Enterprise PL/I for z/OS
PL/I for AIX
WebSphere Developer for System z PL/I for Windows

言語解説書

SC88-9126-04
(英文原典：SC27-1460-06)

Enterprise PL/I for z/OS
PL/I for AIX
WebSphere Developer for System z PL/I for Windows

言語解説書

SC88-9126-04
(英文原典：SC27-1460-06)

お願い

本書および本書で紹介する製品をご使用になる前に、 579 ページの『特記事項』に記載されている情報をお読みください。

本書は、Enterprise PL/I for z/OS バージョン 3 リリース 6 (5655-H31)、IBM PL/I for AIX V2.0.0.0、および WebSphere Developer for System z バージョン 7、PL/I for Windows、ならびに新しい版または TNL で明記されていない限り、以降のすべてのリリースに適用されます。製品のレベルに合った正しい版をご使用ください。

IBM 発行のマニュアルに関する情報のページ

`http://www.ibm.com/jp/manuals/main/mail.html`

こちらから、日本語版および英語版のオンライン・ライブラリーをご利用いただけます。また、マニュアルに関するご意見やご感想を、上記ページよりお送りください。今後の参考にさせていただきます。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： SC27–1460–06
Enterprise PL/I for z/OS
PL/I for AIX
WebSphere Developer for System z PL/I for Windows
Language Reference

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2006.11

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、平成角ゴシック体™W7

©International Business Machines Corporation 1998,2006. All rights reserved.

© Copyright IBM Japan 2006

目次

表 ix

図 xi

第 1 章 本書について 1
本書で用いられる表記規則. 1
セマンティクス 4
工業規格. 4
本リリースにおける機能強化 4
V3R5 からの機能拡張 5
V3R4からの機能拡張. 5
V3R3 からの機能拡張 5
V3R2 からの機能拡張 5
V3R1 からの機能拡張 6

第 2 章 プログラムのエレメント 9
1 バイト文字セット 9
SBCS ステートメント・エレメント 13
ステートメント 16
グループ 19
2 バイト文字セット 19

第 3 章 データ・エレメント. 23
データ項目 23
データ・タイプとその属性 24
計算データ・タイプとその属性. 29

第 4 章 式および参照 55
計算の順序 57
ターゲット 58
演算式 59
配列式 74
構造式 75
制限付き式 76

第 5 章 データ変換 77
計算データ変換の組み込み関数. 78
ストリングの長さの変換 79
算術値の精度の変換 80
モードの変換. 80
その他のデータ属性の変換 80
ソースからターゲットへの変換規則 82
例 90

第 6 章 プログラムの編成. 93
プログラム 93
ブロック 95
パッケージ 96
プロシージャー 99
サブルーチン 112
組み込みサブルーチン 113
関数 113
プロシージャーへの引数の引き渡し 116
開始ブロック 119
入り口データ 120
入り口呼び出しまたは入り口値 133
CALL ステートメント 133
RETURN ステートメント 134
OPTIONS オプションとその属性 135
RETURNS オプションとその属性. 144

第 7 章 タイプ定義 147
ユーザー定義のタイプ (別名) 147
序数の定義 148
タイプ付き構造体と共用体の定義. 150
タイプ変数の宣言 152
タイプ付き構造体修飾 153
序数の使用 155
タイプ付き関数. 158

第 8 章 データ宣言 159
明示宣言 159
暗黙宣言 162
宣言の有効範囲. 163
RESERVED 属性 170
データの位置合わせ 171
属性のデフォルト 175
配列 181
構造体. 185
共用体. 186
構造体/共用体の修飾 187
LIKE 属性 188
例 189
NOINIT 属性 190
集合体の組み合わせおよびマッピング 191

第 9 章 ステートメントとディレクティブ 201
ALLOCATE ステートメント 201
代入ステートメントと複合代入ステートメント . . 202
ATTACH ステートメント 207
BEGIN ステートメント 207
CALL ステートメント 207
CLOSE ステートメント 207
DECLARE ステートメント 208
DEFINE ALIAS ステートメント 208
DEFINE ORDINAL ステートメント 208
DEFINE STRUCTURE ステートメント 208
DEFAULT ステートメント 208
DELAY ステートメント. 208
DELETE ステートメント 209
DETACH ステートメント 209

DISPLAY ステートメント 209
DO ステートメント 209
END ステートメント 221
ENTRY ステートメント 222
EXIT ステートメント 222
FETCH ステートメント 222
FLUSH ステートメント 222
FORMAT ステートメント 222
FREE ステートメント 222
GET ステートメント 222
GO TO ステートメント 223
IF ステートメント 223
%INCLUDE ディレクティブ 225
ITERATE ステートメント 226
LEAVE ステートメント 226
%LINE ディレクティブ 227
LOCATE ステートメント 227
%NOPRINT ディレクティブ 227
%NOTE ディレクティブ 228
ヌル・ステートメント 228
ON ステートメント 228
OPEN ステートメント 229
%OPTION ディレクティブ 229
OTHERWISE ステートメント 229
PACKAGE ステートメント 229
%PAGE ディレクティブ 230
%POP ディレクティブ 230
%PRINT ディレクティブ 230
PROCEDURE ステートメント 230
%PROCESS ディレクティブ 231
*PROCESS ディレクティブ 231
%PUSH ディレクティブ 231
PUT ステートメント 232
READ ステートメント 232
RELEASE ステートメント 232
RESIGNAL ステートメント 232
RETURN ステートメント 232
REVERT ステートメント 233
REWRITE ステートメント 233
SELECT ステートメント 233
SIGNAL ステートメント 235
%SKIP ディレクティブ 235
STOP ステートメント 235
WAIT ステートメント 235
WHEN ステートメント 235
WRITE ステートメント 236
%XINCLUDE ステートメント 236

第 10 章 ストレージ制御 237
ストレージのクラス、割り振り、および割り振り解除 237
静的ストレージとその属性 239
自動ストレージとその属性 239
被制御ストレージとその属性 241
基底付きストレージとその属性 245
区域データとその属性 255
リスト処理 259
ASSIGNABLE 属性と NONASSIGNABLE 属性 . . 260
NORMAL 属性と ABNORMAL 属性 261
BIGENDIAN 属性と LITTLEENDIAN 属性 . . . 261
HEXADEC 属性と IEEE 属性 262
CONNECTED 属性と NONCONNECTED 属性 . . 263
DEFINED 属性と POSITION 属性 264
INITIAL 属性 270

第 11 章 入出力 277
データ・セット 278
ファイル 279
ファイルのオープンとクローズ 285
SYSPRINT および SYSIN 291

第 12 章 レコード単位データ伝送 . . . 293
データ伝送 293
データ伝送ステートメント 294
データ伝送ステートメントのオプション 297
処理モード 301

第 13 章 ストリーム指向データ伝送 303
データ伝送ステートメント 304
データ伝送ステートメントのオプション 305
データ・リスト項目の伝送 311
データ・ディレクティブのデータ指定 312
データ・ディレクティブ・データの制限 313
編集ディレクティブのデータ指定 317
リスト・ディレクティブ・データ指定 321
PRINT 属性 324
ストリーム入出力での DBCS データ 326

第 14 章 編集ディレクティブのフォーマット項目 327
A フォーマット項目 327
B フォーマット項目 328
C フォーマット項目 329
COLUMN フォーマット項目 329
E フォーマット項目 330
F フォーマット項目 332
G フォーマット項目 334
L フォーマット項目 335
LINE フォーマット項目 335
P フォーマット項目 335
PAGE フォーマット項目 336
R フォーマット項目 336
SKIP フォーマット項目 337
V フォーマット項目 338
X フォーマット項目 338

第 15 章 ピクチャー指定文字 341
ピクチャー反復因数 341
文字データ用のピクチャー文字 342
数字データ用のピクチャー文字 343

第 16 章 条件処理 357

条件接頭語 357
ON ユニット 360
REVERT ステートメント 363
SIGNAL ステートメント 364
RESIGNAL ステートメント 364
複数条件 365
CONDITION 属性 365

第 17 章 条件 367
ANYCONDITION 条件 367
AREA 条件 369
ATTENTION 条件 369
CONDITION 条件 370
CONVERSION 条件 371
ENDFILE 条件 373
ENDPAGE 条件 374
ERROR 条件 375
FINISH 条件. 375
FIXEDOVERFLOW 条件. 376
INVALIDOP 条件 377
KEY 条件 378
NAME 条件 378
OVERFLOW 条件 379
RECORD 条件 380
SIZE 条件 381
STORAGE 条件 382
STRINGRANGE 条件. 382
STRINGSIZE 条件. 383
SUBSCRIPTRANGE 条件 384
TRANSMIT 条件 385
UNDEFINEDFILE 条件 386
UNDERFLOW 条件 387
ZERODIVIDE 条件 387

第 18 章 マルチスレッド化機能 389
スレッドの作成. 390
ATTACH ステートメント 390
スレッドの終了. 391
スレッドの完了の待機 392
スレッドの切り離し 392
条件処理 393
タスク・データとタスク属性 393
スレッド間のデータの共用 394
スレッド間のファイルの共用 394

第 19 章 組み込み関数、疑似変数、およびサブルーチン 395
組み込み関数、疑似変数、および組み込みサブルーチンの宣言および呼び出し 398
組み込み関数、疑似変数、および組み込みサブルーチンの引数の指定 400
数学関数の精度. 401
組み込み関数のカテゴリー 401
ABS 415
ACOS 415
ACOSF 415
ADD 415
ADDR 416
ADDRDATA. 417
ALL 417
ALLOCATE 417
ALLOCATION 418
ALLOCSIZE 418
ANY 418
ASIN 418
ASINF. 419
ATAN 419
ATAND 420
ATANF 420
ATANH 420
AUTOMATIC 421
AVAILABLEAREA 421
BINARY 421
BINARYVALUE 422
BIT. 422
BITLOCATION 422
BOOL 423
BYTE 423
CDS 423
CEIL 424
CENTERLEFT 424
CENTRELEFT 425
CENTERRIGHT. 425
CENTRERIGHT. 425
CHARACTER 426
CHARGRAPHIC 426
CHARVAL 427
CHECKSTG 428
COLLATE 428
COMPARE 428
COMPLEX 429
CONJG 430
COPY 430
COS 430
COSD 431
COSF 431
COSH 431
COUNT 431
CS 432
CURRENTSIZE 433
CURRENTSTORAGE 434
DATAFIELD. 434
DATE 435
DATETIME 435
DAYS 435
DAYSTODATE 436
DAYSTOSECS 437
DECIMAL 437
DIMENSION. 438
DIVIDE 438
EDIT 438
EMPTY 439

ENDFILE 440
ENTRYADDR 440
ENTRYADDR 疑似変数 440
EPSILON 441
ERF 441
ERFC 441
EXP 441
EXPF 442
EXPONENT 442
FILEDDINT 442
FILEDDTEST 443
FILEDDWORD 443
FILEID 444
FILEOPEN 445
FILEREAD 445
FILESEEK 445
FILETELL 446
FILEWRITE 446
FIXED 446
FLOAT 447
FLOOR 447
GAMMA 447
GETENV 448
GRAPHIC 448
HANDLE 449
HBOUND 449
HEX 450
HEXIMAGE 451
HIGH 451
HUGE 451
IAND 452
IEOR 452
IMAG 453
IMAG 疑似変数 453
INDEX 453
INOT 454
IOR 454
ISIGNED 455
ISLL 455
ISMAIN 456
ISRL 456
IUNSIGNED 457
LBOUND 457
LEFT 458
LENGTH 458
LINENO 459
LOCATION 459
LOG 460
LOGF 460
LOGGAMMA 460
LOG2 461
LOG10 461
LOG10F 461
LOW 461
LOWERCASE 462
LOWER2 462
MAX 463
MAXEXP 463
MAXLENGTH 464
MEMINDEX 465
MEMSEARCH 466
MEMSEARCHR 466
MEMVERIFY 467
MEMVERIFYR 468
MIN 469
MINEXP 469
MOD 470
MPSTR 471
MULTIPLY 472
NULL 472
OFFSET 472
OFFSETADD 473
OFFSETDIFF 473
OFFSETSUBTRACT 473
OFFSETVALUE 474
OMITTED 474
ONCHAR 474
ONCHAR 疑似変数 474
ONCODE 475
ONCONDCOND 475
ONCONDID 475
ONCOUNT 476
ONFILE 476
ONGSOURCE 477
ONGSOURCE 疑似変数 477
ONKEY 477
ONLOC 478
ONSOURCE 478
ONSOURCE 疑似変数 479
ONSUBCODE 479
ONWCHAR 480
ONWCHAR 疑似変数 480
ONWSOURCE 480
ONWSOURCE 疑似変数 481
ORDINALNAME 481
ORDINALPRED 481
ORDINALSUCC 482
PACKAGENAME 482
PAGENO 482
PICSPEC 482
PLACES 483
PLIASCII 484
PLICANC 484
PLICKPT 485
PLIDELETE 485
PLIDUMP 485
PLIEBCDIC 485
PLIFILL 486
PLIFREE 486
PLIMOVE 487
PLIOVER 487
PLIREST 488

PLIRETC 488
PLIRETV 488
PLISAXA 488
PLISAXB 489
PLISRTA 489
PLISRTB 490
PLISRTC 490
PLISRTD 490
POINTER 490
POINTERADD 491
POINTERDIFF 492
POINTERSUBTRACT 492
POINTERVALUE 492
POLY 492
PRECISION 493
PRED 493
PRESENT 494
PROCEDURENAME 494
PROD 494
PUTENV 495
RADIX 495
RAISE2 495
RANDOM 496
RANK 496
REAL 496
REAL 疑似変数 497
REM 497
REPATTERN 497
REPEAT 498
REPLACEBY2 498
REVERSE 499
RIGHT 499
ROUND 500
SAMEKEY 501
SCALE 501
SEARCH 501
SEARCHR 503
SECS 503
SECSTODATE 504
SECSTODAYS 505
SIGN 505
SIGNED 505
SIN 506
SIND 506
SINF 506
SINH 506
SIZE 507
SOURCEFILE 508
SOURCELINE 508
SQRT 508
SQRTF 509
STORAGE 509
STRING 509
STRING 疑似変数 510
SUBSTR 511
SUBSTR 疑似変数 511
SUBTRACT 512
SUCC 512
SUM 512
SYSNULL 512
SYSTEM 513
TALLY 513
TAN 514
TAND 514
TANF 514
TANH 514
THREADID 515
TIME 515
TINY 515
TRANSLATE 516
TRIM 517
TRUNC 517
TYPE 518
TYPE 疑似変数 518
UNALLOCATED 518
UNSIGNED 519
UNSPEC 519
UNSPEC 疑似変数 521
UPPERCASE 521
VALID 522
VALIDDATE 522
VARGLIST 523
VARGSIZE 523
VERIFY 524
VERIFYR 524
WCHARVAL 525
WEEKDAY 525
WHIGH 526
WIDECHAR 526
WLOW 527
XMLCHAR 527
Y4DATE 528
Y4JULIAN 529
Y4YEAR 530

第 20 章 タイプ付き関数 531
タイプ付き関数の呼び出し 531
タイプ付き関数の引数の指定 531
タイプ付き関数の要旨 532
BIND 532
CAST 532
FIRST 533
LAST 533
NEW 534
RESPEC 534
SIZE 534

第 21 章 プリプロセッサーの機能 . . . 535
プリプロセッサー・オプション 536
プリプロセッサー走査 537
プリプロセッサー変数とデータ・エレメント . . . 539
プリプロセッサー参照とプリプロセッサー式 . . . 540

プリプロセッサー名の有効範囲 541
プリプロセッサー・プロシージャー 541
プリプロセッサー組み込み関数 547
プリプロセッサー・ステートメント 558
プリプロセッサーの例 569

付録. 制限値 575

特記事項. 579
商標 580

参考文献. 581
Enterprise PL/I 資料 581
PL/I for MVS & VM 581
z/OS 言語環境プログラム 581
CICS Transaction Server 581
DB2 UDB for z/OS 582
DFSORT 582
IMS/ESA 582
z/OS MVS 582
z/OS UNIX システム・サービス 582
z/OS TSO/E 582
z/Architecture. 583
Unicode および文字表現 583

用語集 585

索引 601

表

1. 英字と同等の値 10
2. 10 進数と同等の値 11
3. 特殊文字と同等の値. 11
4. 複合記号の説明 12
5. 区切り文字. 14
6. 演算子 15
7. 定数タイプによる属性の分類. 28
8. 変数タイプによる属性の分類. 29
9. コード化算術データ属性の省略形 30
10. FIXED BINARY SIGNED データ・ストレージ所要量 33
11. FIXED BINARY UNSIGNED データ・ストレージ所要量 33
12. ストリング・データ属性の省略形 37
13. 1 つまたは複数の FLOAT オペランドの算術演算の結果 63
14. RULES(ANS) における、2 つのスケールのない FIXED オペランド間の算術演算の結果 . . . 64
15. RULES(ANS) における、2 つのスケールされた FIXED オペランド間の算術演算の結果 . . . 65
16. RULES(IBM) における、2 つの FIXED オペランド間の算術演算の結果 65
17. 固定小数点除算と定数式の比較 66
18. 累乗演算の特殊事例. 67
19. ビット演算子 68
20. ビット演算の例 68
21. 演算の優先順位と変換の手引き 73
22. CEIL (n*3.32) の値と CEIL (n/3.32) の値の表 81
23. 序数処理組み込み関数 156
24. タイプ付き関数 158
25. データ宣言の有効範囲 164
26. 入り口およびラベル宣言の有効範囲 165
27. ハーフワード、ワード、およびダブルワードの規定境界の位置合わせ 171
28. 位置合わせ要件 172
29. デフォルトの算術精度 176
30. 複合代入演算子 203
31. 代替ファイル属性 280
32. データ伝送のタイプによる属性 280
33. PL/I ファイルを宣言するときの属性 281
34. 暗黙オープンによって暗黙に定義される属性 288
35. 組み合わせ後の属性と暗黙に定義される属性 289
36. PRINT ファイルのオプションとフォーマット項目 325
37. 文字ピクチャー指定の例 342
38. 数字と小数点文字の例 345
39. ゼロ抑制文字の例 346
40. 挿入文字の例 348
41. 符号と通貨記号の例 352
42. T、I、および R ピクチャー文字の解釈 353
43. 貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字の例 354
44. 指数文字の例 355
45. スケール因数文字の例 355
46. 条件のクラスと状況 358
47. 算術組み込み関数 401
48. 配列処理組み込み関数 402
49. バッファー管理組み込み関数 403
50. 条件処理組み込み関数 403
51. 日付/時刻組み込み関数 405
52. 日付/時刻のパターン 406
53. 浮動小数点の照会組み込み関数 406
54. 浮動小数点演算組み込み関数 407
55. 入出力組み込み関数 407
56. 整数演算組み込み関数 407
57. 数学組み込み関数 408
58. その他の組み込み関数 409
59. 序数処理組み込み関数 410
60. 精度処理組み込み関数 410
61. 組み込み疑似変数 411
62. ストレージ制御組み込み関数 411
63. ストリング処理組み込み関数 413
64. 組み込みサブルーチン 414
65. UNSPEC によって戻されるビット・ストリングの長さ 519
66. タイプ付き関数 532
67. 言語エレメントの制限 575
68. マクロ機能の制限 577

図

1. 名前付き定数 51
2. PL/I アプリケーション構造 94
3. パッケージ・ステートメント. 98
4. パラメーターの付いた配列引数 104
5. 有効な呼び出しステートメントおよび無効な呼び出しステートメント. 125
6. LIST 属性を説明するサンプル・プログラム 127
7. 各種宣言の有効範囲の例 168
8. 構造体のマッピング例 196
9. 小構造体 G のマッピング 197
10. 小構造体 E のマッピング 197
11. 小構造体 N のマッピング 197
12. 小構造体 S のマッピング 198
13. 小構造体 C のマッピング 198
14. 小構造体 M のマッピング 198
15. 大構造体 A のマッピング 199
16. 構造 A の最終的なマッピングにおけるオフセット 200
17. 一方向のチェーンの例 260

第 1 章 本書について

本書で用いられる表記規則. 1
セマンティクス 4
工業規格. 4
本リリースにおける機能強化 4
V3R5 からの機能拡張 5
V3R4からの機能拡張. 5
V3R3 からの機能拡張 5
V3R2 からの機能拡張 5
V3R1 からの機能拡張 6

本書は、以下の IBM 製品で IBM PL/I コンパイラーを使用するプログラマーを対象とした解説書です。

- Enterprise PL/I for z/OS V3R6
- PL/I for AIX V2.0.0.0
- WebSphere Developer for System z バージョン 7、PL/I for Windows

本書は初歩的な手引書ではなく、すでに PL/I 言語に関する知識を持ち、IBM PL/I コンパイラー用のプログラムの作成に参照情報を必要とする読者を対象としています。本書には、ガイダンス情報や汎用プログラミング・インターフェースも含まれています。

本書は解説書であり、最初から順に読まれることを意図していません。したがって、事前に定義されていない用語が使用されている場合があります。用語は定義される箇所で強調表示され、その定義は用語集に記載されています。

WRKSTN➡

このワークステーション開始アイコンとワークステーション終了アイコンで区切られたテキストは、PL/I ワークステーション製品 (AIX および Windows) でのみサポートされる機能を指定しています。

◅日

本書で用いられる表記規則

本書での表記法を、以下に説明します。例およびユーザーが提供する情報は、大文字と小文字の両方を使用して示されています。 本書で使用されている構文図には、以下の規則が適用されます。

矢印記号

構文図は、左から右、上から下へと線をたどって読んでください。

►►── ステートメントはここから始まります。

──► ステートメントの構文は次の行へ続きます。

►── ステートメントは前の行から続いています。

──►◄ ステートメントはここで終わります。

完結したステートメント以外の構文単位の図は、► 記号で始まり、► 記号で終わります。

規則

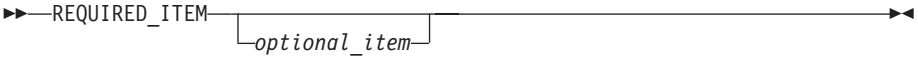
- キーワード、許容される同義語、および予約パラメーターは、MVS プラットフォームでは大文字で示され、UNIX®プラットフォームでは小文字で示されます。これらの項目は示されたとおりに入力する必要があります。
- 変数は、小文字のイタリック体で示します (たとえば、 *column-name*)。これらはユーザー定義のパラメーターまたはサブオプションを表します。
- コマンドの入力において、パラメーターおよびキーワードを区切る句読記号がない場合は、少なくとも 1 つのブランクで区切る必要があります。
- 句読記号 (スラッシュ、コンマ、ピリオド、括弧、引用符、等号) と数字は、示されたとおりに入力する必要があります。
- 脚注は、(1) のように番号を括弧に入れて示します。
- b 記号は、1 つのブランク位置を示します。

必須項目

必須項目は横線 (メインパス) に示します。

オプション項目

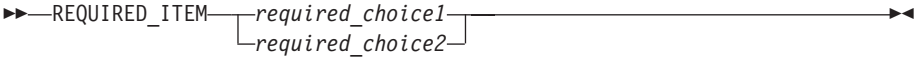
オプション項目は、メインパスの下に示します。



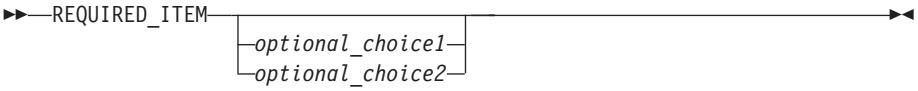
メインパスより上にオプショナル項目を示すこともあります。これは読み易くするためで、ステートメントの実行には影響を及ぼしません。

複数の必須項目またはオプション項目

複数の項目から選択する場合には、それらの項目が縦に重なって、スタックを形成しています。複数の項目からいずれか 1 つを選択しなければならない 場合は、スタック上の項目のうち、1 つがメインパス上に置かれます。

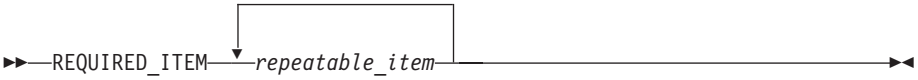


項目がオプショナルである場合、メインパスの下にある支線上に縦に並んだ項目として示されます。

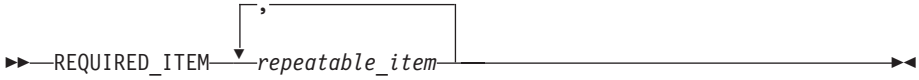


反復可能項目

メインパスの上方を通って左側へ戻る矢印は、項目が反復可能であることを示します。



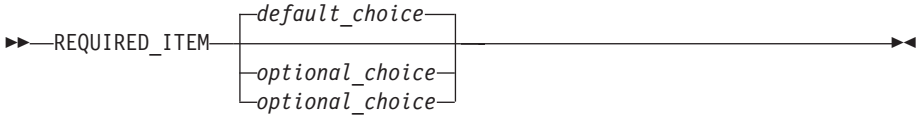
反復矢印の途中にコンマがある場合は、反復される項目をコンマで区切らなければなりません。



スタックの上の繰り返しを示す矢印は、スタックから複数の選択項目を指定できることを示しています。

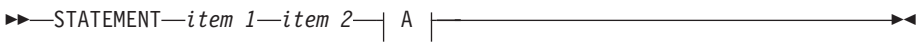
デフォルトのキーワード

IBM 提供のデフォルト・キーワードはメインパスより上に示され、それ以外の選択項目はメインパスより下に示されます。構文図の下にあるパラメーター・リストでは、デフォルト選択項目に下線を付けてあります。

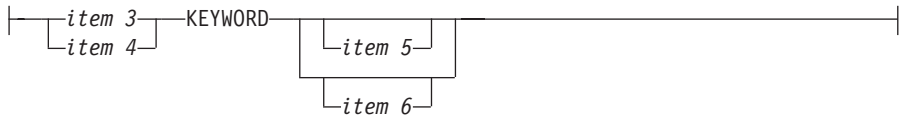


フラグメント

構文図は、フラグメント (部分) に分割する必要がある場合があります。フラグメントは文字またはフラグメント名を用いて | A | のように表します。フラグメントは主図のあとに置かれます。次の例は、フラグメントの使い方を示したものです。



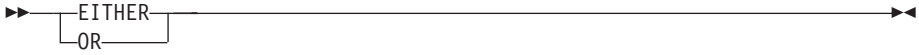
A:



置換ブロック

いくつかのパラメーターの集合を **<A>** のような置換ブロックで表すことがあります。たとえば、/VERB という仮のコマンドで、 /VERB LINE 1、/VERB EITHER LINE 1、または /VERB OR LINE 1 と入力することができます。

ここで、<A> は次のようになります。



パラメーターの終わり

数値を持つパラメーターは記号 '#' で終わり、名前であるパラメーターは 'name' で終わり、汎用とすることができるパラメーターは '*' で終わります。

この例の MSNAME キーワードは名前の値をサポートし、 SYSID キーワードは数値をサポートします。

セマンティクス

PL/I 言語の説明には、以下の規則が使用されます。

- 説明文は簡略化しています。たとえば、正確な意味は「x は変数の名前でなければなりません」であっても、「x は変数でなければなりません」と記されています。同様に、「x の値が伝送されます」の代わりに「x が伝送されます」と記されている場合もあります。構文の説明に「参照」と書かれている場合、あとで「参照された変数」の代わりに「その変数」という表現を使用することがあります。
- 異なる 2 つのソース構成が同等であるというときは、同じ結果をもたらすことを意味しますが、必ずしもその仕組みが同じであるとは限りません。
- 構文の仕様のあとにテキスト内で特に明記していない限り、修飾されていない「式」または「参照」はスカラー式を指しています。スカラー式以外の式の場合には、その式のタイプを明記してあります。たとえば、「配列式」と記されている場合は、スカラー式でも構造式でもありません。
- 「〜*は未定義です*」と記されている場合は、その動作を行っては「ならない」ことを意味します。未定義機能を使用すると、異なる PL/I プロダクトのインプリメンテーションまたはリリースで異なる結果を生成することになります。そのアプリケーション・プログラムはエラーであると見なされます。
- *デフォルト* という用語は、ユーザーが明示的に選択しなかったときにシステムで想定される代替値、属性、またはオプションを説明する場合に使われています。
- *暗黙* という用語は、明示的な指定のないときにプログラムによってとられるアクションを説明する場合に使用されます。
- (単語内以外で使用される) 小文字の b は、ブランク文字を示しています。

工業規格

PL/I コンパイラーは、下記の工業規格 (1987 年 12 月の時点での IBM の解釈にもとづきます) の仕様に従って設計されています。

- 情報交換用米国標準コード (ASCII)、X3.4 - 1977
- 情報交換用ポケット選択文字米国標準規格、レベル 1、X3.77 - 1980 (1979 年 3 月 1 日 ISO への提案事項)
- 情報交換用縦方向キャリッジ位置指定文字米国標準規格、レベル 1 の草案。dpANS X3.78 (1979 年 3 月 1 日 ISO への提案事項)
- 米国標準規格 PL/I 汎用サブセットの選択機能 (ANSI X3.74-1987)

本リリースにおける機能強化

このリリースでは、以下の新しい言語機能を提供しています (その他のプラットフォーム固有の機能強化については、該当の「プログラミング・ガイド」に説明があります)。

- PICSPEC 組み込み関数がサポートされるようになりました。そのため、CHARACTER データをすぐに PICTURE にキャストできます。
- THREADID 組み込み関数が z/OS で使用できるようになりました。変更も加えられており、スレッド ID にポインターを戻すようになりました。また、すべての場合にパラメーターが必須となっています。

V3R5 からの機能拡張

このリリースでは、以下の新しい言語機能を提供しています (その他のプラットフォーム固有の機能強化については、該当の「プログラミング・ガイド」に説明があります)。

- LOCATION 組み込み関数は、構造体が割り振られていなくても、REFER を使用して構造体の中の最初のエレメントを指定できるようになりました。
- DB2 の日付パターンである「YYYY-MM-DD」、「MM/DD/YYYY」および「DD.MM.YYYY」を日時処理の組み込み関数で使用できるようになりました。

V3R4からの機能拡張

このリリースでは、以下の新しい言語機能を提供しています (その他のプラットフォーム固有の機能強化については、該当の「プログラミング・ガイド」に説明があります)。

- DEFAULT ステートメントの意味が、ホスト・コンパイラーでの意味と一致するようになりました。
- ENTRY ステートメントが含まれている PROCEDURES 内の BEGIN ブロックの内部での RETURN ステートメントのサポート。
- REPLACEBY2 組み込み関数
- NOINIT 属性
- MACRO プリプロセッサーの中の以下の組み込み関数
  - LOWERCASE
  - MACNAME
  - TRIM
  - UPPERCASE

V3R3 からの機能拡張

このリリースでは、以下の言語機能を含む、Enterprise PL/I V3R3 で提供されたすべての機能拡張も提供しています。

- 以下の組み込み関数
  - MEMINDEX
  - MEMSEARCH
  - MEMSEARCHR
  - MEMVERIFY
  - MEMVERIFYR
  - XMLCHAR
- GET EDIT の中の V フォーマット項目。

V3R2 からの機能拡張

このリリースでは、以下の言語機能を含む、Enterprise PL/I V3R2 で提供されたすべての機能拡張も提供しています。

- OPTIONS(COBOL) を指定した PROC および ENTRY に対する NOMAP、NOMAPIN、および NOMAP 属性のサポート。
- 異なる RETURNS 属性を持つ ENTRY ステートメントを指定した PROC に対する、旧ホスト・コンパイラーで提供していたのと同じ方法でのサポート。
- OPTIONS(RETCODE) は、OPTIONS(COBOL) を指定した PROC および ENTRY を想定します。
- 未処理の場合、SIZE 条件は ERROR にプロモートされません。
- 新規の USAGE コンパイラー・オプションを使用すると、 RULES(IBM|ANS) オプションの他の影響を受けずに、ROUND および UNSPEC 組み込み関数の IBM または ANS 動作を完全に制御できます。
- PUT LIST と PUT EDIT ステートメントで POINTER が使用できるようになりました。 8 バイトの 16 進値が出力されます。
- ABNORMAL 属性を STATIC 変数で指定すると、STATIC 変数が使用されていなくてもこの変数は保存されます。

V3R1 からの機能拡張

本リリースでは、次を含む Enterprise PL/I V3R1 での機能強化もすべて提供されます。

- z/OS でのマルチスレッド化のサポート
- z/OS での IEEE 浮動小数点のサポート
- マクロ・プリプロセッサーでの ANSWER ステートメントのサポート
- PLISAXA および PLISAXB 組み込みサブルーチンを介した SAX 形式 XML 構文解析
- 追加の組み込み関数
  - CS
  - CDS
  - ISMAIN
  - LOWERCASE
  - UPPERCASE

本リリースでは、次を含む VisualAge PL/I V2R2 での機能強化もすべて提供します。

- WIDECHAR 属性を介した初期 UTF-16 サポート

  以下については、まだサポートされていません。
  - ソース・ファイル内の WIDECHAR 文字
  - W ストリング定数
  - ストリーム入出力内の WIDECHAR 式の使用
  - レコード入出力での WIDECHAR からの暗黙の型変換または WIDECHAR への暗黙の型変換
  - レコード入出力での暗黙の endianness フラグ

    WIDECHAR ファイルを作成する場合は、ファイルの最初の 2 バイトとして endianness フラグ ('fe_ff'wx) を書き込んでください。

- DEFAULT ステートメントでサポートされる DESCRIPTORS オプションと VALUE オプション
- PUT DATA 機能強化
  - POINTER、OFFSET およびサポートされているその他の非計算変数
  - タイプ 3 DO 仕様が使用可能
  - 添え字が使用可能
- DEFINE ステートメントの機能強化
  - 指定されていない構造体の定義
  - CAST および RESPEC タイプ関数
- 追加の組み込み関数
  - ACOSF
  - ASINF
  - ATANF
  - CHARVAL
  - COSF
  - EXPF
  - ISIGNED
  - IUNSIGNED
  - LOG10F
  - LOGF
  - ONWCHAR
  - ONWSOURCE
  - SINF
  - TANF
  - WCHAR
  - WCHARVAL
  - WHIGH
  - WIDECHAR
  - WLOW
- プリプロセッサー機能強化
  - プリプロセッサー・プロシージャーでの配列のサポート
  - %DO ステートメントでの WHILE、UNTIL および LOOP キーワードのサポート
  - %ITERATE ステートメントのサポート
  - %LEAVE ステートメントのサポート
  - %REPLACE ステートメントのサポート
  - %SELECT ステートメントのサポート
  - 追加の組み込み関数
    - COLLATE
    - COMMENT
    - COMPILEDATE
    - COMPILETIME
    - COPY
    - COUNTER
    - DIMENSION
    - HBOUND

- INDEX
- LBOUND
- LENGTH
- MACCOL
- MACLMAR
- MACRMAR
- MAX
- MIN
- PARMSET
- QUOTE
- REPEAT
- SUBSTR
- SYSPARM
- SYSTEM
- SYSVERSION
- TRANSLATE
- VERIFY

第 2 章 プログラムのエレメント

1 バイト文字セット 9
英字および特別言語文字 9
10 進数. 10
2 進数 11
16 進数. 11
特殊文字 11
複合記号 12
大文字小文字の区別 12
SBCS ステートメント・エレメント 13
ID 13
区切り文字と演算子 14
ステートメント 16
単純ステートメント 18
複合ステートメント 18
グループ 19
2 バイト文字セット 19
DBCS ID 20
DBCS ステートメント・エレメント 20
DBCS 継続規則 21

この章では、 PL/I プログラムを作成するときに使用する基本エレメントについて説明します。エレメントには、文字セット、プログラマー定義の ID、キーワード、区切り文字、およびステートメントが含まれます。

PL/I は、1 バイト文字セット (SBCS) および 2 バイト文字セット (DBCS) をサポートしています。

PL/I の言語エレメントに関するインプリメンテーションの制限は、 575 ページの『制限値』にリストされています。

1 バイト文字セット

文字セット とは、*文字* と呼ばれる固有表示の順序付けられたセットを指します。たとえば、モールス符号の記号セットやキリル・アルファベットの文字は文字セットです。 PL/I は、すべての PC 文字セットをサポートします。文字セット 0640 に属する文字はすべての PC コード・ページと同じコード・ポイントを持っているため、*不変* 文字セットとも呼ばれます。*コード・ポイント* とは 256 個の可能文字のうちのいずれかを表す 1 バイト・コードを指し、*コード・ページ* とはすべてのコード・ポイントに対するグラフィック文字と制御機能の意味の割り当てを指します。

PL/I は文字セット 0640 に適合するすべての PC コード・ページをサポートします。ただし、PL/I は、プログラマーが CURRENCY、NAMES、OR、または NOT といったコンパイラー・オプションを使用して指定する文字を除いたすべてのコード・ポイントについても PC コード・ページ 0850 を仮定しています。これらのオプションの詳細については、「プログラミング・ガイド」を参照してください。

コード・ページ 0850 には、英字、10 進数字、特殊文字、およびその他の各国語や制御文字が入っています。定数およびコメントでは、SBCS 文字のすべての値を使用することができます。 PL/I エレメント (たとえば、ステートメント、キーワード、および区切り文字など) は、以下の節で説明されている文字に限定されます。

英字および特別言語文字

PL/I のデフォルトの英字は、英字に特別言語文字が加わります。

英字

英語アルファベットを含む 26 の基本英字があります。それぞれの英字とそれに対応する ASCII および EBCDIC 値 (16 進表記) を表 1 に示します。

表 1. 英字と同等の値

文字	EBCDIC 16 進値 (大文字)	EBCDIC 16 進値 (小文字)	ASCII 16 進値 (大文字)	ASCII 16 進値 (小文字)
A	C1	81	41	61
B	C2	82	42	62
C	C3	83	43	63
D	C4	84	44	64
E	C5	85	45	65
F	C6	86	46	66
G	C7	87	47	67
H	C8	88	48	68
I	C9	89	49	69
J	D1	91	4A	6A
K	D2	92	4B	6B
L	D3	93	4C	6C
M	D4	94	4D	6D
N	D5	95	4E	6E
O	D6	96	4F	6F
P	D7	97	50	70
Q	D8	98	51	71
R	D9	99	52	72
S	E2	A2	53	73
T	E3	A3	54	74
U	E4	A4	55	75
V	E5	A5	56	76
W	E6	A6	57	77
X	E7	A7	58	78
Y	E8	A8	59	79
Z	E9	A9	5A	7A

特別言語文字

デフォルトの特別言語文字は、*番号* 記号 (#)、*単価* 記号 (@)、および*通貨* 記号 ($) です。これらの文字の 16 進値は、コード・ページに応じて変化します。NAMES コンパイラー・オプションを使用すると、独自の特別言語文字を定義することができます。特別言語文字を定義する方法の詳細については、「*プログラミング・ガイド*」を参照してください。

英数字

英数字 は、英字または特別言語文字、あるいは数字のいずれかになります。

10 進数

PL/I は、0 から 9 までの 10 の 10 進数を認識します。10 進数は単に「数字」と呼ばれることもあり、10 進定数およびその他の表示や値を表記する場合に使用され

ます。以下の表では、数字とその 16 進値を表しています。

表 2. 10 進数と同等の値

文字	EBCDIC 16 進値	ASCII 16 進値
0	F0	30
1	F1	31
2	F2	32
3	F3	33
4	F4	34
5	F5	35
6	F6	36
7	F7	37
8	F8	38
9	F9	39

2 進数

PL/I は 0 と 1 の 2 個の 2 進数を認識します。2 進数は「ビット」と呼ばれることもあり、2 進数およびビット定数を表記する場合に使用されます。

16 進数

PL/I は、0 から 9 および A から F までの 16 の 16 進数を認識します。 A から F までは、それぞれ 10 進数値の 10 から 15 までを表します。 16 進数は、16 進数字、または単に 16 進と呼ばれることもあり、定数を 16 進表記で表す場合に使用されます。

特殊文字

表 3 では、特殊文字、それらの PL/I での意味、ならびに ASCII と EBCDIC 値 (16 進表記) を示します。

表 3. 特殊文字と同等の値

文字	意味	デフォルトの EBCDIC 16 進値	デフォルトの ASCII 16 進値
b	ブランク	40	20
=	等号記号または代入記号	7E	3D
+	正符号	4E	2B
-	負符号	60	2D
*	アスタリスクまたは乗算記号	5C	2A
/	スラッシュまたは除算記号	61	2F
(	左括弧	4D	28
)	右括弧	5D	29
,	コンマ	6B	2C
.	小数点またはピリオド	4B	2E
'	単一引用符	7D	27
"	二重引用符	7F	22
%	パーセント	6C	25
;	セミコロン	5E	3B
:	コロン	7A	3A

表 3. 特殊文字と同等の値 (続き)

文字	意味	デフォルトの EBCDIC 16 進値	デフォルトの ASCII 16 進値
¬	NOT 記号、排他的 OR 記号 注 1	5F	5E
&	AND 記号	50	26
|	OR 記号 (注 1)	4F	7C
>	より大記号	6E	3E
<	より小記号	4C	3C
_	区切り文字 (下線)	6D	5F

注 1:

OR (|) および NOT (¬) 記号には、変形コード・ポイントがあります。コンパイラー・オプションの OR および NOT を使用して、これらの演算子を表す代替記号を定義することができます。これらのオプションの詳細については、「プログラミング・ガイド」を参照してください。

複合記号

特殊文字を組み合わせて、複合記号を作成することができます。下の表では、これらの記号とその意味を示しています。複合記号には、ブランクを入れることはできません。

表 4. 複合記号の説明

複合記号	意味
||	連結
**	累乗演算
¬<	より小でない
¬>	より大でない
¬=	等しくない; 評価、排他 OR および代入
<=	より小さいか等しい
>=	より大きいか等しい
/*	コメントの始まり
*/	コメントの終わり
->	ロケーター (ポインターとオフセット)
=>	ロケーター (ハンドル)
+=	式の計算、加算および代入
-=	式の計算、減算および代入
*=	式の計算、乗算および代入
/=	式の計算、除算および代入
|=	式の計算、OR および代入
&=	式の計算、AND および代入
||=	式の計算、連結と代入
**=	式の計算、指数化および代入

大文字小文字の区別

PL/I プログラム内では、大文字と小文字を組み合わせて使用することができます。

キーワードまたは ID の使用では、小文字はそれに対応する大文字と見なされます。 DBCS 文字として小文字を入力した場合でも、大文字に変換されます。

コメントまたは文字、混合または漢字ストリング定数内で大文字と小文字を組み合わせて使用する場合には、小文字は小文字のままです。

SBCS ステートメント・エレメント

このセクションでは、 SBCS を使用する場合に PL/I プログラムを構成するエレメントを説明します。

PL/I ステートメントは、ID、区切り文字、演算子、および定数で構成されます。定数は、 23 ページの『第 3 章 データ・エレメント』で説明します。

ID

ID は、コメントや定数の中には含まれない文字の集まりです。 P、PIC、および PICTURE 以外は、ID が先になり、区切り文字がそのあとに続かなければなりません。 (P、PIC、および PICTURE の各 ID のあとは、文字ストリングが続きます。) ID の最初の文字は、英字、特別言語文字のいずれかにならなければなりません。 ID 名が INTERNAL シンボルである場合、区切り (_) 文字を最初の文字にすることもできます。残りの文字としては、英字、特別言語文字、数字、または下線 (_) 文字が可能です。外部ユーザー名は、IBM、PLI、CEE、_IBM、_PLI、および _CEE 以外の文字で始まる名前ではいけません。

ID は、PL/I キーワードまたはプログラマーが定義する名前となります。 ID がキーワードである場合には、 PL/I は構文から判別することができるので、プログラマーが定義した名前として任意の ID を使用することができます。 PL/I では予約されている 語はありません。

PL/I キーワード

キーワード は、PL/I 内で特定の意味を持つ ID です。取られる処置またはデータの属性などを、キーワードで指定することができます。たとえば、READ、DECIMAL、および ENDFILE などはキーワードです。キーワードによっては省略形を使うことができます。キーワードとその省略形は、大文字で示されます。

プログラマー定義の名前

PL/I プログラムでは、変数やプログラム制御データに*名前* を付けます。そのほかに、組み込み名、条件名、および総称名があります。任意の ID を名前として使用することができます。名前はプログラム・ブロック内でそれぞれ固有でなければなりません。同一ブロック内のファイルと浮動小数点変数の両方に同じ名前を付けることはできません。

読みやすくするために、`Gross_Pay` のように中間に区切り文字 (_) を入れることができます。

名前の例:

```
A                   Rate_of_pay

Record              Loop_3
```

プログラマー定義の外部名に関するその他の要件については、 166 ページの『INTERNAL 属性と EXTERNAL 属性』で説明します。

名前が必要であるにもかかわらず名前を使用しない場合は、アスタリスク (*) を ID として使用することができます。この例については、124 ページを参照してください。

区切り文字と演算子

区切り文字 と *演算子* は、ID と定数を分離するのに使われます。 表 5 は区切り文字、 15 ページの表 6 は演算子を示します。

表 5. 区切り文字

名前	区切り文字	用途
コンマ	,	リスト内の各エレメントを区切ります。BY NAME オプションの前でも使います。
ピリオド	.	修飾名の各エレメントを接続します。10 進数または 2 進数の小数点としても使います。
セミコロン	;	ステートメントの終わりを表します。
等号	=	代入を表すほか、条件式では「等しい」ことを表します。
コロン	:	ステートメントに接頭語をつなぐときに使います。次元属性で、下限と上限を連結するときや、DEFAULT ステートメントの RANGE 指定内でも使います。
ブランク	b	エレメントを区切ります。
括弧	( )	リスト、式、反復因数 (iteration factor)、および反復因数 (repetition factor) を囲みます。各種のキーワードに関連する情報を囲むときも使います。
ロケーター	-> =>	ロケーター修飾 (ポインターとオフセット) を表します。 ロケーター修飾 (ハンドル) も表します。
パーセント	%	% ステートメントと % ディレクティブを表します。

注: 必要な記号を忘れると、トレース困難なエラーが起こることがあります。よくあるエラーとしては、引用符が対になっていない、小括弧が対になっていない、コメント区切り文字が対になっていない、ならびにセミコロンが欠落しているなどがあります。

表 6. 演算子

演算子のタイプ	文字	意味
算術	+	加算または前置加算
	-	減算または前置減算
	*	乗算
	/	除算
	**	累乗演算
比較	=	等しい
	¬=	等しくない
	<	より小
	¬<	より小でない
	>	より大
	¬>	より大でない
	<=	より小さいか等しい
	>=	より大きいか等しい
論理演算子	¬	NOT、排他的 OR
	&	AND
	|	または
ストリング演算子	||	連結

区切り文字として使う文字をほかのコンテキストで使うことができます。たとえば、名前を修飾するときに使うピリオドは区切り文字 (たとえば、`Weather.Temperature`) ですが、算術定数内で使用する場合は 10 進小数点です (たとえば、`3.14`)。

ブランク

各演算子または区切り文字の前後をブランク (b) で囲むことができます。

ブランク以外の区切り文字で区切られていない ID や定数は、1 つ以上のブランクで区切る必要があります。ただし、この規則の唯一の例外として、ID P、PIC、および PICTURE の後にはブランクなしで文字ストリングを続けることができます。ブランクは、入れることが可能なところであれば、いくつでも入れることができます。

ID、複合記号、または定数 (文字定数、混合定数、ワイド文字、および漢字ストリング定数は除く) の中にブランクがあってはなりません。

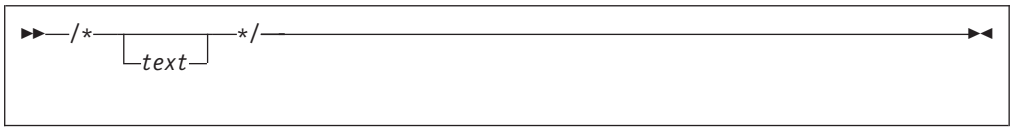
以下に例を示します。

```
ab+bc           と  Ab + Bc は同等
Table(10)       と  TABLEb(b10bbb) は同等
First,Second    と  first,bsecond は同等
AtoB            と  AbtobB は同等ではない
```

以上のほかに、ブランクを必要または許可する場合については、それぞれの言語特性を説明する際に言及します。

コメント

ブランクを区切り文字として使用できるところでは、コメントを使用することができます。コメントは、ブランクと見なされ、区切り文字として使用されます。コメントはプログラム内のロジックでは無視され、影響を及ぼしません。コメントの構文は、以下のとおりです。



/* コメントの始まりを示します。

text
コメントの終わりを示す複合記号 */ 以外の文字は、任意の順序で指定することができます。

***/** コメントの終わりを示します。

コメントは、1 行、または複数行にまたがって入れることができます。たとえば、次のようになります。

```
A = /* This comment is on one line */ 21;

    /* This comment spans
       two lines          */
```

以下の例では、コメントとして表示されるものは、 (引用符で囲まれているので) 実際には文字ストリング定数です。

```
A = '/* This is a constant, not a comment */' ;
```

ステートメント

ID、区切り文字、演算子、および定数を使用して、 PL/I ステートメントを作ることができます。

ソース・プログラムは一連のレコードまたは行から構成されますが、 PL/I は、プログラムを連続した文字のストリームと見なします。 PL/I ステートメントのフォーマットにはほとんど制約がないので、プログラムを作成する際に特別のコーディング規則に注意する必要はなく、各ステートメントを特定の桁位置から始める必要もありません。ステートメントは、前のステートメントのすぐあとの位置から開始できます。また、ブランク (いくつでもよい) で区切ることができます。

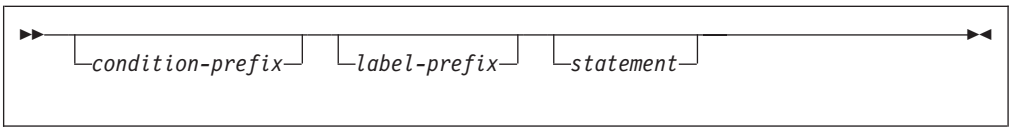
ステートメントの中には、% 記号で始まるものもあります。これらのステートメントは、プリプロセッサーおよびコンパイラーの演算 (ライブラリーからのプログラム・ソース・テキストをはじめとするリストの制御など) を指示する % ディレクティブであるか、または PL/I マクロ機能の % ステートメントであるかのいずれかです。 % ディレクティブは単独で 1 行を占めなければなりません。

プログラムを読みやすく保守しやすくするため、およびソース行の後続ブランクの欠落によって予測できない結果が生じるのを防ぐため、以下のことを実行してください。

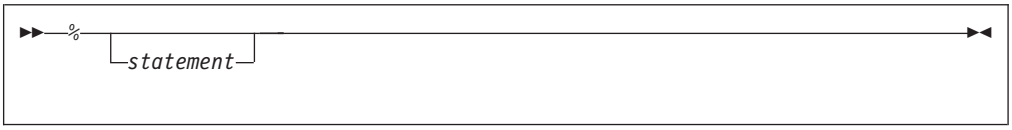
- 言語エレメントを分割して複数行に置くようなことがないようにします。ストリング定数を複数行に書かなければならない場合には、連結記号 (||) を使用してください。
- 1 行に複数のステートメントを記述することはできません。
- % ディレクティブを複数行に分割することはできません。

PL/I ステートメント、マクロ機能 % ステートメント、および % ディレクティブは、201 ページの『第 9 章 ステートメントとディレクティブ』にアルファベット順にリストされています。

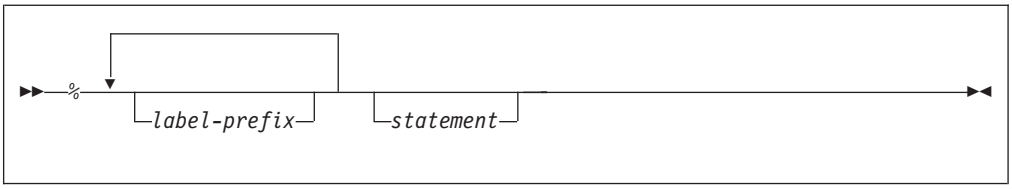
PL/I ステートメントの構文は、以下のとおりです。



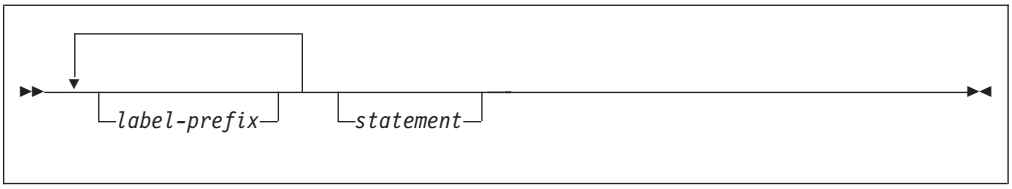
% ディレクティブの構文は、以下のとおりです。



% ステートメントの構文は、以下のとおりです。



マクロ・ステートメントの構文は、以下のとおりです。



各ステートメントは、いずれかのグループまたはブロックに含まれていなければなりません。マクロ・ステートメントは、なんらかのマクロ・グループまたはプロシージャーの中に囲まれていなければなりません。

condition-prefix

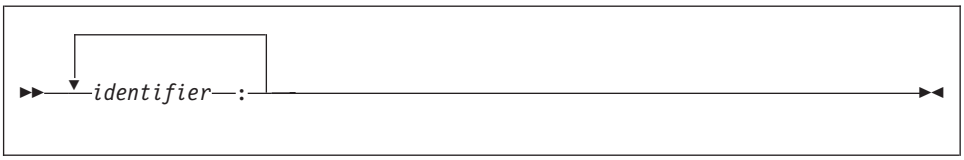
condition-prefix は、PL/I 条件の可能または禁止を指定します (357 ページの『第 16 章 条件処理』を参照)。

label-prefix

label-prefix は、1 つまたは複数のステートメント・ラベルです。ステートメント・ラベルとは、プログラム内のどこか別の場所からそのステートメントを参照

できるように、ステートメントを識別する名前です。ステートメント・ラベルは、ラベル定数 (51 ページの『ラベル・データと LABEL 属性』で取り上げられています)、入り口定数 (120 ページの『入り口データ』で取り上げられています)、またはフォーマット定数 (53 ページの『フォーマット・データと FORMAT 属性』で取り上げられています) のいずれかになります。

DECLARE、DEFAULT、WHEN、OTHERWISE および ON ステートメント以外の任意のステートメントは、ラベル接頭部を持つことができます。ラベル接頭部の構文は以下のとおりです。



一般的にこの資料全体の独立ステートメントの構文は、条件接頭語またはラベル接頭部を示しません。

statement
単純または複合ステートメントがあります。

単純ステートメント

単純ステートメントのタイプは、キーワード、代入、およびヌルです。

キーワード・ステートメント は、キーワードで始まるステートメントです。 このキーワードは、ステートメントの機能を示します。以下の例では、READ および DECLARE はキーワードです。

```
read file(In) into(Input);          /* keyword statement  */
%declare Text char;                 /* keyword %statement  */
```

代入ステートメント には、代入記号 (=) の左側にある 1 つまたは複数の ID、および右側の式が含まれます。このステートメントはキーワードで始まりません。

```
A = B + C;                          /*   assignment statement  */
%Size = 15;                         /* % assignment statement   */
```

ヌル・ステートメント は、1 つのセミコロンだけで構成され、演算できないステートメントです。

```
      ;                             /* null statement         */
Label:;                             /* labeled null statement */
  % ;                               /* % null statement     */
```

複合ステートメント

複合ステートメントはすべてキーワード・ステートメントです。複合ステートメントは、そのステートメントの目的を示すキーワードで始まります。複合ステートメントには、1 つまたは複数の単純ステートメントまたは複合ステートメントが含まれます。 4 つの複合ステートメント、IF、ON、WHEN、および OTHERWISE があります。複合ステートメントは、セミコロンで終了します。このセミコロンは、複合ステートメントの最後のステートメントの終わりも示します。

以下に複合ステートメントの例を示します。

```
on conversion
  onchar() = '0';

if Text = 'stmt' then
  do;
    select(Type);
      when('if') call If_stmt;
      when('do') call Do_stmt;
      when('') /* do nothing */ ;
      otherwise
        call Other_stmt;
    end;
    call Print;
  end;
end;

%if Type = 'AREA' %then
  %Size = Size + 16;
%else;
```

グループ

ステートメントは、グループと呼ばれるさらに大きなプログラム単位内に入れることもできます。グループ は、DO グループまたは SELECT グループのいずれかです。 DO グループとは、DO ステートメントとそれに対応する END ステートメントで区切られた一連のステートメントです。 SELECT グループは、SELECT ステートメントとそれに対応する END ステートメントで区切られた、一連の WHEN ステートメントとオプションの OTHERWISE ステートメントです。区切りステートメントは、グループの一部です。

複合ステートメント内にグループがある場合、グループを単独ステートメントであるかのように処理して、そのグループに制御が渡されるか、あるいはそのグループが迂回されることもあります。

グループ内での制御の流れについては、DO グループの場合は 209 ページの『DO ステートメント』の項で説明し、SELECT グループの場合は 233 ページの『SELECT ステートメント』の項で説明します。

各グループは、いずれかのグループまたはブロック内に入れられなければなりません。グループには、任意の数 (ゼロを含む) のステートメント、グループ、またはグループが入ることができます。

2 バイト文字セット

2 バイト文字セット (DBCS) の各文字は、2 バイトで保存されます。 GRAPHIC コンパイラー・オプションが有効な場合は、 DBCS 文字や SBCS 文字を使用していくつかのソース言語エレメントを書くことができます。特に、ソース・プログラム内の次の場所で DBCS 文字を使用できます。

- コメントの内部
- ステートメント・ラベルまたは ID の一部として
- G または M リテラル内

ただし、INCLUDE ファイル名と FETCH ステートメントの TITLE オプションは、 SBCS でなければなりません。

z/OS プラットフォームでは、DBCS 文字の各ストリングをシフト・コードで直接囲む必要がありますが、コンパイラーでサポートされる他のプラットフォームでは、シフト・コードは不要であり、受け入れられません。以降の例ではこれらのシフト・コードが含まれていますが、z/OS 以外のすべてのプラットフォームの場合にはそれらを省略する必要があります。

DBCS ID

DBCS ID は、DBCS フォーマットの 1 バイト文字、2 バイト文字、またはその両方を組み合わせたものでもかまいません。

DBCS フォーマットの 1 バイト ID

1 バイト文字だけを含む DBCS ID は、先頭文字規則も含めて通常の PL/I 命名規則に従っていなければなりません。 DBCS と同等のものとして表された 1 バイト文字を含む DBCS ID は、 SBCS での同じ ID の同義語です。

注:

1. 本書では記号「**.**」を使用します。 (太字ピリオド) を使用して SBCS としても表すことのできる、2 バイト文字セットの最初のバイトを表します。
2. 本書では、「kk」を使用して、2 バイト文字を表します。
3. 本書では、「<」を使用してシフトアウト文字 ('0E'X) を表します。
4. 本書では、「>」を使用してシフトイン文字 ('0F'X) を表します。

例

```
<.I.B.M> = 3;    /* is the same as IBM=3; */
```

2 バイト文字を含む DBCS ID

DBCS 名に使用されるバイト数は、コンパイラーの LIMITS オプションで名前の最大長として指定された値を超えてはなりません。

DBCS ID 内部の DBCS フォーマットで表された SBCS 文字は、SBCS と見なされます。たとえば、次のようになります。

```
A<kk>B
A<kk.B>
<.Akk>B            /* are all A<kk>B  (SBCS-DBCS-SBCS) */
```

2 バイト文字 ID の用途

DBCS ID は、SBCS ID が許される個所ならどこでも、使用することができます。DBCS ID が EXTERNAL 名および %INCLUDE ファイル名として使用される場合には、その ID をオペレーティング・システムが受け入れるか、または以下のいずれかによって受け入れ可能になるかどうかを確かめなければなりません。

- EXTERNAL 属性と環境名を一緒に使用することによって
- OPEN ステートメントの TITLE オプションを使用することによって

DBCS ステートメント・エレメント

この節では、DBCS を使用しての PL/I 言語エレメントの作成方法に関する補足情報を提供します。この節における言語エレメントの定義および一般的な使用規則は、 13 ページの『SBCS ステートメント・エレメント』の対応する節に記載されています。

DBCS を使用してコード化できる言語エレメント、規則の説明、および使用法の例を以下にリストします。

ID　SBCS、DBCS またはその両方を使用します。

```
dcl Eof                  /* in SBCS, is the same as        */
dcl <.E.o.f>               /* this in DBCS.                  */

dcl <kkkk>X                /* these are all the same, where  */
dcl <kkkk.X>              /* kk is a valid                  */
dcl <kkkk>x               /* DBCS character                 */
dcl <kkkk.x>              /*                                */
```

コメント

SBCS、DBCS またはその両方を使用します。

```
/* comment */            /* all SBCS                       */
/* <.T.y.p.ekk> */        /* DBCS text */
```

コメント区切り文字を SBCS で指定しておく必要があります。

混合文字ストリング定数

SBCS、DBCS またはその両方を使用し、SBCS または DBCS 引用符で囲みます。

引用符に囲まれた DBCS データは SBCS に変換されません。そのデータおよびシフト・コードは、ストリングの内部表記にそのままで保管されます。

```
'<.a.b.c>'M
'<.I.B.M.'.S>'M
'<.I.B.M>''<.S>'M
'IBM<kk>'M
```

グラフィック定数

DBCS のみを使用し、SBCS または DBCS 引用符で囲みます。

```
'<.a.b.c>'G                  /* 6 byte graphic constant  */
'<.I.B.M.'.S>'G            /* 10 byte graphic constant .I.B.M.'.S  */
```

G リテラルは、前後を DBCS 引用符で囲んで指定でき、またその場合は G 自体を DBCS で指定することもできます。そのため、以下はすべて同等です。

```
'<.a.b.c>'G
<.'.a.b.c.'>G
<.'.a.b.c.'.G>
```

DBCS 継続規則

ストリング・リテラルまたは ID に、単一の SBCS ブランクで右マージンと区切られた DBCS 文字がある場合には、そのブランクは無視され、ステートメントのエレメントは次のレコードの左マージンに続きます。

第 3 章 データ・エレメント

データ項目 23
変数 23
定数 24
引用符の使用 24
句読点のついた定数 24
データ・タイプとその属性 24
データ属性 25
計算データ・タイプとその属性 29
コード化算術データとその属性 30
ストリング・データとその属性 37

この章では、PL/I プログラムで使用できるデータの種類と、それらを説明するために使用する属性を説明します。以下の説明について取り上げています。

データ項目の検討
変数および定数の検討
使用可能なデータ・タイプおよびそれらを定義する属性

データを宣言する方法については、159 ページの『第 8 章 データ宣言』を参照してください。

データ項目

データ項目 は、変数または定数のいずれかの値になります。 (これらの用語は、一般の数学的な使用法と必ずしも同じではありません。用語については、次の節で述べています。) データ項目は、*スカラー* と呼ばれる単一の項目です。あるいはデータ*集合体* と呼ばれる項目の集まりであることもあります。

データ集合は、総体的にまたは個別に参照できるデータ項目の集まりです。データ集合の種類は、*配列*、*構造体*、および*共用体* です。任意のタイプの計算データまたはプログラム制御データを使用して、データ集合を作成することができます。

配列は 181 ページの『配列』で、構造体は 185 ページの『構造体』で、共用体は 186 ページの『共用体』で、また共用体と構造体の配列は 191 ページの『配列、構造体、および共用体の組み合わせ』で取り上げています。

変数

変数 は、プログラムの実行中に変化し得る 1 つまたは複数の値を持ちます。変数は宣言で使用され、そこで変数の名前および特定な属性を宣言します。ただし、NONASSIGNABLE 属性の変数は、プログラムの実行中に変化しません。 (詳細については、260 ページの『ASSIGNABLE 属性と NONASSIGNABLE 属性』を参照してください)。*変数参照* は、次のいずれかになります。

- 宣言されている変数名
- 以下の 1 つまたは複数から宣言された名前から派生した参照
  - ポインター修飾
  - 構造体修飾
  - 添え字付け

(55 ページの『第 4 章 式および参照』を参照してください。)

定数

定数 は、変化しない値を持ちます。計算データの定数は、DECLARE ステートメントで定数の値を宣言するか、または定数に名前を付けることによって参照されます。 名前のついた定数の宣言に関する詳細については、 49 ページの『名前付き定数』を参照してください。

プログラム制御データの定数は、名前によって参照されます。

引用符の使用

ストリング定数、16 進定数、およびピクチャー指定は、単一引用符または二重引用符のいずれかで囲まれます。

ストリング内の引用符には、以下の規則が適用されます。

- 組み込まれる引用符が、ストリングを囲むために使用されたものと同じタイプである場合には、組み込まれる 1 つの引用符ごとに 2 つの引用符を入力しなければなりません (つまり、" または "")。
- 組み込まれる引用符が、ストリングを囲むのに使用されたものと異なるタイプである場合は、組み込まれる各インスタンスごとに 1 つの引用符だけを入力します。単一引用符は、データとして処理されます。

例:

```
'Shakespeare''s "Hamlet"' is identical to
"Shakespeare's ""Hamlet"""

PICTURE "99V9" is identical to
PICTURE '99V9'
```

注: 本書の構文図では、単一引用符を示しています。二重引用符は、特に明記されていなければ、置き換えることができます。

句読点のついた定数

読みやすくするために、算術、ビット、および 16 進数の定数には、区切り文字 (_) を使用できます。

'1100_1010'B	は	'11001010'B と同等
1100_1010B	は	11001010B と同等
'C_A'B4	は	'ca'b4 と同等
'C_A'XN	は	'ca'XN と同等
16_777_216	は	16777216 と同等

データ・タイプとその属性

PL/I プログラムで使用されるデータは、計算データまたはプログラム制御データに分類されます。

計算データ
求める結果を得るための計算に使用する値を表します。計算データは算術データとストリング・データから構成されます。

算術データは、コード化算術データまたは数値ピクチャー・データのいずれかです。

コード化算術データ項目は、有理数です。算術データ項目の属性は、*基数* (BINARY または DECIMAL)、*スケール* (FLOAT または FIXED)、*精度* (有効数字と小数点の配置)、およびモード (REAL または COMPLEX) です。

数値ピクチャー・データは、文字フォーマットを持っている数字データであり、45 ページの『数字データ』で説明されています。

ストリング は、単一データ項目と見なされる一連の隣接する文字、ビット、ワイド文字、またはグラフィックのことです。

プログラム制御データ
プログラムの実行を制御するために使用される値を表示します。プログラム制御データは、以下のデータ・タイプ区域、入力、ラベル、ファイル、フォーマット、ポインター、およびオフセットで構成されます。

以下に例を示します。

```
Area = (Radius**2) * 3.1416;
```

`Area` と `Radius` は、計算データのコード化算術変数です。 `2` および `3.1416` は、計算データのコード化算術定数です。

`3.1416` という数字がプログラム中の複数の場所で使用される場合、あるいは特定のデータや精度属性が必要な場合は、この数字を名前付き定数として宣言しておくほうがいいでしょう。したがって、上記のステートメントは以下のようにコーディングすることができます。

```
dcl Pi FIXED DECIMAL (5,4) VALUE(3.1416);
area = (radius**2) * Pi;
```

プログラム制御データの定数は、コンパイラーによって判別される値を持ちます。下記の例では、`loop` という名前はプログラム制御データのラベル定数を表しています。 `loop` という値は、`A=2*B;` ステートメントのアドレスです。

```
loop:  A=2*B;
       C=B+6;
```

データ項目を使用して作業するためには、 PL/I にデータ・タイプとその処理方法を知らせる必要があります。*属性* がこの情報を提供します。属性の種類は、データ属性と非データ属性です。

データ属性

計算データ、プログラム制御データ、およびプログラム特性を示します。

AREA	FILE	OFFSET	STRUCTURE
BINARY	FIXED	ORDINAL	TASK
BIT	FLOAT	PICTURE	TYPE
CHARACTER	FORMAT	POINTER	UNSIGNED
COMPLEX	GRAPHIC	PRECISION	UNION
DECIMAL	HANDLE	REAL	VARYING
DIMENSION	LABEL	RETURNS	VARYINGZ
ENTRY	NONVARYING	SIGNED	WIDECHAR

非データ属性

非データ・エレメント (たとえば、組み込み関数) を示すか、またはほかのデータ属性を持つエレメントに説明を追加します。

ABNORMAL	DEFINED	LIKE	PRINT
ALIGNED	DIRECT	LIST	RECORD
ASSIGNABLE	ENVIRONMENT	LITTLEENDIAN	SEQUENTIAL
AUTOMATIC	EXCLUSIVE	NONASSIGNABLE	STATIC
BASED	EXTERNAL	NONCONNECTED	STREAM
BIGENDIAN	GENERIC	NORMAL	UNALIGNED
BUFFERED	HEXADEC	OPTIONAL	UNBUFFERED
BUILTIN	IEEE	OPTIONS	UPDATE
BYADDR	INITIAL	OUTPUT	VALUE
BYVALUE	INPUT	PARAMETER	VARIABLE
CONDITION	INTERNAL	POSITION	
CONNECTED	KEYED		
CONTROLLED			

たとえば、CHARACTER というキーワードは、計算データのストリング・タイプのデータ属性です。 FILE というキーワードは、プログラム制御データのファイル・タイプのデータ属性です。 INTERNAL という有効範囲属性は、データ項目が宣言されるブロック内でのみ、そのデータ項目が認識されることを指定します。

キーワードと式を使用して属性を指定する方法の詳細については、 159 ページの『第 8 章 データ宣言』を参照してください。属性を指定する方法の概略は次のとおりです。

明示的に DECLARE ステートメントを使用する方法
コンテキストから PL/I が属性を判別する方法
プログラマーが定義したデフォルトまたは言語によって定められたデフォルトを使用する方法

28 ページの表 7 と 29 ページの表 8 は、 PL/I の各種の属性と、計算データおよびプログラム制御データの各タイプとの関連を示しています。 これらの表では、定数と名前のついた定数を表しており、示されたデータ属性と有効範囲属性 (28 ページの表 7) だけを持つことができます。変数を使用して付加属性を指定することができます (29 ページの表 8)。

以下に例を示します。

```
Area = (Radius**2)*3.1416;
```

定数 3.1416 には、次のような属性が指定されます。

明示的に 2 進定数ではないので DECIMAL
固定小数点であるために FIXED
PRECISION(5,4) (5 桁の有効数字で、そのうちの 4 桁は小数点の右側にあること)
虚数部分がないので REAL
INTERNAL および ALIGNED

(28 ページの表 7 の「コード化算術式」の行、および「データ属性」と「有効範囲属性」の列を参照してください。)

定数 1.0 (10 進固定小数点定数) は、定数 1 (もう 1 つの 10 進固定小数点定数)、'1'B (ビット定数)、'1' (文字定数)、 1B (2 進固定小数点定数) または 1E0 (10 進浮動小数点定数) とは異なります。

以下の例では、変数 Pi は 5 桁の 10 進数で、そのうちの 4 桁は小数点の右側にあるという PRECISION のついた、FIXED および DECIMAL というプログラマー定義のデータ属性があります。

```
declare Pi fixed decimal(5,4) initial(3.1416);
```

この DECLARE ステートメントには、Pi についてのほかの属性を持たないので PL/I は、残りの属性のデフォルトを適用します。

データ属性列からの REAL
位置合わせ属性列からの ALIGNED
有効範囲属性列からの INTERNAL
ストレージ属性列からの AUTOMATIC
データ属性列からの SIGNED

(29 ページの表 8 のコード化算術式の行を参照してください。)

非データ属性

表 7. 定数タイプによる属性の分類

定数タイプ	**データ属性** 注 1 および 2	**有効範囲属性** 注 1 および 2
コード化 算術	REAL | 虚数 FLOAT | FIXED BINARY | DECIMAL PRECISION SIGNED	内部
名前つきのコード化 算術	REAL | COMPLEX FLOAT | FIXED BINARY | DECIMAL PRECISION VALUE SIGNED | UNSIGNED	内部
ストリング演算子	BIT | CHARACTER | GRAPHIC | WIDECHAR (長さ)	内部
名前の付いたストリング	BIT | CHARACTER | GRAPHIC | WIDECHAR [(長さ)] NONVARYING VALUE	内部
名前の付いたロケーター	POINTER | OFFSET | HANDLE VALUE	内部
名前の付いたピクチャー	PICTURE REAL | COMPLEX VALUE	内部
ファイル 注 3	FILE ENVIRONMENT STREAM | RECORD INPUT | OUTPUT | UPDATE SEQUENTIAL | DIRECT BUFFERED | UNBUFFERED 注 4 KEYED PRINT	INTERNAL | EXTERNAL
入り口注 5	ENTRY [RETURNS]	INTERNAL | EXTERNAL
フォーマット注 5	FORMAT	内部
ラベル注 5	LABEL	内部

注:

1. この表の中で大文字で表されている属性は、明示的に宣言されます。小文字の属性は、暗黙的にデータ・タイプに指定されます。
2. データ属性のデフォルトには下線が引かれています。リテラル定数のデータ属性はコンテキストに応じて異なるので、デフォルトは適用されません。名前のついた定数およびファイル定数には選択可能な属性があるので、デフォルトが示されます。
3. ファイル属性は、 277 ページの『第 11 章 入出力』に記載されています。
4. BUFFERED は、SEQUENTIAL ファイルのデフォルトです。 UNBUFFERED は、DIRECT ファイルのデフォルトです。
5. フォーマット定数およびラベル定数、ならびに INTERNAL 項目定数は、 DECLARE ステートメントでは宣言されません。

表 8. 変数タイプによる属性の分類

変数 タイプ	データ属性	位置合わせ 属性	有効範囲 属性	ストレージ 属性
区域	AREA(*size*)	ALIGNED	INTERNAL | EXTERNAL (INTERNAL は AUTOMATIC BASED DEFINED PARAMETER に必須)	AUTOMATIC | STATIC | BASED | CONTROLLED (AUTOMATIC は INTERNAL についてデフォルト。 STATIC は EXTERNAL についてデフォルト。) 定義された変数: DEFINED [POSITION] パラメーター: PARAMETER [CONNECTED | NONCONNECTED] [CONTROLLED] [INITIAL [CALL]] [VARIABLE] [NORMAL | ABNORMAL] ASSIGNABLE | NONASSIGNABLE
コード化算術 注 1	REAL | COMPLEX FLOAT | FIXED BINARY | DECIMAL PRECISION [SIGNED | UNSIGNED]	ALIGNED | UNALIGNED		
入り口	ENTRY [RETURNS] [LIMITED]			
ファイル	FILE			
フォーマット	FORMAT			
ラベル	LABEL			
ロケーター	POINTER | HANDLE | {OFFSET [(区域変数)]}			
序数	ORDINAL			
ピクチャー	PICTURE REAL | COMPLEX	ALIGNED | UNALIGNED		
ストリング演算子	BIT | CHARACTER | GRAPHIC | WIDECHAR [(長さ)] [VARYING | VARYINGZ | NONVARYING]			
タスク	TASK	ALIGNED | UNALIGNED		

配列: DIMENSION は任意の変数の宣言に追加できます。詳細については、 181 ページの『配列』を参照してください。

構造体および共用体

- 主要な構造体および共同体の場合: 有効範囲、ストレージ (INITIAL 以外)、位置合わせ、 STRUCTURE または UNION、および LIKE 属性を指定することができます。
- 構造体あるいは共同体であるメンバーの場合: 位置合わせ、STRUCTURE または UNION、および LIKE 属性を指定することができます。
- メンバーは、常に INTERNAL 有効範囲属性を持ちます。

詳細については、 185 ページの『構造体』 および 186 ページの『共用体』を参照してください。

注:

1. 宣言しなかった名前や、データ・タイプを指定せずに宣言した名前は、コード化算術変数と見なされます。デフォルト属性は、175 ページの『属性のデフォルト』に記載されています。示されるデフォルトは、IBM デフォルトです。 ANS デフォルトは、FLOAT と DECIMAL ではなく FIXED と BINARY です。
2. POSITION は、ストリング・オーバーレイ定義を使用した場合だけ使用することができます。

計算データ・タイプとその属性

この節では、計算データとして分類されるデータ・タイプ、およびそれに関連づけられる属性を説明します。

コード化算術データとその属性

コード化算術データの一般情報については、 24 ページの『データ・タイプとその属性』を参照してください。

コード化算術データの構文を次の図に示します。

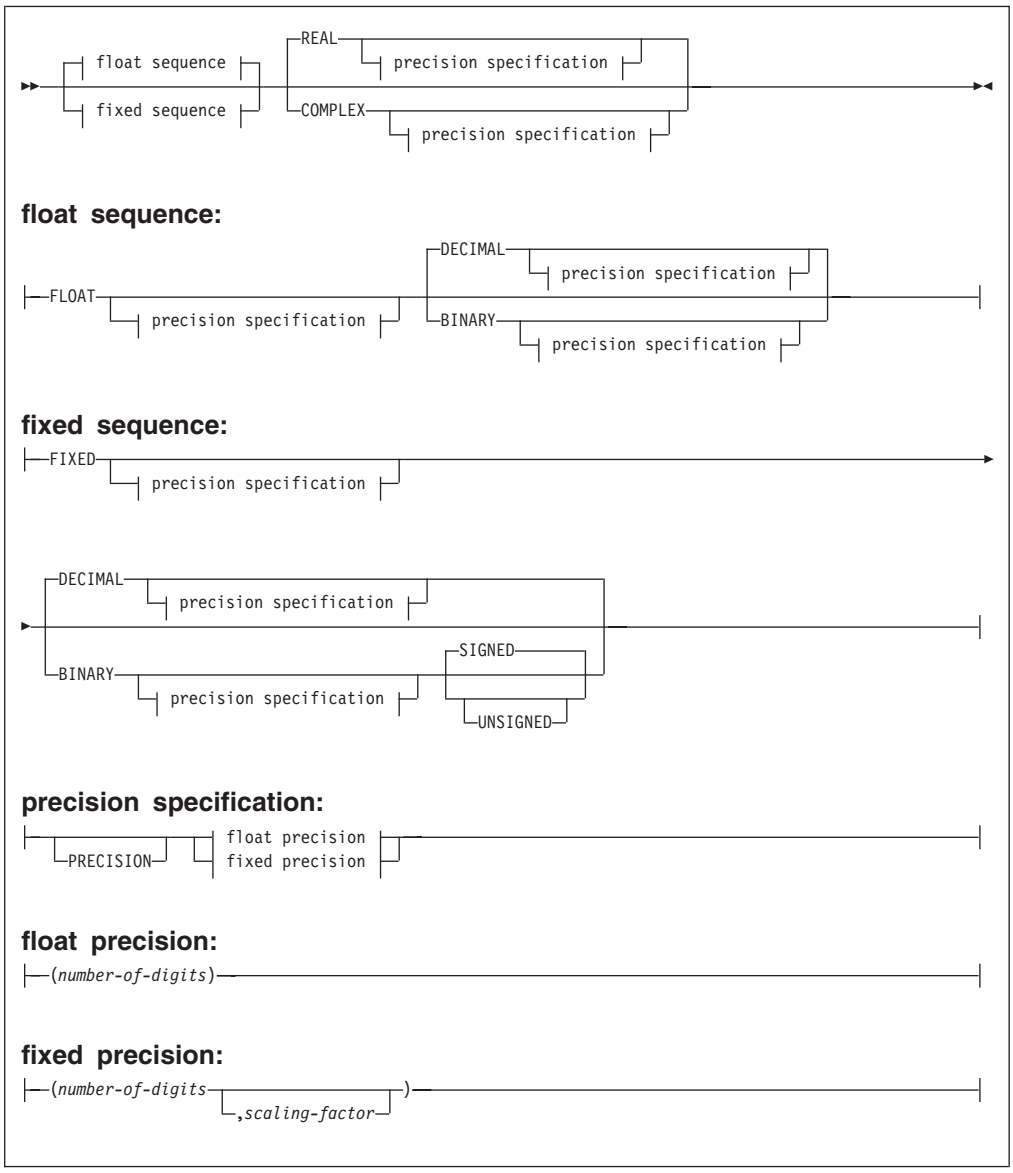


表 9. コード化算術データ属性の省略形

属性	省略形
BINARY	BIN
COMPLEX	CPLX
DECIMAL	DEC
PRECISION	PREC

BINARY 属性と DECIMAL 属性

コード化算術データ項目の*基数* は、10 進数または 2 進数のいずれかです。DECIMAL はデフォルトです。

FIXED 属性と FLOAT 属性

コード化算術データ項目の*スケール* は、固定小数点または浮動小数点のいずれかです。

固定小数点データ項目は有理数であり、その 10 進小数点または 2 進小数点の位置は、定数のときは小数点として示され、変数のときは宣言済みのスケール因数で指定されます。

浮動小数点データ項目は、小数部分と指数部分のフォーマットの有理数です。

PRECISION 属性

コード化算術データ項目の*精度* には、桁数とスケール因数が含まれます。 (スケール因数は、固定小数点項目にのみ使用されます。)

桁数
何桁まで値を持つことができるかを指定する整数です。固定小数点項目の場合、整数は有効数字の数になります。浮動小数点の場合、整数は小数点を排除して保持される有効数字の数になります (小数点の位置によって異なる)。

スケール因数
数値の最右端の桁に関して、10 進または 2 進小数点の想定する位置を指定する整数です。符号は付けても付けなくてもかまいません。スケール因数を指定しない場合、デフォルトは 0 になります。

精度属性の指定を表すとき、通常は (p,q) と書かれています。ただし、*p* は桁数を表し、*q* はスケール因数を表します。

負のスケール因数 (-q) を指定すると、実際の最右端の桁から右方向へ q 桁ずれた位置に小数点がある整数を表します。桁数よりも大きい正のスケール因数 (q) を指定すると、実際の最右端の桁から左方向に q 桁ずれた位置に小数点がある小数を表します。いずれの場合も、間にゼロがあると想定されますが、それらは保管されません。実際に保管されるのは、指定された桁数だけです。

PRECISION を省略する場合は、精度属性を同じ属性分配レベルのスケール (FIXED または FLOAT)、基数 (DECIMAL または BINARY)、またはモード (REAL または COMPLEX) の属性のすぐあとに書き、その間にほかの属性を指定してはなりません。

省略しない場合には、PRECISION は、宣言内の任意の場所に置くことができます。

整数値 は、スケール因数がゼロの固定小数点値を意味します。

REAL 属性と COMPLEX 属性

算術データ項目 (コード化算術データ項目または数字データ項目) の*モード* が実数か複素数のいずれかです。

実数データ項目は、実数の値を表す数です。

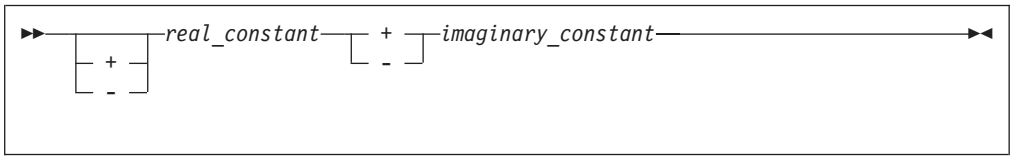
複素数データ項目は、2 つの部分 (実数部分と虚数部分) から構成されます。複素数データ項目を表す変数の場合、実数部分と虚数部分の基数、スケール、および精度は同じです。

算術変数は REAL にデフォルト設定されます。

虚数の定数は、任意のタイプの実定数のすぐ後に文字 I を付けて書きます。以下に例を示します。

```
27I
3.968E10I
11011.01BI
```

これらは、いずれも実数部分がゼロであると見なされます。実数部分がゼロ以外の複合値は、次の構文を使った式で表します。



たとえば、38+27I

下記の y および z という 2 つの複素数が与えられたとき、

```
y = complex(A,B);
z = complex(C,D);
```

x=y/z は、次のように計算されます。

```
real(x) = (A*C + B*D)/(C**2 + D**2);
imag(x) = (B*C - A*D)/(C**2 + D**2);
```

x=y*z は次のように計算されます。

```
real(x) = A*C - B*D;
imag(x) = B*C + A*D;
```

これらの計算の途中で計算条件が起こることもあります。

SIGNED 属性と UNSIGNED 属性

SIGNED 属性と UNSIGNED 属性は、FIXED BINARY 変数と ORDINAL 変数にのみ使用できます。 SIGNED は、変数が負の値と想定できることを示します。 UNSIGNED は、変数が負の値と想定できることを示します。

UNSIGNED は、固定小数点演算のセマンティクスに次のような影響を与えます。

- IAND、IEOR、INOT および IOR のすべてのオペランドが UNSIGNED だと、その結果は UNSIGNED です。
- ISLL と ISRL の第 1 オペランドが UNSIGNED だと、その結果は UNSIGNED です。
- REAL または IMAG のオペランドが UNSIGNED だと、その結果は UNSIGNED です。

RULES(ANS) コンパイラー・オプションを使用している場合、UNSIGNED は、固定小数点演算のセマンティクスに次のような影響を与えます。

- 加算、乗算または除算の両方のオペランドが UNSIGNED だと、その結果は UNSIGNED です。
- MAX または MIN のすべてのオペランドが UNSIGNED だと、その結果は UNSIGNED です。
- REM または MOD のすべてのオペランドが UNSIGNED だと、その結果は UNSIGNED です。

表 10 および表 11 に示されているように、 SIGNED 属性と UNSIGNED 属性はストレージ所要量に影響を及ぼします。

表 10. FIXED BINARY SIGNED データ・ストレージ所要量

精度	ストレージ占有量 (バイト)
精度 <= 7	1
7 < 精度 <= 15	2
15 < 精度 <= 31	4
31 < 精度 <= 63	8

表 11. FIXED BINARY UNSIGNED データ・ストレージ所要量

精度	ストレージ占有量 (バイト)
精度 <= 8	1
8 < 精度 <= 16	2
16 < 精度 <= 32	4
32 < 精度 <= 64	8

2 進固定小数点データ

2 進固定小数点変数を宣言するデータ属性は、BINARY および FIXED です。 以下に例を示します。

```
declare Factor binary fixed (20,2);
```

上記の Factor は、20 個のデータ・ビットの 2 進固定小数点データを表す変数として宣言され、20 桁のうち 2 桁は 2 進小数点の右側のビット数です。

符号付きおよび符号なしの固定 2 進データが占めるストレージについては、 32 ページの『SIGNED 属性と UNSIGNED 属性』を参照してください。

宣言した数のデータ・ビットは、低位桁に入りますが、高位桁の余分な桁位置も、そのデータ項目を使って行われるすべての演算の対象となります。算術オーバーフローがこの余分な高位の桁位置に及んだかどうかが検出されるのは、 SIZE 条件が割り込み可能になっている場合だけです。

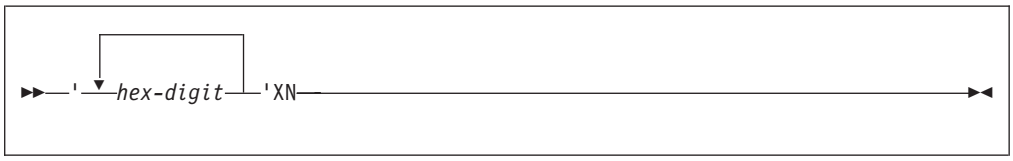
2 進固定小数点定数

2 進固定小数点定数は、オプションの 2 進小数点を持つ 1 つ以上のビット、直後に文字 B によって構成されます。2 進固定小数点定数の精度は (p,q) です。ここで *p* は定数内のデータ・ビットの合計数で、*q* は 2 進小数点の右のビット数です。以下に例を示します。

定数	精度
1011_0B	(5,0)
1111_1B	(5,0)
101B	(3,0)
1011.111B	(7,3)

XN (16 進) 2 進固定小数点定数

XN 定数は、16 進表記の SIGNED REAL FIXED BINARY 定数を表します。定数が 8 桁以下の場合、その精度は 31 です。9 桁以上の場合は精度は 63 です。



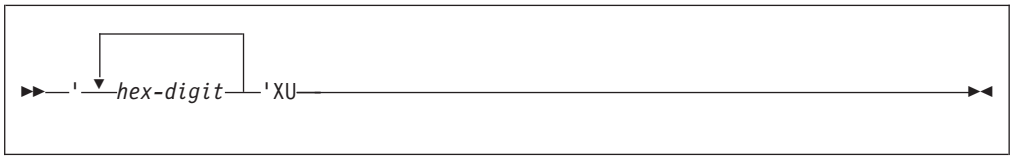
次の例について考えてみます。

```
'100'XN             /*  same as '00000100'XN with value 256      */
'8000'XN            /*  same as '00008000'XN with value 32,768   */
'FFFF'XN            /*  same as '0000FFFF'XN with value 65,535   */
"ffff_ffff"XN       /*  is the value -1                          */
```

必要に応じて、XN 定数の 16 進数値は、指定された値の左側に 16 進のゼロが埋め込まれます。

XU (16 進) 2 進固定小数点定数

XU 定数は、16 進表記の UNSIGNED REAL FIXED BINARY 定数を表します。定数が 8 桁以下の場合、その精度は 32 です。9 桁以上の場合は精度は 64 です。



次の例について考えてみます。

```
'100'XU             /*  same as '00000100'XU with value 256      */
'8000'XU            /*  same as '00008000'XU with value 32,768   */
'FFFF'XU            /*  same as '0000FFFF'XU with value 65,535   */
"ffff_ffff"XU       /*  is the value 2**32-1                     */
```

必要に応じて、XU 定数の 16 進数値は、指定された値の左側に 16 進のゼロが埋め込まれます。

10 進固定小数点データ

10 進固定小数点変数を宣言する場合のデータ属性は、DECIMAL および FIXED です。 以下に例を示します。

```
declare A fixed decimal (5,4);
```

この宣言は、A が 5 桁の 10 進固定小数点データを表すように指定しています。 5 桁のうちの 4 桁は小数点の右側にきます。

次に 2 つの例をあげます。

```
declare B fixed (7,0) decimal;
declare B fixed decimal(7);
```

両方とも、B は 7 桁の整数を示します。

さらに、

```
declare C fixed (7,-2) decimal;
```

これは、C がスケール因数の -2 を示します。つまり、C は少なくとも 7 桁を保持し、 -9999999*100 から 9999999*100 まで (増分は 100) の範囲を意味します。

さらに、

```
declare D decimal fixed real(3,2);
```

この場合は、D が少なくとも 3 桁 (そのうちの 2 桁が小数部分) の固定小数点項目を表すことを指定しています。

10 進固定小数点データは、1 バイトに 2 桁ずつ保管され、最右端バイトの右 4 ビットに符号が保管されます。したがって、10 進固定小数点データ項目は、変数の宣言で桁数 *p* を偶数として指定しても、必ず奇数として保管されます。

偶数の桁数を宣言すると、最高位の桁数に余分な 1 桁分が生じます。この桁位置は、そのデータ項目を使って行われるすべての演算 (たとえば、比較演算) の対象となります。算術オーバーフローまたは割り当てが、この余分な最高位の桁位置にまで及んだかどうかが検出されるのは、SIZE 条件が使用可能になっている場合だけです。

10 進固定小数点定数

10 進固定小数点定数は、1 桁以上の 10 進数で構成されます。小数点はあってもなくても構いません。 10 進固定小数点定数の精度は (p,q) であり、 *p* はその定数の全桁数で、*q* は、小数点の右側に指定された桁数です。以下に例を示します。

定数	精度
3.1416	(5,4)
455.3	(4,1)
732	(3,0)
1_200_300	(7,0)
003	(3,0)
5280	(4,0)
.0012	(4,4)

2 進浮動小数点データ

2 進浮動小数点の変数を宣言するデータ属性は、BINARY および FLOAT です。以下に例を示します。

```
declare S binary float (16);
```

S は、2 進数が 16 桁の精度を持つ 2 進浮動小数点データを表します。

指数は 5 桁の 10 進数を超えることはできません。宣言した精度が (21) 以下であれば、短精度の浮動小数点フォーマットで保管されます。宣言した精度が (21) より

大きく、(53) 以下であれば、長精度の浮動小数点フォーマットが使用されます。宣言した精度が (53) より大きい場合には、拡張精度の浮動小数点フォーマットが使用されます。

2 進浮動小数点定数

2 進浮動小数点定数は、仮数、指数、文字 B の順序で表記します。仮数は、2 進固定小数点定数です。指数は、英字の E、S、D、または Q のあとに 10 進整数 (符号はあってもなくてもよい) を書き、これで 2 の累乗を表します。 E を使用する定数は、精度 (p) があります。*p* は小数部の 2 進数の数字です。 S、D および Q を使用する定数には、必ず最大単精度、最大倍精度、および最大拡張精度があります。以下に例を示します。

定数	精度
101101E5B	(6)
101.101E2B	(6)
11101E-28B	(5)
11.01E+42B	(4)
1S0b	(21)
1D0b	(53)
1Q0b	(64)(**Windows**)
1Q0b	(106)(**AIX**)
1Q0b	(109)(**z/OS**)

10 進浮動小数点データ

10 進浮動小数点の変数を宣言するデータ属性は、DECIMAL および FLOAT です。以下に例を示します。

```
declare Light_years decimal float(5);
```

Light_years の値は、 5 桁の 10 進数を持つ浮動小数点データを表しています。

宣言した精度が (6) 以下であれば、短精度の浮動小数点フォーマットで保管されます。宣言した精度が (6) より大きく、(16) 以下であれば、長精度の浮動小数点フォーマットが使用されます。宣言した精度が (16) より大きい場合には、拡張精度の浮動小数点フォーマットが使用されます。

10 進浮動小数点定数

10 進浮動小数点定数は、小数部のあとに指数が続きます。小数部は、10 進固定小数点定数になります。指数は、E、S、D、または Q のあとに、4 桁以下の 10 進整数 (符号はあってもなくてもよい) を書き、これで 10 の累乗を表します。 E を使用する定数には、精度 (p) があり、この *p* は、小数部の桁数になります。 S、D および Q を使用する定数は、常に単精度、倍精度、および拡張精度を表します。以下に例を示します。

定数	精度
15E-23	(2)
15E23	(2)
4E-3	(1)
1.96E+07	(3)
438E0	(3)
3_141_593E-6	(7)

```
.003_141_593E3          (9)
1s0                     (6)
1d0                     (16)
1q0                     (18)(Windows)
1q0                     (32)(AIX)
1q0                     (33)(z/OS)
```

最後の 5 つの例は、(異なる精度を持っていますが) 同じ値を示しています。

ストリング・データとその属性

ストリングに関する一般情報については、 24 ページの『データ・タイプとその属性』を参照してください。

BIT 属性、CHARACTER 属性、GRAPHIC 属性、および WIDECHAR 属性

BIT 属性は、ビットの変数を指定します。

CHARACTER 属性は、文字変数を指定します。ストリングは、PICTURE 属性を使用して宣言することができます。

WIDECHAR 属性は、UTF-16 データを保持するワイド文字変数を指定します。

GRAPHIC 属性は、グラフィック変数を指定します。

BIT 属性、CHARACTER 属性、GRAPHIC 属性、および WIDECHAR の構文は、次のとおりです。

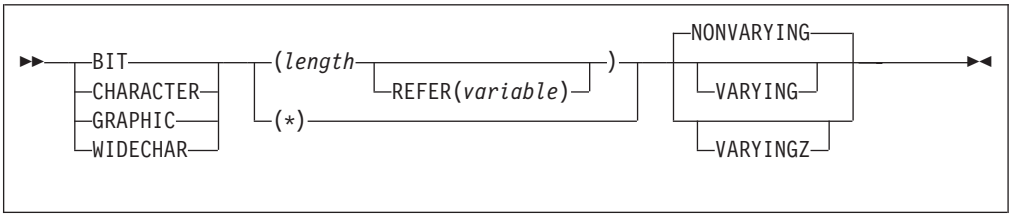


表 12. ストリング・データ属性の省略形

属性	省略形
CHARACTER	CHAR
GRAPHIC	G
WIDECHAR	WCHAR
NONVARYING	NONVAR
VARYING	VAR
VARYINGZ	VARZ

長さ

NONVARYING ストリングの長さ、あるいは VARYING ストリングまたは VARYINGZ ストリングの最大長を指定します。長さは、場合に応じて、ビット、文字、ワイド文字、またはグラフィック (DBCS 文字) 単位になります。

長さは式またはアスタリスクとして指定できます。長さが指定されていない場合には、デフォルトは 1 となります。名前のついた定数の場合には、長さは式の値の長さで判別します。

パラメーターの場合は、パラメーターが CONTROLLED であるときだけ式は有効です。パラメーターにアスタリスクで指定すると、渡される引数から長さが取られます。

長さを式で指定すると、変数にストレージが割り振られるときにその式が計算され、 FIXED BINARY(31,0) に変換されます。変換後の値は正でなければなりません。

STATIC データの場合には、式は、制限付きの式でなければなりません。

BASED データの場合には、ストリングが構造体または共用体のメンバーであり、かつ REFER オプションが使用されていない限り、長さは制限付き式でなければなりません。

REFER

REFER オプションに関しては、 253 ページの『REFER オプション (自己定義データ)』を参照してください。

User を長さ 15 の文字データを表すことができる変数として宣言するステートメントを以下に示します。

```
declare User character (15);
```

以下の例は、ビット変数の宣言を示します。

```
declare Symptoms bit (64);
```

VARYING 属性、VARYINGZ 属性、NONVARYING 属性

VARYING 属性と VARYINGZ 属性は、変数の長さが 0 から宣言済み最大長までの範囲の可変長であることを指定します。 NONVARYING は、宣言した長さと等しい長さを常に持つ変数を指定します。

VARYING ストリングに割り振られるストレージは、宣言した長さよりも 2 バイト長くなります。最左端の 2 バイトには、ストリングの現在長が入ります。

VARYINGZ 文字ストリングに割り振られたストレージは、宣言した長さよりも 1 バイト長くなります。ストリングの現在長は、それに割り振られたストレージの最初の '00'x の前にあるバイト数に等しくなります。

VARYINGZ GRAPHIC ストリングに割り振られるストレージは、宣言した長さよりも 2 バイト長くなります。ストリングの現在長は、それに割り振られたストレージの最初の '0000'gx の前にあるバイト数に等しくなります。

VARYINGZ WIDECHAR ストリングに割り振られるストレージは、宣言した長さよりも 2 バイト長くなります。ストリングの現在長は、それに割り振られたストレージの最初の '0000'wx の前にあるバイト数に等しくなります。

VARYINGZ 属性は BIT ストリングには使用できません。

下記の DECLARE ステートメントで、User と Zuser は、最大長が 15 の可変長文字データを表しています。ただし、User とは異なり、 Zuser は NULL 終止です。割り振られたストレージは、User の場合は 17 バイトであり、 Zuser の場合は 16 バイトです。

```
declare User character (15) varying;
declare Zuser character (15) varyingz;
```

User と Zuser の長さは、それぞれの時点でそれぞれに割り当てられるデータ項目の長さとなります。 MAXLENGTH 組み込み関数および LENGTH 組み込み関数を使用すると、それぞれ宣言した長さと現在長を判別することができます。

VARYINGZ ストリング中の NULL 終止符は、ストリングの長さを判別する場合を除いては、比較または割り当てには使用されません。したがって、下記の宣言によるストリングの内部 16 進表示は同じですが、比較すれば等しくは**なりません**。

```
declare A char(4) nonvarying init( ('abc' || '00'x) );
declare B char(3) varyingz   init( 'abc' );
```

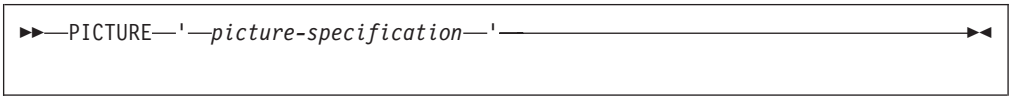
それとは反対に、次の例では、*Z* と *C* は等しいものとして比較されます。

```
dcl Z char(3) nonvarying init('abc');
dcl C char(3) varyingz init('abc');
```

VARYING と VARYINGZ ストリングは、アスタリスクの長さを持つパラメーターとして受け渡しできます。それらが NONASSIGNABLE 属性を持っている場合、記述子なしで渡すことができます。

PICTURE 属性

PICTURE 属性は、ピクチャー文字とデータ項目内の各文字位置に関連付けることにより、文字データ項目の特性を指定します。ピクチャー文字は、その位置を占有できる文字のカテゴリーを指定します。



省略形 PIC

picture-specification
文字データ項目または数字データ項目のいずれかを記述します。有効な文字については、 342 ページの『文字データ用のピクチャー文字』 または 343 ページの『数字データ用のピクチャー文字』を参照してください。

数字ピクチャー指定は、定数の表示によって指定する場合と同じように、数字データの算術属性を指定します。

数字データは、算術値を持ちますが、文字フォーマットで保管されます。数字データは、算術演算が行われる前にコード化算術データに変換されます。

数字データ項目の基数は、10 進数です。そのスケールは、固定小数点または浮動小数点のいずれかです (K または E というピクチャー文字は浮動小数点スケールを表します)。数字データ項目の精度は、有効数字の桁数です (浮動小数点については指数を除きます)。有効数字は、桁位置および条件式のピクチャー文字で指定されま

す。数字データ項目のスケール因数は、V または F のピクチャー文字、もしくは V と F の組み合わせに基づいて決められます。

10 進データだけは、ピクチャー文字で表示することができます。複素数データを宣言するには、固定小数点または浮動小数点データ項目を記述する 1 つのピクチャー指定と共に COMPLEX 属性を指定します。

数字データの詳細については、 45 ページの『数字データ』を参照してください。

文字データ

CHARACTER 属性のついたデータには、文字セットによってサポートされる 256 個のどの文字も入れることができます。 PICTURE 属性のついたデータは、ピクチャー指定文字と一致する文字を持たなければなりません。それぞれの文字は 1 バイトのストレージを占有します。

文字定数

文字定数は、連続した一連の文字を単一引用符または二重引用符で囲んだものです。

定数内に含まれる引用符は、 24 ページの『引用符の使用』でリストされる規則に従います。文字定数の長さは、開始引用符と閉じ引用符の間にある文字の個数です。ただし、2 つの引用符は、1 文字として数えられます。

ヌル文字定数とは、間にブランクを入れずに引用符を 2 つ書いたものです。文字定数の構文は、以下のとおりです。

文字定数の例は以下のとおりです。

定数	長さ
`'Shakespeare''s "Hamlet"'`	22
`"Shakespeare's ""Hamlet"""`	22
`"Page 5"`	6
`'/* This is a comment */'`	27
`''`	0
`(2)'Walla '`	12

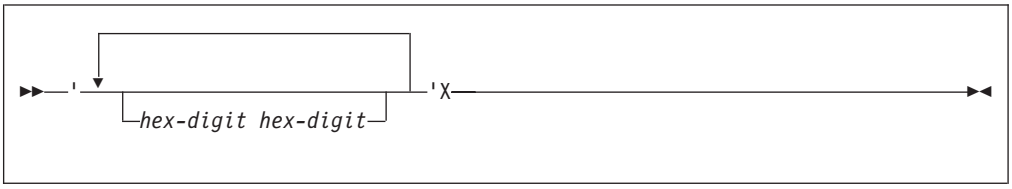
最後の例の、括弧内の数は*ストリング反復因数* であり、そのあとにある一連の文字を指定回数だけ反復することを表します。この例は、定数 `"Walla Walla "` と同等です。ストリング反復因数は、定数であり、括弧で囲まれなければなりません。

X (16 進数) 文字定数

X という文字定数は、単一または二重引用符で囲まれている 16 進数が偶数個連続した列のことです。そのすぐ後ろに X という英字が続きます。各組みの 16 進数は、1 つの文字を表します。

X 定数の長さは、指定された 16 進数の数の半分になります。

ヌルの X 定数は 2 つの引用符として書かれ、そのあとに接尾部 X が続きます。



X 定数の例は以下のとおりです。

定数	長さ
"0d0A"x	2
''X	0

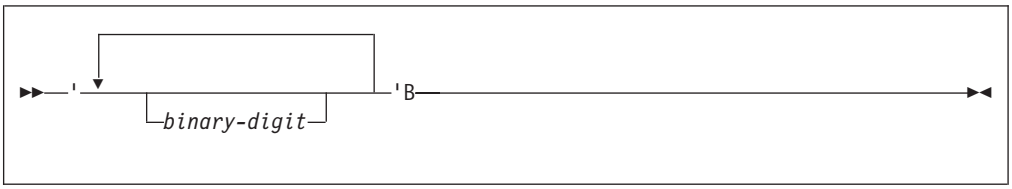
注: X 定数を使用すると、プログラムの移植性が制限されることがあります。

ビット・データ

BIT 属性を持つデータでは、ビット単位でストレージの操作を行うことができます。ストレージの 1 バイトは 8 ビットで構成されます。

ビット定数

ビット定数は、単一または二重引用符で囲まれた一連の 2 進数の列であり、そのすぐ後ろに B が続きます。



ヌルのビット定数は、2 つの引用符で書かれ、そのあとに B が続きます。

ビット定数の例を示します。

定数	長さ
'1'B	1
"1100_1010_11"B	10
(64)'0'B	64
''B	0
'0'B	1

3 番目の例の括弧内の数字はストリング反復因数であり、指定した回数だけ後続のビットを反復することを示します。上記の例では、64 個のゼロのビットからなるストリングになります。

(ビット・データから文字データへの変換、および文字データからビット・データへの変換に関する説明については、 82 ページの『ソースからターゲットへの変換規則』を参照してください。)

B4 (16 進) ビット定数

B4 ビット定数は、単一引用符または二重引用符で囲まれた一連の 16 進数の列であり、そのすぐあとに B4 が続きます。各 16 進数は、4 つのビットを表します。BX は、B4 の同義語です。

B4 定数の例をいくつか示します。

'CA'B4	は	"1100_1010"B と同等
'80'B4	は	'1000_0000'B と同等
'1'B4	は	'0001'B と同等
(2)'F'B4	は	'1111_1111'B と同等
(2)'F'B4	は	'FF'BX と同等
''B4	は	""B と同等

B3 (8 進) ビット定数

B3 ビット定数は、単一引用符または二重引用符で囲まれた一連の 8 進数の列であり、そのすぐ後に B3 が続きます。各 8 進数は、3 つのビットを表します。

B3 定数の例をいくつか示します。

'22'B3	は	"010_010"B と同等
'40'B3	は	'100_000'B と同等
'1'B3	は	'001'B と同等
(2)'7'B3	は	'111_111'B と同等
''B3	は	""B と同等

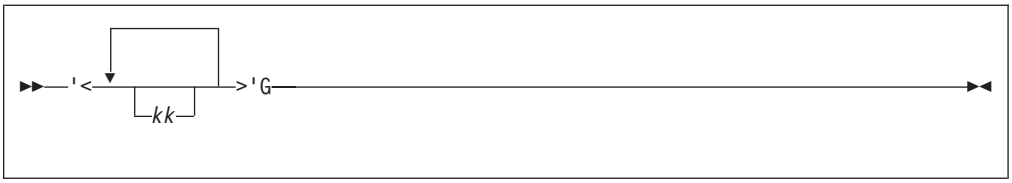
グラフィック・データ

GRAPHIC データには、すべての DBCS 文字を入れることができます。各 DBCS 文字は、2 バイトのストレージを占有します。

グラフィック定数

グラフィック定数は、単一引用符または二重引用符で囲まれた一連の DBCS 文字の列です。グラフィック定数は、定数内の各 DBCS 文字ごとに最大 2 バイトのストレージを使用します。

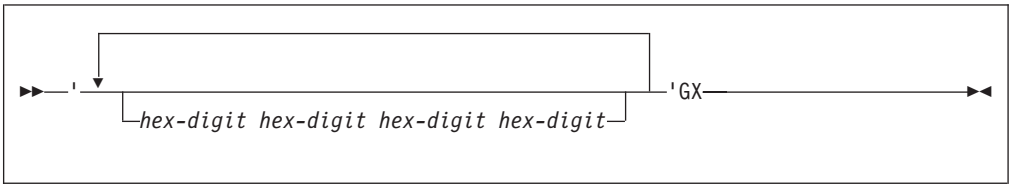
G リテラルは、前後を DBCS 引用符で囲んで指定でき、またその場合は G 全体を DBCS で指定することもできます。



GRAPHIC コンパイラー・オプションは、受け入れられるグラフィック定数に対して有効でなければなりません。 GRAPHIC ENVIRONMENT オプションがグラフィック定数を含む STREAM 入出力ファイルに指定されていない場合には、CONVERSION 条件が生じます。

GX (16 進) グラフィック定数

GX グラフィック定数は、単一引用符または二重引用符で囲まれた一連の 16 進数の列 (4 の倍数) であり、そのすぐあとに GX が続きます。 4 桁の 16 進の各グループは、1 つの DBCS 文字を表します。



例:

```
'81a1'gx        1 個の DBCS 文字を表します。
""gX            ''g と同じです。
```

注: GX を使用すると、プログラムの移植性が制限されることがあります。

混合文字データ

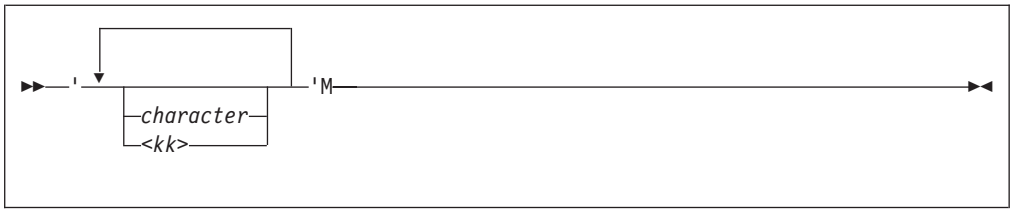
混合文字データには、SBCS および DBCS 文字を入れることができます。混合データは、CHARACTER データ・タイプによって表され、CHARACTER データの処理規則にしたがいます。

OPTIONS 属性の CHARGRAPHIC オプション、および MPSTR 組み込み関数は、混合データ処理を援助するために使用することができます。 CHARGRAPHIC についての詳細は、 135 ページの『OPTIONS オプションとその属性』を参照してください。 MPSTR については、 471 ページの『MPSTR』を参照してください。

M (混合) 文字定数

M 定数は、引用符 (単一または二重) で囲まれた DBCS または SBCS 文字 (あるいはその両方) の連続したシーケンスであり、そのすぐ後に M という英字が続きます。定数に含まれる引用符は、 24 ページの『引用符の使用』にリストされている規則に従います。定数 M の長さは、引用符の間にある SBCS 文字の数です。ただし、2 つの単一引用符は 1 文字として計算され、ストリング内の DBCS 文字の数を 2 倍にして計算します。

ヌルの M 定数は、2 つの引用符で書かれ、そのあとに M が書かれます。



混合文字定数の例は、以下のとおりです。

定数	長さ
'IBM<kk>'M	z/OS では 7 バイト、その他のプラットフォームでは 5 バイト
'<.I.B.M>'M	z/OS では 8 バイト、その他のプラットフォームでは 6 バイト
''M	0

GRAPHIC コンパイラー・オプションは、受け入れられる混合定数に有効でなければなりません。 GRAPHIC ENVIRONMENT オプションは、混合定数を持つ STREAM 入出力ファイルに指定されていない場合は、CONVERSION 条件が生じます。

z/OS では、以下の追加の規則が混合定数に適用されます。

- シフトアウト/シフトインのペアは合致しなければなりません。ペアをネストすることはできません。
- DBCS 部分には、どちらのバイトにも '0E'x または '0F'x が含まれていてはなりません。
- 文字部分には、シフト・コードとして特に意図されていない限り、値 '0E'x または '0F'x が含まれていてはなりません。

注: シフト・コードは z/OS でのみ使用されるため、混合データや M 定数を使用するとプログラムの移植性が制限されることもあります。

ワイド文字データ

WIDECHAR データには、すべての UTF-16 文字を入れることができます。各ワイド文字は、2 バイトのストレージを占有します。

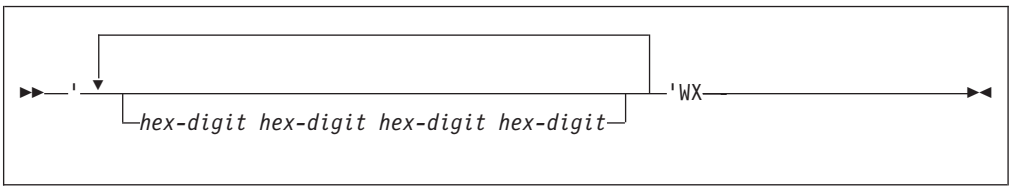
以下については、まだサポートされていません。

- ソース・ファイル内の WIDECHAR 文字
- W ストリング定数
- ストリーム入出力内の WIDECHAR 式の使用
- レコード入出力での WIDECHAR からの暗黙の型変換または WIDECHAR への暗黙の型変換
- レコード入出力での暗黙の endianness フラグ

  WIDECHAR ファイルを作成する場合は、ファイルの最初の 2 バイトとして endianness フラグ ('fe_ff'wx) を書き込んでください。

WX (16 進) ワイド文字定数

WX ワイド文字定数は、単一引用符または二重引用符で囲まれた一連の 16 進数の列 (4 の倍数) であり、そのすぐあとに WX が続きます。 4 桁の 16 進の各グループは、1 つの UTF-16 文字を表します。



例:

```
'0031'wx        1 個の UTF-16 文字を表します。
""wX            ''w と同じです。
```

注: WX 定数は、ビッグ・エンディアン・フォーマットで指定する必要があります (プログラムをリトル・エンディアンフォーマットで実行する場合でも同様です)。このため、たとえば文字 '1' のワイド文字値は、常に '0031'wx ('3100'wx でなく) として指定する必要があります。

注: WX を使用すると、プログラムの移植性が制限されることがあります。

数字データ

数字データ項目は、数字ピクチャー指定と PICTURE 属性を使用して宣言された変数の値です。このデータ項目は、10 進固定小数点の値または 10 進浮動小数点の値を文字で表記したものです。

数字ピクチャー指定は、算術値に変換できるデータだけに割り当てられる文字ストリングを表します。

以下に例を示します。

```
declare Price picture '999V99';
```

Price に割り当てられる値は、5 桁の 10 進数からなる文字ストリングで、仮想小数点は右端から 2 桁目の前にあることを指定します。 Price に割り当てられるデータは、この仮想小数点を基準にして、10 進固定小数点データの場合の小数点位置合わせと同じ方法で位置合わせされます。

数字データは算術属性を持ちますが、コード化算術フォーマットでは保管されません。文字ストリングとして保管されます。したがって、このデータは、10 進固定小数点フォーマットまたは 10 進浮動小数点フォーマットに変換されない限り、算術演算に使用できません。このような変換は自動的に行われますが、余分な処理時間が必要になります。

数字データはストリングのように文字フォーマットで表され、コード化算術データのように小数点位置合わせが行われますが、その処理方法は、コード化算術項目またはストリングのいずれとも異なります。 編集文字を数字データ項目内に挿入するように指定することができ、実際にデータ項目内に保存することができます。したがって、その項目を文字ストリングとして印刷または処理する場合には、代入演算に編集文字も含まれることになります。ただし、数字データ項目を別の数字または

算術変数に割り当てる場合は、編集文字は代入演算には含まれません。割り当てられるのは、実際の数字、符号、および想定小数点の位置だけです。 以下に例を示します。

```
declare Price picture '$99V.99',
        Cost character (6),
        Amount fixed decimal (6,2);
Price = 12.28;
Cost = '$12.28';
```

PRICE のピクチャー指定では、通貨記号 ($) と小数点 (.) が編集文字です。これらの文字は、データ項目内の文字として保管されます。これは、算術値の一部ではありません。両方の代入ステートメントが実行されると、 Price と Cost の実際の内部文字表記は同じであると見なされることがあります。印刷される場合は、全く同じに印刷されます。ただし、常に同じ方法で機能するわけではありません。 以下に例を示します。

```
Amount = Price;
Cost   = Price;
Amount = Cost;
Price  = Cost;
```

最初の 2 つの代入ステートメントが実行されると、 Amount の値は 0012.28 となり、 Cost の値は '$12.28' となります。 Amount への Price の割り当てにおいて、通貨記号と小数点は編集文字であり、割り当ての一部ではありません。 Price の数値は、内部コード化算術フォーマットに変換されます。ただし、Price を Cost に割り当てるときは、文字ストリングへの割り当てなので、数字ピクチャー指定の編集文字も常に割り当ての対象になります。 Price は文字フォーマットで保管されているので、変換は必要ありません。

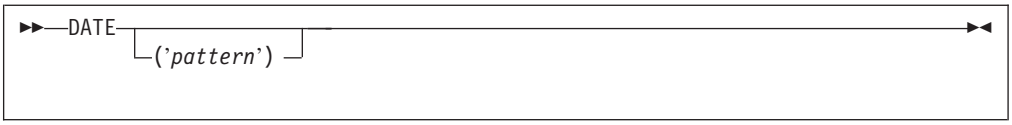
3 番目と 4 番目の代入ステートメントを実行しようとすると、CONVERSION 条件が起こります。ストリング内に通貨記号があれば算術定数としては有効でなくなるため、 Cost の値を Amount に割り当てることはできません。これと同じ理由で、Cost の値を Price に割り当てることもできません。数字ピクチャー指定付きで宣言した変数に割り当てることができるのは、算術タイプの値か、算術タイプに変換できる値だけです。

小数点は編集文字でもあり、文字ストリング内の実際の文字にもなりますが、算術フォーマットへの変換で CONVERSION 条件を引き起こしません。なぜなら、算術定数内に小数点があってもよいからです。正符号 (+) と負符号 (-) についても同様で、算術フォーマットに変換すると算術定数の先頭に符号が付くからです。

数字ピクチャー指定では、その他の編集文字 (ゼロ抑制文字、浮動文字、挿入文字など) を使用することができます。ピクチャー文字の詳細については、341 ページの『第 15 章 ピクチャー指定文字』を参照してください。

日付属性

2 つのオペランドが DATE 属性を持っている場合、コンパイラーは、暗黙の日付比較と変換を行います。 DATE 属性は、変数 (または引数または戻り値) が日付を指定したパターンで保持することを指定します。



パターン
サポートされている日付のパターンの 1 つです。パターンを指定しない場合、デフォルトの値 YYMMDD になります。

DATE 属性は、以下の属性の集合の 1 つを持つ変数がある場合のみ有効です。
- CHAR(n) NONVARYING
- PIC'(n)9' REAL
- FIXED DEC(n,0) REAL

n の長さまたは精度は、 *pattern* の長さと一致しなければなりません。

RESPECT コンパイル時オプションを指定すると (詳細は「*プログラミング・ガイド*」を参照)、次の事柄が起こります。
- DATE 属性を重視することをコンパイラーは認識する。
- DATE 組み込み関数は、属性 DATE('YYMMDD') を持つ値を戻す。

これで、エラー・メッセージが生成されなくても、属性 DATE('YYMMDD') を持つ変数に DATE() を割り当てることができます。 DATE() が DATE 属性を持たない変数に割り当てられる場合には、必ずエラー・メッセージが生成されます。

暗黙の DATE 比較: DATE 属性を指定して宣言された 2 つの変数を比較する場合、 DATE 属性は暗黙の*共通化* を引き起こします。 1 つの変数のみが DATE 属性を持っている場合の比較には、フラグが付けられ、もう一方の被比較数は、通常同じ DATE 属性を持っている場合と同様に処理されます (適用されるいくつかの例外については、後述します)。

暗黙の共通化とは、共通の比較可能な表現に日付を変換するためにコンパイラーがコードを生成することを意味します。このプロセスでは、WINDOW コンパイル時オプションで指定する *window (限定時間帯)* を使用して、2 桁の年を変換します。

次のコード・フラグメントでは、 DATE 属性を有効と見なした場合、 2 番目の DISPLAY ステートメントにある比較は時間帯が限定されます。これは、window を 1900 で開始した場合、この比較が偽を戻すということです。しかし、window を 1950 で開始する場合、この比較は真を戻します。

```
dcl a   pic'(6)9' date;
dcl b   pic'(6)9' def(a);
dcl c   pic'(6)9' date;
dcl d   pic'(6)9' def(c);

b = '670101';
d = '010101';

display( b || ' < ' || d || ' ?' );
display( a < c );
```

日付比較は、以下の位置で発生することがあります。
- IF ステートメントと SELECT ステートメント
- WHILE 文節または UNTIL 文節
- TO 文節が原因で起こる暗黙の比較

同様のパターンを持つ日付の比較: 以下の条件化では、コンパイラーは同一のパターンを持つ日付を比較する特別なコードを何も生成しません。
- 比較演算子 = または ¬= が使われている。
- パターンが、YYYY、YYYYMM、YYYYDDD、または YYYYMMDD と等価である。

異なるパターンを持つ日付の比較: 異なるパターンの日付を含む比較の場合、共通の比較可能な表現に日付を変換するために、コンパイラーはコードを生成します。変換が行われてから、コンパイラーは 2 つの値を比較します。

DATE 属性とリテラルを含む比較: 1 つの被比較数が DATE 属性を持っており、他方の被比較数がリテラルの場合に比較を行うと、コンパイラーは W レベルのメッセージを発行します。次のコンパイラーの動作は、以下に示すようにリテラルの値によって異なります。
- リテラルが有効な日付で表される場合、DATE 属性を持つ被比較数と同じ日付パターンと window を持っている場合と同様に処理されます。
- リテラルが有効な日付で表されない場合、他方の被比較数の DATE 属性は無視されます。

```
dcl start_date char(6) date;
if start_date >= '' then /* no windowing */
...
if start_date >= '851003' then /* windowed */
...
```

DATE 属性と非リテラルを含む比較: 1 つの被比較数が DATE 属性を持っており、他方の被比較数が日付でもリテラルでもない場合の比較では、コンパイラーは E レベルのメッセージを発行します。非日付値は、他方の被比較数と同じ日付パターンと window を持っている場合と同様に処理されます。

```
dcl start_date char(6) date;
dcl non_date char (6);

if start_date >= non_date then /* windowed */
...
```

暗黙の DATE 指定: 日付パターンを指定して宣言された 2 つの変数を割り当てる際に、 DATE 属性は暗黙の変換の原因になることもあります。
- ソースとターゲットが、同じ DATE 属性とデータ属性を持っている場合、どちらも DATE 属性を持っていない場合と同様に、割り当てが進行します。
- ソースとターゲットが異なる DATE 属性を持っている場合、コンパイラーはソースの日付を変換するためにコードを生成してから、割り当てを行います。
- ソースは DATE 属性を持っており、ターゲットが DATE 属性を持っていない場合の割り当てでは、コンパイラーは E レベルのメッセージを発行し、 DATE 属性を無視します。
- ターゲットは DATE 属性を持っており、ソースが DATE 属性を持っていない (かつ、ソースがリテラルではない) 場合の割り当てでは、コンパイラーは E レベルのメッセージを発行し、DATE 属性を無視します。
- ターゲットは DATE 属性を持っており、ソースが DATE 属性を持っていない (かつ、ソースがリテラルである) 場合の割り当てでは、コンパイラーは W レベルのメッセージを発行し、DATE 属性を無視します。

```
dcl start_date char(6) date;
start_date = '';
...
```

- ソースでは 4 桁の年を保持していて、ターゲットでは 2 桁の年を保持している場合、ソースはターゲットの window には存在しない年を保持することがあります。この場合、ERROR 条件が発生します。

```
dcl x char(6) date;
dcl y char(8) date('YYYYMMDD');

y = '20600101';

x = y; /* raises error if window is <= 1960 */
```

- DATE 属性は以下の中では、無視されます。
  - デバッガー
  - レコード入出力ステートメント中で実行される割り当て
  - ストリーム入出力ステートメント (GET DATA など) 中で実行される割り当てと変換

ウィンドウ操作ソリューションを選択しなくても、 2 桁または 4 桁の年の両方を操作しなければならないコードがいくつかあることがあります。次の状況では、複数の日付パターンを使用できます。

```
dcl old_date char(6) date('YYMMDD');
dcl new_date char(8) date('YYYYMMDD');

new_date = old_date;
```

日付の診断: PL/I では、以下の場合に*有効な* 割り当てが起こります。

- 式が、引数として、その引数を記述したエントリー・ポイントに渡される。
- 式が RETURN ステートメント内で使用される。

日付変数を以下のように使用すると、フラグが付けられます。

- 次のどちらかを含む (明示的または有効な) 割り当て
  - 日付から非日付
  - 非日付から日付
- 日付に適用される何らかの算術演算
- BY 文節中の日付の使用 (これは算術演算を暗黙指定する)
- 何らかの算術組み込み関数中での日付の使用
- BINARY、DECIMAL、FIXED、FLOAT、または PRECISION を除く、任意の算術組み込み関数中での日付の使用
- 組み込み関数 SUM、PROD、または POLY 中での日付の使用

上記のすべての場合にコードが生成されますが、ウィンドウ操作は起こりません。結果では、DATE 属性は無視されます。

名前付き定数

名前付き定数は、他のデータ属性とともに VALUE 属性を使用して宣言されるスカラー ID です。名前に対するすべての参照は、該当する定数への参照として論理的に処理されますが、属性の完全セットは明示的に宣言されるか、デフォルトが使用されるかのいずれかです。

注: 名前付き定数を使用すると、名前が付いていない場合とは影響が異なってくることがあります。名前付き定数についての属性は、明示およびデフォルト属性

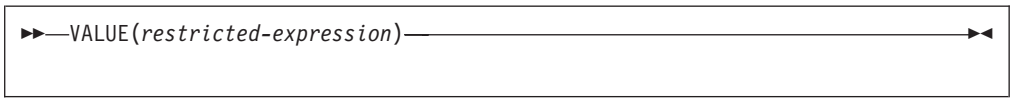
を含む宣言から採用されます。名前の付いていない定数についての属性は、定数の形、書式、およびサイズから推論されます。ストリング・データについては、長さが指定されない場合、またはアスタリスクを使用して指定されている場合、その長さは制限付き式の長さから判別します。

名前付き定数は、アプリケーション・プログラム内で使用するためにより正確でなければなりません。また、それによって、より予測しやすい結果が生じることになります。たとえば、名前が付いた定数 Unit を FIXED BINARY VALUE(1) として定義すると、 FIXED BINARY(15) VALUE(1) という属性を持ちます。単に数字の 1 を使用すると、その属性は FIXED DECIMAL(1,0) になります。起こりうるその他の違いについては、 51 ページの図 1 を参照してください。

また、名前付き定数により、アプリケーションのパラメーターを設定することができます。この結果、デバッグと保守が容易になります。

名前付き定数は、算術データ、ストリング・データとして、およびポインターやオフセットとして宣言することができます。算術データとストリング・データ、ならびにそれらの属性については、それぞれ 37 ページの『ストリング・データとその属性』と 30 ページの『コード化算術データとその属性』を参照してください。名前付き定数は、使用する前に宣言しなければなりません。

VALUE 属性:



restricted expression
式は、計算結果がスカラー値にならなければなりません。制限付きの式に関する詳細については、 76 ページの『制限付き式』を参照してください。

名前付き定数の例: 51 ページの図 1 では、名前付き定数、ならびに名前付き定数と名前の付いていない定数の間に生じる属性や精度の差を示しています。

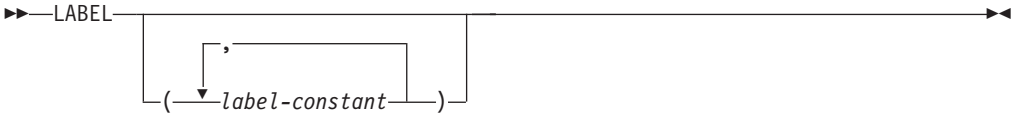
```
Dcl A4 value(148) fixed bin,
    C4 value(261) fixed bin,
    Whole value(800) fixed bin;
Dcl Notes (4) static,
    init(a4,  (Whole/4),   /* 148, 200  */
         c4,  (Whole*2));  /* 261, 1600 */


/* note that "Head" gets length equal to length of VALUE   */


Dcl Head char VALUE('Feel the Power of PL/I'); /* char(22) */
Dcl Headsize fixed bin value(length(Head));     /* 22       */
Dcl 1 Head1 static,
      2 * char(Headsize) initial(Head),        /* char(22) */
      2 * char(20) init(''),
      2 * char(5) init('Page '),
      2 Page_number pic 'zz9',
      2 * char(0);
Dcl TwoHeads char(2*Headsize);                /* char(44) */
Dcl Page0 picture 'zz9' value(0);
Dcl MyNullPtr ptr value(ptrvalue('ffff_ffff'xn));


/* Differences in attributes/results of
   named and unnamed constants                                  */


Dcl Pi float bin value (3.1416);  /* is FLOAT BINARY(21) but ... */
3.1416                            /* is FIXED DECIMAL(5,4)       */


Dcl Unit fixed bin value(1);      /* is FIXED BINARY(15) but ... */
1                                 /* is FIXED DECIMAL(1,0)       */
1.0                               /* is FIXED DECIMAL(2,1)       */
1B                                /* is FIXED BINARY(1)          */
0000_0000_0000_001B               /* is FIXED BINARY(15)         */


Dcl Title char(20) value('SCIDS'); /* is CHAR(20)      but ... */
Dcl Title2 char     value('SCIDS');/* is CHAR(5)               */
'SCIDS'                            /* is CHAR(5)               */
```

図 1. 名前付き定数

名前付き定数は、定数が必要なときはいつでも使用することができます。名前付き定数を、プログラムのあとのほうに出てくる制限付き式の中で使用して、独立定数の計算を行うこともできます。

プログラム制御のデータ・タイプとその属性: この節では、プログラム制御データとそれに関連する属性を説明しています。プログラム制御データは、プログラムの実行を制御する値を指定するのに使います。

ラベル・データと LABEL 属性: ラベル とは、ラベル定数またはラベル変数の値です。



ラベル定数のリストが与えられる場合、変数はリストに含まれる定数を常にその値として持つ必要があります。そして、そのリスト内のラベル定数は、ラベル宣言を含むブロック内で既知である必要があります。括弧で囲まれたラベル定数のリストは、ラベル配列に関する LABEL 属性仕様で使用できます。

ラベル定数は、ステートメント (PROCEDURE、ENTRY、PACKAGE、または FORMAT を除く) のラベル接頭部として書かれる名前です。プログラム実行中に、この名前を参照して、プログラム制御をそのステートメントに移すことができます。 (16 ページの『ステートメント』でラベル接頭部の構文を説明しています。) たとえば、以下のコード行では、Abcde がラベル定数です。

```
Abcde: Miles = Speed*Hours;
```

ラベルのついたステートメントは、通常の命令順序に従ってこのステートメントを実行させることもできますし、プログラム内の別の場所から GO TO ステートメントによってこのステートメントに制御権を移動して実行させることもできます。

ラベル変数には、別のラベル変数の値またはラベル定数を割り当てることができます。そのような割り当てが行われると、ソース・ラベルの環境がターゲット・ラベル変数に割り当てられます。ラベルの静的配列が初期値を持つことを宣言した場合、その配列は割り当て不可と見なされます。

GO TO ステートメントで使用されるラベル変数は、その GO TO ステートメントの実行時にアクティブであるブロック内で使われているラベル定数でなければなりません。次の例を考えてみてください。

```
declare Lbl_x label;
Lbl_a:   statement;

⋮
Lbl_b:   statement;

⋮
Lbl_x = Lbl_a;

⋮
go to Lbl_x;
```

Lbl_a と Lbl_b はラベル定数であり、 Lbl_x はラベル変数です。 Lbl_a を Lbl_x に割り当てることにより、 GO TO Lbl_x ステートメントは制御を Lbl_a ステートメントに移します。それ以外の場合には、Lbl_b を Lbl_x に割り当てるステートメントをプログラムに入れることができます。この割り当てにより、Lbl_x への参照は、 Lbl_b への参照と同じになります。この Lbl_x の値は、別の値が割り当てられるまで保持されます。

ラベル変数に無効な値が割り当てられても、エラーが検出されるとは限りません。以下の例では、値 I が基数となっている配列 Z の特定エレメントに伝送が行われます。

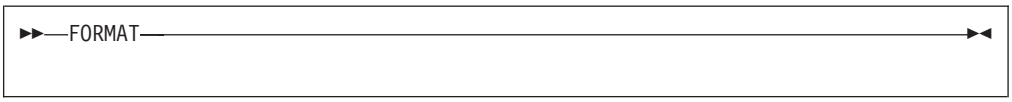
```
go to Z(I);

⋮
Z(1):  if X = Y then return;

⋮
Z(2):  A = A + B + C * D;
```

```
⋮
Z(3):  A = A + 10;
```

Z(2) が省略されると、 I=2 の場合には GO TO Z(I) で ERROR 条件が生じます。 I < LBOUND(Z) または I > HBOUND(Z) のときに GO TO Z(I) を実行した場合、SUBSCRIPTRANGE 条件が使用不能になっていると、予測できない結果が生じます。

フォーマット・データと FORMAT 属性: フォーマット・データ項目とは、フォーマット定数またはフォーマット変数のことです。フォーマット定数は、FORMAT ステートメントのラベル接頭部として書かれる名前です。

FORMAT 属性は、宣言される名前がフォーマット変数であることを指定します。



FORMAT 属性で宣言された名前は、別のフォーマット変数を持つか、またはそれに割り当てられるフォーマット定数を持ちます。そのような割り当てが行われると、ソース・ラベルの環境がターゲット・ラベル変数に割り当てられます。

他の PL/I コンパイラーとの互換性を維持するために、フォーマット変数をラベル変数として宣言することもできます。

次の例を考えてみてください。

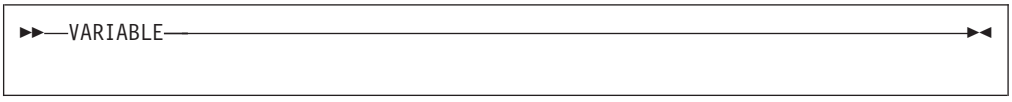
```
Prntexe: format
           ( column(20),A(15), column(40),A(15), column(60),A(15) );
Prntstf: format
           ( column(20),A(10), column(35),A(10), column(50),A(10) );
```

Prntexe と Prntstf はフォーマット定数です。

次の例では、 **4** と **5** は **2** と同じ結果になり、 **6** と **7** は **3** と同じ結果になります。

```
1  dcl Print format;
2  put edit (X,Y,Z) (R(Prntexe) );
3  put edit (X,Y,Z) (R(Prntstf) );
4  Print = Prntexe;
5  put edit (X,Y,Z) (R(Print) );
6  Print = Prntstf;
7  put edit (X,Y,Z) (R(Print) );
```

VARIABLE 属性: VARIABLE 属性は、変数として名前を設定され、ENTRY、FILE および LABEL 変数に対してだけ許可されます。



名前が構造体または共用体のメンバーである場合、または以下のいずれかの属性が指定されている場合、 VARIABLE 属性は暗黙に指定されています。

ストレージ・クラス属性

DIMENSION
PARAMETER
位置合わせ属性
INITIAL

以下の例では、Account1 と Account2 は、ファイル変数であり、File1 と File2 はファイル定数です。

```
declare Account1 file variable,
        Account2 file automatic,
        File1 file,
        File2 file;
```

File1 と File2 は、 Account1 または Account2 に順番に割り振られます。

第 4 章 式および参照

計算の順序 57
ターゲット 58
　変数 58
　疑似変数 58
　中間結果 58
演算式 59
　ポインター演算 59
　算術演算 60
　ビット演算子 67
　比較演算 68
　連結演算 70
　各種演算の組み合わせ 71
配列式 74
　接頭演算子と配列 74
　挿入演算子と配列 74
構造式 75
制限付き式 76
　例 76

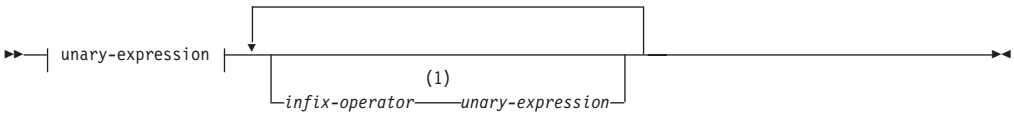
この章では、式および参照のさまざまなタイプを説明しています。

式 は、値を表すものです。式は以下のいずれかになります。

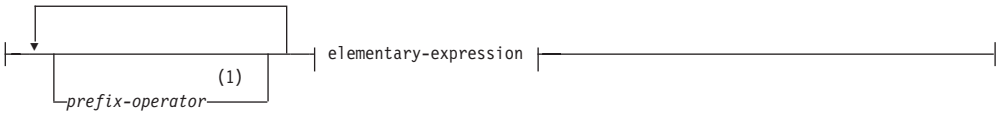
- 単一定数、変数、または関数参照
- 定数、変数、または関数参照を組み合わせたもの。その組み合わせで使用される演算子と括弧を含む。

演算子が含まれている式を*演算式* といいます。 演算式の定数および変数は、*オペランド* と呼ばれます。詳しくは、 59 ページの『演算式』を参照してください。

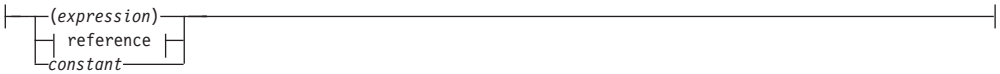
以下の図は、式および参照の構文を示しています。



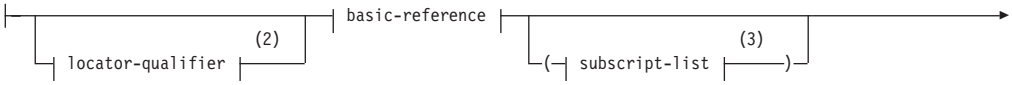
unary-expression:



elementary-expression:

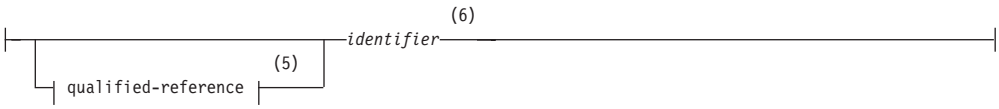


reference:



locator-qualifier:

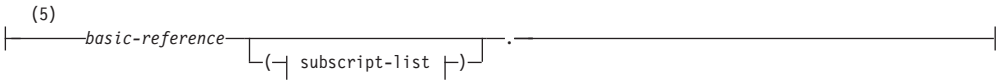
basic-reference:



subscript-list:

argument-list:

qualified-reference:



注:

1 演算子は、 15 ページの表 6 に示されています。

2 ロケーター修飾子は、 248 ページの『ロケーター修飾』と 153 ページの『タイプ付き構造体修飾』で説明されています。

3 添え字は、 181 ページの『配列』で説明されています。

4 引数は、 116 ページの『プロシージャーへの引数の引き渡し』で説明されています。

5 修飾される参照は、 187 ページの『構造体/共用体の修飾』で説明されています。

6 ID は、 13 ページの『ID』で説明されています。

式は*要素式* (スカラー式ともいう)、*配列式*、または*構造式* のいずれかに分類されます。 要素変数および配列変数は、同じ式内に書くことができます。

要素式 単一の値を表します。この定義には、構造体や共用体内の基本名や、配列の単一エレメントを指定する添え字付きの名前も含まれます。

配列式 値の配列を表します。この定義には、次元属性を持つ構造体や共用体のメンバーが含まれます。

構造式 構造化された値のセットを表します。

以下に例を示します。

```
dcl A(10,10) bin fixed(31),
    B(10,10) bin fixed(31),
    1 Rate,
      2 Primary dec fixed(4,2),
      2 Secondary dec fixed(4,2),
    1 Cost(2),
      2 Primary dec fixed(4,2),
      2 Secondary dec fixed(4,2),
    C bin fixed(15),
    D bin fixed(15);
dcl Pi bin float value(3.1416);
```

以下は要素式です。

```
Pi
27
C
C * D
A(3,2) + B(4,8)
Rate.Primary - Cost.Primary(1)
A(4,4) * C
Rate.Secondary / 4
A(4,6) * Cost.Secondary(2)
sum(A)
addr(Rate)
```

以下は配列式です。

```
A
A + B
A * C - D
B / 10B
```

大半の PL/I ステートメントの構文では、式の計算結果がステートメントの構文規則に合ってさえいれば、式を使うことができます。構文の仕様のあとにテキスト内で特に明記していない限り、修飾されていない語 *expression* (式) または *reference* (参照) はスカラー式を指しています。スカラー式以外の式については、式のタイプが明記してあります。たとえば、*array expression* (配列式) という用語は、スカラー式が無効であることを示します。

構造式の例は次のとおりです。

```
Rate = Rate*2
```

計算の順序

PL/I ステートメントには、複数の式または参照が含まれていることがしばしばあります。特定インスタンス (たとえば、代入ステートメントなど) についての説明があるときを除いて、計算の順序は任意であり、(概念的には) 同時に計算することも可能です。

以下に例を示します。

```
dcl (X,Y,Z) entry returns(float), (F,G,H) float;
F = X( Y(G,H), Z(G,H) );
```

関数 Y と Z は、それらに渡された引数の値を変更することがあります。したがって、X が返す値は、どちらの関数が最初に呼び出されるかに応じて異なります。最初のパラメーターが最初に計算されると推定すべきではありません。最後のパラメーターを最初に計算するほうが望ましい場合もあります。

INC 機能がそれに渡された引数の値を増やし、更新された値を戻すと仮定すると、以下の例では、B(1,2) または B(2,1) は、どちらの添え字が最初に計算されるかによって、結果が決まります。どちらの添え字が最初に計算されるかを推定すべきではありません。

```
dcl B(2,2);
I = 0;
put list ( B( INC(I), INC(I) ) );
```

ターゲット

式または変換の計算結果は、*ターゲット・フィールド* に割り当てられます。ターゲット・フィールドは、変数、疑似変数、または中間結果フィールドのいずれかです。

変数

次のようなステートメントで割り当てを行うとします。

```
A = B;
```

代入記号の左側にある変数 (この例では A) がターゲット・フィールドです。変数に対する割り当ては、ストリーム入出力、DO、DISPLAY、およびレコード入出力ステートメント内でも行うことができます。

疑似変数

疑似変数はターゲット・フィールドを表します。たとえば、以下のようになります。

```
declare A character(10),
        B character(30);
substr(A,6,5) = substr(B,20,5);
```

この代入ステートメントでは、SUBSTR 組み込み関数は、ストリング B の 20 番目の文字から始まる長さ 5 のサブストリングを取り出します。 SUBSTR 疑似変数は、ストリング A の中の場所を表しますが、この場所がターゲット・フィールドです。したがって、A の最後の 5 文字は、 B の 20 番目から 24 番目の文字で置き換えられます。 A の最初の 5 文字は、変更されず、もとのままです。

疑似変数については、 395 ページの『第 19 章 組み込み関数、疑似変数、およびサブルーチン』で説明します

中間結果

式が計算されるとき、通常のターゲット属性は、ソース、実行される演算、および第 2 オペランドの属性から部分的に取り出されます。デフォルトが使用されたり、インプリメンテーション制限事項および規則 (たとえば、最大の精度など) が存在していることもあります。これ以降に演算が行われると、中間の結果が変換される場合があります。式の値が計算されたあと、変数または疑似変数に割り当てるために、その結果さらに変換されることもあります。これらの変換は、プログラマー定義のデータが変換される場合と同じ規則に従って行われます。たとえば、以下のようになります。

```
declare A character(8),
        B fixed decimal(3,2),
        C fixed binary(10);
A = B + C;
```

式 B + C の計算時およびその結果の割り当て時に、4 つの異なる結果が生じます。

1. 中間結果。B が 2 進数に変換されて、ここに割り当てられます。
2. 中間結果。2 進数の加算結果がここに割り当てられます。
3. 中間結果。2 進数の結果が 10 進固定小数点フォーマットに変換されて、ここに割り当てられます。
4. A。最終的に結果を入れるためのフィールド。10 進固定小数点の値が文字フォーマットに変換されて、ここに割り当てられます。

最初の結果の属性は、ソース・フィールド B の属性、演算子、および一方のオペランドの属性によって決まります。算術挿入演算子の一方のオペランドが 2 進数であれば、もう一方のオペランドは、演算の前に 2 進数に変換されます。

2 番目の結果の属性は、ソース・フィールド (C と変換された表示の B) の属性から判別します。

3 番目の結果の属性は、ソース・フィールド (2 番目の結果) と最終ターゲット・フィールド A の属性とのかねあいによって決まります。最終ターゲット・フィールドによって決められる属性は、DECIMAL だけです。これは、2 進算術データを文字値に変換するには事前 10 進フォーマットに変換しておかなければならないためです。

A の属性は、DECLARE ステートメントによって決まります。

演算式

演算式は、1 つまたは複数の単一演算で形成されます。単一演算とは、*接頭演算子演算* (単一オペランドの前に演算子がある) または*挿入演算子演算* (2 つのオペランドの間に演算子がある) のどちらかです。 挿入演算子演算の 2 つのオペランドは、通常は、演算実行時にはデータ・タイプが同じでなければなりません。

PL/I では、式に含まれている演算のオペランドは、演算が行われる前に、必要に応じて同じデータ・タイプに変換されます。変換の詳しい規則については、77 ページの『第 5 章 データ変換』で説明します。

1 つの式で異なるデータ・タイプを使用することに関しては、ほとんど制約はありません。ただし、各種のデータ・タイプが混在しているときは変換が行われます。実行時に変換が行われると、プログラムの実行時間が長くなります。さらに、変換によって精度が失われることもあります。データ・タイプが混在している式を使うときは、関係のある変換規則を理解しておいてください。

演算には、ポインター演算、算術演算、ビット演算、比較演算、および連結演算という 5 つのクラスがあります。

ポインター演算

以下のポインター演算を利用できます。

- ポインター式に式を追加したり、またはそこから式を引いたりします。式のタイプは、計算式でなければなりません。必要に応じて、非ポインター・オペランドが FIXED BINARY(31,0) に変換されます。たとえば、以下のようになります。

```
Ptr1 = Ptr1 - 16;
Ptr2 = Ptr1 + (I*J);
```

  これらの演算を行うために、POINTERADD 組み込み関数を使用することもできます。 結果をロケーター参照として使用する場合には、POINTERADD を使用しなければなりません。たとえば、以下のようになります。

```
(Ptr1 + 16) -> Based_ptr            is invalid

pointeradd(Ptr1,16) -> Based_ptr    is valid
```

- ポインターからポインターを引き、ポインター間でその論理的な差を計算します。結果は、FIXED BINARY(31,0) 値になります。

```
Bin31 = Ptr2 - Ptr1;
```

- 挿入演算子を使用して、ポインター式を比較します。

```
if Ptr2 > Ptr1 then
   Bin31 = Ptr2 - Ptr1;
```

- BINARYVALUE 組み込み関数を使用して、算術コンテキストにおいてポインター式を使用します。

```
Bin31 = Bin31 + binaryvalue(Ptr1);
```

- 組み込み関数の POINTERVALUE を使用して、ポインター・コンテキストで計算による式を使用します。

```
dcl 1 Cvtprt pointer based(pointervalue(16));
dcl 1 Cvt based(Cvtptr),
      2 Cvt ...;
```

  必要であれば、式は FIXED BINARY(31,0) に変換されます。

PL/I ブロックでは、ポインター算術を使用して、構造体または配列変数内の任意のエレメントにアクセスすることができます。ただし、ブロックは、構造体あるいは配列変数の入っているものに渡さなければならず、その名前の有効範囲内に参照できる集合を持っていなければなりません。

算術演算

算術演算を指定するには、下記のいずれかの演算子でオペランド同士を結び付けます。

```
+    -    *    /    **
```

正符号 (+) と負符号 (-) は、接頭演算子として使うことも、挿入演算子として使うこともできます。これ以外のすべての算術演算子は、挿入演算子としてのみ使用することができます (ADD、SUBTRACT、DIVIDE、および MULTIPLY 組み込み関数を使用して、算術演算を使用することもできます)。

接頭演算子は、挿入演算子の任意のオペランドの前に付けることができ、そのオペランドと関連があります。たとえば、A*-B という式の場合、負符号 (-) は A の値を B の値の -1 倍で乗じることを示しています。

1 つの変数に複数の接頭演算子を付けることができます。正の接頭演算子を複数付けても、累積的な効果はありませんが、負の演算子を 2 つ続けると、正の接頭演算子を 1 つ使用するのと同じ結果になります。

算術演算におけるデータ変換

算術演算の 2 つのオペランドは、タイプ、基数、モード、精度、およびスケールの点で異なっていても構いません。異なっているときは、以下で説明するように変換が行われます。 (コード化算術オペランドの場合は、62 ページの表 13 を使用して変換方法を判別することができます。各オペランドは、計算結果のタイプ、基数、およびモードに変換されます。計算結果の精度とスケールに変換する必要はありません。)

注: スケールされる FIXED BINARY オペランドは、スケールされる FIXED DECIMAL に変換されてから、それについての演算が行われます。

タイプ: 文字オペランドは、FIXED DECIMAL(N,0) に変換されます。ビット・オペランドは、FIXED BINARY(M,0) に変換されます。 (最大制限については、 575 ページの『制限値』を参照してください。)数字の文字オペランドは、ピクチャー指定によって決まるスケールと精度を持つ DECIMAL に変換されます。

グラフィックとワイド文字の変数とストリングは、すべての計算コンテキストで使用することができます。変換する必要があるときは、従うべき規則は、文字に対しても同様になります。

算術演算の結果は、常に、コード化算術フォーマットになります。算術挿入演算子演算で行われる可能性のある変換は、タイプ変換だけです。

基数: 2 つのオペランドの基数が異なっている場合は、10 進数オペランドが 2 進数に変換されます。

モード: 2 つのオペランドのモードが異なっている場合は、実数オペランドが複素数モードに変換されます。これは、実数部分と同じ基数、スケール、および精度を与えて、虚数部分ゼロを設けるという方法で行われます。ただし、累乗演算で第 2 オペランド (演算における指数) がスケール因数ゼロの固定小数点実数である場合は例外であり、このときは変換は不要です。

精度: 精度やスケール因数だけ (または両方) が異なっている場合は、タイプ変換は不要です。

スケール: 2 つのオペランドのスケールが異なっている場合は、固定小数点オペランドが浮動小数点のスケールに変換されます。ただし、累乗演算で第 1 オペランドが浮動小数点のスケールになっており、第 2 オペランド (演算における指数) がスケール因数ゼロの固定小数点数、つまりは整数または精度 (p,0) として宣言された変数である場合は、例外です。この場合は、変換は必要ではありませんが、結果は浮動小数点数になります。

累乗演算の両オペランドが固定小数点数である場合は、次の変換方法のうちの 1 つが行われます。

- 指数の精度が (p,0) 以外であれば、両オペランドが浮動小数点数に変換されます。

- 指数が符号なし整数でない場合、第 1 オペランドが浮動小数点数に変換されます。
- 固定小数点累乗演算を行うと、使用可能な最大桁数を超えることが両オペランドの精度から明らかになる場合は、第 1 オペランドが浮動小数点数に変換されます。

算術演算の結果

式の中のオペランドが必要に応じて変換されたあと、算術演算が行われ、結果が出されます。この結果は、式の値であることも、さらに後続の演算の対象となる中間結果であることもあります。あるいは、なんらかの条件が起こることもあります。

表 13 と 63 ページの表 14 は、各種の算術演算から帰結する属性と精度を示しています。

67 ページの表 18 は、表 13 と 63 ページの表 14 の右端の欄に注記している累乗演算の特殊事例における結果の属性を示しています。

コンパイラー・オプション RULES(ANS) のもとでは、一方のオペランドが FIXED DECIMAL にスケールされており、他方のオペランドが FIXED BINARY にスケールされていると、FIXED BINARY 値は FIXED DECIMAL に変換されます。 64 ページの表 15 は、コンパイラー・オプション RULES(ANS) のもとでのこの場合の結果の属性および精度を示しています。 RULES コンパイラー・オプションの詳細については、「プログラミング・ガイド」を参照してください。

表 13. 1 つまたは複数の FLOAT オペランドの算術演算の結果

<table>
<tr><th>第 1
オペランド
(p1,q1)</th><th>第 2
オペランド
(p2,q2)</th><th>加算、減算、
乗算または
除算の結果
の属性</th><th>加算
または
減算
精度</th><th>乗算
精度</th><th>除算
精度</th><th>累乗演算
の結果の
属性</th></tr>
<tr><td>FLOAT
DECIMAL
(p1)</td><td>FLOAT
DECIMAL
(p2)</td><td rowspan="3">FLOAT
DECIMAL
(p)</td><td rowspan="6" colspan="3">p = MAX(p1,p2)</td><td rowspan="3">FLOAT
DECIMAL (p)
(特殊な場合以外は、
C が適用される)
p = MAX(p1,p2)</td></tr>
<tr><td>FLOAT
DECIMAL
(p1)</td><td>FLOAT
DECIMAL
(p2,q2)</td></tr>
<tr><td>FIXED
DECIMAL
(p1,q1)</td><td>FLOAT
DECIMAL
(p2)</td></tr>
<tr><td>FLOAT
BINARY
(p1)</td><td>FLOAT
BINARY
(p2)</td><td rowspan="3">FLOAT
BINARY
(p)</td><td rowspan="3">FLOAT
BINARY
(p)
(特殊な場合以外は、
C が適用される)
p = MAX(p1,p2)</td></tr>
<tr><td>FLOAT
BINARY
(p1)</td><td>FIXED
BINARY
(p2,q2)</td></tr>
<tr><td>FIXED
BINARY
(p1,q1)</td><td>FLOAT
BINARY
(p2,q2)</td></tr>
</table>

表 13. 1 つまたは複数の FLOAT オペランドの算術演算の結果 (続き)

第 1 オペランド (p1,q1)	第 2 オペランド (p2,q2)	加算、減算、乗算または除算の結果の属性	加算または減算精度	乗算精度	除算精度	累乗演算の結果の属性
FIXED DECIMAL (p1,q1)	FLOAT BINARY (p2)	FLOAT BINARY (p)	p = MAX(CEIL(p1*3.32),p2)			FLOAT BINARY (p) (特殊な場合以外は、A または C が適用される) p = MAX(CEIL (p1*3.32),p2)
FLOAT DECIMAL (p1)	FLOAT BINARY (p1,q2)					
FLOAT DECIMAL (p1)	FLOAT BINARY (p2)					
FIXED BINARY (p1,q1)	FLOAT DECIMAL (p2)	FLOAT BINARY (p)	p = MAX(p1,CEIL(p2*3.32))			FLOAT BINARY (p) (特殊な場合以外は、B または C が適用される) p = MAX(p1,CEIL (p2*3.32))
FLOAT BINARY (p1)	FIXED DECIMAL (p2,q2)					
FLOAT BINARY (p1)	FLOAT DECIMAL (p2)					

注:

1. 累乗演算の特殊事例は、 67 ページの表 18 で説明しています。
2. CEIL(N*3.32) 値の表については、 81 ページの表 22 を参照してください。

表 14. RULES(ANS) における、2 つのスケールのない FIXED オペランド間の算術演算の結果

第 1 オペランド (p1,q1)	第 2 オペランド (p2,q2)	加算、減算、乗算または除算の結果の属性	加算または減算精度	乗算精度	除算精度	累乗演算の結果の属性
FIXED DECIMAL (p1,0)	FIXED DECIMAL (p2,0)	FIXED DECIMAL (p,q)	p = 1 +MAX(p1,p2) q = 0	p = 1 +p1+p2 q = 0	p = N q = N-p1	FLOAT DECIMAL (p) (特殊な場合以外は、A が適用される) p = MAX(p1,p2)
FIXED BINARY (p1,0)	FIXED BINARY (p2,0)	FIXED BINARY (p,0)	p = 1 +MAX(p1-q1, p2-q2) +q q = 0	p = 1+p1 +p2 q = 0	p = M q = 0	FLOAT BINARY (p) (特殊な場合以外は、B が適用される) p = MAX(p1,p2)
FIXED DECIMAL (p1,0)	FIXED BINARY (p2,0)	FIXED BINARY (p,0)	p = 1 +MAX(r,p2) q = 0	p = 1 +r+p2 q = 0	p = M q = 0	FLOAT BINARY (p) (特殊な場合以外は、A が適用される) p = MAX(CEIL (p1*3.32),p2)

表 14. RULES(ANS) における、2 つのスケールのない FIXED オペランド間の算術演算の結果 (続き)

第 1 オペランド (p1,q1)	第 2 オペランド (p2,q2)	加算、減算、乗算または除算の結果の属性	加算または減算精度	乗算精度	除算精度	累乗演算の結果の属性
FIXED BINARY (p1,0)	FIXED DECIMAL (p2,0)	FIXED BINARY (p,0)	p = 1 +MAX(p1,t) q = 0	p = 1 +p1+t q = 0	p = M q = 0	FLOAT BINARY (p) (特殊な場合以外は、B が適用される) p = MAX(CEIL (p1*3.32),p2)

M は FIXED BINARY の最大精度です。	t = 1 + CEIL(p2*3.32)
N は FIXED DECIMAL の最大精度です。	u = CEIL(ABS(q2*3.32)) * SIGN(q2)
r = 1 + CEIL(p1*3.32)	v = CEIL(p2/3.32)
s = CEIL(ABS(q1*3.32)) * SIGN(q1)	w = CEIL(p1/3.32)

注:

スケール因数は、-128 から +127 の範囲内になければなりません。

1. 累乗演算の特殊事例は、 67 ページの表 18 で説明しています。
2. CEIL(N*3.32) 値の表については、 81 ページの表 22 を参照してください。
3. RULES(ANS) においては、スケールのない FIXED オペランドを持つ除算は、両方のオペランドが FIXED DECIMAL の場合にのみスケールのある結果を生じます。

表 15. RULES(ANS) における、2 つのスケールされた FIXED オペランド間の算術演算の結果

第 1 オペランド (p1,q1)	第 2 オペランド (p2,q2)	加算、減算、乗算または除算の結果の属性	加算または減算精度	乗算精度	除算精度	累乗演算の結果の属性
FIXED DECIMAL (p1,q1)	FIXED DECIMAL (p2,q2)	FIXED DECIMAL (p,q)	p = 1 + MAX(p1-q1, p2-q2) +q q = MAX(q1,q2)	p = 1 +p1+p2 q = q1+q2	p = N q = N-p1+q1-q2	FLOAT DECIMAL (p) (特殊な場合以外は、A が適用される) p = MAX(p1,p2)
FIXED DECIMAL (p1,q1)	FIXED BINARY (p2,0)	FIXED DECIMAL (p,q)	p = 1 +MAX(p1-q1,v) +q q = q1	p = 1 +p2+v q = q1	p = N q = N-q1	FLOAT BINARY (p) (特殊な場合以外は、A が適用される) p = MAX(CEIL (p1*3.32),p2)
FIXED BINARY (p1,0)	FIXED DECIMAL (p2,q2)	FIXED DECIMAL (p,q)	p = 1 +MAX(p2-q2,w) +q q = q2	p = 1 +p2+w q = q1	p = N q = N-q2	FLOAT BINARY (p) (特殊な場合以外は、B が適用される) p = MAX(CEIL (p1*3.32),p2)

M は FIXED BINARY の最大精度です。	t = 1 + CEIL(p2*3.32)
N は FIXED DECIMAL の最大精度です。	u = CEIL(ABS(q2*3.32)) * SIGN(q2)
r = 1 + CEIL(p1*3.32)	v = CEIL(p2/3.32)
s = CEIL(ABS(q1*3.32)) * SIGN(q1)	w = CEIL(p1/3.32)

注:

スケール因数は、-128 から +127 の範囲内になければなりません。

1. 累乗演算の特殊事例は、 67 ページの表 18 で説明しています。
2. CEIL(N*3.32) 値の表については、 81 ページの表 22 を参照してください。
3. RULES(ANS) のもとでは、スケールされた FIXED BINARY は使用できません。

表 16. RULES(IBM) における、2 つの FIXED オペランド間の算術演算の結果

第 1 オペランド (p1,q1)	**第 2 オペランド (p2,q2)**	**加算、減算、乗算または除算の結果の属性**	**加算または減算精度**	**乗算精度**	**除算精度**	**累乗演算の結果の属性**
FIXED DECIMAL (p1,q1)	FIXED DECIMAL (p2,q2)	FIXED DECIMAL (p,q)	p = 1 +MAX(p1-q1, p2-q2) +q q = MAX(q1,q2)	p = 1 +p1+p2 q = q1+q2	p = N q = N-p1+q1-q2	FLOAT DECIMAL (p) (特殊な場合以外は、A が適用される) p = MAX(p1,p2)
FIXED BINARY (p1,q1)	FIXED BINARY (p2,q2)	FIXED BINARY (p,q)	p = 1 +MAX(p1-q1, p2-q2) +q q = MAX(q1,q2)	p = 1 +p1+p2 q = q1+q2	p = M q = M-p1 +q1-q2	FLOAT BINARY (p) (特殊な場合以外は、B が適用される) p = MAX(p1,p2)
FIXED DECIMAL (p1,q1)	FIXED BINARY (p2,q2)	FIXED BINARY (p,q)	p = 1 +MAX(r-s, p2-q2)+q q = MAX(s,q2)	p = 1+r +p2 q = s+q2	p = M q = M-r +s-q2	FLOAT BINARY (p) (特殊な場合以外は、A が適用される) p =MAX(CEIL((p1*3.32),p2)
FIXED BINARY (p1,q1)	FIXED DECIMAL (p2,q2)	FIXED BINARY (p,q)	p = 1 +MAX(p1-q1,t-u) +q q = MAX(s,q1,u)	p = 1 +p1+t q = q1+u	p = M q = M-p1 +q1-u	FLOAT BINARY (p) (特殊な場合以外は、B が適用される) p = MAX(p1,CEIL(p2*3.32))

M は FIXED BINARY の最大精度です。
N は FIXED DECIMAL の最大精度です。
r = 1 + CEIL(p1*3.32)
s = CEIL(ABS(q1*3.32)) * SIGN(q1)
t = 1 + CEIL(p2*3.32)
u = CEIL(ABS(q2*3.32)) * SIGN(q2)
v = CEIL(p2/3.32)
w = CEIL(p1/3.32)

注:

スケール因数は、-128 から +127 の範囲内になければなりません。

1. 累乗演算の特殊事例は、 67 ページの表 18 で説明しています。
2. CEIL(N*3.32) 値の表については、 81 ページの表 22 を参照してください。

次の式を例に、考えてみましょう。

```
A * B + C
```

まず、演算 A * B が行われ、中間結果が生じます。次に、(中間結果) + C という演算が行われて、式の値が求められます。

PL/I は、属性を別の変数に指定するのと同じ方法で中間結果属性を指定します。結果の属性は、2 つのオペランドの属性 (または接頭部演算の場合には単一オペランド)、および関係する演算子から派生します。結果の属性を求める方法については、58 ページの『ターゲット』に説明されています。

組み込み関数 ADD、SUBTRACT、MULTIPLY、および DIVIDE を使用すると、加算、減算、乗算、および除算演算に関してインプリメンテーションで定められている精度の規則を無効にすることができます。

FIXED 除算: 固定小数点除算を行うと、オーバーフローや切り捨てが起こることがあります。たとえば次の場合、

```
25+1/3
```

固定小数点除算の結果が、インプリメンテーションで定義されている最大精度の値になるため、式の計算結果は未定義となり、FIXEDOVERFLOW 条件が起こります。

ただし次の場合、

```
25+01/3
```

定数には作成される精度があるので、式は 25.3333333333333 (最大精度は 15) という結果になります。 2 つの計算の結果は、表 17 に示されている結果になります。

表 17. 固定小数点除算と定数式の比較

項目	精度	結果
1	(1,0)	1
3	(1,0)	3
1/3	(15,14)	0.33333333333333
25	(2,0)	25
25+1/3	(15,14)	未定義 (undefined) (左方で切り捨て。 最大精度が 15 の場合は、 FIXEDOVERFLOW が生じる)
01	(2,0)	01
3	(1,0)	3
01/3	(15,13)	00.3333333333333
25	(2,0)	25
25+01/3	(15,13)	25.3333333333333

PRECISION 組み込み関数を使用することもできます。 以下に例を示します。

```
25+prec(1/3,15,13)
```

注: 正確な精度が必要な状況では、名前のついた定数をお勧めします。

累乗演算の使用

以下の表は、PL/I で累乗演算が扱われる方法を説明しています。

表 18. 累乗演算の特殊事例

事例	第 1 オペランド	第 2 オペランド	結果の属性
A	FIXED DECIMAL (p1,q1)	値が n の整数	FIXED DECIMAL (p,q) (p <= N) ただし、p = (p1 + 1)*n-1 かつ q = q1*n
B	FIXED BINARY (p1,q1)	値が n の整数	FIXED BINARY (p,q) (p <= M) ただし、p = (p1 + 1)*n-1 かつ q = q1*n
C	FLOAT (p1)	FIXED (p2,0)	FLOAT (p1) 基数は第 1 オペランドの基数と同じ

実モードまたは複素数モードでの xy の特殊事例:**

実モード:

複素数モード:

x=0 かつ y>0

結果は 0。x=0、かつ y の実数部分 >0 かつ y の虚数部分 =0 ならば、結果は 0。

x=0 かつ y<=0

ERROR 条件が起こる。 x=0 かつ y の実数部分 <=0 または y の虚数部分 ¬=0 ならば、 ERROR 条件が起こる。

x<0 かつ y が FIXED (p,0) でないとき

ERROR 条件が起こる。 x¬=0 かつ y の実数部分および虚数部分 =0 ならば、結果は 1。

ビット演算子

ビット演算を指定するには、下記のいずれかの論理演算子でオペランドを結び付けます。

```
¬    &    |
```

NOT/排他的 OR 記号 (¬) は、接頭部または挿入演算子として使用することができます。 *AND* (&) 記号および *OR* (|) 記号は、挿入演算子としてのみ使用することができます (これらの演算子は、ブール代数の場合と同じです)。

ビット演算のオペランドは、演算の実行前に、必要に応じてビット・ストリングに変換されます。挿入演算子の演算の両オペランドの長さが異なっている場合は、短い方のストリングの右側に '0'B が埋め込まれます。

ビット演算の演算結果は、両オペランドと同じ長さのビット・ストリングです。

ビット演算は、1 ビットずつ行われます。 68 ページの表 19 は、各演算子のビット位置の結果を説明しています。 68 ページの表 20 は、ビット演算子の例を示しています。

表 19. ビット演算子

A	B	A	B	A&B	A|B	A ¬B
1	1	0	0	1	1	0
1	0	0	1	0	1	1
0	1	1	0	0	1	1
0	0	1	1	0	0	0

表 20. ビット演算の例

オペランドと値	演算	結果
A = '010111'B B = '111111'B C = '110'B D = 5	¬ A	'101000'B
	¬ C	'001'B
	C & B	'110000'B
	A | B	'111111'B
	A ¬ B	''101000''B
	A ¬ C	'100111'B
	C | B	'111111'B
	A | (¬C)	'011111'B
	¬((¬C)|(¬B))	'110111'B
	SUBSTR(A,1,1)|(D=5)	'1'B

BOOL 組み込み関数

演算子 ¬、&、および ¦ を使用する *NOT*、*排他 OR*、*AND*、および *OR* 演算に加えて、 423 ページの『BOOL』で説明している BOOL 組み込み関数を使用してブール演算を行うことができます。

比較演算

比較演算を指定するには、下記のいずれか 1 つの挿入演算子でオペランドを結び付けます。

```
<        ¬<       <=       =        ¬=       >=       >        ¬>
```

比較演算の結果は、常に長さ 1 のビット・ストリングです。オペランドの関係が真であれば、値は '1'B になり、偽であれば '0'B になります。

比較は以下のように定義されます。

代数 コード化算術フォーマットの符号付き算術値を比較します。 2 つのオペランドが基数、スケール、精度、またはモードで異なっている場合は、算術演算変換に似た方法で変換されます。数字データは、比較される前にコード化算術フォーマットに変換されます。演算子 = と ¬= だけが、複素数のオペランドを比較する場合に有効です。

文字 左から右へ文字単位で文字 (バイト単位で 2 進数値) を比較します。

ビット 左から右へ 1 ビットずつ、2 進数を比較します。

グラフィック 左から右へ DBCS 文字を 1 文字 (2 バイト) ずつ比較します。比較は、DBCS 文字の 2 進数値に基づいて行われます。

ワイド文字 左から右へ、2 バイトの対の 2 進数値に基づいてワイド文字を 1 文字ずつ比較します。

序数データ 関係演算子を使用して、同じタイプの序数を比較します。

ポインターおよびオフセット・データ
関連する任意の演算子が入っているポインター値とオフセット値を比較します。ただし、変換を行うことができるのは、オフセットからポインターへの場合です。

プログラム制御データ
内部コード化フォーマットのオペランドを比較します。比較演算子の = と ¬= のみを使用できます。区域変数は比較できません。タイプ変換は起こりません。プログラム制御データの比較では、オペランド間のタイプの相違はすべてエラーになります。

以下のオペランドについては、比較は等しいという結果になります。

入り口 比較演算では、非アクティブ・ブロックのエントリー・ポイントを値として持つ入り口変数を指定しても、エラーにはなりません。同じ PROCEDURE または ENTRY ステートメントの入り口名は、等しいものを比較することはできません。

フォーマット 同じ FORMAT ステートメントに関するフォーマット・ラベルの比較が等しい場合。

ファイル 2 つのオペランドがファイル値を表しており、ファイル値の各部分ごとに等しい場合。

ラベル 同じステートメントに付いている 2 つのラベルを比較すると、等しいという結果になります。比較演算では、これ以上活動しないブロック内のラベル定数を値として持つラベル変数を指定しても、エラーにはなりません。

複合ステートメントのラベルを、その複合ステートメントの本体に含まれているラベルと比較した場合、等しいという結果にはなりません。

計算データの比較で両オペランドのデータ・タイプが別々の比較タイプである場合には、比較タイプに関して優先度の低い方のオペランドが、もう一方のオペランドの比較タイプに合うように変換されます。比較タイプの優先順位は、(1) 代数比較 (最も高い)、(2) ワイド文字比較、 (3) グラフィック比較、(4) 文字比較、(5) ビット比較の順です。たとえば、ビット・ストリングと固定小数点値を比較する場合には、10 進数値との代数比較のためにビット・ストリングが 2 進固定小数点フォーマットに変換されます。さらに、10 進数値も 2 進固定小数点フォーマットに変換されます。

長さの異なる 2 つのストリングの比較では、短い方のストリングの右側に埋め込まれます。この埋め込みは、以下のものから構成されます。

- 文字比較のときのブランク
- ビット比較のときの '0'B
- グラフィック比較のときのグラフィック (DBCS) ブランク
- ワイド文字比較のときのワイド文字ブランク ('0020'wx)

以下に、IF ステートメントでの比較演算の例を示します。

```
if A = B
    then action-if-true;
    else action-if-false;
```

A = B の式の演算結果は、これが真であれば '1'B となり、偽であれば '0'B となります。

以下の代入ステートメントでは、

```
X = A <= B;
```

A が B より小さければ、 '1'B という値が X に割り当てられます。それ以外の場合は、'0'B という値が割り当てられます。

以下の代入ステートメントでは、

```
X = A = B;
```

最初の等号は代入記号です。2 番目の等号は比較演算子です。 A が B と等しければ、 '1'B という値が X に割り当てられます。等しくない場合は、'0'B という値が割り当てられます。

算術式での比較演算の例を示します。

```
(X<0)*A + (0<=X & X<=100)*B + (100<X)*C
```

この式の値は X の値に従って、A、B、または C のいずれかになります。

連結演算

連結演算を指定するには、連結挿入文字でオペランド同士を結び付けます。

||

この連結記号は、左側のオペランドの最後の文字、ビット、グラフィック、またはワイド文字のすぐあとに (間に何もはさまずに)、右側のオペランドの最初の文字、ビット、グラフィック、またはワイド文字が続く形で、2 つのオペランドを結合することを表します。

連結演算はストリング (文字、ビット、グラフィック、またはワイド文字) にしか行えないので、連結演算子を使用すると、ストリング・タイプへの変換が行われることがあります。結果は、RULES コンパイラー・オプションの設定値に応じて異なります。

RULES(IBM) のもとでの結果

RULES(IBM) を指定する場合、連結演算子は以下のように振る舞います。

- いずれかのオペランドがワイド文字である場合は、結果はワイド文字になります。

- そうでなく、いずれかのオペランドがグラフィックである場合は、結果はグラフィックになります。
- そうでなく、いずれかのオペランドがビットまたは 2 進数の場合は、結果はビットになります。
- それ以外の場合は、文字になります。

以下に例を示します。

```
dcl B bin(4)  initial(4),
    C bit(1)  initial('1'b);
put skip list (B || C);

/*  Produces '01001'  not 'bbb41'  */
```

RULES(ANS) のもとでの結果: RULES(ANS) を指定する場合、連結演算子は以下のように振る舞います。

- いずれかのオペランドがワイド文字である場合は、結果はワイド文字になります。
- そうでなく、いずれかのオペランドがグラフィックである場合は、結果はグラフィックになります。
- そうでなく、両方のオペランドがビットである場合には、結果はビットになります。
- それ以外の場合は、文字になります。

以下に例を示します。

```
dcl B bin(4)  initial(4),
    C bit(1)  initial('1'b);
put skip list (B || C);

/*  Produces 'bbb41', not '01001'  */
```

連結演算の結果は、2 つのオペランドの長さの合計に等しい長さを持つストリングであり、そのタイプ (文字、ビット、グラフィック、またはワイド文字) は、2 つのオペランドのタイプと同じです。

連結のためにオペランドの変換が必要であった場合は、オペランド変換後のストリングの長さによって結果の長さが決まります。

オペランドと値	演算	結果
A = '010111'B B = '101'B C = 'xy,Z' D = 'aa/BB'	A || B	'010111_101'B
	A || A || B	'010111_010111_101'B
	C || D	'xy,Zaa/BB'
	D || C	'aa/BBxy,Z'
	B || D	'101aa/BB'

上記の最後の例では、ビット・ストリング '101'B が文字ストリング '101' に変換されてから、連結が行われます。結果は、文字ストリングになります。

各種演算の組み合わせ

1 つの演算式の中で、各種演算を組み合わせることができます。どのような組み合わせでもかまいません。

以下に例を示します。

```
declare Result bit(3),
  A fixed decimal(1),
  B fixed binary (3),
  C character(2), D bit(4);
Result = A + B < C & D;
```

この式の中の計算は、それぞれの計算の規則に従って実行されますが、その前に、下記のように必要とするデータ変換が行われます。

- A の 10 進数値が 2 進数値に変換されます。
- A と B を加算する 2 進数加算が行われます。
- この 2 進数の結果が、C の変換された 2 進数値と比較されます。
- ビット・ストリングで表された比較結果が、ビット変数 D の長さまで広げられてから、& 演算が行われます。
- & 演算の結果 (長さ 4 のビット・ストリング) は、変換されずに、右側で切り捨てられて、Result に割り当てられます。

この例の式は、左から右へ順に計算を行うように記述されています。ただし、計算の順序は、式の中に書かれている演算子の優先順位によって決まります。

演算子の優先順位

式を計算するときの演算子の優先順位は、表 21 に示すとおりです。

表 21. 演算の優先順位と変換の手引き

優先度	演算子	演算のタイプ	注釈
1	**	算術	結果はコード化算術フォーマットです。
	接頭演算子 +、-	算術	オペランドがコード化算術フォーマットであれば、変換は不要です。
			オペランドが CHARACTER ストリングまたは数字 (PICTURE) 表記の固定小数点 10 進数であれば、FIXED DECIMAL に変換されます。
			オペランドが数字 (PICTURE) 表記の 10 進浮動小数点数であれば、FLOAT DECIMAL に変換されます。
			オペランドが BIT ストリングであれば、FIXED BINARY に変換されます。
	接頭演算子 ¬	ビット・ストリング	BIT 以外のデータはすべて BIT に変換されます。
2	*, /	算術	結果はコード化算術フォーマットです。
3	挿入演算子 +、-	算術	結果はコード化算術フォーマットです。
4	‖	連結	71 ページの『RULES(ANS) のもとでの結果』および 70 ページの『RULES(IBM) のもとでの結果』を参照してください。
5	<, ¬<, <=, =, ¬=, >=, >, ¬>	比較	結果は常に '1'B または '0'B です。
6	&	ビット・ストリング	BIT 以外のデータはすべて BIT に変換されます。
7	|	ビット・ストリング	BIT 以外のデータはすべて BIT に変換されます。
	挿入演算子 ¬	ビット・ストリング	BIT 以外のデータはすべて BIT に変換されます。

注:

1. 優先順位が高いものから順に演算子を並べてあります。グループ 1 の優先順位が一番高く、グループ 7 が一番低くなっています。同じ優先順位グループに属する演算子の優先順位はすべて同じです。たとえば、累乗演算子 ** の優先順位は、接頭演算子 + と -、および *NOT* 演算子 ¬ と同じです。
2. 優先順位グループ 1 内の演算子が式の中で複数使われているときは、その式の中での優先順位は右から左への順になります。すなわち最右端の指数または接頭演算子が最高の優先順位を持ち、その隣が次の優先順位を持ち、というようになります。その他の優先順位グループの場合は、同じ優先順位グループ内の演算子が式の中で複数使われているときは、その式の中での計算順序つまり優先順位は、左から右への順になります。

次の式の計算順序は、

```
A + B < C & D
```

下記のように式の各エレメントを括弧で囲んだ場合と同じです。

```
(((A + B) < C) & D)
```

式の計算順序 (したがって、その結果も) は、括弧を使用して変えることができます。括弧で囲まれている式が最初に計算されて 1 つの値が求められ、次にその前後にある演算子に関係づけられています。

たとえば、上記の式を次のように変えることができます。

```
(A + B) < (C & D)
```

A の値が固定小数点 2 進数に変換されてから、加算が行われ、固定小数点 2 進数の結果 (result_1) が生成されます。 C の値がビット・ストリングに変換され、 (このように変換できる場合) *AND* 演算が行われます。この時点で、上の式は次のよう縮小されます。

```
Result_1 < Result_2
```

result_2 が 2 進数に変換されてから、代数比較が行われ、式全体の演算結果として長さ 1 のビット・ストリングが生成されます。

演算子の優先順位は、オペランド間 (またはサブオペランド間) の関係で決められます。次の例を考えてみてください。

```
A + (B < C) & (D || E ** F)
```

この場合、PL/I では、累乗演算を連結より先に実行するということだけが定められています。 (D||E ** F) の計算と他方のオペランド (A + (B < C)) の計算との順序関係は定められていません。

どのような演算式 (接頭演算子の式を除く) も、最終的には、単一の挿入演算子演算に変形されるものでなければなりません。その最終的な演算のオペランドと演算子によって、式全体の計算結果の属性が決まります。次の例では、& 演算子は、最終段階の挿入演算子演算の演算子です。

```
A + B < C & D
```

計算結果は長さ 4 のビット・ストリングになります。

次の例では、括弧で囲まれているため、最終段階の挿入演算子演算の演算子が比較演算子になります。

```
(A + B) < (C & D)
```

計算結果は長さ 1 のビット・ストリングになります。

配列式

配列式には、次のような利用方法があります。

- 割り当てまたは複数の割り当てにおけるソース
- ALL、ANY、POLY、PROD または SUM といった組み込み関数への引数
- (関連パラメーターが、長さが不明なストリングでない場合は) ユーザー・プロシージャーおよびユーザー関数への引数
- PUT LIST ステートメント および PUT EDIT ステートメントのデータ・リストの項目

配列式を計算すると、結果は配列になります。配列を対象とする演算はすべて、行を主体としてエレメントごとに行われます。したがって、1 つの配列式で参照するすべての配列は、次元の数が同じでなければならず、しかも各次元ごとに境界が同じでなければなりません。

配列式では、演算子 (接頭演算子と挿入演算子の両方とも)、要素変数、および定数も使うことができます。演算の組み合わせの規則やオペランドのデータ変換に関する規則は、エレメント演算の場合と同じです。

接頭演算子と配列

配列に接頭演算子演算を行うと、同じ境界を持つ配列になります。この配列の各エレメントは、もとの配列の各エレメントに行われた演算の結果です。 以下に例を示します。

```
If A is the array      5   3  -9
                       1   2   7
                       6   3  -4

then -A is the array  -5  -3   9
                      -1  -2  -7
                      -6  -3   4
```

挿入演算子と配列

挿入演算子演算の一方のオペランドが配列変数のときは、他方のオペランドはエレメント、別の配列でもかまいません。

配列とエレメントの演算

エレメントと配列が挿入演算子で結ばれている式の結果は、もとの配列と同じ境界を持つ配列になります。この配列の各エレメントは、元の配列内の対応するエレメントと単一エレメントとの間で行われた演算の結果になります。 以下に例を示します。

```
If A is the array         5  10   8
                         12  11   3

then A*3 is the array    15  30  24
```

```
                          36  33   9

and 9 > A is the array of   1   0   1
bit strings of length 1     0   0   1
```

配列とエレメントの間で演算を行う場合、そのエレメントは、その配列内のエレメントであってもかまいません。次の代入ステートメントを考えてください。

```
A = A * A(1,2);
```

再度 A に上記の値を使えば、新しく A に割り当てられる値は次のようになります。

```
   50    100   800
 1200   1100   300
```

つまり、A(1,2) の値が再び取り出されます。

配列同士の演算

挿入演算子演算の 2 つのオペランドが共に配列である場合は、この 2 つの配列の次元の数は同じでなければならず、しかも対応する次元ごとに下限と上限が同じでなければなりません。演算結果は、元の 2 つの配列と同じ限界を持つ配列になります。演算は、元の 2 つの配列の対応するエレメント間で行われます。 以下に例を示します。

```
If A is the array          2   4   3
                           6   1   7
                           4   8   2

and if B is the array      1   5   7
                           8   3   4
                           6   3   1

then A+B is the array      3   9  10
                          14   4  11
                          10  11   3

and A*B is the array       2  20  21
                          48   3  28
                          24  24   2

and A>B is the array of    1   0   0
bit strings of length 1    0   0   1
                           0   1   1
```

構造式

構造式は、構造体参照とは異なり、関連するパラメーターに定数範囲がある限りは、割り当てを行うときと、プロシージャーまたは関数への引数としてのみ使用できます。

構造式の中で使われる構造変数はすべて同じように構造化されていなければなりません。すなわち、以下の条件が満たされていなければなりません。

- 構造体は、小構造化がすべて同じで、中に含まれるエレメントと配列の数が同じでなければなりません。
- 構造体内 (および小構造があれば、その内部で) のエレメントや配列の位置指定は、同じでなければなりません。
- 対応する位置にある配列は、同じ境界を持たなければなりません。

制限付き式

PL/I に (おそらく、符号付きの) 定数が必要なときには、*制限付き式* を使用することができます。制限付き式の値はコンパイル時に計算され、定数として使用されます。たとえば、式を使用して、以下に必要な定数を定義することができます。

- 静的、パラメーターおよび基本宣言のエクステント
- 入り口記述のエクステント
- 静的初期設定で使用される値および反復因数

制限付き式は、通常の式と同等ですが、各オペランドが以下のものでなければなりません。

- 定数または名前のついた定数。名前付き定数は、使用する前に宣言しなければなりません。
- 組み込み関数は、制限付きの式 (単数または複数) に適用されます。この組み込み関数は、次のカテゴリーから用いられます。
  - ストリング処理
  - 算術 (RANDOM を除く)
  - 数学
  - 浮動小数点照会
  - 浮動小数点操作
  - 整数操作
  - 精度処理
  - 配列処理機能の DIMENSION、LBOUND、および HBOUND
  - ストレージ制御関数 BINARYVALUE、LENGTH、NULL、OFFSETVALUE、POINTERVALUE、SIZE、STORAGE、および SYSNULL
- タイプ付き関数 BIND、CAST、FIRST、LAST、RESPEC、および SIZE

例

```
dcl Max_names fixed bin value (1000),
    Name_size fixed bin value (30),
    Addr_size fixed bin value (20),
    Addr_lines fixed bin value (4);
dcl 1 Name_addr(Max_names),
      2 Name char(Name_size),
      2 * union,
        3 Address char(Addr_lines*Addr_size), /* address    */
        3 addr(Addr_lines) char(Addr_size),
      2 * char(0);
dcl One_Name_addr char(size(Name_addr(1)));   /* 1 name/addr*/
dcl Two_Name_addr char(length(One_Name_addr)
                       *2);                /* 2 name/addrs  */
dcl Name_or_addr char(max(Name_size,Addr_size)) based;

dcl Ar(10) pointer;
dcl Ex     entry( dim(lbound(Ar):hbound(Ar)) pointer);
dcl Identical_to_Ar( lbound(Ar):hbound(Ar) ) pointer;
```

