210, 217, 224, 226, 232, 236
FINISH 48, 112, 403
<finish-condition> 68, 163, 172, 189, 197
first [tree] 22
≪first-comma≫ 162, 162, 163, 248, 252, 270, 272
FIXED 16, 35, 41, 45, 81, 82, 95, 96, 97, 103, 125, 129, 148, 417, 418, 420
<fixed>
65, 125, 129, 130, 141, 147, 153, 157, 177, 179, 229, 277, 286, 302, 303, 305, 319, 321, 328, 330, 331, 334, 339, 341, 345, 350, 351, 353, 355, 358, 360, 361, 365, 366, 367, 371, 372, 378, 380, 382, 384, 386, 387, 388, 389, 390, 391
fixed-bif 350
<fixed-bif> 72
<fixed-point-format> 71, 20, 71, 277, 286
{fixed-point-format} 51, 51
FIXEDOVERFLOW 44, 62
<fixedoverflow-condition> 68, 196, 305, 385
FLOAT 24, 35, 41, 45, 74, 95, 96, 97, 129, 148, 402
<float>
35, 65, 74, 125, 129, 130, 151, 179, 226, 287, 303, 306, 307, 308, 310, 311, 312, 315, 316, 319, 321, 322, 328, 330, 331, 335, 336, 337, 338, 340, 342, 343, 348, 349, 350, 357, 358, 360, 361, 365, 366, 367, 368, 369, 370, 372, 374, 375 379, 382, 385, 386, 387, 388, 390, 391
float-bif 350
<float-bif> 72
<floating-point-format> 71, 71, 277, 286, 287, 388
{floating-point-format} 51, 51
floor-bif 351, 306
<floor-bif> 72
FOFL 62
follow 22
for each [instruction] 34
form 25
FORMAT 3, 45, 51, 80, 85, 95, 96, 101, 120, 124, 401, 415, 417, 418
<format> 65, 66, 120, 124, 245
≪format-control≫ 162, 162, 192, 273, 283, 296, 297
≪format-control-list≫ 162, 192, 268, 273, 283, 296, 297, 298
<format-item> 71, 71, 115, 273, 284, 296, 297, 391, 392
{format-item} 51, 50, 51, 115, 144
<format-iteration> 71, 71, 115, 296
{format-iteration} 50, 50, 103, 115
<format-iteration-factor> 71, 71, 115, 296
{format-iteration-factor} 51, 50, 51, 115
≪format-iteration-index≫ 162, 162, 296
≪format-iteration-value≫ 162, 162, 296
≪format-list-index≫ 162, 162, 273, 283, 296, 297
<format-specification> 71, 71, 115
{format-specification} 50, 50
{format-specification-commalist} 50, 51, 114, 115

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

<format-specification-list> 70, 71, 114, 115, 273, 283, 296, 297
<format-specification-list-designator> 162, 273, 283, 296, 297
<format-statement> 71, 71, 106, 114, 192, 245, 297
⟨format-statement⟩ 51, 42, 51, 81, 84, 106, 114
<format-statement-designator> 162, 164, 192, 245, 297
<format-statement-list> 63, 67, 106, 245
≪format-value≫ 164, 164, 245, 297
FREE 218, 3, 49, 175, 199, 218, 254, 256, 259, 261, 263, 268, 400, 401
free-based-storage 216, 199, 216
<free-statement> 69, 66, 69, 199, 216
⟨free-statement⟩ 49, 43, 49
<freeing> 69, 69, 115, 216, 218
⟨freeing⟩ 49, 49, 103, 115
⟨freeing-commalist⟩ 49
<freeing-list> 69, 216
FROM 15, 49, 401, 417
<from-option> 70, 69, 70, 258, 259, 263
⟨from-option⟩ 49, 49

G

<ge> 71, 132
generate-scalar-result 301
301, 302, 317, 318, 319, 320, 321, 322, 326, 327, 328, 329, 330, 332, 335, 336, 337, 338, 339, 341, 342, 343, 345, 346, 347, 348, 349, 350, 351, 353, 356, 357, 359, 360, 361, 362, 363, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376 377, 379, 380
≪generation≫ 165
13, 15, 18, 31, 161, 162, 164, 165, 168, 174, 175, 186, 200, 201, 202, 204, 205, 206, 207, 210, 213, 214, 216, 217, 218, 219, 220, 221, 222, 223, 225, 226, 227, 229, 230, 231, 232, 234, 236, 237, 238, 239, 240, 241, 242, 248, 255, 256 257, 258, 259, 260, 261, 263, 265, 268, 293, 294, 295, 300, 313, 336, 344, 346, 351, 352, 356, 359, 371, 379, 383
≪generation-list≫ 293, 295
GENERIC 418
GET 3, 7, 50, 407, 414, 417
get-edit 273, 269, 270
get-established-onvalue 334, 228, 363, 364
<get-file> 70, 70, 269
⟨get-file⟩ 50, 50
get-list 270, 269, 270
<get-statement> 70, 66, 70, 199, 269
⟨get-statement⟩ 50, 43, 50, 81
<get-string> 70, 17, 70, 269, 270, 391
⟨get-string⟩ 50, 50, 81
GO 74
GO TO 3, 48
go to [instruction] 33
GOTO 48
<goto-statement> 68, 17, 66, 68, 74, 175, 187
⟨goto-statement⟩ 48, 43, 48
≪greater-than≫ 322, 323, 327
<group> 67, 66, 67, 108, 158, 175, 177, 190, 191, 199
⟨group⟩ 43, 42, 43, 80, 103, 108, 116
≪group-control≫ 162, 162, 173, 174
<gt> 71, 132

H

HBOUND 410
hbound-bif 351
<hbound-bif> 72
HIGH 402
high-bif 352
<high-bif> 72
high-level-parse 78, 16, 28, 46, 76, 77
higher-model-number 311, 307, 308, 309
highest-positive-model-number 311, 306, 307, 308, 315, 349

I

I 415
IDENT 62
IDENTICAL 62, 410
identical-bif 352
<identical-bif> 72
<identifier> 73, 13, 18, 27, 64, 66, 73, 74, 111, 119, 123, 126, 134, 141, 142, 153, 158, 159, 160, 161, 162, 167, 168, 169, 173, 174, 185, 209, 213, 231, 241, 244, 245
⟨identifier⟩ 57, 15, 18, 44, 45, 46, 47, 48, 49, 51, 52, 55, 56, 57, 73, 77, 79, 80, 82, 83, 84, 86, 89, 90, 91, 92, 98, 99, 100, 101, 103, 104, 111, 113, 114, 115, 117, 119, 123, 126, 130, 131, 134, 148, 149
<identifier-list> 27, 64, 72, 74, 122, 126, 134, 139, 144, 153, 159, 208, 213, 225, 232, 235, 236, 237, 238, 241, 242, 299, 344
⟨identifier-list⟩ 89, 90, 91, 126, 134
⟨identifier-tail⟩ 57, 57
⟨identifier-tail-list⟩ 57
IF 3, 43, 54, 416
if [instruction] 33
⟨if-clause⟩ 43, 43, 54, 116, 117
<if-statement> 68, 66, 68, 116, 117, 175, 184
⟨if-statement⟩ 43, 42, 43, 53, 54, 103, 116
IGNORE 417
IMAG 419
immediate component 21
immediate subnode 21
immediate subtree 21
immediately contain 21, 10
immediately follow 22
immediately precede 22