この例で、名前付き定数のいずれかの値を変更すると、従属宣言はすべて自動的に再計算されます。

第 5 章 データ変換

計算データ変換の組み込み関数 78
ストリングの長さの変換 79
算術値の精度の変換 80
モードの変換 80
その他のデータ属性の変換 80
ソースからターゲットへの変換規則 82
例 90
小数部分がある DECIMAL FIXED から小数部分がある BINARY FIXED への変換 90
算術データからビット・ストリングへの変換 . . 90
算術データから文字への変換 91
変換エラー 91

この章では、計算データのデータ変換について説明します。 PL/I は、特定の属性セットを持つデータ項目が、別の属性セットを持つデータ項目に割り当てられると、データ変換を行います。この章では、ソース は、変換されるデータ項目を意味し、ターゲット は、ソースを変換した属性を意味します。この章では、データの変換について以下の事項を説明します。

- 組み込み関数
- ストリングの長さ
- 算術値の精度
- モード
- ソースからターゲットへの変換規則

章の最後にデータの変換の例が記載されています。

ロケーター・データのデータ変換は、 247 ページの『ロケーターの変換』で説明されています。

計算データ項目の値が変換されると、その内部表記、精度 (算術値の場合)、モード (算術値の場合)、または長さ (ストリング値の場合) が変わることがあります。下の表は、どのような場合にほかの属性に変換が行われるかをまとめたものです。

事例	ターゲットの属性
割り当て	代入記号の左側にある変数の属性
式のオペランド	式を計算するときの規則によって決まる
ストリームの入力 (GET ステートメント)	受け入れフィールドの属性
ストリームの出力 (PUT ステートメント)	ストリームが編集ディレクティブであれば、フォーマット・リストによって決まり、それ以外の場合は文字ストリング
PROCEDURE または ENTRY への引数	対応するパラメーターの属性
組み込み関数または疑似変数への引数	関数または疑似変数によって決まる
INITIAL 属性	初期設定される変数のほかの属性
RETURN ステートメントの式	PROCEDURE ステートメントで指定されている属性
DO ステートメントのオプション BY、TO、または REPEAT	制御変数の属性

次の場合は、文字値に変換されることがあります。

ステートメント	オプション
DISPLAY	
レコード入出力	KEYFROMKEY

ステートメント	オプション
OPEN	TITLE

次の場合は、BINARY 値に変換されることがあります。

ステートメント	オプション/属性/参照
DECLARE、ALLOCATE、DEFAULT	長さ、サイズ、次元、境界、反復因数
DELAY	ミリ秒
FORMAT (および GET や PUT 内のフォーマット項目)	反復因数、d、s、p
OPEN	LINESIZE、PAGESIZE
I/O	SKIP、LINE、IGNORE
ほとんどのステートメント	添え字

ソース・データ項目とターゲット・データ項目のすべての属性 (ストリングの長さは除く) が、コンパイル時に指定されていなければなりません。変換は以下の条件の 1 つを起こすことがあります。 CONVERSION、OVERFLOW、SIZE、または STRINGSIZE。 (367 ページの『第 17 章 条件』を参照してください。)

定数は実行時だけでなくコンパイル時に変換されることもあります。どの場合も、行われる変換はここで述べられているとおりです。

1 つの演算で変換が複数回必要になることがあります。ただし、必ずしも複数回行われるとは限りません。変換規則を理解するには、これらの演算を別々に考えることをお勧めします。たとえば、次のようになります。

```
dcl A fixed dec(3,2) init(1.23);
dcl B fixed bin(15,5);
B = A;
```

この例では、1.23 という 10 進数表記は、まず BINARY (11,7) に変換されて 1.0011101B となります。次に、精度の変換が行われて、BINARY(15,5) の 1.00111B という値になります。

その他の変換例は、この章の終わりに示してあります。

計算データ変換の組み込み関数

式の計算の途中、入出力 GET や PUT 演算、代入演算、および引数とパラメーター間で変換を行うことができます。さらに、以下の組み込み関数を使用すれば、変換を開始させることもできます。

BINARY	FIXED	REAL
BIT	FLOAT	SIGNED
CHAR	GRAPHIC	UNSIGNED
COMPLEX	IMAG	WIDECHAR
DECIMAL	PRECISION	

個々の関数については、 395 ページの『第 19 章 組み込み関数、疑似変数、およびサブルーチン』で説明します。

それぞれの関数は、必要な変換を行って、その関数名で指定される属性を持つ値を戻します。

組み込み関数 COMPLEX、GRAPHIC、および IMAG によって行われる変換は例外ですが、必要な属性を持つ PL/I 変数に割り当てを行っても、上記の組み込み関数で行われる変換と同じ結果を得ることができます。しかし、組み込み関数を使用するほうが、変換を実行するために変数を作成するよりも容易であることは明らかです。

ストリングの長さの変換

ソース・ストリングは、ターゲット・ストリングの左から右へ割り当てられます。ソース・ストリングがターゲット・ストリングよりも長いときは、右側の入りきらなかった文字、ビット、グラフィック、またはワイド文字は無視され、STRINGSIZE 条件が起こります。ターゲット・ストリングが固定長で、ソース・ストリングよりも長いときは、ターゲット・ストリングの右側が埋め込まれます。 STRINGSIZE が割り込み禁止になっており、ソース・ストリングやターゲット・ストリング (またはこれらの両方) の長さが実行時に判別され、しかもターゲット・ストリングが短すぎてソース・ストリングが入りきらない場合には、予測できない結果が発生することがあります。

注: SUBSTR をパラメーターとして変数と一緒に使用し、その変数がターゲットに入っていないストリングを指定すると、 STRINGRANGE 条件が使用可能になっていない場合、予測できない結果になることがあります。

ストリングにはブランクが埋め込まれ、ビット・ストリングには '0'B、漢字ストリングには DBCS ブランク、ワイド文字ストリングにはワイド文字ブランクがそれぞれ埋め込まれます。

```
declare Subject char(10);
Subject = 'Transformations';
```

'Transformations' は、15 文字です。したがって、 PL/I がそのストリングを Subject に割り当てると、ストリングの右端から 5 文字を切り捨てます。これは次の割り当てを実行するのと同等です。

```
Subject = 'Transforma';
```

下に示す最初の 2 つのステートメントは、同じ値を Subject に割り当てます。また、最後の 2 つのステートメントは、同じ値を Code に割り当てます。

```
Subject = 'Physics';
Subject = 'Physics   ';
declare Code bit(10);
Code = '110011'B;
Code = '1100110000'B;
```

次の 2 つのステートメントは、Subject に同等の値を割り当てません。

```
Subject = '110011'B;
Subject = '1100110000'B;
```

最初のステートメントが実行されると、右側のビット定数がまず文字ストリングに変換されてから、ビットのゼロではなくブランク文字が右側に埋め込まれます。このステートメントは次のステートメントと同等です。

```
Subject = '110011bbbb';
```

この 2 つのステートメントの 2 番目のステートメントでは、ビットから文字タイプへの変換が必要です。これは次のステートメントと同等です。

```
Subject = '1100110000';
```

ストリング値が、VARYING 属性を持つストリング変数に割り当てられるときは、ブランク文字またはゼロのビットによる埋め込みは行われません。代わりに、ターゲット・ストリング変数の長さが、割り当てられたストリングの長さにセットされます。ただし、割り当てられたストリングが、可変長ストリング変数で宣言されている最大の長さより長いときは、切り捨てが行われます。

算術値の精度の変換

算術値のデータ属性がターゲットのデータ属性 (精度は除く) と同じであれば、精度の変換が必要になります。

固定小数点データ項目の場合は、10 進数も 2 進数も小数点で位置合わせされて精度が変換されます。したがって、左側または右側で埋め込みや切り捨てが行われることがあります。左側 (最高位桁) でゼロ以外の 2 進数または 10 進数が失われると、SIZE 条件が起こります。

浮動小数点データ項目の場合は、右側 (最低位桁) で、切り捨てまたはゼロによる埋め込みが行われることがあります。

モードの変換

複合値が実数値に変換される場合は、虚数部分が無視されます。実数値が複合値に変換される場合は、虚数部分がゼロになります。

その他のデータ属性の変換

下記のデータ属性を持つデータ項目をソースからターゲットへ変換する場合の規則について、以下に説明します。

- コード化算術フォーマット
  - FIXED BINARY
  - FIXED DECIMAL
  - FLOAT BINARY
  - FLOAT DECIMAL
- 算術文字 PICTURE
- CHARACTER
- BIT
- GRAPHIC
- WIDECHAR

10 進数表記と 2 進数表記の間で変換が行われると、値が変わることがあります。2 進数と 10 進数の間の変換では、係数 3.32 が次のように使用されます。

- n 桁の 10 進数は、CEIL (n*3.32) 桁の 2 進数に変換されます。
- n 桁の 2 進数は、CEIL (n/3.32) 桁の 10 進数に変換されます。

変換の計算には、表 22 にある CEIL 値の表を参照してください。

*表 22. CEIL (n*3.32) の値と CEIL (n/3.32) の値の表*

n	CEIL (n*3.32)	n	CEIL (n/3.32)
1	4	1-3	1
2	7	4-6	2
3	10	7-9	3
4	14	10-13	4
5	17	14-16	5
6	20	17-19	6
7	24	20-23	7
8	27	24-26	8
9	30	27-29	9
10	34	30-33	10
11	37	34-36	11
12	40	37-39	12
13	44	40-43	13
14	47	44-46	14
15	50	47-49	15
16	53[1]	50-53	16
17	57	54-56	17
18	60	57-59	18
19	64	60-63	19
20	67	64-66	20
21	70	67-69	21
22	74	70-73	22
23	77	74-76	23
24	80	77-79	24
25	83	80-83	25
26	87	84-86	26
27	90	87-89	27
28	93	90-92	28
29	97	93-96	29
30	100	97-99	30
31	103	100-102	31
32	107	103-106	32
33	110	107-109	33
		110-112	34
		113-116	35

注 1: ceil(16*3.32) = 54 のときは、53 という値が使用されます。そうでない場合には、浮動 10 進数 (16) が 2 進数に変換され、 (浮動 2 進数 (54) が拡張浮動小数点として表示されるので) 長精度の浮動小数点から拡張精度の浮動小数点に変換されることになります。

固定小数点整数値の場合は、変換しても値は変わりません。小数部分を持つ固定小数点の値の場合は、因数 3.32 によって必要なだけの桁数またはビット数が用意され、変換後の値ともとの値の違いは最低位の 1 桁 (1 ビット) 未満に抑えられます。

たとえば、属性が FIXED DECIMAL (1,1) である 10 進定数 .1 は、.0001B という 2 進数値に変換され、したがって 1/10 は 1/16 に変換されます。属性が FIXED DECIMAL (2,2) である 10 進定数 .10 は、.0001100B という 2 進数値に変換され、したがって 10/100 は 12/128 に変換されます。

ソースからターゲットへの変換規則

ターゲット: コード化算術フォーマット

ソース:

FIXED BINARY、FIXED DECIMAL、
FLOAT BINARY、および FLOAT DECIMAL

これらはすべてコード化算術データです。コード化算術データ間での変換の規則については、ターゲットがこれらの各データ・タイプである場合に、その説明箇所で述べています。

算術文字 PICTURE

データは、まず 10 進数に変換されます。そのスケールと精度は、対応する PICTURE 指定によって決まります。次に、この 10 進値がターゲットの基数、スケール、モード、および精度に変換されます。 FIXED DECIMAL または FLOAT DECIMAL をソースとして使用しているコード化算術データの特定のターゲット・タイプの説明を参照してください。

CHARACTER

ソース・ストリングは有効な算術定数または複素数式でなければなりません。それ以外の場合は、CONVERSION 条件が生じます。定数の頭に符号が付いていてもよく、定数の前後にブランクがあってもかまいません。符号と定数の間には、ブランクがあってはなりません。また、複素数式では、実数部分の終わりと虚数部分の先頭の符号との間にブランクがあってはなりません。

定数は、基数、スケール、モード、および精度の属性を持ちます。割り当ての場合のように、ターゲットの属性がソースの属性と無関係であるときは、定数はターゲットの属性に変換されます。ソースとして定数の属性を使用しているコード化算術データの特定のターゲット・タイプの説明を参照してください。

演算式の計算の場合のように中間結果が必要なときは、中間結果の属性は、もとのストリングの代わりに精度 (M,0) の 10 進固定小数点値が使われている場合の中間結果の属性と同じです。(このため、コンパイラーは、定数の属性には関係なく、あらゆる場合に対処できるコードを生成することができます。) この結果、定数の小数部分が失われることがあります。ソースとして FIXED DECIMAL を使用しているコード化算術データの特定のターゲット・タイプの説明を参照してください。

文字データ項目から中間の 10 進固定小数点への最初の変換では、値が中間結果のデフォルト・サイズを超える可能性があります。超える場合、使用可能になっていれば、SIZE 条件が起こります。

複素数を表す文字ストリングが実数のターゲットに割り当てられる場合、ストリングの実数部分だけがターゲットに割り当てられるので、そのストリングの虚数部分が正しい算術文字であるかどうかの検査は行われず、CONVERSION 条件も起こりません。

ソースがヌル・ストリングまたは 1 つ以上のブランクから成るストリングである場合、ターゲットにはゼロの値が割り当てられます。 CONVERSION 条件は起こりません。

BIT
演算式の計算時に変換が起こる場合は、ソースのビット・ストリングは FIXED BINARY(M,0) の符号なし値に変換されます。 FIXED BINARY をソースとして使用しているコード化算術データの特定のターゲット・タイプの説明を参照してください。

ソース・ストリングが、使用可能な精度よりも長いときは、左側のビットが無視されます。ゼロ以外のビットが失われると、SIZE 条件が起こります。

ヌル・ストリングはゼロの値になります。

GRAPHIC
グラフィック変数とストリングが CHARACTER に変換されたあとは 82 ページで説明されている文字ソースの規則に従ってください。

WIDECHAR
ワイド文字変数とストリングが CHARACTER に変換された後は、82 ページで説明されている文字ソースの規則に従ってください。

ターゲット・フィールド: FIXED BINARY (p2,q2)

ソース:

FIXED DECIMAL (p1,q1)
結果の精度は、p2 = min(N,1+CEIL(p1*3.32)) で、
q2=CEIL(ABS(q1*3.32))*SIGN(q1) です。

FLOAT BINARY (p1)
80 ページの『算術値の精度の変換』で示したように、精度が変換されます。 p1 は宣言または指定されている値で、q1 は、2 進小数点の位置で決められ、指数の値で変更されます。

FLOAT DECIMAL (p1)
FIXED DECIMAL から FIXED BINARY への変換の場合と同じように精度が変換されます。p1 は宣言または指定されている値で、q1 は 10 進小数点の位置で決められ、指数の値で変更されます。

算術文字 PICTURE
82 ページのターゲット: コード化算術を参照してください。

CHARACTER
82 ページのターゲット: コード化算術を参照してください。

BIT
82 ページのターゲット: コード化算術を参照してください。

GRAPHIC
82 ページのターゲット: コード化算術を参照してください。

WIDECHAR
82 ページのターゲット: コード化算術を参照してください。

ターゲット・フィールド: FIXED DECIMAL (p2,q2)

ソース:

FIXED BINARY (p1,q1)
結果の精度は、p2=1+CEIL(p1/3.32) で、q2=CEIL(ABS(q1/3.32))*SIGN(q1) です。

FLOAT BINARY (p1)
FIXED BINARY から FIXED DECIMAL への変換の場合と同じように精度が変換されます。p1 は宣言または指定されている値で、q1 は 2 進小数点の位置で決められ、指数の値で変更されます。

FLOAT DECIMAL (p1)
80 ページの『算術値の精度の変換』で説明したように、精度が変換されます。p1 は宣言または指定されている値で、q1 は 10 進小数点の位置で決められ、指数の値で変更されます。

算術文字 PICTURE
82 ページのターゲット: コード化算術を参照してください。

CHARACTER
82 ページのターゲット: コード化算術を参照してください。

BIT
82 ページのターゲット: コード化算術を参照してください。

GRAPHIC
82 ページのターゲット: コード化算術を参照してください。

WIDECHAR
82 ページのターゲット: コード化算術を参照してください。

ターゲット・フィールド: FLOAT BINARY (p2)

ソース:

FIXED BINARY (p1,q1)
結果の精度は、p2=p1 です。指数は、値の小数部分を表します。

FIXED DECIMAL (p1,q1)
結果の精度は、p2=CEIL(p1*3.32) です。指数は、値の小数部分を表します。

FLOAT DECIMAL (p1)
結果の精度は、p2=CEIL(p1*3.32) です。

算術文字 PICTURE
82 ページのターゲット: コード化算術を参照してください。

CHARACTER
82 ページのターゲット: コード化算術を参照してください。

BIT
82 ページのターゲット: コード化算術を参照してください。

GRAPHIC
82 ページのターゲット: コード化算術を参照してください。

WIDECHAR
82 ページのターゲット: コード化算術を参照してください。

ターゲット・フィールド: FLOAT DECIMAL (p2)

ソース:

FIXED BINARY (p1,q1)
結果の精度は、p2=CEIL(p1/3.32) です。指数は、値の小数部分を表します。

FIXED DECIMAL (p1,q1)
結果の精度は、p2=p1 です。指数は、値の小数部分を表します。

FLOAT BINARY (p1)
結果の精度は、p2=CEIL(p1/3.32) です。

算術文字 PICTURE
82 ページのターゲット: コード化算術を参照してください。

CHARACTER
82 ページのターゲット: コード化算術を参照してください。

BIT
82 ページのターゲット: コード化算術を参照してください。

GRAPHIC
82 ページのターゲット: コード化算術を参照してください。

WIDECHAR
82 ページのターゲット: コード化算術を参照してください。

ターゲット・フィールド: 算術文字 PICTURE

算術文字 PICTURE データ項目は、10 進固定小数点または 10 進浮動小数点の値を文字表記で表したものです。ソースから算術文字 PICTURE への変換に関する以下の説明では、左端または右端の桁を失わずに割り当てを行うのに必要なターゲットの属性を示してあります。

ソース:

FIXED BINARY (p1,q1)
ターゲット・フィールドが次の条件を満たしている必要があります。

```
fixed decimal (1+x+q-y,q) or
float decimal (x)
```

ただし、x>=CEIL(p1/3.32)、y=CEIL(q1/3.32)、かつ q>=y であること。

FIXED DECIMAL (p1,q1)
ターゲット・フィールドが次の条件を満たしている必要があります。

```
fixed decimal (x+q-q1,q) or
float decimal (x)
```

ただし、x>=p1 かつ q>=q1 であること。

FLOAT BINARY (p1)
ターゲット・フィールドが次の条件を満たしている必要があります。

```
fixed decimal (p,q) or
float decimal (p)
```

ただし、p>=CEIL(p1/3.32)、かつ p と q の値はソースの指数で表しうる値の範囲を考慮していること。

FLOAT DECIMAL (p1)
ターゲット・フィールドが次の条件を満たしている必要があります。

```
fixed decimal (p,q) or
float decimal (p)
```

ただし、p>=p1、かつ p と q の値はソースの指数で表しうる値の範囲を考慮していること。

算術文字 PICTURE
ソースの暗黙定義される属性は、FIXED DECIMAL または FLOAT DECIMAL です。該当ターゲットに関する項目を参照してください。

CHARACTER
82 ページのターゲット: コード化算術を参照してください。

BIT(n)
ターゲット・フィールドが次の条件を満たしている必要があります。

```
fixed decimal (1+x+q,q) or
float decimal (x)
```

ただし、x>=ceil(n/3.32) かつ q>=0 であること。

GRAPHIC
82 ページのターゲット: コード化算術を参照してください。

WIDECHAR
82 ページのターゲット: コード化算術を参照してください。

ターゲット: CHARACTER

ソース:

FIXED BINARY、FIXED DECIMAL、
FLOAT BINARY、および FLOAT DECIMAL
以下に説明するように、コード化算術値は 10 進定数 (負数の場合は頭に負符号 (-) が付けられる) に変換されます。その定数は、ソースの属性から長さが導出される中間の文字ストリングに入れられます。次に、この中間のストリングが、ストリングの割り当ての規則に従ってターゲットに割り当てられます。

コード化算術値から文字ストリングへの変換の規則は、リスト・ディレクティブ出力、データ・ディレクティブ出力、およびキーの評価 (REGIONAL ファイルの場合も) でも使用されます。

FIXED BINARY (p1,q1)

まず、2 進数の精度 (p1,q1) が、同等の 10 進数の精度 (p,q) に変換されます。ただし、p=1+CEIL(p1/3.32) および q=CEIL(ABS(q1/3.32))*SIGN(q1) です。そのあとの規則は、FIXED DECIMAL から CHARACTER への変換の規則と同じです。

FIXED DECIMAL (p1,q1)

p1>=q1>=0 の場合

- 定数は、幅 p1+3 のフィールドで右寄せされます。(負符号 (-)、10 進または 2 進小数点、および小数点の値の先行ゼロを入れる場合に備えて、3 桁を用意しておく必要があります。)
- 小数部分の小数点のすぐ前にあるゼロ 1 つを除いて、先行ゼロがブランクで置き換えられます。ソースの値がゼロのときも、ゼロ 1 つが残されます。
- 負数であれば、最初の桁の前に負符号 (-) が付けられます。正の値には符号は付けられません。
- q1=0 のときは、小数点は入りません。 q1>0 のときは、小数点が入れられ、定数の小数部分は q 桁です。

p1<q1 または q1<0 の場合は、定数の右方にスケール因数が付加されます。定数は任意指定の符号付き整数です。項目の値がゼロであっても、スケール因数が付けられ、下記の構文になります。

```
F{+|-}nn
```

ただし、{+|-}nn の値は -q1 です。

中間結果のストリングの長さは p1+k+3 です。ただし、k は、q1 の値が入りきるのに必要な桁数です (符号と英字 F は桁数に数えられません)。

算術値が複素数の場合、中間結果のストリングは、虚数部分が実数部分に連結した形になります。左側の実数部分はソースの実数部分として生成され、右側の虚数部分は、符号と英字 I が必ず付加されます。このようにして生成されたストリングは、2 つのエレメントの間にブランクをはさんでいない複素数式となります。この中間結果のストリングの長さは次のとおりです。

```
2*p1+7     for p1>=q1>=0
2*(p1+k)+7 for p1<q1 or q1<0
```

下の例は、実数および複素数の固定小数点 10 進数値から生成される、中間結果のストリングを示します。

```
Precision     Value     String

(5,0)       2947        'bbbb2947'
(4,1)       -121.7      'b-121.7'
(4,-3)      -3279000    '-3279F+3'
(2,1)       1.2+0.3I    'bbb1.2+0.3I'
```

FLOAT BINARY (p1)

まず、浮動小数点 2 進数の精度 (p1) が同等の浮動小数点 10 進数の精度 (p)

に変換されます。ただし、p=CEIL(p1/3.32) です。そのあとの規則は、FLOAT DECIMAL から CHARACTER への変換の規則と同じです。

FLOAT DECIMAL (p1)
ソースが 10 進浮動小数点数のときは、E(w,d,s) という形の E フォーマット項目で伝送される場合と同様に変換されます。ただし、以下のような条件です。

w、中間ストリングの長さは p1+8 です。

d、小数桁の数は p1-1 です。

s、有効数字の数は p1 です。

算術値が複素数の場合、中間結果のストリングは、虚数部分が実数部分に連結した形になります。左側の実数部分はソースの実数部分として生成され、右側の虚数部分は、符号と英字 I が必ず付加されます。このようにして生成されたストリングは、2 つのエレメントの間にブランクをはさんでいない複素数式となります。中間結果のストリングの長さは 2*p+17 です。

下の例は、実数および複素数の浮動小数点 10 進数値から生成される、中間結果のストリングを示します。

```
Precision    Value          String

(5)          1735*10**5     'b1.7350E+0008'
(5)          -.001663       '-1.6630E-0003'
(3)          1              'b1.00E+0000'
(5)          17.3+1.5I      'b1.7300E+0001+1.5000E+0000I'
```

算術文字 PICTURE
実数の算術文字フィールドは、文字ストリングと解釈され、文字ストリングの長さを変換する際の規則に従ってターゲット・ストリングに割り当てられます。複素数の算術文字フィールドの場合は、実数部分と虚数部分が連結されてから、ターゲット・ストリングに割り当てられます。そのターゲット・ストリングには挿入文字が含まれます。

BIT
ビットの 0 は文字の 0 になり、ビットの 1 は文字の 1 になります。ヌル・ビット・ストリングは、ヌル文字ストリングになります。生成された文字ストリングは、ストリングの長さを変換する際の規則に従ってターゲット・ストリングに割り当てられます。

GRAPHIC
DBCS が SBCS に変換されるのは、対応する SBCS 文字がある場合のみです。そうでない場合には、CONVERSION 条件が起こります。

WIDECHAR
ワイド文字から文字への変換は、すべてのワイド文字の値が '0080'wx より小さい場合にだけ実行されます。そうでない場合には、CONVERSION 条件が起こります。

ターゲット・フィールド: BIT

ソース:

FIXED BINARY、FIXED DECIMAL、
FLOAT BINARY、および FLOAT DECIMAL

必要に応じて、算術値は 2 進数に変換され、符号や小数部分は無視されます。(複素数の算術値の場合は、虚数部分も無視されます。) このようにして生成された 2 進値は、ビット・ストリングと見なされ、ストリングを割り当てる際の規則に従ってターゲット・ストリングに割り当てられます。

FIXED BINARY (p1,q1)

中間結果のビット・ストリングの長さは、次の式で求められます。

```
min(M,(p1-q1))
```

(p1-q1) が負数またはゼロのときは、ヌル・ビット・ストリングが生成されます。

下の例は、固定小数点 2 進数値から生成される、中間結果のストリングを示します。

```
Precision   Value    String

(1)           1      '1'B
(3)          -3      '011'B
(4,2)      1.25      '01'B
```

FIXED DECIMAL (p1,q1)

中間結果のビット・ストリングの長さは、次の式で求められます。

```
min(M,CEIL((p1-q1)*3.32))
```

(p1-q1) が負数またはゼロのときは、ヌル・ビット・ストリングが生成されます。

下の例は、固定小数点 10 進数値から生成される、中間結果のストリングを示します。

```
Precision   Value    String

(1)           1        '0001'B
(2,1)       1.1       '0001'B
```

FLOAT BINARY (p1)

中間結果のビット・ストリングの長さは、次の式で求められます。

```
min(M,p1)
```

FLOAT DECIMAL (p1)

中間結果のビット・ストリングの長さは、次の式で求められます。

```
min(M,ceil(p1*3.32))
```

算術文字 PICTURE

まず、データは、算術 PICTURE 指定で決められたスケールと精度を持つ 10 進数として解釈されます。次に、その項目は、FIXED DECIMAL または FLOAT DECIMAL から BIT への変換規則に従って変換されます。

CHARACTER

文字の 0 はビットの 0 になり、文字の 1 はビットの 1 になります。 0 または 1 以外の文字があると、CONVERSION 条件が起こります。ヌル・ストリングは、ヌル・ビット・ストリングになります。生成されたビット・ストリング

(その長さはソースの文字ストリングの長さと同じ) は、ストリングを割り当てる際の規則に従ってターゲットに割り当てられます。

GRAPHIC
グラフィックの 0 はビットの 0 になり、グラフィックの 1 はビットの 1 になります。0 または 1 以外のグラフィックがあると、CONVERSION 条件が起こります。ヌル・ストリングは、ヌル・ビット・ストリングになります。生成されたビット・ストリング (その長さはソースの漢字ストリングの長さと同じ) は、ストリングを割り当てる際の規則に従ってターゲットに割り当てられます。

WIDECHAR
ワイド文字の 0 ('0030'wx) はビットの 0 になり、ワイド文字の 1 ('0031'wx) はビットの 1 になります。0 または 1 以外のワイド文字があると、CONVERSION 条件が起こります。ヌル・ストリングは、ヌル・ビット・ストリングになります。生成されたビット・ストリング (その長さはソースのワイド文字ストリングの長さと同じ) は、ストリングを割り当てる際の規則に従ってターゲットに割り当てられます。

ターゲット: GRAPHIC

最初に、非グラフィック・ソースは、 86 ページの『ターゲット: Character』で説明した規則に従って文字に変換されます。そのあとで、その結果の文字ストリングが DBCS ストリングに変換されます。

ターゲット: WIDECHAR

最初に、ワイド文字以外のソースは、 86 ページの『ターゲット: Character』で説明した規則に従って文字に変換されます。そのあとで、その結果の文字ストリングがワイド文字ストリングに変換されます。

例

小数部分がある DECIMAL FIXED から小数部分がある BINARY FIXED への変換

```
dcl I fixed bin(31,5) init(1);
    I = I+.1;
```

この結果、I の値は 1.0625 になります。これは、.1 が FIXED BINARY (5,4) に変換され、2 進数の近似値が 0.0001B (端数は丸められません) となるためです。この数の 10 進表記は .0625 となります。 .1 の代わりに .1000 を指定した場合の結果は、これと異なります。

算術データからビット・ストリングへの変換

```
dcl A bit(1),
    D bit(5);
A=1;         /* A has value '0'B     */
```

```
D=1;         /* D has value '00010'B */
D='1'B;      /* D has value '10000'B */
if A=1 then go to Y;
       else go to X;
```

これは X への分岐になります。なぜなら、下記の一連の処置が取られ、A への割り当てが行われるからです。

1. 10 進定数の 1 の属性は FIXED DECIMAL(1,0) です。この定数は、属性が FIXED BINARY(4,0) である一時記憶域に割り当てられ、0001B という値を取ります。
2. 次に、この値が長さ (4) のビット・ストリングに変換されて、'0001'B になります。
3. ビット・ストリングは A に割り当てられています。A は長さ 1 に宣言されており、割り当てられる値は長さ 4 を獲得するので、右方で切り捨てが発生し、最終的には A は '0'B の値を持ちます。

IF ステートメント内の比較演算では、'0'B と 1 が FIXED BINARY に変換され、算術値として比較されます。両者は等しくないので、A=1 の関係は偽 の結果になります。

D への最初の割り当てでは、A への割り当て時と同じような一連の処置が行われます。ただし、割り当てられる値の右側にゼロが埋め込まれる点が異なっています。これは、D の長さが、割り当てられる値の長さより 1 だけ大きい長さとして宣言されているからです。

算術データから文字への変換

次に示す例では、負符号 (-)、10 進または 2 進小数点、および小数点の前の先行ゼロを入れる場合に備えて、3 桁のブランクを用意しておく必要があります。

```
dcl A char(4),
    B char(7);
A='0'; /*A has value '0bbb'*/
A=0;   /*A has value 'bbb0'*/
B=1234567; /*B has value 'bbb1234'*/
```

変換エラー

```
dcl Ctlno char(8) init('0');
do I=1 to 100;
Ctlno=Ctlno+1;
⋮
end;
```

この例では、システムで許可される FIXED DECIMAL の精度の最大値に 15 が使用されました。この例では、下記の一連の処理が行われるので、CONVERSION 条件が起こります。

1. CTLNO の初期値、すなわち '0bbbbbbb' は FIXED DECIMAL(15,0) に変換されます。
2. 属性 FIXED DECIMAL(1,0) を持つ 10 進定数 1 が追加されます。追加の規則に従って、結果の精度は (16,0) です。
3. 次に、この値を再び CTLNO に割り当てるために、この値の長さ 18 の文字ストリングに変換されます。

4. CTLNO の長さは 8 なので、割り当てによって右側が切り捨てられ、CTLNO の最終的な値はブランクだけになります。 2 回目のループでは、この値を算術タイプに正しく変換できません。

第 6 章 プログラムの編成

プログラム 93
プログラム構造 93
プログラムの活動化 94
プログラムの終了 95
ブロック 95
ブロックの活動化 95
ブロックの終了 96
パッケージ 96
プロシージャー 99
PROCEDURE ステートメントおよび ENTRY ステートメント 99
ENTRY ステートメント 100
パラメーター属性 102
プロシージャーの活動化 105
プロシージャーの終了 106
再帰的プロシージャー 107
外部プロシージャーの動的ロード 109
サブルーチン 112
例 1 112
例 2 113
組み込みサブルーチン 113
関数 113
例 114
組み込み関数 115
プロシージャーへの引数の引き渡し 116
BYVALUE および BYADDR の使用 116
仮引数 117
MAIN プロシージャーへの引数の引き渡し . . 118
開始ブロック 119
BEGIN ステートメント 119
開始ブロックの活動化 119
開始ブロックの終了 119
入り口データ 120
入り口定数 120
入り口変数 121
ENTRY 属性 122
OPTIONAL 属性 124
LIST 属性 125
LIMITED 属性 129
総称入り口 130
GENERIC 属性 130
入り口呼び出しまたは入り口値 133
CALL ステートメント 133
RETURN ステートメント 134
サブルーチンからの戻り 134
関数からの戻り 134
OPTIONS オプションとその属性 135
RETURNS オプションとその属性 144

この章では、いくつかのステートメントを各種のブロックにまとめて、PL/I プログラムを編成する方法、ブロック間で制御が移される方法、および異なるブロックで同一データを使用する方法を説明します。

多数のプログラマーが手分けして 1 つのプログラムを作成する場合は特に、プログラムを複数のブロックに適切に分割して作業を進めると、プログラムのコーディングもテストも簡単になります。プログラムを適切に分割すれば、ストレージの有効利用にもつながります。なぜなら、自動ストレージは、ストレージを宣言しているブロックに制御権が移動した時点で割り振られ、ブロックが終了するときに解放されるからです。

プログラム

プログラム構造

PL/I はブロックで編成される言語であり、パッケージ、プロシージャー、開始ブロック、ステートメント、式、および組み込み関数から構成されます。

PL/I *アプリケーション* は、*ロード・モジュール* と呼ばれる、 1 つまたは複数の個別にロード可能なエンティティーから構成されます。 ロード・モジュールは、1 つまたは複数の独立したコンパイル・エンティティー (*コンパイル単位 (CU)* と呼ばれます) から構成されます。 特に明記されていない限り、*プログラム* とは、 PL/I アプリケーションまたはコンパイル単位のことを意味します。

コンパイル単位は、PL/I PACKAGE または外部 PROCEDURE です。各パッケージには、0 個かそれ以上のプロシージャーが含まれます。プロシージャーのいくつかは搬出可能です。 PL/I 外部プロシージャーまたは内部プロシージャーには、 0 個かそれ以上のブロックが入ります。 PL/I ブロックは、 PROCEDURE または BEGIN ブロックのいずれかであり、それぞれのブロックに、ステートメントまたはブロック (あるいはその両方) が含まれます。(設定されない場合もあります)

PL/I ブロックで、高位モジュラー・アプリケーションを処理することができます。なぜなら、ブロックには、変数名やストレージ・クラスを定義する宣言を入れることができるからです。そのため、単一ブロックまたはブロック・グループに対して変数の有効範囲を制限することができます。あるいは、コンパイル単位またはロード・モジュール全体を通してそれを指定することができます。

ブロックをそれ自体に入れる程度を自由に決めることができるので、 PL/I は、コードを再使用できるよう、多数のコンパイル単位やアプリケーションが共用できるブロックを処理することができます。

図 2 で、アプリケーション構造を説明しています。

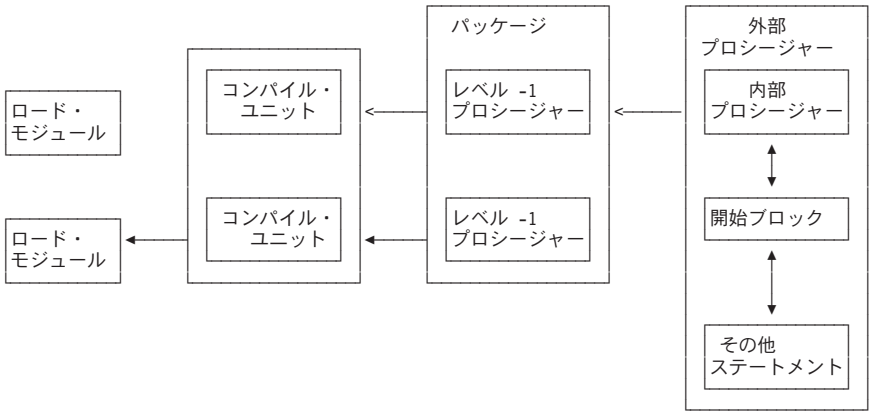


図 2. PL/I アプリケーション構造

パッケージについては、 96 ページの『パッケージ』で記載しています。

プロシージャーについては、 99 ページの『プロシージャー』で記載しています。

開始ブロックについては、 119 ページの『開始ブロック』で記載しています。

プログラムの活動化

PL/I プログラムがアクティブになるのは、呼び出し側プログラムが主プロシージャー を呼び出したときです。 この呼び出し側プログラムは、通常はオペレーティング・システムですが、別のプログラムでもかまいません。主プロシージャーは、外部プロシージャーであり、ステートメントに OPTIONS(MAIN) の指定があります。以下の例では、`Contrl` が主プロシージャーであり、プログラム内のほかの外部プロシージャーを呼び出します。主プロシージャーは、プログラムが実行されている間ずっとアクティブになっています。

```
Contrl: procedure options(main);
  call A;
  call B;
  call C;
end Contrl;
```

プログラムの終了

主プロシージャーが終了すると、プログラムは終了します。プログラムが正常に終了したか異常終了したかに関係なく、終了すると、呼び出し側プログラムに制御が戻されます。前の例では、制御は C プロシージャーから移されて、Contrl プロシージャーに戻り、Contrl プロシージャーで終了します。詳細については、106 ページの『プロシージャーの終了』を参照してください。

ブロック

ブロックとは、以下のことを行う、区切られた一連のステートメントのことです。

- ブロック内に宣言されている名前の有効範囲を設定します。
- 自動変数の割り振りを制限します。
- DEFAULT ステートメントの有効範囲を判別します(175 ページの『属性のデフォルト』に記載されています)。

ブロックには、以下のような種類があります。

- パッケージ
- プロシージャー
- 開始

これらのブロックには、名前のローカル定義と見なされる宣言を入れることができます。これは、名前の有効範囲を設定するため、および自動変数の割り振りを制限するために行われます。これらの宣言は、それ自身のブロックの外では知られることがなく、また、そのブロックを含んでいるブロック内でその名前を参照することはできません。詳細については、163 ページの『宣言の有効範囲』を参照してください。

ストレージは、ストレージが宣言されるブロックの入り口で自動変数に割り振られ、ブロックから出るときに解放されます。詳細については、163 ページの『宣言の有効範囲』を参照してください。

ブロックの活動化

割り振り内、ストレージの解放、名前の有効範囲の削除では、各ブロックは同じ役割を果たします。どのようにして活動化されるかということについては、99 ページの『プロシージャー』 および 119 ページの『開始ブロック』 に記載されています。パッケージは、活動化されず、終了もしません。

ブロックが活動化されているときには、以下のことが行われます。

- 宣言ステートメントに書かれている式は、エクステントおよび初期値 (反復因数も含めて) で計算されます。
- 自動変数用にストレージが割り振られ、指定されていれば初期設定値がセットされます。

- ストレージは仮引数用に割り振られ、このブロックで作成される可能性があるコンパイラー作成の一時域用にも割り振られます。

自動変数の初期値およびエクステントは、同じブロック内で宣言されたほかの自動変数の値やエクステントによって変化してはなりません。たとえば、以下の初期設定を行うと、 J と K についての無効な結果を生成する場合があります。

```
dcl I init(10),J init(K),K init(I);
```

データ項目の宣言は、相互に依存し合うものであってはなりません。たとえば、次のような宣言は正しくありません。

```
dcl A(B(1)), B(A(1));

dcl D(E(1)), E(F(1)), F(D(1));
```

ブロックが活動化される途中でエラーが発生し、ERROR 条件 (またはその他の条件) が起こることがあります。そのような場合は、そのブロックの環境が不完全であることがあります。特に、一部の自動変数が割り振られていないことがあります。 ERROR 条件の発生後に自動変数を参照するステートメントが実行されると、割り振られていないストレージを参照する可能性があります。割り振られていないストレージの参照の結果は未定義です。

ブロックの終了

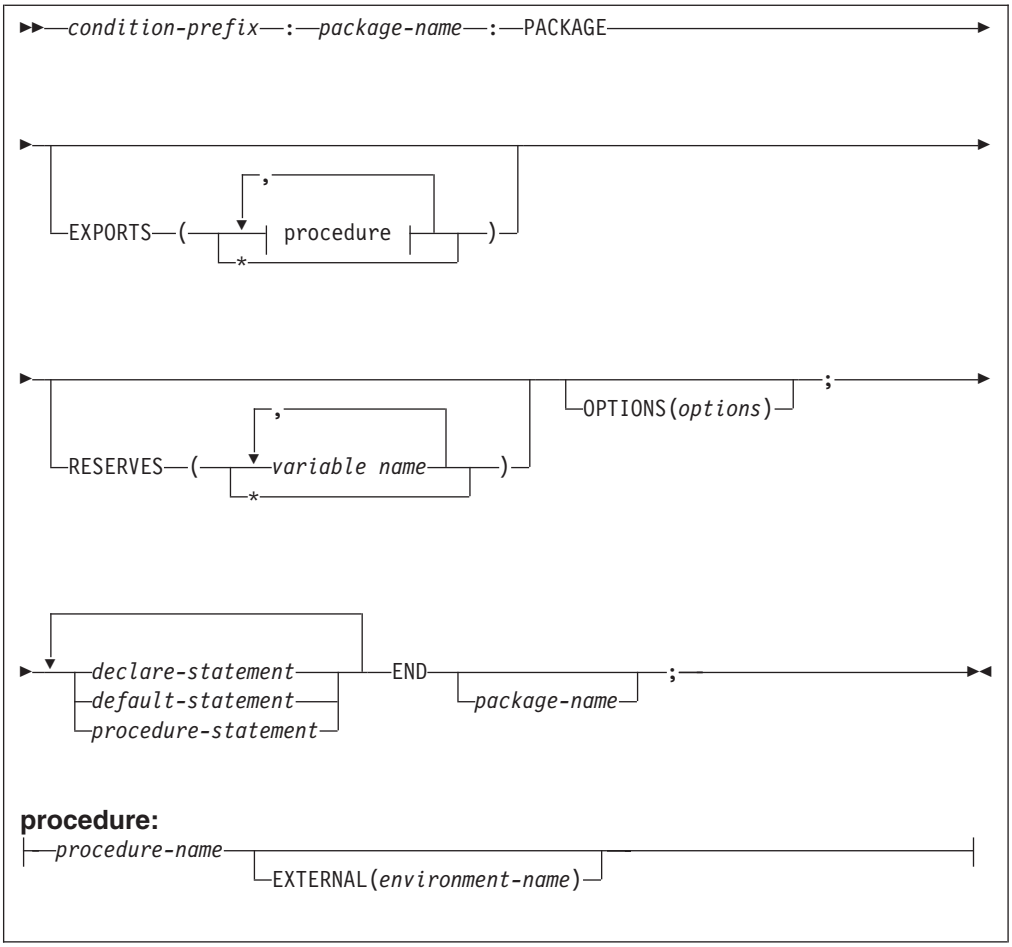
ブロックを終了する方法はいく通りもあります。どのようにして終了されるかということについては、 99 ページの『プロシージャー』 および 119 ページの『開始ブロック』 に記載されています。パッケージは、活動化されず、終了もしません。

ブロック終了時には、次のことが行われます。

- ON ユニット環境が、ブロックの活動化以前の状態に再設定されます。
- そのブロック内のすべての自動変数に割り振られたストレージが解放されます。

パッケージ

パッケージとは、宣言、デフォルト・ステートメント、およびプロシージャー・ブロックだけ入れることができるブロックのことです。名前が再び宣言されない限りは、パッケージは、パッケージ内に含まれるすべての宣言およびプロシージャーによって共用される名前の有効範囲を構成します。レベル 1 のプロシージャーの一部またはすべてが搬出され、外部プロシージャーとしてパッケージの外で認識できるようになります。パッケージは、複数のエントリー・ポイント・アプリケーションを実施する際に使用することができます。



condition-prefix

PACKAGE ステートメントで指定される条件接頭語は、PROCEDURE ステートメントで指定変更されない限り、パッケージ内に含まれているすべてのプロシージャーに適用されます。条件接頭語に関する詳細については、 357 ページの『条件接頭語』を参照してください。

package-name

パッケージの名前です。

EXPORTS

すべての (EXPORTS(*)) または名前のついたプロシージャーは、搬出され、パッケージ外で外部的に認識されることを指定します。 EXPORTS オプションを指定しないと、EXPORTS(*) が想定されます。

procedure name

パッケージ内のレベル 1 のプロシージャーの名前です。

EXTERNAL (environment name)

163 ページの『宣言の有効範囲』に記載されている有効範囲属性です。

RESERVES

このパッケージがすべて (RESERVES(*)) に対してストレージを予約しているか、あるいは RESERVED 属性を持つ名前付きの変数に対してだけストレージを予約しているかを指定します。 170 ページの『RESERVED 属性』を参照してください。

variable name
レベル 1 の外部静的変数の名前です。

OPTIONS オプション
パッケージ・ステートメントに適用する OPTIONS オプションについては、135 ページの『OPTIONS オプションとその属性』を参照してください。

declare statement
パッケージ内で宣言される変数は、その宣言がレベル 1 のプロシージャーの外部で行われる場合には、静的、基底付き、被制御のストレージ・クラスを持たなければなりません。自動変数を使用することはできません。デフォルトのストレージ・クラスは、STATIC です。 159 ページの『第 8 章 データ宣言』を参照してください。

default statement
175 ページの『属性のデフォルト』を参照してください。

procedure statement
99 ページの『PROCEDURE ステートメントおよび ENTRY ステートメント』を参照してください。

パッケージ・ステートメントの例は、図 3 に示されています。

```
*Process S A(F) LANGLVL(SAA2) LIMITS(EXTNAME(31)) NUMBER;
 Package_Demo: Package exports (Factorial);

 /***********************************************/
 /*                Common Data                  */
 /***********************************************/

 dcl N fixed bin(15);
 dcl Message char(*) value('The factorial of ');

 /***********************************************/
 /*                Main Program                 */
 /***********************************************/

 Factorial: proc options (main);
    dcl Result fixed bin(31);
    put skip list('Please enter a number whose factorial ' ||
                  'must be computed ');
    get list(N);
    Result = Compute_factorial(n);
    put list(Message || trim(N) || ' is ' || trim(Result));
 end Factorial;

 /***********************************************/
 /*                 Subroutine                  */
 /***********************************************/

 Compute_factorial: proc (Input) recursive returns (fixed bin(31));
    dcl Input fixed bin(15);
    if Input <= 1 then
       return(1);
    else
       return( Input*Compute_factorial(Input-1) );
 end Compute_factorial;

 end Package_Demo;
```

図 3. パッケージ・ステートメント

プロシージャー

プロシージャーとは、PROCEDURE ステートメントとそれに対応する END ステートメントとで区切られている一連のステートメントです。 プロシージャーは、主プロシージャー、サブルーチン、または関数などです。アプリケーションは、OPTIONS(MAIN) を持つ 1 つの外部プロシージャーを持たなければなりません。以下の例では、プロシージャーの名前は Name であり、プロシージャーのエントリー・ポイント を表しています。

```
Name:
   procedure;
end Name;
```

ENTRY ステートメントは、プロシージャーへの 2 次エントリー・ポイントを定義することができます。以下に例を示します。

```
Name: procedure;
B: entry;
end Name;
```

B は、Name プロシージャーへの 2 次エントリー・ポイントを定義します。ENTRY ステートメントについては、122 ページの『ENTRY 属性』を参照してください。

プロシージャーは、1 つの名前を持たなければなりません。別のプロシージャーまたは開始ブロック内にネストされたプロシージャー・ブロックは、*内部プロシージャー* と呼ばれます。 別のプロシージャーまたは開始ブロック内にネストされていないプロシージャー・ブロックは、*外部プロシージャー* と呼ばれます。パッケージから搬出されたレベル 1 のプロシージャーも、外部プロシージャーになります。外部プロシージャーは、ほかのコンパイル単位のほかのプロシージャーで呼び出すことができます。プロシージャーは、ほかのプロシージャーを呼び出すことができます。

プロシージャーを再帰的にすることができます。つまり、その中から再活動化したり、すでに活動している別の活動プロシージャー内から再活動化したりすることができます。プロシージャーを呼び出すときに引数を渡すことができます。

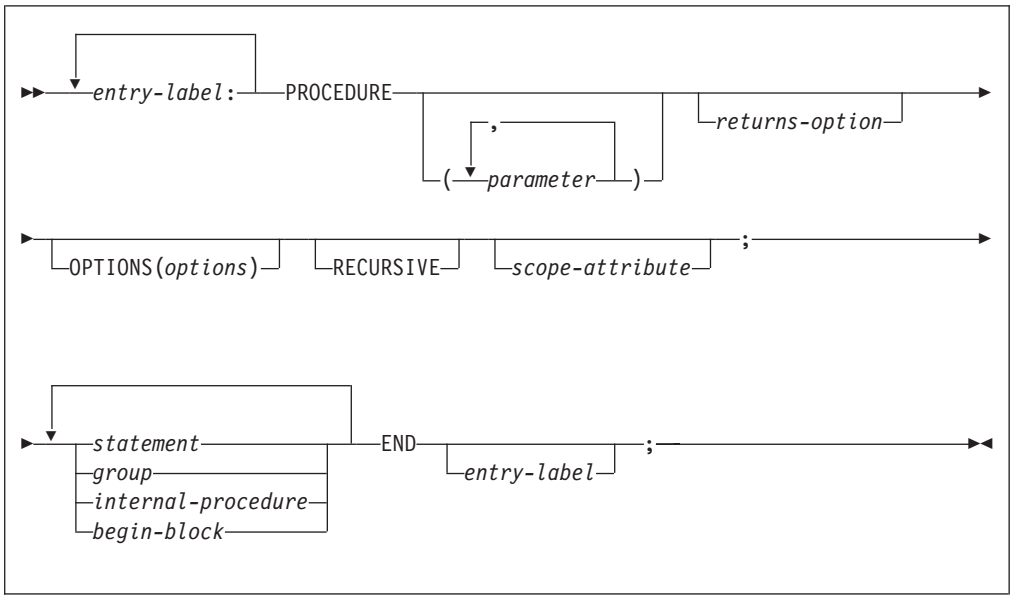
これらのサブジェクトに関する詳細については、以下の節を参照してください。

- 163 ページの『宣言の有効範囲』
- 112 ページの『サブルーチン』
- 113 ページの『関数』
- 116 ページの『プロシージャーへの引数の引き渡し』

PROCEDURE ステートメントおよび ENTRY ステートメント

プロシージャー (サブルーチンまたは関数) は、1 つまたは複数のエントリー・ポイントを持つことができます。プロシージャーへの 1 次エントリー・ポイントは、プロシージャー・ステートメントの左端のラベルによって確立されます。プロシージャーへの 2 次エントリー・ポイントは、 PROCEDURE ステートメントの追加のラベルおよび ENTRY ステートメントによって確立されます。各エントリー・ポイントには 1 つの入り口名があります。外部名の作成規則についての説明は、166 ページの『INTERNAL 属性と EXTERNAL 属性』を参照してください。

PROCEDURE ステートメントは、プロシージャーを主プロシージャー、サブルーチン、または関数として識別します。プロシージャーおよびその他の特性によって予期されるパラメーターも、プロシージャー・ステートメント上で指定することができます。



省略形: PROCEDURE の場合は PROC

entry-label
プロシージャーへのエントリー・ポイントです。外部入り口は、呼び出されるプロシージャー内で明示的に宣言されます。複数の入り口ラベルを指定すると、左端の名前が 1 次エントリー・ポイントとなり、それは PROCNAME と ONLOC 組み込み関数が戻す名前です。入り口データの詳細については、 120 ページの『入り口データ』を参照してください。

parameter
102 ページの『パラメーター属性』および 116 ページの『プロシージャーへの引数の引き渡し』を参照してください。

returns-option
関数プロシージャーについてのみ適用します。 113 ページの『関数』および 144 ページの『RETURNS オプションとその属性』を参照してください。

OPTIONS オプション
135 ページの『OPTIONS オプションとその属性』を参照してください。

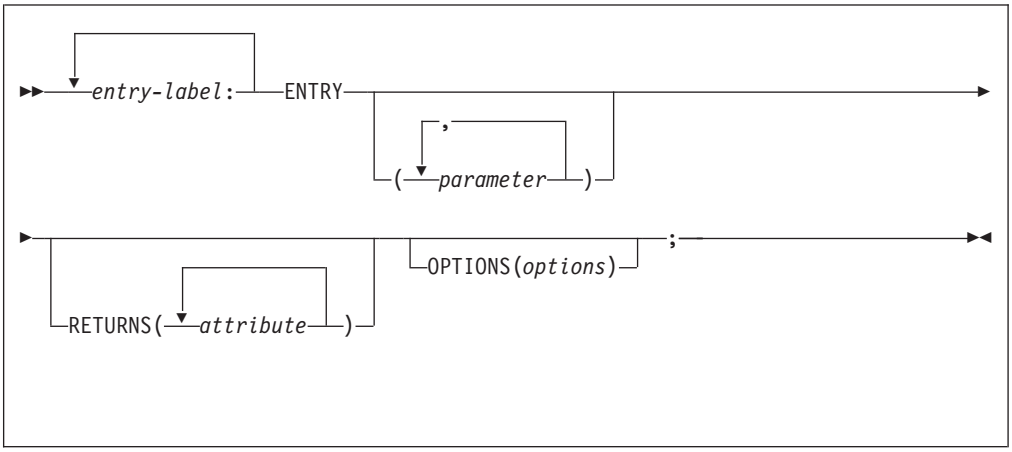
RECURSIVE
107 ページの『再帰的プロシージャー』を参照してください。

scope-attribute
163 ページの『宣言の有効範囲』を参照してください。

ENTRY ステートメント

ENTRY ステートメントは、プロシージャーの 2 次エントリー・ポイントを指定します。 ENTRY ステートメントは、2 次エントリー・ポイントを定義するプロシー

ジャーの内部になければなりません。反復実行を指定する DO グループ内に置くこと、または ON ユニットの内部であることはできません。



entry-label
プロシージャーへの 2 次エントリー・ポイントです。

parameter
102 ページの『パラメーター属性』および 116 ページの『プロシージャーへの引数の引き渡し』を参照してください。

RETURNS オプション
144 ページの『RETURNS オプションとその属性』を参照してください。

OPTIONS オプション
135 ページの『OPTIONS オプションとその属性』を参照してください。

ENTRY ステートメントのパラメーターはすべて BYADDR でなければならず、またプロシージャーが ENTRY ステートメントを含んでいる場合は、そのプロシージャーに対する非ポインター・パラメーターはすべて BYADDR でなければなりません。

ENTRY ステートメントを含むプロシージャーに RETURNS オプションが指定されている場合 (または、プロシージャーに含まれる ENTRY ステートメントに RETURNS オプションが指定されている場合) は、次のことが必須です。

- そのプロシージャーと ENTRY ステートメントの RETURNS オプションすべてに、 BYADDR 属性が指定 (または、コンパイル時オプション DEFAULT(RETURNS(BYADDR)) によって暗黙指定) されている。
- さらに、これらのエントリー・ポイントのいずれかを呼び出すルーチンすべてが、 RETURNS(BYADDR) を指定して入り口を宣言しているか、DEFAULT(RETURNS(BYADDR)) コンパイラー・オプションを指定してコンパイルされている。

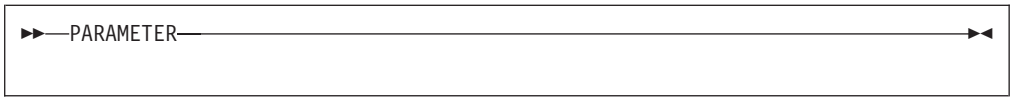
プロシージャーに ENTRY ステートメントがあり、あるエントリー・ポイントには RETURNS 属性が指定されているが、全エントリー・ポイントでは指定されていないときは、次の状況で ERROR 条件が検出されます。

- プロシージャーが RETURNS 属性がないエントリー・ポイントに入力されたときに、コードが式を使用して RETURN ステートメントを実行する場合。

- プロシージャーが、RETURNS 属性を持つエントリー・ポイントに入力されたときに、コードが式を使わずに RETURN ステートメントを実行する場合。

パラメーター属性

パラメーターは、PROCEDURE または ENTRY ステートメント内に指定したパラメーター属性でコンテキストにもとづいて宣言されます。パラメーターは、適切な属性を使用して明示的に宣言されなければなりません。 PARAMETER 属性は宣言内に指定することもできます。属性が DECLARE ステートメント内に指定されていない場合には、デフォルトの属性が適用されます。パラメーター名は、添え字を付けたり、修飾することはできません。



29 ページの表 8 と以下では、パラメーターに対して宣言できる属性を説明します。

パラメーターには、常に INTERNAL 属性があります。

パラメーターが構造体または共用体である場合には、レベル 1 の名前が指定されなければなりません。

パラメーターは、CONTROLLED ストレージ・クラス属性しか持つことができません。被制御パラメーターは、被制御引数を持つ必要があり、 INITIAL 属性も持つことができます。

レコード単位の入出力で使用されるパラメーターまたは DEFINED 項目の基本変数としてのパラメーターは、連結ストレージ内になければなりません。 CONNECTED 属性は、プロシージャー内の宣言、およびプロシージャー入り口宣言の記述子リスト内の両方で指定されなければなりません。

単純パラメーターの境界、長さ、およびサイズ

単純パラメーターの境界、長さ、およびサイズは、アスタリスクまたは制限付き式のいずれかによって指定されなければなりません。実際の長さ、境界、またはサイズが起動のたびに異なるときには、それぞれをアスタリスクで DECLARE ステートメントに指定できます。アスタリスクが使用されている場合は、関連した引数の現行世代から長さ、境界、またはサイズが取得されます。

関連した引数が仮引数である場合、集合体のエレメントであるストリングの長さ指定としてアスタリスクを使用することはできません。ストリングの長さは整数として指定する必要があります。

被制御パラメーターの境界、長さ、およびサイズ

被制御パラメーターの境界、長さ、またはサイズは、アスタリスクや要素式を使用して DECLARE ステートメントに指定することができます。

アスタリスクの表記法: アスタリスクを使用するとき、被制御パラメーターの長さ、境界、またはサイズは、関連した引数の現行世代から取得されます。異なる長さ、境界、またはサイズを ALLOCATE ステートメントに指定しないかぎり、以後

の被制御パラメーターの割り振りは、同じ境界、長さ、またはサイズを使用して行われます。引数の現行世代が存在しない場合、アスタリスクはパラメーターの次元数だけを決定します。そのパラメーターに対してその他の参照を行うには、呼び込まれたプロシージャーの ALLOCATE ステートメントによって、事前に被制御パラメーターの境界、長さ、またはサイズが指定されている必要があります。

式の表記法: パラメーターが割り振られるたびに、式が計算されて、新しい割り振りの現行境界、長さ、またはサイズを指定します。ただし、その DECLARE ステートメントの式は、 ALLOCATE ステートメント自体の境界、長さ、またはサイズの指定によって指定変更されます。

パラメーターの付いた配列引数の例: 104 ページの図 4 では、Sub1 が呼び出されると、割り振られる A および B が渡されます。

```
%process or('|') num  margins(1,72);
Package:package exports(*);


Main: procedure options(main);
declare (A(NA), B(NB), C(NC), D(ND) ) controlled;
declare (NA init(20), NB init(30), NC init(100),
ND init(100) ) fixed bin(31);
declare Sub1 entry((*) controlled, (*) controlled);
declare Sub2 entry ((*) ctl, (*) ctl, fixed bin);


allocate A,B;  /* A(20), B(30) */
display ('Gen1: DIM(A)=' || dim(A) || ', ' || "DIM(B)=" || dim(B));
call Sub1(A,B);


display ('Gen2: Allocn(A)=' || allocn(a) || ', ' ||
'Allocn(B)=' || allocn(B) );
display ('Gen2: DIM(A)=' || dim(A) || ', ' || "DIM(B)=" || dim(B));
free A,B;
display ('Gen1: Allocn(A)=' || allocn(A) || ', ' ||
'Allocn(B)=' || allocn(B) );
display ('Gen1: DIM(A)=' || dim(A) || ', ' || "DIM(B)=" || dim(B));
free A,B;
display ('Gen0: Allocn(A)=' || allocn(A) || ', ' ||
'Allocn(B)=' || allocn(B) );
call Sub2 (C,D,10);


display ('Gen1: Allocn(C)=' || allocn(C) || ', ' ||
'Allocn(D)=' || allocn(D) );
display ('Gen1: DIM(C)=' || dim(C) || ', ' || "DIM(D)=" || dim(D));
free C,D;
display ('Gen0: Allocn(C)=' || allocn(c) || ', ' ||
'Allocn(D)=' || allocn(D) );
end Main;


Sub1: procedure (U,V);
dcl (U(UB), V(*)) controlled,
UB fixed bin(31);
display ('Gen1: Allocn(U)=' || allocn(U) || ', ' ||
'Allocn(V)=' || allocn(V) );
display ('Gen1: DIM(U)=' || dim(U) || ', ' || "DIM(V)=" || dim(V));
UB=200;
allocate U,V; /* U(200), V(30) */
display ('Gen2: Allocn(U)=' || allocn(U) || ', ' ||
'Allocn(V)=' || allocn(V) );
display ('Gen2: DIM(U)=' || dim(U) || ', ' || "DIM(V)=" || dim(V));
end Sub1;


Sub2: procedure (X,Y,N);
dcl (X(N),Y(N)) controlled,
N fixed bin;
display ('Gen0: Allocn(X)=' || allocn(X) || ', ' ||
'Allocn(Y)=' || allocn(Y) );
allocate X,Y;  /* X(10), Y(10) */
display ('Gen1: Allocn(X)=' || allocn(X) || ', ' ||
'Allocn(Y)=' || allocn(Y) );
display ('Gen1: DIM(X)=' || dim(X) || ', ' || "DIM(Y)=" || dim(Y));
end Sub2;


end Package;
```

図 4. パラメーターの付いた配列引数

Sub1 内の ALLOCATE ステートメントは、 A と B の 2 番目の世代を割り振ります。 B は両方の世代に同じ境界を持ち、A は 2 番目の世代には異なる境界を持ちます。

Main へ戻る際、最初の FREE ステートメントは (Sub1 で割り振られている) A と B の 2 番目の世代を解放します。 2 番目の FREE ステートメントは、(Main で割り振られている) A と B の最初の世代を解放します。

Sub2 では、X と Y は、N の値によって異なる境界を使用して宣言されています。X と Y が割り振られたら、それらの値を使用して、割り振られた配列の境界を判別します。

Sub2 から Main へ戻る際、FREE ステートメントは、 (Sub2 で割り振られている) C と D の世代のみを解放します。

プロシージャーの活動化

プログラムの流れではプロシージャーは迂回され、PROCEDURE ステートメントの前にあるステートメントが実行されたあと、そのプロシージャーの END ステートメントのあとにあるステートメントが実行されます。プロシージャーは、*プロシージャー参照* によってしか活動化されません。(94 ページの『プログラムの活動化』では、主プロシージャーを活動化する方法が説明されています。) 呼び出しプロシージャーの実行は、呼び出されたプロシージャーが呼び出すプロシージャーに制御を戻すまで遅らされます。

下記のいずれかのコンテキストの中に入り口式が書かれていることを、プロシージャー参照といいます。

- 133 ページの『CALL ステートメント』で説明されているように、CALL ステートメントを使用してサブルーチンを呼び出すコンテキスト
- 113 ページの『関数』で説明されているように、関数を呼び出すコンテキスト

この節の情報は、これらのコンテキストと関係があります。ただしこの章の例では、CALL ステートメントを使用しています。

プロシージャー参照が実行されると、指定されたエントリー・ポイントを含むプロシージャーは*呼び出される* ことになります。 プロシージャー参照が書かれている場所を*呼び出し点* といい、参照が行われるブロックを*呼び込み側ブロック* といいます。呼び出し側ブロックは、呼び出されたプロシージャーに制御が移されても、アクティブのままです。

1 次エントリー・ポイントからプロシージャーが呼び出されると、引数およびパラメーターは関連付けられ、呼び出されたプロシージャーの最初のステートメントから実行が始まります。プロシージャーが ENTRY ステートメントで 2 次エントリー・ポイントから呼び出されると、 ENTRY ステートメントに続く最初のステートメントから、実行が始まります。 1 次エントリー・ポイントで、あるブロックの入り口について確立された環境は、同じブロックが 2 次エントリー・ポイントで呼び出される時に確立される環境と同じです。

2 つのプロシージャー間の通信は、呼び出しプロシージャーから渡される引数、呼び込まれたプロシージャーから戻される値、および双方のプロシージャーにおいて既知の名前のいずれかによって行われます。したがって、同じプロシージャーでも、呼び出し点が異なれば異なるデータを処理することができます。以下に例を示します。

```
Readin: procedure;
statement-1
statement-2
Errt: entry;
statement-3
statement-4
end Readin;
```

以下の入り口の参照で活動化されます。

```
call Readin;
call Errt;
```

call Readin ステートメントは、その 1 次エントリー・ポイントで Readin を呼び出し、 statement-1 で実行が開始されます。 call Errt ステートメントは 2 次エントリー・ポイント Errt で、 Readin プロシージャーを呼び出し、実行は statement-3 で開始されます。入り口定数 (Readin) を、プロシージャー参照で使用される入り口変数に割り当てることもできます。 以下に例を示します。

```
declare Readin entry,
        Ent1 entry variable;
Ent1 = Readin;
call Ent1;
call Readin;
```

2 つの CALL ステートメントは、同じ効果があります。

プロシージャーの終了

プロシージャーが終了するのは、プロシージャー参照以外の方法で、呼び出し側プログラム、呼び出し側ブロックに制御を戻したとき、または別のアクティブ・ブロックに制御を渡したときです。

次の場合は、プロシージャーは*正常に* 終了します。

- プロシージャー内の RETURN ステートメントに制御が達したとき。 RETURN ステートメントが実行されると、呼び出し側プロシージャー内の呼び出し点に制御が戻されます。呼び出し点が CALL ステートメントであれば、CALL の次のステートメントで、呼び出し側プロシージャー内の実行が再開します。呼び出し点が関数参照であれば、その参照を含んでいるステートメントの実行が再開します。
- プロシージャー内の END ステートメントに制御が達したとき。この場合、RETURN ステートメントが実行されたときと同じ結果になります。

次の場合は、プロシージャーは*異常* 終了します。

- プロシージャーの外へ制御権を移動する GO TO ステートメントに制御が達したとき。 GO TO ステートメントを含んでいるブロック内のラベルか、またはプロシージャーに渡されたラベル引数に関連付けられているパラメーターを指定することができます。 単一スレッド・プログラムの現行スレッド内、またはマルチスレッド・プログラムの任意のスレッド内で、STOP ステートメントが実行されたとき。
- EXIT ステートメントが実行されたとき。
- ERROR 条件が起こり、ERROR または FINISH の ON ユニットが確立されていないとき。さらに、一方または両方の条件の ON ユニットが確立されているが、ON ユニットから出る方法が GO TO ステートメントではなく正常な戻りによって出るようになっているとき。
- プロシージャーが、異常終了する別のプロシージャーを呼び出したとき。

GO TO ステートメントを使用してプロシージャーの外へ制御権を移動すると、場合によっては、いくつかのプロシージャーや開始ブロックが終了することがあります。具体的に言えば、GO TO ステートメントで指定した移動先の場所が、終了しよ

うとしているブロックを直接活動化したのではないブロックに含まれている場合には、活動化の順序においてその 2 つのブロックの間にあるすべてのブロックが終了します。次に例を示します。

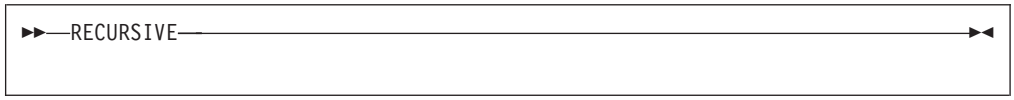
```
A: procedure options(main);
   statement-1
   statement-2
   B: begin;
      statement-b1
      statement-b2
      call C;
      statement-b3
   end B;
   statement-3
   statement-4
   C: procedure;
      statement-c1
      statement-c2
      statement-c3
      D: begin;
         statement-d1
         statement-d2
         go to Lab;
         statement-d3
      end D;
      statement-c4
   end C;
   statement-5
Lab: statement-6
   statement-7
end A;
```

A は B を、B は C を、 C は D を、それぞれ活動化します。 D の中の go to Lab ステートメントは、 A の中の statement-6 に制御を移します。このステートメントは、D、C、または B には含まれていないので、この 3 つのブロックがすべて終了します。 A は活動化されたままです。したがって、D の外へ制御権を移動することによって、ブロック D が終了するだけでなく、介在している B と C も終了します。

再帰的プロシージャー

それ自体から呼び出される、または別のアクティブのプロシージャーから呼び出されるアクティブのプロシージャーは、*再帰的* プロシージャーです。そのような呼び出しは、再帰と呼ばれます。

再帰的に呼び出されたプロシージャーは、PROCEDURE ステートメントで指定された RECURSIVE 属性を持たなければなりません。



再帰プロシージャーの呼び出しごとの環境 (自動変数の値など) は、被制御変数の割り振りのスタッキングと類似した方法で保存されます (241 ページの『被制御ストレージとその属性』を参照)。環境は、再帰的呼び出し時に、*最後に記憶された環境* がスタックに入れられ、その呼び出しの終了時にスタックから*除去される* と考えてください。現行ブロック内のラベル定数は、常に、そのラベルが入っているブロックの現行呼び出しへの参照になります。

ラベル定数が特定呼び出しのラベル変数に割り当てられ、ラベル変数が再帰的プロシージャー内部で宣言されていない場合には、別の呼び出しでその変数を指定する GO TO ステートメントが、現行および介在するプロシージャーや開始ブロックを終了し、割り当てが行われたときの環境を復元します。

再帰的プロシージャー内から入り口変数を通して呼び出された別のプロシージャーの環境は、その入り口定数がその変数に割り当てられたときに存在していた環境です。次の例を考えてみてください。

```
I=1;
call A;                                 /* First invocation of A      */

A: proc recursive;
   declare Ev entry variable static;
   if I=1 then
      do;
        I=2;
        Ev=B;
        call A;                         /* 2nd invocation of A        */
      end;
   else call Ev;                        /* Invokes B with environment */
                                        /* of first invocation of A   */
B: proc;
   go to Out;
end B;
Out: end A;
```

プロシージャー B の中の GO TO ステートメントは、 1 回目に呼び出された A 内の END A ステートメントに制御を移し、B を終了させるほかに、2 つの A を終了させます。

再帰が自動変数に及ぼす影響

再帰的プロシージャーの 1 回の活動化時に割り振られた変数の値は、次にそのプロシージャーが活動化されたときに変更されないようにしなければなりません。これは、変数のスタッキングによって解決されます。 スタッキングは後入れ先出し法で処理されるため、自動変数の最も新しい世代だけを参照することができます。静的変数は、再帰の影響を受けないので、再帰的に呼び出されたプロシージャー相互間の連絡に使用できます。これは、再帰的プロシージャーを含んでいるプロシージャー内で宣言された自動変数にも、被制御変数と基底付き変数にも適用されます。次に例を示します。

```
A: proc;
dcl X;
:
B: proc recursive;
 dcl Z,Y static;
 call B;
:
end B;
end A;
```

変数 X の世代は 1 つであり、プロシージャー B の呼び出しのつど代わることはありません。変数 Z の世代は、プロシージャー B の呼び出しのつど代わります。変数 Y は、プロシージャー B の中でだけ参照することができ、呼び出しのつど再割り振りされることはありません (変数のスタッキングという概念は、 241 ページの『被制御ストレージとその属性』の被制御変数の説明でも重要です)。

外部プロシージャーの動的ロード

モジュールは、FETCH および RELEASE ステートメントを使用して PL/I プログラムによって動的に取り出されたり、(ロードされたり)、解放されたり (削除されたり) します。

プロシージャー参照によって呼び出されたプロシージャーは、通常、プログラムの実行中ずっと主記憶域に常駐します。ただし、必要とされるときにのみ、プロシージャーを主記憶域にロードしておくこともできます。つまり、呼び出しプロシージャーの実行中に、呼び出されたプロシージャーを主記憶域に動的にロードし、あとで主記憶域から動的に削除します。

プロシージャーの動的ロードと動的削除が特に効果的なのは、呼び出しプロシージャーが実行されるつど、呼び出されたプロシージャーが必ず呼び出されるとは限らないときや、実行時間の短縮よりも主記憶域の節減のほうが重要であるときです。

FETCH ステートメントで入り口定数を指定すると、その入り口定数を含んでいるプロシージャーのコピーがすでに主記憶域に存在しているのでない限り、そのプロシージャーをまず主記憶域にロードしなければ実行できないことを表します。 FETCH ステートメントで名前が参照されれば、以下によってディスクからプロシージャーをロードすることも可能です。

- CALL ステートメントまたは INITIAL 属性の CALL オプションの実行
- 関数参照の実行

CALL または関数参照の実行前にも実行後にも、FETCH ステートメントまたは RELEASE ステートメントに制御を渡す必要はありません。

どちらのステートメントがプロシージャーをロードした場合も、CALL ステートメント、CALL オプション、または関数参照が実行されると、普通の方法でプロシージャーが呼び出されます。

プロシージャーがすでに主記憶域にロードされていても、エラーにはなりません。取り出されたプロシージャーは、プログラム全体の実行が完了するまで主記憶域に残しておくことができます。 RELEASE ステートメントを使用すれば、そのプロシージャーが占有していたストレージをいつでも解放することができ、別の目的で使用することができます。

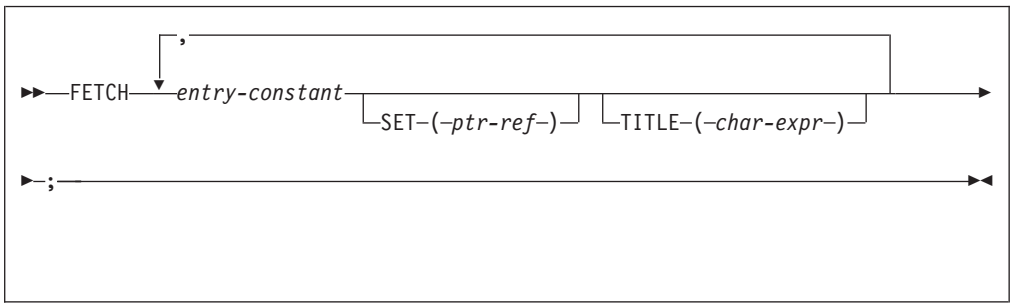
規則と機能

FETCH と RELEASE は以下の規則と機能を備えています。

- 外部プロシージャーだけを取り出すことができます。
- EXTERNAL ファイルと CONDITION 条件は、アプリケーション全体 (主モジュールおよび取り出されたモジュール) で共有されます。それ以外の外部変数は、単一モジュール内でのみ共用されます。
- 取り出されたプロシージャー中の STATIC 変数のストレージは、そのプロシージャーが入っているロード・モジュールがメモリーにロードされるときに割り振られます。ロード・モジュールがメモリーにロードされるたびに、STATIC 変数には、それらの宣言が示す初期値が指定されます。
- FETCH ステートメントおよび RELEASE ステートメントは、入り口定数を指定しなければなりません。取り出されるプロシージャーの入り口定数は、プロシージャーが取り出されている限り、入り口変数に割り当てることができます。

FETCH ステートメント

FETCH ステートメントは、指定されたプロシージャーが主記憶域に存在しているかを調べます。主記憶域内に存在していないプロシージャーは、ディスクからロードされます。



entry-constant

取り出されるプロシージャーの名前で、オペレーティング・システムに知られているものを指定します。取り出し可能なプロシージャーに関連する考慮事項の詳細については、「*プログラミング・ガイド*」を参照してください。

entry-constant (入り口定数) は、対応する CALL ステートメント、 CALL オプション、または関数参照内で使用する入り口定数と同じでなければなりません。

SET

ロードしたモジュールのエントリー・ポイントのアドレスにセットされるポインター参照 (*ptr-ref*) を指定します。このオプションは、テーブル (実行できないロード・モジュール) をロードするために使用することができます。また、非 PL/I プロシージャーへ渡さなければならないアドレスを持つ、取り出された項目に対しても使用することができます。

ロード・モジュールを後に RELEASE ステートメントによって解放して、さらにそのロード・モジュールに (ポインターを使用して) アクセスすると、予期しない結果が生じます。

TITLE

TITLE の場合、char-expr は任意の文字式、または文字式に変換可能な任意の式です。 TITLE が指定されると、指定されたロード・モジュール名が探索され、ロードされます。指定されていない場合には、使用されるロード・モジュール名は、(表示されていれば) 変数の EXTERNAL 属性で指定された環境名、または入り口定数名自体になります。

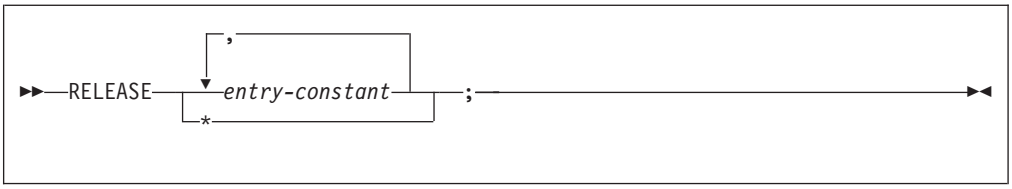
以下に例を示します。

```
dcl A entry;
dcl B entry ext('C');
dcl T char(20) varying;
T = 'Y';

fetch A title('X');           /* X is loaded */
fetch A;                      /* A is loaded */
fetch B title('Y');           /* Y is loaded */
fetch B;                      /* C is loaded */
fetch B title(T);             /* Y is loaded */
```

タイトル・ストリングの詳細については、「*プログラミング・ガイド*」を参照してください。

RELEASE ステートメント

RELEASE ステートメントは、指定した entry constant (入り口定数) が識別するプロシージャーによって占められた主記憶域を解放します。



entry constant
対応する CALL ステートメント、CALL オプション、関数参照、および FETCH ステートメントに使用する入り口定数と同じでなければなりません。RELEASE * は、それ以前にフェッチ (FETCH) されたすべての PL/I モジュールを解放します。 RELEASE ステートメントは、フェッチ (FETCH) されたモジュール内部から実行してはなりません。

以下の例を考えてください。ProgA と ProgB は、ディスクに常駐しているプロシージャーの入り口名です。

```
Prog:  procedure;

1          fetch ProgA;
2          call ProgA;
3          release ProgA;

4          call ProgB;
           go to Fin;

           fetch ProgB;
Fin:  end Prog;
```

1 ProgA は、最初の FETCH ステートメントによって主記憶域にロードされます。

2 ProgA は、最初の CALL ステートメントに達したときに実行されます。

3 ProgA のストレージは、RELEASE ステートメントが実行されると解放されます。

4 ProgB は、このプロシージャーを参照する FETCH ステートメントが実行されなくても、2 番目の CALL ステートメントに達すると、ロードされ、実行されます。

FETCH ProgA ステートメントが省略された場合でも同じ結果になります。RELEASE ステートメント内に ProgA を指定すると、それを呼び出したときと同様に、CALL ProgA ステートメントがプロシージャーをロードします。

取り出されたプロシージャーは、呼び出しプロシージャーとは別々にコンパイルされ、リンクされます。 FETCH ステートメント、RELEASE ステートメント、CALL ステートメント、 CALL オプション、および関数参照で指定される入り口定数は、ディスク上で認識される名前でなければなりません。これは「プログラミング・ガイド」に説明されています。

サブルーチン

サブルーチン は、CALL ステートメントで呼び出される内部または外部プロシージャーです。サブルーチンの構文については、 99 ページの『プロシージャー』を参照してください。

CALL ステートメントの引数は、呼び出されたプロシージャーのパラメーターと関連付けられます。サブルーチンは活動化されると、実行が開始します。引数 (ゼロまたはそれ以上) は、入力専用、出力専用、またはその両方が可能です。

サブルーチンは、RETURN ステートメントまたは END ステートメントで正常に終了します。それから、制御が呼び込み側ブロックに戻ります。サブルーチンは、106 ページの『プロシージャーの終了』で説明しているように、異常終了することもあります。

サブルーチン・プロシージャーは、以下のものでなければなりません。
- プロシージャー・ステートメントに RETURNS オプションを持っていないもの。
- 外部プロシージャーである場合には、RETURNS 属性の付いた入り口として宣言されないもの。
- 関数参照ではなく、CALL ステートメントを使用して呼び出されるもの。
- RETURN ステートメントを使用して結果の値を戻さないもの。

以下の例では、呼び込み側ブロックにとって外部および内部になるサブルーチンの呼び出しを示しています。

例 1

```
Prmain:  procedure;
         declare Name character (20),
         Item bit(5),
4        Outsub entry;
1        call Outsub (Name, Item);
end Prmain;

2    Outsub:  procedure (A,B);
              declare A character (20),
              B bit(5);
3             put list (A,B);
end Outsub;
```

1 Prmain の CALL ステートメントは、 Name と Item という引数で **2** の Outsub プロシージャーを呼び出します。

2 Outsub は、Prmain から渡される Name と Item にそのパラメーターの A と B を関連付けます。 Outsub が実行されると、A に対する各参照は、Name に対する参照と見なされ、B に対する各参照は、 Item に対する参照と見なされます。

3 put list (A,B) ステートメントは、デフォルトの出力ファイル SYSPRINT へ Name と Item の値を送ります。

4 入り口定数として Outsub の宣言では、引数およびパラメーターの属性が突き合わせられるので、パラメーター記述子は ENTRY 属性で指定する必要はありません。 122 ページの『ENTRY 属性』も参照してください。

例 2

```
A:  procedure;
    declare Rate float (10),
            Time float(5),
            Distance float(15),
            Master file;
1       call Readcm (Rate, Time, Distance, Master);

3     Readcm:
2       procedure (W,X,Y,Z);
        declare W float (10),
                X float(5),
                Y float(15), Z file;
        get File (Z) list (W,X,Y);
        Y = W * X;
        if Y > 0 then
          return;
        else
          put list('ERROR READCM');
        end Readcm;

      end A;
```

1 引数 Rate、Time、Distance、および Master は、 **3** の Readcm プロシージャーに渡され、パラメーター W、X、Y、および Z と関連付けられます。

2 W に対する参照は Rate に対する参照と同じであり、 X に対する参照は Time、 Y に対する参照は Distance、 Z に対する参照は Master とそれぞれ同じになります。

3 Readcm は A で明示的に宣言されていないことに注意してください。PROCEDURE ステートメントの指定に ENTRY 属性が暗黙に宣言されています。

組み込みサブルーチン

作成済みのプログラミング・タスクを備えた、*組み込みサブルーチン* を使用することができます。それらの*組み込み名* は、BUILTIN 属性で明示的に宣言することができます。 (BUILTIN 属性の情報または組み込み関数の説明について詳しくは、395 ページの『第 19 章 組み込み関数、疑似変数、およびサブルーチン』を参照してください。)

関数

関数 とは、ゼロまたはそれ以上の引数を持つプロシージャーであり、式で*関数参照*を使用して呼び出されます。関数参照は制御を関数プロシージャーに移します。関数プロシージャーは制御と値を戻します。この値が式の計算で関数参照に置き換わることになります。集合を戻すことはできません。また、ENTRY 変数も、LIMITED 属性を持っていない限り戻すことはできません。式の計算は引き続き行われます。

関数プロシージャーは以下のものでなければなりません。

- プロシージャー・ステートメントに RETURNS オプションがあるもの。
- 外部プロシージャーである場合には、RETURNS 属性の付いた入り口として宣言されるもの。

- 関数参照を使用して呼び出されるもの。 CALL ステートメントは、戻り値に OPTIONAL 属性がある場合だけ、それを呼び出すために使用することができます。この場合には、戻り値は、戻りによって破棄されます。 RETURN の代わりに END を使用すると、予測しない結果になることがあります。
- RETURNS オプション内の属性と RETURNS 属性が合致するもの。
- RETURN ステートメントを使用して、制御と結果の値を戻すもの。

関数が呼び出されるときはいつでも、呼び出された式の引数はエントリー・ポイントのパラメーターと関連付けられます。それから、制御はエントリー・ポイントに戻されます。関数が活動化され、実行が開始します。

RETURN ステートメントは、関数を終了して、その式に指定された値を呼び出し式に戻します。詳細については、 134 ページの『RETURN ステートメント』を参照してください。

106 ページの『プロシージャーの終了』で説明されているように、関数は異常終了することもあります。異常終了した場合は、その関数を呼び出した式の計算は完了されずに、指定のステートメントに制御が移ります。

関数が定義されるので、引数リストを必要としない場合もあります。そのような場合、式の中に書かれている外部関数名が関数参照として認識されるのは、その関数名が入り口名であると明示的に宣言されている場合だけです。詳細については、133 ページの『入り口呼び出しまたは入り口値』を参照してください。

例

以下の例は、呼び込み側ブロックの内部および外部にある関数の呼び出しを示しています。

例 1

以下の例では、代入ステートメントに Sprod 関数への参照が含まれています。

```
Mainp: procedure;
       get list (A, B, C, Y);
1      X = Y**3+Sprod(A,B,C);

2      Sprod: procedure (U,V,W)
       returns (bin float(21));
       dcl (U,V,W) bin float(53);
       if U > V + W then
3        return (0);
       else
3        return (U*V*W);
 end Sprod;
```

1 Sprod が呼び出されると、引数 A、 B、および C はそれぞれ、 **2** のパラメーター U、V、および W と関連付けられます。

2 RETURNS がプロシージャー・ステートメントに指定されているので、Sprod は関数です。これは、内部関数であるため、明示的な入り口宣言は必要ありません。 Sprod が外部関数であれば、 Mainp は RETURNS を指定した入り口宣言を含みます。

3 Sprod は、ゼロを戻すか、または U*V*W が表す値を戻します。制御は Mainp 内の式に戻されます。戻り値は、関数参照の値として用いられ、式の計算は続きます。

例 2

```
Mainp:  procedure;
        dcl Tprod entry (bin float(53),
                         bin float(53),
                         bin float(53),
                         label) external
          returns (bin float(21));
        get list (A,B,C,Y);
1        X = Y**3+Tprod(A,B,C,Lab1);
Lab1:  call Errt;
end Mainp;

1   Tprod:  procedure (U,V,W,Z)
            returns (bin float(21));
            dcl (U,V,W) bin float(53);
            declare Z label;

2           if U > V + W then
                go to Z;
3           else
                return (U*V*W);
     end Tprod;
```

1 Tprod が呼び出されると、 Lab1 はパラメーター Z と関連付けられます。

2 U が V + W よりも大きい場合には、 Lab1 というラベルの付いたステートメントで Mainp に制御を戻します。 **1** の割り当て式の計算は、継続されません。

3 U が V + W より大きくない場合には、 U*V*W は計算されて、通常の方法で Mainp に戻されます。 **1** での割り当て式の計算は、引き続き行われます。

Tprod は外部プロシージャーであることに注意してください。 RETURNS が含まれている Mainp には明示的な入り口宣言があります。

組み込み関数

プログラマーが作成した関数プロシージャーを使用できるようにするため、 PL/I では、*組み込み関数* のセットを提供しています。 組み込み関数には、共通して使用される算術関数のほかに、ストレージなどを使用してストリングおよび配列を処理するための関数があります。組み込み関数の呼び出し方法は、プログラマー定義の関数の呼び出しと同じです。しかし、プログラマー定義の関数は 1 つのエレメント値しか返せませんが、多くの組み込み関数は値の配列を返すことができます。組み込み関数の組み込み名は、BUILTIN 属性で明示的に宣言することができます。(BUILTIN 属性の情報または組み込み関数の説明について詳しくは、 395 ページの『第 19 章 組み込み関数、疑似変数、およびサブルーチン』を参照してください。)

プロシージャーへの引数の引き渡し

関数またはサブルーチンが呼び出されると、パラメーターは、左から右の順に、渡される引数と関連付けられます。

概要は次のとおりです。

- 計算データの引数を、計算データ・タイプのパラメーターに渡すことができます。
- プログラム制御データの引数は、同じタイプのパラメーターに渡さなければなりません。ただし、次の場合はこの例外となります。
  - ポインターとオフセットは、相互に渡されます。
  - LIMITED ENTRY は ENTRY に渡されますが、ENTRY は LIMITED ENTRY に渡されません。
  - ラベル定数の配列は、引数として使用することはできません。

構造体引数が定数エクステントとともに宣言されていないかぎり、構造体から派生する一時的な集合体を必要とする引数は、使用できません。

引数リストの式は、サブルーチンまたは関数が呼び出される前に呼び込み側ブロックで計算されます。パラメーターには、それと関連付けられるストレージはありません。パラメーターは、呼び出されたプロシージャーが、呼び出し側のプロシージャーに割り振られているストレージにアクセスできるようにするための手段を提供するためだけのものです。

BYVALUE および BYADDR の使用

引数に BYVALUE が渡されない限り、引数の値は別として、引数への参照は、一般的にサブルーチンか関数に渡されます。これは、参照または BYADDR によって引数を渡すことによって認識されます。プロシージャー内のパラメーターへの参照は、対応する引数への参照になります。パラメーターの値への変更は、実際には、対応する引数の値への変更になります。ただし、これは常に可能ではなく、好ましいものではありません。たとえば、定数は呼び出されたプロシージャーによって変更されるべきではありません。変更すべきではない引数には、元の引数の値を含む *仮引数* が渡されます。パラメーターへの参照は、仮引数への参照であり、元の引数に対するものではありません。

BYADDR を指定すると、コンパイラーによって、パラメーター・リストに対応する引数のアドレスが書き込まれます。BYVALUE を指定すると、パラメーター・リストに引数の値が書き込まれます。

BYVALUE を指定すると、仮引数は作成されません が、呼び出されたルーチンで対応するパラメーターに行われたすべての変更は、呼び出しルーチンでは不可視になります。これは、仮引数の場合も同様です。

BYVALUE は、コンパイル時に既知の長さとサイズを持つスカラー引数およびスカラー・パラメーターに対してのみ指定できます。

仮引数

仮引数は、引数が以下のいずれかの場合に作成されます。

- 定数 (パラメーターが NONASSIGNABLE 属性を持つ場合を除く)
- 演算子、括弧、または関数参照を使用する式
- データ属性、位置合わせ属性、または接続属性がパラメーターに宣言された属性と異なる変数。

  境界、長さ、またはサイズだけが異なり、しかもこれらがアスタリスクを使用して宣言されている場合、および定数以外の式を使用して被制御パラメーターのエクステントを定義している場合には、上記は制御されていないパラメーターには適用されません。定数以外の式を使用して被制御パラメーターのエクステントを定義している場合、引数とパラメーターのエクステントは一致すると見なされます。

  PICTURE 属性を持つ引数とパラメーターの場合、反復因数が適用されたあと、ピクチャー指定が正確に一致しない限り、仮引数が作成されます。唯一の例外は、スケール因数に正符号が付いている引数またはパラメーターは、正符号の付いていないパラメーターまたは引数と一致することです。
- 被制御ストリングまたは区域 (ALLOCATE ステートメントは長さやエクステントを変更することができるため)。
- 調整可能な長さやサイズを持つストリングまたは区域 (長さやサイズが一定の制御されないパラメーターに関連付けられるもの)

仮引数属性の派生

PL/I では、以下のものから仮引数の属性を派生します。

- 内部プロシージャーの関連パラメーターに宣言された属性。
- 外部入り口宣言の関連パラメーターのパラメーター記述子に指定された属性。このパラメーターの記述子がなかった場合には、定数または式の属性が使用されます。
- 配列境界の引数、ストリングの長さ、または区域のサイズのエクステント (宣言にアスタリスクが付けられている場合)。

仮引数の規則

以下の規則が、仮引数に適用されます。

- パラメーターがエレメント (つまり、構造体または配列のいずれでもない変数) である場合には、引数は要素式でなければなりません。
- VARYING または VARYINGZ ストリング・エレメントを、長さが未定義である (つまり長さがアスタリスクで指定されている) NONVARYING パラメーターに渡した場合、元の引数の長さを持つ仮引数が作成されます。
- 引数として渡される入り口変数は位置合わせされたものと見なされるため、引数とパラメーターの位置合わせだけが異なる場合は、仮引数は作成されません。入り口パラメーターの総称名引数の説明については、130 ページの『総称入り口』を参照してください。
- パラメーターがプログラム制御タイプ (ロケーターを除く) である場合、引数は同じデータ・タイプの参照でなければなりません。

- パラメーターがロケーター (ポインターまたはオフセット) である場合には、引数はロケーターでなければなりません。タイプが異なる場合には、仮引数が作成されます。オフセット・パラメーターのパラメーター記述子で、関連区域を指定してはなりません。
- 制御されていないパラメーターを、任意のストレージ・クラスの引数と関連付けることができます。ただし、複数の引数の世代が存在するときは、パラメーターは呼び出し時に存在する世代とだけ関連付けられます。
- パラメーターが被制御状態のときは、このことを ENTRY 宣言のパラメーター記述子に明示的に指定しなければなりません。さらに、被制御パラメーターは、以下の条件を満たす対応する被制御引数を常に持っていなければなりません。
  - 添え字が付いていない。
  - 構造体のエレメントではない。
  - 仮引数を作成しない。

  複数の引数の世代が呼び出し時に存在している場合には、存在している世代のすべてのスタックにパラメーターは対応します。その結果として呼び出し時に、被制御パラメーターは、対応する引数の現行世代を表します。関連する引数の割り振りスタックの操作を行うために、被制御パラメーターは割り振られ、呼び出されたプロシージャー内で解放されます。

  被制御パラメーターのエクステントがアスタリスクまたは制限なしの式として指定されている場合には、もとの宣言は、制限なしの式として宣言されたエクステントを持たなければなりません。

MAIN プロシージャーへの引数の引き渡し

主プロシージャーの PROCEDURE ステートメントは、パラメーター・リストを持つことができます。 PL/I では、そのようなパラメーターに特定考慮事項は必要ありません。ただし、呼び出し側プログラムの引数の要件 (たとえば、割り当てのターゲットとしてパラメーターを使用してはならないなど) に注意する必要があります。

呼び出し側プログラムがオペレーティング・システムであり、SYSTEM(MVS) コンパイラー・オプションとともにコンパイルされている場合:

- 単一引数がメイン・プロシージャーに渡され、そのパラメーターは CHARACTER VARYING と宣言される必要があります。
- このパラメーターの現行長は、実行時の引数の長さと等しくセットされます。そのため、以下の例では:

  ```
  Tom:  proc (Param) options (main);
     dcl Param char(100) varying;
  ```

  ストレージは、引数の現行長にだけ割り振られます。
- このパラメーターの内容は、OPTIONS(MAIN) とともに指定した 2 番目のオプションによって異なります。
  - OPTIONS(MAIN NOEXECOPS) を指定する場合、オペレーティング・システムによって PL/I に渡されるストリングは、そのままプログラムに渡されます。NOEXECOPS を指定することをお勧めします。
  - OPTIONS(MAIN) のみを指定すると、オペレーティング・システムによって PL/I に渡されるストリングは、最初に '/' を含むストリングまで、すべての文

字を削除したものになります。つまり、ストリングに '/' が含まれていない場合、プログラムはヌル・ストリングを受け取ることになります。

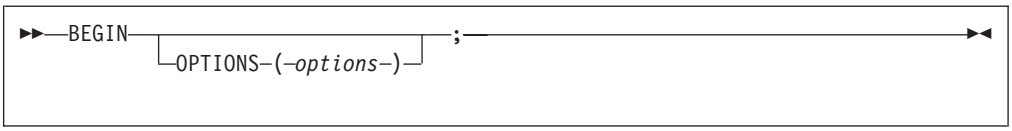
開始ブロック

開始ブロックは、BEGIN ステートメントとそれに対応する END ステートメントによって区切られている一連のステートメントです。 以下に例を示します。

```
B:  begin;
statement-1
statement-2
.
.
.
statement-n
end B;
```

BEGIN ステートメント

BEGIN ステートメントとそれに対応する END ステートメントは、開始ブロックを区切ります。



OPTIONS オプション

開始ブロック・オプションについては、135 ページの『OPTIONS オプションとその属性』を参照してください。

開始ブロックの活動化

開始ブロックは、通常の流れによって、あるいはステートメント IF、ON、WHEN、または OTHERWISE のユニットとして活動化されます。

GO TO ステートメントを使用して、ラベルのついた BEGIN ステートメントに制御権を移動することができます。

開始ブロックの終了

開始ブロックは、プロシージャー参照以外のなんらかの方法で、制御が別のアクティブのブロックに渡されたときに終了します。その方法は以下のとおりです。

- 開始ブロックの END ステートメントを実行すること。ブロックが ON ユニットであるとき以外は、物理的に END のあとのステートメントで制御は継続します。
- 開始ブロック内 (あるいは開始ブロックにとって内部である任意のブロック内) で GO TO ステートメントを実行すること。これにより、制御権はブロックの外部に移動します。
- STOP または EXIT ステートメントを実行すること。
- 制御権が RETURN ステートメントに到達すること。これにより、制御権は開始ブロックの外部および開始ブロックを含むプロシージャーの外部に移動します。

GO TO ステートメントで指定した移動先の場所が、終了しようとしているブロックを直接に活動化したのではないブロックに含まれている場合には、GO TO ステート

メントが実行されると他のブロックも終了します。つまり、活動化の順序においてその 2 つのブロックの間にあるすべてのブロックが終了します。この例については、106 ページの『プロシージャーの終了』を参照してください。

入りロデータ

入りロデータは、入り口定数または入り口変数の値です。

入り口定数は、 PROCEDURE または ENTRY ステートメントに接頭部として付けられた名前、または VARIABLE 属性ではなく ENTRY 属性で宣言された名前です。これは、入り口変数に割り当てられます。以下の例では、P、E1、および E2 が入り口定数です。 Ev は入り口変数です。

```
P:  procedure;
    declare Ev entry variable,
     (E1,E2) entry;


Ev = E1;
call Ev;
Ev = E2;
call Ev;
```

最初の CALL ステートメントは、エントリー・ポイント E1 を呼び出します。 2 番目の CALL はエントリー・ポイント E2 を呼び出します。

以下の例では、添え字のついた入り口変数の F(5) を宣言しています。

A、B、C、D、および E という 5 つの入り口は X、Y、および Z というパラメーターを使用してそれぞれ呼び出されます。

```
declare (A,B,C,D,E) entry,
declare F(5) entry variable initial (A,B,C,D,E);
do I = 1 to 5;
  call F(I) (X,Y,Z);
end;
```

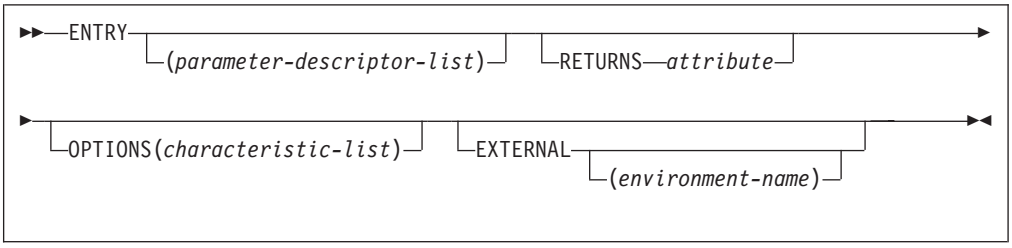
内部プロシージャーのエントリー・ポイントである入り口定数が入り口変数に割り当てられるときに、入り口定数が内部にあったブロックがアクティブのままである間は、割り当てられた値は有効なままです (再帰的プロシージャーの場合は、現行のままです)。

入り口定数

PROCEDURE または ENTRY ステートメントにラベル接頭部を指定すると、入り口定数を明示的に宣言することになります。パラメーター記述子リストは、パラメーター宣言、および (ある場合) デフォルトから得られます。

外部入り口定数は、明示的に宣言されなければなりません。この宣言は、以下のとおりです。

- 外部プロシージャーにエントリー・ポイントを定義します。
- オプションで、(ある場合) エントリー・ポイントのパラメーター記述子リスト (パラメーターの数とその属性) を指定します。
- 入り口が関数の場合にプロシージャーによって戻される値の属性を指定します。



属性は、任意の順序で示すことができます。

ENTRY 属性
ENTRY 属性の完全な構文については、122 ページの『ENTRY 属性』を参照してください。

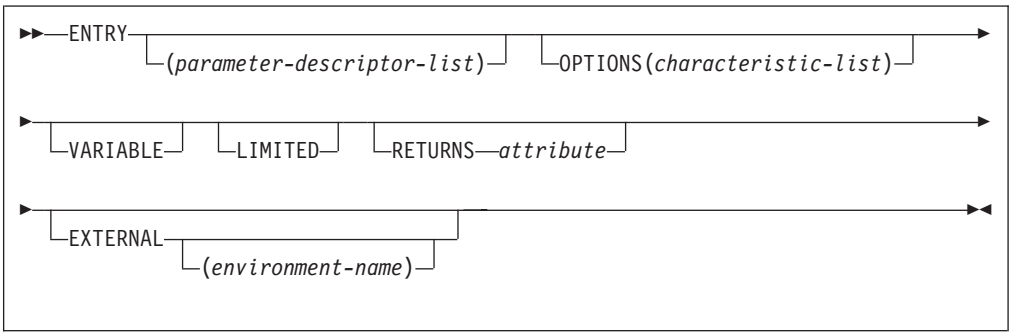
OPTIONS 属性
OPTIONS 属性の完全な構文については、135 ページの『OPTIONS オプションとその属性』を参照してください。

RETURNS 属性
RETURNS 属性の完全な構文については、144 ページの『RETURNS オプションとその属性』を参照してください。

EXTERNAL 属性
environment-name (環境名) を指定しないと、その名前は宣言と同じものになります。 EXTERNAL 属性の詳細については、166 ページの『INTERNAL 属性と EXTERNAL 属性』を参照してください。

入り口変数

入り口変数には、内部および外部入り口値の両方が含まれます。入り口変数は、集合の一部になることができます。構造化および配列の次元属性については、181 ページの『配列』および 185 ページの『構造体』を参照してください。



オプションは、任意の順序で示すことができます。

ENTRY 属性
122 ページの『ENTRY 属性』を参照してください。

OPTIONS 属性
135 ページの『OPTIONS オプションとその属性』を参照してください。

VARIABLE 属性
VARIABLE 属性は、入り口変数として名前を設定します。この変数には、入り口定数および変数が含まれます。構文情報については、 53 ページの『VARIABLE 属性』を参照してください。

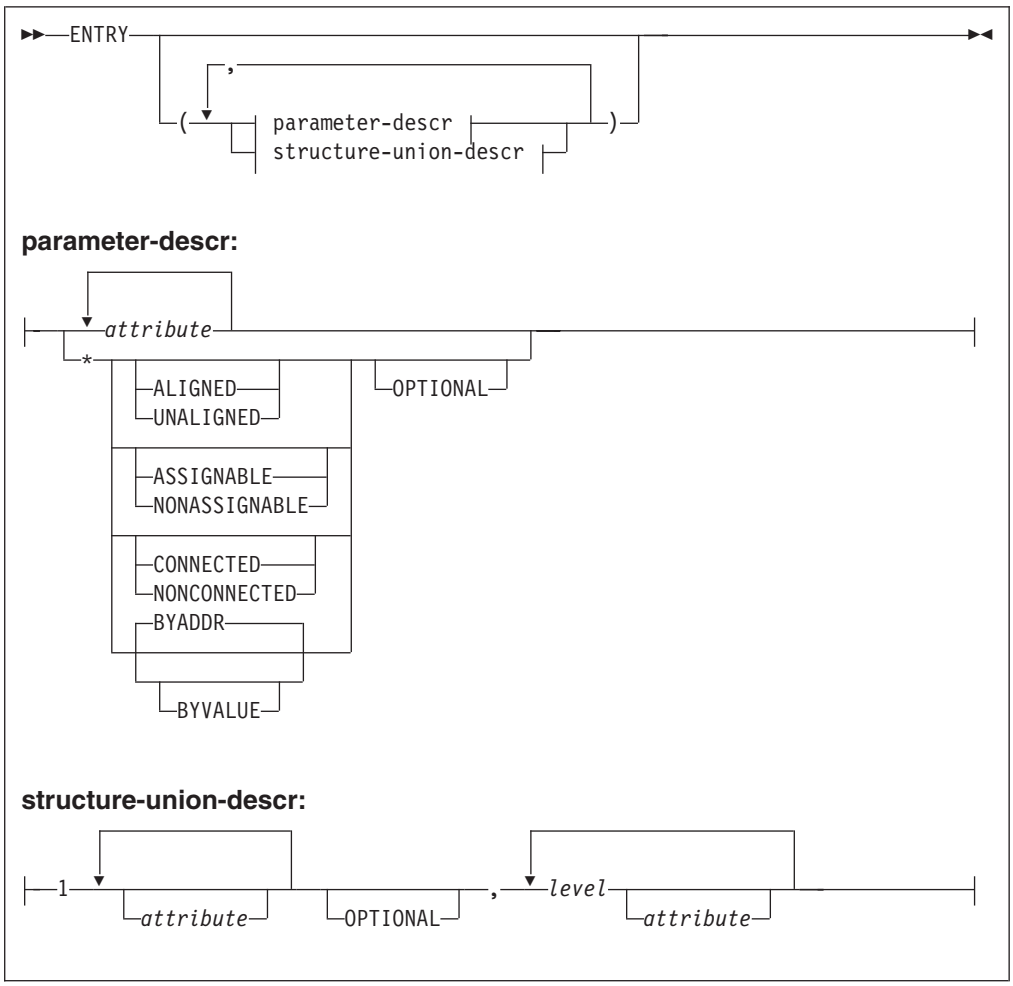
LIMITED 属性
129 ページの『LIMITED 属性』を参照してください。

RETURNS 属性
144 ページの『RETURNS オプションとその属性』を参照してください。

EXTERNAL 属性
163 ページの『宣言の有効範囲』を参照してください。

ENTRY 属性

ENTRY 属性は、宣言される名前が、外部入り口定数または入り口変数のいずれかであることを指定します。さらに、エントリー・ポイントのパラメーター属性も記述します。



ENTRY
パラメーター記述子リストを使用しない ENTRY 属性は、 RETURNS 属性によって暗黙指定されます。

parameter-descr (parameter-descriptor)

パラメーター記述子リストは、関連する外部入り口定数または入り口変数のパラメーターの属性を説明するために指定することができます。引数とパラメーター属性の突き合わせ、および仮引数の作成用に使用されます。

パラメーター記述子リストが指定されない場合には、引数属性のデフォルトとパラメーター属性を突き合わせなければなりません。そのため、引数属性とパラメーター属性を突き合わせない場合には、パラメーター記述子リストを指定しなければなりません。

それぞれのパラメーター記述子は、呼び出されたエントリー・ポイントの 1 つのパラメーターと対応し、(もし指定されていれば) そのパラメーターの属性が指定されます。

パラメーター記述子は、説明されているパラメーターと同じ順序で示さなければなりません。記述子がなければ、引数のデフォルトとパラメーターが突き合わされます。

パラメーターの記述子が不要な場合には、記述子がないということをアスタリスクで示さなければなりません。 以下に例を示します。

`entry(character(10),*,*,fixed dec)`	4 つの引数を示します。
`entry(*)`	1 つの引数を示します。
`entry( )`	入り口名は引数を持ってはならないということを指定します。
`entry`	任意の数の引数を持つことができるということを指定します。
`entry(float binary,*)`	2 つの引数を示します。

attribute

使用可能な属性は、 25 ページの『データ属性』にリストされた任意のデータ属性です。属性は、パラメーター記述子内に任意の順に表示することができます。配列パラメーター記述子の場合は、次元属性が最初に指定されなければなりません。

***** アスタリスクは、そのパラメーターにどのデータ・タイプでも使用できることを指定します。アスタリスク以降の有効な属性は、以下のとおりです。

- ALIGNED または UNALIGNED
- ASSIGNABLE または NONASSIGNABLE
- BYADDR または BYVALUE
- CONNECTED または NONCONNECTED
- OPTIONAL

変換は行われません。

OPTIONAL

この属性については、 124 ページの『OPTIONAL 属性』で説明します。

structure-union-descr (structure-union-descriptor)

構造体共用体記述子の場合は、記述子レベル番号がパラメーターのものと同じである必要はありませんが、構造化は同じでなければなりません。特定レベルの属性は、任意の順序で示すことができます。

アスタリスクが指定されている場合には、デフォルトは適用されません。たとえば、以下の宣言のデフォルトは、2 番目のパラメーターにだけしか適用されません。

```
dcl X entry(* optional, aligned); /* defaults applied for 2nd parm */
```

パラメーター記述子のエクステント (長さ、サイズ、および境界) は、定数またはアスタリスクとして指定されなければなりません。被制御パラメーターは、アスタリスクを持たなければなりません。

RETURNS 属性は、ENTRY 属性を暗黙指定します。 以下に例を示します。

パラメーター記述子の例

```
Test:  procedure (A,B,C,D,E,F);


 declare A fixed decimal (5),
         B float binary (21),
         C pointer,
         1 D,
           2 P,
           2 Q,
             3 R fixed decimal,
         1 E,
           2 X,
           2 Y,
             3 Z,
         F(4) character (10);
end Test;
```

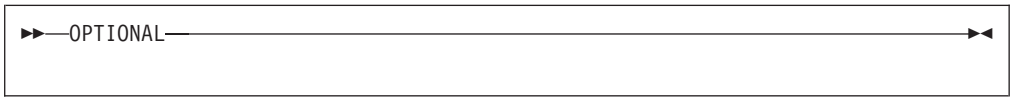
記述子の宣言の例

```
declare Test entry
            (decimal fixed (5),
            binary float (21),
           *,
            1,
             2,
             2,
              3 decimal fixed,
             *,
            (4) char(10));
```

前の例では、C パラメーターおよび E 構造体パラメーターは記述子を持ちません。

OPTIONAL 属性

OPTIONAL は、パラメーター記述子リストの一部として、またはパラメーター宣言の属性として指定することができます。



引数にアスタリスクを指定することによって、 OPTIONAL 引数を呼び出しおよび関数参照で省略することができます。引数リストのどの位置にある項目でも (最後の項目でも構わない) 省くことができます。ただし、省略される項目は引数として数えられます。入り口にその項目が含まれていても、引数の数は入り口で許可されている最大数を超えてはなりません。

項目が LIMITED ENTRY ではない限りは、同じ項目に OPTIONAL と BYVALUE は使用できません。

受け取りプロシージャーは、OMITTED 組み込み関数を使用して、 OPTIONAL パラメーター/引数が入り口の呼び出しで省略されたかどうかを判別することができます。 (OMITTED 組み込み関数についての詳細は、474 ページの『OMITTED』を参照してください。)

ENTRY 宣言の最終パラメーターが OPTIONAL として宣言された場合、その ENTRY はそれらのパラメーターを完全に省略して呼び出すことができます。該当する数のアスタリスクを指定することも不要です。そのため、例えば、ある ENTRY が 5 つのパラメーターを持っていて、そのうち最後の 2 つは OPTIONAL 属性を持っていると宣言された場合、3 つ、4 つ、または 5 つの引数を指定して呼び出すことができます。

呼び出された ENTRY が明示的に宣言されている場合にのみ、そのような末尾の OPTIONAL パラメーターを省略することができます。そのため、ネストされたサブプロシージャーを呼び出す場合には許可されません (明示的に宣言されることがないため)。

図 5 では、プロシージャー Vrtn に対して有効な CALL ステートメントと無効な CALL ステートメントの両方を示します。

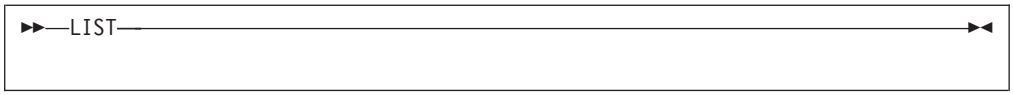
```
  Caller: proc;
    dcl Vrtn entry (
              fixed bin,
              ptr optional,
              float,
              * optional);

/*  The following calls are valid:  */

    call Vrtn(10, *, 15.5, 'abcd');
    call Vrtn(10, *, 15.5, *);
    call Vrtn(10, addr(x), 15.5, *);
    call Vrtn(10, *, 15.5);
    call Vrtn(10, addr(x), 15.5);

/*  The following calls are invalid:  */

    call Vrtn(*, addr(x));
    call Vrtn(10,addr(x));
    call Vrtn(10);
    call Vrtn;
  end Caller;

  Vrtn: proc (Fb, P, Fl, C1);
    dcl Fb fixed bin,
        P ptr optional,
        Fl float,
        C1 char(8) optional;

    if ¬omitted(C1) then display (C1);
    if ¬omitted(P) then P=P+10;
end;
```

図 5. 有効な呼び出しステートメントおよび無効な呼び出しステートメント

Vrtn は、OPTIONAL パラメーターが省略されたかどうかを判別して、適切な処置をとります。

LIST 属性

LIST は、パラメーター記述子リストの最後のパラメーターで指定できます。または、プロシージャーへの最後のパラメーターの属性として指定できます。



LIST 属性が入り口宣言で指定された場合、それは、ゼロまたはさらに追加の引数がそのエントリー・ポイントに渡される可能性があることを示します。たとえば、次の宣言では、 vararg が 1 文字の varyingz パラメーターを指定して呼び出されなければならないこと、および任意の数のほかのパラメーターを指定して呼び出されなければならないことを指定します。

```
dcl vararg  external
            entry( list byaddr char(*) varz nonasgn )
            options( nodescriptor byvalue );
```

LIST 属性がプロシージャーの最後のパラメーターの宣言中で指定される場合、それは、ゼロまたはさらに追加の引数がそのプロシージャーに渡された可能性があることを示します。

LIST 属性が指定された場合、記述子は許可されません。

それらの追加パラメーターの中の最初のパラメーターのアドレスは、 VARGLIST 組み込み関数を介して取得できます。以下のように、このアドレスは、追加パラメーターのアドレスを取得するのに使用されることがあります。

- このプロシージャーへの追加パラメーターが byvalue で渡された場合、この初期アドレスを VARGSIZE 組み込み関数が戻した値で連続的に増やして、追加パラメーターのアドレスを戻します。
- このプロシージャーへの追加パラメーターが byaddr で渡された場合、この初期アドレスをポインターのサイズで連続的に増やして、追加パラメーターのアドレスを戻します。

以下の (printf の簡単なバージョンを実現する) サンプル・プログラムは、 LIST 属性の使用法を示します。ルーチン varg1 は、byvalue パラメーターを持つ変数の引数リストを処理する方法を示します。ルーチン varg2 は、byaddr パラメーターを持つ変数の引数リストを処理する方法を示します。

```
*process rules(ans) dft(ans) gn;
*process langlvl(saa2);

 vararg: proc options(main);

   dcl i1      fixed bin(31) init(1729);
   dcl i2      fixed bin(31) init(6);
   dcl d1      float bin(53) init(17.29);

   call varg1( 'test byvalue' );
   call varg1( 'test1 parm1=%i', i1 );
   call varg1( 'test2 parm1=%i parm2=%i', i1, i2 );
   call varg1( 'test3 parm1=%d', d1 );

   call varg2( 'test byaddr' );
   call varg2( 'test1 parm1=%i', i1 );
   call varg2( 'test2 parm1=%i parm2=%i', i1, i2 );
   call varg2( 'test3 parm1=%d', d1 );
 end;

*process ;
  varg1:
    proc( text )
    options( nodescriptor byvalue );

    dcl text    list byaddr nonasgn varz char(*);

    dcl jx      fixed bin;
    dcl iz      fixed bin;
    dcl ltext   fixed bin;
    dcl ptext   pointer;
    dcl p       pointer;
    dcl i       fixed bin(31) based;
    dcl d       float bin(53) based;
    dcl q       float bin(64) based;
    dcl chars   char(32767) based;
    dcl ch      char(1) based;
```

図 6. LIST 属性を説明するサンプル・プログラム (1/3)

```
    ptext = addr(text);
    ltext = length(text);
    iz = index( substr(ptext->chars,1,ltext), '%' );
    p = varglist();
    do while( iz > 0 );
      if iz = 1 then;
      else
        put edit( substr(ptext->chars,1,iz-1) )(a);
      ptext += iz;
      ltext -= iz;
      select( ptext->ch );
        when( 'i' )
          do;
            put edit( trim(p->i) )(a);
            p += vargsize( p->i );
          end;
        when( 'd' )
          do;
            put edit( trim(p->d) )(a);
            p += vargsize( p->d );
          end;
      end;
      ptext += 1;
      ltext -= 1;
      if ltext <= 0 then leave;
      iz = index( substr(ptext->chars,1,ltext), '%' );
    end;
    if ltext = 0 then;
    else
      put edit( substr(ptext->chars,1,ltext) )(a);
    put skip;
  end;
```

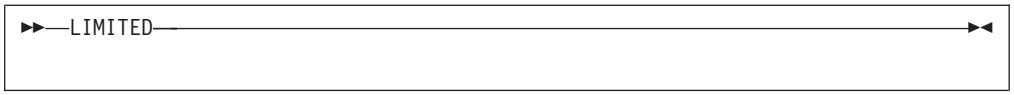
図 6. LIST 属性を説明するサンプル・プログラム (2/3)

```
*process ;
  varg2:
    proc( text )
    options( nodescriptor byaddr );

    dcl text    list byaddr nonasgn varz char(*);

    dcl jx      fixed bin;
    dcl iz      fixed bin;
    dcl ltext   fixed bin;
    dcl ptext   pointer;
    dcl p       pointer;
    dcl p2      pointer based;
    dcl i       fixed bin(31) based;
    dcl d       float bin(53) based;
    dcl q       float bin(64) based;
    dcl chars   char(32767) based;
    dcl ch      char(1) based;

    ptext = addr(text);
    ltext = length(text);
    iz = index( substr(ptext->chars,1,ltext), '%' );
    p = varglist();
    do while( iz > 0 );
      if iz = 1 then;
      else
        put edit( substr(ptext->chars,1,iz-1) )(a);
      ptext += iz;
      ltext -= iz;
      select( ptext->ch );
        when( 'i' )
          do;
            put edit( trim(p->p2->i) )(a);
            p += size( p );
          end;
        when( 'd' )
          do;
            put edit( trim(p->p2->d) )(a);
            p += size( p );
          end;
      end;
      ptext += 1;
      ltext -= 1;
      if ltext <= 0 then leave;
      iz = index( substr(ptext->chars,1,ltext), '%' );
    end;
    if ltext = 0 then;
    else
      put edit( substr(ptext->chars,1,ltext) )(a);
    put skip;
  end;
```

図 6. LIST 属性を説明するサンプル・プログラム (3/3)

LIMITED 属性

LIMITED 属性は、入り口変数がネストされていない入り口定数だけを値として持つことを表します。 LIMITED ではない入り口変数は、入り口定数を値として持つことができます。



例

```
Example: proc options(reorder reentrant);
dcl (Read, Write) entry;
dcl FuncRtn(2) entry limited
static init (Read, Write);


dcl (Prt1) entry;
dcl PrtRtn(2) entry variable limited
static init (Prt1,       /*  legal    */
             Prt2);      /*  illegal  */
Prt2: proc;
:
end Prt2;
end Example;
```

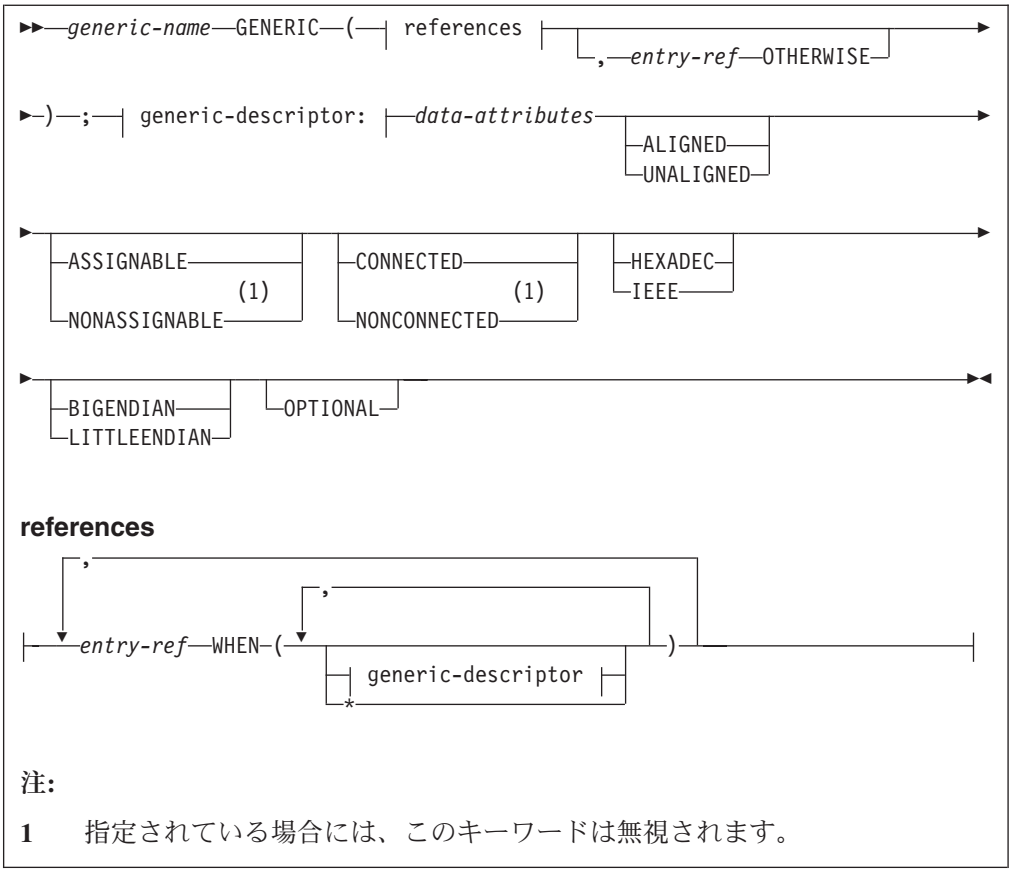
LIMITED 静的入り口変数は、ネストされていない入り口定数の値を使用して初期設定できます。こうすれば、より効率的なコードを生成できます。また、LIMITED 静的入り口変数は、LIMITED ではない入り口変数よりも小さいストレージを使用します。

総称入り口

総称入り口宣言で、入り口参照とそれらの記述子のセットに対して総称名を指定します。コンパイル時に、総称名の呼び出しがセット内の 1 つの入り口に置き換えられます。

GENERIC 属性

総称名は、GENERIC 属性を使用して明示的に宣言されなければなりません。



省略形: OTHERWISE の場合には OTHER

総称宣言の構文については、160 ページの『DECLARE ステートメント』を参照してください。

entry-ref
添え字を付けたり、定義したりする必要はありません。別の記述子のリストを利用して、同じ入り口参照を単一の GENERIC 宣言内で複数回使用することができます。

generic-descriptor
単一引数と対応します。対応する引数を持たなければならないので、関連する入り口参照を置換用に選択できる属性を指定します。

構造体または共用体を指定することはできません。

記述子が不要な場所では、不在の場所をアスタリスクで示さなければなりません。

呼び出されるステートメント内のすべての引数が不在であることを表す記述子は、入り口の WHEN 文節内で総称記述子を省略することによって表されます。フォーマットは次のとおりです。

```
generic (... entry1 when( ) ...)
```

data-attributes
24 ページの『データ・タイプとその属性』にリストされています。

ALIGNED および UNALIGNED
172 ページの『ALIGNED 属性と UNALIGNED 属性』に説明されています。

ASSIGNABLE および NONASSIGNABLE
260 ページの『ASSIGNABLE 属性と NONASSIGNABLE 属性』に説明されています。

CONNECTED および NONCONNECTED
263 ページの『CONNECTED 属性と NONCONNECTED 属性』に説明されています。

HEXADEC および IEEE
262 ページの『HEXADEC 属性と IEEE 属性』に説明されています。

BIGENDIAN および LITTLEENDIAN
261 ページの『BIGENDIAN 属性と LITTLEENDIAN 属性』に説明されています。

OPTIONAL
124 ページの『OPTIONAL 属性』に説明されています。

総称名の呼び出しに達すると、呼び出しに指定されている引数の数およびその属性は、セット内の各入り口の記述子リストと比較されます。数および属性の両方で引数と記述子リストが一致する最初の入り口参照が、総称名と置き換わります。

以下の例では、DECIMAL または FLOAT、および BINARY または FIXED という属性の付いた記述子をちょうど 2 つだけ持つ入り口参照が、探索されます。

```
declare Calc generic (
             Fxdcal when (fixed,fixed),
             Flocal when (float,float),
             Mixed  when (float,fixed),
             Error otherwise);
Dcl     X decimal float (6),
        Y binary fixed (15,0);


        Z = X+Calc(X,Y);
```

正確な属性のついた記述子の正確な数のついた入り口が検出されない場合には、(ある場合) OTHERWISE 文節のついた入り口が選択されます。前の例では、Mixed が選択されて置き換えられています。

同じような方法で、引数の次元数に基づいて、入り口を選択することができます。

```
dcl  D generic (D1 when ((*))),
                D2 when((*,*))),
     A(2),
     B(3,5);
call D(A);      /* D1 selected because A has one dimension  */
call D(B);      /* D2 selected because B has two dimensions */
```

すべての記述子が省略されるか、またはアスタリスクで構成される場合には、正しい数の記述子を持つ最初の入り口参照が選択されます。

総称値への参照で引数として使う入り口式は、ENTRY タイプの記述子とだけ一致します。そのような記述子がないときには、プログラムはエラー状態になります。

入り口呼び出しまたは入り口値

入り口値自体が使用されるか、または入り口呼び出しによって戻される値が使用されるか判然としない場合があります。以下の表および例を使用すると、以下の場合にどちらが起こったかを理解する際に役立ちます。

入り口参照が以下の場合 . . .	次のようになります . . .
組み込み関数の場合	呼び出されます
ヌルであっても、引数リストを持っている場合	呼び出されます
CALL ステートメントで参照される場合	呼び出されます
引数リストがなく、CALL ステートメントで参照されない場合	呼び出されません

以下の例では、A が呼び出され、 B(C) は入り口値として C を渡し、 D(C()) は C を呼び出します。

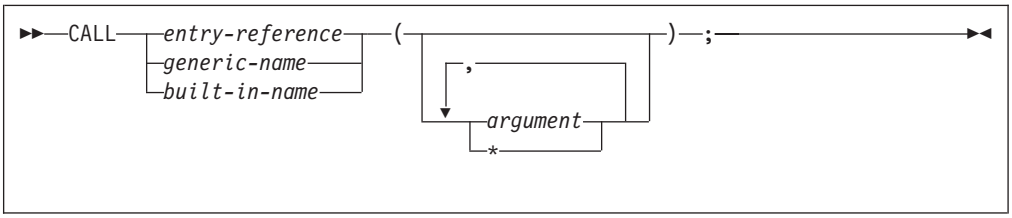
```
dcl ( A, B, C returns (fixed bin), D) entry;

call A;                       /* A is invoked                       */
call B(C);                    /* C is passed as an entry value      */
call D( C() );                /* C is invoked                       */
```

以下の例では、最初の割り当ては、整数への入り口定数を割り当てようとしたことを示しているために、無効になります。 2 番目の割り当ては有効です。

```
dcl A fixed bin,
    B entry returns ( fixed bin );

A = B;
A = B();
```

CALL ステートメント

CALL ステートメントはサブルーチンを呼び出します。



entry-reference
呼び出されるサブルーチンの名前が ENTRY 属性で宣言されることを指定します (120 ページの『入り口データ』で説明されています)。

generic-name
呼び出されるサブルーチンの名前が GENERIC 属性で宣言されることを指定します (130 ページの『総称入り口』で説明されています)。

built-in name
呼び出されるサブルーチンの名前が BUILTIN 属性で宣言されることを指定します (398 ページの『BUILTIN 属性』を参照)。

argument
呼び出されたサブルーチンに渡されるエレメント、要素式、または集合です。
116 ページの『プロシージャーへの引数の引き渡し』を参照してください。

CALL ステートメントの参照または式は、呼び出しが実行されるブロック内で計算されます。この中には、計算結果として入力される ON ユニットの実行も含まれます。

RETURN ステートメント

RETURN ステートメントは、RETURN ステートメントが入っているサブルーチンまたは関数プロシージャーの実行を終了し、呼び出しプロシージャーに制御を戻します。制御は、呼び出し参照の直後の場所に戻されます。

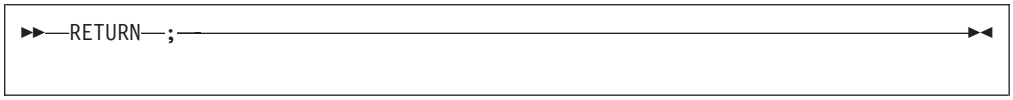
式を伴う RETURN ステートメントを OPTIONS(MAIN) によるプロシージャー内で使用することはできません。

式を伴わない RETURN ステートメントは、RETURNS オプションを指定したプロシージャー内では無効です。反対に、式を伴う RETURN ステートメントは、RETURNS オプションを指定していないプロシージャー内では無効です。

RETURNS オプションを指定したプロシージャーでは、少なくとも 1 つの RETURN ステートメント (もちろん式を伴うもの) が含まれていなければなりません。

サブルーチンからの戻り

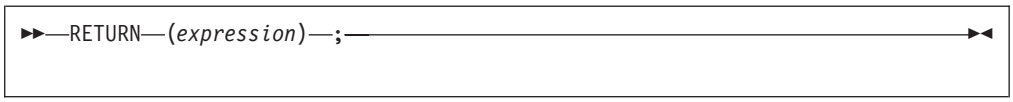
サブルーチンから戻るための、RETURN ステートメント構文は以下のとおりです。



RETURN ステートメントが主プロシージャーを終了すると、プログラムが終了する前に、FINISH 条件が起こります。

関数からの戻り

関数から戻るための RETURN ステートメント構文は以下のとおりです。



関数参照に戻される値は、指定された式の値であり、関数が入力された ENTRY ステートメントまたは PROCEDURE ステートメントの RETURNS オプションで指定されている属性と一致させるために変換されます。以下に例を示します。

```
F: procedure returns(fixed bin(15));
⋮
G: entry returns(fixed dec(7,2));
⋮
dcl D fixed bin(31);
```

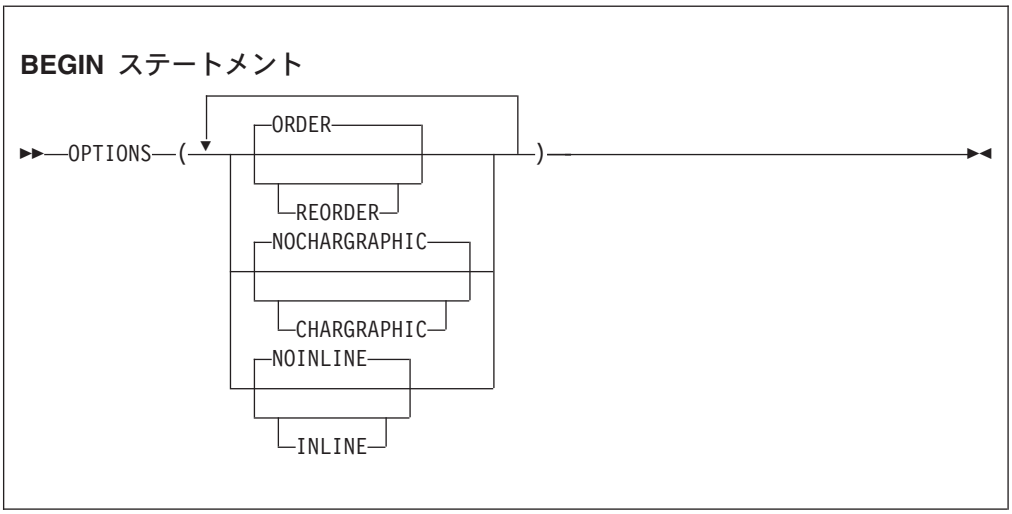
```
⋮
return (D);
```

この関数が F で入力されると、D は、プロシージャー F (FIXED BIN(15)) の RETURNS オプションで指定された属性に変換されます。しかし、この関数が G で入力されると、D は、エントリー G(FIXED DEC(7,2)) の RETURNS オプションで指定された属性に変換されます。

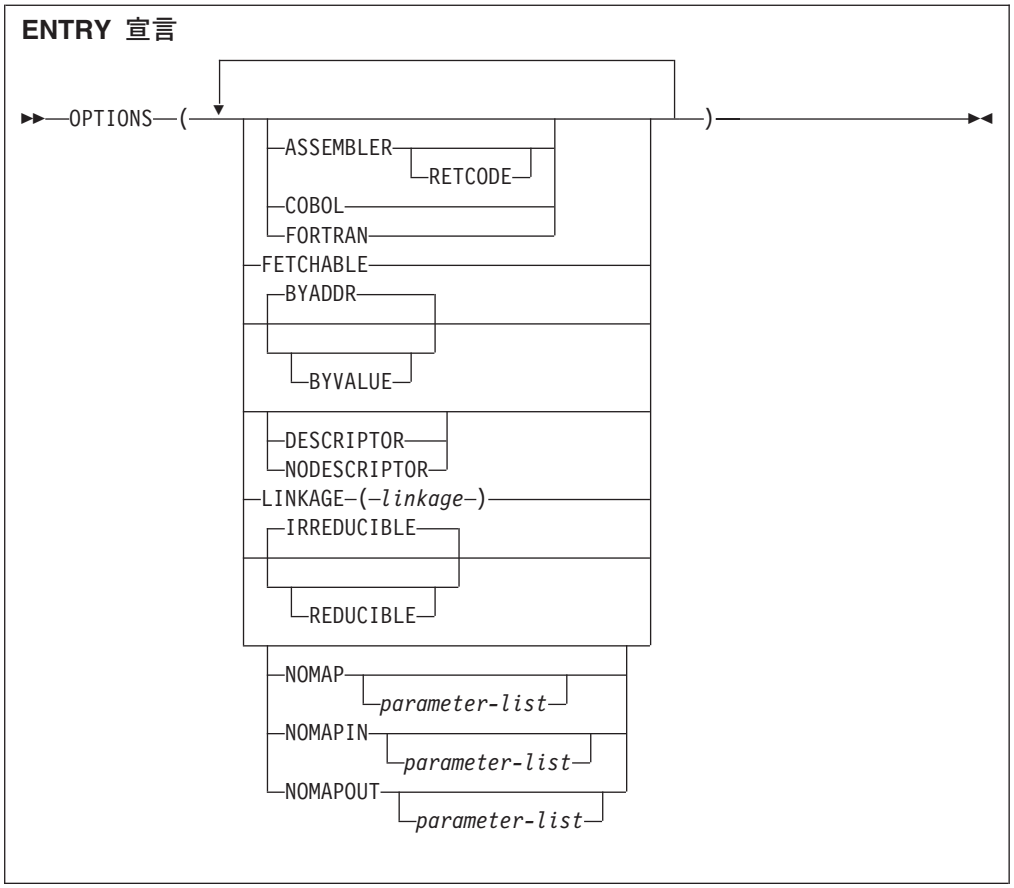
開始ブロック内の RETURN ステートメントに式を指定することはできません。

OPTIONS オプションとその属性

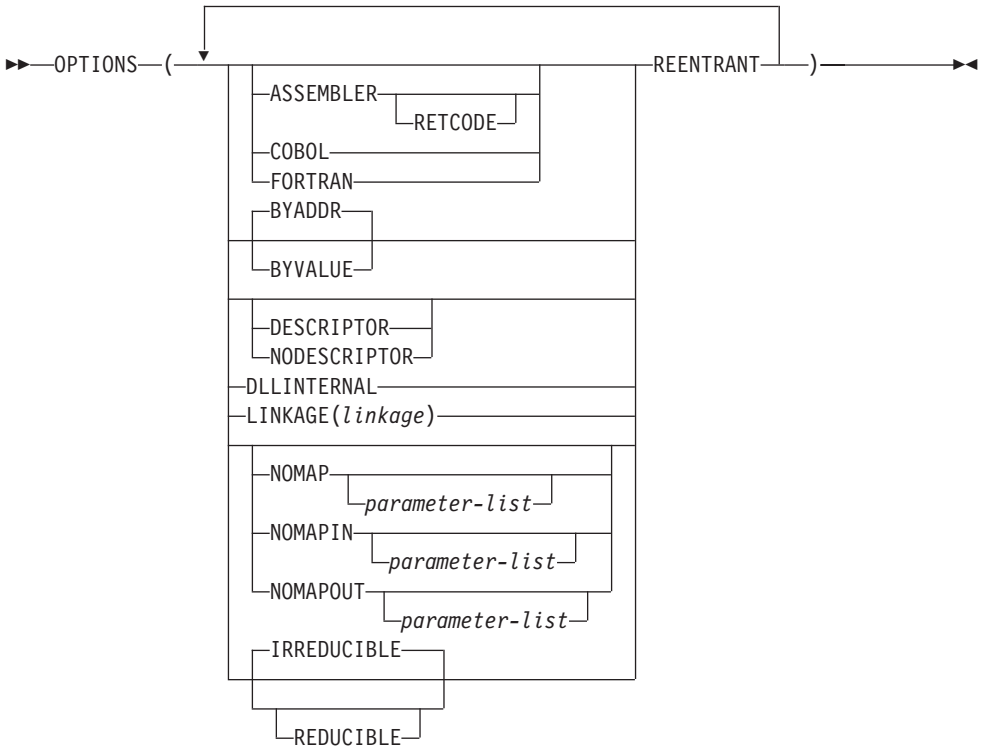
OPTIONS オプションは、ステートメント PACKAGE、PROCEDURE、 ENTRY および BEGIN に指定することができます。 OPTIONS 属性は、ENTRY 宣言で指定することができます。これは、ブロックやプロシージャーの呼び出しに適用する処理の特性を指定するために使用されます。以下の構文図に示されているオプションは、英字順にリストされており、138 ページから説明されています。



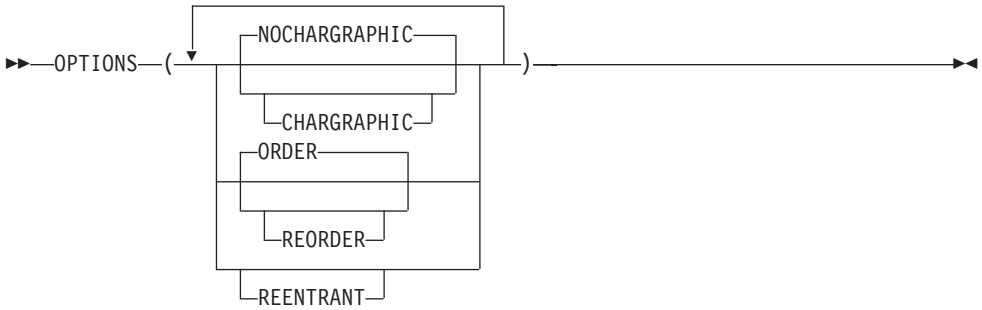
ENTRY 宣言

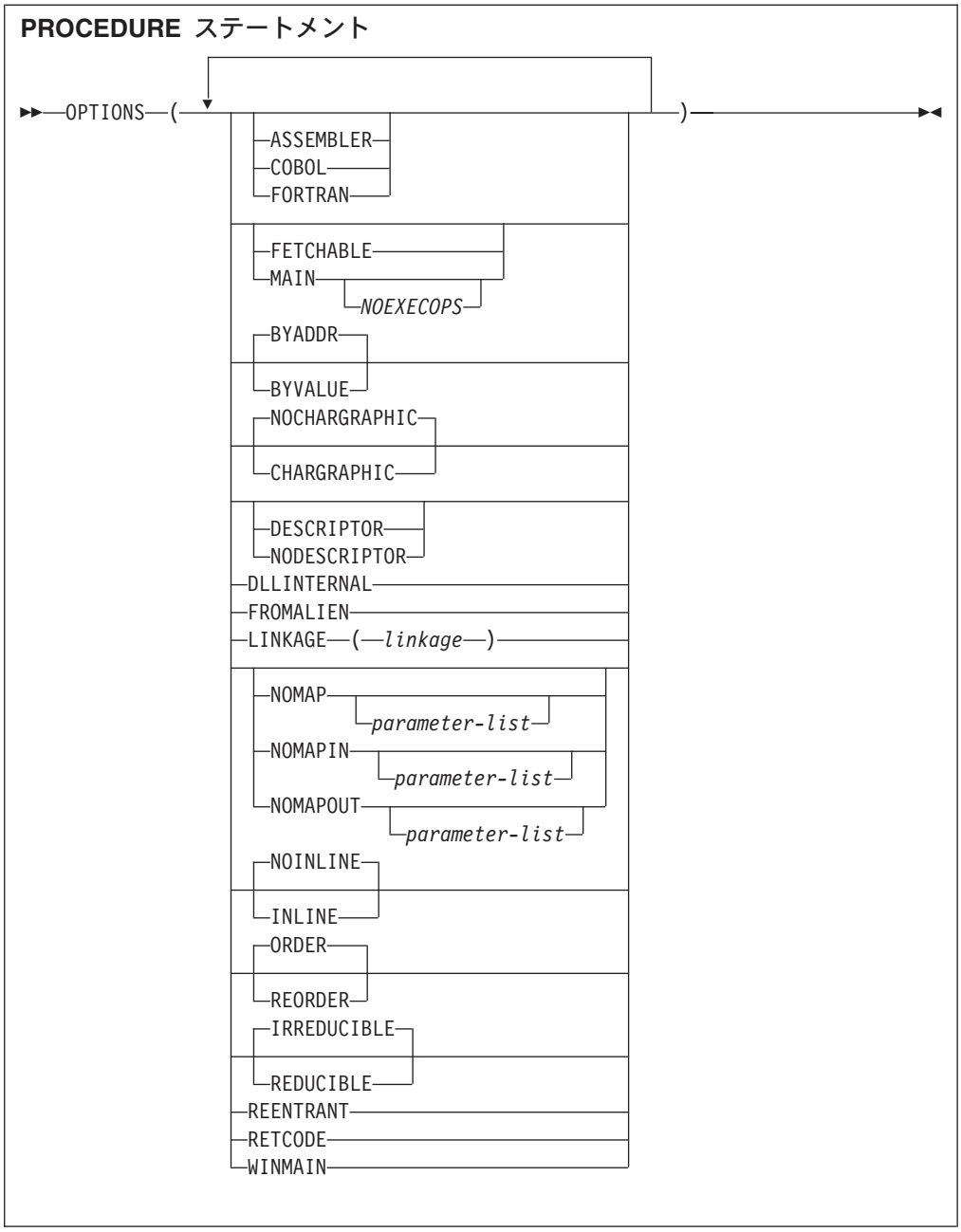

OPTIONS
(
ASSEMBLER
RETCODE
COBOL
FORTRAN
FETCHABLE
BYADDR
BYVALUE
DESCRIPTOR
NODESCRIPTOR
LINKAGE
(
linkage
)
IRREDUCIBLE
REDUCIBLE
NOMAP
parameter-list
NOMAPIN
parameter-list
NOMAPOUT
parameter-list
)

ENTRY ステートメント



PACKAGE ステートメント





オプションは、ブランクまたはコンマで区切られます。これらは、任意の順番に示すことができます。

ASSEMBLER

省略形: ASM

このオプションは NODESCRIPTOR と同じ効果があります。

プロシージャーに ASSEMBLER オプションがある場合、そのプロシージャーの終了によって、 PLIRETV() 値が、そのプロシージャーの戻り値として使用されます。

詳細については、「*プログラミング・ガイド*」を参照してください。

BYADDR または BYVALUE

引数とパラメーターがどのように受け渡しされるのかを指定します。 BYADDR はデフォルトです。

BYVALUE は、既知の長さとサイズを持つスカラー引数およびスカラー・パラメーターに対してのみ指定できます。

BYVALUE および BYADDR 属性は、入り口宣言の記述子リスト、およびパラメーター宣言の属性リスト内にも指定することができます。入り口またはパラメーター宣言に BYVALUE または BYADDR を指定すると、OPTIONS ステートメントに指定されたオプションが無効になります。

以下の例では、入り口宣言と OPTIONS オプションの両方に BYVALUE および BYADDR を表しています。例では、コンパイラー・オプション DEFAULT (BYADDR) が有効であると想定しています。

例 1

```
MAINPR: proc options(main);


  dcl D entry (fixed bin byaddr,
               ptr,
               char(4) byvalue)         /*  byvalue not needed      */
        options(byvalue);
  dcl E2 entry;                          /* default(byaddr) in effect */
  dcl Length fixed bin,
      P      pointer,
      Name   char(4);


  call D(Length, P, Name);              /* Length is passed byaddr */
                                        /* P is passed by value */
                                        /* Name is passed by value */


  call E2(P);                           /* P is passed by address */


D: proc(I, Q, C)
   options(byvalue);
  dcl I fixed bin byaddr,
      Q ptr,
      C char(4) byvalue;


E2: proc(Q);
   dcl Q ptr;
```

例 2

```
dcl F entry (fixed bin byaddr,        /*  byaddr not needed       */
             ptr,
             char(4) byvalue)
      options(byaddr);
dcl E3 entry;
dcl E4 entry (fixed bin byvalue);


call F(Length, P, Name);             /* Length is passed byaddr */
                                     /* P is passed byaddr      */
                                     /* Name is passed by value */
```

```
call E3(Name);                        /* Name is passed byaddr   */
call E4(Length);                      /* Length is passed by value */


F: proc(I,P,C) options(byaddr);
  dcl I fixed bin byaddr;              /*  byaddr not needed       */
  dcl P ptr byaddr;                    /*  byaddr not needed       */
  dcl C char(4) byvalue;               /*  byvalue needed          */


E3: proc(L);
   dcl L char(4);


E4: proc(N);
   dcl N fixed bin byvalue;
```

CHARGRAPHIC または NOCHARGRAPHIC

省略形: CHARG、NOCHARG

外部プロシージャーの省略形は、NOCHARG です。内部プロシージャーと開始ブロックは、それらが含まれているプロシージャーからデフォルトを継承します。

CHARG が有効な場合には、以下のようなセマンティックの変更が起こります。

- すべての文字ストリング割り当てでは、文字の割り当てを混合すると考えられます。
- STRINGSIZE 条件によって、MPSTR 組み込み関数が使用されます。STRINGSIZE は、切り捨てを起こすことができ、かつインテリジェントな DBCS の切り捨てが必要とされる文字の割り当てが生じた場合に使用可能にならなければなりません。 (MPSTR BUILTIN については、 471 ページの『MPSTR』を参照してください。) 以下に例を示します。

```
Name: procedure options(chargraphic);
      dcl A char(5);
      dcl B char(8);

/* the following statement...                              */

      (stringsize): A=B;

/*...is logically transformed into...                      */

      A=mpstr(B,'vs',length(A));
```

NOCHARG が有効な場合は、セマンティックの変更は起こりません。

COBOL

このオプションは NODESCRIPTOR (下記を参照) と同じ効果を持ち、さらに OPTION(COBOL) の効果を加えたものです。

- 入力宣言またはプロシージャー・ステートメントで他のリンケージが指定されていない限り、LINKAGE(SYSTEM) を暗黙指定します。
- NOMAP、NOMAPIN、または NOMAPOUT オプションの使用を許可します。
- プロシージャー・ステートメントで指定されている場合、そのプロシージャーの終了によって、PLIRETV() 値が、そのプロシージャーの戻り値として使用されることを暗黙指定します。

DESCRIPTOR または NODESCRIPTOR

入り口宣言またはプロシージャー・ステートメントで指定したプロシージャーが呼び出されたときに、記述子リストを渡すかどうかを示します。

DESCRIPTOR が示される場合、必要に応じてコンパイラーは記述子を渡します。

NODESCRIPTOR が示される場合には、コンパイラーは記述子を渡しません。

上記のどちらも表示されない場合には、呼び出されたプロシージャーのいずれかのパラメーターがストリング、配列、区域、構造体、または共用体であるときだけ、DESCRIPTOR が想定されます。

いずれかのパラメーターまたはエレメントが、以下の条件に当てはまっている場合、プロシージャー・ステートメントまたは入り口宣言に NODESCRIPTOR があると、エラーになります。

- 範囲、長さまたはサイズを表すのにアスタリスク (*) を使用している場合。
- いずれかのパラメーターが NONCONNECTED の場合。

ただし、範囲が指定されていないパラメーターが NONASSIGNABLE VARYING または VARYINGZ ストリングの場合は、NODESCRIPTOR を利用できます。

DLLINTERNAL

このオプションは、プロシージャーまたは項目が DLL の内部にあることが意図されており、結果として、その名前はコンパイラーによって生成された定義サイド・ファイルに含まれるべきでないことを示します。

DLLINTERNAL 属性は、EXTERNAL プロシージャーまたは ENTRY でのみ有効です。

FETCHABLE

このオプションは、必要に応じて、プロシージャーを呼び出す前に動的に取り出すかどうかを示します。

FETCHABLE 属性は、INTERNAL プロシージャーでは無効です。

FETCHABLE プロシージャーは、MAIN プロシージャーが含まれるロード・モジュールにはリンクしないでください。

FORTRAN

このオプションを使用すると、文字変数以外の記述子は渡されなくなります。

FROMALIEN

このオプションは、プロシージャーを非 PL/I ルーチンから呼び出すことができるかどうかを示します。 FROMALIEN は任意のプロシージャーで指定できます。ただし、その場合は不必要なオーバーヘッドが生じることになります。

INLINE または NOINLINE

INLINE と NOINLINE のオプションは最適化オプションであり、パッケージ内の開始ブロックおよびネストされないレベル 1 のプロシージャー用に指定できます。

INLINE は、開始ブロックまたはプロシージャーがパッケージ内で呼び出されたとき、その開始ブロックまたはプロシージャーのコードが呼び出し点でインライ

ン で実行されることを示します。 INLINE が指定されていても、コンパイラーは、開始ブロックまたはプロシージャーをインラインとしないように選択することができます。

NOINLINE は、開始ブロックまたはプロシージャーがいかなる場合にもインラインで実行されないように指示します。

OPTIONS(INLINE) を使用すれば、適切に構造化された読みやすいコードを書くのが容易になります。たとえば、プログラムをプロシージャーのセットへの一連の呼び出しとして作成した場合、OPTIONS(INLINE) を使えば、これらのプロシージャーを 1 つ 1 つ呼び出す必要がなくなり、オーバーヘッドが少なくなります。

プロシージャーまたは開始ブロックをインラインで実行すると、ONLOC などの組み込み関数によって戻される値は、その関数がインライン化されているプロシージャーの名前を戻します。同様に、トレースバックされた情報には、呼び出されたプロシージャーは含まれません。

インライン化できないプロシージャーと開始ブロックがあります。以下のものを含めますが、これらに限定することはありません。

- 条件の割り込み可能化が可変であるパッケージ内のプロシージャーと開始ブロック
- ON ステートメントまたは REVERT ステートメントを含むプロシージャーと開始ブロック
- データ・ディレクティブの入出力ステートメントを含むプロシージャーと開始ブロック
- ENTRY、FORMAT、または LABEL 定数の割り当てまたは比較を含むプロシージャーと開始ブロック

INLINE オプションを指定したネストされないプロシージャーが外部ではなく参照もされない場合、このプロシージャーのためのコードは作成されません。
INLINE も NOINLINE も指定されていないプロシージャーについては、DEFAULT コンパイラー・オプションがオプションをセットします。

INLINE と NOINLINE の詳しい使用法については、「プログラミング・ガイド」を参照してください。

LINKAGE
このオプションは、使用する呼び出し規約を指定し、PROCEDURE ステートメントおよび ENTRY 宣言で指定されます。

CDECL (INTEL のみ)
このオプションは、32 ビット C コンパイラーが使用する CDECL リンケージ規約を指定します。

OPTLINK
このオプションは最高速リンケージ規約であり、デフォルト設定されています。これは、ほとんどのコンパイラーの場合は標準のリンケージではありません。

STDCALL (Windows のみ)
このオプションは、STDCALL (すべての Windows API が使用する標準のリンケージ規約) を指定します。

SYSTEM
このオプションは、オペレーティング・システムを呼び出すときに使用する呼び出し規則を指定します。このオプションは OPTLINK よりも低速ですが、すべての MVS および AIX アプリケーションの標準オプションです。

呼び出し規則の詳細については、「プログラミング・ガイド」を参照してください。

MAIN
このオプションは、この外部プロシージャーが PL/I プログラムの初期プロシージャーであることを示します。 1 プログラムにつき 1 つの外部プロシージャーにだけ、 MAIN が有効であり、必要とされます。オペレーティング・システムの制御プログラムは、そのプログラムの実行の最初のステップとしてこのプロシージャーを呼び出します。

OPTIONS(MAIN) を指定した、複数のプロシージャーを含んでいる PL/I プログラムは、予測できない結果を生じることがあります。

NOEXECOPS
NOEXECOPS オプションは、MAIN オプションを使用する場合だけ有効です。このオプションは、ランタイム・オプションが、プログラムを呼び出すコマンドやステートメントでは指定できないこと指定します。主プロシージャーのパラメーターだけが指定されます。

NOMAP、NOMAPIN、NOMAPOUT
これらのオプションは、COBOL または FORTRAN と PL/I とのインターフェースでデータ集合体の自動操作を防ぎます。

各オプション引数リストは、適用するオプションのパラメーターを指定します。パラメーターは、どのような順番でも表示でき、コンマまたはブランクによって分離することができます。オプション用の引数リストがない場合、項目名のすべてのパラメーターはデフォルト・リストになります。

NOMAP、NOMAPIN、または NOMAPOUT は、同一の OPTION 指定にすべて表示することができます。この指定は、複数の指定またはデフォルト引数リストの同一パラメーターに含まれてはなりません。

これらのオプションは受け入れられますが、COBOL オプションが適用されない限りは無視されます。

ORDER または REORDER
ORDER と REORDER は、プロシージャーまたは開始ブロックに指定される最適化オプションです。

ORDER は、ブロック内で修正された変数の最も新しく割り当てられた値だけが入力される ON ユニットで利用できることを示します。これは、ステートメントの実行時およびブロック内の式で計算条件が生じたためです。

REORDER オプションを指定すると、コンパイラーが最適化コードを生成し、エラー・フリーの実行があった場合に、ソース・プログラムに指定された結果を生成することができます。

ORDER と REORDER オプションの詳細については、「*プログラミング・ガイド*」を参照してください。

上記のオプションが両方とも外部プロシージャーに指定されない場合には、デフォルトとして DEFAULT コンパイラー・オプションがセットされます。内部ブロックは、含まれているブロックから ORDER または REORDER を継承します。

REDUCIBLE または IRREDUCIBLE

省略形: RED、IRRED

REDUCIBLE は、引数 (1 つまたは複数) が変更されない限り、プロシージャーまたは入り口を複数回呼び出す必要がないこと、およびプロシージャーの呼び出しに副次作用がないことを示します。

たとえば、変更されないデータに基づいて結果を計算するユーザー作成の関数には、REDUCIBLE が宣言されなければなりません。乱数や時刻などの、変更されるデータに基づいて結果を計算する関数は、IRREDUCIBLE として宣言する必要があります。

REENTRANT

このオプションは無視されます。Intel および AIX プラットフォームではすべての PL/I プログラムが再入可能です。 z/OS プラットフォームでは、RENT コンパイラー・オプションによって、コンパイルされたすべてのプログラムが再入可能であり、静的変数 (NOWRITABLE コンパイラー・オプションの利用が必要になる場合もあります。) を変更しないものであれば、その他のプログラムも再入可能です。

RETCODE

このオプションを指定すると、ENTRY 点に ASM または COBOL オプションも指定されている場合、ENTRY は 呼び出された後、PL/I 戻りコードとして保管される値を戻します。基本的に、このような ENTRY が呼び出された場合、その戻り値は PLIRETC サブルーチンに渡されます。

WINMAIN (Windows のみ)

このオプションは、自動的に LINKAGE(STDCALL) および EXT('WinMain') を暗黙指定します。関連ルーチンには、以下の 4 つのパラメーターを含める必要があります。

1. インスタンス操作点
2. 直前の操作点
3. コマンド行へのポインター
4. ShowWindow に渡される整数

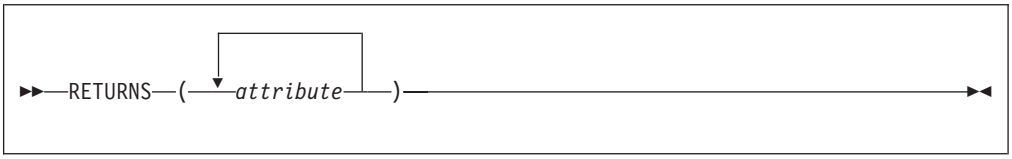
これらは、C WinMain が予期する 4 つのパラメーターと同じであり、このルーチンからの呼び出しは、C ルーチンが予期するものと同じです。

RETURNS オプションとその属性

プロシージャーが関数プロシージャーである場合には、プロシージャー・ステートメントに RETURNS オプションを指定しなければなりません。さらに、呼び出しプロシージャーまたはパッケージで、 RETURNS 属性のついた入り口としてプロシージャーを宣言しなければなりません。 RETURNS オプションおよび RETURNS 属

性は、戻される値の属性を指定するために使用されます。 RETURNS オプションの属性は、RETURNS 属性の属性と一致しなければなりません。

サブルーチンである (そのため、CALL ステートメントを使用して呼び出された) プロシージャーは、プロシージャー・ステートメントに RETURNS オプションを持ってはならず、その入り口宣言は RETURNS 属性を持ってはなりません。



複数の属性を指定する場合は、属性をブランクで区切る必要があります (括弧で囲まれる精度などの属性以外の場合)。

属性は、それが宣言ステートメント内にあるのと同じ方法で指定されます。デフォルトは通常の方法で適用されます。

指定される属性は、(29 ページの表 8 に示されている) スカラー変数のデータ属性や位置合わせ属性になります。 ENTRY 変数は、LIMITED 属性を持たなければなりません。また、TYPE 属性を指定して、ユーザー定義タイプ、序数、およびタイプ付き構造体および共用体を指定することができます。

ストリングの長さおよび区域サイズは、定数で指定されなければなりません。戻り値は、指定される長さまたはサイズを持ちます。

また、BYADDR と BYVALUE も RETURNS の属性のリストに指定することができます。プロシージャーにいずれかの ENTRY ステートメントが含まれており、なおかつ、そのプロシージャーまたは 2 次エントリー・ポイントのどれかが、次のいずれかの値を戻す場合、BYADDR 属性が有効でなくてはなりません。

- 値なし、または
- 集合値

z/OS では、BYADDR が RETURNS のデフォルトです。 C 関数を呼び出す場合、RETURNS の属性リスト内に BYVALUE が指定されていなければなりません。これらの属性については、116 ページの『BYVALUE および BYADDR の使用』を参照してください。

第 7 章 タイプ定義

ユーザー定義のタイプ (別名) 147
DEFINE ALIAS ステートメント 147
序数の定義 148
DEFINE ORDINAL ステートメント 148
タイプ付き構造体と共用体の定義. 150
HANDLE 属性 151
タイプ変数の宣言 152
TYPE 属性 152
ORDINAL 属性. 153
タイプ付き構造体修飾 153
'.' 演算子の使用 154
配列およびタイプ付き構造体または共用体の組み合わせ. 154
ハンドルの使用. 155
序数の使用 155
例 156
タイプ付き関数. 158

プログラミング言語では、*タイプ* は値のセットの記述とこれらの値に対して実行できる操作のセットです。 PL/I には多くの組み込みデータ・タイプがあります。それぞれのタイプは、多くのエレメントの属性を指定できます。 23 ページの『第 3 章 データ・エレメント』に、これらの組み込みデータ・タイプが説明されています。

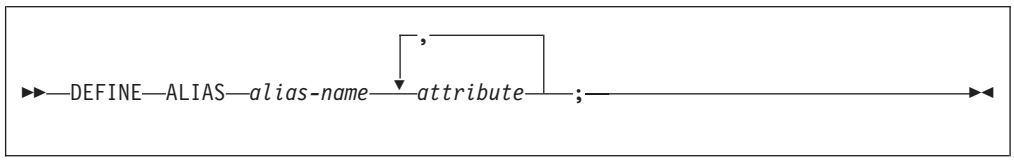
PL/I では、組み込みデータ・タイプを使用して、ユーザー独自のタイプを定義することができます。この章では、ユーザー定義のタイプ (別名、序数、構造体、および共用体)、これらのタイプ、ハンドル、およびタイプ付き関数をとる変数の宣言について説明します。

ユーザー定義のタイプ (別名)

別名 は、明示のデータ・タイプが使用できる所では必ず使用することができるタイプ名です。 DEFINE ALIAS ステートメントを使用して、データ属性の収集について別名を定義することができます。この方法で、データ・タイプに意味のある名前を割り当てて、プログラムの理解度を高めることができます。別名を定義することによって、データ属性のセットについての短い注記を提供し、タイプミスを減らすこともできます。

DEFINE ALIAS ステートメント

DEFINE ALIAS ステートメントは、ユーザーが別名に割り当てたデータ・タイプ属性のセットのために、同義語として使用することができる名前を指定します。



alias-name
指定された属性によって定義された明示のデータ・タイプが許可される場合はいつでも使用することができる名前を指定します。

attributes
指定できる属性は、関数が戻すことのできる変数についてのすべての属性 (たとえば、 RETURNS オプションと属性に有効なそれらの属性) です。これらの有効な属性は、 29 ページの表 8 にリストされています。したがって、ユーザーは

配列または構造体をとる属性リストについて別名を指定することはできません。ただし、DEFINE ORDINAL ステートメント、または DEFINE STRUCTURE ステートメント (『DEFINE ORDINAL ステートメント』および 150 ページの『タイプ付き構造体と共用体の定義』を参照)、または他の DEFINE ALIAS ステートメントに定義されたタイプについては、別名を定義することができます。また、RETURNS オプションと属性内の場合と同様に、任意のストリングの長さと領域のサイズは制限付き式でなければなりません。

欠落しているデータ属性は PL/I デフォルトを使用して提供されます。

例

```
define alias Name   char(31) varying;
define alias Salary fixed dec(7);     /* real by default       */
define alias Zip    char(5)           /* nonvarying by default */
```

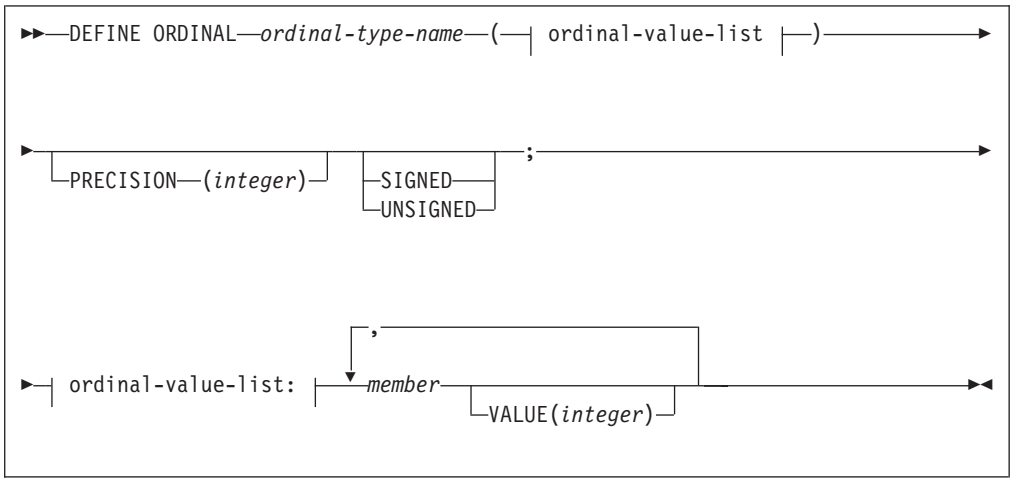
DECLARE ステートメントで Name が使用される場合は常に、 char(31) varying という属性をとります。

序数の定義

序数は順序付けられた値の (名前付き) セットです。 DEFINE ORDINAL ステートメントを使用すれば、序数を定義して、そのセットを参照するための有意義な名前を割り当てることができます。たとえば、「カラー」と呼ばれる序数を定義することができます。序数「カラー」は、「赤」、「黄色」、「青」などのメンバーを持ちます。その後「カラー」セットのメンバーは、関連した固定 2 進値の代わりに、コードをより自己記述しているそれらの名前で参照することができます。また、序数タイプとして宣言された変数は、同じ序数タイプ (またはそのタイプのメンバー) に対してのみ割り当てたり比較することができます。この自動チェック機能により、プログラムの信頼性が向上します。

DEFINE ORDINAL ステートメント

DEFINE ORDINAL ステートメントは、順序付けられた (名前付きの) 値のセットを表すタイプを指定します。



ordinal-type-name

Ordinal-type-name は、序数値のセット名を指定します。この名前は、ORDINAL

属性を持つ DECLARE ステートメントにおいてのみ使用できます。この名前を別の場所で使用すると、ほかの任意の非序数名として処理されます。

member
セット内のメンバーの名前を指定します。

VALUE
VALUE 属性は、セット内の特定のメンバーの値を指定します。最初のメンバーの場合、VALUE 属性を省略すると、ゼロの値が使用されます。ほかのメンバーの場合、VALUE 属性を省略すると、次に大きい整数値が使用されます。

VALUE 属性の値は整数でなければなりません。この整数値は正負の符号を持つことができますが、確実に増加していかなければなりません。指定された (または想定された) VALUE 属性の値を、XN 定数として指定することもできます。

PRECISION
省略形: PREC

PRECISION 属性は、特定の序数値の精度を指定します。この属性を省略すると、序数値の範囲が精度を決定することになります。

最大精度は、FIXED BINARY として宣言されたデータ項目の精度と同じです。

SIGNED または UNSIGNED
これらの属性は、序数値が負の値を取ることができることを示しています。これらを指定しないと、負の値を取るかどうかは序数値の範囲によって決定されることになります。たとえば、負の値が存在している場合は、SIGNED 属性が適用されます。

SIGNED と UNSIGNED の詳細については、32 ページの『SIGNED 属性と UNSIGNED 属性』を参照してください。

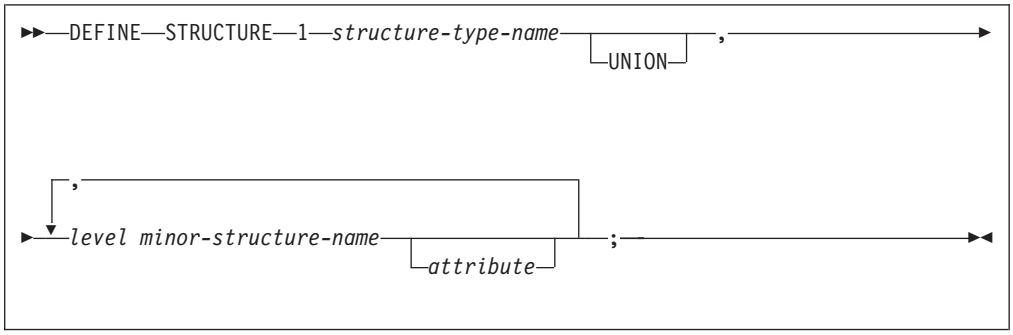
下記の例では、Red の値は 0、Orange の値は 1 であり、以下同様に続きます。しかし、Low の値は 2 であり、 Medium の値は 3 です。

例

```
define ordinal Color ( Red,          /* is 0, since VALUE is omitted */
                       Orange,
                       Yellow,
                       Green,
                       Blue,
                       Indigo,
                       Violet );

define ordinal Intensity ( Low    value(2),
                           Medium,
                           High   value(5));
```

タイプ付き構造体と共用体の定義

DEFINE STRUCTURE ステートメントは名前付き構造体または共用体タイプを指定します。



structure-type-name
この構造体タイプに与えられた名前を指定します (大構造体についての詳細な情報は、 185 ページの『構造体』を参照してください)。この名前は次元を持つことはできませんが、副構造体を持つことはできます。

UNION
187 ページの『UNION 属性』で説明されています。

minor-structure-name
下のレベルに与えられた名前を指定します。 (小構造体についての詳細な情報は 185 ページの『構造体』を参照してください。)

attributes
小構造体名についての属性を指定します。データ属性のみが指定できます。

DEFINE STRUCTURE ステートメントに指定された任意のストリングの長さ、領域サイズ、または配列次元が制限された式でなければなりません。

欠落しているデータ属性は PL/I デフォルトを使用して提供されます。

以下の制約事項に留意してください。

- 定義済み構造体は、その構造体の位置合わせの倍数となるバイト数でなければなりません。
- 定義済み構造体では、最も厳しい位置合わせを持つエレメントの前にあるバイト数が、そのエレメントの位置合わせの倍数でなければなりません。

たとえば、構造体に、最も厳しく位置合わせされた項目である、位置合わせされた fixed bin(31) フィールドが含まれている場合、その制約事項は以下のようになります。

- 構造体は、4 バイトの倍数でなければならない
- 最初の位置合わせされた fixed bin(31) フィールドの前には、4 バイトの倍数のバイト数がなければならない

DEFINE STRUCTURE ステートメントは「強い」タイプを定義します。言いかえると、そのタイプで宣言された変数は同じタイプを持つ変数 (またはパラメーター) に

のみ割り当てることができます。タイプ付き構造体をデータ・ディレクティブ入出力ステートメントに使用することはできません。

定義する構造体名だけを指定し、メンバーを指定しない DEFINE STRUCTURE ステートメントは、「指定されていない構造体」を定義します。

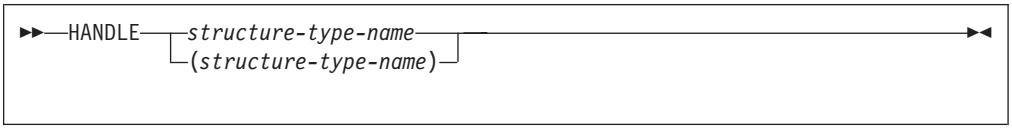
- 指定されていない構造体は参照解除できませんが、HANDLE を宣言するために使用できます (HANDLE はどちらにしても参照解除できません)。
- 指定されていない構造体は、メンバーを指定する後続の DEFINE STRUCTURE ステートメントの対象にすることもできます。

指定されていない構造体の定義は、構造体の定義に 2 つ目の構造体へのハンドルがあって、2 つ目の構造体にも 1 つ目の構造体へのハンドルがある場合に役立ちます。たとえば次の例では、親構造体に子構造体へのハンドルがあり、一方で子構造体にも親構造体へのハンドルがあります。

```
define structure 1 child;

define structure
 1 parent,
   2 first_child   handle child,
   2 parent_data   fixed bin(31);

define structure
 1 child,
   2 parent        handle parent,
   2 next_child    handle child,
   2 child_data    fixed bin(31);
```

HANDLE 属性

HANDLE 属性を使用して、変数を構造体タイプへのポインターとして宣言することができます。そのような変数はハンドル と呼ばれます。



structure-type-name
このハンドルがポイントするタイプ付き構造体を指定します。

定義された構造体と同様に、ハンドルはタイプが強く付きます。ハンドルは同じ構造体タイプのハンドルについてのみ、割り当てられ、比較することができます。ハンドルについての算術演算はできません。

ADDR 組み込み関数を使用して、タイプ付き構造体のアドレスをハンドルに割り当てることはできません。 ADDR 組み込み関数はポインターを戻し、そのポインターをハンドルに割り当てることができないからです。ただし、HANDLE 組み込み関数は、その引数としてタイプ付き構造体を取り、そのタイプにハンドルを戻します。以下の例では、152 ページで定義された tm 構造体タイプを使用しており、tm タイプを見付けるハンドルが宣言され、Daterec のアドレスがそのハンドルに割り当てられます。

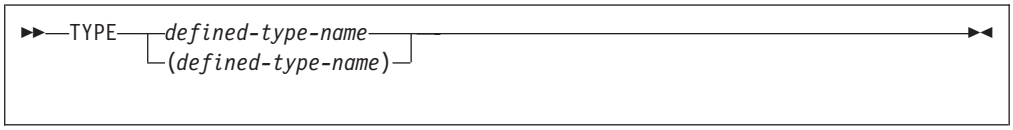
```
dcl P_Daterec handle tm;
dcl Daterec type tm;

P_Daterec = handle(Daterec);
```

ハンドルは POINTERVALUE 組み込み関数を使用して、ポインターに変換することができます。

タイプ変数の宣言

変数は、TYPE 属性を使用して、 DEFINE ALIAS、DEFINE STRUCTURE、または DEFINE ORDINAL ステートメントに指定されたタイプで宣言することができます。

TYPE 属性



defined-type-name
直前の定義済み別名、定義済み構造体、または序数タイプの名前を指定します。

例

```
define alias Name char(31) varying;
               /* Name has attributes char(31) varying */
dcl Employee_Name type Name;
               /* Employee_Name type char(31) varying */
define alias Rate fixed dec(3,2);
               /* Rate has attributes fixed dec real */

define structure
1 Payroll,
2 Name,
3 Last  type Name,
3 First type Name,
2 Hours,
3 Regular  fixed dec(5,2),
3 Overtime fixed dec(5,2),
2 Rate,
3 Regular  type Rate,
3 Overtime type Rate;

dcl Non_Exempt type Payroll;     /* Has Payroll structure type */
dcl Exempt     type Payroll;     /* Has Payroll structure type */
```

TYPE 属性を DEFINE ALIAS ステートメントで指定して、直前の DEFINE ALIAS ステートメントで定義されたタイプのための別名を指定することができます。 以下に例を示します。

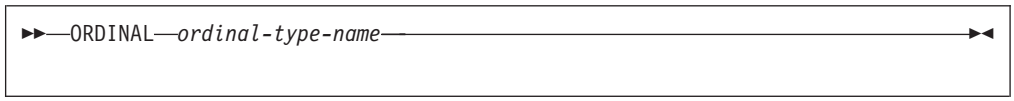
```
define alias Word fixed bin(31);
define alias Short type word;
```

以下の例は、いくつかの名前付きタイプ、構造体タイプ (tm) を定義し、このタイプ付き構造体へのハンドルを取得する C 関数を宣言しています。

```
define alias int     fixed bin(31);
define alias time_t  fixed bin(31);
define structure
  1 tm
   ,2 tm_sec   type int    /* seconds after the minute (0-61)     */
   ,2 tm_min   type int    /* minutes after the hour (0-59)       */
   ,2 tm_hour  type int    /* hours since midnight (0-23)         */
   ,2 tm_mday  type int    /* day of the month (1-31)             */
   ,2 tm_mon   type int    /* months since January (0-11)         */
   ,2 tm_year  type int    /* years since 1900                    */
   ,2 tm_wday  type int    /* days since Sunday (0-6)             */
   ,2 tm_yday  type int    /* days since January 1 (0-365)        */
   ,2 tm_isdst type int    /* Daylight Saving Time flag           */
   ;

dcl localtime    ext('localtime')
            entry( nonasgn byaddr type time_t )
            returns( byvalue handle tm );

dcl time    ext('time')
            entry( byvalue pointer )
            returns( byvalue type time_t );
```

ORDINAL 属性

TYPE 属性または ORDINAL 属性を使用して、変数を序数タイプで宣言することができます。 TYPE 属性の構文については、152 ページの『TYPE 属性』を参照してください。



ordinal-type-name
定義済みの序数タイプ値のセット名を指定します。

以下に例を示します。

```
dcl Wall_color ordinal Color;
```

ORDINAL 属性は FIXED や SIGNED などのほかの属性とは対立しますが、BASED や DIMENSION などとは併用できます。

タイプ付き構造体修飾

. 演算子または => 演算子を持つハンドルを使用して、タイプ付き構造体のメンバーを参照できます。典型的な非タイプ付き構造体内の名前の場合とは異なり、タイプ付き構造体内の名前は独自の「ネーム・スペース」を形成し、それら自体から参照することはできません。たとえば、以下の宣言と定義を指定すると、

```
dcl 1 A,
2 B fixed bin,
2 C fixed bin;

define structure
1 X,
2 Y fixed bin,
2 Z fixed bin;
dcl S type X;
```

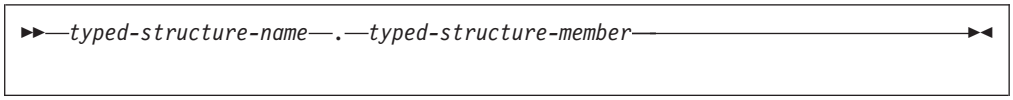
B は有効な参照ですが、Y は無効です。

タイプ名も宣言された名前からは、別のネーム・スペース内にあります。そのため、タイプの名前を変数名としても使用することができます。

```
define alias Hps pointer;
declare Hps type Hps;
```

'.' 演算子の使用

"." 演算子を使用してタイプ付き構造体メンバーを参照する構文は、次のとおりです。



typed-structure-reference
宣言されたタイプ付き構造体の名前です。

typed-structure-member
構造体タイプの、参照された大構造体メンバーまたは小構造体メンバーの名前です。

例えば、152 ページの例の中で定義された構造体タイプ tm と関数 localtime を指定して、以下のコードはシステム日付を入手し、時刻を表示します。

```
     dcl Daterec type tm;

     dcl ltime   type time_t;
     dcl ptime   handle tm;

     ltime = time( null() );
     ptime = localtime( ltime );

     Daterec = ptime => tm;

display ( edit(Daterec.Hours,'99') || ':' ||
edit(Daterec.Minutes,'99') || ':' ||
edit(Daterec.Seconds,'99'));
```

配列およびタイプ付き構造体または共用体の組み合わせ

191 ページの『配列、構造体、および共用体の組み合わせ』に説明されているように、この非タイプ付き構造体を指定すると、

```
dcl 1 a(3),
      2 b(4) fixed bin,
      2 c(5) fixed bin;
```

a(1).b(2)、a.b(1,2)、および a(1,2).b は同じ意味になります。

ただし、以下のタイプ付き構造体を指定すると、

```
define structure
  1 t,
    2 b(4) fixed bin,
    2 c(5) fixed bin;

dcl x(3) type t;
```

x(1).b(2) だけが有効です。さらに、代入ステートメント x.b = 0 は無効ですが、x (1) .b = 0; は有効です。

直前に定義された構造体タイプ t と以下の関数 f を指定すると、

```
dcl f entry returns( type t );
```

display(f().b(2)) は有効です。

ハンドルの使用

ハンドルは => 演算子を持つタイプ付き構造体のメンバーにアクセスします。以下の例では、152 ページで定義された tm タイプを指定すると、そのタイプのハンドルを使用して時間が表示されます。

```
dcl P_Daterec handle tm;
P_Daterec = handle(Daterec);

display ( edit(P_Daterec=>tm_hours,'99') || ':' ||
edit(P_Daterec=>tm_min,'99') || ':' ||
edit(P_Daterec=>tm_sec,'99') );
```

ハンドルは、レベル 1 名 (タイプ名自体) を含むタイプ付き構造体内のすべてのメンバーを見付けることができます。そのタイプ名についての、ハンドルによる参照は、そのハンドルによってポイントされた構造体への参照を構成することかできます。これで、そのハンドルによってこの集合データについての参照ができるようになります。たとえば、H1 と H2 が 2 つの割り振り済み構造体をポイントしている場合、次のようにして 2 つの構造体を交換できます。

```
define structure 1 T, 2 U, 2 V, 2 W;
dcl (H1, H2) handle T;
dcl Temp     type T;

Temp = H1=>T;
H1=>T = H2=>T;
H2=>T = Temp;
```

序数の使用

序数を使用する場合は、以下の点に留意してください。

- 序数は*強いタイプ* です。つまり、序数は同じタイプの別の序数に対してのみ割り当てたり比較することができます。序数は DECLARE ステートメントで宣言されていなければなりません。
- DEFINE ORDINAL ステートメント内の ordinal-type-name は、比較や割り当てには使用できません。
- 序数はほかのデータ・タイプと同様に引数/パラメーターとして受け渡しできます。
- 計算タイプの引数を必要とする組み込み関数では、序数を引数として使用することはできません。ただし、序数をサポートするための組み込み関数が定義されており、 BINARYVALUE が拡張されています。 156 ページの表 23 は、これらの組み込み関数をリストしています。詳細については、 395 ページの『第 19 章 組み込み関数、疑似変数、およびサブルーチン』を参照してください。これらの組み込み関数は 1 つの引数だけを取ります。この引数は ORDINAL タイプの参照でなければなりません。

表 23. 序数処理組み込み関数

関数	説明
BINARYVALUE	序数を 2 進値に変換します。
ORDINALPRED	序数が次に低い値を戻します。
ORDINALSUCC	次に高い序数値を戻します。
ORDINALNAME	序数の名前を指定する文字ストリングを戻します。

例えば、下記のサンプル・コードでは、最初の DO ループが *Color* セットのメンバーを昇順にリストし、2 番目の DO ループが同じセットのメンバーを降順にリストします。この例では、149 ページの『例』に示した序数定義が使用されています。

例

```
dcl Next_color ordinal Color;

do Next_color = first (:Color:)
                repeat ordinalsucc( Next_color )
                until (Next_color = last (:Color:));

  display( ordinalname( Next_color ) );
end;

do Next_color = last (:Color:)
                repeat ordinalpred( Next_color)
                until (Next_color = first(:Color:);

  display( ordinalname( Next_color));
end;
```

最初のループのサンプル出力は次のようになります。

```
RED
ORANGE
YELLOW
GREEN
BLUE
INDIGO
VIOLET
```

序数は、配列の下位境界または上位境界を含めて、配列への索引としては使用できず、変数のエクステントとして定義することもできません。 ただし、BINARYVALUE 組み込み関数を使用して、序数を 2 進値に変換することは可能です。この関数が返す値を使用して、配列の索引を作成したり、エクステントを定義することができます。

たとえば、下記のパッケージでは、配列 *usage_count* の定義によって各カラーが使用される回数が保持され、プロシージャー *Record_usage* によってこの配列が更新され、さらにプロシージャー *Show_usage* によってこの配列内の値が表示されます。

例

```
Usage: package exports(*);

  define ordinal Color ( Red,
                         Orange,
                         Yellow,
                         Green,
                         Blue,
```

```
                        Indigo,
                        Violet );

  dcl Usage_count(   binvalue( first(:Color:))
                   : binvalue( last(:Color:)) )
         static fixed bin(31) init( (*) 0 );
              /* first(:Color:)  = Red    */
              /* last(:Color:) = Violet */

Record_usage: proc (Wall_color );
  dcl Wall_color type Color parm byvalue;

  Usage_count( binvalue(Wall_color) )
      = 1 + Usage_count( binvalue(Wall_color) );
end Record_usage;

Show_usage: proc;
  dcl Next_color type Color;

  do Next_color = Red upthru Violet;
    put skip list( ordinalname( Next_color) );
    put list( Usage_count( binvalue(Next_color) ));
  end;
end Show_usage;

end Usage;
```

序数を使用すれば、保守と拡張が容易で、しかも表索引と同じくらい効率的な関数を作成することができます。

下記の例では、関数 *Is_mellow* により、カラーが「柔らかい」かどうかを示すビットが返されます。もっと多くのカラーが定義される場合は、 SELECT グループに「柔らかい」カラーを追加できます。テーブルを作成した場合とは異なり、SELECT グループでは、カラーの順序は DEFINE ステートメントでの順序と同じでなくてもかまいません。あるいは、カラーを特定の順序に並べる必要はありません。

SELECT グループ内部のステートメントはすべて定数値を返す RETURN ステートメントですから、コンパイラーが SELECT グループ全体をテーブル参照に変換します。

例

```
Is_mellow: proc( Test_color ) returns( bit(1) aligned );

  dcl Test_color type Color parm byvalue;

  select (Test_color);
    when( Yellow, Indigo)
        return( '1'b );
    otherwise
        return( '0'b );
  end;

end;
```

この機能を使用して、ユーザー独自の ORDINALNAME 組み込み関数を作成することも可能です。この独自の関数は、各序数値について、ユーザーが表示させたい名前を返すことができます。たとえば、下記の関数 Color_name は、最初の文字を大文字とする名前に関連付けられたカラー名を返します。

```
Color_name: proc( Test_color ) returns( char(8) varying );

  dcl Test_color type Color parm byvalue;

  select (Test_color);
    when ( Blue   ) return( 'Blue');
    when ( Green  ) return( 'Green');
    when ( Orange ) return( 'Orange');
    when ( Red    ) return( 'Red');
    when ( Yellow ) return( 'Yellow');
    otherwise return ('');
  end;

end;
```

タイプ付き関数

タイプ名は宣言名とは別のネーム・スペースにあるので、それらを変数参照が必要な所で使用することはできません。特に組み込み関数の引数として使用することはできません。ただし、タイプ名は*タイプ付き関数* への引数として使用することはできます。 (ANSI 用語では、これらのタイプ付き関数は*照会関数* として知られています。) これらのタイプ付き関数は、表 24 にまとめられています。

表 24. タイプ付き関数

関数	説明
BIND	ポインターをタイプのハンドルに変換します。
CAST	C の変換規則を使用して、式を指定のタイプに変換します。
FIRST	最初の値を序数セットに戻します。
LAST	最後の値を序数セットに戻します。
NEW	構造体タイプのためにストレージを獲得し、獲得されたストレージにハンドルを戻します。
RESPEC	式のビット・パターンを変更せずに、式の属性を指定のタイプに変更します。
SIZE	タイプを表すために必要なストレージの量を戻します。

これらのタイプ付き関数についての説明は、531 ページの『第 20 章 タイプ付き関数』に記載されています。

第 8 章 データ宣言

明示宣言 159
DECLARE ステートメント 160
属性分配 162
暗黙宣言 162
宣言の有効範囲. 163
INTERNAL 属性と EXTERNAL 属性 166
RESERVED 属性 170
データの位置合わせ 171
ALIGNED 属性と UNALIGNED 属性 172
属性のデフォルト 175
言語に固有のデフォルト. 176
DEFAULT ステートメント 177
言語に固有のデフォルトの復元 181
配列 181
DIMENSION 属性 182
配列の例 182
添え字. 183
配列のクロスセクション. 184
構造体. 185
共用体. 186
UNION 属性. 187
構造体/共用体の修飾 187
LIKE 属性 188
例 189
NOINIT 属性 190
集合体の組み合わせおよびマッピング 191
配列、構造体、および共用体の組み合わせ. . . 191
構造体または共用体の配列のクロスセクション 192
構造体および共用体の操作 192
構造体および共用体のマッピング. 192

PL/I プログラムを実行すれば、特定のデータ・タイプのさまざまなデータ項目が処理されます。名前が付いていない算術定数やストリング定数を除き、プログラム内ではデータ項目はすべて名前によって参照されます。各データ名には、(明示的または暗黙的な) 宣言によって、属性と意味が与えられます。

データ項目のほとんどの属性は、プログラムがコンパイルされるときに判別されます。非静的項目の場合は、属性の値 (配列の次元の境界、ストリングの長さ、区域のサイズ、初期値) と一部のファイル属性は、プログラムの実行時に判定されます。詳細については、95 ページの『ブロックの活動化』を参照してください。

データ項目、タイプ、および属性については、23 ページの『第 3 章 データ・エレメント』で紹介しています。

この章では、明示宣言および暗黙宣言、スカラー、配列、構造体、および共用体の宣言、名前の有効範囲、データの位置合わせ、デフォルトの属性を説明します。

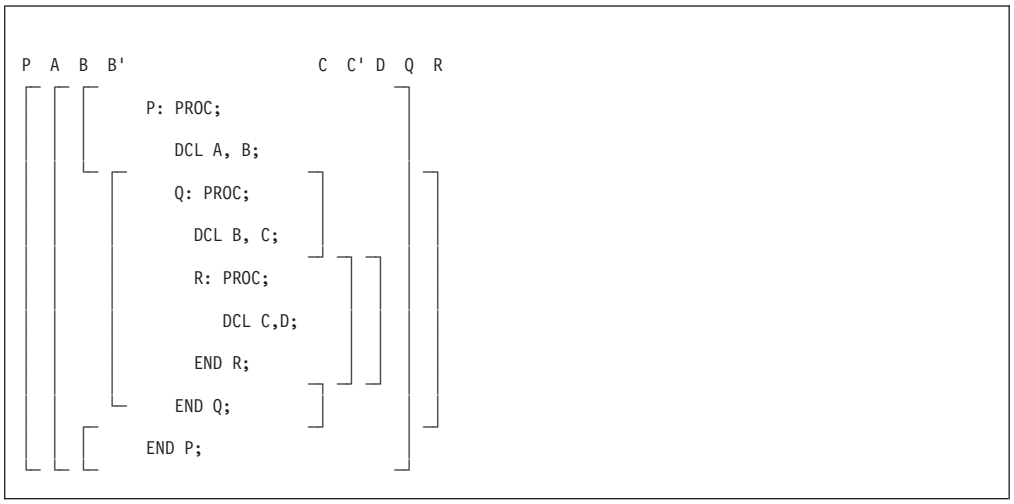
明示宣言

以下のように名前が示されている場合には、明示的に宣言されます。

- DECLARE ステートメントに示されている場合。 DECLARE ステートメントが明示的に名前の属性を宣言します。
- 入り口定数として示されている場合。 PROCEDURE ステートメントおよび ENTRY ステートメントのラベルは、含まれるプロシージャー内の入り口定数の宣言を制定します。
- ラベル定数として示されている場合。ラベル定数は、明示的にラベルを宣言します。
- フォーマット定数として示されている場合。 FORMAT ステートメントのラベルは、ラベルの明示宣言を制定します。

名前の明示宣言の有効範囲は、宣言を含むブロックです。つまり、同じ名前のもう 1 つの明示宣言が内部にあるブロック (および、それらに含まれるブロック) を除い

た、名前が含まれるすべてのブロックです。次の図では、ラインで名前の宣言の有効範囲を示します。



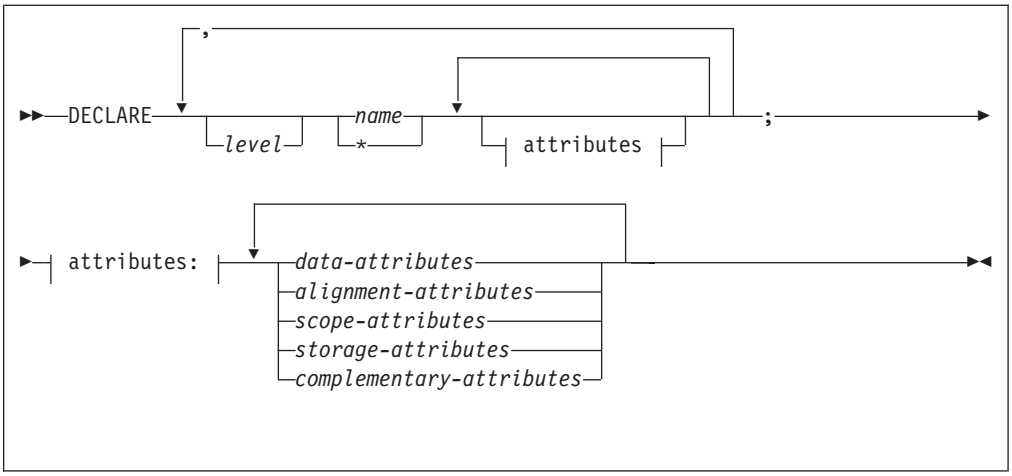
B と B' は、名前 B の 2 つの別個の使用法について示しています。 C と C' は、名前 C の 2 つの使用法について示しています。

有効範囲に関する詳細は、 163 ページの『宣言の有効範囲』を参照してください。

DECLARE ステートメント

DECLARE ステートメントは、名前の一部の属性またはすべての属性を指定します。属性が明示的に宣言されず、コンテキストによって判別できない場合は、デフォルト属性が適用されます。

DECLARE ステートメントは、プログラムの文書の重要な部分になることがあります。同様に、デフォルト属性で十分なとき、または暗黙宣言が可能なときであっても、多数の宣言を使用することができます。 DECLARE ステートメントの数に関しては制限がないので、さまざまな名前のグループに異なる DECLARE ステートメントを使用することができます。どんなに多くの名前でも、1 つの DECLARE ステートメント内で宣言することができます。



省略形: DCL

配列、構造体、および共用体の詳細については、 181 ページの『配列』、 185 ページの『構造体』、または 186 ページの『共用体』を参照してください。

* EXTERNAL 属性に環境名を指定しない限り、INTERNAL または EXTERNAL スカラーの *name* として、あるいはレベル 1 の EXTERNAL 構造体または共用体として使用することはできません (166 ページの『INTERNAL 属性と EXTERNAL 属性』を参照)。

attributes

属性は、任意の順序で示すことができます。

名前に明示的に与える属性はすべて、DECLARE ステートメント内に一緒に宣言されていなければなりません。ただし、以下はこの例外となります。

FILE 属性を持つ名前には、OPEN ステートメントで属性を与える (あるいは暗黙のオープンによって属性を暗黙に与える) こともできます。 OPEN ステートメントの詳細については、 286 ページの『OPEN ステートメント』を参照してください。

パラメーター・リスト内の名前には、パラメーター属性がコンテキストによって与えられます。そのブロックの内部にある DECLARE ステートメントで、その他の属性を指定することができます。

それぞれ別個のブロックとコンパイルの外部名の属性は整合性を必要とします。

グループの属性とメンバーについては、 24 ページの『データ・タイプとその属性』を参照してください。

level

ゼロ以外の整数値です。レベル番号を指定しないと、*レベル 1* が要素変数と配列変数のためのデフォルトとなります。*レベル 1* は、大構造体と共用体の名前について指定しなければなりません。

name

レベル 1 のそれぞれの名前は、ブロック内で固有でなければなりません。レベル 1 の名前の詳細については、 185 ページの『構造体』を参照してください。

条件接頭語とラベルは、DECLARE ステートメント上には指定できません。

属性分配

いくかの名前に共通する属性を分配して、同じ属性を繰り返し指定する手間を省くことができます。属性分配は、いくつかの名前を括弧で囲み、そのあとに、それらすべての名前に適用される属性のセットを置くことによって実現します。属性分配はネストできます。また、次元属性を分配することができます。属性分配は、構造体および共用体の基本名でも使用することができます。分配されるレベル番号は、括弧で囲まれたリストの前に付けます。

括弧で囲まれたリスト内の名前は、コンマで区切られます。分配される属性によって名前を指定変更することはできませんが、リスト内の任意の名前には、分配される属性と対立しない限り、他の属性を指定することができます。

下記の例は属性分配を示しています。この例の最後の宣言は、ネストされる属性分配を示します。

```
declare (A,B,C,D) binary fixed (31);

declare (E decimal(6,5), F character(10)) static;

declare 1 A, 2(B,C,D) (3,2) binary fixed (15);

declare ((A,B) fixed(10),C float(5)) external;
```

暗黙宣言

プログラム内で使用する名前を明示的に宣言しなかった場合、その名前は暗黙のうちに宣言されます。暗黙宣言の有効範囲は、その名前が使われている外部プロシージャーの PROCEDURE ステートメントのすぐあとで DECLARE ステートメントによってその名前を宣言した場合と同様にして決まります。

ファイル、入り口、組み込み関数を除き、暗黙宣言は、名前が最外部のプロシージャーで宣言された場合と同様の効果を持ちます。ファイルと組み込み関数については、暗黙宣言は、名前が任意のプロシージャーの外側の論理パッケージに宣言された場合と同様の効果を持ちます。

注: 組み込み関数およびファイル SYSIN と SYSPRINT 以外のものに暗黙宣言を使用するのは、1987 ANSI 規格に対する違反であり、避けなければなりません。

暗黙のうちに宣言された名前の属性の中には、その名前が使われているコンテキスト上から判定することができます。これは*コンテキスト宣言* と呼ばれ、次のような場合です。

- 組み込み関数の名前。
- CALL ステートメント、または INITIAL の CALL オプション内に示される名前、あるいは引数リストの前にくる名前には、 ENTRY 属性と EXTERNAL 属性が与えられます。
- PROCEDURE ステートメントまたは ENTRY ステートメントのパラメーター・リストに示される名前には、 PARAMETER 属性が与えられます。
- FILE または COPY オプションにおいて示される名前、もしくはファイル名が必要な条件に対するステートメント ON、SIGNAL または REVERT に示される名前には、FILE 属性が与えられます。

- ステートメント ON CONDITION、SIGNAL CONDITION、または REVERT CONDITION に示される名前には、CONDITION 属性が与えられます。
- BASED 属性、SET オプションに示される名前、またはロケーター修飾記号の左側に現れる名前には、POINTER 属性が与えられます。
- IN オプションまたは OFFSET 属性に示される名前には、AREA 属性が与えられます。

コンテキスト宣言の例は、以下のとおりです。

```
read file (PREQ) into (Q);

allocate X in (S);
```

これらのステートメントでは、PREQ に FILE 属性、 S に AREA 属性が与えられます。

コンテキスト宣言ではない暗黙宣言は、 175 ページの『属性のデフォルト』に示すように、すべてデフォルトによって属性を取得します。コンテキスト宣言は明示宣言の有効範囲内には存在することができないため、明示宣言内の名前について確立された属性に、名前のコンテキストが別の属性を追加することはできません。

宣言の有効範囲

名前の適用対象となるプログラム部分は、その名前についての*宣言の有効範囲* と呼ばれます。ほとんどの場合、名前の宣言の有効範囲は、プログラム内で名前が宣言される位置によって完全に決定されます。暗黙宣言は、名前が外部プロシージャーの PROCEDURE ステートメントの直後にある DECLARE ステートメントで宣言されたものと見なされます。

名前がプログラム全体に対して同じ意味を持つ必要はありません。ブロック内で明示的に宣言された名前は、そのブロック内だけで意味を持ちます。そのブロックの外部では、同じ名前がそこでも宣言されていない限りその名前は認識されません。名前を宣言するたびに、その有効範囲が確立されます。この場合、その外側ブロック内の名前は、別のデータ項目を参照することになります。このため、ローカル定義を指定することにより、プログラムのほかの部分で使用されているすべての名前を知らなくても、プロシージャーまたは開始ブロックを作成することが可能になります。

下記の例で、A の実際の出力は C.A、つまり 2 です。 B の出力は、プロシージャー X で宣言されたとおり 1 です。

```
X: proc options(main);
  dcl (A,B) char(1) init('1');
  call Y;
  return;

  Y: proc;
    dcl 1 C,
        3 A char(1) init('2');
    put data(A,B);
    return;
  end Y;
end X;
```

したがって、ネストされたプロシージャーでは、 PL/I は現行ブロック内に宣言された変数を使用してから、包含ブロックで宣言された変数を使用することになります。

名前の宣言の有効範囲を理解するためには、*包含された* および*内部にある* という 2 つの用語を理解しなければなりません。

PACKAGE、PROCEDURE、または BEGIN ステートメントから、それに対応する END ステートメント (BEGIN、PACKAGE、および PROCEDURE ステートメントの条件接頭語も含む) までの間にあるブロックの全テキストは、そのブロックに包含されると見なされます。 ただし、ブロックに適用する任意の ENTRY ステートメントのラベルと同様に、ブロックの先頭にある BEGIN または PROCEDURE ステートメントのラベルは、そのブロックに含まれません。ネストされたブロックは、それらを包むブロックに包含されます。

ブロックに包含されるテキストのうち、そのブロック内のネストされているほかのブロックに包含されていないものは、そのブロックの内部にあると見なされます。プロシージャーの入り口名 (および BEGIN ステートメントのラベル) は、そのブロックに含まれません。したがって、プロシージャーの入り口名は、包含ブロックの内部にあることになります。外部プロシージャーの入り口名は、その外部プロシージャーの外部にあると見なされます。

表 25 は、データ宣言の有効範囲を示しています。

表 25. データ宣言の有効範囲

```
          A_and_D:  package exports(*);
1,2         dcl X static,
                    Y based;
                                                P   Q   Q'  R   R'  S   I
2           ┌─ A:  procedure;                 ┐  ┐      ┐           ┐
3           │     declare P, Q;                │  ┘      │           │
            │ ┌─  B:  Procedure;               │      ┐  │           │
            │ │       declare Q;               │      │  ┘           │
3           │ │       R = Q;                   │      │              │
4           │ │   ┌─ C:  begin;                │      │     ┐        │
            │ │   │     declare R;             │      │     │        │
            │ │   │     do I = 1 to 10;        │      │     │        │
            │ │   │     end;                   │      │     ┘        │
            │ │   └─ end C;                    │      ┘              │
            │ └─  end B;                       │      ┘              │
            │ ┌─  D:  procedure;               │  ┐      ┐      ┐    │
5           │ │       declare S;               │  │      │      ┘    │
            │ └─  end D;                       │  │      │      ┘    │
            └─ end A;                          ┘  ┘      ┘           ┘

          end A_and_D;
```

左方の大括弧はブロック構造体を示しています。右方の大括弧は名前の各宣言の有効範囲を示しています。 Q と R の 2 つの宣言の有効範囲は、 Q と Q'、および R と R' となっています。

X と Y は、パッケージに包含されるすべてのプロシージャーにとって可視となっています。

1 P はブロック A で宣言され、再宣言されていないため、A 全体で認識されます。

2 Q はブロック A で宣言され、さらにブロック B で再宣言されています。 Q の最初の宣言の有効範囲は、 B を除く A のすべてです。 Q の 2 番目の宣言の有効範囲は、ブロック B のみです。

3 R はブロック C で宣言されていますが、 R への参照がブロック B で行われています。ブロック B での R への参照により、外部プロシージャーの A で R の暗黙宣言が行われます。したがって、異なる有効範囲を持つ 2 つの別個の名前 (表 25 の R と R') が存在することになります。明示的に宣言された R の有効範囲は、ブロック C です。ブロック B 内の暗黙的に宣言された R の有効範囲は、ブロック C を除く A のすべてです。

4 I はブロック C で参照されています。このため、外部プロシージャー A で暗黙宣言されていることになります。この結果、包含されたプロシージャー B、C、および D を含むすべての A に宣言が適用されます。

5 S はプロシージャー D で明示的に宣言され、 D の内部でのみ認識されます。

表 26 は、入り口定数とステートメント・ラベルの宣言の有効範囲を示しています。

表 26. 入り口およびラベル宣言の有効範囲

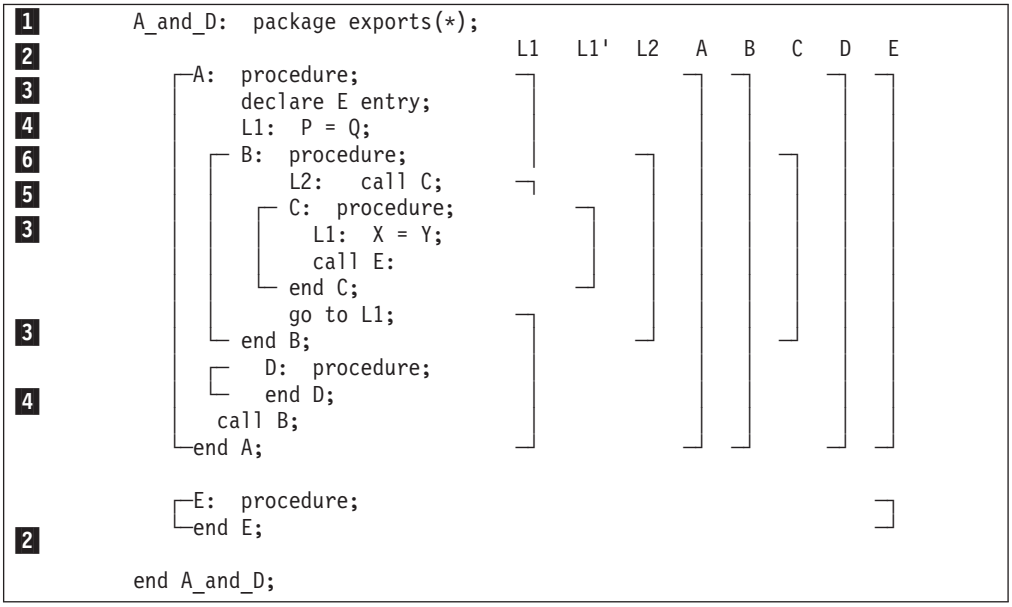


表 26 は、外部プロシージャーの A と E を示しています。

1 名前 A の宣言の有効範囲は、ブロック A のすべてに限られており、E は有効範囲ではありません。

2 E は A で外部入り口定数として明示的に宣言されています。 E の明示宣言は、ブロック A の全体に適用されます。ブロック E 全体に適用される E の明示宣言にはリンクしていません。名前 E の宣言の有効範囲は、ブロック A の全体とブロック E の全体です。

3 ラベル L1 は、A と C の内部のステートメントに示されています。その結果、2 つの別々の宣言が確立されます。最初のものは、ブロック C を除くブロック A のすべてに適用され、2 番目のものはブロック C にのみ適用さ

れます。したがって、ブロック B の GO TO ステートメントが実行されると、制御はブロック A の L1 に移され、ブロック B は終了します。

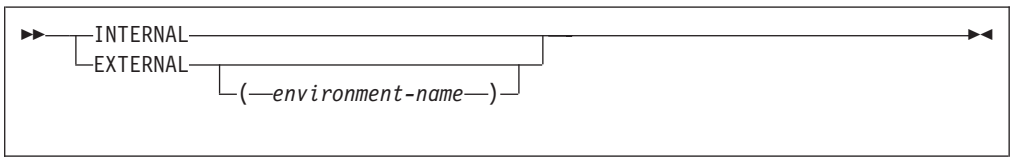
4 D と B は、明示的にブロック A 内で宣言され、 A 内の任意の場所で参照することができます。ただし、それらは INTERNAL であり、ブロック E 内で参照することはできません。

5 C は B で明示的に宣言され、 B の内部から参照できますが、B の外側からは参照できません。

6 L2 は B で宣言され、ブロック B、および B に包含される C で参照できますが、 B の外側からは参照できません。

INTERNAL 属性と EXTERNAL 属性

INTERNAL 属性と EXTERNAL 属性は、名前の有効範囲を指定します。



省略形: INTERNAL は INT、EXTERNAL は EXT

environment-name

コンパイル単位の外側で認識されるプロシージャー名または変数名を指定します。

この指定により、宣言中の名前は実質的に内部となり、コンパイル単位の外側では認識されなくなります。宣言中の名前の代わりに環境名が知られるようになります。

環境名は文字ストリング定数でなければならず、大文字への変換なしにそのまま使用されます。

以下に例を示します。

```
dcl X entry external ('koala');
```

環境名を区切り文字 (_) で開始することは避けてください。下線文字で開始される名前は、ライブラリー用として予約されています。

リンカーが環境名を*修飾*するプラットフォームでは、外部属性を持つ環境名を指定しても、その名前が変数名と大文字/ 小文字の違いしかない場合は、その環境名は修飾されます。次の宣言では、

```
dcl abc ext('kLm'), xyz ext('xYz' );
```

xyz の名前が修飾されます。環境名の装飾について詳しくは、PL/I for Windows の「*プログラミング・ガイド*」の「呼び出し規則」という章の「リンケージに関する考慮事項」を参照してください。

INTERNAL は、内部プロシージャーの入り口名、および任意のストレージ・クラス (被制御を除く) を持つ変数のデフォルトです。 INTERNAL は、その名前が、それを宣言しているブロック内でだけ認識されることを指定します。別の場所で同じ名

前を明示的に宣言すると、別の有効範囲を持つ新しいオブジェクトを参照することになります。これらの 2 つの有効範囲は相互に重なることはありません。

注: INTERNAL は、パッケージ内のレベル 1 プロシージャー上で指定することができます。パッケージが EXPORTS(*) を指定して宣言してある場合、INTERNAL プロシージャーはパッケージの外部から不可視になります。

EXTERNAL は、ファイル定数、入り口定数、プログラマー定義の条件、および被制御変数のデフォルトです。 EXTERNAL 属性を持つ名前は、別々の外部プロシージャー内で、または外部プロシージャーに含まれているブロック内で複数回宣言することができます。 EXTERNAL 属性を持つ同じ名前を 2 回以上宣言した場合、それらの宣言はすべて同じデータを参照します。 EXTERNAL 属性を持つ名前の各宣言の有効範囲は、そのアプリケーション内でのその名前 (EXTERNAL 属性を持つもの) のすべての宣言の有効範囲を包含します。

大構造体または共用体の名前を複数のブロックで EXTERNAL として宣言する場合、対応するメンバー名は異なっていてもかまいませんが、メンバーの属性は同じでなければなりません。

次に例を示します。

```
ProcA: procedure;
declare 1 A external,
2 B,
2 C;
.
.
.


end ProcA;


%process;
ProcB: procedure;
declare 1 A external,
2 B,
2 D;
.
.
.
end ProcB;
```

A.B が ProcA 内で変換されると、それは ProcB についても変換され、逆の場合も同様です。 A.C が ProcA 内で変換されると、 A.D は ProcB について変換され、逆の場合も同様です。

構造体と共用体のメンバーは、常に INTERNAL 属性を持ちます。

同じ名前の外部宣言はすべて同じデータを共用するため、それらの宣言の結果はすべて同じセットの属性にならなければなりません。 EXTERNAL の名前が別の外部プロシージャーで宣言されるときは、ユーザーは属性が確実に一致するようにしなければなりません。 168 ページの図 7 は、さまざまな宣言とその有効範囲を説明しています。

```
          Scope_Example:  package exports(*);
1           A: procedure;
2              declare S character (20);
7              dcl Set entry(fixed decimal(1)),
7                  Out entry(label);
               call Set (3);
9           E: get list (S,M,N);
8           B: begin;
4,5              declare X(M,N), Y(M);
                 get list (X,Y);
                 call C(X,Y);

9,5         C:  procedure (P,Q);
                   declare
                     P(*,*),
                     Q(*),
12,2                 S binary fixed external;
                   S = 0;
6                  do I = 1 to M;
                     if sum (P(I,*)) = Q(I) then
8                       go to B;
                     S = S+1;
                     if S = 3 then
9                       call Out (E);
                     Call D(I);
8           B:  end;
            end C;

9           D:  procedure (N);
                  put list ('Error in row ',
2,3                     N, 'Table Name ', S);
            end D;
            end B;
            go to E;
            end A;
9          Out:  procedure (R);
                 Declare
                   R Label,
11                 (K static internal,
11,7                  L static external) init (0),
12                 S binary fixed external,
                   Z fixed decimal(1);
                 K = K+1; S=0;
                 if K<L then
                   stop;
10               else go to R;
           end;
           Set:  procedure (Z);
                 declare Z fixed dec(1);
7                L=Z;
                 declare L external init(0);
                 return;
                 end;
             end Scope_Example;
```

図 7. 各種宣言の有効範囲の例

1 A は外部プロシージャー名です。この有効範囲は、ブロック A の全体と、A が外部として宣言されているほかの任意のブロックです。

2 S は、ブロック A とブロック C で明示的に宣言されています。文字変数の宣言は、ブロック C を除くブロック A 全体に適用されます。固定 2 進値宣言は、ブロック C の内部でのみ適用されます。 D はブロック C から呼

び出されますが、D 内の PUT ステートメントでの S への参照は文字変数 S への参照であり、ブロック C 内で宣言されている S への参照ではありません。

3 N はブロック D 内のパラメーターとして書かれていますが、このブロックの外でも使用されます。パラメーターとして書かれているため、D 内で N の明示宣言が確立されます。 D の外部での参照により、ブロック A 内での N の暗黙宣言が行われています。名前 N の 2 つの宣言は、別々のオブジェクトを参照します。ただし、この場合、これらのオブジェクトは同じデータ属性を持ちます。データ属性は、デフォルトにより、FIXED BINARY(15,0) と INTERNAL です。DEFAULT(ANS) の場合の精度は (31,0) です。

4 X と Y は B 全体で認識され、B 内部のブロック C または D から参照できます。ただし、B の外側の A 部分からは参照できません。

5 P と Q はパラメーターです。したがって、ブロック内でこれらの名前の宣言がほかになければ、パラメーター・リストに指定されているだけでコンテキスト宣言がされていることになります。しかし、P と Q が配列であることを指定するには、別個の明示宣言が必要です。引数 X と Y は配列として宣言されており、ブロック C 内で認識されていますが、P と Q も配列であることを指定するには、DECLARE ステートメントでそれらを宣言しなければなりません。 (アスタリスク表記は、パラメーターの境界が引数の境界と同じであることを示します。)

6 I と M は、外部プロシージャー A では明示的に宣言されていません。したがって、これらは暗黙に宣言され、A 全体で認識されます。ただし、I はブロック C の中でしか使われません。

7 例の中の外部プロシージャー Out と Set には、これら両方に共通の L という外部宣言があります。これらは、プロシージャー A で ENTRY 属性を付けて明示的に宣言しなければなりません。ENTRY は EXTERNAL を暗黙指定するため、2 つの入り口定数 Set と Out は、2 つの外部プロシージャー全体で認識されます。

8 ラベル B はこのプログラムに 2 回現れます。最初は A 内で開始ブロックのラベルとしてであり、明示宣言となります。次にブロック C 内のラベルとして宣言され、END ステートメントの接頭部として使用されます。したがって、C 内の go to B ステートメントは、C 内の END ステートメントのラベルを参照することになります。ブロック C の外側では、B への参照は開始ブロックのラベルへの参照となります。

9 ブロック C と D は、B 内部のどこからでも呼び出せますが、B の外の A 部分や別の外部プロシージャーからは呼び出せません。同様に、ラベル E は外部プロシージャー A 全体で認識されているため、E への制御の伝送は A 内部のどこからでも行うことができます。ただし、ブロック C 内のラベル B は、 C の中からのみ参照できます。GO TO ステートメントによってブロックの外への伝送ができます。しかし、ネストされたブロックへのそのような伝送は、一般的に行えません。この例外となるケースは外部プロシージャー Out に示されています。ここでは、ラベル E がブロック C から引数としてラベル・パラメーター R に渡されています。

GET ステートメントと PUT ステートメントにはファイルが指定されていないため、SYSIN と SYSPRINT は暗黙宣言されていることに注意してください。

10 ステートメント `else go to R;` は、制御をラベル `E` に伝送します。`E` が `A` 内で宣言されており、`Out` 内では知られていない場合でも同様です。

11 変数 `K` (INTERNAL) と `L` (EXTERNAL) は、`Out` プロシージャー・ブロック内で STATIC として宣言されています。それらの値は `Out` への呼び出しの間で保護されています。

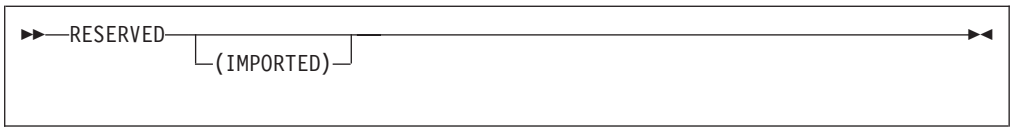
12 プロシージャー `Out` の中の `S` をプロシージャー `C` の中の `S` と同じものとして識別するために、どちらも属性 EXTERNAL を付けて宣言されています。

RESERVED 属性

RESERVED 属性は、STATIC EXTERNAL を暗黙指定します。さらに、変数が RESERVED 属性を持つ場合、アプリケーションは以下の条件に従います。

- 変数のすべての宣言は、RESERVED を指定しなければなりません。
- 変数名は、必ず 1 つのパッケージの RESERVES オプション中に現れなければなりません。

変数が RESERVED 属性を持つ場合、その変数がパッケージ内に保存されていない限り、INITIAL 値は無視されます。



コンパイル単位に RESERVED 属性の変数があり、それがその変数用の保存パッケージではない場合、そのコンパイル単位は、保存パッケージを含むロード・モジュールの一部分であるか、または保存パッケージを含む別のロード・モジュールから変数をインポートしなければなりません。後者の場合、宣言で RESERVED 属性の IMPORTED オプションを指定しなければなりません。

```
owns_x:
  package
  exports(*)
  reserves(x);

  dcl x char(256) reserved init( ... );
  dcl y char(256) reserved init( ... );
  dcl z char(256) reserved(imported) init( ... );

end;

owns_y:
  package
  exports(*)
  reserves(y);

  dcl x char(256) reserved init( ... );
  dcl y char(256) reserved init( ... );
  dcl z char(256) reserved(imported) init( ... );
```

```
end;

owns_z:
  package
  exports(*)
  reserves(z);

  dcl z char(256) reserved(imported) init( ... );

end;
```

上記の例では、パッケージ *owns_x* が変数 *x* のストレージを保存し初期設定します。このパッケージは、パッケージ *owns_y* と同じロード・モジュールに連係されていなければなりません。また、このロード・モジュールは、パッケージ *owns_z* が連係されたロード・モジュールから変数 *z* をインポートしなければなりません。

データの位置合わせ

コンピューターは情報を 8 ビットの倍数で保持します。この 8 ビットの情報単位をバイト といいます。

コンピューターによるアクセスは、1 バイト、ハーフワード、ワード、またはダブルワードを単位として行われます。ハーフワード は連続している 2 バイト、フルワード は連続している 4 バイト、ダブルワード は連続している 8 バイトをそれぞれ指します。 ストレージのバイト位置は 0 から開始される連続した番号が付けられます。各番号は対応するバイトのアドレスです。ハーフワード、ワード、およびダブルワードは、それぞれの一番左のアドレスでアドレス指定されます。

ハーフワード、ワード、およびダブルワードが、主記憶域でそれぞれの情報単位の規定境界に置かれていると、プログラムの実行が高速になります。つまり、表 27 に示すように、情報単位のアドレスがその情報のバイト数の倍数となる場合、プログラムの実行が高速になります。

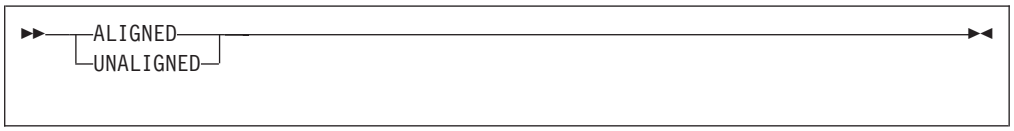
表 27. ハーフワード、ワード、およびダブルワードの規定境界の位置合わせ

ストレージ・セクションのアドレス							
5000	5001	5002	5003	5004	5005	5006	5007
バイト	バイト	バイト	バイト	バイト	バイト	バイト	バイト
ハーフワード		ハーフワード		ハーフワード		ハーフワード	
フルワード				フルワード			
ダブルワード							

PL/I は規定境界でのデータ位置合わせが可能です。ただし、連続するデータ・エレメント間の未使用バイトがあると、ストレージの使用が増加します。たとえば、データ項目がデータ・セットを作成するために使用される集合のメンバーである場合、未使用バイトによって補助記憶域の必要量が増えることになります。
ALIGNED 属性と UNALIGNED 属性を使用すれば、適切な規定境界にデータを位置合わせするかどうかを選択できます。

ALIGNED 属性と UNALIGNED 属性

ALIGNED を指定すると、データ・エレメントがそのデータ・タイプ要件に合うストレージ境界で位置合わせされます。 UNALIGNED は、各データ・エレメントが次のバイト境界でマップされることを指定します。ただし、固定長のビット・ストリングは例外であり、次のビットでマップされます。



デフォルトはエレメント・レベルで適用されます。 UNALIGNED はビット・データ、文字データ、グラフィック・データ、ワイド文字データ、および数字データのデフォルトです。 ALIGNED は、その他のすべてのデータ・タイプのデフォルトです。

ALIGNED 属性の要件は、表 28 に記載されています。

表 28. 位置合わせ要件

変数タイプ	内部での格納	ストレージ所要量 (バイト)	位置合わせ要件	
			ALIGNED データ	UNALIGNEDData

注:

プログラム制御データの位置合わせおよびストレージ所要量で、サポートされるシステムを通して変更することができます。

複素数データは実数データの 2 倍のストレージを必要としますが、位置合わせ要件は同じです。

BIT(n)	ALIGNED: 8 ビットのグループ (またはその一部) ごとに 1 バイト UNALIGNED: バイト境界に関係なく、必要なだけのビット数	ALIGNED: CEIL(n/8) UNALIGNED: n ビット	バイト (データは、0 から 7 までの任意のバイトで開始できます)	ビット (データは、バイト内の 0 から 7 までの任意のビットで開始できます)

表 28. 位置合わせ要件 (続き)

変数タイプ	内部での格納	ストレージ所要量 (バイト)	位置合わせ要件	
			ALIGNED データ	UNALIGNEDData
CHARACTER(n)	1 文字につき 1 バイト	n	バイト (データは、0 から 7 までの任意のバイトで開始できます)	バイト (データは、0 から 7 までの任意のバイトで開始できます)
CHARACTER (n)VARYINGZ	1 文字につき 1 バイト、および、ヌル終了文字の 1 バイト	n+1		
GRAPHIC(n)	1 グラフィックにつき 2 バイト	2n		
GRAPHIC (n)VARYINGZ	1 グラフィックにつき 2 バイト、および、ヌル終了文字の 2 バイト	2n+2		
WIDECHAR (n)	1 ワイド文字につき 2 バイト	2n		
WIDECHAR (n)VARYINGZ	1 ワイド文字につき 2 バイト、およびヌル終了文字の 2 バイト	2n+2		
PICTURE	PICTURE 文字 (V、K、および F スケール因数を除く) につき 1 バイト	V、K、および F 指定以外の PICTURE 文字の数		
DECIMAL FIXED (p,q)	パック 10 進フォーマット (1 桁につき 1/2 バイト、および、符号の 1/2 バイト)	CEIL((p+1)/2		
BINARY FIXED(p,q) SIGNED 1 <= p <= 7 UNSIGNED 1 <= p <= 8	1 バイト	1		
ORDINAL SIGNED 1 <= p <= 7 UNSIGNED 1 <= p <= 8				

表 28. 位置合わせ要件 (続き)

変数タイプ	内部での格納	ストレージ所要量 (バイト)	位置合わせ要件	
			ALIGNED データ	UNALIGNEDData
BIT(n) VARYING	2 バイトの接頭部、および、宣言される最大長の各 8 ビット・グループ (またはその一部) のための 1 バイト	ALIGNED:2+CEIL(n/8) UNALIGNED: 2 バイト +n ビット	ハーフワード (データは 0、2、4、または 6 バイトから開始できます)	バイト (データは、0 から 7 までの任意のバイトで開始できます)
CHARACTER(n)VARYING	2 バイトの接頭部、および、宣言される最大長の各文字につき 1 バイト	n+2		
GRAPHIC(n)VARYING	2 バイトの接頭部、および、宣言される最大長の各グラフィックにつき 2 バイト	2n+2		
WIDECHAR(n)VARYING	2 バイトの接頭部、および、宣言される最大長の各ワイド文字につき 2 バイト	2n+2		
BINARY FIXED(p,q) SIGNED 8 <= p <= 15 UNSIGNED 9 <= p <= 16	ハーフワード	2		
ORDINAL SIGNED 8 <= p <= 15 UNSIGNED 9 <= p <= 16				
BINARY FIXED(p,q) SIGNED 16 <= p <= 31 UNSIGNED 17 <= p <= 32	フルワード	4	フルワード (データは 0 または 4 バイトから開始できます)	バイト (データは、0 から 7 までの任意のバイトで開始できます)
ORDINAL SIGNED 16 <= p <= 31 UNSIGNED 17 <= p <= 32				
BINARY FLOAT(p)1<=p<=21	短浮動小数点			
DECIMAL FLOAT(p)1<=p<=6				
POINTER	-	4	フルワード (データは 0 または 4 バイトから開始できます)	バイト (データは、0 から 7 までの任意のバイトで開始できます)
HANDLE	-			
OFFSET	-			
FILE	-			
ENTRY LIMITED	-			
ENTRY	-	8		
LABEL or FORMAT	-			
TASK	-	16		

表 28. 位置合わせ要件 (続き)

変数タイプ	内部での格納	ストレージ所要量 (バイト)	位置合わせ要件	
			ALIGNED データ	UNALIGNEDData
AREA	-	16+size	ダブルワード (データは 0 バイトから開始できます)	AREA データは、位置合わせなしにすることはできません。
BINARY FIXED(p,q) SIGNED 32 <= p <= 63 UNSIGNED 33 <= p <= 64	-	8		バイト (データは、0 から 7 までの任意のバイトで開始できます)
BINARY FLOAT(p) 22 <= p <= 53	長浮動小数点			
DECIMAL FLOAT(p) 7 <= p <= 16				
BINARY FLOAT(p) 54<=p	拡張精度の浮動小数点	16		
DECIMAL FLOAT(p) 17<=p				

ALIGNED または UNALIGNED は、エレメント、配列、構造体、または共用体変数について指定できます。構造体または共用体にいずれかの属性を適用すれば、明示的には ALIGNED とも UNALIGNED とも宣言されていないすべての被包含エレメントに対してその属性を適用することになります。

以下に、構造体とそのエレメントについての、 ALIGNED と UNALIGNED 宣言の効果を図示した例を示します。

```
declare 1 S,
          2 X bit(2),      /* unaligned by default  */
          2 A aligned,     /* aligned explicitly    */
            3 B,           /* aligned from A        */
            3 C unaligned, /* unaligned explicitly  */
              4 D,         /* unaligned from C      */
              4 E aligned, /* aligned explicitly    */
              4 F,         /* unaligned from C      */
            3 G,           /* aligned from A        */
          2 H;             /* aligned by default    */
```

構造体と共用体の詳細については、 185 ページの『構造体』および 186 ページの『共用体』を参照してください。

属性のデフォルト

PL/I プログラムで使用する名前には、それぞれ完全な属性のセットが必須になります。プロシージャーに渡された引数は、プロシージャーのパラメーターと一致する属性を持っていなければなりません。関数が返す値は、予測されるとおりの属性を持っていなければなりません。ただし、ユーザーが属性の完全なセットを指定するのはまれです。

特に、
- 明示的に宣言された名前
- 暗黙に宣言された (コンテキストによる宣言を含む) 名前

- パラメーター記述子の中に含まれる属性
- 関数プロシージャーから返される値

上記の属性セットは、言語に固有のデフォルトを使用することにより、または言語に固有のデフォルトの変更もしくはまったく新しいデフォルト・セットの作成を通じてユーザーが (DEFAULT ステートメントを使用し) 定義するデフォルトを使用することにより完成することができます。

デフォルトによって適用される属性が、明示宣言またはコンテキスト宣言によって名前に適用された属性に優先することはありません。

言語に固有のデフォルト

データ属性を指定しないで宣言された変数には、デフォルトにより算術属性が与えられます。モード、スケール、基数が DECLARE ステートメントまたは DEFAULT ステートメントによって指定されていない場合は、DEFAULT コンパイラー・オプションがこれらの属性を次のように決定します。

- DEFAULT(IBM) が有効な場合は、文字 I から N で開始される名前を持つ変数は属性 REAL FIXED BINARY(15,0) が与えられます。ほかのすべての変数は属性 REAL FLOAT DECIMAL(6) が与えられます。
- DEFAULT(ANS) が有効な場合、どの変数にも属性 REAL FIXED BINARY(31,0) が与えられます。

スケール因数が精度属性に指定されていない場合、ほかのすべての属性に先行して、FIXED 属性が適用されます。したがって、BINARY (p,q) 属性を指定した宣言は、常に FIXED BINARY (p,q) 属性を指定した宣言と同じになります。

算術宣言で精度が指定されていない場合、 DEFAULT コンパイラー・オプションにより、表 29 に従って精度が決定されます。有効範囲、ストレージ、および位置合わせ属性に関する言語に固有のデフォルトは、 29 ページの表 8 と 28 ページの表 7 に示されています。

ENTRY 宣言で記述リストが指定されていない場合、引数の属性は、呼び出されたプロシージャー内のパラメーターの属性と一致しなければなりません。たとえば、下記の宣言の場合、

```
dcl X entry;
call X( 1 );
```

引数は属性 REAL FIXED DECIMAL(1,0) を持っています。次の例に示すように、プロシージャー x がそのパラメーターを別の属性で宣言した場合には、エラーが発生します。

```
X: proc( Y );
  dcl Y fixed bin(15);
```

入り口宣言によってすべてのパラメーターの属性を指定すれば、こうしたエラーの可能性を回避することができます。

表 29. デフォルトの算術精度

属性	DEFAULT(IBM)	DEFAULT(ANS)
DECIMAL FIXED	(5,0)	(10,0)
BINARY FIXED	(15,0)	(31,0)

表 29. デフォルトの算術精度 (続き)

属性	DEFAULT(IBM)	DEFAULT(ANS)
DECIMAL FLOAT	(6)	(6)
BINARY FLOAT	(21)	(21)

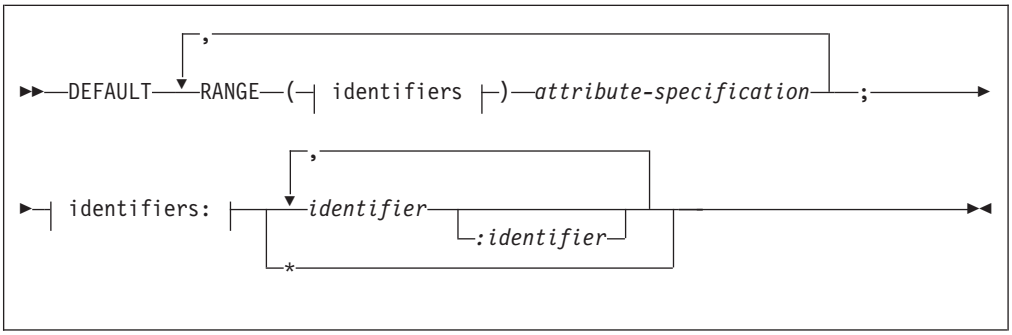
DEFAULT ステートメント

DEFAULT ステートメントは、(属性セットが完全ではない場合は) データ属性のデフォルトを指定します。部分的に完成された明示宣言やコンテキスト宣言、あるいは暗黙宣言の場合、 DEFAULT ステートメントが適用されない属性については、言語に固有のデフォルトが適用されます。

DEFAULT ステートメントはほかのすべての属性指定を無効にします。ただし、ENTRY 属性または FILE 属性を付けて、かつ VARIABLE 属性を暗黙指定する属性は付けずに宣言された名前については、 DEFAULT ステートメントの適用に先立ち、PL/I が暗黙の CONSTANT 属性を指定します。したがって、下記の例の場合は、PL/I によって Xtrn に CONSTANT 属性が指定され、STATIC 属性は指定されません。

```
Sample: proc;

  default range(*) static;
  dcl Xtrn entry;

end;
```

構造体および共用体のエレメントには、修飾されたエレメント名ではなく、エレメントの名前に応じてデフォルト属性が指定されます。 DEFAULT ステートメントは、構造体や共用体を作成するためには使用できません。



省略形: DFT

RANGE(identifier)

指定した ID と同じ文字で始まる名前にデフォルトが適用されるように指定します。 以下に例を示します。

```
RANGE (ABC)
```

上記の場合、下記の名前には適用されます。

```
ABC
ABCD
ABCDE
```

ただし、下記の名前には適用されません。

```
ABD
ACB
AB
A
```

したがって、範囲指定に 1 文字の ID を指定した場合は、その文字で始まるすべての名前に ID が適用されます。 RANGE の ID は DBCS で指定できます。

RANGE(identifier : identifier)

名前の最初の文字が指定した 2 つの ID と一致するか、またはアルファベット順で 2 つの ID の中間にある名前に、デフォルトが適用されるように指定します。文字は DBCS で指定できますが、 RANGE の指定が名前に適用されるかどうか判断するための比較基準はすべて、関係のある文字の 16 進値だけです。指定に使用する文字は、昇順のアルファベット順に並べる必要があります。 以下に例を示します。

```
RANGE(A:G,I:M,T:Z)
```

RANGE(*)

DEFAULT ステートメントの有効範囲内のすべての名前を指定します。 以下に例を示します。

```
DFT RANGE (*) PIC '99999';
```

このステートメントは、すべての名前について、デフォルト属性 REAL PICTURE '99999' を指定します。

範囲オプションを使用した*分配指定* の例を次に示します。

```
DEFAULT (RANGE(A)FIXED, RANGE(B)
        FLOAT)BINARY;
```

このステートメントは、最初の文字が A である名前にデフォルト属性 FIXED BINARY を指定し、最初の文字が B である名前に FLOAT BINARY を指定します。

DESCRIPTORS

明示的なエントリー宣言のパラメーター記述子リストにある、任意のパラメーター記述子に属性を組み込むように指定します。

- 同じクラスの代替属性が存在していても、この属性の組み込みは禁止されません。
- 少なくとも 1 つの属性がすでに存在している必要があります (DESCRIPTORS のデフォルト属性は、ヌル記述子には適用されません)。

以下に例を示します。

```
DEFAULT DESCRIPTORS BINARY;
DCL X ENTRY (FIXED, FLOAT);
```

リスト内のそれぞれのパラメーター記述子に属性 BINARY が追加され、次のように同等なリストが作成されます。

```
(FIXED BINARY, FLOAT BINARY)
```

attribute-list

特定の範囲内の名前に適用する属性を選択する際に使う、属性リストを指定します。リスト内の属性はどの順序で指定してもかまいませんが、相互にブランクで区切るようにしてください。

この属性リストからは、データ項目の宣言を完結させるために必要な属性だけが取り出されます。

FILE を使用した場合は、属性 VARIABLE と INTERNAL が暗黙指定されます。

次元属性は指定可能ですが、属性指定の最初の項目としてだけ指定できます。境界は算術定数または式として指定でき、REFER オプションを組み込むことができます。 以下に例を示します。

```
DFT RANGE(J) (5);
DFT RANGE(J) (5,5) FIXED;
```

DEFAULT ステートメントは、明示的に宣言されていない名前の次元属性を指定できますが、属性 BUILTIN を指定すると、添え字付きの名前がコンテキスト的に宣言されます。したがって、デフォルトでは次元属性は明示的に宣言された名前にだけ適用できます。

INITIAL 属性を指定できます。

データ項目に適用すると相互に対立する属性でも、属性指定では対立するとは限りません。 以下に例を示します。

```
DEFAULT RANGE(S) BINARY VARYING;
```

この場合、文字 S で始まり、BIT、CHARACTER、または GRAPHIC 属性を指定して明示的に宣言された名前には VARYING 属性が指定され、その他のすべての名前 (算術データ以外のものとして明示的にまたはコンテキストで宣言されていない) には BINARY 属性が指定されます。

VALUE

次元属性の前を除いて、attribute-specification の中のどこに書いてもかまいません。

VALUE は、区域のサイズ、ストリングの長さ、および数値精度に関するデフォルトの規則を確立します。

DEFAULT ステートメントで、VALUE オプションは、区域のサイズ、ストリングの長さ、または数値精度を指定できる場所にのみ置きます。

VALUE 文節の中のこれらのサイズ、長さ、および精度の指定は、システム・デフォルト属性の後で、サイズ、長さ、および精度に関するシステム・デフォルトより前に適用されます。したがって、たとえば DCL I; および DEFAULT RANGE(*) VALUE(FIXED BIN(31)); があるとした場合、変数 I はシステム・デフォルト属性である FIXED BIN を受け取りますが、精度 31 は (システム・デフォルトである 15 ではなく) VALUE オプションから受け取ります。

AREA データのサイズや、BIT、CHARACTER、または GRAPHIC データの長さは、式または整数で表すことができ、REFER オプションを使用することもできます。また、アスタリスクとして指定することもできます。

以下に例を示します。

```
DEFAULT RANGE(A:C)
        VALUE (FIXED DEC(10),
               FLOAT DEC(14),
               AREA(2000));
DECLARE B FIXED DECIMAL,
        C FLOAT DECIMAL,
        A AREA;
```

これらのステートメントは、下記のステートメントと同等です。

```
DECLARE B FIXED DECIMAL(10),
        C FLOAT DECIMAL(14),
        A AREA(2000);
```

value-specification では、特定の属性における精度指定であることがわかるように、基数属性とスケール属性を記入する必要があります。基数属性とスケール属性は属性配分することができます (162 ページの『属性分配』を参照のこと)。

VALUE オプションの影響を受ける属性は、区域のサイズ、ストリングの長さ、および精度だけです。上記の例での CHARACTER や FIXED BINARY など、オプションのその他の属性は、単に値を関連付ける属性を示すに過ぎません。次の例を考えてみてください。

```
DEFAULT RANGE(I) VALUE(FIXED DECIMAL(8,3));
I = 1;
```

I が明示的に宣言されていない場合、I には、言語で定められているデフォルトの属性 FIXED BINARY (15,0) が与えられます。 I は DEFAULT ステートメントの影響を受けません。なぜなら、このステートメントは、FIXED DECIMAL である名前のデフォルトの精度が (8,3) であることだけを指定しているからです。

以下に例を示します。

```
DFT RANGE(*) VALUE(FIXED BINARY(31));
```

一方、次の例は、デフォルトの属性として FIXED BINARY を指定し、同時に精度を指定しています。

```
DFT RANGE(*) FIXED BINARY VALUE(FIXED BINARY(31));
```

1 つのブロック内に複数の DEFAULT ステートメントを指定することができます。DEFAULT ステートメントの有効範囲は、そのステートメントが指定されているブロック、および、そのブロック内で、同じ範囲の別の DEFAULT ステートメントを含まず、なおかつ同じ範囲の DEFAULT ステートメントを持つブロックに包含されていないすべてのブロックです。

内部ブロックの DEFAULT ステートメントは、明示的に宣言された名前に対してのみ効果があります。これは、暗黙宣言の場合、その名前が使われている外部プロシージャーの PROCEDURE ステートメントのすぐ後にある DECLARE ステートメントでその名前が宣言されているかのように、宣言の有効範囲が決められるからです。

あるブロック内の DEFAULT ステートメントで指定した範囲が、そのブロックに含まれているブロック内の DEFAULT ステートメントで指定した範囲と部分的に重なり合ってもかまいません。このような場合、包含ブロック内の DEFAULT ステート

メントの範囲は、被包含ブロック内の DEFAULT ステートメントの範囲分だけ減少することになります。 以下に例を示します。

```
 P:  PROCEDURE;
L1:  DEFAULT RANGE (XY) FIXED;
 Q:  BEGIN;
L2:  DEFAULT RANGE (XYZ) FLOAT;
     END P;
```

DEFAULT ステートメント L1 の有効範囲は、プロシージャー P および包含ブロック Q です。DEFAULT ステートメント L1 の範囲は、文字 XY で始まるプロシージャー P 内のすべての名前、および文字 XY で始まる開始ブロック Q 内のすべての名前です。ただし、文字 XYZ で始まる名前は除きます。

DEFAULT ステートメントの接頭部としてラベルを指定できます。このようなラベルへの分岐は、ヌル・ステートメントへの分岐として扱われます。条件接頭語は、DEFAULT ステートメントに付加できません。

言語に固有のデフォルトの復元

次のステートメントを例に考えてみましょう。

```
dft range(*) system;
```

上記の場合、すべての名前について、包含ブロック内で確立されたプログラマー定義のデフォルト規則より優先されます。これを使用すれば、被包含ブロックのために言語固有のデフォルトを復元することができます。

配列

配列 は同一の属性を持つエレメントからなる n 次元の集合です。 配列そのものにだけ名前を付けます。配列の個々の項目を参照するときは、配列内でのその項目の位置を使用します。次元属性を提供することによって、名前が*配列変数* であることを指定できます。

REFER を使用して指定しない限り、すべての配列のすべての次元に、少なくとも 1 つのエレメントが必要です。 REFER を使用して配列の境界を指定する場合、以下の条件を満たす限り、配列のエレメント数をゼロに定義することができます。

- 配列はアクセスされたり、割り当てられたりすることがない
- 配列に 1 つの次元しかない (継承した次元を除く)

そのため、例えば以下のコードでは、ab3、abc1、および abc2 がアクセスされたり割り当てられたりしない限り、n1 がゼロであるときに配列 a を割り振るのは有効です。

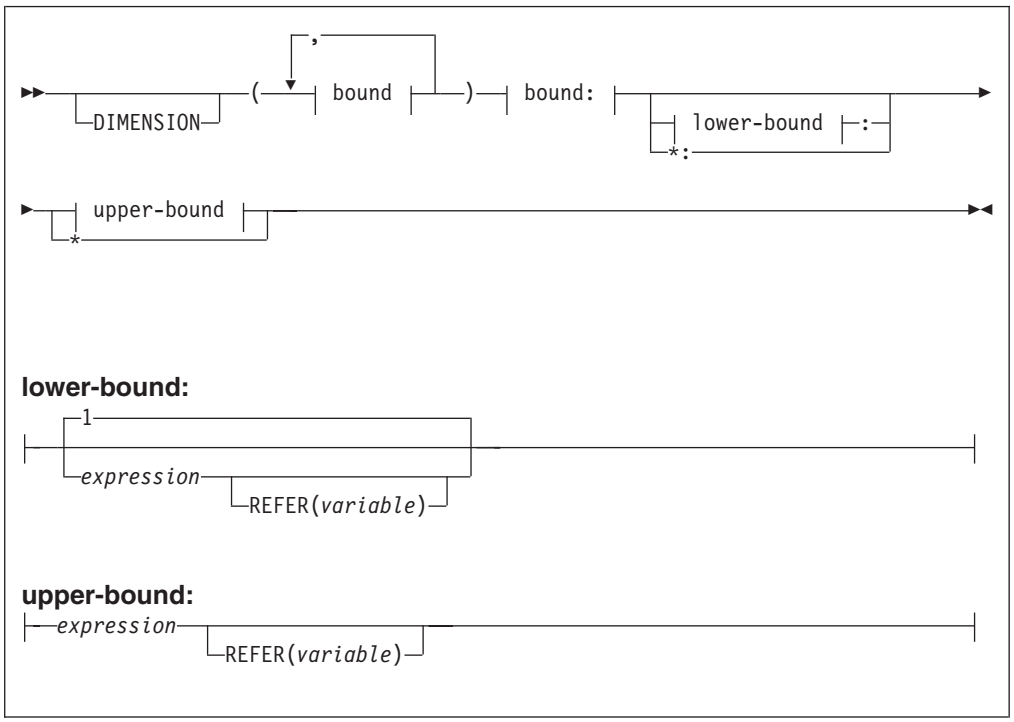
```
dcl n1          fixed bin(31);
dcl p           pointer;
dcl
  1 a based(p),
    2 ab1       fixed bin(31),
    2 ab2       fixed bin(31),
    2 ab3( n1 refer(ab2) ),
      3 abc1    char(40) var,
      3 abc2    char(40) var,
    2 ab4       char(40) var;
```

DIMENSION 属性

次元属性 には、配列の次元の数、各次元の上限と下限を指定します。

境界 は制限なしの式で表され、配列にストレージが割り振られるときに計算され、FIXED BINARY(31,0) に変換されます。

エクステント とは、上限値と下限値を含め、それらの値の間にある整数の個数です。



省略形: DIM

DIMENSION キーワードが省略される場合には、次元は宣言内の名前 (もしくは括弧で囲まれた名前のリスト) のすぐあとになければなりません。

境界 (bound) 指定の数が、その配列の次元の数を表します。ただし、宣言される変数が構造体または共用体の配列に含まれている場合は例外で、そのときはその変数を含んでいる構造体または共用体から次元の数を継承します。

境界の指定は、境界を次のようにして示します。

- 上限だけを指定すると、下限にはデフォルトの 1 が設定されます。
- 下限は上限以下でなければなりません。
- アスタリスク (*) は、上限または下限 (またはその両方) がそのパラメーターに関連する引数によって決定されることを表します。

配列の例

次の宣言について考えてみましょう。

```
declare List fixed decimal(3) dimension(8);
```

List は、8 つのエレメントからなる 1 次元の配列であり、それぞれ 3 桁の 10 進固定小数点エレメントであると宣言されています。 List の 1 つの次元は、境界値として 1 と 8 を持ち、エクステントは 8 です。

以下に例を示します。

```
declare Table (4,2) fixed dec (3);
```

Table は、8 つの 10 進固定小数点エレメントからなる 2 次元の配列であると宣言されています。 Table の 2 つの次元は、1 と 4 および 1 と 2 の境界値を持ち、エクステントは 4 と 2 です。

その他の例を示します。

```
declare List_A dimension(4:11);
declare List_B (-4:3);
```

最初の例では、境界は 4 と 11 で、2 番目の例では -4 と 3 です。どちらの場合も下限から上限までに整数が 8 つあるので、両者のエクステントは同じです。

複数の配列を使って配列データを処理する場合は (74 ページの『配列式』を参照)、単にエクステントだけでなく、両者の境界が同じでなければなりません。前述の List、 List_A、 List_B は、すべてエクステントは同じですが、境界が異なっています。

添え字

配列エレメントの参照方法は、配列の上限と下限によって決まります。たとえば、次のようなデータ項目があるとします。

```
20 5 10 30 630 150 310 70
```

上記データ項目を、前述で宣言した List という配列に割り当てたとします。さまざまなエレメントが、次のように参照されます。

参照	エレメント
LIST (1)	20
LIST (2)	5
LIST (3)	10
LIST (4)	30
LIST (5)	630
LIST (6)	150
LIST (7)	310
LIST (8)	70

LIST のあとに書かれている括弧内の数字を*添え字* といいます。 添え字を括弧で囲んで配列名のあとに書くと、その配列内の特定のデータ項目を指すことになります。 LIST(4) のように、添え字付きの名前を参照すると、1 つのエレメントを参照することになります。これは要素変数です。添え字が付いていない配列名 (たとえば、LIST) を書けば、配列全体を参照することができます。

前述で宣言した List_A と List_B に同じデータを割り当てることができます。この場合は、次のように参照します。

参照	エレメント	参照
LIST_A (4)	20	LIST_B (-4)
LIST_A (5)	5	LIST_B (-3)
LIST_A (6)	10	LIST_B (-2)
LIST_A (7)	30	LIST_B (-1)
LIST_A (8)	630	LIST_B (0)
LIST_A (9)	150	LIST_B (1)
LIST_A (10)	310	LIST_B (2)
LIST_A (11)	70	LIST_B (3)

同じデータを TABLE (2 次元の配列として宣言されている) に割り当てたとします。 TABLE は、次のように 4 行 2 列の行列として表せます。

TABLE(m,n)	(m,1)	(m,2)
(1,n)	20	5
(2,n)	10	30
(3,n)	630	150
(4,n)	310	70

TABLE のエレメントを参照するときは、コンマで区切った 2 つの添え字を括弧で囲んで、配列名のあとに書きます。たとえば、TABLE(2,1) は、2 行目の最初の項目 (この例では 10 というデータ項目) を指します。

TABLE を図示するのに行列を使用したのは、概念的にその方がわかりやすいためで、実際にはデータ項目はストレージでそのように編成されているわけではありません。データ項目は、行を主体とした順序で割り当てられます。つまり列を表す添え字がまず変化するということを意味します。たとえば、TABLE への割り当て順序は、TABLE(1,1)、TABLE(1,2)、TABLE(2,1)、TABLE(2,2) というようになります。

添え字を使用して配列を参照するときは、添え字の数は配列の次元の数と同じでなければなりません。

有効な算術値が得られる式であれば、添え字として使うことができます。式の値は、必要に応じて FIXED BINARY(31,0) に変換されます。たとえば、TABLE(I,J*K) を使用し、I、J、K の値を変えていけば、TABLE の個々のエレメントを参照できます。

配列のクロスセクション

添え字の代わりにアスタリスクを使えば、配列のクロスセクションを参照できます。アスタリスクは、そのエクステント全体を使用することを意味します。たとえば、TABLE(*,1) は TABLE の第 1 列のすべてのエレメントを指します。つまり、TABLE(1,1)、TABLE(2,1)、TABLE(3,1)、および TABLE(4,1) からなるクロスセクションを表します。添え字付きの名前 TABLE(2,*) は TABLE の第 2 行のすべてのデータ項目を指します。 TABLE(*,*) は、TABLE と同様に、配列全体を指します。

添え字としてアスタリスクが付いている名前は、1 つのデータ・エレメントを表すのではなく、アスタリスクと同数の次元を持つ配列を表します。したがって、そのような名前は要素式ではなく配列式です。

配列のクロスセクションを参照すると、ストレージで隣接していない複数のエレメントを参照することがあります。そのようなクロスセクションによって表されるストレージを、*非連結* ストレージといいます (263 ページの『CONNECTED 属性と NONCONNECTED 属性』を参照)。次のような規則があります。左端の境界をアスタリスクにし、その右側の境界をアスタリスク以外のものにすると、配列のクロスセクションは非連結ストレージになります。たとえば、A(4,*,*) は連結ストレージになりますが、A(*,2,*) はなりません。

構造体

構造体 とは、構造体、共用体、要素変数、および配列であるメンバー・エレメントの集合です。

構造変数 は、データの集合全体を参照するときに使用される名前です。ただし配列とは異なり、構造体内の各メンバーにも名前を付けられます。また、各メンバーの属性を別々にすることもできます。参照されることがなければ、構造体やメンバーの名前としてアスタリスクを使用できます。たとえば、予約項目や埋め込み文字項目を、アスタリスクで指定することができます。

構造体は、いくつかの*レベル* を持ちます。 レベル 1 の名前は、*大構造体* と呼ばれます。それより下のレベルの名前は、*小構造体または共用体* になります。最下位のレベルの名前は、*基本* 名と呼ばれ、要素変数または配列変数を表すことができます。共用体の説明については、 186 ページの『共用体』の項を参照してください。

構造体は、DECLARE ステートメントで、それぞれの名前の前にレベル番号を付けるという方法によって指定します。レベル番号は、整数でなければなりません。

大構造体名は、レベル番号 1 を付けて宣言します。小構造体名、共用体、および基本名は、2 以上のレベル番号を付けて宣言します。区切り文字によって、レベル番号と関連する名前を区切らなければなりません。たとえば、給与計算レコードの項目を次のように宣言することができます。

```
declare 1 Payroll,                    /*  major structure name */
          2 Name,                     /*  minor structure name */
            3 Last char(20),          /*  elementary name      */
            3 First char(15),
          2 Hours,
            3 Regular fixed dec(5,2),
            3 Overtime fixed dec(5,2),
          2 Rate,
            3 Regular fixed dec(3,2),
            3 Overtime fixed dec(3,2);
```

上の例では、Payroll は大構造体で、それ以外の名前はこの構造体のメンバーとなります。 Name、Hours、および Rate は小構造体で、それ以外のすべてのメンバーは、要素変数となります。構造体全体は、名前 Payroll を使用すれば参照することができ、構造体内の特定の部分は、その小構造体名を使用すれば参照できます。メンバーは、メンバー名の参照によって参照することができます。

上の例では、読みやすくするために字下げをしています。このステートメントを、次のストリングのように続けて書くこともできます。

```
Declare 1 Payroll, 2 Name, 3 Last char(20), . . .
```

後続の下位レベルに付けるレベル番号は、連続している必要はありません。レベル n の小構造体には、その小構造体名と n 以下のレベル番号を持つ次の名前との間にある、n より大きいレベル番号を持つすべての名前が含まれます。

たとえば、次の宣言では、先の例の宣言とまったく同じ構造体になります。

```
Declare 1 Payroll,
          4 Name,
            5 Last char(20),
            5 First char(15),
          3 Hours,
            6 Regular fixed dec(5,2),
            5 Overtime fixed dec(5,2),
          2 Rate,
            9 Regular fixed dec(3,2),
            9 Overtime fixed dec(3,2);
```

大構造体の記述は、通常、DECLARE ステートメントの終わりのセミコロンで終了します。また、コンマで終了する場合もあり、この場合はあとに別の項目の宣言が続きます。

共用体

共用体 とは、相互にオーバーレイし合い、同じストレージを有するメンバー・エレメントの集合ですメンバーは、構造体、共用体、要素変数、および配列であることができます。それらは同じ属性を持つ必要はありません。

共用体全体には、データの集合全体で参照できる名前が付けられます。構造体と同様、共用体の各エレメントもそれぞれ名前が付けられます。アスタリスクは、共用体の名前、または共用体が参照されない場合はメンバーの名前として使用できます。たとえば、予約項目や充てん文字項目をアスタリスクで指定することができます。

構造体と同様、共用体はどのようなレベル (レベル 1 を含む) にもなります。次に低いレベルの共用体のすべてのエレメントは、共用体のメンバーであり、同じストレージを有します。共用体によって占められるストレージは、最大メンバーに必要なストレージと等しくなります。通常、どのような場合でも使用されるメンバーは 1 つだけで、プログラマーが使用されるメンバーを決定します。

共用体は、構造体と同様、それぞれの名前の前にレベル番号を付けるという方法によって指定します。

共用体は、一般的に共通部、選択部、および可変部からなる可変レコードの宣言に使用されます。たとえば、クライアント・ファイルのレコードは、次のように宣言されます。

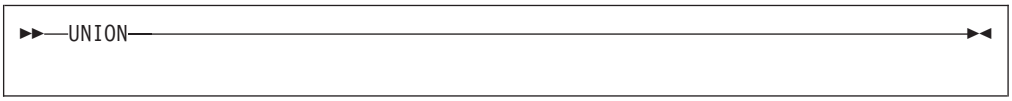
```
Declare 1 Client,
          2 Number pic '999999',
          2 Type bit(1),
          2 * bit(7),
          2 Name union,
            3 Individual,
              5 Last_Name char(20),
              5 First_Name union,
```

```
      7 First   char(15),
      7 Initial char(1),
  3 Company char(35),
2 * char(0);
```

この例では、Client が大構造体です。構造体 Individual とエレメント Company は、共用体 Name のメンバーです。これらメンバーのうちの 1 つが、Type に応じてアクティブになります。構造体 Individual には、共用体 First_name とエレメント Last_name が含まれています。共用体 First_name は、 First と Initial をメンバーとして持ち、両方ともアクティブです。またこの例では、アスタリスクを名前として使用しています。 共用体の記述は、DECLARE ステートメントを終了するセミコロンまでで終了するか、コンマによって終了します (コンマの場合、別の項目の宣言があとに続きます)。

UNION 属性

UNION 属性を使用すると、ある変数が共用体であり、そのメンバーが変数のあとに続き、論理的に次に高位のレベルであることを指定することができます。 CELL は、UNION の同義語として受け入れられます。



構造体/共用体の修飾

構造体のメンバーまたは共用体のメンバーは、その名前が固有であれば、名前だけで参照できます。別のメンバーが同じ名前を持つ場合には、そのレベルが同じか、異なるのか、またはあいまいになります。あいまいになる場合、正しいメンバーを固有に識別するには修飾された参照を行わなければなりません。

修飾された参照 とは、ピリオドによって接続された親メンバーにあたる 1 つ以上の名前で修飾されているメンバー名のことです (修飾された参照の構文については、55 ページの『第 4 章 式および参照』を参照)。ピリオドの前後には、ブランクがあってもかまいません。

修飾は、レベルの順番に従わなければなりません。つまり、最高位レベルの名前は最初に、最低位レベルの名前は最後になければなりません。

レベル 1 の構造体名または共用体名は、ブロックの有効範囲内で固有でなければなりませんが、メンバー名は、直接の親のもとで同じ論理レベルに現れない限り、固有である必要はありません。修飾名は、それが現れるブロック内で、同じ構造体で固有に参照される必要がある場合にしか使用することができません。次の例では、x.y (19) の値は表示されますが、値 (17) は表示されません。

```
dcl Y fixed init(17);


  begin;
dcl
          1 X,
2 Y fixed init(19);
display( Y );
  end;
```

参照の実行は、通常、参照を含む一番内側にあるブロック内で宣言された名前に適用されます。

以下の例では、あいまいな参照とあいまいでない参照の両方を説明しています。以下の例では、 A.C は内側のブロックの C を参照しています。 D.E は外側のブロックの E を参照しています。

```
declare 1 A, 2 C, 2 D, 3 E;
   begin;
   declare 1 A, 2 B, 3 C, 3 E;
A.C = D.E;
```

次の例では、D は 2 回宣言されています。 A.D の参照を行うと 2 番目の D が参照されます。これは、A.D が、2 番目の D のみの完全修飾だからです。最初の D は、A.C.D として参照されます。

```
declare 1 A,
          2 B,
          2 C,
            3 D,
          2 D;
```

次の例では、A.C の参照はあいまいです。これは、C がこの参照によって完全に修飾されないためです。

```
declare 1 A,
          2 B,
            3 C,
          2 D,
            3 C;
```

次の例では、A の参照を行うと最初の A が参照され、 A.A の参照を行うと 2 番目の A が参照され、 A.A.A の参照を行うと 3 番目の A が参照されます。

```
declare 1 A,
          2 A,
            3 A;
```

次の例では、X の参照を行うと最初の DECLARE ステートメントが参照されます。Y.Z の参照はあいまいなものとなります。 Y.Y.Z は 2 番目の Z を参照し、 Y.X.Z は最初の Z を参照します。

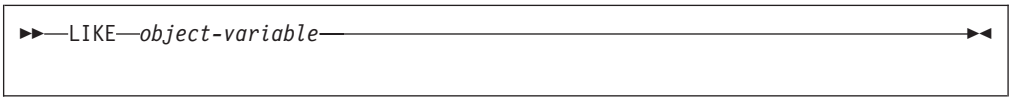
```
declare X;
declare 1 Y,
          2 X,
            3 Z,
            3 A,
          2 Y,
            3 Z,
            3 A;
```

名前の修飾については、 163 ページの『宣言の有効範囲』を参照してください。

LIKE 属性

LIKE 属性は、宣言される名前が、参照される構造体または共用体 (LIKE 属性のオブジェクト) と論理的に同じ編成であることを示します。オブジェクト変数のメンバー名およびその属性 (次元属性を含む) は、有効にコピーされ、宣言される名前のメンバーになります。必要であれば、コピーされたメンバーのレベル番号は自動的

に調整されます。オブジェクト変数名およびその属性 (次元属性を含む) は無視されます。



object-variable
大構造体、小構造体、または共用体のいずれでもかまいません。 LIKE 属性の指定を含むブロック内になければなりません。修飾をすることはできますが、添え字を付けることはできません。オブジェクトまたはそのすべてのメンバーは、LIKE 属性もしくは REFER オプションを持つことはできません。

すべての LIKE 属性のオブジェクトは、その LIKE 属性が拡張される前に宣言された名前に対応しています。

新しいメンバーを、作成された構造または共用体に追加することはできません。LIKE 属性のオブジェクト変数のすぐあとのレベル番号は、 LIKE 属性を持つ名前のレベル番号以下でなければなりません。

次の宣言では、X に対し同じ構造体が生成されます。

```
dcl
  1 A(10) aligned static,
    2 B    bit(4),
    2 C    bit(4),
  1 X like A;


dcl
  1 X,
    2 B bit(4),
    2 C bit(4);
```

次元 (DIM(10))、ALIGNED、および STATIC 属性は、 LIKE 拡張の一部としてコピーされないことに注意してください。

LIKE 属性は、デフォルトが適用される前、もしくは ALIGNED 属性および UNALIGNED 属性が LIKE オブジェクト変数に含まれるエレメントに適用される前に拡張されます。

例

```
declare 1 A,
          2 C,
            3 E(3) union,
              5 E1,
              5 E2,
            3 F;
declare 1 B(10) union,
          2 C, 3 G, 3 H,
          2 D;
begin;
declare 1 C like B;
declare 1 D(2),
          5 BB like A.C;
end;
```

宣言 C および D の結果を、次の例に示します。

```
dcl
    1 C,               /* DIM and UNION not copied. */
      2 C, 3 G, 3 H,
      2 D;


dcl 1 D(2),
      5 BB,
        6 E(3) union,  /* DIM(3) and UNION copied.      */
          7 E1,        /* Note adjusted level-numbers.  */
          7 E2,
        6 F;
```

次の例は、C.E が LIKE 属性を持つため無効です。

```
declare 1 A like C,
        1 B,
          2 C,
            3 D,
            3 E like X,
          2 F,
        1 X,
          2 Y,
          2 Z;
```

次の例は、A の LIKE 属性が、 LIKE 属性によって宣言された構造体 G の副構造体 G.C を指定するため無効です。

```
declare 1 A like G.C,
        1 B,
          2 C,
            3 D,
            3 E,
          2 F,
        1 G like B;
```

次の例は、A の LIKE 属性が、LIKE 属性を持つ副構造体 F を含む構造体 B 内の構造体 C を指定するため無効です。

```
declare 1 A like C,
        1 B,
          2 C,
            3 D,
            3 E,
          2 F like X,
        1 X,
          2 Y,
          2 Z;
```

NOINIT 属性

NOINIT 属性は、どの INITIAL 属性も無視されることを指定します。

NOINIT 属性は、レベル 1 構造体でのみ有効です。

NOINIT 属性は特に LIKE 属性に便利です。これは、新しい変数が古い変数の LIKE (類似) として宣言されたが NOINIT 属性も持つ場合、新しい変数は、古い変数からすべてのサブ構造を継承しますが、その INITIAL 値は継承しないからです。

集合体の組み合わせおよびマッピング

配列、構造体、および共用体の組み合わせ

構造体または共用体に次元属性を指定すると、それぞれ*構造体の配列*、または*共用体の配列* になります。このような配列のエレメントは、同一の名前、レベル、およびメンバーを持つ構造体または共用体です。たとえば、ある構造体が 20 世紀から 21 世紀までの毎月の気象データを保管するために使用されている場合、これは次のように宣言されます。

```
Declare 1 Year(1901:2100),
          3 Month(12),
            5 Temperature,
              7 High decimal fixed(4,1),
              7 Low decimal fixed(4,1),
            5 Wind_velocity,
              7 High decimal fixed(3),
              7 Low decimal fixed(3),
           5 Precipitation,
             7 Total decimal fixed(3,1),
             7 Average decimal fixed(3,1),
         3 * char(0);
```

1991 年 7 月の気象データは、Year(1991,7) と指定すれば参照することができます。 7 月の気象データの各部分は、 Temperature(1991,7) や Wind_Velocity(1991,7) を指定すれば参照することができます。 Precipitation.Total(1991,7) または Total(1991,7) のいずれを指定しても、1991 年 7 月の総雨量を参照することができます。

Temperature.High(1991,3) は 1991 年 3 月の最高気温を参照します。これは添え字付きの修飾された参照です。

構造体または共用体の配列の中に、配列であるメンバーが含まれていると、添え字付きの修飾された参照の必要性が明確になります。次の例では、A および B は共に構造体です。

```
declare 1 A (2,2),
          (2 B (2),
             3 C,
             3 D,
           2 E) fixed bin;
```

データ項目を参照するために、3 つの名前と 3 つの添え字が必要になることがあります。 以下に例を示します。

A(1,1).B　　は構造体の配列 B を参照します。
A(1,1)　　は構造体を参照します。
A(1,1).B(1)　　は構造体を参照します。
A(1,1).B(2).C　　はエレメントを参照します。

このような参照における添え字は、添え字の順序さえ変えなければ、高位レベルの名前に付けても低位レベルの名前に付けてもかまいません。たとえば、上に示した構造体の配列の場合、 A.B.C(1,1,2) と A(1,1,2).B.C は、 A(1,1).B(2).C と同じ意味になります。すべての添え字を最下位レベルに移したのでなければ、その参照には*介在添え字* があるといいます。そこで、たとえば A.B(1,1,2).C には介在添え字があります。

構造体または共用体の配列の中で宣言されている項目は、親のもとで宣言されている次元を継承します。たとえば、上に示した構造体の配列 A では、配列 B は、A で宣言されている 2 つの次元を継承するので、3 次元の構造体になります。 B が固有の名前で、修飾を付ける必要がない場合、特定の B を参照するには 3 つの添え字が必要です。このうちの 2 つは特定の A を識別し、残りの 1 つはその A の中の特定の B を識別するためのものです。

構造体または共用体の配列のクロスセクション

構造体または共用体の配列のクロスセクションを参照することはできません。つまり、すべての添え字がアスタリスクでない限り、参照するときにアスタリスク表記を使用することはできません。

構造体および共用体の操作

構造体は、要素変数を参照することができるほとんどのコンテキストで参照することができます。たとえば、割り当て、入出力ステートメント、その他で構造体参照を行うことができます。ただし、共用体への参照、または共用体を含む構造体への参照は、次のものに制限されます。

- パラメーターおよび引数
- 組み込み関数とサブルーチンのうちストレージ制御や構造体を使用することができるもの

構造体および共用体のマッピング

共用体の各メンバーは、構造体のメンバーと同じ方法でマップされます。つまり、それぞれのメンバー (共用体でない場合) は、構造体のメンバーの場合と同様にマップされます。これによって、共用体の各メンバーの最初の記憶場所は、各メンバーがそれぞれ異なる位置合わせを必要とし、さらに (それによって) メンバーの開始位置より前にそれぞれ異なる埋め込みを必要とする場合は、互いにオーバーレイすることはありません。

次の共用体について考えてみましょう。

```
dcl
  1 A union,
    2 B,
      3 C char(1),
      3 D fixed bin(31),
    2 E,
      3 F char(2),
      3 G fixed bin(31);
```

3 バイトの埋め込みが、A および B の間に追加されます。 A と E の間には 2 バイト追加されます。

共用体の各メンバーの先頭の記憶位置を同じにするには、各先頭メンバーがそれぞれ同じ位置合わせ要件を持ち、さらにメンバー (またはそのメンバーのメンバー) 内での最高位の位置合わせと同じになるようにしてください。

以降の説明は、小構造体または要素変数のいずれかである構造体または共用体のメンバーに適用されます。

どの構造 (大構造体または小構造体) でも、その長さ、位置合わせの要件、および 8 バイト境界からの相対的な位置は、構造体のメンバーの長さ、位置合わせの要件、および相対位置によって決まります。各レベルおよび構造体全体でこれらの要件を決定するプロセスを、*構造体のマッピング* といいます。

レコード単位の入出力を使用するとき、ある構造体にどのくらいの長さのレコードが必要であるかを判別します。また、位置指定モードの入出力では、構造体の位置合わせを正しく行うために必要な埋め込みまたは再配置の量を判別します。このような場合、構造体のマッピングを通して判別することができます。

構造体のマッピングのプロセスは、構造体のメンバー相互間に生じる使用されないストレージ (埋め込み) の量を最小にします。これは、以下に説明する規則に (事実上) 従ってこの処理全体を完了させてから、構造体にストレージを割り振ります。

構造体のマッピングは、物理的な処理ではありません。「シフトする」や「*相対位置変更する*」などの用語は、単に説明の便宜で使用するもので、実際にストレージ内で移動が行われるわけではありません。構造体にストレージが割り振られるとき、マッピング処理の結果として相対位置がすでに知られています。

構造体全体のマッピングでは、項目 (エレメントまたはすでに個々のマッピングが判別されている小構造体) が順に対として組み合わされていきます。 1 つの対が作られると、それは 1 つの単位となり、別の単位と組み合わされます。構造体全体のマッピングが終わるまで、この操作が続けられます。したがって、このプロセスの規則は、次の 2 種類に分類されます。

- 対を作る順序を決めるための規則
- 1 つの対のマッピングを行うための規則

以下にこれらの規則を説明し、これらの規則の適用例を示して詳しく説明します。構造体エレメントの*論理レベル* と*レベル番号* の違いを理解する必要があります。構造体の宣言が一定のレベル番号で行われたり適切な字下げをして行われていれば (規則の説明のあとに詳しい例示があります)、論理レベルについてはすぐに明らかになります。いずれにしても、構造体宣言の始まりから順番に各項目に下記の規則を適用していけば、その構造体の各項目の論理レベルを判別することができます。

注: ある項目の論理レベルは、その項目を直接に含んでいる構造体の論理レベルよりも、常に 1 単位だけ低くなるという規則です。

次の例では、下の行の数字は、宣言内の各項目の論理レベルを示しています。

```
dcl 1 A, 4 B, 5 C, 5 D, 3 E, 8 F, 7 G;

    1   2   3   3   2   3   3
```

対を作る順序の規則

対を作る順序を決める際の手順は、次のとおりです。

1. 最も低い論理レベル (論理レベル n と呼ぶことにする) にある小構造体を見つけます。
2. 論理レベル n の小構造体が複数あるときは、宣言内で最初に書いたものをとります。

3. この小構造体内の最初の 2 つのエレメントを対にして、1 つの単位を形成します。このとき、1 つの対をマッピングするための規則を使用します (『1 つの対のマッピングを行うための規則』を参照)。
4. この単位と、この小構造体で宣言されている次のエレメント (ある場合) とを対にして、より大きい単位を形成します。
5. この小構造体の全エレメントが 1 つの単位に組み合わされるまで、ステップ 4 を繰り返します。これで、この小構造体のマッピングは完了です。あらゆる埋め込みを含め、その位置合わせの要求と長さが決定され、構造体宣言を変更しなければ変更されません。ダブルワード境界からのオフセットも、ここで決められます。このオフセットは、含まれている構造体をマッピングするときに有効です。また、このようなマッピングの結果として変わることがあります。
6. 宣言内の論理レベル n にある次の小構造体 (ある場合) について、ステップ 3 からステップ 5 を繰り返します。
7. 論理レベル n のすべての小構造体のマッピングが終わるまで、ステップ 6 を繰り返します。これらの小構造体のおのおのは、構造体のマッピングを行う上で 1 つのエレメントと考えることができます。
8. 次の高い論理レベルで小構造体のためにペア化処理を繰り返します。すなわち n を (n-1) に等しくし、ステップ 2 から 7 まで繰り返します。
9. n = 1 までステップ 8 を繰り返します。その後ステップ 3 から 5 を大構造体について繰り返します。

1 つの対のマッピングを行うための規則

ここでは説明の便宜上、ストレージをいくつかの連続したダブルワードと考えます。各ダブルワードは 8 バイトで、0 から 7 までの番号が付いているものとし、この番号でダブルワード境界からのオフセットを表します。さらに、構造体の始まりからの長さやオフセットを算出できるように、各バイトには、任意のバイトから始めて 0 から順に連続した番号が付いているものとします。

1. ダブルワード境界の対の最初のエレメントを開始するか、またはエレメントがすでにマップされている小構造体の場合は、示された量によってダブルワード境界からその分を相対位置変更します。
2. この 1 対の 2 番目のエレメントを、最初のエレメントの終わりの後に続く最初の有効な位置から始まるように合わせます。この位置は、2 番目のエレメントの位置合わせ要件に左右されます。 (2 番目のエレメントが小構造体である場合、その位置合わせ要件はすでに決定されています。)
3. 最初のエレメントの位置合わせ要件が満たされる限り、最初のエレメントを 2 番目のエレメントの方向にシフトします。どれだけシフトしたかによって、この 1 対がダブルワード境界からどれだけ離れているか (オフセット) が決まります。

この作業が完了すると、この 2 つのエレメントの間の埋め込みは最少量になっており、後続の作業中に変わることはありません。対は固定長の単位および位置合わせ要件となります。その長さは 2 つの長さの合計と埋め込みであり、その位置合わせ要件は 2 つの位置合わせ要件の高い方 (それらが異なる場合) です。

UNALIGNED 属性による影響

以下に示す構造体のマッピングの例は、ALIGNED と宣言されている構造体に適用される規則を示したものです。 ALIGNED と宣言された構造体のマッピングは、多くの位置合わせ要件があるため、より複雑です。 UNALIGNED 属性を指定すると、ハーフワード、フルワード、およびダブルワードの位置合わせ要件が 1 バイトに引き下げられ、ビット・ストリングの位置合わせ要件が 1 ビットに引き下げられます。構造体のマッピングについては同じ規則が適用されますが、引き下げられた位置合わせ要件が使われます。使われないストレージが生じるのは、構造体にビット・ストリングが含まれていて、バイト内でビットの埋め込みが行われたときだけです。

AREA データは、位置合わせなしにすることはできません。

構造体に UNALIGNED 属性があり、位置合わせできないエレメントがある場合は、 UNALIGNED はそのエレメントについては無視されます。エレメントが位置合わせされ、エラー・メッセージが出されます。たとえば、プログラムで次のように宣言した場合、 C には ALIGNED 属性が与えられます (継承される属性 UNALIGNED は AREA と矛盾するからです)。

```
declare 1 A unaligned,
          2 B,
          2 C area(100);
```

構造体のマッピング例

次の例では、例のように宣言された構造体に、構造体のマッピングの規則がどのように適用されるかを示します。

```
declare 1 A aligned,
          2 B fixed bin(31),
          2 C,
            3 D float decimal(14),
            3 E,
              4 F entry,
              4 G,
                5 H character(2),
                5 I float decimal(13),
              4 J fixed binary(31,0),
            3 K character(2),
            3 L fixed binary(20,0),
          2 M,
            3 N,
              4 P fixed binary(15),
              4 Q character(5),
              4 R float decimal(2),
            3 S,
              4 T float decimal(15),
              4 U bit(3),
              4 V char(1),
            3 W fixed bin(31),
          2 X picture '$9V99';
```

最も低い論理レベルにある小構造体は G です。したがって、G が最初にマッピングされます。次に、E がマッピングされ、N、S、 C、M がこの順序でマッピングされます。

それぞれの小構造体ごとに、プロセスのステップを表 図 8 に示し、そのプロセスを視覚的に捕えた図を 197 ページの図 9 に示します。最後に、199 ページの図 15 に示すように、大構造体 A がマッピングされます。

この例の終わりには、各メンバーが A の始まりからどのくらい離れているか (オフセット) を示す表の形で、 A の構造体マップを示します (200 ページの図 16)。

	エレメントの名前	位置合わせの要件	長さ	ダブルワードのオフセット 開始	ダブルワードのオフセット 終了	埋め込みの長さ	小構造体からのオフセット
ステップ 1	H	バイト	2	0	1		
	I	ダブルワード	8	0	7		
ステップ 2	*H	バイト	2	6	7		0
	I	ダブルワード	8	0	7	0	2
小構造体	G	ダブルワード	10	6	7		
ステップ 1	F	フルワード	8	0	7		
	G	ダブルワード	10	6	7		
ステップ 2	*F	フルワード	8	4	3		0
	G	ダブルワード	10	6	7	2	10
ステップ 3	F & G	ダブルワード	20	4	7		
	J	フルワード	4	0	3	0	20
小構造体	E	ダブルワード	24	4	3		
ステップ 1	P	ハーフワード	2	0	1		0
	Q	バイト	5	2	6		2
ステップ 2	P & Q	ハーフワード	7	0	6		
	R	フルワード	4	0	3	1	8
小構造体	N	フルワード	12	0	3		
ステップ 1	T	ダブルワード	8	0	7		0
	U	バイト	1	0	0	0	8
ステップ 2	T & U	ダブルワード	9	0	0		
	V	バイト	1	1	1	0	9
小構造体	S	ダブルワード	10	0	1		
ステップ 1	D	ダブルワード	8	0	7		0
	E	ダブルワード	24	4	3	4	12
ステップ 2	D . E	ダブルワード	36	0	3		
	K	バイト	2	4	5	0	36
ステップ 3	D, E, & K	ダブルワード	38	0	5		
	L	フルワード	4	0	3	2	40
小構造体	C	ダブルワード	44	0	3		
ステップ 1	N	フルワード	12	0	3		
	S	ダブルワード	10	0	1		
ステップ 2	*N	フルワード	12	4	7		0
	S	ダブルワード	10	0	1	0	12
ステップ 3	N & S	ダブルワード	22	4	1		
	W	フルワード	4	4	7	2	24
小構造体	M	ダブルワード	28	4	7		

* 右方にシフトされた最初の項目

図 8. 構造体のマッピング例

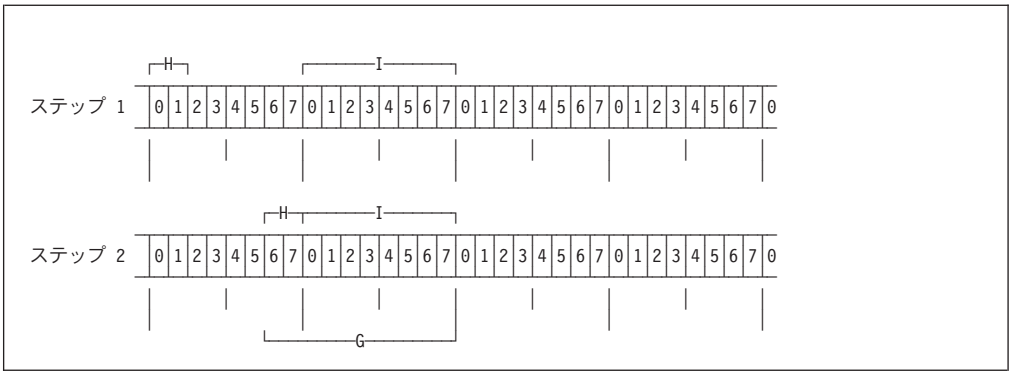


図 9. 小構造体 G のマッピング

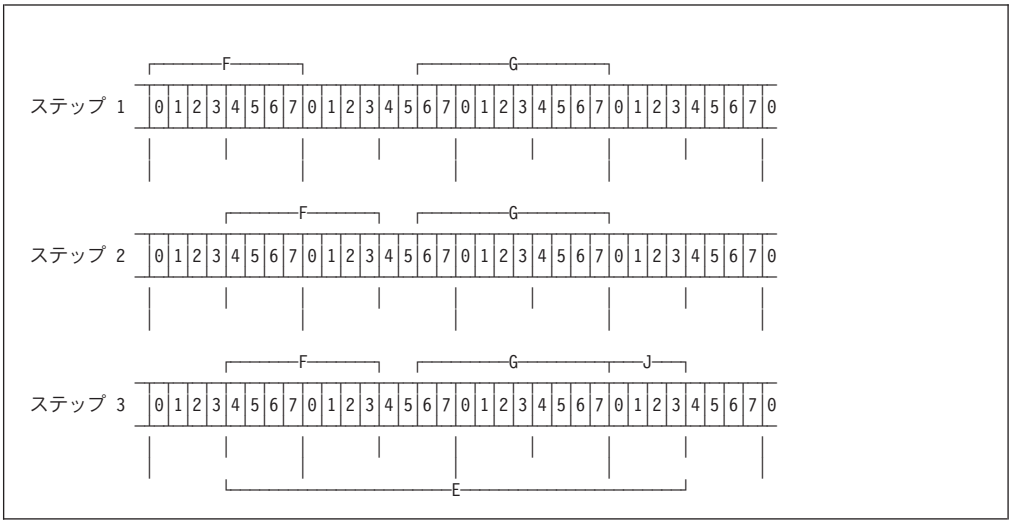


図 10. 小構造体 E のマッピング

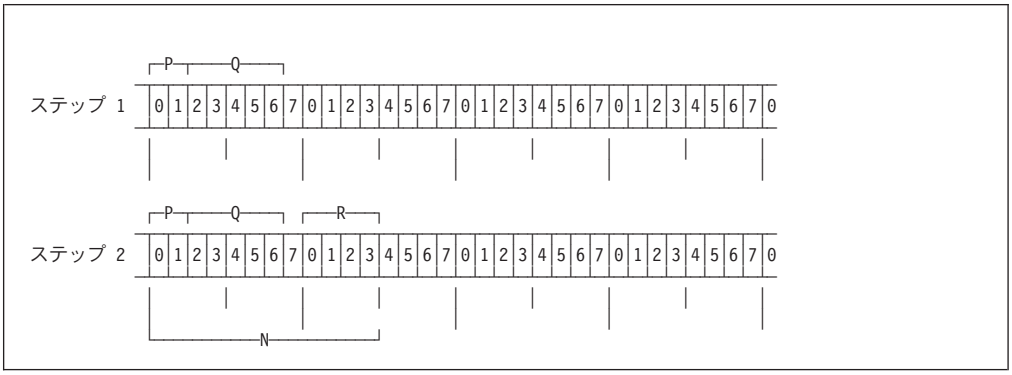


図 11. 小構造体 N のマッピング

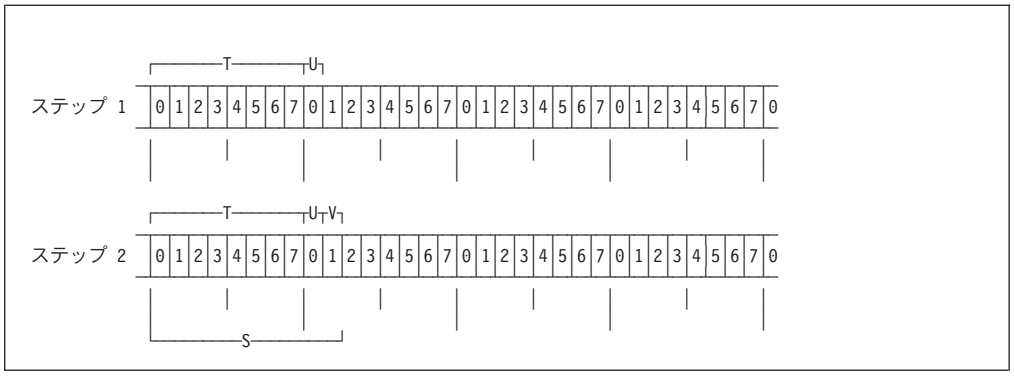


図 12. 小構造体 S のマッピング

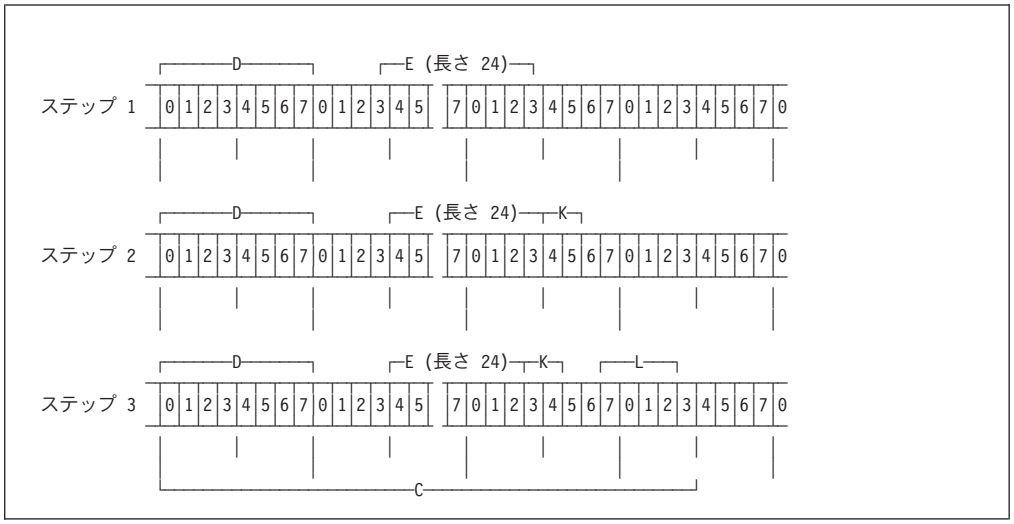


図 13. 小構造体 C のマッピング

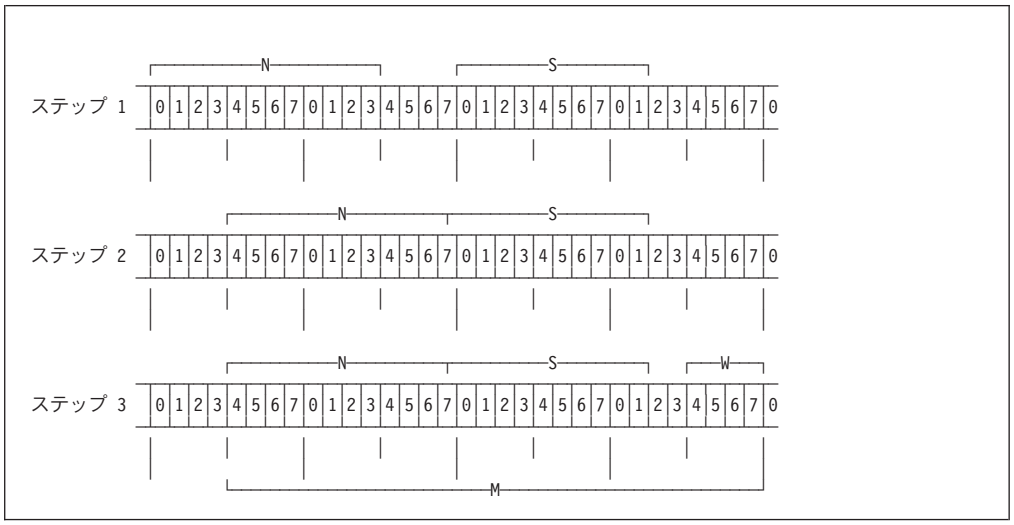


図 14. 小構造体 M のマッピング

	項目の名前	位置合わせ要件	長さ	ダブルワードのオフセット 開始	ダブルワードのオフセット 終了	埋め込み長さ	A のオフセット
ステップ 1	B	フルワード	4	0	3		
	C	ダブルワード	44	0	3		
ステップ 2	B*	フルワード	4	4	7		0
	C	ダブルワード	44	0	3	0	4
ステップ 3	B & C	ダブルワード	48	4	3		
	M	ダブルワード	28	4	7	0	48
ステップ 4	B, C, & M	ダブルワード	76	4	7		
	X	バイト	4	0	3	0	76
	A	ダブルワード	80	4	3		

* 右方にシフトされた最初の項目

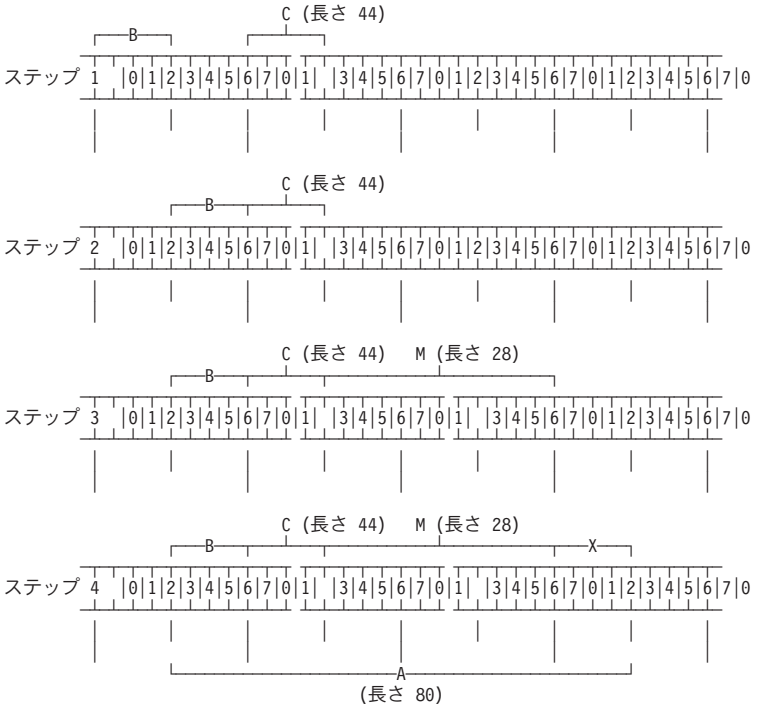


図 15. 大構造体 A のマッピング

A				**A** から
B				0
C			**C** から	4
D			0	4
埋め込み (4)			8	12
E		**E** から	12	16
F		0	12	16
埋め込み (2)		8	20	24
G	**G** から	10	22	26
H	0	10	22	26
I	2	12	24	28
J		20	32	36
K			36	40
埋め込み (2)			38	42
L			40	44
M			**M** から	48
N		**N** から	0	48
P		0	0	48
Q		2	2	50
埋め込み (1)		7	7	55
R		8	8	56
S		**S** から	12	60
T		0	12	60
U		8	20	68
V		9	21	69
埋め込み (2)			22	70
W			24	72
X				76

図 16. 構造 A の最終的なマッピングにおけるオフセット

第 9 章 ステートメントとディレクティブ

ALLOCATE ステートメント 201
代入ステートメントと複合代入ステートメント . . 202
代入ステートメント 202
複合代入ステートメント. 202
ターゲット変数. 204
割り当ての実行方法 204
複数割り当て 206
内部データ移動の例 207
式の値の割り当て例 207
BY NAME を使用した構造体の割り当て例 . . 207
ATTACH ステートメント 207
BEGIN ステートメント 207
CALL ステートメント 207
CLOSE ステートメント 207
DECLARE ステートメント 208
DEFINE ALIAS ステートメント 208
DEFINE ORDINAL ステートメント 208
DEFINE STRUCTURE ステートメント 208
DEFAULT ステートメント 208
DELAY ステートメント. 208
DELETE ステートメント 209
DETACH ステートメント 209
DISPLAY ステートメント 209
DO ステートメント 209
タイプ 1 210
タイプ 2 およびタイプ 3 210
タイプ 4 217
基本的な繰り返しの例 217
WHILE、UNTIL を指定した DO ステートメントの例. 218
UPTHRU と DOWNTHRU を指定した DO の例 219
REPEAT の例 220
END ステートメント. 221
ENTRY ステートメント. 222
EXIT ステートメント 222
FETCH ステートメント 222
FLUSH ステートメント 222
FORMAT ステートメント 222
FREE ステートメント 222
GET ステートメント 222
GO TO ステートメント 223
IF ステートメント. 223
例 225
%INCLUDE ディレクティブ 225
ITERATE ステートメント 226
LEAVE ステートメント 226
例 226
%LINE ディレクティブ 227
LOCATE ステートメント 227
%NOPRINT ディレクティブ 227
%NOTE ディレクティブ. 228
ヌル・ステートメント 228
ON ステートメント 228
OPEN ステートメント 229
%OPTION ディレクティブ 229
OTHERWISE ステートメント 229
PACKAGE ステートメント. 229
%PAGE ディレクティブ. 230
%POP ディレクティブ 230
%PRINT ディレクティブ 230
PROCEDURE ステートメント. 230
%PROCESS ディレクティブ 231
*PROCESS ディレクティブ. 231
%PUSH ディレクティブ. 231
PUT ステートメント. 232
READ ステートメント 232
RELEASE ステートメント 232
RESIGNAL ステートメント 232
RETURN ステートメント 232
REVERT ステートメント 233
REWRITE ステートメント 233
SELECT ステートメント 233
例 234
SIGNAL ステートメント 235
%SKIP ディレクティブ 235
STOP ステートメント 235
WAIT ステートメント 235
WHEN ステートメント 235
WRITE ステートメント 236
%XINCLUDE ステートメント 236

この章には、すべての PL/I ステートメントと % ディレクティブがリストされています。 % ステートメントとマクロ・ステートメントについては、 535 ページの『第 21 章 プリプロセッサーの機能』の章に説明されています。

ALLOCATE ステートメント

ALLOCATE ステートメントについては、 237 ページの『第 10 章 ストレージ制御』を参照してください。

代入ステートメントと複合代入ステートメント

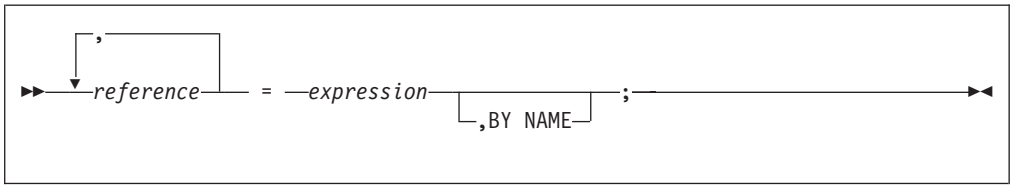
代入ステートメントは、式を計算し、その値を 1 つまたは複数のターゲット変数に割り当てます。

データを内部で移動させたり、計算を指定したりするときにこれらのステートメントを使用します。 (データを内部で移動させるときは、STRING オプションが指定された GET または PUT ステートメントを使用することもできます。さらに、PUT ステートメントでは実行する必要のある計算を指定することもできます。 『第 13 章 ストリーム指向データ伝送』を参照してください。)

ターゲット変数または疑似変数の属性は、ソース (変数、定数、または式の計算結果) の属性と異なっている可能性があるので、代入ステートメントで変換が必要になる場合があります (『第 5 章 データ変換』を参照)。

代入ステートメント

代入ステートメントには、以下の構文を使用します。



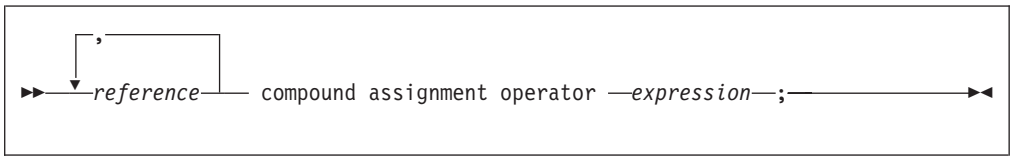
reference
割り当てを指定するターゲットを指定します。

expression
計算され、変換される可能性がある式を指定します。

BY NAME
構造体割り当ての場合は、BY NAME オプションが、 205 ページで概説されているステップに従う割り当てを指定します。

複合代入ステートメント

複合代入ステートメントには、以下の構文を使用します。



reference
割り当てを指定するターゲットを指定します。

compound assignment operator
割り当てが行われる前に、参照および計算された式に適用する演算子を指定します。 203 ページの表 30 に、複合割り当てで使用できる複合代入演算子をリストしています。

expression
計算され、変換される可能性がある式を指定します。

区域の割り当てについては、 255 ページの『区域データとその属性』の項で説明しています。

表 30. 複合代入演算子

複合代入演算子	意味
+=	式の計算、加算および代入
-=	式の計算、減算および代入
*=	式の計算、乗算および代入
/=	式の計算、除算および代入
|=	式の計算、OR および代入
&=	式の計算、AND および代入
||=	式の計算、連結と代入
**=	式の計算、指数化および代入
¬=	式の計算、排他 OR および代入

最初に演算子がターゲットおよびソースに適用され、次にその結果がターゲットに割り当てられます。

以下に例を示します。

```
X += 1          は          X = X+(1) と等しい
X *= Y+Z        は          X = X*(Y+Z) と等しい
```

ただし、

```
X *= Y+Z        は          X = X*Y+Z と等しくない
```

複合割り当てでは、ターゲット変数に指定された任意の添え字またはロケーター式が、一度だけ計算されます。

`f` が関数であり、`X` が配列の場合は、

```
X(f()) += 1     は          X(f()) = X(f())+1 と等しくない
```

関数 `f` は一度だけ呼び出されます。

以降では、次のトピックについて説明しています。
- ターゲット変数の要件
- エレメント割り当ておよび集合割り当ての実行方法
- BY NAME 割り当ての実行方法
- 複数割り当ての実行方法

割り当ての例は、 207 ページの『内部データ移動の例』から記載されています。

ターゲット変数

ターゲット変数にすることができるのは、要素変数、配列変数、または構造体変数、あるいは疑似変数です。

配列ターゲット

配列割り当てでは、各ターゲット変数は、スカラーまたは構造体の配列でなければなりません。 ソースは、そのターゲットと同じ次元数であり、またすべての次元についても同じ境界を持つ、スカラーまたは式でなければなりません。

共用体のターゲット

共用体の割り当てはできません。

構造体ターゲット

BY NAME を使用する構造体割り当てでは、各ターゲット変数は構造体である必要があります。右辺は、構造体参照でなければなりません。

BY NAME を使用しない構造体割り当てでは、各ターゲット変数は構造体である必要があります。右辺は、次に示すようにターゲットの構造体と同じ構造体を持ったスカラー式または構造式でなければなりません。

- 構造体は、小構造化がすべて同じで、中に含まれるエレメントと配列の数が同じでなければなりません。
- 構造体内 (および小構造があれば、その内部で) のエレメントや配列の位置指定は、同じでなければなりません。
- 対応する位置にある配列は、同じ境界を持たなければなりません。

BY NAME を使用しない構造体割り当てでは、ターゲット構造体に非計算データが含まれている場合でも、ヌル・ビット・ストリング (''b) をソースにすることができます。この場合、割り当ては次のことを仮定して実行されます。

1. ターゲットはすべて '00'x で埋められている
2. 数値ターゲット・フィールドはすべて 0 に設定されている
3. 非可変文字、ワイド文字、およびグラフィックのフィールドは、すべてブランクで埋められている

割り当ての実行方法

エレメントの割り当て

エレメントの割り当ては、次のように行われます。

1. 最初に評価されるのは、ターゲット変数の添え字、POSITION 属性式、およびロケーター修飾、ならびに SUBSTR 疑似変数参照の 2 番目と 3 番目の引数です。
2. 次に右辺の式が評価されます。
3. それぞれのターゲット変数ごとに (左から右の順で)、式の値がデータ変換規則に従ってターゲット変数の特性に変換されます。そのあと、変換後の値がターゲット変数に割り当てられます。

集合の割り当て

集合の割り当て (配列割り当ておよび構造体割り当て) は、次のように一連のエレメントの割り当てに展開されます。

1. 元のステートメントのラベル接頭部が、生成された別のステートメントの先頭にあるヌル・ステートメントに付けられます。
2. 配列割り当てや構造体割り当ては、複数あるときは、繰り返し行われます。
3. 先行の配列割り当てまたは構造体割り当てによって、なんらかの代入ステートメントが生成されることがあります。集合割り当てにおける最初のターゲット変数を、マスター変数といいます (これは疑似変数の最初の引数であることもあります)。 マスター変数が配列の場合は、配列式が実行されます。それ以外の場合は、構造式が実行されます。
4. 集合割り当てが所定の 1 組の条件を満たしているときは、集合割り当ては、一連のエレメント割り当てに展開されずに、全体として行われます。 1 組の条件とは、配列が介在配列でないか、または複数の構造体が隣接しており、しかも同一フォーマットになっているかということです。

配列割り当てでは、すべての配列オペランドの次元数および境界が同じでなければなりません。 配列の割り当ては、次のようにループに展開されます。

```
  do J1 = lbound(Master-variable,1) to
          hbound(Master-variable,1);
  do J2 = lbound(Master-variable,2) to
          hbound(Master-variable,2);
⋮
  do Jn = lbound(Master-variable,N) to
          hbound(Master-variable,N);

  generated assignment statement

  end;
```

この展開において、n は、この割り当てに関与するマスター変数の次元数です。生成された代入ステートメントでは、すべての配列オペランドに、ダミー変数 j1 から jn を使用して (左から右へ) 完全に添え字が付けられます。添え字を付けずに配列オペランドを書いた場合、その配列オペランドは、j1 から jn の添え字だけを持つことになります。クロスセクション表記を使用する場合、アスタリスクは、j1 から jn で置き換えられます。元の代入ステートメントにこの条件接頭語が付いている場合は、生成された代入ステートメントにこの条件接頭語が付けられます。

生成された代入ステートメントが構造体の割り当てであれば、そのステートメントは次のように展開されます。

BY NAME オプションの指定がない構造体割り当ての場合、

- どのオペランドも、配列であってはなりませんが、配列を含む構造体にすることはできます。
- 構造体に直接含まれる項目数は、すべての構造体オペランドについて同数 *k* でなければなりません。
- 代入ステートメントは、生成された *k* 個の代入ステートメントで置き換えられます。

- i 番目の生成された代入ステートメントが、それぞれの構造体オペランドを i 番目に含まれる項目で置き換えることによって、もとの代入ステートメントから派生します。そのように生成された代入ステートメントは、さらに展開を要求することができます。
- 元のステートメントの条件接頭語が、生成されたすべての代入ステートメントに付けられます。

BY NAME オプションの指定がある構造体割り当ての場合、構造体の割り当ては下記のステップに従って展開されます。これらのステップでは、さらに配列の割り当てや構造体の割り当てを生成することができます。どのオペランドも配列であってはなりません。

1. マスター変数に直接含まれている最初の項目が検討の対象になります。
2. それぞれの構造体オペランドとターゲット変数に同じ名前の項目が直接含まれている場合は、以下に述べるようにして代入ステートメントが生成されます。
   a. それぞれの構造体オペランドとターゲット変数を、それぞれに直接含まれている同名の項目で置き換えるという方法で、代入ステートメントが派生します。いずれかの構造体に同名の項目が含まれていない場合は、ステートメントは生成されません。
   b. 生成された代入ステートメントが構造体の割り当てまたは構造体の配列の割り当てである場合は、BY NAME が付けられます。
   c. 元の代入ステートメントの条件接頭語が、生成されたすべての代入ステートメントに付けられます。
   d. ターゲット構造体には、共用体は含まれません。
3. マスター変数に直接含まれている各項目ごとに、ステップ 2 が繰り返されます。マスター変数に含まれている項目の順に、代入ステートメントが生成されます。

複数割り当て

割り当ては、1 つの代入ステートメントで複数の変数に対して行うことができます。たとえば、次のようになります。

```
A,X = B + C;
```

B + C の値が A と X の両方に割り当てられます。普通は、このステートメントは、下記のステートメントが実行された場合と同じ結果になります。

```
Temporary = B + C;
A = Temporary;
X = Temporary;
```

代入ステートメント内のソースは、スカラーまたはスカラー配列でなくてはなりません。さらに、ソースが配列の場合は、すべてのターゲットも配列でなくてはなりません。ソースが定数である場合、各ターゲットに対して左から右に割り当てられます。ソースが定数ではない場合、一時変数に割り当てられ、このあと各ターゲットに対して左から右に割り当てられます。

ターゲットは、単一の割り当てで可能な参照であればどの参照でもかまいません。

BY NAME は、複数の割り当てでは使用できません。

内部データ移動の例

次の例の代入ステートメントを使用すれば、内部データを移動することができます。割り当ての記号の右側にある式の値は、左側の変数に割り当てられます。

```
NTOT=TOT;
```

式の値の割り当て例

次の例には、値が代入記号の左側にある変数に割り当てられる式が含まれます。

```
Av=(Av*Num+Tav*Tnum)/(Num+Tnum);
```

BY NAME を使用した構造体の割り当て例

次の例は、BY NAME オプションを使用した構造体割り当てを示します。

```
declare      declare       declare
1 One,       1 Two,        1 Three,
 2 Part1,     2 Part1,      2 Part1,
  3 Red,       3 Blue,       3 Red,
  3 Orange,    3 Green,      3 Blue,
 2 Part2,     3 Red,        3 Brown,
  3 Yellow,  2 Part2,      2 Part2,
  3 Blue,      3 Brown,      3 Yellow,
  3 Green;     3 Yellow;     3 Green;

1          One = Two, by name;
2          One.Part1 = Three.Part1, by name;
```

1 最初の代入ステートメントは、次のステートメントと同じです。

```
One.Part1.Red    = Two.Part1.Red;
One.Part2.Yellow = Two.Part2.Yellow;
```

2 2 番目の代入ステートメントは、次のステートメントと同じです。

```
One.Part1.Red = Three.Part1.Red;
```

ATTACH ステートメント

ATTACH ステートメントについては、 389 ページの『第 18 章 マルチスレッド化機能』を参照してください。

BEGIN ステートメント

BEGIN ステートメントについては、 93 ページの『第 6 章 プログラムの編成』を参照してください。

CALL ステートメント

CALL ステートメントについては、 133 ページの『CALL ステートメント』を参照してください。

CLOSE ステートメント

CLOSE ステートメントについては、 277 ページの『第 11 章 入出力』を参照してください。

DECLARE ステートメント

DECLARE ステートメントについては、160 ページの『DECLARE ステートメント』を参照してください。

DEFINE ALIAS ステートメント

DEFINE ALIAS ステートメントについては、147 ページの『第 7 章 タイプ定義』を参照してください。

DEFINE ORDINAL ステートメント

DEFINE ORDINAL ステートメントについては、147 ページの『第 7 章 タイプ定義』を参照してください。

DEFINE STRUCTURE ステートメント

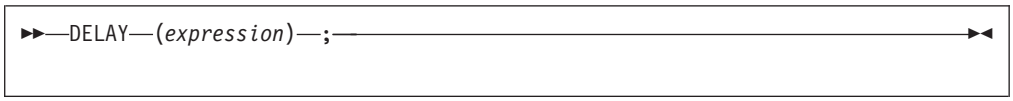
DEFINE STRUCTURE ステートメントについては、147 ページの『第 7 章 タイプ定義』を参照してください。

DEFAULT ステートメント

DEFAULT ステートメントについては、177 ページの『DEFAULT ステートメント』を参照してください。

DELAY ステートメント

DELAY ステートメントは、指定された時間の間、アプリケーション・プログラム内の次のステートメントの実行を一時中断させます。



expression

式を指定します。この式は、計算されて FIXED BINARY(31,0) に変換されます。指定されたミリ秒の間、実行が一時中断します。

最大待機時間は、23 時間 59 分です。

以下に例を示します。

```
delay (20);
```

上の例は、20 ミリ秒間、実行を一時中断させます。

```
delay (10**3);
```

上の例は、1 秒間、実行を一時中断させます。

```
delay (10*10**3);
```

上の例は、10 秒間、実行を一時中断させます。

DELETE ステートメント

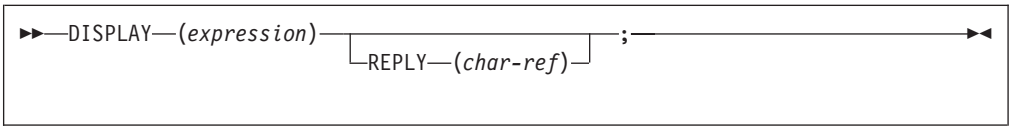
DELETE ステートメントについては、 293 ページの『第 12 章 レコード単位データ伝送』を参照してください。

DETACH ステートメント

DETACH ステートメントについては、 389 ページの『第 18 章 マルチスレッド化機能』を参照してください。

DISPLAY ステートメント

DISPLAY ステートメントは、ユーザーの表示画面にメッセージを表示し、任意指定により、そのメッセージに対する応答を入力するようユーザーに要求することもできます。



expression
必要に応じて、文字ストリングに変換されます。この文字ストリングが表示されます。このストリングに混合文字データが含まれていてもかまいません。式が GRAPHIC 属性を持つ場合、この式は変換されません。

REPLY (char-ref)
ユーザーが入力した応答を受け取る文字参照を指定します。応答には、CHARACTER、GRAPHIC、または混合データのいずれを含んでいてもかまいません。

REPLY オプションは、ユーザーが応答を入力するまでプログラムの実行を一時中断させることになります。

REPLY で GRAPHIC データが入力された場合、そのデータは、混合データを含む文字データとして受け取られます。このような文字データは、 GRAPHIC BUILTIN を使用すれば GRAPHIC データに変換することができます。

例

```
display ('Communication link established.');
```

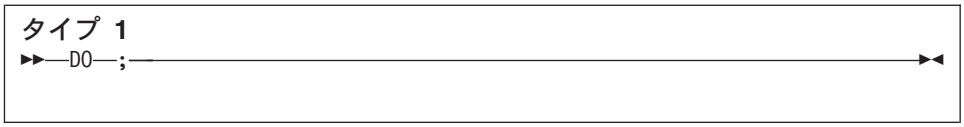
このステートメントは、次のメッセージを表示します。

```
Communication link established.
```

DO ステートメント

DO ステートメントとそれに対応する END ステートメントは、 DO グループと呼ばれるステートメント・グループを形成します。

注: 条件接頭語は、DO ステートメントでは使用することはできません。



expn
expression n の省略形です。

タイプ 1

タイプ 1 の DO グループは、グループ内のステートメントが実行されることを示します。グループ内のステートメントを反復して実行することはできません。

タイプ 2 およびタイプ 3

タイプ 2 および 3 では、DO グループ内でステートメントを反復して実行することができます。

WHILE (exp4)
DO グループの実行を反復すると、反復前に *exp4* が計算され、必要に応じてビット・ストリングに変換されます。結果のビット・ストリングのいずれかのビットが 1 であれば、その DO グループが実行されます。すべてのビットが 0 であるか、ストリングがヌルであれば、タイプ 2 の DO グループの実行が終了し、タイプ 3 の場合はこの WHILE オプションを含む指定に対応する実行だけが終了し、次の指定がある場合には、それが開始されます。

UNTIL (exp5)
DO グループの実行を反復すると、反復後に *exp5* が計算され、必要に応じてビット・ストリングに変換されます。結果のビット・ストリングのすべてのビットが 0 であるか、ストリングがヌルであれば、その DO グループの次の繰り返しが実行されます。いずれかのビットが 1 であれば、タイプ 2 の場合は DO グループの実行が終了し、タイプ 3 の場合はこの UNTIL オプションを含む指定に対応する実行だけが終了し、次の指定がある場合には、それが開始されます。

reference
参照として使用できる疑似変数は、SUBSTR、REAL、IMAG および UNSPEC のみです。どのデータ・タイプでもかまいません。

参照の世代 g は、DO グループの始まりで、初期値を表す式 (exp1) が計算される直前に、1 回だけ確立されます。その DO グループ内で参照の世代が h に変更されても、 DO グループの実行は、世代 g から派生した参照を使用して続行します。ただし、DO グループ内でその参照が参照されると、世代 h が参照されます。 DO グループ内で世代 g を解放しようとすると、エラーになります。

最後の繰り返しの完了後に参照を参照した場合、その参照の値は指定でセットした境界の範囲外の値になっています。次の条件がアプリケーションに設定された境界に適合する場合は、前述の内容が真になります。

- BY 値が正数で、参照の値が、指定でセットした境界の TO 値よりも大きい。
- BY 値が負数で、参照の値が、指定でセットした境界の TO 値よりも小さい。

参照がプログラム制御データ変数であり、ロケーターでなければ、BY オプションと TO オプションは、指定では使用できません。

参照がプログラム制御変数であり、ロケーターまたは序数でなければ、UPTHRU オプションと DOWNTHRU オプションは、指定では使用できません。

exp1
参照の初期値を指定します。

TO、BY、および REPEAT の指定が省略された場合には、 DO グループは 1 回だけ実行され、参照は値として exp1 を持ちます。 WHILE(exp4) を指定した場合は、 exp4 が真でなければ 1 回も実行されません。

TO exp2
exp2 は、その指定に制御権が移動したときに計算され、保管されます。この保管された値は、参照の終了値を示します。 DO グループの始めにテストされた参照の値が範囲内になければ、ただちに、指定に関して DO グループ内のステートメントの実行が終了します。その後、次の指定があれば、その実行が開始されます。

TO exp2 の指定が省略され、BY exp3 が指定されていると、WHILE オプションまたは UNTIL オプションによって実行が終了するまで、あるいはほかのステートメントが DO グループの外へ制御権を移動するまで、実行が繰り返されます。

BY exp3
exp3 は、その指定に制御権が移動したときに計算され、保管されます。この保管された値は、DO グループの実行が反復されるたびに参照に加算される増分を示しています。

BY exp3 の指定が省略され、 TO exp2 が指定されていると、exp3 はデフォルトの 1 になります。

BY 0 を指定すると、WHILE または UNTIL オプションによって実行が停止される場合と DO グループの外へ制御が移される場合とを除いて、DO グループの実行は無限に繰り返されます。

UPTHRU exp2
exp2 は、その指定に制御権が移動したときに計算され、保管されます。この保管された値は、参照の終了値を示します。 DO グループの終わりでテストされた参照の値が所定の範囲内になければ、ただちに、その指定に関して DO グループ内のステートメントの実行が終了します。その後、次の指定があれば、その実行が開始されます。

UPTHRU が指定されていると、ループ内のステートメントが実行されたあと、想定し得る次の値で更新される前に、参照は exp2 と比較されます。ループは、少なくとも一度実行されます。

UPTHRU は、主に序数の処理にループが使用されているときに使用されますが、参照が算術変数やロケーター制御変数である場合に、その参照でも使用されます。参照が序数を示さない場合には、DO グループの実行が行われるたびに参照に加えられる増分は、+1 と見なされます。

DOWNTHRU exp2
exp2 は、その指定に制御権が移動したときに計算され、保管されます。この保

管された値は、参照の終了値を示します。 DO グループの終わりでテストされた参照の値が所定の範囲内になければ、ただちに、その指定に関して DO グループ内のステートメントの実行が終了します。その後、次の指定があれば、その実行が開始されます。

DOWNTHRU が指定されていると、ループ内のステートメントが実行されたあと、参照が想定し得る次の値で更新される前に、参照は exp2 と比較されます。ループは、少なくとも一度実行されます。

DOWNTHRU は、主に序数の処理にループが使用されているときに使用されますが、参照が算術変数やロケーター制御変数である場合に、その参照でも使用されます。参照が序数を示さない場合には、DO グループの実行が行われるたびに参照に加えられる増分は、-1 と見なされます。

REPEAT exp6
DO グループの実行が反復されるたびに、exp6 が計算され、参照に割り当てられます。 WHILE または UNTIL オプションによって実行が停止されるまで、あるいはほかのステートメントが DO グループの外へ制御権を移動するまで、実行が繰り返されます。

DO グループの外から DO グループの中へ制御権を移動することができるのは、その DO グループがタイプ 1 の DO ステートメントで区切られている場合だけです。したがって、タイプ 2 とタイプ 3 の DO グループには、ENTRY ステートメントを含むことはできません。 DO グループ内から呼び出したプロシージャーまたは ON ユニットから、その DO グループへ制御を戻すことはできます。

以降の節では、タイプ 2 およびタイプ 3 の DO グループの使用について説明します。 DO グループの例は、 217 ページの『基本的な繰り返しの例』 ページから記載されています。

タイプ 2 WHILE と UNTIL の使用方法

タイプ 2 の DO の指定で、WHILE オプションと UNTIL オプションの両方を使用すると、その DO ステートメントによって、次に示すように実行が反復されます。

```
  Label:  do while (Exp4)
until (Exp5)
statement-1
⋮
statement-n
end;
  Next:   statement /* Statement following the do-group */
```

上記のものは、次のように展開されたものと同等です。

```
  Label:  if (Exp4) then;
else
go to Next;
statement-1
⋮
statement-n
  Label2: if (Exp5) then;
else
go to Label;
  Next:   statement /* Statement following the do-group */
```

WHILE オプションが省略されると、ラベル Label の IF ステートメントは、ヌル・ステートメントで置き換えられます。 WHILE オプションが省略されると、ステートメント 1 からステートメント n までが少なくとも一度は実行されます。

UNTIL オプションを省略したときは、展開フォーマット内のラベル Label2 の IF ステートメントが GO TO Label ステートメントで置き換えられます。

タイプ 3 を 1 つの指定で使用する場合

1 つの指定で DO グループを実行した場合の結果を、以下に要約して示します。

1. 参照を指定し、BY、TO、UPTHRU、または DOWNTHRU のいずれかのオプションも指定した場合は、 exp1 の参照への割り当てに先立って、exp1、 exp2、および exp3 が計算されます。次に、初期値が参照に割り当てられます。たとえば、次のようになります。

   ```
   do Reference = Exp1 to Exp2 by Exp3;
   ```

   疑似変数ではない変数の場合、前述の例の DO グループ定義によって取られる処置は、下記の展開と同等です。

   ```
   E1=Exp1;
   E2=Exp2;
   E3=Exp3;
   V=E1;
   ```

   変数 V は、参照と同じ属性を持つ、コンパイラー生成の基底付き変数です。E1、E2、および E3 はコンパイラー生成の変数です。
2. TO オプションがあると、制御変数の値が、前に計算された TO オプション内の式 E2 と比較されます。
3. WHILE オプションを指定した場合は、WHILE オプション内の式が計算されます。その結果が、*偽* であれば、DO グループから制御が離れます。
4. DO グループ内のステートメントが実行されます。
5. UNTIL オプションを指定した場合は、 UNTIL オプション内の式が計算されます。その結果が、*真* であれば、DO グループから制御が離れます。
6. UPTHRU オプションが指定されていると、制御変数の値が、前に計算された UPTHRU オプション内の式と比較されます。
7. DOWNTHRU オプションが指定されていると、制御変数の値が、前に計算された DOWNTHRU オプション内の式と比較されます。
8. 参照がある場合は、以下のようになります。
   a. TO オプションまたは BY オプションが指定されていると、前に計算された *exp3* (E3) が参照に加算されます。
   b. REPEAT オプションが指定されていると、 *exp6* が計算されて、参照に割り当てられます。
   c. TO、BY、REPEAT のどのオプションも指定していないときは、 DO グループから制御が離れます。
   d. UPTHRU オプションが指定されており、参照が序数を表していれば、現行値の後続値が参照に割り当てられます。参照が序数を表していないと、1 が参照に加算されます。

e. DOWNTHRU オプションが指定され、参照が序数を表していれば、現行値の先行値が参照に割り当てられます。参照が序数を表していなければ、そこから 1 が減算されます。

f. TO、BY、UPTHRU、DOWNTHRU、および REPEAT のオプションのうち何も指定されていなければ、DO グループから制御が離れます。

9. 2 (213 ページ) へ進みます。

タイプ 3 を 2 つまたはそれ以上の指定で使用する場合

DO ステートメント内に複数の指定がある場合、 2 番目の指定も最初の指定とあらゆる点で同じように展開されます。ただし、DO グループ内のステートメントがプログラム内で実際に重複することはありません。後続の指定が実行されるのは、先行の指定が終了したあとです。

最後の指定の実行が終了すると、制御はその DO グループのあとのステートメントに移ります。

タイプ 3 を TO、BY、REPEAT を指定して使用する場合

TO オプションや BY オプションを使用すれば、参照を一定の増分 (正または負) で変化させることができます。一方、TO オプションや BY オプションの代わりに REPEAT オプションを使用すれば、制御変数を非線形に変化させることができます。 REPEAT オプションは、算術タイプ以外の制御変数 (ポインターなど) にも使用することができます。

タイプ 3 の DO 指定で TO オプションと BY オプションの両方を使用すると、その DO グループの処置は次のように定義されます。

```
  Label:  do Variable=
Exp1
to Exp2
by Exp3
while (Exp4)
until(Exp5);
statement-1
⋮
statement-m
  Label1:  end;
  Next:    statement
```

前述の DO グループの定義によって取られる処置は、次のように展開されたものとまったく同等です。この展開において、V はコンパイラー生成変数で、Variable と同じ属性を持ちます。また、E1、E2、および E3 も、コンパイラー生成変数です。

```
  Label:  E1=Exp1;
  E2=Exp2;
  E3=Exp3;
  V=E1;
  Label2: if (E3>=0)&(V>E2)|(E3<0)&(V<E2) then
go to Next;
if (Exp4) then;
else
 go to Next;
statement-1
⋮
statement-m
  Label1: if (Exp5) then
```

```
go to Next;
  Label3: V=V+E3;
go to Label2;
  Next:   statement
```

この指定で REPEAT オプションを使用したときは、DO グループの処置は次のように定義されます。

```
  Label:  do Variable=
Exp1
repeat Exp6
while (Exp4)
until(Exp5);
statement-1
⋮
statement-m
  Label1:  end;
  Next:    statement
```

前述の DO グループの定義によって取られる処置は、次のように展開されたものとまったく同等です。

```
  Label:  E1=Exp1;
V=E1;
  Label2: ;
if (Exp4) then;
else
  go to Next;
statement-1
⋮
statement-m
  Label1: if (Exp5) then
   go to Next;
  Label3: V=Exp6;
   go to Label2;
  Next:   statement
```

例に示した展開では、さらに次のような規則があります。

1. 前述の展開は、1 つの指定の結果だけを示しています。 DO ステートメント内に複数の指定がある場合は、展開フォーマット内のラベル NEXT が付いているステートメントが、次の指定の最初のステートメントになります。 2 番目の展開も、最初の展開とあらゆる点で同じように展開されます。しかし、プログラムでは、statement-1 から m は実際には重複しません。
2. WHILE 文節を省略したときは、それぞれの展開フォーマット内の statement-1 のすぐ前にある IF ステートメントも省略されます。
3. UNTIL 文節を省略したときは、それぞれの展開フォーマット内の statement-m のすぐあとにある IF ステートメントも省略されます。

タイプ 3 を UPTHRU を指定して使用する場合

タイプ 3 の DO 指定に UPTHRU オプションが含まれる場合、DO グループの処置は、次のようにして定義されます。

```
  Label:  do Variable = Exp1 upthru Exp2 while (Exp4) until (Exp5);
statement1
⋮
statementn
  Label1: end;
  Next: statement
```

上述の DO グループの処置は、次のように展開されたものと同等です。この展開において、*V* はコンパイラー生成変数で、 *Variable* と同じ属性を持ちます。また、*E1* と *E2* も、コンパイラー生成変数です。

```
  Label:  E1 = Exp1;
E2 = Exp2;
V = E1;
  Label2: if (Exp4) then;
else
go to next;
statement1
⋮
statementn


  Label1: if (Exp5) then
go to Next;
if V ≥ E2 then
go to Next;
V = V + 1;
go to Label2;
  Next: statement
```

参照が序数を表していれば、ステートメント `V = V + 1` は、`V = ordinalsucc(V)` で置き換えられます。

タイプ 3 を DOWNTHRU を指定して使用する場合

タイプ 3 の DO 指定に DOWNTHRU オプションが含まれる場合、DO グループの処置は、次のようにして定義されます。

```
  Label:  do Variable = Exp1 downthru Exp2 while (Exp4) until (Exp5);
statement1
⋮
statementn
  Label1: end;
  Next: statement
```

上述の DO グループの処置は、次のように展開されたものと同等です。この展開において、*V* はコンパイラー生成変数で、 *Variable* と同じ属性を持ちます。また、*E1* と *E2* も、コンパイラー生成変数です。

```
  Label:  E1 = Exp1;
E2 = Exp2;
V = E1;
  Label2: if (Exp4) then;
else
go to Next;
statement1
⋮
statementn


  Label1: if (Exp5) then
go to Next;
if V ≤ E2 then
go to Next;
V = V - 1;
go to Label2;
  Next: statement
```

参照が序数を表していれば、ステートメント `V = V - 1` は、`V = ordinalpred(V)` で置き換えられます。

タイプ 4

LOOP

無限の繰り返しを指定します。 FOREVER は、LOOP の同義語です。

以下に例を示します。

```
dcl Payroll file;
dcl 1 Payrec,
     2 Type char,
     2 Subtype char,
     2 * char(100);

Readfile:
  do loop;

    read file(Payroll) into(Payrec);

    If Payrec.type = 'E'
      then leave; /* like goto After_ReadFile */

    If Payrec.type = '1' then
      do;
           /* process first part of record */

        If Payrec.subtype = 'S'
          then iterate Readfile; /* like goto End_ReadFile */

           /* process remainder of record */
      end;

End_ReadFile:
  end;
After_ReadFile:;
```

このループから出るには、LEAVE または GO TO を使用するか、もしくはプロシージャーまたはプログラムを終了するしかありません。

基本的な繰り返しの例

次の例では、DO グループは 10 回実行され、参照 I は 1 から 10 までの範囲を値にとります。

```
  do I = 1 to 10;
⋮
  end;
```

DO および END ステートメントを使用すると、次のように書く場合と同等の結果になります。

```
    I = 1;
  A: if I > 10 then go to B;
⋮
     I = I +1;
     go to A;
  B: next statement
```

以下の DO ステートメントは DO グループを、3 回実行します。 Name に 'Tom'、'Dick'、および 'Harry' をそれぞれ割り当てる度に 1 回です。

```
  do Name = 'Tom', 'Dick', 'Harry';
```

以下のステートメントは、DO グループを 13 回実行することを指定します。 I の値を 1 から 10 に等しくして 10 回、また I の値を 13 から 15 に等しくして 3 回です。

```
  do I = 1 to 10, 13 to 15;
```

参照を添え字として使用した反復

DO ステートメントの参照は、DO グループ内のステートメントで添え字として使用することができます。そのため、各実行では、表または配列の連続するエレメントを処理します。

次の例では、 A の最初の 10 個のエレメントは、順次 1 から 10 にセットされます。

```
  do I = 1 to 10;
     A(I) = I;
  end;
```

TO および BY を使用した反復

次の例では、DO グループを 5 回実行することを指定しています。 I の値は、2、4、6、8、10 となります。

```
  do I = 2 to 10 by 2;
```

参照の値を減らしていく場合は、BY オプションを使用しなければなりません。たとえば、次の例は、I が 10、8、6、4、2、0、-2 の値で実行されます。

```
  do I = 10 to -2 by -2;
```

次の例では、I が 1、3、5 値で DO グループが実行されます。

```
  I=2;
  do I=1 to I+3 by I;
:
end;
```

前述の例は、次のように書く場合と同等の結果になります。

```
  do I=1 to 5 by 2;
:
end;
```

WHILE、UNTIL を指定した DO ステートメントの例

WHILE オプションや UNTIL オプションを使用すると、指定の条件に従って、 DO グループを連続的に実行することができます。たとえば、次のようになります。

```
  do while (A=B);
:
end;
```

これは、次のように書く場合と同等です。

```
  S:  if A=B then;
else goto R;
:
goto S;
  R:  next statement
```

さらに、

```
  do until (A=B);
⋮
end;
```

これは、次のように書く場合と同等です。

```
  S:
⋮
if (A=B) then goto R;
goto S;
  R:  next statement
```

ほかのオプションが付いていない限り、DO UNTIL ステートメントで始まる DO グループは最低 1 回は実行されますが、DO WHILE ステートメントで始まる DO グループは 1 回も実行されないことがあります。つまり、ステートメントの DO WHILE (A=B) と DO UNTIL (A¬=B) は、同じではないということです。

次の例では、DO ステートメントに最初に制御が到達したときに A¬=B であれば、DO グループは 1 回も実行されません。

```
  do while(A=B) until(X=10);
```

ただし、A=B であれば、DO グループは実行されます。 DO グループの実行後、X=10 であれば、それ以上実行されません。 X=10 でない場合は、A がまだ B に等しい限り、さらに実行されます。

次の例では、DO グループは最低 1 回は実行され、そのとき I は 1 です。

```
  do I=1 to 10 until(Y=1);
```

DO グループの実行後、Y=1 であれば、それ以上実行されません。 Y=1 でない場合は、デフォルトの増分値 (BY 1) が I に加算され、I の新しい値が 10 と比較されます。 I が 10 より大きいときは、それ以上実行されませんが、そうでないときは、新たに実行されます。

次のステートメントは、 C(I) がゼロより小さければ DO グループを 10 回実行し、次に A が B に等しい場合にのみ、もう 1 回実行することを指定しています。

```
  do I = 1 to 10 while (C(I)<0),
         11 while (A = B);
```

UPTHRU と DOWNTHRU を指定した DO の例

次の例では DO グループを 5 回実行すると、ループの終わりに達して *i* の値が 5 になります。

```
  do i = 1 upthru 5;
⋮
end;
```

UPTHRU オプションを使用すると、参照は、更新前に終了値と比較されます。これは、終了値以降に何も値がない場合に非常に役立ちます。たとえば、FIXEDOVERFLOW 条件は、次に示すようなループでは発生しません。

```
  do i = 2147483641 upthru 2147483647;
⋮
  end;
```

同様に、次のループは、符号なしの値をゼロまで減分していく際に生じる可能性がある問題を予防します。

```
  dcl U unsigned fixed bin;
  do U = 17 downthru 0;
⋮
  end;
```

UPTHRU と DOWNTHRU は、序数とともに使用すると特に有効です。次の例を考えてみてください。

```
  define ordinal Color ( Red value (1),
Orange,
Yellow,
Green,
Blue,
Indigo,
Violet);
  dcl C ordinal Color;


  do C = Red upthru Violet;
⋮
  end;


  do C = Violet downthru Red;
⋮
  end;
```

最初のループでは、*c* は、red から violet まで昇順で連続する各色を想定しています。 2 番目のループでは、*c* は、violet から red まで降順で連続する各色を想定しています。

REPEAT の例

次の例では、DO グループは I は 1、2、4、8、16 などの値で実行されます。

```
  do I = 1 repeat 2*I;
⋮
end;
```

前述の例は、次のように書く場合と同等の結果になります。

```
    I=1;
  A:
⋮
I=2*I;
goto A;
```

次の例では、DO グループの最初の実行は I=1 で行われます。

```
  do I=1 repeat 2*I until(I=256);
```

この実行のあと、およびそれ以後 DO グループが実行されるたびに、UNTIL 式がテストされます。 I=256 であれば、それ以上実行されません。 I=256 でない場合は、 REPEAT 式が計算されて I に割り当てられ、新たに実行されます。

次の例では、チェーニングしたリストで特定の項目を探索するために使用される DO ステートメントを示します。

```
do P=Phead repeat P -> Fwd
     while(P¬=null())
     until(P->Id=Id_to_be_found);
end;
```

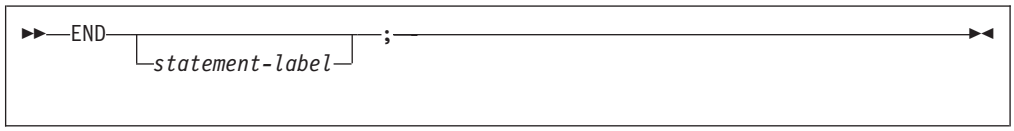
DO グループの最初の実行で、値 Phead が P に割り当てられます。それ以降 DO グループが実行されるたびに、その実行前に値 P -> Fwd が P に割り当てられます。 DO グループの最初の実行前および後続のそれぞれの実行前に、 P の値がテストされ、その値が NULL であれば、それ以上実行されません。

以下のステートメントは、 DO グループを 9 回実行することを指定します。 I の値を 1 から 9 に等しくして 9 回 です。その後、引き続き I の値を 10、20、40、などに等しくします。 10000 より大きい I の値で DO グループが実行されたあと、実行の繰り返しが終わります。

```
do I = 1 to 9, 10 repeat 2*I
               until (I>10000);
⋮
end;
```

END ステートメント

END ステートメントは、1 つまたは複数のブロックやグループを終了させます。それぞれのブロックごと、またはグループごとに、 END ステートメントがなければなりません。



statement-label

添え字を付けることはできません。 END のあとにステートメント・ラベル (statement-label) を指定すると、そのステートメント・ラベルを持ち、DO、SELECT、PACKAGE、BEGIN、または PROCEDURE ステートメントで始まっていて、まだ閉じられていないグループまたはブロックのうち、その END ステートメントより前で、最も近くにあるものを閉じます。 DO、SELECT、PACKAGE、BEGIN、または PROCEDURE ステートメントを持つすべてのブロックには、対応する END ステートメントがなければなりません。

END のあとにステートメント・ラベルがない場合は、その END ステートメントより前で、最も近くにある DO、SELECT、PACKAGE、BEGIN、または PROCEDURE ステートメントで始まっているグループまたはブロックのうち、対応する END ステートメントがほかにないものが閉じられます。

ブロックの END ステートメントに制御が到達すると、そのブロックの実行は終了します。ただし、各ブロックには END ステートメントがなければなりませんが、ブロックの実行を終了させる方法はほかにもあります。 (詳細については、 99 ページの『プロシージャー』 および 119 ページの『開始ブロック』を参照してください。)

プロシージャーの END ステートメントに制御が到達したときは、その END ステートメントは RETURN ステートメントと見なされます。

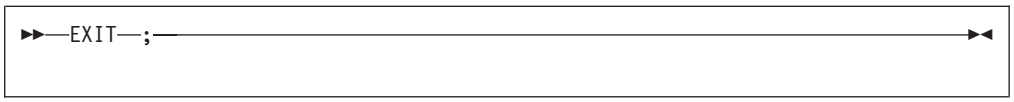
主プロシージャーの END ステートメントに制御が到達したときは、プログラムが正常終了します。

ENTRY ステートメント

ENTRY ステートメントについては、122 ページの『ENTRY 属性』を参照してください。

EXIT ステートメント

EXIT ステートメントは、現行のスレッドを停止します。



FETCH ステートメント

FETCH ステートメントについては、110 ページの『FETCH ステートメント』を参照してください。

FLUSH ステートメント

FLUSH ステートメントについては、277 ページの『第 11 章 入出力』を参照してください。

FORMAT ステートメント

FORMAT ステートメントについては、303 ページの『第 13 章 ストリーム指向データ伝送』を参照してください。

FREE ステートメント

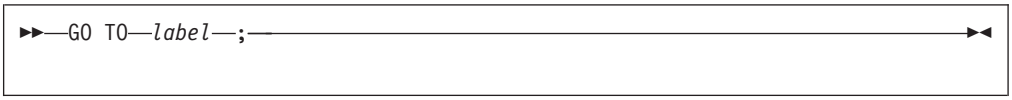
FREE ステートメントについては、237 ページの『第 10 章 ストレージ制御』を参照してください。

GET ステートメント

GET ステートメントについては、303 ページの『第 13 章 ストリーム指向データ伝送』を参照してください。

GO TO ステートメント

GO TO ステートメントは、指定されたラベル参照によって識別されるステートメントに制御を移します。 GO TO ステートメントは無条件分岐です。



省略形: GOTO

label
ラベル定数、ラベル変数、またはラベル値を返す関数参照を指定します。ラベル変数の場合、GO TO ステートメントの実行のたびに異なる値になっていることがあるので、常に同じステートメントに制御が移されるとは限りません。

GO TO ステートメントによって、あるブロック内からそのブロックに含まれていない場所に制御が移されると、そのブロックは終了します。制御の移動先が、終了するブロックを直接に活動化したのではないブロック内にあると、活動化に際して介在したすべてのブロックも終了します (106 ページの『プロシージャーの終了』を参照してください)。

GO TO ステートメントが、複数回活動化を行うブロックに含まれるラベル定数を指定する場合には、その GO TO ステートメントの実行時に活動中のものに制御が移されます (107 ページの『再帰的プロシージャー』を参照してください)。

GO TO ステートメントを使用しても、次に示すものへ制御権を移動することはできません。

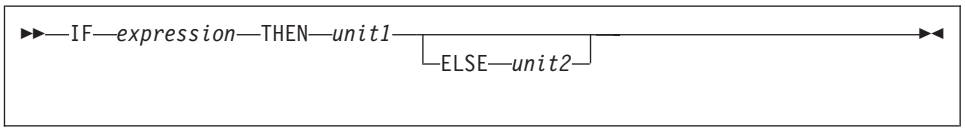
- 非アクティブ・ブロックへ。このようなエラーが検出されるという保証はありません。
- DO グループの外から、タイプ 2 またはタイプ 3 の DO グループ内のステートメントへ。ただし、DO グループ内から呼び出されたプロシージャーまたは ON ユニットを GO TO で終了させる場合は別です。
- FORMAT ステートメントへ。

GO TO の行先をラベル変数で指定し、ラベル定数をそのラベル変数に割り当てれば、スイッチとして使用することができます。また、そのラベル変数に添え字を付けておけば、添え字を変化させることによってスイッチを制御することができます。ラベル変数または関数参照を使用すれば、きわめて複雑なスイッチ機能を実行することができます。ただし、普通は、制御ステートメントは単純である方が効率はよくなります。あるブロックから別のブロックへの GOTO 操作によって、ターゲット・ブロックでの多くの最適化が妨げられます。ただし、ターゲット・ラベルがそのブロック内の最後のステートメントである場合は例外です。

IF ステートメント

IF ステートメントは、式を計算し、計算結果に応じて実行の流れを制御します。したがって、IF ステートメントは、条件分岐です。

注: 条件接頭語は、ELSE ステートメントでは使用することはできません。



expression
式は、RULES(LAXIF) を使用しない場合、ビット (ビット・ストリングではない) を評価しなければなりません。

式の中で & または | (もしくはその両方の) 演算子が使用されている場合は、それらの演算子は、71 ページの『各種演算の組み合わせ』に説明した方法で計算されます。

unit
有効な単一のステートメント、グループ、または開始ブロックのいずれかです。DECLARE、DEFAULT、END、FORMAT、PROCEDURE、または % ステートメントを除いて、すべての単一ステートメントは有効と見なされ、実行可能です。非実行ステートメントを指定すると、結果は予想できません。それぞれのユニットには、制御の移動を指定するステートメント (たとえば、GO TO) が含まれていてもかまいません。そのようなステートメントを使用すれば、IF ステートメントの正規の順序を無効にすることができます。

それぞれのユニットには、ラベルを付けることも、条件接頭語を付けることもできます。

IF は複合ステートメントです。最後のユニットを終了させるセミコロンは、IF ステートメントも終了させます。

ストリング式のいずれかのビットが '1'B であれば、 *unit1* が実行され、*unit2* は存在していても無視されます。すべてのビットが '0'B であるか、ストリングがヌルであれば、*unit1* は無視され、 *unit2* (存在していれば) が実行されます。

IF ステートメントはネストさせることができます。つまり、一方または両方のユニットが IF ステートメントであってもかまいません。それぞれの ELSE は、同じブロックまたは DO グループに含まれている IF ステートメントのうち、最も内側にあって、しかも対応する ELSE を持っていない IF に関連付けられます。したがって、所要の制御順序を指定するために、ヌル・ステートメントを持つ ELSE が必要になることがあります。たとえば、B および C が定数である場合、

```
if A = B then
⋮
else
if A = C then
⋮
else
⋮
```

上記の例は、次の例と同等ですが、より巧みにコード化されています。

```
select( A );
when ( B )
⋮
when ( C )
⋮
```

```
:
end;
```

例

次の例では、比較の結果が真 (A が B に等しい) であれば、 D の値が C に割り当てられ、ELSE ユニットは実行されません。

```
if A = B then
  C = D;
else
  C = E;
```

比較の結果が偽 (A が B に等しくない) であれば、 THEN ユニットは実行されず、E の値が C に割り当てられます。

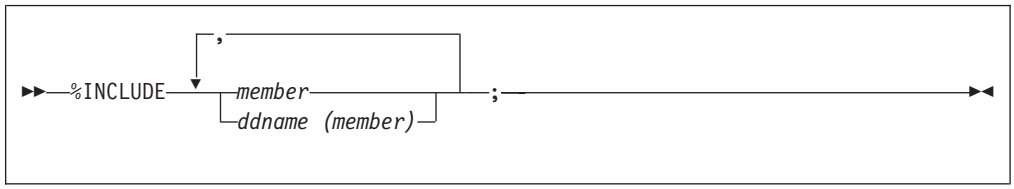
THEN ユニットと ELSE ユニットのどちらにでも、制御権を移動する (条件付きで、または無条件で) ステートメントを書くことができます。 THEN ユニットが GO TO ステートメントで終わる場合には、 ELSE ユニットを指定する必要はありません。たとえば、次のようになります。

```
if all(Array1 = Array2) then
  go to LABEL_1;
next-statement
```

この式が真であれば、THEN ユニットの GO TO ステートメントによって、LABEL_1 に制御が移されます。この式が偽であれば、THEN ユニットは実行されず、次のステートメントに制御が移されます。

%INCLUDE ディレクティブ

%INCLUDE ディレクティブは、外部のテキストをソース・プログラムに合体するために使用します。



組み込みメンバーには、絶対ファイル名を指定してもかまいません。絶対ファイル名を単一引用符または二重引用符で囲みます。たとえば、次は有効です。

INTEL

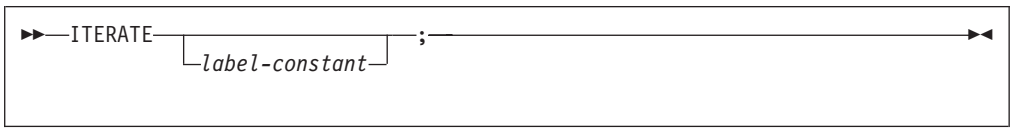
```
%include "¥ibmpli¥include¥sqlcodes.inc"
```

AIX および z/OS UNIX

```
%include "/ibmpli/include/sqlcodes.inc"
```

ITERATE ステートメント

ITERATE ステートメントは、それに含まれる反復 DO グループを区切る END ステートメントに制御を移します。現在行われている反復は完了され、次の反復が (必要であれば) 開始されます。 ITERATE ステートメントは、反復 DO グループが非反復 DO グループを含んでいる場合には、その非反復 DO グループ内で指定することができます。

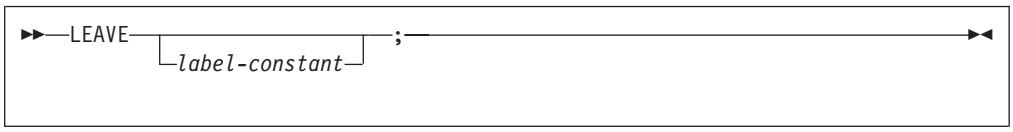


label-constant
このステートメントを含んでいる DO グループのラベルでなければなりません。省略した場合、ITERATE ステートメントを含む最新の反復 DO グループの END ステートメントに制御が移ります。

例については、 217 ページの『タイプ 4』を参照してください

LEAVE ステートメント

単純な DO グループに含まれているか、または単純な DO グループを指定してある場合に、 LEAVE ステートメントは、そのグループから制御を離します。反復 DO グループに含まれているか、またはそのようなグループを指定してある場合には、 LEAVE ステートメントは、現在の繰り返しも含め、そのグループのすべての繰り返しを終了します。制御の流れは、DO グループが END ステートメントに達して終了した場合に通常移動するはずの位置に移動します。この位置は、DO グループの END ステートメントに続く次のステートメントである必要はありません (『例』 を参照してください)。



label-constant
このステートメントを含んでいる DO グループのラベルでなければなりません。ここで指定するラベルを持つ DO グループが終了します。 *label-constant* が省略されると、 LEAVE ステートメントを含むグループが終了する DO グループです。

LEAVE ステートメントおよび参照される (または暗黙に指定される) DO ステートメントが、別々のブロックにあってはなりません。

次の例の詳細については、 217 ページの『タイプ 4』の項を参照してください。

例

次の例では、ステートメント `leave A;` は、 `C` に制御を移します。

```
  If Time_to_process_X then


  A:  do I = lbound(X,1) to hbound(X,1);
         do J = lbound(X,2) to hbound(X,2);
            If X(I,J)=0 then
               leave A;          /* control goes to C, not B */
            else
               do;
.
.
.
               end;
         end;
      end;


    Else


  B:  do;
.
.
.
      end;


  C:  statement after group A;
```

%LINE ディレクティブ

%LINE ディレクティブは、メッセージおよびデバッグ用に生成される情報の中で、その次の行を指定の行とファイルから取得した場合と同様に処理するように指定します。

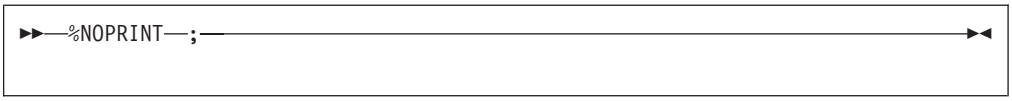
ディレクティブが認識されるためには、文字 '%LINE' が入力行の列 1 から 5 までにあることが必要です (また逆に、これらの 5 文字で始まる行はすべて %LINE ディレクティブとして扱われます)。 `line-number` は 7 桁以下の整数値であることが必要で、 `file-specification` は引用符で囲んではなりません。セミコロンの後に指定された文字はすべて無視されます。

LOCATE ステートメント

LOCATE ステートメントについては、 293 ページの『第 12 章 レコード単位データ伝送』を参照してください。

%NOPRINT ディレクティブ

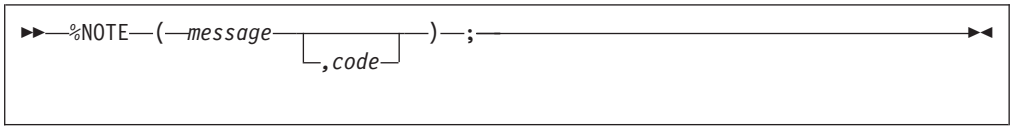
%NOPRINT ディレクティブは、%PRINT ディレクティブに制御権が移動するか、または以前の %PRINT ディレクティブを保管している %POP ディレクティブに制御権が移動するまで、ソース・リストの印刷を中断します。



%NOPRINT ディレクティブの例については、 231 ページの『%PUSH ディレクティブ』を参照してください。

%NOTE ディレクティブ

%NOTE ディレクティブは、指定のテキストおよび重大度の診断メッセージを生成します。



message
診断メッセージを必要とする値を持つ文字式。

code
メッセージの重大度を表す値が入っている固定式。重大度は次のとおりです。

コード	重大度
0	I
4	W
8	E
12	S
16	U

code を省略すると、0 と見なされます。

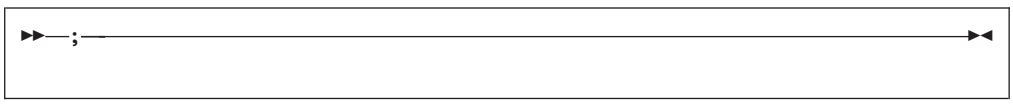
code が上のリストに示した値以外の値を持つ場合、診断メッセージが作成されます。この場合、その結果として起こるシステムの処置は未定義です。

生成されたメッセージは、ほかのメッセージとともにファイルにされます。特定のメッセージを続いて印刷するかどうかは、その重大度レベルとコンパイラー FLAG オプション (「プログラミング・ガイド」に説明されている) の設定に依存します。

重大度 U のメッセージが生成されると、プリプロセスとコンパイル処理が直ちに終了します。重大度 S、E、または W のメッセージが生成されたときにコンパイル処理が終了するかどうかは、コンパイラー・オプションの設定値によってそれぞれ異なります。

ヌル・ステートメント

ヌル・ステートメントは、何も行わず、ステートメントの実行順序を変更することもしません。ヌル・ステートメントは、THEN 文節および ELSE 文節ならびに WHEN ステートメントおよび OTHERWISE ステートメントのヌル処置を示すために、しばしば使用されます。



ON ステートメント

ON ステートメントについては、 357 ページの『第 16 章 条件処理』を参照してください。

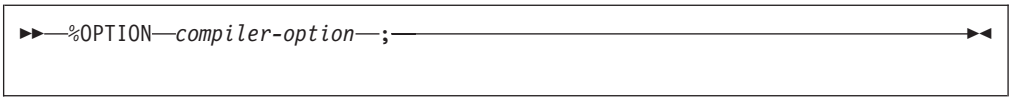
OPEN ステートメント

OPEN ステートメントについては、277 ページの『第 11 章 入出力』を参照してください。

%OPTION ディレクティブ

%OPTION ディレクティブは、ソース・コードのセグメント向けに選択したコンパイラー・オプションのサブセットのうちから 1 つを指定するときに使用します。このディレクティブによって指定したオプションが有効になると、それは以下の時点まで続きます。

- 別の %OPTION ディレクティブが、相補的なコンパイラー・オプションを指定する (そのため最初のものが指定変更される)。
- (%PUSH ディレクティブにより) 保管されていたコンパイラー・オプションが、%POP ディレクティブによって復元される。



compiler-option
有効にするコンパイラー・オプションを指定します。

使用可能なコンパイラー・オプションについては、「*プログラミング・ガイド*」を参照してください。

%OPTION の例に関しては、231 ページの『%PUSH ディレクティブ』を参照してください。

OTHERWISE ステートメント

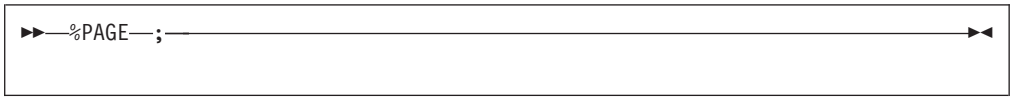
OTHERWISE ステートメントについては、233 ページの『SELECT ステートメント』を参照してください。

PACKAGE ステートメント

PACKAGE ステートメントについては、93 ページの『第 6 章 プログラムの編成』を参照してください。

%PAGE ディレクティブ

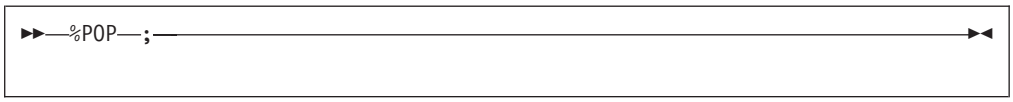
%PAGE ディレクティブを使用すれば、コンパイラーのソース・リストを新しいページから開始できます。



%POP ディレクティブ

%POP ディレクティブを使用すれば、最近の %PUSH ディレクティブによって保管した %PRINT、%NOPRINT、および %OPTION の各ディレクティブの状況を復元することができます。

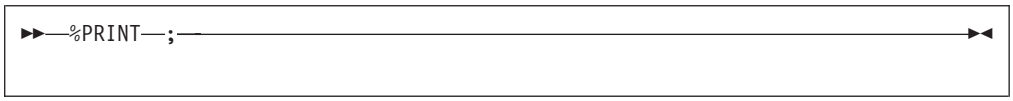
%PUSH ディレクティブと %POP ディレクティブの最も一般的な用途は、インクルード・ファイルやマクロにあります。



例については、 231 ページの『%PUSH ディレクティブ』を参照してください

%PRINT ディレクティブ

%PRINT ディレクティブは、ソース・リストの印刷を再開させます。



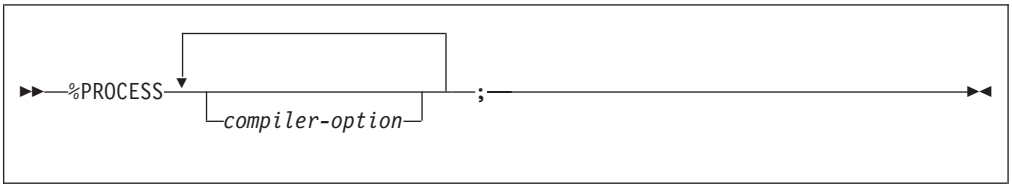
関係のあるコンパイラー・オプションが指定されていれば、%PRINT が有効になります。 %PRINT ディレクティブの例については、 231 ページの『%PUSH ディレクティブ』を参照してください。

PROCEDURE ステートメント

PROCEDURE ステートメントについては、 93 ページの『第 6 章 プログラムの編成』を参照してください。

%PROCESS ディレクティブ

%PROCESS ディレクティブは、コンパイラー・オプションを指定変更するために使用します。



% または * が、ソース・レコードの最初のデータ・バイトでなければなりません。 %PROCESS ディレクティブと *PROCESS ディレクティブはいくつでも指定できますが、最初の言語エレメントが現れるより前に、それらをすべて指定しておかなければなりません。さらに詳しくは「プログラミング・ガイド」を参照してください

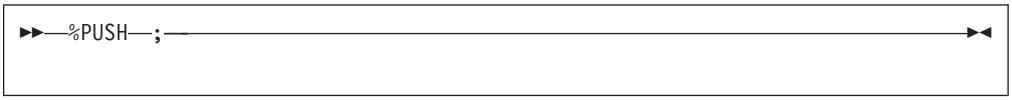
*PROCESS ディレクティブ

*PROCESS ディレクティブは、%PROCESS ディレクティブの同義語です。 %PROCESS ディレクティブに関する詳細は、『%PROCESS ディレクティブ』を参照してください。

%PUSH ディレクティブ

%PUSH ディレクティブを使用すれば、%PRINT、%NOPRINT、および %OPTION の各ディレクティブの現行の状況を後入れ先出し方式で「プッシュダウン」スタック内に保管できます。 %POP ディレクティブを使用すれば、後ほどこの保管した状況を同じく後入れ先出し方式で復元できます。

%PUSH ディレクティブと %POP ディレクティブは、一般的にインクルード・ファイルやマクロで使用されます。



次の例では、ステートメント 1、2、3、S4、S5、および 4 が、リストに印刷されます。それ以外のものは印刷されません。まず LANGLVL(SAA) が有効になります。次いでインクルード・ファイル Second 全体を対象に LANGLVL(SAA2) が有効になります。

```
┌──────────Source Program──────────────────────────┐
│                                                  │
│ *process langlvl(saa);                           │
│                                                  │
│  statement 1;                                    │
│  statement 2;                                    │
│  %include First;   /* statement 3 */             │
│  ┌──────────First──────────────────────────────┐ │
│  │  %push;                        /* F1 */     │ │
│  │  %noprint;                     /* F2 */     │ │
│  │  dcl A entry (ptr,fixed bin); /* F3 */      │ │
│  │  statement F4;                              │ │
│  │  %include Second; /* stmt F5 */             │ │
│  │    ┌──────Second──────────────────────┐     │ │
│  │    │  %push;                   /* S1 */    │ │
│  │    │  %print;                  /* S2 */    │ │
│  │    │  %option langlvl(saa2);   /* S3 */    │ │
│  │    │  dcl B  entry (ptr,fixed bin)        │ │
│  │    │     options(byvalue);     /* S4 */    │ │
│  │    │  statement S5;                       │ │
│  │    │  %pop;                               │ │
│  │    └──────────────────────────────────────┘ │
│  │  statement F6;                              │ │
│  │  %pop;                                      │ │
│  └─────────────────────────────────────────────┘ │
│  statement 4;                                    │
└──────────────────────────────────────────────────┘
```

Second 内の %POP ディレクティブの次に元の設定値が復元されます。

PUT ステートメント

PUT ステートメントについては、 303 ページの『第 13 章 ストリーム指向データ伝送』を参照してください。

READ ステートメント

READ ステートメントについては、 293 ページの『第 12 章 レコード単位データ伝送』を参照してください。

RELEASE ステートメント

RELEASE ステートメントについては、 110 ページの『FETCH ステートメント』を参照してください。

RESIGNAL ステートメント

RESIGNAL ステートメントについては、 357 ページの『第 16 章 条件処理』を参照してください。

RETURN ステートメント

RETURN ステートメントについては、 134 ページの『RETURN ステートメント』を参照してください。

REVERT ステートメント

REVERT ステートメントについては、357 ページの『第 16 章 条件処理』を参照してください。

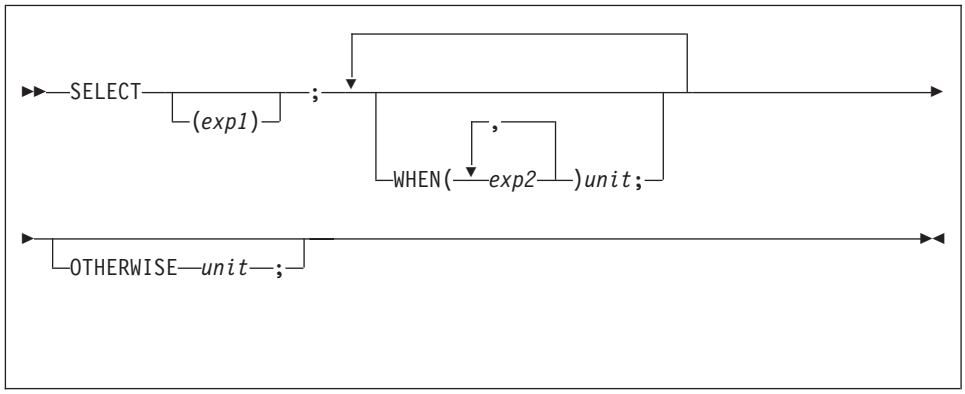
REWRITE ステートメント

REWRITE ステートメントについては、293 ページの『第 12 章 レコード単位データ伝送』を参照してください。

SELECT ステートメント

SELECT グループを使用すれば、複数経路の条件付き分岐を行うことができます。SELECT グループは、SELECT ステートメント、オプションの WHEN ステートメント (1 つまたは複数)、オプションの OTHERWISE ステートメント、および END ステートメントで構成されます。

注: 条件接頭語は、OTHERWISE ステートメントでは使用することはできません。



省略形: OTHERWISE の場合は OTHER

SELECT (exp1)

SELECT ステートメントとこれに対応した END ステートメントが、 SELECT グループと集合的に呼ばれるステートメントのグループを区切ります。
SELECT ステートメント内の式が計算され、その値が保管されます。

WHEN (exp2, exp2, ...) unit

1 つまたは複数の式を指定します。これらの式は計算されて、 SELECT ステートメントにより保管された値と比較されます。

保管されたものと同じ値を持つ式が見付かると、その時点で、 WHEN ステートメント内の式の計算は終了し、その WHEN ステートメントのユニットが実行されます。保管されたものと同じ値を持つ式が見付からなかった場合は、OTHERWISE ステートメントのユニットが実行されます。

WHEN ステートメントにラベルを付けることはできません。

OTHERWISE unit

先行の WHEN ステートメントのすべてのテストが失敗した場合に実行されるユニットを指定します。

OTHERWISE ステートメントを省略した場合、SELECT グループが実行された結果としてどのユニットも選択されなかったときは、 ERROR 条件が起こります。

OTHERWISE ステートメントにラベルまたは条件接頭語を付けてはなりません。

unit
各ユニットは、有効な単一ステートメント、グループ、または開始ブロックのいずれかです。 DECLARE、DEFAULT、END、ENTRY FORMAT、PROCEDURE、および %statement ステートメントは無効です。それぞれのユニットには、制御の移動を指定するステートメント (たとえば、GO TO) が含まれていてもかまいません。そのようなステートメントを指定すれば、 SELECT ステートメントの正規の順序を指定変更することができます。

exp1 を省略した場合、それぞれの *exp2* が計算され、必要に応じてビット・ストリングに変換されます。そのストリングのいずれかのビットが '1'B であれば、その WHEN ステートメントのユニットが実行されます。すべてのビットが 0 であるか、そのストリングがヌルであれば、 OTHERWISE ステートメントのユニットが実行されます。

WHEN または OTHERWISE ステートメントのユニットが実行されたあとは、そのユニット内で制御の通常の流れが変更されたのでない限り、 SELECT グループの次のステートメントに制御が渡されます。

exp1 を指定した場合は、それぞれの *exp2* は、次の比較式がスカラー・ビット値を持たねばなりません。

```
(exp1) = (exp2)
```

exp1 と *exp2* のいずれでも、配列オペランド、構造オペランド、および共用体オペランドを使用することはできません。

例

次の例で、E、E1 などは式を表します。 SELECT ステートメントに制御が到達すると、式 E が計算され、その値が保管されます。次に、WHEN ステートメント内のそれぞれの式が順に (書かれている順に) 計算され、その値が E の値と比較されます。

E の値と等しい値が見付かると、対応する THEN ステートメントの後に指定されている処置が行われます。先の WHEN ステートメント式は計算されません。

WHEN ステートメントのどの式も、SELECT ステートメントの式と等しくない場合は、 OTHERWISE ステートメントの後に指定されている処置が実行されます。

```
select (E);
  when (E1,E2,E3) action-1;
  when (E4,E5) action-2;
  otherwise action-n;
end;
N1: next statement;
```

exp1 を省略した場合の例を以下に示します。

```
select;
  when (A>B) call Bigger;
  when (A=B) call Same;
  otherwise call Smaller;
end;
```

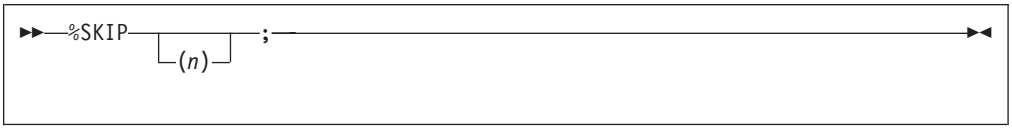
SELECT グループ内に WHEN ステートメントが 1 つもない場合は、OTHERWISE ステートメント内の処置が無条件に実行されます。 OTHERWISE ステートメントを省略した場合に、 SELECT グループが実行され、WHEN ステートメントのどのユニットも選択されなかったときは、ERROR 条件が起こります。

SIGNAL ステートメント

SIGNAL ステートメントについては、 357 ページの『第 16 章 条件処理』を参照してください。

%SKIP ディレクティブ

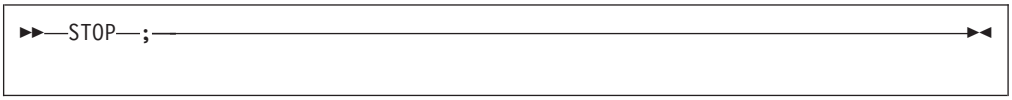
%SKIP ディレクティブは、コンパイラーのソース・リストで指定の行数をブランクのままにします。



n スキップしたい行数を指定します。 1 から 999 までの範囲内の整数でなければなりません。 *n* が省略されると、デフォルトは 1 になります。*n* がそのページに残っている行数よりも大きい場合、%SKIP ディレクティブの代わりに、同じ結果になる %PAGE ディレクティブが実行されます。

STOP ステートメント

STOP ステートメントは、現在のアプリケーションを停止します。



WAIT ステートメント

WAIT ステートメントについては、 389 ページの『第 18 章 マルチスレッド化機能』を参照してください。

WHEN ステートメント

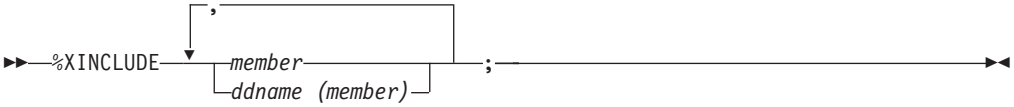
WHEN ステートメントについては、 233 ページの『SELECT ステートメント』を参照してください。

WRITE ステートメント

WRITE ステートメントについては、 293 ページの『第 12 章 レコード単位データ伝送』を参照してください。

%XINCLUDE ステートメント

%XINCLUDE ディレクティブは、外部のテキストを、(以前に組み込まれていなければ) ソース・プログラムに組み入れるために使用します。



第 10 章 ストレージ制御

ストレージのクラス、割り振り、および割り振り解除 237
静的ストレージとその属性 239
自動ストレージとその属性 239
被制御ストレージとその属性 241
被制御変数のための ALLOCATE ステートメント 241
被制御変数のための FREE ステートメント . . 243
被制御変数の複数世代 244
アスタリスクの表記法 244
調節可能エクステント 245
被制御変数のための組み込み関数. 245
基底付きストレージとその属性 245
ロケーター・データ 247
POINTER 変数とその属性 250
基底付き変数のための組み込み関数 250
基底付き変数のための ALLOCATE ステートメント 251
基底付き変数のための FREE ステートメント 252
REFER オプション (自己定義データ) 253
区域データとその属性 255
オフセット・データとその属性 256
区域変数のための組み込み関数 258
区域の割り当て. 258
区域の入出力 258
リスト処理 259
ASSIGNABLE 属性と NONASSIGNABLE 属性 . . 260
NORMAL 属性と ABNORMAL 属性 261
BIGENDIAN 属性と LITTLEENDIAN 属性 . . . 261
HEXADEC 属性と IEEE 属性 262
CONNECTED 属性と NONCONNECTED 属性 . . 263
DEFINED 属性と POSITION 属性 264
非連結ストレージ 266
単純定義 266
iSUB 定義 267
ストリング・オーバーレイ定義 268
POSITION 属性. 269
INITIAL 属性 270
配列変数の初期設定 273
共用体の初期設定 274
静的変数の初期設定 274
自動変数の初期設定 275
基底付き変数と被制御変数の初期設定 275
例 275

変数は、すべてストレージを必要とします。変数に指定する属性は、必要なストレージの大きさ、およびそのストレージをどのように解釈するかを表します。次の例では、X への参照は、すなわち、固定小数点 2 進数として解釈される値を含むストレージの一部への参照です。

```
dcl X fixed binary(31,0) automatic;
```

X が自動であるため、このストレージはその宣言ブロックが活動化されたときに割り振られ、割り振られたストレージは、ブロックが非活動化されるまで有効です。

ストレージのクラス、割り振り、および割り振り解除

ストレージの割り振りとは、変数で表されるデータ項目を内部で記録しておくために、ストレージの 1 つの区域をその変数に関連付ける作業のことです。ある変数にストレージが関連付けられると、その変数は*割り振られた* ことになります。

変数の割り振りは、*静的に* (プログラムの実行前に) 行われることもあれば、*動的に* (プログラムの実行中に) 行われることもあります。 静的に割り振られた変数は、アプリケーション・プログラムの実行中ずっと割り振られたままになっています。動的に割り振られた変数は、その変数を含んでいるブロックが終了するとき、またはアプリケーションから明示的に要求されたときに、占有していたストレージを解放します。

変数に割り当てられたストレージ・クラスによって、その変数に適用されるストレージの制御の程度、およびその変数が持つストレージの割り振り方法やその解放方法は異なります。 4 つのストレージ・クラスがあります。自動、静的、被制御、お

よび基底付きです。ストレージ・クラスは、明示的に、または暗黙に、もしくはコンテキスト宣言でその対応する属性を使用することによって割り当てることができます。

- AUTOMATIC を指定すると、そのストレージ宣言を持つブロックに入るたびに、ストレージが割り振られます。 そのブロックから出る時点で、割り振られていたストレージが解放されます。ブロックが、再帰的に呼び出されるプロシージャーの場合は、直前に割り振られたストレージは、このプロシージャーの入り口でスタックに入れられます (プッシュダウン)。ストレージの最新の割り振りは、各世代が終了した時に、再帰プロシージャー内でスタックから除去されます (ポップアップ)。 (スタック処理の際のプッシュダウンとポップアップについては、 107 ページの『再帰的プロシージャー』を参照してください。)
- STATIC を指定すると、プログラムがロードされるときに、ストレージが割り当てられます。 プログラムの実行が完了するまで、そのストレージは解放されません。取り出されたプロシージャーの場合は、そのプロシージャーが解放されるまで、そのストレージは解放されません。
- CONTROLLED を指定すると、ALLOCATE ステートメントおよび FREE ステートメントを使用してストレージの割り当ておよび解放を制御できます。 同一プログラムで同一の被制御変数を複数回割り振り、しかもその間に解放処理をしない場合は、その変数のその間の複数世代がスタックされます。最新の世代を解放することによってのみ、初期の世代にアクセスすることができます。
- BASED は、CONTROLLED と同じように、ユーザーがストレージの割り振りと解放を制御することを指定します。 CONTROLLED と異なるのは、複数回割り振った場合に、割り振られたストレージはスタックされるのではなく、どれも常に使用可能である点です。それぞれの割り振りは、ポインター変数の値で識別することができます。また基底付き変数ストレージの区域と関連づけたり、オフセット変数の値によって識別することができるという点でも異なります。

  区域外にある基底付き変数は、ALLOCATE 組み込み関数を使用すれば割り振りが、また PLIFREE 組み込みサブルーチンを使用すれば解放が可能です。基底付き変数は、AUTOMATIC 組み込み関数を使用しても割り振ることができます。この組み込み関数を使用して割り振った変数は、その変数の割り振られているブロックが終了すると、自動的に解放されます。

要素変数、配列変数、大構造変数、および共用体変数の場合は、ストレージ・クラス属性を明示的に宣言することができます。配列変数、大構造変数、および共用体変数に対して宣言されたストレージ・クラスは、その配列、大構造体、または共用体のすべてのエレメントに適用されます。

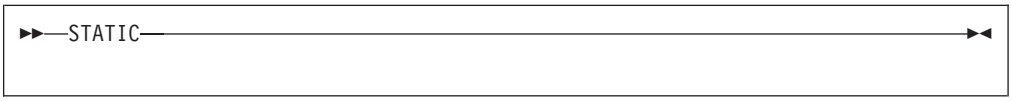
ストレージ・クラス属性は、次のものについては指定することができません。

- CONDITION 条件
- 定義済みデータ項目
- 入り口定数
- ファイル定数
- フォーマット定数
- DEFINE ステートメント内に定義された ID
- ラベル定数
- 構造体または共用体のメンバー
- 名前付き定数

変数へのストレージの割り振りは、PL/I によって管理されます。ユーザーは、ストレージのどこに割り振りを行うかを指定することはできません。ただし、変数が既存の AREA に割り振られるように指定することはできます。詳細については、255 ページの『区域データとその属性』を参照してください。

静的ストレージとその属性

STATIC 属性を持つと宣言された変数は、プログラムの実行の前に割り振られます。それらは、プログラムが終了するまで割り振られたままになっています。静的変数の割り振りについての制御は、実行中のプログラムには与えられません。



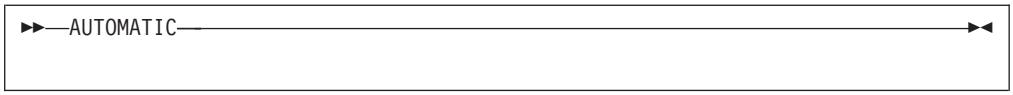
STATIC は外部変数のデフォルトですが、内部変数が STATIC であってもかまいません。またこれは、パッケージ内で宣言された変数のデフォルトでもあり、いずれのプロシージャーの範囲内でもありません。静的変数を参照できるかどうかについては、有効範囲に関する通常の規則に従います。次の例では、変数 X は、プログラムの実行中は継続して割り振られていますが、プロシージャー B の中、または B に含まれているブロックの中だけで参照することができます。変数 Y は STATIC 属性を獲得し、またプログラムが実行している間割り振られています。

```
Package: Package exports (*);
  dcl Y char(10);

  A: proc options(main);
    B: proc;
      declare X static internal;
    end B;
  end A;

  C: proc;
    Y = 'hello';
  end C;

end Package;
```

INITIAL 属性を使用して静的変数を初期設定する場合、初期値は制限付き式でなければなりません。またエクステントも、制限付き式として指定されなければなりません。