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

implementation 1, 4, 14, 15, 16, 17, 18, 20, 35, 36, 59, 61, 207, 230, 252, 259, 261, 265, 266, 314, 333, 344, 354, 379, 391, 402, 405, 410
implementation-defined 18
14, 15, 17, 18, 29, 35, 46, 58, 66, 78, 79, 96, 97, 120, 125, 160, 164, 168, 185, 186, 195, 196, 197, 231, 247, 251, 252, 253, 254, 256, 257, 258, 260, 261, 265, 266, 267, 286, 287, 302, 312, 314, 315, 341, 344, 345, 355, 363
375, 379, 388, 397, 405, 409, 410
implementation-dependent 15, 23, 105, 106, 111, 196, 197, 307, 308, 309, 310, 324, 325, 384, 388
IN 49, 400
in any order 31
<in-option> 69, 69, 114, 116, 206, 216, 218
{in-option} 49, 48, 49, 114, 116
INCLUDE 3, 18, 58, 414
{include} 58, 57, 58, 78
INDEX 410
index-bif 353
<index-bif> 72
infix-add 320, 9, 306
See also infix-f
infix-and 320
See also infix-f
infix-and-logical 320, 319
See also infix-f
infix-cat 321
See also infix-f
infix-divide 322, 306
See also infix-f
infix-eq 322
See also infix-f
<infix-expression> 71, 71, 132, 305, 318, 319
infix-f 319
See also infix-add
See also infix-and
See also infix-and-logical
See also infix-cat
See also infix-divide
See also infix-eq
See also infix-ge
See also infix-gt
See also infix-le
See also infix-lt
See also infix-multiply
See also infix-ne
See also infix-or
See also infix-or-logical
See also infix-power
See also infix-subtract
See also infix-xor
infix-ge 326
See also infix-f
infix-gt 327
See also infix-f
infix-le 327
See also infix-f
infix-lt 328
See also infix-f
infix-multiply 328, 306
See also infix-f
infix-ne 329
See also infix-f
<infix-operator> 71, 71, 132, 318, 319
infix-or 329
See also infix-f
infix-or-logical 330, 319
See also infix-f
infix-power 330
See also infix-f
infix-subtract 332, 306
See also infix-f
infix-xor 332
See also infix-f
INIT 62
INITIAL 9, 46, 53, 62, 95, 102, 399, 400, 405, 408, 417
<initial> 65, 13, 64, 65, 107, 126, 139, 144, 152, 153, 159, 160, 169, 174, 213, 217, 235
{initial} 46, 45, 46, 103, 107, 123, 126
{initial-constant-one} 46, 46, 127
{initial-constant-two} 46, 46, 127, 128
{initial-designator} 65, 106, 107, 123
<initial-element> 65, 64, 65, 127, 128, 152, 154, 159, 214, 215
{initial-element} 46, 46, 103, 127
{initial-element-commalist} 46, 126, 128
<initial-element-list> 64, 65, 126, 128, 159, 214, 215
initialize-array 214, 9, 213
initialize-generation 213, 13, 169, 174, 213
initialize-interpretation-state 167, 167
initialize-refer-options 210, 205
initialize-scalar-element 214, 213
initialize-spec-options 178, 178
INPUT 45, 49, 95, 120, 413
<input> 24, 25, 66, 69, 119, 120, 163, 248, 251, 252, 256, 269, 291
<input-specification> 70, 17, 70, 269
{input-specification} 50, 50
input-stream-item 278, 272, 274, 275, 279, 291
input-stream-item-for-edit 279, 274, 275
<input-target> 70, 70, 293, 295
{input-target} 50, 50
{input-target-commalist} 7, 50
<input-target-list> 70, 270, 273
insert-record 265, 259
{insertion-character} 145, 145, 146, 395
{insertion-string} 145, 145, 146, 392, 393, 394, 395, 396
{insertion-symbol} 145, 145, 392, 396
{insertion-symbol-list} 145
install-arguments 173, 173
instruction 32
INT 62
<integer> 73, 65, 73, 74, 128, 130, 131, 391
{integer} 57, 44, 45, 51, 57, 73, 79, 99, 103, 115, 128, 130, 131, 145, 277
<integer-value> 164
64, 71, 72, 113, 129, 131, 141, 144, 159, 162, 163, 164, 165, 185, 186, 195, 200, 207, 208, 210, 211, 212, 214, 215, 217, 222, 224, 225, 227, 228, 230, 232, 237, 238, 240, 241, 243, 244, 246, 250, 252, 255, 257, 261, 265, 267, 269, 273, 274

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

275, 277, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 296, 297, 298, 299, 301, 302, 310, 311, 312, 315, 316, 325, 326, 334, 349, 353, 356, 359, 364, 390
≪integer-value-list≫ 163, 252
INTERNAL 45, 62, 83, 86, 93, 95, 97, 101, 105, 119
<internal> 64, 119, 209, 244
interpret 166, 8, 18, 40, 161, 171
interpretation [of a production rule] 28
interpretation-phase 40, 8, 34, 38, 39
≪interpretation-state≫ 161, 38, 39, 161, 166, 167, 187
interrupt 195, 18, 191, 192, 193, 194
INTO 15, 49, 417
<into-option> 69, 69, 255, 263
≪into-option≫ 49, 49
≪invalid≫ 275, 276, 277, 278
<io-condition> 68, 68, 112, 113, 163, 194, 248, 249
≪io-condition≫ 48, 48, 112, 113
ISOCHAR 18, 354, 409
isochar-bif 355
<isochar-bif> 72
iSUB 418, 425
<item-data-description> 64, 64, 66, 74, 113, 123, 132, 134, 135, 136, 137, 140, 143, 149, 150, 151, 153, 157, 159, 181, 200, 201, 202, 205, 207, 208, 209, 210, 211, 213, 214, 217, 219, 223, 224, 226, 230, 232, 233, 236, 271, 276, 295, 299, 301, 365, 372
<item-data-description-list> 233
<iteration-factor> 65, 64, 65, 159, 214, 215
≪iteration-factor≫ 46, 46, 128
<iterative-group> 67, 67, 108, 188

K

KEY 19, 48, 49
≪key≫ 166, 165, 166, 247, 252, 255, 256, 259, 260, 261, 262, 264, 265, 266, 267
<key-condition> 68, 193, 249, 259, 266
<key-option> 70, 69, 70, 256, 260, 261, 264
≪key-option≫ 49, 49
KEYED 20, 45, 49, 95, 120, 407
<keyed> 66, 69, 119, 120, 163, 251, 252, 256, 258, 259, 260, 262, 266
≪keyed-dataset≫ 165, 165, 252
≪keyed-record≫ 165, 165, 248, 252, 255, 256, 257, 258, 259, 260, 261, 262, 265, 266, 267
≪keyed-record-list≫ 165, 253, 265, 267
≪keyed-sequential-dataset≫ 165, 165, 252
KEYFROM 50
<keyfrom-option> 70, 70, 258, 264
≪keyfrom-option≫ 50, 49, 50
KEYTO 7, 49
<keyto-option> 70, 7, 69, 70, 256, 264
≪keyto-option≫ 49, 7, 49
keyword 78
known [operation] 36

L

L 51, 407
<l-format> 71, 275, 285
≪l-format≫ 51, 51
LABEL 4, 45, 52, 85, 95, 96, 101, 102, 120, 124, 150, 151, 410, 417
<label> 65, 66, 120, 124, 143, 151, 245, 316
≪label-value≫ 164, 164, 187, 188, 191, 245, 316
last [tree] 22
LBOUND 410
lbound-bif 355
<lbound-bif> 72
<le> 71, 132
≪leading-delimiter≫ 273, 273
≪leading-delimiter-list≫ 272
LEAVE 3, 48, 158, 190, 406
<leave-statement> 68, 66, 68, 117, 158, 190
≪leave-statement≫ 48, 43, 48, 103, 117, 118
LENGTH 409
length [of a string] 304
length-bif 356
<length-bif> 72
≪less-than≫ 322, 323, 327, 328
let [instruction] 32
≪letter≫ 59, 57, 58, 59, 79, 145, 273, 398
≪level≫ 44, 44, 45, 82, 84, 87, 88, 90, 99, 102, 122, 123
LIKE 418
LINE 50, 51
<line-format> 71, 71, 274, 284
≪line-format≫ 51, 51
<line-option> 70, 70, 280
≪line-option≫ 50, 50
≪linemark≫ 166, 272, 273, 274, 275, 280, 288, 290, 291, 292
≪linemark-list≫ 273
lineno-bif 356
<lineno-bif> 72
LINESIZE 20, 49
<linesize-option> 69, 69, 250
≪linesize-option≫ 49, 49
≪linkage-part≫ 163, 13, 161, 163, 173, 174, 175, 189, 192, 209
LIST 7, 50, 53, 54, 414
<list-directed-input> 70, 70, 178, 269, 270
≪list-directed-input≫ 50, 50
<list-directed-output> 70, 70, 178, 281, 282
≪list-directed-output≫ 50, 50
LOCAL 418
local-goto 188, 187
local-tree 30
local-variable-name 30
LOCATE 417, 424
<locator> 65, 65, 124, 136, 150, 204, 206
≪locater≫ 95, 95
<locator-qualifier> 72, 69, 72, 74, 116, 136, 216, 218, 234
≪locater-qualifier≫ 52, 49, 52, 115, 136

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

log-bif 357
<log-bif> 72
log10-bif 357
<log10-bif> 72
log2-bif 358
<log2-bif> 72
LOOP 47, 108, 406
<looping-group> 67, 67, 108, 177, 190
LOW 402
low-bif 358
<low-bif> 72
low-level-parse 77, 28, 58, 76, 78
<lower-bound> 64, 27, 64, 122, 200, 208, 222, 238, 240, 243, 244, 246, 295, 300, 301, 346, 356
{lower-bound} 44, 44, 86, 121, 122
lower-model-number 311, 307, 308, 309
lowest-positive-model-number 311, 306, 307, 308, 315, 349
{lparen} 146, 145, 146
{lparen-list} 146
<lt> 71, 132

M

machine-state syntax 4
≪machine-state≫ 39, 12, 13, 14, 30, 36, 37, 38, 39, 40, 166, 167, 199, 208, 247, 252
make-allocation-unit 209, 207
map-external-name-to-environment-name 160, 159
map-identifier-to-environment-name 160, 159
MAX 418
max-bif 359
<max-bif> 72
<maximum-length> 65, 65, 114, 124, 128, 139, 144, 147, 148, 153, 154, 186, 202, 208, 210, 217, 223, 224, 229, 230, 232, 236, 242, 253, 256, 264, 270, 271, 275, 276, 281, 282, 285, 320, 323, 324, 325, 329, 332, 359, 383, 384
{maximum-length} 45, 45, 93, 124, 129
MAXLENGTH 409
maxlength-bif 359
<maxlength-bif> 72
MAXVAL 19, 35, 53, 411
{maxval-arguments} 53, 52, 53
See also {XX-arguments}
<maxval-enquiry> 73, 150, 156, 315
See also {XX-enquiry}
{maxval-enquiry-name} 53, 52, 53, 150
See also {XX-enquiry-name}
{maxval-enquiry-reference} 52, 52
MEMBER 45, 80, 87, 88, 89, 90, 91, 94, 95, 96, 102, 106, 114, 115, 123, 125, 134, 149, 406, 417
≪member-aggregate-type≫ 165, 165, 299
≪member-aggregate-type-list≫ 165
<member-description> 64, 64, 123, 134, 153, 200, 208, 212, 213, 299, 301
<member-description-list> 64, 122, 134, 200, 201, 212, 213, 235, 236, 237, 238
MIN 418
min-bif 360
<min-bif> 72
<minus> 72, 133
{minus} 146, 146
{minus-list} 146
MINVAL 19, 35, 53, 411
{minval-arguments} 53, 53
See also {XX-arguments} 149
<minval-enquiry> 73, 150, 156, 315
See also {XX-enquiry} 151
{minval-enquiry-name} 53, 53, 150
See also {XX-enquiry-name} 149
{minval-enquiry-reference} 53, 52, 53
MOD 413
mod-bif 361
<mod-bif> 72
model-base 312, 35, 227, 310, 311, 315, 316, 349
model-maximum-exponent 312, 35, 227, 310, 311
model-minimum-exponent 312, 35, 227, 310, 311
model-numbers 35
model-parameters 35
model-precision 312, 35, 310, 311, 315
multiple-constraint 156
MULTIPLY 402
<multiply> 71, 132, 319
multiply-bif 362, 305, 306
<multiply-bif> 72
must 34
4, 7, 8, 12, 14, 17, 40, 42, 45, 47, 58, 70, 77, 78, 79, 80, 84, 86, 87, 90, 92, 96, 99, 101, 102, 104, 108, 109, 110, 113, 114, 117, 118, 119, 123, 124, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 141, 142
143, 145, 147, 148, 149, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 172, 177, 179, 184, 185, 186, 187, 188, 189, 192, 202, 205, 206, 208, 211, 217, 218, 219, 220, 221, 222, 223, 225, 226, 227, 228, 229, 230, 231, 233, 234, 235, 240, 241
242, 244, 246, 248, 250, 252, 256, 257, 258, 259, 260, 262, 263, 267, 269, 274, 275, 279, 280, 284, 285, 286, 287, 289, 290, 291, 297, 303, 315, 316, 317, 318, 319, 320, 321, 322, 324, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337
338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 386, 389, 391
must not 34
4, 8, 17, 40, 58, 64, 65, 69, 70, 78, 79, 80, 82, 84, 87, 88, 93, 94, 100, 101, 102, 104, 112, 114, 115, 118, 119, 126, 129, 130, 131, 134, 135, 136, 137, 139, 143, 145, 147, 149, 151, 153, 154, 155, 158, 160, 185, 187, 189
192, 196, 207, 209, 217, 221, 227, 228, 230, 231, 234, 237, 242, 246, 248, 250, 252, 259, 260, 262, 271, 274, 275, 277, 284, 286, 287, 296, 297, 312, 313, 321, 334, 337, 339, 341, 342, 344, 349, 352, 353, 356, 358, 361, 364, 370, 374, 375
390