自動ストレージとその属性

自動変数は、それが宣言されているブロックに制御が移されたときに割り振られます。自動変数の割り振りは、プログラムの実行中に何回も行うことができます。ユーザーは、プログラムのブロック構造体をどのように設計するかによって、自動変数の割り振りを制御することができます。



省略形: AUTO

AUTOMATIC はデフォルトです。自動変数は常に内部変数です。

以下の例では、プロシージャー B が呼び出されるたびに、変数 X と Y はストレージを割り振られます。 B が終了するとストレージが解放され、 X および Y に入っていた値は失われます。

```
A: proc;
:
call B;
B: proc;
declare X,Y auto;
:
end B;
:
call B;
```

解放されたストレージは、別の変数への割り振りのために使用可能になります。このように、ブロック (プロシージャー・ブロックまたは開始ブロック) がアクティブである間は、そのブロック内で AUTOMATIC と宣言されたすべての変数にストレージが割り振られています。ブロックが非アクティブであれば、そのブロック内の自動変数にストレージは割り振られていません。再帰的に呼び出されるか、または複数のプログラムによって呼び出されるプロシージャーを除いて、個々の自動変数の割り振りは 1 つだけ存在することができます。

自動変数のエクステントは、式で指定することができます。したがって、必要なときに、特定量のストレージを割り振ることができます。次の例では、文字ストリング STR の長さは、プロシージャー B が呼び出されたときに存在していた変数 N の値によって定義された長さになります。

```
A: proc;
declare N fixed bin;
:
B: proc;
declare STR char(N);
```

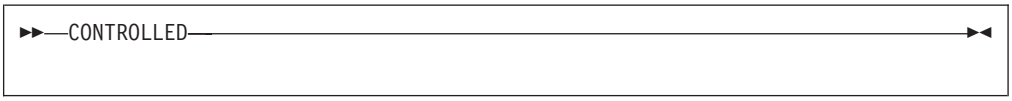
DECLARE ステートメントが同じブロック内にある場合、 PL/I は変数 N を制限付き式または初期設定済みの静的変数のいずれかに初期設定しておかなければなりません。次の例では、割り振られた長さは Str1 に対して正しく割り振られますが、Str2 には正しくありません。 PL/I では、このタイプの宣言の依存性は解決できません。

```
dcl N fixed bin (15) init(10),
M fixed bin (15) init(N),
Str1 char(N),
Str2 char(M);
```

被制御ストレージとその属性

CONTROLLED として宣言された変数は、ALLOCATE ステートメントで指定されたときにだけ割り振られます。被制御変数は、その変数を指定している FREE ステートメントに制御権が到達するか、またはプログラムが終了するまで、割り振られたままになっています。

被制御変数は、プログラムのブロック構造体から、部分的に独立しています (ただし、完全に無関係というわけではありません)。被制御変数の有効範囲は、INTERNAL または EXTERNAL のいずれでもかまいません。 INTERNAL と宣言されている場合の有効範囲は、その変数を宣言しているブロックと、そのブロックに含まれているブロックです。まだ割り振られていない被制御変数を参照すると、エラーになります。



省略形: CTL

以下の例では、変数 X はプロシージャー B 内と、プロシージャー A の CALL ステートメントの実行に続く部分内で有効に参照することができます。

```
A: proc;
dcl X controlled;
call B;
⋮
B: proc;
allocate X;
⋮
end B;
end A;
```

一般に、被制御変数が有用なのは、調整可能なエクステントを持つ大きなデータ集合がプログラムに必要なときです。次の例のステートメントは、入力データに応じて必要なだけのストレージを割り振り、不要になったときにそのストレージを解放します。

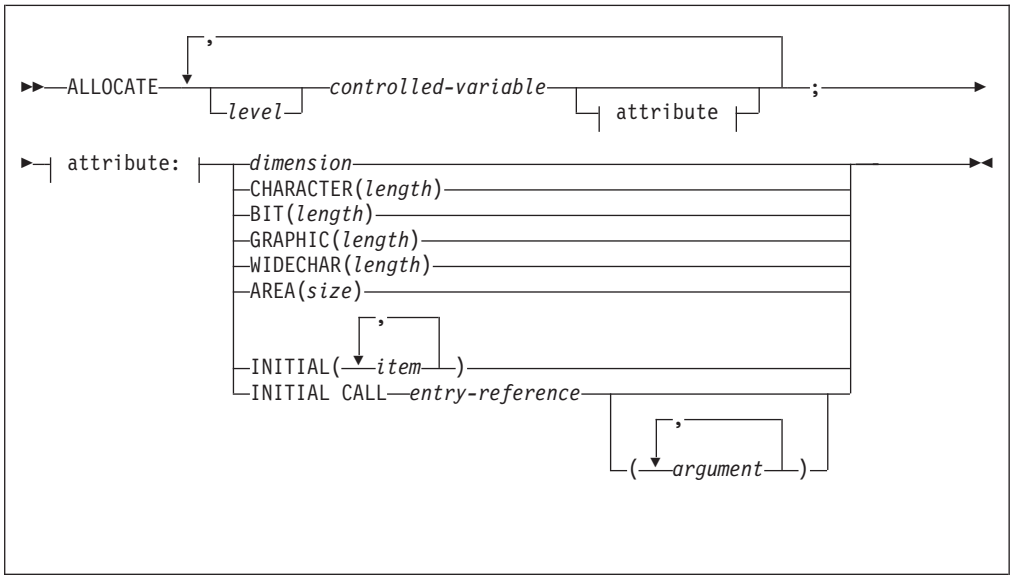
```
  dcl A(M,N) ctl;
  get list(M,N);
  allocate A;
  get list(A);
⋮
  free A;
```

この方法を採用すると、ブロックを活動化したり終了したりする必要がないので、開始ブロックを設ける方法よりも効率がよくなります。

被制御変数のための ALLOCATE ステートメント

ALLOCATE ステートメントは、プロシージャー・ブロックの境界とは関係なく、被制御変数にストレージを割り振ります。同時に、被制御パラメーターも割り振ることができます。制御されている配列の境界、制御ストリングの長さ、制御域のサイ

ズは、その初期値と同様に ALLOCATE ステートメントで指定することができます。



省略形: ALLOC

level
レベル番号を示します。レベル番号を指定しない場合、指定される被制御変数はレベル 1 変数でなければなりません。

controlled-variable
被制御変数または制御された小構造体のエレメントを指定します。大構造体以外の構造体エレメントは、エレメントを含む大構造体全体の相対構造化が、その構造のための DECLARE ステートメント内にあるかのように示される場合にのみ示すことができます。この場合は、次元属性は、その次元属性で宣言されているすべての名前に指定されていなければなりません。

被制御変数と基底付き変数の両方を、同一の ALLOCATE ステートメントで割り振ることができます。基底付き変数の構文については、251 ページの『基底付き変数のための ALLOCATE ステートメント』を参照してください。

配列の境界、ストリングの長さ、および区域のサイズ (エクステント) は、ALLOCATE ステートメントの実行時に計算されます。

- ALLOCATE ステートメントまたは DECLARE または DEFAULT ステートメントは、変数について、すべての必要な次元、サイズ、または長さ属性 (エクステント) を指定しなければなりません。 DECLARE ステートメントから取られた式の計算は、ALLOCATE ステートメントの実行時に割り込み可能になっている条件を使用して、割り振り時に行われます。ただし、式の中の名前は、その DECLARE または DEFAULT ステートメントを有効範囲の中に持つ変数を参照しています。
- 1 つの ALLOCATE ステートメントに、境界値、長さ、またはサイズを明示的に指定すると、その変数について DECLARE ステートメントで指定した値を指定変更します。

- 1 つの ALLOCATE ステートメントに、境界値、長さ、またはサイズをアスタリスクを指定して指定すると、そのエクステントは現行世代から取得されます。変数の世代が存在しない場合は、エクステントは未定義となり、プログラムはエラーになります。
- ALLOCATE または DECLARE ステートメントのいずれかで、割り振ろうとしている変数の参照を含む式を使って、境界値、長さ、またはサイズを指定している場合は、変数の最新の世代の値を使用して式が計算されます。 以下に例を示します。

```
declare X(N) fixed bin ctl;
N = 20;
allocate X;
allocate X(X(1));
```

  X の最初の割り振りでは、 DECLARE ステートメントと N = 20; で、上限が指定されます。 2 番目の割り振りでは、 X の最初の世代の最初のエレメントの値によって、上限が指定されます。

次元属性は、宣言したものと同じ次元数を指定しなければなりません。次元属性は他の任意の属性とともに指定することができ、指定する属性の最初のものでなければなりません。 以下に例を示します。

```
declare X(M) char(N) controlled;
M = 20;
N = 5;
allocate X(25) char(6);
```

BIT、CHARACTER、GRAPHIC、WIDECHAR、および AREA 属性は、同じ属性を持つ変数についてのみ、それぞれ指定することができます。

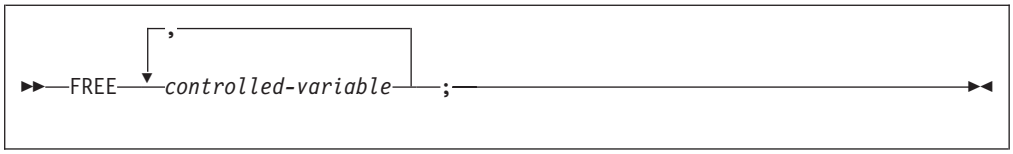
変数が DECLARE または ALLOCATE ステートメントのいずれかに INITIAL 属性を持つ場合は、初期値は割り振り時に変数に割り当てられます。 INITIAL 属性の式または CALL オプションは、 ALLOCATE ステートメントで使用可能な条件を使用して、割り振り時に計算されます。ただし、名前は宣言の環境で解釈されます。INITIAL 属性が DECLARE と ALLOCATE ステートメントの両方に指定される場合は、ALLOCATE ステートメントの INITIAL 属性が使用されます。初期設定時に、割り振られる変数が参照されている場合、その変数の新しい世代を参照することになります。初期設定の詳細については、270 ページの『INITIAL 属性』を参照してください。

ALLOCATE ステートメントの実行時に行われる計算 (たとえば、INITIAL 属性内の式の計算) は、相互に依存し合ってはなりません。

被制御変数のためのストレージが使用可能ではない場合、STORAGE 条件が発生します。

被制御変数のための FREE ステートメント

FREE ステートメントは、被制御変数に割り振られているストレージを解放します。解放されたストレージは、別の割り振りに使用できます。このとき、1 つ前に割り振られた被制御変数が使用可能になり、このあとでその変数を参照すると、その割り振りを参照することになります。



controlled-variable
レベル 1 の添え字なし変数です。

基底付き変数と被制御変数の両方を、同一の FREE ステートメントで解放することができます。基底付き変数の構文については、252 ページの『基底付き変数のための FREE ステートメント』を参照してください。

暗黙の開放

被制御変数は、必ずしも FREE ステートメントで明示的に解放する必要はありません。しかし、被制御変数を (FREE ステートメントで) 明示的に解放することをお勧めします。

被制御ストレージはすべて、プログラムの終了時に解放されます。

被制御変数の複数世代

ストレージがすでに割り振られている (まだ解放されていない) 変数を指定する ALLOCATE ステートメントが実行されると、その変数のストレージはスタックに入れられ (プッシュダウン) ます。つまり、ストレージがスタックされます。スタッキングが行われると、その変数の新しい世代データが作られます。この新しい世代が現行世代となり、現行世代が解放されない限り、前の世代を直接にアクセスすることはできません。 FREE ステートメントによって、または変数にストレージを割り振ったプログラムが終了したために、その変数のストレージが解放されると、1 つ前のストレージがスタックから除去 (ポップアップ) されます。つまり、スタックから取り除かれます。

アスタリスクの表記法

ALLOCATE ステートメントでは、次元、長さ、またはサイズがアスタリスクで示されたときは、1 つ前の世代から継承されます。配列の場合、アスタリスクは、配列の 1 つではなくすべての次元に使用しなければなりません。 以下に例を示します。

```
dcl X(M,N) char(A) ctl;
  M=10;
  N=20;
  A=5;

allocate X;
allocate X(10,10);
allocate X(*,*);
```

X の最初の世代は (10,20) の境界を持ちます。2 番目と 3 番目の世代は (10,10) の境界を持ちます。 X の各世代のエレメントは、すべて長さ 5 の文字ストリングです。

DECLARE ステートメントでもアスタリスク表記を使用することができますが、その場合は、異なる意味を持ちます。 以下に例を示します。

```
dcl Y char(*) ctl,
N fixed bin;

N=20;
allocate Y char(N);
allocate Y;
```

文字ストリング Y の長さが ALLOCATE ステートメントで指定されていない場合は、直前の世代からそれを取ります。その場合は、Y には指定された長さが与えられます。そのため、ユーザーはストリングの長さの指定を、ストレージの実際の割り振りまで延期することができます。

調節可能エクステント

被制御スカラー、配列、および構造体や共用体のメンバーは、調整可能な配列エクステント、ストリング長、および区域サイズを持つことができます。たとえば、次の例において、構造体が割り振られると、 A.B のエクステントは 1 から 10 になり、 A.C は最大の長さが 5 の VARYING 文字ストリングになります。

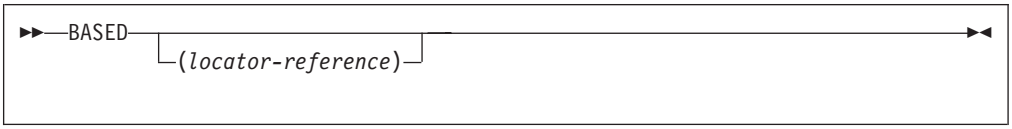
```
dcl 1 A ctl,
2 B(N:M),
2 C char(*) varying;
N = -10;
M = 10;
alloc 1 A,
2 B(1:10),
2 C char(5);
free A;
```

被制御変数のための組み込み関数

ALLOCATION 組み込み関数を使用すれば、特定の被制御変数に割り振られた世代の数を判別することができます。変数が割り振られていない場合は、ゼロの値が戻されます。

基底付きストレージとその属性

基底付き変数の宣言は世代 (すなわち必要なストレージの量とその属性) の宣言です。 (基底付き変数は、主記憶域での世代の位置を示しません。) 世代の位置は、ロケーター変数で示します。割り振られていない基底付き変数の値を参照すると、エラーになります。



locator-reference
データの位置を指定します。

基底付き変数を参照するとき、データ属性と位置合わせ属性についてはその基底付き変数の属性が使用されますが、データの位置は、その変数を修飾しているロケーター変数が示します。

基底付き変数に EXTERNAL 属性を付けることはできませんが、基底付き変数のロケーター参照には、基底付きストレージ・クラスを含む任意のストレージ・クラスを付けることができます。

REFER オプションを使用すれば、調整可能な区域サイズ、配列の境界、およびストリングの長さの指定を持つ基底付き構造体もしくは共用体を宣言することができます。 253 ページの『REFER オプション (自己定義データ)』を参照してください。 基底付きの VARYING ストリングや VARYINGZ ストリングの最大長は、それらのストリングがオーバーレイするストリングの最大長に等しくなければなりません。 以下に例を示します。

```
declare A char(50) varying based(Q),
        B char(50) varying;
        Q=addr(B);
```

基底付き VARYING ストリングは、VARYING ストリング上にのみオーバーレイすることができ、基底付き VARYINGZ ストリングは、VARYINGZ ストリング上にのみオーバーレイすることができます。

基底付き変数用のストレージは、ALLOCATE ステートメント、ALLOCATE 組み込み関数、 AUTOMATIC 組み込み関数、または LOCATE ステートメントを使用して割り振ることができます。 基底付き変数は、(SET オプションを指定した) READ ステートメント、FETCH ステートメント、または ADDR 組み込み関数を使用することによって、既存のデータをアクセスするときにも使用できます。

基底付き AREA 変数は、ALLOCATE ステートメントを使用して割り振ることができます。PL/I は割り振りを行うときに自動的に区域を EMPTY に初期設定します。ただし、ALLOCATE または AUTOMATIC 組み込み関数を使用して区域変数にストレージを獲得すると、ストレージを獲得したあとでその変数に EMPTY を割り当てる必要があります。

割り振られた世代の位置はロケーター変数で示されるので、適切なロケーター値を使用すれば、プログラム内の任意の場所で基底付き変数のどの世代でも参照することができます。次の例では、参照が明示的に修飾されているときを除いて、X を参照するときにはロケーター変数 P によって X の記憶場所が示されることを宣言しています。

```
dcl X fixed bin based(P);
```

このようにしてロケーター参照に関連付けても、その関連付けは永続的なものではありません。このロケーター参照を使用して別の基底付き変数の位置を示すこともできますし、別のロケーター参照を使用して変数 X の別の世代を示すこともできます。ロケーター参照を指定せずに基底付き変数を宣言した場合は、その基底付き変数を参照するときには必ず、ロケーターを使って明示的に修飾する必要があります。

次の例では、配列 A と配列 C は同じストレージを指します。エレメント B とエレメント C(2,1) も同じストレージを指します。

```
dcl A(3,2) character(5) based(P),
    B char(5) based(Q),
    C(3,2) character(5);
P = addr(C);
Q = addr(A(2,1));
```

注: 基底付き変数をこの方法でオーバーレイするときは、新しいストレージは割り振られません。基底付き変数は、オーバーレイされる変数 (この例では C(3,2)) と同じストレージを使用します。

DEFINED 属性および UNION 属性のいずれを使用しても変数をストレージにオーバーレイすることができますが、DEFINED および UNION によるオーバーレイは永続的なものとなります。ロケーター参照によって基底付き変数をオーバーレイしたときは、ロケーター変数に新しい値を割り当てれば、プログラムの実行中いつでも、その結びつきを変更できます。

DEFINED 属性および UNION 属性の詳細については、264 ページの『DEFINED 属性と POSITION 属性』および 186 ページの『共用体』を参照してください。

基底付き変数に INITIAL 属性を指定してもかまいません。初期値が割り当てられるのは、ALLOCATE または LOCATE ステートメントによって基底付き変数が明示的に割り振られたときだけです。

ロケーター・データ

ロケーター・データには 2 つのタイプ、ポインターとオフセットがあります。

ポインター変数 の値は、ストレージの位置のアドレスです。いくつかの異なる記憶場所でストレージを割り振られた変数を参照するときは、この値を使って参照を修飾することができます。

オフセット変数 の値は、区域変数内の位置を指定し、その区域がストレージの別の場所に割り当てられている場合には引き続き有効です。

ロケーター値は、ロケーター変数にだけ割り当てることができます。オフセット値がオフセット変数に割り当てられるときは、OFFSET 属性で指定された区域変数は無視されます。

ロケーターの変換

ロケーター・データをほかのデータ・タイプに変換することはできません。ただし、次の場合は変換することができます。

- BINARYVALUE、POINTERVALUE、および OFFSETVALUE 組み込み関数を使用して、REAL FIXED BINARY (p,0) との間で行われる変換。
- POINTER および OFFSET 組み込み関数を明示的または暗黙的に使用して、ポインターとオフセット間で行われる変換。

オフセット変数が参照されると、OFFSET 属性およびオフセット変数で指定された区域変数のアドレスによって、オフセット変数はポインター値に暗黙に変換されます。 POINTER 組み込み関数を使用すれば、オフセットをポインター値に明示的に変換することができます。次の例では、ステートメントは、区域 B 内のオフセット O によって識別される基底付き変数の記憶位置を示すポインター値を、P に割り当てます。

```
dcl P pointer, O offset(A),B area;
P = pointer(O,B);
```

この区域変数は、オフセット変数に関連付けられた区域変数とは異なっているので、そのオフセット値が別の区域で有効であるかを確認しなければなりません。たとえば、区域 A が区域 B に割り当てられたあとでこの関数が呼び出された場合は、そのオフセット値は有効です。

OFFSET 組み込み関数は、POINTER 組み込み関数とは対照的に、与えられたポインターと区域から得られたオフセット値を返します。この場合、そのポインター値は、その区域内での基底付き変数の位置を示していなければなりません。

ポインター値は、そのポインター値およびその区域のアドレスを使用して、オフセットに変換されます。したがって、この変換ができるのは、OFFSET 属性内で指定された区域内のアドレスに結び付けられたポインター値に限られます。

NULL 組み込み関数または SYSNULL 組み込み関数の値に割り当てる場合を除いて、区域に関連付けられていないオフセット変数への変換や、このオフセット変数からの変換を試みると、エラーになります。

複数割り当ての場合、ロケーターは暗黙に変換されることはありません。

ロケーター参照

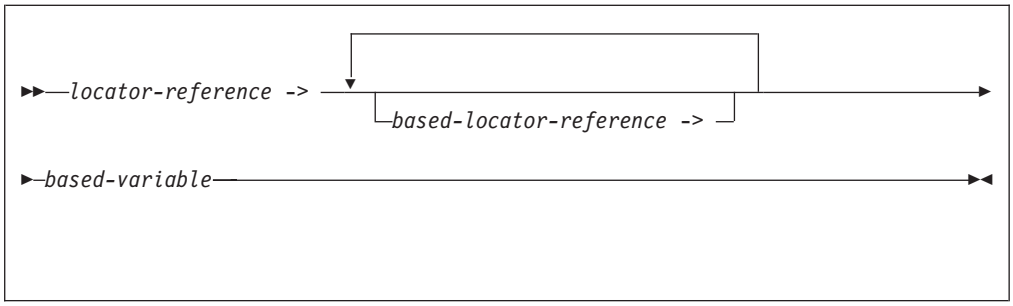
ロケーター参照とは、ロケーター変数 (修飾付きまたは添え字付きでもよい) または、ロケーター値を返す関数参照のいずれかのことです。

ロケーター参照は、次のように使用することができます。
- 基底付き変数の宣言に関連付けて、ロケーター修飾子として使用できます。
- 比較演算で使用できます。たとえば、IF ステートメント内で使用できます。
- プロシージャー参照内で引数として使用することができます。

PL/I はオフセットをポインター値に暗黙に変換するため、オフセット参照とポインター参照のどちらも使用することができます。

ロケーター修飾

ロケーター修飾とは、ロケーター基底付き変数の特定の世代を識別するために、1 つ以上のロケーター参照を基底付き参照に関連付けることです。これを、ロケーターで修飾された参照といいます。複合記号 -> は、「～で修飾された」または「～を指す」という意味を表します。明示的に修飾された参照の構文図を次に示します。



locator-reference
based-locator-reference
データの位置を指定します。

次の例では、X は基底付き変数であり、P はロケーター変数であり、Q は基底付きロケーター変数です。

```
P -> Q -> X
```

この参照は、基底付きロケーター Q、およびロケーター P の値によって識別される X の世代を意味しています。 X と Q は、*明示的にロケーター修飾されている* といいます。

複数のロケーター修飾子を使用している場合、それらの値の計算は順に左から右へ行われます。

基底付き変数の参照を*暗黙に修飾する* こともできます。暗黙に修飾された基底付き変数の世代を判別するために使われるロケーター参照は、基底付き変数と一緒に宣言されたものです。次の例では、ALLOCATE ステートメントは、ポインター変数 P の値をセットします。したがって、X を参照すると、割り振られたストレージが参照されます。

```
dcl X fixed bin based(P) init(0);
allocate X;
X = X + 1;
```

代入ステートメントで X を参照していますが、これは P によって暗黙にロケーターで修飾されます。 X を参照するとき、次の例のように明示的にロケーターで修飾することもできます。

```
P->X = P->X + 1;
```

次の代入ステートメントが実行されると、上記の場合と同じ結果になります。

```
Q = P;
Q->X = Q->X + 1;
```

基底付き変数と一緒に宣言されるロケーターも基底付きであってもかまいません。したがって、ロケーター修飾子のチェーンになることがあります。たとえば、次のようなポインターおよび基底付き変数を使用することができます。

```
declare (P(10),Q) pointer,
         R pointer based (Q),
         V based (P(3)),
         W based (R),
         Y based;
allocate R,V,W;
```

先のように宣言し、割り振りを行った場合、下記の参照はすべて正しい参照です。

```
P(3) -> V
V
Q -> R -> W
R -> W
W
```

最初の 2 つの参照は同じことを意味し、その次の 3 つも同じことを意味します。Y を参照するときは、それを修飾するロケーター変数を指定しなければなりません。

ロケーター修飾のレベル

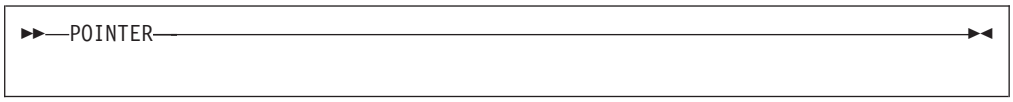
基底付き変数を修飾するポインターは、1 レベルのロケーター修飾を表します。オフセットは、区域内で暗黙に修飾されているので、2 レベルのロケーター修飾を表

します。レベルの数は、添え字付きのロケーターであるかどうか、もしくは構造体または共用体のエレメントであるかどうかとは無関係です。次の例では、X、P -> X、および Q -> P -> X という参照は、 3 レベルのロケーター修飾を表しています。

```
declare X based (P),
        P pointer based (Q),
        Q offset (A);
```

POINTER 変数とその属性

ポインター変数は、基底付き変数の宣言内に書いたとき、ロケーター修飾子として書いたとき、BASED 属性内に書いたとき、あるいは ALLOCATE、LOCATE、READ、または FETCH の各ステートメントの SET オプション内に書いたときにコンテキスト上から宣言されます。ポインター変数を明示的に宣言することもできます。



省略形: PTR

基底付き変数の世代をもはや識別しないポインター変数、たとえば、基底付き変数がすでに解放された場合の値は、未定義です。ポインターで修飾される変数を参照するときは、そのポインターに値を入れておかなければなりません。

基底付き変数のための組み込み関数

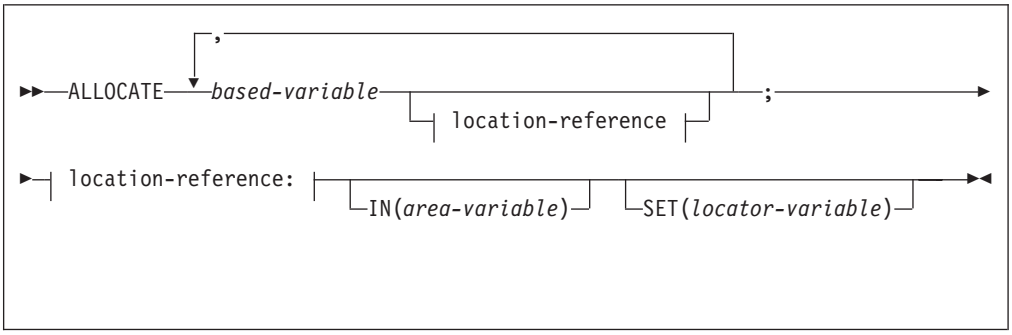
ALLOCATE 組み込み関数を使用すれば、基底付き変数のストレージを獲得できます。また PLIFREE 組み込みサブルーチンを使用すれば、そのストレージを解放できます。 AUTOMATIC 組み込み関数を使用しても、基底付き変数のストレージを獲得できますが、この関数を使用して獲得したストレージは、明示的に解放してはなりません。 AUTOMATIC 組み込み関数で割り振られたストレージは、それが割り振られているブロックが終了したときに自動的に解放されます。

ADDR 組み込み関数は、変数の先頭バイトを指すポインター値を返します。ENTRYADDR 組み込み関数は、入り口が呼び出されたときに、最初に実行される命令のアドレスを示すポインター値を返します。 NULL 組み込み関数は、PL/I のヌル・ポインターを返し、SYSNULL 組み込み関数は、システムのヌル・ポインターを返します。

注: NULL 組み込み関数と SYSNULL 組み込み関数は、同じように比較を行えますが、必ずしも等しいというわけではありません。アプリケーション・プログラムの作成において、関数の等価性に依存するような記述をすべきでは**ありません**。

基底付き変数のための ALLOCATE ステートメント

ALLOCATE ステートメントは、プロシージャー・ブロックの境界とは関係なく、基底付き変数にストレージを割り振り、その記憶位置を示すために使用されるロケーター変数をセットします。



省略形: ALLOC

based variable
レベル 1 の添え字なし変数

IN

ストレージを割り振る区域変数を指定します。区域の詳細については、 255 ページの『区域データとその属性』を参照してください。

SET

割り振られたストレージの位置を示す値にセットされるロケーター変数を指定します。 SET オプションを省略する場合は、基底付き変数の宣言で指定されたロケーターを使用しなければなりません。基底付き変数を宣言する場合の構文の詳細については、 245 ページの『基底付きストレージとその属性』および 247 ページの『ロケーター・データ』を参照してください。

基底付き変数と被制御変数の両方を、同一の ALLOCATE ステートメントで割り振ることができます。被制御変数の構文については、 241 ページの『被制御変数のための ALLOCATE ステートメント』を参照してください。

IN オプションを指定したときや、SET オプションでオフセット変数を指定したときは、区域内でストレージが割り振られます。これらのオプションの指定順序は任意です。区域内での割り振りの場合は、次のことに注意してください。

- 基底付き変数に必要なストレージがその区域内にないときは、AREA 条件が起こります。
- オフセット変数使用時に IN オプションを指定しない場合、オフセット変数を宣言するときに区域参照を指定しておく必要があります。

区域を使用しない場合は、ロケーター変数はポインター変数でなければなりません。基底付き変数のためのストレージが使用可能でない場合、STORAGE 条件が起こります。

基底付き変数が REFER を使用している場合、そのサイズは実行時に計算されます。計算の結果、値が大きすぎて FIXED BIN(31) 変数に適合しない場合、プログ

ラムはエラーとなり、訂正する必要があります。この状態では、STORAGE 条件は起こりません。その代わり、 ONCODE=3809 が設定された ERROR 条件が以下のいずれかの場合に起こります。

- SIZE 条件が割り込み可能になっている
- BASED 構造体がライブラリー呼び出しを介してマップされている

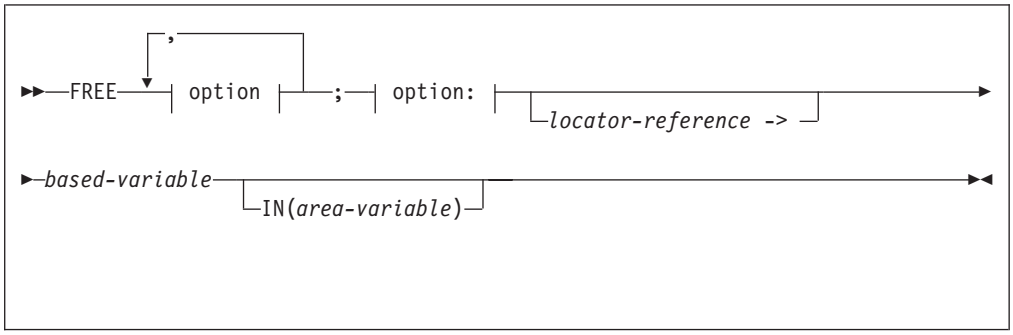
これらの条件のどちらも該当しない場合、予測不能な結果が生じます。

基底付き変数に割り振られるストレージの大きさは、その変数の属性に左右されるほか、その変数の次元、長さ、またはサイズが割り振り時に指定されるときは、それらにも左右されます。これらの属性は、基底付き変数の宣言から判別されます。

基底付きの構造体や共用体には、調整可能な配列境界、ストリングの長さ、または区域のサイズを含むことができます (253 ページの『REFER オプション (自己定義データ)』を参照)。基底付き変数の場合、エクステントとしてアスタリスクを表記することはできません。

基底付き変数のための FREE ステートメント

FREE ステートメントは、基底付き変数や被制御変数に割り振られているストレージを解放します。



locator-reference ->
基底付き変数の特定の世代を解放します。複合記号 -> は、「～で修飾された」または「～を指す」という意味を表します。基底付き変数が明示的にロケーターで修飾されていない場合は、BASED 属性で宣言されたロケーター変数が、解放されるデータの世代を識別するために使われます。ロケーターが 1 つも宣言されていない場合、このステートメントはエラーになります。

based variable
添え字なしの、レベル 1 の基底付き変数でなければなりません。

IN 解放されるストレージが区域内に割り振られている場合は、このオプションを指定するか、または基底付き変数を、対応する区域と一緒に宣言されたオフセットで修飾しなければなりません。基底付き変数が区域内に割り振られていない場合は、IN オプションを指定することはできません。割り振り時に区域を指定すると、基底付きストレージがターゲット区域の中に割り振られます。このような割り振りは、IN オプションでターゲット区域を指定すれば解放することができます。

基底付き変数と被制御変数の両方を、同一の FREE ステートメントで解放することができます。被制御変数の構文については、 243 ページの『被制御変数のための FREE ステートメント』を参照してください。

基底付き変数を使用してストレージを解放することができるのは、それと同じデータ属性を持つ基底付き変数に割り振られているストレージを解放する場合だけです。

解放されるストレージの大きさは、基底付き変数の属性 (ストレージ解放の時点での境界または長さ、あるいはその両方を含む) に左右されます。ユーザーは、この大きさが割り振られたストレージの大きさと一致しているかを判別する必要があります。変数が割り振られていない場合の結果は予測不能です。

暗黙の開放

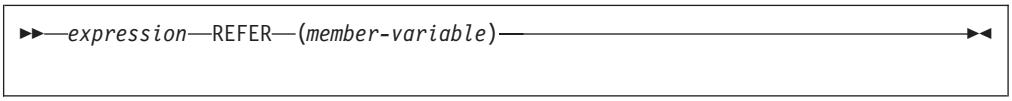
基底付き変数は、必ずしも FREE ステートメントで明示的に解放する必要はありません。しかし、(FREE ステートメントで) 基底付き変数を明示的に解放することをお勧めします。

基底付きストレージはすべて、プログラムの終了時に解放されます。

REFER オプション (自己定義データ)

自己定義構造体または共用体とは、それ自身のフィールドについての情報 (ストリングの長さなど) を含んでいる構造体または共用体です。このデータを処理できるように、基底付き構造または共用体を宣言することができます。ストリングの長さ、配列の境界、区域のサイズはすべて、構造体内または共用体内で宣言した変数 (*参照オブジェクト*) によって定義することができます。その構造体または共用体が割り振られる (ALLOCATE ステートメントまたは LOCATE ステートメントによって) とき、式の値が参照オブジェクト変数に割り当てられます。それ以外にこの構造体または共用体を参照したときは、参照オブジェクトの値が使われます。

基底付き構造体または共用体を宣言するときに REFER オプションを指定して、その構造体または共用体を割り振る時点で式の値を参照オブジェクトに割り当てること、およびその式の値で構造体内または共用体内の別の変数の長さ、境界、もしくはサイズを表すことができます。 REFER オプションで長さ、境界、またはサイズを指定する場合の構文は次のとおりです。



expression

この expression (式) の値は、構造体または共用体が (ALLOCATE もしくは LOCATE を使用して) 割り振られたときに、メンバーの長さ、境界、あるいはサイズを定義します。この式が計算され、FIXED BINARY (31,0) に変換されます。この式のオペランドとして使用する変数は、この REFER オプションを含んでいる構造体または共用体に属していてはなりません。

構造体または共用体への後続の参照では、REFER オプションのメンバーの長さ、境界、もしくはサイズを、*member-variable* (参照オブジェクト) の現行値から獲得します。

member-variable

参照オブジェクトは、次の規則に従わなければなりません。

- 同じレベル 1 の構造体または共用体のメンバーでなければならず、REFER オプションの中でそのオブジェクトを指定するメンバーの前になければなりません。
- 計算できるものでなければなりません。
- ロケーター修飾 (248 ページの『ロケーター修飾』を参照) をしたり、添え字を付けたりすることはできません。
- 配列の一部であってはなりません。

次の例の宣言では、基底付き構造体 STR が配列 Y とエレメント X で構成されていることを指定しています。

```
declare 1 STR based(P),
          2 X fixed binary(31,0),
          2 Y (L refer (X)),
        L fixed binary(31,0) init(1000);
```

STR が割り振られるとき、上限は、X に割り当てられている L の現行値にセットされます。ほかの場所 (P をセットするための READ ステートメントなどで) Y を参照すると、X に入っている制限値が使われます。

REFER オプションを持つメンバーに INITIAL 属性を指定すると、参照オブジェクトに値が割り当てられてから、メンバーの初期設定が行われます。

構造体または共用体の宣言で使用できる REFER オプションの数に制限はありません。

参照オブジェクトの値は、プログラムの実行中に変更してはなりません。参照オブジェクトの値が変更された場合、集合などを解放するエラーが発生します。

また、REFER エクステントを定義する式の中で使用する変数は、その REFER を使用する DECLARE を含むブロック (またはその親ブロックの 1 つ) 内で宣言されている必要があります。変数のいずれかが宣言されていない場合、その変数は暗黙宣言に関する通常の規則に従って暗黙的に宣言されます。つまり、DECLARE を含む最外部のブロックに、その変数の DECLARE が追加されます。

したがって、次のコードの中で、サブルーチン inner_proc 内での変数 m の宣言と割り当ては、ALLOCATE ステートメントに影響しません。 ALLOCATE ステートメントは、メイン・ブロックで暗黙的に宣言された未初期化の m を使用します。

```
refertst: proc options(main);

   dcl
     1 a based,
       2 n fixed bin(31),
       2 c char(m refer(n));

   call inner_proc;

   inner_proc: proc;

     dcl m fixed bin(31);
     dcl p pointer;
```

```
    m = 15;
    allocate a set(p);
  end;
end;
```

区域データとその属性

区域変数は、基底付き変数を割り振るために予約される主記憶域の区域を定義するものです。 ALLOCATE ステートメントや FREE ステートメントを使用すれば、この予約されたストレージを基底付き変数に割り振ったり、基底付き変数から解放したりすることができます。区域変数のストレージ・クラスはどれでもかまいませんが、区域変数に ALIGNED を指定しなければなりません。

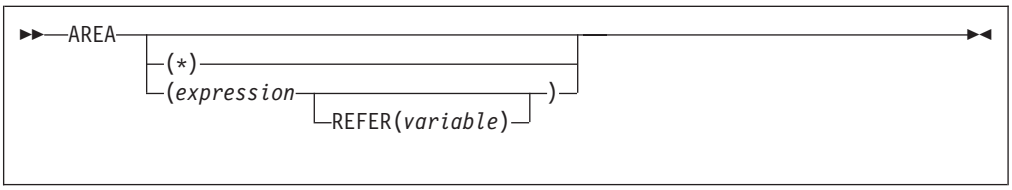
基底付き変数を割り振るときに区域の指定を省略すると、使用可能などこかのストレージが取得されます。したがって、割り振られた複数の基底付き変数が主記憶域全体にわたって分散していることもあります。内部処理の場合は、項目はポインターの使用によって容易にアクセスされるので、分散していても支障はありません。しかし、割り振られた項目をデータ・セットに伝送する場合は、それらを 1 つにまとめる必要があります。区域変数に割り振られた項目はすでに 1 つにまとまっているので、1 つの単位として伝送や割り当てを行うことができ、しかもそれらを個別に処理することもできます。

ある区域変数内での基底付き変数の位置は、その区域変数の開始位置からの変位として識別できると便利です。この目的のためにオフセット変数が用意されています。

区域はその含まれる割り振りで、完全に割り当てまたは伝送することができます。そのため、基底付き割り振りのセットは、割り当てと入出力のための単位として扱うことができ、各割り振りは独自の識別を保持します。

区域のサイズは、ストリングの長さや配列の境界と同じ方法で調節できるので、式やアスタリスク (被制御区域パラメーターの場合)、または REFER オプション (基底付き区域の場合) を使用して指定することができます。

OFFSET 属性または IN オプション内に書いた変数には、コンテキスト上から、AREA 属性が与えられます。変数の AREA 属性を明示的に宣言することもできます。



expression
区域のサイズを指定します。 *expression* またはアスタリスクが指定されていなければ、デフォルトは 1000 になります。

***** 宣言される区域変数がパラメーターの場合は、サイズをアスタリスクで指定することができます。

REFER
REFER オプションの説明については、 253 ページの『REFER オプション (自己定義データ)』を参照してください。

ストレージ・クラスが AUTOMATIC または CONTROLLED である区域のサイズは、予約されるストレージのバイト数を表す値になる式で指定します。

区域が BASED 属性を持っているときは、区域のサイズは定数で指定しなければなりません。ただし、その区域が基底付き構造体または共用体のメンバーで、しかも REFER オプションを使用する場合は例外です。

静的ストレージ・クラスの区域サイズは、制限付き式で指定しなければなりません。

AREA 属性を宣言する例を次に示します。

```
declare area1 area(2000),
        area2 area;
```

宣言されたサイズの区域が予約され、その予約された区域の前にさらに 16 バイトが付けられます。この 16 バイトには、使用中のストレージの大きさなどの制御情報の詳細が入ります。

実際に使用される予約済みのストレージの大きさは、区域のエクステント といいます。区域変数が割り振られた時点では、そこにはなにも入っていないので、区域のエクステントはゼロです。最大エクステントはその区域のサイズです。実行中いつでも、区域内で基底付き変数を割り振ることも解放することもできます。したがって、区域のエクステントは変化します。

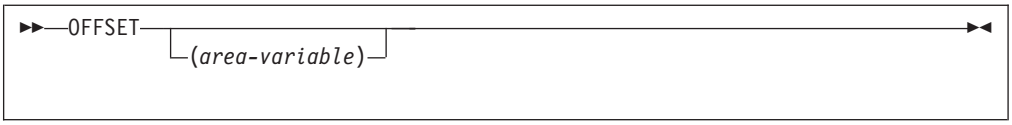
基底付き変数が解放されると、その変数が占有していたストレージは別の割り振りに使用可能になります。ある区域内の使用可能なストレージのチェーンは保持されます。チェーンの先頭は制御情報内に保持されます。基底付き変数に必要なストレージの容量はそれぞれ異なっているので、それらを割り振ったり解放したりすれば、割り振りの容量と使用可能なスペースの容量が一致しないときには必然的にすき間が生じます。これらのすき間は区域のエクステントに含まれます。

どのような演算子 (比較演算子を含む) も、区域変数に使用することはできません。

オフセット・データとその属性

オフセット・データは、必ず区域変数と一緒に使われます。オフセット変数の値は、区域変数内での基底付き変数の位置を、その区域変数の開始位置からの変位として示します。基底付き変数は相対的な位置で示されるので、区域変数が主記憶域の別の部分に割り当てられても、オフセット値は無効になりません。

オフセット変数を使用しても、区域内でポインター変数を使用できなくなるわけではありません。



区域変数とオフセット変数の関連付けは、永続的なものではありません。 POINTER 組み込み関数を使用すれば、オフセット変数を任意の区域変数に関連付けることができます (247 ページの『ロケーターの変換』を参照)。このような関連付けを宣言しておくことの利点は、このオフセット変数を参照すると、関連付けられている区域変数を参照することになるという点です。区域変数の指定を省略すると、このオフセット変数は、POINTER 組み込み関数の使用を介してしかロケーター修飾子として使用することができません。

オフセット変数のセット

オフセット変数の値は、下記のいずれかの方法でセットすることができます。

- ALLOCATE ステートメントによってセットします。
- 別のロケーター変数の値、またはユーザー定義の関数から戻されたロケーター値の割り当てによってセットします。
- 組み込み関数 NULL、SYSNULL、ADDR、ENTRYADDR、OFFSETADD、OFFSETSUBTRACT、OFFSETVALUE、または OFFSET によってセットします。

区域変数の指定を省略したときは、そのオフセット変数は、POINTER 組み込み関数の使用を介してロケーター修飾子として使用することしかできません。

オフセット変数の例

次の例を考えてみてください。

```
dcl X based(O),
    Y based(P),
    A area,
    O offset(A);

    allocate X;
    allocate Y in(A);
```

区域 A とオフセット O のストレージ・クラスは、デフォルトによって AUTOMATIC になります。最初の ALLOCATE ステートメントは、次のステートメントと同等です。

```
allocate x in(A) set(O);
```

2 番目の ALLOCATE ステートメントは、次のステートメントと同等です。

```
allocate Y in(A) set(P);
```

次の例は、オフセット変数を使用して区域変数内にリストを作成するときの方法を示しています。

```
dcl A area,
    (T,H) offset(A),
    1 STR based(H),
      2 P offset(A),
      2 data;

    allocate STR in(A);
    T=H;
```

```
    do loop;
      allocate STR set(T->P);
      T=T->P;
:
:
  end;
```

区域変数のための組み込み関数

EMPTY 組み込み関数は、区域変数を初期設定してこれを空にし、それに対して行われている可能性があるすべての割り振りを解放します。これが区域変数の初期状態で、そこにはまだなにも割り振りがされていません。 AVAILABLEAREA 組み込み関数は、その区域内で行うことができる割り振りの最大サイズを返します。

区域の割り当て

代入ステートメントを使用して、区域参照の値を 1 つ以上の区域変数に割り当てることができます。区域から区域への割り当てを行うと、ターゲット区域内のすべての割り振りが解放され、次に、ソース区域のエクステントがターゲット区域に割り当てられ、ソース区域のすべてのオフセットがターゲット区域で有効になります。

次に例を示します。

```
declare X based (O(1)),
        O(2) offset (A),
        (A,B) area;

        alloc X in (A);
        X = 1;
        alloc X in (A) set (O(2));
        O(2) -> X = 2;
        B = A;
```

POINTER 組み込み関数を使用すると、 POINTER (O(2),B)->X と O(2)->X という参照は、それぞれ区域 B と区域 A で割り振られた同じ値を表します。

割り振りを含んでいない区域がターゲット区域に割り当てられると、ターゲット区域内のすべての割り振りが解放されるだけです。

区域割り当てを使用すれば、元の区域の境界を超えて基底付き変数のリストを拡張することができます。ある区域内に基底付き変数を割り振ろうとしても、その変数を収容するのに必要なフリー・ストレージがその区域内にないか、またはある区域を別の区域に割り当てようとして、十分な大きさがないと、AREA 条件が起こります。この条件の ON ユニットを使用すれば、不適当な区域の参照を修飾しているポインターの値を変更して、別の区域を指すようにすることができます。ON ユニットからの戻りにおいて、割り振りが新規の区域内で再度試されます。あるいは別の方法として、AVAILABLEAREA 組み込み関数を使用して、これから行おうとする割り振りが、AREA 条件を起こさずに対象とする区域内で可能であるかどうかを判別することができます。また、ON ユニットはその区域を書き出して、それを EMPTY にリセットすることもできます。

区域の入出力

区域の機能を使用すれば、基底付き変数のリスト全体を 1 つの単位として、RECORD ファイルとの間で入出力を行うことができます。出力の場合、区域のエクステントと 16 バイトの制御情報が伝送されます (ただし、区域が構造体内または

共用体内にあり、しかもその構造体または共用体の最後の項目でない場合は例外で、そのときは宣言されているサイズ分が伝送されます)。したがって、区域内の未使用部分がデータ・セットのスペースを取ることはありません。

区域のエクステントは変化するので、可変長レコードを使用してください。必要になる最大レコードの長さは、区域の長さ (区域サイズ +16) です。

リスト処理

リスト処理とは、データの集まりを処理するための多くの手法に付けられた名前です。配列、構造体、および共用体もデータの集まりを処理するために使われますが、リスト処理手法は、プログラムの実行中にデータの集まりを無限に再配列したり、広げたりすることができるという点で、融通性があります。ここでは、これらの手法については説明しませんが、基底付き変数とロケーター変数がこの種の処理の基礎としてどのような働きをするかを説明します。

リスト処理では、多数の基底付き変数 (それぞれが 1 つ以上の世代を持っている) を 1 つのリスト内に入れることができます。このリスト内のメンバーは、ほかのメンバーまたはほかのリストの位置を指し示す 1 つ以上のポインターをメンバー内に持つことによって、相互に関連しています。基底付き変数を割り振るときには、主記憶域内のどこにその変数を割り振るかを指定することはできません (ただし、どの区域内に割り振りたいかを指定することはできます)。実際には、チェーニングしている項目は主記憶域全体にわたって分散していることもあります。しかし、それぞれのポインターをアクセスすれば、次のメンバーが見つかります。 リスト内のメンバーは、通常は、ポインター変数を含む構造体または共用体です。次の例では、構造体のリストが作成されます。

```
 dcl 1 STR based(H),
       2 P pointer,
       2 data,
     T  pointer;

       allocate STR;
       T=H;

       do loop;
         allocate STR set(T->P);
           T=T->P;
           T->P=null;
⋮
      end;
```

これらの構造体は、STR のいくつかの世代であり、各世代の中のポインター変数 P によって相互にリンクされます。リストを作成している間、ポインター変数 T は前の世代を識別します。最初の ALLOCATE ステートメントは、世代を示すようポインター H をセットします。つまり、ポインター H はリストの開始点 (リストの先頭) を示します。 2 番目の ALLOCATE ステートメントは、前の世代の中にあるポインター P がこの新しい世代の位置を示すよう、ポインター P をセットします。代入ステートメント T=T->P; はポインター T を更新し、新規の世代の位置を識別できるようにします。代入ステートメント T->P=NULL; は、リストの最後の肯定のディレクティブを指定して、 NULL の最後の世代内にポインターを設定します。

260 ページの図 17 は、一方向のチェーンを図解したものです。

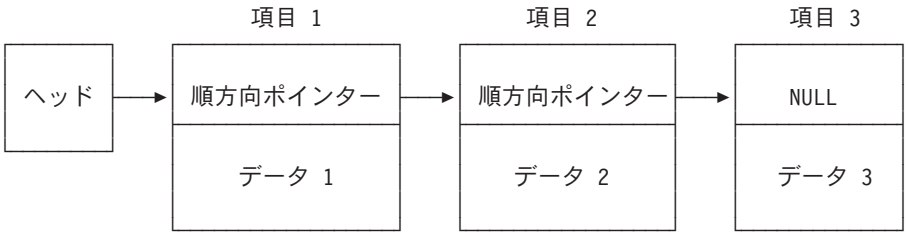


図 17. 一方向のチェーンの例

各世代の中にある P の値が、各世代ごとに別々のポインター変数に割り当てられているのでない限り、STR の各世代は、リストが作成された際の順序でしかアクセスできません。上の例の場合、下記のステートメントを使用すれば、各世代を順にアクセスすることができます。

```
  do T=H
    repeat(T->P)
    while (T¬=null);
⋮
      T->data;
⋮
end;
```

先に挙げたいくつかの例は、一方向のリストを作成する簡単なリスト処理手法です。 別のポインター変数をこの構造体内または共用体内に追加すれば、さらに複雑なリストを作成することができます。別のポインター変数を追加したとすれば、それで前の世代を指すことができます。それにより、リストは双方向となります。リストの任意の項目から直前の項目と次の項目に、該当するポインター値を使用してアクセスすることができます。最新のポインター値を NULL の値にセットする代わりに、リスト内の最初の項目を指すようにセットすれば、リング状リスト (循環リスト) を作成することができます。

リストは、1 つの基底付き変数の世代だけで構成する必要はありません。複数のポインター値を適切にセットすれば、複数の異なる基底付き構造体または共用体の世代を 1 つのリスト内に入れることができます。それらのポインターの値を操作することによって、リストに項目を追加したり、リストから項目を削除したりすることができます。ポインターを操作することによってリストを再構成することができ、その結果リスト内のデータの処理が簡素化されます。

ASSIGNABLE 属性と NONASSIGNABLE 属性

ASSIGNABLE 属性と NONASSIGNABLE 属性は、関連づけられた変数が、割り当てのターゲットとなり得るかどうかを指定します。

省略形: ASGN、NONASGN

デフォルト: ASSIGNABLE

変数が NONASSIGNABLE 属性を持つ場合には、この変数は割り当てることはできません。

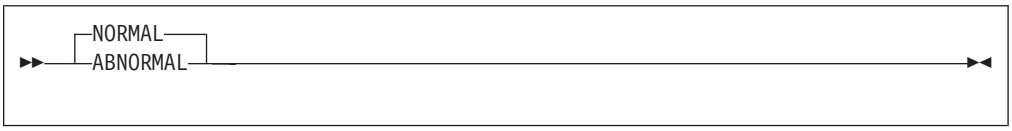
項目記述子に NONASSIGNABLE 属性がある場合、その引数は、対応する ENTRY が呼び出されても変更されないものと見なされます。引数が定数であると、仮引数は作成されません。

ASSIGNABLE 属性および NONASSIGNABLE 属性は、構造体または共用体のメンバーに影響を与えます。

NORMAL 属性と ABNORMAL 属性

NORMAL 属性と ABNORMAL 属性は、関連づけられた変数が、いつでも変更の対象となるかどうかを指定します。

ABNORMAL 属性は、変数の値が、複数のステートメント間または 1 つのステートメント内で変更可能であることを指定します。 ABNORMAL 属性を持つ変数は、必要とされるたびにストレージから取り出され、変更されるたびにストレージに記憶されます。 ABNORMAL 属性を持つ変数の場合、どのような最適化も行うことはできません。



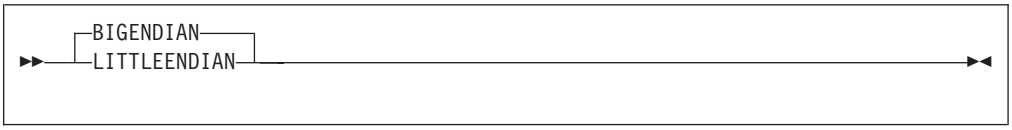
デフォルト: NORMAL

NORMAL 属性および ABNORMAL 属性は、構造体または共用体のメンバーに影響を与えます。

ABNORMAL 属性が、INITIAL 値を持つ INTERNAL STATIC 変数に適用される場合、その変数 (INITIAL 値とともに) は、他の用途に使用されていない場合でも、生成オブジェクト・コードに表示されます。

BIGENDIAN 属性と LITTLEENDIAN 属性

BIGENDIAN 属性と LITTLEENDIAN 属性は、関連した変数を格納する際に、最上位数字または最下位数字のどちらを最初に格納するかを指定します。 BIGENDIAN 属性と LITTLEENDIAN 属性は、FIXED BINARY 変数、ORDINAL 変数、OFFSET 変数、および AREA 変数と、VARYING ストリング変数以外では無視されます。



デフォルト: BIGENDIAN。ただし、z/OS の場合は LITTLEENDIAN がデフォルトです。

BIGENDIAN および LITTLEENDIAN

BIGENDIAN は、変数 (可変ストリングの場合、変数の長さを表す接頭部) が格納されるときに、最上位バイトが最初に格納されることを表します。このフォーマットは、z/OS と RS/6000 に固有のスタイルです。

LITTLEENDIAN は、それとは逆に変数の最下位バイトが最初に格納されることを表します。このフォーマットは、Windows に固有のスタイルです。

AREA 変数に LITTLEENDIAN または BIGENDIAN 属性が設定されると、コンパイラーおよびライブラリーによって管理される制御値を保持するフォーマットだけに影響を与えます。 AREA 変数に格納されているユーザー変数、または AREA 変数のユーザー変数を指すのに使われるオフセット変数には、影響しません。

次の例は、BIGENDIAN 変数と LITTLEENDIAN 変数が保管される方法を示したものです。組み込み関数 HEXIMAGE は、X と Y が、実際に保管される方法を明らかにします。

```
dcl X fixed bin(15) bigendian;
dcl Y fixed bin(15) littleendian;


X = 258;
Y = 258;


display( heximage( addr(X), stg(X) ) );     /* displays 0102 */
display( heximage( addr(Y), stg(Y) ) );     /* displays 0201 */
```

対照的に HEX 組み込み関数は、上の例で示された X と Y に関して、次のような処理をします。

```
display (hex(X));                            /* displays 0102   */
display (hex(Y));                            /* displays 0102   */
```

BIGENDIAN と LITTLEENDIAN は、命令のセマンティクスにはなにも影響を及ぼさず、また変数のストレージ所要量にも影響しません。

BIGENDIAN 属性と LITTLEENDIAN 属性は、構造体または共用体のメンバーに影響を与えます。

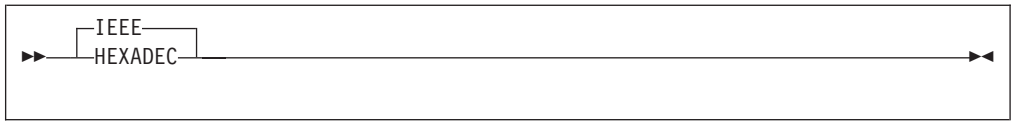
BIGENDIAN と LITTLEENDIAN の使用に関する詳細は、「*プログラミング・ガイド*」を参照してください。

NATIVE 属性と NONNATIVE 属性は、BIGENDIAN と LITTLEENDIAN の同義語ですが、これらの属性が BIGENDIAN と LITTLEENDIAN のどちらを意味するかは、システムによって異なります。

- z/OS と RS/600 では、NATIVE は BIGENDIAN を意味します。
- Windows では、NATIVE は LITTLEENDIAN を意味します。

HEXADEC 属性と IEEE 属性

HEXADEC と IEEE は、関連付けられた変数を、z/OS の場合と同じフォーマットを使用して保管するか、Windows または AIX フォーマットを使用して保管するかを指定します。 HEXADEC 属性と IEEE 属性は、浮動小数点変数の場合以外には無視されます。



デフォルト: IEEE。ただし、z/OS の場合は HEXADEC がデフォルトです。

HEXADEC は、変数が 16 進数 (z/OS) フォーマットで保管されることを表します。

IEEE は、変数が IEEE フォーマットを使用して保管されることを表します。

DEFAULT コンパイラー・オプションの HEXADEC と IEEE サブオプションを使用して、この属性のデフォルトを変更することができます。

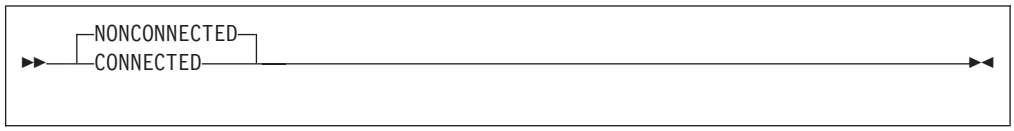
Windows および AIX プラットフォームでは、すべての計算は、IEEE の浮動小数点を使用して行われるので、HEXADEC で宣言された変数は必要に応じて IEEE に変換されます。

z/OS プラットフォームでは、すべての計算は、DEFAULT コンパイラー・オプションの HEXADEC と IEEE サブオプションによって指定されたフォーマットに応じて、浮動小数点命令を使用して行われます。それで、DEFAULT(HEXADEC) オプションでは、すべての計算は 16 進浮動小数点命令を使用して行われるので、IEEE で宣言された変数は HEXADEC に変換されます。一方、DEFAULT(IEEE) オプションでは、すべての計算は、IEEE の 2 進浮動小数点命令を使用して行われるので、HEXADEC で宣言された変数は必要に応じて IEEE に変換されます。

CONNECTED 属性と NONCONNECTED 属性

エレメント、配列、および大構造体または複数の共用体は、常に連結ストレージに割り振られます。非連結ストレージへの参照は、より大きな集合からの不連続項目から構成される集合へ参照を行う場合にのみ行われます (184 ページの『配列のクロスセクション』を参照。) たとえば、次の構造体では、介在配列 A.B および A.C は、両方とも非連結ストレージにあります。

```
1 A(10),
  2 B,
  2 C;
```



省略形: CONN、NONCONN

デフォルト: NONCONNECTED

CONNECTED 属性は、被制御集合パラメーターにしか適用されず、レベル 1 の名前でしか指定することができません。この属性によって、パラメーターの参照が連結ストレージに限られます。そのため、パラメーターをレコード単位の入出力のターゲットまたはソースとして使用するか、もしくはストリング・オーバーレイ定義

の基数として使用することができます。パラメーターが連結であり、CONNECTED 属性が使用される場合、この連結パラメーターへの参照用に、より効率の良いオブジェクト・コードが生成されます。

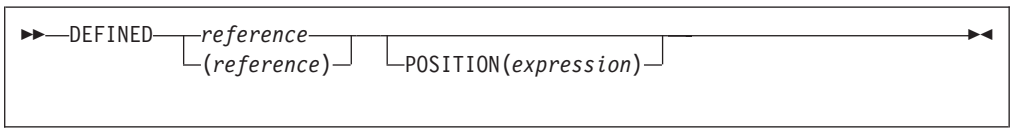
パラメーターが不連続ストレージを占有している場合、NONCONNECTED を指定しなければなりません。次の例では、NONCONNECTED 属性は、sum_Slice ルーチンが、連続していない可能性のあるエレメントを持つ 1 次元の配列を処理することを示しています。最初の呼び出しでは、sum_Slice は最初の行に渡されます。これは連結ストレージにあります。しかし、2 番目の呼び出しでは、sum_Slice は最初の列に渡されます。これは非連結ストレージにあります。

```
dcl A(10,10) fixed bin(31);

display( sum_Slice( A(1,*) ) );     /* first row    */
display( sum_Slice( A(*,1) ) );     /* first column */

sum_Slice:proc(X) returns(fixed bin(31));

  dcl X (*) fixed bin(31) nonconnected; /* default */
return(sum(X) );
end;
```

DEFINED 属性と POSITION 属性

DEFINED 属性は、宣言される変数が、指定された基本変数のストレージの一部または全体に関連付けられることを指定します。

UNION 属性を指定すると、より明快な方法で同じ目的を達することができ、また属性や精度の異なる変数にオーバーレイすることもできます。また、DEFINED 属性では、定義済み変数または基本変数によるアクセスがすべての定義済み変数で反映されますが、共用体では、共用体のメンバーは常に 1 つしか有効ではありません。UNION 属性の構文の詳細については、187 ページの『UNION 属性』を参照してください。



省略形: DEF (DEFINED の場合)、POS (POSITION の場合)

reference
宣言される変数に関連付けられるストレージを持つ変数 (*基本変数*) に対する参照。宣言される変数は、*定義済み変数* です。基本変数は EXTERNAL でも INTERNAL でもよく、パラメーターでも構いません (ストリング・オーバーレイ定義の中で、パラメーターは連結ストレージを参照している必要があります)。ただし、BASED または DEFINED であってはなりません。基本変数の値を変更すると、定義済み変数の値がそれに応じて変更されます。その逆も同様です。

基本変数がデータ集合である場合、定義済み変数はデータ全体で構成することも、データの指定部分だけで構成することもできます。

定義済み変数は、基本変数の属性を継承しません。定義済み変数は INTERNAL にする必要があり、レベル 1 の ID でなければなりません。この変数は次元属性を持つことができます。属性は INITIAL、AUTOMATIC、BASED、CONTROLLED、STATIC、またはパラメーターであってはなりません。

単純、iSUB、およびストリング・オーバーレイの 3 つの定義タイプがあります。

有効になる定義タイプは、次のように決定されます。

1. POSITION 属性が指定されている場合は、ストリング・オーバーレイ定義が有効になります。
2. 基本変数に指定されている添え字に iSUB 変数への参照が含まれている場合、iSUB 定義が有効になります。
3. iSUB 変数も POSITION 属性も存在せず、基本変数と定義済み変数が後述の基準に従って一致している場合は、単純定義が有効になります。
4. どれも当てはまらない場合は、ストリング・オーバーレイ定義が有効になります。

POSITION 属性が指定されている場合は、基本変数に iSUB 参照が含まれていてはなりません。

基本変数と定義済み変数が一致 するのは、引数として渡されたときの基本変数が、定義済み変数の属性 (DEFINED 属性を除く) を持つパラメーターと一致している場合です。このためには、パラメーターの配列の境界、ストリングの長さ、および区域サイズがすべてアスタリスクによって指定されていることが前提になります。

単純定義と iSUB 定義の場合、PICTURE 属性は反復因数を除いて同一の PICTURE 属性にだけ一致します。ストリング・オーバーレイ定義内で参照が有効な基本変数を指定するためには、参照が連結ストレージ内にあることが必要です。マッチング規則は完全にオーバーライドできますが、オーバーライドによってプログラムに意図しない副次作用が出る可能性があります。

配列の境界、ストリングの長さ、または区域サイズに対して指定される値、または派生する値は、基本変数の値と必ずしも一致していなくても構いません。ただし、定義済み変数は、対応する基本配列、ストリング、または区域に収まるようにする必要があります。

定義済みデータを参照する際には、STRINGRANGE、SUBSCRIPTRANGE、および STRINGSIZE の条件が、基本変数ではなく、定義済み変数の配列境界とストリングの長さに生じます。

値の決定と名前の解釈は、次の順序で行われます。

1. 変数を宣言するブロックに入ったときに、定義済み変数の配列境界、ストリングの長さ、および区域サイズが計算されます。
2. 定義済み変数を参照するということは、基本変数の現行世代を参照することです。仮引数が作成されずに、定義済み変数が引数として渡されると、それに対応するパラメーターは、引数が渡された時点での基本変数の現行世代を参照します。呼び込まれたプロシージャー内で基本変数が再割り振りされた場合にも、このことは引き続き当てはまります。

3. 定義済み変数への参照が行われたときに、基本変数と定義済み変数の添え字を評価する順序は定義されていません。

定義済み変数が、位置合わせされていない固定長 BIT であるエレメントが入っている構造体または共用体である場合は、定義済み変数の中のすべての配列境界およびストリング長を、定数として指定する必要があります。

定義済み変数に BIT 属性がある場合、次の条件下では予期しない結果が生じる可能性があります。

- 基本変数がバイト境界上にない
- 定義済み変数が基本変数の最初の位置に定義されておらず、定義済み変数が次の用途に使用されている場合
  - サブルーチン呼び出しのパラメーター (つまり、内部保管データとして参照されている)
  - PUT ステートメントの引数
  - 組み込み関数の引数 (ライブラリー呼び出し)
  - 基本変数が被制御であり、定義済み変数が次元付きで可変の配列境界を指定して宣言されている場合
- 定義済み変数がすべて、位置合わせされていない固定長ビット・ストリングからなる場合、定義済み変数の配列境界、ストリング長、および領域サイズがコンパイル時に分かっていなければなりません。

非連結ストレージ

DEFINED 属性は配列をオーバーレイできます。このため、配列式は非連結ストレージ内の配列エレメント (ストレージ内で隣接していない配列エレメント) を参照できます。次の場合に、連続したエレメントを含む配列式が非連結ストレージを参照できます。

- ストリング配列を、より長いエレメントを持つストリング配列に対して定義する場合。定義済み配列の連続エレメントは、基本配列と定義済み配列のエレメントの長さに差があるために分離され、非連結ストレージに保管されます。

別の配列に対してオーバーレイ定義された配列は、常に非連結ストレージに入れられることが前提になります。

単純定義

単純定義によって、要素変数、配列変数、または構造変数を別の名前で参照できます。

定義済み変数と基本変数は、任意のデータ・タイプで構成できますが、一致している必要があります (前述のとおり)。定義済み変数のエレメントと、対応する基本変数のエレメントごとに、 ALIGNED 属性と UNALIGNED 属性が一致している必要があります。

定義済み変数は次元属性を持つことができます。

配列の単純定義では、次のことが当てはまります。

- 基本変数には、配列のクロスセクションを指定できます。

- 定義済み変数に対して指定する次元数は、基本変数に対して指定する次元数と等しくなければなりません。
- 定義済み配列の境界の対によって指定される範囲は、対応する基本配列の境界の対によって指定される範囲と等しいか、範囲の中に含まれている必要があります。

ストリングの単純定義では、定義済みストリングの長さは基本ストリングの長さ以下であることが必要です。

区域の単純定義では、定義済み区域のサイズは基本区域のサイズと等しくなければなりません。

定義済み変数の対応部分が同じ最大長の可変ストリングであれば、基本変数は可変ストリングであっても、また可変ストリングを含んでいても構いません。

例:

```
DCL A(10,10,10),
    X1(2,2,2) DEF A,
    X2(10,10) DEF A(*,*,5),
    X3 DEF A(L,M,N);
```

X1 は、A のそれぞれの行、列、および平面のうち、最初の 2 つのエレメントからなる 3 次元配列です。X2 は、A の 5 つ目の平面からなる 2 次元配列です。X3 は、添え字式 L、M、および N によって識別されるエレメントからなるエレメントです。

```
DCL B CHAR(10),
    Y CHAR(5) DEF B;
```

Y は、B の最初の 5 文字からなる文字ストリングです。

```
DCL C AREA(500),
    Z AREA(500) DEF C;
```

Z は、C 上で定義される区域です。

```
DCL 1 D UNALIGNED,
         2 E,
         2 F,
              3 G CHAR(10) VAR,
              3 H,
    1 S UNALIGNED DEF D,
         2 T,
         2 U,
              3 V CHAR(10) VAR,
              3 W;
```

S は、D 上で定義される構造体です。単純定義の場合、2 つの構造体の編成は同一でなければなりません。 T への参照は E への参照になり、V は G に、その他も同様です。

iSUB 定義

iSUB 定義を使用すると、基本配列の指定したエレメントからなる定義済み配列を作成できます。定義済み配列と基本配列はスカラー配列でなければならず、任意のデータ・タイプで構成でき、同一の属性を持っている必要があります (次元属性を除く)。

定義済み変数には、次元属性が必要です。定義済み配列の宣言の中で、基本配列には添え字を付ける必要があり、添え字の位置をアスタリスクとして指定することはできません。

iSUB 変数は、基本配列の添え字リスト内での、定義済み配列の次元に対する参照です。基本配列の添え字リストにある少なくとも 1 つの添え字は、 iSUB 式でなければなりません。この式は、評価時に基本配列に必要な添え字を指定します。i の値は、n を定義済み配列の次元数として、1 から n までの範囲です。基本配列の添え字の数は、基本配列の次元数と等しくなければなりません。

定義済み配列への参照が添え字式を指定していない場合、添え字の評価は、参照を含む式または割り当ての評価時に行われます。

i の値は整数として指定されます。 iSUB 式の中で、iSUB 変数は REAL FIXED BINARY(31,0) 変数として扱われます。

定義済み変数への参照にある添え字は、基本変数の添え字リストに対応する iSUB がない場合でも評価されます。

iSUB 定義済み変数は、GET DATA または PUT DATA ステートメントのデータ・リストに入れることはできません。

例:

```
DCL A(10,10) FIXED BIN
    X(10) FIXED BIN DEF( A(1SUB,1SUB) );
```

X は、A の対角線で構成される 1 次元配列です。 X(i) は、A(i,i) と同じストレージを参照します。

```
DCL B(5,10) FIXED BIN
    Y(10,5) FIXED BIN DEF( A(2SUB,1SUB) );
```

Y は、境界を転置した B のエレメントで構成される 2 次元配列です。 Y(i,j) と X(j,i) は同じストレージを参照します。

ストリング・オーバーレイ定義

ストリング・オーバーレイ定義を使用すると、定義済み変数を基本変数のストレージに関連付けることができます。定義済み変数と基本変数は、両方ともストリングまたはピクチャー・データでなければなりません。

定義済み変数と基本変数のどちらも、ALIGNED 属性または VARYING 属性を持つことはできません。

定義済み変数と基本変数はともに、次のいずれかのクラスに属していなければなりません。

- 次のものからなるビット・クラス
  - 固定長のビット変数
  - 固定長のビット変数の集合体
- 次のものからなる文字クラス
  - 固定長の文字変数
  - 文字ピクチャー変数または数字ピクチャー変数

上記 2 つの集合体

- 次のものからなるグラフィック・クラス
  固定長のグラフィック変数
  固定長のグラフィック変数の集合体
- 次のものからなるワイド文字クラス
  固定長のワイド文字変数
  固定長のワイド文字変数の集合体

例:

```
DCL A CHAR(100),
    V(10,10) CHAR(1) DEF A;
```

V は、文字ストリング A のすべてのエレメントからなる 2 次元の配列です。

```
DCL B(10) CHAR(1),
    W CHAR(10) DEF B;
```

W は、配列 B のすべてのエレメントからなる文字ストリングです。

POSITION 属性

POSITION 属性は、ストリング・オーバーレイ定義だけで使用でき、定義済み変数の先頭になる基本変数内のビット、文字、グラフィック、またはワイド文字を指定します。

POSITION 属性の式は、基本変数の先頭からの相対的な位置を指定します。この式で指定する値は、1 から *n* の範囲内の値にすることができます。ただし、*n* は次のように定義されます。

```
n = N(b) - N(d) + 1
```

ここで、N(b) は基本変数のビット、文字、グラフィック、またはワイド文字の数、N(d) は定義済み変数のビット、文字、グラフィック、またはワイド文字の数です。

定義済み項目を参照するたびに、この式が計算され、整数値に変換されます。

POSITION 属性を省略すると、POSITION(1) と見なされます。

定義済み変数がビット・クラス集合である場合、以下の点に注意してください。

- POSITION 属性には整数しか指定できません。
- 基本変数は、添え字付きであってはなりません。

基本変数は、連結ストレージ内のデータを参照している必要があります。

例:

```
DCL C(10,10) BIT(1),
    X BIT(40) DEF C POS(20);
```

X は、C の 20 番目のエレメントから 40 個のエレメントで構成されるビット・ストリングです。

```
DCL E PIC'99V.999',
    Z1(6) CHAR(1) DEF (E),
    Z2 CHAR(3) DEF (E) POS(4),
    Z3(4) CHAR(1) DEF (E) POS(2);
```

Z1 は、10 進数字ピクチャー E のすべてのエレメントからなる文字ストリング配列です。 Z2 は、ピクチャー E のエレメント '999' からなる文字ストリングです。 Z3 は、ピクチャー E のエレメント '9.99' からなる文字ストリング配列です。

```
DCL A(20) CHAR(10),
    B(10) CHAR(5) DEF (A) POSITION(1);
```

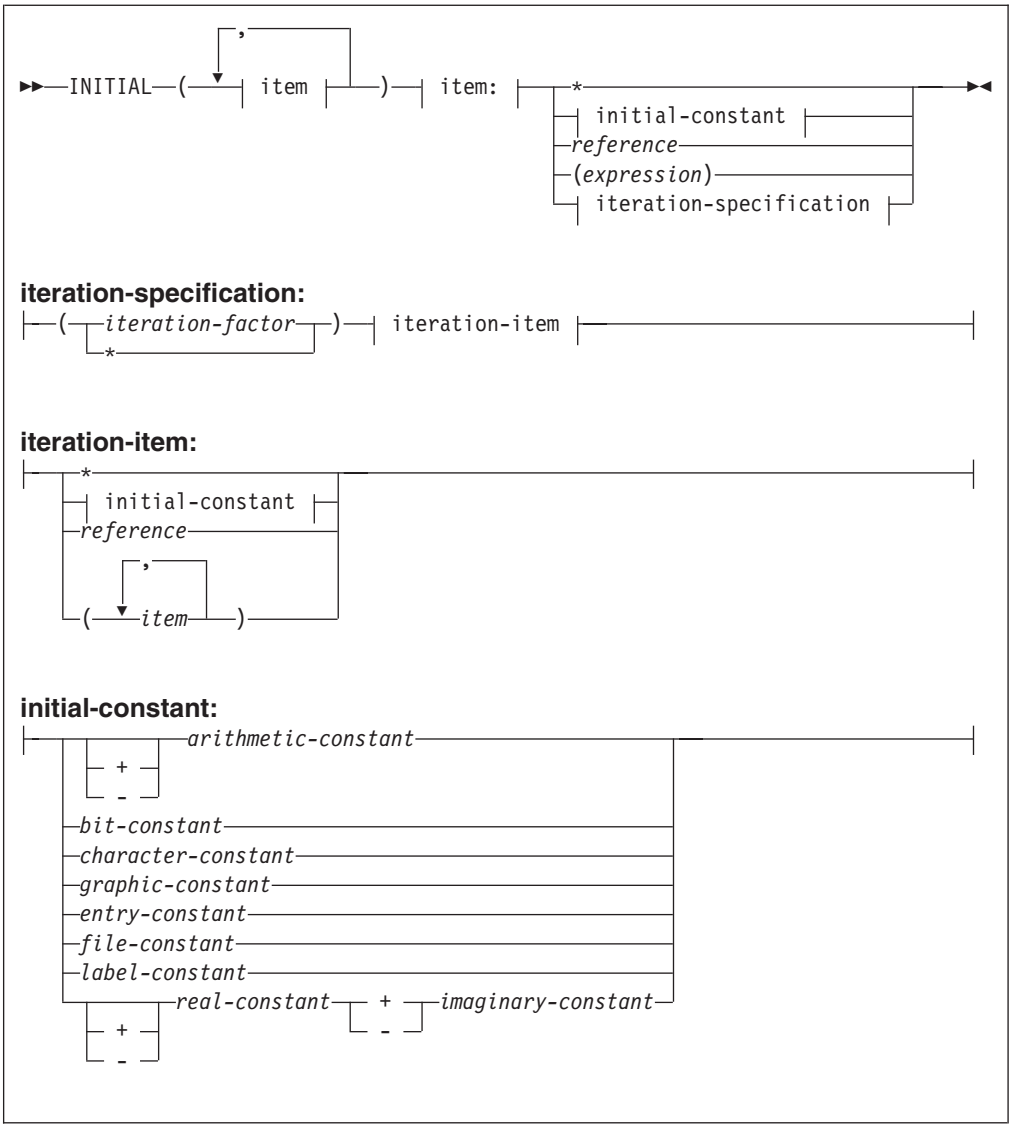
B の最初の 50 文字は、A の最初の 50 文字からなります。POSITION(1) を明示的に指定しなければなりません。そうしない場合は、単純定義が使用され、結果は異なったものになります。

INITIAL 属性

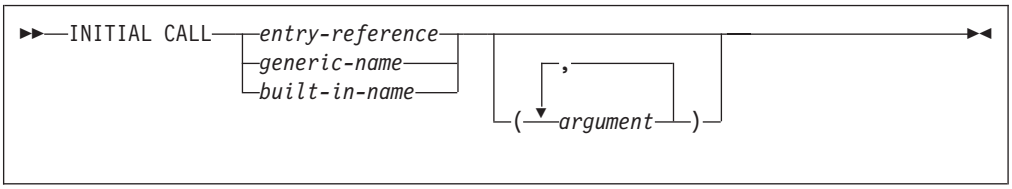
INITIAL 属性は、変数にストレージが割り振られるときにその変数に割り当てられる初期値を指定するものです。要素変数には、初期値を 1 つしか指定することができません。配列変数には複数の初期値を指定することができます。構造変数または共用体変数の場合は、その基本名 (要素変数か配列変数かの別を問わず) を使って個別に初期設定するという方法でのみ、初期設定することができます。定数、定義済みデータ、制御なしパラメーター、および非 LIMITED 静的入り口変数に、INITIAL 属性を与えることはできません。

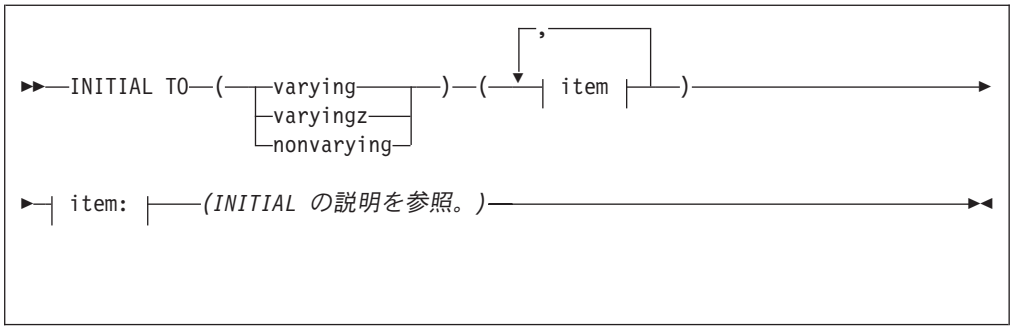
INITIAL 属性には、3 つのフォーマットがあります。

1. 最初のフォーマット INITIAL では、初期定数、式、または関数参照を指定します。その値が、ストレージ割り振り時に変数に割り当てられます。
2. 2 番目のフォーマット INITIAL CALL では、初期設定を行うためにプロシージャーを呼び出すことを、CALL オプションによって指定します。変数の初期設定は、呼び出されたルーチンの実行中に割り当てによって行われます (ルーチンが関数として呼び出されて値を呼び出し点に戻すという方法で初期設定が行われるのではありません)。
3. 3 番目のフォーマット INITIAL TO では、ポインター (またはポインターの配列) が INITIAL LIST に指定された文字ストリングのアドレスで初期化されることを指定します。また、ストリングにも TO キーワードで示された属性があります。


INITIAL
(
,
item
)
item:
*
initial-constant
reference
(expression)
iteration-specification
iteration-specification:
(
iteration-factor
*
)
iteration-item
iteration-item:
*
initial-constant
reference
(
,
item
)
initial-constant:
arithmetic-constant
+
-
bit-constant
character-constant
graphic-constant
entry-constant
file-constant
label-constant
real-constant
+
-
+
-
imaginary-constant

および


INITIAL CALL
entry-reference
generic-name
built-in-name
(
,
argument
)



省略形: INIT、INIT CALL、INIT TO

* エレメントを、反復因数として使用する場合を除き、初期化しないままにしておくことを指定します。

反復因数
配列のエレメントを初期設定するときに、項目を何回反復するかを指定します。

反復因数は、式であってもアスタリスクであってもかまいません。

- 式は、FIXED BINARY(31) に変換されます。静的変数の場合は、定数でなければなりません。
- アスタリスクは、残りのエレメントを指定された値に初期設定しなければならないことを示します。

反復因数に負またはゼロの値を指定すると、初期設定は行われません。

constant
reference
expression
これらは、初期設定される変数に割り当てられる初期値を指定します。

INITIAL CALL
INITIAL CALL では、渡される入り口参照および引数リストは、 95 ページの『ブロックの活動化』に記載されているような、ブロックを活動化するために設定された条件を、満たしていなければなりません。

INITIAL CALL は、静的データの初期設定に使用することはできません。

次の例では、 A のすべてのエレメントは、どのエレメントも INITIAL 属性を必要とすることなく、X'00' に初期設定されています。

```
dcl  1  A automatic,
        2  ...,
        2  ...,
        2  * char(0) initial call plifill( addr(A), '00'X, stg(A) );
```

INITIAL CALL 属性を持つ AUTOMATIC 変数は、他の用途に使用されない場合でも保存されます (プログラムのロジックによってコールを実行する必要が生じた場合に備えて)。

INITIAL CALL ステートメントによって呼び出されたプロシージャーが、 FETCH ステートメントまたは RELEASE ステートメントで指定されており、主記憶域に存在しない場合、INITIAL CALL ステートメントは、そのプロシージャーの動的ロー

ドを開始します。 (動的ロードの詳細については、 109 ページの『外部プロシージャーの動的ロード』を参照してください。)

INITIAL TO
静的固有のポインターにのみ使用してください。ポインター (または、ポインターの配列) が INITIAL LIST で指定された文字ストリングのアドレスで初期化されることを指定します。また、ストリングに TO キーワードで示された属性があることも指定します。

次の例では、pdays は曜日を含む varyingz ストリングのアドレスの文字で初期化されます。

```
dcl pdays(7) static ptr init to(varyingz)
               ('Sunday',
                'Monday',
                'Tuesday',
                'Wednesday',
                'Thursday',
                'Friday',
                'Saturday'  );
```

INITIAL TO で初期化されたポインターによって識別された値を変更しないでください。この値を読み取り専用のストレージに入れることができますが、値を変更しようとすると記憶保護例外になることがあります。前述の例の配列 pdays では、次の例は正しくない割り当てです。

```
dcl x char(30) varz based;

pdays(1)->x = 'Sonntag';
```

配列変数の初期設定

配列に指定した初期値は、行を主体とした順序で (最後の添え字が最も頻繁に変化する)、配列の連続している各エレメントに割り当てられます。指定されている初期値が多過ぎる場合は、超過している値は無視されます。指定が十分でない場合は、残りの配列は初期設定されません。

ストリングの配列を初期設定するときは、ストリング反復因数と反復因数の両方を指定することができます。どちらか一方だけを指定したときは、ストリング定数が括弧で囲まれていない限り、ストリング反復因数と見なされます。

反復因数は * で指定できます。この場合は、残りのエレメントがすべて指定された値で初期設定されます。

以下の例は、ストリング反復因数と反復因数の使用法 (および両者の違い) を示しています。

`((2)'A')` は `('AA')` と同等です。

`((2)('A'))` は `('A','A')`と同等です。

`((2)(1)'A')` は `('A','A')`と同等です。

`((*)(1)'A')` は `('A','A'...'A')`と同等です。

区域変数は、割り振られると EMPTY 組み込み関数の値で初期設定されます。区域変数の INITIAL 文節は無視されます。

INITIAL 属性内の項目の属性とデータ項目自体の属性とが異なっている場合は、それらの属性の間に互換性がある限りにおいて、変換が行われます。

INITIAL 属性は、REFER 文節の対象として使用することはできません。

共用体の初期設定

共用体メンバーは、初期値を持つことができます。ただし、共用体が静的である場合、INITIAL 属性を持つことができる共用体のメンバーは 1 つだけです。共用体が静的でない場合、INITIAL 属性は指定された順に適用されます。後続の初期値は、その前の値を重ね書きします。

次の例では、NT1 の宣言は、これが静的ストレージ属性を持つ場合は無効になります。

```
dcl
  1 NT1 union automatic,
    2 Numeric_translate_table1 char(256)
                  init( (256)'00'X),
    2 *,
      3 * char(240),
      3 * char(10) init('0123456789'),
    2 * char(0);

dcl
  1 NT2 union static,
    2 Numeric_translate_table2 char(256),
    2 *,
      3 * char(        rank('0')    )
          init((1)(low(rank('0'))) ),
      3 * char(10) init('0123456789'),
      3 * char(         (256-(rank('0'))-10)    )
          init((1)(low( (256-(rank('0'))-10) )) ),
```

NT2 の宣言は、これが静的ストレージ・クラスを持っていても有効です。さらに、NT2 宣言は、EBCDIC の実行モードと ASCII の実行モードの間で移植性があります。

静的変数の初期設定

プログラムのロード時に割り振られる変数 (つまり、静的変数) の場合、その変数はプログラムの実行中割り振られたままになっているので、INITIAL 属性で指定した値は 1 回だけ割り当てられます。 (フェッチされるプロシージャーの静的ストレージは、そのプロシージャーがロードされるたびに割り振られ、初期設定されます。)

静的変数が INITIAL 属性を使用して初期設定される場合、その初期値は制限付き式として指定されなければなりません。エクステントの指定は、制限付き式でなければなりません。

静的変数を初期設定する際の制約事項は、次のとおりです。

- STATIC ENTRY 変数には LIMITED 属性がなければなりません (129 ページの『LIMITED 属性』を参照してください)。
- INITIAL は、静的フォーマット変数には使用することができません。
- INITIAL は、構造体または共用体の一部ではないラベル変数に使用することができます。この場合、ラベル変数は CONSTANT 属性になります。
- INITIAL は、AREA 変数には無効です。

- 静的共用体のただ 1 つのメンバーだけが、INITIAL を指定できます。
- RESERVED 属性を持たない STATIC EXTERNAL 項目が、複数の宣言で INITIAL 属性を指定された場合、指定された値はどのような場合でも同じでなければなりません。

自動変数の初期設定

自動変数の場合、その変数は、それを宣言しているブロックが活動化されるたびに割り振られるので、指定した初期値は割り振りのたびに割り当てられます。

基底付き変数と被制御変数の初期設定

基底付き変数と被制御変数の場合、その変数は ALLOCATE ステートメント (基底付き変数のときは LOCATE ステートメントも可) の実行時に割り振られるので、指定した初期値は割り振りのたびに割り当てられます。

基底付き変数のストレージが、ALLOCATE や AUTOMATIC 組み込み関数を使用して割り振られると、初期値は割り当てられません。また区域変数の場合では、区域は暗黙にも EMPTY に初期設定されません。

例

次の例では、Name にストレージが割り振られたときに、文字定数 'John Doe' (右側にブランクが埋め込まれて 10 文字になる) が Name に割り当てられます。

```
dcl Name char(10) init('John Doe');
```

次の例では、Pi が割り振られたときに、それに 3.1416 という値が初期設定されます。

```
dcl Pi fixed dec(5,4) init(3.1416);
```

次の例では、式 B*C の値で A を初期設定することを指定しています。

```
declare A init((B*C));
```

次の例では、A の最初の 920 個のエレメントはそれぞれ 0 にセットされ、次の 80 個のエレメントは、5,5,5,9 を 20 回繰り返したものになります。

```
declare A (100,10) initial
        ((920)0, (20) ((3)5,9));
```

次の例では、A の第 1、第 3、および第 4 のエレメントが初期設定されます。配列のその他の部分は初期設定されません。配列 B は全体が初期設定され、最初の 25 個のエレメントは 0 に、次の 25 個は 1 に、残りのエレメントは 0 にそれぞれセットされます。構造体 C では、次元 (8) が D と E に継承されており、D の最初のエレメントだけが初期設定されます。 E は全部のエレメントが初期設定されます。

```
declare A(15) character(13) initial
          ('John Doe',
           *,
           'Richard Row',
           'Mary Smith'),

         B (10,10) decimal fixed(5)
           init((25)0,(25)1,(*)0),
```

```
        1 C(8),
          2 D initial (0),
          2 E initial((*)0);
```

構造体または共用体の配列を宣言するときに、別の構造体または共用体と同じ構造体フォーマットにするための LIKE 属性を指定した場合、その別の構造体または共用体のエレメントがすでに初期設定されていると、この構造体または共用体の配列における最初の構造体または共用体だけが初期設定されます。

次の例では、構造体の配列において J(1).H と J(1).I だけが初期設定されます。

```
declare 1 G,
          2 H initial(0),
          2 I initial(0),
        1 J(8) like G;
```

第 11 章 入出力

データ・セット 278
連続 279
索引付き 279
相対 279
領域内 279
ファイル 279
FILE 属性 279
RECORD 属性と STREAM 属性 283
INPUT 属性、OUTPUT 属性、および UPDATE 属性 283
SEQUENTIAL 属性と DIRECT 属性 284
BUFFERED 属性と UNBUFFERED 属性 . . . 284
ENVIRONMENT 属性 285
KEYED 属性 285
PRINT 属性 285
ファイルのオープンとクローズ 285
OPEN ステートメント 286
暗黙オープン 288
CLOSE ステートメント 290
FLUSH ステートメント 291
SYSPRINT および SYSIN 291

PL/I の入出力ステートメント (READ、WRITE、GET、 PUT など) を使用すれば、計算機の主記憶装置と補助記憶装置との間でデータを伝送することができます。プログラムの外部にあるデータの集まりをデータ・セット といいます。 データ・セットからプログラムへデータを伝送することを入力 といい、プログラムからデータ・セットへデータを伝送することを*出力* といいます。 (端末を使用している場合には、「データ・セット」は端末も意味することがあります。)

PL/I の入出力ステートメントは、データ・セットの論理的な編成に関係するものであり、データ・セットの物理特性とは関係がありません。したがって、プログラムの実行時に使用される入出力装置についての特別な知識がなくても、プログラムを設計することができます。データ・セット内の物理編成ではなく、データの論理的な側面を主眼にソース・プログラムを処理することができるように、 PL/I ではファイル と呼ばれるデータ・セットのモデルを採用しています。同じファイルを、プログラムの実行中の別々の時点で、別々のデータ・セットに関連づけることができます。

PL/I には、ストリームとレコードの 2 通りのデータ伝送方法があります。

ストリーム指向データ伝送では、データ・セット内のデータの編成はプログラムでは無視され、データは、文字フォーマットの個々のデータ値が連続するストリームと見なされます。入力の場合は、データは文字フォーマットから内部フォーマットに変換され、出力の場合は、内部フォーマットから文字フォーマットに変換されます。

ストリーム指向データ伝送の詳細については、 303 ページの『第 13 章 ストリーム指向データ伝送』を参照してください。

ストリーム指向データ伝送は、文字フォーマットで用意された入力データを処理したり、可読フォーマットの出力を作成したりする (編集する必要がある) 場合に使用することができます。プログラムが対話式システムのもとで実行されているときは、ストリーム指向データ伝送を使用すれば、実行時に端末からプログラムと同期通信することができます。

データの書式作成が可能という点で、ストリーム指向データ伝送はレコード単位データ伝送よりも多様性があります。しかし、その結果、実行時間は増加します。

レコード単位データ伝送では、データ・セットは個別レコードの集まりと見なされます。外部メディア上のレコードは、普通は内部ストレージに入っているレコードをそのままコピーしたものです。レコード単位データ伝送時にはデータ変換は行われません。すなわち、入力の場合は、データはデータ・セットに記録されているとおりの形で伝送され、出力の場合は、データは内部で記録されているとおりの形で伝送されます。

レコード単位データ伝送の詳細については、 293 ページの『第 12 章 レコード単位データ伝送』を参照してください。

レコード単位データ伝送は、2 進数、10 進数、または文字などのフォーマットで表記されているデータが入っているファイルを処理する場合に使用することができます。

レコード単位データ伝送は、ストリーム指向データ伝送に比べて、データの処理方法の点と、処理できるデータ・セットのタイプの点で多様性があります。データは、それが主記憶域にあるとおりの形でデータ・セットに記録されるため、どのようなデータ・タイプでも可能です。なにも変換は行われませんが、ユーザーはデータ構造について確実に把握している必要があります。

同じデータ・セットを、ある時点ではストリーム指向データ伝送で処理し、別の時点ではレコード単位データ伝送で処理することもできます。ただし、その場合は、データ・セット内の項目はすべて文字フォーマットでなければなりません。

この章の以降の節では、データ・セットの種類、ファイルを記述するための属性、およびデータを伝送するためのファイルのオープンとクローズの方法について説明します。 PL/I で認識されるデータ・セット編成のタイプの詳細については、「*プログラミング・ガイド*」を参照してください。

データ・セット

データ・セットは、端末から入力したり端末に出力したりする場合に使用されるほかにも、磁気テープ装置、直接アクセス記憶装置 (DASD) などの各種の補助記憶装置メディアに記憶されます。これらの記憶メディアは、その種類に関係なく、データの収集、記憶、伝送を行うための共通の方法を使用できるという特性を持っています。データがデータ・セットにどのように記録され、そのあと、プログラムに伝送されるためにどのように取り出されるかは、データ・セットの編成の仕方によって異なります。データ・セットへのレコードの記憶と、データ・セットからのレコードの取り出しは、連続する物理位置または論理位置に基づいて順次に行われるか、またはデータ伝送ステートメントで指定されたキーを使用して直接行われます。

PL/I は、次のタイプのデータ・セット編成をサポートします。

- 連続
- 索引付き
- 相対
- 領域内

このようなデータ・セット編成は、データの記憶方法とデータのアクセスに使用する手法において異なります。

連続

連続データ・セット編成では、レコードは、それらの連続する物理的な位置のみをもとに編成されています。データ・セットが作成されると、レコードは、それらが示される順に続けて書き込まれます。レコードは、それらが書き込まれた順でしか取り出されません。

索引付き

索引付きデータ・セット編成では、レコードは、各レコードのキーにもとづいて、論理的な順序で書き込まれます。索引付きデータ・セットは、直接アクセス装置に置かれる必要があります。文字ストリング・キーによって、レコードが識別され、レコードの直接検索、置換、追加、および削除を行うことができます。また、順次処理も行うことができます。

相対

相対データ・セット編成では、番号付きレコードが互いに相対的な位置に書き込まれます。レコードは、1 から始まり、続き番号で番号付けされます。相対データ・セットは、直接アクセス装置に置かれる必要があります。レコード番号を指定するキーによって、レコードが識別され、レコードの直接検索、置換、追加、および削除が行われます。また、順次処理も行うことができます。

領域内

領域内データ・セット編成は、番号付けされた複数の領域に分割され、その各領域は、レコードを 1 つ持つことができます。領域は、ゼロから始まり、順番に番号付けされます。領域は、データ伝送ステートメント内でその領域番号 (また、おそらくキー) を指定することによってアクセスすることができます。キーは領域番号を指定し領域を識別することによって、レコードのすべて直接検索、置換、追加、および削除を最適化することができます。

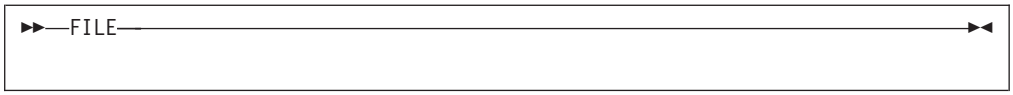
ファイル

データ・セット内の物理編成ではなく、データの論理的な側面を主眼にソース・プログラムを処理することができるように、 PL/I ではファイル と呼ばれるデータ・セットのモデルを採用しています。入出力ステートメントがどのように関連データ・セットにアクセスし、処理するかは、これらのモデルによって異なります。データ・セットとは異なり、ファイルは原始プログラム内でしか意味を持たず、プログラムの外部で物理的に存在するものではありません。

ファイルを表す名前は、FILE 属性を持っています。

FILE 属性

FILE 属性は、対応する名前がファイル定数またはファイル変数であることを指定します。



ファイル定数に、いずれかのファイル記述属性が付いていると、そのファイル定数には FILE 属性が暗黙に付けられます。ある名前を、入出力ステートメントの FILE オプションに書くか、または任意の入出力条件に対する ON ステートメント内に書けば、その名前をファイル定数としてコンテキスト上から宣言することができます。

ファイル定数

PL/I プログラムで処理するデータ・セットは、それぞれファイル定数に関連づけられていなければなりません。

それぞれのファイル定数の個々の特性は、ファイル記述属性によって記述されます。 これらの属性は、代替属性と追加属性の 2 つのカテゴリーに分類されます。

代替 属性とは、属性のグループ内から選択される属性です。あるグループ内の択一属性の 1 つが明示的にも暗黙的にも宣言されておらず、しかもそのグループ内の属性が必須である場合には、デフォルトの属性が使用されます。

表 31 では、PL/I の代替ファイル属性をリストしています。

表 31. 代替ファイル属性

グループ・タイプ	代替属性	デフォルトの属性
使用法	STREAM または RECORD	STREAM
関数	INPUT または OUTPUT または UPDATE	INPUT
アクセス	SEQUENTIAL または DIRECT	SEQUENTIAL
バッファリング	BUFFERED または UNBUFFERED	BUFFERED (SEQUENTIAL ファイルの場合) UNBUFFERED (DIRECT ファイルの場合)
有効範囲	EXTERNAL または INTERNAL	EXTERNAL

追加 属性とは、明示的に宣言しなければならない属性、または明示的に宣言された別の属性によって暗黙に定義される属性です。 追加属性には、ENVIRONMENT、KEYED および PRINT があります。追加属性 KEYED は、DIRECT 属性によって暗黙に定義されます。追加属性 PRINT は、出力ファイル名 SYSPRINT によって暗黙に定義することができます。

表 32 は、それぞれのデータ伝送タイプに適用される属性を示したものです。

表 32. データ伝送のタイプによる属性

伝送のタイプ	属性
ストリーム指向	ENVIRONMENT INPUT および OUTPUT PRINT

表 32. データ伝送のタイプによる属性 (続き)

伝送のタイプ	属性
レコード単位	STREAM BUFFERED および UNBUFFERED DIRECT および SEQUENTIAL ENVIRONMENT INPUT、OUTPUT、および UPDATE KEYED RECORD

表 33 には、ファイル属性の正しい組み合わせが示されています。

表 33. PL/I ファイルを宣言するときの属性

ファイル・タイプ	STREAM	RECORD					凡例:
		SEQUENTIAL			DIRECT		
データ・セットの編成	Consecutive	Consecutive	Relative	Indexed	Relative	Indexed	I 指定または暗黙指定が必要 D デフォルト O オプション S 指定が必要 - 無効
ファイル属性							暗黙指定属性
FILE	I	I	I	I	I	I	
INPUT[1]	D	D	D	D	D	D	FILE
OUTPUT	O	O	O	O	O	O	FILE
ENVIRONMENT	O	O	O	O	O	O	FILE
STREAM	D	-	-	-	-	-	FILE
PRINT[1]	O	-	-	-	-	-	FILE STREAM OUTPUT
RECORD	-	I	I	I	I	I	FILE
UPDATE	-	O	O	O	O	O	FILE RECORD
SEQUENTIAL	-	D	D	D	-	-	FILE RECORD
KEYED[2]	-	-	O	O	I	I	FILE RECORD
DIRECT	-	-	-	-	S	S	FILE RECORD KEYED

注:

[1] INPUT 属性を持つファイルは、PRINT 属性を持つことはできません。

[2] KEYED は、*INDEXED* 出力と *RELATIVE* 出力の場合は必須です。

有効範囲については、 163 ページの『宣言の有効範囲』で説明されています。

ファイル定数に、この章で説明しているいずれかのファイル記述属性が付いていると、そのファイル定数には FILE 属性が暗黙に付けられます。ある名前を、入出力ステートメントの FILE オプションに書くか、または任意の入出力条件に対する ON ステートメント内に書けば、その名前をファイル定数としてコンテキスト上から宣言することができます。

次の例では、名前 Master は、ファイル定数として宣言されています。

```
declare Master file;
```

ファイル変数

ファイル変数は、FILE 属性と VARIABLE 属性を持ちますが、どのファイル定数記述属性も持つことはできません。ファイル変数には、ファイル定数を割り当てることができます。割り当て後、ファイル変数を参照することは、割り当てられたファイル定数を参照することと同じになります。

レコード単位の伝送ステートメントを使用して、データ・セットのファイル変数の値を伝送することができます。伝送後は、そのデータ・セットのファイル変数の値は有効でないことがあります。

VARIABLE 属性は、 53 ページの『VARIABLE 属性』で説明されている環境において暗黙に定義されます。

次の例に示す宣言 Account はファイル変数として宣言され、 Acct1 と Acct2 は、ファイル定数として宣言されています。このあと、これらのファイル定数をこのファイル変数に割り当てることができます。

```
declare Account file variable,
  Acct1 file,
  Acc2 file;
```

構文については、 53 ページの『VARIABLE 属性』を参照してください。

ファイル参照の指定

ファイル参照は、ファイル定数、ファイル変数、または FILE 属性を持つ値を戻す関数参照のいずれかです。ファイル参照は、次のように使用することができます。

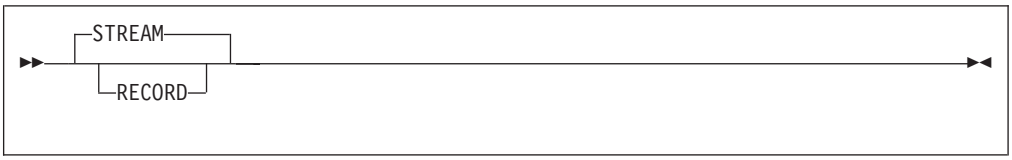
- FILE または COPY オプション内で、使用されます。
- 関数またはサブルーチンに渡される引数として使用されます。
- ON、SIGNAL、および REVERT ステートメントの入出力条件を修飾するために使用されます。
- RETURN ステートメント内の式として、使用されます。

ファイル定数の値を表すファイル変数を利用して、そのファイル定数用の ON ユニットを設定できます (362 ページの『ファイル変数のための ON ユニット』を参照してください)。次の例では、L1 と L2 のラベルが付いている 2 つのステートメントは、同じファイルに対してヌルの ON ユニットを指定しています。

```
dcl F file,
    G file variable;
    G=F;
L1:  on endfile(G);
L2:  on endfile(F);
```

RECORD 属性と STREAM 属性

RECORD 属性と STREAM 属性は、ファイルに使用されるデータ伝送の種類を表します。



デフォルト設定: STREAM

RECORD は、そのファイルが物理的に別個のレコードの集まりから構成されており、各レコードは任意のフォーマットの 1 つまたは複数のデータ項目から構成されていることを示すものです。各レコードは、1 つのエンティティーとして、変数との間で伝送されます。

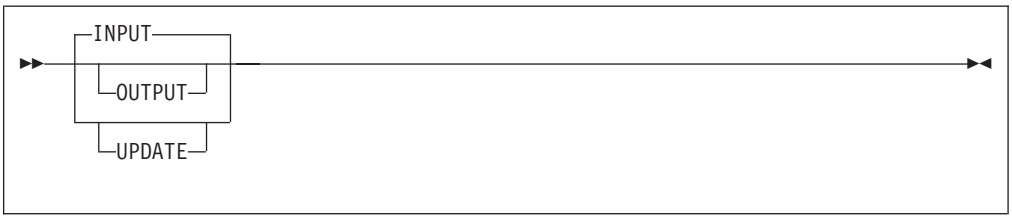
STREAM は、そのファイルのデータが文字フォーマットのデータ項目の連続するストリームであり、そのストリームから各変数へ、またはそれぞれの式からそのストリームへ割り当てられることを示すものです。

STREAM 属性を持つファイルは、OPEN、CLOSE、GET、および PUT 入出力ステートメントの FILE オプションでのみ指定することができます。

RECORD 属性を持つファイルは、OPEN、CLOSE、READ、WRITE、REWRITE、LOCATE、および DELETE 入出力ステートメントの FILE オプションでのみ指定することができます。

INPUT 属性、OUTPUT 属性、および UPDATE 属性

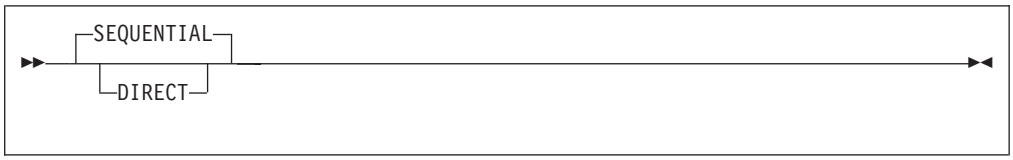
INPUT、OUTPUT および UPDATE の各機能属性は、ファイルに対して行うことができるデータ伝送の方向を表します。 INPUT は、データ・セットからプログラムへデータを伝送することを表します。 OUTPUT は、新しいデータ・セットを作成するために、または既存データ・セットを拡張するために、プログラムからデータ・セットへデータを伝送することを表します。 UPDATE (RECORD ファイルにのみ適用される) は、どちらの方向へもデータを伝送できることを表します。 SEQUENTIAL ファイルの場合、UPDATE 属性の宣言は「その場所で更新」モードを表します。



デフォルト設定: INPUT

SEQUENTIAL 属性と DIRECT 属性

SEQUENTIAL アクセス属性と DIRECT アクセス属性は、RECORD ファイルにのみ適用され、ファイル内のレコードをどのようにアクセスするかを表します。



省略形: SEQUENTIAL の場合は SEQL

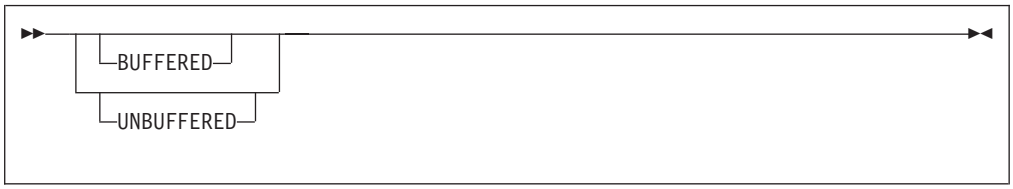
デフォルト設定: SEQUENTIAL

DIRECT 属性は、データ・セットのレコードを直接アクセスできることを表します。データ・セット内のレコードの位置は、文字ストリングであるキーによって突き止められます。したがって、DIRECT 属性を指定すると、KEYED 属性が暗黙に定義されます。関連データ・セットは、直接アクセス記憶装置上になければなりません。

SEQUENTIAL 属性は、連続データ・セットまたは相対データ・セットのレコードを物理的な順序でアクセスすること、および索引付きデータ・セットのレコードをキー・シーケンス順でアクセスすることを表します。ある一定のデータ・セット編成では、SEQUENTIAL 属性を持つファイルを直接アクセスのために使用したり、ランダム・アクセスと順次アクセスの混合のために使用したりすることができます。この場合、そのファイルは追加属性 KEYED を持っていなければなりません。SEQUENTIAL UPDATE ファイルの場合は、データ・セットの既存のレコードを変更することも、無視することも、(データ・セットが INDEXED のときに) 削除することもできます。

BUFFERED 属性と UNBUFFERED 属性

バッファリング (緩衝技法) 属性は、RECORD ファイルにのみ適用されます。



省略形: BUFFERED の場合は BUF、また UNBUFFERED の場合は UNBUF

デフォルト設定: BUFFERED は SEQUENTIAL ファイルのデフォルトです。UNBUFFERED は、DIRECT ファイルのデフォルトです。

BUFFERED 属性は、RECORD ファイルの各レコードが、データ・セットへ伝送されるとき、またはデータ・セットから伝送されるときに、中間記憶バッファーを通らなければならないことを表します。これによって、移動モード処理と位置指定モード処理の両方を行うことができます。

UNBUFFERED 属性は、データ・セットのレコードがバッファーを通る必要がなく、変数に関連づけられている主記憶域へ、またはその主記憶域から、直接に伝送できることを表します。このため、移動モード処理しか行うことができません。

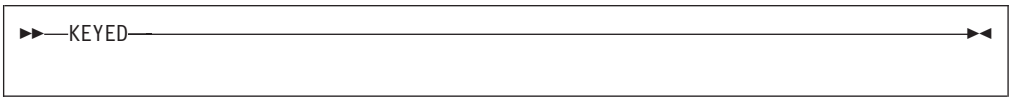
ENVIRONMENT 属性

ENVIRONMENT 属性の特性リストは、PL/I 言語の一部でない各種のデータ・セット特性を指定します。特性と使用法の完全なリストと説明については、「プログラミング・ガイド」を参照してください。

注: 特性は PL/I 言語の一部ではないため、それらの特性をファイル宣言で使用すると、アプリケーション・プログラムの移植性が制限されることがあります。

KEYED 属性

KEYED 属性は、RECORD ファイルにのみ適用され、索引付きデータ・セットおよび相対データ・セットに関連づけられなければなりません。この属性は、ファイル内のレコードを、レコード入出力ステートメントのキー・オプションのいずれか 1 つ (KEY、KEYTO、または KEYFROM) によってアクセスできることを表します。



どのキー・オプションも使用しない場合は、KEYED 属性を指定する必要はありません。

PRINT 属性

PRINT 属性については、324 ページの『PRINT 属性』を参照してください。

ファイルのオープンとクローズ

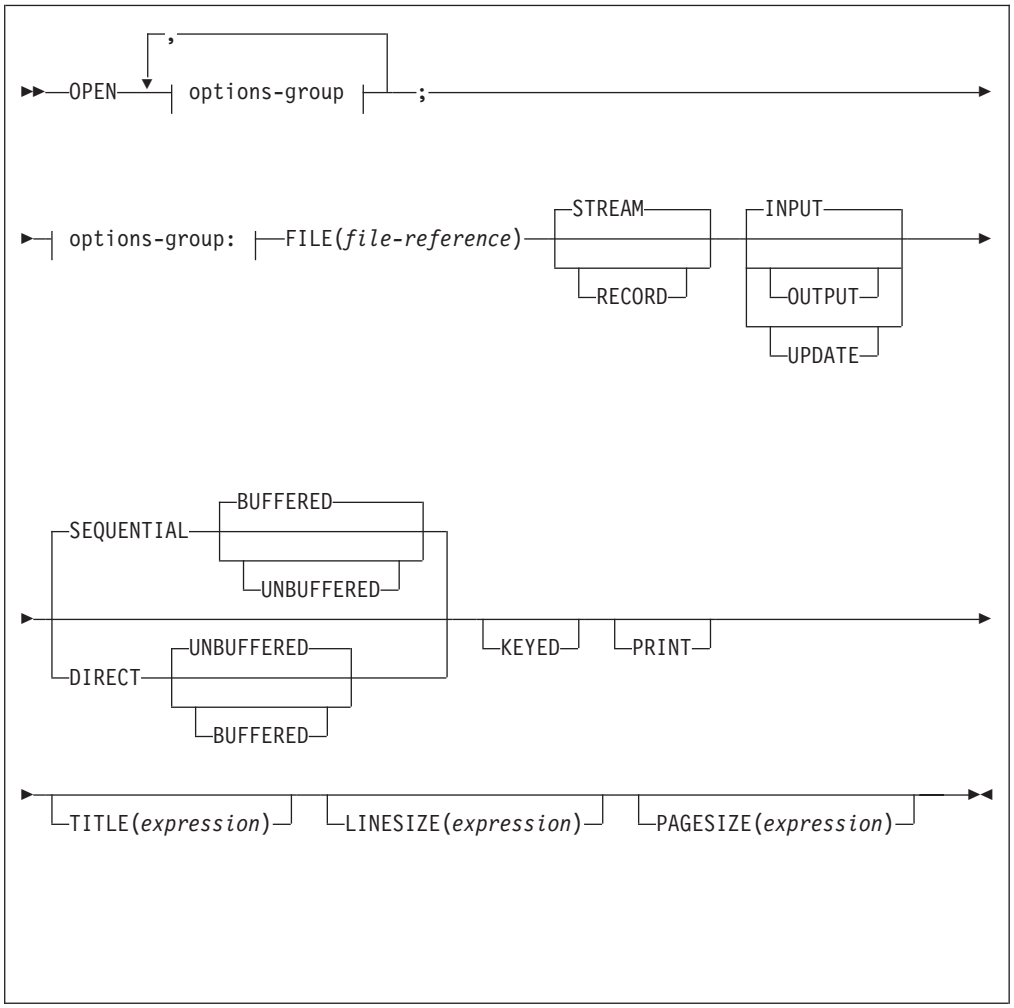
ファイルがデータ・セットに関連づけられていないうちは、そのファイルを入出力ステートメントによるデータ伝送に使用することはできません。ファイルをオープンするということは、ファイルをデータ・セットに関連づけることであり、外部記憶メディアが使用可能かどうかを調べ、そのメディアの位置を決め、必要なオペレーティング・システム・サポートを割り振るということです。処理の完了時には、ファイルはクローズされなければなりません。ファイルをクローズするということは、ファイルとデータ・セットの関係を解くことです。

PL/I には、これらの機能を行うために、 OPEN と CLOSE という 2 つのステートメントがあります。ただし、これらのステートメントを使用するかどうかはオプションです。 OPEN ステートメントが実行されなかったファイルは、そのファイルを参照する最初のデータ伝送ステートメントが実行されている間に暗黙にオープンされます。この場合、DECLARE ステートメントで指定されたファイル情報 (および伝送ステートメントから派生するデフォルトの値) を使用してファイルがオープンされます。同様に、ファイルが PL/I の終了前にクローズされない場合、 PL/I は終了処理中にそのファイルをクローズします。

ストリーム入力、順次入力、または順次更新のファイルがオープンされたときは、関連データ・セットは最初のレコードに位置付けられます。

OPEN ステートメント

OPEN ステートメントは、ファイルをデータ・セットに関連づけます。これによって、OPEN ステートメントで指定された属性と DECLARE ステートメントで指定された属性が組み合わされます。また、オープンされるファイルに必要な属性の一部がまだ宣言されていない場合は、このステートメントでファイルの属性の指定を完成させることができます。



OPEN ステートメントのオプションの指定順序は任意です。

FILE
データ・セットに関連付けられるファイルの名前を指定します。

STREAM、RECORD、
INPUT、OUTPUT、UPDATE、
DIRECT、 SEQUENTIAL、
BUFFERED、UNBUFFERED、
KEYED、および PRINT
これらのオプションは、ファイル宣言で指定した属性を補足する属性を表しま

す。同一ファイルを参照する OPEN ステートメントと DECLARE ステートメントの両方で、同じ属性を指定する必要はありません。レコード入出力とストリーム入出力の属性のリストは、280 ページの表 32 を参照してください。

STREAM ファイルがオープンされると、最初の GET または PUT ステートメントは、ステートメント・オプションまたはフォーマット項目により、アクセスする最初のレコードを指定できます。ステートメント・オプションまたはフォーマット項目は、*n* 行をスキップしたあとにレコードにアクセスすることを指定します。ついで、ファイルは *n* 番目のレコードの先頭に位置づけられます。ステートメント・オプションまたはフォーマット項目がない場合、初期ファイルは、最初の行もしくは最初のレコードの先頭に位置づけられます。そのファイルが PRINT 属性を持っているときは、最初の行または最初のレコードの 1 桁目に物理的に位置づけられます。

すでにオープンされているファイルをオープンしても、そのファイルに影響はありません。

TITLE

expression (式) の内容によって、なにが指定されているかがわかります。TITLE 属性の詳細については、「*プログラミング・ガイド*」を参照してください。

LINESIZE

このファイルに対する後続の処理での行の長さ (バイト数) を表す値で、整数値に変換されます。印刷フォーマット項目や制御フォーマット項目を使用することによって、あるいは GET または PUT ステートメントのオプションによって、新しい行を開始することができます。新しい行を開始するための明示的な処置が取られる前に、行の終わりを超えた位置にファイルを位置付けようとすると、新しい行が開始され、この新しい行の始まりにファイルが位置付けられます。PRINT ファイルのデフォルトの行サイズは 120 です。

LINESIZE オプションは、STREAM OUTPUT ファイルの場合にのみ指定することができます。 GET ステートメントで SKIP オプションを指定したときに使用される行サイズは、データ・セットを作成する際に使用された行サイズです。そのような行サイズがない場合は、論理レコードの現在の長さから制御バイトの長さを引いたものが行サイズになります。

PAGESIZE

計算後、整数値に変換される、1 ページあたりの行数を表す値です。この限界値を超えるような操作が試みられた最初の時点で ENDPAGE 条件が起こります。そのあと、PRINT ファイルへの伝送時に、PAGE フォーマット項目を使用することによって、あるいは PUT ステートメントの PAGE オプションによって、新しいページを開始することができます。デフォルトのページ・サイズは 60 です。

PAGESIZE オプションは、PRINT 属性を持っているファイルにのみ指定することができます。

暗黙オープン

OPEN ステートメントがまだ実行されていないファイルを参照する GET、PUT、READ、WRITE、LOCATE、REWRITE、または DELETE ステートメントが実行されると、そのファイルは暗黙にオープンされます。

COPY オプション付きの GET ステートメントが実行されると、COPY オプションで指定したファイル、または出力ファイル SYSPRINT (COPY オプションでファイルの指定を省略したとき) が暗黙にオープンされることがあります。 COPY オプションで指定したファイルが暗黙にオープンされると、STREAM 属性と OUTPUT 属性が暗黙に定義されます。

表 34 は、左欄のステートメントによってファイルが暗黙にオープンされたとき、暗黙に定義される属性を示したものです。

表 34. 暗黙オープンによって暗黙に定義される属性

ステートメント	暗黙に定義される属性
GET	STREAM、INPUT
PUT	STREAM、OUTPUT
READ	RECORD、INPUT[注]
WRITE	RECORD、OUTPUT[注]
LOCATE	RECORD、OUTPUT、SEQUENTIAL
REWRITE	RECORD、UPDATE
DELETE	RECORD、UPDATE

注:

UPDATE が明示的に宣言されていない場合に限り、INPUT および OUTPUT が READ および WRITE ステートメントのデフォルト属性になります。

表 34 のいずれかのステートメントによってファイルを暗黙にオープンするということは、上記と同じ属性が指定されているファイルに明示的な OPEN ステートメントを使用することと同じ効果があります。

ファイル宣言内に指定した属性と、ファイルをオープンした結果として暗黙に定義される属性との間に、矛盾があってはなりません。たとえば、INPUT 属性と UPDATE 属性は矛盾しており、UPDATE 属性と STREAM 属性も矛盾しています。

表 34 にリストされたデフォルトの属性が適用される前に、前述の暗黙の属性が適用されます。暗黙に定義される属性が矛盾を引き起こすこともあります。デフォルトの属性が適用されたあとで属性が矛盾していると、UNDEFINEDFILE 条件が起こります。

表 35. 組み合わせ後の属性と暗黙に定義される属性

組み合わせ後の属性	暗黙に定義される属性
UPDATE	RECORD
SEQUENTIAL	RECORD
DIRECT	RECORD、KEYED
PRINT	OUTPUT、STREAM
KEYED	RECORD

次の 2 つの例で、ファイル定数とファイル変数を使用して明示的にオープンしたときに組み合わせられる属性を示します。

ファイル定数の例

```
declare Listing file stream;
open file(Listing) print;
```

OPEN ステートメントが実行されると、属性が組み合わされて、STREAM と PRINT という属性が付けられます。暗黙の属性が適用されて、属性は STREAM、PRINT、および OUTPUT となります。デフォルトが適用されたあとの属性は、STREAM、PRINT、OUTPUT、および EXTERNAL となります。

ファイル変数の例

```
declare Account file variable,
        (Acct1,Acct2)  file
        output;

Account = Acct1;
open file(Account) print;

Account = Acct2;
open file(Account) record unbuf;
```

ファイル Acct1 がオープンされて、その属性 (明示的に宣言された属性と暗黙に定義された属性) は STREAM、 EXTERNAL、PRINT、および OUTPUT になります。ファイル Acct2 が、RECORD、EXTERNAL、および OUTPUT の各属性を付けてオープンされます。

暗黙オープンの例

```
declare Master file keyed internal;

read file (Master)
     into (Master_Record)
     keyto(Master_Key);
```

組み合わせ (READ ステートメントの実行による暗黙オープンのため) 後の属性は、 KEYED、INTERNAL、RECORD、INPUT です (そのほかに暗黙に定義される属性はありません)。デフォルトが適用されたあとの属性は、KEYED、INTERNAL、RECORD、INPUT、SEQUENTIAL です。

ファイル定数の宣言の例

```
declare File3 input direct environment( regional(1) )
```

この宣言は 3 つのファイル属性、INPUT、DIRECT、および ENVIRONMENT を指定します。その他の暗黙に定義される属性は、FILE (3 つのうちどの属性によっても暗黙に定義される)、および RECORD と KEYED (DIRECT によって暗黙に定義される) です。有効範囲は、デフォルトによって EXTERNAL となります。ENVIRONMENT 属性は、データ・セットが REGIONAL(1) 編成であることを表します。

先の宣言では、必要なすべての属性が DECLARE ステートメントで指定されているか、または暗黙に定義されています。これらの属性を、OPEN ステートメントで変更 (または無効に) することはできません。

次の例のように宣言しておけば、invntry を別々の目的でオープンすることができます。

```
declare invntry file;
```

次の例では、ファイル属性は、先の例の DECLARE ステートメントで指定された (あるいは暗黙に定義される) 属性と同じになります。

```
open file (Invntry)
     update sequential;
```

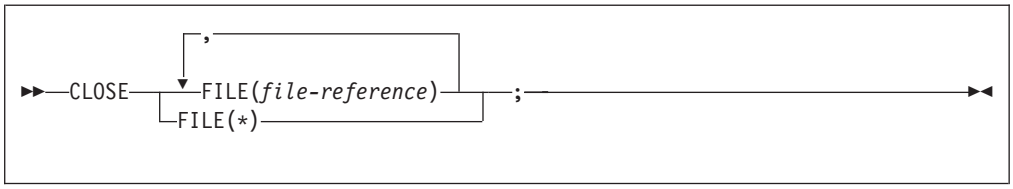
このようにしてオープンされたファイルがクローズされたあとであれば、別の一組の属性を指定して同じファイルをオープンすることができます。たとえば、次の OPEN ステートメントを使用すれば、KEYTO または KEY オプションを使用してレコードを読み取ることができます。

```
open file (Invntry)
     input sequential keyed;
```

ファイルは SEQUENTIAL であるので、このデータ・セットを順次にアクセスすることも、さらに、READ ステートメントで KEY オプションを指定すれば直接にアクセスすることもできます。この種のファイルの場合、READ ステートメントで KEY オプションを指定すれば、指定したレコードが読み取られます。後続の READ ステートメントで KEY オプションを省略すると、KEY 順の次のレコードから順次に読み取られます。

CLOSE ステートメント

CLOSE ステートメントは、そのデータ・セットからオープンされたファイルを分離します。



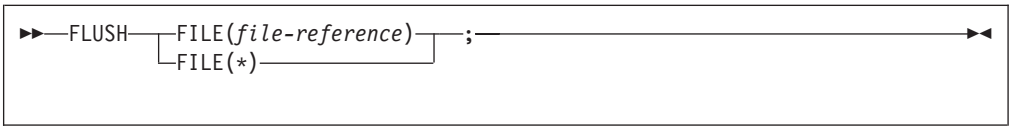
FILE
データ・セットから分離されるファイルの名前を指定します。 CLOSE FILE(*) と指定すると、オープンしているファイルがすべてクローズされます。

CLOSE ステートメントは、明示的にまたは暗黙にオープンされたときにセットされたすべてのファイル属性を、ファイルから切り離すこともします。必要ならば、後続の OPEN ステートメントで、この同じファイルに対して新しい属性を指定することができます。ただし、DECLARE ステートメントでこのファイル定数に明示的に与えた属性は、そのまま効力を持ち続けます。

すでにクローズされたファイルをクローズしても、なんの影響もありません。クローズされたファイルは再びオープンすることができます。 CLOSE ステートメントを使用してファイルをクローズしなくても、ファイルは、プログラムの終了時にクローズされます。

FLUSH ステートメント

FLUSH ステートメントは、1 つまたはすべてのファイルをフラッシュするのに使用します。



FILE
出力ファイルの名前を指定します。

FLUSH ステートメントはオープンしている 1 つの出力ファイル (* を指定した場合はオープンしているすべての出力ファイル) に関連したバッファーをフラッシュします。このような処理は通常はファイルのクローズ時やプログラムの終了時に起きますが、FLUSH ステートメントを指定すると、確実にバッファーがフラッシュされてから他の処理に進むようにすることができます。

SYSPRINT および SYSIN

すべての PL/I プログラム用に使用できるファイルが 2 つ提供されています。 1 つは入力ファイル SYSIN、もう 1 つは出力ファイル SYSPRINT です。これらのファイルは、明示的に宣言したりオープンしたりする必要はありません。 SYSIN のデフォルト属性は STREAM INPUT で、SYSPRINT のデフォルト属性は STREAM OUTPUT PRINT です。SYSPRINT は 7 文字を超えていますが、SYSIN と SYSPRINT のどちらのファイル名もデフォルト属性 EXTERNAL を持っています。

コンパイラーは、上に説明した特殊用途のために SYSIN および SYSPRINT という名前を予約していません。これらの名前は、SYSIN ファイルと SYSPRINT ファイルを識別すること以外の用途にも使用できます。SYSIN と SYSPRINT には他の属性を適用できますが、 SYSPRINT が STREAM OUTPUT ファイルとして宣言またはオープンされる場合は、INTERNAL 属性が宣言されていなければデフォルトで PRINT 属性が適用されます。

第 12 章 レコード単位データ伝送

データ伝送 293
位置合わせされていないビット・ストリング . . 293
可変長ストリング 294
区域変数 294
データ伝送ステートメント 294
READ ステートメント 295
WRITE ステートメント 295
REWRITE ステートメント 295
LOCATE ステートメント 296
DELETE ステートメント 296
データ伝送ステートメントのオプション 297
FILE オプション 297
FROM オプション. 297
IGNORE オプション 298
INTO オプション 298
KEY オプション 299
KEYFROM オプション 299
KEYTO オプション 300
SET オプション 300
処理モード 301
移動モード 301
位置指定モード. 301

この章では、レコード単位データ伝送に使用される入出力ステートメントの機能を説明します。レコード単位データ伝送とストリーム指向データ伝送の両方に同じように適用される PL/I の機能 (ファイルの宣言、ファイル属性、およびファイルのオープンとクローズなど) については、『第 11 章 入出力』で説明しています。ENVIRONMENT 属性の構文については、285 ページの『ENVIRONMENT 属性』を参照してください。それぞれのデータ・セット編成ごとの環境特性とレコード入出力データ伝送ステートメントの詳細については、「*プログラミング・ガイド*」を参照してください。

レコード単位データ伝送では、データ・セット内のデータは、オペレーティング・システムで受け入れ可能な任意のフォーマットで記録されているレコードの集まりです。レコード単位データ伝送時には、データ変換は行われません。つまり、入力の場合は、READ ステートメントが実行されると、1 つのレコードがデータ・セット内に記録されているとおりの形でプログラム変数に伝送されるか、あるいはレコードを指すポインターがセットされます。出力の場合は、ステートメント WRITE、REWRITE、または LOCATE が実行されると、1 つのレコードが内部的に記録されているとおりの形でプログラム変数から伝送されます。ファイルに伝送された情報の長さが N (設定されたレコード長 M より小さい) である場合、最新の M から N を引いたバイト数の結果の値は未定義です。

データ伝送

パラメーターや DEFINED 変数も含むほとんどの変数を、レコード単位データ伝送ステートメントで伝送することができます。一般に、この章で説明することがらは、すべての変数にあてはまります。

注: データ集合は、連結ストレージ内になければなりません。 入力または出力に漢字ストリングを指定する場合は、そのファイルに SCALARVARYING オプションを指定しなければなりません。 データについてのその他の考慮事項は、以降の節で説明しています。

位置合わせされていないビット・ストリング

次のものを伝送することはできません。

- 位置合わせされていない固定長ビット・ストリングである BASED 変数、DEFINED 変数、パラメーター変数、添え字付き変数、または構造体基本エレメント変数。
- 位置合わせされていない固定長ビット・ストリングを最初または最後の基本エレメントとして持つ小構造体 (ただし、その基本エレメントが、その小構造体を含んでいる大構造体の最初または最後のエレメントである場合を除きます)。
- DEFINED 属性を持つ大構造体か、またはパラメーターである大構造体のうち、位置合わせされていない固定長ビット・ストリングを最初または最後のエレメントとして持つもの。

可変長ストリング

可変長ストリングを指定した位置指定モードの出力ステートメント (296 ページの『LOCATE ステートメント』を参照してください) は、そのストリングの最大長に等しい長さを持つフィールドを伝送します。 VARYINGZ ストリングでは、ヌル終了文字も伝送されます。 VARYING ストリングでは、ストリングの現行の長さを示す 2 バイトの接頭部も伝送されます。このため、ENVIRONMENT 属性の SCALARVARYING オプションをファイルに指定する必要があります。

可変長ストリング変数を指定してある移動モードの出力ステートメント (295 ページの『WRITE ステートメント』および 295 ページの『REWRITE ステートメント』を参照してください) は、ストリングの現行の長さだけ伝送します。 VARYINGZ ストリングでは、ヌル終了文字も伝送されます。 VARYING ストリングでは、ENVIRONMENT 属性の SCALARVARYING オプションがファイルに指定されている場合のみ、2 バイトの接頭部が一緒に伝送されます。

読み取りと書き込みに可変長ストリングを使用すると、未定義または未知の長さを持つことができるレコードにアクセス可能になります。

区域変数

位置指定モードの出力ステートメントで区域変数を指定すると、宣言された区域サイズに 16 バイト (制御情報が入っている接頭部の長さ) を加えた長さのフィールドが伝送されます。

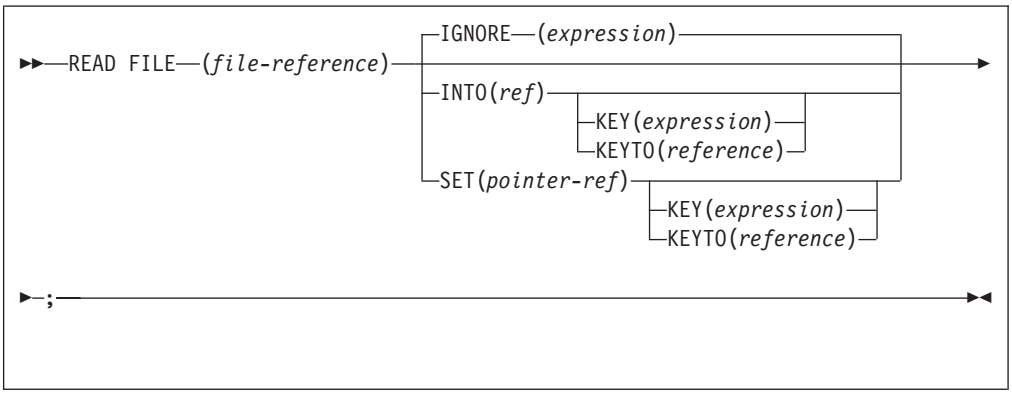
移動モードの出力ステートメントでエレメント区域変数を指定するか、または最後のエレメントが区域変数である構造体を指定すると、区域の現在のエクステントと 16 バイトの接頭部だけが伝送されます。

データ伝送ステートメント

データ・セットとの間でレコードを伝送するためのデータ伝送ステートメントは、READ、WRITE、LOCATE、および REWRITE です。 DELETE ステートメントは、UPDATE ファイルからレコードを削除します。どのデータ伝送ステートメントを使用できるかは、ファイルの属性によって決まります。ステートメント・オプションについては、297 ページの『データ伝送ステートメントのオプション』を参照してください。データ伝送ステートメントの変数については、「*プログラミング・ガイド*」を参照してください。

READ ステートメント

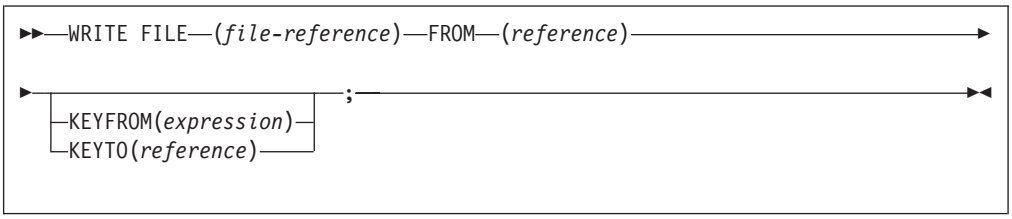
READ ステートメントは、任意の INPUT または UPDATE ファイルに対して使用することができます。これによって、レコードがデータ・セットからプログラム変数へ伝送されるか、またはストレージ内のレコードを指すポインターがセットされます。



キーワードの指定順序は任意です。 INTO、SET、IGNORE のどのオプションも付いていなければ、READ ステートメントは、READ IGNORE(1) と同等です。

WRITE ステートメント

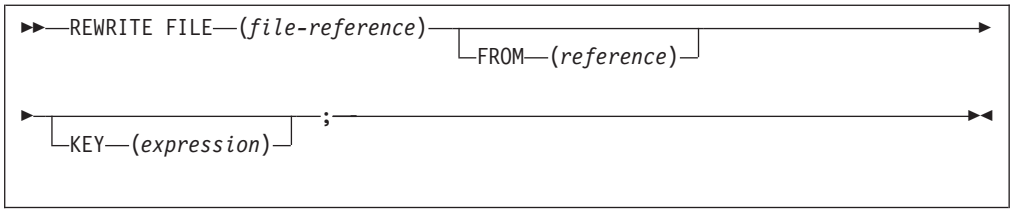
WRITE ステートメントは、任意の OUTPUT ファイル、DIRECT UPDATE ファイル、または SEQUENTIAL UPDATE ファイルに対して使用することができます。このステートメントは、プログラムからレコードを伝送し、それをデータ・セットに追加します。



キーワードの指定順序は任意です。

REWRITE ステートメント

REWRITE ステートメントは、UPDATE ファイルのレコードを置き換えます。SEQUENTIAL UPDATE ファイルの場合は、REWRITE ステートメントは、ファイルから読み込まれた最後のレコードが再書き込みされることを指定します。必然的に、再書き込みまでにレコードが読み込まれている必要があります。 DIRECT UPDATE ファイルの場合は、どのレコードも、まず読み取られたかどうかに関係なく、再書き込みができます。



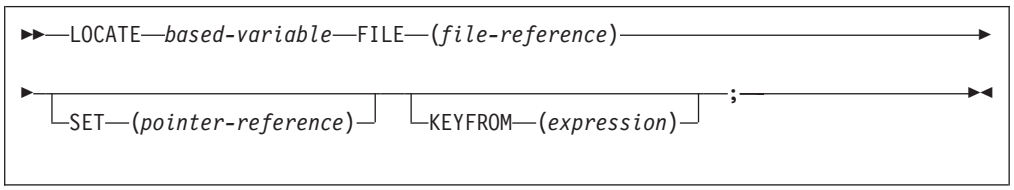
キーワードの指定順序は任意です。 DIRECT 属性を持つ UPDATE ファイルや、SEQUENTIAL 属性と UNBUFFERED 属性の両方を持つ UPDATE ファイルの場合、FROM オプションを指定しなければなりません。

FROM オプションの指定のない REWRITE ステートメントを実行すると、次のような結果になります。

- 最後に読み取られたレコードが、INTO オプションが指定された READ ステートメントで読み取られたものである場合は、FROM オプションが指定されていない REWRITE ステートメントを実行しても、データ・セット内のそのレコードは変わりません。
- 最新のレコードが、SET オプションが指定された READ ステートメントで読み取られた場合、そのレコードは、SET オプション内のポインター変数によって識別される変数でどのような割り当てが行われても更新されます。

LOCATE ステートメント

LOCATE ステートメントは、OUTPUT SEQUENTIAL BUFFERED ファイルを位置指定モードで処理する場合にのみ使用することができます。このステートメントは、出力バッファー内のストレージを基底付き変数に割り振り、次のレコードの位置を指すポインターをセットします。位置指定モードの処理の詳細については、301 ページの『位置指定モード』を参照してください。

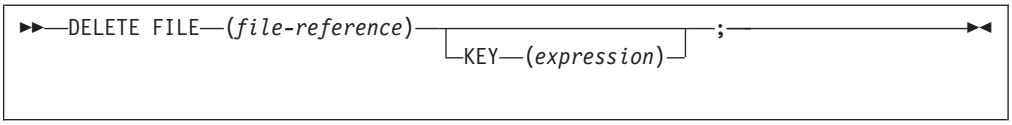


キーワードの指定順序は任意です。

based-variable
添え字なしのレベル 1 の基底付き変数でなければなりません。

DELETE ステートメント

DELETE ステートメントは、UPDATE ファイルからレコードを削除します。



キーワードの指定順序は任意です。 KEY オプションを省略すると、最後に読み取られたレコードが削除されます。別の READ ステートメントが処理されるまで、KEY が指定されていない後続の DELETE または REWRITE ステートメントを実行することはできません。 KEY オプションが含まれる場合、キーによってアドレス指定されたレコードが検出されると、これを削除します。

データ伝送ステートメントのオプション

レコード単位データ伝送ステートメントでどのオプションを使用できるかは、ファイルの属性や、関連データ・セットの特性によって異なります。

FILE オプション

どのレコード単位データ伝送ステートメントでも、FILE オプションを指定しなければなりません。このオプションは、行われる操作の対象となるファイルを表します。 FILE オプションの例は、本節の各ステートメントごとに記載しています。指定したファイルが現行処理でまだオープンされていないときは、暗黙にオープンされます。

FROM オプション

FROM オプションは、レコードをどの要素変数または集合変数から書き出すかを指定します。このオプションは、任意の OUTPUT または DIRECT UPDATE ファイルを参照する WRITE ステートメント内で使用しなければなりません。また、任意の UPDATE ファイルを参照する REWRITE ステートメント内で使用することもできます。

変数が集合の場合、その変数は連結ストレージ内になければなりません。位置合わせされていない固定長ビット・ストリングは、場合によっては許可されないことがあります (詳細は、 293 ページの『データ伝送』を参照してください)。

FROM で参照する変数は、可変長のエレメント・ストリング変数であってもかまいません。 FROM オプションを指定して WRITE ステートメントを使用すると、現行の長さの可変長ストリングだけが、データ・セットに伝送されます。 VARYINGZ ストリングでは、ヌル終了文字が付加され、これも伝送されます。 VARYING ストリングでは、 ENVIRONMENT 属性の SCALARVARYING オプションがファイルに指定されている場合のみ、長さを指定する 2 バイトの接頭部が付加されます。

レコードは、整数バイトの長さとして伝送されます。したがって、バイト境界に合わせられていないビット・ストリング (または、最初または最後のエレメントがビット・ストリングである構造体) が伝送される場合は、レコードの最初または最後に、ストリングの一部ではないビットが含まれることになります。

SEQUENTIAL UPDATE ファイルを参照する REWRITE ステートメントでは、FROM オプションを省略することができます。最後に読み取られたレコードが、INTO オプションが指定された READ ステートメントで読み取られたものである場

合は、FROM オプションが指定されていない REWRITE ステートメントを実行しても、データ・セット内のそのレコードは変わりません。しかし、最後に読み取られたレコードが、SET オプションが指定された READ ステートメントで読み取られたものである場合は、そのレコード (どのような割り当てによっても更新される) がデータ・セットに書き戻されます。

次の例では、ステートメントは変数 Mas_Rec の値を出力ファイル Master に書き出すように指定しています。

```
write file (Master) from (Mas_Rec);
```

REWRITE ステートメントは、UPDATE ファイルから最後に読み取られたレコードを Mas_Rec で置き換えることを指定しています。

```
rewrite file (Master) from (Mas_Rec);
```

IGNORE オプション

IGNORE オプションは、任意の SEQUENTIAL INPUT または SEQUENTIAL UPDATE ファイルを参照する READ ステートメント内で使用することができます。

IGNORE オプション内の式が計算され、整数値 *n* に変換されます。 *n* がゼロよりも大であると、*n* 個のレコードが無視されます。つまり、同じファイルを参照する後続の READ ステートメントは、 (*n*+1) 番目のレコードをアクセスすることになります。 *n* が 1 未満であると、READ ステートメントにはなにも影響しません。

次の例では、ファイル内の次からの 3 つのレコードを無視することを指定しています。

```
read file (In) ignore (3);
```

INTO オプション

INTO オプションは、論理レコードをどの要素変数または集合変数の中に読み込むかを指定します。このオプションは、任意の INPUT または UPDATE ファイルを参照する READ ステートメント内で使用することができます。

変数が集合の場合、その変数は連結ストレージ内になければなりません。位置合わせされていない固定長ビット・ストリングは、場合によっては許可されないことがあります (詳細は、 293 ページの『データ伝送』を参照してください)。

INTO で参照する変数は、可変長のエレメント・ストリング変数であってもかまいません。 VARYINGZ ストリングでは、各レコードにヌル終了文字が含まれます。VARYING ストリングでは、ファイルに ENVIRONMENT 属性の SCALARVARYING オプションが指定されていると、各レコードにはストリング・データの長さを指定する 2 バイトの接頭部が含まれます。

入力時に SCALARVARYING が宣言されていなければ、ストリングの長さはレコードの長さから計算され、2 バイトの接頭部として付加されます (VARYING ストリングの場合)。 VARYING ビット・ストリングの長さは、この計算で 8 の倍数に切り上げられるので、計算後の長さは宣言された最大の長さより大きくなることがあります。

次の例は、次の順序のレコードを変数 RECORD_1 の中に読み込むことを指定しています。

```
read file (Detail) into (Record_1);
```

KEY オプション

KEY オプションは、レコードを識別するための文字キー、グラフィック・キーまたはワイド文字キーを指定します。このオプションは、INPUT または UPDATE ファイルを参照する READ ステートメント内や、DIRECT UPDATE ファイルを参照する REWRITE ステートメント内で使用することができます。

KEY オプションは、KEYED ファイルにのみ適用されます。 KEY オプションは、ファイルが DIRECT 属性を持つ場合には必ず指定しなければなりませんが、ファイルが SEQUENTIAL 属性および KEYED 属性を持つ場合には指定してもしなくてもかまいません。

KEY オプション内の式が計算され、それが文字、グラフィック、またはワイド文字でないときは、キーを表す文字値に変換されます。どのレコードが読み取られるかは、この文字値、グラフィック値、またはワイド文字値によって決まります。

次の例では、変数 Stkey に入っている文字値で識別されるレコードを、変数 Item の中に読み込むように指定しています。

```
read file (Stpck) into (Item) key (Stkey);
```

KEYFROM オプション

KEYFROM オプションは、レコードの伝送先となるデータ・セット上のレコードを識別するための文字キー、グラフィック・キー、またはワイド文字キーを指定します。このオプションは、ファイル KEYED OUTPUT または DIRECT UPDATE を参照する WRITE ステートメント内、もしくは LOCATE ステートメント内で使用することができます。

KEYFROM オプションは、KEYED ファイルにのみ適用します。式が計算され、それが文字、グラフィック、またはワイド文字でないときは、文字ストリングに変換されて、そのレコードが次に書き出されるときのキーとして使用されます。

相対データ・セットは、KEYFROM オプションを使用して作成することができます。この場合、レコード番号をキーとして指定します。

KEYFROM オプションを使用して、REGIONAL(1) データ・セットを作成することができます。この場合、領域番号をキーとして指定します。

索引付きデータ・セットの場合、KEYFROM は、記録されるキーを指定しますが、その長さはデータ・セットに指定されたキーの長さと等しくなければなりません。

次の例では、Loanrec の値をファイル Loans のレコードとして書き出すことと、Loanno の文字ストリング値をキーとして使用する (このキーを使用すれば、このレコードを取り出すことができる) ことを指定しています。

```
write file (Loans) from (Loanrec) keyfrom (Loanno);
```

KEYTO オプション

KEYTO オプションは、レコードのキーが割り当てられる文字変数、グラフィック変数、またはワイド文字変数を指定します。 KEYTO オプションでは、STRING 以外の任意のストリング疑似変数を指定することができます。数字ピクチャー指定を付けて宣言した変数を指定することはできません。 KEYTO オプションは、SEQUENTIAL INPUT ファイル、または SEQUENTIAL UPDATE ファイルを参照する READ ステートメントで指定することができます。

KEYTO オプションは、KEYED ファイルにのみ適用されます。

KEYTO 変数への割り当ては、常に、INTO 変数への割り当て後に行われます。キー指定が正しくないことが検出されると、KEY 条件が起こります。値は次のように割り当てられます。

- 索引付きデータ・セットの場合、記録されたキーは、文字変数の宣言された長さに合わせて、右側が埋め込まれるかまたは切り捨てられます。
- *相対* データ・セットの場合、レコード番号は文字ストリングに変換されます。このとき、先行ゼロは、文字変数の宣言された長さに合わせて、消去されるか、切り捨てられるか、または左側に埋め込まれます。
- REGIONAL(1) データ・セットの場合、8 文字の領域番号は、文字変数の宣言された長さに合わせて、左側に埋め込まれるかまたは切り捨てられます。 文字変数が可変長のときは、領域番号の先行ゼロは切り捨てられ、ストリングの長さは有効数字の桁数にセットされます。領域番号がすべてゼロであるときは、ゼロが 1 つだけになるように切り捨てられます。

このような埋め込みまたは切り捨てに関しては、KEY 条件は起こりません。

次の例では、ファイル `Detail` 内の次のレコードを変数 `Invntry` の中に読み込むことと、そのレコードのキーを変数 `Keyfld` に割り当てることを指定しています。

```
read file (Detail) into (Invntry) keyto (Keyfld);
```

SET オプション

SET オプションは、READ ステートメントまたは LOCATE ステートメントで使用することができます。 READ ステートメントの場合、読み取られるレコードを指すためにセットされるポインター変数を指定します。 LOCATE ステートメントの場合、次に出力するレコードを指すためにセットされるポインター変数を指定します。

LOCATE ステートメントで SET オプションを省略すると、レコード変数と一緒に宣言されているポインターがセットされます。 VARYING ストリングを伝送するときは、そのファイルに対して SCALARVARYING オプションを指定しておかなければなりません。

次の例では、次の順次レコードのバッファー内の場所を指すように、ポインター変数 `P` の値をセットすることを指定しています。

```
read file (X) set (P);
```

処理モード

レコード単位データ伝送には、2 通りのデータ処理方法があります。

移動モード　データを変数に移動させたり、変数から移動させたりして、データを処理します。

位置指定モード
データをバッファーに入れたまま処理することができます。データ伝送ステートメントが実行されると、バッファー内のそのレコードに割り振られているストレージの位置がポインター変数に割り当てられます。位置指定モードは BUFFERED ファイルにのみ適用されます。

どちらの処理モードが使用されるかは、指定するデータ伝送ステートメントとオプションによって決まります。

移動モード

移動モードでは、READ ステートメントは、レコードをデータ・セットから INTO オプションで指定された変数に伝送します。 WRITE ステートメントまたは REWRITE ステートメントは、レコードを FROM オプションで指定された変数からデータ・セットに伝送します。 INTO オプションや FROM オプションで指定する変数は、どのストレージ・クラスであってもかまいません。

次に、移動モードの入力の例を示します。

```
Eof_In = '0'b;
on endfile(In) Eof_In = '1'B;
read file(In) into(Data);
do while (¬Eof_In);
⋮
      /* process record  */
read file(In) into(Data);
end;
```

位置指定モード

位置指定モードでは、バッファーの位置がポインター変数に割り当てられます。基底付き変数は、レコードについて記述しています。別々の基底付き変数を使用すれば、同じデータを別々に解釈させることができます。また、位置指定モードを使用すれば、自己定義レコードを読み取ることもできます。自己定義レコードでは、そのレコードの一部に入っている情報が、そのレコードの残りの部分の構造体を表します。たとえば、そのような情報としては、配列の境界や、そのデータの属性に合わせてどの基底付き構造体を使用する必要があるかを表すコードなどがあります。

SET オプションが指定された READ ステートメントは、レコードが入っているバッファーを示すために、SET オプションで指定されたポインター変数をセットします。したがって、このポインター変数で修飾された基底付き変数を使用すれば、レコードのデータを参照することができます。

このポインター値は、同じファイルを参照する次の READ ステートメントまたは CLOSE ステートメントが実行されるまで有効です。

SET オプションで指定したポインター変数が使用されるか、または SET を省略した場合は基底付き変数の宣言内で指定したポインター変数が使用されます。このポインター値は、同じファイルを参照する次のステートメント LOCATE、WRITE、または CLOSE が実行されるまで有効です。 REFER オプションで指定された基底付き変数のコンポーネントを初期設定します。

LOCATE ステートメントは、次のレコードを構築することができるだけの大きさを持つ区域に、ポインター変数をセットします。

LOCATE ステートメントを実行したあとは、LOCATE ステートメントによってセットされたポインター変数で修飾された基底付き変数に、値を直接割り当てることができます。

次に、位置指定モードの入力の例を示します。

```
dcl 1 Data based(P),
2
:
;

on endfile(In)
;
read file(In) set(P);
do while (¬endfile(In));
:
                    /* process record */
read file(In) set(P);
end;
```

次に、位置指定モードの出力の例を示します。

```
dcl 1 Data based(P);
 2
:
;

do while (More_records_to_write);
locate Data file(Out);
:
            /* build record */
end;
```

第 13 章 ストリーム指向データ伝送

データ伝送ステートメント 304
GET ステートメント 304
PUT ステートメント 305
データ伝送ステートメントのオプション 305
COPY オプション 305
データ指定オプション 306
FILE オプション 308
LINE オプション 308
PAGE オプション 309
SKIP オプション 309
STRING オプション 310
データ・リスト項目の伝送 311
データ・ディレクティブのデータ指定 312
データ・ディレクティブ・データの制限 313
データ・ディレクティブ・データの構文 . . . 313
GET データ・ディレクティブ 314
PUT データ・ディレクティブ 316
編集ディレクティブのデータ指定. 317
GET 編集ディレクティブ 319
PUT 編集ディレクティブ 320
FORMAT ステートメント 321
リスト・ディレクティブ・データ指定 321
リスト・ディレクティブ・データの構文 . . . 322
GET リスト・ディレクティブ 322
PUT リスト・ディレクティブ 323
PRINT 属性 324
ストリーム入出力での DBCS データ 326

この章では、ストリーム指向データ伝送に使用される入出力ステートメントを説明します。ストリーム指向データ伝送とレコード単位データ伝送の両方に適用される機能 (ファイル、ファイル属性、ファイルのオープンとクローズなど) については、277 ページの『第 11 章 入出力』で説明しています。

ストリーム指向データ伝送では、データ・セットは、文字フォーマット、グラフィック・フォーマット、または混合フォーマットのデータ値の連続するストリームとして処理されます。プログラムでは、一般的にレコード境界が無視されます。ただし、データ・セットは一連のデータ行から構成されていると見なされるので、ストリーム指向データ伝送によって作成またはアクセスされるデータ・セットにはそれぞれ、行のサイズが対応しています。通常、1 行はデータ・セットの 1 レコードと同じですが、行サイズは必ずしもレコード・サイズと同じではありません。

ストリーム指向データ伝送ステートメントで FILE オプションを指定する代わりに STRING オプションを指定すれば、内部でデータを移動させることができます。STRING オプションを指定したときは入出力操作ではありませんが、この章では STRING オプションの使用方法も説明します。

ストリーム指向データ伝送は、リスト・ディレクティブ伝送、データ・ディレクティブ伝送、編集ディレクティブ伝送の 3 種類に分けられます。

リスト・ディレクティブ・データ伝送
データ・リスト項目の値が伝送されます。ストリーム内の値のフォーマットを指定する必要はありません。それらの値は、外部では、ブランクまたはコンマで区切られた定数のリストとして記録されます。

データ・ディレクティブ・データ伝送
データ・リスト項目の値だけでなく、それらの名前も伝送されます。ユーザーがストリーム内の値のフォーマットを指定する必要はありません。たとえ DBCS データが存在しなくても、変数名に DBCS 文字が含まれる場合は、ENVIRONMENT 属性の GRAPHIC オプションを指定する必要があります。

編集ディレクティブ・データ伝送
データ・リスト項目の値が伝送されますが、ユーザーがストリーム内の値のフォーマットを指定する必要があります。それらの値は、外部では、フォーマット・リストに従って 1 文字ずつ (またはグラフィック 1 文字ずつ) 処理される文字または漢字ストリングとして記録されます。

以降の節では、データ伝送ステートメントとそれらのオプション、およびリスト・ディレクティブ・データ、データ・ディレクティブ・データ、編集ディレクティブ・データの指定方法について説明します。 2 バイト文字の使用方法については、326 ページの『ストリーム入出力での DBCS データ』を参照してください。

データ伝送ステートメント

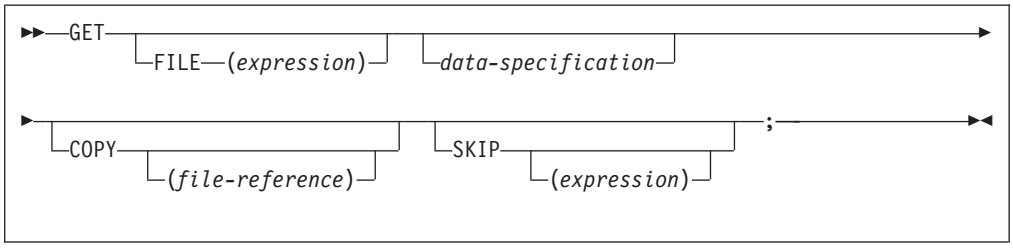
ストリーム指向データ伝送では、GET ステートメントと PUT ステートメントを使用します。 GET ステートメントおよび PUT ステートメントを使用して処理できるのは連続ファイルだけです。

データ値が割り当てられる変数または疑似変数や、データ値の伝送元の式は、通常、それぞれの GET または PUT ステートメントのデータ指定で指定します。 また、これらのステートメントでは、データ値の送り出し元や受け入れ先を表すオプションを指定したり、ストリーム内で先行のデータ値から相対的にどの位置にデータ値があるかを示すオプションを指定したりすることもできます。ストリーム方式のデータ伝送ステートメントのオプションについては、305 ページの『データ伝送ステートメントのオプション』を参照してください。

GET ステートメント

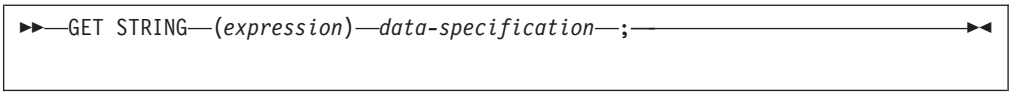
GET ステートメントは、STREAM 入力データ伝送ステートメントであり、次のいずれかを行います。
- データ・セットからのデータ値を、1 つまたは複数の変数に割り当てます。
- ストリングからのデータ値を、1 つまたは複数の変数に割り当てます。

ストリーム入力ファイルの場合、次の GET ステートメントの構文を使用してください。



キーワードの指定順序は任意です。 SKIP オプションを指定しない場合、データ指定を書かなければなりません。

ストリングから伝送する場合、次の GET ステートメントの構文を使用してください。

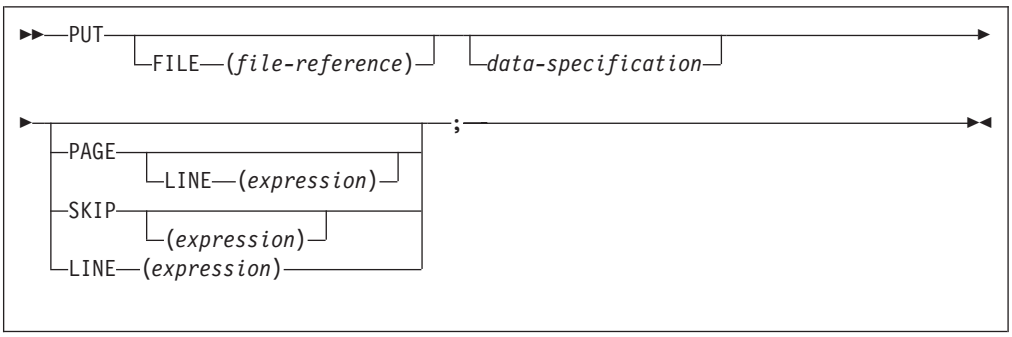


FILE オプションまたは STRING オプションが指定されない場合、FILE(SYSIN) が想定されます。SYSIN は暗黙に宣言された FILE STREAM INPUT EXTERNAL です。

PUT ステートメント

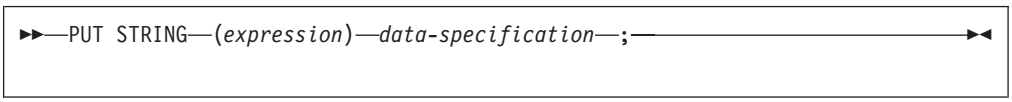
PUT ステートメントは、次のことが可能な STREAM 出力データ伝送ステートメントです。
- 値をストリーム出力ファイルへ伝送します。
- 値を文字変数に割り当てます。

ストリーム出力ファイルを使用する場合、次の PUT ステートメントの構文を使用してください。



キーワードの指定順序は任意です。いずれかの制御オプション (PAGE、SKIP、または LINE) を指定するときにのみ、データ指定を省略することができます。

しかし、文字ストリングへ伝送する場合は次の PUT ステートメントの構文を使用してください。



データ伝送ステートメントのオプション

COPY オプション

COPY オプションは、読み取られたソース・データ・ストリームを変更せずに、STREAM OUTPUT ファイルへ書き出すことを指定します。ファイル参照の指定を省略すると、出力ファイル SYSPRINT が想定されます。入力ストリーム中に新しいレコードが現れると、COPY ファイルで新しいレコードが始められます。 以下に例を示します。

```
get file(sysin) data(A,B,C) copy(DPL);
```

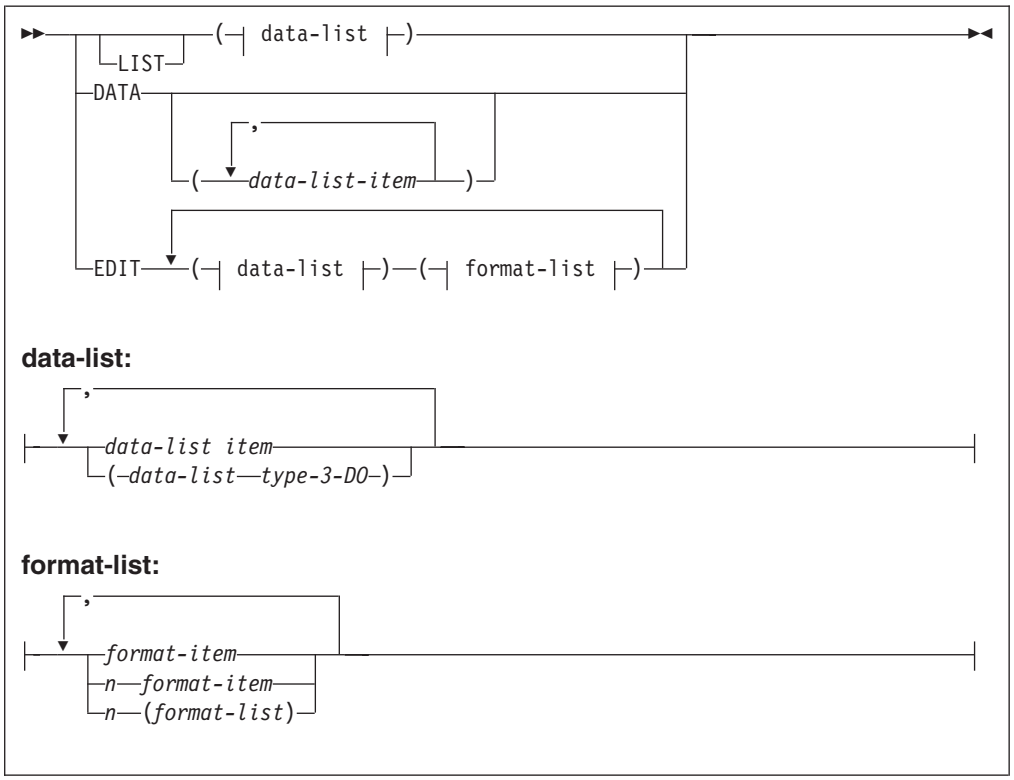
この例では、入力ストリーム中の A、B、および C に割り当てられている値が、同じ名前を持つ変数に伝送されるだけでなく、入力ストリーム中にあるときとまったく同じ形のままでファイル DPL に書き出されます。入力時にスキップされ、内部変数に伝送されないデータ値も、そのまま出力ストリーム中にコピーされます。

COPY オプションが指定された GET ステートメントの実行中になんらかの条件が起こり、ON ユニットに制御が移され、その ON ユニット内で同じファイルを参照する別の GET ステートメントが実行されたとします。次に、その ON ユニットから最初の GET ステートメントに制御が戻されると、その GET ステートメントは、COPY オプションに指定されなかった場合と同様に実行されます。 ON ユニット内で、COPY オプションに関連付けられているファイルを参照する PUT ステートメントが実行された場合には、伝送されるデータの位置は、最後に伝送された COPY データ項目のすぐあとになるとは限りません。

COPY オプションで指定したファイルが現行プログラム内でまだオープンされていないときは、そのファイルは、ストリーム出力伝送用に現行プログラム内で暗黙にオープンされます。

データ指定オプション

GET ステートメントや PUT ステートメント内のデータ指定は、伝送されるデータを指定するためのものです。



GET または PUT ステートメントでデータ・リストを記入し、キーワード LIST、DATA、EDIT のいずれも付けなかったときには、LIST が想定されます。

重要: データ・リストの前に LIST、DATA、または EDIT が付いていないステートメントでは、データ・リストは、GET キーワードまたは PUT キーワードの**直後に**なければなりません。必須オプションは、データ・リストのあとに指定してください。

DATA 312 ページの『データ・ディレクティブのデータ指定』を参照してください。

EDIT 317 ページの『編集ディレクティブのデータ指定』を参照してください。

LIST 321 ページの『リスト・ディレクティブ・データ指定』を参照してください。

data-list item

入力の場合、編集ディレクティブ伝送とリスト・ディレクティブ伝送のデータ・リスト項目 (data-list item) は、エレメント、配列、または構造変数のいずれかになることができます。データ・ディレクティブ・データ指定のデータ・リスト項目は、要素変数、配列変数、または構造変数のいずれでもかまいません。データ・ディレクティブのデータ・リスト内の名前は、添え字付きまたはロケーター修飾付きであってはなりません。ただし、修飾付き (つまり、構造体のメンバー) またはストリング・オーバーレイ定義済みの名前を使用することはできます。

出力の場合、編集ディレクティブ・データおよびリスト・ディレクティブ・データ指定のデータ・リスト項目は、要素式、配列式、または構造式のいずれでもかまいません。データ・ディレクティブ・データ指定のデータ・リスト項目は、要素変数、配列変数、または構造変数のいずれでもかまいません。ロケーター修飾付きであってはなりませんが、修飾付き (つまり、構造体のメンバー) またはストリング・オーバーレイ定義済みの名前を使用することはできます。

データ・リスト項目のデータ・タイプとしては、任意の計算データを使用できます (CONSTANT または VALUE 属性がない場合に限り)。また PUT ステートメントでは、データ・タイプとして
POINTER、HANDLE、OFFSET、ENTRY、FILE、または LABEL も使用できます。データ・タイプがこれらの非計算タイプのいずれかである場合、項目の内容は、16 進イメージによって伝送されます (PUT DATA の場合、その 16 進イメージは引用符で囲まれ、その後に接尾部 BX が付けられます)。

データ・リスト内の配列変数または構造変数は、データ・リスト内に *n* 個の項目 (*n* は、その配列または構造体のエレメント項目の数) を書くのと同じことです。編集ディレクティブ伝送のときは、それぞれのエレメント項目の 1 つがデータ・フォーマット項目の別々の使用方法に関連付けられます。

data-list type-3-DO

Type 3 DO 指定の構文は、 209 ページの『DO ステートメント』で説明しています。 type 3 DO が指定されたデータ・リスト項目は、GET ステートメントのデータ・ディレクティブ・データのリストでは使用できません。

最後の反復指定が処理されると、次のデータ・リスト項目の処理が続けられます。

構文図で示されているように、それぞれの反復指定は括弧で囲まなければなりません。データ指定として 1 つの反復指定だけを書くときは、データ・リストを囲むための括弧と反復指定を囲むための括弧が別々に必要になるので、外側に 2 組の括弧が必要になります。

反復指定がネストされているときは、最右端の DO が、ネスティングの外側のレベルになります。 以下に例を示します。

```
get list (((A(I,J)
     do I = 1 to 2)
     do J = 3 to 4));
```

上の例では、添え字を区別するための括弧の他に、三組の括弧があります。最外部の一組の括弧は、データ指定に必要な括弧です。次の一組は、外側の反復指定に必要な括弧です。 3 番目の一組は、内側の反復指定に必要な括弧です。

このステートメントは、次のネストされた DO グループと機能的に同等です。

```
do J = 3 to 4;
   do I = 1 to 2;
     get list (A (I,J));
   end;
end;
```

上の例では、次の順序で、配列 A のエレメントに値が割り当てられます。

```
A(1,3), A(2,3), A(1,4), A(2,4)
```

format list

format-list (フォーマット・リスト) の説明については、317 ページの『編集ディレクティブのデータ指定』を参照してください。

FILE オプション

FILE オプションは、行われる操作の対象となるファイルを指定します。そのファイルは、STREAM ファイルでなければなりません。ファイル・タイプ・データ項目の宣言方法については、279 ページの『ファイル』を参照してください。

GET ステートメントに FILE オプションあるいは STRING オプションのいずれも指定されていない場合は、入力ファイル SYSIN がデフォルトです。PUT ステートメントにいずれも指定されていない場合は、出力ファイル SYSPRINT がデフォルトです。

LINE オプション

LINE オプションは、PRINT ファイルの場合にのみ指定することができます。
LINE オプションは、データ・セットの新しい現在行を定義します。式が計算され、整数値 *n* に変換されます。現行ページの *n* 番目の行が新しく現在行になります。現行ページで少なくとも *n* 行がすでに書き出されている場合や、OPEN ステートメントの PAGESIZE オプションで指定した限界値より *n* の方が大きい場合は、ENDPAGE 条件が起こります。 *n* がゼロ以下の場合には、値として 1 が使用されます。 *n* が現在行を指定していると、ENDPAGE 条件が起こります。ただし、ファ

イルが 1 桁目に位置づけられているときは例外で、その場合は、SKIP(0) オプションを指定したときと同じ結果になります。

データ指定 (ある場合) によって定義された値が伝送される前に、LINE オプションの効力が生じます。 1 つの PUT ステートメントで PAGE オプションと LINE オプションの両方を指定したときは、まず PAGE オプションが適用されます。 以下に例を示します。

```
put file(List) data(P,Q,R) line(34) page;
```

この例では、変数 P、Q、R の値が、新しいページの 34 行目から、データ・ディレクティブ・フォーマットで印刷されます。

ファイルのオープン後、最初の GET ステートメントで LINE オプションを指定したときの効果については、 286 ページの『OPEN ステートメント』を参照してください。

対話モードで端末に出力する場合、LINE オプションを指定すると 3 行スキップされます。

PAGE オプション

PAGE オプションは、PRINT ファイルの場合にのみ指定することができます。このオプションは、データ・セット内の新しい現行ページを定義します。 1 つの PUT ステートメントで PAGE と LINE の両方を指定したときは、まず PAGE オプションが適用されます。データ指定 (ある場合) によって定義された値が伝送される前に、PAGE オプションの効力が生じます。

PAGE オプションが指定された PUT ステートメントが実行されるか、PAGE フォーマット項目が検出されるか、または ENDPAGE 条件が起こるまで、現行ページのままです。上記のいずれかが起こると、新しいページが定義されます。新しいページでは 1 行目が暗黙に定義されます。

対話モードで端末に出力する場合、PAGE オプションを指定すると 3 行スキップされます。

SKIP オプション

SKIP オプションは、データ・セット内の新しい現在行 (またはレコード) を指定します。式が計算され、整数値 *n* に変換されます。データ・セットは、現在行 (レコード) から数えて *n* 番目の行 (レコード) の先頭に位置付けられます。 *expression* (式) を省略すると、デフォルトとして SKIP(1) が使用されます。

データ指定 (ある場合) によって定義された値が伝送される前に、SKIP オプションの効力が生じます。 以下に例を示します。

```
put list(X,Y,Z) skip(3);
```

この例では、変数 X、Y、Z の値が、出力ファイル SYSPRINT の現在行の後の 3 行目の行から印刷されます。

PRINT 以外のファイルや入力ファイルの場合に、SKIP オプションの式の値がゼロ以下のときには、1 の値が使用されます。 PRINT ファイルの場合に、*n* がゼロ以下であると、現在行の先頭に位置づけられます。

ファイルのオープン後、最初の GET ステートメントで SKIP オプションを指定したときの効果については、 286 ページの『OPEN ステートメント』を参照してください。

SKIP(n) が発行されたときに現行ページに *n* 行未満しか残っていないと、ENDPAGE 条件が起こります。

会話モードを使って端末で印刷する場合、SKIP(n) の *n* が 3 より大きいときは、SKIP(3) と見なされます。3 行より多くスキップすることはできません。

STRING オプション

GET ステートメントおよび PUT ステートメントで STRING オプションを指定すると、主記憶域とデータ・セットの間をデータが伝送されるのではなく、主記憶域のある記憶場所から別の記憶場所へデータが伝送されます。 DBCS データ項目は、STRING オプションを指定して使用することはできません。

STRING オプションが指定された GET ステートメントは、文字ストリングに変換したあとで、式から得られるデータ値をデータ・リスト項目に割り当てることを指定します。このオプションを使用する各 GET 操作は、常に、そのストリングの左端の文字位置から開始します。そのストリングの文字数が、データ指定で指定した文字の総数より少ないときは、ERROR 条件が起こります。

STRING オプションが指定された PUT ステートメントは、データ・リスト項目の値を、指定された文字変数または疑似変数に割り当てることを指定します。 PUT 操作は、必要に応じて変換を行ったあと、ストリングの左端の文字位置から値の割り当てを開始します。ブランクや区切り文字は、通常の入出力操作と同様に挿入されます。ストリングにデータが入りきらないときは、ERROR 条件が起こります。

GET DATA ステートメントに STRING オプションが付いているときは、NAME 条件は起こりません。その代わりに、GET DATA ステートメントに FILE オプションがついており、NAME 条件が起こるというような状況では、ERROR 条件が起こります。

STRING オプションには、次のような制約事項があります。
- COLUMN 制御フォーマット・オプションと STRING オプションを併用することはできません。
- PUT ステートメントの STRING オプションに疑似変数を指定することはできません。

STRING オプションは編集ディレクティブ伝送で使用する場合に最も役立ちます。このオプションを使用すれば、データ収集またはデータ分散処理を 1 つのステートメントで行うことができ、また、レコード単位のステートメントによって伝送される文字ストリングをストリーム指向で処理することもできます。

以下に例を示します。

```
read file (Inputr) into (Temp);
get string(Temp) edit (Code) (F(1));
If Code = 1 then
   get string (Temp) Edit (X,Y,Z)
      (X(1), 3 F(10,4));
```

この READ ステートメントは、入力ファイル Inputr からレコードを読み取ります。最初の GET ステートメントは STRING オプションを使用して、そのレコードの最初のバイトからコードを取り出し、それを Code に割り当てます。そのコードが 1 のときは、2 番目の GET ステートメントが STRING オプションを使用して、そのレコードに入っている値を X、 Y、および Z に割り当てます。 2 番目の GET ステートメントは、ストリング Temp の最初の文字を無視するように指定しています (フォーマット・リスト内の X(1) フォーマット項目)。無視された文字は、最初の GET ステートメントで Code に割り当てられた文字と同じものです。

PUT ステートメント内の STRING オプションの例は次のとおりです。

```
put string (Record) edit
   (Name)    (X(1), A(12))
   (Pay#)    (X(10), A(7))
   (Hours*Rate) (X(10), P'$999V.99');

write file (Outprt) from (Record);
```

PUT ステートメントはスペース処理フォーマット項目 X(1) を使用して、文字変数に割り当てられる最初の文字が 1 つのブランクであることを指定しています。このブランクは、印刷前にシングル・スペースの行送りをすることを表す ANS 縦方向キャリッジ位置指定文字コードです。このあと、変数 Name の値、変数 Pay# の値、および式 Hours*Rate の値が割り当てられます。 WRITE ステートメントは、レコード単位の伝送を使用してレコードをファイル Outprt に書き出すように指定します。

STRING オプションで参照した変数を、データ・リスト内の名前や別名として使用してはなりません。 以下に例を示します。

```
declare S char(8) init('YYMMDD');
put string (S) edit
    (substr (S, 3, 2), '/',
    substr (S, 5, 2), '/',
    substr (S, 1, 2))
    (A);
```

この PUT ステートメントが実行されると、S の値は 'MM/DD/YY' ではなく、'MM/bb/MM' になります。最初のデータ項目の伝送後に S がブランクにされるためです。 STRING オプションで指定した変数にもとづく基底付き変数または定義済み変数をデータ・リスト内に書いた場合も、上記と同じような結果になります。

データ・リスト項目の伝送

データ・リスト項目が複素数モードのときは、実数部分が伝送されたあと虚数部分が伝送されます。

データ・リスト項目が配列式の場合は、配列のエレメントは行の大きい順に伝送されます。すなわち、配列の右端の添え字が最も頻繁に変化します。

データ・リスト項目が構造式のときは、構造体のエレメントは、構造体の宣言で指定された順序で伝送されます。

次に例を示します。

```
declare 1 A (10),
        2 B,
        2 C;
put file(X) list(A);
```

このステートメントは、次の順序で出力されます。

```
A.B(1) A.C(1) A.B(2) A.C(2) A.B(3)
    A.C(3)...
```

しかし、次のように宣言する場合は、

```
declare 1 A,
          2 B(10),
          2 C(10);
```

同じ PUT ステートメントを使用しても、次の順序で出力されます。

```
A.B(1) A.B(2) A.B(3) ... A.B(10)
A.C(1) A.C(2) A.C(3) ... A.C(10)
```

リスト・ディレクティブ伝送または編集ディレクティブ伝送の入力ステートメントで使用するデータ・リスト内で、ある変数に値が割り当てられ、同じデータ・リスト内の後方でこの変数が参照されると、その新しい値が使用されます。 以下に例を示します。

```
get list (N,(X(I) do I=1 to N),J,K,);
     substr (Name, J,K));
```

このステートメントが実行されると、下記の順序で値が伝送され、割り当てられます。

1. 新しい値が N に割り当てられます。
2. 反復指定に指定されているとおりに、 X(1)、X(2)、...X(N) の順に、エレメントが配列 X に割り当てられます。割り当てられる項目の数を指定するために、N の新しい値が使われます。
3. 新しい値が J に割り当てられます。
4. 新しい値が K に割り当てられます。

データ・ディレクティブのデータ指定

DATA データ指定の構文については、 306 ページの『データ指定オプション』を参照してください。

データ・リスト項目内の構造体エレメントの名前は、あいまいさが生じない程度に修飾すればよいので、完全に修飾する必要はありません。

データ・リストを省略すると、データ・ディレクティブ・ステートメントで指定可能なすべての計算関係の変数からなるデータ・リストが想定されます。

出力の場合は、データ・リスト内のすべての項目が伝送されます。

データ・ディレクティブ・データの制限

添え字付きの変数はデータ・ディレクティブ入力では、使用できません。

データ・ディレクティブ入出力用のデータ・リストにおける基底付き変数の参照を、明示的にロケーター修飾することはできません。 以下に例を示します。

```
dcl Y based(Q), Z based;
put data(Y);
```

変数 Z は伝送できません。伝送するには、明示的にロケーターで修飾される必要があります。

データ・リストの基底付き変数には、以下の制限があります。

- 変数は、OFFSET 変数を基底とすることはできません。
- 変数を基底付きにするポインターは、DEFINED ストレージであることはできません。
- 変数を基底付きにするポインターが BASED 自身である場合は、基底付きポインターのチェーンは、BASED でも DEFINED でもないポインターで終了しなければなりません。

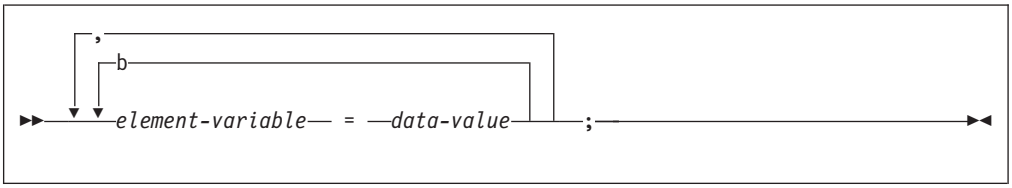
データ・リストの定義済み変数は、以下の制限があります。

- ピクチャー変数または文字変数でなければならない。
- 被制御変数上で定義されていてはならない。
- エレメントまたは配列のクロスセクション上で定義されていてはならない。
- 非定数 POSITION 属性で定義されていてはならない。

タイプ付き構造体をデータ・ディレクティブ入出力ステートメントに使用することはできません。

データ・ディレクティブ・データの構文

データ・ディレクティブ・データの伝送を行うときのストリームは、エレメント割り当てのリストという形をとります。エレメント割り当ては定数の符号の有無にかかわらず、変数名や等号と同じように、文字フォーマットまたはグラフィック・フォーマットで表します。



入力の場合は、エレメント割り当てをブランクまたはコンマのどちらかで区切ることができます。修飾された名前の中のピリオド、添え字、添え字の括弧、および代入記号の前後には、ブランクがいくつあってもかまいません。出力の場合は、エレメント割り当てはブランクで区切られます。 PRINT ファイルの場合、項目はプログラムのタブ設定値に従って区切られます。

ストリーム中のそれぞれのデータ値は、リスト・ディレクティブ伝送のところで説明したいずれかの構文フォーマットをとります。リスト・ディレクティブ伝送の構文についての説明は、322 ページの『リスト・ディレクティブ・データの構文』を参照してください。

ストリーム中のデータ値の長さは、対応する変数の宣言されている属性および (名前も含まれているので) 完全に修飾された添え字付きの名前の長さによって決まります。コード化算術データ、数字データ、ビット・ストリング・データから変換された出力項目の長さは、リスト・ディレクティブ伝送での出力データの長さと同じであり、『第 5 章 データ変換』で説明した文字タイプへのデータ変換の規則に従います。

入力ストリーム中の修飾された名前は、完全に修飾されていなければなりません。

ストリーム中の修飾された名前に介在添え字が付いていてはなりません。たとえば、Y が次のように宣言されているとします。

```
declare 1 Y(5,5),
          2 A(10),
            3 B,
            3 C,
            3 D;
```

この場合は、エレメント名をストリーム中に入れるときは、次のようなフォーマットでなければなりません。

```
Y.A.B(2,3,8)= 8.72
```

GET データ・ディレクティブ

GET ステートメントの詳細については、304 ページの『GET ステートメント』を参照してください。

データ・リストを使用するときは、各データ・リスト項目は要素変数、配列変数、または構造変数でなければなりません。データ・リスト内の名前に添え字が付いていてはなりませんが、修飾された名前を使用することはできます。ストリーム内のすべての名前は、データ・リストに現れている必要があります。ただし、名前の順序は同じである必要はなく、またデータ・リストはストリーム内にない名前を含むことができます。

データ・リスト内の名前がストリーム中にない場合は、その名前を持つ変数の値は変りません。

データ・リスト内にない名前や識別不能な要素変数がストリーム中にあると、NAME 条件が起こります。

セミコロン (引用符で囲まれていないもの) またはファイルの終わりが認識されると、伝送が終了し、データ・リストの指定の有無に関係なく、そのステートメントで実際に伝送されたエレメント割り当ての数が判別されます。

たとえば、次のデータ・リストを考えてみましょう。 A、B、C、 D はそれぞれ要素変数の名前です。

```
Data (B, A, C, D)
```

このデータ・リストを、次のような入力データ・ストリームに関連付けたとします。

```
A= 2.5, B= .0047, D= 125, Z= 'ABC';
```

この場合、C は、データ・リスト内に書かれていますが、ストリーム中にはありません。したがって、C の値は変わりません。 Z はデータ・リスト内に書かれていないので、NAME 条件が起こります。

データ・リスト内に配列名を書く場合、添え字付きの名前をデータ・リスト内に書くことはできませんが、ストリーム中にはその配列の添え字付き参照があってもかまいません。配列全体がストリーム内に現れる必要はありません。実際にストリーム内に現れるこれらのエレメントのみが割り当てられます。添え字が範囲外であるか、欠落しているときは、NAME 条件が起こります。

以下に例を示します。

```
declare X (2,3);
```

このとき、下記のデータ・リストと入力データ・ストリームについて考えてみましょう。

データ指定	**入力データ・ストリーム**
data (X)	X(1,1)= 7.95,
	X(1,2)= 8085,
	X(1,3)= 73;

データ・リストには配列名しか書いていませんが、入力ストリームには配列の個々のエレメントの値を入れることができます。この場合は、3 つのエレメントのみが割り当てられます。配列のその他の部分は変更されません。

データ・リスト内に構造体、小構造体、または構造体エレメントの名前を書く場合、データ・リスト内では完全に修飾する必要はありませんが、ストリーム中には完全修飾名を入れなければなりません。 以下に例を示します。

```
dcl 1 In,
      2 Partno,
      2 Descrp,
      2 Price,
        3 Retail,
        3 Whsl;
```

In.Price.Retail の中に値を読み込むときは、入力データ・ストリームは次のフォーマットになっていなければなりません。

```
In.Price.Retail=1.23;
```

データ指定は、次のうちのどれでもかまいません。

```
data(In)
data(Price)
data(In.Price)
data(Retail)
data(Price.Retail)
data(In.Retail)
data(In.Price.Retail)
```

PUT データ・ディレクティブ

PUT ステートメントの詳細については、305 ページの『PUT ステートメント』を参照してください。

データ・リスト項目は、要素変数、配列変数、構造変数、反復指定のどれでもかまいません。データ・リスト内に書かれている名前が、その値と一緒に、ブランクで区切られたセミコロンで終了するエレメント割り当てのリストの形で伝送されます。 PRINT ファイルの場合は、項目はプログラムのタブ設定値によって区切られます (324 ページの『PRINT 属性』を参照してください)。

それぞれの PUT ステートメントで伝送された最後のデータ項目のあとで、セミコロンがストリーム中に書き出されます。

名前は混合ストリングとして伝送されます。この混合ストリングには、 SBCS 文字または DBCS 文字 (あるいは両方) を含むことができます。 DBCS フォーマットで表現された SBCS 文字がある場合は、まずそれが SBCS に変換されます。 以下に例を示します。

```
put data (<.A>B<.Ckk>);
```

これは次のようにして伝送されます。

```
ABC<kk>=value-of-variable
```

注: 上の例では、<.A>B<.Ckk> はスカラー変数です。

数字変数の文字ストリング値が、有効な算術定数 (符号付きまたは符号なしの) もしくは複素数式を表していないとき、データ・ディレクティブ出力の出力結果を後続のデータ・ディレクティブ入力で使用することはできません。

文字データの場合、文字ストリングの内容は引用符で囲まれて書き出されます。文字ストリング内にあるそれぞれの引用符は、連続する 2 つの引用符で表されます。

次に、データ・ディレクティブ伝送 (入力と出力) の例を示します。

```
declare (A(6), B(7)) fixed;
get file (X) data (B);
do I  = 1 to 6;
     A (I) = B (I+1) + B (I);
end;
put file (Y) data (A);
```