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

N

NAME 416
⟨name⟩ 83, 83, 84
⟨NAME-enquiry-name⟩ 133
See also ⟨XX-enquiry-name⟩
⟨name-list⟩ 83, 84, 85, 86
<named-constant> 66, 24, 25, 64, 66, 106, 108, 117, 119, 120, 128, 129, 134, 135, 139, 152, 155, 158, 167, 185, 244
⟨named-constant⟩ 95, 95
<named-constant-reference> 72, 68, 72, 117, 135, 138, 139, 140, 141, 143, 155, 157, 158, 191, 244, 313
<named-io-condition> 68, 68, 113, 191, 193, 194, 195, 196, 249
⟨named-io-condition⟩ 48, 48, 92, 113
<ne> 71, 132
next [tree] 22
NOCONV 62
NOCONVERSION 44, 62
node 21
NOFIXEDOVERFLOW 44, 62
NOFOFL 62
⟨non-blank-comma⟩ 273, 273
⟨non-blank-comma-list⟩ 270, 273
⟨non-blank-comma-quote⟩ 273, 270, 273
<non-computational-type> 65, 65, 109, 114, 124, 140, 150, 322, 324, 329
⟨non-computational-type⟩ 95, 95
⟨non-delimiter⟩ 57, 57, 77, 78
⟨non-drifting-field⟩ 145, 145
<non-iterative-group> 67, 67, 108, 177, 190
non-terminal 26
non-trivial-basic-value-index 326, 324
<none> 103, 104, 233, 234, 270, 273, 282, 284, 293, 294
NONPRINT 45, 49, 95, 120
<nonprint> 66, 69, 120, 163, 251, 252
NONRECURSIVE 46, 406
NONVAR 62
NONVARYING 45, 62, 95, 97, 400
<nonvarying> 65, 124, 128, 139, 147, 148, 202, 207, 210, 226, 228, 230, 233, 234, 253, 256, 264, 270, 271, 281, 282, 285, 320, 323, 329, 332, 384
NOOFL 62
NOOVERFLOW 44, 62
normal-sequence 176, 8, 175, 177, 184, 188, 190, 191, 193, 195, 203, 216, 220, 250, 254, 256, 259, 260, 262, 268, 269, 280
NOSIZE 44
NOT 156
<not> 72, 133
<not-dummy> 162, 186
<not-equal> 322, 183, 322, 323, 324, 325, 327, 328, 329, 353
NOUFL 62
NOUNDERFLOW 44, 62
NOZDIV 62
NOZERODIVIDE 44, 62
NULL 52, 150, 410
<null> 164, 234, 316, 324, 325, 362, 383
⟨null-arguments⟩ 52, 52
See also ⟨XX-arguments⟩
null-bif 362
<null-bif> 72, 150, 154, 155, 333
<null-bit-string> 32, 147, 164, 202, 210, 224, 233, 279, 304, 305, 318, 320, 329, 332, 348, 386, 388, 389
<null-character-string> 147, 164, 193, 202, 210, 224, 233, 253, 270, 272, 275, 277, 278, 279, 281, 285, 289, 304, 305, 352, 358, 363, 364, 386, 388
<null-enquiry> 73, 151, 156, 316
See also ⟨XX-enquiry⟩
⟨null-enquiry-name⟩ 52, 52, 150
See also ⟨XX-enquiry-name⟩
⟨null-enquiry-reference⟩ 52, 52
<null-entry> 164, 185, 316
<null-file> 164, 228, 250, 316, 349, 356, 364
<null-label> 164, 316
<null-statement> 66, 188
⟨null-statement⟩ 48, 43, 48
null-string 305
<number-of-digits> 65, 19, 20, 65, 74, 130, 147, 179, 277, 287, 302, 303, 304, 305, 319, 330, 333, 334, 341, 350, 351, 358, 359, 360, 361, 365, 366, 372, 378, 379, 384, 385, 386, 387, 388, 389, 390, 391, 392
⟨number-of-digits⟩ 45, 45, 130
number-of-scalar-elements 300
<numeric-picture-specification> 73, 73, 147, 391, 392
⟨numeric-picture-specification⟩ 145, 145, 147
<numeric-string> 389, 389, 390

O

<off> 162, 272, 393, 394, 395, 396
OFFSET 4, 45, 52, 95, 96, 124, 150, 399, 402, 410, 419
<offset> 65, 65, 69, 107, 124, 136, 144, 150, 153, 154, 159, 185, 206, 217, 218, 225, 232, 233, 234, 235, 316, 362, 363, 365, 381, 383
offset-bif 363
<offset-bif> 72
<offset-value> 164, 164, 316, 324, 381, 382, 383, 384, 391
OFL 62
<omega> 165, 166, 247, 248, 260, 262, 267, 274, 275, 278, 279, 291, 292
ON 3, 8, 19, 44, 403
<on> 162, 252, 270, 272, 393, 394, 395, 396
<on-statement> 68, 66, 68, 118, 194, 249
⟨on-statement⟩ 44, 42, 43, 44, 103, 116, 118
<on-unit> 68, 17, 18, 68, 118, 172, 194
⟨on-unit⟩ 44, 44, 118
ONCHAR 419
ONCODE 19, 408
oncode-bif 363
<oncode-bif> 72, 155, 333
<oncode-value> 163, 163, 195, 334, 363

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

≪one-bit≫ 32, 149, 164, 184, 318, 320, 321, 322, 323, 327, 328, 329, 330, 332, 348, 349, 353, 370, 380, 386, 388, 389
ONFIELD 419
ONFILE 408
onfile-bif 363
<onfile-bif> 72, 155, 333
≪onfile-value≫ 163, 163, 192, 195, 196, 249, 334, 363
ONKEY 408
onkey-bif 364
<onkey-bif> 72, 155, 333
≪onkey-value≫ 163, 163, 193, 249, 334, 364
ONLOC 419
ONSOURCE 402, 408
onsource-bif 364, 221
<onsource-bif> 72, 155, 333
<onsource-pv> 72, 219, 221, 227
≪onsource-value≫ 163, 163, 175, 193, 228, 249, 334, 364, 386, 389
OPEN 251, 3, 9, 17, 49, 251, 256, 258, 260, 262, 269, 280, 401, 413, 417
≪open≫ 163, 169, 228, 248, 251, 253, 254, 256, 258, 259, 260, 261, 262, 269, 280, 349, 356, 364
≪open-state≫ 163, 163, 168, 253, 254
<open-statement> 69, 9, 17, 66, 69, 119, 250
{open-statement} 49, 43, 49, 103, 119
operand-name 30
operands 30
≪operation≫ 39, 13, 36, 37, 38, 39, 172, 173, 174, 175, 187, 189, 268
≪operation-list≫ 13, 36, 37, 38, 39, 162, 172, 173, 174, 175, 187, 189, 268
operations 30
optionally 31
OPTIONS 18, 46, 95, 102, 125, 400
<options> 66, 65, 66, 67, 105, 106, 111, 125, 139, 144, 160, 185, 225
{options} 46, 45, 46, 47, 100, 105, 106, 111
{options-specification} 46, 46, 79
OR 156
<or> 71, 132
<or-logical> 16, 71, 132
or-symbol 30
OTHERWISE 43, 406
otherwise [instruction] 33
{otherwise-clause} 43, 43
{otherwise-statement} 43, 43, 109
<otherwise-unit> 67, 67, 109, 182, 188
OUTPUT 20, 45, 49, 95, 120, 413
<output> 66, 69, 119, 120, 163, 251, 252, 253, 258, 280, 291
<output-source> 70, 70, 294, 295
{output-source} 50, 50
{output-source-commalist} 50
<output-source-list> 70, 71, 282, 283
<output-specification> 70, 17, 70, 281
{output-specification} 50, 50
output-stream-item 289, 288, 290, 291, 292
output-string 288, 283, 284, 285, 286, 290
output-string-item 288, 282, 288
output-tab 290, 289
OVERFLOW 35, 44, 62
<overflow-condition> 68, 191, 196, 307, 309, 310
overlay-strings 236, 235, 242

P

P 51, 58, 405, 415
PACKAGE 3, 45, 46, 80, 83, 95, 96, 101, 120, 407, 424
<package> 63, 40, 63, 75, 76, 104, 106, 192
{package} 42, 39, 41, 42, 75, 76, 80, 83, 84, 89, 92, 97, 100, 101, 104, 106
<package-constant> 66, 120, 139
{package-statement} 46, 42, 46, 104
packages 4
PAGE 50, 51
<page> 70, 71, 274, 280, 284
≪page-number≫ 164, 163, 164, 228, 248, 252, 289, 364
≪page-top-flag≫ 163, 268, 288, 289, 290, 291
≪pagemark≫ 166, 247, 252, 289, 291, 292
pageno-bif 364, 221
<pageno-bif> 72
<pageno-pv> 72, 219, 221, 228
PAGESIZE 20, 49
<pagesize-option> 69, 69, 250
{pagesize-option} 49, 49
PARAMETER 45, 62, 84, 95, 99, 101, 102, 121, 400, 419
<parameter> 64, 121, 152, 173, 209, 231
<parameter-descriptor> 65, 65, 111, 125, 143, 144, 152, 155, 185, 208, 241, 242
<parameter-descriptor-list> 65, 111, 125, 143, 144, 185, 225
≪parameter-directory≫ 162, 161, 162, 173, 209, 218, 231
≪parameter-directory-entry≫ 162, 162, 173, 175, 209, 231
≪parameter-directory-entry-list≫ 162, 173
<parameter-name> 66, 66
{parameter-name} 46, 46, 99, 111
{parameter-name-commalist} 46, 83, 84, 99, 100, 111
<parameter-name-list> 8, 40, 66, 111, 173
<parenthesized-expression> 72, 65, 71, 72, 127, 128, 134, 152, 157, 214, 215, 314
{parenthesized-expression} 52, 46, 52, 55, 127, 131, 133
PARM 62
parse 76, 28, 41, 75, 76, 81, 98, 347
parse-list-input 271, 270, 272
partial tree 26
perform [instruction] 33
PIC 62
{pic-key} 58, 58, 77
{pic-spec} 58, 57, 58, 77
≪pic-status≫ 393, 393
PICTURE 4, 45, 58, 62, 95, 96, 97, 102, 124, 400, 408, 417, 424
{picture} 46, 45, 46, 51, 103, 124, 144, 145, 287, 347, 387, 388
{picture-content} 144, 144, 145, 146
{picture-element} 145, 73, 145, 387, 392, 393