入力ストリーム:

```
B(1)=1, B(2)=2, B(3)=3,
B(4)=1, B(5)=2, B(6)=3, B(7)=4;
```

出力ストリーム:

```
A(1)= 3 A(2)= 5 A(3)= 4 A(4)= 3
A(5)= 5 A(6)= 7;
```

次に例を示します。

```
dcl 1 A,
      2 B FIXED,
      2 C,
```

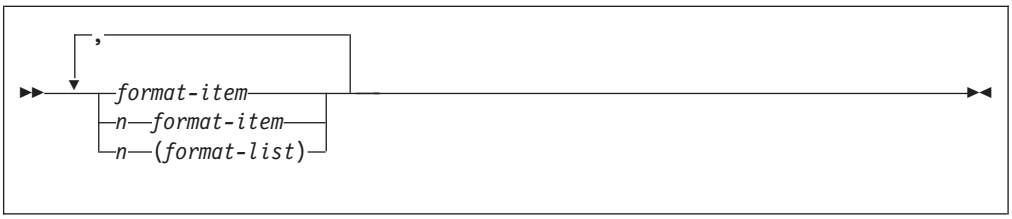
```
        3 D FIXED;
A.B = 2;
A.D = 17;
put data (A);
```

出力ストリーム中のデータ・フィールドは次のようになります。

```
A.B= 2 A.C.D= 17;
```

編集ディレクティブのデータ指定

EDIT データ指定の構文については、306 ページの『データ指定オプション』を参照してください。



n 反復因数を指定します。括弧で囲んだ式か、整数のどちらかです。整数を指定するときは、整数とそのあとのフォーマット項目 (format-item) をブランクで区切らなければなりません。

反復因数は、関連づけられたフォーマット項目またはフォーマット・リストが、連続して *n* 回使用されることを指定します。ゼロまたは負数の反復因数は、対応するフォーマット項目またはフォーマット・リストをスキップし、使用しないことを表します (そのデータ・リスト項目は次のデータ・フォーマット項目に関連付けられます)。

反復因数を式で表したときは、その式が計算され、整数に変換されます。この計算と変換は、一組の繰り返しにつき 1 回だけ行われます。

反復因数に対応するフォーマット項目またはフォーマット・リストは、反復因数のすぐ右に書かれている項目または項目リストです。

format item

データ・フォーマット項目、制御フォーマット項目、またはリモート・フォーマット項目を指定します。フォーマット項目の構文と詳細については、『第 14 章 編集ディレクティブのフォーマット項目』で説明します。

Data-format items

単一データ項目の文字表記またはグラフィック表記を記述するものです。以下のとおりです。

A 文字
B ビット
C 複素数
E 浮動小数点
F 固定小数点
G グラフィック
L 行
P ピクチャー
V 行の表示

制御フォーマット項目

ファイルに関連付けられているデータ・セットのレイアウトを指定するものです。以下のとおりです。

COLUMN
LINE
PAGE
SKIP
X

リモート・フォーマット項目

別の場所にある FORMAT ステートメントのラベル定数を値として持つラベル参照を指定するものです。その FORMAT ステートメント内にその場所に適応するフォーマット項目を書きます。ラベル参照項目は次のものです。

R(ラベル参照)

このラベルとは、FORMAT ステートメントのラベル定数名のことです。 R フォーマット項目の指定については、 336 ページの『R フォーマット項目』を参照してください。

最初のデータ・フォーマット項目は最初のデータ・リスト項目に、2 番目のデータ・フォーマット項目は 2 番目のデータ・リスト項目に、というように対応づけられます。フォーマット・リスト内のデータ・フォーマット項目の数が、関連データ・リスト内の項目の数より少ないときは、そのフォーマット・リストが再使用されます。余分なフォーマット項目は無視されます。

たとえば、フォーマット・リスト内に 5 つのデータ・フォーマット項目があり、関連データ・リストでは、10 個の項目を伝送することを指定したとします。このとき、データ・リスト内の 6 番目の項目は最初のデータ・フォーマット項目に対応づけられます (以下同様)。フォーマット・リスト内に 10 個のデータ・フォーマット項目があり、関連データ・リストには 5 つの項目しかないとすれば、 6 番目から 10 番目までのフォーマット項目は無視されます。

制御フォーマット項目が検出されると、制御処置が実行されます。

PAGE 制御フォーマット項目と LINE 制御フォーマット項目は、PRINT ファイルの場合にのみ使用でき、したがって PUT ステートメント内でだけ使用できます。SKIP、COLUMN、および X フォーマット項目は、入力のときも出力のときも使用できます。

フォーマット項目 PAGE、SKIP、および LINE はそれぞれ、PUT ステートメント (SKIP の場合は GET ステートメントも) の対応するオプションと同じ働きをします。ただし、フォーマット項目は、フォーマット・リストで検出されたときに効力を生じますが、オプションは、データが伝送される前に効力を生じる点が異なっています。

COLUMN フォーマット項目は、GET STRING ステートメントまたは PUT STRING ステートメントで使用することはできません。

ファイルのオープン後、最初の GET または PUT ステートメントで制御フォーマット項目を指定したときの効果については、 286 ページの『OPEN ステートメント』を参照してください。

変数の中に読み込まれた値を、同じデータ・リスト内の後方にある別の変数に対応するフォーマット項目内で使用することができます。

```
get edit (M,String_A,I,String_B)(F(2),A(M),X(M),F(2),A(I));
```

この例では、最初の 2 文字が M に割り当てられます。この M の値は、String_A に割り当てられる文字の個数、および I に 2 文字が割り当てられる前に無視される文字の個数を表します。 I に割り当てられた値は、String_B に割り当てられる文字の個数を示すために使用されます。

入力処理中に変数に割り当てられた値を、後方にあるデータ項目に対応するフォーマット項目内の式で使用することができます。フォーマット項目内の式は、そのフォーマット項目が使用されるたびに計算され、整数に変換されます。

最後のデータ・リスト項目の処理が終わると、伝送が完了します。後続のフォーマット項目 (制御フォーマット項目も含めて) は無視されます。

GET 編集ディレクティブ

GET ステートメントの詳細については、304 ページの『GET ステートメント』を参照してください。

ストリーム中のデータは、文字やグラフィックの連続したストリングであり、ある値と次の値の間に区切り文字はありません。それぞれのデータ値の文字数は、フォーマット・リスト内のフォーマット項目によって指定します。それらの文字は、対応するフォーマット項目に従って解釈されます。データ・リストの処理が終わると、GET ステートメントの実行が終わり、フォーマット項目が残っていてもそれらは無視されます。

それぞれのデータ・フォーマット項目は、データ・リスト項目に関連付けられる文字またはグラフィックの個数を表し、また、データ値をどのように解釈するかも表します。データ値は、必要に応じて変換され、関連データ・リスト項目に割り当てられます。

入力ストリーム中の固定小数点 2 進データ値と浮動小数点 2 進データ値は、必ず 10 進数で表記しなければなりません。 F、P、E のフォーマット項目を使用してそれらの値をアクセスすれば、それらの値は割り当ての時点で 2 進表記に変換されます。

ストリーム中のブランクや引用符はすべて、文字と見なされます。ストリングを引用符で囲まないこと、 引用符を二重にしないこと、およびビット・ストリングを識別するために文字 B や漢字ストリングを識別するために文字 G を使用しないことに注意してください。ストリーム中の文字を、指定された方法で解釈できない場合には、CONVERSION 条件が起こります。

例

```
get edit (Name, Data, Salary)(A(N), X(2), A(6), F(6,2));
```

この例では、次のことを指定します。

- ストリーム中の最初の N 個の文字は、文字ストリングと見なされ、Name に割り当てられます。
- 次の 2 文字はスキップされます。

- その次の 6 文字は、文字フォーマットで Data に割り当てられます。
- さらに次の 6 文字は 10 進固定小数点定数 (符号の有無は任意) と見なされ、Salary に割り当てられます。

PUT 編集ディレクティブ

PUT ステートメントの詳細については、305 ページの『PUT ステートメント』を参照してください。

それぞれのデータ・リスト項目の値が、対応するフォーマット項目によって指定された文字表記またはグラフィック表記に変換され、フォーマット項目によって指定された幅のフィールドに入れてストリーム中に書き出されます。データ・リストの処理が終わると、PUT ステートメントの実行が終わり、フォーマット項目が残っていてもそれらは無視されます。

出力の場合、2 進数項目は 10 進数値に変換されるので、対応する F フォーマット項目や E フォーマット項目では、変換後の 10 進数に換算してフィールドの幅と小数点の位置を指定しなければなりません。 P フォーマット項目の場合は、これらをピクチャー指定で指定します。

出力の場合、出力ストリーム中のデータ値を区切るためにブランクが挿入されるわけではありません。ストリング・データは、フィールド内で左寄せされ、指定の幅を占めます。 算術データは右寄せされます。算術データを文字タイプに変換するときの規則に従うので、(先行ゼロがブランクで置き換えられるほかに) 最高 3 つの先行ブランクが挿入されることがありますが、フィールド内の変換後の算術項目には、通常、最低 1 つはブランクが先頭に付いています。ただし、指定したフィールド幅が短くて先行ブランクを挿入できない場合は、先行ブランクはストリーム中に挿入されません。指定したフィールド幅が短すぎるときは、算術項目では左側が切り捨てられ、ストリング項目では右側が切り捨てられ、 SIZE 条件または STRINGSIZE 条件が起こります。

例 1

```
put edit('Inventory='||Inum,Invcode)(A,F(5));
```

この例では、'Inventory=' が Inum の値と連結されて、その結果の文字ストリングの長さと同じ幅のフィールドに入れてストリーム中に書き出されます。そのあと、Invcode の値が、F フォーマット項目で指定されているとおりに文字に変換され、5 文字の幅のフィールド内で右寄せされて (先行文字はブランク)、ストリーム中に書き出されます。

例 2

次の例では、COLUMN、LINE、PAGE、および SKIP フォーマット項目を、相互に組み合わせて使用する方法を示しています。

```
put edit ('Quarterly Statement')
    (page, line(2), A(19))(Acct#, Bought, Sold, Payment, Balance)
    (skip(3), A(6), column(14), F(7,2), column(30), F(7,2),
    column(45), F(7,2), column(60), F(7,2));
```

この PUT ステートメントは、次のことを指定しています。

1. 見出しとして Quarterly Statement が、出力ファイル SYSPRINT の新しいページの 2 行目に書き出されます。
2. 2 行スキップします。出力内の次の行は、見出しから 3 行目か、もしくは報告書の 5 行目になります。
3. 次の値が書き出されます。
   Acct#、 1 桁目から開始します。
   Bought、 14 桁目から開始します。
   Sold、 30 桁目から開始します。
   Payment、 45 桁目から開始します。
   Balance 60 桁目です。

例 3

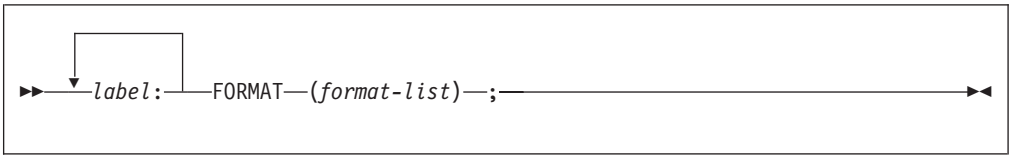
この例では、 Name の値は N 文字のフィールド内で左寄せされた文字ストリングとしてストリーム中に挿入されます。

```
put edit (Name,Number,City) (A(N),A(N-4),A(10));
```

Number は N-4 文字のフィールド内で左寄せされます。また City は 10 文字のフィールド内で左寄せされます。

FORMAT ステートメント

FORMAT ステートメントは、伝送されるデータのフォーマットを制御するために、編集ディレクティブ・データ伝送ステートメントで使用することのできるフォーマット・リストを指定するものです。



label これは、 336 ページの『R フォーマット項目』で説明されているリモート・フォーマット項目 R のラベル参照と同じです。

format list
317 ページの『編集ディレクティブのデータ指定』に説明されているとおりに指定します。

GET または PUT EDIT ステートメントでは、このフォーマット・リスト・オプション内に、R フォーマット項目を記入することができます。 R フォーマット項目で表されるフォーマット・リストのその部分は、そこで識別された FORMAT ステートメントによって与えられます。

FORMAT ステートメントに関連付けられた条件接頭語は無効です。

リスト・ディレクティブ・データ指定

LIST データ仕様の構文については、 306 ページの『データ指定オプション』を参照してください。

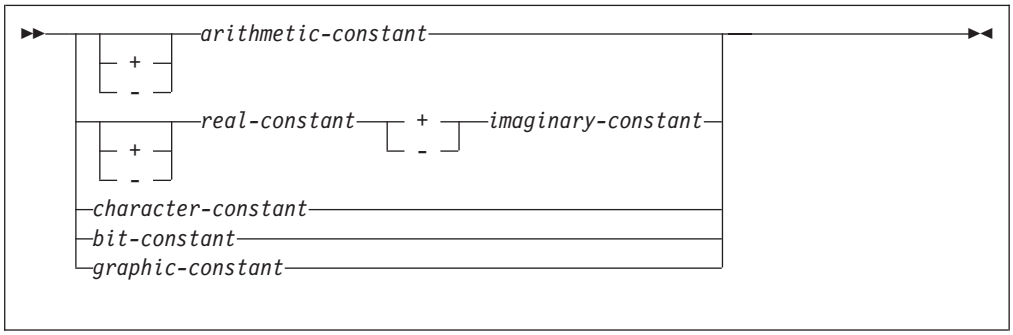
リスト・ディレクティブ・データの指定の例は次のとおりです。

```
list (Card_Rate, Dynamic_Flow)

list ((Thickness(Distance)
     do Distance = 1 to 1000))

list (P, Z, M, R)

list (A*B/C, (X+Y)**2)
```

最後の例のデータ指定には式が書かれているので、これは出力の場合にしか使用できません。これらの式は、ステートメントの実行時に計算され、結果の値がストリームの中に入れられます。

リスト・ディレクティブ・データの構文

入力の場合も出力の場合も、ストリーム中のデータ値は、文字またはグラフィックで表記されます。



ストリングの反復因数を使用することはできません。実定数の前に付ける符号のあとや、複素数式の中央にある (+) または (-) の前後に、ブランクがあってはなりません。

ストリーム中のデータ値の長さは、そのデータ値の属性 (精度や長さも含めて) によって決まります。変換規則およびそれが精度に及ぼす影響の詳細については、77 ページの『第 5 章 データ変換』の中の文字タイプへの変換の箇所で説明しています。

GET リスト・ディレクティブ

GET ステートメントについては、304 ページの『GET ステートメント』を参照してください。

入力の場合、ストリーム中のデータ値はブランクまたはコンマで区切られなければなりません。この分離文字の前後には、ブランクがいくつあってもかまいません。データ・ストリーム中のブランク以外の最初の文字がコンマである場合や、2 つのコンマがブランク (いくつでも構いません) で区切られている場合、それはストリーム中のヌル・フィールドを表します。ヌル・フィールドは、関連データ・リスト項目の値を変更せずそのままにしておくことを表します。

入力の場合、定数または複素数式のリストの伝送が終了するのは、リストが使い尽くされたとき、またはファイルの終わりに達したときです。定数の伝送の場合は、次の GET ステートメントに備えて、ファイルがストリームの中で位置付けられます。

項目をコンマで区切ったとき、次の GET ステートメントの実行時に走査される最初の文字は、コンマのすぐあとの文字です。

```
Xbb,bbbXX
   —
```

項目をブランクだけで区切ったとき、走査される最初の文字は、次の非ブランク文字です。

```
XbbbbXXX
    —
```

ただし、これは、レコードの終わりに達していない場合にあてはまります。レコードの終わりに達した場合は、ファイルはレコードの終わりに位置づけられます。

```
Xbb-bbXXX
   —
```

ただし、レコードの終わりが (コンマ以外の) 非ブランク文字のすぐあとにあり、次のレコードがブランクで始まる場合は、ファイルは次のレコード内の最初の非ブランク文字に位置づけられます。

```
X-bbbXXX
    —
```

レコードがコンマで終っているときは、後続レコードは、次の GET ステートメントで要求されるまで読み取られません。

データが文字定数のときは、前後の引用符が取り除かれて、囲まれていた文字が文字ストリングとして解釈されます。二重引用符は一重引用符と見なされます。

データがビット定数のときは、前後の引用符と末尾の文字 B が取り除かれて、囲まれていた文字がビット・ストリングとして解釈されます。

データが 16 進定数 (X、BX、B4、GX) のときは、前後の引用符と接尾部が取り除かれて、囲まれていた文字が、文字、ビット、または漢字ストリングの 16 進表記として解釈されます。

データが混合定数のときは、前後の引用符と接尾部 M が取り除かれたあと、囲まれていた定数が文字ストリングとして解釈されます。

データがグラフィック定数のときは、前後の引用符と末尾の文字 G が取り除かれて、囲まれていたグラフィックが漢字ストリングとして解釈されます。

データが算術定数または複素数式のときは、定数によってまたは式を計算するときの規則によって暗黙に定義される基数、スケール、モード、および精度を持つコード化算術データとして解釈されます。

PUT リスト・ディレクティブ

PUT ステートメントの詳細については、305 ページの『PUT ステートメント』を参照してください。

データ・リスト項目の (グラフィック以外の) 値が文字表記に変換されて、データ・ストリームに伝送されます。伝送された連続したデータ値は、ブランクで区切られ

ます。 PRINT ファイルの場合は、項目はプログラムのタブ設定値によって区切られます (『PRINT 属性』を参照してください)。

算術値は文字に変換されます。

2 進データ値は、10 進表記に変換されてから、ストリーム中に書き出されます。

数字値の場合は、文字値が伝送されます。

ビット・ストリングは、文字ストリングに変換されます。その文字ストリングは引用符で囲まれて、あとに文字 B が付きます。

ストリングは、次のようにして書き出されます。

- ファイルに PRINT 属性が付いていない場合は、前後に引用符が付けられ、ストリング内の一重引用符やアポストロフィは 2 つの引用符で置き換えられます。フィールドの幅は、ストリングの現在の長さに、付加された引用符の数を加えた長さになります。
- ファイルに PRINT 属性が付いている場合は、前後の引用符が付けられることはなく、ストリング内の一重引用符やアポストロフィが変更されることもありません。フィールドの幅は、ストリングの現在の長さです。

混合ストリングは、次のようにして書き出されます。

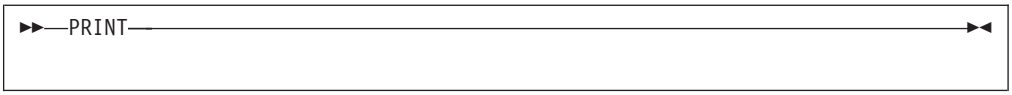
- ファイルに PRINT 属性が付けられていない場合は、SBCS 引用符と文字 M が付けられます。含まれている SBCS 引用符は、2 つの引用符で置き換えられます。
- ファイルが PRINT 属性を持つと、前後の引用符と文字 M が付けられることはなく、含まれている一重引用符は変更されません。

漢字ストリングは次のようにして書き出されます。

- ファイルに PRINT 属性が付けられていない場合は、前後の SBCS 引用符と文字 G が付けられます。前後の引用符は SBCS 引用符なので、ストリング内のグラフィック引用符は一重グラフィック引用符で表されます (つまり、変更されません)。
- ファイルが PRINT 属性を持つと、前後の引用符と文字 G は付けられず、グラフィック引用符は、一重グラフィック引用符になります (つまり変更されません)。

PRINT 属性

PRINT 属性は、STREAM 属性と OUTPUT 属性を持つファイルの場合に指定できます。この属性はそのファイルが印刷される予定であることを示します。すなわち、ファイルと関連したデータを、最初はほかのメディアに書き出されていても、印刷されたページに現れることを示します。



PRINT を指定したときは、PRINT ファイルの各レコードの最初のデータ・バイトは米国標準規格 (ANS) 印刷制御文字を入れるために予約されます。この制御文字は、PL/I によって挿入されます。

リスト・ディレクティブ・データ伝送またはデータ・ディレクティブ・データ伝送で伝送されるデータ値は、自動的に左マージンに合わせられ、また、システムで定義され、事前設定されているタブ位置に合わせられます。

PRINT ファイルのレイアウトは、表 36 にリストされているオプションおよびフォーマット項目を使用して制御することができます。

表 36. PRINT ファイルのオプションとフォーマット項目

ステートメント	ステートメントのオプション	編集ディレクティブのフォーマット項目	結果
OPEN	LINESIZE(n)	-	行の文字数を設定します。
OPEN	PAGESIZE(n)	-	ページの行数を設定します。
PUT	PAGE	PAGE	新しいページにスキップします。
PUT	LINE(n)	LINE(n)	指定された行にスキップします。
PUT	SKIP[(n)]	SKIP[(n)]	指定された行数だけスキップします。
PUT	-	COLUMN(n)	行の中の指定された桁にスキップします。
PUT	-	X(n)	行の中にブランク桁を作って位置を設定します。

LINESIZE と PAGESIZE は、ページの印刷区域 (脚注を除く) の大きさを設定するためのものです。 LINESIZE オプションは、各印刷行に印刷される文字の最大文字数を表します。 PRINT ファイルの場合、このオプションを省略すると、120 文字と見なされます。 PRINT 以外のファイルではデフォルトの値はありません。 PAGESIZE オプションは各印刷ページの最大の印刷行数を指定します。指定しない場合は、デフォルトの値の 60 が使用されます。 以下に例を示します。

```
open file(Report) output stream print PAGESIZE(55) LINESIZE(110);
on endpage(Report) begin;
     put file(Report) skip list (Footing);
     Pageno = Pageno + 1;
     put file(Report) page list ('Page '||Pageno);
     put file(Report) skip (3);
end;
```

この OPEN ステートメントは、ファイル Report を PRINT ファイルとしてオープンします。 PAGESIZE(55) という指定は、各ページに最大 55 行書き出せることを示しています。すでに 55 行が書き出された (またはスキップされた) あと、同じページへの書き出しが試みられると、ENDPAGE 条件が起こります。 ENDPAGE 条件が起こったとき、暗黙に取られる処置は新しいページへのスキップですが、この例に示したように、ON ステートメントを使用してユーザー独自の処置を設定することもできます。

LINESIZE(110) は、ページの各行に最大 110 文字書き出せることを示しています。110 文字を超える行を書き出そうとすると、余分の文字は次の行に書き出されます。

56 行目に書き出そうとすると (または 55 行目を超えてスキップしようとすると)、ENDPAGE 条件が起こり、ここに示した開始ブロックが実行されます。 ENDPAGE 条件は、1 ページあたり 1 回しか起こりません。したがって、ENDPAGE 条件が起こったあと、指定の PAGESIZE を超えて印刷を続けることができます。たとえば、各ページの最下部にフッティングを書き出すような場合には、この方法が役立ちます。

最初の PUT ステートメントでは、1 行スキップすることと、文字ストリングと推定されるフッティングの値を 57 行目に印刷する (ENDPAGE 条件が起こったとき、現在行は常に PAGESIZE+1 です) ことを指定しています。ページ番号 `Pageno` が 1 つだけ増やされ、ファイル `Report` が次のページにセットされ、文字定数 `'Page'` と新しいページ番号が連結されて印刷されます。最後の PUT ステートメントが実行されると 3 行スキップされるので、次は 4 行目から印刷されることになります。制御は、開始ブロックから、ENDPAGE 条件を引き起こした PUT ステートメントに戻されます。ただし、その PUT ステートメントに SKIP または LINE オプションが指定されていても、それらは影響を及ぼしません。

ストリーム入出力での DBCS データ

リスト・ディレクティブ伝送またはデータ・ディレクティブ伝送で DBCS データを使用する場合は、そのファイルの ENVIRONMENT 属性で GRAPHIC オプションを指定しておかなければなりません。また、データ・ディレクティブ伝送では、DBCS データがないときでも、DBCS 名を指定する場合は、GRAPHIC オプションを指定しておかなければなりません。このとき DBCS 継続規則が適用されますが、この規則は 21 ページの『DBCS 継続規則』で説明した規則と同じものです。編集ディレクティブ伝送でグラフィックがどのように処理されるかについては、317 ページの『編集ディレクティブのデータ指定』を参照してください。

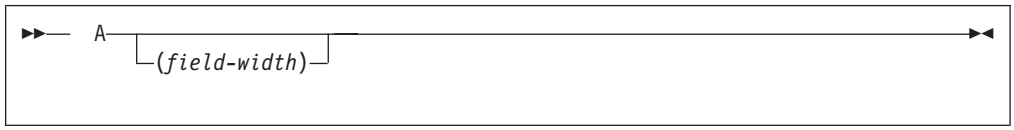
第 14 章 編集ディレクティブのフォーマット項目

A フォーマット項目 327
B フォーマット項目 328
C フォーマット項目 329
COLUMN フォーマット項目 329
E フォーマット項目 330
F フォーマット項目 332
G フォーマット項目 334
L フォーマット項目 335
LINE フォーマット項目 335
P フォーマット項目 335
PAGE フォーマット項目 336
R フォーマット項目 336
例 337
SKIP フォーマット項目 337
V フォーマット項目 338
X フォーマット項目 338

この章では、GET、PUT、または FORMAT ステートメントのフォーマット・リスト内に書くことができる編集ディレクティブの各フォーマット項目を説明します。(317 ページの『編集ディレクティブのデータ指定』も参照してください。) フォーマット項目は、アルファベット順に説明してあります。

A フォーマット項目

文字 (A) フォーマット項目は、文字値の表記を記述するものです。



field-width
データ・ストリーム中で、そのストリングが入っている (または、これから入る) 文字桁の数を指定します。この式は、このフォーマット項目が使用されるたびに計算され、整数値 (負であってはならない) に変換されます。

GET EDIT ステートメントで長さを指定せずに A フォーマット項目を指定すると、コンパイラーから警告メッセージが発行され、その項目は L フォーマット項目として扱われます (エラー・メッセージが発行され、長さ 1 が割り当てられるのではありません)。

入力の場合は、指定した文字数がデータ・ストリームから取り出され、データ・リスト項目に割り当てられます (必要に応じて、変換、切り捨て、または埋め込みが行われます)。入力の場合、フィールド幅を必ず指定しなければなりません。フィールド幅がゼロのときは、ストリングはヌル・ストリングになります。ストリーム中の引用符は、ストリング内の文字と見なされます。

次の例を考えてみてください。

```
get file (Infile) edit (Item) (A(20));
```

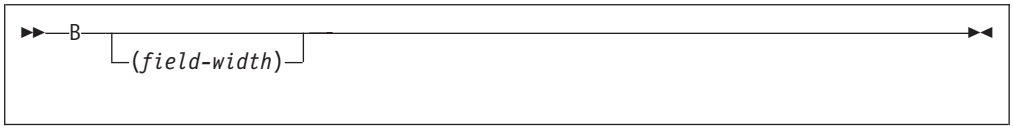
この GET ステートメントは、Infile 内にある次の 20 文字を Item に割り当てます。この値は、フォーマット項目 A(20) で指定された文字表記から、 Item 用に宣言された属性によって指定される表記に変換されます。

出力の場合は、データ・リスト項目が必要に応じて文字ストリングに変換され、指定のフィールド幅に合わせて、切り捨てまたはブランクの追加が右側で行われたあ

と、データ・ストリーム中に書き出されます。フィールド幅がゼロのときは、データ・ストリーム中になにも書き出されません。ストリングを囲むための引用符が挿入されることはなく、ストリング内の引用符も二重にされません。フィールド幅を省略すると、データ・リスト項目の文字ストリングの長さに等しい幅が (必要に応じて、『第 5 章 データ変換』に示した規則に従って変換後に) 想定されます。

B フォーマット項目

ビット (B) フォーマット項目は、ビット値の文字表記を記述するものです。各ビットは、0 または 1 という文字で表されます。



field-width

データ・ストリーム中で、ビット・ストリングが入っている (または、これから入る) 文字桁の数を指定します。この式は、このフォーマット項目が使用されるたびに計算され、整数値 (負であってはならない) に変換されます。

入力の場合は、文字表記のビット・ストリングは指定のフィールド内のどこにあってもかまいません。フィールド内のビット・ストリングの前後にブランクがあってもよく、それらのブランクは無視されます。ビット・ストリングがデータ・リスト項目に割り当てられるとき、必要に応じて変換が行われます。入力の場合、フィールド幅を必ず指定しなければなりません。フィールド幅がゼロのときは、ストリングはヌル・ストリングになります。ストリング中に 0 と 1 以外の文字 (埋め込まれたブランク、引用符、文字 B など) があると、CONVERSION 条件が起こります。

出力の場合は、文字表記のビット・ストリングが指定のフィールド内で左寄せされ、必要に応じて右側で切り捨てまたはブランクの追加が行われます。また、必要に応じて、ビット・ストリングへの変換も行われます。引用符や識別文字 B が挿入されることはありません。フィールド幅がゼロのときは、データ・ストリーム中に何も書き出されません。フィールド幅を省略すると、データ・リスト項目のビット・ストリングの長さに等しい幅が (必要に応じて、『第 5 章 データ変換』に示した規則に従って変換後に) 想定されます。

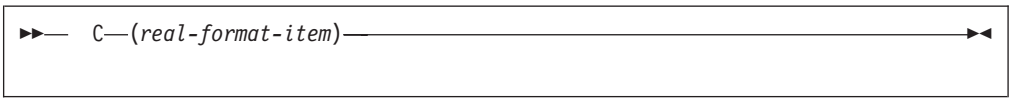
以下に例を示します。

```
declare Mask bit(25);
put file(Maskfle) edit (Mask) (B);
```

この PUT ステートメントは、Mask の値を 0 と 1 からなる 25 文字のストリングとして Maskfle に書き出します。

C フォーマット項目

複素数 (C) フォーマット項目は、複素数データ値の文字表記を記述するものです。実数フォーマット項目 (real-format-item) で、データ・ストリーム中の複素数データ値の実数部分と虚数部分を記述します。



real-format-item
F フォーマット項目、E フォーマット項目、または P フォーマット項目のいずれか 1 つを使用して指定します。 P フォーマット項目では、数字データを記述しなければなりません。

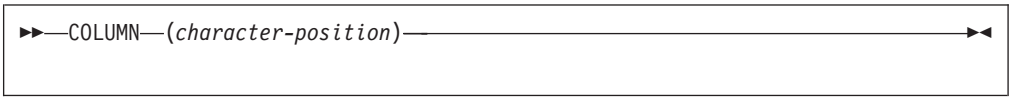
入力の場合は、入力ストリーム中に文字 I があると、CONVERSION 条件が起こります。

出力の場合は、文字 I が虚数部分に付けられることはありません。 2 番目の実数フォーマット項目 (1 つしかないときは最初の実数フォーマット項目) が F 項目か E 項目の場合、虚数部分の値がゼロより小さいときにのみ符号が伝送されます。実数フォーマット項目が P 項目の場合は、S、-、+ のいずれかのピクチャー文字が指定されているときにのみ符号が伝送されます。

I を付けたいときは、データ・リスト内の別個のデータ項目として、複素数項目を指定する変数のすぐあとに I を指定します。そして、I にフォーマット項目 (A または P) を対応させます。 2 番目の実数フォーマット項目が指定された場合、これは無視されます。

COLUMN フォーマット項目

COLUMN フォーマット項目は、ファイルを、現在行または次の行の指定された桁位置に位置付けます。



character-position
式を指定します。この式は、このフォーマット項目が使用されるたびに計算され、整数値 (負であってはならない) に変換されます。

ファイルが現在行の指定された桁位置をまだ通り過ぎていない場合、ファイルはその位置に位置付けられます。ファイルがすでに指定された文字位置の後ろに位置付けされている場合は、現在行は完了し新しい行が開始されます。フォーマット項目は、その後、次の行に適用されます。

指定した桁位置が現在行の最右端の桁位置より右にある場合や、桁位置を表す式の値が 1 より小さい場合は、桁位置 1 が想定されます。

最右端の桁位置は、次のようにして決められます。

- 出力ファイルのときは、行サイズによって決められます。
- 入力ファイルのときは、現在の論理レコードの長さから行サイズが決められ、それによって最右端の桁位置が決められます。

GET STRING ステートメントまたは PUT STRING ステートメントで COLUMN を使用してはなりません。

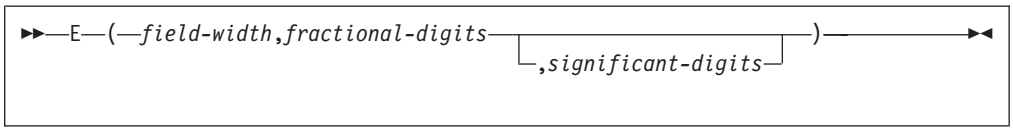
グラフィックまたはワイド文字を含む入力行や出力行に対しては、 COLUMN を使用することはできません。

入力の場合、介在文字桁は無視されます。

出力の場合、介在文字桁にはブランクが入ります。

E フォーマット項目

浮動小数点 (E) フォーマット項目は、実数の浮動小数点 10 進算術データ値の文字表記を記述するものです。



field-width
フィールド内の桁の総数を表します。この値は、フォーマット項目が使用されるたびに計算されて整数値 *w* に変換されます。

fractional-digits
小数点の右側にある小数部の桁数を表します。この値は、フォーマット項目が使用されるたびに計算されて整数値 *d* に変換されます。

significant-digits
小数部になければならない数字の桁数を表しています。この値は、フォーマット項目が使用されるたびに計算されて整数値 *s* に変換されます。

次の関係が成立していなければなりません。

```
w >= s = d+1 or w = 0
```

w ¬= 0 のときは、次のとおりでなければなりません。

```
s > 0, d >= 0
```

w の値は field-width、*d* の値は fractional-digits、 *s* の値は significant-digits です。

入力の場合は、データ・ストリーム中のデータ値は、実数の 10 進浮動小数点定数または実数の 10 進固定小数点定数 (どちらの場合も符号はあってもなくてもよい) でなければならず、それ以外のときは CONVERSION 条件が起こります。データ値は、指定したフィールド内のどこにあってもかまいません。 (便宜上、符号付き指数の前に付ける **E** は省略することができます。)

フィールド幅を表す値には、先行ブランクや後書きブランク、指数部の桁、正符号 (+) または負符号 (-) (オプション) の桁、文字 **E** (オプション) の桁、および小数部の小数点 (オプション) の桁を計算に入れます。

データ値は、指定されたフィールドの任意の場所に現れることができます。ブランクをフィールド内のデータ値の前後に置くことができ、それらは無視されます。フィールド全体がブランクのときには、CONVERSION 条件が起こります。小数点がない場合、*fractional-digits* は小数部の桁のうち仮想小数点の右側にある桁の桁数を示します。小数点が数の中にある場合は、*fractional-digits* の指定は無効にされます。

field-width が 0 のときは、データ・リスト項目にはなにも割り当てられません。

次に例を示します。

```
get file(A) edit (Cost) (E(10,6));
```

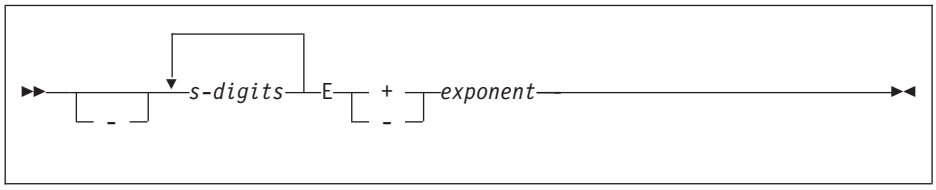
このステートメントは、 A から次の 10 文字を取り出して、それらを浮動小数点 10 進数として解釈します。小数点は小数部の右端 6 桁の前にあると見なされます。この数の値は変数 COST の属性に変換され、この変数に割り当てられます。

出力の場合は、データ・リスト項目が浮動小数点に変換され、必要に応じて四捨五入されます。データの丸めは次のようにして行われます。まず、切り捨てによって右から数字が切り捨てられ、この数字が 5 より大きいか等しい場合は、切り捨てられる数字の左の数字に 1 が追加されます。この加算が行われると、指数が調整されることがあります。

出力ストリーム中に書き出された文字ストリングは、下記のいずれかの構文になります。

注: 文字ストリングのエレメントの間にブランクを入れることはできません。

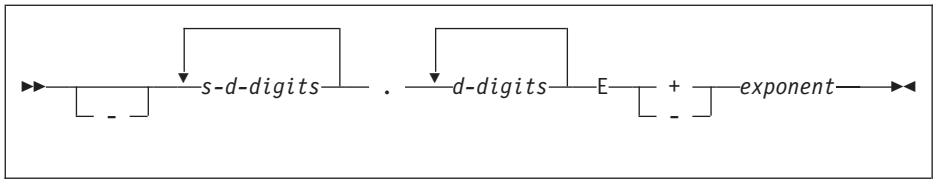
- d=0 の場合



  正の値のときは w >=s+6、負の値のときは w >=s+7 でなければなりません。

  値がゼロ以外のときは、小数部の先頭行がゼロ以外の値になるように指数が調整されます。値がゼロのときは、小数部のすべての桁 (最右端の桁は除く) でゼロ消去が行われます。

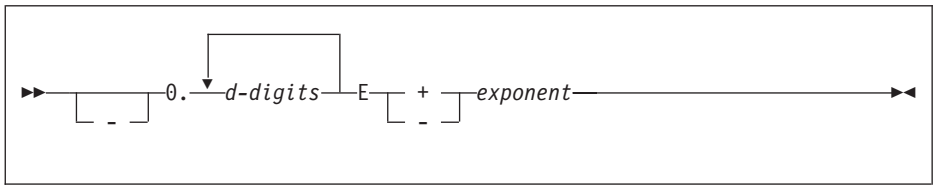
- 0<d<s の場合



正の値のときは w >=s+7、負の値のときは w >=s+8 でなければなりません。

値がゼロ以外のときは、小数部の先頭行がゼロ以外の値になるように指数が調整されます。値がゼロのときは、小数点の左側にあるすべての桁 (最初の桁は除く) でゼロ消去が行われます。ほかのすべての桁にはゼロが入ります。

- d=s の場合



正の値のときは w >=d+8、負の値のときは w >=d+9 でなければなりません。

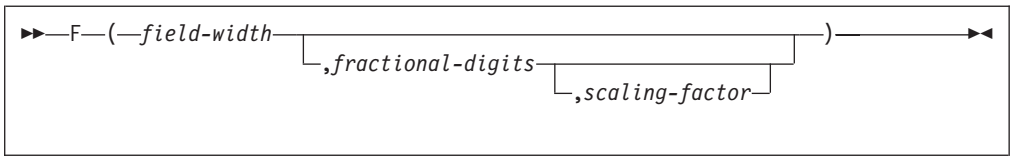
値がゼロ以外の値のときは、小数部分の最初の桁がゼロ以外の値になるように指数が調整されます。値がゼロの値のときは、どの桁にもゼロが入ります。

指数は 4 桁の整数で、4 桁ともゼロである場合もあります。

フィールド幅が短すぎて有効数字や符号が失われる場合は、SIZE 条件が起こります。出力の場合、指定したフィールドが文字ストリングでいっぱいにならないときは、文字ストリングは右寄せされ、左側にブランクが埋め込まれます。

F フォーマット項目

固定小数点 (F) フォーマット項目は、実数の固定小数点 10 進算術値の文字表記を記述するものです。



field-width
フィールド内の桁の総数を表します。この値は、フォーマット項目が使用されるたびに計算されて整数値 *w* に変換されます。変換された値は、負数であってはなりません。

fractional-digits
小数点の右側にある小数部の桁数を表します。この値は、フォーマット項目が使

用されるたびに計算されて整数値 *d* に変換されます。変換された値は、負数であってはなりません。 *fractional-digits* が省略された場合、この値は 0 と見なされます。

scaling-factor
小数部になければならない数字の桁数を表しています。この値は、フォーマット項目が使用されるたびに計算されて整数値 *p* に変換されます。

入力の場合は、データ・ストリーム中のデータ値は、実数の 10 進固定小数点定数 (符号はあってもなくてもよい) でなければならず、それ以外のときは、CONVERSION 条件が起こります。データ値は、指定したフィールド内のどこにあってもかまいません。フィールド内のデータ値の前後にブランクがあってもよく、それらのブランクは無視されます。フィールド全体がブランクのときは、ゼロと解釈されます。

scaling-factor を省略し、しかもフィールド内に小数点がない場合、 *fractional-digits* を表す式は、データ値の桁のうち仮想小数点の右側にある桁の桁数を示します。データ値の中に小数点がある場合は、*fractional-digits* を表す式は無効にされます。

scaling-factor を指定すると、データ・ストリーム中のデータ値に、10 の *p* 乗 (p はスケール因数を表す整数値) が乗算されます。したがって、*p* が正の数ならば、そのデータ値の小数点は、指定された位置から右へ *p* 桁分移動したところにあるものと見なされます。また、*p* が負の数ならば、そのデータ値の小数点は、与えられた位置から左へ *p* 桁分移動したところにあるものと見なされます。小数点がつく位置は、実際に小数点があればその位置であり、小数点が示されていなければ *fractional-digits* を表す式によって指定された位置です。

field-width が 0 のときは、データ・リスト項目にはなにも割り当てられません。

出力の場合は、データ・リスト項目が必要に応じて固定小数点に変換されます。浮動小数点データは、FIXED DECIMAL (N,q) に変換されます。ここで *q* は、*fractional-digits* に指定した値です。ストリーム中のデータ値は、必要に応じて四捨五入された文字表記の実数の 10 進固定小数点であり、指定したフィールドに右寄せで入れられます。

10 進固定小数点タイプから文字タイプへの変換は、通常の変換規則に従って行われます。余分な桁は、変換後のストリングの数値の前にブランクとして示されます。また先行ゼロはブランクに変換されるので (ただし小数点のすぐ左のゼロは除く)、数値の前に余分のブランクが現れることがあります。小数点または負符号 (-) が書き出されるときは、どちらも先行ブランクの 1 つと置き換わります。

field-width だけを指定したときは、数の整数部分だけが書き出されます。小数点は示されません。

field-width と *fractional-digits* を指定したときは、数の整数部分と小数部分の両方が書き出されます。 *fractional-digits* の値 (*d*) がゼロより大きければ、右端 *d* 桁の前に小数点が挿入されます。 *fractional-digits* が *d* より小さければ、後続のゼロが付けられます (値 *d* は *field-width* より小さくなければなりません)。項目の絶対値が 1 より小さいときは、小数点の前に 0 が 1 つ付けられます。小数点の左側にあるすべての桁 (すぐ左の 1 桁は除く) では、先行ゼロのゼロ消去が行われます。

データの丸めは次のようにして行われます。切り捨てによって、右の切り捨てられる数字が 5 以上の場合は、切り捨てられる数字の左の数字に 1 が追加されます。

出力において、データ・リスト項目が 0 より小さい場合は、負符号 (-) が文字表示の前に付けられます。0 より大きいか等しい場合は、符号は付けられません。したがって、負の値の場合には、符号、小数点、および小数点のすぐ左の 0 を桁数として *field-width* に含めることを考慮する必要があります。

field-width が短すぎて文字が失われる場合は、SIZE 条件が起こります。

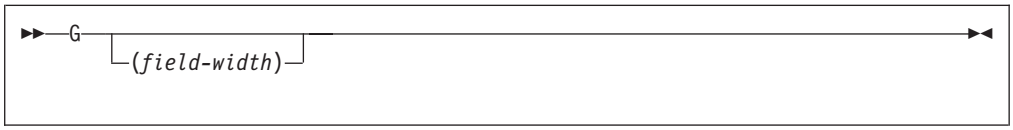
以下に例を示します。

```
declare Total fixed(4,2);
put edit (Total) (F(6,2));
```

この PUT ステートメントでは、Total の値を文字表記の固定小数点数に変換し、出力ファイル SYSPRINT に書き出すことを指定しています。最後の 2 つの数字の前に小数点が挿入され、この数は 6 文字フィールドに右寄せで入れられます。先行ゼロ (小数点のすぐ左にあるゼロは除く) はブランクに変えられ、必要に応じて負符号 (-) が最初の数字の左に付きます。

G フォーマット項目

コンパイラーの場合、グラフィック (G) フォーマット項目は、漢字ストリングの表記を記述するものです。



field-width
データ・ストリーム中で、漢字ストリングが入っている (または入れられる) グラフィック桁 (2 バイト) の個数を指定します。この式は、このフォーマット項目が使用されるたびに計算され、整数値 (負であってはならない) に変換されます。 1 つのグラフィックを形成する 2 バイトの中間で行の終わりになってはなりません。

入力の場合は、指定した個数のグラフィックがデータ・ストリームから取り出され、データ・リスト項目に割り当てられます (必要に応じて、切り捨てや埋め込みが行われます)。入力の場合、*field-width* を必ず指定しなければなりません。*field-width* がゼロのときは、ストリングはヌル・ストリングになります。

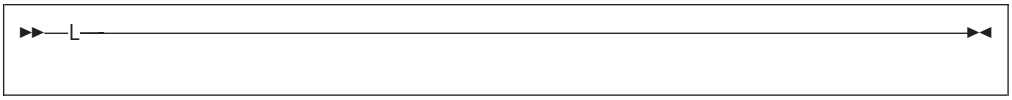
出力の場合は、データ・リスト項目が *field-width* に合わせて右側で切り捨てられたか、(埋め込み用グラフィックで) 拡張されたあと、データ・ストリーム中に書き出されます。ストリングを囲むための引用符が挿入されることはなく、識別用接尾部 *G* が挿入されることもありません。 *field-width* がゼロのときは、データ・ストリーム中にはなにも書き出されません。 *field-width* を省略したときは、データ・リスト項目の漢字ストリングの長さに等しい幅が想定されます。

次の例では、ファイル OUT に GRAPHIC オプションが指定してあると、 6 バイトが伝送されます。

```
declare A graphic(3);
put file(Out) edit (A) (G(3));
```

L フォーマット項目

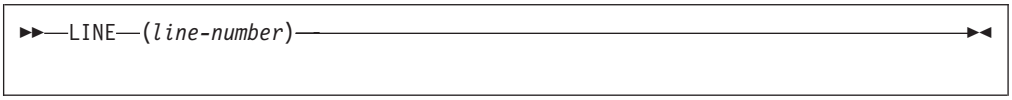
入力の場合は、L は、行の終わりまでのすべてのデータがデータ項目に割り当てられるものであることを表します。



出力の場合は、L は、データ項目 (必要に応じて、右側をブランクで埋め込まれている) が出力行の残りを埋めるものであることを表します。

LINE フォーマット項目

LINE フォーマット項目は、PRINT ファイルの現行ページのどの行に次のデータ・リスト項目を印刷するかを指定します。または ENDPAGE 条件を引き起こします。



line-number
式で表すことができます。この式は、このフォーマット項目が使用されるたびに計算され、整数値 (負であってはならない) に変換されます。

必要に応じて、ブランク行が挿入されます。

指定した *line-number* (行番号) が現在行の行番号以下である場合や、指定した行が OPEN ステートメントの PAGESIZE オプションでセットされた限界 (またはデフォルトの限界) を超えている場合は、ENDPAGE 条件が起こります。ただし、指定した *line-number* が現在行の行番号に等しく、しかも 1 桁目の文字がまだ伝送されていない場合は例外であり、そのときは、 SKIP(0) 項目を指定した場合と同じ結果になります (つまり、行送りせずに復帰します)。

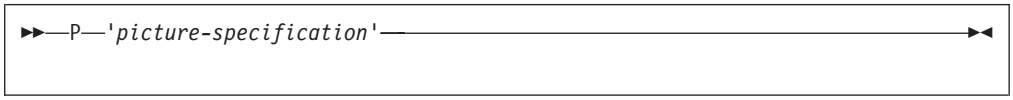
line-number がゼロのときは、1 と見なされます。

P フォーマット項目

ピクチャー (P) フォーマット項目は、実数の数字の値や文字の値の文字表記を記述するものです。

入力の場合は、P フォーマット項目のピクチャー指定によって、データ・ストリーム中にあるデータ項目のフォーマットを記述したり、数字指定の場合は、項目の算術値がどのように解釈されるかを記述したりします。ここに指定した文字がストリーム中にないときは、CONVERSION 条件が起こります。

出力の場合は、データ・リスト内の対応するエレメントの値が、ピクチャー指定で指定したフォーマットに変換されてから、データ・ストリーム中に書き出されます。



picture-specification
341 ページの『第 15 章 ピクチャー指定文字』で詳しく説明しています。

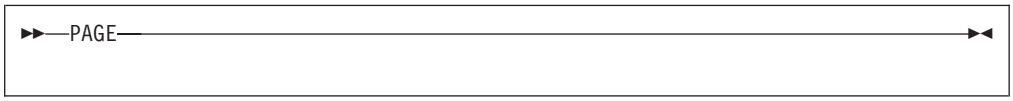
以下に例を示します。

```
get edit (Name, Total) (P'AAAAA',P'9999');
```

このステートメントが実行されるとき、ファイル SYSIN が入力ファイルであると見なされます。 SYSIN から入力された最初の 5 文字は、英字またはブランクでなければならず、それらは Name に割り当てられます。次の 4 文字は数字でなければならず、それらは Total に割り当てられます。

PAGE フォーマット項目

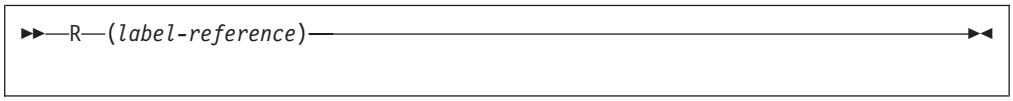
PAGE フォーマット項目は、新しいページを設定することを指定します。これは、PRINT ファイルの場合にのみ使用できます。



新しいページを開始すると、ファイルは次のページの 1 行目に位置づけられます。

R フォーマット項目

リモート (R) フォーマット項目は、FORMAT ステートメント内のフォーマット・リストを使用することを指定します (321 ページの『FORMAT ステートメント』に説明があります)。



label-reference
FORMAT ステートメントのラベル定数を値として持つラベル参照です。

R フォーマット項目とそこで指定した FORMAT ステートメントは、同一ブロックの内部になければならず、両方ともそのブロックの同一の呼び出し内にある必要があります。

リモート FORMAT ステートメントでは、label-reference (ラベル参照) としてそれ自体を参照する R フォーマット項目を書くことはできません。また、別のリモート FORMAT ステートメントを参照することによって、結果的にもとの FORMAT ステートメントを参照することもできません。

GET ステートメントまたは PUT ステートメントに対して割り込み可能な条件は、参照されるリモート FORMAT ステートメントに対しても割り込み可能になっていなければなりません。

ON ユニットが 1 つの GET ステートメントまたは PUT ステートメントだけで構成されているときは、そのステートメントが 1 つのブロックになるので、そのステートメント内にリモート・フォーマット項目を書くことはできません。

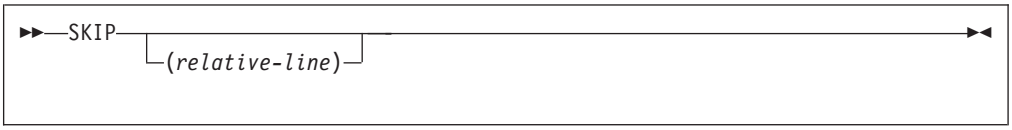
例

```
declare Switch label;
get file(In) list(Code);
if Code = 1 then
  Switch = L1;
else
  Switch = L2;
get file(In) edit (W,X,Y,Z)
    (R(Switch));
L1:  format (4 F(8,3));
L2:  format (4 E(12,6));
```

Switch は、ラベル変数として宣言されています。 2 番目の GET ステートメントは、2 つの FORMAT ステートメントのどちらでも使うことができます。

SKIP フォーマット項目

SKIP フォーマット項目は、新しい行を現在行として定義することを指定します。



relative-line
式を指定します。この式は、このフォーマット項目が使用されるたびに計算され、整数値 *n* に変換されます。変換された値は、負の数であってはならず、また、32,768 未満でなければなりません。 PRINT ファイル以外のファイルのときは、この値はゼロより大きくなければなりません。この値がゼロであるか、または省略されていると、1 と見なされます。

新しい行は、現在行から *n* 行だけあとにある行になります。

n が 1 より大きい場合は、1 行または複数の行が入力時に無視されます。出力時には、1 行または複数のブランク行が挿入されます。

値 *n* は、PRINT ファイルに限り、ゼロであることがあります。この場合には、位置付けは、現在行の先頭になります。前に印刷された文字に上重ね印刷されることがあります。

PRINT ファイルの場合、指定した *relative-line* が OPEN ステートメントの PAGESIZE オプションでセットされた限界 (またはデフォルトの限界) を超えていると、ENDPAGE 条件が起こります。

SKIP フォーマット

SKIP フォーマット項目が、ファイルのオープン後に最初に実行される項目である場合は、最初のページの *n* 行目から出力が開始されます。 *n* がゼロまたは 1 である場合、最初のページの最初の行から出力が開始されます。

以下に例を示します。

```
get file(In) edit(Man,Overtime)
     (skip(1), A(6), COL(60), F(4,2));
```

このステートメントは、ファイル In に関連付けられているデータ・セットを新しい行に位置づけます。その行の最初の 6 文字が Man に割り当てられ、60 桁目から始まる 4 文字が Overtime に割り当てられます。

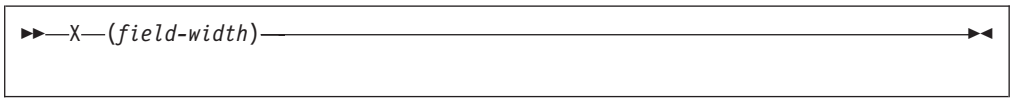
V フォーマット項目

入力の場合は、V は、行の終わりまでのすべてのデータがデータ項目に割り当てられるものであることを表します。ただし、V フォーマット項目で読み取られた文字は、フラッシュされず、表示されるだけです。これらの文字は、他のフォーマット項目で読み取られた場合にのみフラッシュされます。

V フォーマット項目は、出力では無効です。

X フォーマット項目

間隔 (X) フォーマット項目は、データ・ストリーム中のあるデータ値と次のデータ値の間隔を指定します。



field-width

式を指定します。この式は、このフォーマット項目が使用されるたびに計算され、整数値 (負であってはならない) に変換されます。この整数値は、データ・ストリーム中の現在位置から次のフィールドまでの間にある桁の桁数を表します。

入力の場合は、指定した桁数だけデータ・ストリーム中で水平送りされ、それらの桁はプログラムに伝送されません。

以下に例を示します。

```
get edit (Number, Rebate)
       (A(5), X(5), A(5));
```

入力ファイル SYSIN からの次の 15 文字は次のように処理されます。最初の 5 文字は Number に割り当てられ、次の 5 文字は無視され、残りの 5 文字は Rebate に割り当てられます。

出力の場合は、指定した個数のブランク文字がストリーム中に挿入されます。

以下に例を示します。

```
put file(Out) edit (Part, Count) (A(4), X(2), F(5));
```

Part の値を表す 4 文字、2 つのブランク文字、Count の固定小数点値を表す最後の 5 文字は、Out という名前のファイルに書き出されます。

第 15 章 ピクチャー指定文字

ピクチャー反復因数 341
文字データ用のピクチャー文字 342
数字データ用のピクチャー文字 343
数字と小数点 344
ゼロ消去 346
挿入文字 347
通貨記号の定義. 349
符号と通貨記号. 350
貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字. 353
指数文字 354
スケール因数 355

ピクチャー指定とは、一連のピクチャー文字を一重または二重引用符で囲んだものです。この文字は、文字データ項目または数字データ項目の各位置の内容、および出力の内容を記述します。この指定は、次の 2 つの方法で行われます。

- 宣言内の PICTURE 属性の一部として行われます。
- 編集ディレクティブ入出力の P フォーマット項目 (335 ページの『P フォーマット項目』に説明があります) の一部として行われます。

ピクチャー指定は、文字データ項目または数字データ項目を記述します。 A または X のピクチャー文字は、文字ピクチャー指定としてピクチャー指定を定義します。それ以外の場合は、数字ピクチャー指定です。

文字ピクチャー項目 とは、英字、10 進数、ブランク、通貨記号、および句読点文字で形成することができる項目です。

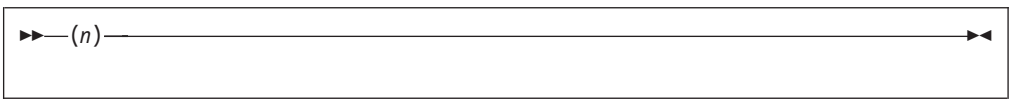
数字ピクチャー項目 とは、10 進数、オプションの小数点、オプションの文字 E、およびオプションの 1 つまたは 2 つの正符号 (+) と負符号 (-) だけで形成することができる項目です。 算術データには普通に対応づけられるその他の文字 (通貨記号など) も指定できますが、そのような文字は数字変数の算術値の一部ではありません。ただし、そのような文字は、数字と一緒にストレージに保管され、その変数の文字値の一部となります。

値がピクチャー変数に割り当てられるときや、P フォーマット項目によって印刷されるときに、ピクチャー指定が異なれば値の表記がどのように異なるかを、この章に記載されている図で示します。それらの図には、データの元の値、そのデータが割り当てられる (または書き出される) 際に入っていた変数の属性、ピクチャー指定、および数字変数またはピクチャー文字変数の文字値が示されています。

2 種類のピクチャー指定の概念については、以降で個別に説明します。

ピクチャー反復因数

ピクチャー反復因数は、そのすぐあとにあるピクチャー文字を繰り返す回数を指定します。



n 整数です。括弧内にブランクがあってはなりません。 n が 0 のときは、ピクチャー文字は無視されます。

たとえば、次の 2 つのピクチャー指定は、同じことを記述しています。

```
'999V99'
'(3)9V(2)9'
```

文字データ用のピクチャー文字

文字ピクチャー指定は、無変更の文字データ項目を記述します。これによって、使用可能な文字セット全体の特定のサブセットに属する文字しかデータ項目のその文字位置に入れられないことを指定できます。データは、英字、10 進数、およびブランクから形成できます。

文字ピクチャー指定内に記入できる文字は、X と A と 9 だけです。これらの各文字は、文字値の中の 1 文字桁を表し、そこには次のような文字を入れることができます。

X 1 バイトをなす 8 ビットで表されており、ビットを組み合わせてできる 256 文字のうちの任意の文字。

A 任意の英字、または特別言語 (#、@、$) 文字、またはブランク。

9 任意の数字またはブランク (9 ピクチャー指定文字は、文字データの場合にのみブランクを使用することができます)。

文字値がピクチャー文字データ項目に割り当てられる (または伝送される) ときには、1 文字ずつ、対応するピクチャー指定文字で指定されたとおりであるかが検査されます。文字データがその位置の指定と一致しない場合、先行ゼロの除外が、CONVERSION 条件が起こります。 (ただし、レコード単位の伝送を行うか、別名を使用するかして値を変更した場合は、検査は行われません。) 以下に例を示します。

```
declare Part# picture 'AAA99X';
put edit (Part#) (P'AAA99X');
```

次の値は、Part# の値として有効です。

```
'ABC12M'
'bbb09/'
'XYZb13'
```

次の値は、Part# の値としては有効ではありません (下線で示した文字が無効文字です)。

```
'AB123M'
'ABC̲1/2'
'Mb#A̲5̲;'
```

表 37 に、文字ピクチャー指定の例を示します。

表 37. 文字ピクチャー指定の例

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
CHARACTER(5)	'9B/2L'	XXXXX	9B/2L
CHARACTER(5)	'9B/2L'	XXX	9B/
CHARACTER(5)	'9B/2L'	XXXXXXX	9B/2Lbb

表 37. 文字ピクチャー指定の例 (続き)

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
CHARACTER(5)	'ABCDE'	AAAAA	ABCDE
CHARACTER(5)	'ABCDE'	AAAAAA	ABCDEb
CHARACTER(5)	'ABCDE'	AAA	ABC
CHARACTER(5)	'12/34'	99X99	12/34
CHARACTER(5)	'L26.7'	A99X9	L26.7

数字データ用のピクチャー文字

数字データは数値を表します。ピクチャー指定には、文字データ・ピクチャー文字 X または A を含むことはできません。数字データ用のピクチャー文字は、データの編集を指定することもできます。

数字変数は、その使用方法に応じて、2 種類の値を持つと考えられます。変数のタイプは次のとおりです。

算術 算術値は、データ項目の 10 進数、仮想小数点の位置、符号 (省略可能)、および指数 (符号は省略可能) かスケール因数で表される値です。数字変数の算術値が使用されるのは、次の場合です。

- 数字変数が、コード化算術値またはビット値を求める式の中で使われているとき (これには、¬、&、|、および比較演算子を用いる式も含まれます。文字ストリングを持つ式でも数字変数の算術値を使用します。)
- 数字変数がコード化算術変数、数字変数、またはビット変数に割り当てられるとき
- 編集ディレクティブの入出力において、C、E、F、B、および P (数字) フォーマット項目とともに使用されるとき

数字変数の算術値が内部コード化算術表記に変換されます。

文字値 文字値は、データ項目の 10 進数と、ピクチャー指定内に書かれているすべての編集文字と挿入文字とで表される値です。ただし、ピクチャー文字 V、K、または F で指定される仮想小数点の位置は文字値に含まれません。数字変数の文字値が使用されるのは、以下の場合です。

- 数字変数が文字式の中で使用されているとき
- 文字変数に割り当てられるとき
- リスト・ディレクティブ出力またはデータ・ディレクティブ出力によって、そのデータが印刷されるとき
- 数字変数で定義された、または数字変数にもとづいた文字変数が参照されたとき
- 編集ディレクティブ出力で A または P (文字) フォーマット項目によって、数字変数が印刷されるとき

データ変換は不要です。

数字データは、10 進数、小数点 (省略可能)、文字 E (省略可能)、および 1 つまたは 2 つの正符号 (+) または負符号 (-) だけから形成することができます。 算術デ

ータには普通に関連づけられるその他の文字 (通貨記号など) も指定できますが、そのような文字は数字変数の算術値の一部ではありません。ただし、そのような文字は、数字と一緒にストレージに保管され、その変数の文字値の一部となります。

数字指定は、1 つまたは複数のフィールドから構成され、各フィールドが固定小数点を記述します。 浮動小数点指定は、2 つのフィールド (小数部を記述するフィールドと指数を記述するフィールド) からなります。第 1 フィールドは、V ピクチャー指定文字を挿入することによって、サブフィールドに分けることができます。 V の前のデータ (ある場合) と V のあとのデータ (ある場合) が数字指定のサブフィールドです。

数字データのピクチャー指定では、各フィールドに、数字桁を指定するピクチャー文字が最低 1 つはなければなりません。ただし、このピクチャー文字は数字 9 である必要はなく、他のピクチャー文字、たとえばゼロ抑制文字 (Z または *) などで数字桁を指定してもかまいません。

注: K、V、F 以外のすべての文字は、文字表記の中に文字が存在することを表します。

数字指定用のピクチャー文字については、次の節を参照してください。

- ピクチャー文字 9 および V で指定されたデータについては、『数字と小数点』を参照してください。
- ピクチャー文字 Z およびアスタリスク (*) で指定されたピクチャー・データについては、346 ページの『ゼロ消去』を参照してください。
- 挿入文字 (小数点、コンマ、スラッシュ、および B) の使用については、347 ページの『挿入文字』を参照してください。
- 小数点および挿入文字を V ピクチャー文字と使用する場合については、348 ページの『挿入文字と小数点文字』を参照してください。
- ユーザー独自の文字 (複数の場合もある) を通貨記号として定義する方法については 349 ページの『通貨記号の定義』を、符号および通貨記号の使用については350 ページの『符号と通貨記号』を参照してください。
- 貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字の機能に使用されるピクチャー文字 CR、DB、T、I、R、および Y については、353 ページの『貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字』を参照してください。
- 指数に使用されるピクチャー文字 K および E については、354 ページの『指数文字』を参照してください。
- スケール因数に使用されるピクチャー文字 F については、355 ページの『スケール因数』を参照してください。
- ピクチャー反復文字については、341 ページの『ピクチャー反復因数』を参照してください。

数字と小数点

固定小数点 10 進数値を表す数字指定では、ピクチャー文字 9 と V を使用します。

9 関連データ項目内のこの位置に 10 進数が入ることを指定します。 (数字データ

のピクチャー指定文字 9 は、文字データの場合は対応する文字がブランクであってはならないという点で文字データの指定とは異なります。)

n 個のピクチャー文字 9 からなるストリングは、その項目が長さ n の文字ストリング (変更不能) で、各文字は数字 (0 から 9) であることを表します。 以下に例を示します。

```
dcl digit picture'9',
    Count picture'999',
    XYZ picture '(10)9';
```

使用例は次のとおりです。

```
dcl 1 Record,
      2 Data char(72),
      2 Identification char(3),
      2 Sequence pic'99999';
dcl Count fixed dec(5);
⋮
Count=Count+1;
Sequence=Count;
write file(Output) from(Record);
```

V 関連データ項目内のこの位置に小数点があると見なすことを指定します。ただし、実際の小数点または 10 進コンマをこの位置に挿入することを指定するわけではありません。割り当てられる値の整数部分と小数部分は、スケール因数 F (±x) が指定されている場合はそれによって修正されてから、文字 V に位置合わせされます。したがって、割り当てられる値は左端と右端のどちらかで切り捨てられたり、数字ゼロで拡張されたりすることがあります。 (左端で有効数字が切り捨てられたときは、結果は未定義となり、可能であれば SIZE 条件が起こります。)

固定小数点 10 進数値のピクチャー指定 (または浮動小数点 10 進数値のピクチャー指定の第 1 フィールド) 内に文字 V がないときは、フィールド指定の最右端に V があると見なされます。したがって、割り当てられる値は必要に応じて切り捨てられ、整数になることがあります。

文字 V は、1 つのピクチャー指定に 1 回しか記入できません。

以下に例を示します。

```
dcl Value picture 'Z9V999';
Value = 12.345;
dcl Cvalue char(5);
Cvalue = Value;
```

Cvalue には、Value の値が割り当てられた後、'12345' が入っています。

表 38 に、数字と小数点文字の例を示します。

表 38. 数字と小数点文字の例

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FIXED(5)	12345	99999	12345
FIXED(5)	12345	99999V	12345
FIXED(5)	12345	999V99	未定義 (undefined)

表 38. 数字と小数点文字の例 (続き)

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FIXED(5)	12345	V99999	未定義 (undefined)
FIXED(7)	1234567	99999	未定義 (undefined)
FIXED(3)	123	99999	00123
FIXED(5,2)	123.45	999V99	12345
FIXED(7,2)	12345.67	9V9	未定義 (undefined)
FIXED(5,2)	123.45	99999	00123

注: 文字値が未定義のときは、SIZE 条件が起こります。

ゼロ消去

ピクチャー文字 Z とアスタリスク (*) は、文字値の条件付き数字桁を指定します。これを使用すれば、先行ゼロをアスタリスクまたはブランクで置き換えることができます。 先行ゼロとは、固定小数点数の左端の数字桁または浮動小数点数の 2 つの部分の左端にある数字桁に入っているゼロのうち、仮想小数点の左側にあり、しかもその前に 1 から 9 の数字がないようなゼロのことです。数字の左端にあるゼロ以外の桁、および数値の右側にあるすべての桁 (ゼロであってもゼロでなくてもかまわない) は、有効数字を表します。

Z 条件付き数字桁を指定するもので、データのこの位置に先行ゼロがあれば、それはブランクで置き換えられます。先行ゼロ以外の数字はそのまま変りません。ピクチャー文字 Z は、ピクチャー文字 * または浮動文字が記入されているのと同じフィールド内に記入することはできません。また、フィールド内で任意のピクチャー文字の右側に記入することもできません。

***** 条件付き数字桁を指定します。先行ゼロがアスタリスクで置き換えられる点を除けば、その使用方法はピクチャー文字 Z と同じです。ピクチャー文字 * は、ピクチャー文字 Z や浮動文字が記入されているのと同じフィールド内に記入することはできません。また、フィールド内で任意のピクチャー文字の右側に記入することもできません。

表 39 に、ゼロ抑制文字の例を示します。

表 39. ゼロ抑制文字の例

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FIXED(5)	12345	ZZZ99	12345
FIXED(5)	00100	ZZZ99	bb100
FIXED(5)	00100	ZZZZZ	bb100
FIXED(5)	00000	ZZZZZ	bbbbb
FIXED(5,2)	123.45	ZZZ99	bb123
FIXED(5,2)	001.23	ZZZV99	bb123
FIXED(5)	12345	ZZZV99	未定義 (undefined)
FIXED(5,2)	000.08	ZZZVZZ	bbb08
FIXED(5,2)	000.00	ZZZVZZ	bbbbb

表 39. ゼロ抑制文字の例 (続き)

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FIXED(5)	00100	*****	**100
FIXED(5)	00000	*****	*****
FIXED(5,2)	000.01	***V**	***01
FIXED(5,2)	95	$**9.99	$**0.95
FIXED(5,2)	12350	$**9.99	$123.50

注: 文字値が未定義のときは、SIZE 条件が起こります。

ピクチャー文字 Z またはアスタリスクをピクチャー文字 V の右側に記入するときは、その指定の小数部分の全数字桁と整数部分の全数字桁に、それぞれ Z またはアスタリスク・ピクチャー文字を記入しなければなりません。ピクチャー文字 V の右側にある全数字桁がゼロ消去ピクチャー文字であるとき、値の小数部分のゼロが抑制されるのは、小数部分のすべての桁にゼロが入っており、しかも整数部分のすべての桁が抑制されたときだけです。したがって、そのデータ項目の文字値はブランクまたはアスタリスクだけになります。小数部分に 1 つでも有効数字が含まれているときは、小数部分の数字はブランクやアスタリスクでは置き換えられません。

挿入文字

ピクチャー文字のコンマ (,)、小数点 (.)、スラッシュ (/)、およびブランク (B) を指定すると、数字データ内の対応する位置にその文字が挿入されます。 これらの文字は、数字桁や文字桁を示すものではなく、数字や文字の間に挿入されます。ただし、文字値の中では、いずれの文字も、その文字が消去されるか否かに関係なく 1 つの文字桁を表しています。コンマ、小数点、およびスラッシュは条件付き挿入文字であり、一連のゼロ抑制文字の間にあるときには消去されることもあります。ブランクは無条件挿入文字であり、対応する位置にブランクを常に挿入することを指定します。

挿入文字は、文字値にだけ適用され、そのデータ項目の算術値に関してはなにも指定しません。挿入文字を使用しても、固定小数点 10 進数のピクチャー指定内で小数点 (または 10 進コンマ) 位置合わせを指定したことにはなりませんし、挿入文字はデータ項目の算術値の一部ではありません。小数点位置合わせは、ピクチャー文字 V および F によって制御されます。

コンマ (,)、小数点 (.)、またはスラッシュ (/)

ゼロ消去が行われない限り、数字データ内の対応する位置にこれらの文字が挿入されます。ゼロ消去が行われたときは、下記の場合にのみこれらの文字が挿入されます。

- 挿入位置より左側に、抑制されていない数字がある場合。
- 挿入位置のすぐ左に V があり、データ項目の小数部分に有効数字がある場合。
- ピクチャー指定の先頭が挿入位置である場合。
- 挿入位置より前にあるのが、数字桁を指定しない文字だけである場合。

上記以外の場合にゼロ消去が行われるときは、コンマ、小数点、またはスラッシュ挿入文字は、その前にある文字と等しくゼロ抑制文字と見なされます。

B 数字データの文字値の対応する位置に、ブランク文字を挿入することを指定します。

挿入文字と小数点文字

小数点、コンマ、またはスラッシュを V と一緒に使用すれば、固定小数点数 (または浮動小数点数) の整数部分の終わりと小数部分の始まりを区切る位置に、小数点 (またはコンマ、スラッシュ) を挿入できます。この機能は、V を使用しても小数点の印刷ができないので、印刷時に必要になる場合があります。小数点は、V の直前または直後になければなりません。小数点が V の直前にある場合は、すべての小数桁が有効な場合でも、未消去の数字が V の左に現れる場合にのみ挿入されます。V のすぐあとにある小数点は、V の右側にあるすべての数字が消去されたときに消去されますが、小数桁に消去されていない数字がある (間にゼロがあってもよい) ときは挿入されます。

次に各国で使用されている 10 進数の規則の例を示します。

```
declare A picture 'Z,ZZZ,ZZZV.99',
        B picture 'Z.ZZZ.ZZZV,99',
        C picture 'ZBZZZBZZZV,99';
A,B,C = 1234;
A,B,C = 1234.00;
```

A、B、C はそれぞれ 9 桁の数を表し、 7 番目と 8 番目の数字の間に小数点または小数コンマが想定されています。小数点挿入文字によって指定された実際の小数点は、算術値の一部ではありませんが、文字値ではその一部です。 2 つの代入ステートメントは、次のように A、B、C に同じ文字値を割り当てます。

```
1,234.00     /* value of A */
1.234,00     /* value of B */
1 234,00     /* value of C */
```

次の例では、割り当て中に小数点の位置合わせが文字 V で行われます。 Rate が印刷される場合、'762.00' のように示されますが、その算術値は 7.6200 です。

```
declare Rate picture '9V99.99';
Rate = 7.62;
```

表 40 に、挿入文字の例を示します。

表 40. 挿入文字の例

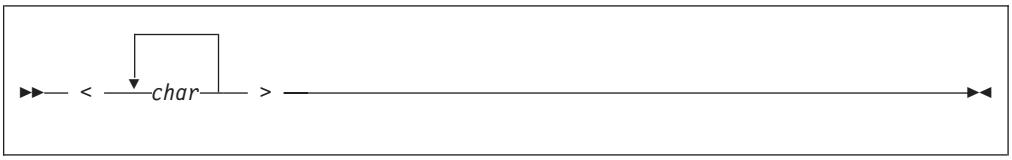
ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FIXED(4)	1234	9,999	1,234
FIXED(6,2)	1234.56	9,999V.99	1,234.56
FIXED(4,2)	12.34	ZZ.VZZ	12.34
FIXED(4,2)	00.03	ZZ.VZZ	bbb03
FIXED(4,2)	00.03	ZZV.ZZ	bb.03
FIXED(4,2)	12.34	ZZV.ZZ	12.34
FIXED(4,2)	00.00	ZZV.ZZ	bbbbb
FIXED(9,2)	1234567.89	9,999,999.V99	1,234,567.89
FIXED(7,2)	12345.67	**,999V.99	12,345.67
FIXED(7,2)	00123.45	**,999V.99	***123.45
FIXED(9,2)	1234567.89	9.999.999V,99	1.234.567,89
FIXED(6)	123456	99/99/99	12/34/56

表 40. 挿入文字の例 (続き)

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FIXED(6)	123456	99.9/99.9	12.3/45.6
FIXED(6)	001234	ZZ/ZZ/ZZ	bbb12/34
FIXED(6)	000012	ZZ/ZZ/ZZ	bbbbbb12
FIXED(6)	000000	ZZ/ZZ/ZZ	bbbbbbbb
FIXED(6)	000000	**/**/**	********
FIXED(6)	000000	**B**B**	**b**b**
FIXED(6)	123456	99B99B99	12b34b56
FIXED(3)	123	9BB9BB9	1bb2bb3
FIXED(2)	12	9BB/9BB	1bb/2bb

通貨記号の定義

通貨記号は、数字データの文字値を表示するピクチャー文字として使用することができます。この記号には、ドル符号 ($) またはユーザーが選択した任意の記号を使用できます。また、< と > の文字で囲まれた一連の文字も記号とすることができます。



< 通貨記号の開始を表します。この文字は、エスケープ文字としての役割を果たします。 < を文字として使用したい場合は、<< と指定しなければなりません。

char これは、通貨記号 (複数の場合もある) の一部となる任意の文字です。

> 通貨記号の終わりを表します。文字 > を使用する場合には、<> と指定する必要があります。

複数の > は、浮動ストリングを表します (351 ページで説明されています)。

一般的な挿入ストリングには、次のものがあります。

<DM> ドイツ・マルクを表します。

<Fr> フランス・フランを表します。

<K$> カリスタン・ドルを表します。

<Sur.f> スリナム・ギルダーを表します。

<$> ドル符号を表します。

文字 < または > がシーケンス内になければならない場合、新たに別の < をその文字の前に付けなければなりません。こうすることによって、< はエスケープ文字としての役目も果たします。

< > で囲まれたシーケンス全体で 1 つの「記号」を表し、それによってある数字の文字値を表します。記号を浮動ピクチャー文字として表す必要がある場合は、「< >」のあとに > を指定して、各オカレンスを表してください。

以下に例を示します。

Pic '<DM>>>.>>9,V99'
10 文字の数字ピクチャーを表し、割り当てのあとは 11 文字になります。

Pic '<Sur.f>999,V99'
7 文字の数字ピクチャーを表し、割り当てのあとは 11 文字になります。

Pic '<K$>>>,>>9.V99'
10 文字の数字ピクチャーを表し、割り当てのあとは 11 文字になります。

Pic '<$>>>,>>9.V99'
10 文字の数字ピクチャーを表し、割り当てのあとは 10 文字になります。

Pic '$$$,$$9.V99'
上のピクチャー指定と同じ値になります。

通貨記号の定義例をさらに挙げます。

```
dcl P pic'<DM>9.999,V99';
P = 1234.40;                          /* Yields 'DM1.234,40'   */


dcl P pic'<DM>9.999,V99';
P =   34.40;                          /* Yields 'DM   34,40'   */


dcl P pic'<DM>>.>>9,V99';
P = 1234.40;                          /* Yields 'DM1.234,40'   */


dcl P pic'<DM>>.>>9,V99';
P =   34.40;                          /* Yields '   DM34,40'   */


dcl P pic'9.999,V99<K$>';
P = 1234.40;                          /* Yields '1.234,40K$' */
```

この章では、*通貨記号* という用語と $ 記号は、ドル符号またはユーザー定義の任意の通貨記号を指します。

符号と通貨記号

ピクチャー文字 S、+、および - は、数字データ内の符号を指定します。ピクチャー文字 $ (通貨記号) は、数字データの文字値の中の通貨記号を指定します。各フィールドでは、1 種類の符号文字しか使用できません。

通貨記号
通貨記号を指定します。

次に例を示します。

```
dcl Price picture '$99V.99';
Price = 12.45;
```

Price の文字値は、'$12.45' です。 Price の算術値は 12.45 です。

文字を通貨記号として指定する場合の詳細については、 349 ページの『通貨記号の定義』を参照してください。

S データ値が >=0 の場合は正符号 (+) を指定します。それ以外の場合は、負符号 (-) を指定します。使用規則は、通貨記号の場合と同じです。

次の例を考えてみてください。

```
dcl Root picture 'S999';
```

値 50 は '+050' として、値 0 は '+000' として、値 -243 は '-243' としてそれぞれ保持されます。

+ データ値が >=0 の場合は正符号 (+) を指定します。それ以外の場合は、ブランクを指定します。使用規則は、通貨記号の場合と同じです。

- データ値が <0 の場合は負符号 (-) を指定し、それ以外の場合はブランクを指定します。使用規則は、通貨記号の場合と同じです。

符号および通貨記号は、固定的に使用することも浮動的に使用することもできます。

固定的使用

固定的に使用すると、符号、通貨記号、またはブランクが対応する位置に入ります。固定文字として使用される S、+、または - は、浮動小数点指定の小数部フィールド内や指数フィールド内の全数字桁の右側または左側に記入することができ、固定小数点の全数字桁の右側または左側に記入することができます。

浮動的使用

浮動的に使用すると、先行ゼロが消去されます。 この場合、ピクチャー文字と対応した右端の抑制位置に符号、ブランク、または通貨記号が含まれます (すべての数字桁が浮動文字で占められ、データ項目の値がゼロであり、浮動文字が挿入されない場合は除きます)。

浮動文字であることを示すには、ピクチャー・フィールド内にその文字を複数記入します。浮動文字が浮動する範囲内のすべての数字桁に、その浮動文字を記入しなければなりません。浮動文字を記入するときは、同じ浮動文字を連続して記入しなければなりませんが、任意で、V およびいずれか 1 つの挿入文字 (コンマ、小数点、スラッシュ、または B) を使用することができます。浮動ストリングの途中またはすぐ後にある挿入文字のスラッシュ、コンマ、または小数点は、浮動ストリングの一部と見なされます。文字 B が記入されている位置には常に、ブランクが挿入されます。データ項目の算術値がゼロの場合以外は、V は浮動ストリングを終了します。その場合、V は無視されます。ピクチャー指定の 1 つのフィールドにつき、浮動ストリングを 1 つだけ記入することができます。浮動ストリングの前に数字桁があってはならず、ピクチャー文字 * や Z が記入されているフィールド内に浮動ストリングがあってはなりません。

浮動文字のストリング内にスラッシュ、コンマ、または小数点がある場合、データ内の対応する桁には下記のいずれかの文字が入ります。

- その位置の左側に有効数字があるときは、スラッシュ、コンマ、または小数点が入ります。

- その位置のすぐ右側に、フィールドの最左端の有効数字があるときは、浮動記号が入ります。
- その位置より複数桁右側に、フィールドの最左端の有効数字があるときは、ブランクが入ります。

浮動ストリングに *n* 個の浮動文字が含まれていると、そのストリングは、*n*-1 個の条件付き数字桁に対応させられます。最左端の浮動文字に関連付けられる桁には、その浮動文字かブランクしか入れてはならず、数字を入れることはできません。 1 つのフィールド内で 2 つの異なるピクチャー文字を浮動的に使用することはできません。

浮動ストリング内に V がある場合、V の前にある部分が 1 つのサブフィールドになり、V のあとにあるサブフィールドの全数字桁もまた、第 2 サブフィールドとなる浮動ストリングの一部でなければなりません。

V のあとのすべての数字桁に浮動文字を記入した場合、そのサブフィールドでゼロ消去が行われるのは、整数桁の数字も小数桁の数字もすべてゼロのときだけです。このとき、編集されたデータ項目はすべてブランクになります (ただし、フィールドの先頭に挿入文字がある場合、その挿入文字は例外です)。小数桁にゼロ以外の数字があると、小数桁全体がゼロ消去されずに残っています。

ピクチャーへの割り当ての途中または割り当て前に、10 進数の小数桁の数字が切り捨てられ、その結果、ゼロの値になった場合は、切り捨てが行われる前の 10 進数の値に対応する符号がピクチャー内に挿入されます。したがって、ピクチャー内の符号は、10 進数値がどのように計算されたかに左右されます。

352 ページの表 41 に、符号と通貨記号の例を示します。

表 41. 符号と通貨記号の例

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FIXED(5,2)	123.45	$999V.99	$123.45
FIXED(5,2)	012.00	99$	12$
FIXED(5,2)	001.23	$ZZZV.99	$bb1.23
FIXED(5,2)	000.00	$ZZZV.ZZ	bbbbbbb
FIXED(1)	0	$$$.$$	bbbbbb
FIXED(5,2)	123.45	$$$9V.99	$123.45
FIXED(5,2)	001.23	$$$9V.99	bb$1.23
FIXED(2)	12	$$$,999	bbb$012
FIXED(4)	1234	$$$,999	b$1,234
FIXED(5,2)	2.45	SZZZV.99	+bb2.45
FIXED(5)	214	SS,SS9	bb+214
FIXED(5)	-4	SS,SS9	bbbb-4
FIXED(5,2)	-123.45	+999V.99	b123.45
FIXED(5,2)	-123.45	-999V.99	-123.45
FIXED(5,2)	123.45	999V.99S	123.45+
FIXED(5,2)	001.23	++B+9V.99	bbb+1.23
FIXED(5,2)	001.23	- - -9V.99	bbb1.23
FIXED(5,2)	-001.23	SSS9V.99	bb-1.23

貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字

同じフィールド内で、ピクチャー文字 CR、DB、T、I、および R をほかの符号文字と一緒に使用することはできません。

貸方と借方

CR (貸方) と DB (借方) は、実数の数字データ項目の符号を指定します。

CR データ値が <0 の場合は、対応する位置に CR という文字を入れることを指定します。それ以外の場合は、その位置に 2 つのブランクが入ります。CR を記入できる位置は、フィールドの全数字桁の右側だけです。

DB データ値が <0 の場合は、対応する位置に DB という文字を入れることを指定します。それ以外の場合は、その位置に 2 つのブランクが入ります。DB を記入できる位置は、フィールドの全数字桁の右側だけです。

オーバーパンチ

任意のピクチャー文字 T、I、または R (オーバーパンチ文字として知られる) は、1 つの文字が対応する数字およびデータ項目の符号を表すことを指定します。浮動小数点指定は、2 つのフィールド (小数部を記述するフィールドと指数を記述するフィールド) からなります。オーバーパンチ文字は、フィールド内の任意の数字桁について指定することができます。

T、I、および R ピクチャー文字は、入力文字が解釈される方法を指定します。表 42 に示します。

表 42. T、I、および R ピクチャー文字の解釈

T または I	T または R	数字
+ 付きの数字	- 付きの数字	
文字	文字	
{	}	0
A	J	1
B	K	2
C	L	3
D	M	4
E	N	5
F	O	6
G	P	7
H	Q	8
I	R	9

T、I、および R は以下の値を指定します。

T 入力時には、T は、文字 { から I と数字 0 から 9 が正の値を表し、文字 } から R が負の値を表すことを指定します。

出力時には、T は、入力データが正の値を表す場合には、対応する位置に文字 { から I の 1 つを含み、入力データが負の値を表す場合には文字 } から R の 1 つを含むことを指定します。 T は '9' ピクチャー指定文字が生じるすべての場所に記入することができます。 以下に例を示します。

```
dcl Credit picture 'ZZV9T';
```

文字表示は 4 文字です。+21.05 は '210E' として保持されます。 -0.07 は 'bb0P' として保持されます。

I 入力時には、I は、文字 { から I と数字 0 から 9 が正の値を表すことを指定します。

出力時には、I は、入力データが正の値を表す場合には、対応する位置に文字 { から I の 1 つを含み、入力データが負の値を表す場合には数字 0 から 9 の 1 つを含むことを指定します。

R 入力時には、R は、文字 } から R が負の値を表し、数字 0 から 9 が正の値を表すことを指定します。

出力時には、R は、入力データが負の値を表す場合には、対応する位置に文字 } から R の 1 つを含み、入力データが正の値を表す場合には数字 0 から 9 の 1 つを含むことを指定します。以下に、例を示します。

```
dcl X fixed decimal(3);
get edit (x) (P'R99');
```

上記の指定は、入力ストリームの次の 3 つの位置内で、 '132' を見付けると X を 132 に設定し、 'J32' を見付けると X を -132 に設定します。

ゼロ置き換え文字

Y 指定された数字桁にあるゼロを、無条件にブランク文字で置き換えることを指定します。

表 43 に、貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字の例を示します。

表 43. 貸方記号、借方記号、オーバーパンチ、およびゼロ置き換え文字の例

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FIXED(3)	-123	$Z.99CR	$1.23CR
FIXED(4,2)	12.34	$ZZV.99CR	$12.34bb
FIXED(4,2)	-12.34	$ZZV.99DB	$12.34DB
FIXED(4,2)	12.34	$ZZV.99DB	$12.34bb
FIXED(4)	1021	999I	102A
FIXED(4)	-1021	Z99R	102J
FIXED(4)	1021	99T9	10B1
FIXED(5)	00100	YYYYY	bb1bb
FIXED(5)	10203	9Y9Y9	1b2b3
FIXED(5,2)	000.04	YYYVY9	bbbb4

指数文字

ピクチャー文字 K と E は、浮動小数点 10 進数を記述する数字指定の指数フィールドを区切ります。指数フィールドは、浮動小数点の数字ピクチャー指定の最後のフィールドです。ピクチャー文字 K と E を、同じピクチャー指定内に記入することはできません。

K 対応する桁の右側が指数フィールドであることを指定します。この文字は、数字データ項目内の文字を指定するものではありません。

E 対応する桁に、指数フィールドの始まりを示す文字 E が入ることを指定します。

文字値の中では指数の値が調整されて、第 1 フィールド (小数部) の最初の有効数字が、ピクチャー指定内の最初の数字指定子 (それがゼロ抑制文字であっても) に対応する桁に入るようにされます。

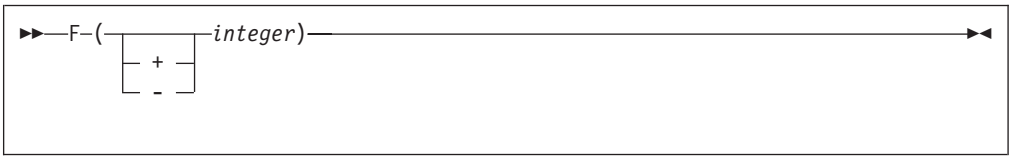
表 44 に、指数文字の例を示します。

表 44. 指数文字の例

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FLOAT(5)	.12345E06	V.99999E99	.12345E06
FLOAT(5)	.12345E-06	V.99999ES99	.12345E-06
FLOAT(5)	.12345E+06	V.99999KS99	.12345+06
FLOAT(5)	-123.45E+12	S999V.99ES99	-123.45E+12
FLOAT(5)	001.23E-01	SSS9.V99ESS9	+123.00Eb-3
FLOAT(5)	001.23E+04	ZZZV.99KS99	123.00+02
FLOAT(5)	001.23E+04	SZ99V.99ES99	+123.00E+02
FLOAT(5)	001.23E+04	SSSSV.99E-99	+123.00Eb02

スケール因数

ピクチャー文字 F は、固定小数点 10 進数のピクチャー・スケール因数を指定します。この文字は、ピクチャー指定の右端に 1 回だけ記入することができます。



F ピクチャー・スケール因数を指定します。ピクチャー・スケール因数は、変数の算術値における小数点の位置が、文字値における仮想小数点の位置からどれだけ右に (ピクチャー・スケール因数が正の場合) または左に (負の場合) あるかを表します。

ピクチャー文字 V のあとにある数字の個数から、F で指定した整数を引いた値が、-128 から 127 の範囲内になければなりません。

表 45 に、ピクチャー・スケール因数文字の例を示します。

表 45. スケール因数文字の例

ソース属性	ソース・データ (定数フォーマット)	ピクチャー指定	文字値
FIXED(4,0)	1200	99F(2)	12
FIXED(7,0)	-1234500	S999V99F(4)	-12345
FIXED(5,5)	.00012	99F(-5)	12
FIXED(6,6)	.012345	999V99F(-4)	12345

第 16 章 条件処理

条件接頭語 357
条件接頭語の有効範囲 359
OPTIMIZATION によって起こる条件 359
ON ユニット 360
ON ステートメント 360
ヌル ON ユニット 361
ON ユニットの有効範囲. 361
動的に派生した ON ユニット 362
ファイル変数のための ON ユニット 362
REVERT ステートメント 363
SIGNAL ステートメント 364
RESIGNAL ステートメント 364
複数条件 365
CONDITION 属性 365

PL/I プログラムの実行中には、さまざまなイベントが発生します。ユーザーは、それらのイベントに対してテスト、応答、または回復アクションを行うことができます。これらのイベントは、*条件* と呼ばれ、検出されたときに*発生* します。条件には、起こることが予期できないエラー (オーバーフローや入出力伝送エラーなど) もあれば、起こると分かっているエラー (入力ファイルの終わりなど) もあります。SIGNAL ステートメントを使用して条件をプログラム中に直接発生させることができます (この方法はテスト中に非常に役立ちます)。

条件をアプリケーションで制御できるようにするには、その条件を*割り込み可能* にします。また、割り込み可能な条件が起こるとアクションが*確立* されます。 条件が割り込み禁止になっていると、その条件が起こってもなんのアクションも実行されません。プログラムはそのイベントが起こったことを認知しません。確立されるアクションは、ON ユニットであることもあれば、その条件に対して定義された暗黙アクションである場合もあります。

呼び出された ON ユニットは、パラメーターのないプロシージャーと見なされます。 ON ユニットを使用する場合には、条件を引き起こした原因を調べる際に役立つ組み込み関数と疑似変数が提供されます。疑似変数は、エラーを訂正して回復する際によく使用されます。 395 ページの『第 19 章 組み込み関数、疑似変数、およびサブルーチン』に、組み込み関数と疑似関数を示しています。

多くの条件では、暗黙アクションとして ERROR 条件が発生します。 したがって、ERROR 条件は、それぞれの条件を個別に検査するのではなく、多数の異なる条件を検査するのに使用される共通の条件です。この点で ONCODE 組み込み関数は特に便利なもので、これを使用すればそれらの条件を起こしている状況を区別することができます。 検出される条件やエラーに対応するコードについては、「メッセージおよびコード」にリストが記載されています。

条件接頭語

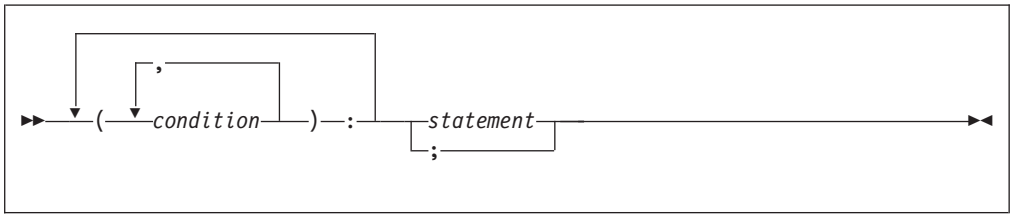
条件によっては、その条件を割り込み可能にするのか、割り込み禁止にするのかを指定することができます。条件が割り込み可能な場合、コンパイラーは、条件の検出に必要な追加のコードを生成します。条件が割り込み禁止の場合は、追加のコードは生成されません。

条件を割り込み禁止にするということは、つまり条件が起こらないようにするということです。条件が起こった場合は、プログラムにエラーがあるということを意味します。

たとえば、SUBSCRIPTRANGE 条件が割り込み可能な場合、コンパイラーは追加のコードを生成して、あらゆる配列指標が必ずその配列の境界内に収まるようにします。 SUBSCRIPTRANGE 条件が割り込み禁止だと、追加コードは生成されないため、無効な配列指標が使用されて予期不能な結果となります。

条件がハードウェアによって検出されると、条件の割り込み禁止は無効になります。

条件接頭語を使用すれば、上記のような条件について、割り込み可能にするか、割り込み禁止にするかを指定できます。



condition
条件の中には、常に割り込み可能になっており、割り込み禁止にすることができないものもあります。さらに、割り込み禁止にするのでない限り、割り込み可能になっているものもあります。逆に、割り込み可能にするのでない限り、割り込み禁止になっているものもあります。条件は、367 ページの『第 17 章 条件』にリストされています。

statement
条件接頭語は、ステートメント DECLARE、DEFAULT、FORMAT、OTHERWISE、END、ELSE、ENTRY、および % ステートメントでは使用することができません。条件接頭語の有効範囲については、359 ページの『条件接頭語の有効範囲』を参照してください。

次の例では、`(size) :` が条件接頭語です。条件付き接頭部は、対応する条件が接頭部の有効範囲内で割り込み可能であることを表します。

```
(size): L1: X=(I**N) / (M+L);
```

条件は、条件名を指定する条件接頭語を使用して、割り込み可能にすることができます。また、条件名を指定する条件接頭語の前にブランクを入れずに NO を付ければ、条件を割り込み禁止にすることができます。条件のタイプと状況を表 46 に示します。

表 46. 条件のクラスと状況

クラスと条件	状況
計算 (データの処理、式の計算、および計算)	
CONVERSION	デフォルトにより割り込み可能
FIXEDOVERFLOW	デフォルトにより割り込み可能
INVALIDOP	デフォルトにより割り込み可能
OVERFLOW	デフォルトにより割り込み可能
UNDERFLOW	常に割り込み可能
ZERODIVIDE	デフォルトにより割り込み可能
入出力	
ENDFILE	常に割り込み可能

表 46. 条件のクラスと状況 (続き)

クラスと条件	状況
ENDPAGE	常に割り込み可能
KEY	常に割り込み可能
NAME	常に割り込み可能
RECORD	常に割り込み可能
TRANSMIT	常に割り込み可能
UNDEFINEDFILE	常に割り込み可能
プログラム・チェックアウト (プログラムの開発、デバッグに有効)	
SIZE	デフォルトにより割り込み禁止
STRINGRANGE	デフォルトにより割り込み禁止
STRINGSIZE	デフォルトにより割り込み禁止
SUBSCRIPTRANGE	デフォルトにより割り込み禁止
その他	
ANYCONDITION	常に割り込み可能
AREA	常に割り込み可能
ATTENTION	常に割り込み可能
CONDITION	常に割り込み可能
ERROR	常に割り込み可能
FINISH	常に割り込み可能
STORAGE	常に割り込み可能

条件を割り込み可能にしたり、割り込み禁止にしたりすることがパフォーマンスに与える影響については、「*プログラミング・ガイド*」を参照してください。

条件接頭語の有効範囲

条件接頭語の有効範囲 (つまり、その接頭部が適用されるプログラムの部分) は、その接頭部が付けられているステートメントまたはブロックです。このステートメントの実行時に呼び出される可能性のあるプロシージャーや ON ユニットには、その接頭部は必ずしも適用されません。

ステートメント PACKAGE、PROCEDURE または BEGIN に付けられた条件接頭語は、対応する END ステートメントまでのすべてのステートメント (END も含めて) に適用されます。これには、そのブロック内にネストされている他の PROCEDURE ステートメントや BEGIN ステートメントも含まれます。

あるブロック内で条件の状況を再定義するには、そのブロック内のステートメント (PROCEDURE ステートメントや BEGIN ステートメントを含む) に接頭部を付けます (このようにすれば、ネストされたブロック内で条件の割り込み可能または割り込み禁止を再定義することができます)。そのような再定義は、その接頭部が付けられているステートメントを実行する場合にのみ適用されます。ネストされた PROCEDURE ステートメントまたは BEGIN ステートメントの場合は、そのステートメントが定義されているブロックとそのブロックに含まれている他のブロック (ある場合) にのみ適用されます。

OPTIMIZATION によって起こる条件

OPTIMIZATION が有効である場合、ある同じ式について複数回出てくる条件は、 1 回しか起こらない可能性があります。次の例では、IX の条件 SUBSCRIPTRANGE は、1 回しか起こりません。

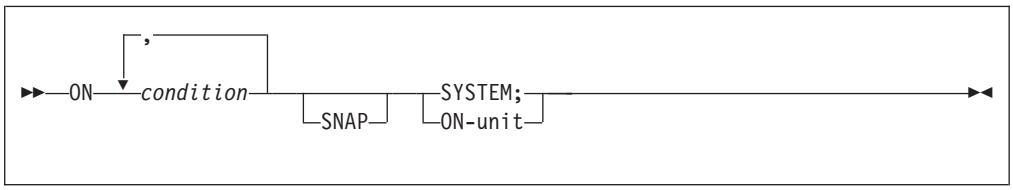
```
call P (55);
(subscriptrange): P: proc (IX);
  dcl (Ar, Br, Cr) (10);
  Ar(IX) = Ar(IX) + Br(IX);
  T = Cr(IX);
End P;
```

ON ユニット

どの条件についても暗黙アクションが定められています。割り込み可能な条件が起こると、この暗黙アクションが実行されます。ただし、その条件についての ON ユニットが有効になっているときは別です。暗黙アクションが ERROR を起こすもので、この条件に対して ON ユニットが確立されていない場合は、エラー条件が起こる前にメッセージが書き込まれます。

ON ステートメント

ON ステートメントは、あるアクションを設定するためのステートメントです。このアクションは、引き続いて設定された条件の有効範囲内で割り込み可能な条件が引き続いて起こると実行されます。



condition

367 ページの『第 17 章 条件』に説明されている条件、または CONDITION 属性によって定義されている条件のいずれかです。

SNAP

割り込み可能条件が起こったとき、その条件に関しての診断情報が印刷されることを指定します。 SNAP オプションのアクションが行われてから、ON ユニットのアクションが行われます。

SNAP と SYSTEM を指定すると、暗黙アクションが取られた直後に SNAP 情報が印刷されます。

SYSTEM

暗黙アクションを実行することを指定します。すべての条件に関して同じ暗黙アクションが取られるとは限りませんが、ほとんどの条件の場合、メッセージが印刷され、ERROR 条件が引き起こされます。条件ごとの暗黙アクションについては、 367 ページの『第 17 章 条件』で説明します。

ON-unit

指定された条件が起こり、しかも割り込み可能であるときに実行されるアクションを指定します。 ON-unit そのものの中にステートメント (複数も可) を書いて、実行されるアクションを定義します。 ON ステートメントが実行されることを、指定の条件に関する ON ユニットが*確立される* といいます。 ON ユニットは、ON ステートメントが実行される時点では実行されません。指定された条件が生じた場合のみ実行されます。

ON ユニットは、ラベルなしの単一の単純ステートメント、またはラベルなしの開始ブロックのどちらでもかまいません。単純ステートメントの場合は、BEGIN、DECLARE、DEFAULT、DO、END、ENTRY、FORMAT、ITERATE、LEAVE、OTHERWISE、PROCEDURE、RETURN、SELECT、WHEN、または % ステートメント以外の任意のステートメントとすることができます。 ON ユニットが開始ブロックの場合は、 RETURN ステートメントは開始ブロック内のネストされたプロシージャー内にのみ現れます。 LEAVE ステートメントは開始ブロック内のネストされた DO グループ内にのみ現れます。

ただし、セミコロン (;) または RESIGNAL ステートメントのいずれかでのみ形成されている ON ユニットを除き、ON ユニットは、それがあるブロックに対して内部的なプロシージャー (パラメーターなしの) と見なされます。 ON ユニット内で参照された名前は、その条件が起こったときの環境で知られている名前ではなく、その ON ユニットを指定した ON ステートメントが実行されたときの環境で知られている名前です。

ON ユニットの実行が完了すると、通常は、その ON ユニットに制御を渡したブロックに制御が戻されます。プロシージャーの場合と同様に、GO TO ステートメントを使用して ON ユニットの外に制御を移すことができ、その場合は、GO TO ステートメントで指定した場所に制御権が移され、正常な戻りは行われません。

ON ユニットからどこの場所に制御が戻されるかは、それぞれの条件によって異なります。条件ごとの正常な戻りについては、 367 ページの『第 17 章 条件』で説明します。

ヌル ON ユニット

ヌル・ステートメント ON ユニットが実行されると、その条件からの正常な戻りが実行されます。

ヌル ON ユニットを使用することは、次の 2 つの理由から、条件を割り込み禁止にすることと同じではありません。

- ヌル ON ユニットはどの条件に対しても指定できますが、条件のなかには割り込み禁止にすることができないものがあります。
- 条件を割り込み禁止にした場合、その条件が発生したかどうかは検査されないため、実行時間を節約することができます。 (ヌル ON ユニットを指定した場合は、PL/I はその条件が発生したかどうかを検査します。)

ON ユニットの有効範囲

ON ステートメントが実行されると、条件の発生時に取られる処置が指定されます。設定されたアクションは、別の ON ステートメントまたは REVERT ステートメントの実行によって無効にされない限り、また ON ステートメントが実行されるブロックが終了しない限り、そのブロック全体および動的に派生したすべてのブロック全体で設定されたままになっています。 (動的に派生したすべての ON ユニットの詳細については、 362 ページの『動的に派生した ON ユニット』を参照してください。)

別の ON ステートメントで同じ条件を指定したときは、次のようになります。

- 前に実行された ON ステートメントで指定したものと同じ条件を、あとで実行されるステートメントで指定し、前の ON ステートメントを含んでいるブロックから動的に派生したブロック内であとの ON ステートメントが実行されたときは、前の ON ステートメントで指定した処置が一時的に中断され、スタックされます。 REVERT ステートメントが実行されるか、またはあとの ON ステートメントを含んでいるブロックが終了すると、保留されていたアクションが復元されます。

  ブロックから制御が戻されると、そのブロックを活動化したときにすでに設定されていたすべてのアクションが再設定されます。したがって、あるブロックですでに設定されているアクションを、そのブロックから呼び出されたサブルーチンで変更することはできません。
- 後ろの ON ステートメントと前の ON ステートメントの双方が、同一の呼び出しで呼び出された同一のブロックの内部にある場合は、前の ON ステートメントによって設定されたアクションは論理的に無効になります。別の ON ステートメントを実行する (または、無効にされた ON ステートメントを再び実行する) のでない限り、無効にされたアクションが再び設定されることはありません。

動的に派生した ON ユニット

ON ユニットの実行中に条件が起こり、その条件によりさらに別の ON ユニットを指定することがあります。ある ON ユニット内で条件が起こったか、または SIGNAL ステートメントによって条件が引き起こされたために別の ON ユニットに入った場合、その別の ON ユニットを、動的に派生した ON ユニットといいます。動的に派生した ON ユニットから正常に戻ると、その条件を引き起こした ON ユニットの環境が再び設定されます。

ERROR ON ユニット内で ERROR 条件が起こったとき、同一の ERROR ON ユニットが実行されると、再び ERROR 条件が起こってループ状態になります。ループ状態になって最大ネスト・レベルを超えると、メッセージが印刷され、アプリケーションは終了します。次の方法を使用すれば、このような状況で発生するループを防止することができます。

```
on error begin;
on error system;
:
end;
```

ファイル変数のための ON ユニット

ON ステートメントでファイル変数を指定した場合は、その ON ユニットの設定時にその変数の現行値であるファイル定数が参照されます。

例 1

```
dcl  F file,
     G file variable;
     G = F;
L1:  on endfile(G);
L2:  on endfile(F);
```

ラベル L1 と L2 が付いている 2 つのステートメントは同等です。

例 2

```
declare FV file variable,
FC1 file,
FC2 file;
FV = FC1;
on endfile(FV) go to Fin;
:
FV = FC2;
read file(FC1) into (X1);
read file(FV) into (X2);
```

最初の READ ステートメントの実行中に ENDFILE 条件が起こると、 ON ユニットに入ります。これは、この ON ユニットがファイル FC1 を参照しているためです。しかし、2 番目の READ ステートメントの実行中に ENDFILE 条件が起こっても、この READ ではファイル FC2 を参照しているので、この ON ユニットには入りません。

例 3

```
E:  procedure;
declare F1 file;
on endfile (F1) goto L1;
call E1 (F1);
:
E1:  procedure (F2);
declare F2 file;
on endfile (F2) go to L2;
read file (F1);
read file (F2);
end E1;
```

E1 の中で F1 がファイルの終わりになると、 E1 の中の F2 用の ON ユニットに入ります。 E1 の中で ON ユニットを指定しなかったとき、 F1 または F2 の ENDFILE 条件が起こった場合には、 E の中の F1 用の ON ユニットに入ることになります。

例 4

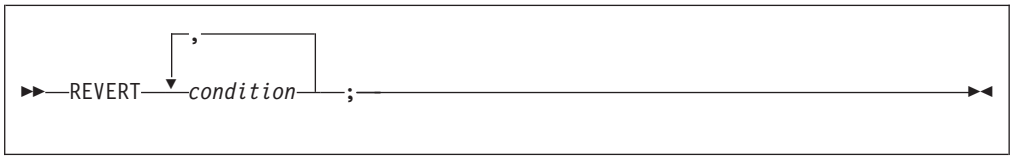
```
declare FV file variable,
FC1 file,
FC2 file;


do FV=FC1,FC2;
on endfile(FV) go to Fin;
end;
```

ファイル変数を指定している ON ステートメントが複数回実行され、なおかつ、その変数がそのつど異なる値を取る場合は、それぞれの実行のたびに別々の ON ユニットが設定されます。

REVERT ステートメント

あるブロックで REVERT ステートメントが実行されると、指定の条件に関してそのブロック内で実行された ON ユニットが取り消されます。そして、そのブロックが活動化されたときにすでに設定されていた ON ユニットが再び設定されます。 REVERT ステートメントの影響を受ける ON ステートメントは、その REVERT ス

テートメントを含んでいるブロックの内部にある ON ステートメントのうち、同一の呼び出しで呼び出されたそのブロック内ですでに実行されたものだけです。



condition

367 ページの『第 17 章 条件』に説明されている条件、または CONDITION 属性によって定義されている条件のいずれかです。

REVERT ステートメントが ON ユニットを取り消すのは、次の 2 つの条件が両方とも成立する場合に限ります。

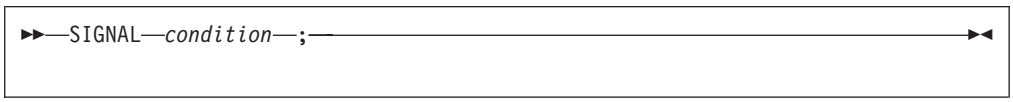
1. 派生元に適用可能であり、かつ REVERT ステートメントでリストされた条件を指定する ON ステートメントが、ブロックの活動化後に実行された。
2. 指定された条件付きの REVERT ステートメントが、同じブロック内で実行されたことがない。

これらの 2 つの条件のどちらかが成立しない場合には、REVERT ステートメントはヌル・ステートメントと見なされます。

SIGNAL ステートメント

SIGNAL ステートメントを使用すれば、条件を引き起こすことができます。 プログラムのテスト時にこのステートメントを使用して、ON ユニットのアクションが妥当であるかどうかを検査したり、指定の条件に正しいアクションが対応しているかを判別したりすることができます。条件が割り込み禁止になっていない限り、設定されているアクションが取られます。

指定した条件が割り込み禁止になっているときは、SIGNAL ステートメントはヌル・ステートメントと同等になります。

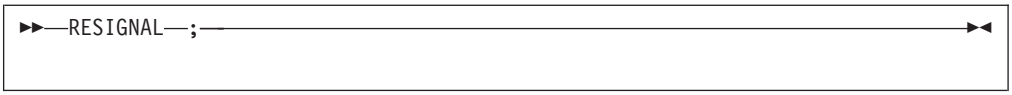


condition

367 ページの『第 17 章 条件』に説明されている条件、または CONDITION 属性によって定義されている条件のいずれかです。

RESIGNAL ステートメント

RESIGNAL ステートメントによって、現行の ON ユニットは終了して、別の ON ユニットが同一の条件で制御を獲得できるようになります。処理は、RESIGNAL を実行している ON ユニットが存在せず、制御を獲得しなかった場合と同様に続けられます。これによって、複数の ON ユニットが同一条件で制御を獲得することができます。



ただし、RESIGNAL は 1 個の ON ユニット内、またはそこから動的に派生したものにおいてのみ有効です。

複数条件

複数条件とは、複数の条件が同時に起こることをいいます。

複数条件が起こり得るのは、次の条件の場合です。
RECORD (380 ページの『RECORD 条件』を参照)。
TRANSMIT (385 ページの『TRANSMIT 条件』を参照)。

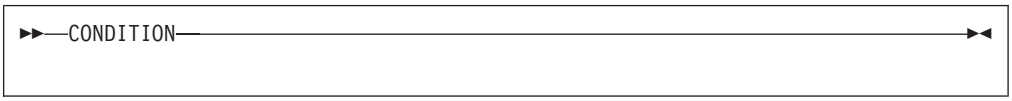
TRANSMIT 条件が常に最初に処理されます。 TRANSMIT の ON ユニットから正常に戻らなかったときは、RECORD 条件は無視されます。

複数条件は、順番に処理されます。次のイベントのいずれかが起こると、後続の条件は処理されません。

- ある条件の処理中に、その条件の暗黙アクション、ON ユニットからの正常な戻り、または ON ユニット内の異常終了によって、プログラムが終了した場合。
- GO TO ステートメントによって ON ユニットの外に制御が移され、その結果、正常に戻ることができなかった場合。

CONDITION 属性

CONDITION 属性は、宣言された名前がプログラマー定義の条件を示すことを指定します。



ON、SIGNAL、または REVERT ステートメント内に CONDITION 条件と一緒に書いた名前は、条件名であることがコンテキスト上から宣言されます。

デフォルトの有効範囲は EXTERNAL です。 CONDITION 条件の例は、370 ページに示されています。

第 17 章 条件

ANYCONDITION 条件 367
AREA 条件 369
ATTENTION 条件 369
CONDITION 条件 370
CONVERSION 条件 371
ENDFILE 条件 373
ENDPAGE 条件 374
ERROR 条件 375
FINISH 条件 375
FIXEDOVERFLOW 条件 376
INVALIDOP 条件 377
KEY 条件 378
NAME 条件 378
OVERFLOW 条件 379
RECORD 条件 380
SIZE 条件 381
STORAGE 条件 382
STRINGRANGE 条件 382
STRINGSIZE 条件 383
SUBSCRIPTRANGE 条件 384
TRANSMIT 条件 385
UNDEFINEDFILE 条件 386
UNDERFLOW 条件 387
ZERODIVIDE 条件 387

この章では、各条件をアルファベット順に説明します。一般的には、それぞれの条件ごとに下記のことがらを説明します。

- **状況** — プログラムの開始時に、条件が割り込み可能/割り込み禁止のどちらになっているかを示す標識、および条件を割り込み禁止にする方法 (できる場合) または割り込み可能にする方法。 358 ページの表 46 は、条件をタイプ別に分類し、状況を示したものです。また、これは、割り込み可能なものを割り込み禁止にする条件をリストします。
- **結果** — 条件を引き起こした操作の結果。条件が割り込み可能になっている場合と割り込み禁止になっている場合に分けて説明します。場合によっては、結果が未定義のこともあります。
- **原因と構文** — どのような場合にその条件が起こるかを含め、条件について述べます。 SIGNAL ステートメントによって起こる条件については、 364 ページの『SIGNAL ステートメント』を参照してください。
- **暗黙アクション** — 割り込み可能条件が起こり、しかもその条件の ON ユニットが確立されていないときに取られる処置。
- **正常な戻り** — ON ユニットが正常終了した場合に制御が戻される場所。 ON ユニットの外へ制御権を移動する GO TO ステートメントは、ON ユニットの異常終了です。 SIGNAL ステートメントによって条件 (ERROR 条件は除く) が引き起こされた場合、正常な戻りでは、SIGNAL のすぐあとのステートメントに戻ります。
- **条件コード** — プログラムの検査で検出される条件やエラーに対応するコード。各コードの解説は、「メッセージおよびコード」の『条件コード』の章に記載されています。

ANYCONDITION 条件

状況

ANYCONDITION は常に使用可能です。

結果

結果は、基礎的な条件の場合と同じです。

原因と構文

SIGNAL ANYCONDITION は使用できません。 ANYCONDITION は、条件をトラップする ON ユニットを確立 (または取り消し) する ON (および REVERT) ステートメントでのみ使用できます。それらの条件には、CONDITION 条件も含みます。この条件はブロック内で起こるもので、そのブロック内にあるほかの適格な ON ユニットによってトラップされません。

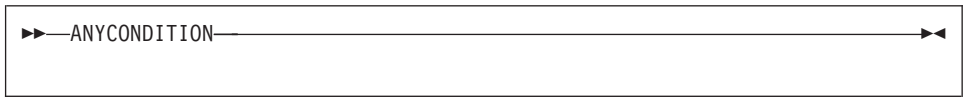
次に示す例では、すべての ERROR 条件は、開始ブロック内で処理され、FINISH 条件は、システムによって処理され、その他のすべての条件は、*handle_All_Others* という名前のルーチンの呼び出しによって処理されることになります。

```
on error
begin;
.
.
.
end;

on finish system;
on anycondition call Handle_all_others;
```

注: 無限ループを避けるために、ON ANYCONDITION を使用する時には ON FINISH (前述の例にあるように) の使用が必要になる場合があります。

条件が発生すると、呼び出しスタックが調べられ (逆方向に)、その条件の ON ユニットを含むブロックが検索されます。そのような ON ユニット、または ON ANYCONDITION ON ユニットを含む最初のブロックが検出されると、検索は停止します。そのような ON が検出されず、その条件が暗黙アクションによって ERROR にプロモートされる場合は (またその場合に限り)、スタックが再度調べられ、ON ERROR ON ユニットが検索されます。

ANYCONDITION ON ユニット内で ONCONDID 組み込み関数を使用して、どの条件が処理されているかを判別することができます。また、ONCONDCOND 組み込み関数を使用すれば、CONDITION 条件の名前を判別することができます。 ONFILE のようなほかの ON 組み込み関数を使用すれば、正確な原因を判別し、その他の関連する情報を知ることができます。これらの組み込み関数については、395 ページの『第 19 章 組み込み関数、疑似変数、およびサブルーチン』にまとめてあります。



省略形

ANYCOND

暗黙アクション

暗黙アクションは、基礎的な条件の場合と同じです。

正常な戻り

正常な戻りは、基礎的な条件の場合と同じです。

条件コード

ANYCONDITION に固有の条件コードはありません。

AREA 条件

状況

AREA 条件は常に割り込み可能です。

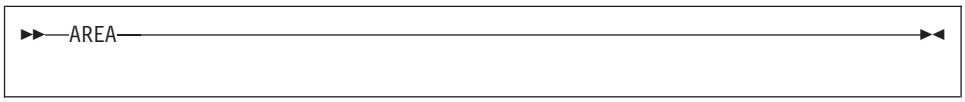
結果

AREA 条件を引き起こす割り振りまたは割り当ての試みは、なんの影響も及ぼしません。

原因と構文

AREA 条件は、次のいずれかの場合に起こります。

- ある区域内に基底付き変数を割り振ろうとしたが、その割り振りに必要なフリー・ストレージがその区域内にない場合。
- 区域の割り当てを実行しようとしたが、ソース区域内の割り振りを収容するのに必要なストレージがターゲット区域内にない場合。



暗黙アクション

メッセージが印刷され、ERROR 条件が起こります。

正常な戻り

ON ユニットからの正常な戻りに際して、次の処置が取られます。

- この条件が割り振りによって起こり、しかも ON ユニットがヌル ON ユニットである場合は、割り振りが再び試みられることはありません。
- この条件が割り振りによって起こった場合は、割り振りが再び試みられますが、その前に、区域参照が計算し直されます。したがって、ON ユニットで、不適当な区域を参照するよう修飾しているポインターの値を変更して、そのポインターが別の区域を指し示すようにしておけば、新しい区域内で割り振りが再び試みられます。
- この条件が区域割り当てまたは SIGNAL ステートメントによって起こった場合は、この条件が起こった場所から実行が続行されます。

条件コード

360、361、362

ATTENTION 条件

状況

ATTENTION 条件は常に割り込み可能です。

結果

この条件が起こると、ATTENTION の ON ユニットに入ります。 ATTENTION ON ユニットがない場合は、この条件は無視されるため、アプリケーションは終了されます。

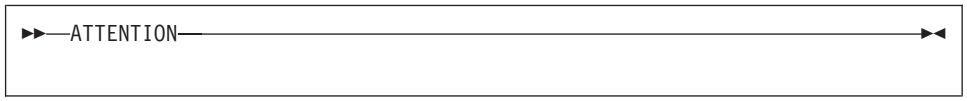
原因と構文

ATTENTION 条件は、ユーザーが、特定のキーの組み合わせを押してアプリケ

ーションに割り込んだときに起こります。この特定のキーは、オペレーティング・システムによって以下のように異なります。

- Windows では、CTRL-BRK および CTRL-C。ユーザーが CTRL-BRK または CTRL-C キーの組み合わせを入力した結果、ATTENTION ON ユニットは Windows 上でドリブンされません。暗黙アクションが実行されます。
- ホストでは、(使用可能であれば) ATTN キー。

またこの条件は、SIGNAL ATTENTION ステートメントによっても起こります。



省略形
ATTN

暗黙アクション
アプリケーションは終了します。

正常な戻り
ATTENTION の ON ユニットから戻ると、プログラム内の、この条件が起こった場所のすぐあとから処理が再開されます。

条件コード
400

CONDITION 条件

状況
CONDITION 条件は常に割り込み可能です。

結果
CONDITION 条件を使用すると、該当する CONDITION 条件の SIGNAL ステートメントが実行されるたびに実行されるような ON ユニットを確立することができます。

デバッグの援助として、この条件を使用して、プログラムの現在の状況を示す情報を印刷する ON ユニットを確立することができます。

原因と構文
CONDITION 条件は、SIGNAL ステートメントによって引き起こされます。SIGNAL ステートメントにどの名前が指定されているかによって、どの CONDITION 条件が引き起こされるかが決まります。 SIGNAL ステートメントを使用すれば、プログラム内のどこからでもその ON ユニットを実行させることができます。名前の有効範囲に関しては、通常の規則が適用されます。条件名はデフォルトでは EXTERNAL ですが、INTERNAL と宣言することもできます。

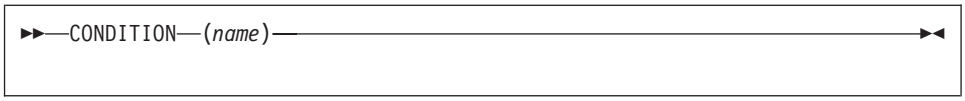
次に、CONDITION 条件の使用例を示します。

```
dcl Test condition;


on condition (Test)
```

```
begin;
:
end;
```

開始ブロックは、次に示すステートメントが実行されるときは必ず実行されます。

```
signal condition (Test);
```



省略形

COND

暗黙アクション

メッセージが印刷され、SIGNAL ステートメントの次のステートメントから実行が続行されます。

正常な戻り

SIGNAL ステートメントの次のステートメントから実行が続けられます。

条件コード

500

CONVERSION 条件

状況

CONVERSION は、NOCONVERSION 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み可能です。 ONSOURCE、ONCHAR、および ONWSOURCE 疑似変数を CONVERSION の ON ユニットで使用すれば、変換エラーを訂正することができます。

結果

CONVERSION 条件が起こったとき、結果フィールド全体の内容は未定義です。

原因と構文

CONVERSION 計算条件は、文字データ、ワイド文字、またはグラフィック・データを変換する際に無効な変換が試みられると起こります。この試みは、内部的に行われることも、入出力処理中に行われることもあります。たとえば、次のような場合、この条件が起こります。

- ビット・データに変換される文字データ内に、0 でも 1 でもない文字がある場合。
- 数字フィールドまたはコード化算術値に変換される文字値の中に、算術定数 (符号は任意) の表記でない文字や、複素定数を表す式が含まれている場合。
- 文字に変換されるグラフィック (DBCS) ストリングが、SBCS に変換できないグラフィックを含む場合。
- 文字ピクチャー項目に変換される値の中に、そのピクチャー指定で許可されていない文字が含まれている場合。

文字データの変換は、必ず左から右へ文字単位で行われ、無効文字が検出されるたびに、この条件が起こります。また、すべての文字がブランクのときもこの条件が起こります。ただし、下記の場合は例外です。

- F フォーマット項目の入力の場合。ゼロの値が想定されます。
- E フォーマット項目の入力の場合。場合によっては、ON ユニットに繰り返し入ることがあるので注意してください。

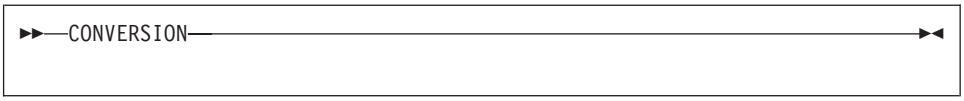
ヌル・ストリングまたは 1 つ以上のブランクから成るストリングが数値変数に割り当てられた場合、CONVERSION 条件は起こらないことに注意してください。

先行ゼロの除外が、この条件に対して指定されている現在の処置が実行されます (CONVERSION が割り込み禁止でない場合)。指定された処置が ON ユニットである場合、無効文字はその ON ユニット内で置き換えることができます。

- ソース・データが文字のときは、ONSOURCE または ONCHAR 疑似変数を使用します。
- ソース・データがワイド文字のときは、ONWSOURCE または ONWCHAR 疑似変数を使用します。
- ソース・データがグラフィックのときは、ONGSOURCE 類似変数を使用します。

CONVERSION 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。

グラフィック・データから非グラフィック・データへの変換中に CONVERSION 条件が起きた場合、ONCHAR および ONSOURCE 組み込み関数には有効なソース・データは含まれません。 ONGSOURCE 組み込み関数には、もとのグラフィック・ソース・データが入っています。グラフィック変換は、ONGSOURCE 疑似変数が CONVERSION の ON ユニットで使用され、CONVERSION 条件を起こしたグラフィック・データの修正を試みる場合に再試行されます。 ONGSOURCE 疑似変数を CONVERSION の ON ユニットで使用しない場合、ERROR 条件が起こります。



省略形

CONV

暗黙アクション

メッセージが印刷され、ERROR 条件が起こります。

正常な戻り

CONVERSION 条件が文字ストリング・ソース (グラフィック・ソースではない) で引き起こされ、かつ ONSOURCE または ONCHAR 疑似変数のいずれかが ON ユニットで使用される場合、プログラムは ON ユニットからの戻り時に変換を再試行します。

CONVERSION がグラフィック・ソースで引き起こされ、かつ ONGSOURCE 疑似変数が ON ユニットで使用される場合、プログラムは ON ユニットからの戻り時に変換を再試行します。

CONVERSION がワイド文字ソースで引き起こされ、かつ ONWSOURCE 疑似変数が ON ユニットで使用される場合、プログラムは ON ユニットからの戻り時に変換を再試行します。

変換エラーがこれらの疑似変数によって修正されないと、プログラムはループします。

条件コード

600-672

ENDFILE 条件

状況

ENDFILE 条件は常に割り込み可能です。

結果

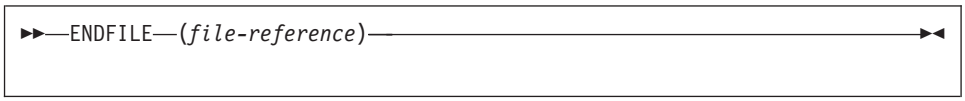
指定されたファイルがこの条件が起こってからクローズされない場合、このファイルに対する後続の GET または READ ステートメントは失敗し、これによって新たな ENDFILE 条件が引き起こされます。

原因と構文

ENDFILE 入出力条件は処理中に起こる可能性があり、GET ステートメントまたは READ ステートメントで指定したファイルの終わりを超えて読み取ろうとすると起こります。この条件は、SEQUENTIAL INPUT ファイル、SEQUENTIAL UPDATE ファイル、および STREAM INPUT ファイルにのみ適用されます。

レコード単位データ伝送では、READ ステートメントの実行中にファイルの終わりが検出されると、ENDFILE 条件が起こります。

ストリーム指向データ伝送では、GET ステートメントの実行中に、GET ステートメントのデータ・リスト内の項目の伝送が完了する前、またはデータ項目が伝送されたあと次のデータ項目が伝送される前のいずれかでファイルの終わりが検出されると、ENDFILE 条件が起こります。データ項目の処理中にファイルの終わりが検出された場合や、X フォーマット項目の処理中にファイルの終わりが検出された場合は、ERROR 条件が起こります。



file-reference (ファイル参照) はスカラー参照でなければなりません。

暗黙アクション

メッセージが印刷され、ERROR 条件が起こります。

正常な戻り

ENDFILE を引き起こした GET または READ ステートメントのすぐあとのステートメントから実行が続けられます。

この条件の ON ユニット内でファイルをクローズした場合は、正常な戻りの結果がどうなるかはわかりません。ファイルをクローズした ON ユニットから出るときは、GO TO ステートメントを使用しなければなりません。

条件コード

70

ENDPAGE 条件

状況

ENDPAGE 条件は常に割り込み可能です。

結果

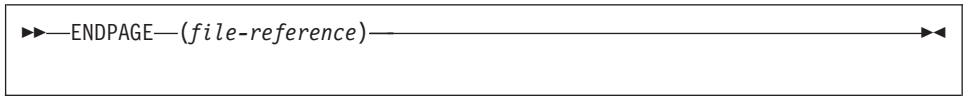
ENDPAGE 条件が起こると、現在行の行番号は、PAGESIZE オプション (デフォルトは 60) で指定した行番号より 1 だけ大きい値になります。したがって、同じページに書き続けることができます。ON ユニットで PAGE オプションまたは PAGE フォーマット項目を実行して現在行を 1 にセットすれば、新しいページを開始することができます。

ON ユニットで新しいページを開始しない限り、現在行の行番号は無限に増えてゆく可能性があります。後続の LINE オプションまたは LINE フォーマット項目で現在行の行番号以下の行番号を指定すれば、ENDPAGE 条件は起こらず、現在行が 1 にセットされて新しいページが開始されます。ただし例外として、現在行の行番号が指定された行番号に等しく、しかもファイルが現在行の 1 桁目に位置づけられている場合は、ENDPAGE 条件は起こりません。

データ伝送中に ENDPAGE 条件が起こったときは、ON ユニットから戻ると、データが現在行 (ON ユニットで変更された可能性がある) に書き出されます。ENDPAGE 条件が LINE または SKIP オプションによって引き起こされたときは、ON ユニットから戻ると、LINE または SKIP で指定された処置は無視されます。

原因と構文

PUT ステートメントが実行された結果として、現行ページに関して指定されている限界を超えて新しい行を開始しようとすると、ENDPAGE 入出力条件が起こります。この限界は OPEN ステートメントの PAGESIZE オプションで指定することができます。PAGESIZE を指定しないと、デフォルトの限界である 60 が適用されます。データ伝送時に (編集ディレクティブの PUT ステートメントの場合は対応するフォーマット項目も含む)、LINE オプションまたは SKIP オプションを使用すれば、この限界を超えることができます。また、LINE オプションまたは LINE フォーマット項目で、現在行の行番号より小さい行番号を指定したときも、ENDPAGE 条件が起こることがあります。 ENDPAGE 条件は、SIGNAL ステートメントによって引き起こされる場合を除いて、1 ページに 1 回しか起こりません。



file-reference (ファイル参照) はスカラー参照でなければなりません。

暗黙アクション
新しいページが開始されます。この条件が SIGNAL ステートメントによって引き起こされた場合、実行は影響を受けず、SIGNAL ステートメントの次のステートメントから実行は継続されます。

正常な戻り
上で述べたようにして PUT ステートメントの実行が続けられます。

条件コード
90

ERROR 条件

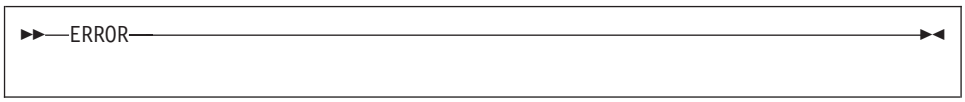
状況
ERROR 条件は、常に割り込み可能です。

結果
エラー・メッセージは、ON ユニットがアクティブでない場合に ERROR 条件が起こったときか、または ON ユニットが GOTO (ブロックから出る) を使用してこの条件からの回復を行わないときに出されます。

原因と構文
ERROR 条件は、多くの条件の暗黙アクションです。したがって、ERROR 条件は、それぞれの条件を個別に検査するのではなく、多数の異なる条件を検査するのに使用される共通の条件です。

ERROR 条件は、次の場合に起こります。

- ある条件で暗黙アクションが取られ、それによって ERROR 条件が引き起こされた場合。
- SUBSCRIPTRANGE CONVERSION などの条件の正常な戻り処置が取られたとき、または再試行が行われない場合。
- プログラムの実行中にエラー (ほかの PL/I 定義の条件を引き起こさないもの) が起こった場合。
- SIGNAL ERROR ステートメントを実行した場合。



暗黙アクション
メッセージが印刷され、FINISH 条件が起こります。

正常な戻り
暗黙アクションが取られます。

条件コード
1000 以上のすべてのコードは ERROR 条件です。

FINISH 条件

状況
FINISH 条件は常に割り込み可能です。

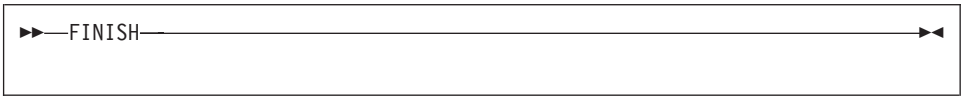
結果

制御が FINISH の ON ユニットに移り、処理を続行します。

原因と構文

FINISH 条件は、プロシージャーを終了するステートメントの実行中に起こります。次の処置が行われます。

- 正常終了の場合 — 主プロシージャーが PL/I の場合にのみ、FINISH ON ユニット (確立されている場合) に制御が移ります。
- 異常終了の場合 — FINISH ON ユニットがアクティブ・ブロックにある場合は、FINISH ON ユニットに制御が移ります。



暗黙アクション

- この条件がメジャー・タスクで起こった場合、なんの処置も取られません。つまり、この条件が起こった場所から処理が続けられます。
- この条件が別の条件の暗黙アクションの一部として起こった場合、プログラムは終了されます。

正常な戻り

処理は、その条件が起こった場所から再開します。条件が SIGNAL ステートメントによって引き起こされた場合、その SIGNAL ステートメントのあとのステートメントで処理が再開されます。

条件コード

4

FIXEDOVERFLOW 条件

状況

NOFIXEDOVERFLOW を指定している条件接頭語の有効範囲を除けば、FIXEDOVERFLOW 条件はプログラム全体にわたって割り込み可能です。

結果

無効な FIXED DECIMAL 操作の結果は未定義です。

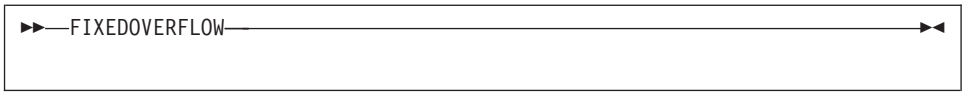
原因と構文

計算関係の条件である FIXEDOVERFLOW 条件は、FIXED DECIMAL 算術演算の結果の長さが、有効な最大の長さを超えたときに起こります。

FIXED BINARY 操作の場合は、FIXEDOVERFLOW 条件は起こりません。

FIXEDOVERFLOW 条件と SIZE 条件は次の点で異なっています。SIZE 条件は、演算結果が変数の宣言されているサイズを超えたときに起こります。一方、FIXEDOVERFLOW 条件は、演算結果の長さが計算機で許可されている最大の長さを超えたときに起こります。

FIXEDOVERFLOW 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。



省略形

FOFL

暗黙アクション

メッセージが印刷され、ERROR 条件が起こります。

正常な戻り

この条件が起こったすぐあとの位置に制御が戻されます。

条件コード

310

注: SIZE 条件が割り込み禁止になっているときに、固定小数点 10 進変数に大きすぎる数を割り当てようとすると、FIXEDOVERFLOW 条件が起こることがあります。

INVALIDOP 条件

状況

INVALIDOP 条件は、NOINVALIDOP 条件接頭語の有効範囲を除いて、プログラムの全体にわたって割り込み可能です。

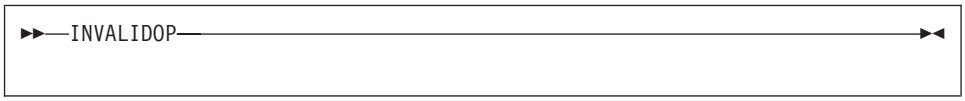
結果

無効な操作の結果は未定義です。

原因と構文

計算関係の条件である INVALIDOP 条件は、次のいずれかが IEEE 浮動小数点式の計算中に検出されたときに起こります。

- 無限大と無限大の減算
- 無限大に 0 を乗算
- 無限大と無限大の除算
- ゼロをゼロで除算
- 無効な浮動小数点データ



暗黙アクション

エラー条件が発生します。

正常な戻り

メッセージが印刷され、ERROR 条件が起こります。

条件コード

290

KEY 条件

状況

KEY 条件は常に割り込み可能です。

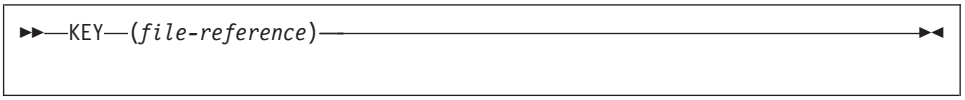
結果

キー順レコードが未定義のため、キー順レコードが入っているステートメントは無視されます。

原因と構文

KEY 入出力条件は、指定されたキーを持つレコードが検出されないときに起こります。この条件は、キー順レコードの処理中にのみ起こる可能性があります。また、以下にリストした条件コードの場合にこの条件が起こります。

データ・セットを参照する LOCATE ステートメントを使用したとき、この LOCATE ステートメントに対する KEY 条件は、そのファイルを参照する次の WRITE または LOCATE ステートメントが実行されるときまで、もしくはファイルがクローズされるときまで起こりません。



file-reference (ファイル参照) はスカラー参照でなければなりません。

暗黙アクション

メッセージが印刷され、ERROR 条件が起こります。

正常な戻り

KEY 条件を引き起こしたステートメントのすぐあとのステートメントに制御が渡されます。

この条件の ON ユニット内でファイルをクローズした場合は、正常な戻りの結果がどうなるかはわかりません。ファイルをクローズした ON ユニットから出るときは、GO TO ステートメントを使用しなければなりません。

条件コード

50-58

NAME 条件

状況

NAME 条件は常に割り込み可能です。

結果

指定されたデータは未定義です。

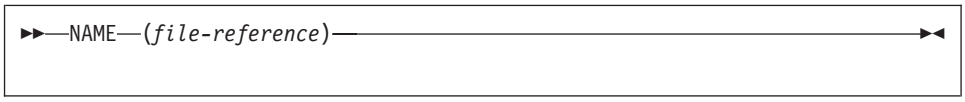
原因と構文

NAME 入出力条件は、FILE オプションが指定されたデータ・ディレクティブ GET ステートメントの実行中に起こる可能性があります。この条件は、次のいずれの場合にも起こります。

- 313 ページの『データ・ディレクティブ・データの構文』で説明した構文規則に従っていない場合。
- 名前が欠落しているか、正しくない場合。次に例を示します。

  - データ・リスト中に対応する名前がない。
  - データ・リストがない場合に、その名前がこのブロック内で知られていない。
  - 修飾された名前が完全には修飾されていない。
  - DBCS のどちらかのバイトの値が、有効な値の範囲 (X'41' から X'FE') 内にない。
- 添え字リストが欠落しているか、正しくない場合。次に例を示します。
  - 添え字が欠落している。
  - 添え字の個数が正しくない。
  - 添え字が 10 桁を超える (先行ゼロは無視される)。
  - 添え字が、変数の現在の割り振りで使用できる範囲内にない。

ON ユニット内で DATAFIELD 組み込み関数を使用すると、不正なデータ・フィールドを検索する場合があります。



file-reference (ファイル参照) はスカラー参照でなければなりません。

暗黙アクション

正しくないデータ・フィールドが無視され、メッセージが印刷されて、GET ステートメントの実行が続行されます。

正常な戻り

ストリーム中の次の名前を使用して GET ステートメントの実行が続けられます。

条件コード

10

OVERFLOW 条件

状況

NOOVERFLOW を指定している条件接頭語の有効範囲を除けば、OVERFLOW 条件はプログラム全体にわたって割り込み可能です。

結果

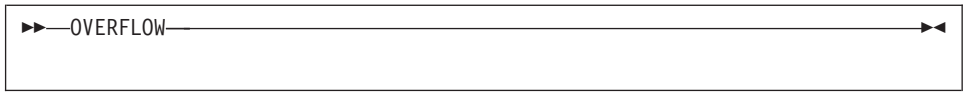
無効な浮動小数点演算の結果は未定義です。

原因と構文

計算関係の条件である OVERFLOW 条件は、浮動小数点数の大きさがシステムで定められている最大の大きさを超えたときに起こります。

OVERFLOW 条件と SIZE 条件は次の点で異っています。SIZE 条件は、演算結果が変数の宣言されているサイズを超えたときに起こります。一方 OVERFLOW 条件は、演算結果が計算機で許可されている最大の大きさを超えたときに起こります。

OVERFLOW 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。



省略形
OFL

暗黙アクション
メッセージが印刷され、ERROR 条件が起こります。

正常な戻り
エラー条件が発生します。

条件コード
300

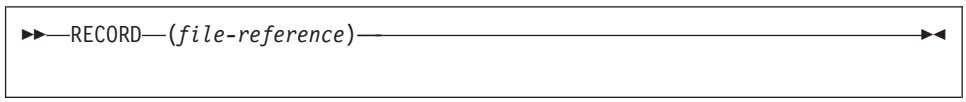
RECORD 条件

状況
RECORD 条件は常に割り込み可能です。

結果
指定されたファイルの長さ接頭部が間違って伝送される可能性があります。

原因と構文
RECORD 入出力条件は、指定されたレコードが切り捨てられたときに起こります。この条件は、READ、WRITE、LOCATE、または REWRITE 処理中にのみ起こる可能性があります。

SCALARVARYING オプションがファイルに適用されている場合は (位置指定モードを使用して可変長ストリングを伝送するときのファイルにはこのオプションを適用しなければならない)、2 バイトの長さの接頭部が可変長ストリング・エレメントと一緒に伝送されます。 RECORD 条件が起こったとき、長さ接頭部はリセットされません。 SCALARVARYING オプションがファイルに適用されていない場合は、長さ接頭部は伝送されません。つまり、入力の場合、可変長ストリングの現在の長さは、レコードの長さとストリングの最大の長さのうち短い方にセットされます。



file-reference (ファイル参照) はスカラー参照でなければなりません。

暗黙アクション
メッセージが印刷され、ERROR 条件が起こります。

正常な戻り
RECORD 条件を引き起こしたステートメントのすぐあとのステートメントから実行が続けられます。

この条件の ON ユニット内でファイルをクローズした場合は、正常な戻りの結果がどうなるかはわかりません。ファイルをクローズした ON ユニットから出るときは、GO TO ステートメントを使用しなければなりません。

条件コード
20-24

SIZE 条件

状況
SIZE 条件は、SIZE 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み禁止です。

結果
割り当ての結果は未定義です。

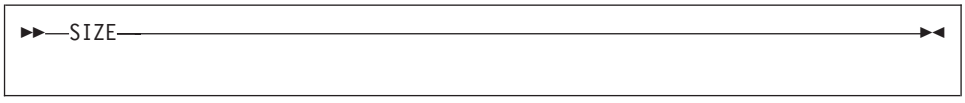
原因と構文
計算関係の条件である SIZE 条件は、変数や中間結果への割り当て時に、または入出力処理中に、高位の (つまり、左端の) 有効数字 (2 進数または 10 進数) が失われたときにのみ起こります。データ・タイプ、基数、スケール、または精度が異なっているときの変換時にも有効数字が失われることがあります。 SIZE 条件が使用不可であっても、インラインで実行されない変換であれば、条件が起こる可能性があります。

SIZE 条件は、あるデータ項目に割り当てられているサイズの値が、(この値がその項目が占有しているストレージの実際のサイズではない場合も) そのデータ項目の宣言されている (またはデフォルトの) サイズを超えるときに起こります。たとえば、精度 (20) の固定小数点 2 進数項目は、ストレージではフルワードを占有しますが、FIXED BINARY (20) を超える大きさの値をこの項目に割り当てようとすると SIZE 条件が起こります。

SIZE 条件の検査が行われると、ストレージ・スペースと実行時間の両方の点でオーバーヘッドが必要です。したがって、この条件は主にプログラム・テストに使用されます。実稼働プログラムではこの条件を取り除くことができます。テストと実稼働アプリケーション・プログラムについての詳細は、「*プログラミング・ガイド*」を参照してください。

SIZE 条件と FIXEDOVERFLOW 条件は次の点で異なっています。FIXEDOVERFLOW 条件は、計算された固定小数点値の値が、システムの許可する最大値を超えるときに起こります。値を割り当てるとき、その値の計算で FIXEDOVERFLOW 条件が起こったか否かには関係なく、SIZE 条件が起こる可能性があります。

SIZE 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。



暗黙アクション
メッセージが印刷され、ERROR 条件が起こります。

正常な戻り
この条件が起こったすぐあとの位置に制御が戻されます。

条件コード
340、341

STORAGE 条件

状況

STORAGE は常に使用可能です。

結果

結果は、この条件を起こしたストレージ割り振りの試みを示す変数のタイプによって異なります。

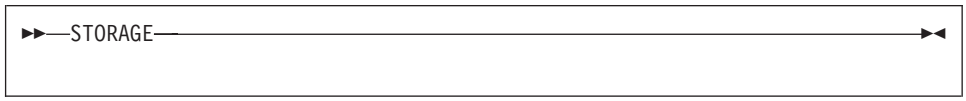
- 被制御変数の ALLOCATE ステートメントの後には、その変数の世代は割り振られません。その被制御変数を参照すると、ALLOCATE ステートメントの前にあるその変数の世代 (ある場合) をアクセスすることになります。
- 基底付き変数の ALLOCATE ステートメントのあとには、その変数は割り振られないため、対応するポインターも未定義です。
- 基底付き変数の ALLOCATE 組み込み関数のあとには、その変数は割り振られないため、対応するポインターの使用も未定義です。

原因と構文

STORAGE 条件が起こると、プログラムは、 BASED または CONTROLLED のストレージを AREA の外部に割り振ろうと試みる ALLOCATE 組み込み関数または ALLOCATE ステートメントが失敗した場合に制御を獲得します。AREA 内の ALLOCATE ステートメントの失敗によって、AREA 条件が引き起こされます。

AUTOMATIC 組み込み関数が失敗しても、STORAGE 条件は起こりません。

STORAGE 条件の ON ユニットは、割り振り済みストレージの解放を試みることができます。 ON ユニットは、十分なストレージを解放できない場合、必要なステップを行い、診断情報を失うことなくプログラムを終了することができます。



暗黙アクション

エラー条件が発生します。

正常な戻り

エラー条件が発生します。

条件コード

450、451

STRINGRANGE 条件

状況

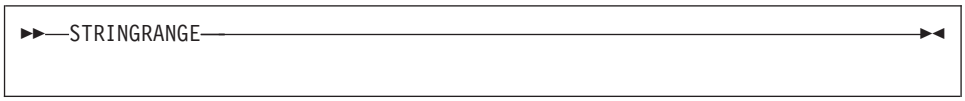
STRINGRANGE 条件は、STRINGRANGE 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み禁止です。

結果

指定された SUBSTR の値は変更されます。

原因と構文

SUBSTR 参照についての引数の値が、SUBSTR 組み込み関数の規則に従わなかった場合は、STRINGRANGE プログラム・チェックアウト条件が常に起こります。無効な引数への参照が行われるたびに、この条件が起こります。



省略形

STRG

暗黙アクション

メッセージが印刷され、「正常な戻り」の説明どおりに処理が続けられます。

正常な戻り

次のように定義される値を持つよう訂正された SUBSTR 参照を使用して、実行が続けられます。

ソース・ストリングの長さ (ON ユニットが指定されている場合はその実行後) を *k*、開始点を *i*、サブストリングの長さを *j* と仮定します。

- *i* が *k* より大きいと、この値は、ヌル・ストリングとなります。
- *i* が *k* 以下であると、この値は、ソース・ストリングの *m* 番目の文字、ビット、ワイド文字、またはグラフィックから開始するサブストリングになり、*m* および *n* が以下によって定義されている場合に、このサブストリングは、*n* 個の文字、ビット、ワイド文字、またはグラフィックになります。

```
M = max( I,1 )
N = max( 0,min( J + min(I,1) - 1,K - M + 1 ))
```

上記は、J を指定した場合です。

```
N = K - M + 1
```

上記は、J を指定しない場合です。

新しい引数は必ず範囲内にあります。

i と *j* の値は、ON ユニットに入る前に確立されます。これらの値は、ON ユニットから戻ったときに計算し直されません。

k の値は、SUBSTR の最初の引数が可変長ストリングである場合には、ON ユニット内で変更になることがあります。値 *n* は、ON ユニットから戻る際に *k* の新しい値を使用して再計算されます。

条件コード

350

STRINGSIZE 条件

状況

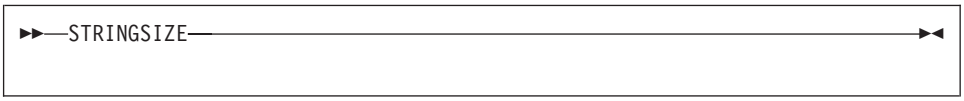
STRINGSIZE 条件は、STRINGSIZE 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み禁止です。

結果

この条件の処置が取られたあと、切り捨てられたストリングがそのターゲット・ストリングに割り当てられます。ソース・ストリングがターゲット・ストリング内に収まるように、ソース・ストリングの右端の文字、ビット、ワイド文字、またはグラフィックが切り捨てられています。

原因と構文

STRINGSIZE プログラム・チェックアウト条件は、あるストリングを、それより短い最大長を持つターゲット・ストリングに割り当てようとすると起こります。



省略形

STRZ

暗黙アクション

メッセージが印刷され、処理が続けられます。

正常な戻り

この条件が起こった位置から実行が続けられます。

条件コード

150、151

SUBSCRIPTRANGE 条件

状況

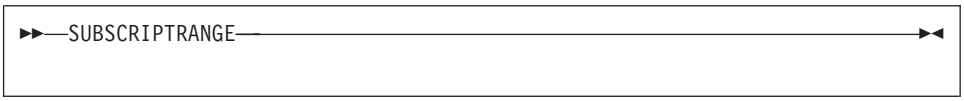
SUBSCRIPTRANGE 条件は、SUBSCRIPTRANGE 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み禁止です。

結果

SUBSCRIPTRANGE 条件が起こった場合、無効な添え字の値は未定義のため、どのような参照になるかわかりません。

原因と構文

SUBSCRIPTRANGE プログラム・チェックアウト条件は、計算後の添え字が指定された限界内に存在しないことが検出されたときに起こります。
SUBSCRIPTRANGE 条件の発生とその他の添え字の計算がどのような順序で行われるかはわかりません。



省略形

SUBRG

暗黙アクション

メッセージが印刷され、ERROR 条件が起こります。

正常な戻り

SUBSCRIPTRANGE の ON ユニットから正常に戻ると、ERROR 条件が起こります。

条件コード

520

TRANSMIT 条件

状況

TRANSMIT 条件は常に割り込み可能です。

結果

TRANSMIT 条件が起こると、伝送されたデータのいずれかが正しくない可能性があります。

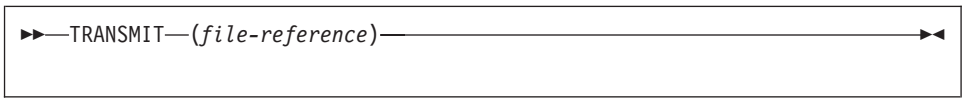
原因と構文

TRANSMIT 入出力条件は、どの入出力処理時にも起こる可能性があります。この条件は、レコード (ブロック化されていればブロック) の訂正不能な伝送エラーによって引き起こされます。この訂正不能な伝送エラーとは、実行中に訂正できなかった入出力エラーのことです。記録メディアが損傷していたり、指定やセットアップが正しくないような場合、このエラーが起こることがあります。

入力の場合は、潜在的に不正なレコードが伝送されたあとで、TRANSMIT 条件が起こります。レコードがブロック化されているときは、そのブロック内の後続の各レコードごとに TRANSMIT 条件が起こります。

出力の場合は、伝送後に TRANSMIT 条件が起こります。レコードがブロック化されているときの伝送は、それぞれの出力ステートメントが実行されたときではなく、ブロックがいっぱいになったときに行われます。

スパン・レコードの更新処理時には、レコードの最後のセグメントでのみ TRANSMIT 条件が起こります。同一ブロック内の後続のレコードではこの条件は起こりませんが、それらのレコードの保全性は保証できません。



file-reference (ファイル参照) はスカラー参照でなければなりません。

暗黙アクション

メッセージが印刷され、ERROR 条件が起こります。

正常な戻り

エラーが起こらなかった場合と同様に処理が続けられます。したがって、TRANSMIT 条件を引き起こしたステートメントまたはデータ項目によって、別の条件 (たとえば、RECORD 条件) が引き起こされることがあります。

この条件の ON ユニット内でファイルをクローズした場合は、正常な戻りの結果がどうなるかはわかりません。ファイルをクローズした ON ユニットから出るときは、GO TO ステートメントを使用しなければなりません。

条件コード

40-46

UNDEFINEDFILE 条件

状況

UNDEFINEDFILE 条件は常に割り込み可能です。

結果

指定されたファイルは、アプリケーション・プログラムに対して未定義です。

原因と構文

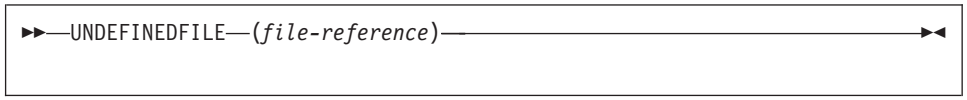
UNDEFINEDFILE 入出力条件は、ファイルのオープンに失敗するたびに起こります。複数のファイルを指定している OPEN ステートメントによってオープンしようとした場合、指定されているすべてのファイルのオープンが試みられたあとで、この条件が起こります。

同じ OPEN ステートメント内の複数のファイルについて UNDEFINEDFILE 条件が起こった場合は、その OPEN ステートメントにファイル名が記述されている順に (右から左へ)、ON ユニットが実行されます。

データ伝送ステートメントで、ファイルが暗黙にオープンされるときにこの条件が起こった場合は、ON ユニットから正常に戻ると、そのデータ伝送ステートメントの残りの部分から処理が続けられます。 ON ユニット内でファイルがオープンされなかった場合は、そのステートメントは処理を続けられず、ERROR 条件が起こります。

UNDEFINEDFILE 条件は、属性の指定が矛盾している場合 (たとえば、DIRECT と PRINT をともに指定する) に起こるだけでなく、次の場合にも起こります。

- ブロック・サイズがレコード・サイズより小さいとき (ただし、スパン・レコードの場合は除く)。
- LINESIZE が許可されている最大値を超えているとき。
- INDEXED データ・セットの作成時に、KEYLENGTH がゼロであるか、または指定されていないとき。
- INDEXED データ・セットの場合に、KEYLENGTH + KEYLOC の値がレコード長を超えるような KEYLOC オプションを指定したとき。
- STREAM データ・セットの場合に、V フォーマットの論理レコードの長さとして 18 バイト未満の値を指定したとき。
- FB フォーマット・レコードのブロック・サイズとして、レコード・サイズの整数倍でない値を指定したとき。
- VB フォーマット・レコードの場合に、指定した論理レコードの長さが、指定したブロック・サイズより少なくとも 4 バイト小さい値になっていないとき。



file-reference (ファイル参照) はスカラー参照でなければなりません。

省略形

UNDF

暗黙アクション
メッセージが印刷され、ERROR 条件が起こります。

正常な戻り
最後の ON ユニットが正常に完了すると、この条件を引き起こしたステートメントのすぐあとのステートメントに制御が移されます。

条件コード
80-89、91-95

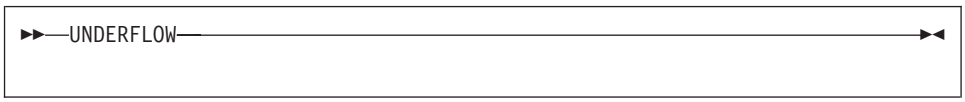
UNDERFLOW 条件

状況
UNDERFLOW 条件は、NOUNDERFLOW 条件接頭語の有効範囲を除き、プログラム全体にわたって割り込み可能です。

結果
無効な浮動小数点数値が 0 にセットされます。

原因と構文
計算関係の条件である UNDERFLOW 条件は、浮動小数点数の大きさが、許可されている最小値より小さいときに起こります。等しい数同士の減算の場合は、UNDERFLOW 条件は起きません (この種のエラーを有効桁エラーといいます)。

式 $\mathbf{X}^{(-Y)}$ (ここで、Y>0) は、$\mathbf{X}^{Y}$ の逆数を取ることによって計算できます。そのため、OVERFLOW 条件は、UNDERFLOW 条件の代わりに起こることがあります。



省略形
UFL

暗黙アクション
z/OS 上で IEEE に準拠して実行されている場合を除き、メッセージが出力され、条件が発生した位置から実行が継続されます。 z/OS 上で IEEE に準拠している場合は、メッセージが出力され、ERROR 条件が発生します。

正常な戻り
z/OS 上で IEEE に準拠して実行されている場合を除き、この条件が発生した直後の位置に制御が戻されます。z/OS 上で IEEE に準拠している場合は、ERROR 条件が発生します。

条件コード
330

ZERODIVIDE 条件

状況
ZERODIVIDE 条件は、NOZERODIVIDE 条件接頭語の有効範囲内を除き、プログラム全体にわたって割り込み可能です。

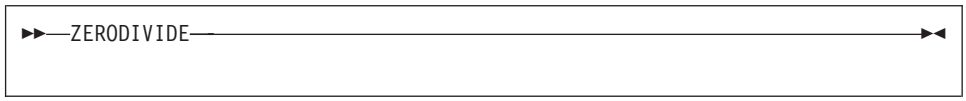
結果

ゼロによる除算の結果は未定義です。

原因と構文

計算関係の条件である ZERODIVIDE 条件は、ゼロによる除算を試みたときに起こります。この条件は、固定小数点の除算でも浮動小数点の除算でも起こります。浮動小数点の除算で割られる数もゼロである場合は、INVALIDOP 条件が起こります。

ZERODIVIDE 条件が起こったとき、この条件が割り込み禁止になっていると、プログラムはエラーになります。



省略形

ZDIV

暗黙アクション

メッセージが印刷され、ERROR 条件が起こります。

正常な戻り

エラー条件が発生します。

条件コード

320

第 18 章 マルチスレッド化機能

スレッドの作成. 390
ATTACH ステートメント 390
例 391
スレッドの終了. 391
スレッドの完了の待機 392
スレッドの切り離し 392
条件処理 393
タスク・データとタスク属性 393
THREADID 組み込み関数 393
スレッド間のデータの共用 394
スレッド間のファイルの共用 394

PL/I のプログラムは、1 つまたは複数のプロシージャーからできています。プログラムは、普通は単一の実行単位として実行されるか、単一の実行単位の一部として実行されます。プロシージャーが別のプロシージャーを呼び出す場合、制御は呼び出されたプロシージャーに渡され、呼び出した側のプロシージャーの実行は、呼び出されたプロシージャーが制御を戻すまで中断されます。このような単一の制御フローをともなう実行は、*同期* フローといいます。

PL/I のマルチスレッド機能を使用すると、呼び出す側のプロシージャーから、呼び出されたプロシージャーに制御が解放されません。その代わりに、さらに追加の制御フローが確立され、両方のプロシージャーが並行して実行されます。このような並行プロシージャーの実行は、*非同期* フローといいます。

PL/I のマルチスレッド機能によって、PL/I プログラムの各部分の実行が複数のスレッドを使用して非同期に行えるようになります。**スレッド**は、コンピューター・システムのリソースを獲得するために競合する作業単位です。 スレッドは、OS PL/I におけるタスクとは同等ではありません。あるプログラム内では主スレッドを除き、そのプログラム内にあるスレッドは必ず他のスレッドから独立しており互いに無関係です。あるプロシージャーが別のプロシージャーをスレッドとして呼び出す場合、この処置は、スレッドの*接続* または作成といいます。

1 つまたは複数のスレッドの実行が、1 つの**プロセス**内で起こることを PL/I プログラムと考えることができます。 PL/I は、複数のプロセスまたはタスクを作成したり管理したりする機能を持っていませんが、単一のプログラム (プロセス) において複数のスレッドを作成したり管理したりすることはできます。

通常は、プロセスごとに 1 つのアプリケーション・スレッドがあります。複数のスレッドを接続 (作成) して、次のようなことができます。

- より短い経過時間内に作業の一部を行う。そのようなアプリケーションはユーザーと対話を行わないバッチ・アプリケーションにすることができます。たとえば、あるプロシージャーは、 PL/I プログラムをコンパイルするスレッドを接続する場合があります。
- 1 つのプログラムの頻繁な応答部分を 1 つ目のスレッドが分担し、入出力を別のスレッドが分担し、通常の応答部分を 3 つ目のスレッドが分担して実行する。
- 場合によっては活動停止中の計算機システムのリソースを使用する。このようなリソースには、入出力装置ばかりでなく CPU も含まれることがあります。
- 応答時間が問題になるリアルタイムのマルチユーザー・アプリケーションを導入する。

- 信頼性を確保するために作業の一部を単独に分離する。つまり、独立しているほかの各部分が処理されている一方で、ジョブの一部に起こった障害を分離できます。

注: オペレーティング・システムのサービスは、PL/I が適切な機能を提供する場合には、直接に使用する必要はありません。

スレッドの作成

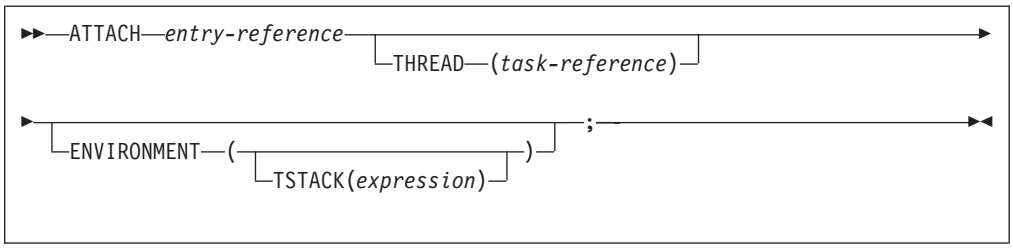
スレッドは次のような性格を持ちます。

- 独立した作業単位です。
- プロセスやシステム内において他のスレッドと並行的にかつそれらから独立して実行されます。
- 他のスレッドを接続できます。
- あるスレッドの完了を待機できます。
- 自分自身または他のスレッドを停止できます。

スレッドによって同期的に呼び出されたプロシージャーや関数は、そのスレッドの実行の一部となります。

ATTACH ステートメント

スレッドの接続 (または作成) は、ATTACH ステートメントを実行して行います。デフォルトを使用したくなければ、スレッドの特性を明示的に指定できます。



entry

LIMITED 入り口変数の名前、または外部の入り口名やレベル 1 のプロシージャー名を指定します。内部プロシージャーや取り出し可能プロシージャーの名前は指定できません。 ATTACH された入り口は、パラメーターを持たない、または厳密に 1 つの BYVALUE POINTER パラメーターを持つものとして宣言されなければなりません。ただし、プロシージャーに LIMITED 入り口変数を割り当て、その入り口変数をスレッドとして接続すれば、プロシージャーを取り出すことはできます。

新しいスレッドに引数を渡すことができます。これはちょうど CALL ステートメントで同期した入り口に引数を渡すのと同様です。

THREAD (task reference)

スレッドと関連づけられるタスク変数の名前を指定します。タスク変数は、名前を付ければスレッドを参照するために使用できます。

明示的に宣言しない限り、コンテキスト宣言で変数に名前が付与されます。

THREAD オプションを指定しない場合、接続されたスレッドを、別のスレッドによって停止させたり待機させたりすることができません。

スレッドが THREAD オプションを指定して接続された場合、そのスレッドに関連付けられたすべてのシステム・リソースを解放するには、DETACH ステートメントを使用して切り離さなければなりません (392 ページの『スレッドの切り離し』を参照してください)。

スレッドの作成にオペレーティング・システムのサービスを直接に使用することはできません。

ENVIRONMENT (省略形: ENV)
環境的な特性を指定します。通常はオペレーティング・システムに依存します。

TSTACK (expression)
Intel 上では、接続されるスレッドで使用されるスタックのサイズを指定します。式は FIXED BINARY(31,0) にする必要があります。スタック・サイズを指定しないと、実行時のデフォルトが使用されます。

z/OS 上では、TSTACK は無視され、スタックのサイズは LE によって決定されます。

例

```
attach input (File1);

attach input (File2)
   thread (T2);
```

付加されたプロシージャーは、サポートされる任意のリンケージを持つことができます。

スレッドの終了

スレッドは、次のような場合に終了します。

- 先頭のプロシージャー (当該スレッド内の初期プロシージャー) に対応している END ステートメントに達する場合。
- ERROR 条件が発生し、ERROR ON ユニットが存在しないか、あるいは ERROR ON ユニットが正常に終了する場合 (その ON ユニットの END ステートメントに達するか、 RESIGNAL ステートメントを実行する場合)。
- 当該スレッド内のいずれかのプロシージャーで EXIT ステートメントが実行される場合。
- プログラム内の開始スレッドが終了する場合。
- そのプログラムのいずれかのスレッドで STOP ステートメントが実行される場合。これはプログラム全体を停止し、主スレッドを含めてすべてのスレッドを終了させます。

FINISH 条件は、プログラム終了を引き起こしたスレッド内でのみ発生します。スレッドが実際に終了する前にそのスレッド内に確立されているいずれかの ON ユニットに制御が渡されます。

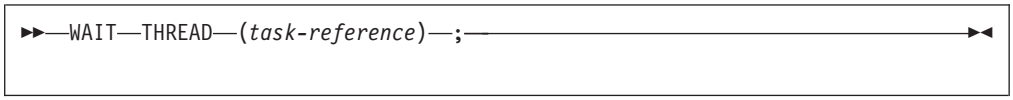
前述の場合を除き、あるスレッドが終了するとき、終了するそのスレッドが主スレッドでない限り、他のスレッドは終了しません。主スレッドが終了する場合は、それが終了する前に、他のすべてのスレッドが停止し終了します。

あるスレッドが終了すると、そのスレッドのスタック空間のみが解放されます。割り振られたストレージ、オープンしたファイルなどのような他のすべてのリソースは、そのまま残されます。スレッドが必要としたなんらかのリソースが解放され、オープンしたファイルがクローズされたことをユーザーは確認しなければなりません。ただし、それらがまだアクティブの他のスレッドで必要であるならば別です。

主スレッドが終了すると、すべてのリソースが解放され、ファイルがクローズされます。

スレッドの完了の待機

スレッドを待機するには、WAIT ステートメントを使用します。

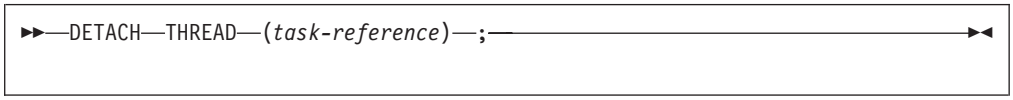


THREAD (task-reference)
THREAD オプションには、プロセスが待機しているスレッドを指定します。現行のスレッドは、指定したスレッドが終了するまで中断されます。指定したスレッドが終了すると、すぐに現行のスレッドが再開します。

```
WAIT THREAD (TI1);
```

スレッドの切り離し

DETACH ステートメントは、THREAD オプションを使用して接続されたスレッドに関連付けられているシステム・リソースを解放するために使用します。



THREAD (task-reference)
THREAD オプションには、切り離すスレッドを指定します。

通常このステートメントは、終了するスレッドの WAIT ステートメントの直後に実行します。

条件処理

新しいスレッドが作成されたとき、ON ユニットが確立されることを想定しているわけではありません。スレッドが作成されたときに有効である ON ユニットは、新しく接続されたスレッドによって継承はされません。あるスレッド内で起こった条件は、そのスレッド内で処理されるだけで、スレッド境界を超えて処理されることはありません。

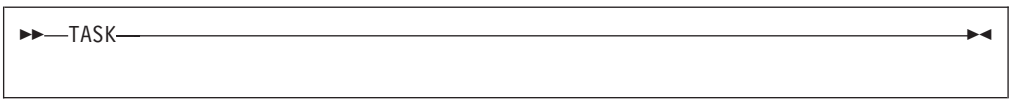
例えば、スレッド **A** がファイル **F** をオープンし、次に **A** がスレッド **T** を作成すると仮定します。すると、**T** は、ENDFILE 条件を発生させます。 ON ENDFILE 条件がスレッド **T** 自体の中に確立されていないと、 ERROR 条件が **T** 内で起こり、通常の条件処理作業がすべてスレッド **T** 内部で行われます。スレッド **A** が ENDFILE や ERROR に ON ユニットを確立しているか否かということは、スレッド **T** の実行に影響を及ぼしません。

スレッドは、条件処理の必要がある場合には、該当する条件に対応した ON ユニットを確立しておかなければなりません。スレッドを超えて条件を知らせるメカニズムはありません。

ATTENTION 条件を起こすために CTRL-BREAK を使用すると、その ATTENTION 条件は、ATTACH ステートメントによって作成されたどのスレッドでもなく、主スレッドの中でのみ起こります。

タスク・データとタスク属性

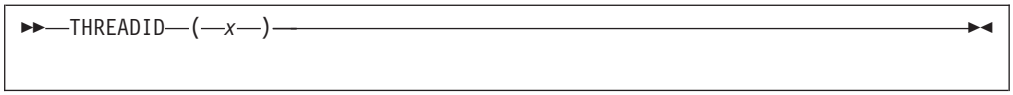
タスク変数は、スレッド ID、サービス・カテゴリー、およびディスパッチング優先順位などのスレッド関連情報を保持しています。明示的な宣言、または THREAD オプションの中に示されたことによる暗黙の指定によって、変数は TASK 属性を付与されます。



タスク変数とスレッドとの関連付けは、そのスレッドを作成する ATTACH ステートメントの THREAD オプションでタスク参照を指定することによって行われます。タスク変数は、それが関連付けられているスレッドがアクティブであれば、アクティブです。タスク変数は、それがスレッドと関連づけられる前に割り振られていなければならず、関連づけられたスレッドがアクティブである間は解放できません。アクティブなタスク変数は、別のスレッドと関連づけることはできません。

THREADID 組み込み関数

THREADID (スレッド ID の短縮形) は、POINTER 値を戻します。この値は、接続されたスレッドのオペレーティング・システム・スレッド ID のアドレスです。



この組み込み関数によって使用される値は、DosSetPriority のようなシステム機能に対するパラメーターとして使用することができますが、DosKillThread へのパラメーターとして使用してはなりません。

x タスク参照。 *x* の値は、ATTACH ステートメントの THREAD オプションですでに設定されているはずです。

スレッド間のデータの共用

すべての静的データと被制御データは、スレッドの間で共用されます。その他のすべてのデータも、それが割り振られており、スレッドのすべてがそのデータの使用を終わりにしない限り、引数/パラメーターおよび基底付き参照を介して共用されます。たとえば、接続する側のスレッドの持つ自動変数は、接続される側のスレッドと共用され、接続する側のブロックは、接続される側のスレッドが共用変数の使用を終わりにするまで終了することはできません。

データの逐次化に関しては、ユーザーが責任を持たねばなりません。被制御データの新しい世代が割り振られたり、既存の世代が解放される場合、一部のスレッドがより古い世代やもはや存在しない世代に依然としてアクセスしていることがあります。このような場合、予期せぬ結果を引き起こします。

すべての割り振られたストレージは、明示的に解放しない限り、プログラムが終了するまで解放されません。

PL/I は、AREA 変数内の ALLOCATE や FREE は逐次化しません。

スレッド間のファイルの共用

すべてのファイルがスレッド間で共用されます。あるスレッドがファイルをオープンすると、そのファイルを明示的にクローズするか、またはプログラムが終了するまで、そのファイルはクローズされません。

逐次化に関しては、ユーザーが責任を持たねばなりません。 『スレッド間のデータの共用』を参照してください。

メッセージ・ファイルと表示ステートメントは、 PL/I によって自動的に逐次化されます。

第 19 章 組み込み関数、疑似変数、およびサブルーチン

組み込み関数、疑似変数、および組み込みサブルーチンの宣言および呼び出し 398
BUILTIN 属性 398
組み込み関数および疑似変数の呼び出し . . . 399
組み込みサブルーチンの呼び出し. 399
組み込み関数、疑似変数、および組み込みサブルーチンの引数の指定 400
引数の集合 400
ヌル引数およびオプショナル引数. 400
数学関数の精度. 401
組み込み関数のカテゴリー 401
算術組み込み関数 401
配列処理組み込み関数 402
バッファー管理組み込み関数 402
条件処理組み込み関数 403
日付/時刻組み込み関数 404
浮動小数点の照会組み込み関数 406
浮動小数点演算組み込み関数 407
入出力組み込み関数 407
整数演算組み込み関数 407
数学組み込み関数 408
その他の組み込み関数 409
序数処理組み込み関数 410
精度処理組み込み関数 410
疑似変数 410
ストレージ制御組み込み関数 411
ストリング処理組み込み関数 412
サブルーチン 414
ABS 415
ACOS 415
ACOSF 415
ADD 415
ADDR 416
ADDRDATA. 417
ALL 417
ALLOCATE 417
ALLOCATION 418
ALLOCSIZE 418
ANY 418
ASIN 418
ASINF 419
ATAN 419
ATAND 420
ATANF 420
ATANH 420
AUTOMATIC 421
AVAILABLEAREA 421
例 421
BINARY 421
BINARYVALUE 422
BIT. 422
BITLOCATION 422
BOOL 423
BYTE 423
CDS 423
CEIL 424
CENTERLEFT 424
例 425
CENTRELEFT 425
CENTERRIGHT. 425
例 425
CENTRERIGHT. 425
CHARACTER 426
例 426
CHARGRAPHIC 426
例 1 427
例 2 427
CHARVAL 427
CHECKSTG 428
COLLATE 428
COMPARE 428
例 429
COMPLEX 429
CONJG 430
COPY 430
COS 430
COSD 431
COSF 431
COSH 431
COUNT 431
CS 432
CURRENTSIZE 433
CURRENTSTORAGE 434
DATAFIELD. 434
DATE 435
DATETIME 435
DAYS 435
例 436
DAYSTODATE 436
DAYSTOSECS 437
DECIMAL 437
DIMENSION. 438
DIVIDE 438
EDIT 438
例 439
EMPTY 439
ENDFILE 440
ENTRYADDR 440
ENTRYADDR 疑似変数 440
EPSILON 441
ERF 441
ERFC 441

EXP 441
EXPF 442
EXPONENT 442
FILEDDINT 442
FILEDDTEST 443
FILEDDWORD 443
FILEID 444
FILEOPEN 445
FILEREAD 445
FILESEEK 445
FILETELL 446
FILEWRITE 446
FIXED 446
FLOAT 447
FLOOR 447
GAMMA 447
GETENV 448
GRAPHIC 448
例 1 448
例 2 449
HANDLE 449
HBOUND 449
HEX 450
例 1 450
例 2 450
HEXIMAGE 451
HIGH 451
HUGE 451
IAND 452
IEOR 452
IMAG 453
IMAG 疑似変数 453
INDEX 453
例 453
INOT 454
例 454
IOR 454
ISIGNED 455
例 455
ISLL 455
例 456
ISMAIN 456
ISRL 456
例 456
IUNSIGNED 457
例 457
LBOUND 457
LEFT 458
例 458
LENGTH 458
LINENO 459
LOCATION 459
例 459
LOG 460
LOGF 460
LOGGAMMA 460
LOG2 461
LOG10 461
LOG10F 461
LOW 461
LOWERCASE 462
LOWER2 462
例 462
MAX 463
MAXEXP 463
例 (Intel の値) 463
例 (AIX の値) 464
例 (z/OS の 16 進値) 464
例 (z/OS の IEEE 値) 464
MAXLENGTH 464
例 464
MEMINDEX 465
例 465
MEMSEARCH 466
例 466
MEMSEARCHR 466
例 467
MEMVERIFY 467
例 468
MEMVERIFYR 468
例 469
MIN 469
MINEXP 469
例 (Intel の値) 469
例 (AIX の値) 470
例 (z/OS の 16 進値) 470
例 (z/OS の IEEE 値) 470
MOD 470
例 471
MPSTR 471
MULTIPLY 472
NULL 472
OFFSET 472
OFFSETADD 473
OFFSETDIFF 473
OFFSETSUBTRACT 473
OFFSETVALUE 474
OMITTED 474
ONCHAR 474
ONCHAR 疑似変数 474
ONCODE 475
ONCONDCOND 475
ONCONDID 475
ONCOUNT 476
ONFILE 476
ONGSOURCE 477
ONGSOURCE 疑似変数 477
ONKEY 477
ONLOC 478
ONSOURCE 478
ONSOURCE 疑似変数 479
ONSUBCODE 479

ONWCHAR 480
ONWCHAR 疑似変数 480
ONWSOURCE 480
ONWSOURCE 疑似変数 481
ORDINALNAME 481
ORDINALPRED 481
ORDINALSUCC 482
PACKAGENAME 482
PAGENO 482
PICSPEC 482
PLACES 483
例 (Intel の値) 483
例 (AIX の値) 483
例 (z/OS の 16 進値) 484
例 (z/OS の IEEE 値) 484
PLIASCII 484
PLICANC 484
PLICKPT 485
PLIDELETE 485
PLIDUMP 485
PLIEBCDIC 485
PLIFILL 486
例 486
PLIFREE 486
PLIMOVE 487
例 487
PLIOVER 487
PLIREST 488
PLIRETC 488
PLIRETV 488
PLISAXA 488
PLISAXB 489
PLISRTA 489
PLISRTB 490
PLISRTC 490
PLISRTD 490
POINTER 490
POINTERADD 491
POINTERDIFF 492
POINTERSUBTRACT 492
POINTERVALUE 492
POLY 492
PRECISION 493
PRED 493
PRESENT 494
PROCEDURENAME 494
PROD 494
PUTENV 495
RADIX 495
RAISE2 495
例 495
RANDOM 496
RANK 496
REAL 496
REAL 疑似変数 497
REM 497
REPATTERN 497
REPEAT 498
REPLACEBY2 498
REVERSE 499
例 499
RIGHT 499
例 500
ROUND 500
コンパイラー・オプション USAGE(ROUND(ANS)) のもとでの結果 500
コンパイラー・オプション USAGE(ROUND(IBM)) のもとでの結果 500
SAMEKEY 501
SCALE 501
SEARCH 501
例 502
例 502
SEARCHR 503
例 503
SECS 503
例 504
SECSTODATE 504
SECSTODAYS 505
SIGN 505
SIGNED 505
SIN 506
SIND 506
SINF 506
SINH 506
SIZE 507
例 508
SOURCEFILE 508
SOURCELINE 508
SQRT 508
SQRTF 509
STORAGE 509
STRING 509
STRING 疑似変数 510
SUBSTR 511
SUBSTR 疑似変数 511
SUBTRACT 512
SUCC 512
SUM 512
SYSNULL 512
SYSTEM 513
TALLY 513
例 513
TAN 514
TAND 514
TANF 514
TANH 514
THREADID 515
TIME 515
TINY 515
TRANSLATE 516
例 516

TRIM 517
例 517
TRUNC 517
TYPE 518
TYPE 疑似変数. 518
UNALLOCATED 518
UNSIGNED 519
UNSPEC 519
UNSPEC 疑似変数. 521
例 521
UPPERCASE. 521
VALID 522
VALIDDATE 522
例 522
VARGLIST 523
VARGSIZE 523
VERIFY 524
例 524
VERIFYR. 524
例 525
WCHARVAL 525
WEEKDAY 525
WHIGH 526
WIDECHAR 526
WLOW 527
XMLCHAR 527
例 528
Y4DATE 528
Y4JULIAN 529
Y4YEAR 530

多くの共通タスクが組み込み関数、サブルーチン、および疑似変数の形で使用可能です。これらの共通タスクを使用することにより、作成するコードの数が減り、信頼性の高い迅速な処理ができます。

この章では、組み込み関数、サブルーチン、および疑似変数をアルファベット順にリストしています。一般的に、次のようなフォーマットで記述します。
- 参照構文を示す見出し
- 戻される値の説明、または、疑似変数の値のセット
- 引数の説明
- 関数または疑似変数の使用時のその他の修飾

組み込み関数の省略形には、宣言 (明示的またはコンテキスト) と名前の有効範囲が分かれています。次に例を示します。

```
dcl (Dim, Dimension) builtin;
```

上記は、多重宣言ではありません。

```
dcl Binary file;
X = Bin (var, 6,3);
```

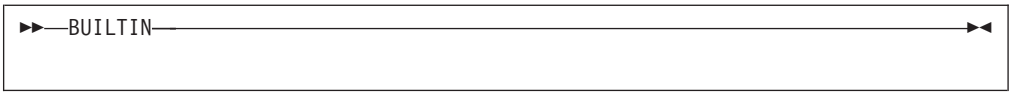
上記は、*Bin* が *Binary* 組み込み関数の省略形であっても、有効です。

組み込み関数、疑似変数、および組み込みサブルーチンの宣言および呼び出し

組み込み関数、疑似変数、およびサブルーチンは、内容に従って宣言するか明示的に宣言することができます。

BUILTIN 属性

BUILTIN 属性は、その名前が組み込み関数、疑似変数、またはサブルーチンであることを指定します。



組み込まれた名前は、プログラマー定義の名前として使用できます。プログラマー定義の宣言または同じ名前の使用を、含まれているブロックから継承している、どのブロック内でも、組み込まれた名前に対して BUILTIN を宣言できます。以下に BUILTIN 属性を持つ組み込み名の例を示します。

例 1

```
1   A: procedure;
       declare Sqrt float binary;
2      X = Sqrt;

3     B:  Begin;
         Declare Sqrt builtin;
         Z = Sqrt(P);
       end B;

     end A;
```

次に例を示します。

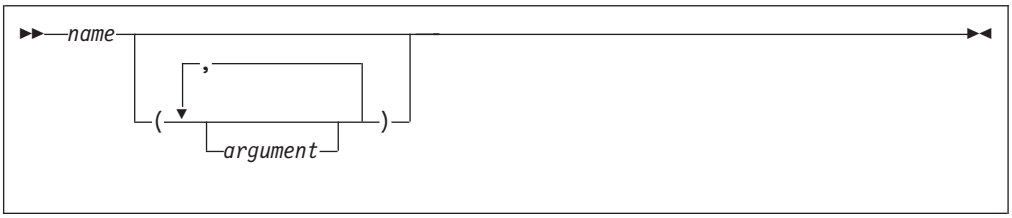
1 Sqrt は、プログラマー定義の名前です。

2 変数 X への割り当ては、プログラマー定義の名前 Sqrt への参照です。

3 Sqrt は、B 内の Sqrt への参照が組み込み関数への参照で、**1** で宣言されたプログラマー定義の名前 Sqrt でないことが認識されるように、BUILTIN 属性によって宣言されます。

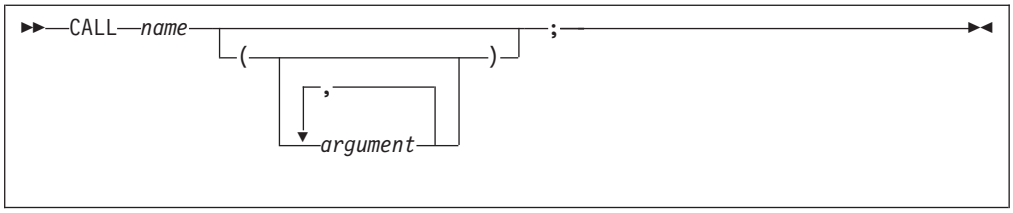
組み込み関数および疑似変数の呼び出し

組み込み関数および疑似変数の呼び出しに使われる構文は、次のとおりです。



組み込みサブルーチンの呼び出し

組み込みサブルーチンの呼び出しに使われる構文は、次のとおりです。



組み込み関数、疑似変数、および組み込みサブルーチンの引数の指定

引数は式で表すことができ、評価、およびデータ変換規則に従って組み込み関数に適したデータ・タイプに変換されます。

引数の集合

引数を持つすべての組み込み関数および疑似変数は、配列引数を持つことができます (複数の引数が配列の場合は、境界は同一でなければなりません)。

- ADDR、ALLOCATION、CURRENTSIZE、SIZE、STRING、および配列処理組み込み関数は、エレメント値を戻します。
- コンパイラー・オプション USAGE(UNSPEC(ANS)) の下では、UNSPEC はエレメント値を戻します。USAGE(UNSPEC(IBM)) は値の配列を戻します。
- その他のすべての関数は、値の配列を戻します。

配列引数の指定は、1 つ以上の引数が制御変数によって修正される添え字付き配列参照である DO グループに、関数参照または疑似変数を指定するのと同等です。

以下に例を示します。

```
dcl A(2) char(2) varying;
dcl B(2) char(2)
   init('AB','CD');
dcl C(2) fixed bin
   init(1,2);
A=substr(B,1,C);
```

結果は、A(1) には値 A が入り、A(2) には値 CD が入ります。

構造体または共用体の引数を受け入れることができる組み込み関数および疑似変数は、ADDR、ALLOCATION、CURRENTSIZE、SIZE、STRING、および UNSPEC です。 UNSPEC を構造体または共用体に適用できるのは、コンパイラー・オプション USAGE(UNSPEC((ANS)) が有効な場合のみです。

ヌル引数およびオプショナル引数

組み込み関数には、引数が必要ないものもあります。それらを BUILTIN 属性によって明示的宣言するか、参照にリストされているヌル引数 (たとえば、ONCHAR()) を内容に従って宣言する必要があります。それ以外の場合は、名前は組み込み関数として認識されません。

数学関数の精度

結果の精度は、次の 2 つの要因に影響されます。
- 引数の精度
- アルゴリズムの精度

ほとんどの引数には、エラーが含まれています。与えられた引数のエラーは、関数を評価する前に、いくつかのステップで累積されていることがあります。入力変換による新しいデータにもエラーが含まれていることがあります。結果の精度に対する引数のエラーの影響は、結果を計算するアルゴリズムではなく、すべて数学関数の性質によって決まります。本書では、このタイプの引数エラーは、説明しません。

インライン機械命令を使って組み込まれた数学組み込み関数では、異なる精度の結果が生成されます。

組み込み関数のカテゴリー

以降の節に、組み込み関数、サブルーチン、および疑似変数をリストしてあります。

これらの表には、完全な関数名だけがリストされています。既存の省略形は、各組み込み関数、サブルーチン、および疑似変数の説明の節に記述されています。

算術組み込み関数

算術組み込み関数により、次のことが可能になります。
- 算術値の特性を判別する。たとえば、SIGN 関数は、算術変数の符号を示します。
- ルーチンの算術演算を実行する。

表 47 には、算術組み込み関数とそれぞれの簡単な説明をリストしています。

算術関数には、1 つまたは複数の引数から結果のデータ・タイプを派生するものもあります。引数のデータ・タイプが異なるときは、77 ページの『第 5 章 データ変換』で説明されているように変換されます。

表 47. 算術組み込み関数

関数	説明
ABS	値の絶対値を計算する。
CEIL	ある値より大きいか等しい最小の整数を計算する。
COMPLEX	与えられた実数および虚数部分によって複素数を戻す。
CONJG	ある値の共役複素数を戻す。
FLOOR	ある値より小さいか等しい最大の整数を計算する。
IMAG	複素数の虚数部分を戻す。
MAX	2 つまたはそれ以上の値の最大値を計算する。
MIN	2 つまたはそれ以上の値の最小値を計算する。
MOD	ある値で別の値を除算したときの剰余のモジュラー等価を戻す。
RANDOM	類似ランダム値を戻す。

表 47. 算術組み込み関数 (続き)

関数	説明
REAL	複素数の実数部分を戻す。
REM	ある値を別の値で除算したときの剰余を計算する。
ROUND	指定された桁で値を丸める。
SIGN	値が負、ゼロ、または正の場合に、それぞれ -1, 0 または 1 を戻す。
TRUNC	ある値を切り捨ててできる整数を計算する。

配列処理組み込み関数

配列処理組み込み関数は、配列引数を演算して、1 つのエレメント値を戻します。次の関数に必要な引数の変換は、関数の説明に示されています。 402 ページの表 48 には、配列処理組み込み関数がリストされています。

表 48. 配列処理組み込み関数

関数	説明
ALL	配列のすべてのエレメントのビット単位の「and」を計算する。
ANY	配列のすべてのエレメントのビット単位の「or」を計算する。
DIMENSION	配列のある次元のエレメントの数を戻す。
HBOUND	配列の次元の上限を戻す。
LBOUND	配列の次元の下限を戻す。
POLY	2 つの配列の浮動小数点近似値を戻す。
PROD	配列のすべてのエレメントの積を計算する。
SUM	配列のすべてのエレメントの和を計算する。

バッファー管理組み込み関数

バッファー管理組み込み関数は、アドレスと複数バイトによって指定するストレージ域である、「バッファー」で演算を行います。また、PLIFILL、PLIMOVE、および PLIOVER 組み込みサブルーチンは、バッファーの管理に役立ちます。

表 49 には、バッファー管理組み込み関数がリストされています。

表 49. バッファー管理組み込み関数

関数	説明
COMPARE	2 つのバッファーを比較する。
HEXIMAGE	バッファーの 16 進表示の文字ストリングを戻す。
MEMINDEX	バッファー中にある 1 つのストリングまたはバッファーの位置を検出する。
MEMSEARCH	バッファー内で、あるストリングの 1 つのエレメントの最初のオカレンスを検索する。
MEMSEARCHR	バッファー内でストリングの 1 つのエレメントの最初のオカレンスを右から検索する。
MEMVERIFY	バッファー内で、あるストリングの 1 つのエレメントの最初の非オカレンスを検索する。
MEMVERIFYR	バッファー内でストリングの 1 つのエレメントの最初のオカレンスを右から検索する。
XMLCHAR	構造体に対応する XML をバッファーに書き込む。

条件処理組み込み関数

条件処理組み込み関数を使うと、発生した条件の原因を判別できます。

これらの関数の使用は、次の ON ユニットまたは動的子孫の範囲内でのみ有効です。

- 組み込み関数への条件特定
- 暗黙処置として発生したときの、ERROR 条件または FINISH 条件

その他の用途は、状況に応じて判別されます。

表 50. 条件処理組み込み関数

関数	説明
DATAFIELD	NAME 条件が発生したときのストリングの値を戻す。
ONCHAR	変換条件の原因となる文字の値を戻す。
ONCODE	条件コード値を戻す。
ONCONDCOND	処理中の CONDITION 条件の名前を戻す。
ONCONDID	特定の条件を識別する番号を戻す。
ONCOUNT	未解決の条件の番号を戻す。
ONFILE	条件が発生したファイルの名前を戻す。
ONGSOURCE	変換条件の原因となる漢字ストリングの値を戻す。
ONKEY	条件が発生したレコードのキーを戻す。
ONLOC	条件の原因となるプロシージャーの名前を戻す。
ONSOURCE	変換条件の原因となるストリングの値を戻す。
ONSUBCODE	特定の入出力エラーに関する追加情報を提供する整数値を戻す。
ONWCHAR	変換条件の原因となるワイド文字の値を戻す。
ONWSOURCE	変換条件の原因となるワイド文字ストリングの値を戻す。

日付/時刻組み込み関数

これらの組み込み関数は、日付および時刻の情報を日、秒数、および文字の日付/タイム・スタンプで戻しまたは処理します。これらの組み込み関数のなかには、使われる日付/時刻のパターンを指定できるものがあります。 405 ページの表 51 には、サポートされている日付/時刻組み込み関数をリストしてあります。 406 ページの表 52 には、サポートされている日付/時刻のパターンをリストしてあります。

これらの関数の時間帯と精度は、システムよって決まります。

リリアン形式: リリアン形式という名前は、グレゴリオ暦の作成者である Luigi Lilio の名前から付けられています。このフォーマットでは、グレゴリー・カレンダーの始めからの日数または秒数で日付を表します。この形式は、経過時間を含む計算を実行するのに役立ちます。

リリアン形式では、経過した日を 1582 年 10 月 14 日から数えます。したがって、1 日目は 1582 年 10 月 15 日金曜日です。たとえば、1988 年 5 月 16 日は、リリアン形式では 148138 です。リリアン形式の日付の有効範囲は、1 から 3,074,324 (1582 年 10 月 15 日から 9999 年 12 月 31 日) までです。

経過秒数の場合は、リリアン形式では、1582 年 10 月 14 の 00:00:00 で開始されます。例えば、1582 年 10 月 15 日の 00:00:01 は 86,401 (24*60*60+1) リリアン秒で、1988 年 5 月 16 日の 19:01:01 は 12,799,191,661 リリアン秒です。リリアン形式の秒の有効範囲は、86,400 から 265,621,679,999.999 (9999 年 12 月 31 日 23:59:59:) 秒です。

表 51. 日付/時刻組み込み関数

関数	説明
DATE	YYMMDD のパターンの現在の日付を戻す。
DATETIME	ユーザー指定のパターンまたはデフォルトのパターン YYYYMMDDHHMISS999 で現在の日付と時刻を戻す。
DAYS	日付/時刻パターンのストリングに対応する日を示す数、または今日の日付を示す数を戻す。
DAYSTODATE	日を示す数を日付/時刻パターンのストリングに変換する。
DAYSTOSECS	日を示す数を秒を示す数に変換する。
REPATTERN	1 つのパターンでは日付を保持している値をとり、2 番目のパターンでは日付に変換された値を戻す。
SECS	日付/時刻パターンのストリングに対応する秒数、または今日の日付を示す秒数を戻す。
SECSTODATE	秒数を日付/時刻パターンのストリングに変換する。
SECSTODAYS	秒数を日を示す数に変換する。
TIME	パターン HHMISS999 で現在の時刻を戻す。
VALIDDATE	ストリングに有効な日付があるかどうかを示す。
WEEKDAY	現在の日または指定された DAYS 値に対応する曜日を戻す。
Y4DATE	パターン 'YYMMDD' を使って日付の値をとり、2 桁の年を 4 桁の年にして日付の値を戻す。
Y4JULIAN	パターン 'YYDDD' を使って日付の値をとり、2 桁の年を 4 桁の年にして日付の値を戻す。
Y4YEAR	パターン 'YY' を使って日付の値をとり、2 桁の年を 4 桁の年にして日付の値を戻す。

406 ページの表 52 は、次のフォーマットを使用しています。

YYYY 4 桁の年
YY 2 桁の年
MM 2 桁の月
MMM 3 文字の月 (例 :DEC)
Mmm 3 文字の月 (例 :Dec)
DD 指定された月での 2 桁表示の日付
DDD 指定された年での日数
HH 指定された日での時間数
MI 指定された時間内の分数
SS 指定された分内の秒数
999 指定された秒内のミリ秒の数

注: 3 文字の月パターンでは、大文字と小文字を正確に一致させる必要があります。

HH、MI、SS、または 999 を使用していてもサポートされる唯一のパターンは、YYYYMMDDHHMISS999 のパターンです。

表 52. 日付/時刻のパターン

	4 桁の年	2 桁の年
年から書き始める場合	YYYYMMDDHHMISS999	YYMMDD
	YYYYMMDD	YYMMMDD
	YYYYMMMDD	YYMmmDD
	YYYYMmmDD	YYDDD
	YYYYDDD	YYMM
	YYYYMM	YYMMM
	YYYYMMM	YYMmm
	YYYYMmm	YY
	YYYY	
月から書き始める場合	MMDDYYYY	MMDDYY
	MMMDDYYYY	MMMDDYY
	MmmDDYYYY	MmmDDYY
	MMYYYY	MMYY
	MMMYYYY	MMMYY
	MmmYYYY	MmmYY
日から書き始める場合	DDMMYYYY	DDMMYY
	DDMMMYYYY	DDMMMYY
	DDMmmYYYY	DDMmmYY
	DDDYYYY	DDDYY
DB2 フォーマット	YYYY-MM-DD	
	MM/DD/YYYY	
	DD.MM.YYYY	

日がパターンから省略されている場合、値が 1 であると仮定します。月と日の両方が省略されている場合、両方とも 値が 1 であると仮定します。

MMM を使用しているときは日付を 3 つの大文字で書き、Mmm を使用しているときは最初の文字を大文字で書き後の 2 文字を小文字で書かなければなりません。

浮動小数点の照会組み込み関数

浮動小数点の照会組み込み関数は、指定した浮動小数点変数の引数についての情報を戻します。

表 53. 浮動小数点の照会組み込み関数

関数	説明
EPSILON	1 と次の正数間のスペースを戻す。
HUGE	浮動小数点変数が保持できる最大の正の有限値を戻す。
MAXEXP	指数の最大値を戻す。
MINEXP	指数の最小値を戻す。
PLACES	浮動小数点の値のモデル精度を戻す。
RADIX	浮動小数点の値のモデル基数を戻す。
TINY	浮動小数点変数が保持できる正の最小値を戻す。

浮動小数点演算組み込み関数

浮動小数点演算組み込み関数は、指定した浮動小数点変数で数学演算を実行し、演算結果を戻します。

表 54. 浮動小数点演算組み込み関数

関数	説明
EXPONENT	浮動小数点の値の指数部分を戻す。
PRED	浮動小数点の値の前の次に表示可能な値を戻す。
SCALE	浮動小数点数を基数の整数乗で乗算する。
SUCC	浮動小数点の値の後の次に表示可能な値を戻す。

入出力組み込み関数

入力および出力組み込み関数により、ファイルの現在の状態を判別できます。

表 55. 入出力組み込み関数

関数	説明
COUNT	最後の GET または PUT 時に伝送されたデータ項目の数を戻す。
ENDFILE	ファイルがオープンされていて、ファイルの終わりに到達しているかどうかを示す。
FILEDDINT	指定したファイル属性の値を戻す。
FILEDDTEST	指定したファイルに指定の属性が適用される場合、値 1 を戻す。
FILEDDWORD	指定したファイル属性の文字ストリングを戻す。
FILEID	ファイルのシステム・トークン値を戻す。
FILEOPEN	ファイルがオープンしているかどうかを示す。
FILEREAD	ファイルから読み取る。
FILESEEK	現在のファイル位置を新しい位置に変更する。
FILETELL	現在のファイル位置を示している値を戻す。
FILEWRITE	ファイルに書き込む。
LINENO	プリント・ファイルに関連した現在の行番号を戻す。
PAGENO	プリント・ファイルに関連した現在のページ番号を戻す。
SAMEKEY	同じキーで別のレコードが続くレコードかどうかを示す。

整数演算組み込み関数

整数演算組み込み関数は、整変数で演算を実行し、演算結果を戻します。

表 56. 整数演算組み込み関数

関数	説明
IAND	複数の固定バイナリー値のビット単位の「and」を計算する。
IEOR	2 つの固定バイナリー値のビット単位の「exclusive」を計算する。
INOT	固定バイナリー値のビット単位の「not」を計算する。
IOR	複数の固定バイナリー値のビット単位の「or」を計算する。
ISIGNED	ビット・パターンを変更せずに、整数を符号付き整数にキャストする。
ISLL	固定バイナリー値を「論理的に」左に桁移動する。

表 56. 整数演算組み込み関数 (続き)

関数	説明
ISRL	固定バイナリー値を「論理的に」右に桁移動する。
IUNSIGNED	ビット・パターンを変更せずに、整数を符号なし整数にキャストする。
LOWER2	固定バイナリー値を 2 の整数乗によって除算する。
RAISE2	固定バイナリー値を 2 の整数乗で乗算する。

数学組み込み関数

これらの関数はすべて、浮動小数点の値で演算を行い、浮動小数点の結果を生成します。浮動小数点でない引数は、変換されます。これらの関数の精度については、401 ページの『数学関数の精度』で説明されています。 表 57 に、数学組み込み関数をリストします。

表 57. 数学組み込み関数

関数	説明
ACOS	逆余弦を計算する。
ACOSF	ACOS をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
ASIN	逆正弦を計算する。
ASINF	ASIN をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
ATAN	逆正接を計算する。
ATAND	逆正接を度で計算する。
ATANF	ATAN をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
ATANH	双曲線逆正接を計算する。
COS	余弦を計算する。
COSD	余弦を度で計算する。
COSF	COS をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
COSH	双曲線余弦を計算する。
ERF	誤差関数を計算する。
ERFC	誤差関数の補数を計算する。
EXP	e の累乗を計算する。
EXPF	EXP をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
GAMMA	ガンマ関数を計算する。
LOG	自然対数を計算する。
LOGF	LOG をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
LOG10	底 10 の常用対数を計算する。
LOG10F	LOG10 をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
LOG2	底 2 の対数を計算する。

表 57. 数学組み込み関数 (続き)

関数	説明
LOGGAMMA	ガンマ関数のログを計算する。
SIN	正弦を計算する。
SIND	正弦を度で計算する。
SINF	SIN をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
SINH	双曲線正弦を計算する。
SQRT	平方根を計算する。
SQRTF	SQRT をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
TAN	正接を計算する。
TAND	正接を度で計算する。
TANF	TAN をインラインで計算する (ハードウェア・アーキテクチャーによって処理可能な場合)。
TANH	双曲線正接を計算する。

その他の組み込み関数

これまでのカテゴリーのいずれにも該当しない組み込み関数を 表 58 にリストします。

表 58. その他の組み込み関数

関数	説明
BYTE	CHARVAL の同義語。
CDS	*ダブル比較およびスワップ* の以前の値と現行値が等しいかどうかを示す FIXED BINARY(31) 値を戻す。
CHARVAL	整数に対応する文字値を戻す。
COLLATE	照合順序を指定する文字 (256) ストリングを戻す。
CS	*比較およびスワップ* の以前の値と現行値が等しいかどうかを示す FIXED BINARY(31) 値を戻す。
GETENV	指定した環境変数を表す値を戻す。
HEX	値の 16 進表示の文字ストリングを戻す。
ISMAIN	現行プロシージャーがメインかどうかを示す。
OMITTED	呼び出しでパラメーターが指定されていないかどうかを示す。
PACKAGENAME	包含パッケージの名前を戻す。
PLIRETV	PL/I の戻りコード値を戻す。
PRESENT	呼び出しでパラメーターが提供されていないかどうかを示す。
PROCEDURENAME	ネストされた最も近いプロシージャーの名前を戻す。
PUTENV	新しい環境変数を追加するか、または既存の環境変数の値を変更する。
RANK	文字、あるいはワイド文字に対応する整数値を戻す。
SOURCEFILE	ステートメントが含まれているファイルの名前を戻す。
SOURCELINE	ステートメントが含まれている行の番号を戻す。

表 58. その他の組み込み関数 (続き)

関数	説明
STRING	ストリングの集合のすべてのエレメントの連結であるストリングを戻す。
SYSTEM	コマンドによって戻される値を戻す。
THREADID	タスクのスレッド ID を戻す。
UNSPEC	値の内部表示のビット・ストリングを戻す。
VALID	変数の内容が宣言に有効であるかどうかを示す。
WCHARVAL	整数に対応するワイド文字値を戻す。

序数処理組み込み関数

序数処理データ組み込み関数は、指定された序数に関する情報を戻します。

表 59. 序数処理組み込み関数

関数	説明
ORDINALNAME	序数の名前を指定する文字ストリングを戻します。
ORDINALPRED	序数が次に低い値を戻します。
ORDINALSUCC	次に高い序数値を戻します。

精度処理組み込み関数

精度処理組み込み関数により、指定された精度で変数を演算し、演算結果の値を戻します。

表 60. 精度処理組み込み関数

関数	説明
ADD	指定された精度で、2 つの値を追加する。
BINARY	値を 2 進数に変換する。
DECIMAL	値を 10 進数に変換する。
DIVIDE	指定された精度で、2 つの値を除算する。
FIXED	値を固定に変換する。
FLOAT	値を浮動に変換する。
MULTIPLY	指定された精度で、2 つの値を乗算する。
PRECISION	値を、指定の精度に変換する。
SIGNED	値を符号付き固定 2 進数に変換する。
SUBTRACT	指定された精度で、2 つの値を減算する。
UNSIGNED	値を符号なし固定 2 進数に変換する。

疑似変数

疑似変数は、受信フィールドを表します。疑似変数をネストすることはできません。たとえば、次のものは正しくありません。

```
unspec(substr(A,1,2)) = '00'B;
```

疑似変数が存在できるのは、次のような場合のみです。

- 代入ステートメントの左側。
- DO 指定内のターゲットであり、なおかつ、その疑似変数が SUBSTR、REAL、IMAG または UNSPEC のいずれかである場合。
- GET ステートメントのデータ・リスト内、または PUT ステートメントの STRING オプション内。
- KEYTO オプションまたは REPLY オプション内のストリング名として。

疑似変数には次のものがあります。

表 61. 組み込み疑似変数

関数	説明
ENTRYADDR	呼び出される入り口のアドレスで入り口変数を設定する。
IMAG	複素数の虚数部分を割り当てる。
ONCHAR	変換条件の原因となる文字の値を設定する。
ONGSOURCE	変換条件の原因となる漢字ストリングの値を設定する。
ONSOURCE	変換条件の原因となるストリングの値を設定する。
REAL	複素数の実数部分を割り当てる。
STRING	ストリングの集合のすべてのエレメントの連結であるストリングを割り当てる。
SUBSTR	ストリングのサブストリングを割り当てる。
ONWCHAR	変換条件の原因となるワイド文字の値を設定する。
ONWSOURCE	変換条件の原因となるワイド文字ストリングの値を設定する。
TYPE	タイプ付き構造体または共用体をハンドルによって位置付けられたストレージに割り当てる。
UNSPEC	値の内部表示であるビット・ストリングを割り当てる。

ストレージ制御組み込み関数

ストレージ制御組み込み関数により、ストレージ所要量および変数の位置を判別して、特殊値の区域およびロケーター変数への割り当て、オフセット値とポインター値間の変換の実行、被制御変数の世代番号の入手、およびオブジェクトとクラスのデータおよび手法の参照が可能です。表 62 に、ストレージ制御組み込み関数をリストします。

表 62. ストレージ制御組み込み関数

関数	説明
ADDR	変数のアドレスを戻す。
ADDRDATA	可変ストリングに適用されるときに、ストリングの最初のデータ・バイトのアドレスを戻す。
ALLOCATE	ヒープ内に、指定されたサイズのストレージを割り振る。
ALLOCATION	被制御変数の現在の世代番号を戻す。
ALLOCSIZE	特定のポインターとともに割り振られたストレージ量を示す FIXED BIN(N,0) 値を戻す。
AUTOMATIC	スタック内に、指定されたサイズのストレージを割り振る。
AVAILABLEAREA	区域内で、単一の作成可能な最大割り振りサイズを戻す。

表 62. ストレージ制御組み込み関数 (続き)

関数	説明
BINARYVALUE	ポインター、オフセット、または序数を整数に変換する。
BITLOCATION	バイト内での変数のビット・オフセットを戻す。
CHECKSTG	割り振られていたストレージが破壊されていないかどうかを判別するビット (1) 値を戻す。
CURRENTSIZE	変数の現行サイズを戻す。
CURRENTSTORAGE	CURRENTSIZE の同義語。
EMPTY	「空の」区域を戻す。
ENTRYADDR	入り口に関連したルーチンのアドレスを戻す。
HANDLE	タイプ付き構造体または共用体にハンドルを戻す。
LOCATION	構造体内の変数のバイト・オフセットを戻す。
NULL	ヌル・ポインター値を戻す。
OFFSET	ポインターをオフセットに変換する。
OFFSETADD	整数をオフセットに追加する。
OFFSETDIFF	2 つのオフセットを減算する。
OFFSETSUBTRACT	オフセットから整数を減算する。
OFFSETVALUE	整数をオフセットに変換する。
POINTER	オフセットをポインターに変換する。
POINTERADD	整数をポインターに加える。
POINTERDIFF	2 つのポインターを減算する。
POINTERSUBTRACT	ポインターから整数を減算する。
POINTERVALUE	整数またはハンドルをポインターに変換する。
SIZE	変数の最大サイズを戻す。
STORAGE	SIZE の同義語。
SYSNULL	システム・ヌル・ポインターの値を戻す。
TYPE	ハンドルによって位置付けされるタイプ付き構造体または共用体を戻す。
UNALLOCATED	指定されたポインター値が割り振り済みストレージの開始点であるかどうかを示すビット (1) 値を戻す。
VARGLIST	プロシージャーに渡された最初のオプショナル・パラメーターのアドレスを戻す。
VARGSIZE	byvalue パラメーターが占めるバイト数を戻す。

ストリング処理組み込み関数

ストリング処理組み込み関数は、ビット・ストリング、文字ストリング、漢字ストリング、およびワイド文字ストリングの処理を簡素化するものです。ストリングの引数は、データ変換規則、または関数の記述で与えられた規則のどちらかに従ったストリングに変換される算術式で表されます。

注: LOWERCASE、TRANSLATE、TRIM、または UPPERCASE など、CHARACTER データのみをサポートする関数もあります。

表 63. ストリング処理組み込み関数

関数	説明
BIT	値をビットに変換する。
BOOL	2 ビット・ストリングでブール演算を実行する。
CENTERLEFT	値が中央から左方向に配置されたストリングを戻す。
CENTERRIGHT	値が中央から右方向に配置されたストリングを戻す。
CENTRELEFT	CENTERLEFT の同義語。
CENTRERIGHT	CENTERRIGHT の同義語。
CHARACTER	値を文字ストリングに変換する。
CHARGRAPHIC	GRAPHIC ストリングを混合文字ストリングに変換する。
COPY	ストリングのコピーが n 個含まれるストリングを戻す。
EDIT	与えられたピクチャー指定に変換された値で構成されるストリングを戻す。
GRAPHIC	値をグラフィックに変換する。
HIGH	照合順序の最も高い文字を n 回コピーして構成される文字ストリングを戻す。
INDEX	ストリング中にある 1 つのストリングの位置を検出する。
LEFT	左詰めの値が入っているストリングを戻す。
LENGTH	ストリングの現行の長さを戻す。
LOW	照合順序の最も低い文字を n 回コピーして構成される文字ストリングを戻す。
LOWERCASE	A から Z までの文字すべてを対応する小文字に変換した文字ストリングを戻す。
MAXLENGTH	ストリングの最大長を戻す。
MPSTR	論理境界でストリングを切り捨て、混合文字ストリングを戻す。
PICSPEC	与えられたピクチャー指定を持つと想定される値で構成されるストリングを戻す。
REPEAT	n+1 回コピーしたストリングを戻す。
REPLACEBY2	一部の文字が、文字のペアに「変換された」ストリングを戻す。
REVERSE	ストリングの反転表示を戻す。
RIGHT	右寄せの値が入っているストリングを戻す。
SEARCH	別のストリング内で、あるストリングの 1 つのエレメントの最初のオカレンスを検索する。
SEARCHR	別のストリング内でストリングの 1 つのエレメントの最初のオカレンスを右から検索する。
SUBSTR	ストリングのサブストリングを割り当てる。
TALLY	ストリングで 1 つのストリングが発生する回数を戻す。
TRANSLATE	2 つの変換ストリングに基づいてストリングを変換する。
TRIM	指定された文字をストリングの左および右からトリムする。
UPPERCASE	a から z までの文字すべてを対応する大文字に変換した文字ストリングを戻す。
VERIFY	別のストリング内でストリングの 1 つのエレメントの最初の非オカレンスを検索する。

表 63. ストリング処理組み込み関数 (続き)

関数	説明
VERIFYR	別のストリング内でストリングの 1 つのエレメントの最初の非オカレンスを右から検索する。
WHIGH	最上位のワイド文字 ('ffff'wx) を n 回コピーして構成されるワイド文字ストリングを戻す。
WIDECHAR	値をワイド文字ストリングに変換する。
WLOW	最下位のワイド文字 ('0000'wx) を n 回コピーして構成されるワイド文字ストリングを戻す。

サブルーチン

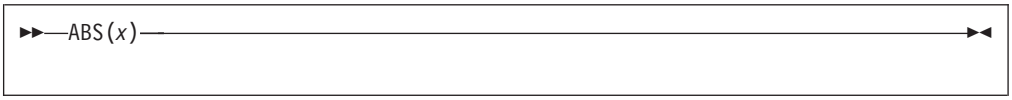
組み込みサブルーチンは、組み込み関数が結果を戻すのに対して、結果を戻す必要がないその他の演算を実行します。

表 64. 組み込みサブルーチン

関数	説明
PLIASCII	EBCDIC から ASCII に変換する。
PLIDELETE	ハンドルに関連したストレージを解放する。
PLICANC	自動再始動機能を取り消し (z/OS のみ)
PLICKPT	実行時リスタートのためのチェックポイント (z/OS のみ)
PLIDUMP	現在のオープン・ファイル、現在場所への呼び出しパスなどについての情報をダンプする。
PLIEBCDIC	ASCII から EBCDIC に変換する。
PLIFILL	指定されたバイト値によって、アドレスに n バイトを埋める。
PLIFREE	ヒープ・ストレージを指すポインターに関連したストレージを解放する。
PLIMOVE	1 つのアドレスから別のアドレスへ n バイト移動する。
PLIOVER	ソースとターゲットの、可能性のあるオーバーラップを補いながら、1 つのアドレスから別のアドレスへ n バイト移動する。
PLIREST	プログラム実行を再始動 (z/OS のみ)
PLIRETC	PL/I 戻りコード値を設定する。
PLISAXA	プログラムのバッファー内にある XML 文書に対して、SAX フォーマットの構文解析を実行する。
PLISAXB	ファイル内にある XML 文書に対して、SAX フォーマットの構文解析を実行する。
PLISRTA	DFSORT を使用することにより、入力ファイルをソートして、ソートされた出力ファイルの作成を可能にする。
PLISRTB	DFSORT を使用することにより、E15 PL/I 終了プロシージャーで提供される入力レコードをソートして、ソートされた出力ファイルの作成を可能にする。
PLISRTC	DFSORT を使用することにより、入力ファイルをソートして、E35 PL/I 終了プロシージャーで処理されるソートされたレコードの作成を可能にする。
PLISRTD	DFSORT を使用することにより、E15 PL/I 終了プロシージャーで提供される入力レコードをソートして、E35 PL/I 終了プロシージャーで処理されるソートされたレコードの作成を可能にする。

ABS

ABS は、*x* の絶対値を戻します。ABS は、*x* の正の値です。

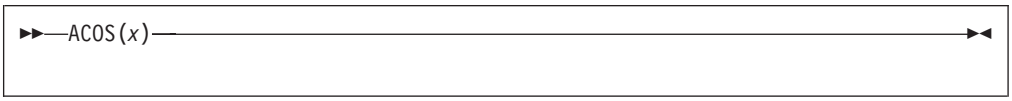


x 式。

結果は、REAL モードです。結果には、*x* の基数、スケール、および精度が含まれています。ただし、x が COMPLEX FIXED(p,q) の場合は除きます。 x が COMPLEX FIXED(p,q) の場合、結果は REAL FIXED(min(n,p+1),q) になります。n には、DECIMAL では N が、BINARY では M が入ります。

ACOS

ACOS は、*x* ラジアンの逆余弦の近似を実数の浮動小数点の値で戻します。



x 実数式。ABS(x) <= 1

結果は、次の範囲内の値です。

```
0 ≤ ACOS(x) ≤ π
```

また、結果は、*x* の基数と精度になります。

ACOSF

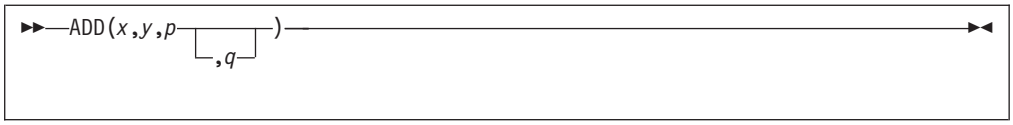
ACOSF は、次の点を除いては ACOS とまったく同じです。

- ACOSF は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 無効な引数が指定された場合、INVALIDOP 条件が発生したり、その他のハードウェア例外が生成されたりするなど、予期できない結果が生じることがある。
- 結果の正確度は、ハードウェアによって設定される。

定義および構文については、『ACOS』を参照してください。

ADD

ADD は、*x* と *y* の和を *p* と *q* で指定された精度で戻します。結果の基数、スケール、およびモードは、PRECTYPE コンパイラー・オプションによって規則が変更されないかぎり、式の評価規則によって決まります。



x および y
式。

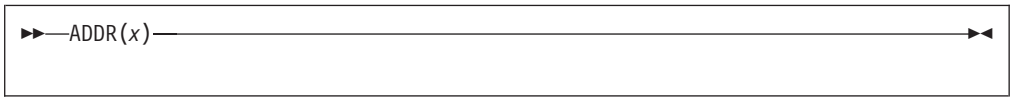
p 制限付き式。演算を通して維持される桁数を指定します。

q 結果のスケール因数を指定する制限付き式。結果が固定小数点の場合に、*q* が省略されるときは、スケール因数ゼロがデフォルトです。結果が浮動小数点の場合は、*q* は省略します。

ADD は、減算されるオペランドの接頭部に負符号 (-) を付けて、減算に使用できます。

ADDR

ADDR は、*x* の世代を識別するポインター値を戻します。



x 参照。 ADDR は、次のもの以外のデータ・タイプ変数、データ編成変数、位置合わせ変数、およびストレージ・クラス変数を参照します。

- 位置合わせされていない固定長のビット・ストリングである変数への添え字付き参照。
- DEFINED 変数または BASED 変数、または位置合わせされていない固定長のビット・ストリングの単純パラメーターへの参照。
- 最初の基本エレメントが位置合わせされていない固定長のビット・ストリングである小構造体または共用体 (ただし、その小構造体を含んでいる大構造体または共用体の最初のエレメントになっている場合を除く)。
- DEFINED 属性を持つ大構造体か共用体、またはパラメーターである大構造体か共用体で、位置合わせされていない固定長のビット・ストリングを最初のエレメントとするもの。
- 連結ストレージを参照しない参照。

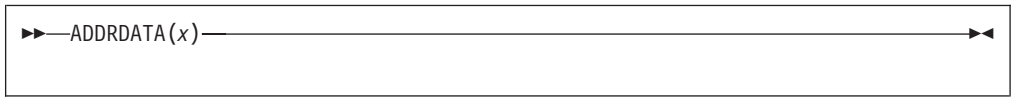
x の参照先と結果を次に示します。

- 集合パラメーターの場合は、CONNECTED 属性を保持している必要があります。
- 集合の場合は、戻り値は最初のエレメントを識別します。
- コンポーネントまたはセクション間の集合の場合は、戻り値では添え字および構造体または共用体の修飾が考慮されます。
- 可変ストリングの場合は、戻り値は 2 バイトの接頭部を識別します。
- 区域。戻り値は、制御情報を示しています。
- 現行プログラムに割り振られていない被制御変数の場合は、ヌル・ポインター値が戻されます。

- 基底付き変数の場合は、結果は明示的に *x* を修飾するポインター (表示される場合) か、*x* の宣言 (存在する場合) で *x* に関連付けられたポインター、またはヌル・ポインターの値です。
- パラメーターであり、仮引数が作成されている場合は、戻り値は仮引数を識別します。

ADDRDATA

ADDRDATA は *x* の世代を識別するポインター値を戻します。



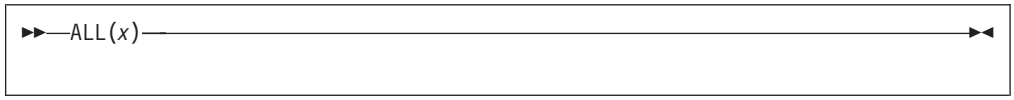
x 参照。

ADDRDATA は、以下のインスタンスを除き、ADDR 組み込み関数と同じです。

- 可変ストリングに適用されるときに、ADDRDATA は、(長さフィールドの最初のデータ・バイトのアドレスではなく) ストリングの最初のデータ・バイトのアドレスを戻す。

ALL

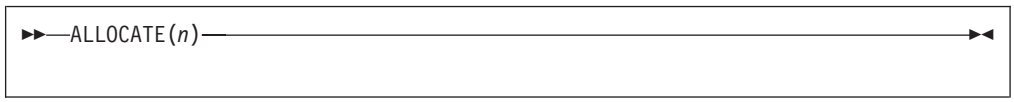
ALL は、*x* の各エレメントに対応するビットがあり、1 の場合は、各ビットが 1 のビット・ストリングを戻します。結果の長さは、最も長いエレメントと同じです。



x 計算配列式。 *x* がビット・ストリング配列ではない場合、*x* はビット・ストリングに変換されます。

ALLOCATE

ALLOCATE は、ヒープ・ストレージにサイズ *n* のストレージを割り振り、割り振り済みストレージにポインターを戻します。



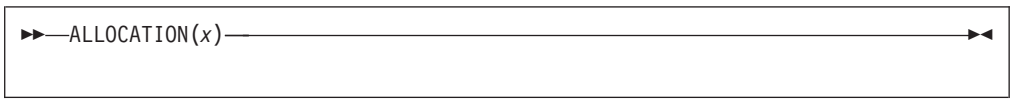
省略形: ALLOC

n 式。 *n* は、負以外でなければなりません。必要な場合には、*n* は、REAL FIXED BINARY(31,0) に変換されます。

ストレージで要求量が使用可能でない場合は、STORAGE 条件が発生します。

ALLOCATION

ALLOCATION は、*x* の現行プログラムでアクセス可能な世代番号を指定する FIXED BINARY(31,0) を戻します。



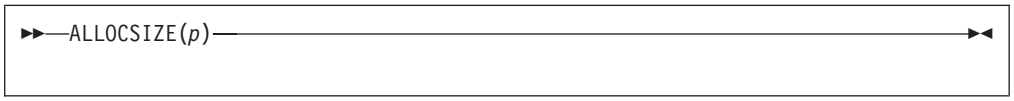
省略形: ALLOCN

x レベル 1 の添え字なし被制御変数です。

現行プログラムで *x* が割り振られていない場合は、結果はゼロになります。

ALLOCSIZE

ALLOCSIZE は、指定されたポインターとともに割り振られたストレージ量を与えている FIXED BIN(31,0) 値を戻します。この組み込み関数を使用するには、CHECK(STORAGE) コンパイル時オプションも指定しなければなりません。

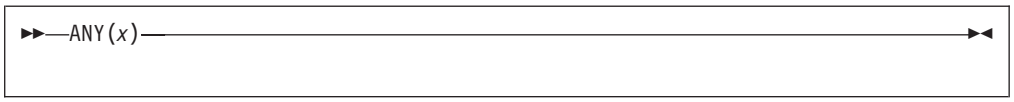


p ポインターの式。

ポインターが割り振り済みストレージの始まりを示していない場合、ALLOCSIZE は 0 を戻します。

ANY

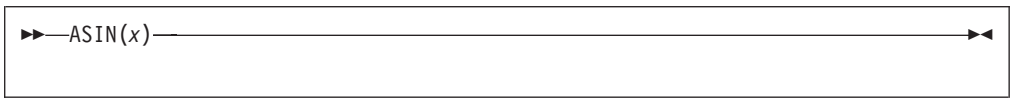
ANY は、*x* の任意のエレメントの対応するビットがあり、1 の場合は、各ビットが 1 のビット・ストリングを戻します。結果の長さは、最も長いエレメントと同じです。



x 計算配列式。 *x* がビット・ストリング配列ではない場合、*x* はビット・ストリングに変換されます。

ASIN

ASIN は、*x* ラジアンでの逆正弦の近似を実数の浮動小数点の値で戻します。



x 実数式。ABS(x) <= 1

結果は、次の範囲内の値です。

```
-π/2 ≤ ASIN(x) ≤ π/2
```

また、結果は、*x* の基数と精度になります。

ASINF

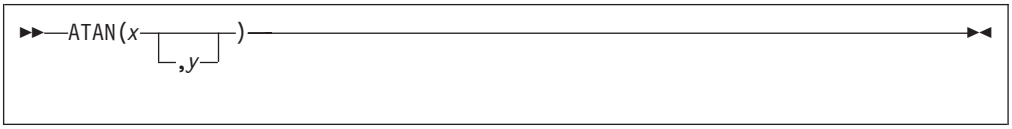
ASINF は、次の点を除いては ASIN とまったく同じです。

- ASINF は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 無効な引数が指定された場合、INVALIDOP 条件が発生したり、その他のハードウェア例外が生成されたりするなど、予期できない結果が生じることがある。
- 結果の正確度は、ハードウェアによって設定される。

定義および構文については、 418 ページの『ASIN』を参照してください。

ATAN

ATAN は、*x* ラジアンまたは比率 *x/y* の逆正接の近似を浮動小数点の値で戻します。



x および y
式。

x が単独で指定されている場合は、結果は *x* の基数と精度になり、次の範囲内の値です。

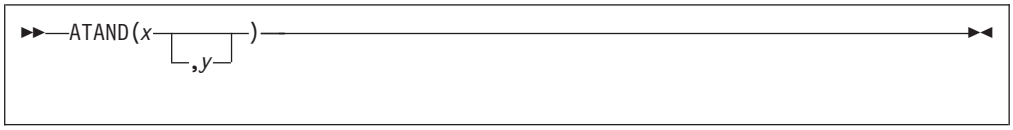
```
-π/2 < ATAN(x) < π/2
```

x と *y* が指定される場合は、それぞれ実数でなければなりません。 *x* と *y* が両方ともゼロの場合は、エラーになります。 *x* と *y* のその他のすべての値の結果には、長い方の引数の精度、式の規則で決まる基数、および次の式で指定される値が含まれます。

ATAN(x/y)	y>0 の場合
π/2	y=0 および x>0 の場合
-π/2	y=0 および x<0 の場合
π + ATAN(x/y)	y<0 および x>=0 の場合
-π + ATAN(x/y)	y<0 および x<0 の場合

ATAND

ATAND は、*x* ラジアンまたは比率 *x/y* の逆正接の近似を実数の浮動小数点の値で戻します。



x および y
式。

x が単独で指定される場合は、*x* は実数でなければなりません。結果は、*x* の基数と精度になり、次の範囲内の値です。

```
-90 < ATAND(x) < 90
```

x と *y* が指定される場合は、それぞれ実数でなければなりません。結果の値は、次の式で指定されます。

```
(180/π) * ATAN(x,y)
```

引数の要件および結果の属性については、419 ページの『ATAN』を参照してください。

ATANF

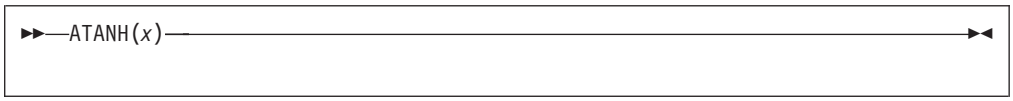
ATANF は、次の点を除いては ATAN とまったく同じです。

- ATANF は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 実数の引数を 1 つだけ指定できる。
- 無効な引数が指定された場合、INVALIDOP 条件が発生したり、その他のハードウェア例外が生成されたりするなど、予期できない結果が生じることがある。
- 結果の正確度は、ハードウェアによって設定される。

定義および構文については、419 ページの『ATAN』を参照してください。

ATANH

ATANH は、*x* の基数、モード、および精度を保持する浮動小数点数で、 *x* の双曲線逆正接の近似値を戻します。



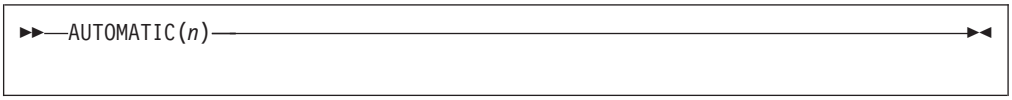
x　式。 ABS(x)<1

結果は、次の式で指定される値になります。

```
LOG((1 + x)/(1 - x))/2
```

AUTOMATIC

AUTOMATIC は、自動ストレージにサイズ *n* のストレージを割り振り、割り振り済みストレージにポインターを戻します。



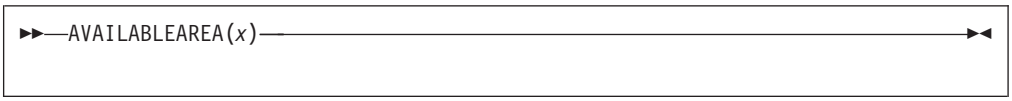
省略形: AUTO

n 式。 *n* は、負以外でなければなりません。必要な場合には、*n* は、REAL FIXED BINARY(31,0) に変換されます。

獲得されたストレージを明示的に解放することはできません。ストレージは、そのブロックが終了したときに自動的に解放されます。

AVAILABLEAREA

AVAILABLEAREA は、FIXED BINARY(31,0) 値を戻します。 AVAILABLEAREA によって戻される値は、区域 *x* から入手可能な最も大きい単一割り振りサイズです。



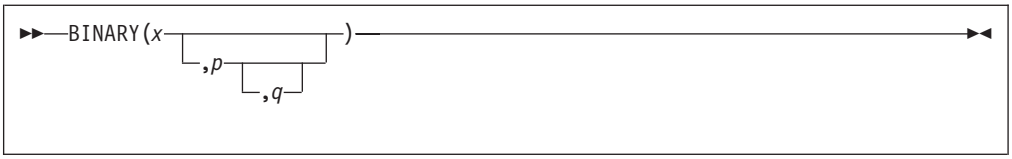
x AREA 属性付き参照。

例

```
dcl Uarea area(1000);
dcl Pz ptr;
dcl C99z char(99) varyingz based(Pz);
dcl (SizeBefore, SizeAfter) fixed bin(31);
SizeBefore = availablearea(Uarea);         /* returns 1000    */
Alloc C99z in(Uarea);
SizeAfter = availablearea(Uarea);          /* returns 896     */
dcl C9 char(896) based(Pz);
Alloc C9 in(Uarea);
```

BINARY

BINARY は、*p* および *q* で指定される精度で、*x* の 2 進数の値を戻します。結果には、*x* のモードとスケールが含まれています。



省略形: BIN

x 式。

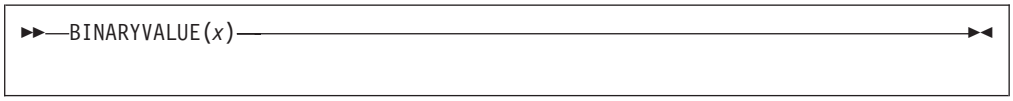
p 制限付き式。演算を通して維持する桁数を指定します。実行の限界を超えることはできません。

q 制限付き式。結果のスケール因数を指定します。固定小数点の結果の場合は、*p* が指定され、*q* が省略されると、スケール因数ゼロがデフォルトになります。結果が浮動小数点の場合は、*q* は省略します。

p と *q* が両方とも省略される場合は、結果の精度は基数変換の規則により決まります。

BINARYVALUE

BINARYVALUE は、*x* の変換された値である FIXED BINARY(31,0) 値を戻します。*x* は、ポインター、オフセット、または序数です。

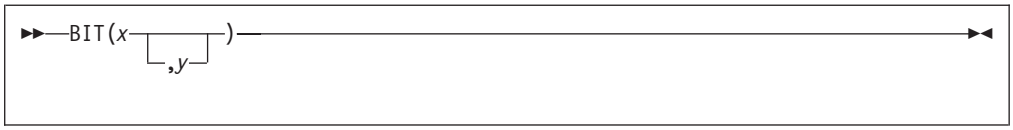


省略形: BINVALUE

x 式。

BIT

BIT は、*x* のビット値である結果を戻し、*y* で指定される長さをとります。

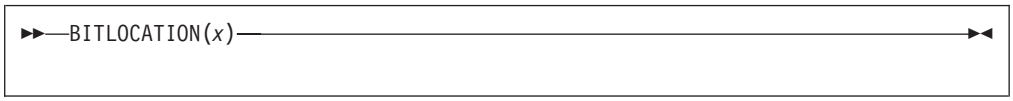


x 式。

y 式。必要な場合には、*y* は、実数の固定小数点の 2 進数の値に変換されます。*y* が省略される場合は、長さはタイプ変換の規則によって決められます。 *y* = 0 の場合は、結果はヌルのビット・ストリングです。 *y* は、負以外でなければなりません。

BITLOCATION

BITLOCATION は、*x* を含むバイト内で、ビット *x* の位置である FIXED BINARY(31,0) の結果を戻します。戻り値は、必ず、0 から 7 の間 (0 ≤ 値 ≤ 7) です



省略形: BITLOC

x 位置合わせされていないビット・タイプの参照。 *x* に位置合わせがされていないビット・タイプがある場合は、値 0 が戻されます。

x は、添え字であってはいけません。

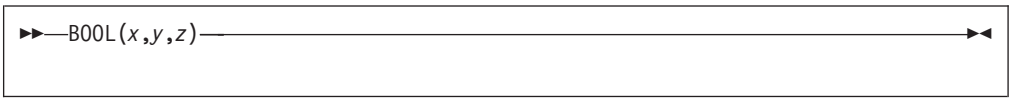
BITLOCATION は、制限付きの式で使用できます。制限は次のとおりです。
*BITLOC(x) が
- 定数エクステントを保持する必要がある変数 *y* のエクステント
- または、定数値を保持する必要がある変数 *y* の値

BITLOC(x) を使用して上記の制限を設定する場合は、*y* の前に *x* を宣言する必要があります。

例については、459 ページの『LOCATION』を参照してください。

BOOL

BOOL は、*x* と *y* のブール演算 *z* の結果であるビット・ストリングを戻します。結果の長さは、長いオペランド *x* または *y* と同じです。



x および y
式。必要な場合には、*x* および *y* は、ビット・ストリングに変換されます。 *x* および *y* の長さが異なる場合は、長さが同じになるように短い方の右側がゼロで埋められます。

z 式。必要な場合には、*z* は、長さ 4 のビット・ストリングに変換されます。 *x* からのビットが *y* からのビットと一致するときは、結果の対応するビットは *z* のビットによって次のように選択されます。

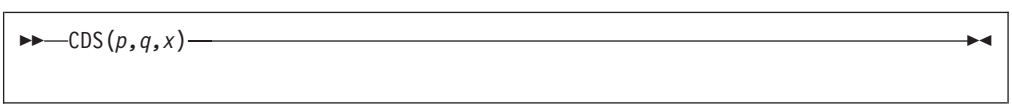
x	y	結果
0	0	z のビット 1
0	1	z のビット 2
1	0	z のビット 3
1	1	z のビット 4

BYTE

BYTE は、CHARVAL の同義語です。詳細については、427 ページの『CHARVAL』を参照してください。

CDS

CDS は、*ダブル比較およびスワップ* の以前の値と現行値が等しいかどうかを示す FIXED BINARY(31) 値を戻します。



p 以前の FIXED BINARY(63) 値のアドレス。

q 現行の FIXED BINARY(63) 値のアドレス。

x 新しい FIXED BINARY(63) 値。

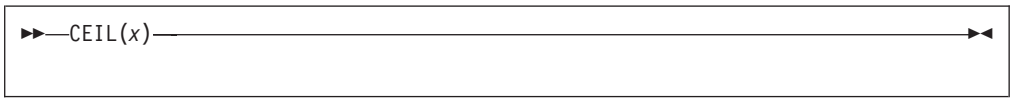
CDS は、「現行」値と「以前の」値を比較します。これらの値が等しい場合は、「現行」値を上書きして「新しい」値がコピーされ、値 0 が戻されます。これらの値が等しくない場合は、「以前の」値を上書きして「現行」値がコピーされ、値 1 が戻されます。

z/OS 上では、CDS 組み込み関数は CDS 命令をインプリメントしています。この関数について詳しくは、「*Principles of Operations*」の付録を参照してください。

Intel 上では、CDS 組み込み関数は Intel cmpxchg8 命令を使用します。同じように、CS 組み込み関数は cmpxchg4 命令を使用します。

CEIL

CEIL は、*x* より大きいか等しい最も小さい整数値を判別し、結果にこの値を割り当てます。



x 実数式。

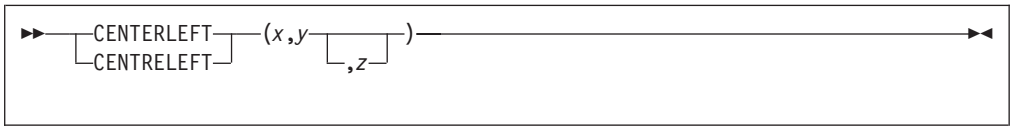
結果は、*x* のモード、スケール、および精度になります。ただし *x* が精度 (*p,q*) の固定小数点のとき、結果の精度は、次の式で表されます。

```
(min(N,max(p-q+1,1)),0)
```

この場合 N は、使用できる最大の桁数です。

CENTERLEFT

CENTERLEFT は、長さ y で必要な場合に左側と右側が文字 *z* で埋められたストリングの中央 (または中央から 1 つ左よりの位置) に、ストリング *x* を挿入した結果のストリングを戻します。 *z* の値の指定は、オプショナルです。



省略形: CENTER

x 文字に変換される式。

y FIXED BINARY(31,0) に変換される式。

z オプションの式。 *z* が指定された場合は、CHARACTER(1) NONVARYING タイプでなければなりません。

例

```
dcl Source char value('Feel the Power');
dcl Target20 char(20);
dcl Target21 char(21);

Target20 = center (Source, length(Target20), '*');
           /* '***Feel the Power***' - exactly centered */


Target21 = center (Source, length(Target21), '*');
           /* '***Feel the Power****' - leaning left! */
```

z が省略される場合は、埋め込み文字にブランクが使われます。

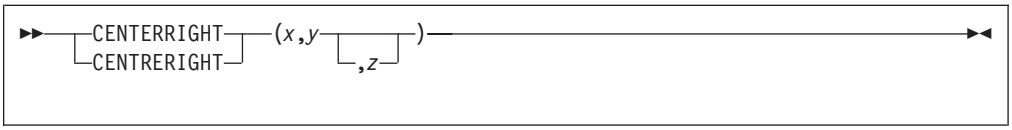
CENTRELEFT

省略形: CENTRE

CENTRELEFT は、CENTERLEFT の同義語です。

CENTERRIGHT

CENTERRIGHT は、長さ *y* で必要な場合に左側と右側が文字 *z* で埋められたストリングの中央 (または中央より 1 つ右よりの位置) に、ストリング *x* を挿入した結果のストリングを戻します。 *z* の値の指定は、オプショナルです。



x 文字に変換される式。

y FIXED BINARY(31,0) に変換される式。

z オプションの式。 *z* が指定された場合は、CHARACTER(1) NONVARYING タイプでなければなりません。

例

```
dcl Source char value('Feel the Power');
dcl Target20 char(20);
dcl Target21 char(21);

Target20 = centerright (Source, length(Target20), '*');
           /* '***Feel the Power***' - exactly centered */

Target21 = centerright (Source, length(Target21), '*');
           /* '****Feel the Power***' - leaning right! */
```

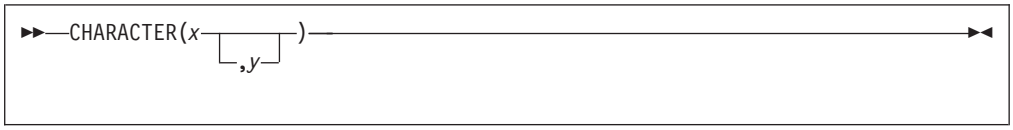
z が省略される場合は、埋め込み文字にブランクが使われます。

CENTRERIGHT

CENTRERIGHT は、CENTERRIGHT の同義語です。

CHARACTER

CHARACTER は、*y* で指定された長さで文字値 *x* を戻します。また、CHARACTER は、グラフィックから文字タイプへの変換もサポートします。



省略形: CHAR

x 式。

n は、計算タイプでなくてはなりません。

x がグラフィックでない場合は、CHARACTER は 文字に変換された *x* を戻します。

x が GRAPHIC のときは、CHARACTER は SBCS 文字に変換された *x* を戻します。 DBCS 文字が SBCS 等価なものに変換できない場合は、CONVERSION 条件が発生します。

x の値は、チェックされません。

y 式。必要な場合には、y は、実数の固定小数点の 2 進数の値に変換されます。

y が省略される場合は、長さはタイプ変換の規則によって決められます。

y は、負以外でなければなりません。

y = 0 の場合は、結果はヌル文字ストリングです。

例

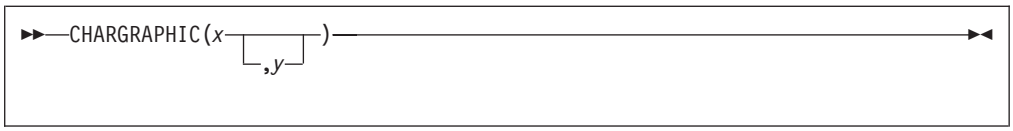
GRAPHIC から文字への変換

```
dcl X graphic(6);
dcl A char (6);
A = char(X);
```

X の値	中間結果	A に割り当てられる値
.A.B.C.D.E.F	ABCDEF	ABCDEF

CHARGRAPHIC

CHARGRAPHIC は、GRAPHIC (DBCS) ストリング *x* を *y* で指定された長さで混合文字ストリングに変換します。



省略形: CHARG

x 式。

x は、GRAPHIC ストリングでなければなりません。 CHARACTER は、混合文字ストリングに変換される *x* を戻します。

y 式。必要な場合には、y は、実数の固定小数点の 2 進数の値に変換されます。

y が省略される場合は、長さはタイプ変換の規則によって決められます。

y は、負以外でなければなりません。

y = 0 の場合は、結果はヌル文字ストリングです。

次の規則が適用されます。

- *y* = 1 の場合は、結果は 1 ブランクの文字ストリングです。
- *y* が文字ストリングを入れるのに必要な長さより大きい場合は、結果は SBCS ブランクで埋まります。
- *y* が文字ストリングを入れるのに必要な長さより小さい場合は、結果が切り捨てられます。 GRAPHIC の後ろで切り捨て、必要ならば SBCS のブランクを追加してストリングの長さを完全にすることによって、保全性が保持されます。

例 1

GRAPHIC から文字へ変換します。*y* は、結果を入れるのに必要な長さです。

```
dcl X graphic(6);
dcl A char (12);
A = char(X,12);
```

X の値	中間結果	A に割り当てられる値
.A.B.C.D.E.F	.A.B.C.D.E.F	.A.B.C.D.E.F

例 2

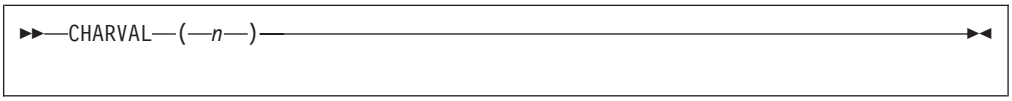
GRAPHIC から文字へ変換します。*y* は、結果を入れるには長さが短過ぎます。

```
dcl X graphic(6);
dcl A char (12);
A = char(X,11);
```

X の値	中間結果	A に割り当てられる値
.A.B.C.D.E.F	.A.B.C.D.E.F	.A.B.C.D.Eb

CHARVAL

CHARVAL は、整数に対応する CHARACTER(1) 値を戻します。



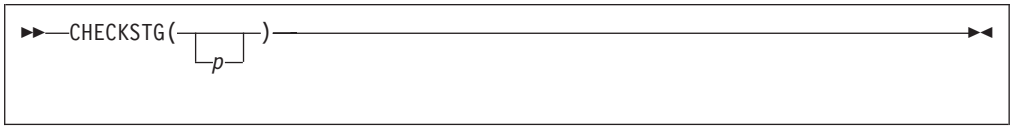
n 式。必要な場合には、UNSIGNED FIXED BIN(8) に変換されます。

CHARVAL(n) のビット値は n と同じですが (つまり、 UNSPEC(CHARVAL(n)) は UNSPEC(n) と等しい)、属性は CHARACTER(1) です。

CHARVAL は RANK の逆です (文字に適用した場合)。

CHECKSTG

CHECKSTG は、指定されたポインター値が、割り振られた正常なストレージの開始点であるかどうかを示すビット (1) 値を戻します。ポインター値が与えられていない場合、CHECKSTG は、すべての割り振り済みストレージが正しいかどうかを判別します。この組み込み関数を使用するには、CHECK(STORAGE) コンパイル時オプションも指定しなければなりません。



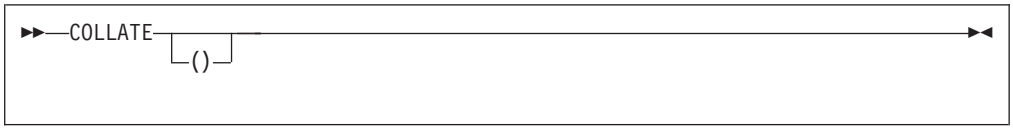
p ポインターの式。

割り振りが行なわれる場合、その後ろに、'ff'x に設定される余分の 8 バイトが続きます。それらのバイトが変更されていない場合、割り振りは、*正しい* と見なされます。

ポインターの式は、BASED 変数に割り振られたストレージを示さなければなりません。

COLLATE

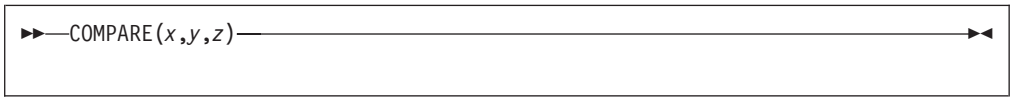
COLLATE は、256 の可能な CHARACTER(1) 値を各照合命令で 1 回比較する CHARACTER(256) ストリングを戻します。



COMPARE

COMPARE は、次のような FIXED BINARY(31,0) 値を戻します。

- アドレス x とアドレス y の z バイトが同一の場合は、ゼロ。
- アドレス x の z バイトがアドレス y の z バイトよりも小さい場合は、負の値。
- アドレス x の z バイトがアドレス y の z バイトよりも大きい場合は、正の値。



x および y
式。両方に POINTER タイプまたは OFFSET タイプを保持していなければなりません。 OFFSET の場合は、式を AREA 修飾を使用して宣言する必要があります。

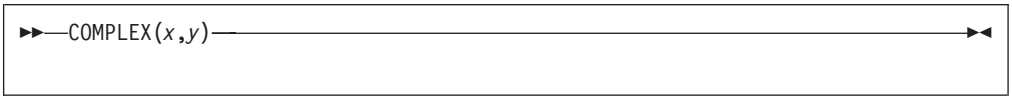
z FIXED BINARY(31,0) に変換される式。

例

```
dcl Result fixed bin;
dcl 1 Str1,
      2 B fixed bin(31),
      2 C pointer,
      2 * union,
        3 D char(4),
        3 E fixed bin(31),
        3 *,
          4 * char(3),
          4 F fixed bin(8) unsigned,
      2 * char(0);
dcl 1 Template nonasgn static,
      2 * fixed bin(31) init(16),     /* ''X */
      2 * pointer init(null()),
      2 * char(4) init(''),
      2 * char(0);

call plimove(addr(Str1), addr(Template), stg(Str1));
Result = compare(addr(Str1), addr(Template), stg(Str1));   /*  0 */
D = 'DSA ';
Result = compare(addr(Str1), addr(Template), stg(Str1));   /*  1 */
B = 15;      /* '00000F00'X */
D = 'DSA ';
Result = compare(addr(Str1), addr(Template), stg(Str1));   /* -1 */
```

COMPLEX

COMPLEX は、複合値 *x* + *y*I を戻します。



省略形: CPLX

x および y
実数式。

x および *y* の基数が異なる場合は、10 進数の引数は 2 進数に変換されます。スケールが異なる場合は、固定小数点の引数は浮動小数点に変換されます。結果は、共通の基数およびスケールになります。

固定小数点の場合は、結果の精度は次の式で指定されます。

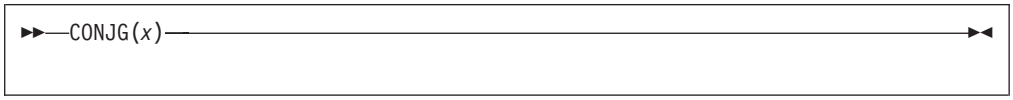
```
(min(N,max(p1-q1,p2-q2)+max(q1,q2)),max(q1,q2))
```

上記の場合、(p1,q1) と (p2,q2) は、それぞれ *x* と *y* の精度であり、 N は許可される最大の桁数です。

必要な変換が行われたあとで、引数が浮動小数点の場合には、結果には長い方の引数の精度になります。

CONJG

CONJG は、*x* の共役複素数、つまり、虚数部の符号を逆にした式の値を戻します。

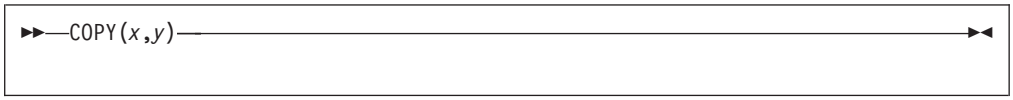


x 式。

x が実数の場合は、*x* は複素数に変換されます。結果は、*x* の基数、スケール、モードおよび精度になります。

COPY

COPY は、ストリング *x* のコピーが連結された *y* で構成されるストリングを戻します。



x 式。

x に、計算タイプ (必須) とストリング・タイプを保持している必要があります。そうでないと、文字に変換されます。

y 負以外の値を使用した整数式。繰り返し回数を指定します。 *y* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

y がゼロの場合は、結果はヌル・ストリングです。

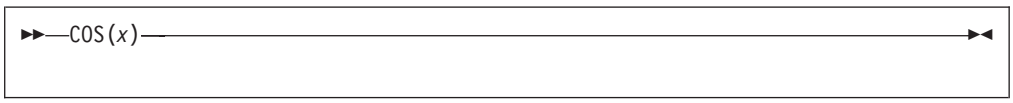
以下のコードを例として示します。

```
copy('Walla  ',1)        /*  returns 'Walla  '  */

repeat('Walla  ',1)      /*  returns 'Walla  Walla  '  */
```

`repeat(x,n)` は、`copy(x,n+1)` と同等です。

COS

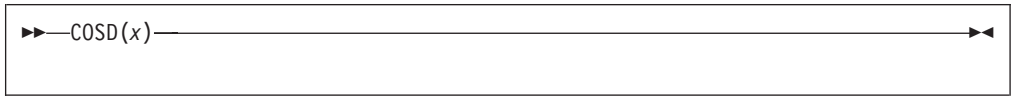
COS は、*x* の基数、精度、およびモードの浮動小数点で、*x* の余弦の近似を戻します。



x ラジアン値を使用した式。

COSD

COSD は、*x* の基数および精度の浮動小数点で、 *x* の余弦の近似を戻します。



x 度の値の実数式。

COSF

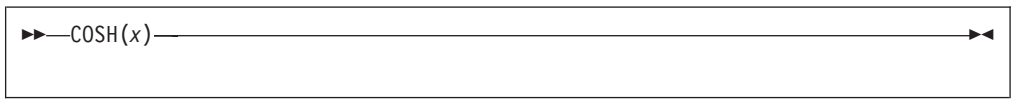
COSF は、次の点を除いては COS とまったく同じです。

- COSF は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 引数は実数表現でなければならない。
- サポートされる引数の最大絶対値は、ハードウェアによって設定される。
- 無効な引数が指定された場合、INVALIDOP 条件が発生したり、その他のハードウェア例外が生成されたりするなど、予期できない結果が生じることがある。
- 結果の正確度は、ハードウェアによって設定される。

定義および構文については、 430 ページの『COS』を参照してください。

COSH

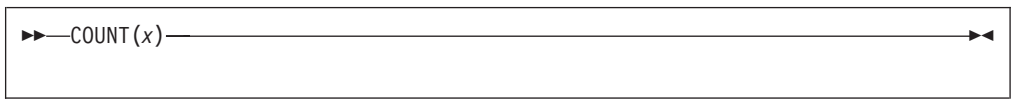
COSH は、*x* の基数、精度、およびモードの浮動小数点で、 *x* の双曲線余弦の近似を戻します。



x 式。

COUNT

COUNT は、*x* に対する最新の GET 命令または PUT 命令での、伝送されるデータ項目の数を指定するスケールのない REAL FIXED BINARY 値を戻します。



x ファイル参照。ファイルは、オープン状態で、STREAM 属性を持っている必要があります。

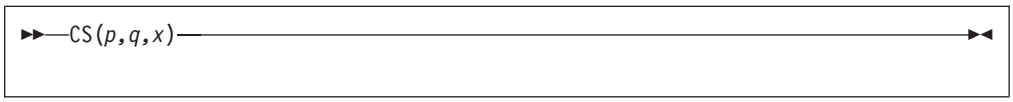
x に対する GET 命令または PUT 命令の伝送項目のカウントは、最初のデータ項目が伝送される前にゼロに初期化され、リストの各データ項目が伝送された後で 1 ずつ増加されます。 *x* が、現行プログラムでオープンされていない場合は、ゼロが戻されます。

GET 命令または PUT 命令時に ON ユニットまたはプロシージャーに入り、 ON ユニットまたはプロシージャー内で、*x* に対して GET 命令または PUT 命令が実行される場合には、COUNT の値は新しい命令を実行するためにリセットされ、オリジナルの GET 命令または PUT 命令がカウントされるときに復元されます。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

CS

CS は、*比較およびスワップ* の以前の値と現行値が等しいかどうかを示す FIXED BINARY(31) 値を戻します。



p 以前の FIXED BINARY(31) 値のアドレス。

q 現行の FIXED BINARY(31) 値のアドレス。

x 新しい FIXED BINARY(31) 値。

CS は、「現行」値と「以前の」値を比較します。これらの値が等しい場合は、「現行」値を上書きして「新しい」値がコピーされ、値 0 が戻されます。これらの値が等しくない場合は、「以前の」値を上書きして「現行」値がコピーされ、値 1 が戻されます。

したがって、CS は次の PL/I 関数としてインプリメントできますが、この場合 CS はアトミックではなくなります。:

```
 cs: proc( old_Addr, current_Addr, new )
  returns( fixed bin(31) byvalue )
  options( byvalue );

  dcl old_Addr     pointer;
  dcl current_Addr pointer;
  dcl new          fixed bin(31);

  dcl old          fixed bin(31) based(old_addr);
  dcl current      fixed bin(31) based(current_addr);

  if current = old then
    do;
      current = new;
      return( 0 );
    end;
  else
    do;
      old = current;
      return( 1 );
    end;
end;
```

z/OS 上では、CS 組み込み関数は CS 命令をインプリメントしています。この関数について詳しくは、「*Principles of Operations*」の付録を参照してください。

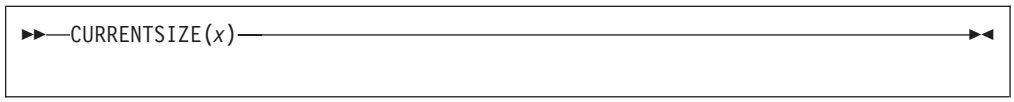
Intel 上では、CDS 組み込み関数は cmpxchg4 命令を使用します。 cmpxchg4 関数は、「現行」値、「新しい」値、および「以前の」値のアドレスを受け取ります。関数は元の「現行」値を戻し、「現行」値が「以前の」値と等しい場合だけ、「現行」値を「新しい」値に更新します。

このため Intel 上では、CS 組み込み関数は次のインライン関数によってインプリメントされます。

```
 cs: proc( old_Addr, current_Addr, new )
  returns( fixed bin(31) byvalue )
  options( byvalue );

  dcl old_Addr     pointer;
  dcl current_Addr pointer;
  dcl new          fixed bin(31);

  dcl old          fixed bin(31) based(old_addr);
  dcl current      fixed bin(31) based(current_addr);

  if cmpxchg4( current_Addr, new, old ) = old then
    do;
      return( 0 );
    end;
  else
    do;
      old = current;
      return( 1 );
    end;
end;
```

CURRENTSIZE

CURRENTSIZE は、処理系定義ストレージを与える FIXED BINARY(31,0) 値を、バイトで戻します。この値は、*x* によって要求されます。



x 任意のデータ・タイプ、データ編成、およびストレージ・クラスを持つ変数。ただし、下記のものは除きます。

- 位置合わせされていない固定長ビット・ストリングである BASED 変数、DEFINED 変数、パラメーター変数、添え字付き変数、構造変数、または共用体基底付き変数。
- 最初または最後の基本エレメントが位置合わせされていない固定長のビット・ストリングである小構造体または共用体 (ただし、その小構造体を含んでいる大構造体または共用体の最初または最後のエレメントになっている場合を除く)。
- BASED 属性、DEFINED 属性、またはパラメーター属性を持つ大構造体か共用体、またはパラメーターである大構造体か共用体で、位置合わせされていない固定長のビット・ストリングを最初または最後のエレメントとするもの。
- 連結ストレージにない変数。

CURRENTSIZE(x) によって戻される値は、次の状況で伝送されるバイト数として定義されます。

```
declare F file record output
        environment(scalarvarying);
write file(F) from(S);
```

x がスカラー可変長ストリングの場合は、戻り値にはストリングの長さの接頭辞と現在使われているバイト数が含まれます。 *x* には、ストリングの未使用のバイト数は含まれません。

x がスカラー区域の場合は、戻り値には、区域制御バイトと区域の現在のエクステントが含まれます。 *x* には、区域の最後の未使用のバイト数は含まれません。

x が区域または可変長ストリングを含む集合の場合は、戻り値には、区域制御バイト、区域の最大サイズ、ストリングの長さの接頭辞、およびストリングの最大長でのバイト数が含まれます。この規則の例外を次に示します。

x が、非次元区域を最後のエレメントとして持つ構造体または共用体の場合は、戻り値には、区域の制御バイトおよび区域の現行エクステントが含まれます。 *x* には、区域の最後での未使用のバイト数は含まれません。

変数が割り振られていない場合、CURRENTSIZE 組み込み関数を調節可能エクステントとともに、BASED 変数上で使用してはなりません。

CURRENTSIZE 組み込み関数の例については、507 ページの『SIZE』を参照してください。

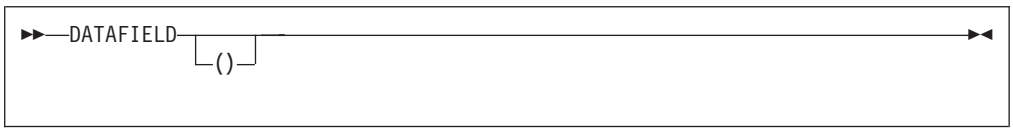
CURRENTSTORAGE

省略形: CSTG

CURRENTSTORAGE は、CURRENTSIZE の同義語です。詳細については、433 ページの『CURRENTSIZE』を参照してください。

DATAFIELD

DATAFIELD は、NAME 条件の ON ユニット (またはその動的子孫) のコンテキストにあります。文字ストリングを返しますが、その値は条件を発生したフィールドのコンテキストです。また、DATAFIELD も、NAME 条件の暗黙処置の一部として発生した ERROR 条件または FINISH 条件の ON ユニット (またはその動的子孫) のコンテキストにもあります。

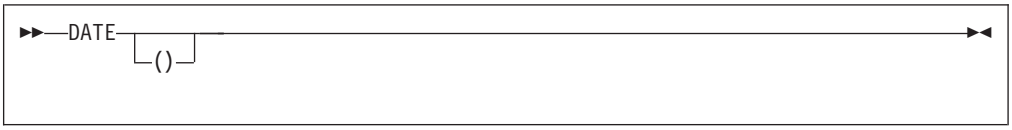


条件が発生したストリングに DBCS ID、GRAPHIC データ、または混合文字データが含まれる場合は、DATAFIELD は混合文字ストリングを戻します。

コンテキスト以外で DATAFIELD が使われている場合は、ヌル・ストリングが戻されます。

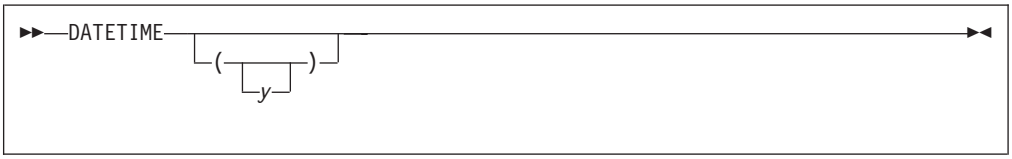
DATE

DATE は、YYMMDD フォーマットの日付を含む非可変文字 (6) のストリングを戻します。



DATETIME

DATETIME は、デフォルトまたはユーザー指定のフォーマットのどちらかで、今日の日付の文字ストリング・タイム・スタンプを戻します。



y 式。

存在する場合は、戻される日付/時刻のパターンを指定します。 *y* が指定されない場合は、デフォルトの日付/時刻のパターン「YYYYMMDDHHMISS999」が想定されます。

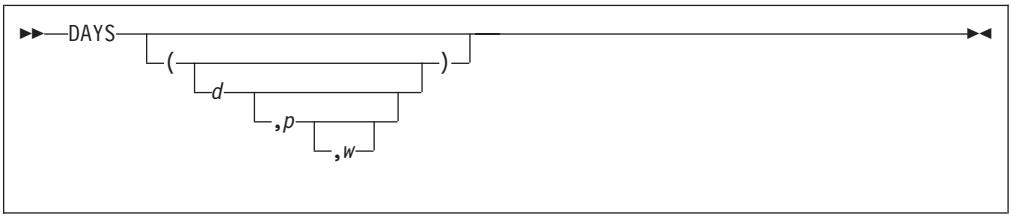
許可されるパターンについては、 406 ページの表 52 を参照してください。

y に計算タイプ (必須) および文字タイプを保持する必要があります。そうでないと、文字に変換されます。

DATETIME の使用例については、『DAYS』を参照してください。

DAYS

DAYS は、日付 *d* に対応する日数 (リリアン形式) である FIXED BINARY(31,0) 値を戻します。



d 日付を表すストリング式。省略される場合は、DATETIME() によって戻される値が想定されます。

d の値に、計算タイプ (必須) および文字タイプを保持している必要があります。そうでない場合は、*d* は文字に変換されます。

p サポートされている日付/時刻パターンの 1 つ。省略される場合は、「YYYYMMDDHHMISS9999」が想定されます。

p に計算タイプ (必須) および文字タイプを保持する必要があります。そうでないと、文字に変換されます。

w 2 桁の年フォーマットを処理するために使用されるように世紀ウィンドウを定義する整数式。

- 値が正 (1950 など) の場合、その値は年として扱われます。
- 値が負またはゼロの場合、その値は、現行のシステム提供の年から減算されるようにオフセットを指定します。
- 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値にデフォルトで設定されます。

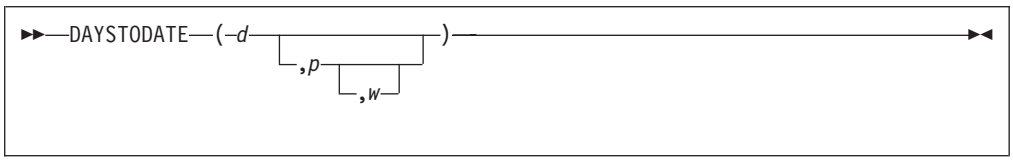
例

```
dcl Date_format value ('MMDDYYYY') char;
dcl Todays_date char(length(Date_format));
dcl Sep2_1993 char(length(Date_format));
dcl Days_of_July4_1993 fixed bin(31);
dcl Msg char(100) varying;
dcl Date_due char(length(Date_format));

Todays_date = datetime(date_format);          /* e.g. 06161993  */

Days_of_July4_1993 = days('07041993','MMDDYYYY');
Sep2_1993 = daystodate(days_of_July4_1993 + 60, Date_format);
             /*  09021993    */

Date_due = daystodate(days() + 60, Date_format);
        /* assuming today is July 4, 1993, this would be Sept. 2, 1993  */

Msg = 'Please pay amount due on or before ' ||
         substr(Date_due, 1, 2) || '/' ||
         substr(Date_due, 3,2)  || '/' ||
         substr(Date_due, 5);
```

許可されるパターンを 406 ページの表 52 にリストしてあります。リリアン形式の説明については、404 ページの『日付/時刻組み込み関数』を参照してください。

DAYSTODATE

DAYSTODATE は、*d* 日 (リリアン形式) に対応する *p* で指定されたフォーマットの日付を含む非可変文字ストリングを戻します。



d 日数 (リリアン形式)。 *d* は、計算タイプでなければなりません。また、必要な場合には、FIXED BINARY(31,0) に変換されます。

p サポートされている日付/時刻パターンの 1 つ。

省略される場合、*p* は、デフォルトの日付/時刻パターン「YYYYMMDDHHMISS999」(DATETIME で戻されるデフォルトのフォーマット) が想定されます。

w 2 桁の年フォーマットを処理するために使用されるように世紀ウィンドウを定義する整数式。

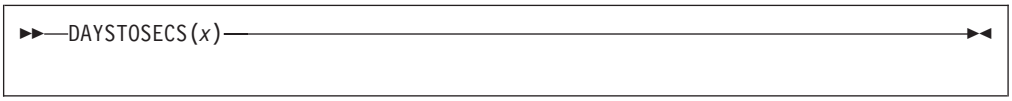
- 値が正 (1950 など) の場合、その値は年として扱われます。
- 値が負またはゼロの場合、その値は、現行のシステム提供の年から減算されるようにオフセットを指定します。
- 省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値にデフォルトで設定されます。

許可されるパターンを 406 ページの表 52 にリストしてあります。リリアン形式の説明については、404 ページの『日付/時刻組み込み関数』を参照してください。

DAYSTODATE の使用例については、435 ページの『DAYS』を参照してください。

DAYSTOSECS

DAYSTOSECS は、日数 *x* に対応する秒数である FLOAT BINARY(53) の値を戻します。



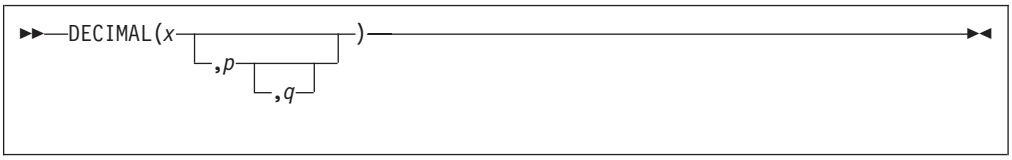
x 式。

x は、計算タイプでなければなりません。また、必要な場合には、 FIXED BINARY(31,0) に変換されます。

DAYSTOSECS(x) は、*x**(24*60*60) と同じです。

DECIMAL

DECIMAL は、*p* および *q* で指定される精度で、*x* の 10 進数の値を戻します。結果には、*x* のモードとスケールが含まれています。



省略形: DEC

x 参照。

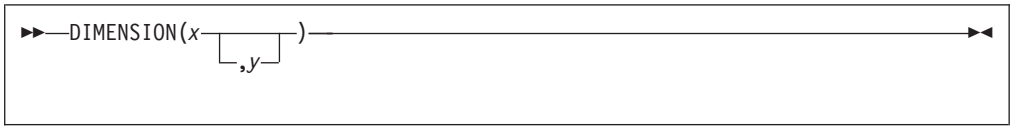
p 演算で維持される桁数を指定する制限付き式。

q 結果のスケール因数を指定する制限付き式。結果が固定小数点の場合に、*p* が与えられ、*q* が省略されるときは、スケール因数はゼロと見なされます。結果が浮動小数点の場合は、*q* は省略します。

p と *q* が両方とも省略される場合は、結果の精度は基数変換の規則により決まります。

DIMENSION

DIMENSION は、*x* の次元 *y* の現行エクステントを指定する FIXED BINARY(31,0) 値を戻します。



省略形: DIM

x 配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

y *x* の特定の次元を指定する式。必要な場合には、*y* は FIXED BINARY(31,0) に変換されます。 *y* は、1 以上でなければなりません。*y* が提供されない場合は、デフォルトを 1 にします。

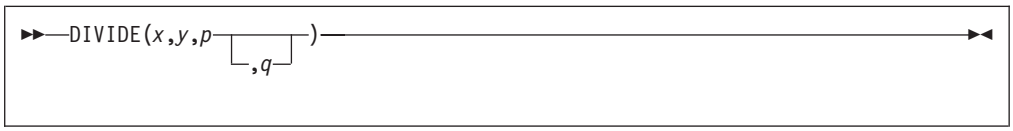
配列が 1 次の場合のみ、*y* を省略できます。

y が *x* の次元よりも大きい場合は、DIMENSION 関数は未定義の値を戻します。

DIMENSION の代わりに LBOUND および HBOUND を使用することをお勧めします。

DIVIDE

DIVIDE は、*x*/*y* の商を *p* と *q* で指定された精度で戻します。結果の基数、スケール、およびモードは、PRECTYPE コンパイラー・オプションによって規則が変更されないかぎり、式の評価規則に従います。



x 式。

y 式。 *y* = 0 の場合は、ZERODIVIDE 条件が発生します。

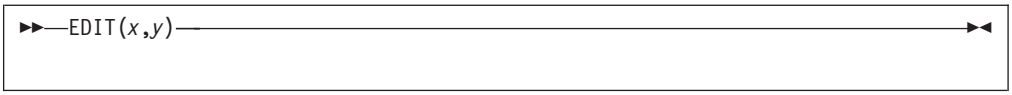
p 演算で維持される桁数を指定する制限付き式。

q 結果のスケール因数を指定する制限付き式。結果が固定小数点の場合に、*q* が省略されるときは、スケール因数ゼロがデフォルトです。結果が浮動小数点の場合は、*q* は省略します。

EDIT

EDIT は、長さ LENGTH(y) の文字ストリングを戻します。 EDIT の値は、*y* で与えられたピクチャー指定を使用して宣言された変数に *x* が指定された場合の結果と同じです。

有効なピクチャー文字については、 341 ページの『第 15 章 ピクチャー指定文字』を参照してください。



x 式。

x に、計算タイプを保持する必要があります。

y ストリング式。

y に文字タイプを保持し、PICTURE データ項目に有効なピクチャー文字を指定する必要があります。 *y* に有効なピクチャー指定がない場合は、ERROR 条件が発生します。

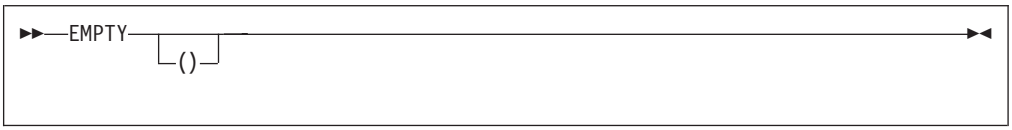
例

```
dcl pic1 char(9) init ('ZZZZZZZZ9');
dcl pic2 char(7) init ('ZZ9V.99');
dcl num fixed dec (9) init (123456789);
z = edit (num, pic1);                    /* '123456789'       */
z = edit (num, pic2);                    /*    '789.00'       */
z = edit (num, substr(pic1,8));          /*     '89'          */
z = edit (num, substr(pic2,1,5));        /*    '789.'         */
z = edit (num, substr(pic1,7,3));        /*    '789'          */
z = edit (num, substr(pic2,3,5));        /*    '9.00'         */
z = edit ('1', substr(pic1,7,3));        /*    '  1'          */
z = edit ('PL/I', 'AAXA');               /*    'PL/I'         */
z = edit ('PL/I', 'AAAA');               /* raises conversion */
```

x を *y* で指定したピクチャー指定に編集できない場合は、発生した条件は *x* が *y* に設定されているピクチャー指定と同じピクチャー指定がある PICTURE データ項目に割り当てられた場合に発生する条件です。

EMPTY

EMPTY は、ゼロ・エクステントの区域を戻します。 EMPTY は、区域のすべての割り振りを解放するのに使われます。

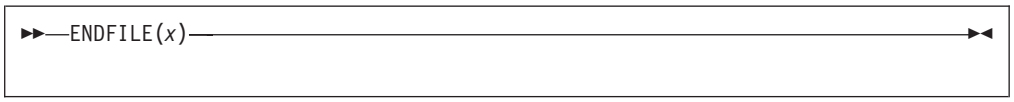


この関数の値は、変数が割り振られたときに、区域変数に割り当てられます。以下に例を示します。

```
declare A area,
        I based (P),
        J based (Q);

allocate I in(A), J in (A);
A = empty();

/* Equivalent to:  free I in (A), J in (A); */
```

ENDFILE

ENDFILE は、ファイルの最後に到達したときに、'1'B を戻します。到達しない場合は、'0'B を戻します。ファイルがオープンされていない場合は、ERROR 条件が発生します。

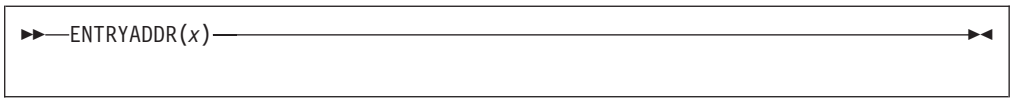


x ファイル参照。

ENDFILE は、バイトストリーム・ファイル (たとえば、FILEREAD 組み込み関数の使用を必要とするファイル) のファイルの終わり条件を検出するために使用できます。

ENTRYADDR

ENTRYADDR は、入り口 *x* が呼び出された場合に、最初に実行された命令のアドレスであるポインター値を戻します。入り口 *x* は、ネストされていないプロシージャーを表す必要があります。

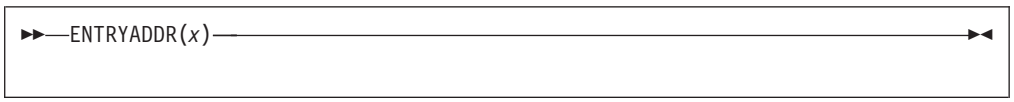


x 入り口参照。

x がフェッチ可能な入り口定数の場合は、ENTRYADDR が実行される前に *x* をフェッチする必要があります。

ENTRYADDR 疑似変数

ENTRYADDR 疑似変数は、呼び出される入り口のアドレスを使用して、入り口変数 *x* を初期化します。

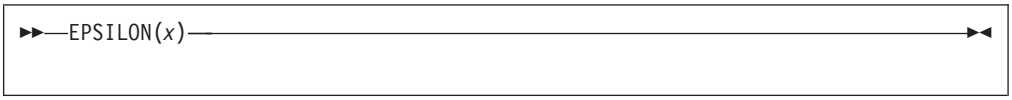


x 入り口参照。

注: ENTRYADDR 変数に提供されたアドレスが内部プロシージャーのアドレスの場合、結果は保証されません。

EPSILON

EPSILON は、*x* が 1 のときに、*x* と次の正の数の間のスペースである浮動小数点値を戻します。戻り値は、*x* の基数、モード、および精度になります。

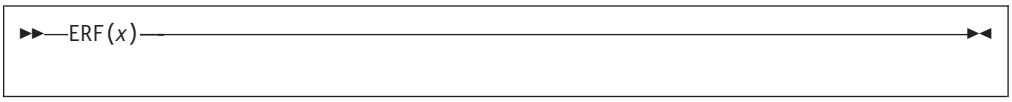


x REAL FLOAT 式。

EPSILON(x) は定数であり、制限付き式に使用できます。

ERF

ERF は、*x* の誤差関数の近似を実数の浮動小数点の値で戻します。



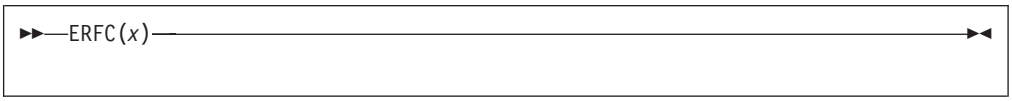
x 実数式。

結果は *x* の基数と精度であり、次の式で指定されます。

$(2/\sqrt{(\pi)}) \int_0^x EXP(-(t^2))dt$

ERFC

ERFC は、*x* の誤差関数の補数の近似を実数の浮動小数点の値で戻します。



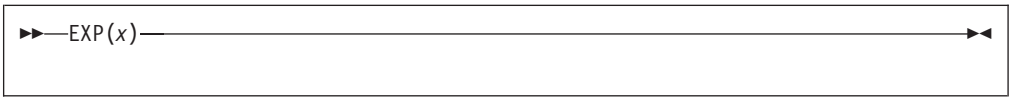
x 実数式。

結果は *x* の基数と精度であり、次の式で指定されます。

```
1 - ERF(x)
```

EXP

EXP は、自然対数の底 e の *x* 乗の近似を浮動小数点の値で戻します。



x 式。

結果は、*x* の基数、モード、および精度になります。

EXPF

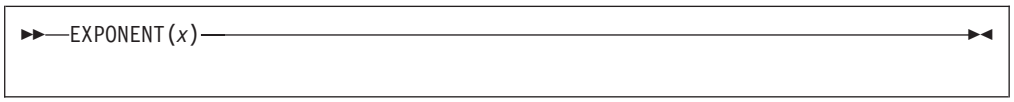
EXPF は、次の点を除いては EXP とまったく同じです。

- EXPF は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 引数は実数表現でなければならない。
- 無効な引数が指定された場合、INVALIDOP 条件が発生したり、その他のハードウェア例外が生成されたりするなど、予期できない結果が生じることがある。
- 結果の正確度は、ハードウェアによって設定される。

定義および構文については、 441 ページの『EXP』を参照してください。

EXPONENT

EXPONENT は、x の指数部分である FIXED BINARY(31,0) 値を戻します。



x 式。 *x* は、REAL FLOAT として宣言しなければなりません。

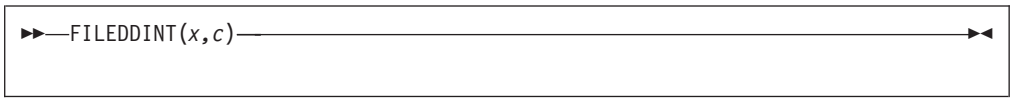
EXPONENT(x) は、*x* の「数学」指数ではありません。 *x* = 0 の場合は、EXPONENT(x) = 0 です。 *x* が 0 以外の場合は、EXPONENT(x) は次のような固有の数 *e* です。

```
       (e-1)                    e
radix(x)      <= abs(x) < radix(x)
```

したがって、EXPONENT(1e0) は 1 であり、0 ではありません。

FILEDDINT

FILEDDINT は、ファイル *x* の属性 *c* の値である FIXED BIN(31) 値を戻します。



x ファイル参照。

c 照会される属性を保持する文字ストリング。

FILEDDINT を使用している場合、以下は、*c* の有効な値です。

blksize	keylen
bufsize	keyloc
delay	recsize
filesize	retry

ONCODE 1010 の ERROR 条件は、ファイルがオープンしていないか、照会されている属性がファイルに無効な場合に発生します。

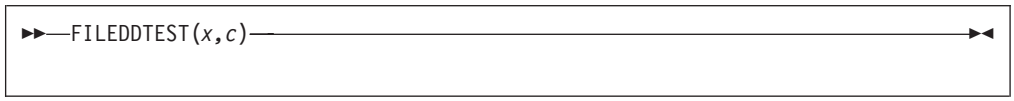
FILEDDINT(x,'BLKSIZE') は、z/OS でのみ有効です。 FILEDDINT(x,'BLKSIZE') は CONSECUTIVE ファイルのブロック・サイズを戻します。 HFS ファイルの場合は 0 を戻し、VSAM ファイルの場合にも 0 を戻します。

FILEDDINT(x,'FILESIZE') は、z/OS で、HFS ファイルの場合を除き 0 の値を戻します。

FILEDDINT(x,'KEYLOC') および FILEDDINT(x,'KEYLEN') は VSAM KSDS ファイルにのみ有効です。

FILEDDTEST

FILEDDTEST は、属性 *c* がファイル *x* に適用する場合に値 1 を保持する FIXED BIN(31) 値を戻します。属性 *c* がファイル *x* に適用しない場合は、0 の値を戻します。



x ファイル参照。

c 照会される属性を保持する文字ストリング。

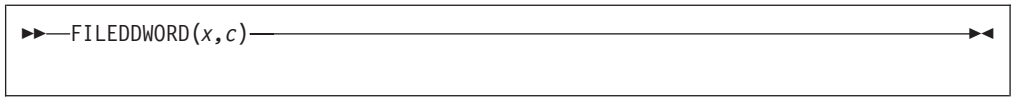
FILEDDTEST を使用している場合、以下は、*c* の有効な値です。

append
bkwd
ctlasa
delimit
descendkey
genkey
graphic
lrmskip
print
prompt
scalarvarying
skip0

ONCODE 1010 の ERROR 条件は、ファイルがオープンしていないか、照会されている属性がファイルに無効な場合に発生します。

FILEDDWORD

FILEDDWORD は、ファイル *x* の属性 *c* の値である文字ストリングを戻します。



x ファイル参照。

c 照会される属性を保持する文字ストリング。

FILEDDWORD

FILEDDWORD を使用している場合、以下は、*c* の有効なオプションです。

access	putpage
amthd	recfm
action	share
charset	type
filename	typef
organization	

これらのオプションは以下の値を戻します。

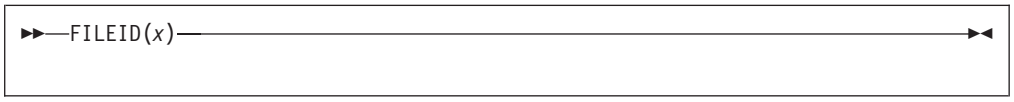
- ACCESS は、SEQUENTIAL または DIRECT を戻します。
- ACTION は、INPUT、OUTPUT、または UPDATE を戻します。
- AMTHD は、z/OS プラットフォームでは VSAM KSDS、VSAM ESDS、または VSAM RRDS を戻し、Windows または AIX プラットフォームでは FILESYS、DDM、BTRIEVE、または ISAM を戻します。
- CHARSET は ASCII または EBCDIC を戻します。
- z/OS プラットフォームの場合、 FILENAME は、HFS ファイルの完全修飾パス名、およびその他のすべてのファイルの MVS データ・セット名を戻します。ただし、DSN=NULLFILE と DD DUMMY のいずれかで指定されたファイルについては、値「NULLFILE」を戻します。 PDS または PDSE のメンバーである MVS データ・セットについては、戻された名前にメンバー名が含まれています。Windows および AIX プラットフォームでは、ファイルの完全修飾パス名を戻します。
- ORGANIZATION は、CONSECUTIVE、RELATIVE、REGIONAL(1)、または INDEXED を戻します。
- RECFM は、ファイルの適切なレコード・フォーマット設定、および VSAM ファイルについては U を戻します。このオプションは z/OS でのみ有効です。
- SHARE は NONE、READ、または ALL を戻します。
- TYPE は、RECORD または STREAM を戻します。
- TYPEF は、固有ファイルのタイプを戻します。

ONCODE 1010 の ERROR 条件は、ファイルがオープンしていないか、照会されている属性がファイルに無効な場合に発生します。

FILEDDWORD(x,'RECFM') は、z/OS でのみ有効です。

FILEID

FILEID は、PL/I ファイル定数またはファイル変数のシステム・トークンである FIXED BIN(31) 値を戻します。



x ファイル参照。

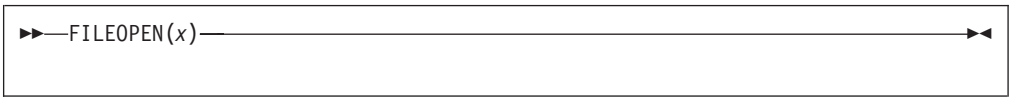
このトークンを PL/I ステートメントによって達成するどのような目的のためにも使用しないでください。

z/OS では、トークンは、RECORD または STREAM ファイルに関連した DCB のアドレスを保持します。トークンは他のファイルでは無効です。注: アプリケーションが DCB を読み取れるように、DCB アドレスが提供されています。 DCB を変更してはなりません。

ONCODE 1010の ERROR 条件は、ファイルがオープンしていない場合に発生します。

FILEOPEN

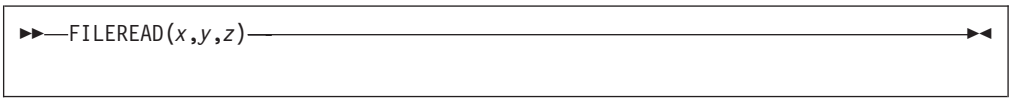
FILEOPEN に戻される値は、ファイル *x* がオープンされている場合は '1'B、ファイルがオープンされていない場合は '0'B です。



x ファイル参照。

FILEREAD

FILEREAD により、ファイル *x* から位置 *y* に *z* ストレージの単位 (バイト) が読み取られます。そして、実際に読み込まれたストレージの単位の数が戻されます。



x ファイル参照。

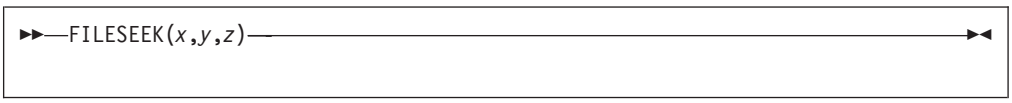
y タイプ POINTER または OFFSET による式。タイプが OFFSET の場合は、式は、AREA 属性によって宣言される OFFSET 変数でなければなりません。

z FIXED BIN(31,0) に変換される計算タイプによる式。

FILEREAD が読み取れるのは、TYPE(U) のファイルのみです。

FILESEEK

FILESEEK は、ファイル *x* と関連した現在のファイル位置をファイル内の新しい位置に変更します。ファイル上での次の操作は、新しい位置で行なわれます。
FILESEEK は、C での fseek 関数と同等です。



x ファイル参照。

y ファイル・ポインターが *z* に対して相対的に移動されることになっている位置の数を示す FIXED BINARY(31) 値。

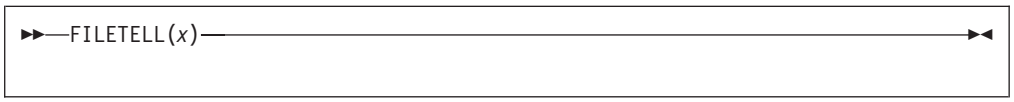
z ファイル・ポインターの移動の起点を示す FIXED BINARY(31) 値。次の値が有効です。

-1 ファイルの始め
0 ファイル・ポインターの現在位置
1 ファイルの終わり

FILESEEK は、TYPE(U) のファイルでのみ使用できます。

FILETELL

FILETELL は、ファイル *x* の現在位置を示す FIXED BINARY(31) 値を戻します。戻される値は、ファイルの始めを相対的に示すオフセットです。 FILETELL は、C での ftell 関数と同等です。

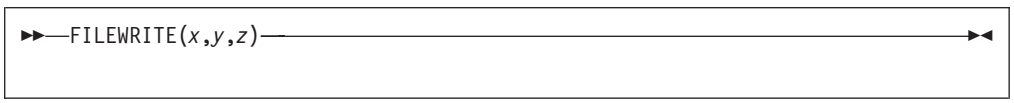


x ファイル参照。

FILETELL は、TYPE(U) のファイルでのみ使用できます。

FILEWRITE

FILEWRITE により、位置 *y* からファイル *x* に *z* ストレージの単位 (バイト) が書き込まれます。そして、実際に書き込まれたストレージの単位の数が戻されます。



x ファイル参照。

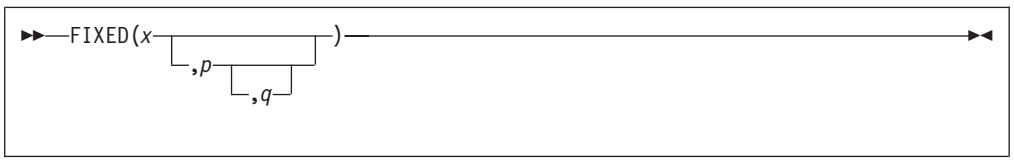
y タイプ POINTER または OFFSET による式。タイプが OFFSET の場合は、式は、AREA 属性によって宣言される OFFSET 変数でなければなりません。

z FIXED BIN(31,0) に変換される計算タイプによる式。

FILEWRITE は、TYPE(U) のファイルにのみ書き込みを行います。

FIXED

FIXED は、*p* および *q* で指定される精度で、*x* の固定小数点の値を戻します。結果には、*x* の基数とモードが含まれています。



x 式。

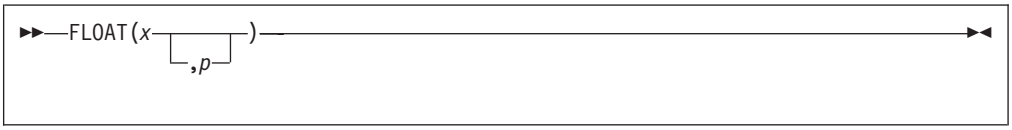
p 結果の合計桁数を指定する制限付き式。これは、実行の限界を超えてはいけません。

q 結果のスケール因数を指定する制限付き式。 *q* が省略される場合は、スケール因数はゼロと想定されます。

p と *q* が両方とも省略される場合は、結果の精度は基数変換の規則により決まります。

FLOAT

FLOAT は、*p* で指定される精度で、*x* の浮動小数点の近似値を戻します。結果には、*x* の基数とモードが含まれています。



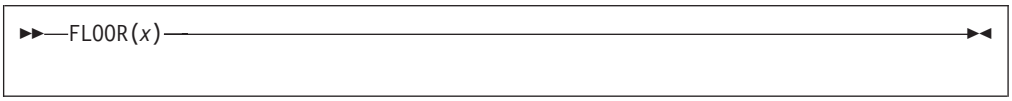
x 式。

p 結果の最小桁数を指定する制限付き式。

p が省略される場合は、結果の精度は基数変換の規則により決まります。

FLOOR

FLOOR は、*x* 以下の最も大きい整数を判別し、この値を結果に割り当てます。



x 実数式。

結果のモード、基数、スケール、および精度は、引数と一致します。 *x* が精度 (*p*,*q*) を使用した固定小数点のときは、結果の精度は次の式で与えられます。

```
(min(n,max(p-q+1,1)),0)
```

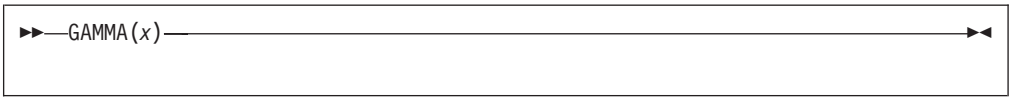
この場合、n は、使用できる最大桁数です。FIXED DECIMAL の場合は N が入り、FIXED BINARY の場合は M が入ります。

GAMMA

GAMMA は、*x* のガンマの近似であり、次の式で指定されます。

$gamma(x) = \int_0^\infty (u^{x-1})(e^{-x})du$

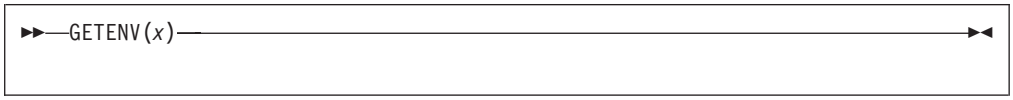
GAMMA は、*x* の基数、モード、および精度で浮動小数点の値を戻します。



x 実数式。 *x* の値は、ゼロより大きくなければなりません。

GETENV

GETENV は、指定した環境変数を表す文字値を戻します。



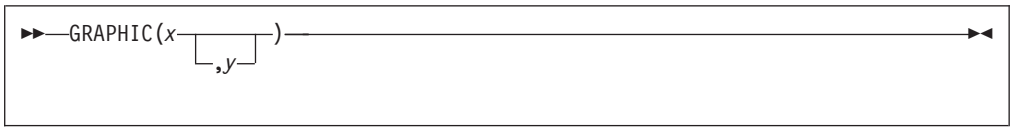
x 環境変数を命名する式。

GRAPHIC

GRAPHIC を使用すれば、文字 (または混合文字) データを GRAPHIC データに明示的に変換することができます。その他のすべてのデータは、最初に文字に変換され、次に GRAPHIC データ・タイプに変換されます。

GRAPHIC は、*y* で指定されるグラフィック記号での長さをとる *x* のグラフィック値を戻します。

文字は、グラフィックに変換されます。 *x* の内容は、変換時に妥当性がチェックされます。このとき、グラフィック定数と混合文字定数をチェックするときに使用するのと同じ規則が使われます。



x 式。 *x* が GRAPHIC のときは、*x* は長さ変更の対象で、適用可能な埋め込みまたは切り捨てが行われます。 *x* がグラフィック以外のときは、必要な場合には文字に変換されます。 SBCS 文字は、同等の DBCS 文字に変換されます。

y 式。必要な場合には、*y* は、実数の固定小数点の 2 進数の値に変換されます。 *y* が省略される場合は、長さはタイプ変換の規則によって決められます。

y は、負以外でなければなりません。

y = 0 の場合は、結果はヌルの漢字ストリングです。

次の規則が適用されます。

- *y* が漢字ストリングを入れるのに必要な長さより大きい場合は、結果はグラフィックのブランクで埋まります。
- *y* が漢字ストリングを入れるのに必要な長さより小さい場合は、結果が切り捨てられます。

例 1

結果を入れるのに十分な大きさのターゲットでの CHARACTER から GRAPHIC の変換の場合

```
dcl X char (11) varying;
dcl A graphic (11);
A = graphic(X,8);
```

X の値	中間結果	A に割り当てられる値
ABCDEFGHIJ	.A.B.C.D.E.F.G.H.I.J	.A.B.C.D.E.F.G.H.b.b.b
123	.1.2.3	.1.2.3.b.b.b.b.b.b.b.b
123A.B.C	.1.2.3.A.B.C	.1.2.3.A.B.C.b.b.b.b.b

この場合、.b は、DBCS ブランクです。

例 2

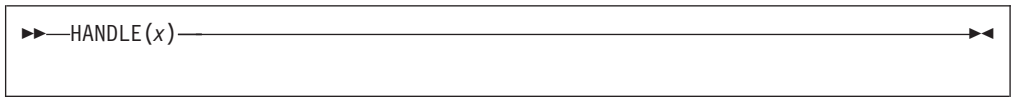
結果を入れるには短すぎるターゲットでの CHARACTER から GRAPHIC の変換の場合

```
dcl X char (10) varying;
dcl A graphic (8);
A = graphic(X);
```

X の値	中間結果	A に割り当てられる値
ABCDEFGHIJ	.A.B.C.D.E.F.G.H.I.J	.A.B.C.D.E.F.G.H

HANDLE

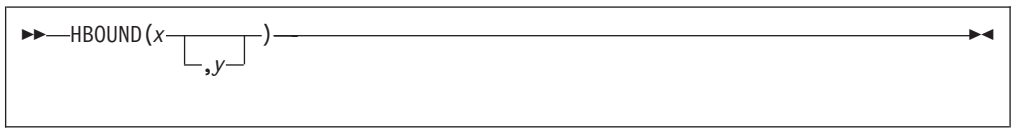
HANDLE は、タイプ付き構造体 *x* にハンドルを戻します。



x タイプ付き構造体。

HBOUND

HBOUND は、*x* の次元 *y* の現行の上限を FIXED BINARY(31,0) 値で戻します。



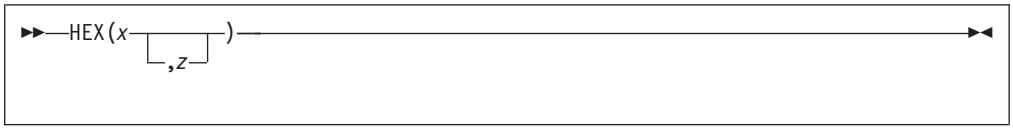
x 配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

y *x* の特定の次元を指定する式。必要な場合には、*y* は FIXED BINARY(31,0) に変換されます。 *y* は、1 以上でなければなりません。*y* が提供されない場合は、デフォルトを 1 にします。

配列が 1 次の場合のみ、*y* を省略できます。

HEX

HEX は、*x* が入っているストレージの 16 進数表示である文字ストリングを戻します。



HEX(x) は、長さ 2 * サイズ (x) の文字ストリングを戻します。

HEX(x,z) は、出力ストリングの 8 文字のすべてのセット間に挿入された 文字 *z* 付きの *x* を含む文字ストリングを戻します。その長さは、2 * サイズ (x) + ((サイズ (x) - 1)/4) です。

x 変数を表す式。 *x* を含む全バイト数は、16 進数に変換されます。

z 式。 *z* が指定された場合は、CHARACTER(1) NONVARYING タイプでなければなりません。

例 1

```
dcl Sweet char(5) init('Sweet');
dcl Sixteen fixed bin(31) init(16);
dcl XSweet char(size(Sweet)*2+(size(Sweet)-1)/4);
dcl XSixteen char(size(Sixteen)*2+(size(Sixteen)-1)/4);

XSweet = hex(Sweet,'-');
          /* '53776565-74' */

XSweet = heximage(addr(Sweet),length(Sweet),'-');
          /* '53776565-74' */

XSixteen = hex(Sixteen,'-');
          /* '10000000' - bytes NOT reversed */

XSixteen = heximage(addr(Sixteen),stg(Sixteen),'-');
          /* '00000010' - bytes reversed */
```

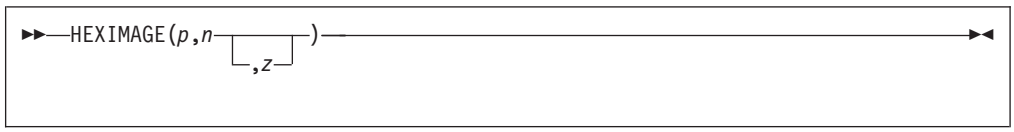
例 2

```
dcl X fixed bin(15) littleendian;
dcl Y fixed bin(15) bigendian;

X = 258;                                    /* stored as '0201'B4 */
Y = 258;                                    /* stored as '0102'B4 */

display (hex(X));                           /* displays 0102 */
display (hex(Y));                           /* displays 0102 */

display (heximage( addr(X), stg(X) ));      /* displays 0201 */
display (heximage( addr(Y), stg(Y) ));      /* displays 0102 */
```

注: この関数は、ストレージの *x* の正確なイメージを戻しません。正確なイメージが必要な場合は、HEXIMAGE 組み込み関数を使用してください。

HEXIMAGE

HEXIMAGE は、指定された位置のストレージの 16 進数表示である文字ストリングを戻します。



HEXIMAGE(p,n) は、位置 *p* でのストレージの *n* バイトの 16 進表示である文字ストリングを戻します。その長さは、2 * n です。

HEXIMAGE(p,n,z) は、出力ストリングの 8 文字の各セット間に挿入された文字 *z* 付きの位置 *p* でのストレージの *n* の 16 進表示である文字を戻します。その長さは、(2 * n) + ((n - 1)/4) です。

p ロケーター・タイプ (POINTER または OFFSET) を保持していなければならない制限付き式。 *p* が OFFSET の場合、AREA 属性を保持している必要があります。

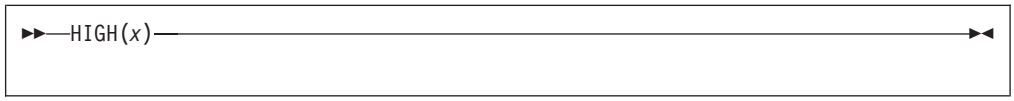
n 式。 *n* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

z *z* が指定された場合は、CHARACTER(1) NONVARYING タイプでなければなりません。

HEXIMAGE 組み込み関数の例については、450 ページの『HEX』を参照してください。

HIGH

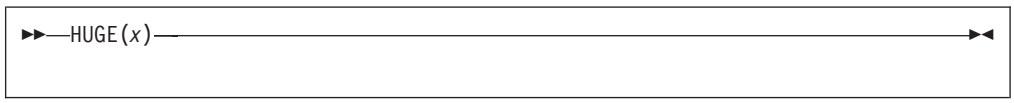
HIGH は、長さ *x* の文字ストリングを戻します。各文字は、照合順序の最上位の文字 (16 進数の FF) です。



x 式。必要な場合には、*x* は正の実数固定小数点の 2 進数の値に変換されます。*x* = 0 の場合は、結果はヌル文字ストリングです。

HUGE

HUGE は、*x* の想定可能な正の最大値を浮動小数点の値で戻します。戻り値は、*x* の基数、モード、および精度になります。

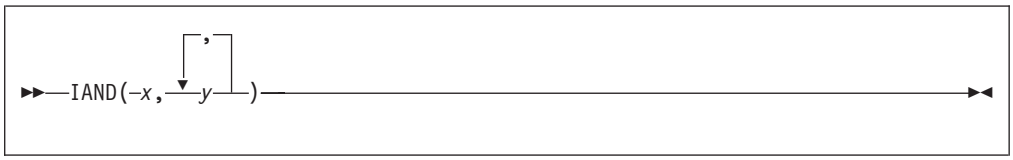


x 式。 *x* は、属性 REAL FLOAT を保持する必要があります。

HUGE(x) は定数であり、制限付き式で使用できます。

IAND

IAND は、引数の論理 AND を戻します。



x および y
計算タイプを保持する必要がある式。

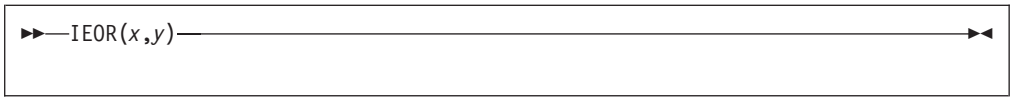
REAL FIXED BIN(p,0) ではない引数があると、その引数は SIGNED REAL FIXED BIN(M,0) に変換されます。

引数のいずれかが SIGNED の場合、UNSIGNED の引数はすべて SIGNED に変換されます。

結果は REAL FIXED BIN(max(p1,p2,...), 0) となります。すべての引数が UNSIGNED の場合は、結果も UNSIGNED です。

IEOR

IEOR は、*x* と *x* の論理排他 OR を戻します。すべての引数が UNSIGNED の場合は、結果も UNSIGNED です。



x および y
計算タイプを保持する必要がある式。

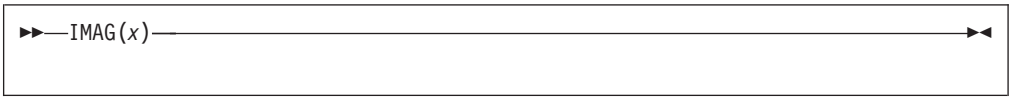
REAL FIXED BIN(p,0) ではない引数があると、その引数は SIGNED REAL FIXED BIN(M,0) に変換されます。

引数のいずれかが SIGNED の場合、UNSIGNED の引数はすべて SIGNED に変換されます。

結果は REAL FIXED BIN(max(p1,p2,...), 0) となります。すべての引数が UNSIGNED の場合は、結果も UNSIGNED です。

IMAG

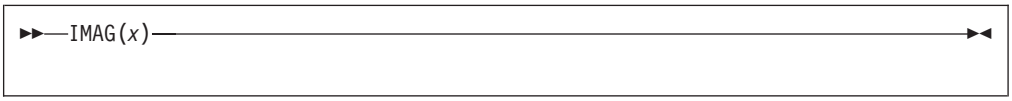
IMAG は、*x* の虚数部分の係数を戻します。結果のモードは実数です。結果は、*x* の基数、スケール、および精度になります。



x 式。 *x* が実数の場合は、*x* は複素数に変換されます。

IMAG 疑似変数

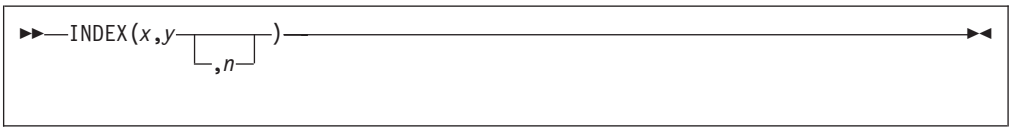
IMAG 疑似変数は、実数値または複素数の実数部分を *x* の虚数部分の係数に割り当てます。



x 複素数参照。

INDEX

INDEX は、*y* と等しいサブストリングの *x* 内の開始位置を示すスケールのない REAL FIXED BINARY 値を戻します。また、処理を開始する *x* 内の位置を指定することもできます。



x 検索されるストリング式。

y 検索元のストリング式。

n *n* は、処理を開始する *x* 内の位置を指定します。 *y* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

x 内で *y* が発生しない場合、または *x* か *y* のどちらかが長さゼロがある場合は、値ゼロが戻されます。

n が 1 より小さい場合、または *n* が 1 + length(x) よりも大きい場合は、STRINGRANGE 条件が発生し、結果は 0 になります。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

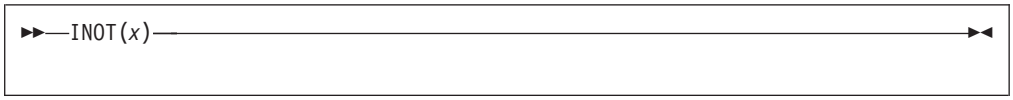
例

```
dcl tractatus char
      value( 'Wovon man nicht sprechen kann, ' ||
             'darueber muss man schweigen.' );

dcl pos fixed bin init(1);
```

```
pos = index( tractatus, 'man', pos+1 ); /* pos = 07 */

pos = index( tractatus, 'man', pos+1 ); /* pos = 46 */

pos = index( tractatus, 'man', pos+1 ); /* pos = 00 */
```

INOT

INOT は、*x* の論理 NOT を戻します。



x 式。 *n* は、計算タイプでなくてはなりません。

x が REAL FIXED BIN(p,0) の場合、その結果は REAL FIXED BIN(p,0) であり、*x* が UNSIGNED の場合、結果も UNSIGNED です。それ以外の場合は、*x* は SIGNED REAL FIXED BIN(M,0) に変換されて、その結果も同じ属性を持ちます。

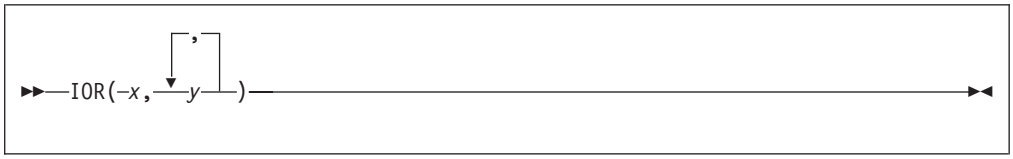
INOT(x) には *x* と反対の符号が付きますが、INOT(x) は *-x* と同じではありません。

例

```
inot(0)        /*  produces -1  */
inot(-1)       /*  produces  0  */
inot(+1)       /*  produces -2  */
```

IOR

IOR は、引数の論理 OR を戻します。



x および y
計算タイプを保持する必要がある式。

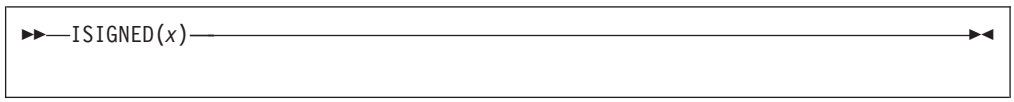
REAL FIXED BIN(p,0) ではない引数があると、その引数は SIGNED REAL FIXED BIN(M,0) に変換されます。

引数のいずれかが SIGNED の場合、UNSIGNED の引数はすべて SIGNED に変換されます。

結果は REAL FIXED BIN(max(p1,p2,...), 0) となります。すべての引数が UNSIGNED の場合は、結果も UNSIGNED です。

ISIGNED

ISIGNED(x) は、*x* のビット・パターンを変更せずに *x* を符号付き整数値にキャストした結果を戻します。



x 式。 *n* は、計算タイプでなくてはなりません。

x が整数でない場合 (つまり、*x* がスケール因数ゼロの REAL FIXED BIN でない場合)、*x* は REAL FIXED BIN(M,0) に変換されます。

ISIGNED(*x*) は、整数 *x* に対して、 *x* と同じビット・パターンを持つ値を戻しますが、その値の属性は SIGNED FIXED BIN(p) になります。

x が UNSIGNED の場合、p は次のように指定されます。

precision(*x*) = 8、16、32、または 64 の場合、 p = precision(*x*) - 1。そうでなければ、p = precision(*x*)。

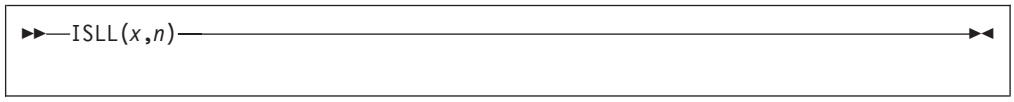
x が SIGNED の場合、p は *x* の精度に等しい値。

例

```
ISIGNED('ff_ff_ff_ff'xu) は、SIGNED FIXED BIN(31) 値 -1 と同等です。
```

ISLL

ISLL(x,n) は、論理的に *x* を左に *n* 桁シフトして、右側にゼロを埋め込んだ結果を戻します。



x 式。 *n* は、計算タイプでなくてはなりません。

n 式。 *n* は、計算タイプでなくてはなりません。

x が REAL FIXED BIN(p,0) であり、なおかつ、

- *x* が SIGNED の場合、その結果は SIGNED REAL FIXED BIN(M,0) です。
- *x* が UNSIGNED の場合、その結果は UNSIGNED REAL FIXED BIN(M+1,0) です。

それ以外の場合は、*x* は SIGNED REAL FIXED BIN(M,0) に変換されて、その結果も同じ属性を持ちます。

n が負の場合、または *n* が M より大きい場合は、結果は定義されません。

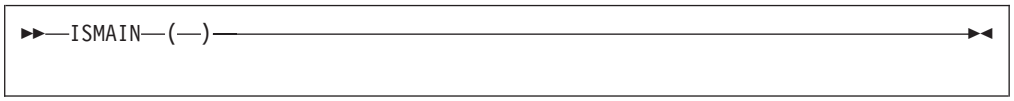
注: RAISE2(x,n) と異なり、ISLL(x,n) は、*x* とは別の符号が付く場合があります。

例

```
isll(+6,1)              /*  produces 12   */
isll(2147483645,1)      /*  produces  -6   */
```

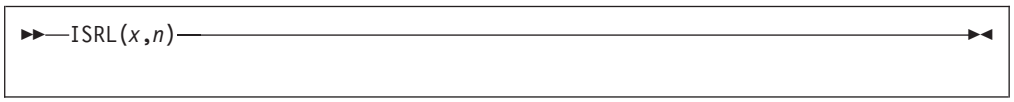
ISMAIN

ISMAIN()) は、呼び出し元のプロシージャーが OPTIONS(MAIN) 属性を持つ場合に、'1'B を戻します。そうでなければ、'0'B を戻します。



ISRL

ISRL(x,n) は、論理的に *x* を右に *n* 桁シフトして、左側にゼロを埋め込んだ結果を戻します。



x 式。 *n* は、計算タイプでなくてはなりません。

n 式。 *n* は、計算タイプでなくてはなりません。

x が REAL FIXED BIN(p,0) であり、なおかつ、

- *x* が SIGNED の場合、その結果は SIGNED REAL FIXED BIN(p,0) です。
- *x* が UNSIGNED の場合、その結果は UNSIGNED REAL FIXED BIN(p,0) です。

それ以外の場合は、*x* は SIGNED REAL FIXED BIN(M,0) に変換されて、その結果も同じ属性を持ちます。

n が負の場合、または *n* が M より大きい場合は、結果は定義されません。

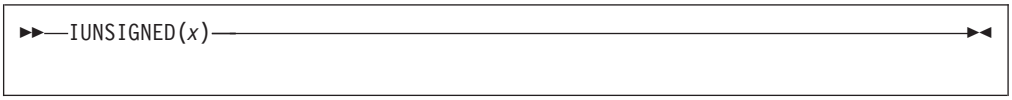
x が負でない場合は、ISRL(x,n) は LOWER2(x,n) と等しく、 *x* が負の場合は、ISRL(x,n) は *n*=0 でなければ正になります。

例

```
isrl(+6,1)           /*  produces 3           */
isrl(-6,1)           /*  produces 2147483645  */
```

IUNSIGNED

IUNSIGNED(x) は、*x* のビット・パターンを変更せずに *x* を符号なし整数値にキャストした結果を戻します。



x 式。 *n* は、計算タイプでなくてはなりません。

x が整数でない場合 (つまり、*x* がスケール因数ゼロの REAL FIXED BIN でない場合)、*x* は REAL FIXED BIN(M,0) に変換されます。

IUNSIGNED(*x*) は、整数 *x* に対して、 *x* と同じビット・パターンを持つ値を戻しますが、その値の属性は UNSIGNED FIXED BIN(p) になります。

x が SIGNED の場合、p は次のように指定されます。

precision(*x*) = 7、15、31、または 63 の場合、 p = precision(*x*) + 1。そうでなければ、p = precision(*x*)。

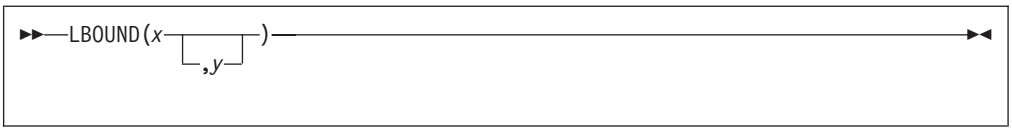
x が UNSIGNED の場合、p は *x* の精度に等しい値になります。

例

IUNSIGNED('ff_ff_ff_ff'xn) は、最大の UNSIGNED FIXED BIN(32) 値と同等です。

LBOUND

LBOUND は、*x* の次元 *y* の現行の下限を FIXED BINARY(31,0) 値で戻します。



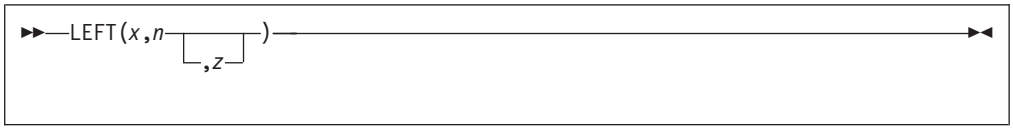
x 配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

y *x* の特定の次元を指定する式。必要な場合には、*y* は FIXED BINARY(31,0) に変換されます。 *y* は 1 以上でなければなりません。*y* が提供されない場合は、デフォルトを 1 にします。

配列が 1 次の場合にのみ、*y* を省略できます。

LEFT

LEFT は、ストリングの左端に、長さ *n* のストリング *x* を挿入し、必要なら文字 *z* で右側を埋めた結果のストリングを戻します。



x 式。 *x* に、計算タイプ (必須) と文字タイプを保持している必要があります。そうでない場合は、*x* は、CHARACTER に変換されます。

n 式。 *n* に、計算タイプ (必須) と文字タイプを保持している必要があります。*n* に属性 FIXED BINARY(31,0) がない場合は、*n* はこの属性に変換されます。

z 式。 *z* が指定された場合、*z* は CHARACTER(1) NONVARYING タイプでなければなりません。

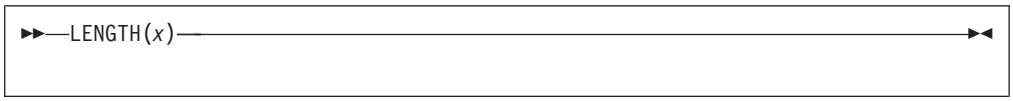
例

```
dcl Source char value('One Hundred SCIDS Marks');
dcl Target char(30);

Target = left (Source, length(Target), '*');
           /* 'One Hundred SCIDS Marks*******'               */
```

z が省略される場合は、埋め込み文字にブランクが使われます。

LENGTH

LENGTH は、*x* の現行の長さを指定するスケールのない REAL FIXED BINARY 値を戻します。



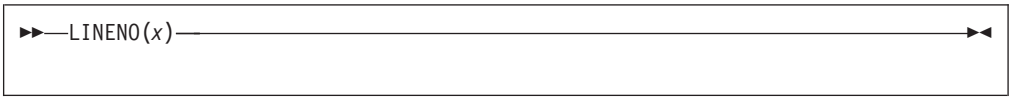
x ストリング式。 *x* が 2 進数の場合はビット・ストリングに変換され、それ以外の場合は、文字ストリングに変換されます。

LENGTH 組み込み関数の例については、464 ページの『MAXLENGTH』を参照してください。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

LINENO

LINENO は、*x* の現行の行数を指定するスケールのない REAL FIXED BINARY 値を戻します。



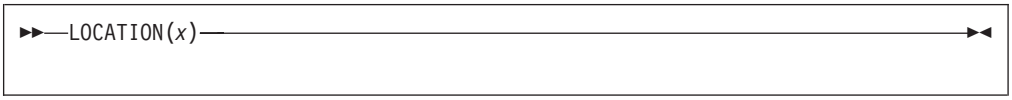
x ファイル参照。

ファイルは、オープン状態で、PRINT 属性を持っている必要があります。ファイルがオープンされていない場合、または PRINT 属性がない場合は、 '0'B が戻されます。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

LOCATION

LOCATION は、メンバー *x* があるレベル -1 の構造体または共用体内での、*x* のバイト位置を指定する FIXED BINARY(31,0) 値を戻します。



省略形: LOC

x 構造体または共用体のメンバー名 *x* が構造体または共用体のメンバーでない場合は、値 0 が戻されます。 *x* に BIT 属性がある場合は、LOCATION によって戻される値は、 *x* を含むバイト位置です。

x は、添え字であってはいけません。

LOCATION は、制限付きの式で使用できます。 LOC(x) が次のいずれかに設定するのに使われる場合は、 *y* の前に *x* を宣言する必要があります。
- 定数エクステントを保持する必要がある変数 *y* のエクステント
- 定数値を保持する必要がある変数 *y* の値

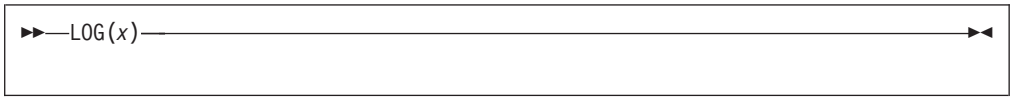
例

```
dcl 1 Table static,
      2 Tab2loc fixed bin(15) nonasgn init(loc(Tab2)),
                /* location is 0; gets initialized to 8  */
      2 Tab3loc fixed bin(15) nonasgn init(loc(Tab3)),
                /* location is 2; gets initialized to 808 */
      2 Length fixed bin nonasgn init(loc(End)),
                /* location is 4 */
      2 * fixed bin,
      2 Tab2(20,20)    fixed bin,
                /* location is 8 */
      2 Tab3(20,20)    fixed bin,
                /* location is 808 */

      2 F2_loc fixed bin nonasgn init(loc(F2)),
                /* location is 1608; gets initialized to  1612 */
```

```
2 F2_bitloc fixed bin nonasgn init(bitloc(F2)),
            /* location is 1610; gets initialized to 1 */

2 Flags,        /* location is 1612 */
  3 F1 bit(1),
  3 F2 bit(1),  /* bitlocation is 1 */
  3 F3 bit(1),
2 Bits(16) bit, /* location is 1613 */
2 End char(0);
```

LOG

LOG は、*x* の自然対数 (底 e の対数) の近似を浮動小数点の値で戻します。戻り値は、*x* の基数、モード、および精度になります。



x 式。 *x* は、ゼロより大きくなければなりません。

LOGF

LOGF は、次の点を除いては LOG とまったく同じです。

- LOGF は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 引数は実数表現でなければならない。
- 無効な引数が指定された場合、INVALIDOP 条件が発生したり、その他のハードウェア例外が生成されたりするなど、予期できない結果が生じることがある。
- 結果の正確度は、ハードウェアによって設定される。

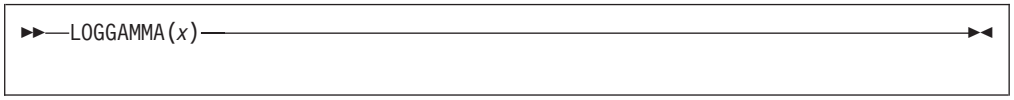
定義および構文については、『LOG』を参照してください。

LOGGAMMA

LOGGAMMA は、*x* のガンマ関数の対数の近似を浮動小数点の値で戻します。 *x* のガンマは、次の式で指定されます。

$gamma(x) = \int_0^\infty (u^{x-1})(e^{-x})du$

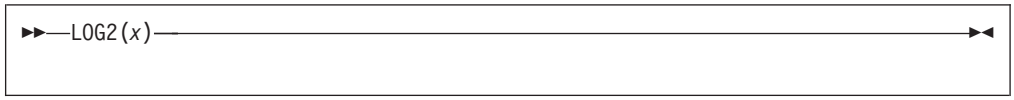
LOGGAMMA は、*x* の基数、モード、および精度になります。



x 実数式。 *x* の値は、ゼロより大きくなければなりません。

LOG2

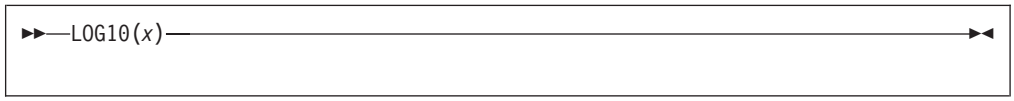
LOG2 は、*x* の 2 進対数 (底 2 の対数) の近似を実数の浮動小数点で戻します。戻り値は、*x* の基数と精度になります。



x 実数式。 *x* の値は、ゼロより大きくなければなりません。

LOG10

LOG10 は、*x* の常用対数 (底 10 の対数) の近似を実数の浮動小数点で戻します。戻り値は、*x* の基数と精度になります。



x 実数式。この値はゼロより大きくなければなりません。

LOG10F

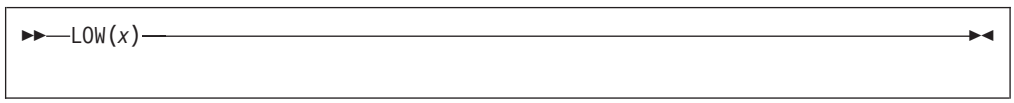
LOG10F は、次の点を除いては LOG10 とまったく同じです。

- LOG10F は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 引数は実数表現でなければならない。
- 無効な引数が指定された場合、INVALIDOP 条件が発生したり、その他のハードウェア例外が生成されたりするなど、予期できない結果が生じることがある。
- 結果の正確度は、ハードウェアによって設定される。

定義および構文については、『LOG10』を参照してください。

LOW

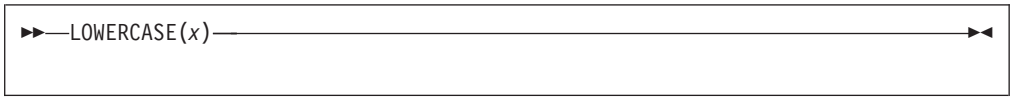
LOW は、長さ *x* の文字ストリングを戻します。各文字は、照合順序の最下位の文字 (16 進数の 00) です。



x 式。必要な場合には、*x* は正の実数固定小数点の 2 進数の値に変換されます。*x* = 0 の場合は、結果はヌル文字ストリングです。

LOWERCASE

LOWERCASE は、A から Z までの英字すべてを同等な小文字に変換した文字ストリングを戻します。



x 式。必要に応じ、*x* は文字に変換されます。

LOWERCASE(x) は次のステートメントと同等です。

```
TRANSLATE( x,
           'abcdefghijklmnopqrstuvwxyz',
           'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  )
```

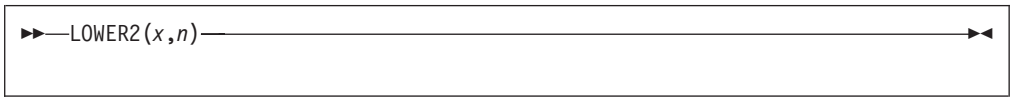
LOWER2

LOWER2(x,n) は、次の値を戻します。

```
          -n
floor(x * (2  ))
```

LOWER2(x,n) は、次の値を戻します。

```
                -n
floor(x * (2  ))
```



注: LOWER2(x,n) は、アセンブラー SRA(x,n) と同等です。

x 式。 *n* は、計算タイプでなくてはなりません。

n 式。 *n* は、計算タイプでなくてはなりません。

x が SINGED REAL FIXED BIN(p,0) の場合、その結果は同じ属性です。それ以外の場合は、*x* は SIGNED REAL FIXED BIN(M,0) に変換されて、その結果も同じ属性を持ちます。

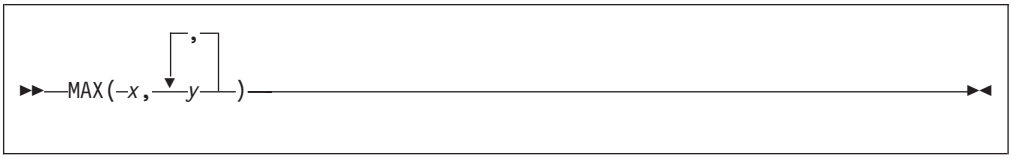
n が負の場合、または *n* が M より大きい場合は、結果は定義されません。

例

```
lower2 (+6,1)                    /*  Produces 3   */

lower2 (-6,1)                    /*  Produces -3  */

lower2 (-7,1)                    /*  Produces -4  */
```

MAX

MAX は、複数の式の集合から最大値を戻します。



x および y
式。

すべての引数は、実数でなければなりません。結果は、引数の共通の基数およびスケールを持つ実数です。

引数が次の精度を使用した固定小数点の引数である場合は、次のようになります。

```
(p1,q1),(p2,q2),...,(pn,qn)
```

結果の精度は、次の式によって指定されます。

```
(min(N,max(p1-q1,p2-q2,...,pn-qn) + max(q1,q2,...,qn)),max(q1,q2,...,qn))
```

この場合 N は、使用できる最大の桁数です。

引数が次の精度を使用した浮動小数点である場合は、次のようになります。

```
p1,p2,p3,...pn
```

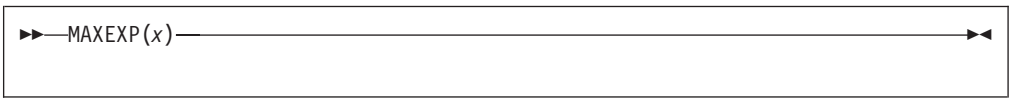
結果の精度は、次の式によって指定されます。

```
max(p1,p2,p3,...pn)
```

許可される引数の最大数は 64 です。

MAXEXP

MAXEXP は、EXPONENT(x) が取ることができる最大値 FIXED BINARY(31,0) 値を戻します。



x 式。 *x* は、REAL 属性および FLOAT 属性をとる必要があります。

例 (Intel の値)

```
maxexp(x) = 00128      for x float bin(p), p <= 21
maxexp(x) = 01024      for x float bin(p), 21 < p <= 53
maxexp(x) = 16384      for x float bin(p), 53 < p

maxexp(x) = 00128      for x float dec(p), p <= 6
maxexp(x) = 01024      for x float dec(p), 6 < p <= 16
maxexp(x) = 16384      for x float dec(p), 16 < p
```

例 (AIX の値)

```
maxexp(x) = 0128      for x float bin(p), p <= 21
maxexp(x) = 1024      for x float bin(p), 21 < p <= 53
maxexp(x) = 1024      for x float bin(p), 53 < p

maxexp(x) = 0128      for x float dec(p), p <= 6
maxexp(x) = 1024      for x float dec(p), 6 < p <= 16
maxexp(x) = 1024      for x float dec(p), 16 < p
```

例 (z/OS の 16 進値)

```
maxexp(x) = 63      for x float bin(p), p <= 21
maxexp(x) = 63      for x float bin(p), 21 < p <= 53
maxexp(x) = 63      for x float bin(p), 53 < p

maxexp(x) = 63      for x float dec(p), p <= 6
maxexp(x) = 63      for x float dec(p), 6 < p <= 16
maxexp(x) = 63      for x float dec(p), 16 < p
```

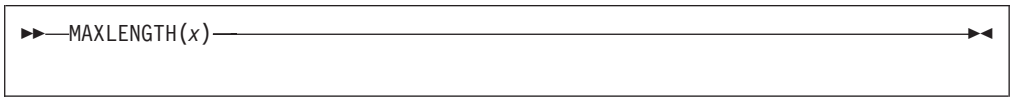
例 (z/OS の IEEE 値)

```
maxexp(x) = 128     for x float bin(p), p <= 21
maxexp(x) = 1024    for x float bin(p), 21 < p <= 53
maxexp(x) = 16384   for x float bin(p), 53 < p

maxexp(x) = 128     for x float dec(p), p <= 6
maxexp(x) = 1024    for x float dec(p), 6 < p <= 16
maxexp(x) = 16384   for x float dec(p), 16 < p
```

MAXEXP(x) は定数で、制限付き式に使用できます。

MAXLENGTH

MAXLENGTH は、ストリングの最大長を戻します。



x 式。 *x* に、計算タイプ (必須) とストリング・タイプを保持している必要があります。そうでないと、文字に変換されます。

例

```
dcl x char(20);
dcl y char(20) varying;

x, y = '';

x = copy( '*', length(x) );    /* fills x with '*'   */
y = copy( '*', length(y) );    /* leaves y unchanged */

x = copy( '-', maxlength(x) ); /* fills x with '-'   */
y = copy( '-', maxlength(y) ); /* fills y with '-'   */
```

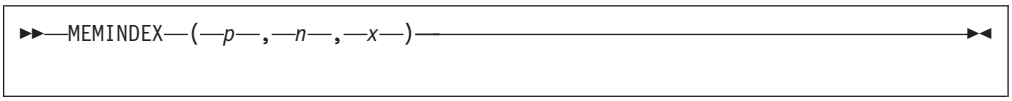
y の最初の割り当ては、未変更のままになります。上記のコードの断片で使用される場合、*length(y)* は、ゼロを戻す（*y* は VARYING であり、事前に ’ ’ とセットされている）ためです。

しかし、*y* への 2 番目の割り当てでは、*maxlength(y)* が 20（*y* の宣言された長さ）を戻すため、- 符号が 20 個入ります。

MEMINDEX

MEMINDEX は、指定したサブストリングのバッファー内で開始位置を示すスケールのない REAL FIXED BINARY 値を戻します。

3 つの引数を使用すると、関数の構文は以下のようになります。



p 検索されるバッファーのアドレス。

n 検索されるバッファーの長さ。

x 検索のターゲットとして使用するストリング式。

4 つの引数を使用すると、関数の構文は以下のようになります。

p 検索される最初のバッファーのアドレス。

n 検索される最初のバッファーの長さ。

q 検索のターゲットとして使用する 2 番目のバッファーのアドレス。

m 検索のターゲットとして使用する 2 番目のバッファーの長さ。

バッファー長は、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

バッファー長は、負数であってはなりません。

3 つの引数を使用する場合は、ターゲットのストリング式は、タイプ CHARACTER (PICTURE を含む)、GRAPHIC、または WIDECHAR を保持している必要があります。バッファー長は、そのストリング・タイプの単位の数として解釈されます。

4 つの引数を使用する場合は、バッファー長はバイト数を指定し、文字検索が行われます。

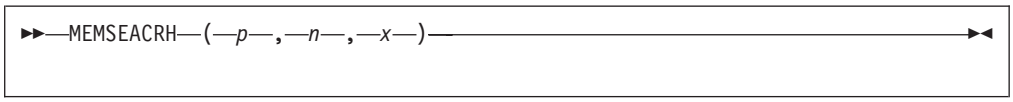
VARYING または VARYINGZ のストリング *X* およびストリング *Y* では、関数 MEMINDEX(ADDRDATA(X)、LENGTH(X)、Y) は、INDEX(X, Y) と同じ値を戻します。

例

```
dcl cb(128*1024) char(1);
dcl wb(128*1024) widechar(1);
dcl pos fixed bin(31);
/* 128K bytes searched for the character string 'test' */
pos = memindex( cb, stg(cb), 'test' );
/* 256K bytes searched for the string 'test' as widechar */
pos = memindex( wb, stg(wb), wchar('<') );
```

MEMSEARCH

MEMSEARCH は、指定のストリングにある任意の文字、グラフィック、またはワイド文字がバッファーにある場合、そのバッファー内の (左側の) 最初の位置を指定するスケールのない REAL FIXED BINARY 値を戻します。



p 検索されるバッファーのアドレス。

n 検索されるバッファーの長さ。

x ストリング式。

バッファー長は、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

バッファー長は、負数であってはなりません。

ストリング式 *x* は、タイプ CHARACTER (PICTURE を含む)、GRAPHIC、または WIDECHAR を保持している必要があります。バッファー長は、そのストリング・タイプの単位の数として解釈されます。

アドレス *p* および長さ *n* は、*x* に存在する任意の文字、グラフィック、ワイド文字を検索する「ストリング」を指定します。

バッファー長 *n* がゼロであるか、または *x* がヌル・ストリングである場合は、結果はゼロです。

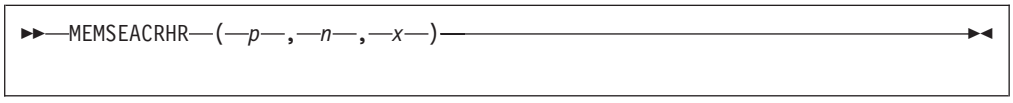
バッファーに *x* がない場合は、結果はゼロです。

例

```
dcl cb(128*1024) char(1);
dcl wb(128*1024) widechar(1);
dcl pos fixed bin(31);

/* 128K bytes searched from the left for a numeric */
pos = memsearch( cb, stg(cb), '012345789' );

/* 256K bytes searched from the left for a widechar '0' or '1' */
pos = memsearch( wb, stg(wb), '0030_0031'wx );
```

MEMSEARCHR

MEMSEARCHR は、指定のストリングにある任意の文字、グラフィック、またはワイド文字がバッファーにある場合、そのバッファー内の (右側の) 最初の位置を指定するスケールのない REAL FIXED BINARY 値を戻します。



p 検索されるバッファーのアドレス。

n 検索されるバッファーの長さ。

x ストリング式。

バッファー長は、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

バッファー長は、負数であってはなりません。

ストリング式 *x* は、タイプ CHARACTER (PICTURE を含む)、GRAPHIC、または WIDECHAR を保持している必要があります。バッファー長は、そのストリング・タイプの単位の数として解釈されます。

アドレス *p* および長さ *n* は、*x* に存在する任意の文字、グラフィック、ワイド文字を検索する「ストリング」を指定します。

バッファー長 *n* がゼロであるか、または *x* がヌル・ストリングである場合は、結果はゼロです。

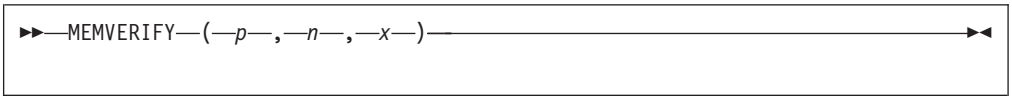
バッファーに *x* がない場合は、結果はゼロです。

例

```
dcl cb(128*1024) char(1);
dcl wb(128*1024) widechar(1);
dcl pos fixed bin(31);

/* 128K bytes searched from the right for a numeric */
pos = memsearchr( cb, stg(cb), '012345789' );

/* 256K bytes searched from the right for a widechar '0' or '1' */
pos = memsearchr( wb, stg(wb), '0030_0031'wx );
```

MEMVERIFY

MEMVERIFY は、指定したストリング内にはない 文字、グラフィック、またはワイド文字の (左側から) 最初のバッファーで位置を指定するスケールのない REAL FIXED BINARY 値を戻します。



p 検索されるバッファーのアドレス。

n 検索されるバッファーの長さ。

x ストリング式。

バッファー長は、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

バッファー長は、負数であってはなりません。

ストリング式 *x* は、タイプ CHARACTER (PICTURE を含む)、GRAPHIC、または WIDECHAR を保持している必要があります。バッファー長は、そのストリング・タイプの単位の数として解釈されます。

アドレス *p* および長さ *n* は、*x* に存在しない任意の文字、グラフィック、ワイド文字を検索する「ストリング」を指定します。

バッファー長 *n* がゼロであるか、または *x* がヌル・ストリングである場合は、結果はゼロです。

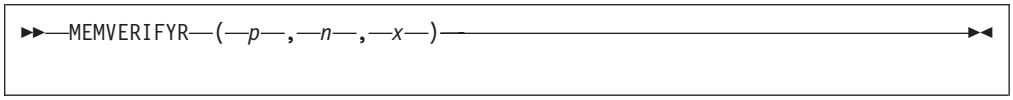
バッファー内のすべての文字、グラフィック、またはワイド文字が *x* に存在する場合は、結果はゼロです。

例

```
 dcl cb(128*1024) char(1);
dcl wb(128*1024) widechar(1);
dcl pos fixed bin(31);

/* 128K bytes searched from the left for a non-numeric */
pos = memverify( cb, stg(cb), '012345789' );

/* 256K bytes searched from the left for the a non-blank widechar */
pos = memverify( wb, stg(wb), '0020'wx );
```

MEMVERIFYR

MEMVERIFYR は、指定したストリング内にはない 文字、グラフィック、またはワイド文字の (右側から) 最初のバッファーで位置を指定するスケールのない REAL FIXED BINARY 値を戻します。



p 検索されるバッファーのアドレス。

n 検索されるバッファーの長さ。

x ストリング式。

バッファー長は、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

バッファー長は、負数であってはなりません。

ストリング式 *x* は、タイプ CHARACTER (PICTURE を含む)、GRAPHIC、または WIDECHAR を保持している必要があります。バッファー長は、そのストリング・タイプの単位の数として解釈されます。

アドレス *p* および長さ *n* は、*x* に存在しない任意の文字、グラフィック、ワイド文字を検索する「ストリング」を指定します。

バッファー長 *n* がゼロであるか、または *x* がヌル・ストリングである場合は、結果はゼロです。

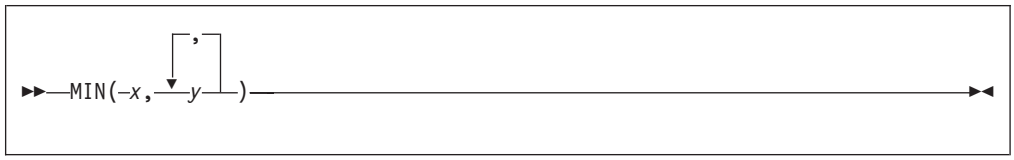
バッファー内のすべての文字、グラフィック、またはワイド文字が *x* に存在する場合は、結果はゼロです。

例

```
dcl cb(128*1024) char(1);
 dcl wb(128*1024) widechar(1);
 dcl pos fixed bin(31);

 /* 128K bytes searched from the right for a non-numeric */
 pos = memverify( cb, stg(cb), '012345789' );

 /* 256K bytes searched from the right for the a non-blank widechar */
 pos = memverify( wb, stg(wb), '0020'wx );
```

MIN

MIN は、1 つ以上の式の集合から、最小値を戻します。



x および y
式。

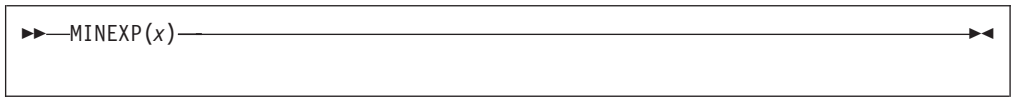
すべての引数は、実数でなければなりません。結果は、引数の共通の基数およびスケールを持つ実数です。

結果の精度は、463 ページの『MAX』の説明と同じです。

許可される引数の最大数は 64 です。

MINEXP

MINEXP は、EXPONENT(x) が取ることができる最小値の FIXED BINARY(31,0) 値を戻します。



x 式。 *x* は、REAL 属性および FLOAT 属性をとる必要があります。

MINEXP(x) は定数で、制限付き式に使用できます。

例 (Intel の値)

```
minexp(x) = -00125    for x float bin(p), p <= 21
minexp(x) = -01021    for x float bin(p), 21 < p <= 53
minexp(x) = -16831    for x float bin(p), 53 < p
```

```
minexp(x) = -00125    for x float dec(p), p <= 6
minexp(x) = -01021    for x float dec(p), 6 < p <= 16
minexp(x) = -16831    for x float dec(p), 16 < p
```

例 (AIX の値)

```
minexp(x) = -0125     for x float bin(p), p <= 21
minexp(x) = -1021     for x float bin(p), 21 < p <= 53
minexp(x) = -0968     for x float bin(p), 53 < p

minexp(x) = -0125     for x float dec(p), p <= 6
minexp(x) = -1021     for x float dec(p), 6 < p <= 16
minexp(x) = -0968     for x float dec(p), 16 < p
```

例 (z/OS の 16 進値)

```
minexp(x) = -64    for x float bin(p), p <= 21
minexp(x) = -64    for x float bin(p), 21 < p <= 53
minexp(x) = -50    for x float bin(p), 53 < p

minexp(x) = -64    for x float dec(p), p <= 6
minexp(x) = -64    for x float dec(p), 6 < p <= 16
minexp(x) = -50    for x float dec(p), 16 < p
```

例 (z/OS の IEEE 値)

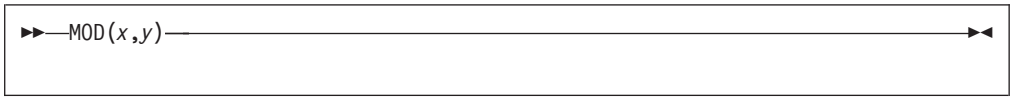
```
maxexp(x) = -125    for x float bin(p), p <= 21
minexp(x) = -1021   for x float bin(p), 21 < p <= 53
minexp(x) = -16381  for x float bin(p), 53 < p

minexp(x) = -125    for x float dec(p), p <= 6
minexp(x) = -1021   for x float dec(p), 6 < p <= 16
minexp(x) = -16381  for x float dec(p), 16 < p
```

MOD

MOD は、次を満たす負以外の最小値 R を戻します。

```
(x - R)/y = n
```

n は、整数値です。すなわち、R は *y* で割り切れるように、*x* から減算する負でない最小値です。



x 実数式。

y 実数式。 *y* = 0 の場合は、ZERODIVIDE 条件が発生します。

結果 R は、引数と共通の基数およびスケールを持つ実数です。結果が浮動小数点の場合は、精度は *x* および *y* の大きい方になります。結果が固定小数点の場合は、精度は次の式で指定されます。

```
(min(n,p2-q2+max(q1,q2)),max(q1,q2))
```

(p1,q1) と (p2,q2) は、それぞれ *x* と *y* の精度であり、n には、FIXED DECIMAL の場合は N が入り、FIXED BINARY の場合は M が入ります。

x および *y* のスケール因数が異なる固定小数点の場合は、R が計算される前に、小さいスケール因数の引数が大きいスケール因数に変換されます。変換が失敗すると、結果は不定になります。

次の例では、MOD 組み込み関数と REM 組み込み関数を対比しています。

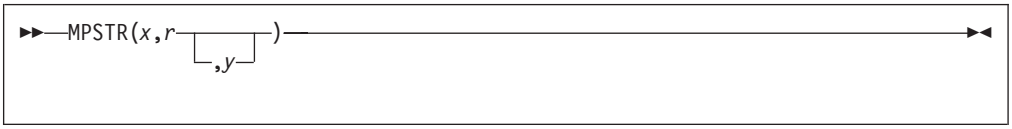
REM 組み込み関数については、497 ページの『REM』を参照してください。

例

```
rem( +10, +8 ) = 2
mod( +10, +8 ) = 2

rem( +10, -8 ) = 2
mod( +10, -8 ) = 2

rem( -10, +8 ) = -2
mod( -10, +8 ) = 6

rem( -10, -8 ) = -2
mod( -10, -8 ) = 6
```

MPSTR

MPSTR は、論理境界でストリングを切り捨て、混合文字ストリングを戻します。たとえば、MPSTR は、2 バイト文字をバイト間では切り捨てません。戻されるストリングの長さは、式 *x* の長さ、または *y* で指定された値と等しくなります。ストリングの処理は、式 *r* で選択された規則によって次のように判別されます。



x 文字ストリングの結果が出る式。必要な場合には、*x* の値は文字に変換されます。

r 1 文字の結果が出る式。式は GRAPHIC であることはできません。必要な場合には、文字に変換されます。

式 *r* は、ストリングの処理に使われる規則を指定します。 *r* で使われる文字およびその規則を次に示します。

V または v
混合ストリング *x* を検証し、混合ストリングを戻します。

S または s
ヌル DBCS ストリングを除去して新しいストリングを作成し、混合ストリングを戻します。

V および S の両方が指定される場合は、順番が指定された順番に関係なく、V は S より優先されます。

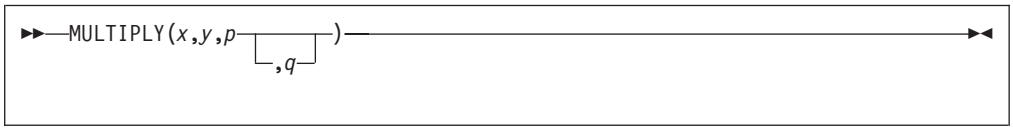
V を指定せずに S が指定された場合は、ストリング x は有効なストリングであると想定されます。ストリングが無効の場合は、結果は不定になります。

注: Intel および AIX では、パラメーター *r* は無視されます。

y 式。必要な場合には、*y* は、実数の固定小数点の 2 進数の値に変換されます。*y* が省略される場合は、長さはタイプ変換の規則によって決められます。 *y* の値は、負であってはいけません。 *y* = 0 の場合は、結果はヌル文字ストリングです。 *y* が *x* を入れるのに必要な長さより大きい場合は、結果は ブランクで埋められます。 *y* が *x* を入れるのに必要な長さより小さい場合は、結果は右から超過した文字を廃棄する (SBCS 文字の場合) か、必要な DBCS 文字 (2 バイト一組) を廃棄して切り捨てます。

MULTIPLY

MULTIPLY は、*x* と *y* の積を *p* と *q* で指定された精度で戻します。結果の基数、スケール、およびモードは、PRECTYPE コンパイラー・オプションによって規則が変更されないかぎり、式の評価規則によって決まります。



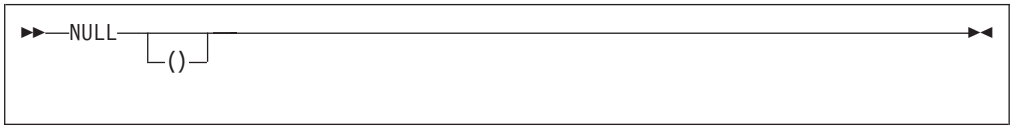
x および y
式。

p 演算を通して維持する桁数を指定する制限付き式。

q 結果のスケール因数を指定する制限付き式。固定小数点の場合に、*q* が省略されると、スケール因数ゼロが想定されます。結果が浮動小数点の場合は、*q* は省略します。

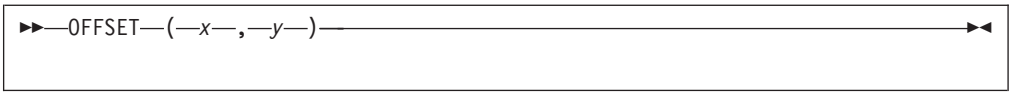
NULL

NULL は、ヌル・ポインター値を戻します。ヌル・ポインター値は、変数の生成を識別しません。ヌル・ポインター値は、ハンドルに割り当てられ、ハンドルと比較されます。ヌル・ポインター値は、組み込み関数の値のオフセット変数への割り当てによって OFFSET に変換されます。



OFFSET

OFFSET は、ポインター参照 *x* から派生するオフセット値および区域 *y* に関連したオフセット値を戻します。x がヌル・ポインター値の場合は、ヌル・オフセット値が戻されます。



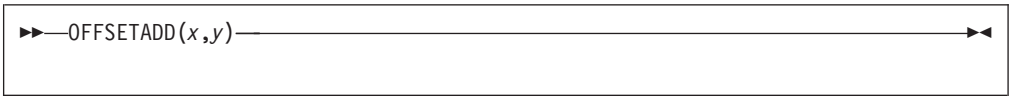
x ポインター参照。 *x* は、区域 *y* 内で基底付き変数の生成を識別するか、ヌル・ポインター値でなければなりません。

y 区域参照。

x がエレメント参照の場合は、*y* は要素変数でなければなりません。

OFFSETADD

OFFSETADD は、引数の合計を戻します。

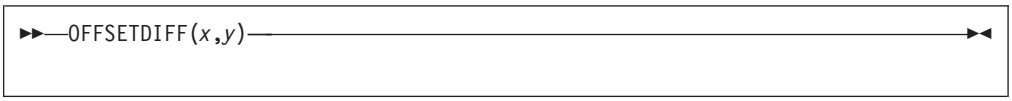


x 式。 *x* は、OFFSET として指定されなければなりません。

y 式。 *y* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

OFFSETDIFF

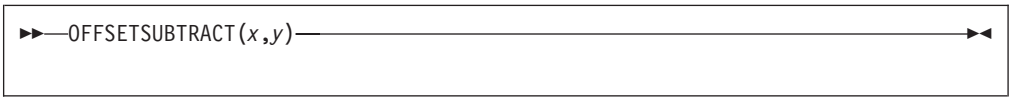
OFFSETDIFF は、引数間の算術差を表す FIXED BINARY(31,0) 値を戻します。



x および y
式。両方とも OFFSET として指定されなければなりません。

OFFSETSUBTRACT

OFFSETSUBTRACT は、OFFSETADD(x,-y) と同等です。

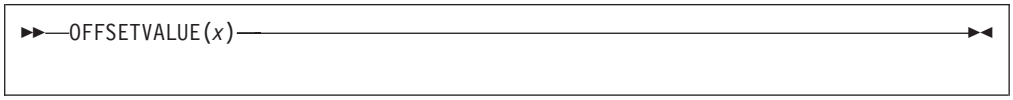


x 式。 *x* は、OFFSET として指定されなければなりません。

y 式。 *y* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

OFFSETVALUE

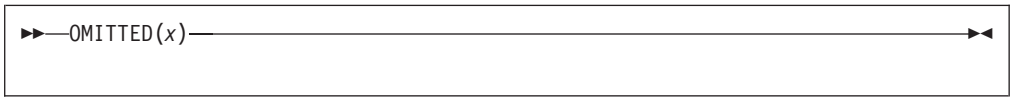
OFFSETVALUE は、*x* の変換された値であるオフセット値を戻します。



x 式。 *x* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

OMITTED

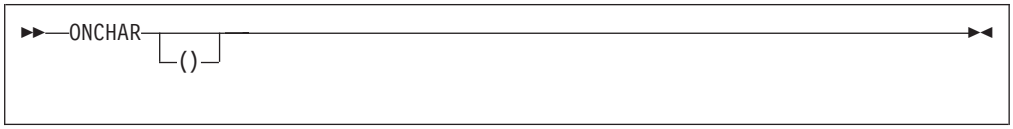
OMITTED が含まれているプロシージャーの呼び出しで、*x* という名前のパラメーターが省略される場合は、OMITTED は '1'B である BIT(1) 値を戻します。



x BYADDR 属性付きのレベル -1 の添え字なしパラメーター。

ONCHAR

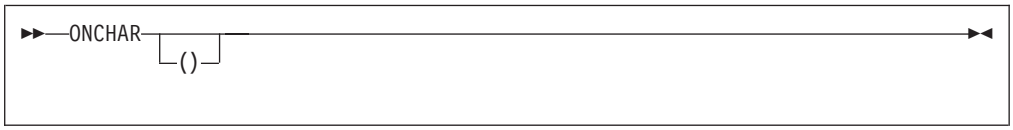
ONCHAR は、CONVERSION 条件が発生する原因となる文字を含む character(1) ストリングを戻します。この関数は、CONVERSION 条件の ON ユニットの中か、CONVERSION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニット (またはそれらの ON ユニットから動的に派生したもの) の中にあります。



ONCHAR 組み込み関数がコンテキスト以外で使われる場合は、ブランクが戻されます。

ONCHAR 疑似変数

ONCHAR 疑似変数は、ONCHAR 組み込み関数の現行値を設定します。疑似変数に割り当てられたエレメントの値は、長さ 1 の文字値に変換されます。再変換されるときには、新しい文字が使われます (77 ページの『第 5 章 データ変換』の変換を参照してください)。

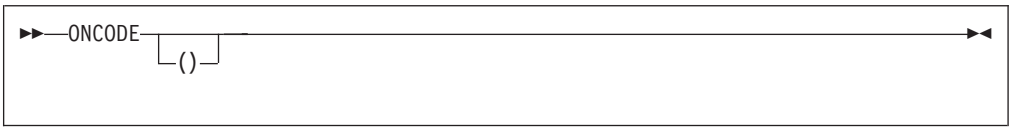


疑似変数は、コンテキスト以外では使用してはいけません。

ONCODE

ONCODE 組み込み関数は、最後の条件の原因によって決まる固定小数点の 2 進数の値を提供します。 ONCODE は、特定の条件、たとえば ERROR 条件を発生するさまざまな状況を区別するのに使用できます。検出された条件およびエラーに対応するコードについては、特定の条件を参照してください。

ONCODE は、条件コードである実数の固定小数点の 2 進数の値を戻します。 ONCODE は、ON ユニットまたはその動的子孫のコンテキストにあります。条件コードのすべての定義は「メッセージおよびコード」に記載されています。

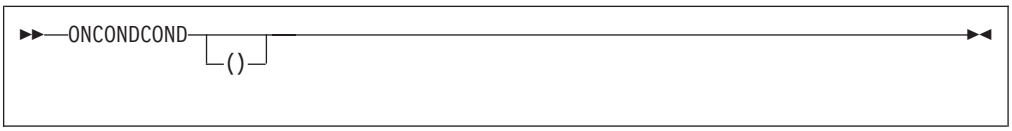


コンテキスト以外で ONCODE が使われている場合は、ゼロが戻されます。

ONCONDCOND

ONCONDCOND は、値が CONDITION 条件が発生する条件の名前である不変文字ストリングを戻します。名前が DBCS 名の場合は、ONCONDCOND は混合文字ストリングとして戻されます。 ONCONDCOND は、次の状況のコンテキストにあります。

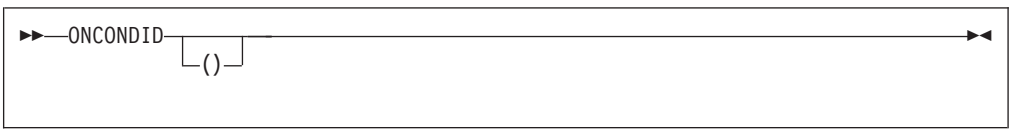
- CONDITION ON ユニットまたはそれらから動的に派生したもの
- CONDITION 条件をトラップする ANYCONDITION ON ユニット、またはこのような ON ユニットから動的に派生したもの



コンテキスト以外で ONCONDCOND が使われている場合は、ヌル・ストリングが戻されます。

ONCONDID

ONCONDID (ON 条件 ID の省略形) は、ON ユニットでハンドルされている条件を識別する FIXED BINARY(31,0) 値を戻します。 ONCODE は、ON ユニットまたはそれらから動的に派生したものの 1 つのコンテキストにあります。



ONCONDID によって戻される値は、次の DECLARE ステートメントで指定されます。

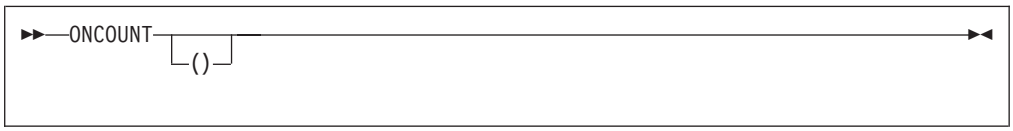
ONCONDID

```
declare (   condid_area              value(1),
            condid_attention         value(2),
            condid_condition         value(3),
            condid_conversion        value(4),
            condid_endfile           value(5),
            condid_endpage           value(6),
            condid_error             value(7),
            condid_finish            value(8),
            condid_fixedoverflow     value(9),
            condid_invalidop         value(10),
            condid_key               value(11),
            condid_name              value(12),
            condid_overflow          value(13),
            condid_record            value(14),
            condid_size              value(15),
            condid_storage           value(16),
            condid_stringrange       value(17),
            condid_stringsize        value(18),
            condid_subscriptrange    value(19),
            condid_transmit          value(20),
            condid_undefinedfile     value(21),
            condid_underflow         value(22),
            condid_zerodivide        value(23)
        ) fixed bin(31);
```

コンテキスト以外で ONCONDID が使われている場合は、ゼロが戻されます。

ONCOUNT

ONCOUNT は、ON ユニットが入力されたときに、依然としてハンドルされる条件の数を指定するスケールのない FIXED BINARY 値を戻します (365 ページの『複数条件』を参照)。ONCOUNT は、ON ユニットまたは ON ユニットから動的に派生したものの中にあります。

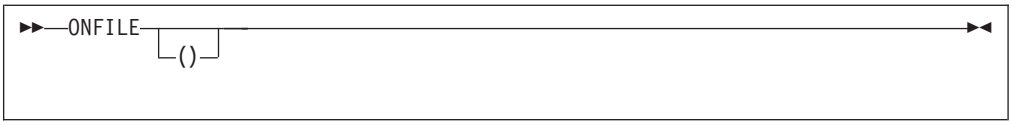


コンテキスト以外で ONCOUNT が使われている場合は、ゼロが戻されます。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

ONFILE

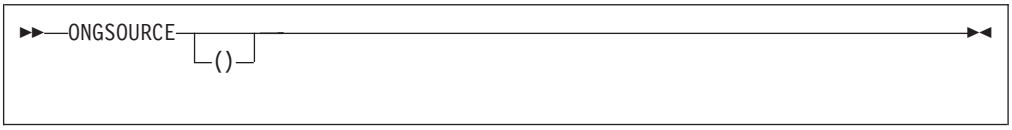
ONFILE は、値が入力条件または出力条件が発生するファイル名である文字ストリングを戻します。名前が DBCS 名の場合は、ONFILE は混合文字ストリングとして戻されます。また、ONFILE は、入力条件または出力条件か、入力条件または出力条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニット (またはそれらから動的に派生したもの) の中にあります。



コンテキスト以外で ONFILE が使われている場合は、ヌル・ストリングが戻されます。

ONGSOURCE

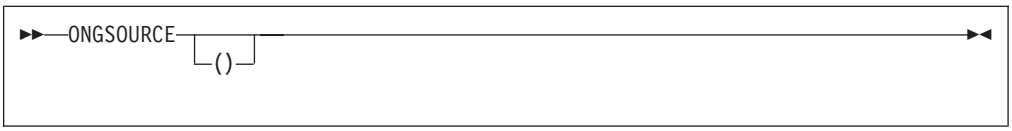
ONGSOURCE は、CONVERSION 条件が発生する原因となる DBCS 文字を含む漢字ストリングを戻します。また、この関数は、CONVERSION 条件か、CONVERSION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニット (またはそれらから動的に派生したもの) の中にあります。



ONGSOURCE 組み込み関数がコンテキスト以外で使われる場合は、ヌル GRAPHIC ストリングが戻されます。

ONGSOURCE 疑似変数

ONGSOURCE 疑似変数は、ONGSOURCE 組み込み関数の現行値を設定します。疑似変数に割り当てられたエレメントの値は、グラフィックに変換されます。再変換されるときには、このストリングが使われます。



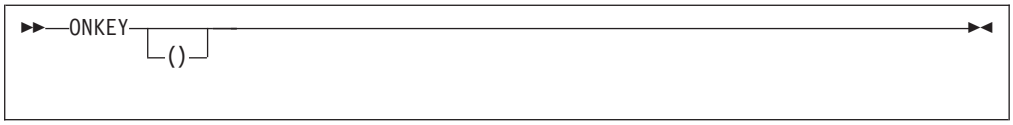
疑似変数は、コンテキスト以外では使用してはいけません。

ONKEY

ONKEY は、値が入出力条件が発生するレコードのキーである文字ストリングを戻します。索引付きファイルの場合に、キーが GRAPHIC の場合は、ストリングは混合文字ストリングとして戻されます。 ONKEY は、次のコンテキストにあります。

- ON ユニットまたはその動的子孫
- 入出力条件 (ENDFILE 以外)
- 入出力条件の暗黙処置として発生した ERROR 条件または FINISH 条件

ONKEY は、条件が発生するステートメントで KEY オプション、KEYTO オプション、または KEYFROM オプションが指定されていない場合でも、必ず、KEYED ファイルでの演算に設定されます。



ONKEY の指定結果を次に示します。

- 入出力条件 (ENDFILE 以外) の場合、または、ERROR 条件か FINISH 条件の暗黙処置として発生したこれらの条件の場合は、結果はエラーを発生する入出力ステートメントで記録されたキーの値です。
- 相対データ・セットの場合は、結果は相対レコード数の文字ストリング表示です。キーが間違えて指定された場合は、結果はソース・キーの最後の 8 文字です。ソース・キーが 8 文字より小さい場合は、8 文字になるように右側がブランクで埋まります。キーが正しく指定された場合は、文字ストリングは、必要ならば、左側がブランクで埋められた文字フォーマットの相対レコード数で構成されます。
- 更新済みレコードのキーが入力レコードのキーと異なるときに、索引付きデータ・セットに更新済みレコードを書き込む REWRITE ステートメントの場合は、結果は入力レコードの組み込みキーの値です。

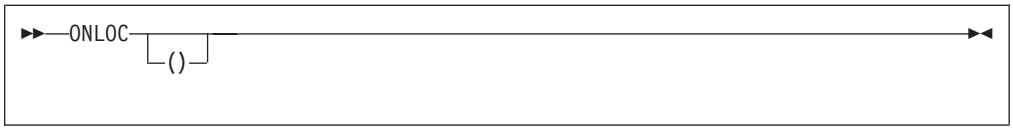
コンテキスト以外で ONKEY が使われている場合は、ヌル・ストリングが戻されます。

ONLOC

ONLOC は、値が条件が発生するプロシージャーの現行呼び出しに使われるエントリー・ポイントの名前である文字ストリングを戻します。

ONLOC は、CALL ステートメントまたは GOTO ステートメントに名前が表示されていても、常に複数のラベル指定の左端の名前を戻します。

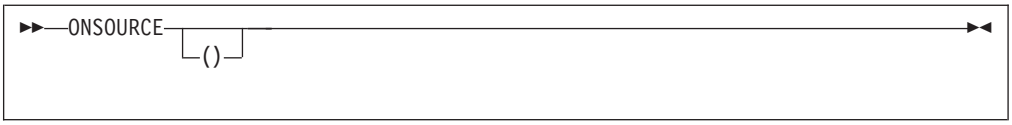
名前が DBCS 名の場合は、ONLOC は混合文字ストリングとして戻されます。ONLOC は、ON ユニットまたはその動的子孫のコンテキストにあります。



コンテキスト以外で ONLOC が使われている場合は、ヌル・ストリングが戻されます。

ONSOURCE

ONSOURCE は、値が CONVERSION 条件が発生したときに処理中のフィールドの内容である文字ストリングを戻します。また、この関数は、CONVERSION 条件か、CONVERSION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニット (またはそれらから動的に派生したもの) の中にあります。

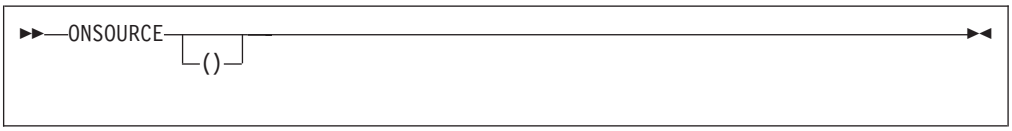


コンテキスト以外で ONSOURCE が使われている場合は、ヌル・ストリングが戻されます。

失敗した変換の中のソースが COMPLEX 値である場合、ONSOURCE() はその値の REAL 部分または IMAG 部分のみを表示します。

ONSOURCE 疑似変数

ONSOURCE 疑似変数は、ONSOURCE 組み込み関数の現行値を設定します。疑似変数に割り当てられたエレメントの値は、文字ストリングに変換されます。また、必要な場合には、CONVERSION 条件が発生したフィールドの長さと一致するように、右側がブランクで埋められるか切り捨てられます。再変換されるときには、このストリングが使われます。



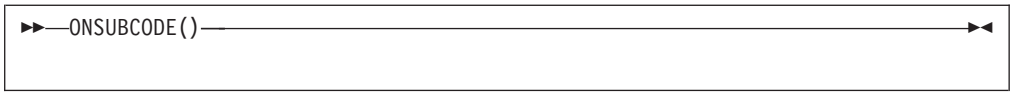
変換が再試行されるときは、疑似変数に割り当てられたストリングは、単一データ項目として処理されます。このため、エラーの訂正処理は、GET LIST ステートメントまたは GET DATA ステートメントの実行時に変換が発生するときに、複数のデータ項目を含むストリングを割り当てることはできません。ストリングにブランクまたはコンマがあると、CONVERSION が再発生します。

疑似変数は、コンテキスト以外では使用してはいけません。

ONSOURCE が 2 進定数でない場合は、ONSOURCE 疑似変数を 1 に設定してはなりません。たとえば、ONSOURCE() が 'ERR' である場合、ONSOURCE() を '0'B に設定してはなりません。

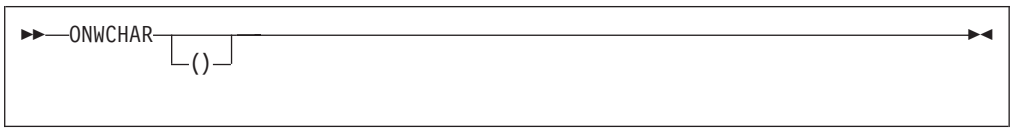
ONSUBCODE

ONSUBCODE は、発生した入出力エラーに関する詳細を提供する FIXED BINARY(31,0) を戻します。これは、メッセージ IBM0236I および IBM0265I のために文書化された SUBCODE1 値に対応します。これらの値の定義は「メッセージおよびコード」に記載されています。



ONWCHAR

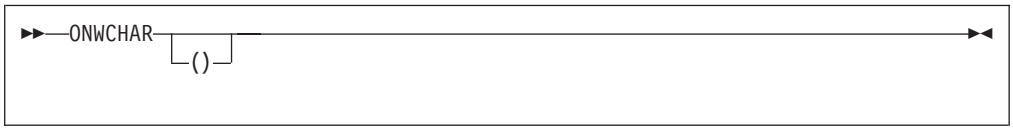
ONWCHAR は、CONVERSION 条件が発生する原因となるワイド文字を含む widechar(1) ストリングを戻します。この関数は、CONVERSION 条件の ON ユニットの中か、CONVERSION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニット (またはそれらの ON ユニットから動的に派生したもの) の中にあります。



ONWCHAR 組み込み関数がコンテキスト以外で使われる場合は、ワイド文字ブランクが戻されます。

ONWCHAR 疑似変数

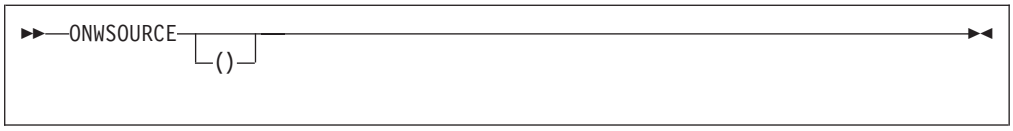
ONWCHAR 疑似変数は、ONWCHAR 組み込み関数の現行値を設定します。疑似変数に割り当てられたエレメントの値は、長さ 1 のワイド文字値に変換されます。再変換されるときには、新しいワイド文字が使われます (77 ページの『第 5 章 データ変換』の変換を参照してください)。



疑似変数は、コンテキスト以外では使用してはいけません。

ONWSOURCE

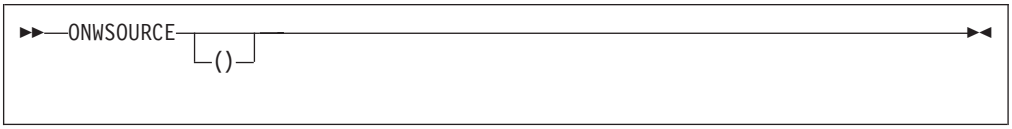
ONWSOURCE は、値が CONVERSION 条件が発生したときに処理中のフィールドの内容であるワイド文字ストリングを戻します。また、この関数は、CONVERSION 条件か、CONVERSION 条件の暗黙処置として発生した ERROR 条件または FINISH 条件の ON ユニット (またはそれらから動的に派生したもの) の中にあります。



コンテキスト以外で ONWSOURCE が使われている場合は、ヌル・ストリングが戻されます。

ONWSOURCE 疑似変数

ONWSOURCE 疑似変数は、ONWSOURCE 組み込み関数の現行値を設定します。疑似変数に割り当てられたエレメントの値は、ワイド文字ストリングに変換されます。また、必要な場合には、CONVERSION 条件が発生したフィールドの長さと一致するように、右側がワイド文字ブランクで埋められるか切り捨てられます。再変換されるときには、このストリングが使われます。

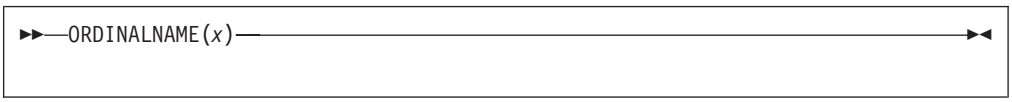


変換が再試行されるときは、疑似変数に割り当てられたストリングは、単一データ項目として処理されます。このため、エラーの訂正処理は、GET LIST ステートメントまたは GET DATA ステートメントの実行時に変換が発生するときに、複数のデータ項目を含むストリングを割り当てることはできません。ストリングにブランクまたはコンマがあると、CONVERSION が再発生します。

疑似変数は、コンテキスト以外では使用してはいけません。

ORDINALNAME

ORDINALNAME は、序数 *x* に関連したセットのメンバーである不変文字ストリングを戻します。

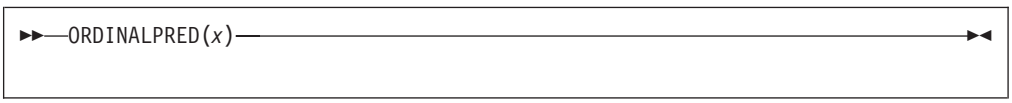


x 参照。この関数には、序数タイプを保持している必要があります。

ORDINAL を計算式内で使用してはなりません。また、文字へ変換してもなりません。しかし、ORDINALNAME によって、ORDINAL に表示可能な値を取得することが可能なので、デバッグにおいて非常に役立ちます。