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

{picture-element-list} 73, 147, 391, 392, 393, 397, 398
<picture-format> 71, 71, 275, 276, 286, 397
{picture-format} 51, 51
≪picture-invalid≫ 276, 380, 397, 398
{picture-item} 145, 145, 146
{picture-item-list} 144, 146
≪picture-valid≫ 397, 276, 380, 386, 389, 397, 398
picture-validation syntax 145
≪picture-validity≫ 397, 276, 380, 386, 389, 397, 398
<pictured> 65, 65, 71, 124, 144, 145, 151, 202, 207, 210, 224, 226, 229, 232, 234, 286, 347, 353, 380
{pictured} 95, 95
<pictured-numeric> 73, 65, 73, 144, 145, 146, 147, 226, 229, 303, 323, 339, 341, 349, 353, 367, 371, 380, 382, 384, 385, 386, 387, 388, 389, 391, 392, 397
{pli-text} 57, 57, 58, 76, 78
<plus> 72, 133
{plus} 146, 146
{plus-list} 146
POINTER 4, 45, 52, 62, 95, 96, 124, 150, 399, 402, 410, 419
<pointer> 65, 124, 150, 177, 205, 206, 234, 257, 336, 362, 365, 381, 383
pointer-bif 365
<pointer-bif> 72
<pointer-set-option> 69, 69, 248, 255, 263
{pointer-set-option} 49, 49
≪pointer-value≫ 164, 164, 200, 205, 234, 257, 324, 336, 362, 381, 382, 383
POSITION 418, 425
position-file 266, 256, 260, 262
≪position-index≫ 165, 165, 203, 207, 224, 230, 232, 233, 237, 326
<power> 71, 132, 319
PREC 62
precede 22
PRECISION 45, 62, 80, 81, 82, 95, 96, 97, 102, 129, 417, 419
<precision> 65
35, 65, 74, 107, 125, 129, 130, 147, 148, 178, 179, 181, 303, 304, 312, 317, 318, 319, 320, 322, 323, 328, 330, 331, 332, 333, 335, 336, 337, 338, 339, 340, 341, 342, 343, 345, 346, 348, 350, 351, 357, 358, 360, 361, 362, 366, 367
368, 369, 370, 372, 374, 375, 378, 379, 386, 387, 390, 391
{precision} 45, 45, 81, 82, 107, 125, 129, 130, 148
{precision-designator} 65, 76, 106, 107, 125, 134, 135
<prefix-expression> 72, 71, 72, 133, 317
{prefix-expression} 52, 52, 131, 133
prefix-f 317
See also prefix-minus
See also prefix-not
See also prefix-plus
prefix-minus 317, 306
See also prefix-f
prefix-not 318
See also prefix-f
<prefix-operator> 72, 72, 133, 317
{prefix-operator} 52, 46, 52
prefix-plus 318
See also prefix-f
{primary-case-letter} 59, 59, 61, 62, 79
{primitive-expression} 52, 52, 55, 131, 132
PRINT 9, 20, 45, 49, 95, 120, 411
<print> 9, 66, 69, 119, 120, 163, 228, 248, 251, 252, 282, 283, 288, 289, 290, 291, 356, 364
PROC 62
PROCEDURE 3, 46, 53, 62, 403, 406, 415, 418
<procedure> 63, 10, 40, 63, 68, 75, 76, 105, 106, 118, 166, 172, 175, 185, 188, 189, 192, 244, 245
{procedure} 42, 39, 41, 42, 53, 75, 76, 80, 83, 84, 85, 89, 92, 98, 100, 105, 106, 110
<procedure-function-reference> 72, 16, 72, 139, 143, 172, 185, 313
<procedure-list> 63, 67, 104, 106, 244
{procedure-statement} 46, 42, 46, 53, 84, 98, 105, 110
processor 36
PROD 402
prod-bif 365
<prod-bif> 72
production-rule 25
<program> 63, 8, 12, 38, 39, 40, 63, 75, 159, 166, 167, 168, 176, 305
≪program-directory≫ 161, 161, 167
program-epilogue 169, 167, 171
program-run 14
≪program-state≫ 161, 38, 161, 167, 171, 172
<programmer-named-condition> 68, 68, 113, 163, 191, 193, 195, 196, 197
{programmer-named-condition} 48, 48, 113
prologue 174, 12, 13, 172, 173, 174, 187, 199
≪prologue-flag≫ 13, 163, 174, 175, 192, 209
promotable 222
promote-and-convert 222, 186, 189, 222, 227, 391
propagate-alignment 88, 87
proper for assignment 114, 152
<pseudo-variable> 72, 72, 142, 165, 220, 222, 226, 227, 228, 229, 231
<pseudo-variable-reference> 72, 7, 72, 136, 137, 142, 220, 226
PTR 62
PUT 3, 7, 50, 53, 54, 407, 414, 417
put-edit 283, 281
<put-file> 70, 7, 70, 280
{put-file} 50, 50
put-line 290, 280, 284
put-list 282, 281
put-page 291, 197, 280, 284, 290
<put-statement> 70, 7, 66, 70, 280
{put-statement} 50, 8, 43, 50, 81
<put-string> 70, 17, 70, 280, 281, 391
{put-string} 50, 50, 81
{putative-list-constant} 273, 272, 273