ORDINALPRED

ORDINALPRED は、序数 *x* が取ることができる 2 番目に低い値である序数を戻します。

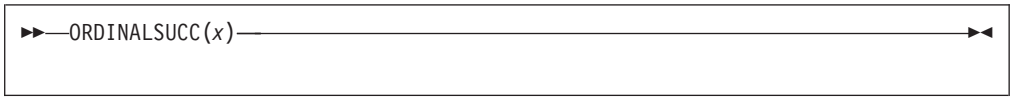


x 参照。この関数には、序数タイプを保持している必要があります。

戻される序数には、序数 *x* と同じタイプを保持している必要があります。

ORDINALSUCC

ORDINALSUCC は、序数 *x* が取ることができる 2 番目に高い値である序数を戻します。

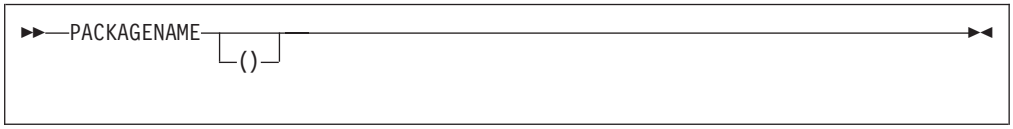


x 参照。この関数には、序数タイプを保持している必要があります。

戻される序数には、序数 *x* と同じタイプを保持している必要があります。

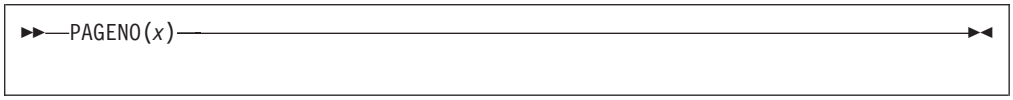
PACKAGENAME

PACKAGENAME は、呼び出されたパッケージ名を含む不変文字ストリングを戻します。現行のコンパイル単位にパッケージがない場合は、PACKAGENAME は最外部のプロシージャーの名前を戻します。



PAGENO

PAGENO は、ファイル *x* に関連した現行ページ番号であるスケールのない FIXED BINARY31 値を戻します。



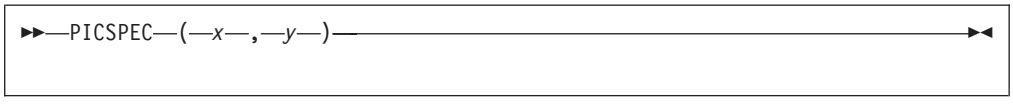
x ファイル参照。ファイルは、オープン状態で、PRINT 属性を持っている必要があります。

ファイルが PRINT ファイルでない場合は、ERROR 条件が発生します。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

PICSPEC

PICSPEC 組み込み関数は、CHARACTER から PICTURE タイプへデータをキャストします。



x 式。

y ピクチャー指定。

式 *x* は、コンパイル時に既知となる長さの CHARACTER NONVARYING でなければなりません。

y は、最初の引数と同じ長さの外部表記を持つ、有効な PICTURE を指定する文字リテラルでなければなりません。

結果は、2 番目の引数によって指定された PICTURE タイプになります。

EDIT 組み込み関数とは異なり、変換は行われず、最初の引数がピクチャーに有効なデータを保持しているかを調べるチェックも行われません。

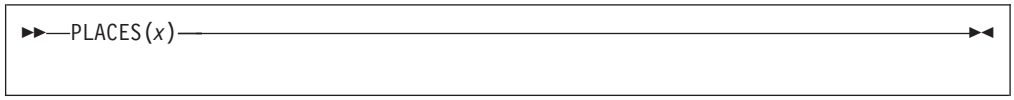
UNSPEC 組み込み関数のように、データの「タイプ」だけが変更されます。

そのため、例えば PICSPEC(x,'(5)9') では、*x* は CHAR(5) でなければなりませんが (ピクチャー指定 '(5)9' の長さは 4 文字であるものの、外部表記では 5 つの文字が必要なため)、*x* は実際に 5 つの数字を含んでいるかどうかチェックされることはありません。

N = N + PICSPEC(X,'(5)9') という記述によって *x* が CHAR から PIC'(5)9' に変換されることはありませんが (この変換ではライブラリー呼び出しが必要)、*x* の内容は、PIC'(5)9' であると宣言されたかのように扱われます。

PLACES

PLACES は、浮動小数点の式 *x* を表すのに使われるモデル精度である FIXED BINARY(31,0) 値を戻します。



x 式。 *x* は、REAL FLOAT として宣言しなければなりません。

PLACES(x) は定数で、制限付き式に使用できます。

例 (Intel の値)

```
places(x) = 24        for x float bin(p), p <= 21
places(x) = 53        for x float bin(p), 21 < p <= 53
places(x) = 64        for x float bin(p), 53 < p

places(x) = 24        for x float dec(p), p <= 6
places(x) = 53        for x float dec(p), 6 < p <= 16
places(x) = 64        for x float dec(p), 16 < p
```

例 (AIX の値)

```
places(x) = 024        for x float bin(p), p <= 21
places(x) = 053        for x float bin(p), 21 < p <= 53
places(x) = 106        for x float bin(p), 53 < p

places(x) = 024        for x float dec(p), p <= 6
places(x) = 053        for x float dec(p), 6 < p <= 16
places(x) = 106        for x float dec(p), 16 < p
```

例 (z/OS の 16 進値)

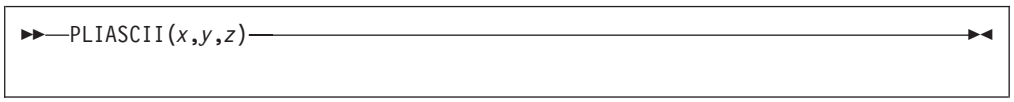
```
places(x) = 6         for x float bin(p), p <= 21
places(x) = 14        for x float bin(p), 21 < p <= 53
places(x) = 28        for x float bin(p), 53 < p

places(x) = 6         for x float dec(p), p <= 6
places(x) = 14        for x float dec(p), 6 < p <= 16
places(x) = 28        for x float dec(p), 16 < p
```

例 (z/OS の IEEE 値)

```
places(x) = 24        for x float bin(p), p <= 21
places(x) = 53        for x float bin(p), 21 < p <= 53
places(x) = 113       for x float bin(p), 53 < p

places(x) = 24        for x float dec(p), p <= 6
places(x) = 53        for x float dec(p), 6 < p <= 16
places(x) = 113       for x float dec(p), 16 < p
```

PLIASCII

PLIASCII は、位置 y にある Z ストレージの単位 (バイト) を位置 x で EBCDIC から ASCII に変換します。位置 x および y にあるストレージは、同じ位置を指定しない限りオーバーラップしません。

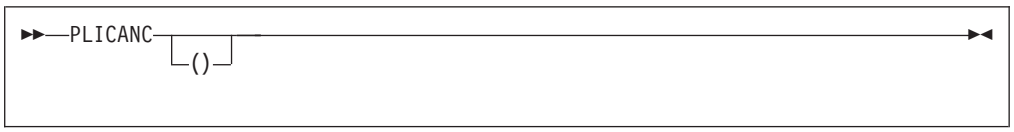


x および y
タイプ POINTER または OFFSET を持つ式。タイプが OFFSET の場合は、式は、AREA 属性によって宣言される OFFSET 変数でなければなりません。

z FIXED BIN(31,0) に変換される計算タイプによる式。

PLICANC

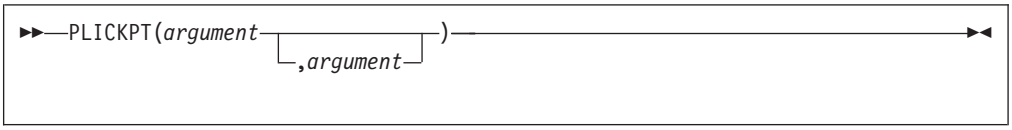
この組み込みサブルーチンを使って、自動再始動機能を取り消すことができます。



PLICANC の使用に関する情報については、「プログラミング・ガイド」を参照してください。

PLICKPT

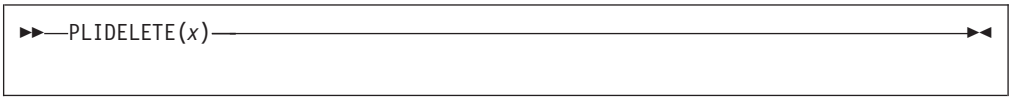
この組み込みサブルーチンを使って、実行時再始動用チェックポイントを使用することができます。



PLICKPT の使用に関する情報については、「プログラミング・ガイド」を参照してください。

PLIDELETE

この組み込みサブルーチンは、ハンドル *x* に関連したストレージを解放します。



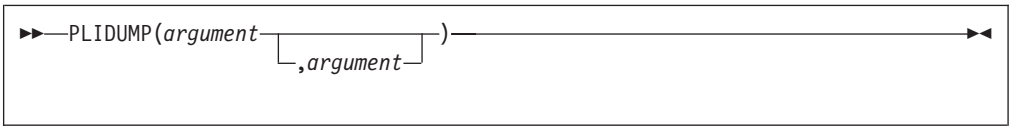
x ハンドル式。

PLIDELETE(x) は、ハンドルに関連したストレージを解放するには最適な方法です。このストレージは、通常、NEW タイプ関数によって取得されます。

CALL PLIDELETE(x) は、CALL PLIFREE(PTRVALUE(x)) と同等です。

PLIDUMP

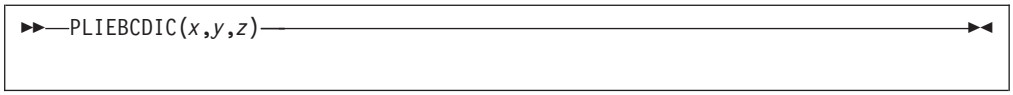
このサブルーチンにより、プログラムで使われるストレージの選択部分の定様式ダンプを獲得することができます。



PLIDUMP の使用に関する情報については、「*プログラミング・ガイド*」を参照してください。

PLIEBCDIC

PLIEBCDIC は、位置 y にある Z ストレージの単位 (バイト) を位置 x で ASCII から EBCDIC に変換します。位置 x および y にあるストレージは、同じ位置を指定しない限りオーバーラップしません。

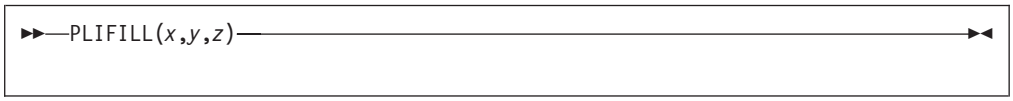


x および y
タイプ POINTER または OFFSET を持つ式。タイプが OFFSET の場合は、式は、AREA 属性によって宣言される OFFSET 変数でなければなりません。

z FIXED BIN(31,0) に変換される計算タイプによる式。

PLIFILL

この組み込みサブルーチンは、バイト *y* の *z* コピーを変換、埋め込み、または切り捨てを行わずに位置 *x* に移動します。



x 式。 *x* は、POINTER または OFFSET として宣言されなければなりません。 *x* が OFFSET の場合は、x を AREA 属性で宣言する必要があります。

y CHARACTER(1) NONVARYING として宣言されなければなりません。

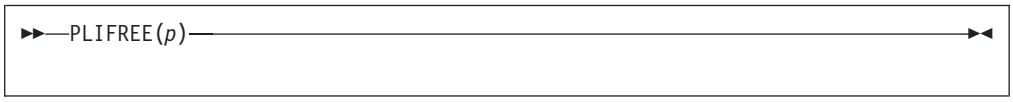
z FIXED BINARY(31,0) に変換される式。

例

```
dcl 1 Str1,
      2 B  fixed bin(31),
      2 C  pointer,
      2 * union,
        3 D  char(4),
        3 E  fixed bin(31),
        3 *,
          4 * char(3),
          4 F fixed bin(8) unsigned,
      2 * char(0)
           initial call plifill( addr(Str1), '00'x, stg(Str1) );
```

PLIFREE

この組み込みサブルーチンは、ALLOCATE 組み込み関数を使用して割り当てられたポインター *p* に関連したヒープ・ストレージを解放します。

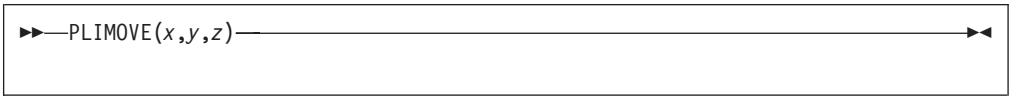


p ロケーター式。

PLIFREE は、ALLOCATE (ALLOC) の反対です。

PLIMOVE

この組み込みサブルーチンは、位置 *y* の *z* ストレージの単位 (バイト) を変換、埋め込み、または切り捨てを行なわずに位置 *x* に移動します。 PLIOVER 組み込みサブルーチンと異なり、位置 *x* および *y* でのストレージは固有であると仮定されます。ストレージがオーバーラップする場合は、予期しない結果が発生することがあります。



x および y
POINTER または OFFSET として宣言された式。タイプが OFFSET の場合は、*x* または *y* を AREA 属性によって宣言する必要があります。

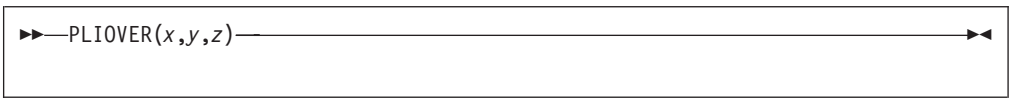
z 式。 *z* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

例

```
dcl 1 Str1,
      2 B  fixed bin(31),
      2 C  pointer,
      2 * union,
        3 D  char(4),
        3 E  fixed bin(31),
        3 *,
          4 * char(3),
          4 F fixed bin(8) unsigned,
          2 * char(0);
dcl 1 Template nonasgn static,
      2 * fixed bin(31) init(200),
      2 * pointer init(null()),
      2 * char(4) init(''),
      2 * char(0);

call plimove(addr(Str1), addr(Template), stg(Str1));
```

PLIOVER

この組み込みサブルーチンは、位置 *y* の *z* ストレージの単位 (バイト) を変換、埋め込み、または切り捨てを行なわずに位置 *x* に移動します。 PLIMOVE 組み込みサブルーチンと異なり、位置 *x* および *y* でのストレージはオーバーラップ可能です。



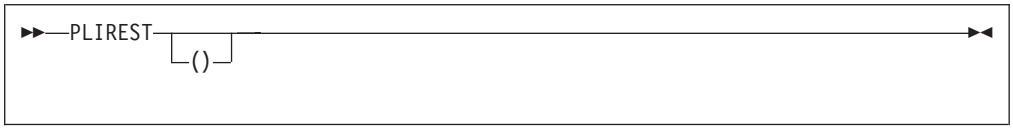
x および y
POINTER または OFFSET として宣言された式。タイプが OFFSET の場合は、*x* または *y* を AREA 属性によって宣言する必要があります。

z 式。 *z* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

PLIOVER の使用法は、*x* および *y* のストレージをオーバーラップできることを除き、PLIMOVE と同じです (487 ページの『PLIMOVE』を参照)。

PLIREST

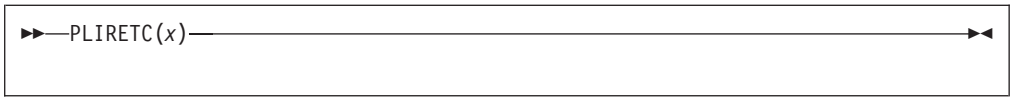
この組み込みサブルーチンを使って、プログラム実行を再開することができます。



PLIREST の使用に関する情報については、「プログラミング・ガイド」を参照してください。

PLIRETC

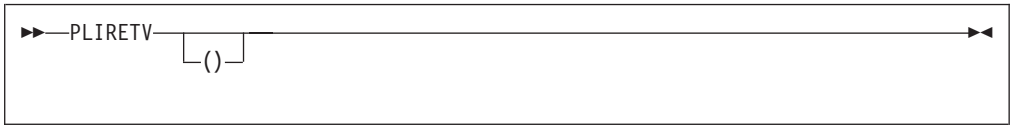
この組み込みサブルーチンにより、呼び出されたこの PL/I プログラムまたは PLIRETV 組み込み関数を介した別の PL/I プロシージャーで調べることができる、戻りコードを常に設定できます。



x FIXED BINARY(31,0) 戻りコードを指定する式。

PLIRETV

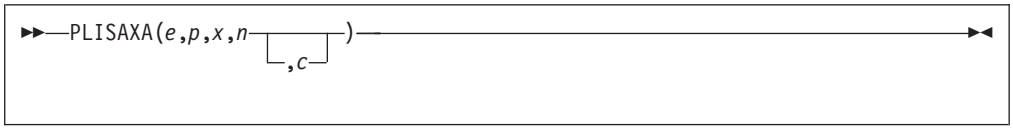
PLIRETV は、PL/I の戻りコードである FIXED BINARY(31,0) 値を戻します。



PL/I の戻りコードの値は、CALL PLIRETC ステートメントで指定された最新の値です。

PLISAXA

この組み込みサブルーチンを使用すると、プログラムのバッファー内にある XML 文書に対し、SAX フォーマットの構文解析を実行できます。



e イベント構造体

p 構文解析イベントに戻されるポインター値または「トークン」

x 入力 XML が入っているバッファーのアドレス

n そのバッファーにあるデータのバイト数

c その XML のコード・ページの名称を指定する数値表現

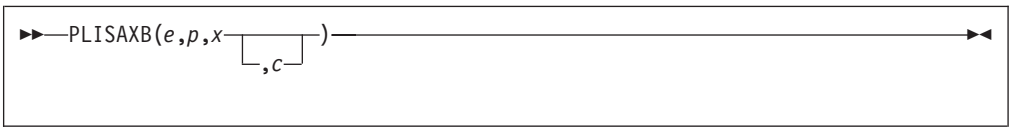
XML が CHARACTER VARYING ストリングまたは WIDECHAR VARYING ストリングに含まれている場合は、ADDRDATA 組み込み関数を使用して、最初のデータ・バイトのアドレスを取得する必要があります。

また、XML が WIDECHAR ストリングに含まれている場合、バイト数の値は LENGTH 組み込み関数によって戻される値の 2 倍になることに注意してください。

詳細については、「*プログラミング・ガイド*」を参照してください。

PLISAXB

この組み込みサブルーチンを使用すると、ファイル内にある XML 文書に対し、SAX フォーマットの構文解析を実行できます。



e イベント構造体

p 構文解析イベントに戻されるポインター値または「トークン」

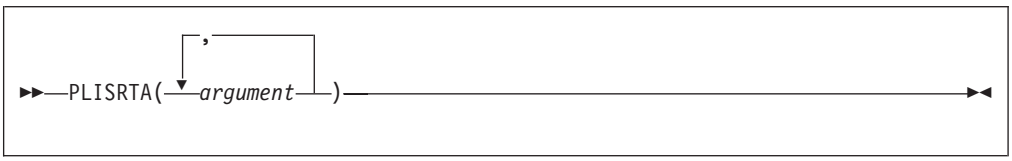
x 入力ファイルを指定する文字ストリング式

c その XML のコード・ページの名称を指定する数値表現

詳細については、「*プログラミング・ガイド*」を参照してください。

PLISRTA

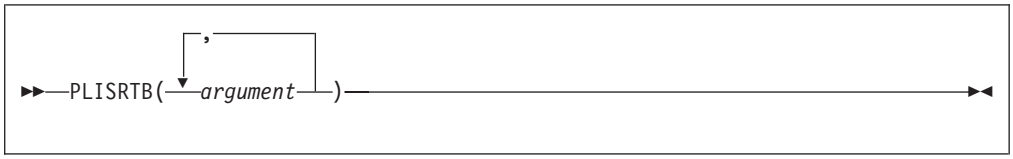
この組み込みサブルーチンにより、入力ファイルをソートして、ソートされた出力ファイルを作ることができます。



詳細については、「*プログラミング・ガイド*」を参照してください。

PLISRTB

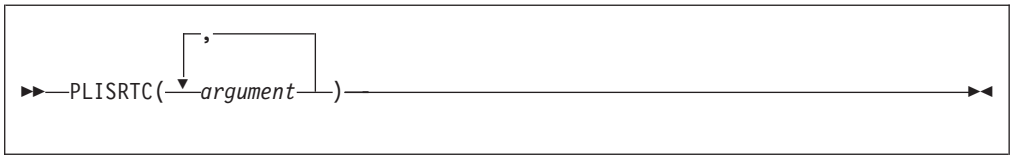
この組み込みサブルーチンにより、E15 PL/I 出口プロシージャーによって提供される入力レコードをソートし、ソートされた出力ファイルを作ることができます。



詳細については、「*プログラミング・ガイド*」を参照してください。

PLISRTC

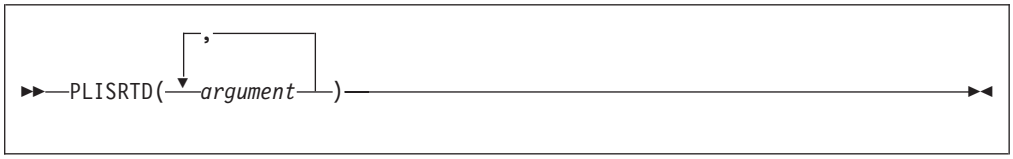
この組み込みサブルーチンにより、入力ファイルをソートして、E35 PL/I 出口プロシージャーによって処理されるソートされたレコードを作ることができます。



詳細については、「*プログラミング・ガイド*」を参照してください。

PLISRTD

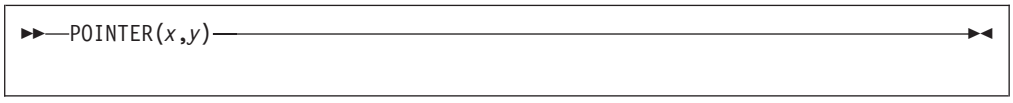
この組み込みサブルーチンにより、E15 PL/I 出口プロシージャーによって提供される入力レコードをソートして、E35 PL/I 出口プロシージャーによって処理されるソートされたレコードを作ることができます。



詳細については、「*プログラミング・ガイド*」を参照してください。

POINTER

POINTER は、*y* によって指定された区域で、オフセット参照 *x* によって指定される生成を識別するポインター値を戻します。*x* がヌル・オフセット値の場合は、ヌル・ポインター値が戻されます。



省略形: PTR

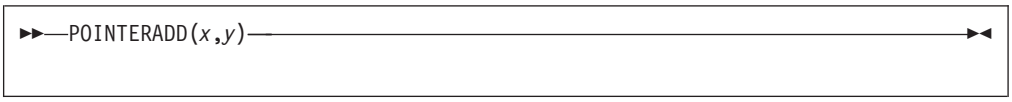
x オフセット参照。 *x* は、ヌル・オフセット値です。そうでない場合は、*x* は、基底付き変数の生成を識別する必要があります。ただし、*y* にある必要はありません。*y* でそうでない場合は、生成は *y* での生成と同等でなければなりません。

y 区域参照。

最後の生成の割り当てまで、それぞれ同じ回数だけ変数が割り当てられ、解放された場合は、異なる区域での基底付き変数の生成は同等です。

POINTERADD

POINTERADD は、その引数の和であるポインター値を戻します。



省略形: PTRADD

x ポインターの式。

y 計算タイプを保持する必要がある式。この式は、FIXED BINARY(31,0) に変換されます。

POINTERADD は、基底付き変数のロケーターとして使用できます。

POINTERADD は、減算されるオペランドの接頭部に負符号 (-) を付けることにより、減算に使用できます。

ポインターを増分するために、POINTERADD を使用する必要はありません。整数の場合と同じようにポインターを増分させるだけです。たとえば、以下のように書き込む必要はありません。

```
p = pointeradd(p,2);
```

その代わりに、以下の同等ステートメントのどちらか 1 つを書き込むことができます。

```
p = p + 2;
p += 2;
```

しかし、POINTERADD は、以下の例にあるように、ポインターのロケーション・オフセットでのストレージ参照を取り消す際に便利です。

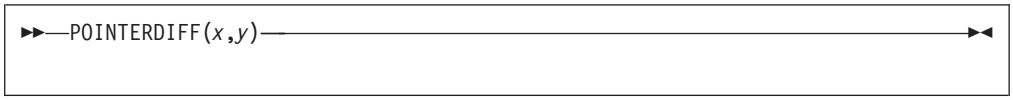
```
dcl x fixed bin(31), b based fixed bin(31);
x = pointeradd(p,2)->b;
```

PL/I のロケーターは参照でなければならないため、書き込みができないことにご注意ください。

```
x = (p + 2)->b;
```

POINTERDIFF

POINTERDIFF は、2 つのポインター *x* および *y* 間の差である FIXED BINARY(31,0) 結果を戻します。

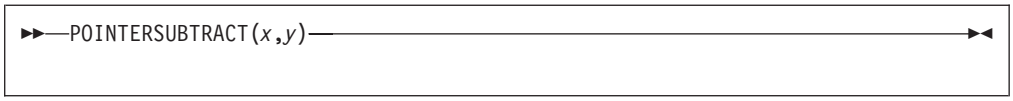


省略形: PTRDIFF

x および y
POINTER として宣言された式。

POINTERSUBTRACT

POINTERSUBTRACT は、POINTERADD(x,-y) と同等です。



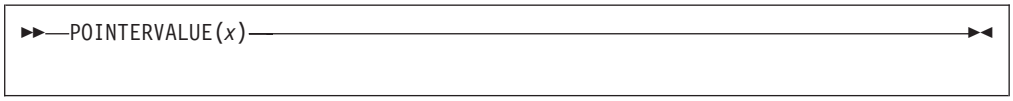
省略形: PTRSUBTRACT

x ポインター式でなければなりません。

y 計算タイプを保持する必要がある式。この式は、FIXED BINARY(31,0) に変換されます。

POINTERVALUE

POINTERVALUE は、変換された *x* の値であるポインター値を戻します。



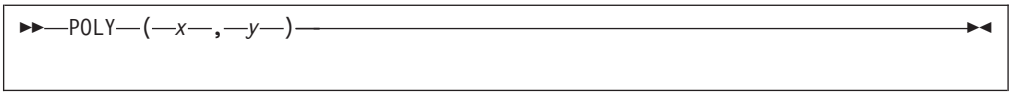
省略形: PTRVALUE

x HANDLE 属性か計算タイプのどちらかを保持する必要がある式。 *x* に計算タイプがある場合は、FIXED BINARY(31,0) に変換されます。

POINTERVALUE(x) は、*x* が定数の場合に、静的ポインター変数を初期化するのに使われます。

POLY

POLY は、1 次元配列式 x から作られた 1 つの多項式の近似値を表す浮動小数点数値を返します。戻り値は、最初の引数と同じ属性を持ちます。 POLY の構文は次のとおりです。



x 配列式。

y 要素式。

x は REAL FLOAT でなくてはなりません。また、必要に応じて **y** は x の属性に変換されます。

x の下限が 0、上限が n である場合の結果は、係数が x によって与えられ、 y の次元を n とする古典多項式になります。つまり、次のような結果になります。

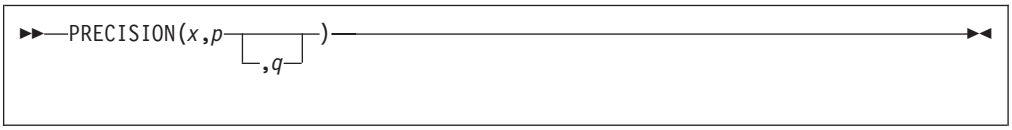
```
x(0) + x(1)*y + x(2)*y**2 + ... + x(n)*y**n
```

x の下限が m、上限が n である一般的なケースでは、結果は次の多項式になります。

```
x(m) + x(m+1)*y + x(m+2)*y**2 + ... + x(n)*y**(n-m)
```

PRECISION

PRECISION は、*p* および *q* で指定される精度で、*x* の値を戻します。戻り値の基数、モードおよびスケールは、*x* の場合と同じです。



省略形: PREC

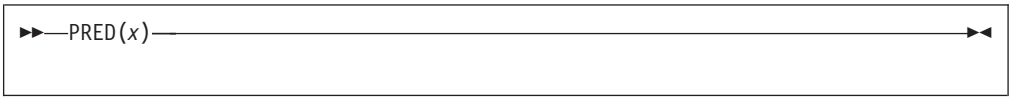
x 式。

p 制限付き式。 *p* は、式 *x* が変換後に持つ式の値である桁数を指定します。

q 制限付き式。結果のスケール因数を指定します。固定小数点の場合に、*q* が省略されると、スケール因数ゼロが想定されます。結果が浮動小数点の場合は、*q* は省略します。

PRED

PRED は、*x* より小さい表示可能な最大数である浮動小数点の値を戻します。戻り値は、*x* の基数、モード、および精度になります。OVERFLOW は、そのような数がない場合に発生します。

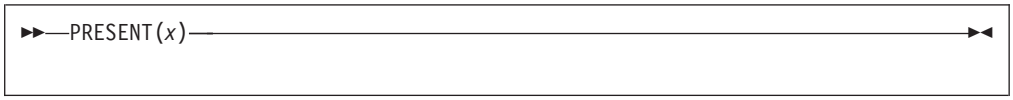


x REAL FLOAT 式。

PRED(TINY(X)) は、ゼロを戻し、UNDERFLOW を発生させることはありません。

PRESENT

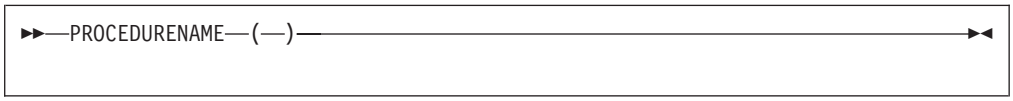
PRESENT(x) が含まれているプロシージャーの呼び出しで、*x* という名前のパラメーターが存在する場合は、present(x) は '1'B である BIT(1) 値を戻します。



x レベル 1 の添え字が付いていない BYADDR パラメーター。

PROCEDURENAME

PROCEDURENAME() は、この組み込み関数を呼び出すプロシージャー名を含む不変文字ストリングを戻します。

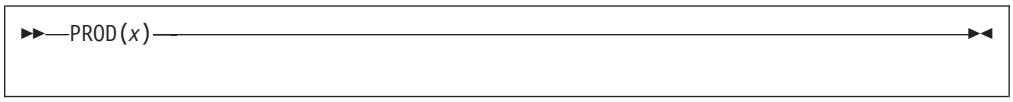


省略形: PROCNAME

PROCEDURENAME は、CALL ステートメントまたは GOTO ステートメントに名前が表示されていても複数のラベル指定の左端の名前を常に戻します。

PROD

PROD は、*x* のすべてのエレメントの積を戻します。



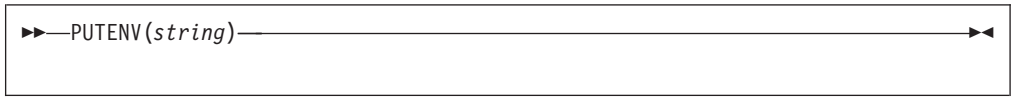
x 配列式。 *x* のエレメントがストリングの場合は、それらのエレメントは固定小数点の整数値に変換されます。

x のエレメントが固定小数点の整数値またはストリングでない場合は、浮動小数点に変換され、結果は浮動小数点になります。

結果は、*x* の精度になります。ただし、固定小数点の整数値およびストリングの結果は精度 (n,0) の固定小数点になります。ここで、*n* は許可される最大桁数です。基数およびモードは、変換後の引数 *x* と一致します。

PUTENV

PUTENV は、C の putenv 関数と同じように機能します。この関数は、新しい環境変数を追加するか、または既存の環境変数の値を変更します。

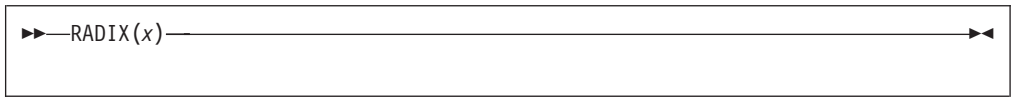


string
フォーマット *envvarname=value* の文字ストリング。

成功した場合には、PUTENV は真 ('1'B) を戻し、そうでない場合には、偽 ('0'B) を戻します。

RADIX

RADIX は、浮動小数点の式 *x* を表すのに使われるモデルの基数である FIXED BINARY(31,0) 値を戻します。

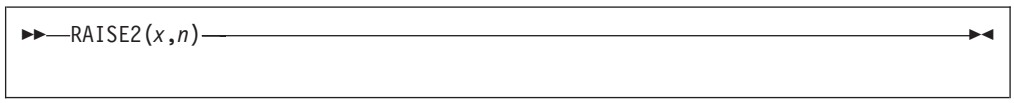


x REAL FLOAT 式。

RADIX(x) は 2 (例外として、z/OS 上の 16 進数の場合は 16) で、制限付き式に使われます。

RAISE2

RAISE2(x,n) は、値 x*(2**n) を戻します。



注: RAISE2(x,n) は、アセンブラー SLA(x,n) と同様です。

x 式。 *n* は、計算タイプでなくてはなりません。

n 式。 *n* は、計算タイプでなくてはなりません。

x は SIGNED REAL FIXED BIN(M,0) に変換されて、その結果は同じ属性を持ちます。

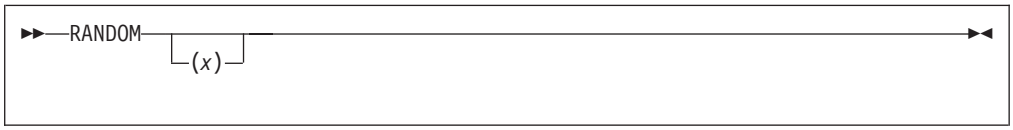
n が負の場合、または *n* が M より大きい場合は、この関数は不定です。

例

```
raise2(6,1)                    /*  produces 12  */
```

RANDOM

RANDOM は、与えられたシード *x* を使用して生成されたランダム数 FLOAT BINARY(53) を戻します。 *x* を省略すると、ランダム数は、シードを与えた最後の RANDOM 呼び出しのシードに基づくか、シードを与えた RANDOM がこれまでに呼び出されていない場合は、デフォルトの初期シード 1 に基づいて生成されます。



x 式。 *x* に、計算タイプ (必須) および算術タイプを保持している必要があります。 *x* は数値で、実数でなければなりません。 *x* が指定された FIXED BINARY(31,0) でない場合は、*x* は変換されます。

0 < *x* < 2,147,483,646 でなければ ERROR 条件が発生します。

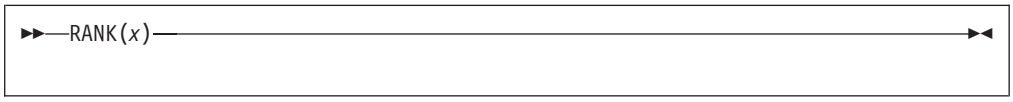
RANDOM によって生成される値は、0 < random(x) < 1 で 0 と 1 の間に均等に分散します。値は、下記のとおり乗算合同式法を使用して生成されます。

```
seed(x) = mod(950706376 * seed(x - 1), 2147483647)
random(x) = seed(x) / 2147483647
```

シードは、マルチスレッド化・アプリケーションの各スレッド内ではなく、プログラム・レベルで維持されます。

RANK

RANK は、文字またはワイド文字に対応する整数値を戻します。



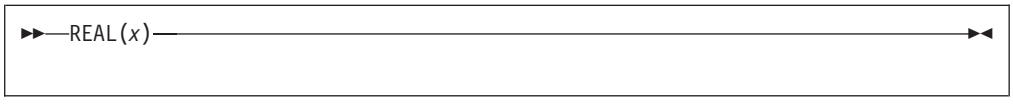
x 属性 CHAR (1) NONVARYING または WCHAR (1) NONVARYING を保持しなければなりません。

x が文字である場合、RANK(x) は index(collate(),x)-1 として定義され、Rank は BYTE の逆になります。

x がワイド文字である場合、ビッグ・エンディアン・フォーマットで格納された x を y とすると、RANK(x) は UNSPEC(y) に等しくなります。

REAL

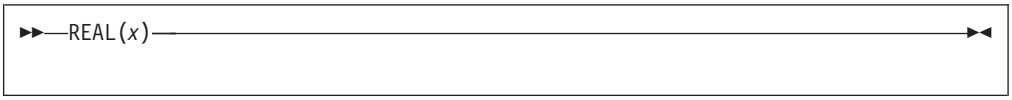
REAL は、*x* の実数部分を戻します。結果には、*x* の基数、スケール、および精度が含まれます。



x 式。 *x* が実数の場合は、*x* は複素数に変換されます。

REAL 疑似変数

REAL 疑似変数は、実数値または複合値の実数部分を *x* の実数部分に割り当てます。

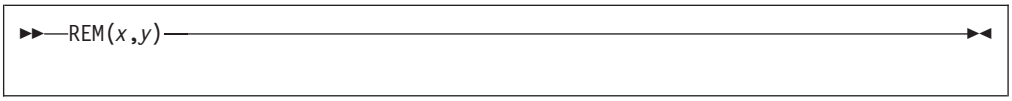


x 複素数参照。

REM

REM は、*x* を *y* で除算した剰余を戻します。これは、次の式で計算できます。

```
x - y * trunc(x/y)
```

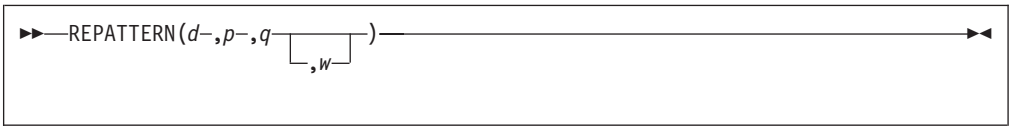


x および y
式。 *x* および *y* は、計算タイプでなければなりません。また、算術タイプでも構いません。

REM 組み込み関数と MOD 組み込み関数の対比の例については、 470 ページの『MOD』を参照してください。

REPATTERN

1 つのパターンでは日付を保持している値をとり、2 番目のパターンでは日付に変換された値を戻します。



d 日付を表すストリング式。 *d* の長さは、少なくともソース・パターン *q* の長さと同じでなければなりません。 *d* のほうが長い場合は、余分の文字を先行ブランクにしなければなりません。

d に、計算タイプ (必須) と文字タイプを保持している必要があります。そうでないと、文字に変換されます。

p ターゲット・パターン。サポートされている日付/時刻パターンの 1 つでなければなりません。

q ソース・パターン。サポートされている日付/時刻パターンの 1 つでなければなりません。

w 整数に変換できる式 (1950 など) を指定します。負の場合、コードを実行する

ときに年の値から減算されるようにオフセットを指定します。省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値にデフォルトで設定されます。

戻り値は、属性 CHAR(m) NONVARYING を持っています。 *m* は、ターゲット・パターン *p* の長さです。

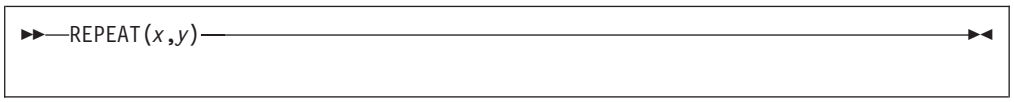
許可されるパターンを 406 ページの表 52 にリストしてあります。リリアン形式の説明については、 404 ページの『日付/時刻組み込み関数』を参照してください。

以下に、REPATTERN を使用して、年を 2 桁で表す日付パターンと 4 桁で表す日付パターンの間で変換を行う方法について、いくつかの例を示します。ただし、両方のパターンが年の値を保持するのに同じ桁数を使用している場合でも、この組み込み関数を使用して、日付をあるサポートされたパターンから別のサポートされたパターンへ変換することができます。

REPATTERN('990101','YYYYMMDD','YYMMDD', 1950) returns '19990101'
REPATTERN('000101','YYYYMMDD','YYMMDD', 1950) returns '20000101'
REPATTERN('19990101','YYMMDD','YYYYMMDD', 1950) returns '990101'
REPATTERN('20000101','YYMMDD','YYYYMMDD', 1950) returns '000101'
REPATTERN('19490101','YYMMDD','YYYYMMDD', 1950) raises ERROR

REPEAT

REPEAT は、*y* で指定された数だけ連結した *x* で構成されるビット・ストリング、文字ストリング、漢字ストリング、またはワイド文字ストリングを戻します。したがって、*x* の (*y* + 1) オカレンスがあります。



x 繰り返されるビット、文字、漢字、またはワイド文字の式。 *x* が算術の場合は、次の変換が発生します。
- 2 進数の場合は、*x* はビット・ストリングに変換されます。
- 10 進数の場合は、*x* は文字ストリングに変換されます。

y 式。必要な場合には、*y* は、実数の固定小数点の 2 進数の値に変換されます。

y がゼロまたは負の場合は、ストリング *x* が戻されます。 REPEAT 組み込み関数の例については、 430 ページの『COPY』を参照してください。

REPLACEBY2

REPLACEBY2 は、*x* の中の一部の文字を文字のペアで置き換えることによって形成される、固定長ストリングを戻します。

x 置換できる文字があるかどうかを検索される文字式。

y 置換するペアの値を含む文字式。

z 置換される文字を含む文字式。

REPLACEBY2 は、*x* の各文字を次のように処理します。

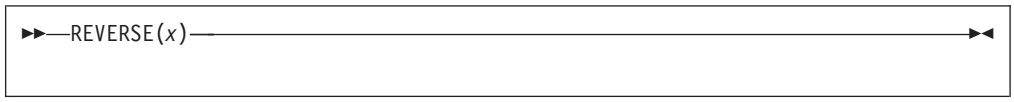
x の中にある文字が *z* の中で検出されると、*y* にある *z* に対応する文字ペアが結果にコピーされます。それ以外の場合は、 *x* の中の文字が結果にコピーされます。 *z* に複数の同一文字が入っている場合は、左端のオカレンスが使われます。

ストリング *y* は、ストリング *z* の 2 倍の長さでなければなりません。

例として、REPLACEBY2('Rätsel', 'aeoeuess', 'äöüß') はストリング 'Raetsel' を戻します。

REVERSE

REVERSE は、順序を反対にした *x* のエレメントを含む不変ストリングを戻します。



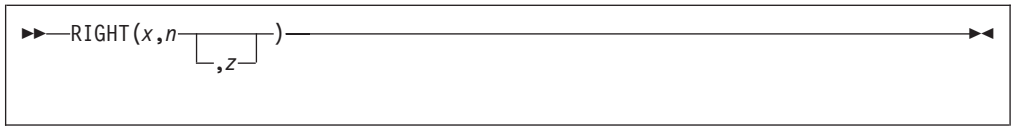
x 式。 *x* に、計算タイプ (必須) とストリング・タイプを保持している必要があります。 *x* にストリング・タイプがない場合は、連結の規則に従って、ストリング (すなわち、数値からビット、文字、漢字、またはワイド文字) に変換されます。

例

```
dcl Source char value('HARPO');
dcl Target char(length(Source));

Target = reverse (Source);     /* 'OPRAH' */
```

RIGHT

RIGHT は、長さ *n* のストリング *x* の右端にストリング x を挿入し、必要な場合に左側が文字 *z* を埋めた結果のストリングを戻します。 *z* が省略される場合は、埋め込み文字にブランクが使われます。



x 式。 *x* に、計算タイプ (必須) を保持している必要があります。また、文字タイプを保持することもできます。そうでないと、文字に変換されます。

n 式。 *n* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

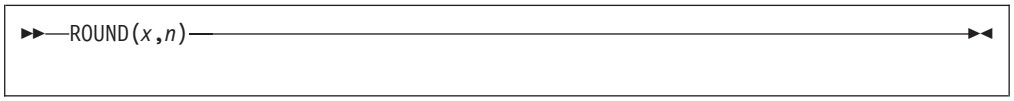
z 式。 *z* が指定された場合、*z* は CHARACTER(1) NONVARYING タイプでなければなりません。

例

```
dcl Source char value('One Hundred SCIDS Marks');
dcl Target char(30);

Target = right (Source, length(Target), '*');
           /* '*******One Hundred SCIDS Marks'               */
```

ROUND

ROUND は、*n* で指定された桁で丸められた *x* の値を戻します。結果は、*x* のモード、基数、およびスケールになります。



x 実数式。 *x* が負の場合は、絶対値が丸められてから、符号が戻ります。

n オプションの符号付き整数。丸める桁を指定します。 *n* は、FIXED データのスケール因数の制限に従う必要があります。 *n* が 0 より大きい場合は、小数点の右から (*n*) 番目の桁が丸められます。 *n* がゼロまたは負の場合は、小数点の左から (1-*n*) 番目の桁が丸められます。

固定小数点の結果の精度は、次の式で与えられます。 (*p,q*) は *x* の精度、N は許可される桁数の最大数です。

```
(max(1,min(p-q+1+n,N)),n)
```

n は、結果のスケール因数を指定します。

以下に例を示します。値 6.67 が出力です。

```
dcl X fixed dec(5,4) init(6.6666);
put (round(X,2));
```

コンパイラー・オプション USAGE(ROUND(ANS)) のもとでの結果

x が FIXED の場合。この例では、*x* が BINARY の場合 b = 2、*x* が DECIMAL の場合は b = 10 です。

$$round(x,n) = sign(x)*(b^{-n})* floor(abs(x)* (b^{n}) + 1/2)$$

x が FLOAT で 0 以外の場合。この例では、b = radix(x) 、e = exponent(x) です。

$$round(x,n) = sign(x)*(b^{(e-n)})* floor(abs(x)* (b^{(n-e)}) + 1/2)$$

x が FLOAT で 0 の場合。

```
round (x,n) = 0
```

コンパイラー・オプション USAGE(ROUND(IBM)) のもとでの結果

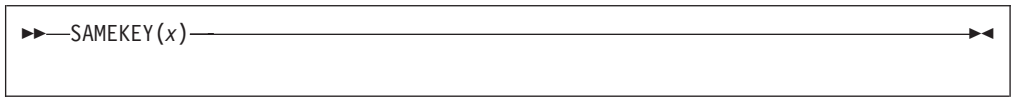
x が FIXED の場合。この例では、*x* が BINARY の場合 b = 2、*x* が DECIMAL の場合は b = 10 です。

$$round(x,n) = sign(x)*(b^{-n})* floor(abs(x)* (b^{n}) + 1/2)$$

x が FLOAT の場合は、処理は行われません。

SAMEKEY

SAMEKEY は、アクセスされたレコードに同じキーを持つ別のレコードが続いているかどうかを示す、長さ 1 のビット・ストリングを戻します。



x ファイル参照。ファイルは、RECORD 属性をとる必要があります。

ファイル *x* での入出力操作が正常終了時、または RECORD 条件が発生する直前では、同じキーを持つ別のレコードが続いている処理レコードの場合は、SAMEKEY によってアクセスされる値が '1'B に設定されます。そうでない場合は、'0'B に設定されます。

また、SAMEKEY でアクセスされる値は、次の場合に '0'B に設定されます。

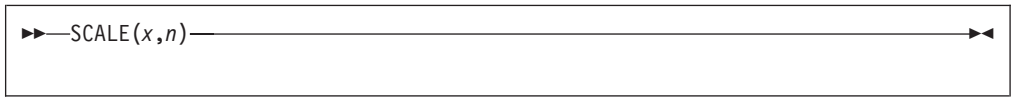
- RECORD 以外の条件が発生する入出力演算が、ファイルの位置が変更または失われる原因になる場合
- ファイルがオープンされていない場合
- 現行カーソル位置がファイル内にない場合

SCALE

SCALE は、次の式に基づいて浮動小数点の値を戻します。

$$x*(\mathrm{radix}(x))^{n}$$

結果は、*x* の基数、モード、および精度になります。

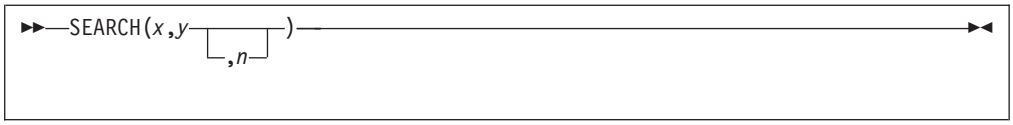


x REAL FLOAT 式。

n 計算タイプを保持する必要がある式。この式は、FIXED BINARY(31,0) に変換されます。

SEARCH

SEARCH は、別のストリングにある任意の 文字、ビット、グラフィック、またはワイド文字が指定のストリングにある場合、そのストリング内の最初の位置を指定するスケールのない REAL FIXED BINARY 値を戻します。この関数に、検索を開始する位置を指定することもできます。



x および y
式。 *x* は、ストリング *y* に表示される文字、ビット、漢字、またはワイド文字を検索するストリングを指定します。

x または *y* はヌル・ストリングで、結果はゼロです。

x に *y* がない場合は、結果はゼロです。

n 式。 *n* は、検索を開始する *x* 内の位置を指定します。 *y* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

1 ≤ *n* ≤ LENGTH(*x*)+1 でない場合は、 STRINGRANGE 条件 (割り込み可能な場合) が発生します。暗黙処置と通常の戻りは、結果ゼロになります。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

SEARCH は、数字のストリングにある区切り文字を検出するのに使われます。

例

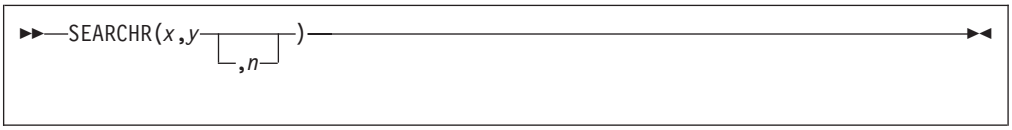
```
  dcl Source char value(' Our PL/I wields the Power ');
  dcl Pos fixed bin(31);

/* Find occurrences of any of the characters 'P','o',or 'w' in source *  /

  Pos = search (Source, 'Pow');              /*  returns 6 for the 'P'    */
  Pos = search (Source, 'Pow', Pos+1);    /*  returns 11 for the 'w'   */
  Pos = search (Source, 'Pow', Pos+1);    /*  returns 22 for the 'P'   */
  Pos = search (Source, 'Pow', Pos+1);    /*  returns 23 for the 'o'   */
  Pos = search (Source, 'Pow', Pos+1);    /*  returns 24 for the 'w'   */

  Pos = index (source, 'Pow',1);             /*  returns 22 for the 'Pow' */
```

上記の例では、SEARCH は 3 文字 (P, o, または w) のどれかが表れる位置を戻します。 INDEX はストリング Pow が表れる位置を戻します。

例

```
  dcl Source char value (' 368,475;121.,856,478')
  dcl Delims char(3) init (',;.');             /* string of delimiters */
  dcl Number(5) char(3);
  dcl Start fixed bin(31);
  dcl End fixed bin(31);

  /* Extract the three-digit numbers from the source string */
  /* by searching for the delimiters                        */
  Start = verify (Source, ' ');
                    /* find start of first number */
  End   = search (Source, ',;.', Start );
                    /* find end of first number */
  if End = 0 then
     End = length (Source) + 1;
  Number(1) = substr (Source, Start, 3);        /* 368 */
  Start = verify (Source, Delims, End);
               /* find start of second number */
  End   = search (Source, Delims, Start );
  Number(2) = substr (Source, Start, 3);        /* 475 */
```

SEARCHR



SEARCHR 関数は、次のこと以外は SEARCH 組み込み関数と同じ演算を行います。

- 検索は、右から左に行われる。
- *n* のデフォルトは LENGTH(*x*)。
- 0 ≤ *n* ≤ LENGTH(*x*) でないと、STRINGRANGE 条件 (割り込み可能な場合) が発生します。暗黙処置と通常の戻りは、結果ゼロになります。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

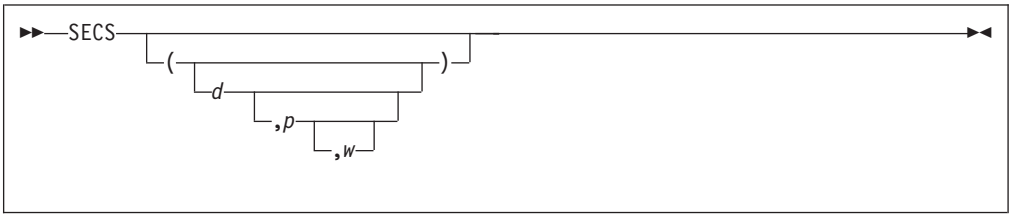
SEARCHR の構文は、501 ページの『SEARCH』で説明します。

例

```
dcl Source char value (' 555 Bailey Ave, San Jose, CA 95141, USA');
dcl Digits char value ('0123456789');
dcl (Start, End) fixed bin(31);
dcl Num char(20) var;

   /*   Find last number (i.e., zip code)    */

End   = searchr (Source, Digits);        /* returns 35 for the '1' */
Start = verifyr (Source, Digits, End);   /* returns 30 for the ' ' */
Num   = substr (Source, Start + 1, End - Start);  /* extract number */
```

SECS

SECS は、日付 *d* に対応する秒数 (リリアン形式に基づく) を FLOAT BINARY(53) 値で戻します。



d 日付を表すストリング式。存在する場合は、*d* は入力日付をパターン *p* に指定された日付/時刻を表示する文字ストリングとして指定します。*d* が指定されない場合は、「DATETIME()」が想定されます。

d に、計算タイプ (必須) と文字タイプを保持している必要があります。そうでないと、文字に変換されます。

p サポートされている日付/時刻パターンの 1 つ。 *p* が省略される場合は、デフォルトの日付/時刻のパターン「YYYYMMDDHHMISS999」が想定されます。

p に、計算タイプ (必須) と文字タイプを保持している必要があります。そうでないと、文字に変換されます。

w 整数に変換できる式 (1950 など) を指定します。負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値にデフォルトで設定されます。

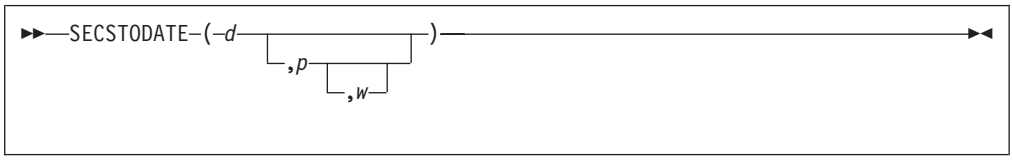
許可されるパターンを 406 ページの表 52 にリストしてあります。リリアン形式の説明については、404 ページの『日付/時刻組み込み関数』を参照してください。

例

```
dcl Dayname (7) char(9) var static nonasgn init( 'Sunday',
                                                 'Monday',
                                                 'Tuesday',
                                                 'Wednesday',
                                                 'Thursday',
                                                 'Friday',
                                                 'Saturday');
dcl Jul4_1776_Secs float bin(53);
dcl Age_Tot_Secs pic 'Z,ZZZ,ZZZ,ZZZ,ZZ9';

Jul4_1776_Secs = secs('17760704','YYYYMMDD');     /* seconds */
Age_Tot_Secs   = secs() - Jul4_1776_Secs;   /* seconds since */
display ('USA became independent on ' ||
          dayname(weekday(secstodays(Jul4_1776_Secs))) ||
         ', July 4, 1776 and at this very moment it has been ' ||
         Age_Tot_Secs, ||     ' seconds.');
```

SECSTODATE

SECSTODATE は、*d* 秒 (リリアン形式に基づく) に対応する *p* で指定された日付/時刻パターンを含む不変文字ストリングを戻します。



d 秒数 (リリアン形式)*d*は、計算タイプでなければなりません。また、必要な場合には、FLOAT BIN(53) に変換されます。

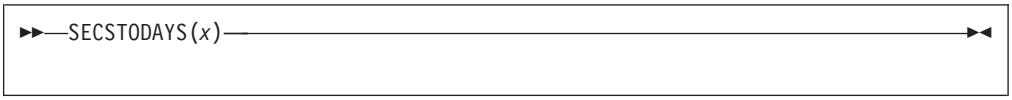
p サポートされている日付/時刻パターンの 1 つ。省略される場合、*p* は、デフォルトの日付/時刻パターン「YYYYMMDDHHMISS999」(DATETIME で戻されるデフォルトのフォーマット) が想定されます。

w 整数に変換できる式 (1950 など) を指定します。負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値にデフォルトで設定されます。

許可されるパターンを 406 ページの表 52 にリストしてあります。リリアン形式の説明については、404 ページの『日付/時刻組み込み関数』を参照してください。

SECSTODAYS

SECSTODAYS は、秒数 *x* を日数に変換した数を表す FIXED BINARY(31,0) 値を戻します。1 日に達しない秒は無視します。



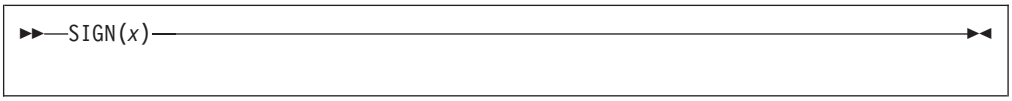
x 式。 *x* は、計算タイプ (必須) を保持し、FLOAT BINARY(53) である必要があります。そうでない場合は、*x* は、FLOAT BINARY(53) に変換されます。

SECSTODAYS(x) は、*x*/(24*60*60) と同じです。

例については、 503 ページの『SECS』を参照してください

SIGN

SIGN は、*x* が正、ゼロ、または負であるかどうかを示すスケールのない REAL FIXED BINARY 値を戻します。



x 実数式。

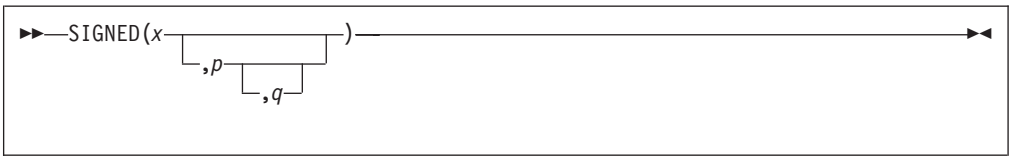
戻り値は、次の式で指定されます。

x の値	戻される値
x > 0	+1
x = 0	0
x < 0	-1

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

SIGNED

SIGNED は、*p* および *q* で指定された精度で、*x* の符号付き FIXED BINARY 値を戻します。



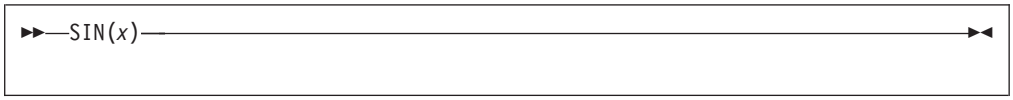
x 式。

p 演算を通して維持する桁数を指定する制限付き式。

q 結果のスケール因数を指定する制限付き式。固定小数点の結果の場合は、*p* が指定され、*q* が省略されると、スケール因数ゼロがデフォルトになります。

SIN

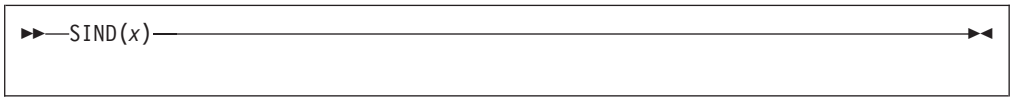
SIN は、*x* の正弦の近似を浮動小数点の値で戻します。戻り値は、*x* の基数、モードおよび精度になります。



x ラジアン値の式。

SIND

SIND は、*x* の正弦の近似を実数の浮動小数点の値で戻します。戻り値は、*x* の基数および精度になります。



x 度で示される値を持つ実数式。

SINF

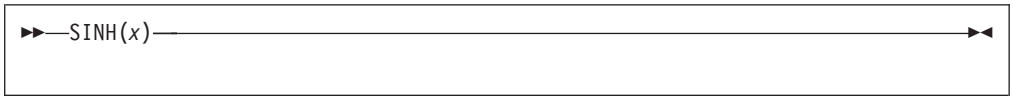
SINF は、次の点を除いては SIN とまったく同じです。

- SINF は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 引数は、実数でなければならない。
- サポートされる引数の最大絶対値は、ハードウェアによって設定される。
- 無効な引数が指定された場合、INVALIDOP 条件が発生したり、その他のハードウェア例外が生成されたりするなど、予期できない結果が生じることがある。
- 結果の正確度は、ハードウェアによって設定される。

定義および構文については、『SIN』を参照してください。

SINH

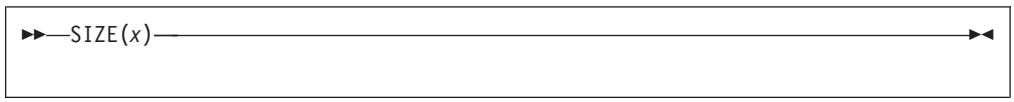
SINH は、*x* の双曲線正弦の近似を表す浮動小数点の値を戻します。戻り値は、*x* の基数、モードおよび精度になります。



x ラジアン値の式。

SIZE

SIZE は、変数 *x* に割り当てる処理系定義ストレージを与える FIXED BINARY(31,0) 値を、バイトで戻します。



x 任意のデータ・タイプ、データ編成、位置合わせ、およびストレージ・クラスを持つ変数。ただし、下記のものは除きます。

x は、次のものであることはできません。

- 位置合わせされていない固定長ビット・ストリングである BASED 変数、DEFINED 変数、パラメーター変数、添え字付き変数、構造変数、または共用体基底付き変数。
- 最初または最後の基本エレメントが位置合わせされていない固定長のビット・ストリングである小構造体または共用体 (ただし、その小構造体を含んでいる大構造体または共用体の最初または最後のエレメントになっている場合を除く)。
- BASED 属性、DEFINED 属性、またはパラメーター属性を持つ大構造体か共用体、またはパラメーターである大構造体か共用体で、位置合わせされていない固定長のビット・ストリングを最初または最後のエレメントとするもの。
- 連結ストレージにない変数。

SIZE(x) によって戻される値は、次の状況で伝送される最大バイト数です。

```
declare F file record input
        environment(scalarvarying);
read file(F) into(x);
```

x が次のものである場合

- 可変長ストリングの場合は、戻り値には、ストリングの長さの接頭辞、ストリングの最大長のバイト数が含まれています。
- 区域の場合は、戻り値には、区域制御バイト、区域の最大サイズが含まれています。
- 区域または可変長ストリングを含む集合の場合は、戻り値には、区域制御バイト、区域の最大サイズ、ストリングの長さ接頭辞、およびストリングの最大長のバイト数が含まれています。

変数が割り振られていない場合、SIZE 組み込み関数を調節可能エクステントとともに、BASED 変数上で使用してはなりません。

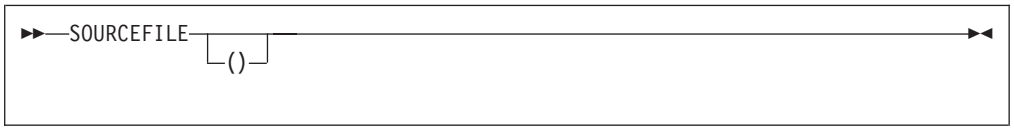
割り振られたバイト数を無視して、変数が現在必要とするバイト数を取得するには、CURRENTSIZE 組み込み関数を使用します。詳細については、433 ページの『CURRENTSIZE』を参照してください。

例

```
dcl Scids   char(17)         init('See you at SCIDS!') static;
dcl Vscids  char(20) varying init('See you at SCIDS!') static;
dcl Stg fixed bin(31);

Stg = storage   (Scids);           /* 17 bytes */
Stg = currentsize (Scids);         /* 17 bytes */
Stg = size (Vscids);               /* 22 bytes */
Stg = currentsize (Vscids);        /* 19 bytes */
Stg = size (Stg);                  /* 4  bytes */
Stg = currentsize (Stg);           /* 4  bytes */
```

SOURCEFILE

SOURCEFILE は、この関数が呼び出されるステートメントを含むファイルの名前をとる不変文字ストリングを戻します。

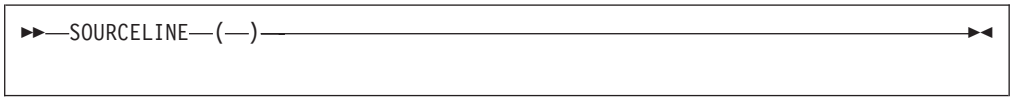


SOURCEFILE は、制限付き式に使用できます。

戻されるストリングは、システムによって異なります。これらのストリングは、トレースおよびデバッグの目的にのみ使用する必要があります。

SOURCELINE

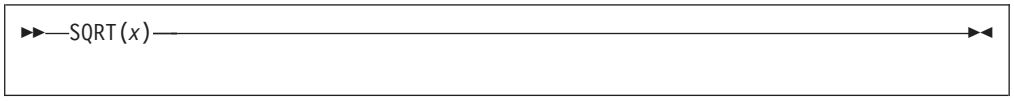
SOURCELINE() は、ステートメントが含まれているこのファイル内で、この関数が呼び出されるステートメントの行番号である FIXED BINARY(31,0) 値を戻します。ステートメントが、いくつかのソース行に拡張している場合は、ステートメントが開始する行の番号が戻されます。



SOURCELINE() は、制限付き式に使用できます。

SQRT

SQRT は、*x* の正の平方根の近似を浮動小数点の値で戻します。戻り値は、*x* の基数、モードおよび精度になります。



x 式。 *x* が実数の場合は、*x* はゼロより小さくてはいけません。

SQRTF

SQRTF は、次のこと以外は、SQRT とまったく同じです。
- SQRTF は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 引数は、実数でなければならない。
- 無効な引数は、ハードウェア例外を生成する。
- 結果の正確度は、ハードウェアによって設定される。

定義および構文については、 508 ページの『SQRT』を参照してください。

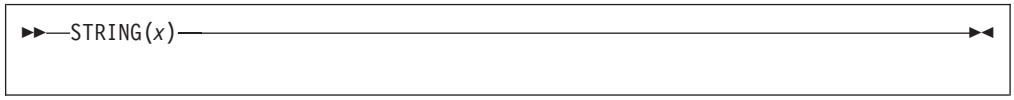
STORAGE

省略形: STG

STORAGE は、SIZE の同義語です。 507 ページの『SIZE』の構文を参照してください。

STRING

STRING は、*x* のすべてのエレメントを連結したストリングを戻します。



x 集合体参照またはエレメント参照。

STRING には、次の制限があります。
- 共用体を含む共用体または構造体には適用できない。
- スカラーに適用した場合は、スカラーはビット・ストリング、文字ストリング、ピクチャー漢字ストリング、ピクチャー数値ストリング、漢字ストリング、またはワイド文字ストリングでなければなりません。
- 構造体に適用する場合は、構造体に埋め込みバイトがなく、構造体のエレメントは次のどちらかでなければなりません。
  - すべて位置合わせされていないビット・ストリング。
  - すべて文字ストリングで、それぞれ文字ストリング、ピクチャー付き列、またはピクチャー付き数値ストリングのどれか。
  - すべて漢字ストリング。
  - すべてワイド文字ストリング。
- 配列に適用する場合は、配列内のすべてのエレメントは、以前に説明した制限を受けます。

戻されるストリングのタイプは、以下の例外がありますが、基本エレメントの 1 つと同じタイプになります。
- 基本エレメントのいずれかが PICTURE である場合、戻されるタイプは CHARACTER タイプになります。

- 基本エレメントのいずれかが GRAPHIC タイプである場合、CHARACTER である必要がある STRINGOFGRAPHIC コンパイラー・オプションが指定されていなければ、戻されるタイプは GRAPHIC になります。

有効な STRING ターゲットを次に示します。

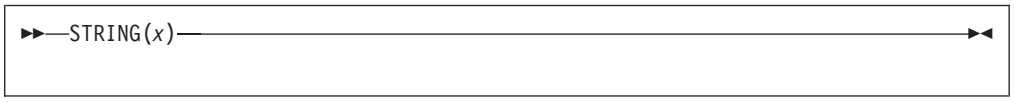
```
dcl
  1 A,
    2 B  bit(8),
    2 C  bit(2),
    2 D  bit(8);

dcl
  1 W,
    2 X  char(2),
    2 Y  pic'aa',
    2 Z  char(6);

dcl
  1 W,
    2 X  char(2),
    2 Y  pic'99',
    2 Z char(6);
```

無効な STRING ターゲットを次に示します。

```
dcl
  1 A,
    2 B  bit(8) aligned,
    2 C  bit(2),
    2 D  bit(8) aligned;
```

STRING 疑似変数

STRING 疑似変数は、*x* がストリングのスカラーであるように、ストリングを *x* に割り当てます。 *x* の残りのストリングは、ブランクまたはゼロ・ビットで埋められるか、可変長の場合は長さゼロが与えられます。



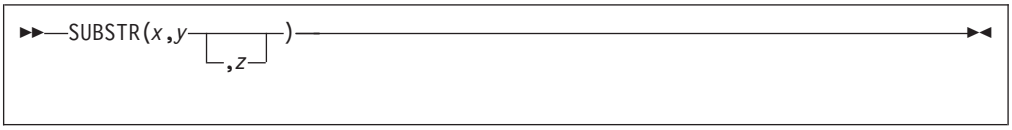
x 集合体参照またはエレメント参照。 *x* の各基本エレメントは、すべてビット・ストリングかすべて文字ストリングでなければなりません。

STRING 疑似変数は、コンテキスト以外では使用してはいけません。

また、疑似変数は、STRING 組み込み関数に制限されます。制限の詳細については、509 ページを参照してください。

SUBSTR

SUBSTR は、サブストリングを戻します。このサブストリングは、*x* の *y* および *z* によって指定されます。



x ストリング式。サブストリングが抜き出されるストリングを指定する。 *x* がストリングでない場合は、文字に変換されます。

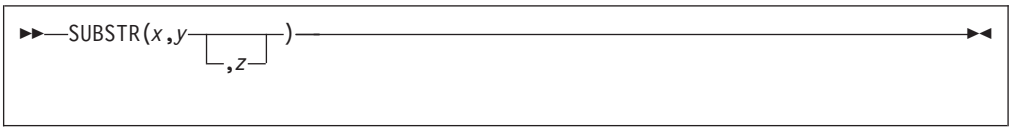
y FIXED BINARY(31,0) に変換される式。 *y* は、*x* の中のサブストリングの開始位置を指定する。

z FIXED BINARY(31,0) に変換される式。 *z* は、*x* の中のサブストリングの長さを指定します。*z* がゼロの場合は、ヌル・ストリングが戻されます。 *z* が省略される場合は、戻されるサブストリングは *x* の位置 *y* から *x* の終わりまでです。

z が負または、*y* および *z* の値が *x* の現行の長さに全体が入らないサブストリングなどの場合は、STRINGRANGE 条件が発生します。 *y* = LENGTH(*x*)+1 および *z* = 0 のときには、この条件は発生しません。 SUBSTR 組み込み関数の例については、 501 ページの『SEARCH』を参照してください。

SUBSTR 疑似変数

SUBSTR 疑似変数は、*x* の *y* および *z* で指定されるストリングの値をサブストリングに割り当てます。*x* の剰余は、変更されません。可変ストリングへの割り当ては、ストリングの長さを変更しません。



x ストリング参照。 *x* は、数字であってはいけません。

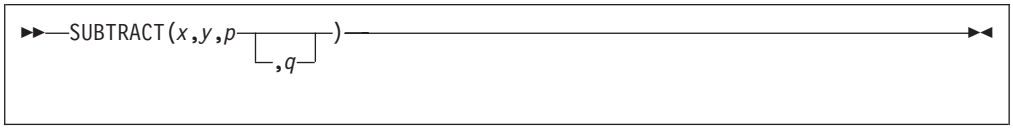
y 式。 *x* のサブストリングの開始位置を指定する FIXED BINARY 値に変換可能な *y* の式。

z 式。 *z* は、*x* の中のサブストリングの長さを指定します。*z* は実数の固定小数点のバイナリー値に変換できます。 *z* がゼロの場合は、ヌル・ストリングが戻されます。 *z* が省略される場合は、戻されるサブストリングは *x* の位置 *y* から *x* の終わりまでです。

y および *z* は、*x* が配列である場合にのみ、配列であることが可能です。

SUBTRACT

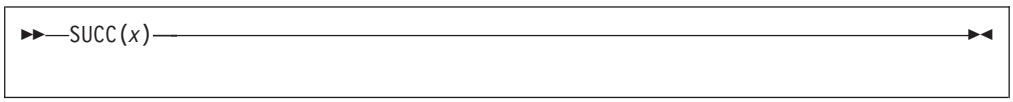
SUBTRACT は、ADD(x,-y,p,q) と同等です。



引数の詳細については、 415 ページの『ADD』の引数の説明を参照してください。

SUCC

SUCC は、*x* より大きい表示可能な最小数である浮動小数点の値を戻します。戻り値は、*x* の基数、モード、および精度になります。OVERFLOW 条件は、そのような数がない場合に発生します。



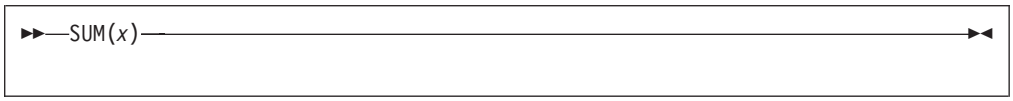
x REAL FLOAT 式。

SUCC は、次の関係を満たしています。

```
pred(succ(x)) = x
succ(pred(x)) = x
succ(x)       = -pred(-x)
succ(0d0)     = tiny(0d0)
```

SUM

SUM は、*x* のすべてのエレメントの和を戻します。結果の基数、モード、およびスケールは、*x* の基数、モード、およびスケールと同じです。



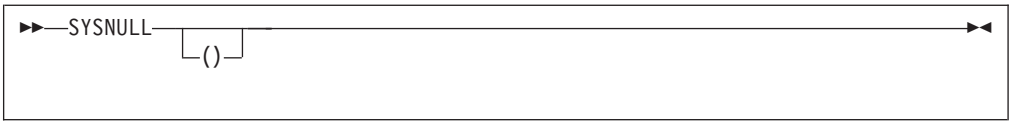
x 配列式。 *x* のエレメントがストリングの場合は、それらのエレメントは固定小数点の整数値に変換されます。

x のエレメントが固定小数点の場合は、結果の精度は (*N*,*q*) です。 N は、許可される桁の最大数で、*q* は *x* のスケール因数です。

x のエレメントが浮動小数点の場合は、結果の精度は *x* と一致します。

SYSNULL

SYSNULL は、システム・ヌル・ポインターの値を戻します。 SYSNULL をハンドルに割り当て、SYSNULL とハンドルを比較することができます。 SYSNULL を使用して静的ポインターおよびオフセット変数を初期化できます。

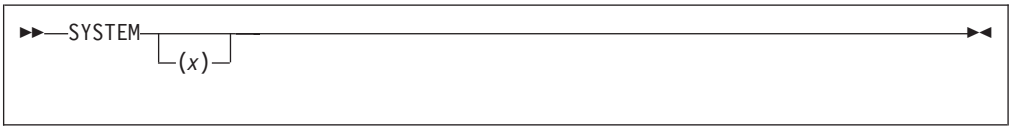


注: NULL および SYSNULL を比較すると等しい場合があります。ただし、これらが等しいということに基づいてコーディングしないでください。

472 ページの『NULL』も参照してください。

SYSTEM

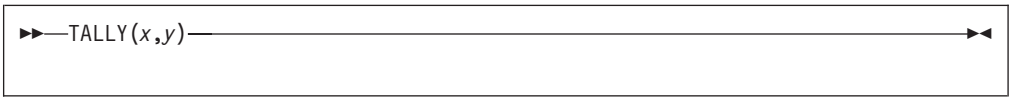
SYSTEM(x) は、*x* に含まれているコマンドで呼び出されるときにコマンド・プロセッサーからの戻り値である FIXED BIN(31,0) 値を戻します。



x *x* に、計算タイプ (必須) と文字タイプを保持している必要があります。そうでない場合は、*x* は文字に変換されます。

TALLY

TALLY は、ストリング *x* にストリング *y* が表示される回数を示す FIXED BINARY(31,0) 結果を戻します。 *x* 内に *y* が表示されない場合は、値 0 が戻されます。



x および y
ストリング式。

x および *y* の両方が計算タイプであり、文字、ビット、グラフィック、またはワイド文字タイプである必要があります。

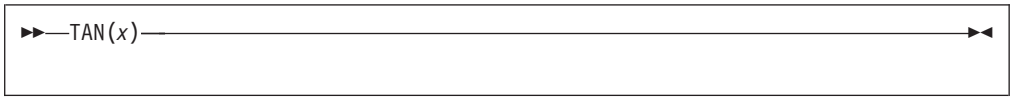
x または *y* はヌル・ストリングで、結果はゼロです。

例

```
TALLY ('We''ve got the Power!', 'power');      /* returns 0 */
TALLY ('We''ve got the Power!', 'Power');      /* returns 1 */
TALLY ('We''ve got the Power!', ' ');          /* returns 3 */
TALLY ('We''ve got the Power!', 'e');          /* returns 4 */
TALLY ('1001'B, '1'B);                         /* returns 2 */
```

TAN

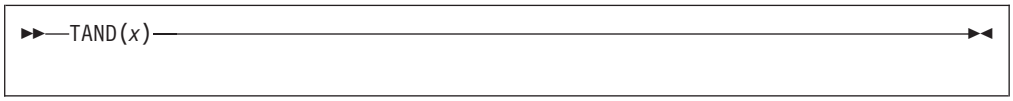
TAN は、*x* の正接の近似を浮動小数点の値で戻します。戻り値は、*x* の基数、モードおよび精度になります。



x ラジアン値の式。

TAND

TAND は、*x* の正接の近似を実数の浮動小数点の値で戻します。戻り値は、*x* の基数および精度になります。



x 度で示される値を持つ実数式。

TANF

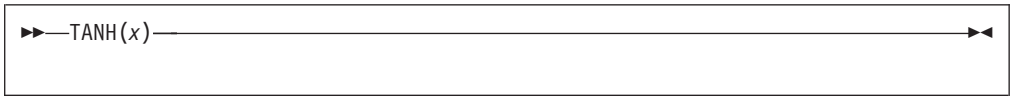
TANF は、次の点を除いては TAN とまったく同じです。

- TANF は、ハードウェア・アーキテクチャーによって処理可能ならば、結果をインラインで計算する。
- 引数は、実数でなければならない。
- サポートされる引数の最大絶対値は、ハードウェアによって設定される。
- 無効な引数が指定された場合、INVALIDOP 条件が発生したり、その他のハードウェア例外が生成されたりするなど、予期できない結果が生じることがある。
- 結果の正確度は、ハードウェアによって設定される。

定義および構文については、『TAN』を参照してください。

TANH

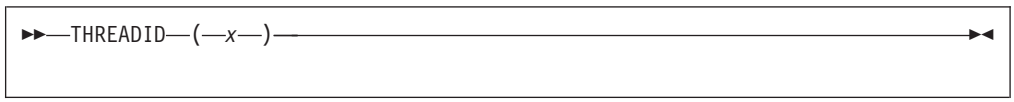
TANH は、*x* の双曲線正接の近似を浮動小数点の値で戻します。戻り値は、*x* の基数、モードおよび精度になります。



x ラジアン値の式。

THREADID

THREADID (スレッド ID の短縮形) は、POINTER 値を戻します。この値は、接続されたスレッドのオペレーティング・システム・スレッド ID のアドレスです。



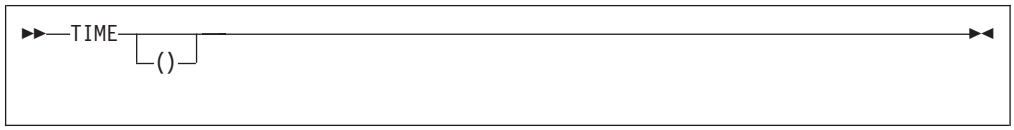
x タスク参照。 *x* の値は、ATTACH ステートメントの THREAD オプションですでに設定されているはずです。

この組み込み関数によって戻される値を使用して、DosSetPriority などのシステム関数 (Windows の場合)、または posix 関数 (z/OS の場合) を呼び出すことができます。

現在実行中のスレッドのシステム・スレッド ID を取得するには、そのスレッドが実行されているプラットフォームに適した関数を呼び出す必要があります。 そのため、Windows では GetCurrentThreadId、z/OS では pthread_self を呼び出してください。

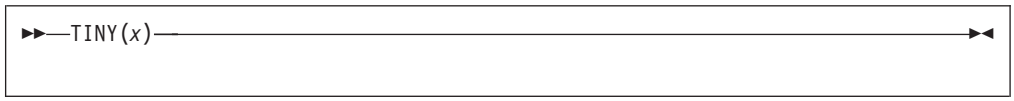
TIME

TIME は、HHMISS999 のフォーマットで、文字ストリング・タイム・スタンプを戻します。



TINY

TINY は、*x* が取ることができる正の最小値を浮動小数点の値で戻します。戻り値は、*x* の基数、モード、および精度になります。

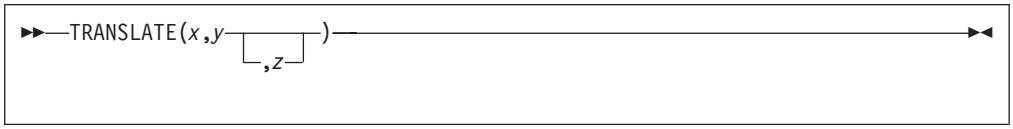


x REAL FLOAT 式。

TINY(x) は定数で、制限付き式に使用できます。

TRANSLATE

TRANSLATE は、*x* と同じ長さの文字ストリングを戻します。



x 変換できる文字があるかどうかを検索される文字式。

y 文字の変換値を含む文字式。

z 変換される文字を含む文字式。 *z* が省略される場合は、デフォルトは collate() です。

TRANSLATE は、*x* の各文字を次のように処理します。

x の中にある文字が *z* に検出される場合は、*y* にある *z* に対応する文字が結果にコピーされます。それ以外の場合は、 *x* の文字が結果にコピーされます。 *z* に複数の同一文字が入っている場合は、左端のオカレンスが使われます。

y は、*z* の長さと一致するように、右がブランクで埋められるか、切り捨てられます。

すべての算術引数またはビット引数は、文字に変換されます。 TRANSLATE は、GRAPHIC または WIDECHAR のデータをサポートしません。

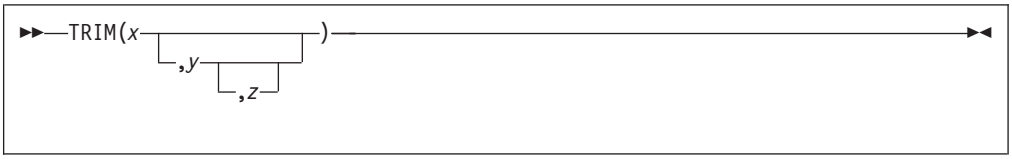
例

```
dcl source char value("Ein Raetsel gibt es nicht.");
dcl target char(length(source));
dcl (to   value ('ABCDEFGHIJKLMNOPQRSTUVWXYZ'),
     from value ('abcdefghijklmnopqrstuvwxyz')) char;

target = translate(source, to, from);
         /* "EIN RAETSEL GIBT ES NICHT." */
```

上記の例における TRANSLATE 組み込み関数と同じ目的で UPPERCASE を使用することもできます。しかし、UPPERCASE 組み込み関数が規格英字のみを変換するのに対して、TRANSLATE は他の文字への変換に使用することができます。例えば、「Raetsel」が a のウムラウトを使用してつづられている場合、TRANSLATE は a のウムラウトを A のウムラウトに変換します (これらの文字が *from* ストリングおよび *to* ストリングにそれぞれ追加されている場合)。

TRIM

TRIM は、片側または両端がトリムされた文字を使用した不変文字ストリングを戻します。



x、y、および z
式。

それぞれに、計算タイプ (必須) および属性 CHARACTER を保持している必要があります。そうでない場合は、変換されます。

x は、*y* によって定義された文字が左からトリムされ、*z* によって定義された文字が右からトリムされたストリングです。

z が省略される場合は、1 つのブランクを含む CHARACTER(1) NONVARYING ストリングをデフォルトにします。

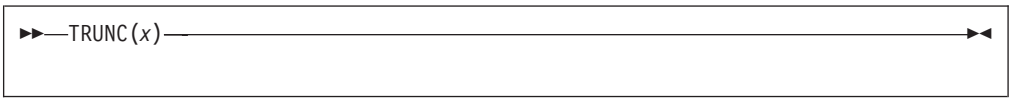
y と *z* が両方とも省略される場合は、両方とも 1 つのブランクを含む CHAR(1) NONVARYING ストリングをデフォルトにします。

例

```
dcl Source char value(" *** PL/I's got the Power!  ***  ");
dcl Target char(length(Source)) varying;

Target = trim(Source, ' ', '* ');
             /* "*** PL/I's got the Power!" */
```

TRUNC

TRUNC は、*x* を切り捨ててできる整数値を戻します。 *x* が、正または 0 の場合は、*x* より小さいか等しい最も大きい整数値です。*x* が負の場合、*x* より大きいか等しい最も小さい整数値です。この値は、結果に割り当てられます。



x 実数式。

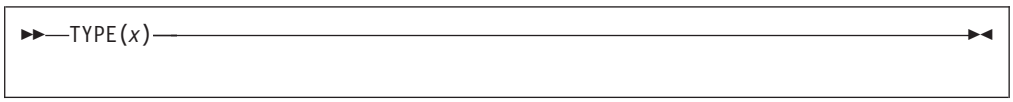
結果の基数、モード、スケール、精度は、*x* の基数、モード、およびスケールと同じです。*x* が精度 (*p,q*) を使用した固定小数点であるときは、結果の精度は次の式によって指定されます。

```
(min(N,max(p-q+1,1)),0)
```

この場合 N は、使用できる最大の桁数です。

TYPE

TYPE は、ハンドル *x* によって位置付けされるタイプ付き構造体または共用体を戻します。

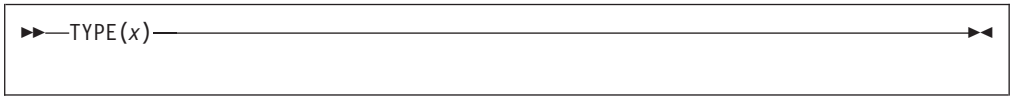


x ハンドル

TYPE(x) は、タイプ付き構造体 (または共用体) *x* の参照を解除します。 TYPE 組み込み関数の例については、『TYPE 疑似変数』を参照してください。

TYPE 疑似変数

TYPE 疑似変数は、タイプ付き構造体または共用体をハンドル *x* によって位置付けられたストレージに割り当てます。



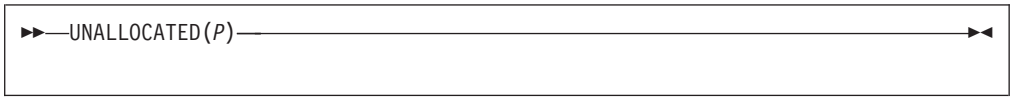
x ハンドル

定義済み構造体 *T* が与えられると、次の割り当てが有効です。

```
dcl P1 handle T;
dcl P2 handle T;
dcl D1 type T;
dcl D2 type T;

D1 = type(P2);          /* Assigns the storage located by P2 to D1 */
type(P1) = type(P2);
type(P1) = D2;          /* Assigns D2 to the storage located by P1 */
```

UNALLOCATED

UNALLOCATED は、指定されたポインター値が割り振り済みストレージの始まりであるかどうかを示すビット (1) 値を戻します。この組み込み関数を使用するには、CHECK(STORAGE) コンパイル時オプションも指定しなければなりません。

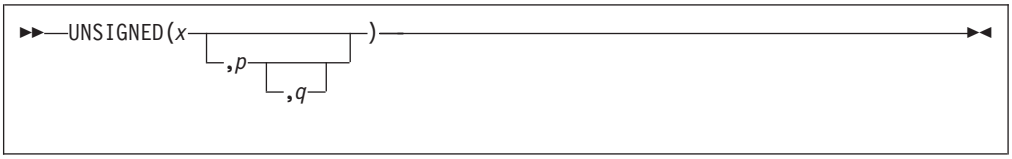


p ポインターの式。

指定されたポインター値が、ALLOCATE ステートメントまたは ALLOCATE 組み込み関数によって取得したストレージの始まりである場合に、UNALLOCATED は bit(1) 値 '1'b を戻します。

UNSIGNED

UNSIGNED は、*p* および *q* で指定された精度で、*x* の符号なしの FIXED BINARY 値を戻します。



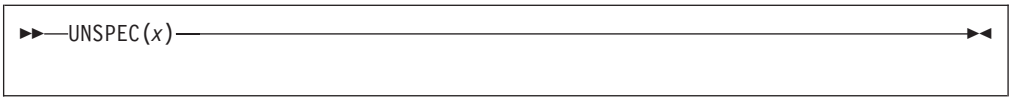
x 式。

p 整数。演算を通して維持される桁数を指定します。

q オプションの符号付き整数。結果のスケール因数を指定します。固定小数点の結果の場合は、*p* が指定され、*q* が省略されると、スケール因数ゼロがデフォルトになります。

UNSPEC

UNSPEC は、*x* の内部コードであるビット・ストリングを戻します。



x スカラー、配列、構造体、または共用体の式。

UNSPEC 組み込み関数は、次の規則に従います。

- コンパイラー・オプション USAGE(UNSPEC(IBM)) の場合
  - 構造体参照と式の UNSPEC は使用できません。
  - 配列の UNSPEC 結果は、BIT の配列になります。
- コンパイラー・オプション USAGE(UNSPEC(ANS)) の場合
  - 集合体の場合、UNSPEC は埋め込みバイトまたはビットを含まないものにだけ使用できます。
  - 結果は常に BIT スカラーです。配列の UNSPEC 結果は、BIT の配列にはなりません。

注: UNSPEC の使用により、プログラムの移植性に影響を与えることがあります。

戻されるビット・ストリングの長さは、表 65 に示すように、 *x* の属性によって決まります。

表 65. UNSPEC によって戻されるビット・ストリングの長さ

ビット・ストリングの長さ	*x* の属性
8	SIGNED FIXED BINARY(p,q), 1 <= p <= 7 UNSIGNED FIXED BINARY(p,q), 1 <= p <= 8 ORDINAL SIGNED PRECISION(p), 1 <= p <= 7 ORDINAL UNSIGNED PRECISION(p), 1 <= p <= 8

表 65. UNSPEC によって戻されるビット・ストリングの長さ (続き)

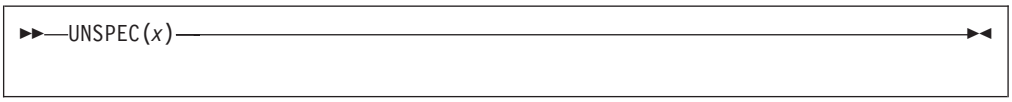
ビット・ストリングの長さ	*x* の属性
16	SIGNED FIXED BINARY(p,q), 8 <= p <= 15 UNSIGNED FIXED BINARY(p,q), 9 <= p <= 16 ORDINAL SIGNED PRECISION(p), 8 <= p <= 15 ORDINAL UNSIGNED PRECISION(p), 9 <= p <= 16
32	ENTRY LIMITED SIGNED FIXED BINARY(p,q), 16 <= p <= 31 UNSIGNED FIXED BINARY(p,q), 17 <= p <= 32 ORDINAL SIGNED PRECISION(p), 16 <= p <= 31 ORDINAL UNSIGNED PRECISION(p), 17 <= p<= 32 FLOAT DECIMAL(p), 1 <= p <= 6 FLOAT BINARY(p), 1 <= p <= 21 OFFSET FILE 定数または変数 POINTER HANDLE
64	SIGNED FIXED BINARY(p), 31 < p UNSIGNED FIXED BINARY(p), 32 < p FLOAT BINARY(p), 21 < p < 53 FLOAT DECIMAL(p), 7 <= p <= 16 LABEL 定数または変数 ENTRY 定数または変数
128	FLOAT BINARY(p), 54 <= p FLOAT DECIMAL(p), 17 <= p TASK
n	BIT(n)
8*n	CHARACTER(n)PICTURE (長さ *n* の文字ストリング値付き)
16*n	GRAPHIC(n)WIDECHAR(n)
16+n	*n* が *x* の最大の長さの BIT(n) VARYING
16+(8*n)	*n* が *x* の最大の長さの CHARACTER(n) VARYING
8+(8*n)	*n* が *x* の最大の長さの CHARACTER(n) VARYINGZ
16+(16*n)	*n* が *x* の最大の長さの GRAPHIC(n) VARYING、*n* が *x* の最大の長さの WIDECHAR(n) VARYING
16+(16*n)	*n* が *x* の最大の長さの GRAPHIC(n) VARYINGZ、 *n* が *x* の最大の長さの WIDECHAR(n) VARYINGZ
8*(n+16)	AREA (n)
8*FLOOR(n)	n = (p+2)/2 の FIXED DECIMAL (p,q)

プログラム制御データの位置合わせおよびストレージ所要量は、サポートされるシステムをまたがって変更することができます。

x が可変長ストリングの場合は、その 2 バイト接頭部が戻されるビット・ストリングに含まれます。 *x* が区域の場合は、戻り値は制御情報を含みます。

UNSPEC 疑似変数

UNSPEC 疑似変数は、直接、ビット値を *x* に割り当てます。変換は行いません。必要な場合には、ビット値は、*x* の長さが合うように、表 65 に従って、'0'B で右側が埋まります。



x 参照。

x が可変長ストリングの場合は、その 2 バイト接頭部が、ビット値が割り当てられるフィールドに含まれます。 *x* が区域の場合は、制御情報は、受信フィールドに含まれます。

疑似変数は、 519 ページの『UNSPEC』で説明されている UNSPEC 組み込み関数の規則に従います。

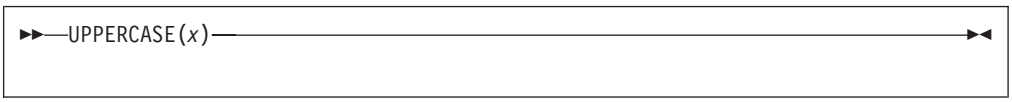
注: UNSPEC の使用により、プログラムの移植性に影響を与えることがあります。

例

```
dcl 1 Str1 nonasgn static,
      2 * fixed bin(15) littleendian init(16),  /* '1000'X */
      2 * char init('33'x),
      2 * bit init('1'b),
      2 Ba(4) bit init('1'b, '0'b, '1'b, '0'b),
      2 B3 bit(3) init('111'b),
      2 * char(0);
dcl Bit_Str1 bit(size(Str1)*8);
dcl Bit_Ba   bit(dim(Ba)*length(Ba(1)));
dcl Bit_B3   bit(length(B3));

Bit_Ba   = unspec(Ba);   /* result is scalar '1010'B not an array */
Bit_B3   = unspec(B3);   /* '111'B                                */
Bit_Str1 = unspec(Str1); /* '100033D7'B4 or
                            '100033'B4 || '11010111'B             */
```

UPPERCASE

UPPERCASE は、a から z までの英字すべてを同等な大文字に変換した文字ストリングを戻します。



x 式。必要に応じ、*x* は文字に変換されます。

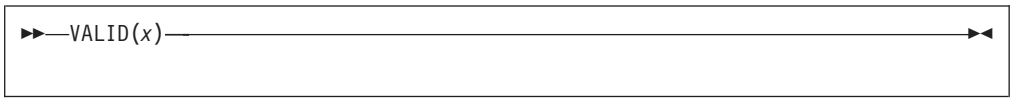
UPPERCASE(x) は次のステートメントと同等です。

```
TRANSLATE( x,
           'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
           'abcdefghijklmnopqrstuvwxyz' )
```

VALID

VALID は、次の条件のもとで、'1'B である BIT(1) 値を戻します。
- *x* がピクチャーでその内容が *x* のピクチャー指定に有効な場合。
- *x* が FIXED DECIMAL で、*x* のデータは有効な固定小数点である場合。

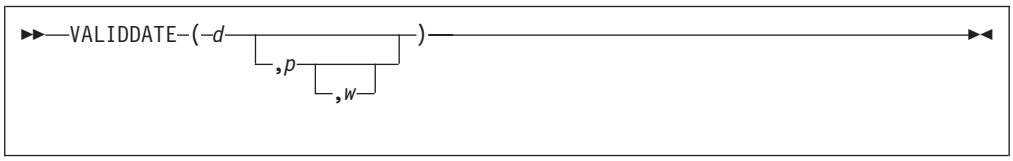
これらの条件と合わない場合は、結果は '0'B です。



x 各ピクチャーまたは固定小数点タイプでの参照。

VALIDDATE

ストリング d がパターン p と一致する日付/時刻値を保持する場合は、VALIDDATE は、'1'B を戻します。



d 日付を表すストリング式。

存在する場合は、*d* は入力日付をパターン *p* によって決まる日付/時刻を表示する文字ストリングとして指定します。 *d* が指定されない場合は、「DATETIME()」が想定されます。

d に計算タイプ (必須) および文字タイプを保持する必要があります。そうでない場合は、*d* は文字に変換されます。

p サポートされている日付/時刻パターンの 1 つ。

存在する場合は、*d* の日付/時刻パターンを指定します。 *p* が指定されない場合は、デフォルトの日付/時刻のパターン「YYYYMMDDHHMISS999」が想定されます。

p に計算タイプ (必須) および文字タイプを保持する必要があります。そうでない場合は、*p* は文字に変換されます。

w 整数に変換できる式 (1950 など) を指定します。負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値にデフォルトで設定されます。

許可されるパターンを 406 ページの表 52 にリストしてあります。リリアン形式の説明については、404 ページの『日付/時刻組み込み関数』を参照してください。

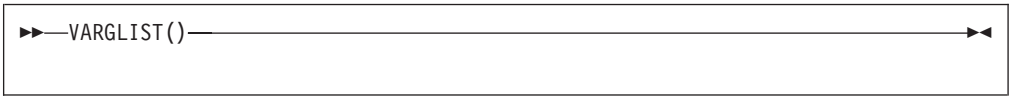
例

```
dcl duedate   char(8);
dcl (b1,b2)   bit(1);
```

```
duedate = '19950228';
b1 = validdate( duedate, 'YYYYMMDD' ); /* b1 = '1'b */

duedate = '02301995';
b2 = validdate( duedate, 'DDMMYYYY' ); /* b2 = '0'b */
```

VARGLIST

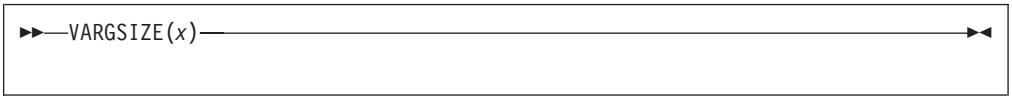
VARGLIST は、引数の変数番号を持つプロシージャーに渡された最初のオプショナル・パラメーターのアドレスを戻します。



VARGLIST は、最後のパラメーターが LIST 属性をとるプロシージャーの中でのみ使用されます。

VARGSIZE

VARGSIZE は、byvalue を渡された場合に変数がスタック上で占めるバイト数を戻します。



x 任意のデータ・タイプ、データ編成、位置合わせ、およびストレージ・クラスを持つ変数。ただし、下記のものは除きます。

x は、次のものであることはできません。

- 位置合わせされていない固定長ビット・ストリングである BASED 変数、DEFINED 変数、パラメーター変数、添え字付き変数、構造変数、または共用体基底付き変数。
- 最初または最後の基本エレメントが位置合わせされていない固定長のビット・ストリングである小構造体または共用体 (ただし、その小構造体を含んでいる大構造体または共用体の最初または最後のエレメントになっている場合を除く)。
- BASED 属性、DEFINED 属性、またはパラメーター属性を持つ大構造体か共用体、またはパラメーターである大構造体か共用体で、位置合わせされていない固定長のビット・ストリングを最初または最後のエレメントとするもの。
- 連結ストレージにない変数。

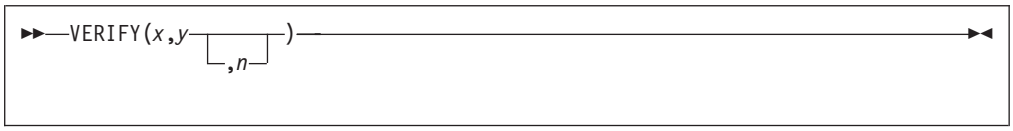
VARGSIZE(x) は、byvalue を渡された場合に x がスタック上で占めるバイト数を戻します。この値は、少なくとも SIZE(x) と同じ大きさになります。 SIZE(x) によって戻される値が 4 バイトの倍数まで切り上げられる必要がある場合にはこの値はそれより大きくなります。

VARGSIZE は、最後のパラメーターが LIST 属性をとるプロシージャーの中でのみ使用されなければなりません。

VERIFY

VERIFY は、*y* にない左端の文字、ワイド文字、グラフィック、またはビットの *x* 内での位置を示すスケールのない REAL FIXED BINARY 値を戻します。この関数により、処理を開始する *x* 内の位置を指定できます。

x のすべての文字、ワイド文字、グラフィック、またはビットが *y* にある場合は、値ゼロが戻されます。 *x* がヌル・ストリングの場合は、値ゼロが戻されます。 *x* がヌル・ストリングでなく、*y* がヌル・ストリングの場合は、 *n* の値が戻されます。 *n* のデフォルトは 1 です。



x ストリング式。

y ストリング式。

n *n* は、処理を開始する *x* 内の位置を指定します。 *n* には、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

1 ≤ *n* ≤ LENGTH(*x*)+1 でない場合は、 STRINGRANGE 条件 (割り込み可能な場合) が発生します。暗黙処置と通常の戻りでは、結果は 0 になります。*n* = LENGTH(*x*) + 1 の場合は、結果はゼロです。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

例

```
X = '  a  b';         /* Two blanks in each space */
Y = ' ';              /* One blank                */
N = 1;
I = verify(X,Y,N);    /* I = 3 */

do while (I > 0);
  display ( 'Nonblank at position ' || trim(I) );
  N = I + 1;
  I = verify(X,Y,N);
end;
```

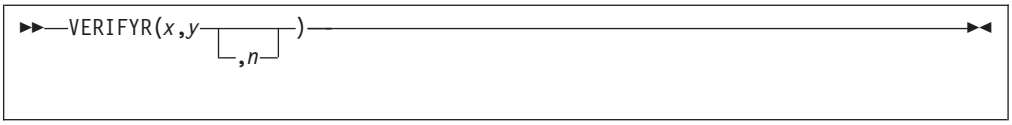
DO ループの最初のパススルーのあとで、N=4 で VERIFY(X,Y,N) は 6 を戻します。2 回目のパスのあとで、N=7 (LENGTH(x)+1) で VERIFY(X,Y,N) は 0 を戻し、ループが終了します。

他の VERIFY 組み込み関数の例については、 501 ページの『SEARCH』を参照してください。

VERIFYR

VERIFYR 関数は、次のこと以外は VERIFY 組み込み関数と同じ演算を行います。

- 検査は、右から左に行われます。
- *n* のデフォルトは LENGTH(*x*)。



0 ≤ *n* ≤ LENGTH(*x*) でないと、STRINGRANGE 条件 (割り込み可能な場合) が発生します。 *n* = 0 の場合は、結果はゼロです。

BIFPREC コンパイラー・オプションによって、戻される結果の精度が決まります。

引数の説明については、 524 ページの『VERIFY』を参照してください。

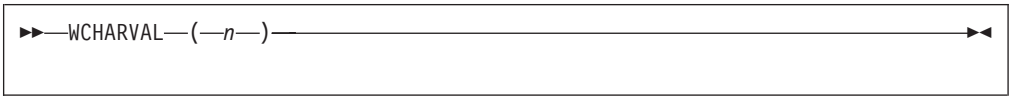
例

```
X = 'a  b  ';          /* Two blanks in each space */
Y = ' ';               /* One blank                */
N = length(X);         /* N = 6                 */
I = verifyr(X,Y,N);    /* I = 4                 */

do while (I > 0);
  display ( 'Nonblank at position ' || trim(I) );
  N = I - 1;
  I = verifyr(X,Y,N);
end;
```

DO ループの最初のパススルーのあとで、N=3 および VERIFYR(X,Y,N) は 1 を戻します。2 回目のパスのあとで、N=0 で VERIFYR(X,Y,N) は 0 を戻し、ループが終了します。別の例については、 503 ページの『SEARCHR』を参照してください。

WCHARVAL

WCHARVAL は、整数に対応する WIDECHAR(1) 値を戻します。



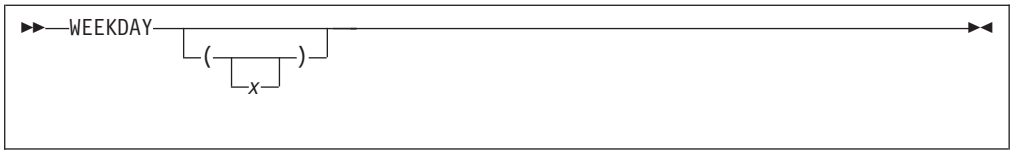
n 式。必要な場合には、UNSIGNED FIXED BIN(16) に変換されます。

n がビッグ・エンディアン・フォーマットの場合、WCHARVAL(n) のビット値は n と同じですが (つまり、 UNSPEC(WCHARVAL(n)) は UNSPEC(n) と等しい)、属性は WIDECHAR(1) です。

WCHARVAL は RANK の逆です (ワイド文字に適用した場合)。

WEEKDAY

WEEKDAY は、曜日に変換された日数 *x* を FIXED BINARY(31,0) 値で戻します。ここで 1= 日曜日、2= 月曜日、 . . 7= 土曜日です。 *x* が指定されない場合は、今日の日付と見なされます。



x 式。存在する場合は、*x* は、入力データを日数として指定します。存在しない場合は、*x* は、DAYS() であると想定されます。

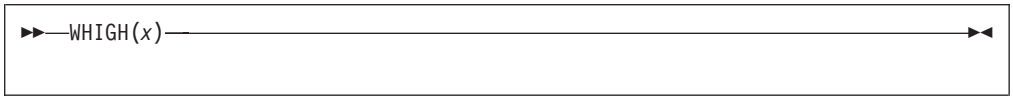
x がなく、今日の日付がシステムから利用できない場合は、結果のゼロが戻されます。

x は、計算タイプでなければなりません。また、必要な場合には、 FIXED BINARY(31,0) に変換されます。

WEEKDAY の例については、 503 ページの『SECS』を参照してください。

WHIGH

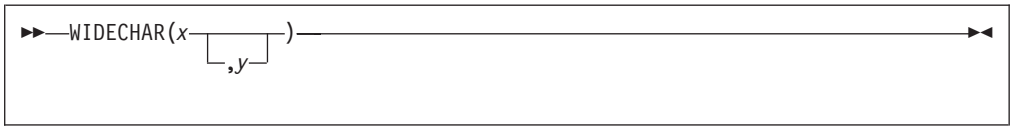
WHIGH は、それぞれ最大のワイド文字値 (16 進 FFFF) を持つワイド文字で構成される、長さ *x* のワイド文字ストリングを戻します。



x 式。必要な場合には、*x* は正の実数固定小数点の 2 進数の値に変換されます。*x* = 0 の場合は、結果はヌルのワイド文字ストリングです。

WIDECHAR

WIDECHAR は、*y* で指定された長さでワイド文字値 *x* を戻します。



省略形: WCHAR

x 式。

n は、計算タイプでなくてはなりません。

x の値は、チェックされません。

y 式。必要な場合には、y は、実数の固定小数点の 2 進数の値に変換されます。

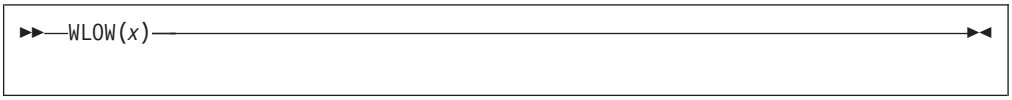
y が省略される場合は、長さはタイプ変換の規則によって決められます。

y は、負以外でなければなりません。

y = 0 の場合は、結果はヌルのワイド文字ストリングです。

WLOW

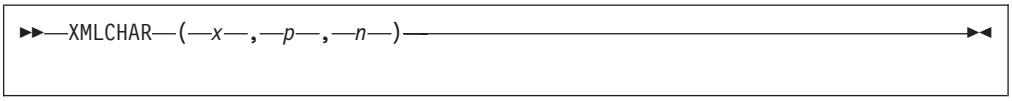
WLOW は、それぞれ最小のワイド文字値 (16 進 0000) を持つワイド文字で構成される、長さ *x* のワイド文字ストリングを戻します。



x 式。必要な場合には、*x* は正の実数固定小数点の 2 進数の値に変換されます。*x* = 0 の場合は、結果はヌルのワイド文字ストリングです。

XMLCHAR

XMLCHAR 組み込み関数は、構造体のデータを XML としてバッファーにダンプします。XMLCHAR は、バッファーに書き込まれたバイトの数を戻します。バッファーが小さすぎる場合、構造体データは切り捨てられ、バッファーが構造体を収容するのに必要なバイト数が戻されます。



x 構造体参照。

p ターゲット・バッファーのアドレス。

n ターゲット・バッファーの長さ。

バッファー長は、計算タイプを保持している必要があり、FIXED BINARY(31,0) に変換されます。

バッファー長は、負数であってはなりません。

構造体参照の *x* には、ストリングや数値データなどの計算可能なデータのみが含まれていなければなりません。構造体参照の *x* には配列が含まれていることがありますが、構造体参照自体が配列である場合は、完全に添え字が付いていなければなりません。

構造体参照の *x* には、副構造体が含まれていることがありますが、含まれている副構造体は名前の変わりに * を使用することはできません。ただし、基本エレメントの名前として * を使用することはできますが、その場合、ターゲット・バッファーに名前のないエレメントは書き込まれません。

xml が作成されると、以下のようになります。

- 構造体のすべての名前が、最初は「<」と「> 」で、最後は「</」と「>」で囲まれて書き込まれます。
- 数値とビット・データは、文字に変換されます。
- 先行ブランクと末尾ブランクは、可能な限りトリムされます。

例

このコードがフラグメント化されている場合

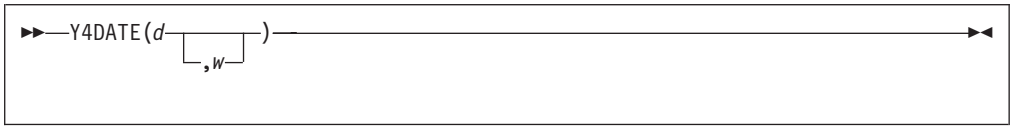
```
dcl buffer   char(800);
dcl written  fixed bin(31);
dcl next     pointer;
dcl left     fixed bin(31);
dcl
  1 a,
   2 a1,
     3 b1 char(8),
     3 b2 char(8),
   2 a2,
     3 c1 fixed bin,
     3 c2 fixed dec(5,1);

b1 = ' t1';
b2 = 't2';
c1 = 17;
c2 = -29;
next = addr(buffer);
left = stg(buffer);
written = xmlchar( a, next, left );
next += written;
left -= written;
```

以下のバイトがバッファーに書き込まれ、書き込みは、72 にセットされます。

```
<A><A1><B1>t1</B1><B2>t2</B2></A1><A2><C1>17</C1><C2>-29.0</C2></A2></A>
```

Y4DATE

Y4DATE は、パターン 'YYMMDD' を使って日付の値をとり、2 桁の年を 4 桁の年にして日付の値を戻します。



d 日付を表すストリング式。

d に計算タイプ (必須) および文字タイプを保持する必要があります。そうでない場合は、*d* は文字に変換されます。

w 整数に変換できる式 (1950 など) を指定します。負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値にデフォルトで設定されます。

戻り値は、属性 CHAR(8) NONVARYING をとっており、次のように計算されます。

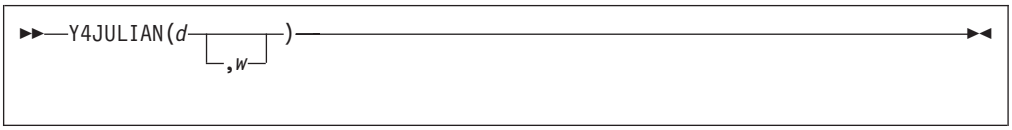
```
dcl y2 pic'99';
dcl y4 pic'9999';
dcl cc pic'99';

y2 = substr(d,1,2);
cc = w/100;

if y2 < mod(w,100) then
```

```
  y4 = 100*cc + 100 + y2;
else
  y4 = 100*cc + y2;

return( y4 || substr(d,3) );
```

Y4DATE('990101',1950) は '19990101' を戻します。
Y4DATE('000101',1950) は '20000101' を戻します。

Y4JULIAN

Y4JULIAN は、パターン 'YYDDD' を使って日付の値をとり、2 桁の年を 4 桁の年にして日付の値を戻します。



d 日付を表すストリング式。 *d* の長さは、少なくとも 5 でなければなりません。5 より長い場合は、余分の文字を先行ブランクにしなければなりません。

d に計算タイプ (必須) および文字タイプを保持する必要があります。そうでないと、文字に変換されます。

w 整数に変換できる式 (1950 など) を指定します。負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値にデフォルトで設定されます。

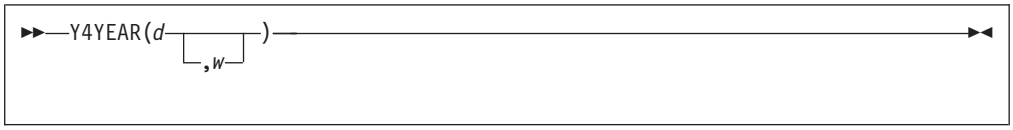
戻り値は、属性 CHAR(7) NONVARYING をとっており、次のように計算されます。

```
dcl y2 pic'99';
dcl y4 pic'9999';
dcl c  pic'99';

y2 = substr(d,1,2);
cc = w/100;

if y2 < mod(w,100) then
  y4 = 100*cc + 100 + y2;
else
  y4 = 100*cc + y2;

return( y4 || substr(d,3) );
```

Y4JULIAN('99001',1950) は '1999001' を戻します。
Y4JULIAN('00001',1950) は '2000001' を戻します。

Y4YEAR

Y4YEAR は、パターン 'YY' を使って日付の値をとり、2 桁の年を 4 桁の年にして日付の値を戻します。



d 日付を表すストリング式。 *d* の長さは、少なくとも 2 でなければなりません。2 より長い場合は、余分の文字を先行ブランクにしなければなりません。

d に計算タイプ (必須) および文字タイプを保持する必要があります。そうでないと、文字に変換されます。

w 整数に変換できる式 (1950 など) を指定します。負の場合、コードを実行するときに年の値から減算されるようにオフセットを指定します。省略される場合は、*w* は、WINDOW コンパイル時オプションに指定された値にデフォルトで設定されます。

戻り値は、属性 CHAR(4) NONVARYING を持っており、次のように計算されます。

```
dcl y2 pic'99';
dcl y4 pic'9999';
dcl c  pic'99';

y2 = d;
cc = w/100;

if y2 < mod(w,100) then
  y4 = 100*cc + 100 + y2;
else
  y4 = 100*cc + y2;

return( y4 );
```

Y4YEAR('99',1950) は '1999' を戻します。
Y4YEAR('00',1950) は '2000' を戻します。

第 20 章 タイプ付き関数

タイプ付き関数の呼び出し 531
タイプ付き関数の引数の指定 531
タイプ付き関数の要旨 532
BIND 532
CAST 532
FIRST 533
例 533
LAST 533
例 533
NEW 534
RESPEC 534
SIZE 534

タイプ付き関数を使用して、定義されたタイプを操作できます。タイプ付き関数は、次の方法で組み込み関数と区別します。

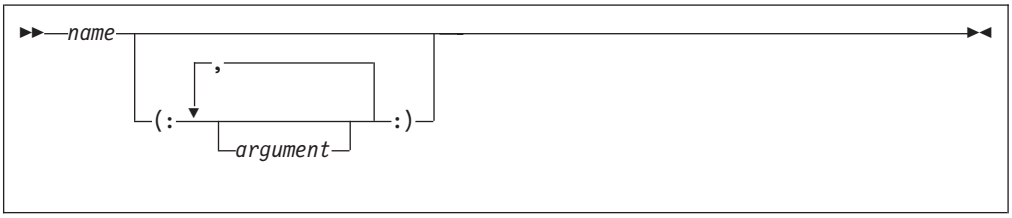
- 1 つ以上の引数が定義済みタイプである。
- それらを宣言することはできない。
- 引数は、(と) との記号ではなく、(: と :) との複合記号で囲む。

この章では、タイプ付き関数がアルファベット順にリストされています。一般的に、次のようなフォーマットで記述します。

- 参照構文を示す見出し
- 戻り値の説明
- 引数の説明
- 関数の使用時のその他の修飾

タイプ付き関数の呼び出し

次の構文を使ってタイプ付き関数を呼び出します。



タイプ付き関数の引数は、区切り文字 (: および :) で囲みます。

タイプ付き関数の引数の指定

タイプ付き関数の引数には、タイプ名 (別名、名前付き構造体および共用体、序数) と、その他のデータ・タイプが可能です。

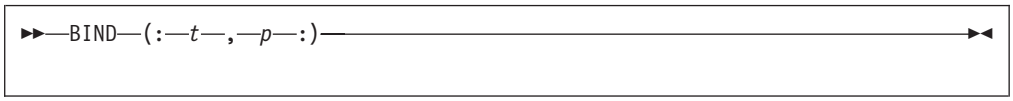
タイプ付き関数の要旨

表 66. タイプ付き関数

関数	説明
BIND	ポインターをタイプのハンドルに変換します。
CAST	C の変換規則を使用して、式を指定のタイプに変換します。
FIRST	最初の値を序数セットに戻します。
LAST	最後の値を序数セットに戻します。
NEW	構造体タイプのためにストレージを獲得し、獲得されたストレージにハンドルを戻します。
RESPEC	式のビット・パターンを変更せずに、式の属性を指定のタイプに変更します。
SIZE	タイプを表すために必要なストレージの量を戻します。

BIND

BIND は、ポインター *p* を構造体のタイプ *t* のハンドルに変換します。BIND は、タイプ付き構造体のメンバーのロケーターとして使用できます。

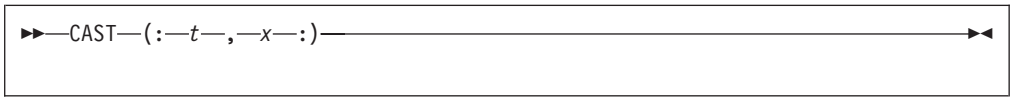


t 構造体タイプの名前

p ポインターの式

CAST

CAST は、C の変換規則を使用して、式 *x* をタイプ *t* に変換します。



t 「C タイプ」のスカラーの名前

x 同じく「C タイプ」のスカラー式

サポートされる「C タイプ」は次のとおりです。

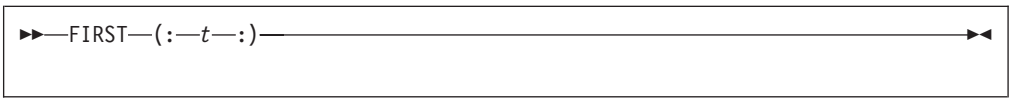
- REAL FIXED BIN(p,0)
- REAL FIXED DEC(p,q) ただし p >= q かつ q>= 0
- NATIVE FLOAT
- ORDINAL
- POINTER または HANDLE
- LIMITED ENTRY

x が FLOAT または FIXED DEC の場合、 *t* は FLOAT、FIXED、または ORDINAL でなければならず、 *t* が FLOAT または FIXED DEC の場合、 *x* は FLOAT、FIXED、または ORDINAL でなければなりません。

必要な変換はすべて、ANSI C 規則に従って行われます。つまり、たとえば CAST によって SIZE は発生せず、また負の値が UNSIGNED にキャストされた場合、結果は大きな正数になります。

FIRST

FIRST は、序数セット *t* の最初の値を戻します。



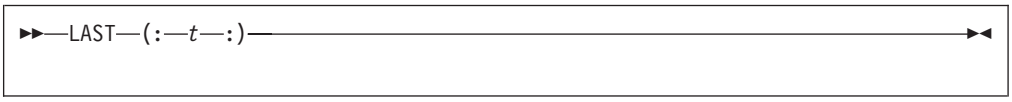
t 序数タイプの名前

例

```
define ordinal Color ( Red,
                       Orange,
                       Yellow,
                       Green,
                       Blue,
                       Indigo,
                       Violet );

display (ordinalname( first(:Color:) ));  /* RED */
```

LAST

LAST は、序数セット *t* の最後の値を戻します。



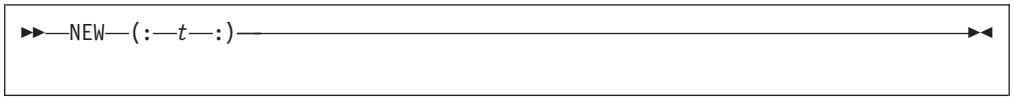
t 序数タイプの名前

例

```
define ordinal Color ( Red,
                       Orange,
                       Yellow,
                       Green,
                       Blue,
                       Indigo,
                       Violet );

display (ordinalname( last(:Color:) )); /* VIOLET */
```

NEW

NEW は、構造体タイプ *t* のヒープ・ストレージを獲得し、獲得したストレージにハンドルを戻します。

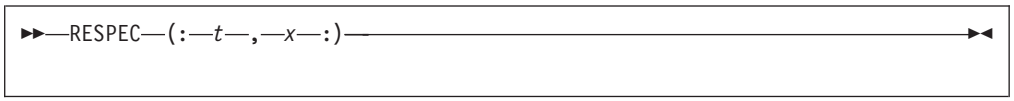


t 構造体タイプの名前

NEW(:t:) は、 BIND(: t, ALLOC(SIZE(:t:)) :) と等価です。

RESPEC

RESPEC は、式のビット値を変更せずに、式 *x* の属性をタイプ *t* に変更します。



t スカラー型の名前

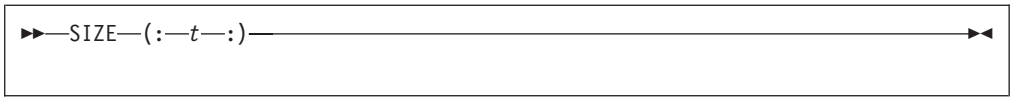
p スカラー式

x は *t* と同じでなければならず、 *x* または *t* のどちらかが UNALIGNED BIT である場合は、両方とも UNALIGNED BIT にする必要があります (この場合、関数は何も行わないので意味のないものになります)。

たとえば、*t* が属性 LIMITED ENTRY を持つタイプである場合、 RESPEC(t, sysnull()) は「ヌル」関数ポインターを戻します。

SIZE

SIZE は、タイプ *t* で宣言された変数に必要なストレージ必要量を戻します。



t 構造体タイプまたは共用体タイプの名前

第 21 章 プリプロセッサーの機能

プリプロセッサー・オプション 536
プリプロセッサー走査 537
　プリプロセッサー・ステートメント 538
プリプロセッサー変数とデータ・エレメント . . . 539
プリプロセッサー参照とプリプロセッサー式 . . . 540
プリプロセッサー名の有効範囲 541
プリプロセッサー・プロシージャー 541
　プリプロセッサー・プロシージャーの引数とパラメーター 542
　%PROCEDURE ステートメント 543
　プリプロセッサー RETURN ステートメント . . 544
　プリプロセッサー ANSWER ステートメント 545
プリプロセッサー組み込み関数 547
　COLLATE 548
　COMMENT 548
　COMPILEDATE. 548
　COMPILETIME 549
　COPY 549
　COUNTER 550
　DIMENSION 550
　HBOUND 550
　INDEX 551
　LBOUND 551
　LENGTH 552
　LOWERCASE 552
　MACCOL 552
　MACLMAR 552
　MACNAME 553
　MACRMAR 553
　MAX 553
　MIN 553
　PARMSET 554
　QUOTE 554
　REPEAT 554
　SUBSTR 555
　SYSPARM 555
　SYSTEM 556
　SYSVERSION 556
　TRANSLATE 556
　TRIM 557
　UPPERCASE. 557
　VERIFY 557
プリプロセッサー・ステートメント 558
　%ACTIVATE ステートメント 558
　%代入ステートメント 559
　%DEACTIVATE ステートメント 559
　%DECLARE ステートメント 560
　%DO ステートメント 562
　%END ステートメント 562
　%GO TO ステートメント 563
　%IF ステートメント 563
　%INCLUDE ステートメント 564
　%INSCAN ステートメント 565
　%ITERATE ステートメント 566
　%LEAVE ステートメント 566
　%NOTE ステートメント. 567
　%ヌル・ステートメント 567
　%REPLACE ステートメント 568
　%SELECT ステートメント 568
　%XINCLUDE ステートメント 568
　%XINSCAN ステートメント 569
プリプロセッサーの例 569
　例 1 569

コンパイラーには、ソース・プログラムを変更できるように、MACRO プリプロセッサーが備わっています。 MACRO あるいは PP(MACRO) コンパイル時オプションを指定すれば、コンパイルの前にプリプロセッサーが実行されます。 MACRO プリプロセッサーは、プリプロセッサー入力を走査し、プリプロセッサー出力を生成します。プリプロセッサー出力は、コンパイラーへの入力として利用することができます。

この章では、読者が本書全体にわたって説明されている PL/I 言語をすでに理解していることを前提に、プリプロセッサーを説明します。

プリプロセッサー入力 は、文字ストリングです。 下記のものが混在しています。

- *プリプロセッサー・ステートメント*。[1] プリプロセッサー・ステートメントは、プリプロセッサー走査によって検出されたときに実行されます (ただし、プリプ

1. 説明を明確にするために、本章では、プリプロセッサー・ステートメントにはすべて % 記号が付いています (実際には、プリプロセッサー・プロシージャー内で使用するときは、% 記号を付けません)。

ロセッサー・プロシージャーは例外であり、これは呼び出されてから実行されます)。プリプロセッサー・ステートメント (プリプロセッサー・プロシージャー内にある場合を除く) の頭にはパーセント記号 (%) を付けます。ブランクを使用して、パーセント記号とそのステートメントの残りの部分とを区切ることもできます。

プリプロセッサーは、プリプロセッサー・ステートメントを実行し、それに応じて入力テキストを変更します。プリプロセッサー・ステートメントを使用すれば、下記のいずれかの方法で入力テキストの一部を変更することができます。

- 入力テキスト内の任意の ID (およびオプションの引数リスト) をテキストの任意のストリングに変更することができます。
- 入力テキストのどの部分を、プリプロセッサー出力の中にコピーするかを指定することができます。
- ライブラリーに収められている文字のストリングを、プリプロセッサー入力の中に組み込むことができます。

- *リスト制御*ステートメント。プログラムの印刷リストのレイアウトを制御するためのものです。これらのステートメントは、インソース・リスト (プリプロセッサー入力) とソース・リスト (プリプロセッサー出力) の両方に影響を与えます。これらについては、201 ページの『第 9 章 ステートメントとディレクティブ』で説明しています。
- 入力テキスト。プリプロセッサー・ステートメントでもなくリスト制御ステートメントでもないプリプロセッサー入力です。入力テキストは PL/I 原始プログラムでもその他のテキストでも構いませんが、その入力テキストは、プリプロセッサー走査による入力テキストの処理 (下記を参照) に矛盾するものであってはなりません。

プリプロセッサー出力 [2] は、文字のストリングで、下記のものが混在しています。

- *リスト制御*ステートメント。 プリプロセッサー入力に含まれていて、走査されたリスト制御ステートメントが、プリプロセッサー出力の中にコピーされます。
- *出力*テキスト。 走査され、おそらく変更された入力テキストが、プリプロセッサー出力に組み入れられます。

コンパイル時オプションを指定することによって、プリプロセッサー入力を印刷したり、プリプロセッサー出力または入出力を印刷したり、データ・セットに書き出したりすることができます。

プリプロセッサー・オプション

MACRO または PP(MACRO) コンパイル時オプションを指定すると、プリプロセッサーが呼び出されます。

また、プリプロセッサーだけに影響するコンパイラー・オプションを指定することもできます。オプションによっては、プリプロセッサーの振る舞いを大きく変更するものもあります。特に重要なオプションは次のものです。

2. プリプロセッサーによって置き換えられた出力は、定様式で示してあります。実際に実行したときに生成される置き換え後の出力は定様式になっていません。

FIXED
FIXED 変数の処理方法を指定します。このオプションには、次の 2 つのサブオプションがあります。

BINARY
FIXED 変数を BINARY として処理するように指定します。

DECIMAL
FIXED 変数を DECIMAL として処理するように指定します。

CASE
入力テキストを大文字に変換するかどうかを指定します。このオプションには、次の 2 つのサブオプションがあります。

ASIS
入力テキストを「現状のまま」にするように指定します。

UPPER
入力テキストを大文字に変換するように指定します。

RESCAN
このオプションは、プリプロセッサーがテキストを再スキャンするときに、ID の大/小文字を扱う方法を指定します。このオプションには、次の 2 つのサブオプションがあります。

ASIS
再スキャンは大文字と小文字を区別します。

UPPER
再スキャンは大文字と小文字を区別しません。

DBCS
このオプションは、プリプロセッサーが、テキストの置換時に DBCS を正規化するかどうかを指定します。このオプションには、次の 2 つのサブオプションがあります。

EXACT
入力テキストは「現状のまま」残され、プリプロセッサーは <kk.B> と <kk>B を異なる名前として扱います。

INEXACT
入力テキストは「正規化」され、プリプロセッサーは <kk.B> と <kk>B を同じ名前の 2 つのバージョンとして扱います。

これらのオプションのデフォルトは、
FIXED(DECIMAL)、CASE(UPPER)、RESCAN(ASIS) および DBCS(INEXACT) です。

これらのオプションの指定方法について詳しくは、「プログラミング・ガイド」を参照してください。

プリプロセッサー走査

プリプロセッサーは、プリプロセッサー入力の始まりから走査を開始し、 1 文字ずつ順に走査します。

デフォルトでは CASE(UPPER) オプションが有効で、プリプロセッサーは、入力に含まれている小文字 (コメントおよびストリング定数内の小文字を除く) を大文字に変換します。ただし、CASE(ASIS) サブオプションが有効になっている場合、テキストは現状のままになります。

プリプロセッサー・ステートメント

プリプロセッサー・ステートメントは検出されたときに実行されます。以下の処理を行うことができます。

- %DECLARE ステートメントを使用して、およびラベル接頭語として書くことによって、プリプロセッサーを定義する。

  プリプロセッサー変数を明示的に宣言しなければ、診断メッセージが出され、変数にはデフォルト属性 CHARACTER が与えられます。ただし、それ以後に実行される %ACTIVATE ステートメント内にその変数を書かない限り、その変数は置き換えのために活動化されません。変数は、プリプロセッサー・ステートメント内で参照できます。
- %DECLARE または %ACTIVATE ステートメントを使用して ID を活動化することにより、置き換え処理を開始させる。これについては、『入力テキスト』の項で説明します。
- %DEACTIVATE ステートメントを使用して ID を非活動化することにより、置き換え処理を終了させる。
- %NOTE ステートメントを使用してコンパイラー・リスト内にメッセージを生成する。
- 文字ストリングをプリプロセッサー入力に入れる。
- %GOTO、%IF、%NULL、%DO、または %END ステートメントを使用してプリプロセッサー入力内の別の場所からプリプロセッサー走査を続行させる。
- % 割り当て、または %DO ステートメントを使用してプリプロセッサー変数の値を変更する。
- %PROCEDURE、%RETURN、および %END ステートメントを使用してプリプロセッサー・プロシージャーを定義する。プリプロセッサー式の中の関数参照によって、プリプロセッサー・プロシージャーを呼び出すことができます。あるいは、関数プロシージャー名がアクティブであれば、プリプロセッサーによる入力テキストの走査中に関数参照が検出されると、プリプロセッサー・プロシージャーが呼び出されます。

リスト制御ステートメント

プリプロセッサー・プロシージャーに含まれていないリスト制御ステートメントは、プリプロセッサー出力の中にコピーされます (各ステートメントごとに 1 行ずつ使われます)。

入力テキスト

アクティブ ID が新しい値で書き換えられた後の入力テキストが、プリプロセッサー出力の中にコピーされます。無効文字 (文字定数またはコメントの一部) は、プリプロセッサー出力内ではブランクで置き換えられます。置き換えを行うかどうかを判別するために、入力テキストが走査されて、下記のものが探索されます。

- この PL/I 文字セットの一部ではない文字。これらは区切り文字と見なされ、変更されずにそのままプリプロセッサー出力の中にコピーされます。
- PL/I 文字定数や PL/I コメント。これらは変更されずに、プリプロセッサーによって入力テキストからプリプロセッサー出力に移されます。活動プリプロセッサー・プロシージャーへの引数リスト内に書かれている場合は、この限りではありません。ただし、この場合、入力マージンと出力マージンが異なっているときには、ストリングやコメントが数行にまたがっていると、入力行数と出力行数が一致しないことがあります。出力は常に F フォーマットで行われ、そのマージンは 2 桁目と 72 桁目に決まっているので、特に入力が V フォーマットであれば、上記のことが起こります。このような場合、出力行の番号付けからも、入力行数と出力行数が一致していないことが分かります。
- アクティブ ID。新しい値で置き換えられる必要のある ID は、まず*活動化* しておかなければ置き換えられません。最初に、ID を %DECLARE ステートメントに挿入することで、その ID を活動化することができます。 %DEACTIVATE ステートメントを実行すれば、その ID を非活動化することができます。その後、%ACTIVATE ステートメントまたは %DECLARE ステートメントを実行すれば、その ID を再び活動化することができます。

  アクティブなプリプロセッサー変数の名前に一致する ID は、プリプロセッサー出力内ではその変数の値で置き換えられます。

  ID が、アクティブなプリプロセッサー関数 (プログラマー作成の関数でも組み込み関数でもよい) の名前に一致したときは、そのプロシージャーが呼び出されて、その呼び出しが、戻り値で置き換えられます。

  ID を活動化するとき、RESCAN オプションと NORESCAN オプションのどちらかを指定することができます。 NORESCAN オプションが適用されていると、値がただちにプリプロセッサー出力の中に挿入されます。 RESCAN オプションが適用されていると、再走査が行われて、その値がテストされ、その値 (または値の一部) を別の値で書き換えることができるかどうかが判別されます。別の値で置き換えられないときは、値がプリプロセッサー出力の中に挿入されます。別の値で置き換えることができるときは、置き換えをそれ以上できなくなるまで置き換え処理が続けられます。このようにして、置き換え可能なものをすべて置き換えたあとにだけ、プリプロセッサー出力への値の挿入が行われます。

置き換わる値には、% 記号、対になっていない引用符、対になっていないコメント区切り文字などが含まれていてはなりません。

プリプロセッサー入力内の最後の文字を超えて走査が試みられたときに、走査が終了します。その時点で、プリプロセッサー出力が完成しているので、コンパイルを開始することができます。

プリプロセッサー変数とデータ・エレメント

プリプロセッサー変数は、%DECLARE ステートメントで FIXED 属性または CHARACTER 属性を付けて指定します。他の属性をプリプロセッサー変数に対して宣言することはできません。また、属性を繰り返してはなりません。(ただし、プリプロセッサーでは他の属性を提供しています。) すべての変数は、STATIC ストレージ・クラスと同等のストレージを持ちます。

プリプロセッサー・データのタイプは、コード化算術データとストリング・データであり、次のいずれかです。

FIXED
FIXED 属性を指定して宣言したプリプロセッサー変数には、デフォルトで属性 DECIMAL(5,0) が与えられます。

FIXED(BINARY) が有効になっている場合、変数には属性 BINARY(31,0) が与えられます。

小数値はサポートされていません。

CHARACTER
CHARACTER 属性を付けて宣言したプリプロセッサー変数には、VARYING 属性が与えられます。

また、プリプロセッサーは、X 文字ストリング定数をサポートします。

文字定数に、ストリングの反復因数を使用することはできません。ただし、COPY 組み込み関数を使用して定数を複製できます。

BIT
プリプロセッサー・ビット変数はありませんが、ビット定数を使用することはでき、比較演算子、連結演算子 (ビット・オペランドとともに使用するとき)、*NOT* 演算子、および PARMSET 組み込み関数が実行されるとビット値が作られます。 %IF ステートメント内のプリプロセッサー式は、ビット値に変換されます。

プリプロセッサー参照とプリプロセッサー式

『第 4 章 式および参照』で説明したのと同じ方法で、プリプロセッサー参照をコーディングできますし、プリプロセッサー式が計算されます。ただし、以下の点に注意してください。

- プリプロセッサー式のオペランドとして指定できるのは、プリプロセッサー変数、プリプロセッサー・プロシージャー参照、固定小数点 10 進定数、ビット定数、文字定数、およびプリプロセッサー組み込み関数参照だけです。
- 配列をプリプロセッサー・プロシージャーの外部で宣言する (複数のプロシージャー間で共有できるように) ことはできますが、プロシージャーの外部から参照することはできません (配列照会組み込み関数への最初の引数として参照する場合を除き)。
- 累乗演算記号 (**) は使えません。
- FIXED(DECIMAL) オプションを指定した場合
  - 算術演算の場合、精度 (5,0) の 10 進数演算だけが行われます。つまり、各オペランドが精度 (5,0) の 10 進固定小数点整数値に変換された後で演算が行われ、10 進固定小数点の演算結果も精度 (5,0) に変換されます。たとえば、3/5 という式は、0.6 ではなく 0 と評価されます。

    算術値に変換される文字値は、整数 (符号は省略可能) の形をしていなければなりません。ヌル・ストリングは 0 に変換されます。
  - 固定小数点値がビット値に変換されると、常に、長さが CEIL(3.32*5)、つまり 17 のストリングになります。

  - 固定小数点値を文字値に変換すると、結果は常に長さ 8 のストリングになり、PL/I プログラム内で FIXED DEC(5,0) 値を CHARACTER に変換した結果と同じ値が得られます。
- FIXED(BINARY) オプションを指定した場合
  - 算術演算の場合、精度 (31,0) の 2 進演算だけが行われます。つまり、各オペランドが精度 (31,0) の 2 進固定小数点整数値に変換された後で演算が行われ、2 進固定小数点の演算結果も精度 (31,0) に変換されます。たとえば、3/5 という式は、0.6 ではなく 0 と評価されます。

    算術値に変換される文字値は、整数 (符号は省略可能) の形をしていなければなりません。ヌル・ストリングは 0 に変換されます。
  - 固定小数点値がビット値に変換されると、常に、長さが 31 のストリングになります。
  - 固定小数点値が文字値に変換されると、先行ブランクが削除されるので、結果は可変長のストリングになります。

プリプロセッサー名の有効範囲

プリプロセッサー名の有効範囲は、その名前がどこで宣言されているかによって決まります。プリプロセッサー・プロシージャー内で宣言されている名前の有効範囲は、そのプロシージャーです。組み込まれたストリング内で宣言されている名前の有効範囲は、そのストリングと、そのストリングが組み込まれた後で走査されたすべての入力テキスト (ただし、同じ名前の宣言を含んでいるプリプロセッサー・プロシージャーは除く) です。その他の名前の有効範囲は、プリプロセッサー入力全体です (ただし、同じ名前の宣言を含んでいるプリプロセッサー・プロシージャーは除きます)。

プリプロセッサー・プロシージャー

プリプロセッサー・プロシージャーは、%PROCEDURE ステートメントと %END ステートメントで区切られます。プロシージャーが RETURNS 属性を指定して定義されていない場合、プロシージャーに ANSWER ステートメントがなくても構いませんが、 RETURN ステートメントがあってはなりません。逆に、プロシージャーが関数である場合は、プロシージャーに少なくとも 1 つの RETURN ステートメントが必要で、ANSWER ステートメントがあってはなりません。

プリプロセッサー・プロシージャー内で使用できるステートメントおよびグループは、次のとおりです。

- プリプロセッサー ANSWER ステートメント
- プリプロセッサー代入ステートメント
- プリプロセッサー DECLARE ステートメント
- プリプロセッサー DO グループ
- プリプロセッサー GO TO ステートメント (プリプロセッサー・プロシージャー内の GO TO ステートメントは、そのプロシージャーの外へ制御権を移動することはできません。)
- プリプロセッサー IF ステートメント

- プリプロセッサー ITERATE ステートメント
- プリプロセッサー LEAVE ステートメント
- プリプロセッサー・ヌル・ステートメント
- プリプロセッサー NOTE ステートメント
- プリプロセッサー REPLACE ステートメント
- プリプロセッサー RETURN ステートメント
- プリプロセッサー SELECT グループ
- %PAGE、%SKIP、%PRINT、および %NOPRINT リスト制御ステートメント

プリプロセッサー・プロシージャー内のプリプロセッサー・ステートメントの頭には、% 記号を付けません。

プリプロセッサー・プロシージャーをネストすることはできません。プリプロセッサー・プロシージャー内にプリプロセッサー ENTRY 宣言があってはなりません。

プリプロセッサー・プロシージャーの入り口名と、そのプロシージャーへ渡される引数リストをまとめて、*関数参照* といいます。プリプロセッサー式の中の関数参照によって、プリプロセッサー・プロシージャーを呼び出すことができます。あるいは、関数プロシージャー名がアクティブであれば、プリプロセッサーによる入力テキストの走査中に関数参照が検出されると、プリプロセッサー・プロシージャーが呼び出されます。プリプロセッサー・プロシージャーの入り口名を %DECLARE ステートメントで指定する必要はありません。

入り口名がアクティブであれば、そのプリプロセッサー・プロシージャーは呼び出される前に走査される必要はありません。ただし、そのプリプロセッサー・プロシージャーは下記のどちらかの中になければなりません。
- プリプロセッサー入力
- 呼び出し点の前に挿入されていたストリング

プリプロセッサー・プロシージャーがプリプロセッサー入力に組み込まれる前に、そのプロシージャーの参照が検出された場合、どのような結果になるかは分かりません。

プリプロセッサー出力では、関数参照とその引数リストが、プリプロセッサー関数から返された値 (つまり、RETURN ステートメント内のプリプロセッサー式の値) で書き換えられます。

プリプロセッサー・プロシージャーの引数とパラメーター

プロシージャー参照内の引数の個数と %PROCEDURE ステートメント内のパラメーターの個数とが同じである必要はありません。引数は、パラメーター・リストとの突き合わせが行われる前に計算されます。定位置引数の個数がパラメーターの個数よりも多い場合には、右側にある余分の引数が無視されます。 (引数が関数参照であるときは、その引数が最終的には無視されるものであっても、その関数が呼び出されて実行されます。) 関数参照によってセットされなかったパラメーターには、ゼロの値 (FIXED パラメーターの場合) またはヌル・ストリング (CHARACTER パラメーターの場合) が指定されます。

パラメーターを、関数参照によって複数回設定しないでください。ただし、あるパラメーターの値を複数回指定した (たとえば、定位置とキーワードの両方で) ときは、エラーと診断されますが、左端に設定した値が呼び出しで使われます。

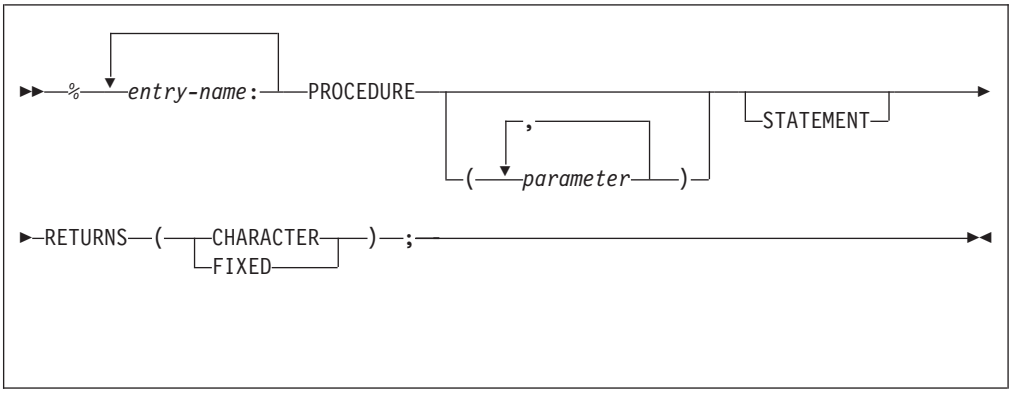
関数参照がプリプロセッサー・ステートメント内にあるときは、通常の方法で引数がパラメーターに関係付けられます。 116 ページの『プロシージャーへの引数の引き渡し』の項で説明したのと同じ方法で、仮引数が作られることがありますし、引数は対応するパラメーターの属性に変換されます。

関数参照が入力テキスト内に存在すると、仮引数が常に作成されます。引数は、ストリングとして解釈されます。コンマまたは右括弧を区切り文字として使用してください。ただし、対になっている括弧、対になっている単一引用符、または対になっているコメント区切り文字の間にあるコンマや右括弧は、区切り文字の働きをしません。たとえば、定位置引数リスト (A(B,C),D) の引数は、ストリング A(B,C) とストリング D の 2 つです。引数内のブランク (先行ブランクや末尾ブランクも) は意味がありますが、そのようなブランクが行の終わりまで続いており、しかも引用符やコメント区切り文字で囲まれていない場合は、1 つのブランクで置き換えられます。

入力テキスト内で関数参照が検出されると、各引数が走査され、置き換えが可能かどうか調べられます。この置換アクティビティーは、その関数に渡された引数の個数に影響しません。置き換え処理によって引数内にコンマや括弧が入れられても、それらは区切り文字とは見なされず、単に引数内の文字と見なされます。キーワード呼び出しが使われている場合、そのキーワード自体は置換アクティビティーに適しません。置き換えがすべて終わると、その結果の各引数が、その関数入り口名を持つプリプロセッサー・プロシージャー・ステートメント内の対応するパラメーターの属性で指定されるタイプに変換されます。

%PROCEDURE ステートメント

%PROCEDURE ステートメントは、%END ステートメントと対になっており、プリプロセッサー・プロシージャーを区切る働きをします。 %PROCEDURE ステートメントの構文は次のとおりです。



省略形: %PROC

parameter
関数プロシージャーのパラメーターを指定します。

STATEMENT

関数参照が入力テキスト内にあり、しかもこの STATEMENT オプションが指定されているときは、次のとおりです。

- 引数を定位置引数リスト内に指定することも、キーワード参照によって指定することもできます。
- 参照の終わりは、セミコロンで示します。置き換えが行われると、このセミコロンは保持されません。

たとえば、次のステートメントで始まるプリプロセッサー・プロシージャーがあるとします。

```
%FIND:PROC(A,B,C) STATEMENT...;
```

これをプリプロセッサー式から呼び出すときは、関数参照の形式は次のとおりでなければなりません。

```
FIND(arg1,arg2,arg3)
```

関数参照が入力テキスト内にあるときは、下記のどの形式 (または類似の形式) で参照しても、このプロシージャーを呼び出すことができ、すべて同じ結果になります。

```
FIND(X,Y,Z);

FIND B(Y) C(Z) A(X);

FIND(X) C(Z) B(Y);

FIND(,Y,Z) A(X);
```

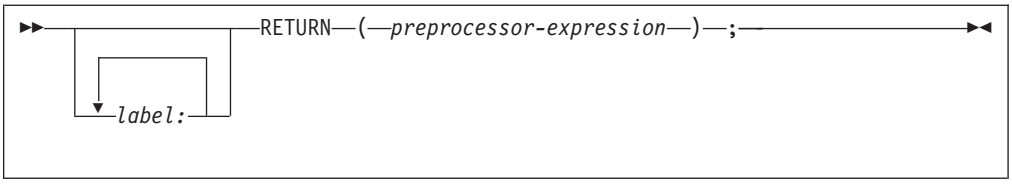
RETURNS

CHARACTER または FIXED 属性を RETURNS 属性リスト内に指定することにより、この関数プロシージャーから返される値の属性を示さなければなりません。

プリプロセッサー RETURN ステートメント

プリプロセッサー RETURN ステートメントは、プリプロセッサー・プロシージャー内だけで、プロシージャーに RETURNS 属性が指定されている場合に限って使用できます。したがって、このステートメントの頭には % 記号を付けてはなりません。このステートメントは、このプリプロセッサー・プロシージャーの呼び出し点に値と制御を返します。 RETURNS 属性を持つそれぞれのプリプロセッサー・プロシージャー内には、 1 つ以上の RETURN ステートメントがなければなりません。

プリプロセッサー関数プロシージャーから呼び出し点に返される値は、そのプロシージャー内の RETURN ステートメント内のプリプロセッサー式 (preprocessor-expression) で指定された値です。プリプロセッサー RETURN ステートメントの構文は、次のとおりです。

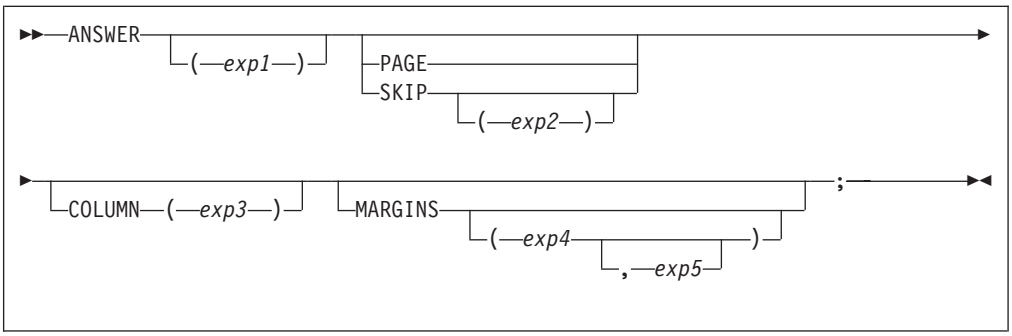


preprocessor-expression
この値は、%PROCEDURE ステートメント内に指定されている RETURNS 属性に合わせて変換されてから、呼び出し点に返されます。

プリプロセッサー ANSWER ステートメント

プリプロセッサー ANSWER ステートメントは、 RETURNS 属性のないプリプロセッサー・プロシージャー内でだけ使用できます。

ANSWER ステートメントは、テキストを作成するか、またはほかのプリプロセッサー・プロシージャーを呼び出します。あるいは、その両方を行います。応答されたテキストは、ソース・テキスト内のプリプロセッサー・プロシージャーの呼び出しを置換します。 ANSWER ステートメントは 1 つのプリプロセッサー・プロシージャー内で何回でも使うことができます。



省略形 : ANSWER の場合は ANS、COLUMN の場合は COL、 MARGINS の場合は MAR です。

exp1
ANSWER テキストを表す文字式を表します。 ANSWER テキストは、単一文字ストリング定数か、または合成されたプリプロセッサー式のいずれかにすることができます。

テキストが式の場合、式の計算は通常の方法で行われ、結果は単一文字ストリングに変換されます。

SCAN または RESCAN が有効な場合には、置換とプリプロセッサー・プロシージャー呼び出しのために、その文字ストリングが走査されます。この置換はそのプリプロセッサー・プロシージャーの範囲内で行われ、応答されたテキストが戻される範囲内で行われることは**ありません**。それから、応答されたテキストがソース・テキスト内のプリプロセッサー呼び出しの点に挿入されます。テキストがソース・テキストに戻されたあとは、いかなる置換アクティビティーが行われても、そのテキストが走査されることはありません。

ストリング内の置換アクティビティーは、ソース・テキストの走査と置換のときに用いられるのと同じ規則に従って行われます。 『例』を参照してください。

PAGE
%PAGE ディレクティブを生成することによって、応答テキストを出力ソースの新規のページに強制的に入れるようにします。

SKIP
応答テキストを出力ソースの新規の行に強制的に入れるようにします。 *exp2* は、スキップされる行数を指定している算術式を表します。 *exp2* を省略した場合、デフォルトは 1 です。

COLUMN
ソース・プログラム行に応答テキストが入れられるときの開始カラムを指定します。 *exp3* の値は、ソース・プログラム行内の応答テキストの開始カラム番号を示す算術式を表します。

MARGINS
出力テキストが出力レコード内に置かれる場所を指定します。 *exp4* の値は、出力テキストの左マージンを示す算術式を表します。 *exp5* の値は、出力テキストの右マージンを示す算術式を表します。

exp5 に指定される値は、 MACLMAR (左マージン) 組み込み関数と MACRMAR (右マージン) 組み込み関数によって戻される範囲内になければなりません。

ANSWER ステートメントの MARGINS オプションを指定しない場合の、デフォルトの値は MARGINS(MACLMAR,MACRMAR) です。オペランドを指定しないで MARGINS オプションを指定する場合は、デフォルトの値は MARGINS(MACCOL,MACRMAR) です。

同じプリプロセッサー・プロシージャーの中で、式を指定した RETURN ステートメントと ANSWER ステートメントの両方を一緒に使うことはできません。

例

```
%dcl (Expression, Single_string) entry;
%dcl (Deactivated_macro, Statement_function) entry;
%dcl Deactivated_variable character;
%deact Deactivated_variable, Deactivated_macro;
%Deactivated_variable = '** value of deactivated variable **';

%Deactivated_macro: procedure returns( character );
   return( '** value of deactivated macro **' );
%end;

%Statement_function: procedure( key1 ) stmt returns( fixed );
   dcl key1 fixed;
   return( key1 + key1 );
%end;

%Expression: procedure;
   ANS( Counter ) skip;
   ANS( Deactivated_macro ) skip;
   ANS( Deactivated_variable ) skip;
   /* The following is invalid:                                  */
   /* ANS( Statement_function Key1(7););                         */
%end;

%Single_string: procedure;
```

```
   ANS( 'Counter' ) skip;
   ANS( 'Deactivated_macro' ) skip;
   ANS( 'Deactivated_variable' ) skip;
   ANS( 'Statement_function Key1( 7 );' ) skip;
%end;


Expression              /* Generates:                            */
                        /*   00001                               */
                        /*   ** value of deactivated macro **    */
                        /*   ** value of deactivated variable ** */

Single_string           /* Generates:                            */
                        /*   Counter                             */
                        /*   Deactivated_macro                   */
                        /*   Deactivated_variable                */
                        /*         14                            */
```

プリプロセッサー組み込み関数

プリプロセッサー組み込み関数 と呼ばれる 1 組の事前定義関数の 1 つを、関数参照で呼び出すことができます。これらの組み込み関数の呼び出し方法は、プログラマー定義の関数の呼び出し方法と同じですが、正しい数の引数を指定しなければならない点だけが異なります。

プリプロセッサー組み込み関数は、次のとおりです。

COLLATE	INDEX	MAX	SYSTEM
COMMENT	LBOUND	MIN	SYSVERSION
COMPILEDATE	LENGTH	PARMSET	TRANSLATE
COMPILETIME	LOWERCASE	QUOTE	TRIM
COPY	MACCOL	REPEAT	UPPERCASE
COUNTER	MACLMAR	SUBSTR	VERIFY
DIMENSION	MACNAME	SYSPARM	
HBOUND	MACRMAR		

プリプロセッサーは、入力テキスト内のプリプロセッサー組み込み関数参照を、その組み込み関数名がアクティブであるときにだけ、実行します。 %DECLARE ステートメントまたは %ACTIVATE ステートメントを使用すれば、組み込み関数を活動化することができます。

プリプロセッサー・ステートメント内のプリプロセッサー組み込み関数名は、別の意味を持つように宣言されていない限り、組み込み関数として常にアクティブになっています。

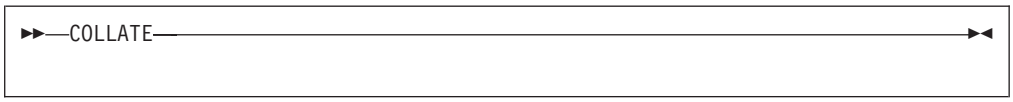
プリプロセッサー組み込み関数名をユーザー定義のプリプロセッサー・プロシージャーの名前として使用している場合は、その名前を参照すると、組み込み関数ではなくそのプロシージャーを参照することになります。そのような場合に、あるプリプロセッサー・プロシージャー内でその組み込み関数を使用するときは、BUILTIN 属性を指定してその ID を宣言しておかなければなりません。

次のプリプロセッサー組み込み関数の場合、引数は不要であり、ヌル引数を指定してはなりません。

COLLATE	COUNTER	MACNAME	SYSTEM
COMPILEDATE	MACCOL	MACRMAR	SYSVERSION
COMPILETIME	MACLMAR	SYSPARM	

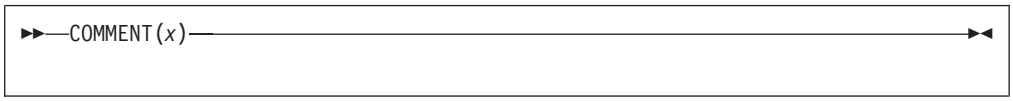
COLLATE

COLLATE は、あり得る 256 の文字値を照合順序で 1 つずつ含む、長さ 256 の CHARACTER ストリングを戻します。



COMMENT

COMMENT は、CHARACTER 式をコメントに変換します。



x コメントに変換される式。

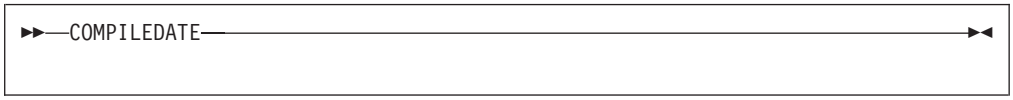
x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

x は /* と */ で囲まれます。

x に /* または */ の複合記号が入っている場合には、それぞれ、/> と </ によって置き換えられます。

COMPILEDATE

COMPILEDATE は、コンパイル日時が入った長さ 17 の CHARACTER ストリングを戻します。



COMPILEDATE から戻されるストリングのフォーマットは、次のとおりです。

yyyy 現在の年
mm 現在の月
dd 現在の日
hh 現在の時
mm 現在の分
ss 現在の秒
ttt 現在のミリ秒

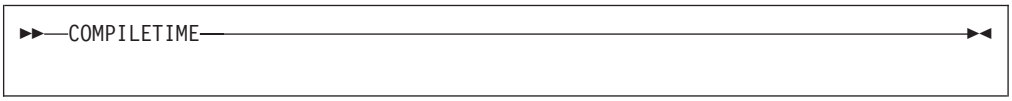
時間帯と精度は、システムよって決まります。

次の例は、プログラムの実行中に、COMPILEDATE から返されたストリングを印刷する方法を示しています。

```
%DECLARE COMP_DATE CHAR;
%COMP_DATE=QUOTE(COMPILEDATE);
PUT EDIT (COMP_DATE) (A);
```

COMPILETIME

COMPILETIME は、コンパイル日時が入った長さ 18 の CHARACTER ストリングを戻します。



COMPILETIME から戻されるストリングのフォーマットは、次のとおりです。

DD 日
. ピリオド
MMM 月 (JAN、FEB、MAR など)
. ピリオド
YY 年
b ブランク
HH 時
. ピリオド
MM 分
. ピリオド
SS 秒

「日」の先行ゼロはブランクで置き換えられます。その他の先行ゼロは消去されません。

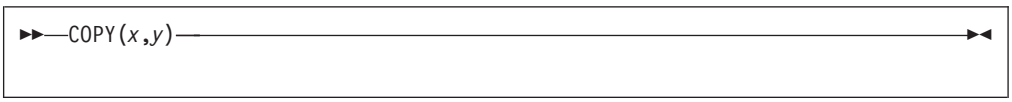
時刻機構が使用可能でない場合は、返されるストリングの最後の 8 文字は 00.00.00 となります。

次の例は、プログラムの実行中に、COMPILETIME から返されたストリングを印刷する方法を示しています。

```
%DECLARE COMP_TIME CHAR;
%COMP_TIME=QUOTE(COMPILETIME);
PUT EDIT (COMP_TIME) (A);
```

COPY

COPY は、ストリング *x* のコピーを *y* 回連結して構成される CHARACTER ストリングを戻します。



x 式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

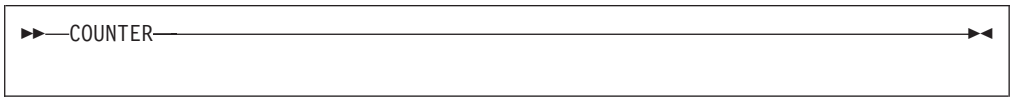
y 繰り返し回数を指定する式。 *y* のタイプは FIXED でなければならず、それ以外の場合はこのタイプに変換されます。

y は、負以外でなければなりません。

y がゼロの場合は、結果はヌル・ストリングです。

COUNTER

COUNTER は、10 進数が入った長さ 5 の CHARACTER ストリングです。返される値は、最初の呼び出しの場合は 00001 で、その後呼び出されるたびに 1 ずつ増えます。

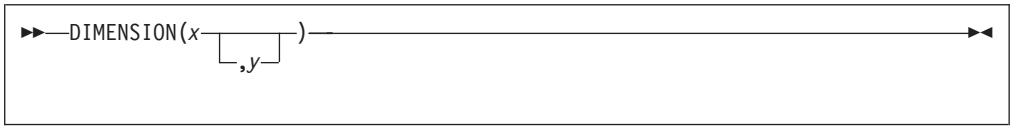


COUNTER が 99999 回呼び出されると、次回の呼び出し時に診断メッセージが出され、00000 が返されます。その直後の呼び出しは、最初の呼び出しとして扱われます。

COUNTER 組み込み関数は、固有の名前を生成したり、数を数えたりするときに使用することができます。

DIMENSION

DIMENSION は、*x* の次元 *y* の現行エクステントを指定する FIXED 値を戻します。



省略形: DIM

x 配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

y *x* の特定の次元を指定する式。

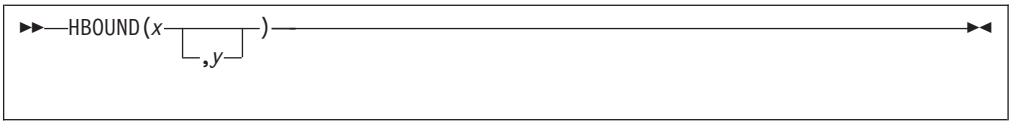
y のタイプは FIXED でなければならず、それ以外の場合はこのタイプに変換されます。

y は、1 以上でなければなりません。*y* が提供されない場合は、デフォルトを 1 にします。

配列が 1 次の場合のみ、*y* を省略できます。

HBOUND

HBOUND は、*x* の次元 *y* の現行の上限を FIXED 値で戻します。



x 配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

y *x* の特定の次元を指定する式。

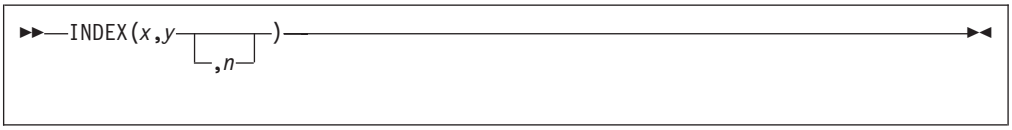
y のタイプは FIXED でなければならず、それ以外の場合はこのタイプに変換されます。

y は、1 以上でなければなりません。*y* が提供されない場合は、デフォルトを 1 にします。

配列が 1 次の場合のみ、*y* を省略できます。

INDEX

INDEX は、*y* と等しいサブストリングの *x* 内の開始位置を示す FIXED 値を戻します。また、*x* の処理を開始する位置を指定することもできます。



x 検索される式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

y 検索のターゲット式。

y のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

n *n* は、処理を開始する *x* 内の位置を指定します。

n のタイプは FIXED でなければならず、それ以外の場合はこのタイプに変換されます。

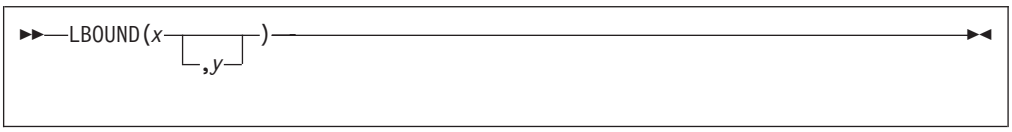
x 内で *y* が発生しない場合、または *x* か *y* のどちらかが長さゼロがある場合は、値ゼロが戻されます。

n は 0 より大きく、かつ 1 + LENGTH(*x*) を超えないことが必要です。

n = LENGTH(*x*) + 1 の場合は、結果はゼロです。

LBOUND

LBOUND は、*x* の次元 *y* の現行の下限を FIXED 値で戻します。



x 配列参照。 *x* の次元数は、*y* よりも小さくてはなりません。

y *x* の特定の次元を指定する式。

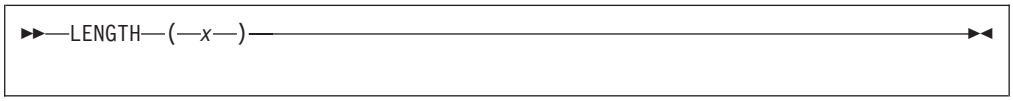
y のタイプは FIXED でなければならず、それ以外の場合はこのタイプに変換されます。

y は、1 以上でなければなりません。*y* が提供されない場合は、デフォルトを 1 にします。

配列が 1 次の場合のみ、*y* を省略できます。

LENGTH

LENGTH は、指定の文字式 x の現在の長さを表す FIXED 値を戻します。



x 式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

LOWERCASE

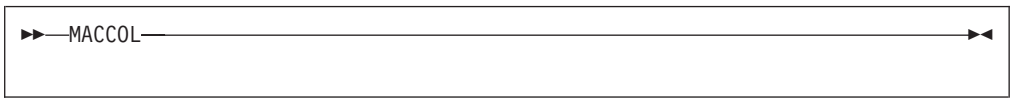
LOWERCASE は、A から Z の英字すべてを同等の小文字に変換した CHAR ストリングを戻します。

x 式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

MACCOL

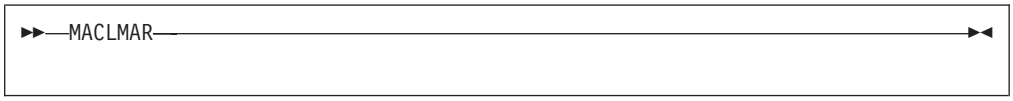
MACCOL は、最外部のマクロ呼び出しが入っているソース・テキスト内で、そのマクロ呼び出しが始まっているカラムを表す FIXED 値を戻します。



戻される値は、ネストされたマクロ呼び出しによって影響されません。

MACLMAR

MACLMAR は、MARGINS コンパイラー・オプションにおける左ソース・マージンのカラム番号を表す FIXED 値を戻します。



「*プログラミング・ガイド*」の MARGINS オプションの説明を参照してください。

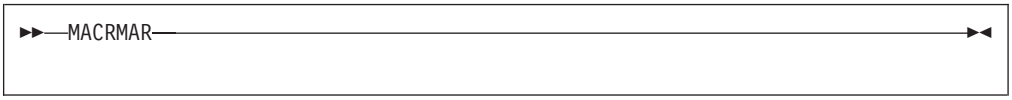
MACNAME

MACNAME は、呼び出し元のプリプロセッサー・プロシージャーの名前を戻します。

プリプロセッサー・プロシージャーの外部からの MACNAME の呼び出しは無効です。

MACRMAR

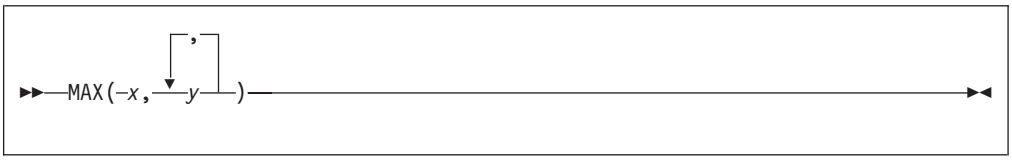
MACRMAR は、MARGINS コンパイラー・オプションの右ソース・マージンのカラム番号を表す FIXED 値を戻します。



「*プログラミング・ガイド*」の MARGINS オプションの説明を参照してください。

MAX

MAX は、複数の式の集合から最大値を戻します。

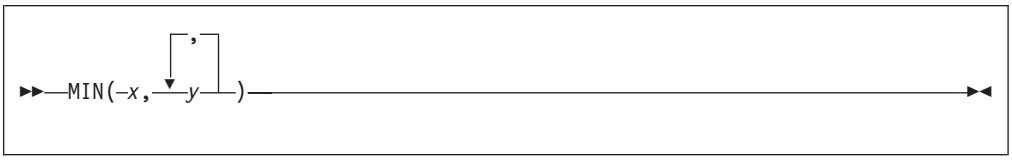


x および y
式。

引数はすべて FIXED でなければならず、FIXED でないものはすべて FIXED に変換されます。

MIN

MIN は、複数の式の集合から、最小値を戻します。

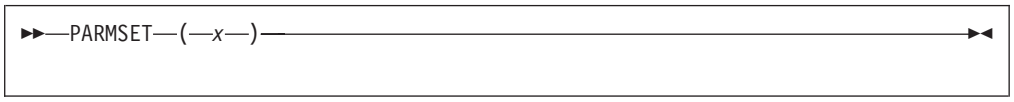


x および y
式。

引数はすべて FIXED でなければならず、FIXED でないものはすべて FIXED に変換されます。

PARMSET

PARMSET は、指定のパラメーターがプロシージャーの呼び出し時に設定されたかどうかを示す BIT 値を戻します。



x そのプリプロセッサー・プロシージャーのパラメーターでなければなりません。

PARMSET 組み込み関数は、プリプロセッサー・プロシージャー内でだけ使用することができます。

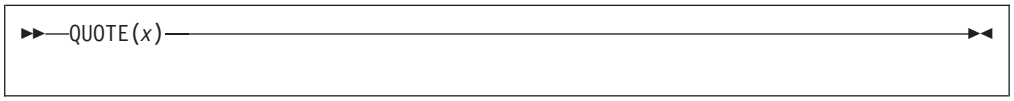
PARMSET から返される値は、そのプロシージャーを呼び出した関数参照によってパラメーター x が明示的にセットされている場合はビット値 '1'B であり、セットされていない (つまり、プリプロセッサー式の中の関数参照で対応する引数が指定されなかったか、または入力テキスト内の関数参照で対応する引数がヌル・ストリングであった) 場合は、ビット値 '0'B です。

対応する引数が関数参照内に指定されていても、その関数参照が別のプリプロセッサー・プロシージャー内にある場合は、PARMSET から '0'B が返されることがあります。次のようになります。

- その引数そのものが呼び出しプロシージャーのパラメーターでないときは、値 '1'B が返されます。
- その引数が呼び出しプロシージャーのパラメーターであるときは、呼び出しプロシージャーそのものが呼び出されたときの指定されたパラメーターについての値が返されます。

QUOTE

QUOTE は、*x* を有効な引用符付きストリングとして表す CHARACTER ストリングを戻します。



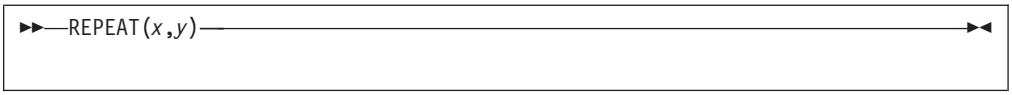
x 引用符付きストリングに変換される式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

x に一重引用符が入っている場合には、各引用符が、それぞれ 2 つずつ連続する一重引用符に置き換えられます。

REPEAT

REPEAT は、ストリング *x* のコピーを (*y* + 1) 回連結して構成される CHARACTER ストリングを戻します。



x 式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

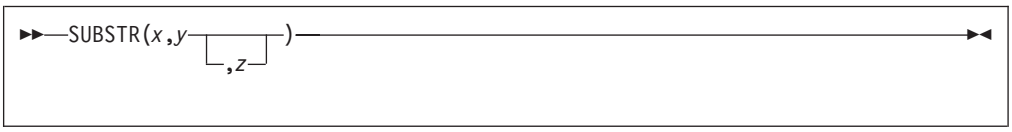
y 繰り返し回数を指定する式。 *y* のタイプは FIXED でなければならず、それ以外の場合はこのタイプに変換されます。

y は、負以外でなければなりません。

y がゼロの場合は、結果は *x* です (必要に応じ、文字に変換される)。

SUBSTR

SUBSTR は、サブストリングを戻します。このサブストリングは、*x* の *y* および *z* によって指定されます。



x サブストリングが抜き出されるストリングを指定する式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

y *x* の中のサブストリングの開始位置を指定する式。

y のタイプは FIXED でなければならず、それ以外の場合はこのタイプに変換されます。

z *x* の中のサブストリングの長さを指定する式。

z のタイプは FIXED でなければならず、それ以外の場合はこのタイプに変換されます。

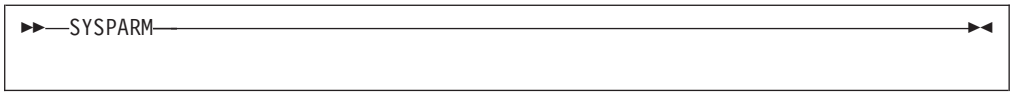
z がゼロの場合は、ヌル・ストリングが戻されます。 *z* が省略される場合は、戻されるサブストリングは *x* の位置 *y* から *x* の終わりまでです。

z は負でなければならず、*y* および *z* の値は、サブストリング全体が *x* の現在の長さに入るような値でなければなりません。

y = LENGTH(*x*)+1 かつ *z* = 0 の場合は、ヌル・ストリングが戻されます。

SYSPARM

SYSPARM は、SYSPARM コンパイラー・オプションの CHARACTER ストリング値を戻します。

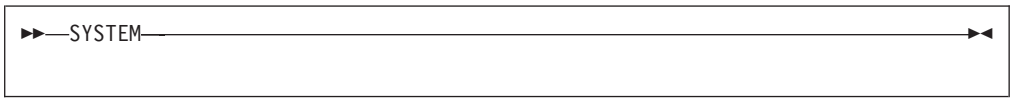


戻された値は大文字に変換されることはありません。コンパイラー・オプションに指定されたとおりの値が戻されます。詳細については、「*プログラミング・ガイド*」の「SYSPARM コンパイラー・オプション」の説明を参照してください。

SYSPARM を使用すると、プログラムの外部情報はソース・プログラムを変更しないでアクセスできます。

SYSTEM

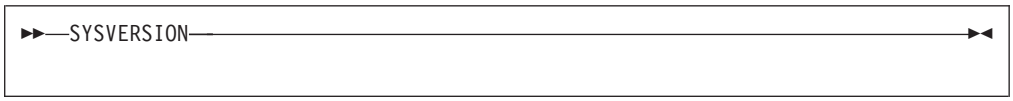
SYSTEM は、有効な SYSTEM コンパイラー・オプションの値が入っている CHARACTER ストリングを戻します。



詳細については、「*プログラミング・ガイド*」の「SYSTEM コンパイラー・オプション」の説明を参照してください。

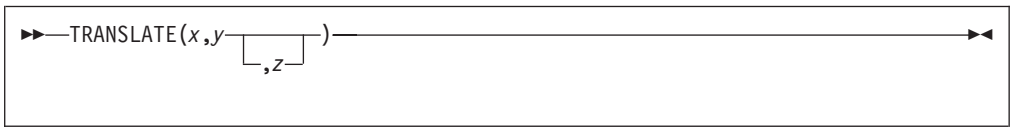
SYSVERSION

SYSVERSION は、製品名、バージョン、リリース、およびモディフィケーション・レベルを含む CHARACTER ストリングを戻します。



TRANSLATE

TRANSLATE は、選択した文字を変換した、*x* と同じ長さの CHARACTER ストリングを戻します。



x 変換できる文字を検索する対象を指定する式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

y 文字の変換値を含む式。

y のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

z 変換される文字を含む式。 *z* が省略された場合のデフォルトは COLLATE です。

z のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

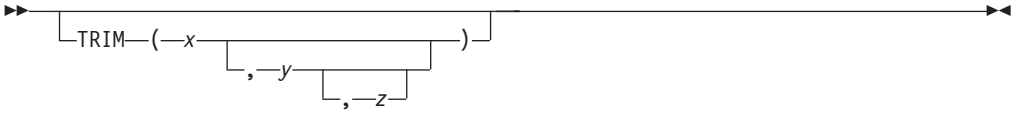
TRANSLATE は、*x* の各文字を次のように処理します。

x の中にある文字が *z* に検出される場合は、*y* にある *z* に対応する文字が結果にコピーされます。それ以外の場合は、 *x* の文字が結果にコピーされます。 *z* に複数の同一文字が入っている場合は、左端のオカレンスが使われます。

y は、*z* の長さと一致するように、右がブランクで埋められるか、切り捨てられます。

TRIM

TRIM は、入力ストリングの一方の端または両端から文字を切り取った CHAR ストリングを戻します。



UPPERCASE

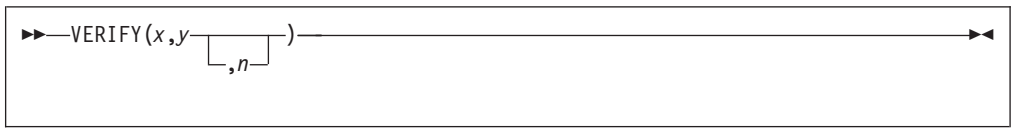
UPPERCASE は、a から z の英字すべてを同等の大文字に変換した CHAR ストリングを戻します。

x 式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

VERIFY

VERIFY は、*y* にない左端の文字の *x* 内での位置を示す FIXED 値を戻します。この関数により、処理を開始する *x* 内の位置を指定できます。



x 式。

x のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

y 式。

y のタイプは CHARACTER でなければならず、それ以外の場合はこのタイプに変換されます。

n *n* は、処理を開始する *x* 内の位置を指定します。

n のタイプは FIXED でなければならず、それ以外の場合はこのタイプに変換されます。

x のすべての文字が *y* にある場合は、値ゼロが戻されます。 *x* がヌル・ストリングの場合は、値ゼロが戻されます。 *x* がヌル・ストリングでなく、*y* がヌル・ストリングの場合は、 *n* の値が戻されます。 *n* のデフォルトは 1 です。

n は 0 より大きく、かつ 1 + LENGTH(*x*) を超えないことが必要です。

n = LENGTH(*x*) + 1 の場合は、結果はゼロです。

プリプロセッサー・ステートメント

ここでは、プリプロセッサー・ステートメントをアルファベット順に取り上げて説明します。

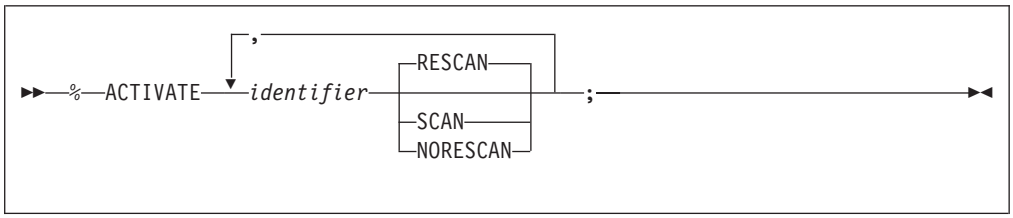
プリプロセッサー・ステートメント内でブランクを使用できるところであればどこにでも、コメントを記入できます。このようなコメントは、プリプロセッサー出力テキストには挿入されません。

すべてのプリプロセッサー・ステートメントにラベルを付けることができます。

%CONTROL ステートメントはサポートされません。このキーワードを記述してもエラーにはなりませんが、無視されます。

%ACTIVATE ステートメント

%ACTIVATE ステートメントは、ID をアクティブにして、それに置換適格性を与えます。それ以後、その ID がアクティブになっている間は、入力テキスト内でその ID が検出されると、置き換え処置が開始されます。



省略形: %ACT

identifier
プリプロセッサー変数名、プリプロセッサー・プロシージャー名、またはプリプロセッサー組み込み関数を指定します。

配列変数を参照する ID は指定できません。

RESCAN
結果が出力に入れられる前に、すべての活動 ID を置換するのに必要な回数だけこの ID が置換されることを指定します。

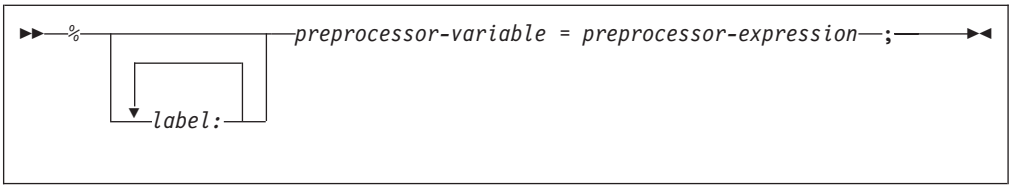
SCAN
結果が出力に入れられる前に、この ID が 1 回だけ置換されることを指定します。

NORESCAN
SCAN の同義語です。

すでにアクティブの ID に対して %ACTIVATE を使用しても特に影響はありませんが、走査状況が変わる可能性はあります。

%代入ステートメント

% 代入ステートメントは、プリプロセッサー式を評価して、その結果をプリプロセッサー変数に割り当てます。

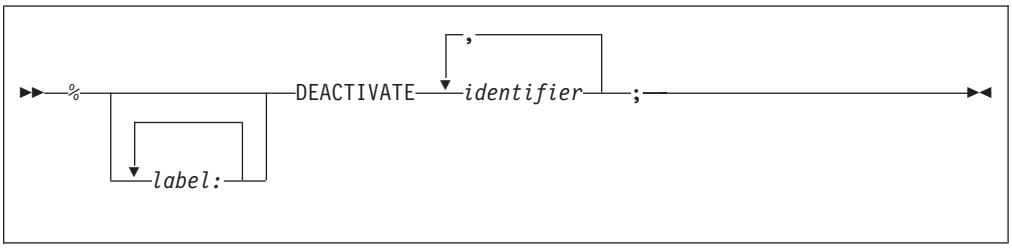


複合割り当てと多重割り当ては許可されていません。

配列を割り当てのターゲットにすることはできませんが、配列エレメントをターゲットにすることはできます。

%DEACTIVATE ステートメント

%DEACTIVATE ステートメントは、ID を非活動化します。



省略形: %DEACT

identifier
プリプロセッサー変数名、プリプロセッサー・プロシージャー名、またはプリプロセッサー組み込み関数名を指定します。

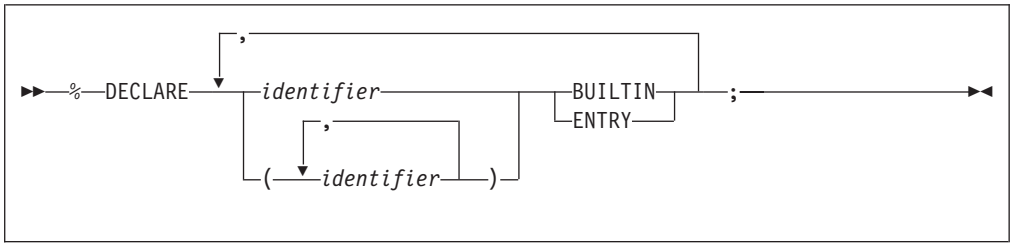
ID を非活動化すると、その ID は置換適格性を失いますが、その値は失われません。したがって、その ID を再活動化したとき、置き換え値を割り当てる必要はありません。

ID を非活動化しても、後続のプリプロセッサー・ステートメント内でその ID に別の値を割り当てることができます。

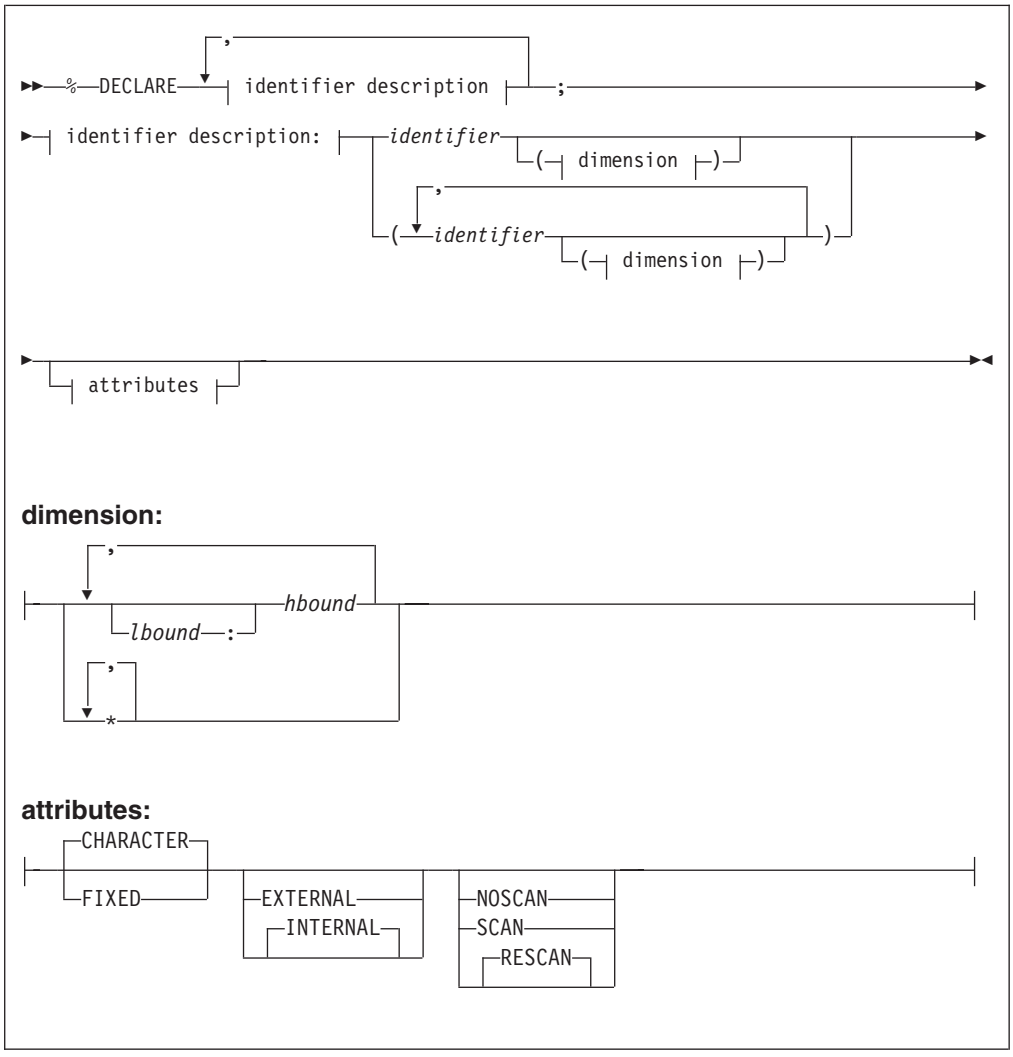
非活動化された ID の非活動化は無効です。

%DECLARE ステートメント

%DECLARE ステートメントは、ID をマクロ変数、マクロ・プロシージャー、または組み込み関数として設定します。また、マクロ変数の走査状況を指定することができます。