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

R

R 51
⟨R⟩ 395
RADIX 19, 35, 53, 411
⟨radix-arguments⟩ 53, 53
See also ⟨XX-arguments⟩
<radix-enquiry> 73, 151, 156, 316
See also ⟨XX-enquiry⟩
⟨radix-enquiry-name⟩ 53, 53, 151
See also ⟨XX-enquiry-name⟩
⟨radix-enquiry-reference⟩ 53, 52, 53
<radix-factor> 71, 71, 103, 277, 285
⟨radix-factor⟩ 58, 51, 58, 61, 79, 81, 103, 148, 279
raise-condition 191, 172, 182, 189, 191, 197, 205, 207, 223, 249, 256, 258, 260, 262, 269, 272, 278, 279, 280, 287, 289, 305, 306, 307, 308, 309, 310, 322, 331, 347, 385, 386, 389, 393
raise-io-condition 249, 17, 251, 256, 257, 258, 259, 260, 261, 262, 266, 267, 269, 271, 272, 276, 279, 280, 288, 290, 291, 292
⟨rangle⟩ 146, 146
⟨rangle-list⟩ 146
rdd 317, 319, 332
READ 256, 3, 7, 49, 256, 401, 407, 417
<read-statement> 69, 66, 69, 199, 248, 255
⟨read-statement⟩ 49, 43, 49
REAL 414, 417, 419
⟨real-constant⟩ 57, 46, 57, 79, 148, 390
<real-format> 71, 71, 275, 276
⟨real-format⟩ 51, 51
≪real-number≫ 164, 34, 164, 227, 306, 307, 308, 309, 310, 311, 315, 316, 349, 359
≪real-value≫ 164
34, 141, 148, 162, 164, 179, 181, 227, 228, 257, 277, 286, 287, 302, 305, 306, 307, 308, 309, 310, 315, 316, 344, 346, 351, 356, 359, 363, 364, 367, 368, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 397, 398
RECORD 9, 15, 19, 45, 48, 49, 95, 120, 403
<record> 9, 66, 69, 119, 120, 163, 248, 251, 252, 256, 258, 260, 262
≪record≫ 165, 19, 165, 247, 248, 255, 256, 257, 258, 259, 261, 262, 265, 266, 267
<record-condition> 68, 247, 249, 257, 261, 266
≪record-dataset≫ 165, 165, 247, 248, 252, 255, 258, 259, 261, 262, 266
≪record-list≫ 165, 253, 265, 267
⟨record-set⟩ 95, 95
RECURSIVE 46, 105, 406
<recursive> 63, 105, 172
REFER 44, 399, 418, 424
<refer-option> 64, 64, 122, 124, 126, 144, 153, 154, 155, 205, 208, 210, 211, 217, 235, 237
⟨refer-option⟩ 44, 44, 90, 122, 124, 126, 129
⟨reference⟩ 52, 7, 15, 45, 46, 47, 48, 49, 50, 52, 55, 56, 89, 91, 92, 99, 100, 101, 104, 107, 113, 115, 121, 124, 132, 134, 135, 136, 138
⟨reference-designator⟩ 64, 65, 76, 106, 107, 121, 124, 136
≪remote-block-state≫ 162, 162, 192, 211, 268, 297, 298
<remote-format> 71, 71, 115, 297
⟨remote-format⟩ 51, 51, 103, 115
reorganize 81, 80
REPEAT 47, 414, 416
<repeat-option> 67, 27, 27, 67, 177, 181
⟨repeat-option⟩ 47, 47
⟨repetition-factor⟩ 145, 144, 145
replace [instruction] 33
replace-concrete-designators 107, 76, 106, 111
replace-concrete-named-constant-designators 108, 106
replace-concrete-precision-designators 107, 76, 106, 111
RESERVED 45, 95, 97, 102, 121, 405
<reserved> 64, 121, 160, 168
result 30
RETURN 3, 48, 416
return [instruction] 33
<return-statement> 68, 66, 68, 118, 171, 175, 185, 189
⟨return-statement⟩ 48, 43, 48
≪returned-onsource-value≫ 163, 163, 175, 271, 276, 386, 389
≪returned-value≫ 163, 163, 189, 313
RETURNS 47, 95, 97, 98, 102, 125, 400, 418, 419
<returns-descriptor> 66, 40, 65, 66, 111, 125, 143, 144, 152, 155, 189, 208, 225, 241
⟨returns-descriptor⟩ 47, 45, 46, 47, 98, 99, 100, 111
REVERSE 402
reverse-bif 366
<reverse-bif> 72
REVERT .3, 8, 48, 403
<revert-statement> 68, 66, 68, 194, 195, 249
⟨revert-statement⟩ 48, 43, 48
REWRITE 260, 3, 19, 49, 260, 401
<rewrite-statement> 69, 66, 69, 259
⟨rewrite-statement⟩ 49, 43, 49
root-node 21, 10
ROUND 19, 402
round-bif 366, 306
<round-bif> 72
rounded-result 308, 306, 367
⟨rparen⟩ 146, 145, 146

S

⟨S⟩ 393, 394, 396, 397
⟨S-list⟩ 146
scalar 300
scalar data-type 302
≪scalar≫
114, 156, 165, 181, 186, 204, 205, 211, 216, 226, 227, 228, 229, 230, 231, 246, 271, 275, 276, 299, 300, 302, 315, 316, 317, 318, 319, 320, 321, 322, 326, 327, 328, 329, 330, 331, 332, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346 347, 348, 349, 350, 351, 352, 353, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

scalar-element 300, 200
scalar-elements-of-data-description 200, 200, 201, 207, 209, 212, 217, 223, 236, 237, 239, 240, 300
scalar-result. 302
scalar-result-type 302
scalar-value 302
<scale> 65
19, 35, 65, 74, 125, 129, 130, 147, 148, 178, 179, 226, 303, 304, 305, 317, 318, 319, 320, 321, 322, 323, 328, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 345, 346, 348, 349, 350, 351, 357, 358, 359, 360, 361, 362 365, 366, 367, 368, 369, 370, 372, 374, 375, 378, 379, 384, 385, 386, 390, 391
<scale-factor> 65, 65, 130, 141, 147, 157, 179, 229, 277, 286, 302, 303, 304, 305, 319, 330, 334, 339, 341, 346, 350, 351, 353, 355, 359, 360, 361, 365, 366, 367, 371, 372, 378, 379, 380, 384, 386, 387, 389, 390, 392, 397
⟨scale-factor⟩ 45, 45, 82, 130, 131
⟨scaled-digits-field⟩ 146, 146
<scope> 64, 64, 74, 119, 209
⟨scope⟩ 95, 95
SEARCH 409, 410
search-bif 367
<search-bif> 72
⟨secondary-case-letter⟩ 59, 59, 61, 62, 79
SELECT 3, 47, 406, 415
select-based-generation 234, 231
select-qualified-reference 238, 213, 232, 241, 242
⟨select-statement⟩ 47, 43, 47, 109
select-subscripted-reference 239, 232, 241, 242
<selection> 67, 66, 67, 109, 175, 182
⟨selection⟩ 43, 42, 43, 80, 103, 109, 116
SEQL 62
SEQUENTIAL 20, 45, 49, 62, 95, 120, 407
<sequential> 66, 69, 119, 120, 163, 251, 252, 256, 260, 262, 266
≪sequential-dataset≫ 165, 165, 252
⟨sequential-set⟩ 95, 95
SET 15, 48, 49, 400
<set-option> 69, 69, 114, 205
⟨set-option⟩ 48, 48, 114
set-storage 223, 186, 199, 222, 225, 227, 230, 231, 257
set-union 225, 223
⟨sign⟩ 146, 145, 146
sign-bif 368
<sign-bif> 72
SIGNAL 3, 48, 403
<signal-statement> 68, 66, 68, 191, 193, 249
⟨signal-statement⟩ 48, 43, 48
<signed-integer> 73, 66, 73, 111, 245
⟨signed-integer⟩ 44, 44, 80, 85, 86, 111
⟨signs⟩ 146, 146
simple component 21
simple subnode 21
simple subtree 21
⟨simple-bit-string-constant⟩ 58, 52, 57, 58, 79, 147, 271, 279
⟨simple-character-string-constant⟩ 58, 46, 52, 57, 58, 77, 79, 147, 271, 279
⟨simple-statement-name⟩ 44, 42, 43, 44, 53, 80, 83, 84, 85, 86, 89, 98, 110, 111, 114
⟨simple-string-constant⟩ 52, 46, 52
simply contain 21, 10
SIN 53, 54
sin-bif 369
<sin-bif> 72
sind-bif 369
<sind-bif> 72
<single-closing> 69, 69, 254
⟨single-closing⟩ 49, 49
⟨single-closing-commalist⟩ 49
<single-closing-list> 69
<single-opening> 69, 17, 69, 119, 250
⟨single-opening⟩ 49, 49
⟨single-opening-commalist⟩ 49, 119
<single-opening-list> 69, 119
sinh-bif 369
<sinh-bif> 72
SIZE 44, 403
<size-condition> 68, 192, 196, 287, 305, 385, 386, 393
SIZETO 50, 407
<sizeto-option> 70, 69, 70, 255, 265
⟨sizeto-option⟩ 50, 49, 50
SKIP 291, 50, 51, 81, 269, 274, 280, 283, 284, 285, 289, 290, 414
<skip-format> 71, 71, 274, 284
⟨skip-format⟩ 51, 51, 81
<skip-option> 70, 70, 269, 280
⟨skip-option⟩ 50, 50, 81
SNAP 19, 44, 403
<snap> 68, 118, 163, 195, 196
SOME 402
some-bif 370
<some-bif> 72
source-type 384
<space-format> 71, 71, 274, 284
⟨space-format⟩ 51, 51
<spec> 67, 67, 177, 178, 180
⟨spec⟩ 47, 47, 56
<spec-designator> 162, 178, 180
sqrt-bif 370
<sqrt-bif> 72
⟨statement-condition-prefix⟩ 44, 42, 43, 44, 105, 106, 112
≪statement-control≫ 162, 13, 36, 38, 162, 172, 173, 174, 175, 187, 189, 268
<statement-name> 66, 66, 71, 110, 111, 114, 117, 158, 185, 244, 245
⟨statement-name⟩ 44, 42, 43, 44, 47, 80, 83, 84, 85, 86, 89, 110, 111, 117, 118
STATIC 45, 74, 95, 97, 102, 121, 400, 405, 407, 408, 417
<static> 64, 74, 121, 152, 159, 168, 209, 232
≪static-directory≫ 161, 161, 167, 168, 209, 217
≪static-directory-entry≫ 161, 161, 168, 169, 209
≪static-directory-entry-list≫ 161, 169
step 31
STOP 3, 48
stop-program 172, 172
<stop-statement> 66, 171, 172, 175