または



省略形: %DECLARE の場合は %DCL、 CHARACTER の場合は CHAR、INTERNAL の場合は INT、EXTERNAL の場合は EXT です。

identifier description
マクロ機能 ID の名前と属性を指定します。

BUILTIN
この ID が同じ名前を持つプリプロセッサー組み込み関数であることを指定します。

CHARACTER
この ID が、最大の長さを持たない可変長文字ストリングを表していることを指定します。

ENTRY
この ID がプリプロセッサー・プロシージャーであることを指定します。

宣言により、入り口名が活動化されます。

プリプロセッサー・プロシージャーの入り口名を %PROCEDURE ステートメントのラベルとして書けば、その入り口名を明示的に宣言することができます。ただし、このように明示宣言しただけでは、そのプリプロセッサー・プロシージャー名は活動化されません。

FIXED
ID が整数を表すことを指定します。

(デフォルトの) FIXED(DECIMAL) オプションを指定した場合、 ID には属性 DECIMAL(5,0) も与えられます。

FIXED(BINARY) オプションを指定した場合、 ID には属性 BINARY(31,0) も与えられます。

RESCAN
この ID がアクティブであり、必要な回数だけ出力内で置換されることを指定します。

SCAN
この ID がアクティブであり、1 回だけ出力内で置換されることを指定します。

NOSCAN
この ID が非アクティブであり、出力内で置換されないことを指定します。

dimension
配列変数の次元指定です。 15 を超える次元は指定できません。

注: 配列をプリプロセッサー・プロシージャーの外部で宣言する (複数のプロシージャー間で共有できるように) ことはできますが、プロシージャーの外部から参照することはできません (配列照会組み込み関数への最初の引数として参照する場合を除き)。

lbound
この次元にとって必要な下限。デフォルトは、1 です。

hbound
この次元にとって必要な上限。

INTERNAL
この属性は、プロシージャーの内部でだけ有効です。プロシージャーの外部で指定された場合、診断メッセージが出され、変数には EXTERNAL 属性が与えられます。

プロシージャーの外部で宣言された変数はすべて EXTERNAL であり、プロシージャーの内部で宣言された変数はすべて INTERNAL です。

EXTERNAL
この属性は、プロシージャーの外部でだけ有効です。プロシージャーの内部で指定された場合、診断メッセージが出され、変数には INTERNAL 属性が与えられます。

%DO ステートメント

%DO ステートメントは、%END ステートメントと対になっており、プリプロセッサー DO グループを区切る働きをします。また DO グループの反復実行を指定することもできます。

%DO ステートメントの構文は、209 ページの『DO ステートメント』で説明しています。

注: DO ステートメントのフォーマットは、次のものを除いてすべてサポートされます。

- UPTHRU と DOWNTHRU は受け入れられません。
- タイプ 3 DO ステートメントの 『specification』 は、複数回指定できません。

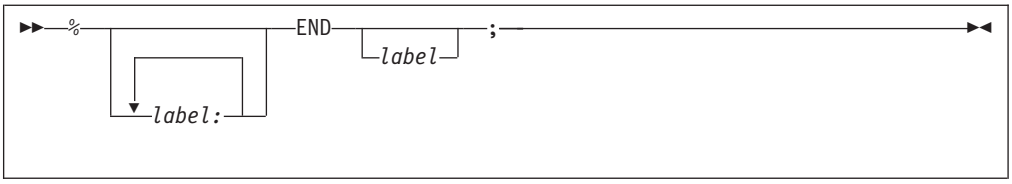
プリプロセッサー DO グループはネストさせることができます。

タイプ 3 のプリプロセッサー DO グループの中に制御権を移動することはできません。ただし、その DO グループ内から呼び出されたプリプロセッサー・プロシージャーから戻ることはできます。

プリプロセッサー・ステートメント、入力テキスト、およびリスト制御ステートメントを、プリプロセッサーの DO グループ内に書くことができます。プリプロセッサー・ステートメントは実行され、入力テキストは走査されて、可能であれば置き換えられます。

%END ステートメント

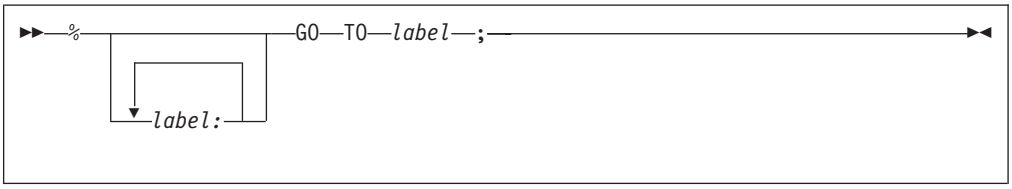
%END ステートメントは、%DO、%SELECT、または %PROCEDURE ステートメントと一緒に使用され、プリプロセッサー DO グループまたはプリプロセッサー・プロシージャーを区切る働きをします。



END の後ろのラベル (label) は、%PROCEDURE、%DO、または %SELECT ステートメントのラベルでなければなりません。複数の閉止も可能です。

%GO TO ステートメント

%GO TO ステートメントを使用すると、プリプロセッサーは指定されたラベルの場所から走査を続行します。



省略形: %GOTO

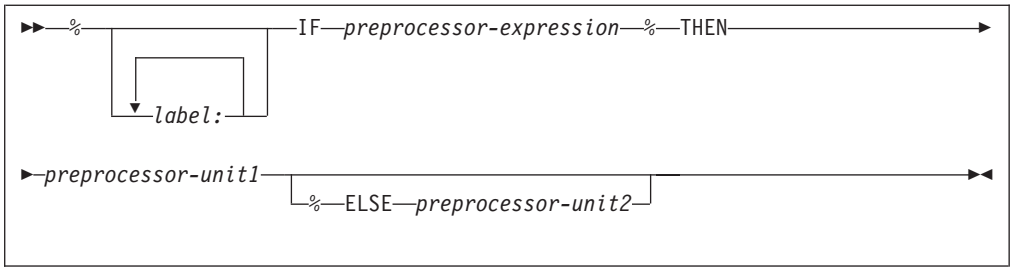
GO TO のあとに続くラベルは、走査の移動先を指定します。ここに指定するラベルは、プリプロセッサー・ステートメントのラベルでなければなりませんが、プリプロセッサー・プロシージャーのラベルであってはなりません。

プリプロセッサー・プロシージャー内のプリプロセッサー GO TO ステートメントは、そのプロシージャーの外へ制御権を移動することはできません。つまり、GO TO の後ろに指定するラベルは、そのプロシージャーに含まれているラベルでなければなりません。

組み込まれたストリングと一緒に %GO TO を使用する場合の制限事項については、後述の 『%INCLUDE ステートメント』の項を参照してください。

%IF ステートメント

%IF ステートメントは、プリプロセッサー式のビット値に従って、走査の流れを制御します。



preprocessor-expression
この式が計算されて、ビット・ストリングに変換されます (変換できないときはエラーになります)。

preprocessor-unit
単一のプリプロセッサー・ステートメント (%DECLARE、%PROCEDURE、%END、%DO 以外のもの)、プリプロセッサー DO グループ、またはプリプロセッサー SELECT グループ。それ以外の場合、記述は、 223 ページの『IF ステートメント』で指定されている記述と同じです。

ストリング内のいずれかのビットの値が '1'B であれば、unit1 が実行され、 unit2 (ある場合) は無視されます。すべてのビットが '0'B であれば、 unit1 が無視され、unit2 (ある場合) が実行されます。

unit1 または unit2 のどちらかで %GO TO ステートメントまたはプリプロセッサー RETURN ステートメントを指定したことによって、走査が別の場所から再開する場合を除けば、%IF ステートメントのすぐ後ろから走査が再開します。

%IF ステートメントは、 223 ページの『IF ステートメント』の項で説明したのと同じ方法でネストさせることができます。

%INCLUDE ステートメント

%INCLUDE が実行されると、そのステートメントで指定された外部テキストが、プリプロセッサー入力の中に、%INCLUDE ステートメントの代わりとして組み込まれます。そのようなテキストがいったん組み込まれると、それは*組み込み済み* テキストと呼ばれます。そのテキストは、プリプロセッサー・ステートメント、リスト制御ステートメント、および PL/I 原始ステートメントで構成することができます。

%INCLUDE ステートメントの構文は、 225 ページの『%INCLUDE ディレクティブ』で説明しています。

dataset と *member name* の各ペアは、ソース・プログラムに組み込まれる外部テキストを識別します。

走査は、包含されるテキストの最初の文字から続行されます。包含されるテキストは、プリプロセッサー入力と同じ方法で走査されます。ですから、包含されるテキストを、形成するプリプロセッサー出力の基にすることができます。

%INCLUDE ステートメントはネストすることができます。 つまり、組み込まれたテキストの中に、%INCLUDE ステートメントがあってもかまいません。

組み込み済みテキスト内の %GO TO ステートメントが制御権を移動することができるのは、同じ組み込みファイルの内部の点に限られています。 %GOTO ステートメントよりも、その %GOTO ステートメントのターゲット・ラベルが先に来てはなりません。

組み込み済みテキスト内のプリプロセッサー・ステートメント、DO グループ、SELECT グループ、およびプロシージャーは完結していなければなりません。たとえば、%IF ステートメントの半分が組み込みテキスト内にあり、残りの半分がプリプロセッサー入力の別の部分にあるということは許されません。

プリプロセッサー入力と組み込み済みテキスト内に、%INCLUDE 以外にはプリプロセッサー・ステートメントが含まれていない場合は、プリプロセッサーの実行を省略することができます。 (その場合、INCLUDE コンパイル時オプションを使用する必要があります。)

たとえば、データ・セット SYSLIB のメンバー PAYRL に下記のテキスト (構造体宣言) が収められているとします。

```
DECLARE 1 PAYROLL,
          2 NAME,
            3 LAST   CHARACTER (30) VARYING,
            3 FIRST  CHARACTER (15) VARYING,
            3 MIDDLE CHARACTER (3) VARYING,
          2 CURR,
            3 (REGLAR, OVERTIME) FIXED DECIMAL (8,2),
          2 YTD LIKE CURR;
```

このとき、下記のプリプロセッサー・ステートメントが実行されると、

```
%DECLARE PAYROLL CHARACTER;
%PAYROLL='CUM_PAY';
%INCLUDE PAYRL;
%DEACTIVATE PAYROLL;
%INCLUDE PAYRL;
```

2 つの構造体宣言がプリプロセッサー出力テキスト内に生成されます。それら 2 つの構造体宣言は名前だけが異なっており、それぞれ CUM_PAY と PAYROLL という名前です。

最初の %INCLUDE ステートメントが実行されると、PAYRL 内のテキストがプリプロセッサー入力の中に組み込まれます。この組み込み済みテキスト内の ID PAYROLL がプリプロセッサー走査で検出されると、ID PAYROLL はアクティブのプリプロセッサー変数 PAYROLL の現行値 (つまり CUM_PAY) で置き換えられます。それ以上の包含されたテキストを走査しても、追加の置換は行われません。次に、%DEACTIVATE ステートメントがプリプロセッサー走査で検出され、プリプロセッサー変数 PAYROLL が非活動化されます。 2 番目の %INCLUDE ステートメントが実行されると、PAYRL 内のストリングがもう一度プリプロセッサー入力の中に組み込まれます。しかし、今回は、この組み込み済みテキストが走査されても置き換えはまったく行われません。

%INSCAN ステートメント

%INCLUDE ステートメントと同様に、%INSCAN ステートメントもファイルを組み込みます (ただし、%INSCAN ステートメント内は除きます)。組み込まれるファイルは、プリプロセッサー変数で指定します。

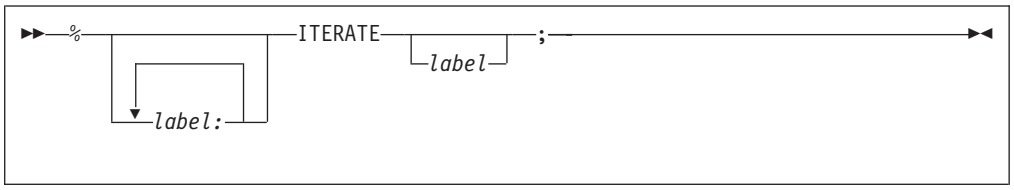
filename
組み込むファイルの名前を指定するプリプロセッサー式。

```
%dcl inname char;
%inname = 'oldform';
%inscan inname; /* includes the file "oldform" */
```

%ITERATE ステートメント

%ITERATE ステートメントは、このステートメントを含む反復 DO グループを区切る %END ステートメントに制御を移します。現在行われている反復は完了され、次の反復が (必要であれば) 開始されます。

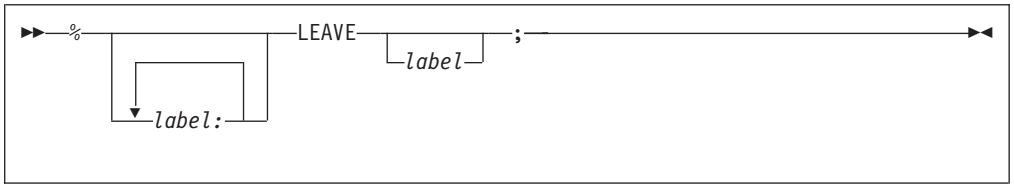
ITERATE ステートメントは、反復 DO グループが非反復 DO グループを含んでいる場合には、その非反復 DO グループ内で指定することができます。



label-constant
このステートメントを含んでいる DO グループのラベルでなければなりません。省略した場合、ITERATE ステートメントを含む最新の反復 DO グループの END ステートメントに制御が移ります。

%LEAVE ステートメント

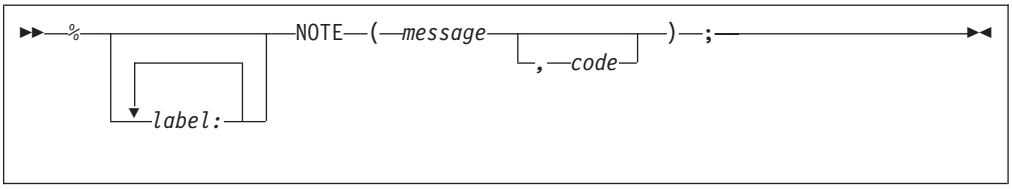
単純な DO グループに含まれているか、または単純な DO グループを指定してある場合に、 %LEAVE ステートメントは、そのグループを終了します。反復 DO グループに含まれているか、またはそのようなグループを指定してある場合には、%LEAVE ステートメントは、現在の繰り返しも含め、そのグループのすべての繰り返しを終了します。制御の流れは、DO グループが END ステートメントに達して終了した場合に通常移動するはずの位置に移動します。



label-constant
このステートメントを含んでいる DO グループのラベルでなければなりません。ここで指定するラベルを持つ DO グループが終了します。 *label-constant* が省略されると、 LEAVE ステートメントを含むグループが終了する DO グループです。

%NOTE ステートメント

%NOTE ステートメントは、指定されたテキストと重大度コードを示すプリプロセッサー診断メッセージを生成します。



message
診断メッセージを必要とする値を持つ文字式。

code
メッセージの重大度を表す値が入っている固定式。重大度は次のとおりです。

コード	重大度
0	I
4	W
8	E
12	S
16	U

code を省略すると、デフォルトとして 0 が設定されます。

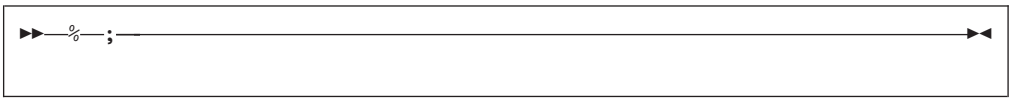
code の値が上記以外の値のときは、診断メッセージが生成され、デフォルトの値が使われます。デフォルトの値は、*code* の値が 0 より小さいか 16 より大きいときは重大度 U であり、その他の場合は、次に低い重大度がデフォルトとなります。

生成されたメッセージは、他のプリプロセッサー・メッセージと一緒にファイルされます。特定のメッセージを続いて印刷するかどうかは、その重大度レベルとコンパイラー FLAG オプションの設定に依存します。

重大度 U のメッセージが生成されると、プリプロセスとコンパイル処理が直ちに終了します。重大度 S、E、または W のメッセージが生成されたときにコンパイル処理が終了するかどうかは、NOSYNTAX および NOCOMPILE コンパイル時オプションがどのようにセットされているかによって異なります。

%ヌル・ステートメント

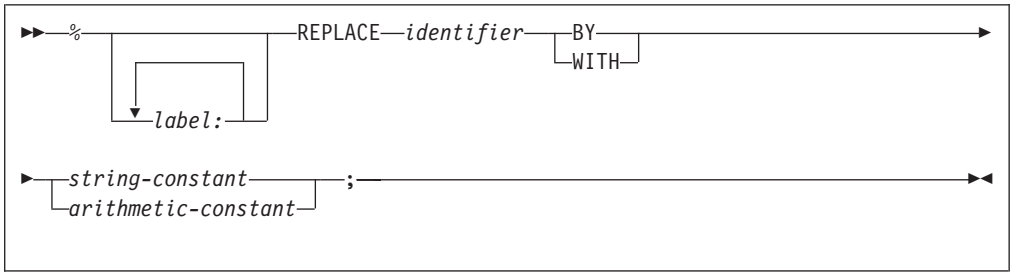
% ヌル・ステートメントは、何も行いません。またステートメントの実行順序を変えることもしません。



注: %PROCEDURE ステートメントおよび RETURN ステートメントの説明については、本章の始めの方に記載されています。

%REPLACE ステートメント

%REPLACE ステートメントを使用すると、名前をストリング定数か数値定数に直ちに置き換えることができます。名前に値を割り当てるために、その名前を変数として宣言する必要はありません。



identifier
置き換えられる名前。

string-constant
名前が宣言されていない場合、CHARACTER 属性が与えられます。

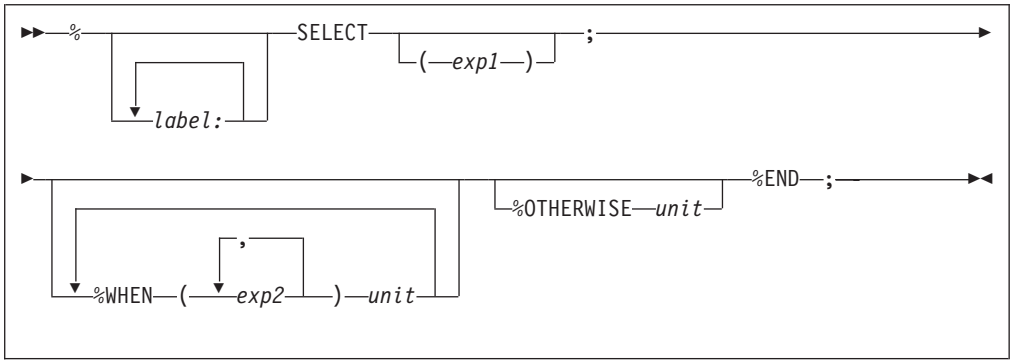
arithmetic-constant
名前が宣言されていない場合、FIXED 属性が与えられます。

FIXED(DEC) オプションが指定されている場合、値は FIXED DEC(5,0) に変換されます。

FIXED(BIN) オプションが指定されている場合、値は FIXED BIN(31,0) に変換されます。

%SELECT ステートメント

%SELECT ステートメントは、%END ステートメントと対になっており、プリプロセッサー SELECT グループを区切る働きをします。



%XINCLUDE ステートメント

%XINCLUDE ステートメントは %INCLUDE ステートメントと同じですが、ファイルがすでに存在している場合はファイルを組み込みません。

%XINCLUDE ステートメントの構文は、236 ページの『%XINCLUDE ステートメント』で説明しています。

%XINSCAN ステートメント

%XINSCAN ステートメントは %INSCAN ステートメントと同じですが、ファイルがすでに存在している場合はファイルを組み込みません。

プリプロセッサーの例

例 1

プリプロセッサー入力に、次のストリングが含まれているとします。

```
%DECLARE A CHARACTER, B FIXED;
%A = 'B+C';
%B = 2;
X = A;
```

すると、次のステートメントがプリプロセッサー出力に挿入されます。

```
X = 2+C;
```

これらのプリプロセッサー・ステートメントは、デフォルトの値である RESCAN を指定して A と B を活動化し、文字ストリング 'B+C' を A に割り当て、さらに定数 2 を B に割り当てます。

4 行目は入力テキストです。 A の現行値、つまり 'B+C' が A の代わりとしてプリプロセッサー出力の中に挿入されます。ただし、このストリングにはプリプロセッサー変数 B が含まれているので、プリプロセッサーは B を再走査し、B が活動化されていることを見つけます。ですから、プリプロセッサー出力内の B は値 2 に置き換わります。プリプロセッサー変数 B の精度は、デフォルトの精度 (5,0) なので、実際には 2 の前に 0 が 4 つ付いています。ストリング 'B+C' 内の B がこの値で置き換えられるときに、この値は文字ストリングに変換されるので、2 の前にブランクが 7 つ付きます。

さらに再走査が行われ、2 をこれ以上置き換えられないことがわかると、 +C から走査が再開されます。これも置き換えることはできません。

上記の例で、プリプロセッサー変数 A が次のステートメントによって活動化されたとします。

```
%ACTIVATE A NORESCAN;
```

すると、プリプロセッサー出力は次のようになります。

```
X = B+C;
```

例 2

プリプロセッサー入力に、次のストリングが含まれているとします。

```
%DECLARE I FIXED, T CHARACTER;
%DEACTIVATE I;
%I = 15;
%T = 'A(I)';
S  = I*T*3;
```

```
%I = I+5;
%ACTIVATE I;
%DEACTIVATE T;
R  = I*T*2
```

すると、プリプロセッサー出力は次のようになります (置き換えのブランクは示していません)。

```
S = I*A(I)*3;
R = 20*T*2;
```

例 3

この例は、プリプロセッサーの諸機能を使用して、次のような DO グループの実行速度を上げる方法を示したものです。

```
DO I=1 TO 10;
Z(I)=X(I)+Y(I);
END;
```

下記のように書けば、上記のように書くのと同じ結果になり、しかも、コンパイル済みプログラムの実行中に増分を加えてテストする必要がなくなります。

```
%DECLARE I FIXED;
%DO I = 1 TO 10;
Z(I)=X(I)+Y(I);
%END;
%DEACTIVATE I;
```

3 行目は入力テキストなので、置き換えをする必要があるかどうかを調べるために走査されます。この行が最初に走査されたときは、I の値は 1 であり、しかも I は活動化されています。ですから、次のステートメントがプリプロセッサー出力に挿入されます。

```
Z(       1)=X(       1)+Y(       1);
```

1 の前には、それぞれブランクが 7 入ります。

I の値が増えて 10 になるまで、そのつど、入力テキストが走査され、 I があれば I の現行値で置き換えられます。その結果、次のステートメントがプリプロセッサー出力に挿入されます。

```
Z(       1)=X(       1)+Y(       1);
Z(       2)=X(       2)+Y(       2);
          .
          .
          .
Z(      10)=X(      10)+Y(      10);
```

I の値が 11 に達すると、制御が %DEACTIVATE ステートメントに渡ります。

例 4

下記のプリプロセッサー入力内の VALUE は、'arg1(arg2)' の形をした文字ストリングを返すプリプロセッサー関数プロシージャーです。ただし、 *arg1* と *arg2* は、この関数に渡される引数を表しています。

```
DECLARE (Z(10), Q) FIXED;
%A='Z';
%ACTIVATE A, VALUE;
Q = 6 + VALUE(A,3);
%DECLARE A CHARACTER;
```

```
%VALUE: PROC(ARG1,ARG2) RETURNS(CHAR);
        DCL ARG1 CHAR, ARG2 FIXED;
        RETURN(ARG1||'('||ARG2||')');
        %END VALUE;
```

4 行目が走査されるとき、A はアクティブになっているので置き換えに対して適格です。 VALUE もアクティブなので、4 行目で VALUE が参照されると、その名前のプリプロセッサー関数プロシージャーが起動します。

ただし、引数 A と 3 が VALUE に渡される前に、A は値 Z (前の代入ステートメントで A に割り当てられた値) で置き換えられ、 3 は対応するパラメーターの属性に合わせて固定小数点フォーマットに変換されます。引数が渡されると、VALUE はこれらの引数と括弧の連結演算を行い、結果の値、つまりストリング Z (3) を呼び出し点に返します。関数参照が戻り値で置き換えられ、その結果がプリプロセッサー出力に挿入されます。その結果、次のようなプリプロセッサー出力が生成されます。

```
DECLARE (Z(10),Q) FIXED;
Q = 6+Z(       3);
```

例 5

下記のように定義されたプリプロセッサー関数プロシージャー GEN は、最高 99 個の入り口名 (パラメーター記述子リスト内に最高 99 個のパラメーター記述子を持つ) を指定する GENERIC 宣言を生成することができます。この例では、4 個だけが生成されます。

```
%DCL GEN ENTRY;
DCL A GEN (A,2,5,FIXED);
  %GEN: PROC(NAME,LOW,HIGH,ATTR) RETURNS (CHAR);
DCL (NAME, SUFFIX, ATTR, STRING) CHAR, (LOW, HIGH, I, J) FIXED;
STRING='GENERIC(';
DO I=LOW TO HIGH;                       /* ENTRY NAME LOOP*/
   IF I>9 THEN
      SUFFIX=SUBSTR(I, 7, 2);
                                        /* 2 DIGIT SUFFIX*/
   ELSE SUFFIX=SUBSTR(I, 8, 1);
                                        /* 1 DIGIT SUFFIX*/
   STRING=STRING||NAME||SUFFIX||' WHEN (';
   DO J=1 TO I;                         /* DESCRIPTOR LIST*/
      STRING=STRING||ATTR;
      IF J<I                            /* ATTRIBUTE SEPARATOR*/
         THEN STRING=STRING||',';
         ELSE STRING=STRING||')';
                                        /* LIST SEPARATOR */
   END;
   IF I<HIGH THEN                       /* ENTRY NAME SEPARATOR*/
      STRING=STRING||',';
   ELSE STRING=STRING||')';
                                       /* END OF LIST /*
END;
RETURN (STRING)
% END;
```

次のようなプリプロセッサー出力が作成されます。

```
DCL A GENERIC(A2 WHEN (FIXED,FIXED),
              A3 WHEN (FIXED, FIXED, FIXED),
              A4 WHEN (FIXED, FIXED, FIXED, FIXED),
              A5 WHEN (FIXED, FIXED, FIXED, FIXED, FIXED));
```

例 6

この例では、下記のフォーマットのステートメントと同じ働きをするプリプロセッサー・プロシージャーを示します。

```
SEARCH TABLE(array) FOR(value)
USING(variable) AND(variable);
```

このステートメントは、配列添え字として指定した (またはデフォルトの) 変数を使用して、指定した値を求めて 2 次元配列を探索するためのステートメントです。このステートメントが実行されると、指定した値がどのエレメントに入っているかが配列添え字変数によって示されます。指定した値がどのエレメントにも入っていない場合は、両方の添え字変数が -22222 にセットされます。

このステートメントと同じ働きをするプリプロセッサー・プロシージャーは、次のとおりです。

```
%SEARCH:
PROC(TABLE,FOR,USING,AND) STATEMENT RETURNS(CHARACTER);

  DECLARE(TABLE,FOR,USING,AND,LABL, DO1,DO2) CHARACTER,
      (PARMSET,COUNTER) BUILTIN;

  IF PARMSET(TABLE) & PARMSET(FOR) THEN;
  ELSE SERR:DO;
  NOTE ('MISSING OR INVALID ARGUMENT(S)'||'FOR ''SEARCH''',4);
  RETURN ('/*INVALID SEARCH STATEMENT*/');
  END;

  IF ¬PARMSET(USING) THEN
    USING='I';
  IF ¬PARMSET(AND)   THEN
    AND='J';
  IF USING = AND  THEN
    GO TO SERR;

  LABL='SL'||COUNTER;
  DO1=LABL||': DO '||USING||'=LBOUND('||TABLE||',1)
    TO HBOUND('||TABLE||',1);';
  DO2='DO '||AND||'=LBOUND('||TABLE||',2)
    TO HBOUND ('||TABLE||',2);';

  RETURN(DO1||DO2||'SELECT('||TABLE
    ||'('||USING||','||AND||'));
  WHEN('||FOR||') LEAVE '||LABL||';
  OTHER;
  END '||LABL||';
  IF '||AND||' > H  BOUND('||TABLE||',2) THEN
    '||USING||','||AND||.' = -22222;');
%END SEARCH;
```

この例では、PARMSET 組み込み関数を使用して、このプロシージャーが呼び出されたときにどのパラメーターがセットされるかを判別しています。 USING がセットされていない場合、配列添え字付き変数 I のデフォルトが使用されます。 AND がセットされていないと、J が使用されます。 TABLE または FOR がセットされていないか、または呼び出しの結果として両方の添え字が同じ変数になった場合は、プリプロセッサー診断メッセージが出され、コメントがプリプロセッサー出力に返されます。

COUNTER 組み込み関数は、このプロシージャーから返されるプリプロセッサー出力に固有のラベルを付けるために使われています。

このプロシージャーを呼び出すときには、キーワード引数、定位置引数、または両者の組み合わせのどれでも指定することができます。次のようにプロシージャーを呼び出すと、同じ結果が生成されます。

```
SEARCH TABLE(LIST.NAME) FOR('J.DOE') USING(I) AND(J);

SEARCH TABLE(LIST.NAME) FOR('J.DOE');

SEARCH(LIST.NAME) FOR('J.DOE');

SEARCH(LIST.NAME,'J.DOE');

SEARCH(,'J.DOE') TABLE(LIST.NAME);
```

どの方法で呼び出した場合も、返されたプリプロセッサー出力は次のようになります。

```
SL00001:
DO I=LBOUND(LIST.NAME,1) TO HBOUND(LIST.NAME,1);
   DO J=LBOUND(LIST.NAME,2) TO HBOUND(LIST.NAME,2);
      SELECT(LIST.NAME(I,J));
   WHEN('J.DOE') LEAVE SL00001;
   OTHER;
END SL00001;
IF J > HBOUND(LIST.NAME,2) THEN
   I,J = -22222;
```

ラベル SL00001 は、最初の呼び出しの場合にだけ返されます。それ以後、このプロシージャーを呼び出すたびに、新しい固有のラベルが返されます。

付録. 制限値

表 67 に、PL/I 言語エレメントの設定制限値を要約しています。また、 577 ページの表 68 に、マクロ機能の言語エレメントの設定制限値を要約しています。

表 67. 言語エレメントの制限

言語エレメント	説明	制限
配列	配列の次元の最大数	15
	最小下限 (注 1)	-2147483648
	最大上限 (注 1)	+2147483647
注 1: コンパイル時オプション CMPAT(V1) が指定されている場合、最小下限は -32768、最大上限は 32767 です。また、これらの限界は注意して使用する必要があります。たとえば、A に最大上限が指定されていて、JX に属性 SIGNED FIXED BIN(31) が指定されている場合、ループ DO JX = LBOUND(A) TO HBOUND(A) は配列の最後のエレメントにヒットした後「ラップ」します。 TO の代わりに UPTHRU を使用した場合、ループはラップしません。		
構造体	構造体内のレベルの最大数	15
	構造体内の最大レベル番号	255
算術値の精度	FIXED DECIMAL の最大精度	31 (注 2)
	FIXED BINARY の最大精度	63 (注 3)
	FLOAT DECIMAL の最大精度	33 (注 4)
	FLOAT BINARY の最大精度	109 (注 5)
	FIXED データの最大スケール因数	127
	FIXED データの最小スケール因数	-128
注 2: これはコンパイル時オプション LIMITS(FIXEDDEC(31)) を指定した場合に限り当てはまります。デフォルトは 15 です。		
注 3: これはコンパイル時オプション LIMITS(FIXEDBIN(63)) を指定した場合に限り当てはまります。デフォルトは 31 です。		
注 4: Intel の FLOAT DECIMAL 最大精度は 18 です。		
注 5: Intel の FLOAT BINARY 最大精度は 64 です。		
ストリング変数/定数と AREA 変数/定数	CHARACTER の最大の長さ	32767
	BIT の最大の長さ	32767
	GRAPHIC の最大の長さ	16383
	WIDECHAR の最大の長さ	16383
	AREA の最大サイズ	2147483647

表 67. 言語エレメントの制限 (続き)

言語エレメント	説明	制限
組み込み関数	IAND、IOR、MAX、および MIN 関数への引数の最大個数	64
	ADD、BINARY、DECIMAL、DIVIDE、FIXED、FLOAT、MULTIPLY、PRECISION、および SUBTRACT 関数内の精度 (p) の最大値	対応する算術精度の制限と同じ
	ADD、BINARY、DECIMAL、DIVIDE、FIXED、MULTIPLY、PRECISION、および SUBTRACT 関数内のスケール (q) の最大値	対応する算術精度の制限と同じ
	CEIL、FLOOR、MAX、MIN、MOD、ROUND および TRUNC 関数内の桁 (N) の最大数	対応する算術精度の制限と同じ
プログラム・サイズ	ID の最大の長さ	100
	プログラム内のプロシージャーの最大数	255
	ステートメント・タイプが変更される前の字句単位 (キーワード、ID、区切り文字など) の最大数	511
	ブロック内の DEFAULT ステートメントの最大数	31
	%PUSH ステートメントの最大数	63
	%INCLUDE ステートメントの最大数	2047
	%INCLUDE ステートメントの最大ネスト	2046
	ソース・ファイル内の最大行数	1048575
	ステートメントの最大数	16777215
	ブロック内の LIKE 属性の最大数	63
	データ・リスト内の出力式の最大数	60
	データ・リスト内の反復 DO 指定の最大数	50
	位置合わせされていないビットを含まないデータ集合の最大サイズ	2147483647
	位置合わせされていないビットを含むデータ集合の最大サイズ	268435455
	CALL または関数参照内の引数の最大数	255
	プロシージャーの最大パラメーター数	4095
	分配された属性の最大ネスト	15
	BEGIN および PROCEDURE ステートメントの最大ネスト	30
	DO グループの最大ネスト	49
	IF ステートメントの最大ネスト	49
	SELECT ステートメントの最大ネスト	49
	式の最大ネスト	383
	%NOTE メッセージの最大の長さ	32767

表 67. 言語エレメントの制限 (続き)

言語エレメント	説明	制限
その他	文字ピクチャー内のピクチャー文字の最大数	511
	数値ピクチャー内の最大バイト数	253
	数値ピクチャー内の数値ピクチャー文字の最大数	31
	CHARACTER、X、BIT、BX、GRAPHIC、GX、WX および M ストリング定数の外部表示の最大バイト数 外部表示には、すべての引用符とストリング接尾部が含まれます。たとえば、ストリング '01010110'B にはその外部指定内に 11 バイトありますが、内部指定には 1 バイトしかありません。同様に、ストリング 'Ain''t Misbehavin''' にはその外部指定内に 21 バイトありますが、内部表示には 17 バイトしかありません。	3072
	KEYTO 文字ストリングの最大の長さ	120
	KEYTO グラフィックまたはワイド文字ストリングの最大長	60
	KEY の最大の長さ	32763
	LINESIZE の最大行サイズ	F フォーマットあるいは U フォーマットでは 32,759、V フォーマットでは 32,751
	LINESIZE の最小行サイズ	1
	PAGESIZE の最大ページ・サイズ	32,767
	PAGESIZE コンパイラー・オプションの最小ページ・サイズ	1
	DISPLAY 文字ストリングの最大サイズ	126
	最大の DISPLAY 応答メッセージ	72 バイト
	IEEE 正規化浮動小数点数値の範囲	+3.30E-4932 から +1.21E+4932、0、-3.30E-4932 から -1.21E+4932
	16 進浮動小数点数値の範囲	+10E-78 から +10E75、0、-10E-78 から +10E+75

表 68. マクロ機能の制限

言語エレメント	説明	制限
配列	最大次元数	15
	下限の最小値	-32768
	上限の最大値	+32767

表 68. マクロ機能の制限 (続き)

言語エレメント	説明	制限
算術値の範囲	最大および最小の FIXED(DECIMAL) オプションによる FIXED 変数	FIXED DECIMAL(5) ID と同じ
	最大および最小の FIXED(BINARY) オプションによる FIXED 変数	FIXED BINARY(31) ID と同じ
マクロ・プロシージャー	最大ネスティング・レベル	1
キー	キーワード・パラメーターの最大個数	4096

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-8711
東京都港区六本木 3-2-12
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。

国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。

本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

商標

以下は、IBM Corporation の商標です。

AIX
CICS
CICS/ESA
DB2
DFSMS
DFSORT
IBM
IMS
IMS/ESA
Language Environment
MVS
z/OS
RACF
System/390
VisualAge
z/OS

Intel および Pentium は、Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Microsoft、Windows、および Windows NT は、Microsoft Corporation の米国およびその他の国における商標です。

Pentium は、Intel Corporation の米国およびその他の国における商標です。

Unicode は、Unicode Consortium の商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

参考文献

Enterprise PL/I 資料

「*プログラミング・ガイド*」、SC88-9123
「*言語解説書*」、SC88-9126
「*メッセージおよびコード*」、SC88-9127
「*診断ガイド*」、GC88-9125
「*コンパイラーおよびランタイム 移行ガイド*」、GC88-9124

PL/I for MVS & VM

「*導入およびカストマイズ (MVS)*」、SC88-7221
「*言語解説書*」、SC88-7219
「*コンパイル時メッセージおよびコード*」、SC88-7224
「*診断の手引き*」、SC88-7223
「*移行の手引き*」、SC88-7220
「*プログラミングの手引き*」、SC88-7218
「*参照要約*」、SX88-7011

z/OS 言語環境プログラム

「*概念*」、SA88-8555
「*デバッグのガイド*」、GA88-8548
「*ランタイム・メッセージ*」、SA88-8554
「*カスタマイズ*」、SA88-8552
「*プログラミング・ガイド*」、SA88-8549
「*プログラミング・リファレンス*」、SA88-8550
「*ランタイム・アプリケーション マイグレーション・ガイド*」、 GA88-8553
「*ILC (言語間通信) アプリケーションの作成*」、SA88-8551

CICS Transaction Server

「*アプリケーション・プログラミング・ガイド*」、SC88-7689
「*アプリケーション・プログラミング・リファレンス*」、SC88-7690
「*カスタマイズ・ガイド*」、SC88-7686
「*外部インターフェース・ガイド*」、SD88-7026

DB2 UDB for z/OS

「管理ガイド」、 SC88-9806

「アプリケーション・プログラミングおよび SQL ガイド」、 SC88-9808

「コマンド解説書」、 SC88-9809

「メッセージ」、 GC88-4061

「コード」、 GC88-4062

「SQL 解説書」、 SC88-9817

DFSORT™

「アプリケーション・プログラミングの手引き」、SC88-7061

「導入およびカストマイズ」、SC88-7163

IMS/ESA®

「アプリケーション・プログラミング: データベース管理プログラム」、SC88-7552

「Application Programming: Database Manager Summary」、SC26-8037

「アプリケーション・プログラミング: 設計の手引き」、SC88-7542

「アプリケーション・プログラミング: トランザクション管理プログラミング」、SC88-7553

「Application Programming: Transaction Manager Summary」、SC26-8038

「アプリケーション・プログラミング: EXEC DL/I コマンド (CICS および IMS™)」、SC88-7554

「Application Programming: EXEC DL/I Commands for CICS and IMS Summary」、SC26-8036

z/OS MVS

「JCL 解説書」、SA88-8569

「JCL ユーザーズ・ガイド」、SA88-8570

「システム・コマンド」、SA88-8593

z/OS UNIX システム・サービス

「z/OS UNIX システム・サービス コマンド解説書」、SA88-8461

「z/OS UNIX システム・サービス プログラミング: アセンブラー呼び出し可能サービス 解説書」、SA88-8642

「z/OS UNIX システム・サービス ユーザーズ・ガイド」、 SA88-8640

z/OS TSO/E

「コマンド解説書」、SA88-8628

「ユーザーズ・ガイド」、 SA88-8638

z/Architecture

「*Principles of Operation*」、SA22-7832

Unicode® および文字表現

「*OS/390 Unicode サポート: 変換サービスの使用法*」、SD88-6163

用語集

この用語集は、PL/I のすべてのプラットフォームとリリースで使用する用語を定義したものです。このマニュアルで使用されていない用語が含まれていることがあります。該当する用語が見つからない場合は、本書の索引を調べるか、「*IBM Dictionary of Computing*」、SC20-1699 を参照してください。

［ア行］

あいまい参照 (ambiguous reference). 参照時点で認識されている名前をただ 1 つだけ識別するためには修飾が不十分な参照。

アクセス (access). データを参照するかまたは取り出すこと。

アクティブ (active). 活動化から終了にいたるまでのブロックの状態。ソース・プログラム・テキスト中の対応する ID をプリプロセッサー変数やプリプロセッサー入り口名の値に置き換えることができるときの、その変数や入り口の状態。イベント変数が非同期操作に関連付けられている間に置かれている状態。タスク変数に関連するタスクが付加されるときにタスク変数が置かれている状態。タスクが終了する前に置かれている状態。

値参照 (value reference). データ項目の値を得るのに使用する参照。

アテンション (attention). タスクに割り込みが生じる原因となるような、タスクにとっては外部の事柄の発生。

暗黙オープン (implicit opening). OPEN ステートメント以外の入力ステートメントまたは出力ステートメントが原因で、ファイルがオープンされること。

暗黙処置 (implicit action). 使用可能な条件が生じたときに、その条件用に現在確立されている ON ユニットがない場合にとられる処置。 *ON ステートメント処置 (ON-statement action)* と対比。

暗黙宣言 (implicit declaration). DECLARE ステートメント内で明示的に宣言されていないか、または内容に従って宣言されていない名前。

暗黙の (implicit). 明示指定のないまま取られる処置。

位置合わせ (alignment). 機械に依存する特定の境界 (たとえば、フルワード境界またはハーフワード境界) に関連付けて、データ項目を保管すること。

イベント (event). 状況および完了を、関連したイベント変数から決定することのできるプログラムの活動。

イベント変数 (event variable). イベントと関連付けることができる EVENT 属性を持つ変数。その値は、処置が完了したかどうか、および完了の状況を示す。

入り口値 (entry value). 入り口定数または入り口変数によって表されるエントリー・ポイント。入り口値には、その入り口定数に関連した活動化環境が含まれる。

入り口参照 (entry reference). 入り口値を返す入り口定数、入り口変数参照、または関数参照。

入り口式 (entry expression). 評価されると入り口名を生じるような式。

入り口データ (entry data). プロシージャーへのエントリー・ポイントを表すデータ項目。

入り口定数 (entry constant). PROCEDURE ステートメントのラベル接頭部 (入り口名)。 ENTRY 属性を指定し、VARIABLE 属性を指定しないで名前を宣言すること。

入り口変数 (entry variable). 入り口値を割り当てる対象となりうる変数。これは、ENTRY 属性と VARIABLE 属性を両方とも持っている必要がある。

入り口名 (entry name). ENTRY 属性を持つものとして明示的または内容に従って宣言された ID (ただし、VARIABLE 属性が与えられていない場合に限る)。または、ENTRY 属性を暗黙指定された入り口変数の値を持った ID。

埋め込み (padding). ストリングの長さを必要な長さまで拡張するために、ストリングの右側に連結される、1 つまたは複数の文字、漢字、またはビット。構造体または共用体の中に挿入される、1 つまたは複数のバイトまたはビット。その構造、または共用体内の後続エレメントが正しい規定境界に位置合わせされるようにするためのもの。

英字 (alphabetic character). A から Z までの任意の英字と、#、$、@ (これらのグラフィック表記は国によって異なる場合がある) の拡張英字。

英数字 (alphameric character). 英字または数字。

エクステント (extent). 配列の次元の境界、ストリング長、または区域サイズによって示される範囲。この区域がターゲット区域に割り当てられる場合は、ターゲット区域のサイズ。

エピローグ (epilogue). ブロックまたはタスクの終了時に自動的に生じる各種処理。

エレメント (element). 配列などのデータ項目の集まりとは対照的な、単一のデータ項目。スカラー項目。

エレメント名 (elementary name). 「*基本エレメント (base element)*」を参照。

演算子 (operator). 実行する演算を指定する記号。

演算式 (operational expression). 1 つまたは複数の演算子から成る式。

エントリー・ポイント (entry point). そこでプロシージャーを呼び出すことができるプロシージャー内の 1 地点。「*1 次エントリー・ポイント (primary entry point)*」および「*2 次エントリー・ポイント (secondary entry point)*」も参照。

オープン (ファイルの) (opening (of a file)). ファイルをデータ・セットに関連付けること。

オブジェクト (object). 単一名で参照されるデータの集まり。

オプション (option). ステートメントの実行や解釈に影響を及ぼすのに使われるステートメント中の指定。

オフセット変数 (offset variable). OFFSET 属性を持ったロケーター変数のことであり、その値は、ストレージ内のある区域の先頭からの相対位置を識別する。

オペランド (operand). ID、定数、または式。式には演算子が、時には他のオペランドとともに使用される。

オン条件 (ON-condition). PL/I プログラムにおける、プログラム割り込みの原因となりうるオカレンス。予期しないエラーが検出されたり、予期できる出来事ではあるものの、予期しない時にそれが起きたときに発生する。

[カ行]

介在添え字 (interleaved subscripts). 添え字付き修飾参照の最下位レベル以外のレベルに存在する添え字。

介在配列 (interleaved array). 非連結ストレージを参照する配列。

開始ブロック (begin-block). BEGIN ステートメントと END ステートメントによって区切られ、名前有効範囲を形成するステートメントの集まり。開始ブロックの活動化は、条件が生じたために行われる (開始ブロックが ON ユニットへの処置指定である場合) か、または GOTO ステートメントの結果の分岐を含め、通常の制御の流れを介して行われる。

外部シンボル (external symbol). それ自身が定義されている制御セクションを除く制御セクション内で参照できる名前。

外部シンボル辞書 (External Symbol Dictionary (ESD)). オブジェクト・モジュール内で使われるすべての外部シンボルの一覧表。

外部プロシージャー (external procedure). 他のいずれのプロシージャーにも組み込まれないプロシージャー。パッケージ内に入っていて同様にエクスポートされるレベル 2 のプロシージャー。

外部名 (external name). 有効範囲が必ずしも 1 つのブロックとその収容ブロックだけに限定されない (EXTERNAL 属性を持つ) 名前。

拡張英字 (extended alphabet). A から Z までの大文字、小文字の英字、$、@、および #、または NAMES コンパイラー・オプションで指定されたもの。

確立された処置 (established action). 条件が生じたときにとられる処置。「*暗黙の処置 (implicit action)*」および「*ON ステートメント処置 (ON-statement action)*」も参照。

下限 (lower bound). 配列次元の下限。

仮想起点 (virtual origin (VO)). すべてゼロの添え字を持った配列のエレメントを保持するための位置。このようなエレメントが配列内になければ、仮想起点は本来それが保持されるべき場所になる。

型変換 (conversion). ある 1 つの表現法から、一組の特定属性に合うよう別の表現法に値を変換すること。たとえば、文字ストリングを FIXED BINARY (15,0) などの算術値に変換すること。

活動化 (プリプロセッサー変数またはプリプロセッサー・エントリー・ポイントの) (activate (a preprocessor variable or preprocessor entry point)). マクロ機能 ID を、それに後続するソース・コード内で置換可能にすること。 %ACTIVATE ステートメントは、プリプロセッサー変数やプリプロセッサー・エントリー・ポイントを活動化する。

活動化 (ブロックの) (activate (a block)). ブロックの実行を開始すること。プロシージャー・ブロックは、呼び出されるときに、活動化される。開始ブロックが活動化するのは、分岐を含め、通常の制御の流れ内に現れたときである。パッケージを活動化することはできない。

仮引数 (dummy argument). 参照によって渡すことのできない引数の値を保持するため自動的に作成される一時記憶域。

環境 (活動化の) (environment (of an activation)). 収容ブロック内で宣言されたデータに関して、呼び出されたブロックと関連し、そのブロック内で使用される情報。

環境 (ラベル定数の) (environment (of a label constant)). ステートメント・ラベル定数への参照が適用されるブロックの個々の活動化の識別情報。この情報が決定されるのは、ステートメント・ラベル定数が、引数として渡されたり、またはステートメント・ラベル変数に割り当てられ、それが定数と一緒に渡されたり割り当てられたときである。

関数 (プロシージャー) (function (procedure)). PROCEDURE ステートメント内に RETURNS オプションのあるプロシージャー。 RETURNS 属性を指定して宣言された名前。これは、関数参照内にその入り口名のうちの 1 つがあると呼び出され、スカラー値を参照点に返す。*サブルーチン (subroutine)* と対比。

関数参照 (function reference). 入り口定数または入り口変数のことで、このどちらも関数を表さなければならないが、その後に空と考えられる引数リストが続く。*サブルーチン呼び出し (subroutine call)* と対比。

完全修飾名 (fully-qualified name). 名前が参照するメンバーより上の階層順序内のすべての名前と、そのメンバー自身の名前が組み込まれている名前。

キー (key). 直接アクセス・データ・セット内のレコードを識別するデータ。「*ソース・キー (source key)*」および「*記録済みキー (recorded key)*」を参照。

キーワード (keyword). PL/I において定義されたコンテキスト内で使用されると特定の意味を持つ ID。

キーワード・ステートメント (keyword statement). ステートメントの機能を示すキーワードで始まる単純ステートメント。

疑似変数 (pseudovariable). ターゲット変数を指定するのに使用できるすべての組み込み関数の名前。これは通常、代入ステートメントの左側にある。

記述子 (descriptor). 区域サイズ、配列境界、またはストリング長などの変数に関する情報を保持する制御ブロック。

基数 (base). 算術値を表現するための数体系。

規定境界 (integral boundary). そこでデータを位置合わせすることができる任意の 8 ビット単位のバイト・マルチアドレス。通常はハーフワード、フルワード、またはダブルワード (2、4、または 8 バイトの長さの整倍数) 境界である。

基底付き参照 (based reference). 基底付きストレージ・クラスを持った参照。

基底付きストレージ割り振り (based storage allocation). 基底付き変数用のストレージの割り振り。

基底付き変数 (based variable). ストレージ・アドレスがロケーターによって与えられる変数。同一変数の複数の世代をアクセスすることができる。これは、ストレージ内の固定位置を識別しない。

起動 (invocation). プロシージャーの活動化。

起動する (invoke). プロシージャーを活動化すること。

基本エレメント (base element). それ自身は別の構造体や共用体ではない、構造体や共用体のメンバー。

基本項目 (base item). 定義変数を定義するための、自動、被制御、または静的変数、またはパラメーター。

境界 (bounds). 任意の配列次元の上限と下限。

共用体 (union). 同一のストレージを占有し、相互にオーバーレイしたデータ・エレメントの集まり。メンバーは構造体、共用体、基本変数、または配列のいずれであっても構わない。それらは、同一の属性を持っていなくても構わない。

切り捨て (truncation). ターゲット変数のストリング長や精度が限度を超えたときに、データ項目の片方の端から 1 つまたは複数の数字、文字、グラフィックス、またはビットを除去すること。

記録済みキー (recorded key). 直接アクセス・データ・セット内でレコードを識別する文字ストリングのことであり、そこでは文字ストリングそのものもデータの一部として記録される。

区切り文字 (break character). 下線記号 (_)。 ID を読みやすくするために使用することができる。たとえば、変数を OLDINVENTORYTOTAL とする代わりに OLD_INVENTORY_TOTAL と記述できる。

区切り文字 (delimiter). すべてのコメントと、パーセント記号、括弧、コンマ、ピリオド、セミコロン、コロン、割り当て記号、ブランク、ポインター、アスタリスク、および単一引用符。これらは ID、定数、ピクチャー指定、iSUB、およびキーワードの限界を定めるものとなる。

区切る (delimit). 1 つまたは複数の項目またはステートメントの前後を、文字またはキーワードで囲むこと。

組み込み関数 (built in function). SQRT (平方根) のような、言語が提供する定義済み関数。

組み込み関数参照 (built-in function reference). オプショナルの引数リストを持つ組み込み関数名。

組み込みサブルーチン (built-in subroutine). コンパイル時に定義され、CALL ステートメントによって呼び出される入り口名を持つサブルーチン。

組み込み名 (built-in name). 組み込みサブルーチンの入り口名。

グループ (group). より大きいプログラム単位に入っているステートメントの集まり。グループは、DO グループまたは選択グループのどちらかであるが、 ON ユニットとしての場合を除き、単一ステートメントを使用できるところでは常に使用することができる。

クローズ (ファイルの) (closing (of a file)). ファイルをデータ・セットまたは装置と切り離すこと。

継承次元 (inherited dimension). 構造体、共用体、またはエレメントでの、収容構造から派生する次元。名前が配列ではないエレメントであれば、その次元全体が継承次元で構成される。名前が配列であるエレメントであれば、その次元は、継承次元および明示的に宣言された次元で構成される。 1 つまたは複数の継承次元を持つ構造体を、非結合集合と呼ぶ。*結合集合 (connected aggregate)* と対比。

現行世代 (current generation). 変数名を参照して、現在使用できる自動変数または被制御変数の世代。

コード化算術データ (coded arithmetic data). 数値を表し、基数 (10 進数または 2 進数)、スケール (固定小数点または浮動小数点)、および精度 (個々に持ちうる桁数) を特徴とするデータ項目。このデータは、変換しなくても、算術計算用に受け入れることのできる形式で保管される。

合成演算子 (composite operator). <=、**、および /* などの特殊文字を複数含む演算子。

構造化 (structuring). メンバー数、配置順、属性、および論理レベルによって表現される構造階層。

構造式 (structure expression). 評価されると構造体の値セットを生成する式。

構造体 (structure). 必ずしも同じ属性を持たなくても差し支えないデータ項目の集まり。*配列 (array)* と対比。

構造体の配列 (array of structures). 次元属性を構造体名に与えて指定される、順番に並べられた同一構造体の集まり。

構造体メンバー (structure member). 「*メンバー (member)*」を参照。

固定小数点定数 (fixed-point constant). 「*算術定数 (arithmetic constant)*」を参照。

コメント (comment). 文書化のために使用され、/* および */ で区切られる、ゼロ以上の文字数の文字ストリング。

コンテキスト宣言 (contextual declaration). DECLARE ステートメントで明示的に宣言されていないが、その使用の前後関係から、特定の属性が ID に関連付けられるような ID の存在。

コンパイラー・オプション (compiler options). コンパイルの特定の面を制御するために指定されるキーワード。たとえば、生成するオブジェクト・モジュールの特徴や、作成する印刷出力のタイプなどがある。

コンパイル時 (間) (compile time). 一般に、ソース・プログラムがオブジェクト・モジュールに変換されている時間。 PL/I では、変更したい場合に、ソース・プログラムを変更して、オブジェクト・プログラムに変換し終わるまでに経過する時間。

［サ行］

再帰的プロシージャー (recursive procedure). そのプロシージャー自身からでも、または別のアクティブ・プロシージャーからでも呼び出すことのできるプロシージャー。

再入可能プロシージャー (reentrant procedure). 複数のタスク、スレッド、またはプロセスから同時に活動化でき、しかもこれらのタスク、スレッド、およびプロセス間で相互に干渉が生じないプロシージャー。

サブタスク (subtask). 特定のタスクによって生成されるタスク、または特定のタスクから最後に生成されたタスクへの直接ライン内の任意のタスク。

サブルーチン (subroutine). PROCEDURE ステートメント内に RETURNS オプションのないプロシージャー。*関数 (function)* と対比。

サブルーチン呼び出し (subroutine call). 後に CALL ステートメント内にあるオプショナルの引数リストが付く、サブルーチンを表さなければならないエントリー参照。*関数参照 (function reference)* と対比。

算術演算子 (arithmetic operators). 接頭演算子の + と -、あるいは挿入演算子 + - * / ** のうちのいずれか。

算術データ (arithmetic data). 基数、スケール、モード、および精度の特性を持つデータ。コード化算術データとピクチャー数字データも含まれる。

算術定数 (arithmetic constant). 固定小数点定数または浮動小数点定数。大部分の算術定数には符号を付けることができるが、符号は定数の一部ではない。

算術比較 (arithmetic comparison). 数値の比較。「*ビット比較 (bit comparison)*」、「*文字比較 (character comparison)*」も参照。

算術変換 (arithmetic conversion). ある 1 つの算術表現から別の表現に値を変換すること。

式 (expression). 値、値の配列、または一連の構造化値セットを表すのに、プログラム内で使われる表記。単独で使用される定数または参照、あるいは、定数または参照あるいはその両方を演算子と組み合わせたもの。

字句単位の (lexically). 単位を左から右への順序に扱うことに関連した用語。

次元属性 (dimension attribute). 配列の次元数を指定し、各次元の境界を示す属性。

自己定義データ (self-defining data). プログラム実行時に決定され、集合のメンバー内に保管される境界、長さ、およびサイズを持つデータ項目を含む集合。

指数文字 (exponent characters). 以下のピクチャー指定文字のこと。

1. K および E。指数フィールドの先頭を示すため、浮動小数点ピクチャー指定内で使用される文字。
2. F。10 進小数点をその想定位置から右方向へ (正定数の場合) かまたは左方向へ (負定数の場合) 移動するときに、小数部の桁数を示す整定数を使って指定されるスケール因数文字。

実際の起点 (actual origin (AO)). 配列または構造体内の最初の項目の位置。

自動ストレージ割り振り (automatic storage allocation). 自動変数用のストレージ割り振り。

自動変数 (automatic variable). ブロックの起動時に自動的にストレージを割り振られ、そのブロックの終了時に自動的にそれを解除される変数。

シフト (shift). ストレージ内のデータを元の位置の左または右へ変更すること。

シフトアウト (shift-out). 2 バイト・ストリングの先頭でコンパイラーにシグナルを送るために使用される記号。

シフトイン (shift-in). 2 バイト・ストリングの終わりをコンパイラーに知らせるために使用される記号。

集合 (aggregate). 「データ集合」を参照。

集合式 (aggregate expression). 配列式、構造式、または共用体式のこと。

集合タイプ (aggregate type). どのデータ項目の場合も、それが構造体、共用体、または配列のいずれであるかの指定。

修飾名 (qualified name). 構造メンバーまたは共用体メンバーの階層順序。ピリオドで結合されていて、構造体の中の名前を識別するのに使用される。どの名前にも添え字を付けることができる。

修正 (fix-up). コンパイル済みプログラムを実行可能にするために、コンパイル時にエラーを検出したあとでコンパイラーが実行する解決手段。

収容ブロック (containing block). 該当する宣言、ステートメント、プロシージャー、またはその他のソース・テキストを収容している、パッケージ、プロシージャー、または開始ブロック。

終了 (タスクの) (termination (of a task)). タスクへの制御の流れを停止すること。

終了 (ブロックの) (termination (of a block)). ブロックの実行が終了して、RETURN ステートメントまたは END ステートメントによって、制御がその起動側ブロックに戻るか、または GO TO ステートメントによって起動側ブロックまたは他のアクティブ・ブロックに制御が渡ること。

主プロシージャー (main procedure). OPTIONS (MAIN) 属性を持った PROCEDURE ステートメントのある外部プロシージャー。このプロシージャーは、プログラム実行の最初のステップで自動的に呼び出される。

使用可能 (enabled). 条件により割り込みが生じて、該当する規定 ON ユニットが呼び出される条件の状態。

条件 (condition). エラー (オーバーフローなど) または予期される状況 (入力ファイルの終わりなど) のいずれかの例外的な状態。条件が発生する (検出される) と、その条件に対する規定のアクションが処理される。「*確立された処置 (established action)*」および「*暗黙処置 (implicit action)*」も参照。

上限 (upper bound). 配列次元の上限。

条件接頭語 (condition prefix). ステートメントの接頭部として付けられる、括弧で囲まれた 1 つまたは複数の条件名のリスト。条件接頭語は、指定した条件を使用可能にするか使用不能にするかを指定する。

条件名 (condition name). PL/I 定義またはプログラマー定義の条件の名前。

小構造 (minor structure). 別の構造体または共用体の中に組み込まれている構造体。小構造の名前は、1 よりも大きくかつ親構造体または親共用体よりも大きいレベル番号を指定して宣言される。

使用不可の (disabled). 割り込みが発生せず、規定の処置も取られないような事態になった状態。

商用文字 (commercial character).

- CR (貸方) ピクチャー指定文字。
- DB (借方) ピクチャー指定文字。

処置指定 (action specification). ON ステートメント内にある、ON ユニットまたは単一のキーワード SYSTEM。該当する条件が発生すれば、2 つのうちいずれかがとるべき処置を指定する。

数字 (digit). 0 から 9 までの文字の 1 つ。

数字データ (numeric-character data). 「*10 進ピクチャー・データ (decimal picture data)*」を参照。

数値ピクチャー・データ (numeric picture data). 算術値と文字値を持ったピクチャー・データ。このタイプのピクチャー・データは、'A' または 'X.' という文字を含むことはできない。

スカラー変数 (scalar variable). 構造、共用体、配列ではない変数。

スケール (scale). 1 つの数値表記体系であり、その算術値は固定小数点または浮動小数点で表現される。

スケール因数 (scale factor). 固定小数点数内の小数桁数の指定。

スケール因数 (scaling factor). 「*スケール因数 (scale factor)*」を参照。

ステートメント (statement). キーワード、区切り文字、ID、演算子、および定数から構成され、セミコロン (;) で終わる PL/I ステートメント。任意で、条件接頭語リストとラベルのリストを付けることができる。「*キーワード・ステートメント (keyword statement)*」、「*代入ステートメント (assignment statement)*」、および「*ヌル・ステートメント (null statement)*」も参照。

ステートメント本体 (statement body). ステートメント本体は、単純ステートメントまたは複合ステートメントのどちらでもかまわない。

ステートメント・ラベル (statement label). 「*ラベル定数 (label constant)*」を参照。

ストリーム指向データ伝送 (stream-oriented data transmission). 文字形式になった個々のデータ値の連続ストリームであるものとしてデータを扱って、データを伝送すること。*レコード単位データ伝送 (record-oriented data transmission)* と対比。

ストリング (string). 単一のデータ項目として処理される、連続した文字、グラフィックス、またはビットの列。

ストリング変数 (string variable).
BIT、CHARACTER、または GRAPHIC 属性を指定して宣言される変数。この変数の値は、ビット・ストリング、文字ストリング、または漢字ストリングのいずれでもかまわない。

制御セクション (control sections). オブジェクト・モジュール内のグループ化された機械命令。

制御の流れ (flow of control). 実行の連なり。

制御フォーマット項目 (control format item). ストリーム内または印刷ページの内での、あるデータ項目の位置付けを指定するために、編集ディレクティブ伝送の中で使用される指定。

制御変数 (control variable). DO ステートメントの反復実行を制御するのに使用する変数。

制御文字 (control character). 特定コンテキスト内に存在することによって制御機能が指定される、文字セット内の文字。 1 つの例としてファイルの終わり (EOF) マーカーがある。

制限付き式 (restricted expression). コンパイル時にコンパイラーによって評価されて定数を生じる式。このような式のオペランドは、定数、指定した定数、および制限付きの式になる。

整数 (integer). 符号を付けるか付けないかは任意の、10 進または 2 進小数点のない一連の数字、または一連のビット。通常は、FIXED BINARY (p,0) または FIXED DECIMAL (p,0) と記述される、符号を付けるか付けないかは任意の整数。

静的ストレージ割り振り (static storage allocation). 静的変数用のストレージの割り振り。

静的変数 (static variable). プログラム実行の開始前に割り振られ、その実行の継続時間中はその割り振りの変わらない変数。

精度 (precision). 固定小数点データ項目内にある桁数またはビット数、または、浮動小数点データ項目での最小確保有効数字 (指数は除く) の数。

世代 (変数の) (generation (of a variable)). 静的変数の割り振り、被制御変数または自動変数の特定の割り振り、または基底付き変数の特定のロケーター修飾で、または定義された変数かパラメーターで指示されるストレージ。

接頭演算子 (prefix operator). オペランドの前に置かれ、そのオペランドにだけ適用される演算子。接頭演算子には、プラス (+)、マイナス (-)、および not (¬) がある。

接頭部 (prefix). ステートメントの先頭に付けられるラベル、または 1 つまたは複数の条件名の括弧で囲まれたリスト。

ゼロ抑止文字 (zero-suppression characters). ピクチャー指定文字の Z と *。これは、対応する桁位置のゼロを抑止し、それぞれをブランクまたはアスタリスクで置き換えるのに使用する。

宣言 (declaration). ID を名前として確立し、その ID 用に一連の属性を (部分的または全体的に) 指定すること。特定名の属性のソース。

先行ゼロ (leading zeroes). 算術値としては意味のないゼロ。ある数値内で最初の非ゼロより左側にあるすべてのゼロ。

選択グループ (select-group). SELECT ステートメントと END ステートメントで区切られたステートメントの連なり。

選択文節 (selection clause). 選択グループの WHEN 文節または OTHERWISE 文節。

ソース (source). 問題データに変換されるデータ項目。

ソース変数 (source variable). 他の演算に使用されるが、その演算で変更されることのない変数。ターゲット変数 *(target variable)* と対比。

ソース・キー (source key). 直接アクセス・データ・セット内で個々のレコードを識別するため、レコード単位伝送ステートメント内で参照されるキー。

ソース・プログラム (source program). ソース・プログラム・プロセッサー、およびコンパイラーへの入力となるプログラム。

総称キー (generic key). キー・クラスを識別する文字ストリング。そのストリングで始まるキーはすべて、そのクラスのメンバーである。たとえば、 'ABCD'、'ABCE'、および 'ABDF'、という記録済みキーは、すべて総称キー 'A' および 'AB' で識別されるクラスのメンバーであり、最初の 2 つは、'ABC' というクラスのメンバーでもある。そして、これら 3 つの記録済みキーは、それぞれ 'ABCD'、'ABCE'、'ABDF' というクラスの固有のメンバーであると見なすことができる。

総称記述子 (generic descriptor). GENERIC 属性内で使用する記述子。

総称名 (generic name). 入り口名ファミリーの名前。総称名への参照は、呼び出し点にある引数リスト内の引数の属性に一致するパラメーター記述子を持った入り口名によって置き換えられる。

相対仮想起点 (relative virtual origin (RVO)). 配列の実際の原点から配列の仮想原点を引いたもの。

挿入演算子 (infix operator). 2 つのオペランド間にある演算子。

挿入点文字 (insertion point character). 関連データを文字ストリングへ割り当てるときに、指示位置に挿入さ

れるピクチャー指定文字。入力のときに P フォーマット項目内で使用される挿入文字は、検査の目的で用いられる。

添え字 (subscript). 配列の次元内の位置を指定するための要素式。添え字がアスタリスクであれば、次元のすべてのエレメントを指定する。

添え字リスト (subscript list). 括弧に入れられた、1 つまたは複数の添え字のリスト。配列のおのおのの次元に対して 1 つの添え字が対応する。これらによって配列の単一エレメントまたはクロスセクションを一意的に識別する。

属性分配 (factoring). 1 つまたは複数の属性を、DECLARE ステートメント内の括弧で囲まれた名前リストに対して適用して、複数の名前に共通する属性を反復する必要をなくすこと。

[タ行]

ターゲット (target). データ項目 (ソース) が変換される属性。

ターゲット参照 (target reference). 受取側変数 (または受取側変数の一部) を指定する参照。

ターゲット変数 (target variable). 値が割り当てられる変数。

大構造 (major structure). レベル番号 1 を指定して宣言された名前を持つ構造。

代替属性 (alternative attribute). 属性グループから選択するファイル記述属性。何も指定しないと、デフォルトがとられる。追加属性 (*additive attribute*) と対比。

タイプ (type). データの世代、値、または項目に対して適用される一連のデータ属性とストレージ属性。

多重宣言 (multiple declaration). 同一ブロックに対して内部であり、別の修飾を持たない同一 ID の複数宣言。同一 ID の複数外部宣言。

タスク (task). 単一の制御の流れによる 1 つまたは複数のプロシージャーの実行。

タスクの生成 (attachment of a task). 呼び込まれたプロシージャー (およびこれが呼び出すプロシージャー) を、呼び出しプロシージャーの実行と一緒に、非同期で実行するために、プロシージャーを呼び出して別に制御の流れを確立すること。

タスク変数 (task variable). TASK 属性を持ち、その値がタスクの相対優先順位を示す変数。

タスク名 (task name). タスク変数を参照するのに使用される ID。

単純ステートメント (simple statement). IF、ON、WHEN、および OTHERWISE 以外のステートメント。

単純パラメーター (simple parameter). ストレージ・クラス属性が指定されていないパラメーター。単純パラメーターはどのストレージ・クラスの引数も表すことができるが、被制御引数の現行世代だけを表すことができる。

ダンプ (dump). エラーの原因のトレースなどの、プログラムが使用するストレージの一部または全部、または他のプログラム情報の印刷出力。

調節可能エクステント (adjustable extent). 関連変数の世代によって異なることのある境界 (配列の)、長さ (ストリングの)、またはサイズ (区域の)。調節可能エクステントは、世代ごとに別々に評価される式またはアスタリスク (ただし、基底付き変数の場合は REFER オプション) で指定される。静的変数に使用することはできない。

追加属性 (additive attribute). デフォルトを持たず、必要であれば明示的に述べるか、または、明示的に述べられた別の属性で暗黙指定しなければならないファイル記述属性。代替属性 (*alternative attribute*) と対比。

データ (data). 処理に適合した形式の情報または値の表現。

データ項目 (data item). 単一の名前付きデータ単位。

データ指定 (data specification). 伝送モード (DATA、LIST、または EDIT) を指示し、さらにデータ・リストと、編集ディレクティブ・モードの場合はフォーマット・リストを含む、ストリーム指向伝送ステートメントの一部分。

データ集合 (data aggregate). 異なったデータ項目の集まりであるデータ項目。

データ属性 (data attribute). FIXED BINARY などの、データ項目が表すデータのタイプを指定するキーワード。

データ伝送 (data transmission). データ・セットからプログラムへ、およびその逆に、データを転送すること。

データ・ストリーム (data stream). ストリーム指向伝送でデータ・セットから、またはデータ・セットへ、文字形式のデータ・エレメントの連続ストリームとして転送されるデータ。

データ・セット (data set). 単一のファイル名の参照によってアクセスすることができる、プログラムの外部にあるデータの集まり。参照されることが可能な装置。

データ・タイプ (data type). 一連のデータ属性。

データ・ディレクティブ伝送 (data-directed transmission). データを伝送するための、ストリーム指向伝送のタイプ。代入ステートメントに似ていて、`name = constant` の形式をとる。

データ・リスト (data list). ストリーム指向伝送における、GET および PUT ステートメント内で使用するデータ項目を括弧で囲んだリスト。フォーマット・リスト *(format list)* と対比。

定義された変数 (defined variable). 指定された基底付き変数用の一部または全部のストレージに関連付けられる変数。

定数 (constant). 名前が付いておらず、変更できない値を持った、算術またはストリング・データ項目。VALUE 属性を指定して宣言された ID。 FILE 属性または ENTRY 属性を指定し、VARIABLE 属性を指定しないで宣言された ID。

定数参照 (constant reference). 対象として定数を持つ参照値。

デバッグ (debugging). プログラムからバグを除去する処理。

デフォルト (default). 指定がされていないときに、とられる値、属性、またはオプション。

同期 (synchronous). プログラムの順次実行での単一の制御の流れ。

特別言語文字 (extralingual character). 英数字にも特殊文字にも分類されない文字 ($、@、および # など)。このグループには、NAMES コンパイラー・オプションで指定された文字も含まれる。

[ナ行]

内部プロシージャー (internal procedure). ブロックの中に組み込まれている別のプロシージャー。*外部プロシージャー (external procedure)* と対比。

内部ブロック (internal block). ブロックの中に組み込まれている別のブロック。

内部名 (internal name). 名前が宣言されたブロック内のみで認識されている名前、またそのブロック内に入っているブロックの中でも認識されている可能性もある名前。

入出力 (input/output). 補助メディアと主記憶装置との間でデータを転送すること。

認識された (名前に関する用語) (known (applied to a name)). 宣言された意味で認識されること。名前は、その有効範囲内で認識される。

ヌル・ステートメント (null statement). セミコロン記号 (;) のみの入ったステートメント。これは、何も処置はとられないことを示す。

ヌル・ストリング (null string). 長さゼロの文字ストリング、漢字ストリング、またはビット・ストリング。

ヌル・ロケーター値 (null locator value). 内部記憶域内のどの位置も識別できない特殊ロケーター値。これは、現在ロケーター変数がデータの世代を識別できないことを示すのに役立つ。

ネスト (nesting). 次のものの発生。

- ブロック内にある別のブロック。
- グループ内にある別のグループ。
- THEN 文節または ELSE 文節内の IF ステートメント。
- 関数参照の引数としての関数参照。
- FORMAT ステートメントのフォーマット・リスト内のリモート・フォーマット項目。
- パラメーター記述子リスト内の別のパラメーター記述子リスト。
- 1 つまたは複数の属性が分配されている括弧で囲まれた名前リスト内の属性の指定。

[ハ行]

配列 (array). 同じ属性を持ち、1 つまたは複数の次元別にグループ分けされた 1 つまたは複数のデータ・エレメントに、名前を付けて順番に並べた集合体。

配列式 (array expression). 評価されると値の配列が生成される式。

配列の共用体 (union of arrays). DIMENSION 属性を持った共用体。

配列のクロス・セクション (cross section of an array). 配列の少なくとも 1 つの次元のエクステントで表すことのできるエレメント。配列参照内に添え字の代わりにアスタリスクがあれば、それはその次元のエクステント全体を表す。

配列の構造体 (structure of arrays). 次元属性を持つ構造体。

配列変数 (array variable). 同じ属性を持っていなければならないデータ項目の集合を表す変数。*構造変数 (structure variable)* と対比。

パック 10 進 (packed decimal). 固定小数点 10 進データ項目の内部表現。

パッケージ定数 (package constant). PACKAGE ステートメントのラベル接頭語。

バッファー (buffer). レコードが入力時に読み込まれ、レコードが出力時に書き出される、入出力操作に使用する中間記憶域。

パラメーター (parameter). PROCEDURE ステートメントの後に続くパラメーター・リスト中の名前。そのプロシージャーが呼び出されれば、渡される引数を指定する。

パラメーター記述子 (parameter descriptor). ENTRY 属性指定内でパラメーター用に指定される一連の属性。

パラメーター記述子リスト (parameter descriptor list). ENTRY 属性指定内のすべてのパラメーター記述子のリスト。

パラメーター・リスト (parameter list). コンマで区切られ、プロシージャー・ステートメント内のキーワード PROCEDURE の後に続くか、または ENTRY ステートメント内のキーワード ENTRY の後に続く、括弧で囲まれた 1 つまたは複数のパラメーターのリスト。このリストは、呼び出し時に渡される引数リストと対応する。

範囲 (デフォルト指定の) (range (of a default specification)). DEFAULT ステートメント内の属性を適用される ID またはパラメーター記述子のどちらか、またはこの両方のセット。

反復 DO グループ (iterative do-group). 制御変数または WHILE や UNTIL オプション、またはこの両方を指定した DO ステートメントを持つ DO グループ。

反復因数 (iteration factor). INITIAL 属性指定において、特定の値を使って初期化されることになっている配列の連続エレメント数を指定するための式。フォーマット・リストにおける、特定のフォーマット項目またはフォーマット項目のリストを連続して使用する回数を指定するための式。

反復因数 (repetition factor). 以下のものを指定する、括弧に入れられた符号なし整定数。

1. 後続するストリング定数を繰り返す回数。
2. 後続するピクチャー文字を繰り返す回数。

反復指定 (repetitive specification). 1 つまたは複数のデータ項目伝送の被制御反復を指定するためのデータ・リストの 1 エレメントであり、通常は配列と同時に使用する。

非アクティブ (deactivated). ある ID の値で、ソース・プログラム・テキスト内のプリプロセッサー ID を置き換えることができない状態。*アクティブ (active)* と対比。

比較演算子 (comparison operator). 関係内の項目を相互に比較するよう指示するための算術、ストリング・ロケーター、または論理関係で使用される演算子。比較演算子は次のとおり。

= (に等しい)
> (より大)
< (より小)
>= (より大か等しい)
<= (より小か等しい)
¬= (等しくない)
¬> (より大ではない)
¬< (より小ではない)

引数 (argument). サブルーチンまたは機能の呼び出しの一部である引数リスト内にある式。

引数リスト (argument list). コンマで区切られ、入り口名定数、入り口名変数、総称名、または組み込み関数名に続く、括弧で囲まれたゼロまたはそれ以上の引数のリスト。そのリストは、エントリー・ポイントのパラメーター・リストである。

ピクチャー指定 (picture specification). PICTURE 属性を指定した宣言内で、または P フォーマット項目内でピクチャー文字を使用して宣言されたデータ項目。

ピクチャー指定文字 (picture specification character). ピクチャー指定で使用できるすべての文字。

ピクチャー・データ (picture data). 文字形式で表された、数値データ、文字データ、またはそれらの混合。

被制御ストレージ割り振り (controlled storage allocation). 被制御変数用のストレージの割り振り。

被制御パラメーター (controlled parameter). DECLARE ステートメント内で CONTROLLED 属性を指定されるパラメーター。これは、CONTROLLED 属性を持った引数としか関連付けることはできない。

被制御変数 (controlled variable). 現行世代にだけアクセスすることができ、 ALLOCATE と FREE ステートメントによって割り振りと解放が制御される変数。

ビット (bit). 0 または 1。コンピューター・ストレージの最小スペース量。

ビット値 (bit value). ビット・タイプを表す値。

ビット比較 (bit comparison). 2 進数字を、左から右へビットごとに比較すること。「*算術比較 (arithmetic comparison)*」、「*文字比較 (character comparison)*」も参照。

ビット・ストリング (bit string). ゼロ以上のビットで構成されたストリング。

ビット・ストリング演算子 (bit string operators). 論理演算子 NOT と 排他 OR (¬)、 AND (&)、および OR (|)。

ビット・ストリング定数 (bit string constant). 囲まれていて、接尾部 B が直後に付いた 2 進数字の連なり。*文字定数 (character constant)* と対比。単一引用符に囲まれ、後に接尾部 B4 が付いた 16 進数字の連なり。

非同期操作 (asynchronous operation). ステートメントの実行と、入出力操作が並行して行われること。各種タスクに複数の制御の流れを使った、プロシージャーの並行実行。

評価 (evaluation). 単一の値、値の配列、または値の構造化値へ式を換算すること。

標準システム処置 (standard system action). 使用可能な条件のための ON ユニットがないときにその条件が発生した場合にとられる、言語で指定された処置。

標準デフォルト (値) (standard default). 属性またはオプションの指定がなく、適用できる DEFAULT ステートメントがない場合の、代替属性または代替オプション。

標準ファイル (standard file). GET ステートメントや PUT ステートメントで FILE オプションまたは STRING オプションがない場合に、PL/I が想定するファイル。 SYSIN が標準入力ファイルであり、SYSPRINT が標準出力ファイルである。

非連結ストレージ (nonconnected storage). 非結合データ項目が占有するストレージ。たとえば、継承次元を持つ介在配列や構造体は、非結合ストレージ内にある。

ファイル (file). プログラムにおいて、単数または複数のデータ・セットを名前付きで表現したもの。ファイルは、オープンするごとに、単数または複数のデータ・セットに関連付けられる。

ファイル記述属性 (file description attribute). 各ファイル定数の個々の特性を記述したキーワード。「*代替属性 (alternative attribute)*」と「*追加属性 (additive attribute)*」も参照。

ファイル式 (file expression). 評価されるとファイル・タイプを生じる式。

ファイル定数 (file constant). FILE 属性を指定し、VARIABLE 属性を指定しないで宣言された名前。

ファイル変数 (file variable). ファイル定数を割り当てることのできる変数。この場合、ファイルは、FILE 属性と VARIABLE 属性を持っていなければならず、ファイル記述属性を持っていることはできない。

ファイル名 (file name). ファイル用に宣言された名前。

フィールド (データ・ストリーム中の) (field (in the data stream)). 単一データまたはスペーシング・フォーマット項目によって、幅 (文字数) を定義されるデータ・ストリームの部分。

フィールド (ピクチャー指定の) (field (of a picture specification)). 任意の文字ストリング・ピクチャー指定、または固定小数点数を記述した数字ピクチャー指定の部分 (または全部)。

フォーマット (format). ストリーム内のデータ項目の表現法を記述したり (データ・フォーマット項目)、またはストリーム内のデータ項目の個々の位置決めを記述する (制御フォーマット項目) ために編集ディレクティブ・データ伝送内で使用される仕様。

フォーマット定数 (format constant). FORMAT ステートメントでのラベル接頭部。

フォーマット・データ (format data). FORMAT 属性を指定された変数。

フォーマット・ラベル (format label). FORMAT ステートメントでのラベル接頭部。

フォーマット・リスト (format list). ストリーム指向伝送における外部メディアでのデータ項目のフォーマットを指定したリスト。データ・リスト *(data list)* と対比。

複合ステートメント (compound statement). 他のステートメントが含まれているステートメント。 PL/I では、IF、ON、OTHERWISE、および WHEN だけが、複合ステートメントである。「*ステートメント本体 (statement body)*」を参照。

複合ネストの深さ (combined nesting depth). プログラム内の PROCEDURE/BEGIN/ON、DO、SELECT、および IF...THEN...ELSE によるネストのレベルをカウントして決定される、最も深いネスト・レベル。

複素数データ (complex data). おのおのの項目が実数部と虚数部で構成された算術データ。

含まれているブロック、宣言、またはソース・テキスト (contained block, declaration, or source text). 開始、プロシージャー、またはパッケージのブロック内のすべてのブロック、プロシージャー、ステートメント、宣言、またはソース・テキスト。パッケージ、プロシージャー、および BEGIN ステートメントとそれに対応する END ステートメント全体は、ブロック内には含まれていない。

符号および通貨記号 (sign and currency symbol characters). ピクチャー指定文字。 S、+、-、および $ (または < と > で囲まれたその他の通貨記号)。

浮動小数点定数 (floating-point constant). 「*算術定数 (arithmetic constant)*」を参照。

部分修飾名 (partially-qualified name). 不完全な修飾名。これには名前が参照する構造メンバーまたは共用体メンバーより上の階層順序内にある名前のうちの全部ではない 1 つまたは複数の名前、およびそれ自身のメンバー名が含まれる。

プリプロセッサー (preprocessor). コンパイルを実行する前に、ソース・プログラムを調べるためのプログラム。

プリプロセッサー・ステートメント (preprocessor statement). プリプロセッサーがとる処置を指定するために、ソース・プログラム内に入れる特殊ステートメント。これは、プリプロセッサーによって検出されると実行される。

プログラム (program). 1 つまたは複数の外部プロシージャーまたはパッケージのセット。外部プロシージャーのうちの 1 つは、PROCEDURE ステートメント内に OPTIONS(MAIN) 指定を持っていなければならない。

プログラム制御データ (program control data). PL/I プログラムの処理を制御するのに使用するための区域、ロケーター、ラベル、フォーマット、項目、およびファイルのデータ。

プロシージャー (procedure). PROCEDURE ステートメントと END ステートメントで区切られたステートメントの集まり。プロシージャーとはプログラムまたはプログラムの一部であり、名前の有効範囲を区切り、そのプロシージャーまたは入り口名の 1 つへの参照によって活動化される。「*外部プロシージャー (external procedure)*」および「*内部プロシージャー (internal procedure)*」も参照。

プロシージャー参照 (procedure reference). 入り口定数または入り口変数。この後に引数リストを続けることができる。プロシージャー参照は、CALL ステートメントや CALL オプションに入れることも、または関数参照として使用することもできる。

ブロック (block). その中で宣言された名前の有効範囲と、その名前用のストレージ割り振りを指定する、1 つの単位として処理される一連のステートメント。ブロックとしては、パッケージ、プロシージャー、または開始ブロックのいずれもありえる。

プロローグ (prologue). ブロックの起動時に自動的に生じる処理。

分離文字 (separator). 「*区切り文字 (delimiter)*」を参照。

編集ディレクティブ伝送 (edit-directed transmission). データが連続した文字ストリームとして中にあり、関連データ・リストに対して行いたい編集を指定するにはフォーマット・リストを必要とするようなタイプのストリーム指向伝送。

変数 (variable). データを参照するのに使用され、値を割り当てる対象となりうる名前付きのエンティティー。その属性は一定のままであるが、場合に応じてそれぞれ異なる値を参照することができる。

変数参照 (variable reference). 変数全体またはその一部を指定する参照。

ポインター (pointer). ストレージ内の位置を識別するための変数のタイプ。

ポインター値 (pointer value). ポインター型を識別する値。

ポインター変数 (pointer variable). ポインター値が入った POINTER 属性を持ったロケーター変数。

[マ行]

マルチプログラミング (multiprogramming). 単一の処理装置を使って、複数のプログラムを並行して処理するのに、計算機システムを使用すること。

マルチプロセッシング (multiprocessing). 複数のプログラムを同時に実行するために、複数の処理装置を備えた計算機システムを使用すること。

未定義 (undefined). ユーザーが行ってはならないことを示す。未定義機能が使用されると、PL/I 製品の個々のインプリメンテーションによって違った結果が出る可能性がある。このような場合、アプリケーション・プログラムはエラーとなる。

名 (name). 変数や定数にユーザーが与える ID。コンテキスト中に現れる、キーワードではない ID。場合によっては、ユーザー定義名とも呼ぶ。

明示宣言 (explicit declaration). ラベル接頭部として DECLARE ステートメント内、またはパラメーター・リスト内に ID (名前) を出すこと。*暗黙宣言 (implicit declaration)* と対比。

メンバー (member). 構造体または共用体の中の、構造体、共用体、あるいはエレメントの名前。ライブラリー内のデータ・セット。

モード (算術データの) (mode (of arithmetic data)). 算術データの属性。これは、*実数* または *複素数* のどちらかである。

文字ストリング定数 (character string constant). 単一引用符で囲まれる一連の文字。たとえば、'Shakespeare''s 'Hamlet:" など。

文字ストリング・ピクチャー・データ (character string picture data). 文字値だけを持ったピクチャー・データ。このタイプのピクチャー・データは、少なくとも 1 つの A または X ピクチャー指定文字を持っていなければならない。*数値ピクチャー・データ (numeric picture data)* と対比。

文字セット (character set). あらかじめ決められた文字の集まり。 *ASCII* と *EBCDIC* を参照。

文字比較 (character comparison). 照合順序に従って、左から右へ文字単位で行われる比較。「*算術比較 (arithmetic comparison)*」、「*ビット比較 (bit comparison)*」も参照。

戻り値 (returned value). 関数プロシージャーから戻される値。

問題状態プログラム (problem-state program). オペレーティング・システムの問題プログラム状態内で稼働するプログラム。これには、入出力指示やその他の特権命令は入らない。

問題データ (problem data). コード化された算術データ、ビット・データ、文字データ、グラフィック・データ、およびピクチャー・データ。

[ヤ行]

有効範囲 (条件接頭語の) (scope (of a condition prefix)). 全体にわたって特定の条件接頭語が適用される、プログラムの部分。

有効範囲 (宣言の) (scope (of a declaration or name)). 全体にわたって特定名が認識されているプログラムの部分。

優先度 (priority). タスクに関連する値で、他のタスクに対するそのタスクの優先順位 (指名順位) を指定する。

要素式 (element expression). 評価されるとエレメント値を生じる式。

要素変数 (element variable). エレメントを表す変数。スカラー変数。

呼び出されたプロシージャー (invoked procedure). 活動化されているプロシージャー。

呼び出し (call). CALL ステートメントまたは CALL オプションを使用してサブルーチンを呼び出すこと。

呼び出し側ブロック (invoking block). プロシージャーを活動化するブロック。

呼び出し点 (point of invocation). 呼び込まれたプロシージャーへの参照が現れる呼び込み側ブロック内の地点。

予備ファイル (spill file). 一時作業ファイルとして使われる SYSUT1 という名前のデータ・セット。

[ラ行]

ライブラリー (library). メンバーと呼ばれるその他のデータ・セットを保管するのに使用できる MVS 区分データ・セット、または CMS MACLIB のこと。

ラベル (label). ステートメントの接頭部として付く名前。 PROCEDURE ステートメント上の名前を、入り口定数と呼び、FORMAT ステートメント上の名前を、フォーマット定数と呼ぶ。その他の種類のステートメント上の名前を、ラベル定数と呼ぶ。 LABEL 属性を持つデータ項目。

ラベル接頭部 (label prefix). ステートメントの接頭部として付けられたラベル。

ラベル定数 (label constant). ステートメント (PROCEDURE、ENTRY、FORMAT、または PACKAGE を除く) のラベル接頭語として書き込まれる名前。実行時に、そのラベル接頭語が参照されれば、そのステートメントにプログラムの制御を渡すことができる。

ラベル変数 (label variable). LABEL 属性を指定して宣言された変数。その値は、プログラム内でのラベル定数である。

ラベル・データ (label data). ラベル定数または、ラベル変数の値。

リスト・ディレクティブ (list-directed). ストリーム内のデータがブランクやコンマで区切られた定数になり、フォーマット設定が自動的に行われるタイプのストリーム指向伝送。

リモート・フォーマット項目 (remote format item). R という文字の後に FORMAT ステートメントのラベル (括弧に囲まれている) のあるもの。フォーマット指定ステートメントは、転送するデータのフォーマットを制御するために、編集ディレクティブ・データ伝送ステートメントにより使用する。

領域 (area). 基底付き変数を割り振ることのできる、ストレージ中の部分。

ループ (loop). 繰り返し実行される一連の命令。

レコード (record). レコード単位入力または出力の操作における、伝送の論理単位。 1 つまたは複数の関連データ項目の集まり。これらの項目には通常、それぞれ異なったデータ属性があり、また通常は構造体か共用体の宣言で記述される。

レコード単位データ伝送 (record-oriented data transmission). 別々のレコードの形式でデータを伝送すること。*ストリーム方式のデータ伝送 (stream data transmission)* と対比。

レベル 1 変数 (level-one variable). 大構造体または共用体の名前。構造体または共用体の中に含まれていない、添え字なし変数。

レベル番号 (level number). DECLARE ステートメント中の名前の前に付く番号で、構造体名の階層内のその相対位置を指定するもの。

連結 (concatenation). 2 つのストリングを指定順に結合し、元の 2 つのストリングの合計長に等しい長さを持った 1 つのストリングを作成する操作。これは、演算子 || で指定する。

連結参照 (connected reference). 連結ストレージに対する参照。プログラムを実行するには、ストレージが連結されていることが明らかでなければならない。

連結集合 (connected aggregate). エレメントが、間にデータ項目の入らない連続したストレージを占有する配列または構造体。*非連結集合 (nonconnected aggregate))* と対比。

連結ストレージ (connected storage). 単一名を使って参照することができる諸項目の非中断かつ線形の連なりの主記憶域。

ロケーター (locator). 変数のアドレスまたはその記述子を保持する制御ブロック。

ロケーター値 (locator value). ストレージ・アドレスを識別する値か、またはストレージ・アドレスを識別するのに使用できる値。

ロケーター修飾 (locator qualification). 基底付き変数への参照において、その参照が参照している基底付き変数の世代を指定するために、基底付き変数の左側に矢印で接続されているロケーター変数または関数参照。これは、暗黙参照であることもある。

ロケーター変数 (locator variable). 変数またはバッファーの主記憶域内の位置を識別する値を持った変数。これは、POINTER 属性または OFFSET 属性を持つ。

ロケーター/記述子. その後に記述子の付いたロケーター。ロケーターは、記述子のアドレスではなく、変数のアドレスを保持する。

ロック・レコード (locked record). EXCLUSIVE DIRECT UPDATE ファイル内のレコードであって、1 つのタスクにだけしか使用することはできず、そのレコードを使用しているタスクによって解放されるまで、他のタスクからアクセスできないレコード。

論理演算子 (logical operators). ビット・ストリング演算子の NOT や排他 OR (¬)、 AND (&)、および OR (|)。

論理レベル (構造体または共用体メンバーの) (logical level (of a structure or union member)). 全レベル番号が直接順序になっているとき (あるレベル番号から次のレベル番号までの増分が 1 のとき) に、レベル番号で示される深さ。

[ワ行]

割り当て (assignment). 値を変数に与える処理。

割り込み (interrupt). 条件やアテンションの発生の結果として、プログラムの制御の流れを変更すること。

割り振られた変数 (allocated variable). 主記憶域が関連付けられ解放されない変数。

割り振り (allocation). 変数用の主記憶域の予約。割り振られた変数の世代。 PL/I ファイルを、システム・データ・セット、装置、またはファイルに関連付けること。

[数字]

1 次エントリー・ポイント (primary entry point). PROCEDURE ステートメントのラベル・リスト内の任意の名前によって識別されるエントリー・ポイント。

10 進 (decimal). 0 から 9 までの数字を使った数体系。

10 進固定小数点値 (decimal fixed-point value). 小数点の想定位置を持つ 10 進数の連なりで構成される有理数。 *2 進固定小数点値 (binary fixed-point value)* と対比。

10 進固定小数点定数 (decimal fixed-point constant). 1 つまたは複数の 10 進数 (および任意で小数点を付けたもの) から成る定数。

10 進ピクチャー文字 (decimal digit picture character). ピクチャー指定文字 9 のこと。

10 進ピクチャー・データ (decimal picture data). 「*数値ピクチャー・データ (numeric picture data)*」を参照。

10 進浮動小数点値 (decimal floating-point value). 10 進小数部と考えることができる仮数形式の実数、および 10 を底とする整数のべき乗と考えることができる指数の近似値。 *2 進浮動小数点値 (binary floating-point value)* と対比。

10 進浮動小数点定数 (decimal floating-point constant). 10 進固定小数点定数から成る仮数と、3 桁を超えないオプショナルの符号付き整定数が後に付いた文字 E から成る指数とで構成される値。

16 進 (hex). 「*16 進数字 (hexadecimal digit)*」を参照。

16 進数 (hexadecimal). 16 の基数を持った数体系。有効数は 0 から 9 の数字と、A は 10 を、F は 15 を表す A から F までの文字。

16 進数字 (hexadecimal digit). 0 から 9 までと A から F までの数字のいずれか。A から F までは、それぞれ 10 進数値の 10 から 15 までを表す。

2 次エントリー・ポイント (secondary entry point). 入り口ステートメントのラベル・リスト内の任意の名前によって識別されるエントリー・ポイント。

2 進固定小数点値 (binary fixed-point value). 2 進数字で構成され、オプショナルの 2 進小数点とオプショナルの符号を持った整数。 *10 進固定小数点値 (decimal fixed-point value)* と対比。

2 進数 (binary). 0 と 1 が唯一の数表示である数体系。

2 進数字 (binary digit). 「*ビット (bit)*」を参照。

2 進浮動小数点値 (binary floating-point value). 2 進小数部と見なせる仮数と、2 の基数に対する整数指数と見なせる指数の形式の実数の近似値。*10 進浮動小数点値 (decimal floating-point value)* と対比。

A

ASCII. 情報交換用米国標準コード (American National Standard Code for Information Interchange)。

attribute. 表明された特性を記述するのに名前と関連付けた記述特性。式の計算の結果の特性を説明するために用いられる記述特性。

D

DBCS. 文字セットにおいて、それぞれの文字は 2 つの連続するバイトで表される。

DO グループ (do-group). DO ステートメントで区切られ、それに対応する END ステートメントで終了する、制御目的に使用される一連のステートメント。*ブロック (block)* と対比。

DO ループ (do-loop). 「*反復 DO グループ (iterative do-group)*」を参照。

E

EBCDIC. 拡張 2 進化 10 進コード (Extended Binary-Coded Decimal Interchange Code)。 8 ビットのコード化文字からなるコード化文字セット。

end-of-step メッセージ (end-of-step message). ジョブ制御ステートメントとジョブ・スケジューラー・メッセ

ージのリストに続いており、各ステップの成功または失敗を示す戻りコードを含むメッセージ。

I

ID (identifier). コメントや定数内に入ることがなく、前後に区切り文字をともなう文字のストリング。 ID の先頭文字は、26 個の英字、または特別言語文字 (ある場合) でなければならない。その他の文字がある場合には、拡張英字、数字、区切り文字を追加して入れることができる。

IEEE. 米国電気電子学会 (Institute of Electrical and Electronics Engineers)。

O

ON ステートメント処置 (ON-statement action). ある条件が生じたときに処置を取れるよう、条件に対して明示的に設定された処置法。プログラムの制御の流れ内で ON ステートメントが見つかると、とられる処置で、その条件に対する処置が設定される。この処置は、ON ユニットが設定されたままであるか、RESIGNAL ステートメントで再設定されて条件が生じたときにとられる。*暗黙の処置 (implicit action)* と対比。

ON ユニット (ON-unit). 該当する条件が起きたときに、とられるよう指定された処置。

P

PL/I プロンプター (PL/I prompter). PL/I コマンドのコマンド・プロセッサー・プログラムで、オペランドを調べ、コンパイラーに必要なデータ・セットを割り振る。

PL/I 文字セット (PL/I character set). PL/I のプログラム・エレメントを表現するために定義されている文字セット。

R

REFER オブジェクト (REFER object). REFER オプション中の変数。メンバーの現行境界、長さ、またはサイズを保持しているか、あるいは保持する予定のもの。REFER オブジェクトは、同一構造体または共用体のメンバーでなければならない。これは、ロケーター修飾したり添え字を付けてはならず、また REFER オプションを持ったメンバーの前になければならない。

REFER 式 (REFER expression). REFER というキーワードの前に付いた式。この式は、REFER オプションを含む基底付き変数が、ALLOCATE ステートメントまたは LOCATE ステートメントのいずれかによって割り振られるときの境界、長さ、またはサイズとして使用される。

reference. 明示宣言を生じることになる 1 つのコンテキスト内以外の名前の出現。

RETURNS 記述子 (RETURNS descriptor). RETURNS 属性内と、PROCEDURE および ENTRY ステートメントの RETURNS オプション内で使用する記述子。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

アスタリスク
  算術演算での使用 60
  説明 346
  添え字としての使用 184
  ID として 14
アプリケーション 93
暗黙
  解放
    基底付き変数の場合 253
    被制御変数の場合 244
  宣言 162
  ファイルのオープン 288
暗黙アクション 357
暗黙にロケーター修飾された参照 249
暗黙の日付
  比較 47
  割り当て 47
異常終了
  プログラム 95
  プロシージャー 106
位置指定モード 301
移動モード 301
入り口参照の呼び出し 133
入り口参照呼び出し 133
入り口データ
  参照の呼び出し 133
  説明 120
  総称入り口宣言 130
  属性
    分類 28, 29
    ENTRY 122
    GENERIC 130
    LIMITED 129
    LIST 125
    OPTIONAL 124
  直接入り口宣言 120
  定数 120
  変数 121
  GENERIC 130
入り口定数 120
引用符 (単一または二重)、ストリング・データを囲む 24
英字 10
英数字 10
エクステント (境界の) 182
エレメント
  式 56
  スカラー 23
  データ 23
  パラメーター 117
  プログラム 9
  変数 23
  割り当て 204
  DBCS の場合 20
  SBCS の場合 13
演算
  組み合わせ 71
  クラス 59
  算術 60
  接頭演算子
    説明 59
    例 74
  挿入演算子 59
  比較
    説明 68
    例 70
  ビット 67
  ポインター 59
  ポインター・サポート拡張の使用 59
  連結 70
  論理 67
演算子
  算術
    使用法 15
    説明 60
  使用法 14
  ストリング 15
  挿入演算子
    説明 74
    ポインター式との併用 60
  比較 15
  ビット 15
  論理 15
演算子の優先順位 72
演算式
  クラス 59
  説明 59
  データ・タイプの制限 59
  定義 55
  変換規則 59
  例 60
エントリー・ポイント 99
オーバーパンチ・ピクチャー文字、I 353
オーバーパンチ・ピクチャー文字、R 353
オーバーパンチ・ピクチャー文字、T 353
応答テキスト 545
大文字小文字の区別 12
オプション
  値の指定 179
  データ伝送ステートメントの 297, 305
  ASSEMBLER 138
  DESCRIPTORS オプション 178
  DLLINTERNAL 141
  EXPORTS 97
  FETCHABLE 141
  GRAPHIC 43
  GRAPHIC ENVIRONMENT 43
  NORESCAN 558
  OPTIONS 135
  RANGE 177
  RECURSIVE 107
  REPEAT 212
  REPLY 209
  RESCAN 558
  RESERVES 97
  RETURNS 144
  SCAN 558
  SET 110
  SNAP 360
  SYSTEM 360
  TITLE 110
オフセット変数 247
オフセット・データ 256
オペランド
  定義 55
  変換 63

[カ行]

介在添え字 191
開始ブロック
  活動化 119
  終了 119
  説明 119
  例 119
外部プロシージャー
  説明 99
  動的ロード 109
各種演算の組み合わせ 71
確立された条件 357
確立された処置 360

貸方 (CR) ピクチャー文字 353
下線 (_) 文字 24
仮想定数 32
括弧 14
活動化
開始ブロック 119
プログラム 94
プロシージャー 105
ブロック 95
仮引数
規則 117
説明 117
属性の導出 117
借方 (DB) ピクチャー文字 353
漢字ストリング定数 43
関数
組み込み 115
制限 113
説明 113
定義 113
プログラマー作成の 115
戻り 134
例 114
キーワード
定義 13
キーワード・ステートメント 18
記号、複合の 12
疑似変数
説明 58
宣言 398
要約 411
呼び出し 399
例 58
ENTRYADDR 440
IMAG 453
ONCHAR 474
ONGSOURCE 477
ONSOURCE 479
ONWCHAR 480
ONWSOURCE 481
REAL 497
STRING 510
SUBSTR 511
TYPE 518
UNSPEC 521
記述子のリスト 123
規定境界 171
基底付きストレージ
組み込み関数 246
説明 238
BASED 属性の構文 245
基底付き変数
組み込み関数 246
説明 245, 250
リストの入出力 258
ALLOCATE ステートメント 251
基底付き変数 *(続き)*
FREE ステートメント 252
基本名 185
境界 182
単純パラメーター 102
被制御パラメーター 102
共用体
修飾名 187
説明 186
宣言 186
名前 186
配列のクロスセクション 192
例 186
レベル 186
UNION 属性、分類 29
共用体のレベル 186
区域
属性 29
データ 255
入出力 258
変数の伝送 294
割り当て 258
EMPTY 組み込み関数 439
IN オプション付きの ALLOCATE ステートメント 251
区切り文字 14
句読点のついた定数 24
組み合わせ配列、構造、および共用体 191
組み込み関数
基底付き変数 250
区域変数 258
算術、要約 401
条件処理、要約 403
序数 156
序数処理データ、要約 410
数学、要約 408
ストリング処理、要約 412
ストレージ制御、要約 411
整数演算、要約 407
精度処理、要約 410
宣言 398
データ変換の開始 78
定義 115
での数学関数の精度 401
入出力、要約 407
ヌル引数および 400
のカテゴリー 401
配列処理、要約 402
引数の集合 400
被制御変数 245
データ変換 78
日付/時刻、要約 404
浮動小数点演算、要約 407
浮動小数点の照会、要約 406
プリプロセッサー 547
組み込み関数 *(続き)*
プリプロセッサー用 547
呼び出し 399
ABS 415
ACOS 415
ACOSF 415
ADD 415
ADDR 416
ADDRDATA 417
ALL 417
ALLOCATE (ALLOC) 417
ALLOCATION (ALLOCN) 418
ALLOCSIZE 418
ANY 418
ASIN 418
ASINF 419
ATAN 419
ATAND 420
ATANF 420
ATANH 420
AUTOMATIC (AUTO) 421
AVAILABLEAREA 421
BINARY
データ変換 78
BINARY (BIN) 421
BINARYVALUE
序数での使用 156
ポインター式との併用 60
BINARYVALUE (BINVALUE) 422
BIT 422
データ変換 78
BITLOCATION (BITLOC) 422
BOOL 423
BYTE 423
CDS 423
CEIL 424
CENTERLEFT (CENTER) 424
CENTERRIGHT 425
CHAR 78
CHARACTER (CHAR) 426
CHARGRAPHIC (CHARG) 426
CHARVAL 427
CHECKSTG 428
COLLATE 428
COMPARE 428
COMPLEX (CPLX) 429
CONJG 430
COPY 430
COS 430
COSD 431
COSF 431
COSH 431
COUNT 431
CS 432
CURRENTSIZE (CSTG) 433
CURRENTSTORAGE 434

組み込み関数 *(続き)*
DATAFIELD 434
DATE 435
DATETIME 435
DAYS 435
DAYSTODATE 436
DAYSTOSECS 437
DECIMAL
データ変換 78
DECIMAL (DEC) 437
DIMENSION (DIM) 438
DIVIDE 438
EDIT 438
EMPTY 439
ENDFILE 440
ENTRYADDR 440
EPSILON 441
ERF 441
ERFC 441
EXP 441
EXPF 442
EXPONENT 442
FILEDDINT 442
FILEDDTEST 443
FILEDDWORD 443
FILEID 444
FILEOPEN 445
FILEREAD 445
FILESEEK 445
FILETELL 446
FILEWRITE 446
FIXED 446
データ変換 78
FLOAT 447
データ変換 78
FLOOR 447
GAMMA 447
GETENV 448
GRAPHIC 448
データ変換 78
HANDLE 449
HBOUND 449
HEX 450
HEXIMAGE 451
HIGH 451
HUGE 451
IAND 452
IEOR 452
IMAG 453
データ変換 78
INDEX 453
INOT 454
IOR 454
ISIGNED 455
ISLL 455
ISMAIN 456

組み込み関数 *(続き)*
ISRL 456
IUNSIGNED 457
LBOUND 457
LEFT 458
LENGTH 458
LINENO 459
LOCATION (LOC) 459
LOG 460
LOG10 461
LOG10F 461
LOG2 461
LOGF 460
LOGGAMMA 460
LOW 461
LOWER2 462
LOWERCASE 462
MAX 463
MAXEXP 463
MAXLENGTH 464
MEMINDEX 465
MEMSEARCH 466
MEMSEARCHR 466
MEMVERIFY 467
MEMVERIFYR 468
MIN 469
MINEXP 469
MOD 470
MPSTR 471
MULTIPLY 472
NULL 472
OFFSET 472
OFFSETADD 473
OFFSETDIFF 473
OFFSETSUBTRACT 473
OFFSETVALUE 474
OMITTED 474
ONCHAR 474
ONCODE 475
ONCONDCOND 475
ONCONDID 475
ONCOUNT 476
ONFILE 476
ONGSOURCE 477
ONKEY 477
ONLOC 478
ONSOURCE 478
ONSUBCODE 479
ONWCHAR 480
ONWSOURCE 480
ORDINALNAME 481
ORDINALPRED 481
ORDINALSUCC 482
PACKAGENAME 482
PAGENO 482
PICSPEC 482

組み込み関数 *(続き)*
PLACES 483
PLIRETV 488
POINTER (PTR) 490
POINTERADD
ポインター演算との併用 60
POINTERADD (PTRADD) 491
POINTERDIFF (PTRDIFF) 492
POINTERSUBTRACT
(PTRSUBTRACT) 492
POINTERVALUE
使用法 60
POINTERVALUE (PTRVALUE) 492
POLY 492
PRECISION
結果の計算 66
データ変換 78
PRECISION (PREC) 493
PRED 493
PRESENT 494
PROCEDURENAME
(PROCNAME) 494
PROD 494
PUTENV 495
RADIX 495
RAISE2 495
RANDOM 496
RANK 496
REAL 496
データ変換 78
REM 497
REPATTERN 497
REPEAT 498
REVERSE 499
RIGHT 499
ROUND 500
SAMEKEY 501
SCALE 501
SEARCH 501
SEARCHR 503
SECS 503
SECSTODATE 504
SECSTODAYS 505
SIGN 505
SIGNED 505
データ変換 78
SIN 506
SIND 506
SINF 506
SINH 506
SIZE 507
SOURCEFILE 508
SOURCELINE 508
SQRT 508
SQRTF 509
STORAGE (STG) 509

組み込み関数 *(続き)*
STRING 509
SUBSTR 511
SUBTRACT 512
SUCC 512
SUM 512
SYSNULL 512
SYSTEM 513
TALLY 513
TAN 514
TAND 514
TANF 514
TANH 514
THREADID 515
TIME 515
TINY 515
TRANSLATE 516
TRIM 517
TRUNC 517
TYPE 518
UNALLOCATED 518
UNSIGNED 519
データ変換 78
UNSPEC 519
UPPERCASE 521
VALID 522
VALIDDATE 522
VARGLIST 523
VARGSIZE 523
VERIFY 524
VERIFYR 524
WCHARVAL 525
WEEKDAY 525
WHIGH 526
WIDECHAR
データ変換 78
WIDECHAR (WCHAR) 526
WLOW 527
XMLCHAR 527
Y4DATE 528
Y4JULIAN 529
Y4YEAR 530
組み込み関数、その他
要約 409
組み込み関数および疑似変数の呼び出し 399
組み込み関数の宣言 398
組み込み疑似、要約 410
組み込みサブルーチン
宣言 398
定義 113
要約 414
呼び出し 399
PLIASCII 484
PLICANC 484
PLICKPT 485
組み込みサブルーチン *(続き)*
PLIDELETE 485
PLIDUMP 485
PLIEBCDIC 485
PLIFILL 486
PLIFREE 486
PLIMOVE 487
PLIOVER 487
PLIREST 488
PLIRETC 488
PLISAXA 488
PLISAXB 489
PLISRTA 489
PLISRTB 490
PLISRTC 490
PLISRTD 490
組み込みサブルーチンの呼び出し 399
組み込み名
組み込み関数との併用 115
サブルーチンとの併用 113
グラフィック定数
構文 42
ストリング 293
説明 42
比較演算 69
グラフィック・データ
定数 42
伝送 293
フォーマット項目 334
変換 90
GX (グラフィック 16 進) ストリング定数 43
グラフィック・データ、変換 (GRAPHIC) 448
グループ、ステートメントの 19
クロスセクション、配列の 184
計算条件
CONVERSION 371
FIXEDOVERFLOW 376
INVALIDOP 377
OVERFLOW 379
UNDERFLOW 387
ZERODIVIDE 387
計算データ
ストリング・データ 25
説明 24
属性 24
変換 78
計算データ・タイプ
ストリングの反復因数 40
ストリング・データ
グラフィック 42
説明 37
ワイド文字 44
BIT 属性 37
CHARACTER 属性 37
計算データ・タイプ *(続き)*
ストリング・データ *(続き)*
GRAPHIC 属性 37
NONVARYING 属性 38
VARYING 属性 38
VARYINGZ 属性 38
WIDECHAR 属性 37
属性 29
BINARY 属性と DECIMAL 属性 31
REAL 属性と COMPLEX 属性 31
結果、算術演算の
説明 62
特殊な場合 67
FLOAT オペランド 62
言語間通信
リンケージ
OPTLINK 142
SYSTEM 142
LINKAGE オプション 142
言語に固有のデフォルト
説明 176
定義 176
復元 181
コード化算術データ
構文 30
属性
省略語 30
タイプ 28, 29
変換ターゲット 82
10 進固定小数点 34
10 進浮動小数点 36
2 進固定小数点データ 33
2 進浮動小数点 35
BINARY 属性と DECIMAL 属性 31
FIXED および FLOAT 属性 31
PRECISION 属性 31
REAL 属性と COMPLEX 属性 31
工業規格 4
構造体
式 56
修飾 153
修飾名 187
宣言 185
属性 29
組織の指定 185
タイプ付き
説明 150
ハンドル 151
HANDLE 組み込み関数 151
定義 150, 185
名前
基本 185
共用体の 186
小 185
説明 185
大 185

構造体 *(続き)*
配列のクロスセクション 192
被制御 245
変数 189
メンバー・エレメント 186
レベル
共用体の 186
共用体の最高位番号 187
共用体の最大番号 187
構造の最高位番号 186
構造の最大番号 186
説明 185
割り当て 204, 205
DEFINE STRUCTURE ステートメント 150
LIKE 属性 188
構造体タイプ、定義 150
構造体または共用体配列のクロスセクション 192
構造体マッピング
説明 192
対を作る順序の規則 193
例 195
1 つの対のマッピングを行うための規則 194
UNALIGNED 属性による影響 195
構文、図、読み方 1
固定小数点
フォーマット項目
説明 332
ピクチャー・スケール因数の指定 355
10 進データ 34
2 進データ 33
コメント
説明 16
コロン記号 14
混合ストリング定数 43
混合データ 43
コンテキスト宣言 162
コンパイル単位 93
コンマ 14

［サ行］

再帰
自動変数に及ぼす影響 108
属性 100
定義 107
再帰が自動変数に及ぼす影響 108
再帰的プロシージャー
自動変数に及ぼす影響 108
説明 107
属性の指定 107
例 108
サイズ
単純パラメーター 102
被制御パラメーター 102
索引データ・セット 279
サブフィールド、数字データの 344
サブルーチン
組み込み 113
制限 112
定義 112
戻り 134
例 112
PROCEDURE ステートメントで識別する 99
サブルーチン、組み込み
呼び出し 399
リスト 414
PLIASCII 484
PLICANC 484
PLICKPT 485
PLIDELETE 485
PLIDUMP 485
PLIEBCDIC 485
PLIFILL 486
PLIFREE 486
PLIMOVE 487
PLIOVER 487
PLIREST 488
PLIRETC 488
PLISAXA 488
PLISAXB 489
PLISRTA 489
PLISRTB 490
PLISRTC 490
PLISRTD 490
算術演算
結果
説明 62
特殊な場合 67
FLOAT オペランド 62
RULES(ANS) のもとでの結果 63
説明 60
データ変換 61
算術演算子
使用法 15
説明 60
算術演算の結果、RULES (ANS) における 63
算術組み込み関数
要約 401
ABS 415
CEIL 424
COMPLEX 429
CONJG 430
FLOOR 447
IMAG 453
MAX 463
算術組み込み関数 *(続き)*
MIN 469
MOD 470
RANDOM 496
REAL 496
REM 497
ROUND 500
SIGN 505
TRUNC 517
算術データ
コード化 24
数値ピクチャー 24
算術ピクチャー指定
使用法 45
説明 39
算術文字データ
使用法 45
挿入編集文字 45
PICTURE データへの変換 85
参照
構文 55
説明 55
プリプロセッサー 540
ロケーター 248
式
値の割り当て 207
エレメント 56
計算の順序 57
構造体 56
構文 55
式の中間結果 65
スカラー 56
制限付き
組み込み関数の適用 76
説明 76
例 76
説明 55
ターゲットの 58
タイプ 56
配列 74
プリプロセッサー 540
命令形式
クラス 59
説明 59
定義 55
式と参照の計算順序 72
式の計算順序 57
式の中間結果
説明 58
例 65
自己定義データ (REFER オプション) 253
指数指定子 354
実行する際の制限事項 575
指定
データ・リスト項目の伝送 311

指定 *(続き)*
反復 307
編集ディレクティブ 317
リスト・ディレクティブ 321
指定文字 341
自動ストレージ
説明 238
AUTOMATIC 属性の構文 239
自動変数、再帰の影響 108
集合の割り当て 205
修飾
共用体 187
構造体 187
説明 248
区切り文字としての使用 14
修飾された参照 187
終了
開始ブロック 119
スレッド 391
プログラム 95
プロシージャー 106
ブロック 96, 221
出力
定義 277
出力と入力
区域の 258
条件 358
説明 277
主プロシージャー
引数の引き渡し 118
呼び出し 94
主プロシージャーの呼び出し 94
順序、計算の
式と参照の 72
条件
クラス 358
計算の 358
出力と入力 358
状況 358
その他 358
入出力 358
プログラム・チェックアウト 358, 359
ANYCONDITION 367
AREA 369
ATTENTION
説明 369
マルチスレッド化の 393
CONDITION 370
CONVERSION 371
ENDFILE 373
ENDPAGE 374
ERROR 375
FINISH 375
FIXEDOVERFLOW 376
INVALIDOP 377
条件 *(続き)*
KEY 378
NAME 378
OPTIMIZATION のもとでの発生 359
OVERFLOW 379
RECORD 380
SIZE 381
STORAGE 382
STRINGRANGE 382
STRINGSIZE 383
SUBCRIPTRANGE 384
TRANSMIT 385
UNDEFINEDFILE 386
UNDERFLOW 387
ZERODIVIDE 387
条件コード
説明 357
条件コード、ONCODE 組み込み関数との併用 475
条件処理
暗黙アクション 357
確立された処置 357
条件接頭語の有効範囲 359
条件の割り込み可能化 357
条件の割り込み禁止化 357
説明 357
複数条件 365
マルチスレッド化 393
割り込み可能条件の確立 357
CONDITION 属性 365
ON ステートメント
確立されたアクションの有効範囲 361
構文 360
説明 360
動的に派生した ON ユニット 362
ヌル ON ユニット 361
ファイル変数のための ON ユニット 362
RESIGNAL ステートメント 364
REVERT ステートメント 363
SIGNAL ステートメント 364
条件処理組み込み関数
要約 403
DATAFIELD 434
ONCHAR 474
ONCODE 475
ONCONDCOND 475
ONCONDID 475
ONCOUNT 476
ONFILE 476
ONGSOURCE 477
ONKEY 477
ONLOC 478
ONSOURCE 478
ONWCHAR 480
条件処理組み込み関数 *(続き)*
ONWSOURCE 480
条件接頭語
構文 358
使用法 357
説明 17
例 358
小構造体名 185
小数点と数字の指定子 344
初期値
共用体の 270
STATIC 変数での 274
初期設定
基底付き変数と被制御変数 275
共用体 274
自動変数 275
静的変数 274
配列変数 273
序数
オプション 148
許容属性 155
組み込み関数 156
説明 148
定義 148
配列との併用 156
例 149
DEFINE ORDINAL ステートメント 148
DO ループの例 156
DOWNTHRU の使用 220
ORDINAL 属性 153
PRECISION 属性 149
SIGNED 属性 149
UNSIGNED 属性 149
VALUE 属性 149
序数処理組み込み関数
要約 410
リスト 156
ORDINALNAME 481
ORDINALPRED 481
ORDINALSUCC 482
序数データ、属性、分類 29
処理モード
位置指定 301
移動 301
説明 301
処理リスト 259
数学組み込み関数
精度 401
要約 408
ACOS 415
ACOSF 415
ASIN 418
ASINF 419
ATAN 419
ATAND 420

数学組み込み関数 *(続き)*
ATANF 420
ATANH 420
COS 430
COSD 431
COSF 431
COSH 431
ERF 441
ERFC 441
EXP 441
EXPF 442
GAMMA 447
LOG 460
LOG10 461
LOG10F 461
LOG2 461
LOGF 460
LOGGAMMA 460
SIN 506
SIND 506
SINF 506
SINH 506
SQRT 508
SQRTF 509
TAN 514
TAND 514
TANF 514
TANH 514
数学組み込み関数の精度 401
数字
および小数点の指定子 344
10 進数 10
16 進数 11
2 進数 11
数字データ
サブフィールド 344
定義 45
ピクチャー指定子 343
フィールド 344
変換 85
数字ピクチャー項目
説明 341, 343
スカラー ID 49
スケール、算術演算での 61
スケール因数
説明 31
文字 355
スタックに入れる 108
ステートメント
お勧めするコーディング 16
キーワード 18
グループ 19
構文 16
説明 201
代入 18, 202
単純 18
ステートメント *(続き)*
ヌル 228
複合 18
ALLOCATE 251
ANSWER
プリプロセッサー・プロシージャーでの使用 545
ATTACH 390
BEGIN 119
CALL 133
CLOSE 290
DECLARE 160
DEFAULT 177
DEFINE ALIAS 147
DEFINE ORDINAL 148
DEFINE STRUCTURE 150
DELAY 208
DELETE 296
DETACH 392
DISPLAY 209
DO 209
END 221
ENTRY 100
EXIT 106
FETCH 109
FORMAT 321
FREE 243, 252
GET
データ・ディレクティブ 314
編集ディレクティブ 319
リスト・ディレクティブ 322
STREAM 入力 304
GET STRING 304
GO TO 223
IF 223
ITERATE 226
LEAVE 226
LOCATE 296
ON 360
OPEN 286
PACKAGE 96
PROCEDURE
主プロシージャーを呼び出すための使用 94
説明 99
PUT
データ・ディレクティブ 316
編集ディレクティブ 320
リスト・ディレクティブ 323
STREAM 出力 305
READ 295
RELEASE
制約事項 109
説明 111
動的にロードする外部 109
例 111
ステートメント *(続き)*
RESIGNAL 364
RETURN
関数からの戻り 134
構文 134
サブルーチンとの併用 134
使用法 106
説明 134
プリプロセッサー・プロシージャーでの使用 544
REVERT 363
REWRITE
説明 295
SELECT
説明 233
例 234
SIGNAL 364
STOP
使用法 106
WAIT 392
WRITE
説明 295
%PROCEDURE 543
ステートメントのオプション 544
ステートメント・エレメント
DBCS の場合 20
SBCS の場合 13
ストリーム指向データ伝送
定義 277
リスト・ディレクティブ 303
ストリング演算子 () 15
ストリング処理組み込み関数
要約 412
BIT 422
BOOL 423
CENTERLEFT 424
CENTERRIGHT 425
CHARACTER 426
CHARGRAPHIC 426
COPY 430
EDIT 438
GRAPHIC 448
HIGH 451
INDEX 453
LEFT 458
LENGTH 458
LOW 461
LOWERCASE 462
MAXLENGTH 464
MPSTR 471
REPEAT 498
REVERSE 499
RIGHT 499
SEARCH 501
SEARCHR 503
SUBSTR 511

ストリング処理組み込み関数 *(続き)*
TALLY 513
TRANSLATE 516
TRIM 517
UPPERCASE 521
VERIFY 524
VERIFYR 524
WHIGH 526
WIDECHAR 526
WLOW 527
ストリング内の引用符 24
ストリング・オーバーレイ定義 268
ストリング・データ
引用符 24
可変長の伝送 294
グラフィック 42
混合 43
属性
省略語 37
長さの指定 37
分類 28
定義 25
反復因数 40, 273
ビット 41
文字データ 40
BIT 属性 37
CHARACTER 属性 37
GRAPHIC 属性 37
NONVARYING 属性 38
PICTURE 属性 39
VARYING 属性 38
VARYINGZ 属性 38
ストレージ
基底付き 245
自動 239
制御 237
静的 239
接続された 263
被制御 241
非連結 185
分類 237
割り振り 237
ストレージ制御組み込み関数
要約 411
ADDR 416
ADDRDATA 417
ALLOCATE 417
ALLOCATION 418
ALLOCSIZE 418
AUTOMATIC 421
AVAILABLEAREA 421
BINARYVALUE 422
BITLOCATION 422
CHECKSTG 428
CURRENTSIZE 433
CURRENTSTORAGE 434
ストレージ制御組み込み関数 *(続き)*
EMPTY 439
ENTRYADDR 440
HANDLE 449
LOCATION 459
NULL 472
OFFSET 472
OFFSETADD 473
OFFSETDIFF 473
OFFSETSUBTRACT 473
OFFSETVALUE 474
POINTER 490
POINTERADD 491
POINTERDIFF 492
POINTERSUBTRACT 492
POINTERVALUE 492
SIZE 507
STORAGE 509
SYSNULL 512
SYSTEM 513
UNALLOCATED 518
スペース・フォーマット項目 338
スレッド
切り離し 392
作成 389
終了 391
使用 389
条件処理 393
待機 392
タスク変数 393
データの共用 394
ファイルの共用 394
ATTACH ステートメント 390
ENVIRONMENT オプション 391
TASK 属性 393
THREAD オプション 390
TSTACK オプション 391
スレッド間のデータの共用 394
スレッド間のファイルの共用 394
制御ストレージ 237
制限、FETCH および RELEASE の
説明 109
制限事項 575
制限付き式
組み込み関数の適用 76
説明 76
例 76
整数
値 31
整数演算組み込み関数
要約 407
IAND 452
IEOR 452
INOT 454
IOR 454
ISIGNED 455
整数演算組み込み関数 *(続き)*
ISLL 455
ISRL 456
IUNSIGNED 457
LOWER2 462
RAISE2 495
静的記憶装置 238, 239
静的割り振り 237
精度処理組み込み関数
要約 410
ADD 415
BINARY 421
DECIMAL 437
DIVIDE 438
FIXED 446
FLOAT 447
MULTIPLY 472
PRECISION 493
SIGNED 505
SUBTRACT 512
UNSIGNED 519
セット、データの 277
接頭演算子
演算 59
条件
構文 358
使用法 357
例 358
セミコロン 14
ゼロ置換文字 353
ゼロ抑制文字 346
宣言
暗黙 162
コンテキスト 162
配列 181
明示 159
有効範囲
説明 163
例 164
INTERNAL 属性と EXTERNAL 属
性での定義 166
DEFINE ORDINAL ステートメント
148
宣言、データの
説明 159
属性分配 162
宣言、DEFINE ALIAS、ステートメント
147
ソースからターゲットへの変換規則
グラフィック 90
コード化算術フォーマット 82
固定 10 進数 84
固定 2 進数 83
算術文字 85
算術文字 PICTURE 85
数字 85

ソースからターゲットへの変換規則 *(続き)*
ビット 88
浮動 10 進数 85
浮動 2 進数 84
文字 86
ワイド文字 90
走査、プリプロセッサー 537
総称入り口宣言 130
総称記述子 131
総称選択 132
総称名 130
相対データ・セット 279
相対ライン 337
挿入演算 59
挿入演算子と配列 74
挿入文字 347
添え字
介在 191
定義 183
配列の 183
添え字付きの修飾された参照 191
属性
共用体データ 29
組み合わせ 288
計算データ 24
コード化算術式 28, 29
構造体データ 29
序数データ 29
ストリング・データ 28, 29
説明 24
代替 280
タスク・データ 29
追加 280
データ
説明 24
リスト 25
データ・タイプに応じた分類 26
デフォルトのデータ属性 175
名前付きのコード化算術式 28
名前付きのストリング・データ 28
配列データ 29
パラメーター 102
非データ 26
ファイル・データ 28, 29
プログラム制御データ 25
ラベル・データ 28, 29
ロケーター・データ 29
ABNORMAL 261
ALIGNED
記憶位置合わせ要件 172
説明 172
例 175
AREA 29
ASSIGNABLE 260
AUTOMATIC 239
属性 *(続き)*
BASED 245
BIGENDIAN 261
BINARY 31
BIT 37
BUFFERED 284
BUILTIN
使用法 113, 398
BYADDR 139
BYVALUE 139
CHARACTER
説明 37
COMPLEX 31
CONDITION 365
CONNECTED 263
CONTROLLED 241
DATE 46
DECIMAL 31
DEFINED 264
DIMENSION 182
DIRECT 284
ENTRY 122
ENVIRONMENT 285
EXTERNAL
使用法 110
説明 166
FILE 279
FIXED
説明 31
FLOAT 31
FORMAT
説明 53
変数タイプによる分類 29
GENERIC 130
GRAPHIC 37
HANDLE 151
HEXADEC 262
IEEE 262
INITIAL 270
INPUT 283
INTERNAL 166
KEYED 285
LABEL 51
LIKE 188
LIMITED 129
LIST 125
LITTLEENDIAN 261
NOINIT 190
NONASSIGNABLE 260
NONCONNECTED 263
NONVARYING 38
NORMAL 261
OFFSET 256
OPTIONAL 124
OPTIONS 135
ORDINAL 153
属性 *(続き)*
OUTPUT 283
PARAMETER 102
PICTURE 39
POINTER 250
POSITION 264
PRECISION 31
PRINT 324
REAL 31
RECORD 283
RECURSIVE 107
RESERVED 170
RETURNS 144
SEQUENTIAL 284
SIGNED
説明 32
データ・ストレージ所要量 33
STATIC 239
STREAM 283
TASK 393
TYPE 152
UNALIGNED
記憶位置合わせ要件 172
説明 172
例 175
UNBUFFERED 284
UNION 187
UNSIGNED
説明 32
データ・ストレージ所要量 33
UPDATE 283
VALUE 49
VARIABLE 53
VARYING 38
VARYINGZ 38
WIDECHAR
説明 37
属性分配 162
その他の組み込み関数
要約 409
BYTE 423
CHARVAL 427
COLLATE 428
GETENV 448
HEX 450
OMITTED 474
PACKAGENAME 482
PLIRETV 488
PRESENT 494
PROCEDURENAME 494
RANK 496
SOURCEFILE 508
SOURCELINE 508
STRING 509
UNSPEC 519
VALID 522

その他の組み込み関数 *(続き)*
WCHARVAL 525
その他の条件
ANYCONDITION 367
AREA 369
ATTENTION 369
CONDITION 370
ERROR 375
FINISH 375
STORAGE 382

[タ行]

ターゲット
疑似変数
説明 58
例 58
構造体 204
説明 58
ターゲット変数の要件 204
中間結果 58
配列 204
変数 58
大構造体名 185
代数比較演算子 68
代替属性 280
定義 280
BUFFERED および UNBUFFERED 284
INPUT、OUTPUT、および UPDATE 283
RECORD と STREAM 283
SEQUENTIAL と DIRECT 284
代入ステートメント
説明 202
ターゲット変数の要件 204
定義 18
BY NAME オプション 202
タイプ
修飾 153
説明 152
タイプ付き関数 158
定義 147
ハンドル 151
変数 152
DEFINE STRUCTURE ステートメント 150
HANDLE 組み込み関数 151
タイプ付き関数 158
説明 531
引数 531
リスト 532
BIND 532
CAST 532
FIRST 533
LAST 533
タイプ付き関数 *(続き)*
NEW 534
RESPEC 534
SIZE 534
タイプ付き関数の呼び出し 531
タイプ付きの名前 147
タイプ付き変数、宣言 152
修飾 153
ハンドル 151
タイプ定義、記述 147
タスク変数 393
タスク・データ、属性、分類 29
ダブルワード、データの位置合わせでの 171
単純
オーバーレイ定義 265, 268
ストリング・オーバーレイ定義 268
単純 102
単純定義 266
定義 265, 266
パラメーター
境界、長さ、およびサイズ 102
被制御 102
iSUB 定義 267
単純ステートメント 18
調節可能エクステント 245
直接入り口宣言 120
追加属性
定義 280
ENVIRONMENT 285
KEYED 285
通貨記号
説明 350
定義 349
通知、条件の 364
データ
位置合わせ 171
入り口 120
エレメント 23, 539
オフセット 256
区域 255
グラフィック 42
計算の 24
項目 23
混合 43
算術文字 45
指定 306
数字 343
スレッド間での共用 394
属性 24
タイプ 24
伝送 277
ビット 41
ビット定数 41
フォーマット 53
フォーマット項目 327
データ *(続き)*
プログラム制御
説明 25
タイプおよび属性 51
変換
エラー 92
組み込み関数を使用した 78
算術演算での 61
説明 77
ソースからターゲットへの変換規則 82
文字 40
文字定数 40
ラベル 51
ロケーター 247
ワイド文字 44
10 進固定小数点 34
10 進浮動小数点 36
2 進固定小数点 33
2 進浮動小数点 35
データ項目
式 56
集合 23
スカラー 23
定義 23
複素数 31
モード 31
データ項目のモード 31
データ指定オプション、ストリーム入出力の
説明 306
データ・ディレクティブ 312
定義 304
伝送されるデータ 293
データ宣言
暗黙 162
共用体 186
言語に固有の属性デフォルト 176
構造体 185
説明 159
配列 181
明示 159
データ伝送
位置合わせされていないビット・ストリング 293
可変長ストリング 294
漢字ストリング 293
区域変数 294
出力 277
ストリーム指向 303
ストリーム指向ステートメント
説明 304
タイプ 3 の DO グループ 307
GET 304
PUT 305
説明 293

データ伝送 (続き)
データ集合 293
データ・ディレクティブ 303
データ・リスト項目 311
入力 277
編集ディレクティブ 304
レコード単位 293
レコード単位ステートメント
説明 294
DELETE 296
LOCATE 296
READ 295
REWRITE 295
WRITE 295
TRANSMIT 条件 385
データ伝送ステートメント・オプション
説明 305
COPY 305
FILE 308
LINE 308
PAGE 309
SKIP 309
STRING 310
データの位置合わせ
ストレージ・アドレス 171
説明 171
ALIGNED 属性と UNALIGNED 属性の使用法 172
データの位置合わせ属性 171
データ変換
エラー 92
規則 78
組み込み関数での開始 78
計算データ 78
算術値の精度 80
算術演算での 61
算術データからビット・ストリングへの変換、例 90
算術データから文字ストリングへの変換、例 91
ストリングの長さ 79
説明 77
ソースからターゲットへの変換規則 82
変換エラー 91
モード 80
データ・エレメント
説明 23
属性 23
データ項目 23
定数
引用符 24
句読法 24
名前付き 24
プリプロセッサー 539
データ・セット
記憶 278
索引付き 279
相対 279
タイプ 278
データの伝送 277
領域 279
連続 279
データ・タイプ
計算の 24
説明 24
データ・ディレクティブのデータ指定
説明 312
GET ステートメントの使用法 314
PUT ステートメントの使用法 316
データ・ディレクティブ・データ伝送 303
定数
入り口
構文 121
使用法 120
説明 120
虚数 31, 32
グラフィック 42
名前付き 49
ビット 41
ファイル 280
文字 40
文字ストリング 40
ラベル 52
10 進固定小数点 35
10 進浮動小数点 36
2 進固定小数点 33
2 進浮動小数点 36
B3 (16 進ビット) ストリング 42
B4 (16 進ビット) ストリング 42
BX (16 進ビット) ストリング 42
GX (グラフィック) ストリング 43
M (混合) ストリング 43
WX (ワイド文字) ストリング 45
XN (2 進 16 進数) 34
XU (2 進 16 進数) 34
ディレクティブ
*PROCESS 231
%INCLUDE 225
%LINE 227
%NOPRINT 227
%NOTE 228
%OPTION 229
%PAGE 230
%POP 230
%PRINT 230
%PROCESS 231
%PUSH 231
%SKIP 235
デフォルト、属性の
言語に固有の 176
言語に固有のデフォルト属性の復元 181
説明 175
データ属性 175
DEFAULT ステートメント 177
伝送、データの 277
等号 14
動的に派生した ON ユニット 362
動的ロード、外部プロシージャーの
FETCH ステートメント 109
RELEASE ステートメント 109
動的割り振り 237
特別言語文字 10

[ナ行]

内部、定義 164
内部プロシージャー 99
長さ
単純パラメーター 102
被制御パラメーター 102
名前
プリプロセッサー 541
名前、タイプ 147
名前付き定数 49
名前付きのコード化算術式の属性 28
名前付きのストリング・データ属性 28
名前付きの定数、説明 24
名前の認識 159
入出力組み込み関数 407
要約 407
COUNT 431
ENDFILE 440
FILEDDINT 442
FILEDDTEST 443
FILEDDWORD 443
FILEID 444
FILEOPEN 445
FILEREAD 445
FILESEEK 445
FILETELL 446
FILEWRITE 446
LINENO 459
ONSUBCODE 479
PAGENO 482
SAMEKEY 501
入力
区域の 258
条件
ENDFILE 373
ENDPAGE 374
KEY 378
NAME 378
RECORD 380

入力 *(続き)*
条件 *(続き)*
TRANSMIT 385
UNDEFINEDFILE 386
説明 277
定義 277
ヌル ON ユニット 361
ヌル引数、組み込み関数での使用 400
ヌル・ステートメント
説明 228
定義 18

[ハ行]

ハーフワード 171
排他的 OR 演算子 67
バイト、定義 171
配列
エクステント 182
境界 182
クロスセクション 184
構造体と共用体 191
式
説明 56, 74
例 57
接頭演算子と 74
挿入演算子と 74
添え字 183
属性 29
ターゲット 204
定義 181
配列同士の演算 75
配列とエレメントの演算 74
変数 181
例 182
割り当て 204, 205
DIMENSION 属性 182
配列式
説明 74
定義 56
例 56
配列処理組み込み関数
要約 402
ALL 417
ANY 418
DIMENSION 438
HBOUND 449
LBOUND 457
POLY 492
PROD 494
SUM 512
配列の上限、獲得 (HBOUND) 449
配列の下限、獲得 (LBOUND) 457
配列変数 181
パッケージ 96
バッファー管理組み込み関数
COMPARE 428
HEXIMAGE 451
MEMINDEX 465
MEMSEARCH 466
MEMSEARCHR 466
MEMVERIFY 467
MEMVERIFYR 468
PICSPEC 482
XMLCHAR 527
パラメーター
エレメント 117
属性 102
配列引数
例 103
引数 116
パラメーター記述子リスト 123
パラメーターの付いた配列引数 103
反復因数 272, 273, 317
ストリングの場合 40, 273
ピクチャー文字の場合 341
ビット・データの 41
反復実行 (DO ステートメント) 210, 217
比較演算
オペランドの変換 68
グラフィック 69
序数データ 69
説明 68
代数 68
ビット 68
プログラム制御データ 69
ポインターおよびオフセット・データ 69
文字 68
例 70
ワイド文字 69
比較演算子 15
引数
受け渡し
主プロシージャーへ 118
プロシージャーへの 116
仮の
規則 117
説明 117
属性の導出 117
指定 139, 400
引数およびパラメーターの関連 116
引数の受け渡し
主プロシージャーへ 118
説明 116
BYVALUE および BYADDR の使用 116
引数の集合 400
ピクチャー指定文字
定義 341
ピクチャー指定文字 *(続き)*
9
数字の場合 344
文字データの場合 342
A 342
B 347
CR 353
DB 353
E 354
F 355
I 353
K 354
R 353
S 350
T 353
V
数字の場合 344
挿入 348
X 342
Y 353
Z 346
$ 350
* 346
+ 350
- 350
/ 347
ピクチャー・データ
指定 39
数字データの指定子 343
スケール因数 355
反復因数 341
文字データの指定子 342
PICTURE 属性の構文 39
ピクチャー・フォーマット項目 335
被制御
構造体と共用体メンバー 245
ストレージ 238, 241
パラメーター 102
変数
説明 241
複数生成 244
ALLOCATE ステートメントの使用法 241
FREE ステートメントの使用法 243
日付/時刻組み込み関数
パターン 405
要約 404
リリアン形式 404
DATE 435
DATETIME 435
DAYS 435
DAYSTODATE 436
DAYSTOSECS 437
REPATTERN 497
SECS 503

日付/時刻組み込み関数 *(続き)*
SECSTODATE 504
SECSTODAYS 505
TIME 515
VALIDDATE 522
VARGLIST 523
VARGSIZE 523
WEEKDAY 525
Y4DATE 528
Y4JULIAN 529
Y4YEAR 530
ビット
演算子
説明 15
ビット演算子での使用 67
データ 41
定数 41
変換
規則 88
説明 80
ビット演算子
使用法 67
例 68
ビット・ストリングの伝送、位置合わせされていない 293
ビット・データ
反復因数 41
ビット・フォーマット項目 328
非データ属性 26
標準 4
ピリオド 14
非連結ストレージ 185, 266
ブール演算子 67
ファイル
暗黙オープン (implicit opening) 288
オープンとクローズ 285
参照の指定 282
スレッド間での共用 394
宣言 279
属性 28
属性の説明 280
代替属性 280
追加属性 280
定義 279
定数 280
変数 282
FILE 属性 279
PRINT 325
SYSIN 291
SYSPRINT 291
ファイルのオープンとクローズ 285
ファイル・データ 29
フィールド 344
フォーマット項目
説明 317
A 327
フォーマット項目 *(続き)*
B 328
C 329
COLUMN 329
E 330
F 332
G 334
L 335
LINE 335
P 335
PAGE 336
R 336
SKIP 337
V 338
X 338
フォーマット表記規則 1
フォーマット・データ 53
復元、言語に固有のデフォルトの 181
複合記号 12
複合ステートメント 18
複合代入演算子 203
複数条件 365
複数世代、被制御変数の 244
複数割り当て 206
複素数
データ項目 31
フォーマット項目 329
含まれている、定義 164
符号
固定的使用 351
異なる符号を持つ CR と DB の使用 353
数字データでの指定 350
浮動的使用 351
浮動小数点
データ変換 85
フォーマット項目 330
10 進データ 36
2 進データ 35
浮動小数点演算組み込み関数
要約 407
EXPONENT 442
PRED 493
SCALE 501
SUCC 512
浮動小数点の照会組み込み関数
要約 406
EPSILON 441
HUGE 451
MAXEXP 463
MINEXP 469
PLACES 483
RADIX 495
TINY 515
浮動文字 351
ブランク
区切り文字としての使用 14
説明 15
プリプロセッサー
機能 535
組み込み関数 547
参照と式 540
出力 536
出力テキスト 536
ステートメント
説明 536
リスト 538
ステートメント、リスト 558
走査
およびプリプロセッサー・ステートメント 538
およびリスト制御ステートメント 538
説明 537
入力テキスト 538
名前の有効範囲 541
入力 535
入力テキスト 536, 538
プリプロセッサー 536
プロシージャー 541
変数とデータ・エレメント 539
リスト制御 536
リスト制御ステートメント 536
例 569
%ACTIVATE 558
%DEACTIVATE 559
%DECLARE 560
%DO 562
%END 562
%GO TO 563
%IF 563
%INCLUDE 564
%INSCAN 565
%ITERATE 566
%LEAVE 566
%NOTE 567
%REPLACE 568
%SELECT 568
%XINCLUDE 568
%XINSCAN 569
%代入 559
%ヌル 567
フルワード 171
プログラマー定義の名前 13
プログラム
エレメント
入り口値 133
入り口データ 120
入り口の呼び出し 133
開始ブロック 119
関数 113

プログラム *(続き)*
エレメント *(続き)*
組み込み関数 115
サブルーチン 112
説明 9
CALL ステートメント 133
OPTIONS オプション 135
RETURN ステートメント 134
活動化 94
構造 93
サブルーチン 99
終了 95
定義 (PL/I の場合の) 93
パッケージ 96
プロシージャー 99
ブロック
活動化 95
説明 95
編成 93
RETURN 134
プログラム制御データ
使用法 51
説明 25
タイプおよび属性 51
プログラムの正常終了 95
プログラムの編成 93
プログラム・エレメント
グループ 19
ステートメント
説明 16
単純 18
複合 18
説明 9
1 バイト文字セット (SBCS)
ステートメント・エレメント 13
説明 9
2 バイト文字セット (DBCS)
ステートメント・エレメント 20
説明 19
プログラム・チェックアウト条件 359
STRINGRANGE 382
STRINGSIZE 383
SUBSCRIPTRANGE 384
プログラム・ブロックの定義 93
プロシージャー
外部 99
活動化 105
再帰的 107
終了 106
制御の伝送 106
説明 99
属性の指定 102
動的ロード
規則 109
説明 109
プロシージャー *(続き)*
動的ロード *(続き)*
FETCH ステートメントの使用法 110
RELEASE ステートメントの使用法 111
内部 99
引数の受け渡し
仮引数の使用 117
説明 116
BYVALUE および BYADDR の使用 116
プリプロセッサー 541
ブロック 95
メインへの引数の引き渡し 118
ブロック
開始 119
活動化 95
終了 96
説明 95
タイプ 95
パッケージ 96
プロシージャー 99
別名
定義 147
DEFINE ALIAS ステートメント 147
変換
エラー 92
オペランド 63
組み込み関数を使用した 78
算術値の精度 80
算術演算での 61
ストリングの長さ 79
ソースからターゲットへの変換規則 82
その他のデータ属性の 80
データ 77
モード 80
連結演算 70
ロケーター・データの 247
変換エラー 91
編集ディレクティブ
データ伝送 304
フォーマット項目 327
編集ディレクティブのデータ指定 317
変数
入り口 121
オフセット 247
基底付き
識別 245
使用法 250
構造体 185
参照 23
自動 108
説明 241
ターゲット 58
変数 *(続き)*
定義 23
配列 181
被制御 241
複素数データ項目を表す 31
プリプロセッサー 539
ポインター 247, 250
変数、タイプ付き 152
変数、ハンドルとしての 151
ポインター演算 59
ポインター記号 14
ポインター変数 247, 250

[マ行]

マクロ機能組み込み関数
COLLATE 548
COMMENT 548
COMPILEDATE 548
COMPILETIME 549
COPY 549
COUNTER 550
DIMENSION 550
HBOUND 550
INDEX 551
LBOUND 551
LENGTH 552
MACCOL 552
MACLMAR 552
MACRMAR 553
MAX 553
MIN 553
PARMSET 554
QUOTE 554
REPEAT 554
SUBSTR 555
SYSPARM 555
SYSTEM 556
SYSVERSION 556
TRANSLATE 556
VERIFY 557
マルチスレッド化
オプション
ENVIRONMENT 391
THREAD 390
TSTACK 391
条件処理 393
スレッド
切り離し 392
作成 389
終了 391
使用 389
待機 392
スレッド間のデータの共用 394
スレッド間のファイルの共用 394
説明 389

マルチスレッド化 *(続き)*
タスク変数 393
リンケージ要件 391
ATTACH ステートメント 390
TASK 属性 393
THREADID 組み込み関数 393
マルチスレッド化、THREADID 組み込み関数の 515
マルチスレッド化機能 389
明示宣言 159
明示的にロケーター修飾された参照 249
モード、処理の
位置指定 301
移動 301
説明 301
文字
英字 10
英数字 10
セット
1 バイト 9
2 バイト 19
ゼロ消去 346
挿入 347
定数 40
特殊 11
特別言語 10
比較演算での使用 68
ピクチャー指定 39
フォーマット項目 327
文字データ
説明 40
ピクチャー指定子 342
変換 80, 86
文字ストリング定数 40
文字セット
説明 9
1 バイト
区切り文字と演算子 14
ステートメント・エレメント用 13
DBCS フォーマットの ID 20
ID 13
2 バイト
ステートメント・エレメント 20
ID 20

[ヤ行]

有効範囲
確立されたアクションの 361
条件接頭語の 359
プリプロセッサー名 541
ラベル宣言の 163
抑止文字 346
呼び出されたプロシージャー 105
呼び出し、タイプ付き関数の 531
呼び出し側ブロック 105
呼び出し規則
OPTLINK 142
SYSTEM 142
呼び出し点、プロシージャーの 105

[ラ行]

ラベル 17
ラベル、言語ステートメントの 52
ラベル定数 52
ラベル・データ
説明 51
属性 28, 29
リスト
処理 259
双方向 260
単一方向 260
チェーン 259
パラメーター記述子 123
リスト制御ステートメント 536
リスト・ディレクティブ
出力 323
データ指定 321
データ伝送 303
入力 322
GET ステートメント 322
PUT ステートメント 323
リモート・フォーマット項目 336
領域データ・セット 279
リリアン形式 404
累乗演算、特殊な場合 67
レコード単位データ伝送
ステートメント 294
説明 293
定義 278
レベル、構造体の
固有名の指定 186
説明 185
レベル番号 (構造体エレメントの) 193
連結
演算 70
演算子 15
連結ストレージ 263
連続データ・セット 279
ロード・モジュール
説明 93
ファイル拡張子 93
ロケーター
参照 248
修飾 248
修飾子 14
修飾のレベル 249
データ
オフセット変数 247
修飾 248
説明 247
ロケーター *(続き)*
データ *(続き)*
属性 29
ポインター変数 247
パラメーター 118
変換 247
論理演算子
使用法 15
説明 67
論理レベル (構造体エレメントの) 193

[ワ行]

ワイド文字ストリング定数 45
ワイド文字データ
変換 90
WX (ワイド文字 16 進) ストリング定数 45
ワイド文字定数
比較演算 69
割り当て
エレメント 204
区域 258
構造体 204, 205
構造体割り当てのための BY NAME の使用 207
式の値 207
集合 205
配列
集合の割り当て 205
ターゲット変数 204
複合 203
複数 206
割り込み可能条件 357
割り振り 237

[数字]

1 バイト文字セット (SBCS)
英字 10
ステートメント・エレメント 13
説明 9
特別言語 10
10 進数字 10
16 進数字 11
2 進数字 11
10 進固定小数点データ
説明 34
変換 84
10 進固定小数点定数 35
10 進数字 10
10 進浮動小数点データ
説明 36
変換 85
10 進浮動小数点定数 36

16 進 (X) の文字ストリング定数 40
16 進数字 11
2 進固定小数点データ
説明 33
変換 83
2 進固定小数点定数 34
2 進数字 11
2 進浮動小数点データ
説明 35
変換 84
2 進浮動小数点定数 36
2 バイト文字セット (DBCS)
グラフィック・データ内の 42
継続規則 21
ステートメント・エレメント 20
説明 19
ソース・プログラム内での使用 19
入出力ストリーム内のデータ 326
ID 20
9 ピクチャー指定文字
使用法 344
文字データの場合 342

A

A ピクチャー指定文字 342
A フォーマット項目 327
ABNORMAL 属性 261
ABS 組み込み関数 415
ACOS 組み込み関数 415
ACOSF 組み込み関数 415
ADD 組み込み関数 415
ADDR 組み込み関数 416
ADDRDATA 組み込み関数 417
ALIGNED 属性
記憶位置合わせ要件 172
説明 172
例 175
ALL 組み込み関数 417
ALLOC (ALLOCATE) ステートメント 241
ALLOCATE (ALLOC)
組み込み関数
基底付き区域変数の場合 246
基底付き変数 246, 250
構文 417
ステートメント 241
ALLOCATION (ALLOCN) 組み込み関数 418
ALLOCSIZE 組み込み関数 418
ANSWER ステートメント
プリプロセッサー・プロシージャーでの使用 545
ANY 組み込み関数 418
ANYCONDITION 条件 367
AREA
条件 369
属性 255
ASIN 組み込み関数 418
ASINF 組み込み関数 419
ASM (ASSEMBLER) オプション 138
ASSEMBLER (ASM) オプション 138
ASSIGNABLE 属性 260
ATAN 組み込み関数 419
ATAND 組み込み関数 420
ATANF 組み込み関数 420
ATANH 組み込み関数 420
ATTACH ステートメント 390
ATTENTION (ATTN) 条件
説明 369
マルチスレッド化 393
AUTOMATIC (AUTO) 組み込み関数 421
AUTOMATIC 組み込み関数
基底付き区域変数の場合 246
基底付き変数の場合 246, 250
AUTOMATIC、(AUTOMATIC) 属性 239
AUTOMATIC、(AUTO) 属性 239
AVAILABLEAREA 組み込み関数
区域変数の場合 258
構文 421

B

B (挿入文字) 348
B フォーマット項目 328
B3 (16 進ビット) ビット・ストリング定数 42
B4 (16 進ビット) ビット・ストリング定数 42
BASED 属性 245
BEGIN ステートメント
説明 119
有効な OPTIONS オプション 135
BIGENDIAN 属性 261
BINARY (BIN) 組み込み関数 421
BINARY (BIN) 属性 31
BINARYVALUE (BINVALUE) 組み込み関数 422
BINARYVALUE 組み込み関数
序数の 156
ポインター式との併用 60
BIND タイプ付き関数 532
BIT 組み込み関数 422
BIT 属性 37
BITLOCATION (BITLOC) 組み込み関数 422
BOOL 組み込み関数 423
BUF (BUFFERED) 属性 284
BUFFERED (BUF) 属性 284
BUILTIN 属性 398
組み込み関数の名前を宣言する 113
BX (16 進ビット) ビット・ストリング定数 42
BY NAME オプション、代入ステートメントの
構造体割り当てで指定された場合 206
構造体割り当てで指定されなかった場合 205
説明 202
BYADDR オプション 139
BYADDR 属性 139
BYTE 組み込み関数 423
BYVALUE オプション 139
BYVALUE 属性 139

C

C フォーマット項目 329
CALL オプション、INITIAL 属性の 272
CALL ステートメント 133
CAST タイプ付き関数 532
CDS 組み込み関数 423
CEIL 組み込み関数 424
CELL、同義語 187
CENTERLEFT (CENTER) 組み込み関数 424
CENTERRIGHT 組み込み関数 425
CHARACTER (CHAR) 組み込み関数 426
CHARACTER (CHAR) 属性
説明 37
CHARGRAPHIC (CHARG) 組み込み関数 426
CHARGRAPHIC オプション 140
CHARVAL 組み込み関数 427
CHECKSTG 組み込み関数 428
CLOSE ステートメント 290
COBOL オプション 140
COLLATE 組み込み関数 428
COLLATE マクロ機能組み込み関数 548
COLUMN キーワード
ANSWER プリプロセッサー・ステートメントの 546
COLUMN フォーマット項目 329
COMMENT マクロ機能組み込み関数 548
COMPARE 組み込み関数 428
COMPILEDATE マクロ機能組み込み関数 548
COMPILETIME マクロ機能組み込み関数 549
COMPLEX (CPLX) 組み込み関数 429
COMPLEX (CPLX) 属性 31
COND (CONDITION) 条件 370
CONDITION (COND) 条件 370
CONDITION 属性 365
CONJG 組み込み関数 430
CONNECTED (CONN) 属性 263

CONTROLLED (CTL) 属性 241
CONV (CONVERSION) 条件 371
CONVERSION (CONV) 条件 371
CONVERSION 条件接頭語 358
COPY オプション 305
COPY 組み込み関数 430
COPY マクロ機能組み込み関数 549
COS 組み込み関数 430
COSD 組み込み関数 431
COSF 組み込み関数 431
COSH 組み込み関数 431
COUNT 組み込み関数 431
COUNTER マクロ機能組み込み関数 550
CS 組み込み関数 432
CTL (CONTROLLED) 属性 241
CURRENTSIZE 組み込み関数 433
CURRENTSTORAGE (CSTG) 組み込み関数 434

D

DATAFIELD 組み込み関数 434
DATE 組み込み関数 435
DATE 属性
説明 46
DATETIME 組み込み関数 435
DAYS 組み込み関数 435
DAYSTODATE 組み込み関数 436
DAYSTOSECS 組み込み関数 437
DBCS (2 バイト文字セット) 19
DBCS の継続規則 21
DCL (DECLARE) ステートメント 560
説明 160
DECIMAL (DEC) 組み込み関数 437
DECIMAL (DEC) 属性 31
DECLARE (DCL) ステートメント
説明 160
DEF (DEFINED) 属性 264
DEFAULT (DFT) ステートメント 177
DEFAULT ステートメントの DESCRIPTORS オプション 178
DEFINE ALIAS ステートメント 147
DEFINE ORDINAL ステートメント
オプション 148
説明 148
DEFINE STRUCTURE ステートメント 150
DEFINED (DEF) 属性 264
DELAY ステートメント 208
DELETE ステートメント 296
DESCRIPTOR オプション 141
DETACH ステートメント 392
DFT (DEFAULT) ステートメント 177
DIM (DIMENSION) 属性 182
DIMENSION (DIM) 組み込み関数 438
DIMENSION (DIM) 属性 182
DIMENSION マクロ機能組み込み関数 550
DIRECT 属性 284
DISPLAY ステートメント 209
DIVIDE 組み込み関数 438
DLLINTERNAL オプション 141
DO グループ
タイプ 3 の DO グループ 210, 213
マクロ機能 209
例 217
DO ステートメント
説明 209
反復実行 210
DO ステートメントの BY オプション 211
DOWNTHRU オプション
序数での使用 220
説明 211
タイプ 3 の DO 指定との併用 216
例 219

E

E ピクチャー文字 354
E フォーマット項目 330
EDIT オプション 317
EDIT 組み込み関数 438
EMPTY 組み込み関数 439
区域変数の場合 258
END ステートメント
説明 221
ENDFILE 組み込み関数 440
ENDFILE 条件 373
ENDPAGE 条件 374
ENTRY ステートメント 100
ENTRY ステートメント、有効な OPTIONS オプション 136
ENTRY 属性
説明 122
有効な OPTIONS オプション 135
ENTRYADDR 疑似変数 440
ENTRYADDR 組み込み関数 440
entry-constant
FETCH ステートメントとの併用 110
ENV (ENVIRONMENT) 属性 285
ENVIRONMENT (ENV) 属性 285
ENVIRONMENT オプション 391
EPSILON 組み込み関数 441
ERF 組み込み関数 441
ERFC 組み込み関数 441
ERROR 条件
説明 375
プロシージャーの異常終了 106
EXE (ファイル拡張子) 93
EXIT ステートメント 106
EXP 組み込み関数 441
EXPF 組み込み関数 442
EXPONENT 組み込み関数 442
EXPORTS オプション 97
EXT (EXTERNAL) 属性 166
EXTERNAL (EXT) 属性
使用法 110
説明 166

F

F ピクチャー文字 355
F フォーマット項目 332
FETCH ステートメント
外部プロシージャーの動的ロード 109
制約事項 109
説明 110
FETCHABLE オプション 141
FILE オプション
ストリーム指向データ伝送の 295
説明 308
レコード単位データ伝送の 297
FILE 指定、OPEN ステートメントでの 286
FILE 属性 279
FILEDDINT 組み込み関数 442
FILEDDTEST 組み込み関数 443
FILEDDWORD 組み込み関数 443
FILEID 組み込み関数 444
FILEOPEN 組み込み関数 445
FILEREAD 組み込み関数 445
FILESEEK 組み込み関数 445
FILETELL 組み込み関数 446
FILEWRITE 組み込み関数 446
FINISH 条件 375
FIRST タイプ付き関数 533
FIXED 組み込み関数 446
FIXED 属性
説明 31
FIXEDOVERFLOW (FOFL) 条件 376
FIXEDOVERFLOW 条件接頭語 358
FLOAT 組み込み関数 447
FLOAT 属性 31
FLOOR 組み込み関数 447
FOFL (FIXEDOVERFLOW) 条件 376
FORMAT ステートメント 321
FORMAT 属性
説明 53
変数タイプによる分類 29
FORTRAN オプション 141
FREE ステートメント
基底付き変数 252
被制御変数 243
IN オプション 252
FROM オプション、データ伝送ステートメントの 297
FROMALIEN オプション 141

G

G フォーマット項目 334
GAMMA 組み込み関数 447
GENERIC 宣言の WHEN オプション 131
GENERIC 属性
  説明 130
  OTHERWISE オプションの使用法 131
GENERIC 属性の OTHERWISE オプション 131
GET STRING ステートメント 304
GET ステートメント
  ストリング 319
  データ・ディレクティブ 314
  編集ディレクティブ 319
  リスト・ディレクティブ 322
GETENV 組み込み関数 448
GO TO (GOTO) ステートメント
  説明 223
GRAPHIC ENVIRONMENT オプション 43
GRAPHIC オプション 43
GRAPHIC 組み込み関数 448
GRAPHIC 属性 (G) 37
GRAPHIC の文字への変換 (CHARGRAPHIC) 427
GX (グラフィック 16 進) ストリング定数 43

H

HANDLE 組み込み関数 449
HANDLE 組み込み関数のタイプ付き構造体 449
HANDLE 属性 151
HBOUND 組み込み関数 449
HBOUND マクロ機能組み込み関数 550
HEX 組み込み関数 450
HEXADEC 属性 262
HEXIMAGE 組み込み関数 451
HIGH 組み込み関数 451
HUGE 組み込み関数 451

I

I (オーバーパンチ) ピクチャー文字 353
IAND 組み込み関数 452
ID
  アスタリスク 14
  キーワードの使用法 13
  スカラー 49
  定義 13
  プログラマー定義の名前 13
  2 バイト文字の DBCS 20
ID *(続き)*
  DBCS 20
  DBCS フォーマットの SBCS 20
IEEE 属性 262
IEOR 組み込み関数 452
IF ステートメント 563
  構文 224
  説明 223
IF ステートメントの ELSE 文節 223
IF ステートメントの THEN 文節 223
IGNORE オプション、データ伝送ステートメントの 298
IMAG 疑似変数 453
IMAG 組み込み関数 453
IN オプション
  ALLOCATE ステートメント 251
  FREE ステートメント 252
IN オプション、基底付き変数の FREE ステートメントに指定する 252
INCLUDE ディレクティブ 225
INDEX 組み込み関数 453
INDEX マクロ機能組み込み関数 551
INITIAL CALL 272
INITIAL (INIT) 属性 270
INITIAL TO 273
INLINE オプション 141
INOT 組み込み関数 454
INPUT 属性 283
INT (INTERNAL) 属性 166
INTERNAL (INT) 属性 166
INTO オプション、データ伝送ステートメントの 298
INVALIDOP 条件 377
INVALIDOP 条件接頭語 358
IOR 組み込み関数 454
IRREDUCIBLE (IRRED) オプション 144
ISIGNED 組み込み関数 455
ISLL 組み込み関数 455
ISMAIN 組み込み関数 456
ISRL 組み込み関数 456
iSUB
  定義 265, 267
  非連結 266
ITERATE ステートメント 226
IUNSIGNED 組み込み関数 457

K

K ピクチャー文字 354
KEY オプション、データ伝送ステートメントの 299
KEY 条件 378
KEYED 属性 285
KEYFROM オプション、データ伝送ステートメントの 299
KEYTO オプション、データ伝送ステートメントの 300

L

L フォーマット項目 335
LABEL 属性
  説明 51
  有効な OPTIONS オプション 52
LAST タイプ付き関数 533
LBOUND 組み込み関数 457
LBOUND マクロ機能組み込み関数 551
LEAVE ステートメント 226
LEFT 組み込み関数 458
LENGTH 組み込み関数 458
LENGTH マクロ機能組み込み関数 552
LIKE 属性 188
LIMITED 属性
  説明 129
  例 130
LINE オプション 308
LINE ディレクティブ 227
LINE フォーマット項目 335
LINENO 組み込み関数 459
LINESIZE 指定、OPEN ステートメントでの 287
LINKAGE オプション 142
LIST 属性
  説明 125
LITTLEENDIAN 属性 261
LOCATE ステートメント 296
LOCATION (LOC) 組み込み関数 459
LOG 組み込み関数 460
LOG10 組み込み関数 461
LOG10F 組み込み関数 461
LOG2 組み込み関数 461
LOGF 組み込み関数 460
LOGGAMMA 組み込み関数 460
LOW 組み込み関数 461
LOWER2 組み込み関数 462
LOWERCASE 組み込み関数 462

M

M (混合) ストリング定数 43
MACCOL マクロ機能組み込み関数 552
MACLMAR マクロ機能組み込み関数 552
MACRMAR マクロ機能組み込み関数 553
MAIN オプション 143
MARGINS キーワード
  ANSWER プリプロセッサー・ステートメントの 546
MAX 組み込み関数 463

MAXEXP 組み込み関数 463
MAXLENGTH 組み込み関数 464
MEMINDEX 組み込み関数 465
MEMSEARCH 組み込み関数 466
MEMSEARCHR 組み込み関数 466
MEMVERIFY 組み込み関数 467
MEMVERIFYR 組み込み関数 468
MIN 組み込み関数 469
MINEXP 組み込み関数 469
MOD 組み込み関数 470
MPSTR 組み込み関数 471
MULTIPLY 組み込み関数 472

N

NAME 条件 378
NEW タイプ付き関数 534
NOCHARGRAPHIC オプション 140
NODESCRIPTOR オプション 141
NOEXECOPS オプション 143
NOINIT 属性 190
NOINLINE オプション 141
NOMAP オプション 143
NONASSIGNABLE 属性 260
NONCONNECTED (NONCONN) 属性 263
NONVARYING (NONVAR) 属性 38
NOPRINT ディレクティブ 227
NORESCAN オプション 558
NORMAL 属性 261
NOT 演算子 67
NOTE ディレクティブ 228
NULL 組み込み関数 472

O

OFFSET 組み込み関数 472
OFFSET 属性 256
OFFSETADD 組み込み関数 473
OFFSETDIFF 組み込み関数 473
OFFSETSUBTRACT 組み込み関数 473
OFFSETVALUE 組み込み関数 474
OFL (OVERFLOW) 条件 379
OMITTED 組み込み関数 474
ON ステートメント 360
ON ユニット
  動的に派生した 362
  ヌル 361
  ファイル変数の 362
  有効範囲 361
ONCHAR 疑似変数 474
ONCHAR 組み込み関数 474
ONCODE 組み込み関数 475
  使用法 357
ONCONDCOND 組み込み関数 475
ONCONDID 組み込み関数 475
ONCOUNT 組み込み関数 476
ONFILE 組み込み関数 476
ONGSOURCE 疑似変数 477
ONGSOURCE 組み込み関数 477
ONKEY 組み込み関数 477
ONLOC 組み込み関数 478
ONSOURCE 疑似変数 479
ONSOURCE 組み込み関数 478
ONSUBCODE 組み込み関数 479
ONWCHAR 疑似変数 480
ONWCHAR 組み込み関数 480
ONWSOURCE 疑似変数 481
ONWSOURCE 組み込み関数 480
OPEN ステートメント 286
OPTIMIZATION、条件の発生 359
OPTION ディレクティブ 229
OPTIONAL 属性 124
OPTIONS オプション
  構文 135
  説明 135
  特性リスト 135
  ASSEMBLER 138
  BEGIN ステートメント 135
  BYADDR 139
  BYVALUE 139
  CHARGRAPHIC 140
  COBOL 140
  DESCRIPTOR 141
  ENTRY 宣言 135
  FORTRAN 141
  FROMALIEN 141
  INLINE 141
  IRREDUCIBLE 144
  LINKAGE 142
  MAIN 143
  NOCHARGRAPHIC 140
  NODESCRIPTOR 141
  NOEXECOPS 143
  NOINLINE 141
  NOMAP 143
  ORDER 143
  PROCEDURE ステートメント 138
  RECURSIVE 107
  REDUCIBLE 144
  REENTRANT 144
  REORDER 143
  RETCODE 144
  WINMAIN 144
OPTIONS オプション、ENTRY ステートメント 136
OPTIONS 属性 135
ORDER オプション 143
ORDINAL 属性 153
ORDINALNAME 組み込み関数 481
ORDINALPRED 組み込み関数 481
ORDINALSUCC 組み込み関数 482
OTHERWISE ステートメント
  SELECT ステートメント内の 233
OUTPUT 属性 283
OVERFLOW (OFL) 条件 379
OVERFLOW 条件接頭語 358

P

P フォーマット項目 335
PACKAGE ステートメント
  説明 96
  有効な OPTIONS オプション 137
  例 98
PACKAGENAME 組み込み関数 482
PAGE オプション 309
PAGE キーワード、ANSWER ステートメントの 546
PAGE ディレクティブ 230
PAGE フォーマット項目 336
PAGENO 組み込み関数 482
PAGESIZE 指定、OPEN ステートメントでの 287
PARAMETER 属性 102
PARMSET マクロ機能組み込み関数 554
PICSPEC 組み込み関数 482
PICTURE (PIC) 属性 39
PLACES 組み込み関数 483
PLIASCII 組み込みサブルーチン 484
PLICANC 組み込みサブルーチン 484
PLICKPT 組み込みサブルーチン 485
PLIDELETE 組み込みサブルーチン 485
PLIDUMP 組み込みサブルーチン 485
PLIEBCDIC 組み込みサブルーチン 485
PLIFILL 組み込みサブルーチン 486
PLIFREE 組み込みサブルーチン 486
  基底付き変数の場合 250
PLIMOVE 組み込みサブルーチン 487
PLIOVER 組み込みサブルーチン 487
PLIREST 組み込みサブルーチン 488
PLIRETC 組み込みサブルーチン 488
PLIRETV 組み込み関数 488
PLISAXA 組み込みサブルーチン 488
PLISAXB 組み込みサブルーチン 489
PLISRTA 組み込みサブルーチン 489
PLISRTB 組み込みサブルーチン 490
PLISRTC 組み込みサブルーチン 490
PLISRTD 組み込みサブルーチン 490
PL/I アプリケーション
  構造の図解 94
  説明 93
POINTER (PTR) 組み込み関数 490
POINTER (PTR) 属性 250
POINTERADD (PTRADD) 組み込み関数 491
  ポインター演算との併用 60

POINTERDIFF (PTRDIFF) 組み込み関数 492
POINTERSUBTRACT (PTRSUBTRACT) 組み込み関数 492
POINTERVALUE (PTRVALUE) 組み込み関数 492
  使用法 60
POLY 組み込み関数 492
POP ディレクティブ 230
POS (POSITION) 属性 264
POSITION (POS) 属性 264
PRECISION (PREC) 組み込み関数 493
PRECISION 組み込み関数
  使用法 66
PRECISION 属性
  序数 149
  説明 31
PRED 組み込み関数 493
PRESENT 組み込み関数 494
PRINT 属性 324
PRINT ディレクティブ 230
PROC (PROCEDURE) ステートメント 99
PROCEDURE (PROC) ステートメント
  使用法 94
  説明 99
  有効な OPTIONS 137
PROCEDURE ステートメント 138
PROCEDURENAME (PROCNAME) 組み込み関数 494
PROCESS ディレクティブ 231
PROD 組み込み関数 494
PTR (POINTER) 属性 250
PTRADD (POINTERADD) 組み込み関数
  ポインター演算との併用 60
PTRVALUE (POINTERVALUE) 組み込み関数
  使用法 60
PUSH ディレクティブ 231
PUT ステートメント
  ストリング 320
  データ・ディレクティブ 316
  編集ディレクティブ 320
  リスト・ディレクティブ 323
  STREAM 出力 305
PUTENV 組み込み関数 495

Q

QUOTE マクロ機能組み込み関数 554

R

R (オーバーパンチ) ピクチャー文字 353
R フォーマット項目 336
RADIX 組み込み関数 495
RAISE2 組み込み関数 495
RANDOM 組み込み関数 496
RANGE オプション 177
RANK 組み込み関数 496
READ ステートメント 295
REAL 疑似変数 497
REAL 組み込み関数 496
REAL 属性 31
RECORD 条件 380
RECORD 属性 283
RECURSIVE オプション 107
RECURSIVE 属性 107
REDUCIBLE (RED) オプション 144
REENTRANT オプション 144
REFER オプション
  説明 253
  AREA 属性での 256
RELEASE ステートメント
  外部プロシージャーの動的ロード 109
  制約事項 109
  説明 111
  例 111
REM 組み込み関数 497
REORDER オプション 143
REPATTERN 組み込み関数 497
REPEAT オプション 212
REPEAT 組み込み関数 498
REPEAT マクロ機能組み込み関数 554
REPLY オプション 209
RESCAN オプション 558
RESERVED 属性 170
RESERVES オプション 97
RESIGNAL ステートメント 364
RESPEC タイプ付き関数 534
RETCODE オプション 144
RETURN ステートメント
  関数からの戻り 134
  サブルーチンとの併用 134
  使用法 106
  説明 134
  プリプロセッサー・プロシージャーでの使用 544
RETURNS オプション
  説明 144
RETURNS 属性 144
REVERSE 組み込み関数 499
REVERT ステートメント 363
REWRITE ステートメント
  説明 295
RIGHT 組み込み関数 499
ROUND 組み込み関数 500

S

S ピクチャー文字 350
SAMEKEY 組み込み関数 501
SCALARVARYING オプション 293
SCALE 組み込み関数 501
SCAN オプション 558
SEARCH 組み込み関数 501
SEARCHR 組み込み関数 503
SECS 組み込み関数 503
SECSTODATE 組み込み関数 504
SECSTODAYS 組み込み関数 505
SELECT グループ 233
SELECT ステートメント
  説明 233
  例 234
SEQL (SEQUENTIAL) 属性 284
SEQUENTIAL (SEQL) 属性 284
SET オプション
  説明 251
  ポインター参照の指定 110
  ALLOCATE ステートメントの使用法 251
  LOCATE ステートメントの使用法 296
  READ ステートメントの使用法 295
SIGN 組み込み関数 505
SIGNAL ステートメント 364
SIGNED 組み込み関数 505
SIGNED 属性
  序数 149
  説明 32
  データ・ストレージ所要量 33
SIN 組み込み関数 506
SIND 組み込み関数 506
SINF 組み込み関数 506
SINH 組み込み関数 506
SIZE 組み込み関数 507
SIZE 条件 381
SIZE 条件接頭語 358
SIZE タイプ付き関数 534
SKIP オプション 309
SKIP キーワード、ANSWER ステートメントの 546
SKIP ディレクティブ 235
SKIP フォーマット項目 337
SNAP オプション、ON ステートメントの 360
SOURCEFILE 組み込み関数 508
SOURCELINE 組み込み関数 508
SQRT 組み込み関数 508
SQRTF 組み込み関数 509
STATIC 属性
  説明 239
  INITIAL 属性との併用 274

STOP ステートメント
使用法 106
STORAGE (STG) 組み込み関数 509
STORAGE 条件 382
STREAM 属性 283
STRG (STRINGRANGE) 条件 382
STRING オプション
説明 310
GET ステートメントでの使用 304
PUT ステートメントでの使用 305
STRING 疑似変数 510
STRING 組み込み関数 509
STRINGRANGE (STRG) 条件 265, 382
STRINGRANGE 条件接頭語 358
STRINGSIZE (STRZ) 条件 265, 383
STRINGSIZE 条件接頭語 358
STRZ (STRINGSIZE) 条件 383
SUBRG (SUBSCRIPTRANGE) 条件 265, 384
SUBSCRIPTRANGE (SUBRG) 条件 265, 384
SUBSCRIPTRANGE 条件接頭語 358
SUBSTR 疑似変数 511
SUBSTR 組み込み関数 511
SUBSTR マクロ機能組み込み関数 555
SUBTRACT 組み込み関数 512
SUCC 組み込み関数 512
SUM 組み込み関数 512
SYSIN 291
SYSNULL 組み込み関数 512
SYSPARM マクロ機能組み込み関数 555
SYSPRINT 291
SYSTEM オプション、ON ステートメントの 360
SYSTEM 組み込み関数 513
SYSTEM マクロ機能組み込み関数 556
SYSVERSION マクロ機能組み込み関数 556

T

T (オーバーパンチ) ピクチャー文字 353
TALLY 組み込み関数 513
TAN 組み込み関数 514
TAND 組み込み関数 514
TANF 組み込み関数 514
TANH 組み込み関数 514
TASK 属性 393
THREAD オプション 390
THREADID 組み込み関数 515
TIME 組み込み関数 515
TINY 組み込み関数 515
TITLE オプション 110
TITLE 指定、OPEN ステートメントでの 287
TO オプション 211
TO オプション、INITIAL 属性の 273
TRANSLATE 組み込み関数 516
TRANSLATE マクロ機能組み込み関数 556
TRANSMIT 条件 385
TRIM 組み込み関数 517
TRUNC 組み込み関数 517
TSTACK オプション 391
TYPE 疑似変数 518
TYPE 組み込み関数 518
TYPE 属性 152

U

UFL (UNDERFLOW) 条件 387
UNALIGNED 属性
記憶位置合わせ要件 172
構造体マッピングへの影響 195
説明および構文 172
例 175
UNALLOCATED 組み込み関数 518
UNBUF (UNBUFFERED) 属性 284
UNBUFFERED (UNBUF) 属性 284
UNDEFINEDFILE (UNDF) 条件 386
UNDERFLOW (UFL) 条件 387
UNDERFLOW 条件接頭語 358
UNDF (UNDEFINEDFILE) 条件 386
UNION 属性 187
UNION、同義語 187
UNSIGNED 組み込み関数 519
UNSIGNED 属性
序数 149
説明 32
データ・ストレージ所要量 33
UNSPEC 疑似変数 521
UNSPEC 組み込み関数 519
UNTIL オプション
説明 210
タイプ 2 の DO 指定との併用 212
UPDATE 属性 283
UPPERCASE 組み込み関数 521
UPTHRU オプション
説明 211
タイプ 3 の DO 指定との併用 215
例 219
UPTHRU、序数との併用 220

V

V ピクチャー指定文字 344
V フォーマット項目 338
VALID 組み込み関数 522
VALIDDATE 組み込み関数 522
VALUE オプション 177, 179
VALUE 属性
序数 149
説明 49
VARGLIST 組み込み関数 523
VARGSIZE 組み込み関数 523
VARIABLE 属性 53
VARYING (VAR) 属性 38
VARYINGZ (VARZ) 属性 38
VERIFY 組み込み関数 524
VERIFY マクロ機能組み込み関数 557
VERIFYR 組み込み関数 524

W

WAIT ステートメント 392
WCHARVAL 組み込み関数 525
WEEKDAY 組み込み関数 525
WHEN ステートメント
説明 233
WHIGH 組み込み関数 526
WHILE オプション
説明 210
タイプ 2 の DO 指定との併用 212
WIDECHAR (WCHAR) 組み込み関数 526
WIDECHAR (WCHAR) 属性
説明 37
WINMAIN オプション 144
WLOW 組み込み関数 527
WRITE ステートメント
説明 295
WX (ワイド文字 16 進) ストリング定数 45

X

X (16 進) の文字ストリング定数 40
X ピクチャー指定文字 342
X フォーマット項目 338
XMLCHAR 組み込み関数 527
XN (2 進 16 進数) 定数 34
XU (2 進 16 進数) 定数 34

Y

Y ゼロ置換ピクチャー文字 353
Y4DATE 組み込み関数 528
Y4JULIAN 組み込み関数 529
Y4YEAR 組み込み関数 530

Z

Z ゼロ消去ピクチャー文字 346
ZDIV (ZERODIVIDE) 条件 387
ZERODIVIDE (ZDIV) 条件 387

ZERODIVIDE 条件接頭語 358

[特殊文字]

$ (ピクチャー文字) 350
& (AND 記号)
  演算子としての使用 15
  ASCII および EBCDIC 値 11
& (ビット演算子: AND) 67
&= (AND と代入)、複合記号の作成 12
( ) (囲み記号)
  区切り文字としての使用 14
  ASCII および EBCDIC 値 11
* (乗算)
  演算子としての使用 15
  算術演算での使用 60
  ASCII および EBCDIC 値 11
* ゼロ消去ピクチャー文字 346
*PROCESS ディレクティブ 231
** (累乗演算)
  演算子としての使用 15
  算術演算での使用 60
  複合記号の作成 12
**= (指数化と代入)、複合記号の作成 12
*/ (コメントの終了)、複合記号の作成 12
*= (乗算と代入)、複合記号の作成 12
+ (加算)
  演算子としての使用 15
  算術演算での使用 60
  ASCII および EBCDIC 値 11
+ (ピクチャー文字) 350
+= (加算と代入)、複合記号の作成 12
, (分離文字)
  区切り文字としての使用 14
  ASCII および EBCDIC 値 11
- (減算)
  演算子としての使用 15
  算術演算での使用 60
  ASCII および EBCDIC 値 11
-= (減算と代入)、複合記号の作成 12
-> (ロケーター)
  区切り文字としての使用 14
  ロケーター修飾 248
-> (ロケーター)、複合記号の作成 12
. (名前修飾子、小数点)
  区切り文字としての使用 14
  ASCII および EBCDIC 値 11
/ (除算)
  演算子としての使用 15
  算術演算での使用 60
  ASCII および EBCDIC 値 11
/ (挿入文字) 347
/* (コメントの開始)、複合記号の作成 12
/* */ (コメント)
  区切り文字としての使用 14
  構文 16
/= (除算と代入)、複合記号の作成 12
: (接頭部、次元、および範囲区切り記号)
  使用法 14
  ASCII および EBCDIC 値 11
; (ステートメント終了文字)
  区切り文字としての使用 14
  ASCII および EBCDIC 値 11
= (等号)
  演算子としての使用 15
  区切り文字としての使用 14
  比較演算での使用 68
  ASCII および EBCDIC 値 11
? (マクロ・トリガー文字)
  ASCII および EBCDIC 値 11
> (より大記号)
  演算子としての使用 15
  ASCII および EBCDIC 値 11
>= (より大か等しい記号) 12
< (より小記号)
  演算子としての使用 15
  比較演算での使用 68
  ASCII および EBCDIC 値 11
<= (より小か等しい記号) 12
  演算子としての使用 15
  比較演算での使用 68
| (ビット演算子: OR) 67
|| (論理 OR 記号)
  演算子としての使用 15
  ASCII および EBCDIC 値 11
|= (OR と代入)、複合記号の作成 12
|| (連結)
  演算子としての使用 15
  複合記号の作成 12
  連結演算子での使用 70
||= (連結と代入)、複合記号の作成 12
% ディレクティブ
  %INCLUDE 225
  %LINE 227
  %NOPRINT 227
  %NOTE 228
  %OPTION 229
  %PAGE 230
  %POP 230
  %PRINT 230
  %PROCESS 231
  %PUSH 231
  %SKIP 235
% (% ステートメント用)
  区切り文字としての使用 14
  ASCII および EBCDIC 値 11
%ACTIVATE ステートメント 558
%DEACTIVATE ステートメント 559
%DECLARE ステートメント 560
%DO ステートメント 562
%END ステートメント 562
%GO TO ステートメント 563
%IF ステートメント 564
%IF ステートメントの ELSE 節 564
%IF ステートメントの THEN 節 564
%INCLUDE ステートメント 564
%INCLUDE ディレクティブ 225
%INSCAN ステートメント 565
%ITERATE ステートメント 566
%LEAVE ステートメント 566
%LINE ディレクティブ 227
%NOPRINT ディレクティブ 227
%NOTE ステートメント 567
%NOTE ディレクティブ 228
%OPTION ディレクティブ 229
%PAGE ディレクティブ 230
%POP ディレクティブ 230
%PRINT ディレクティブ 230
%PROCEDURE ステートメント 543
%PROCESS ディレクティブ 231
%PUSH ディレクティブ 231
%REPLACE ステートメント 568
%SELECT ステートメント 568
%SKIP ディレクティブ 235
%XINCLUDE ステートメント 568
%XINSCAN ステートメント 569
%代入ステートメント 559
%ヌル・ステートメント 567
' ' (囲み定数)
  ASCII および EBCDIC 値 11
_ (下線、区切り)、ASCII および EBCDIC 値 11
¬ (NOT 記号)
  ASCII および EBCDIC 値 11
¬ (ビット演算子: NOT、XOR) 67
¬ (論理 NOT EOR 記号)
  演算子としての使用 15
  ASCII および EBCDIC 値 11
¬= (等しくない記号)
  演算子としての使用 15
  説明 12
  比較演算での使用 68
¬> (より大ではない記号)
  演算子としての使用 15
  説明 12
  比較演算での使用 68
¬< (より小ではない記号)
  演算子としての使用 15
  説明 12
  比較演算での使用 68
' (引用符)
  二重 11
" (二重引用符)
  ASCII および EBCDIC 値 11

プログラム番号: 5655-H31

Printed in Japan

Enterprise PL/I for z/OS

SC27-1456
Licensed Program Specifications

SC88-9123
プログラミング・ガイド

GC88-9124
コンパイラーおよびランタイム 移行ガイド

SC88-9126
言語解説書

SC88-9127
メッセージおよびコード

SC88-9126-04

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12