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

⟨stop-statement⟩ 48, 43, 48
STORAGE 19, 48, 112, 207, 403
<storage-class> 64, 64, 74, 121, 159
⟨storage-class⟩ 95, 95
<storage-condition> 68, 163, 197, 203, 207
⟨storage-index⟩ 165, 165, 201, 202, 207, 211, 232, 237, 238, 243, 257, 294, 326
⟨storage-index-list⟩ 164, 165, 200, 201, 202, 207, 211, 214, 216, 224, 227, 229, 232, 235, 236, 237, 238, 240, 241, 242, 243, 295, 300, 324, 383
<storage-type> 64, 64, 74, 121, 209, 231, 232, 240
⟨storage-type⟩ 95, 95
STREAM 20, 45, 49, 95, 120
<stream> 9, 24, 25, 66, 69, 119, 120, 163, 248, 251, 252, 269, 280, 289
⟨stream-dataset⟩ 166, 165, 166, 248, 252, 391
⟨stream-item⟩ 166, 166, 247, 248, 271, 272, 274, 275, 278, 279, 282, 283, 288, 289, 290, 291, 292
⟨stream-item-list⟩ 166, 247, 253, 271, 272, 275, 282, 283, 284, 285, 286, 288, 289, 290
⟨stream-set⟩ 95, 95
STRING 305, 50
<string> 65, 65, 66, 108, 125, 128, 134, 140, 152, 155, 202, 207, 210, 217, 226, 229, 230, 236, 245, 271, 275, 303, 304, 323, 353, 356, 366, 371, 383, 384
⟨string⟩ 95, 95
string-bif 371
<string-bif> 72
⟨string-constant⟩ 52, 52
<string-format> 71, 71, 275, 276
⟨string-format⟩ 51, 51
⟨string-io-control⟩ 162, 162, 268, 270, 272, 278, 281, 289
⟨string-limit⟩ 162, 162, 281, 289
⟨string-or-picture-symbol⟩ 58, 58, 147, 148, 149, 273
⟨string-or-picture-symbol-list⟩ 58, 145, 147
<string-pv> 72, 228
⟨string-symbol-or-linemark⟩ 273, 273
⟨string-symbol-or-linemark-list⟩ 272, 273
<string-type> 65, 65, 124, 128, 228, 230, 271, 303, 304, 305, 321, 323, 342, 347, 353, 356, 359, 366, 371, 373
STRINGRANGE 416
STRINGSIZE 416
STRUCTURE 41, 45, 80, 87, 88, 89, 95, 96, 122, 123, 417
⟨structure-aggregate-type⟩ 165, 165, 210, 299
<structure-data-description> 64, 64, 123, 134, 135, 144, 153, 159, 200, 201, 208, 212, 213, 225, 235, 236, 237, 238, 295, 299, 301
⟨structure-qualification⟩ 52, 52, 137
⟨structure-qualification-list⟩ 52
subnode 21
subnode-type sequence 28, 26
<subroutine-reference> 73, 68, 73, 104, 120, 134, 136, 137, 143, 144, 184, 185
<subscript> 72, 72, 138, 246, 344, 345, 351, 355
⟨subscript⟩ 52, 27, 52, 135, 137, 138, 140
⟨subscript-commalist⟩ 27, 52, 135, 137, 138, 140
<subscript-list> 72, 138, 217, 232, 239, 241, 242, 246, 344, 345, 351, 355
SUBSCRIPTRANGE 416
substr-bif 371, 221
<substr-bif> 72
<substr-pv> 72, 221, 229
substring, 305
SUBTRACT 402
<subtract> 71, 132, 319
subtract-bif 372, 305, 306
<subtract-bif> 72
subtree 21
⟨succeed⟩ 251, 253
SUM 402
sum-bif 373
<sum-bif> 72
⟨suppression⟩ 393, 393, 394, 395, 396
⟨suppression-type⟩ 393, 393, 394, 395, 396, 397
⟨symbol⟩ 59, 19, 59, 78, 92, 147, 149, 164, 166, 168, 247, 251, 256, 272, 273, 274, 275, 277, 278, 279, 281, 282, 283, 285, 286, 289, 292, 323, 341, 345, 347, 375, 392, 393, 394, 396, 397
⟨symbol-list⟩ 14, 25, 40, 61, 73, 75, 76, 77, 78, 98, 347
syntactic-expression 28
syntactic-unit 28
syntax 26
SYSIN 81, 92, 400, 403, 418
SYSPRINT 8, 81, 92, 251, 400, 403, 418
SYSTEM 44, 118, 403
system-action 196, 191, 196
<system-action> 68, 118, 163, 194, 196

T

TAB 289, 51, 283, 284, 417
<tab-format> 71, 71, 274, 284
⟨tab-format⟩ 51, 51, 81
TALLY 409
tally-bif 373
<tally-bif> 72
tan-bif 374
<tan-bif> 72
tand-bif 375
<tand-bif> 72
tanh-bif 375
<tanh-bif> 72
<target-reference> 72, 7, 16, 63, 69, 72, 104, 113, 120, 134, 136, 137, 157, 219, 220, 226
target-type 384
terminal 26
terminate [instruction] 33
<test> 68, 68, 116, 117, 184
test-attribute-consistency 94, 93, 98
test-connectedness 232, 186, 219, 220, 229, 231, 242, 263, 336, 344, 352, 371, 379
test-constraints 157, 157, 158
test-descriptor-extent-expressions 113, 111
test-enablement 192, 191, 192, 193
test-invalid-duplicates 96, 94
test-matching 144, 143, 243
test-model-number 310, 307, 308, 309, 311

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

test-name-in-description 101, 99, 100
test-spec 180, 180, 181
test-string-class 233, 217, 228, 232, 235, 242, 371
test-union-matching 225, 222, 225
test-when-enablement 182, 182
test-when-matching 183, 182
⟨text-name⟩ 58, 18, 58, 78
THEN 43, 54, 400
then [instruction] 33
<then-unit> 68, 68, 116, 117, 184
TIME 18, 408
time-bif 375
<time-bif> 72, 155, 333
TITLE 49
<title-option> 69, 69, 250, 253
⟨title-option⟩ 49, 49
TO 7, 47, 56, 74, 414, 416
to obtain [instruction] 33
<to-by> 67, 67
⟨to-by⟩ 47, 47, 56
<to-option> 67, 67, 177, 178, 179, 181
⟨to-option⟩ 47, 47, 56
≪to-value≫ 162, 162, 179, 180
translate 75, 40, 152
translate-bif 376
<translate-bif> 72
translation-phase 40, 38, 39
≪translation-state≫ 39, 39, 40, 75
translation-unit 4
⟨translation-unit⟩ 42, 41, 42, 45, 53, 75, 80
TRANSMIT 19, 48, 403
<transmit-condition> 17, 68, 248, 249
tree 21
TRIM 409
trim-bif 377
<trim-bif> 72
trim-dd 139, 139
trim-group-control 188, 188, 191, 268
trim-io-control 268, 187, 268, 269, 280
≪true≫ 12, 94, 95, 96, 98, 144, 157, 158, 177, 178, 180, 181, 182, 183, 184, 186, 190, 219, 220, 222, 225, 229, 231, 232, 233, 241, 242, 243, 263, 294, 307, 308, 309, 310, 311, 320, 321, 330, 336, 344, 352, 371, 379
trunc-bif 379, 306
<trunc-bif> 72
type [of a node] 21

U

UFL 62
UNAL 62
UNALIGNED 15, 16, 45, 62, 88, 89, 95, 97, 123, 400, 417
<unaligned> 65, 123
unconstrained-result 310, 306, 385
≪undefined≫ 18, 162, 164, 173, 199, 204, 207, 210, 221, 223, 226, 227, 230, 231, 243, 244, 245, 246, 248, 253, 257, 260, 262, 266, 270, 273, 282, 283, 293, 313, 353, 379, 381
UNDEFINEDFILE 9, 48, 62, 413
<undefinedfile-condition> 17, 68, 249, 251, 256, 258, 260, 262, 269, 280
UNDERFLOW 35, 44, 62, 403
<underflow-condition> 68, 197, 307, 308, 309, 310
UNDF 62
UNION 45, 87, 88, 89, 95, 96, 122, 123, 406
≪union≫ 164, 164, 199, 211, 223, 225
≪union-aggregate-type≫ 165, 165, 299
≪union-alternative≫ 165, 165, 199, 211, 212, 226
≪union-alternative-list≫ 164, 212
<union-data-description> 64, 64, 65, 123, 134, 135, 144, 153, 157, 159, 199, 200, 201, 208, 210, 211, 212, 222, 225, 228, 235, 238, 242, 295, 299, 300, 301, 352, 371
≪union-list≫ 164, 199, 210, 211, 212, 223
unique-name [of a node] 23, 21
⟨unit⟩ 42, 42, 53, 83, 84, 85, 86, 91, 110, 114
⟨unit-list⟩ 42, 43, 53, 85, 92, 106, 108, 110
UNSPEC 18, 230, 379, 410
unspec-bif 379, 221
<unspec-bif> 72
<unspec-pv> 72, 221, 231
⟨unsubscripted-reference⟩ 52, 44, 52, 89, 101, 104, 126, 134
UNTIL 47, 407, 416
<until-option> 67, 67, 108, 180, 190
⟨until-option⟩ 47, 47, 108
UPDATE 45, 49, 95, 120, 407
<update> 66, 69, 119, 120, 163, 251, 252, 256, 258, 260, 262
<upper-bound> 64, 27, 64, 122, 200, 208, 222, 238, 240, 243, 244, 246, 295, 300, 301, 346, 351
⟨upper-bound⟩ 44, 44, 86, 121, 122
use [of an allocation-unit] 18

V

⟨V⟩ 392, 395
valid-bif 380, 397
<valid-bif> 72
validate-automatic-declaration 153, 152
validate-based-declaration 153, 152, 208
validate-concrete-declarations 101, 81
validate-declaration 152, 152
validate-defined-declaration 154, 152
validate-descriptor 155, 152
validate-external-declaration 159, 159
validate-external-unit 152, 76
validate-field-of-pictured-value 398, 61, 397
validate-input-format 276, 275, 276, 397
validate-leave-statements 158, 152, 191
validate-numeric-pictured-value 397, 276, 380, 386, 389
validate-offset 391, 383
validate-parameter-declaration 154, 152
validate-program 159, 40
validate-static-declaration 154, 152
validate-translation-unit 80, 41, 45, 75

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

VALUE 45, 93, 94, 95, 102, 108, 119, 120
≪value-and-type≫ 389, 148, 275, 276, 277, 286, 382, 385, 389, 390, 391
value-of-evaluated-target 221, 180, 181
value-of-generation 202, 207, 220, 221, 227, 230, 313, 353, 371, 383
value-of-storage-index 202, 202, 238
<value-reference> 72
16, 63, 64, 68, 69, 71, 72, 74, 104, 107, 113, 114, 115, 120, 131, 132, 134, 135, 136, 137, 139, 141, 143, 150, 151, 152, 154, 155, 157, 172, 185, 187, 194, 204, 205, 218, 219, 226, 228, 229, 230, 234, 250, 254, 269, 280, 297
313, 336, 344, 345, 351, 352, 355, 359, 362, 365, 371, 379
VAR 62
VARIABLE 45, 95, 97, 119
<variable> 64, 13, 64, 74, 119, 120, 121, 134, 136, 157, 174, 205, 213, 235, 238, 240, 242, 315
≪variable≫ 95, 95
<variable-reference> 72
7, 15, 16, 17, 31, 64, 65, 67, 69, 70, 72, 74, 104, 107, 114, 134, 136, 137, 138, 139, 143, 144, 153, 154, 156, 157, 159, 177, 178, 185, 186, 200, 204, 205, 206, 216, 217, 218, 219, 220, 225, 226, 227, 228, 229, 230, 231, 232
234, 235, 240, 241, 242, 257, 263, 264, 265, 281, 293, 294, 313, 336, 344, 345, 346, 351, 352, 355, 359, 362, 363, 365, 371, 379, 381, 383, 391
VARYING 45, 62, 95, 124, 418
<varying> 65, 124, 139, 230, 383, 384
VERIFY 410
verify-bif 380
<verify-bif> 72

W

WHEN 43, 406
<when-all-values> 67, 67, 109, 110, 182, 183
<when-any-values> 67, 67, 109, 110, 182, 183
≪when-clause≫ 43, 43, 109
≪when-statement≫ 43, 43, 109
≪when-statement-list≫ 43, 109
<when-unit> 67, 67, 109, 182, 183, 188, 192
<when-unit-designator> 162, 182, 188, 192
<when-unit-list> 67, 109, 110
WHILE 47, 56, 416
<while-option> 67, 67, 108, 177, 180
≪while-option≫ 47, 47, 56, 108
WRITE 259, 3, 19, 49, 259, 401, 407
<write-statement> 70, 66, 70, 258
≪write-statement≫ 49, 43, 49

X

X 51
<x-condition> 112
<xor> 71, 132
≪XX-arguments≫ 149, 151
See also ≪declaresize-arguments≫
See also ≪epsilon-arguments≫
See also ≪maxval-arguments≫
See also ≪minval-arguments≫
See also ≪null-arguments≫
See also ≪radix-arguments≫
≪XX-enquiry≫ 151
See also <declaresize-enquiry>
See also <epsilon-enquiry>
See also <maxval-enquiry>
See also <minval-enquiry>
See also <null-enquiry>
See also <radix-enquiry>
≪XX-enquiry-name≫ 149
See also ≪declaresize-enquiry-name≫
See also ≪epsilon-enquiry-name≫
See also ≪maxval-enquiry-name≫
See also ≪minval-enquiry-name≫
See also ≪NAME-enquiry-name≫
See also ≪null-enquiry-name≫
See also ≪radix-enquiry-name≫

Z

≪Z≫ 393, 394
≪Z-list≫ 146
ZDIV 62
≪zero-bit≫ 32, 149, 164, 285, 318, 320, 321, 322, 323, 327, 328, 329, 330, 332, 348, 349, 353, 370, 380, 383, 386, 388, 389
ZERODIVIDE 44, 62
<zerodivide-condition> 68, 196, 322, 347

Numerics

≪0≫ 386, 388, 393, 394, 395
≪1≫ 121, 386, 388
≪9≫ 393
≪9-list≫ 146

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

UDC 681.3.06:519.682:800.92

Descriptors: data processing, programming (computers), artificial languages, program..ning languages, PL/I.

Price based on 443 pages

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale