iSeries

VisualAge RPG 言語解説書

Windows 版バージョン 5

SC88-5608-04
(英文原典：SC09-2451-04)

iSeries

VisualAge RPG 言語解説書

Windows 版バージョン 5

SC88-5608-04
(英文原典：SC09-2451-04)

本書は、IBM WebSphere Development Studio Client for iSeries のバージョン 4、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本書は、 SC88-5608-03 の改訂版です。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

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

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(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： SC09-2451-04
iSeries
VisualAge RPG Language Reference
Version 5 for Windows

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 2004.4

この文書では、平成明朝体™W3、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™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

目次

本書について

本書には、Windows® オペレーティング・システムで VisualAge RPG コンパイラーを使用して実装される RPG IV 言語に関する情報が記載されています。

本書には以下が含まれます。
- 文字セット、記号名、予約語、コンパイラー指示、標識などの言語基本エレメント
- データ・タイプおよびデータ形式
- エラーおよび例外処理
- サブプロシージャー
- 仕様
- 組み込み関数、式、および命令コード

本書は、VisualAge RPG プログラム言語に詳しいプログラマーを対象としています。

この解説は、VisualAge RPG 言語の詳細説明を提供します。VisualAge RPG コンパイラーの使用法や ILE RPG プログラムの VisualAge RPG プログラムへの変換方法についての情報は提供していません。これらのトピックの詳細については、*VisualAge for RPG プログラミング*, SC88-5607-05 を参照してください。

本書を使用する前に、VisualAge RPG アプリケーションのタスクに慣れておいてください。*VisualAge for RPG プログラミング* およびオンライン・ヘルプを参照してください。

前提条件および関連情報

iSeries および AS/400e の技術情報を調べるための開始点として iSeries Information Center を使用してください。Information Center をアクセスする方法には、以下の 2 つがあります。
- 次の Web サイトから:

  `http://www.ibm.com/eserver/iseries/infocenter`
- OS/400 オーダーで配送された CD-ROM から:

  iSeries Information Center, SK88-8055-03.

iSeries Information Center には、CL コマンド、システム・アプリケーション・プログラミング・インターフェース (API)、論理区画、クラスター、Java™、TCP/IP、Web サービス、およびセキュア・ネットワークなどのアドバイザーおよび重要なトピックが入っています。また、関連した IBM® Redbooks へのリンクおよび Technical Studio や IBM ホーム・ページなどのその他の Web サイトへのインターネット・リンクも含まれています。

VisualAge RPG ライブラリー

VisualAge RPG ライブラリーには、以下の資料が含まれています。

VisualAge for RPG プログラミング

本書には、VisualAge RPG を使用してアプリケーションを作成する場合の具体的な情報が記載されています。これには、アプリケーション開発サイクルのすべての段階で実行する必要があるステップが、設計からパッケージおよび分配まで説明されています。VisualAge RPG アプリケーション開発の概念と工程を明らかにするために、プログラミング例が収録されています。

ADTS/CS VisualAge for RPG パーツ解説書

本書には、VisualAge RPG の**パーツ**、**パーツ属性**、**パーツ・イベント**、および**イベント属性**に関する情報が記載されています。本書は、VisualAge RPG を使用してアプリケーションを開発しているすべての方々の参照資料です。

VisualAge for RPG WINDOWS 版 言語解説書

本書には、VisualAge RPG コンパイラーを使用して実装される RPG IV 言語に関する情報が記載されています。 次の事項が含まれています。

- 文字セット、記号名および予約語、コンパイラー指示、ならびに標識などの言語基本エレメント
- データ・タイプおよびデータ形式
- エラーおよび例外処理
- 仕様
- 組み込み関数、式、および命令コード

製品全体の概要については、*WebSphere Development Studio Client for iSeries ご使用に際して*を参照してください。

関連資料のリストについては、本書の終わりにある参考文献を参照してください。

また以下のオンライン・ソースでも IBM WebSphere Development Studio Client for iSeries についての最新の情報を見つけることができます。

「Development Studio Client」ホーム・ページ

```
ibm.com/software/ad/wdsc/
```

オンライン情報へのアクセス

VisualAge RPG には、種々のオンライン・ブックおよびオンライン・ヘルプが含まれています。プロダクトの使用中にヘルプにアクセスしたり、またプロダクトの使用中または別途に本書を表示することができます。

オンライン・ブックの使用法

オンライン・ブックを表示するためには、次のいずれかを実行してください。

- VisualAge RPG GUI Designer またはエディター・ウィンドウの**ヘルプ** プルダウン・メニューから本書の名前を選択してください。

- **「スタート」**メニューから本書にアクセスしてください。**「プログラム」→「IBM WebSphere Development Studio Client for iSeries」**を選択します。次に**「文書」**を選択します。

PDF 形式の資料

VisualAge RPG 資料は、URL `http://www.ibm.com/eserver/iseries/infocenter` の iSeries Information Center から PDF の形式で使用可能です。

注: ワークステーションで資料を PDF 形式で表示するには、Adobe Acrobat Reader (Windows 版 3.01 以降) が必要です。システム設置場所にこのプログラムがない場合には、Adobe Systems Web サイト (http://www.adobe.com) からコピーをダウンロードできます。

以下の VisualAge RPG 資料は、PDF 形式で使用可能です。

- *VisualAge for RPG プログラミング*
- *ADTS/CS VisualAge for RPG パーツ解説書*
- *VisualAge for RPG WINDOWS 版 言語解説書*

この製品については、*WebSphere Development Studio Client for iSeries ご使用に際して*, SD88-5054-03を参照してください。

オンライン・ヘルプの使用法

オンライン・ヘルプは VisualAge RPG のすべての領域から使用することができます。特定のウィンドウ、ダイアログ・ボックス、またはプロパティー・ノートブックのヘルプを表示するには、**「ヘルプ」**プッシュボタン (使用可能な場合) を選択します。

注: HTML 形式のヘルプを表示するには、ワークステーションには Netscape Navigator 4.04 以降、Microsoft® Internet Explorer 4.01 以降などのフレーム使用可能 Web ブラウザーが必要です。(推奨ブラウザーは、Netscape Navigator 4.6 または Internet Explorer 5.0 です。)

ヘルプ情報の使用法

任意の時点でヘルプ情報を表示するためには、 F1 キーを押してください。表示されるヘルプは、入力フォーカスを持つインターフェースの領域に固有のものです。入力フォーカスはメニュー項目、ウィンドウ、ダイアログ・ボックス、およびプロパティー・ノートブック上にあるか、あるいはこれらの特定の部分にあります。

ダイアログ・ボックスのヘルプ情報については、ウィンドウの右上隅の疑問符 (使用可能な場合) をクリックしてください。 マウスの矢印の隣に疑問符 (?) が現れます。ワードまたはフィールドをクリックすると、その特定のフィールドについてのヘルプ情報が表示されます。

言語依存ヘルプの使用法

言語依存ヘルプを表示するためには、編集ウィンドウで F1 キーを押してください。カーソルが命令コードに位置付けられている場合には、その命令コードのヘルプが表示されます。その他の場合は、現在の指定のヘルプが表示されます。

このリリースの新しい機能

VisualAge RPG は、現在「IBM WebSphere™ Development Studio Client for iSeries」製品の一部になっています。この製品の詳細については、*WebSphere Development Studio Client for iSeries ご使用に際して* オンライン資料および関連したヘルプ・トピックを参照してください。バージョン管理情報は、製品の WebSphere Development クライアント・ファミリーを反映するように変更されています。

この資料には、前のリリースの Readme からの情報と、その他の技術的な修正が含まれています。

第 1 部 VisualAge RPG 言語の紹介

この項では、下記のような VisualAge® RPG (VARPG) 言語の基本エレメントの一部について説明します。

- 文字セット
- 記号名と予約語
- コンパイラー指示
- 標識
- サブプロシージャー

第 1 章 記号名と予約語

VisualAge RPG 言語に有効な文字セットは以下から構成されます。

文字 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

記号名には小文字を使用できますが、それらはコンパイル中に大文字に変換されます。

数字 0 1 2 3 4 5 6 7 8 9

文字 + − * , . ’ & / $ # : @ _ > < = ( ) %

ブランク文字

記号名

記号名は、プログラムまたはプロシージャー中の特定のデータを固有に識別します。その目的は、ユーザーがそのデータにアクセスできるようにすることにあります。すべての記号名には次の規則が適用されます。

- 名前の先頭文字は英字、$、#、または @ でなければなりません。
- 残りの文字は英字、数字、または下線 (_) でなければなりません。
- 名前は仕様フォームの項目内で左寄せされなければなりません。ただし、名前を浮動にできるフィールドの場合 (定義仕様、キーワード・フィールド、および拡張演算項目 2 フィールド) を除きます。
- 記号名を予約語にすることはできません。
- 記号名は 1 - 4096 文字とすることができます。一般的な限界は、名前を定義するために使用する項目のサイズによって決められます。最大 15 文字までの名前は、定義仕様またはプロシージャー仕様の名前項目に指定することができます。15 文字より長い名前の場合には、継続仕様を使用してください。
- 記号名は、それが定義されているプロシージャー内では固有でなければなりません。

表 1 記号名と追加の制約事項をリストします。

表 1. 記号名に関する制約事項

配列	配列名を文字 TAB で始めることはできません。
条件付きコンパイル名	条件付きコンパイルに使用する記号名は、他の記号名と何の関係も持ちません。条件付きコンパイルの名前は最大 50 文字までの長さにすることができます。
データ構造	データ構造名は一度しか定義できません。
例外出力レコード	同じ EXCEPT 名を複数の例外出力レコードに割り当てることができます。
フィールド	• フィールド名は、その名前を使用しているそれぞれの定義が同じデータ・タイプ、同じ長さ、同じ小数点以下の桁数の場合には、複数回定義することができます。同じ名前を使用する定義はすべて単一のフィールド (すなわち、ストレージ内の同じ領域) を参照します。しかし、定義仕様では一度しか定義することができません。 • データ構造が修飾 (QUALIFIED または LIKEDS で定義) されていない限り、フィールドは、データ構造サブフィールドとして一度だけ定義することができます。この場合、サブフィールドが使用されるときには、それを修飾 (*dsnamesubfieldname* の形式で指定) しなければなりません。 • サブフィールド名を *ENTRY PLIST パラメーターで結果フィールドとして指定することはできません。 VisualAge RPG コンパイラーは、パーツと同じ名前で静的テキストと入力フィールドのパーツ用のグローバル・フィールドを作成します。ソースにおけるこれらのフィールド名の明示的定義は、一致しなければなりません。
キー・フィールド・リスト	キー・フィールド・リスト (KLIST) 名には追加の制約事項はありません。
ラベル	ラベル名には追加の制約事項はありません。
名前付き固定情報	名前付き固定情報には追加の制約事項はありません。
パラメーター・リスト	パラメーター・リスト (PLIST) には追加の制約事項はありません。
プロトタイプ名	プロトタイプ名には追加の制約事項はありません。
レコード名	レコード名は、プログラム中の 1 つのファイルのみに存在することができます。
サブルーチン	アクション・サブルーチン名については 488 ページの『BEGACT (アクション・サブルーチンの開始)』 を、ユーザー・サブルーチン名については 492 ページの『BEGSR (ユーザー・サブルーチンの開始)』 を参照してください。
テーブル	テーブル名には 3 - 10 文字を含めることができ、文字 TAB で始まっていなければならず、サブプロシージャー内で定義することはできません。
ウィンドウ	コンポーネントの GUI 定義に定義されているウィンドウ名は、プロシージャー内でも、プログラム内のシンボル名として予約されます。

特殊な機能を持つ語および予約語

以下は、特殊な機能を持つ語の要約です。

組み込み関数の特殊語

*ALL および *NULL 特殊語は組み込み関数で使用されます。組み込み関数の詳細については、399 ページの『第 25 章 組み込み関数』 を参照してください。

日付と時刻の特殊語

次の特殊語は日付、時刻、およびタイム・スタンプ・フィールドで使用されます。

*CDMY	*CMDY	*CYMD
*CYMD0	*DMY	*ISO
*LONGJUL	*MDY	*EUR
*JIS	*USA	*HMS
*JUL	*YMD	

日付形式の詳細については、227 ページの『DATFMT(fmt{区切り文字})』 を参照してください。

式

NOT 特殊語は式で使用することができます。式の詳細については、383 ページの『第 24 章 式』 を参照してください。

ファイルの位置決めの特殊語

*START および *END 特殊語は、ファイルの位置決めに使用することができます。ファイルの位置決めの詳細については、6 ページの『ファイルの位置決め』を参照してください。

暗黙のリテラル

表意定数は、長さを参照しないで指定できる暗黙のリテラルです。表意定数の詳細については、161 ページの『表意定数』を参照してください。

*ALLX'x1..'	*DARKGREEN	*OFF
*ALLG'K1K2'	*DARKGRAY	*ON
*ALL'X..'	*DARKPINK	OK(*O)
*ABORT	*DARKRED	*PALEGRAY
*BLACK	*ENTER	*PINK
*BLANK	*GREEN	*RED
*BLANKS	*HALT	*RETRY
*BLUE	*HIVAL	*WARN
*BROWN	*INFO	*WHITE
*CANCEL	*IGNORE	*YELLOW
*CYAN	*LOVAL	*YESBUTTON

*DARKBLUE	*NOBUTTON	*ZERO
*DARKCYAN	*NULL	*ZEROS

標識の予約語

*IN および *INxx 予約語によって、標識をデータとして参照することができます。詳細については、 27 ページの『データとして参照される標識』 を参照してください。

ジョブ日付の予約語

次の予約語によって、ジョブ日付またはその一部にアクセスすることができます。詳細については、 8 ページの『ユーザー日付の特殊語』 を参照してください

UDATE	*DATE
UMONTH	*MONTH
UYEAR	*YEAR
UDAY	*DAY

ページ番号の予約語

PAGE および PAGE1-PAGE7 予約語は、レポートのページの番号づけまたは出力フィールドの順次番号づけに使用することができます。詳細については、 7 ページの『PAGE、PAGE1-PAGE7 予約語』を参照してください。

パラメーターの受け渡しの特殊語

*OMIT、*RIGHTADJ、*STRING、および *VARSIZE 特殊語は、パラメーターの受け渡しに使用されます。

フィールドの配置

*PLACE によって、出力レコードのフィールドの反復配置が可能です。詳細については、 330 ページの『*PLACE』 を参照してください。

すべてのフィールドの書き出し

*ALL によって、外部記述ファイルに対して定義されたすべてのフィールドを出力で書き出すことができます。表意定数の詳細については、 162 ページの『表意定数の規則』 を参照してください。

ファイルの位置決め

*START および *END は、OS/400™ データベース・ファイルの位置を変更します。

ファイルが非キー付きファイルの場合は、*START および *END はそれぞれファイルの始めと終わりに位置づけられます。ファイルがキー付きファイルの場合は、*START および *END はそれぞれキー順アクセス・パスの始めと終わりに位置づけられます。

PAGE、PAGE1-PAGE7 予約語

PAGE は、レポートのページの番号づけ、ファイルの出力レコードの順次番号づけ、または出力フィールドの順次番号づけに使用されます。ページ排出は起こりません。

使用できる 8 つの PAGE フィールド (PAGE、PAGE1、PAGE2、PAGE3、PAGE4、PAGE5、PAGE6、および PAGE7) は、別のタイプの出力ページの番号づけまたは別の印刷装置ファイルのページ番号づけに使用することができます。

PAGE フィールドは、出力仕様の 30-43 桁目、あるいは入力仕様または演算仕様で指定することができます。

PAGE フィールドには次の規則が適用されます。
- ページ番号は、他に指定のない限り 1 から始まります。
- 新しいページごとに自動的に 1 が加算されます。
- PAGE フィールドは任意の長さにすることができます。
- PAGE フィールドは小数点以下の桁数がゼロでなければなりません。
- PAGE フィールドが出力仕様だけに指定される場合には、4 桁の小数点以下の桁数がゼロの数値フィールドとして取り扱われます。

PAGE の語はさまざまな方法で使用することができます。
- ページ番号を 1 以外から始めるには、PAGE フィールドの値を必要な開始ページより 1 だけ小さく設定します。
- ジョブの任意の位置からページの番号づけを再開するには次のようにします。
  - 後にブランクを指定します (出力仕様の 45 桁目) 。
  - PAGE フィールドを演算仕様の演算の結果フィールドとして指定します。
  - 出力フィールドに出力標識を指定します (図 2 を参照) 。出力標識をオンに設定すると、PAGE フィールドが 1 にリセットされます。PAGE フィールドは常に書き出されるので、PAGE フィールドの印刷の制御に出力標識を使用することはできません。
  - PAGE フィールドを入力フィールドとして指定します (図 1 を参照) 。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++Sq..RiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++....FrPlMnZr....
IINPUT         50   1  CP
I                                  2    5 0PAGE
```

図 1. ページ・レコード記述

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OFilename++EF..N01N02N03Excnam++++B++A++Sb+Sa+.........................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O*  標識 15 がオンになると、PAGE フィールドがゼロに設定されて、フィールド
O*  が印刷される前に 1 に設定されます。標識 15 がオフの場合には、印刷
O*  される前に PAGE フィールドの内容に 1 が加えられます。
OPRINT     E    99                        01
O               15      PAGE          1     75
```

図 2. PAGE フィールドをゼロにリセット

ユーザー日付の特殊語

プログラムの日付は、実行時にユーザー日付の特殊語を使用して指定することができます。ユーザー日付の特殊語は、UDATE、*DATE、UMONTH、*MONTH、UDAY、*DAY、UYEAR、および *YEAR です。

ユーザー日付の特殊語は、ジョブ記述に指定されているジョブ日付にアクセスします。ユーザー日付は出力として書き出すことができます。

ユーザー日付の特殊語は、アプリケーションが実行を開始したときに設定されます。これらは、プログラムが深夜を超えて実行される場合、またはジョブ日付が変更される場合に更新されません。プログラムが実行しているときの時刻と日付を取得するには TIME 命令を使用してください。TIME 命令の詳細については、697 ページの『TIME (時刻)』 を参照してください。

日付形式 UDATE および *DATE を指定するには、制御仕様で DATEDIT キーワードを使用します。

DATEDIT	UDATE 形式	*DATE 形式
*MDY	*MDY	*USA (mmddyyyy)
*DMY	*DMY	*EUR (ddmmyyyy)
*YMD	*YMD	*ISO (yyyymmdd)

このキーワードを指定しない場合には、デフォルトは *MDY です。

ユーザー日付フィールドには次の制約事項が適用されます。

- ユーザー日付フィールドは、日付タイプのフィールドではなく、数値フィールドです。
- ユーザー日付フィールドを変更することはできません。このことは、以下を使用できないことを意味します。
  - 演算の結果フィールドで
  - PARM 命令の演算項目 1 として
  - LOOKUP 命令の演算項目 2 の索引として
  - 出力仕様の後にブランクで
  - 入力フィールドとして
- ユーザー日付フィールド UMONTH、*MONTH、UDAY、*DAY、UYEAR、および *YEAR は、出力仕様の 44 桁目で Y 編集コードで編集することはできません。

ユーザー日付の語はさまざまな方法で使用することができます。

数値フィールドを使用する命令コード
: ユーザー日付特殊語は、数値フィールドを使用する命令コードで、演算仕様の演算項目 1 または演算項目 2 に使用することができます。

UDATE および *DATE の編集	UDATE および *DATE は、出力仕様の 44 桁目に & 編集コードが指定されている場合に、それらを書き込むときに編集することができます。制御仕様の DATEDIT キーワードは、その形式と挿入する区切り文字を決定します。
2 桁の日付フィールドの印刷	2 桁の日付フィールドを印刷するには、出力仕様で UMONTH、*MONTH UDAY、 *DAY、および UYEAR を指定します。
4 桁の日付フィールドの印刷	4 桁の日付フィールドを印刷するには、出力仕様で UMONTH、*MONTH UDAY、*DAY、および UYEAR を指定します。
6 桁の日付フィールドの印刷	6 桁の日付フィールドを印刷するには、出力仕様で UDATE を指定します。月 / 日 / 年、年 / 月 / 日、または日 / 月 / 年の 3 種類の日付形式を使用することができます。制御仕様の DATEDIT キーワードは、その形式を決定します。
8 桁の日付フィールドの印刷	8 桁の日付フィールドを印刷するには、出力仕様で *DATE を指定します。年は 4 桁です。月 / 日 / 年、年 / 月 / 日、または日 / 月 / 年の 3 種類の日付形式を使用することができます。制御仕様の DATEDIT キーワードは、その形式を決定します。
日の印刷	日だけを印刷するには、出力仕様で UDAY または *DAY を指定します。
月の印刷	月だけを印刷するには、出力仕様で UMONTH または *MONTH を指定します。
年の印刷	年だけを印刷するには、出力仕様で YEAR または *YEAR を指定します。

第 2 章 コンパイラー指示

コンパイラー指示ステートメント /FREE... /END-FREE は、フリー・フォームの演算仕様ブロックを示します。コンパイラー指示 /TITLE、/EJECT、/SPACE、/COPY と /INCLUDE、および /INCLUDE によって、コンパイル時にコンパイラー・リストの見出し情報を指定したり、コンパイラー・リストのスペーシングを制御したり、他のファイル・メンバーからレコードを挿入したりできます。条件付きコンパイラー指示ステートメント /DEFINE、/UNDEFINE、/IF、 /ELSEIF、/ELSE、および /EOF によって、ソース・レコードを選択または除外することができます。コンパイラー指示ステートメントは、コンパイル時配列またはテーブル・レコードの前に置かなければなりません。

/FREE... /END-FREE (7 ～ 11 桁目)

桁	項目
7 ～ 11	/FREE または /END-FREE
12 ～ 80	ブランク

/FREE コンパイラー指示は、フリー・フォーム演算仕様ブロックの始まりを指定します。/END-FREE は、そのブロックの終わりを指定します。12 ～ 80 桁目はブランクでなければなりません。残りの桁は、コメントに使用できます。フリー・フォームステートメントの使用の詳細については、318 ページの『フリー・フォーム構文』 を参照してください。

/COPY または /INCLUDE)

/COPY と /INCLUDE 指示は、同じ目的と同じ構文をもちます。使用する指示は、自由に選択できます。

/COPY と /INCLUDE コンパイラー指示を使用すると、コンパイル中のファイル内でその指示が見つかった場所で他のファイルからレコードが挿入されます。このファイルはワークステーションまたは iSeries サーバー上に存在することができます。挿入されたレコードには、COPYNEST キーワードで指定されたネストの最大の深さ (指定がない場合には 32) まで、/COPY および /INCLUDE を含む有効な指定を入れることができます。

アプリケーションの保守を容易にするために、エクスポートされたプロシージャーのプロトタイプを別のソース・メンバーに入れることができます。その場合には、そのメンバーの /COPY または /INCLUDE 指示をエクスポートされたプロシージャーが入っているモジュールとエクスポートされたプロシージャーに対する呼び出しが入っているモジュールの両方に入れるようにしてください。

コピー指示はコンパイラー・リストには印刷されず、指定したファイルの内容と置き換えられます。コピーされたすべてのメンバーは、コンパイラー・リストの COPY メンバー・テーブルに表示されます。

/COPY メンバーは、/COPY 指示がフリー・フォームグループ内でコーディングされていても、デフォルトで固定形式と見なされます。/COPY メンバーにフリー・フォーム仕様が含まれる場合には、それらの仕様を /FREE と /END-FREE 指示で区切らなければなりません。

iSeries サーバーからのファイルのコピー

iSeries サーバーからファイルをコピーするには、以下のように /COPY ステートメントを入力してください。

- 1 スペースだけを続けた /COPY または /INCLUDE
- 1 スペースだけを続けた *REMOTE
- コピー (マージ) されるメンバーの位置。形式: libraryname/filename,membername
  - メンバー名を指定しなければなりません。
  - ファイル名を指定しない場合のデフォルトは QRPGLESRC です。
  - ファイル名とメンバー名はコンマで区切ります。コンマを入れる必要があります。
  - ライブラリーを指定しないと、ライブラリー・リストでファイルが検索されます。メンバーが見つかるか検索が完了するまで、ライブラリー・リストにある指定したソース・ファイルがすべて検索されます。
  - ライブラリーを指定したら、ファイル名も指定しなければなりません。
- 少なくとも 1 つのスペースとコメント (オプション)。

OS/400 ファイルをコピーするための /COPY ステートメントの例を以下に示します。

- ソース・ファイル QRPGLESRC のメンバー MBR1 をコピーするには、次のステートメントを入力します。ファイル QRPGLESRC の検索には現行のライブラリー・リストが使用されることに注意してください。

  ```
  C/COPY *REMOTE MBR1
  ```

- ソース・ファイル SRCFIL のメンバー MBR1 をコピーするには、次のステートメントを入力します。ファイル SRCFIL の検索には現行のライブラリー・リストが使用されることに注意してください。

  ```
  I/COPY *REMOTE SRCFIL,MBR1
  ```

- ライブラリー SRCLIB のソース・ファイル SRCFIL のメンバー MBR1 をコピーするには、次のステートメントを入力します。

  ```
  O/COPY *REMOTE SRCLIB/SRCFIL,MBR1
  ```

- ライブラリー ″srclib″ のファイル ″srcfil″ のメンバー ″mbr1″ をコピーするには、次のステートメントを入力します。

  ```
  O/COPY *REMOTE "srclib"/"srcfil","mbr1"
  ```

ワークステーションからのファイルのコピー

ローカル・ワークステーションからファイルをコピーするには、次のように /COPY ステートメントを入力してください。

- 1 スペースだけを続けた /COPY または /INCLUDE
- コピーされるメンバーの位置。形式: Drive:\pathname\member.CPY

  ドライブおよびパスは任意指定です。
- 少なくとも 1 つのスペースとコメント (オプション)。

以下に、ローカル・ファイルをコピーするための /COPY ステートメントの例を示します。

```
O/COPY D:\PROJECT1\INCLUDES\TOOLS1.CPY
```

ネストされた /COPY または /INCLUDE

/COPY および /INCLUDE 指示は、ネストすることができます。 /COPY または /INCLUDE メンバーには 1 つ以上の /COPY または /INCLUDE 指示 (これらの指示にさらに /COPY または /INCLUDE 指示が含まれる場合があります。以下同様) を入れることができます。ネストできる最大深さは、COPYNEST制御仕様キーワードを使用して設定することができます。デフォルトの最大深さは 32 です。

ネストされた /COPY または /INCLUDE ファイルに相互に無限が含まれていないことを確認する必要があります。ソース行が複数回使用されることのないように /COPY または /INCLUDE ファイルの始めに条件付きコンパイル指示を使用してください。

条件付きコンパイル指示

条件付きコンパイル指示ステートメントによって、ソース・コードのセクションをコンパイルに条件付きで組み込んだり除外することができます。

- 条件指示 /DEFINE および /UNDEFINE の定義を使用して、現在定義されている条件のリストに条件名を追加したり、リストから条件名を除去することができます。
- 条件式 DEFINED(condition-name) および NOT DEFINED(condition-name) は、条件のテスト /IFグループ内で使用します。
- 条件のテスト指示 /IF、/ELSEIF、/ELSE、および /ENDIF は、コンパイラーがどのソース行を読み取るかを制御します。
- /EOF 指示は、コンパイラーに現行ソース・メンバーの残りのソース行を無視するように指示します。

条件の定義

条件名は、条件の定義指示 /DEFINE および /UNDEFINE を使用して、現在定義されている条件のリストに追加したりリストから除去することができます。

/DEFINE (7 ～ 13 桁目)

/DEFINE コンパイラー指示は、条件付きコンパイルの条件を定義します。条件名域の項目はフリー・フォームです (左寄せする必要はありません)。 /DEFINE には次の項目が使用されます。

桁	項目
7 ～ 13	/DEFINE
14	ブランク
15 ～ 80	条件名
81 ～ 100	コメント

/DEFINE 指示は、現在定義されている条件のリストに条件名を追加します。以後の /IF DEFINED(condition-name) は真になります。以後の /IF NOT DEFINED(condition-name) は偽になります。

/UNDEFINE (7 ～ 15 桁目)

/UNDEFINE 指示は、条件がすでに定義されていないことを示すために使用します。条件名域の項目はフリー・フォームです (左寄せする必要はありません)。

桁	項目
7 ～ 15	/UNDEFINE
16	ブランク
17 ～ 80	条件名
81 ～ 100	コメント

/UNDEFINE 指示は、現在定義されている条件のリストから条件名を除去します。以後の /IF DEFINED(condition-name) は偽になります。以後の /IF NOT DEFINED(condition-name) は真になります。

注: DEFINE パラメーターで指定された条件は、/IF および /ELSEIF 指示を処理するときに定義されると見なされます。これらの条件は /UNDEFINE 指示を使用して除去することができます。

事前定義条件

幾つかの条件は、RPG コンパイラーが定義します。こうした条件は、/DEFINE または /UNDEFINE では使用できません。/IF と /ELSEIF の場合にしか使用できません。

コンパイラー・ターゲットに関連した条件

COMPILE_WINDOWS
この条件は、プログラムが Windows ネイティブ・プログラムを作成するためにコンパイル中である場合に定義されます。(EXE または DLL オブジェクト。)

COMPILE_JAVA
この条件は、プログラムが Java で実行するためにコンパイル中である場合に定義されます。

条件式

条件式は次の形式の 1 つをとります。
- DEFINED(condition-name)
- NOT DEFINED(condition-name)

条件式はフリー・フォームですが、次の行に続けることはできません。

条件のテスト

条件は /IF グループを使用してテストされ、これは /IF 指示の後にゼロまたは 1 つ以上の /ELSEIF 指示を続け、さらに任意指定で /ELSE 指示とその後に続く /ENDIF 指示で構成されます。

/IF グループの指示間では、コンパイル時データ以外のソース行が有効です。これにはネストされた /IFグループが含まれます。

注: /IF グループのネスト・レベルには実際上の制限はありません。

/IF 条件式 (7 ～ 9 桁目)

/IF コンパイラー指示は、条件付きコンパイルの条件式のテストに使用します。/IF には次の項目が使用されます。

桁	項目
7 ～ 9	/IF
10	ブランク
11 ～ 80	条件式
81 ～ 100	コメント

条件式が真の場合は、/IF 指示の後のソース行が選択されて、コンパイラーによって読み取られます。そうでない場合は、同じ /IF グループの次の /ELSEIF、/ELSE、または /ENDIF までの行は除外されます。

/ELSEIF 条件式 (7 ～ 13 桁目)

/ELSEIF コンパイラー指示は、/IF または /ELSEIF グループ内の条件式のテストに使用します。 /ELSEIFには次の項目が使用されます。

桁	項目
7 ～ 13	/ELSEIF
14	ブランク
15 ～ 80	条件式
81 ～ 100	コメント

前の /IF または /ELSEIF が満たされないでその条件式が真の場合には、/ELSEIF 指示の後のソース行が選択されて読み取られます。そうでない場合には、同じ /IF グループの次の /ELSEIF、/ELSE または /ENDIF が見つかるまでの各行は除外されます。

/ELSE (7-11 桁目)

/ELSE コンパイラー指示は、失敗した /IF または /ELSEIF テストの後で読み取るソース行を無条件に選択するために使用します。/ELSE には次の項目が使用されます。

桁	項目
7 ～ 11	/ELSE
12 ～ 80	ブランク

81 ～ 100	コメント

前の /IF または /ELSEIF が満たされない場合には、次の /ENDIF までのソース行が選択されます。

前の /IF または /ELSEIF が満たされた場合には、次の /ENDIF までのソース行が除外されます。

/ENDIF (7-12 桁目)

/ENDIF コンパイラー指示は、一番新しい /IF、/ELSEIF、または /ELSE グループを終了するために使用します。/ENDIF には次の項目が使用されます。

桁	項目
7 ～ 12	/ENDIF
13 ～ 80	ブランク
81 ～ 100	コメント

/ENDIF 指示の後で、対応する /IF 指示が行を選択した場合には、各行が無条件に選択されます。そうでない場合には、/IF グループ全体が選択されないので、引き続き各行は選択されません。

条件のテスト規則

- /ELSEIF および /ELSE は /IF グループの外側では無効です。
- /IF グループには最大 1 つの /ELSE 指示を含めることができます。/ELSEIF 指示を /ELSE 指示に続けることはできません。
- /ENDIF は /IF、/ELSEIF、または /ELSE グループの外側では無効です。
- すべての /IF は以後の /ENDIF と対応していなければなりません。
- いずれか 1 つの /IF グループに関連づけられたすべての指示は同じソース・ファイルに入っていなければなりません。/IF を 1 つのファイルに入れて、対応する /ENDIF を別のファイルに入れることは、2 番目のファイルがネストされた /COPY であっても有効ではありません。しかし、完全な /IF グループをネストされた /COPY に入れることはできます。

/EOF 指示

/EOF 指示は、コンパイラーに現行のソース・メンバーの残りのソース行を無視するように指示します。

/EOF (7 ～ 10 桁目)

/EOF コンパイラー指示は、コンパイラーに現行のソース・ファイルのファイルの終わりに達したと見なすように指示するために使用します。 /EOF には次の項目が使用されます。

桁	項目
7 ～ 10	/EOF
11 ～ 80	ブランク
81 ～ 100	コメント

/EOF は、現行のソース・メンバーの読み取り中にアクティブになったアクティブ /IF グループを終了します。/EOF が /COPY ファイルの中にある場合には、その /COPY 指示が読み取られたときにアクティブであった条件は依然としてアクティブのままとなります。

注: 除外された行がリストに印刷される場合には、/EOF の後でソース行が引き続き読み取られてリストされますが、各行の内容はコンパイラーによって完全に無視されます。/EOF の後では、診断メッセージは出されません。

/EOF 指示を使用すると、/COPY メンバー全体を一回だけ使用するときにはコンパイル時のパフォーマンスは高くなりますが、複数回コピーされる場合があります。(これは、除外された行が印刷される場合には該当しません。)

/EJECT (7 ～ 12 桁目)

コンパイラー指示 /EJECT は、コンパイラー・リストで新しいページを開始するために使用します。

注: /EJECT はコンパイラー・リストには印刷されず、新しいページと置き換えられます。コンパイラー・リストがすでに新しいページの先頭にある場合には、新しいページはコンパイラー・リストには印刷されません。

新しいページを指定するには、次のように /EJECT ステートメントを入力してください。

桁	項目
7 ～ 12	/EJECT
13 ～ 49	ブランク
50 ～ 100	コメント

/SPACE (7 ～ 12 桁目)

コンパイラー指示 /SPACE は、コンパイラー・リストのソース・セクション内の行送りを制御するために使用します。

注: /SPACE はコンパイラー・リストには印刷されず、指定した行送りと置き換えられます。/SPACE によって行なわれる行送りは、仕様タイプ間でスキップされる 2 行に加えて行なわれます。

見出し情報を指定するには、次のように /SPACE ステートメントを入力してください。

桁	項目
7 ～ 12	/SPACE
13	ブランク
14 ～ 16	送る行数を定義する 1-112 の正整数値。112 より大きい値を指定すると、112 が /SPACE 値として使用されます。数が現行ページに残されている行数より大きい場合には、以後の仕様は次のページの先頭から開始されます。

17 ～ 49	ブランク
50 ～ 100	コメント

/TITLE (7 ～ 12 桁目)

コンパイラー指示 /TITLE は、ヘッダー情報 (セキュリティーの分類やタイトルなど) を指定するために使用します。このタイトル情報は、コンパイラー・リストの各ページの先頭に表示されます。

1 つのプログラムに 2 つ以上の /TITLE ステートメントを入れることができます。それぞれの /TITLE ステートメントは、別の /TITLE ステートメントが見つかるまで、コンパイラー・リストの見出しを提供します。コンパイラー・リストの最初のページに情報を印刷するためには、/TITLE ステートメントは最初に見つかる指定でなければなりません。/TITLE ステートメントによって指定された情報は、コンパイラーの見出し情報に加えて印刷されます。

注: /TITLE はコンパイラー・リストには印刷されず、見出し情報と置き換えられます。/TITLE ステートメントは、タイトルが印刷される前に次のページにスキップさせます。

見出し情報を指定するには、次のように /TITLE ステートメントを入力してください。

桁	項目
7 ～ 12	/TITLE
13	ブランク
14 ～ 100	タイトル情報

第 3 章 標識

標識は 1 バイトの文字フィールドで、'1' (オン) または '0' (オフ) のいずれかが入ります。標識は、一般に操作の結果を指示するか、あるいは操作の処理を条件づけるために使用されます。

標識は仕様の項目によって定義されます。標識が定義される仕様の位置によって、その標識の使用方法が決定されます。次に、定義された標識を使用して演算および出力命令に条件づけることができます。

標識形式は、標識の変数を定義する定義仕様で指定することができます。標識形式で文字データを定義する方法については、115 ページの『文字データ・タイプ』 および 264 ページの『40 桁目 (内部データ・タイプ)』 を参照してください。

ほとんどの標識の状態は演算命令によって変更することができます。すべての標識は、SETON 命令コードでオンに設定し、SETOFF 命令コードでオフに設定することができます。

この項では、以下について説明します。
- VisualAge RPG 仕様で定義される標識 (レコード識別標識、フィールド標識、演算結果標識)
- 最終レコード標識 (LR)
- フィールドとレコードの関連標識の割り当て
- 演算の条件づけ
- 式での標識の使用
- 出力の条件づけ
- データとして参照される標識

仕様で定義される標識

次の標識は仕様で定義することができます。
- レコード識別標識 (入力仕様の 21 ～ 22 桁目)
- フィールド標識 (入力仕様の 69 ～ 74 桁目)
- 演算結果標識 (演算仕様の 71 ～ 76 桁目)
- *IN 配列、*IN(xx) 配列エレメント、または *INxx フィールド

次に、定義された標識を使用してプログラム内の命令に条件づけることができます。

レコード識別標識

レコード識別標識は、入力仕様の 21-22 桁目の項目で定義され、処理のために対応するレコード・タイプが選択されたときにオンに設定されます。次に、その標識を使用して、特定の演算および出力命令に条件づけすることができます。レコード識別標識は、特定の順序で割り当てる必要はありません。

レコード識別標識は 01-99 と LR です。

外部記述ファイルの場合には、レコード識別標識は任意選択です。指定する場合には、プログラム記述ファイルの場合と同じ規則に従います。

処理のためにレコード・タイプを選択すると、対応するレコード識別標識がオンに設定されます。他のレコード識別標識は、ファイルからのレコードのリトリーブにファイル命令コードが使用される場合を除き、すべてオフになります。レコード識別標識は、レコードが選択された後で入力フィールドが入力域に移動される前にオンに設定されます。標識はいつでもオンに設定することができます。

レコードをリトリーブするためにファイル命令コードが使用されると、レコードがファイルからリトリーブされるとすぐにレコード識別標識がオンに設定されます。同じファイルに複数の命令が出された場合には、同じファイルに対して複数のレコード識別標識とレコードが見つからない標識を同時にオンに設定することが可能です。

レコード識別標識の割り当て規則

プログラム記述ファイルのレコードにレコード識別標識を割り当てるときには、次の規則が適用されます。

- すべてのレコード・タイプで同じ命令が処理される場合には、2 つまたはそれ以上の別のレコード・タイプに同じ標識を割り当てることができます。その場合には、21-22 桁目にレコード識別標識を指定し、OR関係の各レコード・タイプにレコード ID を指定してください。
- レコード識別標識は AND 関係で関連づけることができますが、グループの最初の行に表示されなければなりません。レコード識別標識を AND 行に指定することはできません。
- 未定義のレコード (プログラム記述ファイルに入っている 23 ～ 46 桁目でレコード ID によって記述されていないレコード) はプログラムを停止させる原因になります。
- レコード識別標識は、別のレコード・タイプのレコード識別標識、フィールド標識、または演算結果標識として指定することができます。診断メッセージは出されませんが、この標識の使用によって誤った結果になることがあります。

外部記述ファイルのレコードにレコード識別標識を割り当てるときには、次の規則が適用されます。

- AND/OR 関係をレコード様式名と一緒に使用することはできませんが、複数のレコードに同じレコード識別標識を割り当てることはできます。
- 7 ～ 16 桁目には、ファイル名でなくレコード様式名を指定しなければなりません。

フィールド標識

フィールド標識は、入力仕様の 69 ～ 70 桁目、71 ～ 72 桁目、および 73 ～ 74 桁目の項目によって定義されます。フィールド標識は一般標識 01 ～ 99 です。

フィールド標識は、指定されたフィールドまたは配列エレメントがゼロより大きいか、ゼロより小さいか、ゼロか、あるいはブランクかを判別するために使用することができます。

- 69 ～ 72 桁目は数値フィールドの場合に有効です。
- 73 ～ 74 桁目は数字または文字フィールドの場合に有効です。

- 69 ～ 70 桁目に指定した標識は、数値の入力フィールドがゼロより大きい場合にオンに設定されます。
- 71 ～ 72 桁目に指定した標識は、数値の入力フィールドがゼロより小さい場合にオンに設定されます。
- 73 ～ 74 桁目に指定した標識は、数字の入力フィールドがゼロであるか、または文字の入力フィールドがブランクの場合にオンに設定されます。

次に、フィールド標識を使用して、演算または出力命令に条件づけすることができます。

フィールド標識は、フィールドまたは配列エレメントのデータがレコードから抜き出されて、それが表している条件が入力レコードに存在する場合にオンに設定されます。このフィールド標識は、同じタイプの別のレコードが読み取られて、それが表している条件が入力レコードに存在しないか、あるいは演算の結果としてその標識がオフに設定されるまで、オンのままとなります。

フィールド標識の割り当て規則

フィールド標識を割り当てるときには、次の規則が適用されます。

- プログラムの始めには、プラス、マイナス、ゼロ、またはブランクの標識はオフに設定されています。これらは、読み取ったレコードに対してテストされるフィールドで条件 (プラス、マイナス、ゼロ、またはブランク) が満たされなければオンには設定されません。
- フィールド標識は、配列全体で使用することはできません。しかし、配列エレメントの場合には入力を行なうことができます。フィールド標識を NULL 値可能フィールドに使用できるのは、**ユーザー制御**または **ALWNULL(*USRCTL)** オプションが使用されている場合だけです。NULL 値のサポートについては、145 ページの『データベースのヌル値サポート』 を参照してください。
- 数値の入力フィールドには 2-3 のフィールド標識を割り当てることができます。しかし、そのフィールドのテストの結果を知らせる標識がオンに設定されるだけで、他はオフに設定されます。
- 別のレコード・タイプのフィールドに同じフィールド標識が割り当てられると、その状態 (オンまたはオフ) は常に最後に選択したレコード・タイプに基づいたものとなります。
- 別のレコード・タイプのフィールドに別のフィールド標識を割り当てると、そのタイプの別のレコードが読み取られるまでそのフィールド標識はオンのままとなります。同様に、単一のレコード・タイプ内の 2つ以上のフィールドに割り当てられたフィールド標識は、常に最後に定義されたフィールドの状態を反映しています。
- 同じフィールド標識を、別の入力仕様のフィールド標識、演算結果標識、レコード識別標識、またはフィールドとレコードの関連標識として指定することができます。診断メッセージは出されませんが、この標識の使用によって誤った結果になることがあります。
- 3 つの位置全部に同じ標識を指定すると、そのフィールドが入っているレコードを選択したときに常にその標識がオンに設定されます。

演算結果標識

結果の標識は、従来の形式 (C 仕様) で演算仕様によって使用されます。フリー・フォーム演算仕様によっては使用されません。ほとんどの命令コードの場合、従来

の形式またはフリー・フォームのいずれでも、結果の標識の代わりに組み込み関数を使用できます。詳細については、 399 ページの『第 25 章 組み込み関数』 を参照してください。

演算結果標識は、演算仕様の 71-76 桁目の項目によって定義されます。演算結果標識の目的は、26-35桁目に指定した命令コードによって異なります。結果の標識の目的については、『第 26 章 命令コードの詳細』の個々の命令コードを参照してください。たとえば、演算結果標識は、算術演算の後で結果フィールドをテストし、レコードが見つからない条件を識別し、ファイル命令の例外 / エラー条件を指示し、あるいはファイルの終わり条件を指示するために使用することができます。

演算結果標識は 01-99 と LR です。

演算結果標識は、演算仕様の 3 つの位置 (71-72 桁目、73-74 桁目、および 75-76 桁目) に指定することができます。演算結果標識が定義された位置によって、テストする条件が決定されます。

ほとんどの場合には、演算が処理されると、演算結果標識がオフに設定され、演算結果標識によって指定された条件が満たされると、その標識がオンに設定されます。しかし、この規則には、 582 ページの『LOOKUP (テーブルまたは配列エレメントの検索)』、 674 ページの『SETOFF (標識をオフにセット)』、および 674 ページの『SETON (標識をオンにセット)』 などのいくつかの例外があります。演算結果標識は、同じ演算行あるいは他の演算または出力命令で条件標識として使用することができます。同じ行で使用した場合には、この標識の前の設定によってその演算を処理するかどうかが決定されます。処理されると、結果フィールドがテストされて、この標識の現在の設定が決定されます (23 ページの図 3 を参照) 。

演算結果標識の割り当て規則

演算結果標識を割り当てるときには、次の規則が適用されます。

- 結果フィールドが配列全体を参照しているときには、演算結果標識を使用することはできません。
- 2 つ以上の演算結果をテストするために同じ標識を使用すると、最後に処理された演算によってこの標識の設定が決定されます。
- 指定された命令によって、2 つ以上の条件のテストに同じ標識を使用することができます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C*  減算の演算命令で異なる条件をテストするために 2 つの演算結果標識
C*  が使用されています。これらの標識は、給与計算ジョブで処理されなけ
C*  ればならない演算を条件づけるために使用されます。勤務時間 (HRSWKD)
C*  が 40 を超える場合には、標識 10 がオンに設定され、残業手当を検索
C*  するために必要なすべての演算を条件づけるために使用されます。
C*  標識 20 がオンでない (従業員の作業時間が 40 時間以上である)
C*  場合には、週 40 時間に基づく基本給が計算されます。
C*
C     HRSWKD        SUB       40            OVERTM          3 01020
C*
C  N20PAYRAT        MULT (H)  40            PAY             6 2
C   10OVERTM        MULT (H)  OVRRAT        OVRPAY          6 2
C   10OVRPAY        ADD       PAY           PAY
C*
C*  標識 20 がオンの (社員が 40 時間未満働いた) 場合
C*  には、週 40 時間未満に基づいて計算されます。
C*
C   20PAYRAT        MULT (H)  HRSWKD        PAY
C*
```

図 3. 条件命令に使用した演算結果標識

最終レコード標識 (LR)

LR 標識はプログラムを終了するために使用することができます。この標識は、プログラムを終了するかどうかを決定するために、各アクション・サブルーチンの終わりにテストされます。詳細については、 550 ページの『ENDACT (アクション・サブルーチンの終了)』 を参照してください。

標識の使用

レコード識別標識、フィールド標識、演算結果標識、*IN、*IN(xx)、または *INxx として定義された標識は、ファイル、演算命令、または出力命令の条件づけに使用することができます。標識は、条件標識として使用する前に定義しなければなりません。標識の状態 (オンまたはオフ) は、条件標識として使用する場合には影響を受けません。状態を変更できるのは、特定の条件を表すために標識を定義した場合だけです。

フィールドとレコードの関連標識

フィールドとレコードの関連標識は入力仕様の 67-68 桁目に指定します。有効なフィールドとレコードの関連標識は 01-99 です。

注: フィールドとレコードの関連標識を外部記述ファイルに指定することはできません。

フィールドとレコードの関連は、レコード・タイプが OR 関係のいくつかのうちの 1 つである場合に、フィールドを特定のレコード・タイプと関連づけます。仕様行で記述されたフィールドを入力に使用できるのは、フィールドとレコードの関連項

目で指定された標識がオンか、またはこの項目がブランクの場合だけです。この項目がブランクの場合には、そのフィールドは OR 関係で定義されたすべてのレコード・タイプに共通です。

フィールドとレコードの関連標識の割り当て

フィールドを特定のレコード・タイプと関連づけるには、67-68 桁目にレコード識別標識を指定します。 OR 関係で複数のレコード・タイプが指定されている場合には、67-68 桁目にフィールドとレコードの関連標識がないフィールドはすべて OR 関係のすべてのレコード・タイプと関連づけられます。フィールドを 1 つのレコード・タイプと関連づけるには、67-68 桁目にそのレコード・タイプに割り当てられたレコード識別標識を入力します (図 4 を参照) 。

レコード識別標識以外の標識 (01-99) も、入力域から入力フィールドへのフィールドの移動を条件づけるために 67-68 桁目に使用することができます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++Sq..RiPos1+NCCPos2+NCCPos3+NCC..............................
I.......................Fmt+SPFrom+To+++DcField+++++++++....FrPlMnZr....
IREPORT    AA  14    1 C5
I        OR    16    1 C6
I                                  20   30  FLDB
I                                   2   10  FLDA              07
I*
I*  プログラム内のどこかに標識 07 が指定されています。
I*
I                                  40   50  FLDC              14
I                                  60   70  FLDD              16
```

図 4. フィールドとレコードの関連

図 4 のファイルには、1 つは 1 桁目の 5 ともう 1 つは 1 桁目の 6 で識別される 2 種類のレコードが入っています。FLDC フィールドは、レコード識別標識 14 によって 1 桁目の 5 で識別されたレコード・タイプに関連づけられています。FLDD フィールドは、レコード識別標識 16 によって 1桁目に 6 があるレコード・タイプに関連づけられています。このことは、FLDC が 1 つのレコード・タイプ (1 桁目の 5 で識別される) だけで見つけられ、FLDD がもう 1 つのタイプだけで見つけられることを意味します。FLDA は、前にプログラム内のどこかで定義されている標識 07 によって条件づけされます。FLDBは、レコード識別標識によってどちらのタイプにも関連づけされていないので、両方のレコード・タイプで見つけられます。

演算の条件づけ標識

従来の形式 (C 仕様) での演算仕様には、9 ～ 11 桁目に条件付け標識を組み込むことができます。条件付け標識は、フリー・フォーム演算仕様では使用されません。

演算が行なわれる条件を指定する標識は、プログラム内のどこかで定義されます。

7-8 桁目

演算仕様の 7-8 桁目にブランク、SR、AN、または OR を指定します。 7-8 桁目がブランクの場合には、その演算はプログラム・ロジック、サブルーチン内のステートメント、または宣言命令によって指定された場合に処理されます。

9-11 桁目

演算が処理される条件を制御する標識を指定するには、演算仕様の 9-11 桁目に指定します。 9 桁目に N が指定されている場合には、その標識はオフ ('0') の値についてテストされなければなりません。10-11 桁目には、01-99 または LR を指定することができます。

9-11 桁目に使用する標識は、前に次のタイプの標識の 1 つとして定義されていなければなりません。

- レコード識別標識 (入力仕様、21-22 桁目)
- フィールド標識 (入力仕様、69-74 桁目)
- 演算結果標識 (演算仕様、71-76 桁目)
- *IN 配列、*IN(xx) 配列エレメント、または *INxx フィールド。これらの予約語の 1 つを使用する場合の標識の定義方法については、27 ページの『データとして参照される標識』 を参照してください。

演算の条件づけで標識がオフでなければならない場合には、9 桁目に N を入れます。グループ化された AND/OR 行の標識は、その演算を行なう前にすべて指定されたとおりでなければなりません。

図 5 および 図 6 に、条件標識の例を示します。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++Sq..RiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1FrPlMnZr...*
I.....................Fmt+SPFrom+To+++DcField+++++++++....FrPlMnZr....
I*
I*  フィールド標識は、演算の条件づけに使用することができます。プログラム
I*  が時間外勤務を含む週給を見つけるためのものであると仮定してください。
I*  時間外勤務フィールドを検査して時間外勤務の入力の有無を調べます。
I*  社員が時間外勤務した場合には、このフィールドが正になって
I*  標識 10 がオンに設定されます。すべての場合に通常の週給が計算され
I*  ます。しかし、時間外手当が加算されるのは、標識 10 がオンの場合
I*  だけです。
I*
ITIME      AB  01
I                                  1    7  EMPLNO
I                                  8   10 0OVERTM              10
I                                 15   20 2RATE
I                                 21   25 2RATEOT
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C*
C*  入力仕様にフィールド標識が割り当てられています。
C*  これは、ここでは演算命令の条件づけに使用されています。
C*
C                   EVAL (H)  PAY = RATE * 40
C   10              EVAL (H)  PAY = PAY + (OVERTM * RATEOT)
```

図 5. 演算の条件づけ (フィールド標識)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++Sq..RiPos1+NCCPos2+NCCPos3+NCC.............................
I.....................Fmt+SPFrom+To+++DcField+++++++++....FrPlMnZr....
I*
I*  レコード識別標識は、演算に条件を付けるために使用されます。
I*  レコードが 1 桁目に T で読み取られると、01 標識がオンに設定
I*  されます。この標識がオンの場合には、SAVE という名前のフィールド
I*  が SUM に加算されます。1 桁目に T のないレコードが読み取られると
I*  02 標識がオンに設定されます。次に、加算命令の代わりに、02 で条件
I*  づけられた減算命令が実行されます。
I*
IFILE      AA  01    1 CT
I         OR   02    1NCT
I                                10   15 2SAVE
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C*  入力仕様でレコード識別標識 01 と 02 が割り当てられています。
C*  これらは、ここでは演算命令に条件を付けるために使用されて
C*  います。
C*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C   01              ADD       SAVE          SUM               8 2
C   02              SUB       SAVE          SUM               8 2
```

図 6. 演算の条件づけ (レコード識別標識)

式で使用する標識

標識は、演算仕様の拡張演算項目 2 フィールドの式でブールとして使用することができます。これらは、データとして (すなわち、*IN または *INxx を使用して) 参照されなければなりません。図 7 はこれを示しています。

```
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C* これらの例では、IF 構造は 01 がオンの場合に実行されるだけです。
C* *IN01 は、値がオンまたはオフのブールとして処理されます。
C* 最初の例では、標識の値 ('0' または '1') が検査されます。
C                   IF        *IN01
C* 2 番目の例では、論理式 B < A が評価されます。
C* 真なら 01 がオンに設定され、偽なら 01 がオフに設定されます。
C* これは、A と B で COMP を使用して、適切な演算結果標識位置に
C* 01 を入れたのと同じです。
C                   EVAL      *IN01 = B < A
```

図 7. 式で使用する標識

詳細については、 383 ページの『第 24 章 式』 および 552 ページの『EVAL (式の評価)』 を参照してください。

出力の条件づけ標識

出力レコードまたは出力フィールドが書き出される条件を指定するために使用される標識は、プログラム内で前に定義されていなければなりません。出力に条件を付ける標識は 21-29 桁目に指定します。すべての標識が出力の条件づけに有効です。

出力の条件づけに使用する標識は、次のタイプの標識の 1 つとして前に定義されていなければなりません。

- レコード識別標識 (入力仕様、21-22 桁目)

- 01-99 のように VisualAge RPG プログラムによって設定される標識
- *IN 配列、*IN(xx) 配列エレメント、または *INxx フィールド。

標識がレコード全体に条件を付ける場合には、そのレコード・タイプを指定する行に標識を入力します。標識がフィールドの書き出し時点を条件づけする場合には、フィールド名と同じ行に標識を入力します。

条件づけ標識は出力行には必要ありません。条件標識が指定されていないと、その行は、出力に対してそのレコード・タイプが検査されるたびに出力されます。条件標識を指定する場合には、3 つの別個の出力標識フィールド (22-23 桁目、25-26 桁目、および 28-29 桁目) のそれぞれに 1 つの標識を入力することができます。これらの標識がオンの場合には、出力命令が実行されます。それぞれの標識の前の桁 (21 桁目、24 桁目、または 27 桁目) に N があることは、その標識がオンでない場合にだけ出力命令が実行されることを意味します (否定標識) 。出力行をすべて否定標識で条件づけることはできません。少なくとも 1 つの標識は正でなければなりません。

出力標識は、16-18 桁目に AND/OR を指定して、AND/OR 関係で指定することができます。AND/OR 行の数は制限なしに使用することができます。 AND/OR 行は出力レコードの条件づけに使用することができますが、フィールドの条件づけには使用できません。しかし、1 つのフィールドは、演算で EVAL 命令を使用して、4 つ以上の標識で条件づけることができます。図 8 に、これを示します。

```
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C* 標識 20 は、標識 10、12、14、16、および 18 がオンに設定された
C* 場合にだけオンに設定されます。
C                   EVAL      *IN20 = *IN10 AND *IN12 AND *IN14
C                              AND *IN16 AND *IN18
C                   EXCPT
OFilename++EAddN01N02N03Excnam++++.....................................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O* OUTFIELD は標識 20 で条件づけされ、これは結果的に、EVAL 命令の
O* すべての標識によって条件づけされることを意味します。
OPRINTER   E
O               20      OUTFIELD
```

図 8. 標識での EVAL の使用

データとして参照される標識

標識を参照し操作するもう 1 つの方法は、*IN および *INxx 予約語を使用するものです。

*IN

配列 *IN は、標識 01-99 を表す 99 の 1 桁の文字エレメントの定義済み配列です。配列のエレメントには、文字値 '0' (ゼロ) または '1' (1) しか入れることができません。

入力レコードのフィールド、結果フィールド、または PARM 命令の演算項目 1 としての *IN 配列または *IN(xx) 可変索引配列エレメントの指定は、プログラム内で使用する標識 01-99 を定義します。

単一文字エレメントの配列に有効な演算または参照は、配列 *IN で有効です。ただし、配列 *IN をデータ構造のサブフィールドまたは PARM 命令の結果フィールドとして指定することはできません。

*INxx

フィールド *INxx は定義済みの 1 桁の文字フィールドで、この場合に xx は標識のいずれか 1 つを表します。

入力レコードのフィールド、結果フィールド、または PARM 命令の演算項目 1 としての *INxx フィールドまたは *IN(n) 固定索引配列エレメント (ここで、n = 1 - 99) の指定は、プログラム内で使用する対応する標識を定義します。

*INxx は、1 桁の文字フィールドが有効な場合に指定します。*INxx を、データ構造のサブフィールド、PARM 命令の結果フィールド、または SORTA 命令で指定することはできません。

データとして参照される標識を指定するための規則

*IN、配列エレメント *IN(xx)、またはフィールド *INxx には、次の規則が適用されます。

- 文字 '0' (ゼロ) または *OFF をこれらのフィールドのいずれかに移動すると、対応する標識がオフに設定されます。
- 文字 '1' (1) または *ON をこれらのフィールドのいずれかに移動すると、対応する標識がオンに設定されます。
- '0' (ゼロ) または '1' (1) 以外の値を *INxx に移動してはいけません。
- *IN、*IN01 - *IN99、または *IN(index) のアドレスを取ると、標識 *IN01 ～ *IN99 が定義されます。*INLR などの他の標識のアドレスを取ると、その標識だけが定義されます。

データとして参照される標識の例については、図 9 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C*  このプログラムが呼び出されると、プログラム内のロジックを制御
C*  するために単一のパラメーターが渡されます。このパラメーターは
C*  標識 50 の値を設定します。このパラメーターは文字値の 1 か 0
C*  で渡されなければなりません。
C*
C     *ENTRY        PLIST
C     *IN50         PARM                    SWITCH            1
C*
C*
C*  サブルーチン SUB1 は標識 61-68 を使用します。このサブルーチンが
C*  処理される前に、メインライン・プログラムで使用されるこれらの
C*  標識の状況が保管されます。(このサブルーチンの始めには、標識は
C*  オフに設定されているものとします。) サブルーチンが処理された
C*  後では、標識はそれらの元の状態に戻されます。
C*
C*
C                   MOVEA     *IN(61)        SAV8              8
C                   EXSR      SUB1
C                   MOVEA     SAV8          *IN(61)
C*
C*  コード・フィールド (CODE) には 1-5 の数値が入っていて、標識 71-75
C*  を設定するために使用されます。この 5 つの標識はオフに設定されます。
C*  フィールド X は 70 + CODE フィールドとして計算されます。次に、
C*  フィールド X は 配列 *IN の索引として使用されます。その後で、
C*  標識 71-75 の状況に基づいて別のサブルーチンが使用されます。
C*
C                   MOVEA     '00000'       *IN(71)
C   70              ADD       CODE          X                 3 0
C                   MOVE      *ON           *IN(X)
C   71              EXSR      CODE1
C   72              EXSR      CODE2
C   73              EXSR      CODE3
C   74              EXSR      CODE4
C   75              EXSR      CODE5
```

図 9. データとして参照される標識の例

標識の要約

表 2. 標識がオンおよびオフに設定される場合

標識のタイプ	オンに設定	オフに設定
レコード識別	レコードが読み取られた直後。	プログラマーによって。
フィールド標識	指定されたフィールドのブランクまたはゼロによって、指定されたフィールドのプラスによって、あるいは指定されたフィールドのマイナスによって。	このフィールドの状況が次回テストされる前に。
演算結果	演算が処理されて、その標識が表す条件に適合した場合。	演算結果標識と同じ標識が指定されていて、指定された条件が満たされていない場合に、次回演算が処理されるとき。
LR	プログラマーによって。	プログラマーによって。

第 4 章 コンポーネントの処理

考えられる 3 つのターゲット・オブジェクトの 1 つが、結果としてコンパイルから生じます。結果は、使用される制御仕様キーワードによって異なります。

- NOMAIN および EXE キーワードがない時は、コンポーネントが作成されます。
- NOMAIN キーワードが指定されている時は、ユーティリティー、または NOMAIN、DLL が作成されます。この DLL には RPG サブプロシージャーしか入っていません。
- EXE キーワードが指定されている時は、RPG EXE が作成されます。このモジュールには、メイン・プロシージャーおよびサブプロシージャーが入っています。

このセクションでは、コンポーネントの開始および停止方法について説明するとともに、コンポーネントの初期化および終了方法についても説明します。NOMAIN DLL および EXE の作成および使用法の概要については、65 ページの『第 6 章 サブプロシージャーおよびプロトタイプ』を参照してください。

コンポーネントの開始および停止

START 命令コードおよび STOP 命令コードによって、1 つのアプリケーションで複数のコンポーネントを実行することができます。START 演算命令は、アプリケーション内で新規コンポーネントを開始します。 STOP 演算命令は、コンポーネントの実行を終了します。

これらの命令コードの詳細については、679 ページの『START (コンポーネント開始またはローカル・プログラム呼び出し)』 および 681 ページの『STOP (コンポーネントの停止)』 を参照してください。

コンポーネントの初期化

VisualAge RPG アプリケーションは、1 つまたは複数のコンポーネントから構成することができます。各コンポーネントは、個別に開始されます。アプリケーションが実行されると、最初の (1 次) コンポーネントが開始されます。後続の (2 次) コンポーネントはすべて、ユーザーまたはプログラムによって開始されますが、どちらによって開始されるかは、発生するイベントおよびそのイベントを処理するアクション・サブルーチンによって決まります。2 次コンポーネントは、どのような順序でも開始できます。

アプリケーションの EXE ファイルは、1 次コンポーネントを呼び出します。コマンド行からこのコンポーネントにパラメーターを渡すことができます。各パラメーターは、入力された文字ストリングから *ENTRY PLIST のパラメーターのターゲット・データ・タイプに変換されます。

2 次コンポーネントは、1 次コンポーネントまたは他の 2 次コンポーネントのいずれかから START 命令コードを使用して呼び出されます。PARM 命令コードおよび PLIST 命令コードを使用して、パラメーターを 2 次コンポーネントに渡すことができます。2 次コンポーネントの場合、パラメーターは変換されません。

コンポーネントがパラメーターを受け取ると、以下のことが起こります。

1. プログラム・フィールドが初期化されます。
2. ファイルがオープンされ、データ構造、実行時前配列、および実行時前テーブルがロードされます。
3. *ENTRY PLIST パラメーターの場合、結果フィールドが演算項目 1 に移動されます。
4. ユーザー初期化サブルーチン (*INZSR) が指定されていると、それが実行されます。コンポーネントのパーツおよびイベントを扱うほとんどの演算命令は、コンポーネントの実行時環境がまだ初期化されていないので、この時点では実行されません。
5. *ENTRY PLIST パラメーターの場合、演算項目 2 が結果フィールドにコピーされます。
6. RESET 演算命令によって使用されるデータ構造および変数が保管されます。
7. コンポーネントの実行時環境が初期化されます。
8. コンポーネントに対してアクション・サブルーチンが書かれている場合には、イベントの初期セットがウィンドウおよびそのパーツの初期セットとして処理されます。たとえば、「すぐにオープン」を指定する始動属性をもつすべてのウィンドウおよびそのパーツは、CREATE イベントを起こします。これらのアクション・サブルーチンの実行中に生成されたイベントは、この時点でそれらに対応して書かれたアクション・サブルーチンも呼び出します。

コンポーネントの初期化が完了すると、コンポーネントのパーツがアプリケーションに使用可能になります。エンド・ユーザーは、現在オープンされているどのようなコンポーネントであっても、その中にアクション・サブルーチンを呼び出すイベントを生成することができます。

アプリケーションの初期化中に一定の命令コード、属性、およびデフォルト例外ハンドラーが使用できないような場合があります。たとえば、パーツの作成前に、そのパーツの属性を入手することはできません。詳細については、35 ページの表 3、36 ページの表 4、37 ページの表 5、および 39 ページの表 6 を参照してください。

コンポーネントの終了

コンポーネントは、1 次コンポーネントを終了するか、1 つまたは複数のコンポーネントを開始したコンポーネントを終了すると、終了します。複数のコンポーネントが終了する時には、コンポーネントは階層の逆順で終了します。各コンポーネントには階層の逆順に呼び出される *TERMSR (正常終了) があります。各コンポーネントは、順にその終結処置と終了処理 (たとえばファイルのクローズ) が実行されます。

複数コンポーネント・アプリケーション中のあるコンポーネントが異常終了すると、そのコンポーネントだけが異常終了となります。同時に終了することになっているその他のコンポーネントは、正常に終了します。

正常終了

コンポーネントが正常終了となるのは、次のような状況です。

- ルート・アクション・サブルーチンで ENDACT に達した時に、LR がオンであった場合、あるいはルート・アクション・サブルーチンから RETURN が実行された場合。

ルート・アクション・サブルーチンは、アクション・サブルーチンのネストの最下部にある (最初の) サブルーチンです。アクション・サブルーチンのネストは、別のアクション・サブルーチンの実行中に、イベントで新規アクション・サブルーチンが呼び出される場合に起こります。

たとえばアクション・サブルーチン BUTTON+CLICK+WINDOW1 には SHOWWIN 'window2' 演算命令が含まれています。これにより、アクション・サブルーチン WINDOW2+CREATE+WINDOW2 を呼び出す CREATE イベントが生成されます。 CREATE イベントの処理中に別のイベントが起こると (たとえば 'WINDOWX' SETATR 1 'FOCUS')、アクション・サブルーチン WINDOW2+CREATE+WINDOW2 が中断され、アクション・サブルーチン WINDOWX+FOCUS+WINDOWX が呼び出されます。この呼び出しスタックには次のネストされたアクション・サブルーチンが組み込まれています。

```
1.  FIELD1+FOCUS+WINDOWX
2.  WINDOW2+CREATE+WINDOW2
3.  BUTTON+CLICK+WINDOW1    (ルート・アクション・サブルーチン)
```

以下が起こるまで、LR はチェックされません。

1. アクション・サブルーチン WINDOWX+FOCUS+WINDOWX が終了する
2. アクション・サブルーチン WINDOW2+CREATE+WINDOW2 が終了する
3. BUTTON+CLICK+WINDOW1 アクション・サブルーチンで ENDACT または RETURN 演算命令が実行される

- コンポーネントで STOP が実行された場合。詳細については、 681 ページの『STOP (コンポーネントの停止)』 を参照してください。
- *PSSR が実行され、以下のいずれかで終了した場合:
  - ENDSR '*DEFAULT' または同等のフィールド名。ルート・アクション・サブルーチンの終わりに LR 標識がオンになります。
  - ENDSR '*NODEFAULT' または同等のフィールド名。ルート・アクション・サブルーチンの終わりに LR 標識がオンになります。

  詳細については、 551 ページの『ENDSR (ユーザー・サブルーチンの終了)』 および 60 ページの『コンポーネント・エラー / 例外』 を参照してください。
- デフォルト例外ハンドラーがメッセージ情報ウィンドウを表示し、以下のいずれかが選択された場合:
  - 「デフォルト処理を実行する」。ルート・アクション・サブルーチンの終わりに LR 標識がオンになります。
  - 「デフォルト処理を実行しない」。ルート・アクション・サブルーチンの終わりに LR 標識がオンになります。

  詳細については、 60 ページの『コンポーネント・エラー / 例外』 を参照してください。

正常終了では以下が起こります。
- *TERMSR が存在すれば、それが実行されます
- ファイル、実行時前配列とテーブル、およびデータ域データ構造が書き込まれます
- すべてのファイルがクローズされます
- すべてのデータ域がアンロックされます

*TERMSR は、最終コードの実行が可能なユーザー作成サブルーチンです。
*TERMSR が呼び出されると、アクティブ状態のアクション・サブルーチンがなくなり、現行コンポーネントは終了中としてマークされます。これは、ほとんどのグラフィカル・ユーザー・インターフェース操作が使用できなくなることを意味します。 35 ページの表 3、 36 ページの表 4、 37 ページの表 5、および 39 ページの表 6 を参照してください。

コンポーネントの正常終了の状況値のリストについては、 60 ページの『コンポーネント状況コード』 を参照してください。

異常終了

以下のいずれかの状況が起こると、コンポーネントは異常終了します。
- *PSSR が実行され、以下のいずれかで終了した:
  - ENDSR '*ENDCOMP'、ENDSR '*CANCL'、または同等のフィールド名
  - ENDSR '*ENDAPPL' または同等のフィールド名
- デフォルト例外ハンドラーがメッセージ情報ウィンドウを表示し、以下のいずれかが選択されている:
  - コンポーネントの終了
  - アプリケーションの終了
- GUI での異常条件は実行時に起こります。

異常終了では以下のことが起こります。
- すべてのファイルがクローズされます
- すべてのデータ域がアンロックされます

注: 異常終了の場合、*TERMSR は呼び出されません。

初期化、終了、およびイベント処理の制約

アプリケーションの一部のステージで一定の命令コード、属性、およびデフォルト例外ハンドラーが使用できないような場合があります。以下の表では、初期化、終了、または通常イベント処理の時の各種の制約について説明します。

表 3. 初期化、終了、およびイベント処理の時の命令コードの制約

GUI 演算命令	初期化 (*INZSR)	終了 (*TERMSR)	イベント処理
CLSWIN	使用不能	使用不能	制約なし
DSPLY	制約なし	使用不能	表示される情報ウィンドウは、通知されたイベントを妨げます。CLSWIN または STOP 演算命令の実行後、DSPLY 演算命令が同じサブルーチンから、あるいはネストされたアクション・サブルーチンから実行された場合には (たとえば Close Window または Close Component イベントがまだ保留中である)、保留中のイベントは DSPLY 演算命令で受け取られますが、実行されません。

SHOWWIN	使用不能	使用不能	同じアクション・サブルーチンまたはネストされたアクション・サブルーチン内から同じ命令コードを複数回実行することはできません。たとえばアクション・サブルーチンに次が入っているとします。
			`SHOWWIN 'WIN1'`
			`CLSWIN 'WIN1'`
			`SHOWWIN 'WIN1'`
			この場合、2 番目の SHOWWIN は失敗します。

表 3. 初期化、終了、およびイベント処理の時の命令コードの制約 (続き)

GUI 演算命令	初期化 (*INZSR)	終了 (*TERMSR)	イベント処理

START	制約なし	制約なし	同じアクション・サブルーチンまたはネストされたアクション・サブルーチン内から同じ命令コードを複数回実行することはできません。たとえばアクション・サブルーチンに次が入っているとします。
			`START 'COMP2'`
			`STOP 'COMP2'`
			`STOP 'COMP2'`
			この場合、2 番目の STOP は失敗します。

STOP 'self'	使用不能	使用不能	コンポーネントは、ネストされたアクション・サブルーチンから終了することはできません
STOP 'other'	親コンポーネントは終了できません	親コンポーネントは終了できません	コンポーネントは、ネストされたアクション・サブルーチンから終了することはできません

表 4. 初期化、終了、およびイベント処理の時の属性の制約

属性	初期化 (*INZSR)	終了 (*TERMSR)	イベント処理
パーツ属性 (GETATR, SETATR, %GETATR, %SETATR)	使用不能	使用不能	制約なし
イベント属性 (%PART, ...)	使用不能	使用不能	制約なし
システム属性 (%DSPWIDTH, %DSPHEIGHT)	使用不能	使用不能	制約なし

表 5. 初期化、終了、およびイベント処理の時のデフォルト例外ハンドラーの制約

属性	初期化 (*INZSR)	終了 (*TERMSR)	イベント処理
メッセージ情報ウィンドウ、 デフォルト処理を実行	制約なし	コンポーネントは終了され、非同期情報ウィンドウが表示されます。	表示される情報ウィンドウは、通知されたイベントを妨げます。CLSWIN または STOP 演算命令の実行後、この演算命令が同じサブルーチンから、あるいはネストされたアクション・サブルーチンから実行された場合には (たとえば Close Window または Close Component イベントがまだ保留中である)、保留中のイベントはこの演算命令で受け取られますが、実行されません。
メッセージ情報ウィンドウ、 デフォルト処理を実行しない	制約なし	コンポーネントは終了され、非同期情報ウィンドウが表示されます。	表示される情報ウィンドウは、通知されたイベントを妨げます。CLSWIN または STOP 演算命令の実行後、この演算命令が同じサブルーチンから、あるいはネストされたアクション・サブルーチンから実行された場合には (たとえば Close Window または Close Component イベントがまだ保留中である)、保留中のイベントはこの演算命令で受け取られますが、実行されません。

表 5. 初期化、終了、およびイベント処理の時のデフォルト例外ハンドラーの制約 (続き)

属性	初期化 (*INZSR)	終了 (*TERMSR)	イベント処理
メッセージ情報ウィンドウ、 コンポーネントの終了	制約なし	コンポーネントは終了され、非同期情報ウィンドウが表示されます。	コンポーネントは、ネストされたアクション・サブルーチンから終了することはできません。表示される情報ウィンドウは、通知されたイベントを妨げます。CLSWIN または STOP 演算命令の実行後、この演算命令が同じサブルーチンから、あるいはネストされたアクション・サブルーチンから実行された場合には (たとえば Close Window または Close Component イベントがまだ保留中である)、保留中のイベントはこの演算命令で受け取られますが、実行されません。
メッセージ情報ウィンドウ、 アプリケーションの終了	制約なし	コンポーネントは終了され、非同期情報ウィンドウが表示されます。	表示される情報ウィンドウは、通知されたイベントを妨げます。CLSWIN または STOP 演算命令の実行後、この演算命令が同じサブルーチンから、あるいはネストされたアクション・サブルーチンから実行された場合には (たとえば Close Window または Close Component イベントがまだ保留中である)、保留中のイベントはこの演算命令で受け取られますが、実行されません。

表 6. 初期化、終了、およびイベント処理の時のコンポーネント終了の制約

コンポーネントの終了	初期化 (*INZSR)	終了 (*TERMSR)	イベント処理
*PSSR BEGSR.. ENDSR '*DEFAULT'	制約なし	制約なし	制約なし
*PSSR BEGSR.. ENDSR '*NODEFAULT'	制約なし	制約なし	制約なし
*PSSR BEGSR.. ENDSR '*ENDCOMP' または ENDSR '*CANCL'	制約なし	制約なし	コンポーネントは、ネストされたアクション・サブルーチンから終了することはできません
*PSSR BEGSR.. ENDSR '*ENDAPPL'	制約なし	制約なし	コンポーネントは、ネストされたアクション・サブルーチンから終了することはできません

第 5 章 エラーおよび例外処理

例外 / エラーは、2 つのクラス (プログラムとファイル) に分類されます。ファイルとプログラムの例外 / エラーについての情報によって、VARPG プログラムがファイル情報データ構造とプログラム状況データ構造を個々に使用することができるようになります。これらのタイプの例外 / エラーを処理するよう、ファイルとプログラムの例外 / エラー・サブルーチンを指定することができます。この章では、ファイル、プログラム、およびコンポーネントにエラーおよび例外処理について説明されています。

ファイルの例外 / エラー

以下は、幾つかのファイル例外 / エラーの例です: 不定レコード・タイプ、トリガー・プログラムのエラー、クローズ・ファイルへの入出力操作、装置エラー、および配列 / テーブル・ロード・シーケンス・エラー。これらは、以下の方法のいずれかで処理できます。

- 命令コード拡張 'E' を指定することができます。指定すると、演算命令が開始される前に、この拡張によって %ERROR および %STATUS 組み込み関数がゼロを戻すように設定されます。例外 / エラーが演算命令中に起こった場合には、この演算命令後に %ERROR が '1' を戻し、%STATUS がファイル状況を戻します。オプショナル・ファイル情報データ構造は、例外 / エラー情報で更新されます。%ERROR および %STATUS をテストすることによって、アクションを実行するよう決定することができます。
- 命令コードの演算仕様の 73 および 74 桁に、標識を指定することができます。この標識は、指定した演算命令の処理中に例外 / エラーが起こった場合に、オンに設定されます。オプショナル・ファイル情報データ構造は、例外 / エラー情報で更新されます。この標識をテストすることによって、アクションを実行するよう決定することができます。
- ON-ERROR グループは、MONITOR ブロックで処理されたステートメントのエラーを処理するために使用できます。ステートメントの処理時にエラーが起こった場合には、制御が適切な ON-ERROR グループに渡されます。
- ファイル例外 / エラー・サブルーチンを指定することができます。このサブルーチンは、制御を受け取るサブルーチンの名前をもつ、ファイル仕様上の INFSR キーワードによって定義されます。ファイル例外 / エラーに関する情報は、ファイル仕様上の INFDS キーワードで指定されたファイル情報データ構造を通して使用することができます。プログラムまたはファイル状況の最新の値セットを戻す、%STATUS 組み込み関数を使用することもできます。ファイルを指定した場合には、%STATUS が、指定されたファイルの INFDS *STATUS フィールドに含まれた値を戻します。
- 標識、'E' 拡張、MONITOR ブロック、またはファイル例外 / エラー・サブルーチンが存在しない場合には、どのファイル例外 / エラーも VisualAge RPG デフォルト・エラー処理プログラムによって処理されます。

ファイル情報データ構造

ファイル情報データ構造は、ファイル・エラーについての情報を提供します。ファイル例外、エラー、およびファイル・フィードバック情報がプログラムで使用できるよう、ファイル情報データ構造 (INFDS) をファイルごとに定義することができます。このデータ構造は、ファイルごとに固有でなければなりません。これには、以下のフィードバック情報が含まれます。

- ファイル・フィードバック (1 ～ 80 桁)
- オープン・フィードバック (81 ～ 240 桁)
- 入出力フィードバック (241 ～ 366 桁)
- 装置特定フィードバック (367 桁)

注: INFDS の長さは、INFDS で宣言されたフィールドによって異なります。

ファイル・フィードバック情報

ファイル・フィードバック情報は、INFDS 内の 1 桁から開始され、80 桁で終了します。これには、以下を含む、VisualAge RPG プログラムに特定のファイルについてのデータが含まれています。

- 例外またはエラーが起こったファイルの名前
- 例外またはエラーが起こった時に処理されているレコード、あるいは例外またはエラーの原因となるレコード
- 例外またはエラーが起こった時に処理されている最後の演算命令
- 状況コード
- 例外またはエラーが起こったルーチン

注: ファイル・フィードバック・セクションの上書きによって、後続のエラー処理で予期しない結果となる場合があるので、推奨されません。

ファイル・フィードバック・セクションでよく使用される幾つかのサブフィールドの位置は、特殊キーワードによって定義されます。表 7では、これらのキーワードについて要約されています。

表 7. INFDS のファイル・フィードバック情報

開始 (26 ～ 32 桁)	終了 (33 ～ 39 桁)	フォーマット	長さ	キーワード	情報
1	8	文字	8	*FILE	ファイル名の最初の 8 文字
9	9	文字	1		オープン指示 (1 = オープン)
10	10	文字	1		ファイルの終わり (1 = ファイルの終わり)
11	15	ゾーン 10 進数	5,0	*STATUS	状況コード。 50 ページの『ファイル状況コード』 を参照してください。

表 7. INFDS のファイル・フィードバック情報 (続き)

開始 (26 ～ 32 桁)	終了 (33 ～ 39 桁)	フォーマット	長さ	キーワード	情報
16	21	文字	6	*OPCODE	命令コード。最初の 5 桁 (左寄せ) は、命令コードの文字表現を使用して、演算命令のタイプを指定します。たとえば、READE が処理されていた場合には、READE が左端の 5 桁に入れられます。 6 文字の名前をもつ命令コードは、5 文字に短縮されます。 **DELETE** DELET **EXCEPT** EXCPT **READPE** REDPE **UNLOCK** UNLCK **UPDATE** UPDAT 残りの桁には、次の 1 つが含まれています。 **F** 最後の演算命令がファイル名に指定されています。 **R** 最後の演算命令がレコードに指定されています。 **I** 最後の演算命令が暗黙ファイル操作に指定されています。
22	29	文字	8	*ROUTINE	プロシージャー名の最初の 8 文字、あるいは呼び出しがプロシージャー・ポインターの場合にはゼロ。
30	37	文字	8		ソース・リストの行番号
38	42	ゾーン 10 進数	5,0		SPECIAL ファイル上のエラーについてのユーザー指定の理由

表 7. INFDS のファイル・フィードバック情報 (続き)

開始 (26 ～ 32 桁)	終了 (33 ～ 39 桁)	フォーマット	長さ	キーワード	情報
38	45	文字	8	*RECORD	プログラム記述ファイルの場合、レコード識別標識はフィールドに左寄せで入れられます。残りの 6 桁はブランクで埋められます。 外部記述ファイルの場合、例外またはエラーが起こった時に処理されているレコードの名前の最初の 8 文字。
46	52	文字	7		マシンまたはシステム・メッセージ番号
53	66	文字	14		未使用

ファイル・フィードバック域の内容の完全記述については、Web サイト **http://www.ibm.com/eserver/iseries/infocenter** の **Information Center** にある「データベースおよびファイル・システム」 カテゴリーの「*DB2*® ユニバーサル・データベース™」 セクションを参照してください。

INFDS ファイル・フィードバックの例: ファイル・フィードバック・セクションのフィールドを含む INFDS を定義するには、以下の項目を指定してください。

- ファイル情報データ構造の名前をもつ、ファイル仕様上の INFDS キーワードを指定します。
- 定義仕様で使用したいファイル情報データ構造およびサブフィールドを指定します。
- 定義仕様上の FROM フィールド (26 ～ 32 桁) に左寄せされた特殊キーワードを指定するか、あるいは FROM フィールド (26 ～ 32 桁) と TO フィールド (33 ～ 39 桁) にフィールドの位置を指定します。

```
FFilename++IT.A.FRlen+......A.Device+.Keywords++++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    IF   E             DISK    INFDS(FILEFBK)    REMOTE
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++Comments++++++++++
DFILEFBK          DS
D FILE              *FILE                                                  * File name
D OPEN_IND               9      9                                          * File open?
D EOF_IND               10     10                                          * File at eof?
D STATUS            *STATUS                                                * Status code
D OPCODE            *OPCODE                                                * Last Opcode
D ROUTINE           *ROUTINE                                               * RPG Routine
D LIST_NUM              30     37                                          * Listing line
D SPCL_STAT             38     42S 0                                       * SPECIAL status
D RECORD            *RECORD                                                * Record name
D MSGID                 46     52                                          * Error MSGID
```

図 10. ファイル・フィードバック情報を使用した INFDS のコーディングの例

注: キーワードはラベルではないので、サブフィールドのアクセスには使用できません。短い項目は、右側をブランクで埋め込まれます。

オープン・フィードバック情報

ファイル情報データ構造の 1 - 240 桁には、オープン・フィードバック情報が含まれています。この域の内容は、INFDS と関連したファイルがオープンされる時には常にオープン・フィードバック・セクションにコピーされます。これには、複数メンバー処理ファイル上の読み取り操作の結果としてオープンされるメンバーが含まれます。

注: オープン・フィードバック情報は、印刷装置ファイルには指定されませんが、装置フィードバック情報は印刷装置ファイルに指定されます。オープン・フィードバック域の内容の完全記述については、Web サイト **http://www.ibm.com/eserver/iseries/infocenter** の **Information Center** にある「データベースおよびファイル・システム」 カテゴリーの「*DB2 ユニバーサル・データベース*」 セクションを参照してください。

INFDS オープン・フィードバックの例: オープン・フィードバック・セクションのフィールドを含む INFDS を定義するには、以下の項目を指定してください。

- ファイル情報データ構造の名前をもつ、ファイル仕様上の INFDS キーワードを指定します。
- 定義仕様で使用したいファイル情報データ構造およびサブフィールドを指定します。
- INFDS に組み込みたいフィールドを決定するには、**Information Center** にある「データベースおよびファイル・システム」 カテゴリーの「*DB2 ユニバーサル・データベース*」 セクションの情報を使用してください。 オープン・フィードバック・セクションのサブフィールドを指定する「開始」および「終了」位置 (定義仕様の 26 ～ 32 桁および 33 ～ 39 桁) を計算するには、**Information Center** で指定された「オフセット」、「データ・タイプ」、および「長さ」を使用して以下の計算を行なってください。

```
From = 81 + Offset
To = From - 1 + Character_Length
Character_Length = Length (in bytes)
```

入出力フィードバック情報

ファイル情報データ構造の 41 - 366 桁が、入出力フィードバック情報に使用されます。ファイル共通入出力フィードバック域の内容は、ファイルの POST 後にだけ入出力フィードバック・セクションにコピーされます。詳細については、 638 ページの『POST (転記)』 を参照してください。

入出力フィードバック域の内容の記述は、**Information Center** にある「データベースおよびファイル・システム」 カテゴリーの「*DB2 ユニバーサル・データベース*」 セクションで見つかります。

注: 入出力フィードバック情報は、印刷装置ファイルには指定されませんが、装置特定フィードバック情報は印刷装置ファイルに指定されます。

INFDS 入出力フィードバックの例: オープン・フィードバック・セクションのフィールドを含む INFDS を定義するには、以下の項目を指定してください。

- ファイル情報データ構造の名前をもつ、ファイル仕様上の INFDS キーワードを指定します。
- 定義仕様で使用したいファイル情報データ構造およびサブフィールドを指定します。

- INFDS に組み込みたいフィールドを決定するには、**Information Center** にある「データベースおよびファイル・システム」 カテゴリーの「*DB2* ユニバーサル・データベース」 セクションの情報を使用してください。 入出力フィードバック・セクションのサブフィールドを指定する「開始」および「終了」位置 (定義仕様の 26 ～ 32 桁および 33 ～ 39 桁) を計算するには、**Information Center** で指定された「オフセット」、「データ・タイプ」、および「長さ」を使用して以下の計算を行なってください。

  ```
  From = 241 + Offset
  To = From - 1 + Character_Length
  Character_Length = Length (in bytes)
  ```

  たとえば、ファイルの装置クラスについては、以下が **Information Center** により提供されます。

  ```
  Offset = 30
  Data Type is character
  Length = 2
Therefore,
  From = 241 + 30 = 271,
  To = 271 - 1 + 2 = 272.
  ```

図 11のサブフィールド DEV_CLASS を参照してください。

```
FFilename++IT.A.FRlen+......A.Device+.Keywords++++++++++++++++++++++++++++++Comments++++++++++++
FMYFILE    IF   E             DISK    INFDS(MYIOFBK)    REMOTE
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++Comments++++++++++
DMYIOFBK          DS
D                                                                          * 241-242 not used
D WRITE_CNT             243    246B 0                                      * Write count
D READ_CNT              247    250B 0                                      * Read count
D WRTRD_CNT             251    254B 0                                      * Write/read count
D OTHER_CNT             255    258B 0                                      * Other I/O count
D OPERATION             260    260                                         * Current operation
D IO_RCD_FMT            261    270                                         * Rcd format name
D DEV_CLASS             271    272                                         * Device class
D IO_PGM_DEV            273    282                                         * Pgm device name
D IO_RCD_LEN            283    286B 0                                      * Rcd len of I/O
```

図 11. 入出力フィードバック情報のコーディング

装置特定フィードバック情報

ファイル情報データ構造の装置特定フィードバック情報は、INFDS の 367 桁で開始されます。これには、データベースまたは印刷装置に特定の入出力フィードバック情報が含まれます。

装置特定フィードバック情報が必要な時の INFDS の長さは変数で、ファイルの装置タイプが変数かどうか、およびファイルがキー順かどうか (DISK ファイルの場合) によって異なります。

外部記述 DISK ファイルの場合、INFDS は、401 桁から始まるファイルの最長キーを保持するのに、少なくとも十分長さです。

ファイルの装置特定入出力フィードバック域の内容は、ファイルの POST 後にだけ INFDS の装置特定フィードバック・セクションにコピーされます。詳細については、 638 ページの『POST (転記)』 を参照してください。

INFDS 装置特定フィードバックの例: 装置フィードバック・セクションのフィールドを含む INFDS を定義するには、以下の項目を指定してください。

- ファイル情報データ構造の名前をもつ、ファイル仕様上の INFDS キーワードを指定します。
- 定義仕様で使用したいファイル情報データ構造およびサブフィールドを指定します。
- INFDS に組み込みたいフィールドを決定するには、**Information Center** にある *「データベースおよびファイル・システム」* カテゴリーの *「DB2 ユニバーサル・データベース」* セクションの情報を使用してください。 装置特定フィードバックのサブフィールドを指定する「開始」および「終了」位置 (定義仕様の 26 〜 32 桁および 33 〜 39 桁) を計算するには、**Information Center** で指定された「オフセット」、「データ・タイプ」、および「長さ」を使用して以下の計算を行なってください。

```
From = 367 + Offset
To = From - 1 + Character_Length
Character_Length = Length (in bytes)
```

たとえば、データベース・ファイルの相対レコード番号について、**Information Center** は以下を使用します。

```
   Offset = 30
   Data Type is binary
   Length = 4
Therefore,
   From = 367 + 30 = 397,
   To = 397 - 1 + 4 = 400.
```

図 12の DBFBK データ構造のサブフィールド、DB_RRN を参照してください。

```
FFilename++IT.A.FRlen+......A.Device+.Keywords++++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    IF   E             DISK    INFDS(DBFBK)    REMOTE
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++Comments++++++++++
DDBFBK            DS
D FDBK_SIZE             367    370B 0                                    * Size of DB fdbk
D JOIN_BITS             371    374B 0                                    * JFILE bits
D LOCK_RCDS             377    378B 0                                    * Nbr locked rcds
D POS_BITS              385    385                                       * File pos bits
D DLT_BITS              384    384                                       * Rcd deleted bits
D NUM_KEYS              387    388B 0                                    * Num keys (bin)
D KEY_LEN               393    394B 0                                    * Key length
D MBR_NUM               395    395B 0                                    * Member number
D DB_RRN                397    400B 0                                    * Relative-rcd-num
D KEY                   401   2400                                       * Key value (max
D                                                                        *   size 2000)
```

図 12. データベース特定フィードバック情報を使用した INFDS のコーディングの例

ブロック化の考慮事項: 入出力特定フィードバック域のフィールド、および装置特定フィードバック情報域のフィールド (大抵の場合) は、レコードがブロック化および非ブロック化されたファイル (フィードバック情報および相対レコード番号を除く) に対する演算命令ごとには更新されません。POST 演算命令が実行されると、この例外が起こります。この場合、入出力特定および装置特定フィードバック域のす

べてのフィールドが更新されます。POST 演算命令では、ブロックの最後のレコードではなく、現行レコードでキーおよび相対レコード番号が更新されます。

ファイル例外およびエラー・サブルーチン (INFSR)

ファイル例外またはエラーに従って制御を受け取るサブルーチンを識別するには、そのサブルーチンの名前で「ファイル記述」仕様上に INFSR キーワードを指定します。サブルーチン名は *PSSR であり、これはプログラム例外 / エラー・サブルーチンがこのファイルで例外およびエラーの制御を得ることを示します。

ファイル例外 / エラー・サブルーチンは、73 ～ 74 桁目に標識が指定されておらず、 (E) 拡張がなく、またエラーを処理できる MONITOR グループのモニター・ブロックにないファイル演算命令で例外またはエラーが起こったときに制御を受け取ります。ファイル例外 / エラー・サブルーチンは、EXSR 命令コードによっても実行できます。ファイル例外 / エラー・サブルーチンでは、任意の命令コードが使用できます。BEGSR 演算命令の演算項目 1 および EXSR 演算命令の演算項目 2 には、制御を受け取るサブルーチンの名前 (ファイル仕様上の INFSR キーワードで指定されたのと同じ名前) が含まれていなければなりません。ENDSR 演算命令は、ファイル例外 / エラー・サブルーチンの最後の仕様でなければならず、以下のように指定されなければなりません。

位置	項目
6	C
7 ～ 11	ブランク
12 ～ 25	サブルーチン内の GOTO 使用で使用されるラベルを入れることができます。
26 ～ 35	ENDSR
36 ～ 49	サブルーチンの処理後に制御が戻される場所を指定するオプショナル項目。この項目は、以下の戻りポイントの 1 つを指定する値をもつ文字フィールド、リテラル、または配列エレメントでなければなりません。

注: 戻りポイントがリテラルとして指定されている場合には、これらをアポストロフィで囲まなければなりません。これらが名前定数として指定されている場合には、この定数は文字でなければならず、先行ブランクのない戻りポイントしか入れられません。これらがフィールドまたは配列エレメントで指定されている場合には、この値はフィールドまたは配列エレメントで左寄せされなければなりません。

***DEFAULT**
現行アクション・サブルーチンから制御を戻し、現行イベントと関連したデフォルト処理を実行します。

***NODEFAULT**
現行アクション・サブルーチンから制御を戻します。デフォルト処理は行なわないでください。処理がこのポイントに届いた時に LR がオンの場合には、コンポーネントが終了され、*DEFAULT および *NODEFAULT 戻りポイントは無視されます。

***CANCL**
コンポーネントを異常終了します。

***ENDAPPL**
すべての現行アクティブ・コンポーネントを終了して、アプリケーションを終了します。

***ENDCOMP**
コンポーネントを異常終了します。

ブランク
デフォルト・エラー処理プログラムに制御を戻します。演算項目 2 がブランクの値の時、および演算項目 2 が指定されていない時に、これが適用されます。サブルーチンが EXSR 演算命令で呼び出され、演算項目 2 がブランクの場合には、次に続く命令に制御が戻されます。ブランクは、実行時のみ有効です。

50-76 ブランク。

ファイル例外 / エラー・サブルーチンを指定する場合には、次の点に留意してください。

- EXSR 演算命令の演算項目 2 でサブルーチンの名前を指定することによって、ファイル例外 / エラー・サブルーチンを明示的に呼び出すことができます。
- ファイル例外 / エラー・サブルーチンの ENDSR 演算命令が実行されると、演算項目 2 のフィールドまたは配列エレメントがブランクにリセットされます。サブルーチンの処理中にこのフィールドに値を入れない場合には、デフォルト・エラー処理プログラムは、サブルーチンが EXSR 演算命令によって呼び出されない限り、サブルーチンの処理に従って制御を受け取ります。演算項目 2 がブランクに設定されているので、起こった例外またはエラーに最適なサブルーチン内の戻りポイントを指定することができます。サブルーチンが EXSR 演算命令で呼び出された場合には、EXSR 演算命令の後に続く次の命令に制御が戻されます。 ファイル例外 / エラー・サブルーチンは、複数のファイルのエラーを処理することができます。
- プログラムの開始または終わりの間にファイル例外またはエラーが起こった場合には、ユーザー作成ファイル例外 / エラー・サブルーチン (INFSR) にではなく、デフォルト・エラー処理プログラムに制御が渡されます。
- ファイル例外 / エラー・サブルーチンは、ファイル例外またはエラーが起こった時にいつも制御を受け取ることができるので、エラー状態のファイルで入出力操作が処理される場合には、サブルーチンが実行されている一方で、例外またはエラーが起こります。サブルーチンが実行されている一方で、すでにエラー状態のファイルで例外 / エラーが起こる場合には、このサブルーチンが再度呼び出されます。これによって、この問題を回避するようこのサブルーチンをコーディングしない限り、プログラム・ループが起こります。このようなプログラム・ループを回避する 1 つの方法は、サブルーチンで最初スイッチを設定することです。これがサブルーチンを通して最初でない場合には、LR 標識をオンに設定して、次のとおり RETURN 演算命令を出してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* If INFSR is already handling the error, exit.
C     ERRRTN        BEGSR
C     SW            IFEQ      '1'
C                   SETON                                        LR
C                   RETURN
C* Otherwise, flag the error handler.
C                   ELSE
C                   MOVE      '1'           SW
C                     :
C                     :
C                     :
C                   ENDIF
C* End error processing.
C                   MOVE      '0'           SW
C                   ENDSR
```

注: 入出力エラーが起こった後に、ファイルの処理を続行することはできません。続行するには、そのファイルに CLOSE 演算命令を出してから OPEN 演算命令を出す必要があります。

ファイル状況コード

99 より大きいサブフィールド位置 *STATUS に入れられたコードは、例外またはエラーと見なされます。状況コードが 99 より大きい時には、エラー標識 (73 と 74 桁に指定されている場合) がオンに設定されるか、あるいは%ERROR 組み込み関数 ('E' 拡張が指定されている場合) が '1' を戻すよう設定されます。そうでない場合には、ファイル例外 / エラー・サブルーチンが制御を受け取ります。 位置 *STATUS は、各ファイル操作後に更新されます。

%STATUS 組み込み関数を使用して、例外 / エラーについての情報を取得することができます。これは、プログラムまたはファイル状況の最新の値セットを戻します。ファイルを指定した場合には、%STATUS が、指定されたファイルの INFDS *STATUS フィールドに含まれた値を戻します。

以下のテーブルには、ファイル情報データ構造のサブフィールド位置 *STATUS に入れられたコードが要約されています。

表 8. 通常コード

コード	装置[1]	RC	条件
00000			例外 / エラーなし
00011	D		読み取り (入力) でのファイルの終わり
00012	D		CHAIN、SETLL、および SETGT 演算命令で該当レコードなし条件
00014			ローカル・ファイルの出力レコードが切り捨てられる
00015			ローカル・ファイルの入力レコードが切り捨てられる
注: [1]『装置』 は、条件が適用される装置を参照します。以下の省略形が使用されます: P = PRINTER; D = DISK; SP = SPECIAL			

表 9. 例外 / エラー・コード

コード	装置[1]	RC	条件
01011	D		未定義のレコード・タイプ (入力レコードがレコード識別標識と一致しない)
01021	D		すでに存在するレコードを書き込もうとした (使用されるファイルには固有キーがあり、これが重複している)
01022	D		参照制約がファイル・メンバーで見つかった
01041	n/a		配列 / テーブルのロード・シーケンス・エラー
01042	n/a		配列 / テーブルのロード・シーケンス・エラー
01051	n/a		配列 / テーブルの項目が多すぎる
01211	all		クローズ・ファイルへの入出力操作
01215	all		すでにオープンされているファイルに OPEN が出された
01216[2]	all		暗黙 OPEN/CLOSE 演算命令でのエラー
01217[2]	all		明示 OPEN/CLOSE 演算命令でのエラー
01218	D		レコードがすでにロックされている
01221	D		前の読み取りなしで更新操作が試行された
01222	D		参照制約エラーによりレコードを割り振ることができない
01231	SP		SPECIAL ファイルでのエラー
01235	P		PRTCTL スペースまたはスキップ項目でエラー
01299	D,P		その他の入出力エラーが見つかった。ローカル・ファイルの場合、このメッセージには以下の ID の 1 つが含まれます。 • *LF0001: ファイルをオープンできなかった • *LF0002: ファイルをクローズできなかった • *LF0003: 予期しない入出力結果 • *LF0004: ファイル・ポインターを設定できなかった • *LF0005: 読み取り失敗 • *LF0006: 書き込み失敗 • *LF0007: ファイルのサイズを判別できなかった • *LF0008: ファイルをサイズ変更できなかった • *LF0009: ファイルをコピーできなかった • *LF0010: ファイルを削除できなかった • *LF0011: コンパイル時にローカルとして指定されたファイルが、実行時にリモート・ファイルとして見つかった

表 9. 例外 / エラー・コード (続き)

コード	装置[1]	RC	条件
注: [1]『装置』 は、条件が適用される装置を参照します。以下の省略形が使用されます: P = PRINTER; D = DISK; SP = SPECIAL; [2]オープンまたはクローズ演算命令中に起こったエラーは、*STATUS 値 1216 または 1217 となります。			

プログラム例外およびエラー

プログラム例外およびエラーの幾つかの例は、次のとおりです: 0 除算、負数の SQRT、無効な配列指数、CALL でのエラー、呼び出し先プログラムから戻されたエラー、およびストリング演算命令の開始桁または長さが範囲外。これらは、以下の方法のいずれかで処理できます。

- 一定の命令コードの演算仕様の 73 および 74 桁に、標識を指定することができます。この標識は、指定した演算命令の処理中に例外またはエラーが起こった場合に、オンに設定されます。オプショナル・プログラム状況データ構造は、例外/エラー情報で更新されます。この標識をテストすることによって、アクションの実行を決定することができます。
- 幾つかの命令コードで、命令コード拡張 'E' を指定することができます。指定すると、演算命令が開始される前に、この拡張によって %ERROR および %STATUS 組み込み関数がゼロを戻すように設定されます。例外 / エラーが演算命令中に起こった場合には、この演算命令後に %ERROR が '1' を戻し、%STATUS がプログラム状況を戻します。オプショナル・プログラム状況データ構造は、例外 / エラー情報で更新されます。%ERROR および %STATUS をテストすることによって、アクションを実行するよう決定することができます。
- ON-ERROR グループは、MONITOR ブロックで処理されたステートメントのエラーを処理するために使用できます。ステートメントの処理時にエラーが起こった場合には、制御が適切な ON-ERROR グループに渡されます。
- BEGSR 演算命令の演算項目 1 で *PSSR をコーディングすることによって、プログラム例外 / エラー・サブルーチンを指定することができます。プログラム例外 / エラーに関する情報は、定義仕様のデータ構造ステートメントの 23 桁に S で指定されたプログラム状況データ構造を通して、使用することができます。
- 標識、'E' 拡張、モニター・ブロック、あるいはプログラム例外 / エラー・サブルーチンが存在しない場合には、プログラムの例外およびエラーがデフォルト・エラー処理プログラムによって処理されます。

プログラム状況データ構造

プログラム例外およびエラー情報が VisualAge RPG プログラムで使用できるよう、プログラム状況データ構造を定義することができます。

データ構造は、データ構造ステートメントの 23 桁で S によって、プログラム状況データ構造として定義されます。プログラム状況データ構造には、起こったプログラム例外またはエラーについての情報を提供するサブフィールドが含まれています。これらのサブフィールドの位置は、特殊キーワード、または事前定義された「開始」および「終了」位置によって定義されます。サブフィールドをアクセスするには、各サブフィールドに名前を割り当てます。キーワードを、26 ～ 39 桁に左寄せで指定しなければなりません。

表 10では、データ構造のサブフィールドのレイアウト、およびそのサブフィールドの「開始」と「終了」位置が提供されます。

表 10. プログラム状況データ構造の内容

開始 (26 ～ 32 桁)	終了 (33 ～ 39 桁)	フォーマット	長さ	キーワード	情報
1	10	文字	10	*PROC	コンポーネント名
11	15	ゾーン 10 進数	5,0	*STATUS	状況コード
16	20	ゾーン 10 進数	5,0		以前の状況コード
21	28	文字	8		ソース・リストの行番号
29	36	文字	8	*ROUTINE	例外またはエラーが起こったルーチンの名前。このサブフィールドは、*STATUS サブフィールドが非ゼロ値で更新された時にのみ、ルーチンの開始時またはプログラム呼び出し後に更新されます。以下の名前でルーチンが識別されます。 ***INIT** プログラム初期化 ***TERM** プログラム終了 ***ROUTINE** 呼び出されるプログラムまたはプロシージャーの名前 (最初の 8 文字)。
37	39	ゾーン 10 進数	3,0	*PARMS	呼び出し側プログラムからこのプログラムに渡されるパラメーターの数。
40	42	文字	3		例外タイプ: OS/400® システム例外の場合には CPF、マシン例外の場合には MCH、あるいは実行時ルーチンからのエラー戻りコードの場合には *RT。Windows 例外の場合、このフィールドには *EX が入れられます。
43	46	文字	4		例外番号: CPF 例外の場合、このフィールドには CPF メッセージ番号が入れられます。マシン例外の場合、これにはマシン例外番号が入れられます。Windows 例外の場合、このフィールドには 2 進 9,0 形式の例外番号が入れられます。VisualAge RPG 実行時ルーチンのエラー戻りコードも 2 進 9,0 形式でこのフィールドに入れられます。
47	90		44		予約済み
91	170	文字	80		例外データを検索しました。OS/400 メッセージがこのサブフィールドに入れられます。
171	190		20		予約済み
191	198	文字	8		ジョブがシステムに入力された日付 (*DATE 形式)。この値によって表示される日付は、270 - 275 桁で表示されるのと同じ日付です。

表 10. プログラム状況データ構造の内容 (続き)

開始 (26 ～ 32 桁)	終了 (33 ～ 39 桁)	フォーマット	長さ	キーワード	情報
199	200	ゾーン 10 進数	2,0		4 桁の年の最初の 2 桁。*YEAR の最初の 2 桁と同じです。このフィールドは、270 - 275 桁の日付の世紀部分に適用されます。たとえば、日付 1999-06-27 の場合、UDATE は 990627 となり、この世紀フィールドは 19 となります。このフィールドの値は、270 - 275 桁の値と一緒に、191 -198 桁の値の結合情報をもちます。**注:** この世紀フィールドは、276 - 281 桁または 288 - 293 桁の日付には適用されません。
201	208	文字	8		最後のファイル操作が行なわれたファイルの名前 (エラーが起こった時にのみ更新)
209	243	文字	35		最後に使用されたファイルの状況情報。この情報には、状況コード、命令コード、VisualAge RPG ルーチン名、ソース・リストの行番号、およびレコード名が含まれます。これは、エラーが起こった時にしか更新されません。**注:** 命令コードは、INFDS の *OPCODE と同じ形式です。
244	253		10		予約済み
254	263	文字	10		リモート・ファイル・オープン演算命令の iSeries ホスト・サインオン・ユーザー ID。この値は、別のホストが異なるサインオン・ユーザー ID でアクセスされた時にだけ更新されます。
264	269		10		予約済み
270	275	ゾーン 10 進数	6,0		プログラムがシステムで実行を開始した日付 (UDATE 形式)。(UDATE は、この日付から派生します。) UDATE の記述については、8 ページの『ユーザー日付の特殊語』 を参照してください。これは、「ジョブ日付」としてよく知られてます。この値によって表示される日付は、191 - 198 桁で表示されるのと同じ日付です。
276	281	ゾーン 10 進数	6,0		実行しているプログラムの日付 (UDATE 形式のシステム日付)。この値の年部分が 40 - 99 の場合には、日付は 1940 - 1999 年です。そうでない場合には、日付は 2000 - 2039 となります。199 - 200 桁の「世紀」値は、このフィールドには適用されません。
282	287	ゾーン 10 進数	6 (小数点以下の桁数ゼロ)		実行しているプログラムの時刻 (形式 hhmmss)

表 10. プログラム状況データ構造の内容 (続き)

開始 (26 ～ 32 桁)	終了 (33 ～ 39 桁)	フォーマット	長さ	キーワード	情報
288	293	文字	6		プログラムがコンパイルされた日付 (UDATE 形式のシステム日付)。この値の年部分が 40 - 99 の場合には、日付は 1940 - 1999 年です。そうでない場合には、日付は 2000 - 2039 となります。 199 - 200 桁の「世紀」値は、このフィールドには適用されません。
294	299	文字	6		プログラムがコンパイルされた時刻 (形式 hhmmss)
300	303	文字	4		コンパイラーのレベル
304	313	文字	10		ソース・ファイル名 (最初の 10 文字)
314	429		116		予約済み

プログラム状況コード

99 より大きいサブフィールド位置 *STATUS に入れられたコードは、例外またはエラー条件と見なされます。状況コードが 99 より大きい時には、エラー標識 (73 ～ 74 桁に指定されている場合) がオンに設定されるか、%ERROR 組み込み関数 ('E' 拡張が指定されている場合) が '1' を戻すよう設定されるか、制御が MONITOR ブロック内の適切な ON-ERROR グループに渡されます。そうでない場合には、プログラム例外 / エラー・サブルーチンが制御を受け取ります。*STATUS は、例外またはエラーが起こった時に更新されます。

%STATUS 組み込み関数は、プログラムまたはファイル状況の最新の値セットを戻します。

以下のコードは、プログラム状況データ構造のサブフィールド位置 *STATUS に入れられます。

通常コード:

コード 条件

00000 例外 / エラーが起こらなかった

00031 コンポーネントが終了中。RETURN または ENDACT 演算命令が実行された時の LR 標識

00032 コンポーネントの明示終了 (STOP コンポーネント) の結果としてコンポーネントが終了中

00033 コンポーネントの暗黙終了 (STOP コンポーネントの親または祖父母) の結果としてコンポーネントが終了中

00034 別のコンポーネントからの明示終了要求 (STOP コンポーネント) の結果としてコンポーネントが終了中

00035 別のコンポーネントからの暗黙終了要求 (STOP コンポーネントの親) の結果としてコンポーネントが終了中

00050 変換によって置換された。

例外 / エラー・コード:

コード 条件

00100 値がストリング演算命令の範囲外

00101 負の平方根

00102 0 除算

00103 中間結果が、結果を入れるのに十分な大きさでない

00104 浮動アンダーフロー。中間値が、中間結果フィールドに入れるには小さすぎる。

00112 無効な日付、時刻、タイム・スタンプ値。

00113 日付オーバーフローまたはアンダーフロー。(たとえば、日付演算の結果が、*HIVAL より大きいか、あるいは *LOVAL より小さくなっています)

00114 日付マッピング・エラー。ここで日付は、4 文字の年から 2 文字の年にマップされていて、日付範囲は 1940-2039 ではありません。

00115 可変長文字またはグラフィックス・フィールドに無効な現行長があります。

00120 テーブルまたは配列が順序外です

00121 配列指数が無効です

00122 OCCUR が範囲外です

00123 Reset aプログラムの初期化ステップ中にリセットが試行されました

00202 呼び出し先プログラムまたはプロシージャーが失敗しました

00211 呼び出し側プログラムまたはプロシージャーでエラー

00221 呼び出し先プログラムがこれに渡されていないパラメーターを使用しようとしました

00222 ポインターまたはパラメーター・エラー

00301 メソッド呼び出しでクラスまたはメソッドが見つからないか、メソッド呼び出しでエラー。

00302 Java ネイティブ・メソッドへの入り口点で Java 配列を RPG パラメーターに変換中にエラー。

00303 RPG ネイティブ・メソッドからの出口点で RPG パラメーターを Java 配列に変換中にエラー。

00304 Java メソッド呼び出しの準備で Java パラメーターを Java 配列に変換中にエラー。

00305 Java メソッドの後で Java 配列を RPG パラメーターまたは戻り値に変換中にエラー。

00306 RPG 戻り値を Java 配列に変換中にエラー。

00333 DSPLY 演算命令でエラー

00401 IN/OUT で指定されたデータ域が見つかりません

00411 データ域タイプまたは長さが一致しません

00412 データ域が出力用にロックされていません

00413 IN/OUT 演算命令でエラー

00414 ユーザーにデータ域を使用する権限がありません

00415 ユーザーにデータ域を変更する権限がありません

00421 UNLOCK 演算命令でエラー

00431 データ域が別のプログラムで以前にロックされています

00432 データ域が同じ処理でプログラムによってロックされています

00451 2 つの CCSID 間の変換はサポートされていません。

00501 ソート・シーケンスの検索に失敗しました

00802 コミットメント制御がアクティブではありません

00803 ロールバック操作が失敗しました

00804 COMMIT 演算命令でエラーが起こりました

00805 ROLBK 演算命令でエラーが起こりました

00907 10 進データ・エラー (桁または符号が無効)

00940 ホスト・サービスでエラーが起こりました

00970 プログラムを生成するのに使用されるコンパイラーのレベル番号が、VisualAge RPG 実行時サブルーチンのレベル番号と一致しません。

01400 属性名が無効です

01401 SHOWWIN 演算命令がオープン・ウィンドウで試行されました

01402 パーツ名がアプリケーションで見つかりません

01403 新規属性値が有効範囲外です

01404 属性アクセス・タイプが演算命令で無効です

01405 イベント属性のデータ・タイプが、演算命令と互換性がありません

01406 無効なメッセージ ID

01407 属性のデータ・タイプが、演算命令と互換性がありません

01408 リソースが不十分です

01410 START 演算命令が失敗しました

01411 STOP 演算命令が失敗しました

01420 サブファイル演算命令でエラーが起こりました

01421 サインオン・ダイアログは取り消されました。

01422 操作されるパーツが入っているコンポーネントが開始されていません。

1601 1 つまたは複数の DB2 製品のダイナミック・リンク・ライブラリー (DLL) が見つかりません。

08888 再帰エラー

09001 エラー標識または *PSSR がありません

09998 VisualAge RPG コンパイラーまたは実行時サブルーチンで内部障害

09999 システム・ルーチンでプログラム例外。

プログラム状況データ構造の例

プログラムでプログラム状況データ構造 (PSDS) を指定するには、定義仕様で使用したいプログラム状況データ構造およびサブフィールドをコーディングします。

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++Comments++++++++++
DMYPSDS          SDS
D PROC_NAME         *PROC                                               * Component name
D PGM_STATUS        *STATUS                                             * Status code
D PRV_STATUS             16     20S 0                                   * Previous status
D LINE_NUM               21     28                                      * Src list line num
D ROUTINE           *ROUTINE                                            * Routine name
D PARMS             *PARMS                                              * Num passed parms
D EXCP_TYPE              40     42                                      * Exception type
D EXCP_NUM               43     46                                      * Exception number
D*
D EXCP_DATA              91    170                                      * Exception data
D*
D DATE                  191    198                                      * Date (*DATE fmt)
D YEAR                  199    200S 0                                   * Year (*YEAR fmt)
D LAST_FILE             201    208                                      * Last file used
D FILE_INFO             209    243                                      * File error info
D*
D JOB_DATE              270    275S 0                                   * Date (UDATE fmt)
D RUN_DATE              276    281S 0                                   * Run date (UDATE)
D RUN_TIME              282    287S 0                                   * Run time (UDATE)
D CRT_DATE              288    293                                      * Create date
D CRT_TIME              294    299                                      * Create time
D CPL_LEVEL             300    303                                      * Compiler level
D SRC_FILE              304    313                                      * Source file
D*
```

図 13. PSDS のコーディングの例

注: キーワードはラベルではないので、サブフィールドのアクセスには使用できません。短い項目は、右側をブランクで埋め込まれます。

プログラム例外およびエラー・サブルーチン

プログラム例外またはエラーが起こった時に制御を受け取るサブルーチンを識別するには、サブルーチンの BEGSR 演算命令の演算項目 1 で、*PSSR を指定します。命令コードの 73 ～ 74 桁目に標識が指定されていないか、演算命令に (E) 拡張がないか、ステートメントがエラーを処理できる MONITORブロック内にないか、あるいは命令コードに予期しない例外 (SCAN 操作での配列の指標付けエラー) が起こった場合には、制御は、プログラム例外またはエラーが起こった時点でこのサブルーチンに転送されます。さらに、このサブルーチンを EXSR 演算命令で呼び出すことができます。*PSSR は、ファイル仕様の INFSR キーワードで指定することができ、ファイル例外 / エラーが起こった場合に j を受け取ります。

プログラム例外 / エラー・サブルーチンでは、任意の命令コードが使用できます。ENDSR 演算命令は、サブルーチンの最後の仕様でなければならず、ENDSR 演算命令の演算項目 2 は、サブルーチンの実行に従って戻りポイントを指定します。詳細については、 48 ページの『ファイル例外およびエラー・サブルーチン (INFSR)』を参照してください。

プログラム例外 / エラー・サブルーチンを指定する場合には、次の点に留意してください。

- EXSR 演算命令の演算項目 2 で *PSSR を指定することによって、*PSSR サブルーチンを明示的に呼び出すことができます。
- *PSSR サブルーチンの ENDSR 演算命令が実行されると、フィールド、サブフィールド、配列エレメント、または演算項目 2 のフィールドで指定された配列エレメントがブランクにリセットされます。これによって、起こった例外またはエラーに最適なサブルーチン内に戻りポイントを指定することができます。演算項目 2 がサブルーチンの終わりにブランクを含む場合には、デフォルト・エラー処理プログラムが制御を受け取ります。サブルーチンが EXSR または CASxx 演算命令で呼び出された場合には、EXSR または ENDCS の後に続く次の命令に制御が戻されます。例外がサブプロシージャーで起こった場合には、GOTO 演算命令は ENDSR 演算命令の前には行なわれず、エラー・コード 9001 が出されてアプリケーションが終了します。演算項目 2 は、サブプロシージャー *PSSR の ENDSR 演算命令でサポートされていません。
- 非ファイル例外 / エラーが起こるたびに、プログラム例外 / エラー・サブルーチンが制御を受け取るので、サブルーチンが実行されている一方で、例外またはエラーが起こる可能性があります。サブルーチンが実行されている一方で例外 / エラーが起こる場合には、このサブルーチンが再度呼び出されます。これによって、この問題を回避するようこのサブルーチンをコーディングしない限り、プログラム・ループが起こります。
- サブプロシージャーで *PSSR を定義することができ、各サブプロシージャーは自身の *PSSR をもつことができます。サブプロシージャーの *PSSR は、そのサブプロシージャーでローカルであることに注意してください。このサブプロシージャーを同じ例外ルーチンで共用したい場合には、各 *PSSR が共用プロシージャーを呼び出さなければなりません。
- サブプロシージャー内で定義されていない *PSSR がある場合には、この *PSSR は、サブプロシージャーで例外が起きても実行されません。

コンポーネント・エラー / 例外

以下のセクションでは、イベント中のエラーの処理方法、および VisualAge RPG 例外ハンドラーによってトラップされる例外について説明してあります。

コンポーネント状況コード

以下の *STATUS 値によって、正常終了でコンポーネントを終了する方法が照会できます。

00031 LR がオンなので、コンポーネントが終了されます。ルート ENDACT に達していない時に、LR がチェックされます。ルート・アクション・サブルーチンは、ネストされたアクション・サブルーチンの最下部 (または最初) にあるサブルーチンです。

00032 コンポーネントは直接、自身を終了します。たとえば、コンポーネント 'thiscomp' は STOP 'thiscomp' を出します。コンポーネント'thiscomp' は終了されます。

00033 コンポーネントは間接的に自身を終了します。たとえば、現行コンポーネントを START したコンポーネントを終了するために、 STOP 'myparent' が現行コンポーネントによって出されます。'myparent' のすべての子が、現行コンポーネントを含め、最初に終了されます。

00034 コンポーネントが、別のコンポーネントによって直接終了されます。たとえば、現行コンポーネントを終了するために、STOP 'X' が別のコンポーネントによって出されます。

00035 コンポーネントが、別のコンポーネントによって間接的に終了されます。たとえば、現行コンポーネントの親 'myparent' を終了するために、別のコンポーネントによって STOP 'myparent' が間接的に出され、現行コンポーネントも終了されます。

正常終了が起こった時に、サブルーチン *TERMSR が呼び出されます。*TERMSR は、最終コード実行が起こったユーザー書き込みサブルーチンです。*TERMSR が呼び出された時点で、アクション・サブルーチンはアクティブでなくなり、現行コンポーネントが終了中としてマークされます。これは、ほとんどのグラフィカル・ユーザー・インターフェース操作が使用できなくなることを意味します。詳細については、次を参照してください。

- 35 ページの表 3
- 36 ページの表 4
- 37 ページの表 5
- 39 ページの表 6

イベント・エラー処理

イベントの処理中にエラーが起こった場合には、以下の 2 つのうち 1 つが起こります。

- *PSSR または INFSR が存在しない場合には、デフォルト例外ハンドラーが呼び出されます。
- エラー処理ルーチンが存在する (*PSSR または INFSR) 場合には、エラー処理ルーチンが呼び出されます。

アプリケーションにエラー処理サブルーチンが含まれている場合には、以下の命令コードの 1 つに達するまでこのサブルーチンが実行を続行します。

RETURN ENDACT *DEFAULT 処理が行なわれたのと同じ場所に制御を戻します。他にネストされたアクション・サブルーチンがない場合は、LR がチェックされます。

- LR がオンである場合には、コンポーネントは正常に終了します。 32 ページの『正常終了』を参照してください。
- LR がオンでない場合には、現行アクション・サブルーチンは終了し、イベントのためのすべてのデフォルト・アクションが実行されます。

STOP コンポーネントは正常終了します。幾つかの制約事項が適用されます。 31 ページの『第 4 章 コンポーネントの処理』を参照してください。

ENDSR 演算項目 2 での指定は、実行のフローに影響を与えます。

- 演算項目 2 が指定されていない場合には、デフォルト例外ハンドラーの照会メッセージ・ウィンドウが表示されます。
- *DEFAULT が演算項目 2 に指定されている場合には、ENDACT *DEFAULT 処理が行なわれたのと同じ場所に制御が戻されます。他にネストされたアクション・サブルーチンがない場合は、LR がチェックされます。
  - LR がオンである場合には、コンポーネントは正常に終了します。 32 ページの『正常終了』を参照してください。
  - LR がオンでない場合には、現行アクション・サブルーチンは終了し、イベントのためのすべてのデフォルト・アクションが実行されます。
- *NODEFAULT が演算項目 2 に指定されている場合には、ENDACT *NODEFAULT 処理が行なわれたのと同じ場所に制御が戻されます。 551 ページの『ENDSR (ユーザー・サブルーチンの終了)』 を参照してください。他にネストされたアクション・サブルーチンがない場合は、LR がチェックされます。
  - LR がオンである場合には、コンポーネントは正常に終了します。 32 ページの『正常終了』を参照してください。
  - LR がオンでない場合には、現行アクション・サブルーチンは終了し、イベントのためのすべてのデフォルト・アクションが実行されません。
- *ENDCOMP または *CANCL が演算項目 2 に指定されている場合には、エラーが起こった時に実行されていたアクション・サブルーチンが終了され、コンポーネントは異常終了します。 551 ページの『ENDSR (ユーザー・サブルーチンの終了)』 を参照してください。
- *ENDAPPL が演算項目 2 に指定されている場合には、エラーが起こった時に実行されていたアクション・サブルーチンが終了され、アプリケーション中のすべてのコンポーネントが階層の逆の順序でクローズされます。 551 ページの『ENDSR (ユーザー・サブルーチンの終了)』 を参照してください。エラーが起こった時にアクティブだったコンポーネントは、異常終了されます。その

他のすべてのコンポーネントは、正常終了されます。 32 ページの『正常終了』 および 34 ページの『異常終了』 を参照してください。

デフォルト例外ハンドラーがプロシージャー外で起こった例外のために呼び出されると、以下の 1 つを選択できるウィンドウが表示されます。

- デフォルト処理を行なう (ENDSR *DEFAULT 用の上記の情報が適用される)
- デフォルト処理を行なわない (ENDSR *NODEFAULT 用の上記の同じ情報が適用される)
- 演算命令を再試行する: このオプションは、入出力エラーの小さなセット用にだけ表示されます。これによって、同じ演算命令を再試行することができます。
- コンポーネントを終了する (ENDSR *ENDCOMP 用の上記の同じ情報が適用される)
- アプリケーションを終了する (ENDSR *ENDAPPL 用の上記の情報が適用される)

注: サブプロシージャー内で例外が起こり、ローカル *PSSR またはエラー標識がない場合には、アプリケーションが終了されます。

エラー処理ルーチンまたはデフォルト例外ハンドラーに制御が指定された時、エラーの原因となった現行アクション・サブルーチンはまだアクティブです。エラーの時点で有効だった同じイベント属性をまだアクセスすることができます。たとえば、MouseDown イベントの処理中は、%BUTTON イベント属性は有効です。このイベントの処理中にエラーが起こった場合には、 *PSSR で %BUTTON を参照することができます。

注: %BUTTON が、イベント属性が無効なイベントの *PSSR で参照される場合には、エラーが起こります。この種類のエラーによって、*PSSR が適切にこれを処理するようコーディングされていない場合に、アプリケーションが簡単に永久再帰状態に陥る場合があります。

複数ネストされたがネストされた場合、エラー処理ルーチンは、ENDSR *DEFAULT、*NODEFAULT、または同等のフィールド名が実行された時の最上アクション・サブルーチン呼び出しにのみ影響を与えます。たとえば、SHOWWIN WINDOW2 がアクション・サブルーチン BUTTON+CLICK+WINDOW1 の内側から実行された場合には、 BUTTON+CLICK+WINDOW1 は中断され、アクション・サブルーチン WINDOW2+CREATE+WINDOW2 が呼び出されます。この 2 番目のアクション・サブルーチンの呼び出し中にエラーが起こった場合には、*PSSR またはデフォルト例外ハンドラーは 0xC0000095invoked となります。*DEFAULT が実行された場合には、 WINDOW2+CREATE+WINDOW2 だけが終了され、SHOWWIN WINDOW2 の後の演算命令にある BUTTON1+CLICK+WINDOW1 に制御が戻されます。

例外処理

以下の例外が、VisualAge RPG 例外ハンドラーによってトラップされます。これらの例外は、PSDS の例外タイプ・フィールド (40 ～ 42) に入れられた *EX 付きの 4 バイトの 2 進数として、PSDS の例外番号フィールド (43 ～ 46) に入れられます。

アクセス違反	0xC0000005
0 除算された整数	0xC000009B
0 除算された浮動	0xC0000095
無効な演算命令の浮動	0xC0000097
無許可の命令	0xC000001C
特権命令	0xC000009D
整数オーバーフロー	0xC000009C
浮動オーバーフロー	0xC0000098
浮動アンダーフロー	0xC000009A
浮動デノーマル・オペランド	0xC0000094
浮動の不正確な結果	0xC0000096
浮動スタック検査	0xC0000099
データ・タイプのミスアライメント	0xC000009E
無効なロック順序	0xC000001D
配列境界が超過	0xC0000093

Windows 特定の例外の詳細については、オペレーティング・システムの文書を参照してください。

その他のすべての例外は、以下の方法の 1 つで処理されます。

- CALLB または CALL 中に例外が起こった場合には、状況コードが 202 または 211 に設定されます。
- CALLB または CALL 中に例外が起こらなかった場合には、例外は、以下のように状況コードにマップされます。

0 除算された整数	102
0 除算された浮動	102
浮動オーバーフロー	103
アクセス違反	222
データ・タイプのミスアライメント	222
その他のすべての例外	9999

第 6 章 サブプロシージャーおよびプロトタイプ

考えられる 3 つのターゲット・オブジェクトの 1 つが、結果としてコンパイルから生じます。結果は、使用される制御仕様キーワードによって異なります:
- NOMAIN および EXE キーワードがない時は、コンポーネントが作成されます。
- NOMAIN キーワードが指定されている時は、ユーティリティー、または NOMAIN、DLL が作成されます。この DLL には RPG サブプロシージャーしか入っていません。
- EXE キーワードが指定されている時は、RPG EXE が作成されます。このモジュールには、メイン・プロシージャーおよびサブプロシージャーが入っています。

VisualAge RPG プログラムは、1 つまたは複数のモジュールから構成されます。**プロシージャー**は、CALLP 命令コードによって呼び出すことができる任意のコードです。VisualAge RPG には、メイン・プロシージャーおよびサブプロシージャーの 2 種類のプロシージャーがあります。**メイン・プロシージャー**は、プログラムの入り口点プロシージャーとして指定できるプロシージャーであり、最初に呼び出された時に制御を受け取ります。メイン・プロシージャーは EXE の作成時にしか作成されないことに注意してください。

サブプロシージャーは、メイン・ソース・セクションの後に指定されるプロシージャーです。 (各タイプのコンパイル・ターゲットのメイン・ソース・セクションのレイアウトについては、256 ページの『定義の配置と有効範囲』 を参照してください。) サブプロシージャーは次の点でメイン・プロシージャーとは異なります:
- サブプロシージャー内で定義されている名前は、そのサブプロシージャーの外側ではアクセスできません。
- 呼び出しインターフェースはプロトタイプ化されていなければなりません。
- サブプロシージャーへの呼び出しはバインド済みプロシージャー呼び出しでなければなりません。
- P、D、および C 仕様しか使用できません。

すべてのサブプロシージャーは、メイン・ソース・セクションの定義仕様に対応するプロトタイプが入っていなければなりません。プログラムまたはプロシージャーを正しく呼び出し、呼び出し側が正しいパラメーターを渡すことを保証するために、プロトタイプはコンパイラーによって使用されます。

このセクションでは、サブプロシージャーを次の観点で説明します。
- サブプロシージャーの定義
- NOMAIN および EXE モジュール
- サブルーチンとの比較

サブプロシージャーの定義

サブプロシージャーはメイン・ソース・セクションの後で定義されます。図 14 はサブプロシージャーを表示したものであり、相違点のある部分を強調表示しています。

```
 * プロシージャー FUNCTION のプロトタイプ
 *
D FUNCTION        PR            10I 0                    1
D    TERM1                       5I 0 VALUE
D    TERM2                       5I 0 VALUE
D    TERM3                       5I 0 VALUE
 *
P Function        B                                      2
 *
 *---------------------------------------------------------------
 * このプロシージャーは、値パラメーターとして渡された 3 つの数値
 * に対して関数を実行します。
 *
 * これは、プロシージャーに対してプロシージャー・インターフェースを指定
 * する方法およびプロシージャーから値が戻される方法を図示したものです。
 *---------------------------------------------------------------
 *
D Function        PI            10I 0                    3
D    Term1                       5I 0 VALUE
D    Term2                       5I 0 VALUE
D    Term3                       5I 0 VALUE
D Result          S             10I 0                    4
C                   EVAL      Result = Term1 ** 2 * 17
C                                    + Term2      * 7    5
C                                    + Term3
C                   RETURN    Result * 45 + 23
P                 E                                      6
 *
```

図 14. サブプロシージャーの例

1. 名前、戻り値 (ある場合)、およびパラメーター (ある場合) を指定するプロトタイプ。
2. 開始プロシージャー仕様 (プロシージャー仕様の 24 桁目に B)
3. プロシージャー・インターフェース定義。これは戻り値およびパラメーターがある場合に指定します。 プロシージャー・インターフェースは対応するプロトタイプと一致していなければなりません。サブプロシージャーが値を戻さず、渡されるパラメーターがない場合には、プロシージャー・インターフェース定義は任意指定となります。
4. サブプロシージャーに必要な変数、固定情報、およびプロトタイプのその他の定義仕様。これらの定義はローカル定義です。
5. プロシージャーのタスクを実行するために必要な標準またはフリー・フォームの演算仕様。演算はローカルおよびグローバル定義の両方を参照できます。 サブプロシージャー内に組み込まれているサブルーチンはローカルです。これらはサブプロシージャーの外側では使用できません。 サブプロシージャーが値を戻す場合には、サブプロシージャーに RETURN 演算が含まれていなければなりません。
6. 終了プロシージャー仕様 (プロシージャー仕様の 24 桁目に E)

定義仕様の中に入れられることのあるプロシージャー・インターフェース定義の場合を除いて、サブプロシージャーは上記の順序でコーディングされていなければなりません。

サブプロシージャーとして次のようにコーディングすることはできません:
- 事前実行時およびコンパイル時の配列およびテーブル
- *DTAARA 定義

演算仕様が一度だけ処理され、プロシージャーは演算仕様の終わりに戻ります。詳細については、69 ページの『サブプロシージャー演算』を参照してください。

サブプロシージャーをエクスポートすることができます。すなわち、プログラム中の他のモジュール内のプロシージャーがそれを呼び出すことができます。エクスポートするように指示するためには、プロシージャー開始仕様にキーワード EXPORT を指定してください。指定しない場合は、サブプロシージャーはモジュール内からしか呼び出せません。プロシージャーは NOMAIN DLL からしかエクスポートできないことに注意してください。

プロシージャー・インターフェース定義

プロトタイプ化されたプロシージャーが呼び出し元パラメーターまたは戻り値をもつ場合には、プロシージャー・インターフェース定義をもっていなければなりません。 **プロシージャー・インターフェース定義**は、プロシージャーの定義内のプロトタイプ情報の反復です。 これは、プロシージャーの入力パラメーターを宣言したり、プロシージャーの内部定義が外部定義 (プロトタイプ) と矛盾しないことを保証するために使用されます。

ユーザーは、定義タイプ記入項目 (24 ～ 25 桁目) に PI を入れることによって、プロシージャー・インターフェースを指定します。 24 ～ 25 桁目のブランクによって指示されたパラメーター定義が、PI 仕様の直後に続かなければなりません。プロシージャー・インターフェース定義は、24 ～ 25 桁目にブランクが入っていない最初の定義仕様で終わるか、あるいは未定義の仕様によって終わります。

プロシージャー・インターフェース定義の詳細については、77 ページの『プロシージャー・インターフェース』 を参照してください。

戻り値

値を戻すプロシージャーは基本的にユーザー定義の関数であり、組み込み関数に類似しています。 サブプロシージャーの戻り値を定義するためには、次のようにしなければなりません:
1. サブプロシージャーのプロトタイプおよびプロシージャー・インターフェース定義の両方に戻り値を定義します。
2. 戻される値が入る拡張演算項目 2 フィールドに式を指定して RETURN 演算をコーディングします。

ユーザーは、プロシージャー・インターフェース仕様 (24 ～ 25 桁目に PI が指定されている定義仕様) に戻り値の長さとタイプを指定します。次のキーワードも使用できます:

DATFMT(fmt)
戻り値に、キーワードによって指定された日付形式が入ります。

DIM(N)
戻り値が、N エレメントをもつ配列となります。

LIKE(名前)
戻り値が、キーワードによって指定された項目のように定義されます。

PROCPTR
戻り値がプロシージャー・ポインターとなります。

TIMFMT(fmt)
戻り値に、キーワードによって指定された時刻形式が入ります。

値を呼び出し側に戻すためには、戻り値が入っている式を指定した RETURN 演算をコーディングしなければなりません。 拡張演算項目 2 フィールドの中の式は、EVAL の指定された式と同じ規則に従います。 実際に戻される値は EVAL 式の左辺と同じ役割をもち、RETURN 演算の拡張演算項目 2 は右辺と同じ役割をもちます。サブプロシージャーに戻り値が定義されている場合に RETURN 演算が実行されることを保証しなければなりません。そうしないと、サブプロシージャーの呼び出し側に対して例外が発行されます。

定義の有効範囲

サブプロシージャー内で定義された項目はすべてローカルです。ローカル項目がグローバル・データ項目と同じ名前で定義されている場合には、サブプロシージャー内のその名前に対する参照はローカル定義を使用します。

ただし、以下の事柄に留意してください:
- サブルーチン名およびタグ名は、EXE のメイン・プロシージャーに定義されていても、それらが定義されているプロシージャーに対してしか認識されません。
- 入力および出力仕様に指定されているすべてのフィールドはグローバルです。 サブプロシージャーが入力または出力仕様を使用する時 (たとえば、読み取り操作の処理中) は、同じ名前のローカル変数があっても、グローバル名が使用されます。

サブプロシージャー内のグローバル KLIST または PLIST を使用する時に、一部のフィールドがローカル・フィールドと同じ名前をもつことがあります。この場合には、グローバル・フィールドが使用されます。これによって、KLIST または PLIST を使用する前にそれをセットアップする時に問題が起こる場合があります。

たとえば、次のソースについて考えてみます:

```
D* メイン・プロシージャー定義
D Fld1            S              1A
D Fld2            S              1A
D*
C* 2 フィールド、Fld1 および Fld2 でグローバル・キー・フィールド・リストを定義します
C     global_kl     KLIST
C                   KFLD                    Fld1
C                   KFLD                    Fld2
C*
P* サブプロシージャー・セクション
P Subproc         B
D Fld2            S              1A
D*
C* local_kl は 1 つのグローバル kfld (fld1) と 1 つのローカル (fld2) をもっています
C*
C     local_kl      KLIST
C                   KFLD                    Fld1
```

```
C                   KFLD                    Fld2
C*
C* Fld2 がサブプロシージャー内にローカルに定義されていても、
C* グローバル KLIST が常にグローバル・フィールドを使用するので、
C* global_kl ではグローバル Fld2 が使用されます。 結果として、
C* ローカル Fld2 への割り当ては CHAIN 演算には影響しません。
C*
C                   EVAL      Fld1 = 'A'
C                   EVAL      Fld2 = 'B'
C     global_kl     SETLL     file
C*
C* ローカル KLIST は、その名前のローカル・フィールドがない時にだけ、
C* グローバル・フィールドを使用します。 local_kl はローカル Fld2 を
C* 使用するので、ローカル Fld2 への割り当てが CHAIN 演算に影響します。
C                   EVAL      Fld1 = 'A'
C                   EVAL      Fld2 = 'B'
C     local_kl      SETLL     file
 ...
P                 E
```

定義の配置および有効範囲に対するそれらの影響の詳細については、256 ページの『定義の配置と有効範囲』 を参照してください。

サブプロシージャー演算

次の 1 つが起こると、サブプロシージャーが終了します:

- RETURN 演算が処理される。
- サブプロシージャーの本体の中の最後の演算が処理される。

70 ページの図 15 はサブプロシージャーの通常の処理ステップを表示します。 71 ページの図 16 は例外 / エラー処理の順序を表示します。

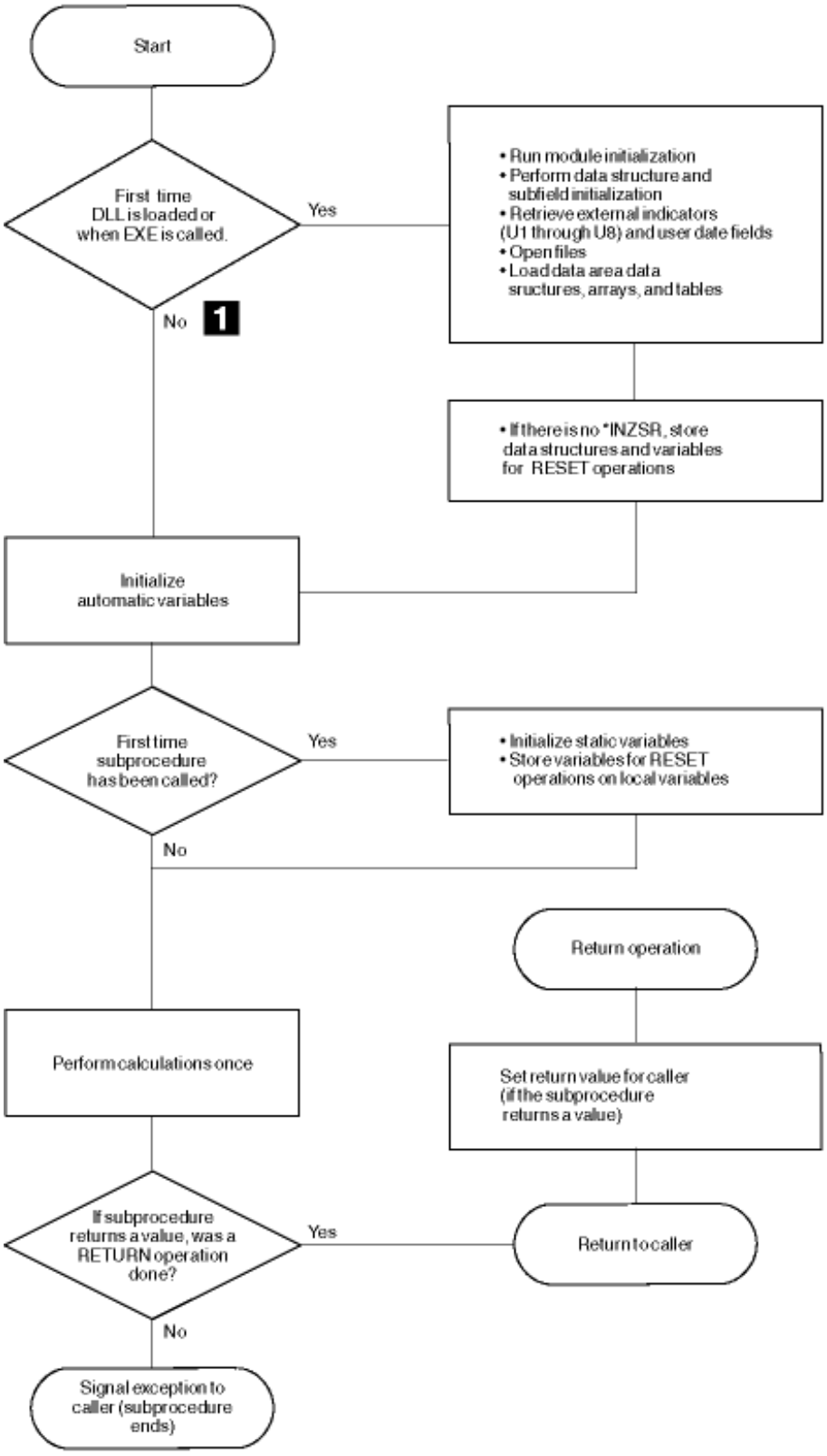


図 15. サブプロシージャーの通常の処理順序

1 [いいえ] ブランチを取るということは、プログラムが活動化されているために別のプロシージャーがすでに呼び出されていることを意味します。別のプ

ロシージャーがファイルをクローズしているか、あるいはデータ域をロック解除している可能性があるので、ファイルやデータ域などの状態について誤った前提事項を作成しないようにしなければなりません。

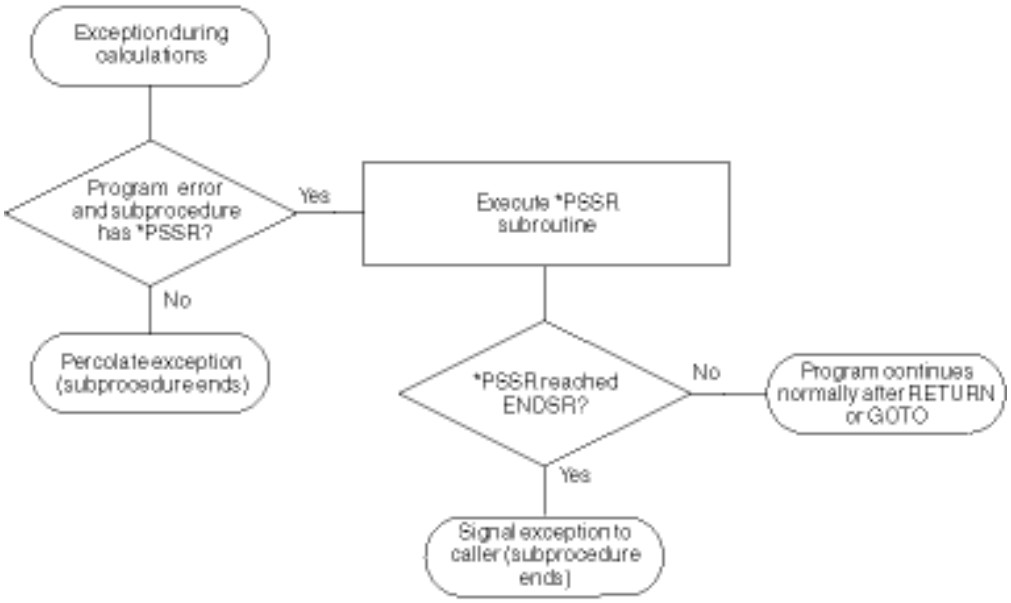


図 16. サブプロシージャーの例外 / エラー処理

サブプロシージャーをコーディングする際に考慮すべき幾つかの点を以下に揚げます:

- サブプロシージャーに対応した *INZSR はありません。 データは、サブプロシージャーが最初に呼び出される時 (ただし、演算が始まる前) に初期化されます (INZ 値またはデフォルト値のいずれかで)。
- サブプロシージャーが通常通りに戻される時は、呼び出し先のプログラムまたはプロシージャーのプロトタイプに指定されている場合は戻り値が呼び出し側に渡されます。他の何かが自動的に行なわれることはありません。すべてのファイルおよびデータ域は手作業でクローズしなければなりません。ファイルは手作業で書き出さなければなりません。EXE の場合には、LR などの標識をオンに設定できますが、EXE のメイン・プロシージャーが終了するまではプログラムの終了は行なわれません。
- サブプロシージャー内の例外処理は、サブプロシージャーにはデフォルトの例外ハンドラーがなく、したがってメイン・プロシージャーに対してデフォルトのハンドラーが呼び出される状態がサブプロシージャーの異常終了と対応するために、メイン・プロシージャーとは異なります。たとえば、サブプロシージャー内の *PSSR サブルーチンの ENDSR 演算の演算項目 2 はブランクでなければなりません。通常は、演算項目 2 をブランクにすることによって、結果的に制御がデフォルト・ハンドラーに渡されますが、サブプロシージャーでは、ENDSR に達した場合には、サブプロシージャーが異常終了します。

  *PSSR の中に RETURN 演算をコーディングするか、あるいはサブプロシージャーの中に GOTO およびラベルをコーディングして処理を続行するようにすることによって、異常終了を避けることができます。

- *PSSR エラー・サブルーチンはサブプロシージャーに対してローカルです。反対に、ファイル・エラーは定義によってグローバルとなり、そのためにサブプロシージャーで INFSR をコード化することや、INFSR がコード化されているファイルを使用することができません。

NOMAIN モジュール

アクション・サブルーチンをコーディングせずに、モジュール内で 1 つまたは複数のサブプロシージャーをコーディングすることができます。このようなモジュールは **NOMAIN モジュール**と呼ばれ、それには、制御仕様に NOMAIN キーワードを指定する必要があります。NOMAIN DLL の概念は OS/400™ サービス・プログラムの概念と類似しています。

NOMAIN DLL については、以下を考慮する必要があります。
- DLL はプロシージャーだけで構成されていなければなりません。すべてのサブルーチン (BEGSR) はプロシージャーに対してローカルでなければなりません。
- ソースでは GUI 命令コードを使用できません。これには、START、STOP、SETATR、GETATR、%SETATR、%GETATR、SHOWWIN、 CLSWIN、および READS があります。DSPLY は使用できます。しかし、それが入っているプロシージャーが VisualAge RPG DLL から呼び出された場合には、DSPLY 命令コードは何も実行しません。
- *INZSR および *TERMSR は許可されません。
- *ENTRY パラメーターは許可されません。

EXE モジュール

モジュールは、制御仕様に EXE キーワードが指定されていることを必要とするので、**EXE モジュール**と呼ばれます。

EXE モジュールは、メイン・プロシージャーとサブプロシージャーで構成されています。すべてのサブルーチン (BEGSR) はプロシージャーに対してローカルでなければなりません。EXE には、名前がソース・ファイルの名前と一致するプロシージャーが入っていなければなりません。これが EXE のメイン入り口点であり、すなわちメイン・プロシージャーとなります。

EXE モジュールについては、以下を考慮する必要があります。
- ソースでは GUI 命令コードを使用することはできません。これには、START、STOP、SETATR、GETATR、%SETATR、 %GETATR、SHOWWIN、CLSWIN および READS が含まれます。DSPLY は使用できます。
- *INZSR および *TERMSR は許可されません。
- *ENTRY パラメーターは許可されません。

  入り口点パラメーターがある場合には、それらはメイン・プロシージャーのパラメーター定義に指定され、それらは VALUE (各パラメーターごとに VALUE キーワードを指定しなければならない) によって渡されなければなりません。UCS-2 パラメーターにはできません。
- 開始 P 仕様では EXPORT キーワードは使用できません。
- メイン・プロシージャーの戻り値は、精度ゼロ (0) の 2 進数または整数として定義しなければなりません。

サブプロシージャーおよびサブルーチン

サブプロシージャーが次の改良点を提供することを除いては、サブプロシージャーとサブルーチンは類似しています:

- ユーザーはパラメーターをサブプロシージャーに渡すことができます (値で渡す場合でも)。

  すなわち、サブプロシージャーとの通信に使用されるパラメーターは必ずしも変更可能なものである必要はありません。 プログラムの場合のように、参照によって渡されるパラメーターは変更可能でなければなりません。したがって信頼度が劣る場合があります。
- サブプロシージャーに渡されるパラメーターおよびそれによって受け取られるパラメーターは、コンパイル時に整合性がチェックされます。 これは実行時エラーを減らすのに役立ちますが、経費が高くなる可能性があります。
- サブプロシージャーを式の中で組み込み関数のように使用することができます。

  この方法で使用すると、値が呼び出し側に戻されます。 これによって基本的に、式の中で必要となる演算子をカスタマイズ定義することができます。
- サブプロシージャー内で定義された名前はそのサブプロシージャーの外側では非可視となります。

  すなわち、他のプロシージャーと共用している項目をプロシージャーが不注意に変更する恐れが少なくなります。さらに、プロシージャーの呼び出し側は、サブプロシージャー内部で使用される項目についてすべてを認識する必要がありません。
- ユーザーはモジュール外からサブプロシージャーを呼び出すことができます (エクスポートされている場合)。
- ユーザーはサブプロシージャーを繰り返し呼び出すことができます。
- プロシージャーは異なる仕様タイプ、すなわちプロシージャー仕様に定義されます。この異なるタイプによって、ユーザーが別の単位を処理していることを即座に認識するために役立ちます。

それにもかかわらず、サブプロシージャーによって提供される改良を必要としない場合には、サブルーチンを使用しなければなりません。 サブルーチンの処理は、サブプロシージャーに対する呼び出しよりもさらに速くなります。

プロトタイプおよびパラメーター

プログラムおよびプロシージャーを呼び出すための推奨される方法は、プロトタイプ化された呼び出しを使用することです。プロトタイプ化された呼び出しによって、コンパイラーは実行時に呼び出しインターフェースをチェックすることができるからです。サブプロシージャーをコーディングする場合には、コンパイラーが呼び出しインターフェースとサブプロシージャーを突き合わせることができるように、プロシージャー・インターフェース定義をコーディングする必要があります。

このセクションでは、プロトタイプ、プロトタイプ化されたパラメーター、およびプロシージャー・インターフェース定義を、それぞれ定義する方法について説明します。

プロトタイプ

プロトタイプは、呼び出しインターフェースの定義です。これには、次の情報が組み込まれています。

- 呼び出しがバインド済み (プロシージャー) か動的 (プログラム) か
- プログラムまたはプロシージャー (外部名) の検索方法
- パラメーターの数とプロパティー
- 渡さなければならないパラメーターと、任意に渡すパラメーター
- プロシージャーの場合に、戻り値のデータ・タイプ (ある場合)

プロトタイプは、呼び出しを行なうプログラムまたはプロシージャーの定義仕様に組み込まれていなければなりません。プログラムまたはプロシージャーを正しく呼び出し、呼び出し側が正しいパラメーターを渡すことを保証するために、プロトタイプはコンパイラーによって使用されます。

次の規則がプロトタイプ定義に適用されます。

- 7 ～ 21 桁目にプロトタイプ名を指定しなければなりません。プロトタイプ定義にキーワード EXTPROC が指定されている場合は、プログラムまたはプロシージャーに対するすべての呼び出しはそのキーワードに指定されている外部名を使用します。いずれのキーワードも指定されていない場合には、外部名は 7 ～ 21 桁目に指定された名前 (大文字) であるプロトタイプ名になります。
- 定義タイプ項目 (24 ～ 25 桁目) に PR を指定します。すべてのパラメーター定義はその PR 仕様の直後に続かなければなりません。プロトタイプ定義は、24 ～ 25 桁目に非ブランクが指定された最初の定義仕様で終わるか、あるいは非定義仕様によって終わります。
- 次のキーワードのいずれかを、呼び出しインターフェースに適するように指定します。

  EXTPROC(名前)

  呼び出しは、キーワードによって指定された外部名を使用するバインド済みプロシージャー呼び出しとなります。

  CLTPGM(名前)

  呼び出しは、キーワードによって指定された外部名を使用する外部プログラム呼び出しとなります。

  DLL(name)

  DLL キーワードは、LINKAGE キーワードと一緒に、Windows API を含む Windows DLL の機能を呼び出すプロシージャーのプロトタイピングに使用されます。

  LINKAGE(name)

  LINKAGE キーワードは、DLL キーワードと一緒に、DLL の機能を呼び出す時に使用されるリンケージ規約 (インターフェース) を指定します。

  STATIC

  STATIC キーワードは、データ項目は静的ストレージに保管され、それにより、そのデータ項目が定義されたプロシージャーに対する呼び出しにまたがって値が保持されることを指定します。
- 戻り値 (ある場合) は PR 定義に指定します。 戻り値の長さとデータ・タイプを指定してください。さらに、戻り値として次のキーワードを指定することができます。

DATFMT(fmt)
戻り値に、キーワードによって指定された日付形式が入ります。

DIM(N)
戻り値が、N エレメントをもつ配列となります。

LIKEDS(データ構造名)
戻り値はデータ構造です。(プロシージャーを呼び出す時には、戻り値のサブフィールドを参照できます。)

LIKE(名前)
戻り値が、キーワードによって指定された項目のように定義されます。

PROCPTR
戻り値がプロシージャー・ポインターとなります。

TIMFMT(fmt)
戻り値に、キーワードによって指定された時刻形式が入ります。

VARYING
文字、グラフィック、または UCS-2 戻り値は、可変長形式になります。

これらのキーワードについては、 265 ページの『定義仕様のキーワード』 を参照してください。

図 17 は、数値入力パラメーターをとり、文字ストリングを戻すサブプロシージャー CVTCHR のプロトタイプを示したものです。戻り値に関連した名前がないことに注意してください。そのために、その内容をプログラムのデバッグ時に表示することはできません。

```
     * 戻り値は、入力パラメーター NUM の
     * 文字表現であり、左寄せされ、右側はブランクで
     * 埋め込まれます。
*
    D CVTCHR          PR            31A
    D   NUM                         30P 0   VALUE
*
     * 次の式は、CVTCHR に対する呼び出しを示します。
     * 変数 rrn の値が 431 である場合には、この EVAL の後で、
     * 変数 msg の値が「レコード 431 が見つかりません。」
     * となります。
*
    C                   EVAL      msg = 'Record '
    C                                 + %TRIMR(CVTCHR(RRN))
    C                                 + ' が見つかりません。'
```

図 17. CVTCHR のプロトタイプ

エクスポートされたサブプロシージャーのプロトタイプまたはメイン・プロシージャーのプロトタイプを書き込む場合は、/COPY ファイルにプロトタイプを入れ、呼び出し元のソース・ファイルとプロシージャーを定義するモジュールのソース・ファイルにプロトタイプをコピーしてください。このコーディング技法は、呼び出し元とプロシージャーの両方が同じプロトタイプを使用するため、その両方に最大限のパラメーター検査が提供されます。

プロトタイプ化されたパラメーター

プロトタイプ化された呼び出しインターフェースにパラメーターの引き渡しが含まれている場合には、PR仕 様の直後にパラメーターを定義しなければなりません。パラメーター定義仕様では、タイプの定義に適用される次のキーワードを使用することができます:

ASCEND
配列が昇順になります。

CCSID(number | *DFT)
グラフィックおよび UCS-2 定義の CCSID を設定します。

CLASS(*JAVA:class_name)
Java の場合だけは、オブジェクトを保管できるフィールドのオブジェクトのクラスを提供します。

DATFMT(fmt)
日付パラメーターの形式が fmt となります。

DIM(N)
パラメーターは、N エレメントをもつ配列となります。

LIKEDS(データ構造名)
パラメーターは、LIKEDS キーワードで識別されるサブフィールドと同じサブフィールドをもつデータ構造です。

LIKE(名前)
パラメーターは、キーワードによって指定された項目のように定義されます。

PROCPTR
パラメーターはプロシージャー・ポインターとなります。

TIMFMT(fmt)
時刻パラメーターの形式が fmt となります。

VARYING
文字、グラフィック、または UCS-2 戻り値は、可変長形式になります。

これらのキーワードについては、265 ページの『定義仕様のキーワード』 を参照してください。

パラメーター定義仕様では、パラメーターを渡す方法を指定する次のキーワードも使用できます。

CONST
パラメーターは読み取り専用の参照で渡されます。CONST で定義されたパラメーターは、呼び出し先プログラムまたはプロシージャーによって変更しないでください。このパラメーター引き渡し方式によって、リテラルおよび式を渡すことができます。

NOOPT
パラメーターは、呼び出し先プログラムまたはプロシージャーで最適化されません。

OPTIONS(opt1 { : opt2 { : opt3 { : opt4 } } })

ここで、opt1 ... opt4 は *OMIT、*VARSIZE、または *STRING とすることができます。たとえば、 **OPTIONS(*VARSIZE)**。

次のパラメーター引き渡しオプションを指定します。

***OMIT** この参照パラメーターの場合は特殊値 *OMIT を渡すことができます。

***VARSIZE** パラメーターに入るデータは、定義で指示されているより少ないことがあります。このキーワードが有効であるのは、文字パラメーター、グラフィック・パラメーター、または参照によって渡される配列の場合だけです。呼び出し先プログラムまたはプロシージャーには、渡されるパラメーターの長さを判別する方法が必要です。

注: 固定長フィールドの場合にこのキーワードを省略すると、パラメーターには定義で指示されているデータ量以上を入れることができます。可変長フィールドの場合は、パラメーターの長さは、定義に指示されている宣言済み最大長と同じでなければなりません。

***RIGHTADJ** CONST または VALUE パラメーターの場合は、*RIGHTADJ はグラフィック、UCS-2、または文字パラメーター値が右寄せされることを示します。

***STRING** 文字値をヌル文字で終了するストリングとして渡します。このキーワードは、値または読み取り専用の参照によって渡される基底ポインター・パラメーターの場合にのみ有効です。

VALUE

パラメーターは値によって渡されます。

上記のキーワードについては、 265 ページの『定義仕様のキーワード』 を参照してください。

プロシージャー・インターフェース

プロトタイプ化されたプロシージャーが呼び出し元パラメーターまたは戻り値をもつ場合には、プロシージャー・インターフェース定義を、メイン・ソース・セクション (メイン・プロシージャーの場合) またはサブプロシージャー・セクションに定義しなければなりません。**プロシージャー・インターフェース定義**は、プロシージャーの定義の中のプロトタイプ情報を反復します。これは、プロシージャーの入力パラメーターを宣言したり、プロシージャーの内部定義が外部定義 (プロトタイプ) と矛盾しないことを保証するために使用されます。

次の規則がプロシージャー・インターフェース定義に適用されます:

- 7 - 21 桁目に指定するプロシージャー・インターフェースの名前は任意指定です。指定する場合には、対応するプロトタイプ定義の 7 - 21 桁目に指定されている名前と一致していなければなりません。
- 定義タイプ記入項目 (24 - 25 桁目) に PI を指定します。プロシージャー・インターフェース定義は、定義仕様の中の任意の位置に指定できます。メイン・プロシージャーでは、プロシージャー・インターフェースの前に、それが参照するプ

ロトタイプがなければなりません。プロシージャーが値を戻す場合、あるいはパラメーターをもっている場合には、サブプロシージャーの中にプロシージャー・インターフェースが必要となります。そうでない場合は任意指定となります。

- 24 - 25桁目のブランクによって指示されたパラメーター定義が、PI 仕様の直後に続かなければなりません。
- プロトタイプに指定された名前と必ずしも一致していなくても、パラメーター名を指定しなければなりません。
- パラメーターのすべての属性 (データのタイプ、長さ、およびディメンションを含む) は、対応するプロトタイプ定義のものと完全に一致していなければなりません。
- パラメーターがデータ構造であることを指示するには、LIKEDS キーワードを使用して別のデータ構造と同じサブフィールドをパラメーターに定義してください。
- PI 仕様およびパラメーター仕様に指定するキーワードは、プロトタイプに指定されたものと一致していなければなりません。

モジュールの中にプロシージャーに対する呼び出しが入っている場合には、ユーザーが呼び出したい各プログラムおよびプロシージャーのプロトタイプ定義がなければなりません。必要なコーディングを最小化する 1つの方法は、共用プロトタイプを /COPY ファイルに保管することです。

プロトタイプ化されたプロシージャーを他のユーザーに提供する場合には、それらをプロトタイプ (/COPYファイルの中にある) と一緒に提供するようにしてください。

第 7 章 SQL サポート

VisualAge RPG アプリケーションに DB2® データベースをアクセスする構造化言語 (SQL) ステートメントが入っている場合には、以下のタスクを実行する必要があります。

1. DB2 をインストールして、これに対するアクセスをセットアップします。DB2 マニュアル *DB2 Universal Database Personal Edition Quick Beginnings*, S10J-8150 および *DB2 Universal Database for Windows NT Quick Beginnings*, S10J-8149 と、**Information Center** (Web サイト **http://www.ibm.com/eserver/iseries/infocenter**) にある 「データベースおよびファイル・システム」 カテゴリーの「*DB2 ユニバーサル・データベース*」 セクションには、ワークステーションおよび iSeries サーバーで DB2 製品を導入およびセットアップする方法が説明されています。
2. ソース・プログラムで SQL ステートメントをコーディングします。 80 ページの『一般構文の規則』 では、VisualAge RPG プログラムでの SQL ステートメントのコーディングする方法について説明しています。
3. アプリケーションをビルドします。GUI Designer からの「ビルド・オプション」ダイアログのオンライン・ヘルプでは、SQL ステートメント付きの VisualAge RPG プログラムで選択できるビルド・オプションについて説明しています。データベースのビルド、実行、および接続の追加情報については、 87 ページの『アプリケーションのビルド』、 88 ページの『アプリケーションの実行』、および 89 ページの『データベースとの接続』を参照してください。
4. ユーザー・アプリケーションをパッケージおよびインストールします。*VisualAge for RPG プログラミング*, SC88-5607-05では、VisualAge RPG アプリケーションをパッケージおよびインストールする方法について説明しています。

VisualAge RPG 組み込み SQL サポートは、コンパイルされる中間ファイルを作成する個別のプリコンパイラーがない点で、他のインプリメンテーションと異なります。組み込み SQL ステートメントは、ビルド処理のコンパイル・ステップ中に処理されます。

ローカル・データベース、他のワークステーション・ノード上のデータベース、または他の iSeries サーバー上のデータベースを使用するようにアプリケーションをビルドすることができます。他のこれらのシステム上でサポートされる SQL のレベルの差は、ビルドが実行されているワークステーション上でサポートされる SQL のレベルによって相補されます。ビルド時ワークステーション上の DB2 によってサポートされている構文だけが使用できます。

アプリケーションを別のワークステーションにポートする場合には、そのアプリケーションは、同じかそれ以上のレベルの DB2 でしか実行できません。

注: VisualAge RPG は、DB2/2 V1.2 で定義された機能のレベルをサポートしています。V1.2 機能だけがアプリケーションで使用されている場合には、最新のリリースの DB2/2 を使用することができます。

一般構文の規則

次の規則は、ユーザーの VisualAge RPG ソース・プログラムに組み込まれる SQL ステートメントの構文を記述しています。

1. SQL ステートメントは演算仕様書にコーディングされます。次のステートメントは例外で、すべてのコンパイル時データ (1 - 2 桁目の **)、すなわち、INCLUDE, BEGIN DECLARE, END DECLARE の前の任意の位置にコーディングすることができます。
2. SQL ステートメントの先頭を指定するには、7 - 15 桁目に **/EXEC SQL** をコーディングしてください。16 桁目はブランクでなければなりません。その行の残りの 17 から 80 桁目は SQL ステートメントとするか、あるいは SQL ステートメントの一部にすることができます。
3. SQL ステートメントの終わりを指定するには、7 - 15 桁目に **/END-EXEC** をコーディングしてください。
4. /EXEC SQL と /END-EXEC の間にコーディングできる SQL ステートメントは 1 つだけです。
5. 1 つの SQL ステートメントを複数の行にコーディングすることができます。/EXEC SQL と /END-EXEC の間の行には、7 桁目には正符号 (+) を、8 桁目にはブランクを入れなければなりません。
6. 文字リテラルは複数の行にスパンすることができます。リテラルは 1 つの行で 80 桁目までにコーディングし、次の行の 9 桁目から続行する必要があります。
7. SQL ステートメントに注釈行を指定するには、7 桁目にアスタリスク (*) をコーディングしてください。
8. SQL ステートメントと同じ行にコメントを指定するには、その SQL ステートメント内で --を使用してください。
9. SQL で始まる名前は、SQL 名と混同される場合があるので、プログラム中では使用しないでください。

次の例は、一般構文の規則を示しています。

```
----+-*--1----+----2----+----3----+----4----+----5----+----6----+
     C/EXEC SQL WHENEVER SQLERROR GO TO ERRLAB
     C/END-EXEC
     C/EXEC SQL                  -- starts SQL statement
     C+   SELECT *
      * this is a normal RPG style comment
     C+      INTO :hvar1,        -- ホスト変数 1
     C+           :hvar2         -- ホスト変数 2
     C+      FROM TABLEX
     C+      WHERE NAME='TESTING'
     C/END-EXEC
```

図 18. SQL ステートメントでの一般構文の規則

ホスト変数宣言

SQL ステートメント BEGIN DECLARE および END DECLARE が、VisualAge RPG プログラムで使用できますが、これらのステートメントは無視されます。宣言されるすべての変数は、候補ホスト変数と見なされます。

ホスト変数は、SQL ステートメント内の先行コロンによって識別されます。

ホスト変数でサポートされているデータ・タイプは、文字、可変長文字、グラフィック、整数パック 10 進数、ゾーン 10 進数、2 進数、日付、時刻、およびタイム・スタンプです。SQL データ・タイプ REAL、DOUBLE、および VARCHAR は、サポートされていません。

以下のテーブルには、VisualAge RPG データ・タイプが SQL データ・タイプにマップされる方法が要約されています。

表 11. ホスト・データ・タイプ

VARPG データ・タイプ	SQL データ・タイプ	説明	注
4 桁の 2 進数	SMALLINT	16 ビットの符号付き整数	小数点以下の桁数なし
9 桁の 2 進数	INTEGER	32 ビットの符号付き整数	小数点以下の桁数なし
パック 10 進数	DECIMAL(m,n)	デフォルトの RPG 数値データ・タイプ	
文字	CHAR(m)	固定長の文字	254 文字まで
グラフィック	GRAPHIC(m)	固定長の DBCS ストリング	127 文字まで
ゾーン 10 進数	DECIMAL(m,n)	固定小数点の数	DB2 演算命令の前または後に、RPG によってパック 10 進数に変換されます。
日付	DATE	日付	以下の様式は、DB2: *ISO、*USA、*EUR、*JIS でサポートされています。他の RPG 様式は、DB2 演算命令の前または後に、RPG によってパック 10 進数に変換されます。
時刻	TIME	時刻	以下の様式は、DB2: *ISO、*USA、*EUR、*JIS でサポートされています。他の RPG 様式は、DB2 演算命令の前または後に、RPG によってパック 10 進数に変換されます。
タイム・スタンプ	TIMESTAMP	タイム・スタンプ	

ホスト変数の規則

以下では、ホスト変数の規則について説明しています:

1. ホスト変数は、プログラムで定義されたスカラー文字、数字、日付、時刻、タイム・スタンプ、またはDBCS フィールドです。ホスト変数を以下にすることはできません:
   - 複数回繰り返しデータ構造
   - 標識フィールド名 (*INxx)
   - テーブル
   - UDATE、UDAY、UMONTH、UYEAR
2. 索引付き配列は、ホスト変数として使用できません。
3. SQL のすべての数値データ・タイプには互換性があり、ホスト変数タイプがカラム定義と一致しない時には適切な変換が行なわれます。これには、浮動小数 (FLOAT) のデータベース・カラムが含まれます。切り捨てを示すメッセージを受け取ることになります。
4. SQL のすべての文字データ・タイプには互換性があり、ホスト変数タイプがカラム定義と正確に一致しない時には、適切な変換が行なわれます。SQL は、固定長と可変長文字間で適切な変換を実行します。切り捨てを示すメッセージを受け取ることになります。
5. SQL のすべての DBCS データ・タイプには互換性があり、ホスト変数タイプがカラム定義と正確に一致しない時には、適切な変換が行なわれます。SQL は、固定長と可変長 DBCS データ間で適切な変換を実行します。切り捨てを示すメッセージを受け取ることになります。
6. SQL の日付、時刻、およびタイム・スタンプ・フィールドは文字フィールドと互換性があります。たとえば、SQL 日付カラムが文字ホスト変数で取り出されると、これは「ビルド」ノートブックの DB2 オプション・ページで指定された日付 / 時刻様式値を使用して様式設定されます。
7. 標識変数は、4 桁の 2 進数値として宣言されなければなりません。
8. サブフィールドをもたない単一オカレンス・データ構造は、正規 RPG 規則 (165 ページの『第 11 章 データ構造』を参照) に従っている文字データ・タイプとして見なされます。サブフィールドをもつデータ構造は、ホスト構造として見なされます。

ホスト変数としてのデータ構造

データ構造が SQL ステートメントのホスト変数として指定されると、この名前はデータ構造のすべてのサブフィールドを参照します。これは、ホスト変数の項目の多いリストを指定する便利な方法です。図 19には、データ構造が使用されない場合のソース・コードが図示されており、図 20には、データ構造が使用される場合のソース・コードが図示されています。

```
----+-*--1----+----2----+----3----+----4----+----5----+----6----+
     C/EXEC SQL
     C+   SELECT *
     C+      INTO :F1, :F2, :F3, :F4, :F5, :F6, :F7
     C+      FROM TABLEX
     C+      WHERE NAME='GASPARE'
     C/END-EXEC
```

図 19. データ構造を使用しないホスト変数のコーディング

データ構造の使用:

```
----+-*--1----+----2----+----3----+----4----+----5----+----6----+
     D ROW             DS
     D   F1                    1     10
     D   F2                   11     20
     D   F3                   21     30
     D   F4                   31     40
     D   F5                   41     50
     D   F6                   51     60
     D   F7                   61     70
      *
     C/EXEC SQL
     C+   SELECT *
     C+      INTO :ROW
     C+      FROM TABLEX
     C+      WHERE NAME='GASPARE'
     C/END-EXEC
```

図 20. データ構造を使用するホスト変数のコーディング

データ構造サブフィールドには他に幾つかコーディングがありますが、SQL ステートメントのホスト変数リストは、より小さいものです。プログラム内に幾つかの SQL ステートメントが存在している可能性があるので、コーディング全体の労力は少なくなります。

標識変数および構造

標識変数が必要な場合 (たとえば、ヌル・カラムによって) には、ホスト構造の名前とともに、短い 2 進数値指定するを指定することができます。

```
----+-*--1----+----2----+----3----+----4----+----5----+----6----+
     D ROW             DS
     D   F1                     1     10
     D   F2                    11     20
     D   F3                    21     30
     D   F4                    31     40
     D   F5                    41     50
     D   F6                    51     60
     D   F7                    61     70
      *
     D STRUCT          DS
     D  AI                      1     14B 0 DIM(7)
      *
     C/EXEC SQL
     C+   SELECT *
     C+      INTO :ROW:AI
     C+      FROM TABLEX
     C+      WHERE NAME='GASPARE'
     C/END-EXEC
```

図 21. 標識変数および構造

これは、各配列エレメントのコーディングの点で、標識変数と同じです。たとえば、field F1 の標識変数は AI(1)、field F2 の標識変数は AI(2) などです。

ホスト構造の規則

以下では、ホスト構造の規則について説明しています:

- 1 つの SQL ステートメントには、1 つまたは複数のホスト構造を入れることができます。
- データ構造には、ホスト構造として認識されるためのサブフィールドが含まれていなければなりません。サブフィールドのないデータ構造は、通常文字フィールドとして見なされます。
- ホスト構造名は、標識配列のすぐ前にすることができます。この配列は小数点以下の桁数がゼロの 2 進数値です。配列の各エレメントは、データ構造のサブフィールドに対応しています。

/EXEC SQL INCLUDE ステートメント

/EXEC SQL INCLUDE ステートメントは、コンパイル時間データ・セクション (1 - 2 桁にある **) より前であれば、プログラム中の任意の場所に表示することができます。ファイル名は、単一名で指定されます。ファイル拡張子は、VPG にデフォルト設定されます。

注: このファイル名は、ローカル・ファイルのみを参照します。

/EXEC SQL INCLUDE SQLCA ステートメント

SQLCA データ構造は、データベース処理が「ビルド」ノートブックの DB2 オプションで指定されている時に、自動的に VisualAge RPG プログラムに組み込まれます。データ構造は、INCLUDE SQLCA ステートメントが指定されていない場合にも組み込まれます。

SQLCA データ構造を使用して、実行後に各 SQL ステートメントの結果を照会することができます。

INCLUDE SQLCA ステートメントが指定されている場合には、データ構造の定義がプログラムのそのポイントに組み込まれます。INCLUDE SQLCA ステートメントの後続のインスタンスは、無視されます。

注: SQLCA データ構造は、/EXEC SQL INCLUDE SQLCA の代わりに /COPY コンパイラー指示を使用して組み込むこともできます。

図 22では、SQLCA データ構造のレイアウトが表示されています:

```
----+-*--1----+----2----+----3----+----4----+----5----+----6----+
SQL  D* Start of SQLCA Data Structure
SQL  D SQLCA           DS
SQL  D  SQLAID                  1      8A
SQL  D  SQLABC                  9     12B 0
SQL  D  SQLCOD                 13     16B 0
SQL  D  SQLERL                 17     18B 0
SQL  D  SQLERM                 19     88A
SQL  D  SQLERP                 19     96A
SQL  D  SQLERRD                97    120B 0 DIM(6)
SQL  D  SQLERR                 97    120A
SQL  D   SQLER1                97    100B 0
SQL  D   SQLER2               101    104B 0
SQL  D   SQLER3               105    108B 0
SQL  D   SQLER4               109    112B 0
SQL  D   SQLER5               113    116B 0
SQL  D   SQLER6               117    120B 0
SQL  D  SQLWRN                121    127A
SQL  D   SQLWN0               121    121A
SQL  D   SQLWN1               122    122A
SQL  D   SQLWN2               123    123A
SQL  D   SQLWN3               124    124A
SQL  D   SQLWN4               125    125A
SQL  D   SQLWN5               126    126A
SQL  D   SQLWN6               127    127A
SQL  D   SQLWN7               128    128A
SQL  D   SQLWN8               129    129A
SQL  D   SQLWN9               130    130A
SQL  D   SQLWNA               131    131A
SQL  D  SQLSTT                132    136A
SQL  D* End of SQLCA Data Structure
```

図 22. SQLCA データ構造のソース拡張

/EXEC SQL WHENEVER ステートメント

/EXEC SQL WHENEVER ステートメントは、SQL ステートメントの実行に続いて行なわれるエラー処理を決定するします。図 23では、/EXEC SQL WHENEVER ステートメントの構文について図示されています。

```
C/EXEC SQL WHENEVER <condition> <action>
C/END-EXEC
```

図 23. SQL WHENEVER ステートメントの構文

注: <condition> は、SQLWARNING、SQLERROR、または NOT FOUND です。<action> は、GOTO <tag-name>、GO TO <tag-name>、または CONTINUE です。

/EXEC SQL WHENEVER は、SQL ステートメントが非ゼロ戻りコードで戻される時のアクションを識別します。これは、次の /EXEC SQL WHENEVER ステートメントまでのすべての後続 SQL ステートメントに適用されます。

ステートメントが入っているコードのセクションに基づいて、アクションが適用できない時には常にメッセージが出されます。図 24 に、これを示します。

```
---+-*--1----+----2----+----3----+----4----+----5----+----6----+
 1  C           SUBR1     BEGSR
 2  C/EXEC SQL WHENEVER SQLERROR GOTO ERRLAB
 3  C/END-EXEC
 4  C           ERRLAB    TAG
 5  C                     ENDSR
 6  C           SUBR2     BEGSR
 7  C/EXEC SQL FETCH ...
 8  C/END-EXEC
 9  C                     ENDSR
```

図 24. SQL WHENEVER を使用したエラー・メッセージ

この例では、WHENEVER アクションが別のサブルーチン内のブランチの原因となるため、ステートメント 7 が無効です。この条件で使用できる値は、次のとおりです。

- SQLWARNING: SQL 戻りコードの値が 0 より大きく 100 より小さい場合に、アクションを起動します。
- SQLERROR: SQL 戻りコードの値が 0 より小さい場合に、アクションを起動します。
- NOT FOUND: SQL 戻りコードの値が 100 の場合に、アクションを起動します。

このアクションで使用できる値は、次のとおりです。

- GOTO <tag-name>: 条件が真の場合に、指定されたタグ名で実行が再開されます。
- CONTINUE: 条件が真の場合に、プログラム内の次の実行可能ステートメントで実行が再開されます。これがデフォルトのアクションです。

注: /EXEC SQL WHENEVER ステートメントは、演算仕様に表示されていなければなりません。

/EXEC SQL BEGIN DECLARE ステートメント

/EXEC SQL END DECLARE ステートメントは、コンパイラーによって無視されますが、中間のステートメントは無視されません。表 12では、SQL データ・タイプが VisualAge RPG データ・タイプにマップされる方法について説明しています。

表 12. SQL タイプのホスト変数へのマッピング

SQL データ・タイプ	VARPG データ・タイプ	データ形式 (43 桁)	長さ (バイト) (44 ～ 51 桁)	小数点以下の桁数 (52 桁)
SMALLINT	4 桁の 2 進数	B	2	0
INTEGER	9 桁の 2 進数	B	4	0
DECIMAL(m,n)	パック 10 進数	P	m/2+1	n
CHAR(m)	文字		m	
DATE	日付		10	
TIME	時刻		8	
TIMESTAMP	タイム・スタンプ		26	
GRAPHIC(m)	グラフィック	G	m*2	

実行時エラー処理

SQL ステートメントが失敗した場合には、実行時にメッセージが出されます。これらのエラーを検出するためには、SQLステートメントをコーディングするか、あるいは SQLCOD 値を明示的にチェックしなければなりません。

アプリケーションのビルド

組み込み SQL を含むアプリケーションをビルドするには、「ビルド」ノートブックで以下のオプションを指定しなければなりません:

- DB2 データベース名
- パッケージ名、あるいはバインド・ファイル名のいずれか

詳細については、 *VisualAge for RPG プログラミング*, SC88-5607-05 を参照してください。

指定したデータベースを、ワークステーションでカタログしなければなりません。データベースを使用する適切な権限をもっていなければなりません。アプリケーションのビルドを開始すると、DB2 Database Manager が自動的に開始されます (ビルド処理が DB2START コマンドを出す)。ただし、クライアント環境からアプリケーションをビルドしようとする場合には、サーバーでデータベース・マネージャーを自身で開始しなければなりません。コンパイル中にデータベースと自動的に接続するには、「ビルド・オプション」ノートブックの DB2connect ページで最初に有効なユーザー ID およびパスワードを指定しなければなりません。後続のビルドで、VARPG がこの情報を使用してデータベースと接続します。

アプリケーションを実行する前に、パッケージを作成しなければなりません。パッケージは、データベース中に保管されているオブジェクトで、この中には、アプリケーションまたはプログラム内の組み込み SQL を実行するための情報が含まれて

います。パッケージがビルド時に作成された場合には、これは、使用可能なバインディングと呼ばれます。これによって、アプリケーションはビルド中に使用されたデータベースだけをアクセスすることができます。アプリケーションが、バインディングを据え置かずにビルドされた場合には、バインド・ファイルが作成され、アプリケーションは多くのデータベースをアクセスすることができます。

アプリケーションの実行

組み込み SQL を含むアプリケーションを実行するには、以下の条件でなければなりません:

- アプリケーションがアクセスするデータベースを、ワークステーションでカタログしなければなりません。
- データベースをアクセスするのに適切な権限をもっていなければなりません。
- アプリケーションがビルドされたタイム・スタンプが、アクセスしようとしているデータベース・パッケージのタイム・スタンプと一致してしなければなりません。

VisualAge RPG アプリケーションのビルドを開始すると、DB2 Database Manager が自動的に開始されます (DB2START)。クライアント環境からアプリケーションを実行しようとする場合には、サーバーでデータベース・マネージャーを自身で開始しなければなりません。

アプリケーションが、バインディングを据え置かずにビルドされた場合には、アプリケーションを実行する前に、作成されたバインド・ファイルをデータベースにバインドしなければなりません。

アプリケーションをビルドすると、タイム・スタンプがその中に組み込まれます。このタイム・スタンプは、アプリケーションが実行される時にデータベース・パッケージと比較されます。タイム・スタンプが等しくない場合には、アプリケーションは実行されません。アプリケーションが古いパッケージに対して実行された場合に、このミスマッチが起こります。

アプリケーションを別のワークステーションにポートする場合には、アプリケーションのタイム・スタンプが、新しいワークステーション上でアクセスしようとしているパッケージのタイム・スタンプと一致していなければなりません。以下のいずれかを行なうことができます:

- コマンド・プロンプトから以下のコマンドを出して、アプリケーションを再バインドします:

  ```
  sqlbind applic1.bnd typesx
  ```

  ここで、*sqlbind* は DB2 コマンドで、*applic1.bnd* はビルド中に作成されたバインド・ファイルで、さらに *typesx* はアクセスしたいデータベースです。
- アプリケーションで使用されるデータベースへのアクセスをセットアップします。新規ワークステーションをアクセスしようとするデータベースをカタログしなければなりません。これは、ビルド中に使用されるのと同じデータベースです。データベースを別のワークステーション上、あるいはリモート・システム上にすることができます。

データベースとの接続

アプリケーションがデータベースをアクセスする前に、アプリケーションをそのデータベースと接続しなければなりません。CONNECT TO ステートメントを使用して、あるいは暗黙接続を使用して、これを行なうことができます。

CONNECT TO ステートメントの使用

アプリケーション中で CONNECT TO ステートメントを使用することによって、接続したいデータベース名を指定することができます。たとえば、次のようになります。

```
C\EXEC SQL CONNECT TO LATONA
C\END-EXEC
```

注: LATONA は、データベースの名前です。

以下の例で表示されているとおり、データベース名の変数を使用することができます:

```
D server          s             10a
D userid          s              8a
D password        s             10a
...
C                   eval      server = 'LATONA'
C                   eval      userid = 'USERID'
C                   eval      password = 'password'
...
C\EXEC SQL
C+     CONNECT TO :server IN SHARE MODE user :userid using :password
C\END-EXEC
```

CONNECT SQL ステートメントの構文の詳細については、DB2 構成の SQL 参照を参照してください。

暗黙接続の使用

暗黙的に接続したいデータベースを指示するよう、環境変数 SQLDBDFT 設定することによって、データベースとの暗黙接続を確立することができます。たとえば、次のようになります。

```
SET SQLDBDFT=LATONA
```

この環境変数は、CONFIG.SYS ファイルで、あるいはセッションのコマンド・プロンプトから設定することができます。

アプリケーションを Windows 環境で実行する場合には、以下を使用してデータベースと接続します:

```
SET DB2DBDFT=LATONA
SET DB2USERID=USERID
SET DB2PASSWORD=password
```

これらの環境変数は、AUTOEXEC.BAT ファイルで設定されます。

注: インストールされた DB2 の構成によって、環境変数名には幾つかの違いがあります。DB2 インストール・マニュアルを参照してください。

第 8 章 ファイルの考慮事項

この章では、VisualAge RPG プログラムでファイルを使用する方法について説明します。プログラムでは、DISK ファイル、PRINTER ファイル、および SPECIAL ファイルを次のように使用できます。

- DISK ファイル:
  - DISK ファイルはリモートまたはローカルのいずれでもよい
  - リモート DISK ファイルは外部記述でなければならない
  - ローカル DISK ファイルはプログラム記述でなければならない
- PRINTER ファイル:
  - 使用できる PRINTER ファイルは最大 8 つまで
  - PRINTER ファイルはプログラム記述でなければなりません。
  - PRINTER ファイルはローカルでなければならない
- SPECIAL ファイル:
  - SPECIAL ファイルはプログラム記述でなければならない
  - SPECIAL ファイルはローカルでなければならない

ファイルを指定する方法の詳細については、以下の章を参照してください。

- 237 ページの『第 17 章 ファイル仕様』
- 255 ページの『第 18 章 定義仕様』
- 299 ページの『第 19 章 入力仕様』
- 323 ページの『第 21 章 出力仕様』

ディスク・ファイル

209 ページの『第 3 部 仕様』 では、さまざまな仕様でローカル・ファイルとリモート・ファイルの両方を定義する方法について説明しています。この章には、VisualAge RPG アプリケーションでローカル・ファイルまたは OS/400 データベース・ファイルを使用する場合の追加の考慮事項について説明しています。

ローカル・ファイル

以下は、ローカル・ファイルの制約事項の要約です:

- ローカル・ファイルはロックできません。
- ローカル・ファイルの数値フィールドは、変換なしでそのまま書き込まれ、読み取られます。
- プログラムが、存在していないローカル・ファイルで入出力を実行した場合には、そのファイルが作成されます。
- ローカル・ファイルが VisualAge RPG によって作成された場合には、このファイルのレコードは、復帰改行で終了されます。復帰改行を含まないローカル・ファイルを使用した場合には、VisualAge RPG アプリケーションは、このファイルで入出力操作を実行することができなくなります。
- ローカル・ファイルのビット・パターンは、そのままストレージに読み取られます。ビット・パターンに復帰改行 (CRLF) の2 進数表記がある場合には、VisualAge RPG プログラムによって読み取られるレコードは 2 つのレコードに分割されます。

- ローカル・ファイルに2 進数が含まれる場合には、その数はバイト・スワップです。

OS/400 ファイル

VisualAge RPG 命令コードは、OS/400 物理データベース・ファイル、ソース物理データベース・ファイル、および論理データベース・ファイルをアクセスできます。OS/400 データベース・ファイルのセットアップ方法の詳細については、Web サイト **http://www.ibm.com/eserver/iseries/infocenter** の **Information Center** にある「データベースおよびファイル・システム」 カテゴリーの「*DB2 ユニバーサル・データベース*」 セクションを参照してください。

アプリケーションが OS/400 データベース・ファイルをアクセスする前に、サーバーをセットアップする必要があります。サーバーをセットアップする方法の詳細については、*WebSphere Development Studio Client for iSeries ご使用に際して*, SD88-5054-03 および *VisualAge for RPG プログラミング*, SC88-5607-05 を参照してください。

VisualAge RPG アプリケーションが OS/400 データベース・ファイルをアクセスする時には、DDM サービス・ジョブがデータベース入出力要求の処理に使用されます。単一 DDM サービス・ジョブは、VisualAge RPG アプリケーションごとにファイルがオープンされる各個別の iSeries サーバーで使用されます。アプリケーションが終了すると、すべての DDM サービス・ジョブは終了されます。

ファイルのオープン・データ・パスの共用

オープン・データ・パスの共用は、サポートされません。VisualAge RPG アプリケーションに OPNQRYF CL コマンドが含まれている場合には、OPNQRYF と関連したオープン・データ・パスは、VisualAge RPG アプリケーションでオープンされたファイルで共用することができません。

VisualAge RPG アプリケーションが、OPNQRYF コマンドを使用するプログラムを呼び出す場合には、オープン・データ・パスを共用することができます。

VisualAge RPG アプリケーションによって実行される各オープンは、固有のオープン・データ・パスを作成します。同じ VisualAge RPG アプリケーション内で同じデータベース・ファイルを複数オープンすると、オープン・データ・パスごとにファイルのインスタンスが異なることになります。

VisualAge RPG プログラムで異なるファイル別名を使用して同じデータベース・ファイルを複数回オープンし、同じ実際の OS/400 データベース・ファイルを参照することができます。「iSeries 情報の定義」ノートブックで、複数のファイル・ページを定義する必要があります。

また、VisualAge RPG アプリケーションの異なるコンポーネントで同じデータベース・ファイルをオープンすることができます。

ファイルの照会および単一 / ブロック化レコード入出力操作

ファイルの照会 (OPNQRYF コマンド) を使用する時には、VisualAge RPG アプリケーションおよび照会ファイルの両方を設定すると、単一レコード入出力操作またはブロック化レコード入出力操作のいずれかが予期されます。これらの設定値が一

致しない場合は、アプリケーションが表示されて入力時のレコードはスキップされます。この不一致は、OPNQRYF コマンドが ファイルを最初にオープンしてから、オープン・データ・パスが VisualAge RPG アプリケーション・ファイル・オープン要求と共用されて、オープン設定値の一部が無視されるために起こる可能性があります。ただし、単一レコード設定またはブロック化レコード設定はコンパイル時の VisualAge RPG アプリケーションで固定されます。

OPNQRYF コマンドおよび OVRDBF コマンドの場合は、SEQONLY(*NO/*YES) キーワードが単一レコード入出力処理またはブロック化レコード入出力処理を判別します。VisualAge RPG アプリケーションの場合は、単一レコード処理またはブロック化レコード処理は以下のいずれかに基づきます。

- F 仕様キーワード BLOCK(*NO/*YES) 値
- プログラムのファイル上にあるランダム・レコード位置決め操作の表示 (たとえば SETLL)。

照会ファイル上の無効データ・エラー

コンパイル時ファイルのキー・フィールド定義と一致しないキー・フィールド定義がある照会ファイル (OPNQRYF コマンド) を使用する場合は、実行時エラーの起こる可能性があります。照会ファイルからの単一レコード入力操作では、操作で戻されるキー・フィードバック値でサーバーからワークステーションへの変換が実行されます。戻されたキー値の形式がコンパイル時ファイルのキー定義から予期される形式と一致しない場合は、これは無効データ・エラーを生成している可能性があります。このエラー状態は、以下のいずれかを実行すると防ぐことができます。

- ブロック化レコード入力操作だけを使用する。
- プログラム実行時に使用された照会ファイルと一致するキー・フィールド定義のあるコンパイル時ファイルを選択する。

組み込みデータベース・ファイル指定変更のあるアプリケーション

VARPG アプリケーションが OVRDBF (データベース・ファイルの指定変更) コマンドを使用して別のライブラリーまたはファイル名を指定する場合は、元の名前のファイルがサーバー上に存在しないとファイル・オープンが失敗します。

VARPG ファイル・オープン要求は iSeries DDM サポートを介して行われ、ファイル指定変更情報を適用する前にファイルを位置指定およびロックしようとします。このロックが元のライブラリー / ファイル名で失敗すると、オープン要求が失敗します。ファイル・ロックが正常に実行されると、その要求は iSeries データベース層に渡され、これはファイル指定変更に適用されて、リダイレクトされたファイル名でオープンを実行します。

OS/400 ファイル・データ変換

iSeries サーバー上でのデータ保管は、ワークステーション上での保管とは異なります。このために、VisualAge RPG アプリケーションが OS/400 プログラムの呼び出し、OS/400 データ域のアクセス、または OS/400 データベース・ファイルのアクセスを実行する場合に、VisualAge RPG データ変換が起こります。この変換により、データはワークステーションと iSeries サーバーの両方で正しく表示されることになります。

データ構造が OS/400 プログラムを呼び出すパラメーターとして渡される場合、あるいはデータ構造が OS/400 データ域を表示する場合には、各サブフィールドはそのデータ・タイプに基づいて変換されます。

注: データ変換を実行するには、すべてのデータベース・ファイルおよび TO/FROM ファイルを外部記述しなければなりません。

VisualAge RPG データ・タイプで以下の変換が実行されます。

文字データ:

文字データが、EBCDIC から ASCII (あるいはその逆) に変換されます。変換を正しく実行するには、データベースに保管されたすべての文字データに、適切な EBCDIC CCSID でタグを付ける必要があります。文字データは、文字フィールドの CCSID、および ASCII 端末のコード・ページに基づいて変換されます。アプリケーションが OS/400 プログラムを呼び出すか、あるいは OS/400 データ域をアクセスする場合は、その呼び出しまたはデータ域要求を提供するジョブの CCSID がデータの変換時に使用されます。

文字データのタグ付けの詳細については、Web サイト **http://www.ibm.com/eserver/iseries/infocenter** の **Information Center** にある「データベースおよびファイル・システム」 カテゴリーの「*DB2 ユニバーサル・データベース*」 セクションを参照してください。

グラフィック・データ・タイプ:

グラフィック・データ・タイプには、SO と SI 文字を除くすべての DBCS 文字が含まれます。これはシステムが予期している様式なので、グラフィック・データ・タイプ全体を使用することができます。

注: アプリケーションがサーバー入出力要求を出す場合に、VisualAge RPG 変換を正しく実行するには、OS/400 グラフィック・フィールドに適切な DBCS CCSID でタグを付ける必要があります。

日付、時刻、およびタイム・スタンプ・データ・タイプ:

変換は、文字データ・タイプと同様です。明示的 CCSID タグ付けは、OS/400 データベース・アクセスでは必要ありません。

ゾーン数値データ・タイプ:

ゾーン数値は、サーバーとワークステーションの両方で表示および印刷できるよう変換されます。

負の EBCDIC ゾーン数値は ASCII に変換され、最後のバイトの符号部分は x'7' に変換されます。 135 ページの『ゾーン 10 進数形式』を参照してください。

パック 10 進数データ・タイプ:

EBCDIC パック数値は ASCII に変換され、パック数値の符号部分は、その数が正の場合には x'C' に、負の場合にはx'D' に変換されます。 133 ページの『パック 10 進数形式』を参照してください。

2 進データ・タイプ:

2 進数フィールドは、このデータがサーバーとワークステーションの間で送信された時に再配列されます。

データベース・タイプにおいて以下の変換が実行されます。

浮動データ・タイプ:

2 進数フィールドおよび浮動フィールドは、このデータがサーバーとワークステーションの間で送信されるとバイト・スワップされます。たとえば、次のようになります。

- サーバーの 2 バイト整数フィールドに入っている '1 2' は、ワークステーションでは '2 1' に変換されます。
- ワークステーションの 2 バイト整数フィールドに入っている '2 1' は、サーバーでは '1 2' に変換されます。
- サーバーの 4 バイト整数フィールドに入っている '1 2 3 4' は、ワークステーションでは '4 3 2 1' に変換されます。
- ワークステーションの 4 バイト整数フィールドに入っている '4 3 2 1' は、サーバーでは '1 2 3 4' に変換されます。

CCSID 65535 のタグを付けられた 16 進データ・タイプおよび文字データ

文字フィールドが CCSID 65535 (変換が行なわれないことを暗黙指定) のタグを付けられている場合には、データ変換は行なわれません。

J、O、または E データ・タイプ:

J (DBCS のみ)、O (混合データ)、および E (すべて単一バイトか、すべて 2 バイトのどちらか) データ・タイプが、VisualAge RPG プログラムの文字フィールドとして扱われます。

J、O、および E データ・タイプは、適切な CCSID のタグが付いていなければなりません。ただし、これらは DBCS 文字が入る可能性があるので特別です。サーバーでは、これらのデータ・タイプの DBCS 文字は、SO (シフトアウト) 文字および SI (シフトイン) 文字で囲まれます。ワークステーションでは、DBCS は SO と SI 文字で囲まれていません。

サーバーからデータが検索されると、これらの文字はストリップされ、文字フィールドは 2 つの追加末尾ブランクで埋め込まれます。

サーバーにデータが送信されると、OS/400 DBCS フィールドに追加される適切な数の SO 文字および SI 文字によって置換できるように、文字フィールドには十分な末尾ブランクがあることを確認する必要があります。

たとえば、O (混合データ) フィールドは、データベースで作成されることを前提としています。ワークステーションでは、データベースに書き込まれる前にこのフィールドには以下が入っています。

```
        sbDBsbDBblblblbl
ここで sb = 単一バイト文字。
       DB = 2 バイト文字。
       bl = 単一バイト・ブランク文字。
```

この例では、このフィールドは以下として変換されます:

```
sbSODBSIsbSODBSI
```

注: 末尾の単一バイト・ブランクは無意味と見なされ、SO および SI 文字で適切に置換されます。

可変長フィールド:

サーバー入出力要求の場合は、このデータがサーバーとワークステーションの間で送信されると 2 進数部分が再配列されます。文字部分は、フィールド CCSID に基づいて変換されます。

OS/400 データベース・ファイル・コミットメント制御

OS/400 コミットメント制御により、データベース変更のグループを 1 単位として処理できます。この単位は、コミット操作を出すことによって、データベースに正常に適用することができます。この単位と関連した変更は、グループとして正常に適用できない場合にはロールバックすることができます。

「iSeries 情報の定義」ノートブックで入力する情報により、VisualAge RPG アプリケーションの複数の OS/400 データベース・ファイルを定義できます。これらのファイルを、複数のサーバーに置くことができます。サーバー情報の定義に関する詳細については、オンライン・ヘルプおよび *VisualAge for RPG プログラミング*, SC88-5607-05 を参照してください。

コミットメント制御環境は、1 つのサーバーだけに開始できます。別のサーバーでは、まだデータベース・ファイルを使用できます。ただし、それらのファイルは、コミットメント制御ではオープンできません。

サーバーにコミットメント制御環境を開始した後で、そのサーバー上でデータベース・ファイルをオープンするには、コミットメント制御下でオープンしたいファイルのファイル仕様で COMMIT キーワードを使用します。コミットメント制御下でオープンされたファイルだけが、後続の COMMIT 演算命令および ROLLBK 演算命令による影響を受けます。COMMIT キーワードの詳細については、245 ページの『COMMIT{(rpg_name)}』 を参照してください。

トランザクションと関連した適切な変更を行なった後で、COMMIT 命令コードを使用してその変更をデータベースにコミットしたり、あるいは ROLLBK 命令コードを使用してデータベースの変更をロールバックすることができます。

VisualAge RPG アプリケーションが終了されると、コミットメント制御は終了されます。データベース上で明示的にコミットまたはロールバックされていない変更が保留中である場合は、アプリケーションの終了時に暗黙ロールバック操作が行なわれます。

コミットメント制御下でファイルをオープンする決定が実行時に行なわれるよう、プログラムを書き込むことができます。ファイル仕様の COMMIT キーワードには、条件付きコミットメント制御を指定できるパラメーターがあります。実行時にコミットメント制御でファイルをオープンする COMMIT キーワードの使用の詳細については、245 ページの『COMMIT{(rpg_name)}』を参照してください。

レベル検査: VisualAge RPG は、使用する VisualAge RPG プログラムとデータベース・ファイルの間のレベル検査をサポートします。

VisualAge RPG コンパイラーはレベル検査に必要な情報を提供します。レベル検査は、データベース・ファイルの作成または変更時に LVLCHK(*NO) を指定しない限り、ファイルをオープンするとレコード様式基準で行なわれます。

注: レベル検査が行なわれると、これは入出力エラーとして処理されます。

浮動小数点: 浮動小数点フィールドは、サポートされていません。浮動小数点フィールドをもつ外部記述 OS/400 ファイルを処理する場合には、VisualAge RPG アプリケーションによってこの浮動小数点フィールドをアクセスすることはできません。新規レコードを作成すると、このレコードの浮動小数点フィールドは 値ゼロで

す。既存のレコードを更新する時には、浮動小数点フィールドは変更されません。キー・フィールドとして浮動小数点フィールドを使用することはできません。

ファイルのロック: OS/400 システムでは、ジョブの実行中に使用されるファイルをロック状態（排他、排他読み取り許可、共用更新、共用非更新、または共用読み取り）にすることができます。ファイル・ロック状態ではジョブ内のプログラムは影響を受けません。ファイル・ロック状態が適用されるのは、別のジョブのプログラムが同時にファイルを使用しようとした時だけです。ファイル・ロック状態は、CL コマンド ALCOBJ (オブジェクト割り振り) で割り振ることができます。リソースおよびロック状態の割り振りの詳細については、Web サイト **http://www.ibm.com/eserver/iseries/infocenter** の **Information Center** にある「*プログラミング*」 カテゴリーの「*CL および API*」 セクションを参照してください。

OS/400 システムは、ファイルのオープン時にデータベース・ファイルを次のロック状態にします:
- INPUT 用にオープン: 共用読み取りロック状態
- UPDATE 用にオープン: 共用更新ロック状態
- ADD 用にオープン: 共用更新ロック状態
- OUTPUT 用にオープン: 共用更新ロック状態

レコードのロック: 更新用にオープンされているファイルからレコードを読み取ると、このレコードにロックが適用されます。サーバー上の他のプログラムと、同じファイルの VisualAge RPG アプリケーションにある他のオープン・インスタンスは、このレコード・ロックが解除されるまで、このレコードを更新用に読み取ることができません。

ファイルが更新ファイルであっても、命令コード名の後ろにある命令コード・フィールドで (N) 命令コード拡張を使用することによって、入力するためにレコードを読み取ることができます。以下の命令コードによって、命令コード拡張 (N) が指定されていない場合には、レコードがロックされます:
- CHAIN
- READ
- READE
- READP
- READPE

次のいずれかが起こるまで、ロックは継続します。
- レコードが更新される
- レコードが削除される
- 別のレコードがファイルから読み取られる (入力または更新のため)
- ファイルに SETLL または SETGT 演算命令が実行される
- ファイルに UNLOCK 演算命令が実行される
- 出力仕様によって定義された、フィールド名が組み込まれていない出力演算命令がファイルに実行される

  注: レコードをファイルに追加する出力演算命令では、レコード・ロックは解放されません。

プログラムが更新用のレコードを読み取ろうとして、そのレコードが他のファイル・オープン・インスタンスによってすでにロックされている場合には、そのレコードがアンロックされるまで、読み取り操作は待機します。待機時間がファイルの WAITRCD パラメーターを超えた場合には、例外が起こります。プログラムがこの例外 (RNQ1218) を処理しない場合には、デフォルト・エラー処理プログラムが制御します。この操作を再試行するオプションがあります。これによって、レコード・ロック・タイムアウトが起こらなかったかのようにプログラムを続行することができます。

注: デフォルト・エラー処理プログラムが制御する前、入出力操作がファイルで実行される時に指定される INFSR サブルーチンがそのファイルにある場合には、再試行される入力操作が順次操作の場合に予期しない結果となる場合があります。ファイル・カーソルが変更されている場合には、これが起こります。

印刷装置ファイル

以下の規則が印刷装置ファイルに適用されます:

- VisualAge RPG アプリケーションの印刷装置のレコード長は、物理ページ幅より小か等しくなければなりません。
- 現在行の値が FORMLEN(n) キーワードによって指定された値と等しい時には、自動用紙送りが挿入されます。ここで nは、ページ当りの行数で、FORMLEN が指定されていない場合には、デフォルトのページ長は 66 行となります。
- 印刷が印刷装置ファイルのページの途中で終了した場合には、用紙送りが自動的に追加されます。

以下の制約事項が印刷装置ファイルに適用先されます:

- PRINT フィードバック域 (ページと行の数) は、POST 操作の後でしか更新されません。
- メジャー / マイナー戻りコード、入出力フィードバック域、およびオープン・フィードバック域は、サポートされていません。
- オーバーフロー / 取り出しは、サポートされていません。ただし、行番号が FORMLEN で指定された値 (またはデフォルトの 66) に至ると、用紙送りが実行されます。
- オーバーフローがサポートされていないので、*OFL ルーチン (ファイル例外およびエラー用) はサポートされません。これは、*ROUTINE はファイル・フィードバック域のこの値で更新が行なわれないことを意味しています。

特殊ファイル

特殊ファイルで、VisualAge RPG 操作で直接サポートされていない入出力装置を指定することができます。ファイルの入出力操作は、ユーザー作成ルーチンによって制御されます。ファイル仕様で PROCNAME キーワードを使用して、特殊ファイル・ハンドラーを定義してください。

注: ユーザー作成ルーチンは、ダイナミック・リンク・ライブラリー (DLL) に含まれている機能です。

100 ページの図 25では、**fspecr** は、コンパイルされて VisualAge RPG アプリケーションにリンクされた C モジュール内の機能です。以下の例では、ワークステーション上のローカル入出力を実行する方法について示してあります。

注: VisualAge RPG プログラムを DLL と接続するには、VisualAge RPG プログラムのプロシージャー・キーワード (PROCNAME) によって参照される C 機能定義を含む DLL を作成しなければなりません。VisualAge RPG アプリケーションをビルドする時に、「ビルド」ノートブックでアプリケーションが呼び出すすべての機能を含むライブラリー (LIB) またはオブジェクト (OBJ)、あるいはその両方のリストを指定しなければなりません。

```
F* ------------------------------------------------------------
F* ---------- ---   VRPG Special Files  --- ----------------------
F* ------------------------------------------------------------
F* Special file declaration. Explicit open and close.
F dino     cf   f   18        SPECIAL  USROPN PLIST(dinoplist)
F                                       PROCNAME('fspecr')
F                                       INFDS(info)
D* ------------------------------------------------------------
D* The INFDS- positions 38 to 42.
D info            DS
d  errinfo               38     42s 0
D* ------------------------------------------------------------
D* Used to display set indicators and read values.
D BoxId           M                   STYLE(*INFO)
D                                     BUTTON(*OK:*ENTER)
D                                     BUTTON(*ABORT:*IGNORE)
D* ------------------------------------------------------------
D* Input field associated with special file.
D fieldc          S             18              inz ('VRPGisGreat')
D* ------------------------------------------------------------
D* Extra, user-defined parameter for special file I/O operations.
D mySFparm        S             10              inz ('VRPGisGreat')
I* ------------------------------------------------------------
I* Input specification, i.e. fieldc is updated for the VRPG program
I* after each I/O operation performed on the special file dino.
I dino     NS
I                                  1   18  fieldc

I*
```

図 25. 特殊ファイルの使用 (1/2)

```
I* ------------------------------------------------------------
C     *INZSR        BEGSR
C* ------------------------------------------------------------
C     dinoplist     PLIST
C                   PARM                    mySFparm
C                   EVAL                    fieldc = 'FIRSTINIT'
C     FIELDC        DSPLY     BoxId         Reply              9 0
C*------------------------------------------------------------------------
C                   OPEN      dino                                  90
C   90'IND90'       DSPLY     BoxId         Reply
C   99'IND99'       DSPLY     BoxId         Reply
c*------------------------------------------------------------------------
C  N90              READ      dino                                  9099
C*------------------------------------------------------------------------
C  N90FIELDC        DSPLY     BoxId         Reply
C   90'IND90'       DSPLY     BoxId         Reply
C   99'IND99'       DSPLY     BoxId         Reply
c*------------------------------------------------------------------------
C                   CLOSE     dino                                  90
C*------------------------------------------------------------------------
C   90'IND90'       DSPLY     BoxId         Reply
C   99'IND99'       DSPLY     BoxId         Reply
C                   ENDSR
```

図 25. 特殊ファイルの使用 (2/2)

```
#include  <stdlib.h>
#include  <memory.h>
#include  <stdio.h>
/* ------------------------------------------------------------,
|                   Special File Function                     |
'------------------------------------------------------------*/
 *
extern void fspecr ( char *option,       // VRPG provided
                     char *status,       // VRPG provided
                     char *error,        // VRPG provided
                     char *area,         // VRPG provided
                     char *mySFparm)     // User provided
{
/* ------------------------------------------------------------,
|                   Local Constants                           |
'------------------------------------------------------------*/
#define REC_SIZE 18              // Size of record to be read.
#define NORMAL_STATUS '0'        // Values for 'status' parameter.
#define ERROR_STATUS  '2'
#define EOF_STATUS    '1'
#define OPEN_ERROR    12345      // Update values for 'error' also
#define READ_ERROR    88888      // used for the *RECORD field in
#define OPTION_ERROR  99999      // in the INFDS.
 *
   static FILE *fp ;
 *
   int radix = 10 ;              // Required for the '_itoa' function.
 *
   int a     = 0 ;
   char temp[6];
   switch  (option[0]) {
     case 'O' :                        // Locally open the file.
       if ((fp=fopen("special.dat", "rb+")) == NULL) {
          a = OPEN_ERROR ;             // ASCII value of an open error
          _itoa( a, temp, radix ) ;    // is set here for the INFDS.
          memcpy( error, temp, 5 ) ;
          status[0] = ERROR_STATUS ; // Return status.
       }
       else {
          status[0] = NORMAL_STATUS ;
       }
 *
       break ;
 *
     case 'C' :                        // Local close ...
       fclose(fp) ;
       status[0] = NORMAL_STATUS ;
       break ;
```

図 26. 特殊ファイル用 C プログラム (1/2)

```
 *
     case 'R' :                       // Local file open... read.
       fread(area, 1, REC_SIZE, fp) ;
       if (feof(fp)) {
         status[0]  = EOF_STATUS ;    // File read and EOF reached.
       }
       else if (ferror(fp)) {         // Check for any errors.
          a = READ_ERROR ;            // ASCII equivalent of a read error
          _itoa( a, temp, radix ) ;   // is set here for the INFDS.
          memcpy( error, temp, 5 ) ;
          status[0] = ERROR_STATUS ; // Return status.
       }
       else {
         status[0] = NORMAL_STATUS ;
       }
       break ;
 *
      default :
          a = OPTION_ERROR ;          // Set the ASCII equivalent of an
          _itoa( a, temp, radix ) ;   // option error for the INFDS.
          memcpy( error, temp, 5 ) ;
          status[0] = ERROR_STATUS ; // Return status.
          break ;
 *
      }
 *
      return ;
 *
#undef REC_SIZE
#undef NORMAL_STATUS
#undef ERROR_STATUS
#undef EOF_STATUS
#undef OPEN_ERROR
#undef READ_ERROR
#undef OPTION_ERROR
}
```

図 26. 特殊ファイル用 C プログラム (2/2)

図 27の例では、ファイル仕様から USROPN キーワードを除外することによって暗黙的に OPEN およびCLOSE 演算命令を実行する方法が示されています。

```
F* ------------------------------------------------------------------
F* ---------- ---   VRPG Special Files  --- ----------------------
F* ------------------------------------------------------------------
F* Special file declaration. Implicit open and close.
F dino     cf   f   18        SPECIAL  PLIST(dinoplist)
F                                       PROCNAME('fspecr')
F                                       INFDS(info)
D* ------------------------------------------------------------------
D* The INFDS- positions 38 to 42.
D info            DS
d  errinfo               38     42s 0
D* ------------------------------------------------------------------
D* Used to display set indicators and field values.
D BoxId           M                   STYLE(*INFO)
D                                     BUTTON(*OK:*ENTER)
D                                     BUTTON(*ABORT:*IGNORE)
D* ------------------------------------------------------------------
D* Input field associated with special file.
D fieldc          S             18            inz ('VRPGisGreat')
D* ------------------------------------------------------------------
D* Extra, user-defined parameter for special file I/O operations.
D mySFparm        S             10            inz ('VRPGisGreat')
I* ------------------------------------------------------------------
I* Input specification, i.e. fieldc is updated for the VRPG program
I* after each I/O operation performed on the special file dino.
I dino     NS
I                                   1   18  fieldc
I* ------------------------------------------------------------------
C     *INZSR        BEGSR
C* ------------------------------------------------------------------
C     dinoplist     PLIST
C                   PARM                    mySFparm
C                   EVAL                    fieldc = 'FIRSTINIT'
C     FIELDC        DSPLY     BoxId         Reply             9 0
C*-------------------------------------------------------------------------
C                   READ      dino                                  9099
C*-------------------------------------------------------------------------
C  N90FIELDC        DSPLY     BoxId         Reply
C   90'IND90'       DSPLY     BoxId         Reply
C   99'IND99'       DSPLY     BoxId         Reply
C*-------------------------------------------------------------------------
C                   ENDSR
```

図 27. 暗黙的な特殊ファイルのオープンおよびクローズ

第 2 部 Data

この項では、プログラムでのデータ使用に関する情報を提供します。

- 107 ページの『第 9 章 データ・タイプおよびデータ形式』 はデータ・タイプおよび形式を説明します。
- 157 ページの『リテラル』 はリテラルを説明します。
- 165 ページの『第 11 章 データ構造』 はデータ構造を説明します。
- 175 ページの『第 12 章 配列およびテーブルの使用』 は配列およびテーブルを説明します。
- 193 ページの『第 13 章 数値フィールドの編集』 数値フィールドの編集方法を説明します。
- 207 ページの『第 14 章 データの初期化』 はデータ初期化を説明します。

第 9 章 データ・タイプおよびデータ形式

このセクションでは、VisualAge RPG によってサポートされるデータ・タイプおよびその特殊なプロパティーについて説明します。サポートされるデータ・タイプは次の通りです。
- 基底ポインター
- 文字
- 日付
- グラフィック
- 数値
- オブジェクト
- プロシージャー・ポインター
- 時刻
- タイム・スタンプ
- UCS-2

さらに、一部のデータ・タイプでは異なるデータ形式を使用することができます。このセクションでは、内部データ形式と外部データ形式との間の相違、各形式、およびその指定方法について説明します。

内部形式と外部形式

数値、日付、およびタイム・スタンプは、外部形式とは独立した内部形式をもっています。 **内部形式**は、データがプログラムに保管される方法です。 **外部形式**は、データがファイルに保管される方法です。

次の場合には、内部形式に注意する必要があります。
- 参照によりパラメーターを渡す
- データ構造中のサブフィールドをオーバーレイする

さらに、算術演算の実行時パフォーマンスが重要である時には、数値フィールドの内部形式について考慮する必要があります。詳細については、 354 ページの『パフォーマンスの考慮』 を参照してください。

数値および日付 / 時刻のデータ・タイプには、デフォルトの内部形式と外部形式があります。定義仕様で特定フィールドの内部形式を指定することができます。同様に、対応する入力仕様または出力仕様でプログラム記述フィールドの外部形式を指定することができます。

外部記述ファイル中のフィールドの場合には、外部データ形式は、データ記述仕様の 35 桁目に指定します。外部記述フィールドの外部形式は変更することができませんが、1 つの例外があります。制御仕様に EXTBININT を指定した場合には、小数点以下の桁数がゼロの 2 進数フィールドは、整数外部形式をもつものとして扱われます。

外部記述データ構造中のサブフィールドの場合、外部記述に指定されたデータ形式がコンパイラーによってサブフィールドの内部形式として使用されます。

内部形式

数値独立フィールドのデフォルトの内部形式はパック 10 進数です。数値データ構造サブフィールドのデフォルトの内部形式はゾーン 10 進数です。異なる内部形式を指定するには、必要な形式をフィールドまたはサブフィールドの定義仕様の 40 桁目に指定してください。

日付、時刻、およびタイム・スタンプのデフォルトの形式は *ISO です。一般的に、特に混合した外部形式タイプがある場合には、デフォルトの ISO 内部形式を使用するようにお勧めします。

日付、時刻、およびタイム・スタンプのフィールドの場合には、必要であれば、制御仕様で DATFMT キーワードおよび TIMFMTキーワードを使用し、プログラム中の すべての 日付 / 時刻フィールドのデフォルトの内部形式を変更することができます。定義仕様で DATFMT キーワードまたは TIMFMT キーワードを使用し、*個別の* 日付 / 時刻フィールドのデフォルトの内部形式を指定変更することができます。

外部形式

プログラム記述ファイルに数値、文字、または日付 / 時刻フィールドがある場合には、その外部形式を指定することができます。有効な外部数値形式は 2 進数、整数、パック 10 進数、ゾーン 10 進数、符号なし、または浮動です。外部形式は、フィールドの処理方法に影響しません。しかし、指定された内部形式によっては、算術演算のパフォーマンスを改善できる場合があります。詳細については、 354 ページの『パフォーマンスの考慮』 を参照してください。

数値フィールドの外部形式の指定

次の表は、数値プログラム記述フィールドの外部形式の指定方法を示したものです。形式の各タイプの詳細については、このセクションの残りの該当部分を参照してください。

表 13. 外部形式を指定するための項目および位置

フィールドのタイプ	仕様	使用する位置
入力	入力	36 桁目
出力	出力	52 桁目
配列またはテーブル	定義	EXTFMT キーワード

表 13 中のこれらのフィールドのいずれであっても、次の有効な外部数値形式のうちの 1 つを指定してください。

B 2 進数
F 浮動
I 整数
L 左符号
P パック 10 進数
R 右符号
S ゾーン 10 進数
U 符号なし

浮動数値データのデフォルトの外部形式は、外部表示表現と呼ばれています。4 バイトの浮動データの形式は次の通りです。

```
+n.nnnnnnnE+ee ここで + は符号 (+ または -) を表します。
                       n は小数部の桁数を表します。
                       e は指数の桁数を表します。
```

8 バイトの浮動データの形式は次の通りです。

```
+n.nnnnnnnnnnnnnnnE+eee
```

4 バイトの浮動値は 14 桁を占め、8 バイトの浮動値は 23 桁を占めるということに注意してください。

浮動以外の数値データの場合、デフォルトの外部形式はゾーン 10 進数です。コンパイル時配列およびテーブルの外部形式は、ゾーン 10 進数であり、左符号または右符号でなければなりません。

浮動コンパイル時配列およびテーブルの場合、コンパイル時データは、数値リテラルまたは浮動リテラルのいずれかとして指定します。4 バイトの浮動配列の各エレメントはソース・レコード中で 14 桁を必要とし、8 バイトの浮動配列の各エレメントは 23 桁を必要とします。

入力仕様、演算仕様、または出力仕様で定義されているが、定義仕様で対応する定義のない浮動でない数値フィールドは、パック 10 進数形式で内部的に保管されます。

文字、グラフィック、または UCS-2 フィールドの外部形式の指定

108 ページの表 13 中のいずれの入力フィールドおよび出力フィールドであっても、次の有効な外部データ形式のうちの 1 つを指定してください。

A 文字 (文字および標識データに有効)

N 標識 (文字および標識データに有効)

G グラフィック (グラフィック・データに有効)

C UCS-2 (UCS-2 データに有効)

配列またはテーブルのデータを UCS-2 形式で指定するには、EXTFMT キーワードを使用することができます。

可変長の文字、グラフィック、または UCS-2 データの場合には、入力仕様の 31 - 34 桁目と出力仕様の 53 - 80 桁目に *VAR データ属性を指定します。

日付 / 時刻フィールドの外部形式の指定

プログラム記述ファイル中に日付、時刻、およびタイム・スタンプのフィールドがある場合には、その外部形式を指定し *なければなりません*。ファイル仕様で DATFMT キーワードおよび TIMFMTキーワードを使用することによって、プログラム記述ファイル中のすべての日付、時刻、およびタイム・スタンプのフィールドにデフォルトの外部形式を指定することができます。同様に特定のフィールドの外部形式を指定することができます。所要の形式を入力仕様の 31 ～ 34 桁目に指定してください。適切なキーワードおよび形式を出力仕様の 53 ～ 80 桁目に指定します。

各形式タイプの詳細については、この章の残りの該当セクションを参照してください。

基底ポインター・データ・タイプ

基底ポインターは、基底付き変数のストレージを見つけるために使用されます。ストレージは、特定の基底ポインター変数に基づいてフィールド、配列、またはデータ構造を定義し、必要な記憶場所を示すようにその基底ポインター変数を設定することによって、アクセスされます。

たとえば、長さが 5 の文字フィールドで、ポインター PTR1 に基づいている基底付き変数 MY_FIELD を考えてください。基底付き変数は、ストレージで固定位置をもっていません。ポインターを使用して、その変数の現在の記憶位置を示さなければなりません。

次がストレージの一部のエリアのレイアウトである場合を想定してください。

```
-------------------------------------------------------
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
-------------------------------------------------------
```

G を指すようにポインター PTR1 を設定したとします。

```
   PTR1-------------------.
                          |
                          V
-------------------------------------------------------
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
-------------------------------------------------------
```

ここで MY_FIELD は 'G' から始まるストレージにあることになり、その値は 'GHIJK' となります。ポインターが 'J' を示すように移動されると、MY_FIELD の値は 'JKLMN' になります。

```
               PTR1-------------------.
                                      |
                                      V
-------------------------------------------------------
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
-------------------------------------------------------
```

ここで MY_FIELD が EVALステートメントによって 'HELLO' に変更された場合には、'J' から始まるストレージは次のように変更されます。

```
               PTR1-------------------.
                                      |
                                      V
-------------------------------------------------------
| A | B | C | D | E | F | G | H | I | H | E | L | L | O | O |
-------------------------------------------------------
```

フィールドの基底ポインターを定義するには、定義仕様で BASED キーワード (267 ページの『BASED(basing_pointer_name)』 を参照) を使用します。基底ポインターは、基底付きフィールドと同じ有効範囲をもちます。

基底ポインター・フィールドの長さは 4 バイト長でなければならず、また 4 バイト境界に位置合わせされなければなりません。境界合わせのこの要件のため、データ構造のポインター・サブフィールドが先行フィールドの直後に続かなくなること

も、複数回繰り返しデータ構造が不連続のオカレンスをもつようになることもあります。サブフィールドの位置合わせの詳細については、 167 ページの『データ構造サブフィールドの位置合わせ』を参照してください。

基底ポインターのデフォルトの初期設定値は *NULL です。

注:

1. 基底ポインターをコーディングする時には、十分な長さで、基底付きフィールドに正しいタイプのストレージを示すポインターを設定しなければなりません。114 ページの図 31 は、基底ポインターをコーディング しない方法の例を一部示したものです。
2. 式中でポインターからのオフセットを加算または減算することができます。たとえば、EVAL ptr = ptr + オフセット。ポインターによる演算を行なう時には、指そうとする項目のストレージ内を今まで通り指示しているようにすることはユーザーの責任であるということに注意してください。ほとんどの場合、この項目の前または後を示している場合には、例外とはなりません。

   2 つのポインターを減算して、それらの間のオフセットを決定する時には、それらのポインターが同じスペースを示すか、あるいは同じタイプのストレージを示さなければなりません。たとえば、静的ストレージ中の 2つのポインター、自動ストレージ中の 2 つのポインター、あるいは同じユーザー・スペース内の 2 つのポインターを減算することができます。

基底ポインターの設定

基底付き変数の位置の設定または変更は、次のいずれかの方法で基底ポインターを設定または変更することによって、行ないます。

- INZ(%ADDR(FLD)) で初期化する。FLD は基底付きでない変数です。
- ポインターを %ADDR(X) の結果に割り当てる。X はいずれかの変数です。
- ポインターを別のポインターの値に割り当てる。
- ALLOC または REALLOC を使用する (例については、485 ページの『ALLOC (ストレージの割り振り)』 および 654 ページの『REALLOC (新規長さのストレージの再割り振り)』 を参照してください)。
- ポインター演算を使用してポインターを次または前に移動する。

```
EVAL        PTR = PTR + offset
```

(″offset″ は、ポインターが移動される距離 (バイト数) です)

例

図 28 は、基底付きデータ構造の定義方法を示したものです。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
 *
 *  基底付きのデータ構造、配列、およびフィールドを定義します。
 *  PTR1 が定義されていない場合には、コンパイラーによって暗黙的に
 *  定義されます。
 *
 *  これらの基底付きのフィールドまたは構造が使用できるようになるには、
 *  基底ポインターが正しい記憶位置を示すように設定されていなければ
 *  ならないということに注意してください。DSbased データ構造が使用
 *  される前に、PTR1 が有効なストレージ・アドレスに設定されます。
 *
D DSbased         DS                  BASED(PTR1)
D    Field1                   1  16A
D     Field2                     2
D
D ARRAY           S              20A   DIM(12) BASED(PTR2)
D
D Temp_fld        S                *   BASED(PTR3)
D
D PTR2                             *   INZ
D PTR3                             *   INZ(*NULL)
```

図 28. 基底付き構造およびフィールドの定義

次は、ポインターからのオフセットをどのように加算および減算して、2 つのポインター間の差を決定するかを示したものです。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.....................................Keywords++++++++++++++++++++++++++++++
 *
D P1              s               *
D P2              s               *
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CSRN01+++++++++++++++Opcode(E)+Extended Factor 2++++++++++++++++++++++++++++
 *
 * ポインター P1 に 20 バイトのストレージを割り振ります。
C                   ALLOC     20            P1
 * ストレージを 'abcdefghij' に初期化します。
C                   EVAL      %STR(P1:20) = 'abcdefghij'
 * このストレージの 9 番目のバイトを示すように P2 を設定します。
C                   EVAL      P2 = P1 + 8
 * P2 が 'i' を指していることを示します。%STR は、このポインター
 * が示しているところまでのデータを戻しますが (最初に見つけた
 * x'00' ヌル終結文字は含みません)、指定の長さ (この場合には 1)
 * を検索するだけです。
C                   EVAL      Result = %STR(P2:1)
C                   DSPLY                   Result            1
 * 前のバイトを示すように P2 を設定します。
C                   EVAL      P2 = P2 - 1
 * P2 が 'h' を指していることを示します。
C                   EVAL      Result = %STR(P2:1)
C                   DSPLY                   Result
 * P1 と P2 がどの程度離れているかを検出します。(7 バイト)
C                   EVAL      Diff = P2 - P1
C                   DSPLY                   Diff              5 0
 * P1 のストレージを解放します。
C                   DEALLOC                 P1
C                   RETURN
```

図 29. ポインターによる演算

図 30 は、年間通算日が必要な場合に、年間通算日形式での日数を入手する方法を示したものです。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7....+....
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DATFMT(*JUL)
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
D JulDate         S               D   INZ(D'95/177')
D                                     DATFMT(*JUL)
D JulDS           DS                  BASED(JulPTR)
D  Jul_yy                        2  0
D  Jul_sep                       1
D  Jul_ddd                       3  0
D JulDay          S              3  0
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CSRN01+++++++++++++++Opcode(E)+Extended Factor 2++++++++++++++++++++++++++++
 *
 * 年間通算日をオーバーレイする構造の基底ポインターを設定します。
C                   EVAL      JulPTR = %ADDR(JulDate)
 * 年間通算日の日の部分を抜き出します。
C                   EVAL      JulDay = Jul_ddd
```

図 30. 年間通算日の入手

基底ポインターをコーディングする時には、十分な大きさで、基底付きフィールドに正しいタイプであるストレージを指すようにポインターを設定するようにしてください。図 31 は、基底ポインターをコーディング しない 方法の例を一部示したものです。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
  *
D chr10           S             10a   based(ptr1)
D char100         S            100a   based(ptr1)
D p1              S              5p 0 based(ptr1)
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CSRN01+++++++++++++++Opcode(E)+Extended Factor 2++++++++++++++++++++++++++++
  *
  *
  * ptr1 を p1 (数値フィールド) のアドレスに設定します。
  * chr10 (ptr1 を基底とするもの) を 'abc' に設定します。
  * データ・タイプに互換性がないために、p1 に書き込まれる
  * データは信頼性のないものになります。
  *
C                   EVAL      ptr1 = %addr(p1)
C                   EVAL      chr10 = 'abc'
  *
  * ptr1 を chr10 (10 バイト・フィールド) のアドレスに設定します。
  * chr100 (100 バイト・フィールド) をすべて 'x' に設定します。
  * 10 バイトが chr10 に書き込まれ、90 バイトが他のストレージ (不明
  * の位置) に書き込まれます。
  *
C                   EVAL      ptr1 = %addr(chr10)
C                   EVAL      chr100 = *all'x'
```

図 31. 基底ポインターをコーディングしない方法

文字データ・タイプ

文字データ・タイプは、文字値を表すもので、次の形式の 1 つとなります。

A 文字

N 標識

G グラフィック

C UCS-2

文字データには、指定された形式によって 1 つまたは複数の単一バイトまたは 2 バイト文字が含まれている場合があります。文字、グラフィック、および UCS-2 フィールドは固定長または可変長のいずれかの形式とすることもできます。ストリングに対して働く命令コードは、文字データを受け入れます。以下の表は、各種の文字データ・タイプ形式を要約したものです。

文字 データ・タイプ	**バイト数**	**CCSID**
文字	1 つまたは複数の**単一バイト**文字で、長さは固定または可変	ワークステーションの CCSID と見なされます
標識	1 つの **1 バイト**文字で、長さは固定	ワークステーションの CCSID と見なされます
グラフィック	1 つまたは複数の **2 バイト**文字で、長さは固定または可変	ワークステーションの CCSID または有効なユーザー定義の 2 バイト CCSID
UCS-2	1 つまたは複数の **2 バイト**文字で、長さは固定または可変	13488 (UCS-2 バージョン 2.0)

文字データの CCSID については、 124 ページの『文字、グラフィック、および UCS-2 の間のデータ変換』を参照してください。

非標識文字フィールドのデフォルトの初期設定値はブランクです。

標識は特殊なタイプの文字データです。標識データは、ファイル仕様の COMMIT キーワードに指定された標識およびフィールドからなります。標識はすべて 1 バイトの長さで、'0' と '1' の文字値だけを含みます。標識のデフォルト値は '0' です。

文字形式

固定長文字形式は、1 つまたは複数のバイトの設定した長さです。

可変長文字形式については、 118 ページの『可変長の文字形式、グラフィック形式、および UCS-2 形式』を参照してください。

文字フィールドは、適切な仕様の「データ・タイプ」項目に A を指定することによって定義します。パラメーターが文字フィールドとなっている定義仕様で LIKE キーワードを使用しても定義することができます。

デフォルトの初期設定値はブランクです。

標識形式

標識形式は特殊なタイプの文字データです。標識はすべて 1 バイトの長さで、'0' (オフ) と '1' (オン)の文字値だけを含みます。標識は、一般的に演算命令の結果を示すか、あるいは演算命令の処理を条件付ける (制御する) ために使用されます。標識のデフォルト値は '0' です。

標識フィールドは、適切な仕様の「データ・タイプ」項目に N を指定することによって定義します。パラメーターが標識フィールドとなっている定義仕様で LIKE キーワードを使用しても標識フィールドを定義することができます。標識フィールドはまた、ファイル仕様の COMMIT キーワードによって暗黙的に定義されます。

標識変数を定義する場合の規則は次の通りです。

- 標識は、独立フィールドとして、サブフィールドとして、プロトタイプ・パラメーターとして、またプロシージャー戻り値として定義することができます。
- 標識変数を実行前またはコンパイル時の配列またはテーブルとして定義する場合には、初期設定データは 'O' と '1' だけから構成されていなければなりません。

  注: 実行時に標識に '0' または '1' 以外の値が含まれている場合には、結果は予測できません。
- キーワード INZ を指定する場合には、値は '0'、*OFF、'1'、または *ON のうちの 1 つでなければなりません。
- 標識フィールドにはキーワード VARYING を指定することはできません。

標識変数を使用する場合の規則は次の通りです。

- 標識フィールドのデフォルトの初期設定値は '0' です。
- 命令コード CLEAR は標識変数を '0' に設定します。
- 標識変数に適用される「後で消去」機能は標識変数を '0' に設定します。
- MOVEA(P) 演算命令の結果として標識の配列が指定されている場合には、埋め込み文字は '0' となります。
- 外部キーが長さ 1 の文字である場合には検索引き数として標識を使用することができます。

グラフィック形式

グラフィック形式は、各文字が 2 バイトによって表されるような文字ストリングです。

単一バイトと 2 バイトのグラフィック・データとの間の相違を以下の図に示してあります。

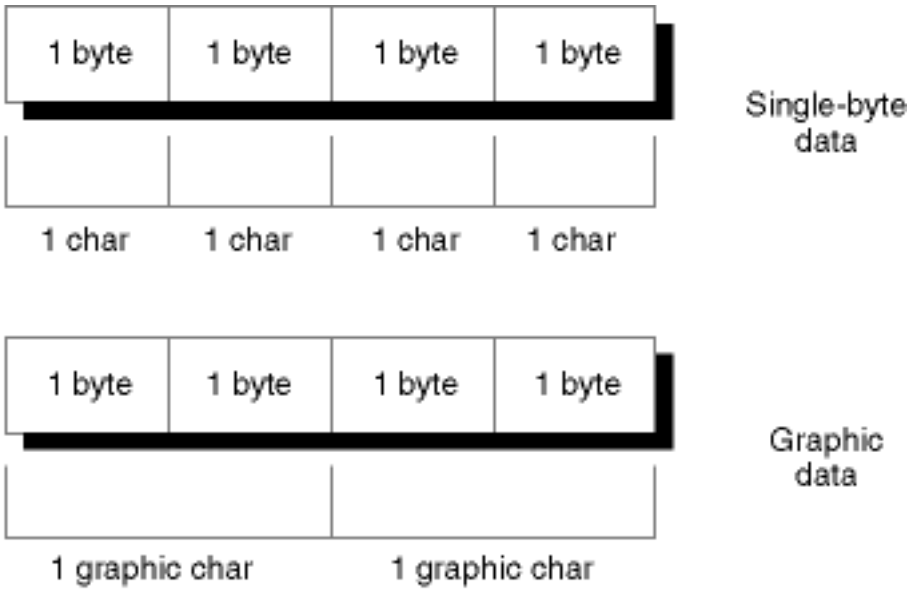


図 32. 単一バイトとグラフィック・データの比較

グラフィック・フィールドの長さ (バイト数) は、フィールド中のグラフィック文字の数を 2 倍したものです。

レコードが追加され、データベース・ファイルおよびグラフィック・フィールドが出力用には指定されていない場合には、出力用にフィールドには 2 バイトのブランクが入れられます。出力フィールドにブランクが入れられるのは次の条件の下です。

- 出力仕様で出力用にフィールドが指定されていない。
- フィールドで条件標識が満たされない。

グラフィック・データは固定長または可変長とすることができます。固定長のグラフィック形式は、設定された長さをもち、各文字が 2 バイトによって表される文字ストリングです。

可変長文字形式については、 118 ページの『可変長の文字形式、グラフィック形式、および UCS-2 形式』を参照してください。

グラフィック・フィールドは、適切な仕様の「データ・タイプ」項目に G を指定することによって定義します。パラメーターがグラフィック・フィールドとなっている定義仕様で LIKE キーワードを使用しても定義することができます。

グラフィック・データのデフォルトの初期設定値は 2 バイトのブランクです。その 16 進値は、ワークステーションにインストールされたコード・ページによります。*HIVAL の値は X'FFFF' で、*LOVAL の値は X'0000' です。

UCS-2 形式

汎用文字セット (UCS-2) 形式は、各文字が 2 バイトによって表される文字ストリングです。この文字セットは、多くの作成言語で文字をエンコードすることができます。

UCS-2 フィールドの長さ (バイト数) は、フィールド中の UCS-2 文字の数を 2 倍したものです。

固定長の UCS-2 形式は、設定された長さをもち、各文字が 2 バイトによって表される文字ストリングです。

可変長の UCS-2 形式については、『可変長の文字形式、グラフィック形式、および UCS-2 形式』 を参照してください。

UCS-2 フィールドは、適切な仕様の「データ・タイプ」項目に C を指定することによって定義します。パラメーターが UCS-2 フィールドとなっている定義仕様で LIKE キーワードを使用しても定義することができます。

UCS-2 データのデフォルトの初期設定値は X'0020' です。*HIVAL の値は X'FFFF' で、*LOVAL の値は X'0020' で、*BLANK の値は X'0020' です。

UCS-2 形式の詳細については、Web サイト **http://www.ibm.com/eserver/iseries/infocenter** の **Information Center** にあるプログラミング ・カテゴリーの *CL* および *API* セクションを参照してください。

可変長の文字形式、グラフィック形式、および UCS-2 形式

可変長の文字フィールドは宣言された最大長をもちますが、その現在の長さはプログラムの実行中に変わることがあります。文字形式の場合にはこの長さは単一バイトで計測され、グラフィック形式および UCS-2 形式の場合には長さは 2 バイトで計測されます。可変長文字フィールドに割り振られるストレージは、宣言された最大長より 2 バイト長くなります。左端の 2 バイトは、文字数、グラフィック文字数、または UCS-2 文字数による現在の長さが入る符号なし整数フィールドです。実際の文字データは、可変長フィールドの 3 番目のバイトから始まります。

図 33 には、可変長文字フィールドがどのように保管されるかが示されています。

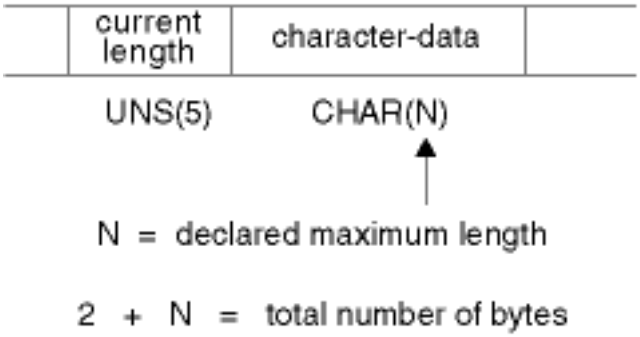


図 33. 可変長形式の文字フィールド

図 34 には、可変長グラフィック・フィールドがどのように保管されるかが示されています。UCS-2 フィールドも同様に保管されます。

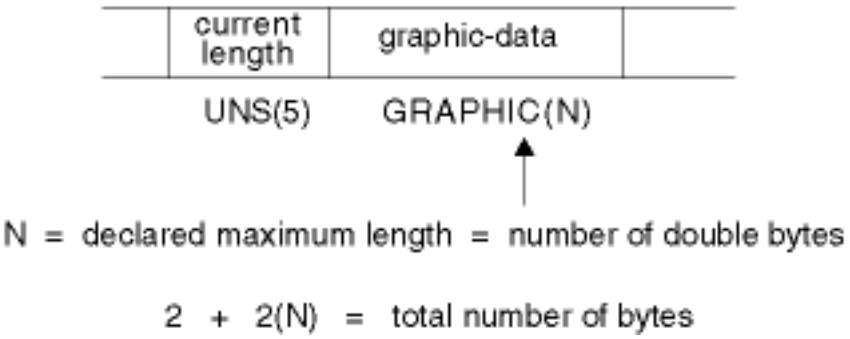


図 34. 可変長形式のグラフィック・フィールド

注: 現在の長さ (これを含む) までのデータだけが有効です。

可変長文字フィールドは、定義仕様で A (文字)、G (グラフィック)、または C (UCS-2) を指定し、さらに定義仕様でキーワード VARYING を指定することによって定義します。パラメーターが可変長文字フィールドとなっている定義仕様で LIKE キーワードを使用しても定義することができます。

*VAR データ属性をもつ外部可変長フィールドは、入力仕様または 出力仕様で参照することができます。

デフォルトの初期設定値は、長さがゼロのヌルストリング ('') です。

可変長フィールドの使用例については、以下を参照してください。
- 122 ページの『可変長フィールドの使用』
- 438 ページの『%LEN (長さの取り出しまたは設定)』
- 408 ページの『%CHAR (文字データに変換)』
- 452 ページの『%REPLACE (文字ストリングの置き換え)』

可変長形式はグラフィック・データでも使用可能です。

可変長の文字形式、グラフィック形式、および UCS-2 形式の規則

可変長フィールドを定義する時には、次の規則が適用されます。

- フィールドの宣言する長さは単一バイトで 1 ～ 65535 文字、2 バイト・グラフィック文字または UCS-2 文字で 1 ～ 16383 文字とすることができます。
- 現在の長さは、0 からフィールドに宣言された最大長までの範囲内の値となります。
- フィールドは、キーワード INZ を使用して初期化することができます。初期値は指定された正確な値で、フィールドの初期長は初期値の長さです。フィールドは初期化のためのブランクを埋め込まれますが、長さにはブランクは含められません。
- 位取り表記法を使用して定義するサブフィールドを除いたすべての場合に、長さ項目 (定義仕様の 33 ～ 39 桁目) には、2 バイト長を含まない最大フィールド長が入ります。
- 位取り表記法を使用して定義するサブフィールドの場合は、長さには 2 バイト長を含めます。結果として、可変長サブフィールドは、名前のないデータ構造の場合に単一バイトで 65537、2 バイトで 16384 の長さとすることができます。
- データ構造にはキーワード VARYING は指定することはできません。
- 可変長の実行前配列の場合、ファイル中の初期設定データは、2 バイトの長さのプレフィックスを含めた可変長形式で保管されます。
- 実行前配列データはファイルから読み取られ、ファイルの最大レコード長は 32766 であるので、可変長の実行前配列の最大サイズは単一バイトで 32764 文字、2 バイトのグラフィック文字または UCS-2 文字で 16382 文字となります。
- 可変長の配列またはテーブルは、コンパイル時データで定義することもできます。データ・フィールド中の末尾ブランクは意味をもちません。データの長さは、フィールド中の最後の非ブランク文字の位置にあります。長さプレフィックスはコンパイル時データには保管できないので、これは実行前初期化とは異なります。
- グラフィック・コンパイル時データの場合には、単一バイトのブランクは有効データと見なされます。グラフィック配列およびテーブルのコンパイル時データは、2 バイトのブランクで埋め込む必要があります。単一バイトのブランクは、非ブランクと見なされます。
- 可変長フィールドのようなフィールドを定義するのに、*LIKE DEFINE を使用することはできません。

以下は、可変長文字フィールドの定義例です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++++

 * 独立フィールド:
D var5            S              5A   VARYING
D var10           S             10A   VARYING INZ('0123456789')
D max_len_a       S          32767A   VARYING

 * 実行前配列:
D arr1            S            100A   VARYING FROMFILE(dataf)

 * データ構造サブフィールド:
D ds1             DS

 *   長さ表記で定義されたサブフィールド:
D   sf1_5                       5A   VARYING
D   sf2_10                     10A   VARYING INZ('0123456789')

 *   位取り表記法を使用して定義されたサブフィールド: A(5)VAR
D   sf4_5              101    107A   VARYING


 *   可変の内部表現を表示するサブフィールド:
D   sf7_25                    100A   VARYING
D   sf7_len                     5I 0 OVERLAY(sf7_25:1)
D   sf7_data                  100A   OVERLAY(sf7_25:3)

 * プロシージャー・プロトタイプ
D Replace         PR         32765A   VARYING
D   String                   32765A   CONSTANT VARYING OPTIONS(*VARSIZE)
D   FromStr                  32765A   CONSTANT VARYING OPTIONS(*VARSIZE)
D   ToStr                    32765A   CONSTANT VARYING OPTIONS(*VARSIZE)
D   StartPos                     5U 0 VALUE
D   Replaced                    5U 0 OPTIONS(*OMIT)
```

図 35. 可変長の文字フィールドおよび UCS-2 フィールドの定義

以下は、可変長のグラフィック・フィールドおよび UCS-2 フィールドの定義例です。

```
* .. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+...
DName++++++++++ETDsFrom+++To/L+++IDc.Functions+++++++++++++++++
 *----------------------------------------------------------
 * グラフィック・フィールド
 *----------------------------------------------------------
 * 独立フィールド:
D GRA20           S             20G   VARYING
D MAX_LEN_G       S          16383G   VARYING
 * 実行前配列:
D ARR1            S            100G   VARYING FROMFILE(DATAF)
 * データ構造サブフィールド:
D DS1             DS
 *   長さ表記で定義されたサブフィールド:
D   SF3_20                     20G   VARYING
 *   位取り表記法を使用して定義されたサブフィールド: G(10)VAR
D   SF6_10               11    32G   VARYING
 *----------------------------------------------------------
 *   UCS-2 フィールド
 *----------------------------------------------------------
D MAX_LEN_C       S          16383C   VARYING
D FLD1            S              5C   INZ(%UCS2('ABCDE')) VARYING
D FLD2            S              2C   INZ(U'01230123') VARYING
D FLD3            S              2C   INZ(*HIVAL) VARYING
D DS_C            DS
D   SF3_20_C                   20C   VARYING
 *   位取り表記法を使用して定義されたサブフィールド: C(10)VAR
D   SF_110_C             11    32C   VARYING
```

図 36. 可変長のグラフィック・フィールドおよび UCS-2 フィールドの定義

可変長フィールドの使用

可変長フィールドの長さ部分は、文字数で計測されたフィールドの現在の長さを表します。文字フィールドの場合、この長さはまた、現在の長さのバイト数で表します。2 バイト・フィールド (グラフィックおよび UCS-2) の場合、これは、フィールドの長さを 2 バイト単位を表します。たとえば、現在の長さが 3 の UCS-2 フィールドは、2 バイト文字 3つ分の長さ、すなわち 6 バイトの長さになります。

以下のセクションでは、可変長フィールドの最適な使用法について説明するとともに、各種の命令コードを使用した時に現在の長さがどのように変わるかについて説明します。

フィールドの長さの設定方法: 可変長フィールドが INZ を使用して初期化された時には、初期の長さは、初期設定値の長さとなるように設定されます。たとえば、長さが 10 の文字フィールドが値 'ABC' に初期化された場合には、初期の長さは 3 に設定されます。

可変長ターゲットの長さは EVAL 演算命令で変更されます。たとえば、長さが 10 の文字フィールドに値 'XY' が割り当てられた場合には、長さは 2 に設定されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D fld                     10A        VARYING
 * EVAL の前では 'fld' がどのような長さになっていても関係ありません。
 * EVAL の後では、長さは 2 になります。
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                   EVAL      fld = 'XY'
```

CLEAR 演算命令は可変長フィールドの長さを 0 に変更します。

PARM 演算命令は、結果フィールドの長さを演算項目 2 (指定されている場合) のフィールド長に設定します。

固定形式の演算命令 MOVE、MOVEL、CAT、SUBST、および XLATE は、可変長の結果フィールドの長さを変更しません。たとえば、MOVE を使用して長さが 10 で、現在の長さが 2 の可変長文字フィールドに値 'XYZ'が移動される場合には、フィールドの長さは変わらず、データは切り捨てられます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D fld                     10A        VARYING
      * MOVEL の前に fld の長さは 2 になっていると想定してください。
      * 最初の MOVEL の後では、'XY' の値をもつことになります。
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                   MOVEL     'XYZ'      fld
      * 2 番目の MOVEL の後では、'1Y' の値をもつことになります。
C                   MOVEL     '1'        fld
```

注: EVAL とは対照的に、MOVE および MOVEL の場合の推奨される用法は、一時的に長さを固定したいフィールドの値を変更することです。例として、サイズが毎日変わるかもしれない欄をもつが、そのサイズがプログラムの一定実行時点では固定しなければならない報告書を作成する場合があります。

フィールドがファイルから読み取られると (入力仕様)、可変長フィールドの長さは入力データの長さに設定されます。

出力仕様の「後で消去」機能は、可変長フィールドの長さを 0 に設定します。

可変長フィールドの長さは、EVAL 演算命令の左側に %LEN 組み込み関数を使用して自身で設定することができます。

フィールド長の使用方法: 値に可変長フィールドが使用される時には、その現在の長さが使用されます。次の例の場合、'result'は、長さが 7 の固定長フィールドであると想定してください。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
D fld                      10A        VARYING
 * 次の EVAL 演算命令の場合
 *    'fld' の値        'fld' の長さ      'result'
 *    --------------    ---------------   -----------
 *    'ABC'             3                 'ABCxxx '
 *    'A'               1                 'Axxx   '
 *    ''                0                 'xxx    '
 *    'ABCDEFGHIJ'      10                'ABCDEFG'
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                   EVAL      result = fld + 'xxx'
 * 次の MOVE 演算命令では MOVE の前に 'result' は値 '.......' を
 * もっていると想定してください。
 *    'fld' の値        'fld' の長さ      'result'
 *    --------------    ---------------   -----------
 *    'ABC'             3                 '....ABC'
 *    'A'               1                 '......A'
 *    ''                0                 '.......'
 *    'ABCDEFGHIJ'      10                'DEFGHIJ'
C                   MOVE      fld           result
```

可変長フィールドを使用する理由: 一時変数に可変長フィールドを使用すると、ストリング演算命令のパフォーマンスが改善されるだけでなく、フィールドの現在の長さを %SUBST の別の変数に保管したり、%TRIM を使用して余分のブランクを無視したりする必要がないので、コードが読みやすくなります。

サブプロシージャーが異なる長さのストリング・データを処理するようになっている場合には、プロトタイプ・プロシージャーのパラメーターおよび戻り値に可変長フィールドを使用すると、呼び出しおよびプロシージャーのパフォーマンスと読みやすさの両方が改善されることがあります。パラメーターの実際の長さを取り出すのに、サブプロシージャー内で長さパラメーターを渡す必要がなくなります。

文字、グラフィック、および UCS-2 の間のデータ変換

注: グラフィック CCSID が無視される (制御仕様に CCSID(*GRAPH:*IGNORE) が指定されているか、あるいは CCSID(*GRAPH) がまったく指定されていない) 場合には、グラフィック・データは CCSIDをもたないと見なされ、グラフィック・データと UCS-2 データの間の変換はサポートされません。

文字、グラフィック、および UCS-2 データは、異なる CCSID (コード化文字セット ID) をもつことがあります。データ・タイプ間の変換は、データの CCSID に依存します。

データの CCSID

文字データの CCSID は、文字データと UCS-2 データの間で変換を行なっている時にのみ考慮されます。

文字データと UCS-2 データまたはグラフィック・データの間で変換を行なう時には、文字データの CCSIDはワークステーションの CCSID と見なされます。

UCS-2 データの CCSID のデフォルトは 13488 です。このデフォルトは、制御仕様で CCSID(*UCS2) キーワードを使用して変更することができます。プログラム記述

UCS-2 フィールドの CCSID は、定義仕様で CCSIDキーワードを使用して指定することができます。外部記述 UCS-2 フィールドの CCSID は、外部ファイルから取られます。

注: UCS-2 フィールドは、DDS 中で G のデータ・タイプおよび 13488 の CCSID を指定して定義します。

グラフィック・データの CCSID のデフォルトは、制御仕様の CCSID(*GRAPH) キーワードに指定された値です。プログラム記述グラフィック・フィールドの CCSID は、定義仕様で CCSID キーワードを使用して指定することができます。外部記述グラフィック・フィールドの CCSID は、外部ファイルから取られます。

日付データ

日付フィールドは、事前に決定されたサイズおよび形式をもちます。これは、定義仕様で定義することができます。日付データにはすべて先行ゼロおよび後続ゼロが必要です。

比較または割り当てで使用される日付固定情報または日付変数は、同じ形式である必要も、同じ区切り文字を使用する必要もありません。入力フィールド、出力フィールド、キー・フィールドなどの I/O 演算命令に使用される日付は、演算命令に必要な形式に変換されます (必要な場合)。

日付変数のデフォルトの内部形式は、*ISO です。このデフォルトの内部形式は、制御仕様キーワードの DATFMT によってグローバルに指定変更することも、定義仕様キーワードの DATFMT によって個別に指定変更することもできます。

日付フィールドの内部日付形式および区切り文字を決定する時に使用される階層は、次の通りです。
1. 定義仕様に指定された DATFMT キーワードから
2. 制御仕様に指定された DATFMT キーワードから
3. *ISO

表現可能な年の範囲によって、3 種類の日付データ形式があります。これにより、演算命令の結果がターゲット・フィールドの有効範囲の外側の日付となる時に、日付のオーバーフローまたはアンダーフロー条件が起こる可能性が出てきます。形式および範囲は次の通りです。

年の桁数	年の範囲
2 (*YMD, *DMY, *MDY, *JUL)	1940 - 2039
3 (*CYMD, *CDMY, *CMDY)	1900 - 2899
4 (*ISO, *USA, *EUR, *JIS, *LONGJUL)	0001 - 9999

126 ページの表 14 には、日付データの形式と区切り文字がリストされています。

日付フィールドのコーディング方法の例については、次の資料の例を参照してください。
- 363 ページの『日付の演算』
- 374 ページの『日付・時刻データの移動』
- 482 ページの『ADDDUR (期間の加算)』

- 587 ページの『MOVE (転送)』
- 560 ページの『EXTRCT (日付 / 時刻 / タイム・スタンプの抽出)』
- 683 ページの『SUBDUR (期間の減算)』
- 690 ページの『TEST (日付、時刻、タイム・スタンプのテスト)』

表 14. 日付データ・タイプの RPG 定義の日付形式および区切り文字

形式名	説明	形式 (デフォルトの区切り文字)	有効な区切り文字	長さ	例
2 桁の年形式					
*MDY	月/日/年	mm/dd/yy	/ - . , '&'	8	01/15/96
*DMY	日/月/年	dd/mm/yy	/ - . , '&'	8	15/01/96
*YMD	年/月/日	yy/mm/dd	/ - . , '&'	8	96/01/15
*JUL	年間通算日	yy/ddd	/ - . , '&'	6	96/015
4 桁の年形式					
*ISO	国際標準化機構	yyyy-mm-dd	-	10	1996-01-15
*USA	IBM® USA 標準規格	mm/dd/yyyy	/	10	01/15/1996
*EUR	IBM 欧州標準規格	dd.mm.yyyy	.	10	15.01.1996
*JIS	日本工業規格西暦	yyyy-mm-dd	-	10	1996-01-15

次の表には、すべての日付形式の *LOVAL、*HIVAL、およびデフォルト値がリストされています。

表 15. 日付値

形式名	説明	*LOVAL	*HIVAL	デフォルト値
2 桁の年形式				
*MDY	月/日/年	01/01/40	12/31/39	01/01/40
*DMY	日/月/年	01/01/40	31/12/39	01/01/40
*YMD	年/月/日	40/01/01	39/12/31	40/01/01
*JUL	年間通算日	40/001	39/365	40/001
4 桁の年形式				
*ISO	国際標準化機構	0001-01-01	9999-12-31	0001-01-01
*USA	IBM USA 標準規格	01/01/0001	12/31/9999	01/01/0001
*EUR	IBM 欧州標準規格	01.01.0001	31.12.9999	01.01.0001
*JIS	日本工業規格西暦	0001-01-01	9999-12-31	0001-01-01

区切り文字

MOVE、MOVEL、または TEST 演算命令で日付形式をコーディングする時には、文字フィールドでは区切り文字は任意指定です。区切り文字がないことを示すには、後にゼロが続く形式を指定します。区切り文字のない日付形式のコーディング方法の詳細については、 587 ページの『MOVE (転送)』、 610 ページの『MOVEL (左につめて転送)』、および 690 ページの『TEST (日付、時刻、タイム・スタンプのテスト)』 を参照してください。

MOVE、MOVEL、および TEST 演算命令の場合の形式

幾つかの形式は、MOVE、MOVEL、および TEST 演算命令によって使用されるフィールドの場合にもサポートされます。このサポートは、すでに 3 桁の年形式および 4 桁の年 *LONGJUL 形式になっている外部定義値との互換性を維持するために提供されています。

表 16 には、MOVE、MOVEL、および TEST 演算命令の演算項目 1 で使用できる有効な外部定義日付形式がリストされています。

表 16. 外部定義日付形式および区切り文字

形式名	説明	形式 (デフォルトの区切り文字)	有効な区切り文字	長さ	例 (2001 年 4 月 25 日)
3 桁年形式(1.)					
*CYMD	世紀 年/月/日	cyy/mm/dd	/ - . , '&'	9	101/04/25
*CMDY	世紀 月/日/年	cmm/dd/yy	/ - . , '&'	9	104/25/01
*CDMY	世紀 日/月/年	cdd/mm/yy	/ - . , '&'	9	125/04/01
4 桁の年形式					
*LONGJUL	長年間通算日	yyyy/ddd	/ - . , '&'	8	2001/115

注:

1. 世紀文字 'c' に有効な値は次の通りです。

```
             'c'            年
            ----------------------
        0          1900-1999
        1          2000-2099
                .              .
                .              .
                .              .
                9          2800-2899
```

*CYMD 形式では文字フィールドの区切り文字は任意指定です。区切り文字がないことを示すには、*CYMD0を指定することができます。

数値データ・タイプ

数値データは、0 個以上の小数点以下の桁数をもつものとして定義されたデータから構成されます。数値データは、次の 1 つの形式をもちます。

- 128 ページの『2 進数形式』
- 130 ページの『浮動形式』
- 132 ページの『整数形式』
- 133 ページの『パック 10 進数形式』
- 134 ページの『符号なし形式』
- 135 ページの『ゾーン 10 進数形式』

数値フィールドのデフォルトの初期設定値はゼロです。

2 進数形式

2 進数形式は、符号 (正または負) がフィールドの左端のビットにあり、整数値がフィールドの残りのビットにあるということを意味します。正数は符号ビットにゼロがあります。負数は符号ビットに 1 があり、2 の補数の形式になっています。2 進数形式では、各フィールドは 2 バイトまたは 4 バイトの長さでなければなりません。

2 進数フィールドは、1 ～ 9 桁の長さにすることができ、小数点以下の桁数付きで定義することができます。フィールドの長さが 1 ～ 4 桁の場合には、コンパイラーは 2 進数フィールドが 2 バイトの長さであると見なします。フィールドの長さが 5 ～ 9 桁の場合には、コンパイラーは 2 進数フィールドが 4 バイトの長さであると見なします。

プログラム記述ファイル

2 進数形式で読み取られるすべての入力フィールドは、コンパイラーによってフィールド長 (桁数) が割り当てられます。4 という長さは 2 バイトの 2 進数フィールドに割り当てられます。9 という長さは 4 バイトの 2 進数フィールドに割り当てられます(ただし、フィールドがプログラムの他の場所に定義されていない場合)。こうした長さの制限のために、2 バイトの 2 進数フィールドに割り当てることができる最大の 10進値は 9999 で、4 バイトの 2 進数フィールドに割り当てることができる最大の 10 進値は 999 999 999 です。一般的に n 桁の 2 進数フィールドの最大値は n 個の 9です。この説明では、小数点以下の桁数がゼロであることを前提としています。

プログラム記述ファイルの場合は、次の項目で 2 進数入力フィールド、2 進数出力フィールド、および 2 進数配列またはテーブル・フィールドを指定してください。

- 2 進数入力フィールド: 入力仕様の 36 桁目に B を指定します。
- 2 進数出力フィールド: 出力仕様の 52 桁目に B を指定します。編集が指定されている場合は、この桁はブランクでなければなりません。

  2 進数形式で書き込まれるフィールドの長さは、9 桁を超えることができません。フィールドの長さが 1 ～ 4 桁の場合には、コンパイラーは 2 進数フィールドが 2 バイトの長さであると見なします。フィールドの長さが 5 ～ 9 桁の場合には、コンパイラーは 2 進数フィールドが 4 バイトの長さであると見なします。

  2 進数形式の 2 バイトのフィールドは、コンパイラーによって 1 ～ 4 桁の 10 進数フィールドに変換されるので、入力値が長くなり過ぎる場合があります。その場合には、数字の左端の桁が脱落します。たとえば、4 桁の 2 進数入力フィールドが 16 進数 6000 の 2 進値をもつ場合には、コンパイラーは、これを 10 進数 24 576 に変換します。ただし 2 は脱落し、結果は 4576 になります。同様に、入力値が 2 進数形式の 4 バイトのフィールドに長過ぎる場合もあります。2 進数フィールドの小数点以下の桁数がゼロ (0) である場合には、2 進数フィールドの代わりに整数フィールドを定義することによって、この変換の問題を回避することができます。

  注: 2 進数入力フィールドは、突き合わせフィールドまたは制御フィールドとして定義することはできません。

- 2 進数配列またはテーブル・フィールド: 定義仕様の 40 桁目に B を指定してください。コンパイル時配列およびテーブルの外部形式は、2 進数であってはなりません。

外部記述ファイル

外部記述ファイルの場合には、データ形式は、データ記述仕様の 35 桁目に指定します。フィールドの桁数は、DDS 記述の長さと正確に同じでなければなりません。たとえば、DDS 仕様で 2 進数フィールドを長さが 7 桁、小数点以下の桁数が 0 として定義した場合には、データは次のように処理されます。

1. フィールドは、入力仕様で 4 バイトの 2 進数フィールドとして定義されます。
2. VisualAge RPG プログラム中のフィールドとしてパック (7,0) フィールドが生成されます。

2 進数フィールドの完全な情報を保存したい場合には、フィールドをデータ構造で 2 進数サブフィールドとして再定義するか、またはフィールドを 2 進数独立フィールドとして再定義してください。外部記述 2 進数フィールドには、VARPG 2 進数フィールドで許される範囲の外側の値が入ることがあります。外部記述 2 進数フィールドの小数点以下の桁数がゼロ (0) である場合には、この問題を回避することができます。これを行なうには、定義仕様で外部記述 2 進数フィールドを定義し、制御仕様で EXTBININT キーワードを指定します。これにより、外部記述フィールドの外部形式が符号付き整数のそれに変更されます。

130 ページの図 37 には、各種の形式で 10 進数 8191 がどのようになるかを示しています。

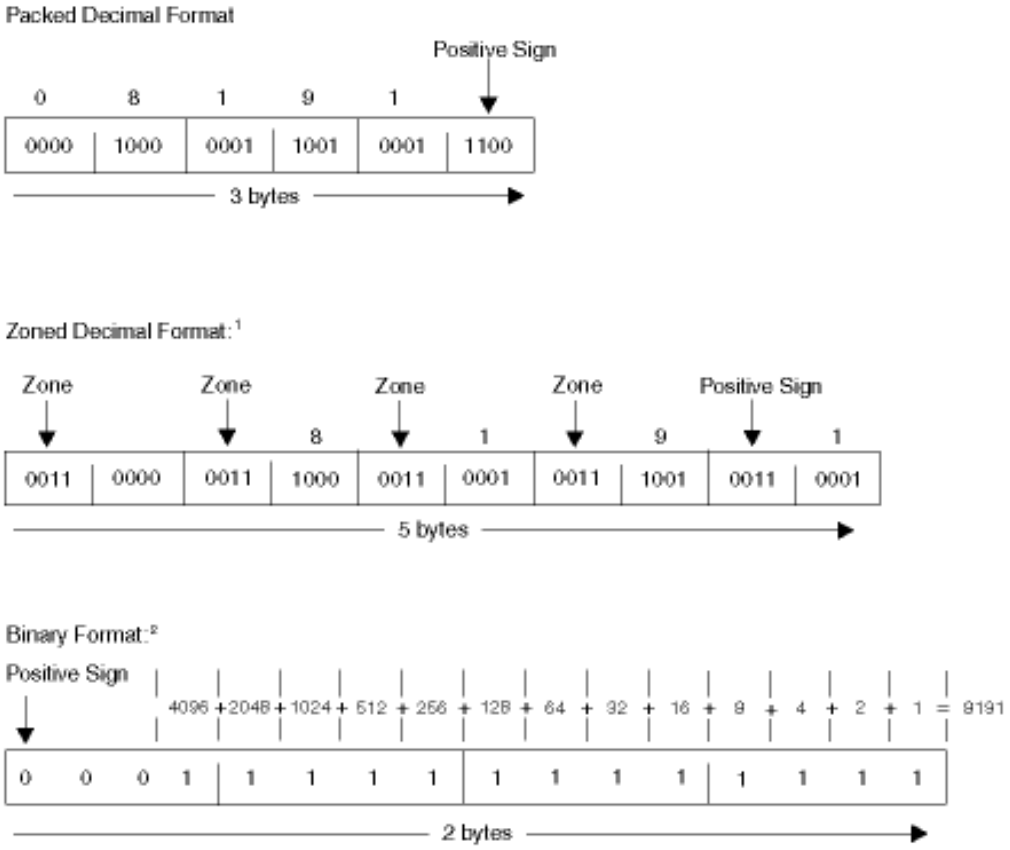


図 37. 2 進数フィールドの定義

[1] ストレージに 8191 がゾーン 10 進数として読み込まれると、それは 4 バイトを占めます。これがパック 10 進数に変換された場合には、3 バイトを占めます。ゾーン 10 進数に変換し戻されると、5 バイトを占めます。

[2] 正の 2 進数の数値を得るには、オン (1) のビットの値を加算します。符号ビットは含めません。負の 2 進数の数値を得るには、オフ (1) のビットの値を加算し、1 を加えます (符号ビットは含めません)。

浮動形式

浮動形式は次の 2 つの部分からなります。

- 仮数
- 指数

浮動小数点フィールドの値は、仮数に 10 を指数べき乗したものを乗算した結果です。たとえば、1.2345が仮数で、5が指数である場合には、浮動小数点フィールドの値は次のようになります。

```
1.2345 * (10 ** 5) = 123450
```

浮動小数点フィールドは、適切な仕様の「データ・タイプ」項目に F を指定することによって定義します。

小数点以下の桁数はブランクのままにしておかなければなりません。ただし、浮動小数点フィールドは、小数点以下の桁数をもつものと見なされます。結果として、

配列指標や DO ループ指標など、小数点以下の桁数がない数値が必要な所で、浮動変数を使用できなくなる場合があります。

浮動小数点フィールドのデフォルトの初期 CLEAR 値は 0E0 です。

浮動小数点フィールドの長さは、バイト数で定義します。これは、4 バイトまたは 8 バイトのいずれかとして指定しなければなりません。正の浮動小数点フィールドの値の許容範囲は次の通りです。

フィールド長	許容最小値	許容最大値
4 バイト	1.175 494 4 E-38	3.402 823 5 E+38
8 バイト	2.225 073 858 507 201 E-308	1.797 693 134 862 315 E+308

注: 負の値は同じ範囲をもちますが、符号は負です。

浮動変数は「科学計算」値を表すようために用いられるようになっていますから、浮動変数に保管されている数値は、パック変数と正確に同じ値を表さない場合があります。通貨金額など、数値を特定の小数点以下の桁数まで正確に表す必要がある場合には、浮動数値は使用しないでください。

浮動小数点フィールドの外部表示表現

外部表示表現の概要については、 108 ページの『数値フィールドの外部形式の指定』を参照してください。

次に対しては浮動値の外部表示表現が適用されます。
- 「データ形式」項目がブランクの浮動データの出力。
- 「データ形式」項目がブランクの浮動データの入力。
- コンパイル時および実行前の配列およびテーブル (キーワード EXTFMT が除外された場合) の外部形式。
- 命令コード DSPLY を使用した浮動値の表示および入力。
- 組み込み関数 %EDITFLT の結果。

出力: 浮動値を出力する時には、外部表現では浮動リテラルと同様の形式が使用されますが、次の例外があります。
- 値は常に仮数と指数の両方に文字 **E** と符号付きで書き込まれます。
- 値は 14 文字または 23 文字のいずれかの長さになります (それぞれ **4F** および **8F**)。
- 値は正規化されます。すなわち、小数点は最上位の有効数字の直後にあります。
- 10 進数区切り記号は、制御仕様キーワード DECEDIT のパラメーターによってピリオドまたはコンマのいずれかになります。

以下は、浮動値がどのように提示されるかの例です。

```
+1.2345678E-23
-8.2745739E+03
-5.722748027467392E-123
+1,2857638E+14             DECEDIT(',') が指定された場合
```

入力: 浮動値を入力する時には、値は、浮動リテラルのように指定します。値は、正規化する必要も、フィールドに合わせて位置合わせする必要もありません。浮動

値が配列 / テーブル初期設定データとして定義されている時には、値は 14 文字または 23 文字のいずれかの長さのフィールドに指定します (それぞれ **4F** および **8F**)。

浮動フィールドについては、以下のことに注意してください。

- 浮動サブフィールドのアクセスのパフォーマンスを改善するために、浮動フィールドの位置合わせが必要になる場合があります。ALIGN キーワードを使用すれば、定義仕様で定義された浮動サブフィールドを位置合わせすることができます。4 バイトの浮動サブフィールドは 4 バイト境界に位置合わせされ、8 バイトの浮動サブフィールドは 8 バイト境界に位置合わせされます。浮動サブフィールドの位置合わせの詳細については、266 ページの『ALIGN』 を参照してください。
- 浮動フィールドのようなフィールドを定義するために LIKE キーワードが使用された時には、長さ調整を行なうことはできません。

整数形式

整数形式は、次の 2 つの点を除けば 2 進数形式と似ています。

- 整数形式では全範囲の 2 進数値を使用することができます
- 整数フィールドの小数点以下の桁数は常にゼロです

整数フィールドは、適切な仕様の「データ・タイプ」項目に I を指定することによって定義します。パラメーターが整数フィールドとなっている定義仕様で LIKE キーワードを使用しても整数フィールドを定義することができます。

整数フィールドの長さは、桁数で定義し、3、5、10、または 20 桁の長さにすることができます。3 桁フィールドは 1 バイトのストレージを占め、5 桁フィールドは 2 バイトのストレージを占め、10 桁フィールドは 4 バイトのストレージを占め、20 桁フィールドは 8 バイトのストレージを占めます。整数フィールドの値の許容範囲は、その長さによります。

フィールド長
許容される値の範囲

3 桁整数
-128 - 127

5 桁整数
-32768 - 32767

10 桁整数
-2147483648 - 2147483647

20 桁整数
-9223372036854775808 - 9223372036854775807

整数フィールドについては、以下のことに注意してください。

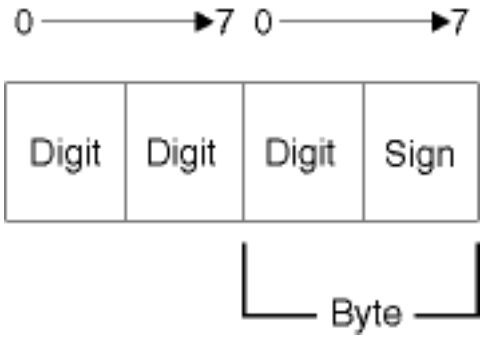
- 整数サブフィールドのアクセスのパフォーマンスを改善するために、整数フィールドの位置合わせが必要になる場合があります。ALIGN キーワードを使用すれば、定義仕様で定義された整数サブフィールドを位置合わせすることができます。

2 バイトの整数サブフィールドは 2 バイト境界に位置合わせされ、4 バイトの整数サブフィールドは 4 バイト境界に位置合わせされ、8 バイトの整数サブフィールドは 8 バイト境界に位置合わせされます。整数サブフィールドの位置合わせの詳細については、 266 ページの『ALIGN』を参照してください。

- 整数フィールドのようなフィールドを定義するために LIKE キーワードを使用した場合には、「長さ」項目に桁数単位での長さ調整を含めることができます。 調整値は、フィールドの結果の桁数が 3、5、10、または 20 となるようなものでなければなりません。

パック 10 進数形式

パック 10 進数形式は、ストレージの各バイトが 2 つの 10 進数を含めることができる (下位バイトは除きます) ということを意味します。下位バイトは左側に 1 つの数字、右側に符号 (正または負) が入ります。すべてのパック 10 進数では優先符号 (正数には 16 進数 C、負数には 16 進数 D) が使用されます。さらに、次の符号がサポートされています。正数には 16 進数 A、E、F、負数には 16 進数 B。パック 10 進数形式は次のようになっています。



130 ページの図 37 では、パック 10 進数形式で 10 進数 8191 がどのようになるかを示しています。

プログラム記述ファイル:

- パック 10 進数入力の場合は、入力仕様の 36 桁目に P を指定します。

  パック 10 進数出力の場合は、出力仕様の 52 桁目に P を指定します。編集が指定されている場合は、この桁はブランクでなければなりません。

  パック 10 進数配列およびテーブルの場合は、定義仕様の 40 桁目に P を指定します。コンパイル時配列およびテーブルの外部形式は、パック 10 進数形式とすることはできません。

外部記述ファイルの場合には、データ形式は、データ記述仕様に指定します。

パック 10 進数の長さ (桁数) の決定

パック 10 進数フィールドの長さ (桁数) を判別するには、次の公式を使用してください。

```
桁数 = 2n - 1
```

... n は、パック入力レコードで使用される桁数です。

この公式では、パック 10 進数形式で表すことができる最大桁数が示されます。上限は 30 です。

パック 10 進数フィールドは、最大 16 バイトの長さにすることができます。 表 17 は、最大 30 桁の長さのゾーン 10 進数フィールドがパックの場合にどのようになるかを示しています。

表 17. 最大 30 桁の長さのゾーン 10 進数フィールドをパックした結果

ゾーン 10 進数の長さ (桁数)	パック 10 進数フィールドで使用されるバイト数
1	1
2, 3	2
4, 5	3
. . . . . .	. . .
28, 29	15
30	16

注: 30 桁だけが使用可能です。

16 バイトのパック 10 進数フィールドに位取り表記法を使用する場合には、PACKEVEN キーワードを使用しなければなりません。そうでない場合には、フィールドを 30 桁をもつものとして定義しなければなりません。

たとえば、パック 10 進数形式で読み取られる入力フィールドは 5 バイトの長さをもちます (入力仕様またはデータ記述仕様で指定)。このフィールドの桁数は、2(5) – 1 すなわち 9 に等しくなります。したがって、フィールドを演算仕様で使用する場合には、結果フィールドは 9 桁の長さでなければなりません。定義仕様の PACKEVEN キーワードを使用すれば、桁数ではなく、開始位置および終了位置を使用してパック 10 進数フィールドを指定する時に、可能な 2 つのサイズのうちのどちらが必要であるかを示すことができます。

符号なし形式

符号なし整数形式は、整数形式と似ていますが、値の範囲に負数が含まれないという点が異なります。符号なし形式は、負数でない整数データが必要な場合にのみ使用してください。

符号なしフィールドは、適切な仕様の「データ・タイプ」項目に U を指定することによって定義します。パラメーターが符号なしフィールドとなっている定義仕様で LIKE キーワードを使用しても符号なしフィールドを定義することができます。

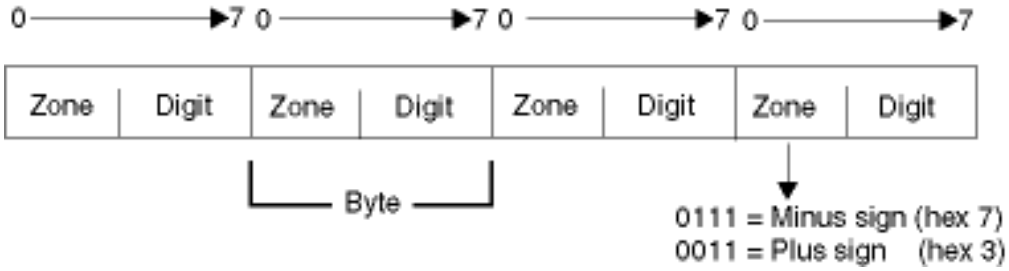
符号なしフィールドの長さは、桁数で定義し、3、5、10、または 20 桁の長さにすることができます。3 桁フィールドは 1 バイトのストレージを占め、5 桁フィールドは 2 バイトのストレージを占め、10 桁フィールドは 4 バイトのストレージを占め、20 桁フィールドは 8 バイトのストレージを占めます。符号なしフィールドの値の許容範囲は、その長さによります。

フィールド長	使用可能な値値の範囲
3 桁符号なし	0 - 255
5 桁符号なし	0 - 65535
10 桁符号なし	0 - 4294967295
20 桁符号なし	0 - 18446744073709551615

位置合わせを含めた符号なしフィールドの使用に関するその他の考慮事項については、 132 ページの『整数形式』を参照してください。

ゾーン 10 進数形式

ゾーン 10 進数形式は、ストレージの各バイトが 1 つの数字または 1 つの文字を含むことができるということを意味します。ゾーン 10 進数では、ストレージの各バイトが 4 ビットのゾーン部分と 4 ビットの数字部分という 2 つの部分に分かれています。ゾーン 10 進数形式は次のようになっています。



右端バイトのゾーン部分は、10 進数の符号 (正または負) を示します。すべてのゾーン 10 進数では優先符号 (正数には 16 進数 3、負数には 16 進数 7) が使用されます。さらに、次の符号がサポートされています。正数には 16 進数 0、1、2、8、9、A、B、負数には 16 進数 4、5、6、C、D、E、F。ゾーン 10 進数では、10 進数の各数字にはゾーン部分があります。ただし右端のゾーン部分は符号として働きます。 130 ページの図 37 では、ゾーン 10 進数形式で 8191 の数がどのようになるかを示しています。

出力仕様の 40 ～ 43 桁目に「終わりの位置」をコーディングし、フィールドがパック形式で出力される時には、フィールド長の変化を考慮しなければなりません。パック後のフィールドの長さを判別するには、次の公式を使用してください。

$$\text{Field length} = \frac{n}{2} + 1$$

. . . where n = number of digits in the zoned decimal field.

(Any remainder from the division is ignored.)

プログラム記述ファイルの場合、ゾーン 10 進数形式は、入力仕様の 36 桁目、出力仕様の 52 桁目、あるいは定義仕様の 40 桁目をブランクにすることによって指定します。外部記述ファイルの場合には、このデータ形式は、データ記述仕様の 35 桁目に指定します。

ゾーン 10 進数形式では代替符号形式を指定することができます。代替符号形式では、数値フィールドの直前または直後に + 符号または – 符号が付いています。正符号は 16 進数 2B で、負符号は 16 進数 2D です。

代替符号形式を指定する時には、フィールド長 (入力仕様で指定) には符号のための追加の位置を含めなければなりません。たとえば、フィールドが 5 桁の長さで、代替符号形式を指定する場合には、6 桁のフィールド長を指定しなければなりません。

数値形式を使用する場合の考慮事項

数値フィールドを定義する時には、以下のことに留意してください。

- 出力仕様の 47 - 51 桁目に「終わりの位置」をコーディングする場合、出力フィールドによって占められるバイト数を計算する時には、外部形式を使用してください。たとえば、5 桁のパック・フィールドは 3 バイトに保管されますが、出力がゾーン形式である時には、5 バイトが必要です。出力が整数形式である時には、2 バイトだけで済みます。
- 文字フィールドをゾーン数値フィールドに移動する場合には、文字フィールドの符号がゾーン正数またはゾーン負数に固定されます。その他のバイトのゾーン部分は、強制的に '3' になります。しかし、文字フィールドのバイトの 1 つの数字部分に有効数字が含まれていない場合には、10 進数データ・エラーが起こります。
- 数値フィールドが編集なしで書き込まれる時には、符号は区切り文字としては印刷されず、数字の最後の桁に符号が含まれます。これは驚くべき結果を生じることがあります。たとえば、-625 が書き出される時には、ゾーン 10 進数値 XX'363275' となります。これは 62u として現われます。
- デフォルトは、4 バイト演算の実行です。コンパイラーが 8 バイト演算を実行するのは、少なくとも 1つのオペランドが 8 バイト整数である場合だけです。2 つの 4 バイト整数が 8 バイトの結果を生じるような算術演算では、オーバーフローという実行時エラーが起こることがあります。この問題を避けるには、1 つのオペランドを 8 バイトにしてください。

フィールドの数値形式を選択するための指針

次の場合には、フィールドに整数形式または符号なし形式を指定してください。

- 演算のパフォーマンスが重要である。

  一定の算術演算では、使用する値を整数とすることが重要になる場合があります。パフォーマンスの改善があるとみられる一部の例では、配列指標の計算および組み込み関数 %SUBST の引き数が含まれる場合があります。
- デフォルトは、4 バイト演算の実行です。コンパイラーが 8 バイト演算を実行するのは、少なくとも 1つのオペランドが 8 バイト整数である場合だけです。パフォーマンスの面から見れば、8 バイト演算は費用がかかるので、避けるようにしてください。
- ILE C のような整数データ・タイプをサポートするその他の言語で書かれたルーチンと対話する。

- 整数として定義され、9999 または 999999999 を超える値が含まれる可能性があるフィールドをファイル・フィードバック域で使用する。

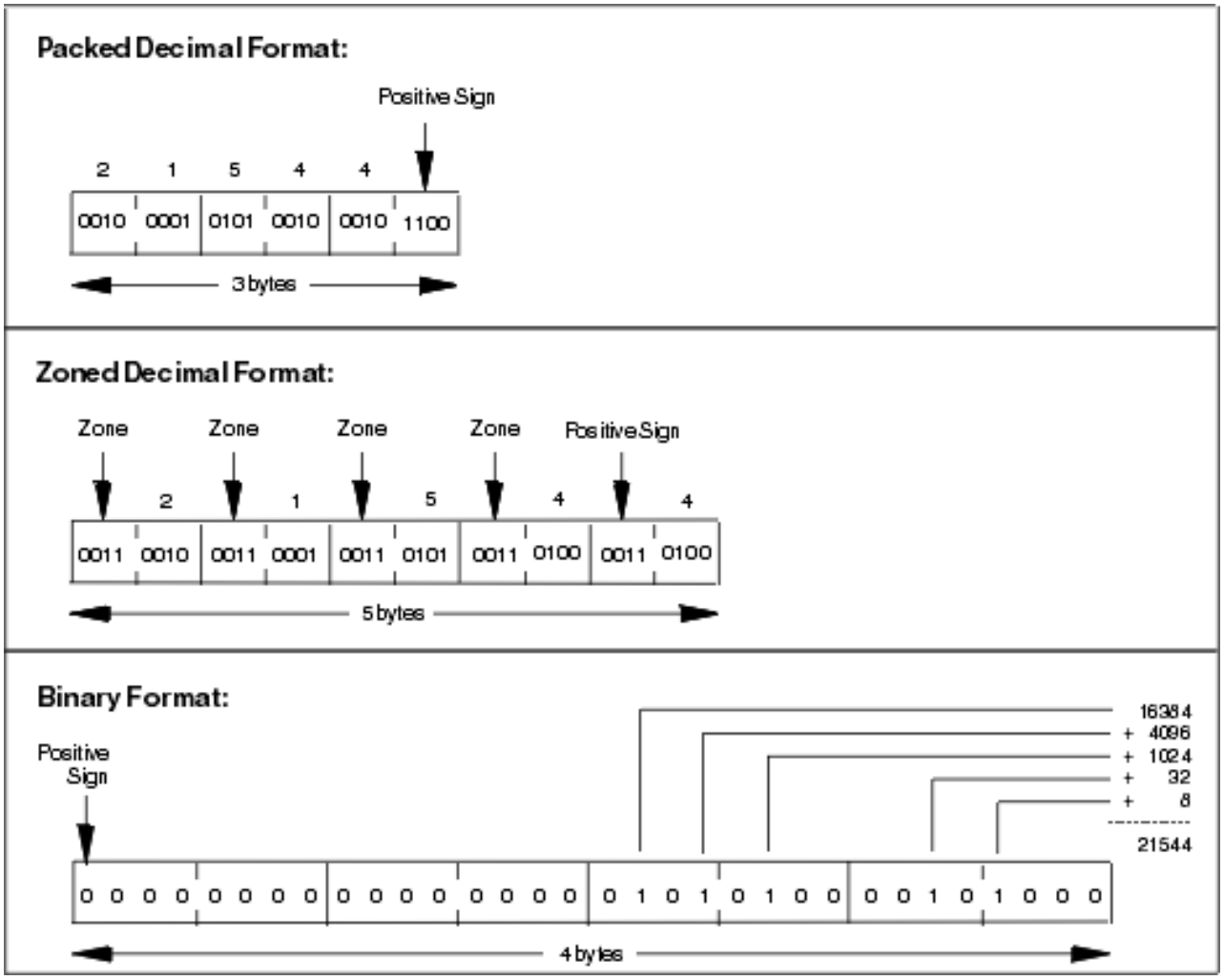
次の場合には、フィールドにパック形式、ゾーン形式、および 2 進数形式を指定してください。
- 通貨値など、小数点以下の桁数が暗黙指定される値を使用する。
- 19 桁を超える値を処理する
- フィールドに特定の桁数を確保することが重要である

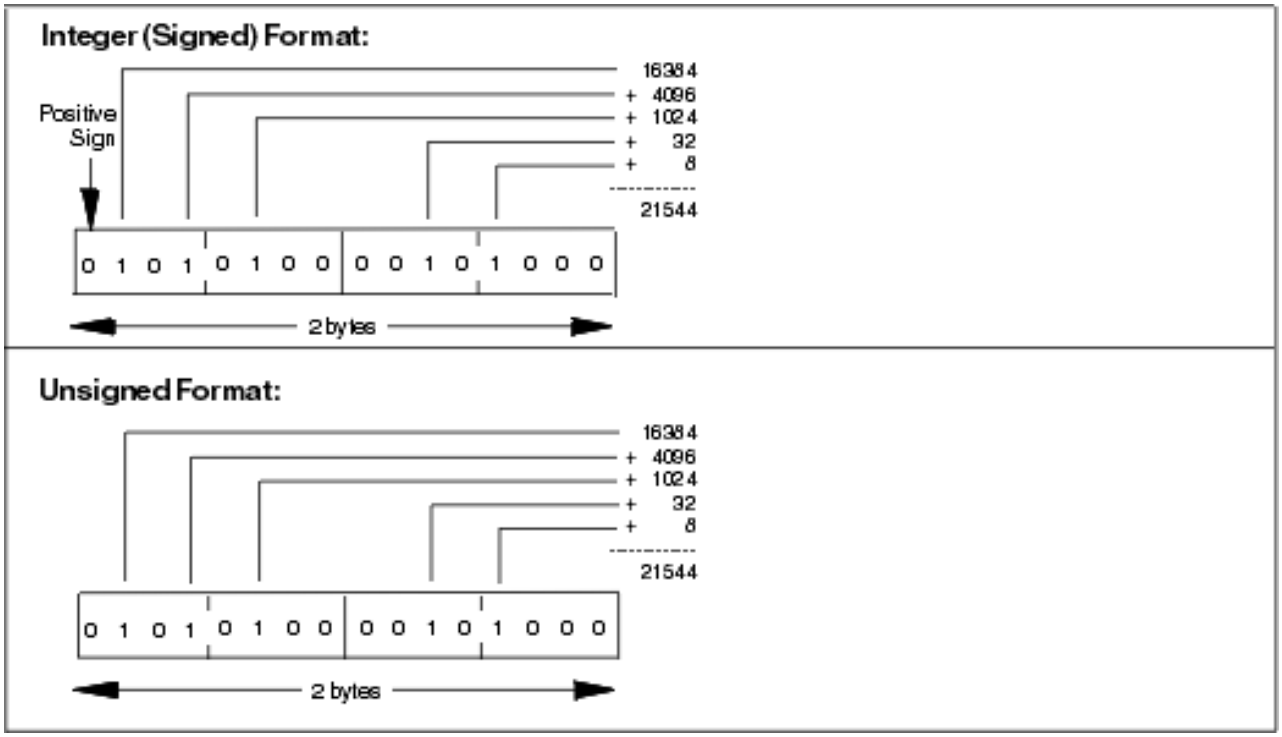
次の場合には、フィールドに浮動形式を使用してください。
- パックまたはゾーン値で表すことができない非常に小さな値または非常に大きな値 (あるいはその両方)を同じ変数に保持する必要がある。

注: 整数形式または符号なし形式を使用して実行される算術演算、特に整数演算がフリー・フォームの式で行なわれる時には、オーバーフローが起こる可能性がより高くなります。これは、中間結果が十分なサイズの一時 10 進数フィールドではなく整数形式または符号なし形式で保存されるためです。

数値形式の表現

次の図は、各種の形式で 10 進数 21544 がどのようになるかを示したものです。


Packed Decimal Format:
Positive Sign
2 1 5 4 4
0010 0001 0101 0010 0010 1100
3 bytes
Zoned Decimal Format:
Zone Zone Zone Zone Positive Sign
2 1 5 4 4
0011 0010 0011 0001 0011 0101 0011 0100 0011 0100
5 bytes
Binary Format:
Positive Sign
16384
+ 4096
+ 1024
+ 32
+ 8
21544
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 1 0 0 0
4 bytes



図中の表現について次の点に注意してください。

- 正の 2 進数または整数の数値 (符号なしの数値) を得るには、オン (1) のビットの値を加算しますが、符号ビット (ある場合) は含めません。符号なしの数値の場合、左端のビットを含めオンになっているビットの値を加算します。
- 値 21544 は、下位 2 バイトのビットしか使用しない場合であっても、2 バイトの 2 進数フィールドで表現することはできません。2 バイトの 2 進数フィールドは 4 桁しか保持することができませんが、21544は 5 桁です。

図 38 は、数値 -21544 を整数形式で示しています。

図 38. 数値 -21544 の整数表現

注: ワークステーション・アーキテクチャーでは、2 進整数、整数形式、および符号なし形式は、プログラム・メモリー中でバイトが逆になった順序で保管されます。この記憶メカニズムは、これらの形式でサブフィールドをオーバーレイするために使用される文字サブフィールドの値に影響します。

オブジェクト・データ・タイプ

オブジェクト・データ・タイプによって、Java オブジェクトを定義することができます。オブジェクト・データ・タイプは、以下のように指定します。

```
 * 変数 MyString は、Java ストリング・オブジェクトです。
D MyString        S               O   CLASS(*JAVA
D                                        :'java.lang.String')
```

あるいは以下のように指定します。

```
D bdcreate        PR              O   EXTPROC(*JAVA
D                                          :'java.math.BigDecimal'
D                                          :*CONSTRUCTOR)
```

40 桁目では、データ・タイプ O を指定します。キーワード・セクションでは、CLASS キーワードを指定して、オブジェクトのクラスを示します。環境に *JAVA を指定し、クラス名を指定してください。

オブジェクトが Java コンストラクターの戻りタイプである場合には、戻されるオブジェクトのクラスはメソッドのクラスと同じなので、CLASS キーワードを指定しません。代わりに、環境 *JAVA、クラス名、およびプロシージャー *CONSTRUCTOR で EXTPROC キーワードを指定します。

オブジェクトは、基底付けできません。データ構造のサブフィールドすることもできません。

オブジェクトが配列またはテーブルである場合には、実行時にロードしなければなりません。タイプがオブジェクトである実行前および実行時配列およびテーブルは、使用できません。

すべてのオブジェクトは *NULL に初期化されます。これは、オブジェクトがそのクラスのインスタンスと関連付けられていないということを意味します。

オブジェクトの内容を変更するには、メソッド呼び出しを使用しなければなりません。オブジェクトによって使用されるストレージに直接にアクセスすることはできません。

クラスは、実行時に解決されます。コンパイラーは、クラスが存在することも、そのクラスが他のオブジェクトと互換性があることも検査しません。

オブジェクト・フィールドを指定できる場合

オブジェクト・フィールドは、以下の場合に使用することができます。

フリー・フォーム評価

EVAL 演算命令を使用して、1 つのオブジェクト項目 (フィールドまたはプロトタイピングされたプロシージャー) をタイプがオブジェクトのフィールドに割り当てることができます。

フリー・フォーム比較

1 つのオブジェクトを別のオブジェクトと比較できます。どの比較でも指定できますが、以下の比較しか意味がありません。

- 別のオブジェクトと等しいか、等しくない。2 つのオブジェクトが等しいのは、それらがまったく同じオブジェクトを表す場合だけです。同じ値をもつ 2 つの異なるオブジェクトは、等しくありません。

  2 つのオブジェクトの値が等しいことをテストしたい場合には、Java の 'equals' メソッドを以下のように使用します。

```
D objectEquals    PR              N   EXTPROC(*JAVA
D                                     : 'java.lang.Object'
D                                     : 'equals')
C                   IF        objectEquals (obj1 : obj2)
C                    ...
C                   ENDIF
```

- *NULL と等しいか、等しくない。オブジェクトが *NULL と等しいのは、そのオブジェクトがそのクラスの特定のインスタンスと関連付けられていない場合です。

フリー・フォーム呼び出しパラメーター

プロトタイプ中のパラメーターがオブジェクトである場合には、オブジェクトを呼び出し演算命令のパラメーターとしてコーディングできます。

注:

1. オブジェクトは、入力フィールドまたは出力フィールドとしては無効です。
2. 割り当ての妥当性は検査されません。たとえば、RPG では、クラス Number のオブジェクトをクラス String で定義されたオブジェクト変数に割り当てることができます。これが正しくなかった場合、String 変数を使用しようとすると、Java エラーが起こります。

```
D Obj             S               O   CLASS(*JAVA
D                                         :'java.lang.Object')
D Str             S               O   CLASS(*JAVA
D                                         :'java.lang.String')
D Num             S               O   CLASS(*JAVA
D                                         :'java.math.BigDecimal')

 * すべての Java クラスがクラス 'java.lang.Object' のサブクラス
 * である場合には、どのオブジェクトもこのクラスの変数に割り当てる
 * ことができます。
 * 以下の 2 つの割り当ては有効です。
C                   EVAL      Obj = Str
C                   EVAL      Obj = Num
 * しかし、Str を Num に割り当てるのはおそらく無効です。
```

図 39. オブジェクト・データ・タイプの例

プロシージャー・ポインターのデータ・タイプ

プロシージャー・ポインターは、プロシージャーまたは機能を指示するために使用されます。プロシージャー・ポインターは、プログラムに結び付けられる入り口点を示します。プロシージャー・ポインターは、定義仕様で定義します。

プロシージャー・ポインター・フィールドの長さは 4 バイトでなければならず、また 4 バイト境界に位置合わせされなければなりません。境界合わせのこの要件のた

め、データ構造のポインター・サブフィールドが先行フィールドの直後に続かなくなることも、複数回繰り返しデータ構造が不連続のオカレンスをもつようになることもあります。プロシージャー・ポインターのデフォルトの初期設定値は *NULL です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D*
D* 基底ポインター・フィールドを定義し、データ構造
D* My_Struct のアドレスに初期化します
D*
D  My_struct      DS
D   My_array                    10    DIM(50)
D
D Ptr1            S               4*   INZ(%ADDR(My_Struct))
D*
D* あるいはその代わりに、長さが定義されていなければ、デフォルトの長さ 4 を使用します
D*
D Ptr1            S                *   INZ(%ADDR(My_Struct))
D*
D* プロシージャー・ポインター・フィールドを定義し、ヌル値に初期化します
D*
D Ptr1            S               4*   PROCPTR INZ(*NULL)
D*
D* プロシージャー・ポインター・フィールドを定義し、プロシージャー
D* My_Proc のアドレスに初期化します
D*
D Ptr1            S               4*   PROCPTR INZ(%PADDR(My_Proc))
D*
D* 複数回繰り返しデータ構造中にポインターを定義し、ストレージを
D* 正確にマップします。
D*
DDataS            DS                   OCCURS(2)
D ptr1                             *
D ptr2                             *
D Switch                          1A
D*
D* ストレージ・マップは次のようになります。
D*
D*               DataS
```

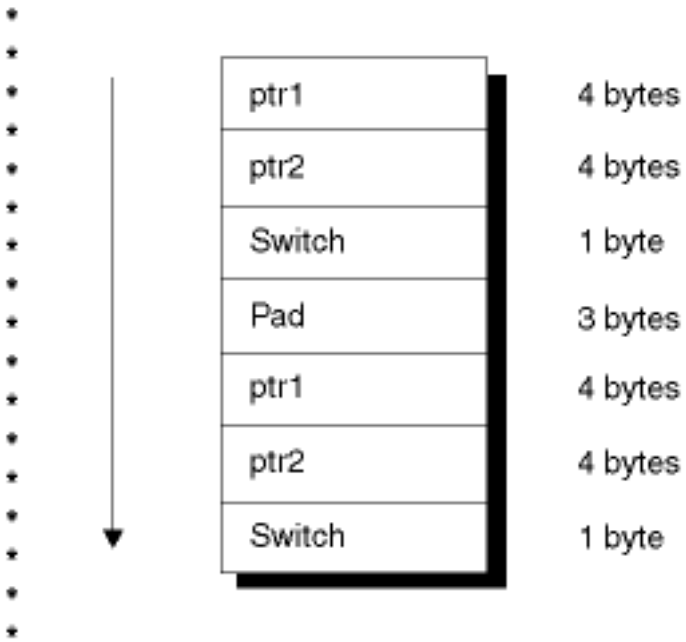


図 40. ポインターの定義

時刻データ

時刻フィールドは、事前に決定されたサイズおよび形式をもちます。これは、定義仕様で定義することができます。時刻データにはすべて先行ゼロおよび後続ゼロが必要です。

比較または割り当てで使用される時刻固定情報または時刻変数は、同じ形式である必要も、同じ区切り文字を使用する必要もありません。入力フィールド、出力フィールド、またはキー・フィールドが演算命令に必要な形式に変換されるような入出力演算命令には、時刻が使用されます (必要な場合)。

時刻変数のデフォルトの内部形式は、*ISO です。このデフォルトの内部形式は、制御仕様キーワードの TIMFMT によってグローバルに指定変更することも、定義仕様キーワードの TIMFMT によって個別に指定変更することもできます。

時刻フィールドの内部時刻形式および区切り文字を決定する時に使用される階層は、次の通りです。
1. 定義仕様に指定された TIMFMT キーワードから
2. 制御仕様に指定された TIMFMT キーワードから
3. *ISO

時刻フィールドのコーディング方法の例については、次の資料の例を参照してください。
- 363 ページの『日付の演算』
- 374 ページの『日付・時刻データの移動』
- 482 ページの『ADDDUR (期間の加算)』
- 587 ページの『MOVE (転送)』
- 683 ページの『SUBDUR (期間の減算)』
- 690 ページの『TEST (日付、時刻、タイム・スタンプのテスト)』

次の表には、時刻データの形式がリストされています。

表 18. 時刻データ・タイプの時刻形式および区切り文字

形式名	説明	デフォルトの区切り文字付き形式	有効な区切り文字	長さ	例
*HMS	時:分:秒	hh:mm:ss	: . , &	8	14:00:00
*ISO	国際標準化機構	hh.mm.ss	.	8	14.00.00
*USA	IBM USA 標準規格。AM および PM は、大文字小文字の混合とすることができます。	hh:mm AM または hh:mm PM	:	8	02:00 PM
*EUR	IBM 欧州標準規格	hh.mm.ss	.	8	14.00.00
*JIS	日本工業規格西暦	hh:mm:ss	:	8	14:00:00

次の表には、すべての日付形式の *LOVAL、*HIVAL、およびデフォルト値がリストされています。

表 19. 時刻値

形式名	説明	*LOVAL	*HIVAL	デフォルト値
*HMS	時:分:秒	00:00:00	24:00:00	00:00:00
*ISO	国際標準化機構	00.00.00	24.00.00	00.00.00
*USA	IBM USA 標準規格。AM および PM は、大文字小文字の混合とすることができます。	00:00 AM	12:00 AM	00:00 AM
*EUR	IBM 欧州標準規格	00.00.00	24.00.00	00.00.00
*JIS	日本工業規格西暦	00:00:00	24:00:00	00:00:00

区切り文字

MOVE、MOVEL、または TEST 演算命令で時刻形式をコーディングする時には、文字フィールドでは区切り文字は任意指定です。区切り文字がないことを示すには、後にゼロが続く形式を指定します。区切り文字のない時刻形式のコーディング方法の詳細については、 587 ページの『MOVE (転送)』を参照してください。

タイム・スタンプ・データ

タイム・スタンプ・フィールドは、事前に決定されたサイズおよび形式をもちます。これは、定義仕様で定義することができます。タイム・スタンプ・データは次の形式でなければなりません。

```
yyyy-mm-dd-hh.mm.ss.mmmmmm (26 桁)
```

タイム・スタンプ・リテラルではマイクロ秒 (.mmmmmm) は任意指定であり、指定されていない場合には、右側にゼロが埋め込まれます。タイム・スタンプ・データにはすべて先行ゼロが必要です。

タイム・スタンプのデフォルトの初期設定値は 0001 年 1 月 1 日深夜 (0001-01-01-00.00.00.000000) です。タイム・スタンプの *HIVAL 値は 9999-12-31-24.00.00.000000 です。同様に、タイム・スタンプの *LOVAL 値は 0001-01-01-00.00.00.00000 です。

区切り文字

MOVE、MOVEL、または TEST 演算命令でタイム・スタンプ形式をコーディングする時には、文字フィールドでは区切り文字は任意指定です。区切り文字がないことを示すには、*ISO0 を指定してください。区切り文字なしに *ISO を使用する方法の例については、 690 ページの『TEST (日付、時刻、タイム・スタンプのテスト)』 を参照してください。

データベースのヌル値サポート

VisualAge RPG プログラムでは、外部記述データベース・ファイルからのヌル値使用可能フィールドを処理するのに 3 つの異なる方法のうちの 1 つを選択することができます。その選択は、*ヌル値可能* オプションまたは ALWNULL 制御仕様キーワードをどのように指定しているかによります。

1. **ユーザー制御**、ALWNULL(*USRCTL) - ヌル値をもつレコードを読み取り、書き込み、更新、および削除し、ヌルキーをもつレコードを検索して、それに位置づけます。
2. **入力専用**、ALWNULL(*INPUTONLY) - ヌル値をもつレコードを読み取り、ヌルフィールド中のデータにアクセスします
3. **使用不可**、ALWNULL(*NO)- ヌル値をもつレコードを処理しません

注: プログラム記述ファイルの場合、*ヌル値可能* オプションまたは ALWNULL キーワードに指定された値に関係なく、レコード中のヌル値は常にデータ・マッピング・エラーの原因となります。

コンパイラー・オプションの指定の詳細については、*WebSphere Development Studio Client for iSeries ご使用に際して*, SD88-5054-03 を参照してください。

ヌル値使用可能フィールドおよびキー・フィールドのユーザー制御サポート

外部記述ファイルにヌル値使用可能フィールドが含まれていて、**ユーザー制御**オプションまたは ALWNULL(*USRCTL) オプションを指定した場合には、次のことを行なうことができます。

- 外部記述データベース・ファイルのヌル値をもつレコードを読み取り、書き込み、更新、および削除する。
- フィールドに関連した KFLD の演算項目 2 に標識を指定することによって、キーによる演算命令を使用してヌルキーをもつレコードを検索して、それに位置付ける。
- 式の右側で %NULLIND 組み込み関数を使用して、ヌル値使用可能フィールドが実際にヌルであるかどうかを判別する。
- 式の左側で %NULLIND 組み込み関数を使用して、ヌル値使用可能フィールドを出力または更新のためにヌルに設定する。

ヌル値を含むフィールドがプログラム内で正しく使用されるようにするのは、ユーザーの責任です。 MOVE演算命令の演算項目 2 としてヌル値使用可能フィールドを使用する場合には、MOVE を行なう前に、それがヌル値であるかどうかを最初にチェックしなければなりません。さもないと、結果フィールドの値が壊れることがあります。また、PRINTER ファイルやプログラム記述ファイルなど、ヌル値使用可能として定義されているフィールドをもたないファイルに対してヌル値使用可能フィールドを出力する場合にも注意が必要です。

注: ヌル値使用可能フィールドのヌル標識の値は、入力、出力、ファイル位置決めなどの演算命令についてだけ考慮されます。ここには、ヌル標識が考慮されない演算命令のいくつかを示してあります。
- ヌル標識がオンであっても、ヌル値使用可能フィールドの DSPLY はフィールドの内容を表示します。
- ヌル値使用可能フィールドを別のヌル値使用可能フィールドに移動し、演算項目 2 フィールドにオンのヌル標識がある場合には、結果フィールドには演算項目 2 フィールドからデータが取り出されます。結果フィールドに対応するヌル標識は、オンに設定されません。
- SORTA および LOOKUP を含め、ヌル値使用可能フィールドとの比較演算では、ヌル標識は考慮されません。

フィールドが外部記述データベース・レコードでヌル値使用可能であり、プログラム中で固定情報として定義されていない場合には、そのフィールドはヌル値使用可能と見なされます。

注: 外部記述データ構造に使用されるファイルにヌル値使用可能フィールドが定義されている場合には、ヌル属性は、VARPG サブフィールドの定義には使用されません。

フィールドが VARPG プログラムでヌル値使用可能と見なされると、ヌル標識がそのフィールドに関連付けられます。次のことに注意してください。

- フィールドが複数発生データ構造またはテーブルである場合には、ヌル標識の配列がそのフィールドに関連付けられます。各ヌル標識は、データ構造中のオカレンスまたはテーブルのエレメントに対応しています。
- フィールドが配列エレメントである場合には、配列全体がヌル値使用可能と見なされます。ヌル標識の配列が配列に関連付けられ、各ヌル標識が配列エレメントに対応します。
- フィールドが複数発生データ構造の配列サブフィールドのエレメントである場合には、ヌル標識の配列がそのデータ構造の各オカレンスの配列に関連付けられます。

ヌル標識はプログラム初期化中にゼロに初期化されるので、プログラムが実行を開始する時点では、ヌル値使用可能フィールドにはヌル値が含まれていません。

ヌル値使用可能フィールドの入力

RPG プログラム中でヌル値使用可能であるフィールドの場合、DISK および SPECIAL ファイルについて入力で次のことが適用されます。

- ヌル値使用可能フィールドが外部記述ファイルから読み取られると、そのフィールドがレコード中でヌルである場合には、そのフィールドのヌル標識がオンに設定されます。そうでない場合には、ヌル標識はオフに設定されます。
- フィールド標識が指定され、ヌル値使用可能フィールドがヌルである場合には、すべてのフィールド標識がオフに設定されます。
- フィールドが 1 つのファイルでヌル値使用可能として定義され、別のファイルではヌル値使用可能でないと定義されている場合には、RPG プログラムではそのフィールドはヌル値使用可能と見なされます。しかし、2 番目のファイルを読み取る時には、そのフィールドに関連したヌル標識は常にオフに設定されます。
- 結果フィールドにデータ構造を使用したプログラム記述ファイルからの入力演算命令は、そのデータ構造またはそのサブフィールドに関連したヌル標識には影響を及ぼしません。
- プログラム記述ファイルの入力仕様を使用してヌル値使用可能フィールドを読み取ると、関連したヌル標識が常にオフに設定されます。
- フィールドとレコードの関連標識のためにヌル値使用可能フィールドが読み取られるように選択されていない場合には、関連したヌル標識は変更されません。

ヌル値使用可能フィールドの出力

ヌル値使用可能フィールドが外部記述ファイルに書き込まれる (出力または更新される) 時には、そのフィールドのヌル標識がその演算命令の時点でオンであれば、ヌル値が書き出されます。

ヌル値使用可能フィールドが外部記述データベース・ファイルに出力される時、あるいは外部記述データベース・ファイルで更新される時には、フィールドがヌルであれば、バッファーに入れられた値は、データ管理機能によって無視されます。

注: 出力の時点でオンのヌル標識をもつフィールドのデータはバッファーに移動されています。これは、フィールドのヌル標識がオンであると、10 進数データ・エラーや、「基底ポインターが設定されていない」などというエラーが起こるということを意味します。

外部記述データベース・ファイルに対する出力演算命令の時に、プログラム中ではヌル値使用可能と見なされ、ファイル中ではヌル値使用可能と見なされないフィールドがファイルに含まれている場合には、そうしたヌル値使用可能フィールドに関連したヌル標識は使用されません。

149 ページの図 41 は、**ユーザー制御**オプションまたは ALWNULL(*USRCTL)キーワードを選択した時に、ヌル値をもつレコードの読み取り、書き込み、および更新を行なう方法を示したものです。

```
H*
H*  制御仕様で ALWNULL(*USRCTL) キーワードを指定するか、あるいは
H*  VARPG プログラムをユーザー制御オプションでコンパイルして
H*  ください。
H*
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H*      H ALWNULL(*USRCTL)
F*
F* DISKFILE には FLD1 と FLD2 という 2 つのフィールドをもつレコード REC
F* が含まれていて、FLD1 と FLD2 はともにヌル値使用可能です。
F*
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++
F*
FDISKFILE  UF A E             DISK
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
C*
C* 最初のレコードを読み取ります。
C* レコードをヌルでないいずれかのフィールドの新規値で更新します。
C                   READ      REC                                    10
C                   IF        NOT %NULLIND(Fld1)
C                   MOVE      'FLD1'        Fld1
C                   ENDIF
C                   IF        NOT %NULLIND(Fld2)
C                   MOVE      'FLD2'        Fld2
C                   ENDIF
C                   UPDATE    REC
C*
C* 別のレコードを読み取ります。
C* すべてのフィールドがヌルとなるように、レコードを更新します。
C* これらのフィールドは無視されるので、その値を設定する必要はありません。
C                   READ      REC                                    10
C                   EVAL      %NULLIND(Fld1) = *ON
C                   EVAL      %NULLIND(Fld2) = *ON
C                   UPDATE    REC
C*
C* Fld 1 がヌルで、Fld 2 がヌルでない新規レコードを書き込みます。
C*
C                   EVAL      %NULLIND(Fld1) = *ON
C                   EVAL      %NULLIND(Fld2) = *OFF
C                   EVAL      Fld2 = 'New value'
C                   WRITE     REC
```

図 41. ヌル値使用可能フィールドの入力および出力

キーによる演算命令

ヌル値使用可能キー・フィールドがある場合には、KFLD 演算命令の演算項目 2 に標識を指定し、その標識をキーによる入力演算命令の前にオンに設定することによって、ヌル値を含むレコードを検索することができます。ヌルキーを選択したくない場合には、標識をオフに設定します。

150 ページの図 42 は、ヌルキーをもつレコードの位置決めおよび検索を行なうために、キーによる演算命令がどのように使用されるかを示したものです。

```
 *
 * 下の File1 には、Key1、Key2、Key3 という 3 つのキー・フィールドから
 * 構成される複合キーをもつレコード Rec1 が含まれていると想定します。
 * Key2 と Key3 はヌル値使用可能です。Key1 はヌル値使用可能でありません。
 * 各キー・フィールドは 2 文字の長さです。
 *
 *..1....+....2....+....3....+....4....+....5....+....6....+....7....+..
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
F*
FFile1     IF   E             DISK
F*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
C*
C     Full_Kl       KLIST
C                   KFLD                    Key1
C                   KFLD      *IN02         Key2
C                   KFLD      *IN03         Key3
C*
C     Partial_Kl    KLIST
C                   KFLD                    Key1
C                   KFLD      *IN05         Key2
C*
C*
C* 下の SETLL 演算命令の場合、*IN02 が ON で、*IN03 が OFF になっています。
C* File1 は、'AA??CC' (?? は、この例ではヌル値を示すために使用されています)
C* に等しいか、それより大きいキーをもつ次のレコードに位置付けられます。
C*
C* *IN02 が ON であるので、Key2 の検索引き数の実際の内容は無視されます。
C*
C* File1 に Key1 が 'AA' で、Key2 がヌル値で、Key3 が 'CC' であるレコード
C* が存在した場合には、標識 90 (Eq 標識) がオンに設定されます。
C*
C                   MOVE      'AA'          Key1
C                   MOVE      'CC'          Key3
C                   EVAL      *IN02 = '1'
C                   EVAL      *IN03 = '0'
C     Full_Kl       SETLL     Rec1                                   90
C*
```

図 42. ヌル値使用可能キー・フィールドを使用するキーによる演算命令の例 (1/2)

```
C*
C* 下の CHAIN 演算命令では、Key1 が 'JJ' で、Key2 が 'KK' で、Key1 が
C* ヌル値のレコードが検索されます。この場合も *IN03 が ON であるので、
C* プログラマーがある種の値 (たとえば 'XX') を Key3 の検索引き数に移動
C* しても、'XX' は使用されません。これは、File1 にキー 'JJKKXX' をもつ
C* レコードが実際にあった場合に、そのレコードが検索されないということ
C* を意味します。
C*
C                   MOVE      'JJ'          Key1
C                   MOVE      'KK'          Key2
C                   EVAL      *IN02 = '0'
C                   EVAL      *IN03 = '1'
C     Full_Kl       CHAIN     Rec1                              80
C*
C*
C* 下の CHAIN 演算命令では検索引き数として部分キーが使用されます。
C* これは、Key1 が 'NN' で、Key2 がヌル値で、Key3 がヌル値を含む任意の
C* 値であるレコードを検索します。
C*
C* データベース中では、ヌル値が照合順序中の最高の位置を占めます。File1
C* 中にキーが昇順になっていると想定してください。File1 にキーとして
C* 'NN??xx' (?? はヌル値を意味し、xx はヌル値以外の値を意味します) をもつ
C* レコードがある場合には、そのレコードが検索されます。このような
C* レコードが File1 に存在しないが、File1 にキーとして 'NN????' を
C* もつレコードがある場合には、'NN????' レコードが検索されます。結果
C* として Key2 と Key3 のヌルフラグが ON に設定されます。
C*
C                   MOVE      'NN'          Key1
C                   SETON                                       05
C     Partial_Kl    CHAIN     Rec1                              70
```

図 42. ヌル値使用可能キー・フィールドを使用するキーによる演算命令の例 (2/2)

ヌル値使用可能フィールドの入力専用サポート

外部記述入力専用ファイルにヌル値使用可能フィールドが含まれ、**入力専用**オプションまたは ALWNULL(*INPUTONLY) キーワードが指定されている時には、次の条件が適用されます。

- レコードがデータベース・ファイルから検索され、そのレコードの中の一部のフィールドにヌル値が含まれている時には、ヌル値使用可能フィールドのデータベースのデフォルト値がヌル値を含むそうしたレコードに入れられます。デフォルト値は、ユーザー定義 DDS デフォルト値またはシステム・デフォルト値となります。
- レコード中の一定のフィールドがヌル値をもっているかどうかを判別できなくなります。
- 入力フィールドが外部記述入力専用ファイルからのヌル値使用可能フィールドである場合には、入力仕様でフィールド標識は使用することができません。
- キーによる入力演算命令の演算項目 1 が外部記述入力専用ファイルのヌル値使用可能キー・フィールドに対応している時には、キーによる演算命令は使用することができません。

ヌルフィールド使用不可オプション

外部記述ファイルにヌル値使用可能フィールドが含まれていて、**使用不可**オプションまたは ALWNULL(*NO) キーワードが指定されている時には、次の条件が適用されます。

- ファイルからヌル値を含むレコードが検索されると、データ・マッピング・エラーが起こり、エラー・メッセージが出されることになります。
- レコード中のデータがアクセス不能で、レコード中のどのフィールドもヌル値を含む入力レコードからの値で更新することができなくなります。
- このオプションでは、レコードの更新または追加用にヌル値使用可能フィールドにヌル値を入れることはできません。ヌル値使用可能フィールドにヌル値を入れたい場合には、**ユーザー制御**オプションを使用してください。

データベース可変長フィールドの変換

VisualAge RPG コンパイラーは、外部記述ファイルまたはデータ構造からの可変長の文字フィールドまたはグラフィック・フィールドを固定長文字フィールドとして内部的に定義することができます。可変長の文字フィールドおよびグラフィック・フィールドを固定長形式に変換することが必要なくても、可変長フィールドがサポートされる前に作成されたプログラムをサポートするために、言語で CVTOPT コンパイラー・オプションが残っています。

CVTOPT 制御仕様キーワードに *VARCHAR (可変長文字フィールドの場合) または *VARGRAPHIC (可変長グラフィック・フィールドの場合) を指定することによって、可変長フィールドを変換することができます。*VARCHAR または *VARGRAPHIC が指定されなかった時、あるいは *NOVARCHAR または *NOVARGRAPHIC が指定された時には、可変長フィールドは固定長文字に変換されず、VisualAge RPG プログラム中で可変長として使用することができます。

*VARCHAR または *VARGRAPHIC が指定された時には、次の条件が適用されます。

- 外部記述ファイルまたは外部記述データ構造から可変長フィールドが抜き出された場合には、それは固定長文字フィールドとして宣言されます。
- 1 バイト文字フィールドの場合、宣言されたフィールドの長さは、DDS フィールドの長さに 2 バイトを加えたものになります。
- DBCS グラフィック・データ・フィールドの場合、宣言されたフィールドの長さは、DDS フィールドの長さを 2 倍して 2 バイトを加えたものになります。
- フィールド中の 2 つの余分なバイトには、可変長フィールドの現在の長さを表す 2 進数を含みます。 153 ページの図 43 には、可変長フィールドのフィールド長が示されています。
- 固定長文字フィールドとして定義された可変長グラフィック・フィールドの場合、長さはグラフィック文字の数を 2 倍したものです。

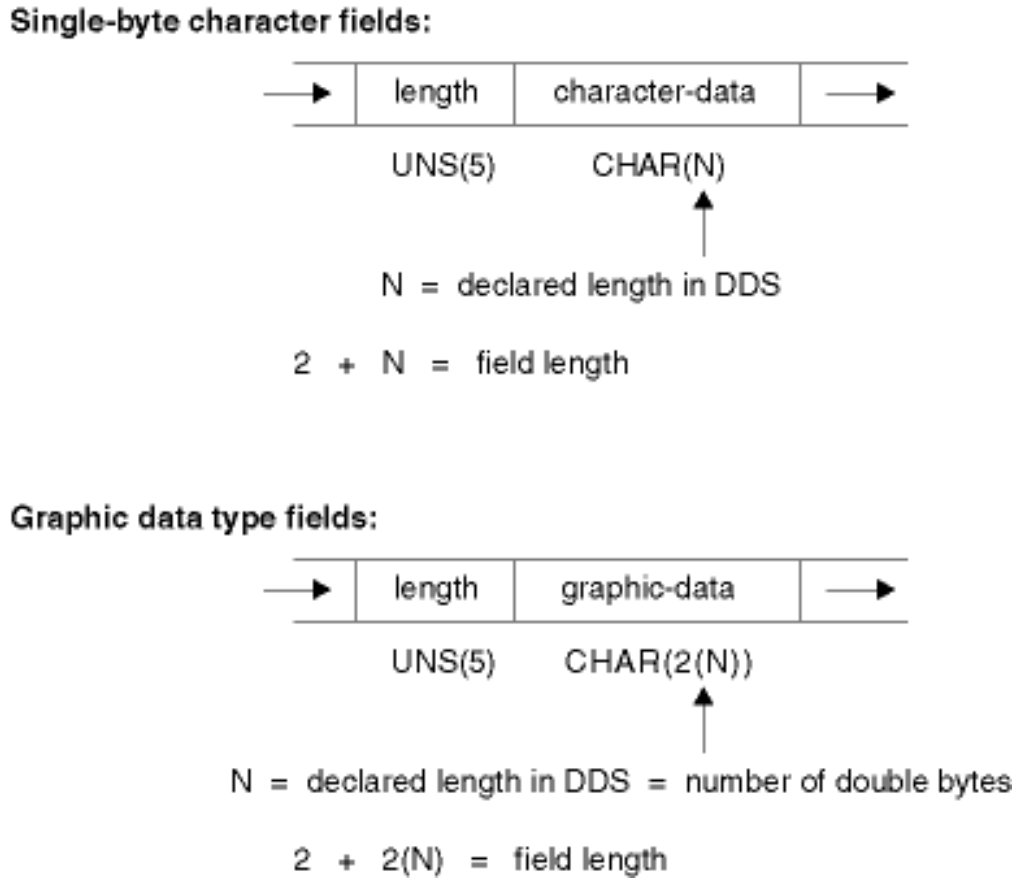


図 43. 変換後の可変長フィールドのフィールド長

- プログラムは、宣言された固定長フィールドに対して有効な文字演算命令を実行することができます。しかし、フィールドの構造のために、フィールドがファイルに書き込まれる時に、フィールドの最初の 2 バイトには、有効な符号なし整数データが含まれていなければなりません。フィールドの最初の 2 バイトに無効なフィールド長データが含まれている場合には、出力演算命令で入出力例外エラーが起こります。
- 可変長フィールドが OS/400 ファイルから GUI オブジェクトにインポートされ、ファイルが *VARCHAR または *VARGRAPHIC オプションの指定されたプログラムで外部記述ファイルとして使用される時には、コンパイル中にフィールド定義矛盾エラーが起こります。データ長のための 2 バイトがファイル・レコード様式から取られたフィールドの定義に追加されますが、これは GUI オブジェクトからのフィールド長定義と矛盾しています。

  この矛盾を回避するには、*VARCHARまたは *VARGRAPHIC オプションを指定しないか、あるいは GUI オブジェクトを名前変更して、適切な場合に 2 つのフィールド間でデータを移動するソース・コードを書いてください。
- 入力フィールドが外部記述入力ファイルからの可変長フィールドである場合には、入力仕様でフィールド標識を使用できません。
- キーによる演算命令の演算項目 1 が外部記述入力ファイルの可変長キー・フィールドに対応している時には、キーによる演算命令は使用することができません。
- レコード中の一定のフィールドを選択的に出力するように選択し、出力仕様で可変長フィールドが指定されていない場合、あるいはプログラム中で可変長フィー

ルドが無視された場合には、新しく追加されたレコードの出力バッファーにデフォルト値が入れられます。最初の 2 バイトではデフォルト値は 0、残りのすべてのバイトではブランクです。

- 変換後の可変長フィールドを変更したい場合には、現在のフィールド長が正しいことを確認してください。これを行なう 1 つの方法は次の通りです。
  1. 可変長フィールド名をサブフィールド名としてデータ構造を定義します。
  2. フィールドの先頭部分をオーバーレイする 5 桁の符号なし整数サブフィールドを定義し、3 桁目からフィールドをオーバーレイする N バイトの文字サブフィールドを定義します。
  3. フィールドを更新します。

この代わりに、別の可変長フィールドを左寄せにしてフィールドに移動することができます。変換後の可変長フィールドを VARPG プログラムで変更する方法の例は次の通りです。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*
A*   ファイル MASTER には、可変長フィールドが含まれています。
A*
AAN01N02N03T.Name++++++Rlen++TDpBLinPosFunctions++++++++++++++++++++
A*
A          R REC
A            FLDVAR       100          VARLEN
```

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7....+.. *
H*
H*  制御仕様で CVTOPT(*VARCHAR) キーワードを指定するか、あるいは
H*  コマンドに CVTOPT(*VARCHAR) を指定して VisualAge RPG プログラムを
H*  コンパイルしてください。
H*
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H*
H CVTOPT(*VARCHAR)
F*
F*   外部記述ファイルの名前は MASTER です。
F*
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++
F*
FMASTER    UF   E             DISK
```

図 44. プログラム中での可変長フィールドの変換 (1/2)

```
D*
D*   FLDVAR は、DDS に定義された 100 という DDS 長をもつ可変長フィールド
D*   です。VARPG フィールド長は 102 であるということに注意してください。
D*
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D*
D                 DS
D FLDVAR                  1    102
D   FLDLEN                       5U 0 OVERLAY(FLDVAR:1)
D   FLDCHR                     100    OVERLAY(FLDVAR:3)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C* 文字値が可変長フィールド FLDCHR に移動されます。
C* CHECKR 演算命令の後、FLDLEN は 5 という値をもちます。
C                   READ      MASTER                                 LR
C                   MOVEL     'SALES'       FLDCHR
C     ' '           CHECKR    FLDCHR        FLDLEN
C  NLR              UPDAT     REC
```

図 44. プログラム中での可変長フィールドの変換 (2/2)

変換済みの可変長グラフィック・フィールドが必要な場合には、長さを入れるための 2 バイトの符号なし整数フィールド、およびフィールドのデータ部分を入れるための長さ N のグラフィック・サブフィールドをコーディングすることができます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
D*
D*   可変長グラフィック・フィールド VGRAPH が長さ 3 として DDS に宣言
D*   されています。これは、フィールドの最大長が 2 バイト 3 つ、すなわち
D*   6 バイトであるということを意味します。長さ部分を数えたフィールド
D*   の合計長は 8 バイトです。
D*
D*   CVTOPT(*VARGRAPHIC) で VARPG プログラムをコンパイルします。
D*
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D*
D                 DS
DVGRAPH                     8
D  VLEN                     4U 0 OVERLAY(VGRAPH:1)
D  VDATA                    3G   OVERLAY(VGRAPH:3)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C*   GRPH が固定長グラフィック・フィールドであり、その長さ 2 バイト
C*   2 つ分である想定してください。GRPH を VGRAPH にコピーし、
C*   VGRAPH の長さを 2 に設定します。
C*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C                   MOVEL     GRPH          VDATA
C                   Z-ADD     2             VLEN
```

図 45. 可変長グラフィック・フィールドの変換

第 10 章 リテラルおよび名前付き固定情報

リテラルおよび名前付き固定情報は、固定情報の一種です。固定情報は、次のいずれかの場所に指定することができます。

- 演算項目 1
- 演算項目 2
- 演算仕様の拡張演算項目 2
- 制御仕様のキーワードに対するパラメーターとして
- 組み込み関数に対するパラメーターとして
- 出力仕様の「フィールド名」、「固定情報」、または「編集語」フィールド
- 配列指標として
- 定義仕様のキーワードとともに

リテラル

リテラルは、プログラムで参照することができる自身そのものを定義する固定情報です。リテラルは、VisualAge RPG データ・タイプのいずれかに属することができます。

文字リテラル

文字リテラルを指定する時には、次の規則が適用されます。

- 文字リテラルでは文字の組み合わせを使用することができます。これには、DBCS 文字が含まれます。 DBCS 文字は偶数バイト数でなければなりません。組み込みブランクは有効です。
- アポストロフィ間に文字のない文字リテラルを使用することができます。
- 文字リテラルは、アポストロフィ (') で囲まなければなりません。
- リテラルの一部として必要なアポストロフィは、2 つのアポストロフィによって表されます。たとえば、リテラル O'CLOCK は 'O''CLOCK' としてコーディングします。
- 文字リテラルは、文字データとだけ互換性があります。
- 標識リテラルは、'1' (オン) または '0' (オフ) のいずれかを含む 1 バイトの文字リテラルです。

16 進数リテラル

16 進数リテラルを指定する時には、次の規則が適用されます。

- 16 進数リテラルは次の形と取ります。

  ```
  X'x1x2...xn'
  ```

  ここで:

  X'x1x2...xn' は、文字 A - F、a - f、および 0 - 9 だけを含んでいなければなりません。
- アポストロフィの間でコーディングされるリテラルは、偶数の長さでなければなりません。
- 文字のそれぞれの対は、単一バイトを定義します。

- 16 進数リテラルは、ENDSR の演算項目 2 として、また編集語としての場合を除いて文字リテラルがサポートされていればどこでも使用することができます。
- 16 進数リテラルは、ビット演算命令の BITON、BITOFF、および TESTB で使用される場合を除いて対応する文字リテラルと同じ意味をもちます。ビット演算命令の場合、演算項目 2 には、1 バイトを表す 16 進数リテラルを指定することができます。16 進数リテラルの規則および意味は、文字フィールドの場合と同じです。
- 16 進数リテラルに単一引用符の 16 進値が含まれている場合には、文字リテラルと違って、2 度指定する必要はありません。たとえば、リテラル A'B は 'A''B' として指定されますが、16 進数バージョンでは X'412742' であって、X'41272742' ではありません。
- 通常、16 進数リテラルは、文字データとだけ互換性があります。しかし、入っている 16 進数字の数が 16 以下である 16 進数リテラルは、数式で使用される時、あるいは数値変数が INZ キーワードを使用して初期化される時には、符号なし数値として扱うことができます。

数値リテラル

数値リテラルを指定する時には、次の規則が適用されます。

- 数値リテラルは、0 - 9 の数字を任意に組み合わせたものから構成されます。小数点または符号を含めることができます。
- 符号 (+ または –) が存在する場合には、それは左端の文字でなければなりません。符号なしリテラルは、正数としてとして扱われます。
- 数値リテラルにブランクを入れることはできません。
- 数値リテラルは、アポストロフィ (') で囲みません。
- 数値リテラルは数値フィールドと同様に使用されますが、数値リテラルには値を割り当てることができません。
- 10 進数区切り記号はコンマまたはピリオドのいずれかでなければなりません。

浮動形式の数値リテラルは、いくぶん指定の仕方が異なります。浮動リテラル次の形を取ります。

```
<仮数>E<指数>
```

ここで、<仮数> は、上で説明した 1 - 16 桁のリテラルです。
<指数> は、-308 ～ +308 の値の小数部のないリテラルです。

- 浮動リテラルは正規化する必要はありません。すなわち、仮数は、小数点の左側に 1 桁ちょうどに書く必要はありません。(小数点は、指定する必要がありません。)
- **E** の代わりに小文字の **e** を使用してもかまいません。
- 小数点としてピリオド ('.') か コンマ (',') のいずれかを使用することができます。
- 浮動リテラルは、数値固定情報が使用できるところならどこでも使用することができます。ただし、浮動データ・タイプが使用できない演算命令の場合は除きます。たとえば、浮動リテラルは、配列指標など、小数点以下の桁数がゼロの数値リテラルが必要な所では使用することができません。
- 浮動リテラルは、通常の数値リテラルの場合と同じ継続規則に従います。リテラルは、リテラル内のどこででも分割することができます。
- 浮動リテラルには、4 ページの 1.6.2 「定義の規則」で説明した制限内の値が必要です。

次に有効な浮動リテラルの例をいくつか示します。

```
1E1                = 10
1.2e-1             = .12
-1234.9E0          = -1234.9
12e12              = 12000000000000
+67,89E+0003       = 67890  (小数点はコンマです)
```

次に無効な浮動リテラルの例をいくつか示します。

```
1.234E                          <--- 指数なし
1.2e-                           <--- 指数なし
-1234.9E+309                    <--- 指数が大き過ぎる
12E-2345                        <--- 指数が小さ過ぎる
1.797693134862316e308           <--- 値が大き過ぎる
179.7693134862316E306           <--- 値が大き過ぎる
0.0000000001E-308               <--- 値が小さ過ぎる
```

日付リテラル

日付リテラルは D'xxxxxx' の形を取ります。

- D は、リテラルが日付タイプであるということを示します。
- xxxxxx は、制御仕様で指定された形式の有効な日付です。
- xxxxxx は、アポストロフィ (') で囲みます。

時刻リテラル

時刻リテラルは T'xxxxxx' の形を取ります。

- T は、リテラルが時刻タイプであるということを示します。
- xxxxxx は、制御仕様で指定された形式の有効な時刻です。
- xxxxxx は、アポストロフィ (') で囲みます。

タイム・スタンプ・リテラル

タイム・スタンプ・リテラルは Z'yyyy-mm-dd-hh.mm.ss.mmmmmm' の形を取ります。

- Z は、リテラルがタイム・スタンプ・タイプであるということを示します。
- yyyy–mm–dd は、有効な日付 (年–月–日) です。
- hh.mm.ss.mmmmmm は、有効な時刻 (時.分.秒.マイクロ秒) です。
- yyyy–mm–dd–hh.mm.ss.mmmmmm は、アポストロフィで囲みます。
- マイクロ秒は任意指定であり、指定しない場合には、デフォルトのゼロとなります。

グラフィック・リテラル

グラフィック・リテラルは G'K1K2' の形を取ります。

- G は、リテラルがグラフィック・タイプであるということを示します。
- K1K2 は偶数バイト数です。
- K1K2 は、アポストロフィ (') で囲みます。

UCS-2 リテラル

UCS-2 リテラルは U'Xxxx...Yyyy' の形を取ります。

- U は、リテラルが UCS-2 タイプであるということを示します。
- 各 UCS-2 リテラルでは、リテラル中の各 UCS-2 文字ごとに 4 バイトが必要です。リテラルの各 4 バイトは、1 つの 2 バイト UCS-2 文字を表します。
- UCS-2 リテラルは、UCS-2 データとだけ互換性があります。

UCS-2 リテラルは、モジュールのデフォルトの UCS-2 CCSID にあると見なされます。

名前付き固定情報

名前付き固定情報は、リテラルに割り当てられた記号名です。名前付き固定情報は、定義仕様で定義します。名前付き固定情報の値は、リテラルに指定された規則に従います。

名前付き固定情報

固定情報に名前を割り当てることができます。この名前は、プログラムの実行中に変更することができない特定の値を表します。

名前付き固定情報の規則

- 名前付き固定情報は、演算仕様の演算項目 1、演算項目 2、および拡張演算項目 2 に、制御仕様のキーワードに対するパラメーターとして、組み込み関数に対するパラメーターとして、また出力仕様の「フィールド名」、「固定情報」、または「編集語」フィールドに指定することができます。また定義仕様の配列指標として、あるいはキーワードとともに使用することもできます。
- 数値名前付き固定情報には事前定義の精度がありません。実際の精度は、指定された文脈によって定義されます。
- 名前付き固定情報は、定義仕様のどこでも指定することができます。

名前付き固定情報の定義例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.....................................Keywords++++++++++++++++++++++++++++++
 *
 *  日付フィールドを定義して、1988 年 9 月 3 日に初期化します。
 *
D  DateField      S               D   INZ(D'1988-09-03')
 *
 *  2 進 9,5 フィールドを定義して、0 に初期化します。
 *
D  BIN9_5         S             9B 5 INZ
 *
 *  値が小文字のアルファベットである名前付き固定情報を定義します。
 *
D  Lower          C                   CONST('abcdefghijklmnop-
D                                     qrstuvwxyz')
 *
 *  キーワード CONST の使用を明示せずに名前付き固定情報を定義します。
 *
D  Upper          C                   'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

図 46. 名前付き固定情報の定義

表意定数

次の表意定数は、表意定数の暗黙の長さおよび小数点以下の桁数が関連したフィールドのそれと同じであるために、長さを指定せずに指定できる暗黙リテラルです。例外のリストについては、 162 ページの『表意定数の規則』を参照してください。

*ALL'x..' , *ALLG'K1K2'	*BLANK/*BLANKS	*HIVAL
*ALLU'XxxxYyyy',		
*ALLX'x1..'		
*LOVAL	*NULL	*ON/*OFF
*ZERO/*ZEROS		

表意定数は、演算仕様の演算項目 1 および演算項目 2 に指定することができます。以下に表意定数の予約語および暗黙の値を示します。

予約語	暗黙値
***BLANK/*BLANKS**	すべてブランク。文字、グラフィック、または UCS-2 フィールドの場合にのみ有効です。
***ZERO/*ZEROS**	文字 / 数値フィールド: 全桁ゼロ。*数値浮動フィールドの場合*: 値は '0 E0' です。
***HIVAL**	文字、グラフィック、または UCS-2 フィールド: システムの最高の照合文字 (16 進数 FF)。 数値フィールド: 正符号をもつすべての 9。 *浮動フィールドの場合* : 4 バイト の浮動フィールドの *HIVAL = 3.402 823 5E38 (x'FF7FFFFF') 8 バイトの浮動フィールドの *HIVAL = 1.797 693 134 862 315 E308 (x'FFEFFFFFFFFFFFFF') 日付、時刻、およびタイム・スタンプのフィールド: 日付、時刻、およびタイム・スタンプの *HIVAL値については、 125 ページの『日付データ』、 144 ページの『時刻データ』、および 145 ページの『タイム・スタンプ・データ』を参照してください。
***LOVAL**	文字、グラフィック、または UCS-2 フィールド: システムの最低の照合文字 (16 進数ゼロ)。 数値フィールド: 負符号をもつすべての 9。 *浮動フィールドの場合* : 4 バイト の浮動フィールドの *LOVAL = -3.402 823 5E38 (x'7F7FFFFF') 8 バイトの浮動フィールドの *LOVAL = -1.797 693 134 862 315 E308 (x'7FEFFFFFFFFFFFFF') 日付、時刻、およびタイム・スタンプのフィールド: 日付、時刻、およびタイム・スタンプの *LOVAL 値については、 125 ページの『日付データ』、 144 ページの『時刻データ』、および 145 ページの『タイム・スタンプ・データ』を参照してください。

***ALL'x..'**	文字 / 数値フィールド: 関連したフィールドの長さに等しくなるように文字ストリング x . .が循環的に反復されます。 フィールドが数値フィールドである場合には、ストリング内のすべての文字が数字 (0 - 9) でなければなりません。*ALL'x..' が数値固定情報として使用される時には、符号も小数点も指定することはできません。 **注:** 浮動形式の数値フィールドでは、*ALL'x..' を使用することはできません。 数値整数フィールドまたは数値符号なしフィールドの場合、値は、対応するフィールドの最大許容値を超えることはできません。
***ALLG'K1K2'**	グラフィック・フィールド: 関連したフィールドの長さに等しくなるようにグラフィック・フィールド K1K2 が循環的に反復されます。
***ALLU'XxxxYyyy'**	**UCS-2 フィールド**: *ALLU'XxxxYyyy' の形式の表意定数は、*ALLU'XxxxYyyy' 固定情報に関連したフィールドの長さによって決定された長さの 'XxxxYyyyXxxxYyyy...'の形式のリテラルを示します。固定情報中の各 2 バイト文字は、4 桁の 16 進数字によって表されます。たとえば、*ALLU'0041'は、反復した UCS-2 'A' のストリングを表します。
***ALLX'x1..'**	文字フィールド: 関連したフィールドの長さに等しくなるように 16 進数リテラル X'x1..' が循環的に反復されます。
***NULL**	基底ポインターまたはプロシージャー・ポインターにヌル値が有効です。
***ON/*OFF**	*ON は '1' で *OFF は '0' です。両方とも文字フィールドだけに有効です。

次の表意定数は、DSPLY 命令コードで使用できる暗黙リテラルです。

*ABORT	*CANCEL	*ENTER	*HALT
*IGNORE	*INFO	*NOBUTTON	OK(*O)
*RETRY	*WARN	*YESBUTTON	

次の表意定数は、アプリケーションの GUI を作成する時に使用できる暗黙リテラルです。

*BLACK	*BLUE	*BROWN	*CYAN
*DARKBLUE	*DARKCYAN	*DARKGREEN	*DARKGRAY
*DARKPINK	*DARKRED	*GREEN	*PALEGRAY
*PINK	*RED	*YELLOW	*WHITE

表意定数の規則

表意定数を使用する時には、次の規則が適用されます。

- 固定長文字フィールドに使用可能な表意定数は、可変長文字フィールド (*BLANK/*BLANKS、 *ZERO/*ZEROS、*HIVAL、*LOVAL、*ALL'x..'、*ALLG'K1K2'、*ALLX'x1..'、*ON/*OFF) にも使用可能です。
- 固定長グラフィック・フィールドに使用可能な表意定数は、可変長グラフィック・フィールド (*BLANK/*BLANKS、 *HIVAL、*LOVAL、*ALLG'K1K2') にも使用可能です。
- 表意定数は、固定長と可変長の両方の文字フィールドおよびグラフィック・フィールドで同じです。

```
*HIVAL   = X'FF'
*LOVAL   = X'00'
*BLANK   = ' ' または X'20' または 2 バイトのブランク
*ZERO    = '0' または X'30'
*OFF     = '0' または X'30'
*ON      = '1' または X'31'
```

- MOVE および MOVEL 演算命令では、文字表意定数を数値フィールドに移動することができます。表意定数は、最初に数値フィールドのサイズをもつゾーン数値として拡張され、必要な場合には、パックまたは 2 進数の数値フィールドに変換され、その後でターゲットの数値フィールドに保管されます。表意定数の各文字の数字部分は、有効でなければなりません。
- 表意定数は、基本項目と見なされます。MOVEA の場合を除いて、表意定数は、配列と一緒に使用されると、フィールドとして働きます。たとえば、MOVE *ALL'XYZ' ARR です。

  ARR に 4 バイトの文字エレメントがある場合には、各エレメントに 'XYZX' が入ります。
- MOVEA は、特殊なケースと見なされます。表意定数は、指定された配列の部分に等しい長さで生成されます。たとえば:
  - MOVEA *BLANK ARR(X)

    エレメント X で始まると、ARR の残りはブランクを含むことになります。
  - MOVEA *ALL'XYZ' ARR(X)

    ARR は、4 バイトの文字エレメントをもちます。エレメント境界は、常に文字 MOVEA演算命令での問題であるので無視されます。エレメント X で始まると、配列の残りは 'XYZXYZXYZ...'を含むことになります。
- SETGT および SETLL 命令コードの場合、演算項目 1 に *HIVAL または *LOVAL 値を使用できないようになっています。

注: MOVEA の結果は、MOVE の結果と異なります。

- 表意定数が適切な長さに設定されるか、設定し直されると、代替照合順序が指定されていれば、通常の照合順序を変更することができます。
- 移動演算命令の MOVE および MOVEL は、表意定数 *ALL'x..'、*ALLG'K1K2'、および *ALLX'x1..' を移動する時と同じ結果になります。ストリングは、関連したフィールドの長さがストリングの長さと同じになるまで、文字が (左側から) 循環的に反復されるものです。
- 表意定数は、演算項目の 1 つが表意定数でない限り、比較演算命令で使用することができます。
- 表意定数 *BLANK/*BLANKS は、MOVE 演算命令でゼロとして数値フィールドに移動されます。

第 11 章 データ構造

ストレージ中のエリアおよびそのエリア内のフィールド (サブフィールド) のレイアウトを定義することができます。ストレージ中のこのエリアは、データ構造と呼ばれています。データ構造を定義するには、定義仕様の 24 - 25 桁目に DS を指定します。

データ構造は、次のために使用することができます。
- 異なるデータ形式を使用して同じ内部域を複数回定義する
- 名前を使用して個々のサブフィールドを処理する
- データ構造の名前を使用してすべてのサブフィールドをグループとして処理する
- データ構造およびそのサブフィールドをレコードの場合と同様に定義する
- データのセットの複数オカレンスを定義する
- 不連続のデータを連続した内部ストレージ位置にグループ化する

それぞれ特定の目的をもった 3 つの特殊なデータ構造があります。
- データ域データ構造 (定義仕様の 23 桁目の U によって識別)。 261 ページの『23 桁目 (データ構造のタイプ)』 を参照してください。
- ファイル情報データ構造 (ファイル仕様のキーワード INFDS によって識別)。 248 ページの『INFDS(DSname)』を参照してください。
- プログラム状況データ構造 (定義仕様の 23 桁目の S によって識別)。 261 ページの『23 桁目 (データ構造のタイプ)』 を参照してください。

データ構造はプログラム記述または外部記述とすることができます。 LIKEDS キーワードを使用すれば、他のデータ構造と似た 1 つのデータ構造を定義することができます。

プログラム記述データ構造は、定義仕様の 22 桁目のブランクによって識別されます。プログラム記述データ構造のサブフィールド定義は、データ構造定義の直後に続かなければなりません。 260 ページの『22 桁目 (外部記述)』 を参照してください。

外部記述データ構造は、定義仕様の 22 桁目の E によって識別され、外部記述ファイルに含まれるサブフィールド記述をもっています。プログラムがコンパイルされる時には、データ構造サブフィールドの外部記述を見つけ、抜き出すために、外部名が使用されます。外部記述の名前を 7 - 21 桁目に指定するか、あるいはキーワード EXTNAME のパラメーターとして指定します。 260 ページの『7-21 桁目 (名前)』および 273 ページの『EXTNAME(file_name{:format_name})』を参照してください。

注: 外部記述中のサブフィールドに指定されたデータ形式は、コンパイラーによってサブフィールドの内部形式として使用されます。これは、外部記述ファイルの扱われ方と異なります。

外部サブフィールド名は、キーワード EXTFLD を使用してプログラム中で名前変更することができます。キーワード PREFIX を使用すれば、EXTFLD によって名前変更されていない外部サブフィールド名にプレフィックスを追加することができま

す。外部ファイル名を使用してデータ構造を定義する時に、ファイル名が EXTNAME キーワードに指定されたパラメーターと同じであっても、データ構造サブフィールドはファイル仕様に指定された PREFIX キーワードによって影響されないということに注意してください。外部サブフィールドのリストの直後にプログラム記述サブフィールドを指定することによって、外部記述データ構造にさらにサブフィールドを追加することができます。 272 ページの『EXTFLD(field_name)』および 292 ページの『PREFIX(prefix{:nbr_of_char_replaced})』を参照してください。

プロトタイプまたはプロシージャー・インターフェースでのデータ構造パラメーターの定義

プロトタイピングされたパラメーターをデータ構造として定義するには、最初に通常のデータ構造を定義して、パラメーターのレイアウトを定義しなければなりません。次に、LIKEDS キーワードを使用して、プロトタイピングされたパラメーターをデータ構造として定義することができます。パラメーターのサブフィールドを使用するには、パラメーター名で修飾したサブフィールド (dsparm.subfield) を指定します。たとえば、以下のようになります。

```
 * PartInfo は、パーツを記述するデータ構造です。
D PartInfo        DS
D  Manufactr                     4
D  Drug                          6
D  Strength                      3
D  Count                         3  0
 * プロシージャー "Proc" にはパラメーター "Part" があります。
 * このパラメーターは、サブフィールドが "PartInfo" 中の
 * サブフィールドと同じであるデータ構造です。このプロシージャー
 * を呼び出すときには、LIKEDS(PartInfo) としても定義されている
 * パラメーターを渡す (あるいは "PartInfo" 自体を渡す) のが最善
 * ですが、コンパイラーは、正しい長さのどんな文字フィールド
 * でも渡せるようにします。
D Proc            PR
D  Part                                LIKEDS(PartInfo)
P Proc            B
 * プロシージャー・インターフェースは、またキーワード
 * LIKEDS(PartInfo) でパラメーター Part を定義します。
 * これは、パラメーターがデータ構造であり、サブフィールドを "Part." で
 * 修飾して指定すれば (たとえば "Part.Strength")、そのサブフィールドを
 * 使用できるということを意味します。
D Proc            PI
D  Part                                LIKEDS(PartInfo)
C                   IF        Part.Strength > getMaxStrength (Part.Drug)
C                   CALLP     PartError (Part : DRUG_STRENGTH_ERROR)
C                   ELSE
C                   EVAL      Part.Count = Part.Count + 1
C                   ENDIF
P Proc            E
```

データ構造サブフィールドの定義

サブフィールドは、定義仕様の「定義タイプ」項目 (24 - 25 桁目) にブランクを指定することによって定義します。サブフィールド定義は、データ構造定義の直後に続かなければなりません。非ブランクの「定義タイプ」項目をもつ定義仕様が見つかるか、あるいは異なるタイプの仕様が見つかると、サブフィールド定義が終了します。

サブフィールドの名前は、7 - 21 桁目に入力します。ソースの読みやすさを改善するために、サブフィールド名を字下げして、それらがサブフィールドであることが視覚的に分かるようにすることができます。

データ構造が QUALIFIED キーワードで定義されている場合には、サブフィールド名は、プログラム内の他の名前と同じにすることができます。サブフィールド名は、使用されるときにそれを所有するデータ構造で修飾されます。

LIKE キーワードを使用して、既存の項目のようにサブフィールドを定義することもできます。このように定義すると、サブフィールドには、基礎となる項目の長さとデータ・タイプが入ります。同様に、LIKEDS キーワードを使用して、既存の項目と似たデータ構造全体を定義することができます。 LIKE キーワードの使用例については、 280 ページの図 87 を参照してください。

OVERLAY キーワードを使用すれば、前に定義されているサブフィールドのストレージを別のサブフィールドのストレージでオーバーレイすることができます。このキーワードは、後のサブフィールド定義で指定します。

サブフィールドの長さの指定

サブフィールドの長さは絶対 (定位置) 表記または長さ表記を使用して指定することができます。

絶対 定義仕様の「開始位置」(26 - 32 桁目) と「終了位置」/「長さ」(33 - 39 桁目) の両方の項目に値を指定します。

長さ 「終了位置」/「長さ」(33 - 39 桁目) 項目に値を指定します。「開始位置」項目はブランクです。

長さ表記を使用すると、サブフィールドは、その開始位置が前に定義されているすべてのサブフィールドの最大「終了位置」より大きくなるように位置付けられます。それぞれの表記の例については、 169 ページの『データ構造の例』を参照してください。

データ構造サブフィールドの位置合わせ

サブフィールドの位置合わせが必要になる場合があります。これは自動的に行なわれる場合もあれば、手操作で行なわなければならない場合もあります。

たとえば、長さ表記を使用してサブフィールド・タイプの基底ポインターまたはプロシージャー・ポインターを定義すると、サブフィールドが正しく位置合わせされるようにするために必要な場合には、コンパイラーが自動的に埋め込みを実行します。

浮動、整数、または符号なしのサブフィールドを定義すると、実行時パフォーマンスを改善するために、位置合わせが必要になる場合があります。サブフィールドが

長さ表記を使用して定義された場合には、データ構造定義にキーワード ALIGN を指定することによって、浮動、整数、または符号なしのサブフィールドを自動的に位置合わせすることができます。しかし、次の例外に注意してください。

- ファイル情報データ構造またはプログラム状況データ構造には ALIGN キーワードを使用することができません。
- キーワード OVERLAY を使用して定義されたサブフィールドは、データ構造にキーワード ALIGN が指定されていても、自動的に位置合わせされることはありません。この場合には、サブフィールドを手操作で位置合わせしなければなりません。

自動位置合わせの場合、フィールドは次の境界に位置合わせされます。

- 5 桁の整数サブフィールドまたは符号なしサブフィールドの場合 2 バイト
- 10 桁の整数サブフィールドまたは符号なしサブフィールドあるいは 4 バイトの浮動サブフィールドの場合 4 バイト
- 20 桁の整数サブフィールドまたは符号なしサブフィールドの場合 8 バイト
- 8 バイトの浮動サブフィールドの場合 8 バイト
- ポインター・サブフィールドの場合 16 バイト

フィールドを手操作で位置合わせする場合には、各フィールドが同じ境界に位置合わせされるようにしてください。 ((桁 - 1) mod n) = 0 の場合には、開始位置は、n バイト境界です。(″x mod y″ の値は、整数演算で x を y で除算した余りです。) これは、X DIV Y の後の MVR 値と同じです。

図 47は、バイトの順序を示し、位置合わせに使用される各種の境界を識別します。

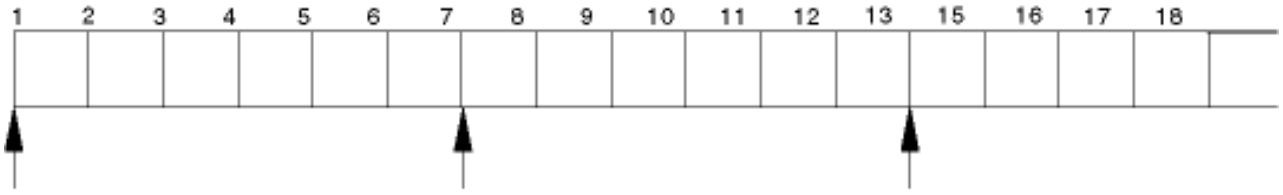


図 47. データの位置合わせの境界

以下では、先行バイトの順序に注意してください。

- 1 桁目が 16 バイト境界、((1-1) mod 16) = 0 であるため。
- 13 桁目は 4 バイト境界、((13-1) mod 4) = 0 であるため。
- 7 桁目は 4 バイト境界ではない、((7-1) mod 4) = 2 であるため。

データ域データ構造

データ域データ構造は、定義仕様の 23 桁目の U によって指定されます。これは、プログラム初期化の時に読み取られ、ロックされた同じデータ域がプログラムの終了時に書き出され、アンロックされなければならないということを示します。その他のデータ構造と同様にデータ域データ構造は、タイプとして文字をもちます。データ域データ構造に読み込まれるデータ域も文字でなければなりません。
*DTAARA DEFINE命令コードまたは DTAARA キーワードを使用して、プログラム中でデータ域の名前を変更しない限り、データ域およびデータ域データ構造は、

同じ名前でなければなりません。 529 ページの『DEFINE (フィールド定義)』 および 271 ページの『DTAARA{(data_area_name)}』 を参照してください。

暗黙的に読み込まれるか、あるいは書き出されるデータ域にデータ域演算命令 (IN、OUT、および UNLOCK)を指定することができます。こうした演算命令でデータ域データ構造を使用する前に、 *DTAARA DEFINE 演算命令の結果フィールドにそのデータ域を指定するか、あるいは DTAARA キーワードでそのデータ域を指定しなければなりません。 529 ページの『DEFINE (フィールド定義)』 および 271 ページの『DTAARA{(data_area_name)}』 を参照してください。

注: *ENTRY PLISTの PARM 演算命令の結果フィールドにはデータ域データ構造を指定することはできません。

ファイル情報データ構造

プログラム中の各ファイルにファイル情報データ構造を指定することができます。ファイル情報データ構造は、ファイル仕様のキーワード INFDS によって定義されます。 248 ページの『INFDS(DSname)』を参照してください。 これによって、起こったファイル例外またはエラーについての状況情報が提供されます。ファイル情報データ構造の名前は、各ファイルに固有でなければなりません。ファイル情報データ構造には、起こったファイル例外またはエラーについての情報を提供するサブフィールドが含まれています。 ファイル情報データ構造およびそのサブフィールドについては、 42 ページの『ファイル情報データ構造』を参照してください。

プログラム状況データ構造

プログラム状況データ構造は、プログラムにプログラム例外およびエラーについての情報を提供します。これは、定義仕様の 23 桁目の S によって識別されます。プログラム状況データ構造およびそのサブフィールドの詳細については、 52 ページの『プログラム状況データ構造』を参照してください。

データ構造の例

次の例は、データ構造を定義して使用する方法を示したものです。

- データ構造を使用してフィールドをさらに再分割する
- データ構造を使用して、フィールドをグループにする
- 絶対表記および長さ表記によるデータ構造
- 外部記述データ構造を名前変更して、初期化する
- PREFIX を使用して、外部データ構造中のすべてのフィールドを名前変更する
- 複数回繰り返しデータ構造を定義する
- データ域データ構造を使用する

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 * 長さ表記を使用して、データ構造サブフィールドを定義します。
 * Partno を使用するか、あるいは個別のサブフィールド Manufactr、Drug、
 * Strength、または Count を使用してデータ構造全体を参照することができます。
 *
D Partno          DS
D  Manufactr                     4
D  Drug                          6
D  Strength                      3
D  Count                         3  0
D
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++Sq..RiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++....FrPlMnZr......
 *
 *  プログラム記述ファイル FILEIN 中のレコードには、Partno というフィールドが
 *  含まれていますが、これはこのプログラムでの処理のために再分割する必要が
 *  あります。これを行なうには、上の定義仕様を使用して、フィールド Partno を
 *  データ構造として記述します。
 *
IFILEIN    NS  01   1 CA    2 CB
I                                   3   18  Partno
I                                  19   29  Name
I                                  30   40  Patno
```

図 48. データ構造を使用して、フィールドを再分割

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 *  データ構造を使用してフィールドをグループ化すると、入力レコードで
 *  隣接していない位置にあるフィールドが隣接した内部位置を占めるように
 *  することができます。ここでこのエリアは、データ構造名または個別の
 *  サブフィールド名によって参照できるようになります。
 *
D Partkey         DS
D  Location                      4
D  Partno                        8
D  Type                          4
D
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++Sq..RiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++....FrPlMnZr......
 *
 *  プログラム記述ファイル TRANSACTN からのフィールドは、Item_Master
 *  から取り出されたフィールドと比較する必要があります。
 *
ITRANSACTN NS  01   1 C1    2 C2
I                                 3   10  Partno
I                                11   16 0Quantity
I                                17   20  Type
I                                21   21  Code
I                                22   25  Location
I*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  フィールド Item_Nbr に比較するには、データ構造名 Partkey を使用
 *  します。
 *
C                   :
C     Partkey       IFEQ      Item_Nbr                             99
C                   :
C*
```

図 49. データ構造を使用して、フィールドをグループ化

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 *  FRED と呼ばれるプログラム記述データ構造を定義します。
 *  このデータ構造は次の 5 つのフィールドから構成されます。
 *   1.   長さが 10 で、ディメンションが 70 のエレメントをもつ配列 (Field1)
 *   2.   長さが 30 のフィールド (Field2)
 *   3/4. Field2 を長さが等しい 2 つのフィールド (Field3 と Field4) に分割
 *   5.   3 番目のフィールドについで 2 進数フィールドを定義
 *  読みやすくするための字下げに注意してください。
 *
 *
 *  絶対表記:
 *
 *  コンパイラーは、合計長 (700) をディメンション (70) で除算することに
 *  よって、配列エレメントの長さ (Field1) を決定します。
 *
D FRED            DS
D  Field1                  1    700    DIM(70)
D  Field2                701    730
D   Field3               701    715
D    Field5              701    704B 2
D   Field4               716    730
 *
 *  長さ表記:
 *
 *  Field2 を再分割するために OVERLAY キーワードが使用されます。
 *
D FRED            DS
D  Field1                           10    DIM(70)
D  Field2                           30
D   Field3                          15    OVERLAY(Field2)
D    Field5                          4B 2 OVERLAY(Field3)
D   Field4                          15    OVERLAY(Field2:16)
```

図 50. 絶対表記および長さ表記のデータ構造

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 *  内部名 FRED をもち、外部名 EXTDS をもつ外部記述データ構造を定義し、
 *  フィールド CUST を CUSTNAME に名前変更し、CUSTNAME を 'GEORGE' に、
 *  PRICE を 1234.89 に初期化します。
 *  サブフィールド ITMARR (外部記述に 100 バイトの文字フィールドとして
 *  定義) に DIM キーワードを割り当てます。
 *
D Fred          E DS                  EXTNAME(EXTDS)
D   CUSTNAME    E                     EXTFLD(CUST) INZ('GEORGE')
D   PRICE       E                     INZ(1234.89)
D   ITMARR      E                     DIM(10)
```

図 51. 外部記述データ構造の名前変更および初期化

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
D
D extds1        E DS                  EXTNAME (CUSTDATA)
D                                     PREFIX (CU_)
D   Name        E                     INZ ('Joe's Garage')
D   Custnum     E                     EXTFLD (NUMBER)
D
 *
 *  前のデータ構造は、次のように拡張されます。
 *  -- すべての外部記述フィールドがデータ構造に含められる
 *  -- 名前変更されたサブフィールドはその新しい名前を保持する
 *  -- 名前変更されていないサブフィールドには、プレフィックス・
 *     ストリングでプレフィックスが付けられる
 *
 *  拡張データ構造:
 *
D EXTDS1        E DS
D   CU_NAME     E               20A   EXTFLD (NAME)
D                                     INZ ('Joe's Garage')
D   CU_ADDR     E               50A   EXTFLD (ADDR)
D   CUSTNUM     E                9S0  EXTFLD (NUMBER)
D   CU_SALESMN  E                7P0  EXTFLD (SALESMN)
```

図 52. PREFIX を使用して、外部データ構造中のすべてのフィールドを名前変更

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 *  次のもので 20 個のエレメントをもつ複数回繰り返しデータ構造を定義します。
 *  -- 文字 20 の 3 つのフィールド
 *  -- 文字 10 の 4 番目のフィールド。これは、2 桁目から 2 番目の
 *     フィールドをオーバーラップします。
 *
 *  名前付き固定情報 'twenty' を使用して、オカレンスが定義されます
 *
 *  絶対表記 (開始/終了位置を使用)
 *
D twenty          C                   CONST(20)
D
DDataStruct       DS                  OCCURS (twenty)
D field1                   1     20
D field2                  21     40
D   field21               22     31
D  field3                 41     60 *
 *  絶対表記と長さ表記の混合
 *
D DataStruct      DS                  OCCURS(twenty)
D  field1                        20
D  field2                        20
D   field21               22     31
D  field3                 41     60
```

図 53. 複数回繰り返しデータ構造の定義

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 *  このプログラムは、データ域データ構造を使用して、一連の合計を累算します。
 *
D  Totals        UDS
D    Tot_amount                  8 2
D    Tot_gross                  10 2
D    Tot_netto                  10 2
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CSRN01Factor1+++++++Opcode(E)+Factor2++++++++++++++++++++++++++++++++++++++++
 *
C                   :
C                   EVAL      Tot_amount = Tot_amount + amount
C                   EVAL      Tot_gross  = Tot_gross  + gross
C                   EVAL      Tot_netto  = Tot_netto  + netto
```

図 54. データ域データ構造の使用

第 12 章 配列およびテーブルの使用

配列およびテーブルは共に、次のものが同じであるデータ・フィールド (エレメント) の集まりです。
- フィールド長
- データ・タイプ
  - 文字
  - 数値
  - 日付
  - 時刻
  - タイム・スタンプ
  - グラフィック
  - 基底ポインター
  - プロシージャー・ポインター
  - UCS-2
- フォーマット
- 小数点以下の桁数 (数値の場合)

配列とテーブルは次の点で異なります。
- 特定の配列エレメントをその位置によって参照することができます
- 特定のテーブル・エレメントは、その位置によって参照することはできません
- 配列そのものは、配列中のすべてのエレメントを指します
- テーブル名は、最後の LOOKUP (テーブル・エレメントまたは配列エレメントの検索) 演算命令で見つかったエレメントを指します。.

注: サブプロシージャーでは、実行時配列だけを定義することができます。テーブル、実行前配列、およびコンパイル時配列はサポートされていません。

次のセクションでは、配列の使用法について説明します。
- 『配列』
- 182 ページの『配列の初期化』
- 182 ページの『関連した配列の定義』
- 184 ページの『配列の検索』
- 186 ページの『配列の使用』
- 188 ページの『配列出力』

189 ページの『テーブル』では、テーブルについて同じことを説明します。

『配列』では、配列のコーディング方法、配列エレメントの初期値の指定方法、配列の値の変更方法、および配列を使用する場合の特殊な考慮事項について説明します。

配列

配列には 3 つのタイプがあります。
- 実行時配列は、プログラムの実行中にロードされます。

- コンパイル時配列は、プログラムが作成された時にロードされます。初期データは、プログラムの永続パーツとなります。
- 実行前配列は、プログラムが実行を開始する時、いずれかの入力、演算、または出力演算命令が処理される前に配列ファイルからロードされます。

実行時配列の場合に配列を定義およびロードする基本事項について説明します。コンパイル時配列および実行前配列の定義およびロードの場合には、この基本事項およびある種の追加仕様を使用します。

配列の名前および指標

配列全体は、配列名を単独で使用して参照します。配列の個々のエレメントは、配列名、次に左括弧、さらに指標、最後に右括弧を付けたものを使用して参照します。たとえば:

```
AR(IND)
```

指標は、配列内のエレメントの位置 (1 から開始) を示すもので、数値または数値を収めるフィールドのいずれかです。

配列名および指標を指定する時には、次の規則が適用されます。
- 配列名は、固有の記号名でなければなりません。
- 指標は、数値フィールドであるか、ゼロより大きく、小数点以下の桁数がゼロの固定情報です。
- 拡張演算項目 2 フィールド中の式内に配列を指定した場合には、指標は、小数点以下の桁数がゼロの数値を戻す式とすることができます。
- 実行時に、プログラムがゼロ、負数、または配列中のエレメントの数より大きい値の指標を使用して配列を参照すると、エラー / 例外ルーチンがプログラムの制御権をもちます。

基本配列仕様

配列は、定義仕様で定義します。
- 配列名を 7 - 21 桁目に指定します。
- DIM キーワードを使用して配列中の項目の数を指定します。
- スカラー・フィールドに必要とする長さ、データ形式、および小数点以下の桁数を指定します。明示的な「開始位置」と「終了位置」項目を指定するか (サブフィールドを定義する場合)、明示的な「長さ」項目を指定することができます。あるいは LIKE キーワードを使用して配列属性を定義することができます。属性は、プログラム中のどこでも指定することができます。
- 分類順序を指定する必要がある場合には、ASCEND キーワードまたは DESCEND キーワードを使用します。

177 ページの図 55は、基本配列仕様の例を示したものです。

実行時配列のコーディング

基本配列仕様以上に仕様が必要ない場合には、実行時配列を定義したことになります。実行時配列の場合には、キーワードの ALT、CTDATA、EXTFMT、FROMFILE、PERRCD、および TOFILE は、使用できないということに注意してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DARC              S              3A   DIM(12)
```

図 55. 実行時配列を定義するための基本配列仕様

実行時配列のロード

定義仕様で INZ キーワードを使用すれば、実行時配列の初期値を割り当てることができます。入力仕様または演算仕様からも実行時配列の初期値を割り当てることができます。この 2 番目の方法は、他のタイプの配列にデータを入れるために使用することもできます。

たとえば、MOVE 演算命令の演算仕様を使用して、配列の各エレメント (あるいは選択したエレメント) に 0 を入れることができます。

入力仕様を使用すれば、ファイルからのデータで配列を埋めることができます。次のセクションでは、ファイルのレコードからこのデータを検索することについて詳しく説明します。

注: 日時の実行時データは、ロードされる日付または時刻配列と同じ形式でなければならず、同じ区切り文字を使用しなければなりません。

1 つのソース・レコードへの実行時配列のロード

配列情報が 1 つのレコードに入っている場合には、情報はレコード中の連続した位置を占めていても、レコード全体に分散していてもかまいません。

配列エレメントが入力レコードで連続している場合には、配列は単一の入力仕様でロードすることができます。 図 56 には、1 つのレコードから 6 つのエレメント (それぞれ 12 文字) からなる配列をロードするための仕様が示されています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DINPARR           S             12A   DIM(6)
IFilename++Sq..RiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++....FrPlMnZr....
IARRFILE   AA  01
I                                1   72  INPARR
```

図 56. 連続したエレメントをもつ実行時配列の使用

配列エレメントがレコード全体に分散している場合には、仕様行に 1 エレメントずつ記述して、一度に 1 つずつエレメントを定義して、ロードすることができます。

178 ページの図 57 には、1 つのレコードから 6 つのエレメント (それぞれ 12 文字) からなる配列をロードするための仕様が示されています。各エレメント間を分離するためにブランクがあります。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DARRX             S             12A   DIM(6)
IFilename++Sq..RiPos1+NCCPos2+NCCPos3+NCC.............................
I.....................Fmt+SPFrom+To+++DcField+++++++++....FrPlMnZr....
IARRFILE   AA  01
I                                  1   12  ARRX(1)
I                                 14   25  ARRX(2)
I                                 27   38  ARRX(3)
I                                 40   51  ARRX(4)
I                                 53   64  ARRX(5)
I                                 66   77  ARRX(6)
```

図 57. 分散したエレメントをもつ実行時配列の定義

複数のソース・レコードを使用して実行時配列をロード

配列情報が複数のレコードにある場合には、各種の方法を使用して配列をロードすることができます。使用する方法は、配列のサイズおよび配列エレメントが入力レコードで連続しているかどうかによります。レコードは、一度に 1 レコードずつ処理されます。したがって、配列全体は、配列情報を含むすべてのレコードが読み取られ、情報が配列フィールドに移されるまで、処理されません。配列全体がプログラムに読み込まれるまで、演算および出力の演算命令の処理を抑制しなければならない場合もあります。

実行時配列の順序付け

実行時配列は、順序検査されません。SORTA (配列の分類) 演算命令を処理した場合には、配列は、その配列を定義する定義仕様 (ASCEND キーワードまたは DESCEND キーワード) に指定された順序に分類されます。順序が指定されていない場合には、配列は昇順に分類されます。LOOKUP 演算命令で高 (演算仕様の 71 ～ 72 桁目)、または低 (演算仕様の 73 ～ 74 桁目) 標識が使用されている時には、配列の順序を指定しなければなりません。

コンパイル時配列のコーディング

コンパイル時配列は、基本配列仕様とキーワード CTDATA を使用して指定します。定義仕様で PERRCDキーワードを使用して、入力レコードに配列項目の数を指定することができます。PERRCD キーワードを指定しなかった場合には、項目数はデフォルトで 1 になります。例については、 179 ページの図 58 中の仕様を参照してください。

EXTFMT(code) キーワードを使用して外部データ形式を指定することができます。詳細については、 272 ページの『EXTFMT(code)』 を参照してください。

注: 配列データがワークステーションにある場合には、EXTFMT キーワードを使用することはできません。浮動コンパイル時配列には、EXTFMT キーワードは使用できません。

TOFILE キーワードを使用すれば、LR がオンになってプログラムが終了した時に、配列が書き込まれるファイルを指定することができます。

コンパイル時配列のロード

コンパイル時配列の場合、配列ソース・データをプログラム・ソース・メンバー中のレコードに入力してください。**CTDATA キーワードを使用した場合には、配列データは、ソース・レコードの後のどこでも入力することができます。このキーワードを使用しない場合には、配列データは、定義仕様でコンパイル時配列およびテーブルが定義されている順序でソース・レコードの後に続かなければなりません。このデータは、プログラムがコンパイルされると、配列にロードされます。プログラムが新しいデータで再コンパイルされるまで、LR がオフで前の呼び出しが終了していない限り、プログラムが呼び出されるたびに、配列は、常に最初に同じ値をもっています。

コンパイル時配列は別々に記述するか、交互形式で (ALT キーワードで) 記述することができます。交互形式は、入力レコード上の 1 つの配列のエレメントが別の配列のエレメントと混合されるということを意味します。

配列ソース・レコードの規則

配列ソース・レコードの規則は次の通りです。

- 各レコードの最初の配列項目は、1 桁目から開始しなければなりません。
- すべてのエレメントは同じ長さでなければならず、介在スペースなしに続いていなければなりません。
- レコード全体を項目で満たす必要はありません。満たされていない場合には、項目の後にブランクまたはコメントを含めることができます。図 58を参照してください。
- 定義仕様に指定された配列中のエレメントの数が提供された項目数より大きくなっている場合には、残りのエレメントには指定されたデータ・タイプのデフォルト値が埋められます。

```
     DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++
     DARC              S              3A   DIM(12) PERRCD(5) CTDATA
**CTDATA ARC
48K16343J64044HComments can be placed here
12648A47349K346Comments can be placed here
50B125         Comments can be placed here
```

48K	163	43J	640	44H	126	48A	473	49K	346	50B	125

This is the compile-time array, ARC.

図 58. コメント付き配列ソース・レコード

- 最後のレコードを除く各レコードは、定義仕様の PERRCD キーワードで指定された数の項目を含んでいなければなりません。最終レコードでは、未使用の項目はブランクでなければならず、未使用項目の後にコメントを含めることができます。
- 各項目は、全体が 1 つのレコードに含まれなければなりません。項目を 2 つのレコードに分割することはできません。1 つの項目の長さは、100 文字という最大長 (ソース・レコードのサイズ) に制限されています。配列が使用され、交互形

式で記述されている場合には、対応するエレメントが同じレコードになければなりません。それらのエレメントはまとめた長さは 100 文字を超えることはできません。
- 日時のコンパイル時配列の場合、データは、ロードされる日付または時刻配列と同じ形式でなければならず、同じ区切り文字を使用しなければなりません。
- 配列データは、次の 2 つの方法のいずれかで指定することができます。
  - **CTDATA arrayname: 配列のデータは、コンパイル時データ・セクションのどこにも指定することができます。
  - **b: (b=ブランク) 配列のデータは、定義仕様に指定したのと同じ順序で指定しなければなりません。

  1 つのプログラムではこれらの手法のうちの 1 つしか使用することができません。
- 配列は、昇順 (ASCEND キーワード)、降順 (DESCEND キーワード)、または順序なし (キーワードの指定なし) にすることができます。
- グラフィック配列および UCS-2 配列は、16 進値によって分類されます。
- 定義仕様で EXTFMT キーワードに L または R を指定した場合には、各エレメントに符号 (+ または –) を含める必要があります。たとえば、L が指定され、2 というエレメント・サイズの配列は、ソース・データに 3 桁を必要とします (+37–38+52–63)。
- 浮動コンパイル時データは、浮動リテラルまたは数値リテラルとしてソース・レコードに指定されます。4 バイト浮動として定義された配列は、各エレメントに 14 桁を必要とします。8 バイト浮動として定義された配列は、各エレメントに 23 桁を必要とします。

実行前配列のコーディング

定義仕様では、基本配列仕様に加えて、FROMFILE キーワードを使用して、配列入力データをもつファイルの名前を指定することができます。TOFILE キーワードを使用して、プログラムの終わりで配列が書き込まれるファイルの名前を指定することができます。ファイルが入出力共用ファイル (ファイル仕様の 17 桁目の C で指定) である場合には、FROMFILE キーワードと TOFILE キーワードのパラメーターが同じでなければなりません。PERRCD キーワードを使用して、入力レコード当たりのエレメント数を指定することができます。

EXTFMT キーワードでは次のものを指定します。
- B (データが 2 進数形式である場合)
- L (符号がデータ・エレメントの左にあることを示すため)
- P (配列データがパック 10 進数形式である場合)
- R (符号がデータ・エレメントの右にあることを示すため)
- S (配列データがゾーン 10 進数形式である場合)

配列入力データをもつファイルのファイル仕様の 18 桁目に T を指定します。

2 つの実行前配列、1 つのコンパイル時配列、および 1 つの実行時配列のコーディングを比較するためには、181 ページの図 59 を参照してください。

ALT キーワードを使用すれば、交互形式で配列を指定することができます。(181 ページの図 59 を参照してください。)

注: 10 桁を超えて定義された配列の場合には、整数形式または符号なし形式を指定することはできません。

```
*....+....1....+....2....+....3....+....4....+....5....+....6....+....*
      HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      H DATFMT(*USA) TIMFMT(*HMS)
      DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++
      D* 実行時配列。ARI は日付タイプの 10 個のエレメントをもちます。これらの
      D* エレメントは、1994 年 9 月 15 日に初期化されます。これは、制御仕様で
      D* 定義されているように、月、日、年の形式で、区切り文字として
      D* スラッシュを使用しています。
      DARI             S               D   DIM(10) INZ(D'09/15/1994')
      D* 交互形式のコンパイル時配列。2 つの配列はともに 8 個のエレメント
      D* (レコード当たり 3 つのエレメント) をもちます。ARC は 15 という長さの
      D* 文字配列で、ARD は事前定義された 8 という長さの時刻配列です。
      DARC             S              15    DIM(8) PERRCD(3)
      D                                     CTDATA
      DARD             S               T   DIM(8) ALT(ARC)
      D*
      D* 実行前配列。ARE は、ファイル DISKIN から読み取られるもので、
      D* 250 個の文字エレメント (レコード当たり 12 個のエレメント) をもちます。各
      D* エレメントは 5 桁の長さです。各レコードのサイズは 60 (5*12) です。
      D* 各エレメントは昇順に配置されています。
      DARE             S              5A   DIM(250) PERRCD(12) ASCEND
      D                                     FROMFILE(DISKIN)
      D*
      D*
      D* 入出力共用ファイルとして指定された実行前配列。ARH は、LR がオン
      D* の状態でプログラムが正常に終了した時に、読み取られたのと同じ
      D* ファイルに書き戻されます。ARH は 250 個の文字エレメント (レコード当たり
      D* 12 個のエレメント) をもちます。各エレメントは 5 桁の長さです。エレメント
      D* は昇順の順序に配置されています。
      DARH             S              5A   DIM(250) PERRCD(12) ASCEND
      D                                     FROMFILE(DISKOUT)
      D                                     TOFILE(DISKOUT)
**CTDATA ARC
Toronto        12:15:00Winnipeg       13:23:00Calgary        15:44:00
Sydney         17:24:30Edmonton       21:33:00Saskatoon      08:40:00
Regina         12:33:00Vancouver      13:20:00
```

図 59. 各種の配列の定義仕様

実行前配列のロード

実行前配列の場合、配列入力データを順次プログラム記述ファイルに入力します。入力、演算、または出力の演算命令が処理される前にプログラムを呼び出すと、配列はファイルからの初期値をロードされます。このファイルを変更することによって、次回のプログラムに対する呼び出しで、プログラムを再コンパイルすることなく、配列の初期値を変更することができます。ファイルは到着順で読み取りが行なわれます。実行前配列データの場合の規則は、コンパイル時配列データと同じですが、ただし各レコードの長さに制限がないという点が異なります。 179 ページの『配列ソース・レコードの規則』 を参照してください。

文字配列の順序検査

文字配列の順序検査が起こると、VisualAge RPG は、デフォルトの ASCII 照合順序を使用します。

配列の初期化

実行時配列の各エレメントを同じ値に初期化するには、定義仕様で INZ キーワードを指定します。配列がデータ構造サブフィールドとして定義されている場合には、データ構造初期化オーバーラップの通常の規則が適用されます (初期化は、フィールドがデータ構造内で宣言された順序で行なわれます)。

コンパイル時および実行前配列

コンパイル時配列または実行前配列の場合、その初期値が他の手段 (コンパイル時データまたは入力ファイルからのデータ) を通して配列に割り当てられるので、INZ キーワードは指定することはできません。コンパイル時配列または実行前配列がグローバルに初期化されたデータ構造に現われる場合には、その配列はグローバル初期化に組み込まれません。

注: 配列がデータ構造で宣言された順序に関係なく、コンパイル時配列は、プログラムの後でデータが宣言された順序で初期化され、実行前配列は、その初期設定ファイルの宣言順序で初期化されます。実行前配列は、コンパイル時配列の後で初期化されます。

サブフィールド初期化でコンパイル時配列または実行前配列のオーバーラップがあった場合には、データ構造内でフィールドが宣言された順序に関係なく、配列はサブフィールドの後で初期化されます。

関連した配列の定義

交互配列の定義で ALT キーワードを使用すれば、2 つのコンパイル時配列または 2 つの実行前配列を交互形式でロードすることができます。基本配列の名前を ALT キーワードのパラメーターとして指定します。このような配列のデータを保管するためのレコードは、最初の配列の最初のエレメントの後に 2 番目の配列の最初のエレメントが続き、最初の配列の 2 番目のエレメントの後に 2 番目の配列の 2 番目のエレメントが続き、最初の配列の 3番目のエレメントの後に 2 番目のエレメントの 3 番目のエレメントが続くといったようになっています。対応するエレメントは、同じレコードになければなりません。主配列定義の PERRCD キーワードは、レコードごとの対応する対の数を指定し、その際にエレメントのそれぞれの対を 1 つの項目としてカウントします。主配列と代替配列の両方に EXTFMT を指定することができます。

183 ページの図 60 は、交互形式の 2 つの配列を示しています。

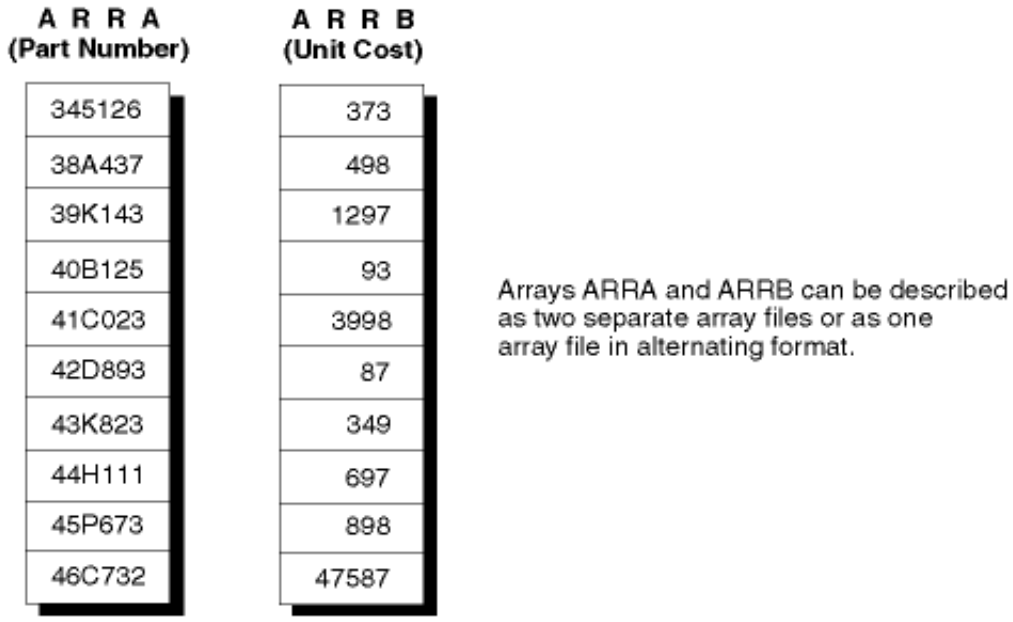


図 60. 交互形式および非交互形式の配列

2 つの個別配列ファイルとして記述されると、ARRA と ARRB のレコードは、図 61 中のレコードのようになります。

このレコードは、1 - 60 桁目に ARRA 項目が入っています。

ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry
1	7	13	19	25	31	37	43	49	55 . . . .

図 61. 2 つの個別配列ファイルの配列レコード

このレコードは、1 - 50 桁目に ARRB 項目が入っています。

ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry
1	6	11	16	21	26	31	36	41	46 . . . .

図 62. 1 つの配列ファイルの配列レコード

交互形式の 1 つの配列ファイルとして記述される時には、ARRA と ARRB のレコードは、下の 184 ページの図 63 中のレコードのようになります。最初のレコードには、交互形式で 1 - 55 桁目に ARRA および ARRB 項目が入っています。 2 番目のレコードは、1 - 55 桁目に ARRA および ARRB 項目が交互形式で入っています。

図 63. 交互形式の 1 つの配列ファイルの配列レコード

```
*....+....1....+....2....+....3....+....4....+....5....+....6....+....*
      DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++
      DARRA             S              6A   DIM(6) PERRCD(1) CTDATA
      DARRB             S              5  0 DIM(6) ALT(ARRA)
      DARRGRAPHIC       S              3G   DIM(2) PERRCD(2) CTDATA
      DARRC             S              3A   DIM(2) ALT(ARRGRAPHIC)
      DARRGRAPH1        S              3G   DIM(2) PERRCD(2) CTDATA
      DARRGRAPH2        S              3G   DIM(2) ALT(ARRGRAPH1)
**CTDATA ARRA
345126  373
38A437  498
39K143 1297
40B125   93
41C023 3998
42D893   87
**CTDATA ARRGRAPHIC
ok1k2k3iabcok4k5k6iabc
**CTDATA ARRGRAPH1
ok1k2k3k4k5k6k1k2k3k4k5k6i
```

図 64. 交互形式の 1 つの配列ファイルの配列レコード

配列の検索

配列を検索するために以下のものを使用することができます。

- LOOKUP 命令コード
- %LOOKUP 組み込み関数
- %LOOKUPLT 組み込み関数
- %LOOKUPLE 組み込み関数
- %LOOKUPGT 組み込み関数
- %LOOKUPGE 組み込み関数

LOOKUP 命令コードの詳細については、以下を参照してください。

- 185 ページの『指標による配列の検索』
- 『指標を使用しない配列の検索』
- 582 ページの『LOOKUP (テーブルまたは配列エレメントの検索)』

%LOOKUPxx 組み込み関数の詳細については、441 ページの『%LOOKUPxx (配列エレメントの検索)』を参照してください。

指標を使用しない配列の検索

指標なしで配列を検索する時には、結果の標識の状況 (オンまたはオフ) を使用して、一定のエレメントが配列に存在するかどうかを判別してください。指標なしの

配列の検索は、フィールドが配列エレメントのリストに存在するかどうかを判別するために、入力データの妥当性検査に使用することができます。一般的には等しい LOOKUP が使用されます。

演算仕様の演算項目 1 に検索引き数 (名前を指定した配列で一致を見つけたいデータ) を指定し、配列名を演算項目 2に指定します。

演算項目 2 には検索される配列の名前を指定します。少なくとも 1 つの結果の標識を指定しなければなりません。同じ LOOKUP 演算命令で高と低の両方に指定を行なってはいけません。配列が順序 (ASCEND キーワードまたは DESCEND キーワード) になっていない場合には、高と低に結果の標識を指定してはいけません。制御レベル標識および条件標識 (7 - 11 桁目に指定) も使用することができます。結果フィールドは使用することができません。

検索は、配列の先頭から始まり、配列の終わりになるか、あるいは検索の条件が満たされると終わります。行なう検索のタイプ (等しい、高、低) を満たす配列エレメントが見つかると、必ず結果の標識がオンに設定されます。

図 65 は、指標なしの配列の LOOKUP を示したものです。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IT.A.FRlen+......A.Device+.Keywords++++++++++++++++++++++++++++++++
FARRFILE   IT   F    5        DISK
F*
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DDPTNOS           S              5S 0 DIM(50) FROMFILE(ARRFILE)
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* LOOKUP 演算命令が処理され、検索引き数 (DPTNUM) に等しいエレメント DPTNOS が
C* 見つかると、標識 20 がオンに設定されます。
C     DPTNUM        LOOKUP    DPTNOS                                  20
```

図 65. 指標なしの配列の LOOKUP 演算命令

部門番号を含む ARRFILE は、配列ファイル指定 (18 桁目の T) のあるファイル仕様に入力ファイルとして (17 桁目の I) 定義されます。このファイルはプログラム記述 (22 桁目の F) で、各レコードは 5 桁の長さ (27 桁目の 5) です。

定義仕様では、ARRFILE は、配列 DPTNOS を含むものとして定義されています。この配列には 50個の項目 (DIM(50)) が入っています。各項目は、小数点以下の桁数 (41 - 42 桁目) がゼロの 5 桁の長さで (33 - 39 桁目) です。レコードごとに 1 つの部門番号を入れることができます (PERRCD のデフォルトは 1)。

指標による配列の検索

どのエレメントが LOOKUP 検索を満たすかを判別するために、検索を配列中の特定のエレメントから開始します。このタイプの検索を行なうには、指標のない配列の場合と同様に演算仕様に指定を行なってください。しかし、演算項目 2 では、検索される配列の名前を後に括弧付き数値フィールド (小数点以下の桁数がゼロ) を付けて入力してください。この数値フィールドは、検索が開始されるエレメントの番号を含むものです。この数値固定情報または数値フィールドは配列中の一定のエレメントを示すので、指標と呼ばれています。この指標は、検索を満たすエレメント番号で更新されるか、あるいは検索が失敗した場合には、ゼロに設定されます。

1 以外のエレメントから始まる検索を満たすエレメントの存在をテストするために、指標として数値固定情報を使用することができます。

指標のない配列に適用される他のすべての規則は、指標付きの配列に適用されます。

図 66 は、指標付きの配列での LOOKUP を示したものです。この例は、 185 ページの図 65 と同じ部門番号の配列 (DPTNOS) を示しています。しかし、部門記述の代替配列 (DPTDSC) も定義されています。DPTDSCの各エレメントは 20 桁の長さです。ファイルに配列全体を初期化するための十分なデータがない場合には、DPTNOS 中の残りのエレメントはゼロを埋め込まれ、DPTDSC 中の残りのエレメントはブランクを埋め込まれます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IT.A.FRlen+......A.Device+.Keywords++++++++++++++++++++++++++++++++
FARRFILE   IT   F   25            DISK
F*
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DDPTNOS           S              5S 0 DIM(50) FROMFILE(ARRFILE)
DDPTDSC           S             20A   DIM(50) ALT(DPTNOS)
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* Z-ADD 演算命令が DPTNOS 中の最初のエレメントで LOOKUP を開始します。
C                   Z-ADD     1             X                 3 0
C* LOOKUP が成功し、検索引き数 DPTNUM に等しい項目を含むエレメントが見つかると、
C* 標識 20 がオンに設定され、MOVE 演算命令で部門番号に対応する部門記述が
C* DPTNAM に入れられます。
C     DPTNUM        LOOKUP    DPTNOS(X)                              20
C* 検索引き数に等しいエレメントが見つかった場合には、DPTDSC のエレメント X が
C* DPTNAM に移動されます。
C                   IF            *IN20
C                   MOVE      DPTDSC(X)     DPTNAM          20
C                   ENDIF
```

図 66. 指標付きの配列の LOOKUP 演算命令

配列の使用

配列は、入力仕様、出力仕様、または演算仕様で使用することができます。

演算での配列の指定

配列全体または配列中の個々のエレメントを演算仕様に指定することができます。個々のエレメントは、フィールドのように処理されます。

OVERLAY キーワードで定義された不連続配列は、MOVEA 演算命令でも、PARM 演算命令の結果フィールドにも使用することはできません。

配列全体を指定するには、配列名だけを使用します。これは、演算項目 1、演算項目 2、または結果フィールドとして使用することができます。配列名では次の演算命令を使用することができます。

ADD	ADDDUR	CHECK	CHECKR	CLEAR
DEFINE	DIV	EVAL	EXTRCT	LOOKUP

MOVE	MOVEL	MOVEA	MULT	PARM
RESET	SCAN	SORTA	SQRT	SUB
SUBDUR	XFOOT	Z-ADD	Z-SUB	

他のいくつかの演算命令は配列エレメントだけで使用することができ、配列名単独で使用することができません。こうした演算命令には次のものが含まれますが、それに限定されていません。

BITON	BITOFF	CABxx	CAT	COMP
DO	DOU	DOUxx	DOW	DOWxx
IF	IFxx	MVR	SUBST	TESTB
TESTN	TESTZ	WHEN	WHENxx	

指標のない配列名 (たとえば、ARRAY) または指標としてアスタリスクの付いた配列名 (たとえば、ARRAY(*)) で指定すると、一定の演算命令が配列中の各エレメントに反復されます。これらは次のものです。

ADD	ADDDUR	DIV	EVAL	EXTRCT
MOVE	MOVEL	MULT	SQRT	SUB
Z-ADD	Z-SUB			

指標なしに配列名を指定すると、こうした演算命令に次の規則が適用されます。

- 演算項目 1、演算項目 2、および結果フィールドが同じ数のエレメントをもつ配列である場合には、演算命令ではすべての配列から最初のエレメントが使用され、次にすべての配列から 2 番目のエレメントが使用されます。このようにして、配列中のすべてのエレメントが処理されます。配列に同じ数の項目がない場合には、一番少ない数のエレメントをもつ配列の最後のエレメントが処理されると、演算命令が終了します。ADD、SUB、MULT、および DIV の演算命令に演算項目 1 が指定されていないと、演算項目 1 は結果フィールドと同じであると見なされます。
- 演算項目の 1 つがフィールド、リテラル、または表意定数であり、他の演算項目および結果フィールドが配列であると、演算命令は短いほうの配列の全エレメントに対して演算命令が一度実行されます。すべての演算命令で同じフィールド、リテラル、または表意定数が使用されます。
- 結果フィールドは常に配列でなければなりません。
- 命令コードで演算項目 2 だけが使用され (たとえば、Z-ADD、Z-SUB、SQRT、ADD、SUB、MULT、または DIV では演算項目 1 が指定されない場合があります)、結果フィールドが配列である場合には、演算命令は配列中のすべてのエレメントに対して 1 回実行されます。演算項目 2 が配列でない場合には、すべての演算命令で同じフィールドまたは固定情報が使用されます。
- 処理される演算命令の数のために結果の標識 (71 - 76 桁目) は使用することができません。

注: EVAL 演算命令で使用される時には、%ADDR(arr) と %ADDR(arr(*))は同じ意味をもちません。詳細については、405 ページの『%ADDR (変数のアドレスの取り出し)』を参照してください。

配列の分類

SORTA 命令コードを使用して、配列を分類することができます。配列は、定義仕様で配列に指定された順序 (昇順または降順) に分類されます。

配列の一部をキーとして使用した分類

OVERLAY キーワードを使用して、1 つの配列を別の配列にオーバーレイすることができます。たとえば、名前と給与が入っている 1 つの基本配列および 2 つのオーバーレイ配列 (1 つは名前用で、1 つは給与用)を使用することができます。ここで、該当のオーバーレイ配列で分類を行なうことによって、基本配列を名前か給与のいずれかで分類することができます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D                 DS
D Emp_Info                      50    DIM(500) ASCEND
D   Emp_Name                    45    OVERLAY(Emp_Info:1)
D   Emp_Salary                   9P 2 OVERLAY(Emp_Info:46)
D
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   C* 次の SORTA は Emp_Info を従業員名で分類します。Emp_Info の
C* エレメントの順序を決定するために、Emp_Name の順序が使用されます。
C                   SORTA     Emp_Name
C* 次の SORTA は Emp_Info を従業員の給与で分類します。Emp_Info の
C* エレメントの順序を決定するために、Emp_Salary の順序が使用されます。
C                   SORTA     Emp_Salary
```

図 67. OVERLAY が指定された SORTA 演算命令

配列出力

配列全体は、LR 標識がオンに設定された時にプログラムの終わりでのみ書き出されます。配列全体が書き出されるということを示すには、定義仕様上に TOFILE キーワードを使用して出力ファイルの名前を指定します。このファイルは、ファイル仕様で順次編成入出力共用ファイルとして記述しなければなりません。

ファイルが入出力共用ファイルで、物理ファイルとして外部記述されている場合には、プログラムの終わりで配列中の情報は、プログラムの始めで配列に読み込まれた情報に置き換わります。論理ファイルは、予期しない結果を生むことがあります。

配列全体が出力レコードに書き込まれる場合には (出力仕様を使用)、レコードの他のフィールドと一緒に配列を記述します。

- 出力仕様の 30 ～ 43 桁目には、定義仕様で使用される配列名が入っていなければなりません。
- 出力仕様の 47 ～ 51 桁目には、配列の最後のエレメントが終わるレコード位置が入っていなければなりません。編集コードを指定した場合には、「終わりの位置」にはブランクおよび編集コードのための拡張部分が含まれていなければなりません (このセクションで次にリストしてある「配列全体の編集」を参照してください)。

出力標識 (21 ～ 29 桁目) を指定することができます。ゼロ抑制 (44 桁目)、後で消去 (45 桁目)、およびデータ形式 (52 桁目) の項目は、配列中のすべてのエレメントに関係します。

配列全体の編集

配列全体に編集が指定されると、配列のすべてのエレメントが編集されます。各種のエレメントに異なる編集が必要な場合には、それを個別に参照してください。

配列全体に編集コードが指定されている時には (44 桁目)、配列中のエレメント間に 2 つのブランクが自動的に挿入されます。最初のエレメントを除いて、配列中のすべてのエレメントの左にはブランクがあります。編集語が指定された時には、ブランクは挿入されません。編集語には、挿入されるすべてのブランクが含まれていなければなりません。

テーブル

配列の説明はテーブルに適用されますが、次の 2 つの相違点があります。

活動	差
定義	テーブル名は、文字 TAB で始まっている固有の記号名でなければなりません。
テーブル・エレメントの使用および変更	テーブルのエレメントは、一度に 1 つしかアクティブにできません。アクティブ・エレメントを参照するためにはテーブル名が使用されます。
検索	LOOKUP 演算命令は、テーブルでは指定の仕方が異なります。テーブルの検索には別の組み込み関数が使用されます。

注: サブプロシージャー中にテーブルを定義することはできません。

テーブルの検索には以下のものを使用することができます。
- LOOKUP 命令コード
- %TLOOKUP 組み込み関数
- %TLOOKUPLT 組み込み関数
- %TLOOKUPLE 組み込み関数
- %TLOOKUPGT 組み込み関数
- %TLOOKUPGE 組み込み関数

LOOKUP 命令コードの詳細については、以下を参照してください。
- 『1 つのテーブルでの LOOKUP』
- 190 ページの『2 つのテーブルでの LOOKUP』
- 582 ページの『LOOKUP (テーブルまたは配列エレメントの検索)』

%TLOOKUPxx 組み込み関数の詳細については、471 ページの『%TLOOKUPxx (テーブル・エレメントの検索)』 を参照してください。

1 つのテーブルでの LOOKUP

1 つのテーブルを検索する時には、演算項目 1、演算項目 2、および少なくとも 1 つの結果の標識を指定しなければなりません。条件標識 (7 ～ 11 桁目に指定) 使用することもできます。

行なう検索のタイプ (等、高、低) を満たすテーブル・エレメントが見つかると、必ずそのテーブル・エレメントがテーブルの現行エレメントとなります。検索が失敗した場合には、前の現行エレメントが現行エレメントのままになります。

LOOKUP が最初に成功するまでは、最初のエレメントが現行エレメントになっています。

結果の標識は、検索の結果を反映します。この標識がオンである場合には、それは検索が成功したことを反映し、検索を満たすエレメントが現行エレメントになります。

2 つのテーブルでの LOOKUP

検索で 2 つのテーブルが使用される時にも、実際には 1 つしか検索されません。検索条件 (高、低、等) が満たされると、対応するエレメントが使用可能になります。

演算項目 1 には検索引き数が含まれていなければならず、演算項目 2 には検索されるテーブルの名前が含まれていなければなりません。結果フィールドは、使用可能になるデータをもつテーブルの名前を指定しなければなりません。結果の標識も使用する必要があります。必要な場合、制御レベル標識および条件標識を 7 - 11 桁目に指定することができます。

使用される 2 つのテーブルは、同数の項目をもっていなければなりません。検索されるテーブルに 2番目のテーブルより多くのエレメントが含まれている場合にも、検索条件が満たされる可能性があります。しかし、検索テーブルで見つかったエレメントに対応するエレメントが 2 番目のテーブルにない場合もあります。望ましくない結果が起こることがあります。

注: LOOKUP 以外の演算命令でテーブル名を指定した場合には、LOOKUP が成功する前には、テーブルは最初のエレメントに設定されています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* LOOKUP 演算命令では、EMPNUM という名前のフィールドの内容に等しい項目
C* を TABEMP から検索します。等しい項目が TABEMP 中で見つかると、標識 09
C* がオンに設定され、TABEMP 項目および TABPAY 中のその関連項目が現行エレメント
C* となります。
C     EMPNUM        LOOKUP    TABEMP        TABPAY                   09
C* 標識 09 がオンに設定されると、HRSWKD という名前のフィールドの内容に
C* TABPAY 中の現行エレメントの値が乗算されます。
C                   IF        *IN09
C     HRSWKD        MULT(H)   TABPAY        AMT              6 2
C                   ENDIF
```

図 68. 等しい項目の検索

LOOKUP 演算命令で検索されるテーブル・エレメントの指定

LOOKUP 以外の演算命令でテーブル名が使用されると、常にテーブル名は、最後に成功した検索によって取り出されたデータを実際に示します。したがって、このようにしてテーブル名が使用されると、テーブルのエレメントを算術演算命令で使用することができます。

テーブルが LOOKUP 演算命令の演算項目 1 として使用された場合には、現行エレメントが検索引き数として使用されます。こうしてテーブルのエレメント自身が検索引き数となることができます。

テーブルは、LOOKUP 演算命令以外の演算命令の結果フィールドとしても使用することができます。この場合には、現行エレメントの値は、演算仕様によって変更されます。このようにしてテーブルの内容を算術演算命令で変更することができます。図 69 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C     ARGMNT        LOOKUP    TABLEA                                  20
C* エレメントが見つかると、1.5 を乗算されます。MULT 演算命令の前のテーブル全体
C* の内容が 1323.5、-7.8、および 113.4 であり、ARGMNT の値が -7.8 である
C* 場合には、2 番目のエレメントが現行エレメントとなります。
C* MULT 演算命令の後、ここでテーブル全体は、1323.5、-11.7、および 113.4
C* という値をもつことになります。
C* 2 番目のエレメントが LOOKUP によって設定された現行エレメントであるので、2 番目の
C* エレメントだけが変更されたということに注意してください。
C                   IF        *IN20
C     TABLEA        MULT      1.5           TABLEA
C                   ENDIF
```

図 69. LOOKUP 演算命令で検索されるテーブル・エレメントの指定

第 13 章 数値フィールドの編集

編集は、通貨記号、コンマ、ピリオド、負符号、浮動負符号などを含めて数値フィールドを読みやすくするための手段です。右端の桁からフィールドの先頭への負符号の移動、ゼロ・フィールドをブランク化、配列中でのスペーシング、日付フィールドの編集、通貨記号またはアスタリスク保護なども行なえるようにします。

フィールドの編集は、編集コードまたは編集語によって行なうことができます。出力仕様を使用してフィールドを編集後の形式で印刷するか、組み込み関数の %EDITC (編集コード) および %EDITW (編集語) を使用して、演算仕様でフィールドの編集後の値を入手することができます。

注: 入力フィールド・パーツおよび静的テキスト・パーツの編集方法については、*VisualAge for RPG プログラミング*, SC88-5607-05を参照してください。

編集されていないフィールドを印刷すると、フィールドは、内部的に表現されるそのままの形で現われます。次の例は、数値出力フィールドを編集する必要がある理由を示しています。

フィールドのタイプ	コンピューター中のフィールド	未編集フィールドの印刷	編集済みフィールドの印刷
英数字	JOHN T SMITH	JOHN T SMITH	JOHN T SMITH
数字 (正)	0047652	0047652	47652
数字 (負)	004765r	004765r	47652-

未編集の英数字フィールドおよび未編集の正数数値フィールドは印刷されても読みやすくなりますが、未編集の負数数値フィールドは、数値でない 'r' を含んでいるので、混乱を生じます。この 'r' は、数字 2 とフィールドの負符号が組み合わさったものです。この数字と負符号の結合は、フィールドの 1 つの桁を符号用に取っておく必要がないように行なわれます。この組み合わせはコンピューターにフィールドを保管するためには便利ですが、出力を読みにくくしています。数値フィールドは、印刷の前に編集する必要があります。

編集コード

編集コードは、事前定義パターンに従って数値フィールドを編集する手段となります。編集コードは、単純 (X, Y, Z) と組み合わせ (1 - 4、A - D、J - Q) という 2 つの類別に分かれています。編集コードは、編集されるフィールドの出力仕様の 44 桁目に入力します。あるいは編集コードを演算仕様で %EDITC 組み込み関数の 2 番目のパラメーターとして指定することができます。

単純編集コード

単純編集コードは、句読点なしに数値フィールドを編集するために使用することができます。これらのコードおよびその働きは次の通りです。

- X 編集コードは、正フィールドの場合に 16 進数 3 が符号となるようにします。しかし、通常ではシステムがこれを行なうので、このコードを指定する必要はありません。先行ゼロは抑制されません。X 編集コードは、負数を変更しません。
- Y 編集コードは通常は 3 ～ 9 桁の日付フィールドを編集するために使用されます。最初の区切り文字の直前までの日付フィールドの左端のゼロを抑制します。年、月、日を区切るために、スラッシュが挿入されます。編集形式を変更するには、制御仕様で DATEDIT(fmt{区切り文字}) および DECEDIT('値')キーワードを使用することができます。
- Y 編集コードは、*YEAR、*MONTH、および *DAY には無効です。
- Z 編集コードは、数値フィールドから符号 (正または負) を除去し、先行ゼロを抑制します。フィールドに小数点は入れられず、印刷されません。

組み合わせ編集コード

組み合わせ編集コード (1 ～ 4、A ～ D、J ～ Q) は、数値フィールドに句読点を入れます。

10 進数区切り記号にどの文字が使用されるか、また先行ゼロが抑制されるかどうかは制御仕様の DECEDITキーワードが決定します。小数点が印刷されるかどうか、印刷されるとしたらどこに印刷されるかは、ソース・フィールドの小数点以下の桁数が決定します。ソース・フィールドに小数点以下の桁数が指定され、さらにゼロ・バランスの抑制が指定されている場合には、10 進数区切り記号は、フィールドがゼロでない場合にのみ印刷されます。ゼロ・バランスが印刷されない場合には、ゼロ・フィールドはブランクとして印刷されます。

ゼロ・バランスが印刷され、フィールドがゼロに等しい場合には、次のいずれかが印刷されます。

- n 個のゼロが後に続く 10 進数区切り記号。n は、フィールドの小数部を示します。
- 小数部が指定されていない場合には、フィールドの単位桁にゼロ。

12 個の組み合わせ編集コードのいずれかで浮動通貨記号またはアスタリスク保護を使用することができます。浮動通貨記号を指定するには、出力仕様の 53 - 55 桁目に通貨記号をコーディングし、さらに編集されるフィールドについて 44 桁目に編集コードをコーディングします。浮動通貨記号は、最初の有効数字の左側に現われます。ゼロ・バランスを抑制する編集コードが使用された時には、ゼロ・バランスでは浮動通貨記号は印刷されません。ゼロ・バランスを抑制する編集コードが使用された時には、ゼロ・バランスでは通貨記号は現われません。

制御使用で CURSYM キーワードによって通貨記号が指定されない限り、 ドル記号 ($) が通貨記号として使用されます。

編集されるフィールドの編集コードと一緒に出力仕様の 53 - 55 桁目にアスタリスク固定情報をコーディングすると、抑制されたゼロごとに 1 つのアスタリスクが印刷されます。ゼロ・バランスのソース・フィールドにはすべてアスタリスクのフィ

ールドが印刷されます。組み込み関数 %EDITC を使用してアスタリスク保護を指定するには、3 番目のパラメーターとして *ASTFILL を指定してください。

単純 (X、Y、Z) 編集コードと一緒にアスタリスク充てんおよび浮動通貨記号を使用することはできません。

組み込み関数 %EDITC の場合、3 番目のパラメーターに浮動通貨記号を指定します。プログラムの通貨記号を使用するには、*CURSYM を指定します。別の通貨記号を使用するには、長さ 1 の文字固定情報を指定してください。

通貨記号は、アスタリスク充てんの前に現われます (固定通貨記号)。これには、次をコーディングした 2つの出力仕様が必要です。

1. 最初の出力仕様の 53 桁目に通貨記号固定情報を入れます。47 - 51 桁目に指定する「終わりの位置」は、編集済みフィールドの先頭の前の 1 スペースとしてください。
2. 2 番目の出力仕様では、30 - 43 桁目に編集フィールド、44 桁目に編集コード、47 - 51 桁目に編集フィールドの「終わりの位置」、53 - 55 桁目に '*' を入れます。

   これは、次のように通貨記号を %EDITC組み込み関数に連結することによって、%EDITC 組み込み関数を使用して行なうことができます。

```
C           EVAL       X = '\' + %EDITC(N: 'A' : *ASTFILL)
```

配列全体を印刷するために編集コードが使用されると、配列の各エレメントの前に 2 つのブランクが付きます (最初のエレメントは除きます)。

注: %EDITC 組み込み関数を使用して配列を編集することはできません。

表 20では、組み合わせ編集コードの働きを要約してあります。これらのコードでは、左にリストされた形式のフィールドが編集されます。負フィールドは、図の一番上に示したように符号なし、CR、負符号 (–)、または浮動負符号によって、読みやすくすることができます。

表 20. 組み合わせ編集コード

		負バランス標識			
グループ区切り文字付きの印刷	**ゼロ・バランスの印刷**	**符号なし**	**CR**	**–**	**浮動負符号**
Yes	Yes	1	A	J	N
Yes	No	2	B	K	0
No	Yes	3	C	L	P
No	No	4	D	M	Q

編集に関する考慮事項

編集コードのいずれかを指定する時には、次を行なってください。

- プリンター・ファイル以外のファイルの場合、フィールドの編集には注意が必要です。プリンター・ファイル以外のファイルのフィールドを編集する場合には、編集済みフィールドの内容およびそれについて行なわれる演算命令の影響に注意

してください。たとえば、このファイルを入力として使用する場合には、編集で書き出されるフィールドは、数値フィールドではなく、文字フィールドと見なされます。

- 編集演算命令によって追加されたデータについて考慮する必要があります。追加される句読点の量によって、出力フィールドの全体の長さが増します。出力仕様で編集する時にこうした追加された文字について考慮しないと、出力フィールドがオーバーラップすることがあります。
- 出力に指定された「終わりの位置」は、編集済みフィールドの「終わりの位置」です。たとえば、J ～ M のいずれかの編集コードを指定した場合には、「終わりの位置」は負符号 (フィールドが正の場合にはブランク) の位置です。
- コンパイラーは、符号なし数値フィールドであっても符号の文字位置を割り当てます。

編集コードの要約

表 21では、編集コードおよびそのオプションが要約されています。このテーブルの単純化されたバージョンは、出力仕様の 45 ～ 70 桁目に印刷されます。 197 ページの表 22は、編集後にフィールドがどのようになるかを示したものです。

198 ページの表 23は、出力に「終わりの位置」が指定された場合に同じフィールドについて、各種の編集コードが及ぼす影響を示しています。

表 21. 編集コード

				DECEDIT キーワードのパラメーター				
編集コード	コンマ	小数点	負バランスの符号	'.'	','	'0,'	'0.'	ゼロ抑制
1	Yes	Yes	符号なし	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	Yes
2	Yes	Yes	符号なし	ブランク	ブランク	ブランク	ブランク	Yes
3		Yes	符号なし	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	Yes
4		Yes	符号なし	ブランク	ブランク	ブランク	ブランク	Yes
A	Yes	Yes	CR	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	Yes
B	Yes	Yes	CR	ブランク	ブランク	ブランク	ブランク	Yes
C		Yes	CR	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	Yes
D		Yes	CR	ブランク	ブランク	ブランク	ブランク	Yes
J	Yes	Yes	– 5 (負)	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	Yes
K	Yes	Yes	– (負)	ブランク	ブランク	ブランク	ブランク	Yes
L		Yes	– (負)	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	Yes
M		Yes	– (負)	ブランク	ブランク	ブランク	ブランク	Yes
N	Yes	Yes	– (浮動符号)	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	Yes

表 21. 編集コード (続き)

				DECEDIT キーワードのパラメーター				
編集コード	**コンマ**	**小数点**	**負バランスの符号**	**'.'**	**','**	**'0,'**	**'0.'**	**ゼロ抑制**
O	Yes	Yes	– (浮動符号)	ブランク	ブランク	ブランク	ブランク	Yes
P		Yes	– (浮動符号)	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	Yes
Q		Yes	– (浮動符号)	ブランク	ブランク	ブランク	ブランク	Yes
X[1]								Yes
Y[2]								Yes
Z[3]								Yes

[1] X 編集コードは、正の値の場合に 16 進数 3 が符号となるようにします。 システムがこれを行なうので、通常はこのコードを指定する必要はありません。

[2] Y 編集コードは、最初の区切り文字直前までの日付フィールドの左端ゼロを抑制します。また、Y 編集コードは、次のパターンに従って月、日、および年の間にスラッシュ (/) を挿入します。

```
   nn/n
   nn/nn
   nn/nn/n
   nn/nn/nn
  nnn/nn/nn
   nn/nn/nnnn
  nnn/nn/nnnn
   nn/nn/nnnn
nnnnn/nn/nn
```

[3] Z 編集コードは、数値フィールドから符号 (正または負) を除去し、先行ゼロを抑制します。

表 22. 編集コードの使用例

編集コード	**正数 - 2 桁の小数点以下の桁数**	**正数 - 小数点以下の桁数なし**	**負数 - 3 桁の小数点以下の桁数**	**負数 - 小数点以下の桁数なし**	**ゼロ・バランス - 2 桁の小数点以下の桁数**	**ゼロ・バランス - 小数点以下の桁数なし**
未編集	1234567	1234567	00012b[4]	000000	000000	
1	12,345.67	1,234,567	.120	120	.00	0
2	12,345.67	1,234,567	.120	120		
3	12345.67	1234567	.120	120	.00	0
4	12345.67	1234567	.120	120		
A	12,345.67	1,234,567	.120CR	120CR	.00	0
B	12.345.67	1,234,567	.120CR	120CR		
C	12345.67	1234567	.120CR	120CR	.00	0
D	12345.67	1234567	.120CR	120CR		
J	12,345.67	1,234,567	.120–	120–	.00	0
K	12,345,67	1,234,567	.120–	120–		
L	12345.67	1234567	.120–	120–	.00	0
M	12345.67	1234567	.120–	120–		

表 22. 編集コードの使用例 (続き)

編集コード	正数 - 2 桁の小数点以下の桁数	正数 - 小数点以下の桁数なし	負数 - 3 桁の小数点以下の桁数	負数 - 小数点以下の桁数なし	ゼロ・バランス - 2 桁の小数点以下の桁数	ゼロ・バランス - 小数点以下の桁数なし
N	12,345.67	1,234,567	–.120	–120	.00	0
O	12,345,67	1,234,567	–.120	–120		
P	12345.67	1234567	–.120	–120	.00	0
Q	12345.67	1234567	–.120	–120		
X[1]	1234567	1234567	00012b[4]	000000	000000	
Y[2]			0/01/20	0/01/20	0/00/00	0/00/00
Z[3]	1234567	1234567	120	120		

[1] X 編集コードは、正の値の場合に 16 進数 F が符号となるようにします。 システムがこれを行なうので、通常はこのコードを指定する必要はありません。

[2] Y 編集コードは、最初の区切り文字の直前までの日付フィールドの左端のゼロを抑制します。 また、Y 編集コードは、次のパターンに従って月、日、および年の間にスラッシュ (/) を挿入します。

```
   nn/n
   nn/nn
   nn/nn/n
   nn/nn/nn
  nnn/nn/nn
   nn/nn/nnnn  19 桁目の M、D、またはブランクで使用される形式
  nnn/nn/nnnn  19 桁目の M、D、またはブランクで使用される形式
 nnnn/nn/nn    19 桁目の Y で使用される形式
nnnnn/nn/nn    19 桁目の Y で使用される形式
```

[3] Z 編集コードは、数値フィールドから符号 (正または負) を除去し、数値フィールドの先行ゼロを抑制します。

[4] b は、ブランクを表します。これは、負のゼロが印刷可能文字に対応していない場合に起こることがあります。

表 23. 「終わりの位置」での編集コードの影響

	負数、2 桁の小数点以下の桁数。「終わりの位置」は 10 として指定されています。								
	出力印刷位置								
編集コード	3	4	5	6	7	8	9	10	
[1] r は負数 2 を表します。									
未編集				0	0	4	1	r[1]	
1					4	.	1	2	
2					4	.	1	2	
3					4	.	1	2	
4					4	.	1	2	
A			4	.	1	2	C	R	
B			4	.	1	2	C	R	
C			4	.	1	2	C	R	

表 23. 「終わりの位置」での編集コードの影響 (続き)

	負数、2 桁の小数点以下の桁数。「終わりの位置」は 10 として指定されています。								
	出力印刷位置								
編集コード	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
D			4	.	1	2	C	R	
J				4	.	1	2	–	
r				4	.	1	2	–	
L				4	.	1	2	–	
M				4	.	1	2	–	
N				–	4	.	1	2	
O				–	4	.	1	2	
P				–	4	.	1	2	
Q				–	4	.	1	2	
X				0	0	4	1	r[1]	
Y			0	/	4	1	/	2	
Z						4	1	2	

編集語

編集コードでは満たすことができない編集要件がある場合には、編集語を使用することができます。編集語は、出力仕様の 53 - 80 桁目に指定する文字リテラルまたは名前付き固定情報です。これは、数字の編集パターンを記述するもので、次のものを直接に指定できるようにします。

- ブランク・スペース
- コンマ、小数点、およびその位置
- 不要ゼロの抑制
- 先行アスタリスク
- 通貨記号およびその位置
- 固定情報文字の追加
- 負標識として負符号または CR を出力

編集語は、システムが出力を作成するためにソース・データに適用するテンプレートとして使用されます。

編集語は、出力仕様に直接に指定するか、出力仕様の編集語フィールドの名前付き固定情報名をもつ名前付き固定情報として指定することができます。組み込み関数 %EDITW (編集語) を使用して、演算仕様でフィールドの編集済みの値を入手することができます。

編集語として使用される名前付き固定情報は、115 文字に制限されています。

編集語のコーディング方法

編集語を使用するには、出力仕様を以下に示しているようにコーディングします。

位置　　項目

21〜29 条件標識を入れることができます。

30〜43 編集されるデータが取られる数値フィールドの名前を入れます。

44 編集コード: ソース・データの編集に編集語を使用しようとする場合には、ブランクでなければなりません。

45 この位置の “B” は、編集および出力された後で、ソース・データがゼロまたはブランクに設定されるということを示します。そうでない場合には、ソース・データは未変更のままです。

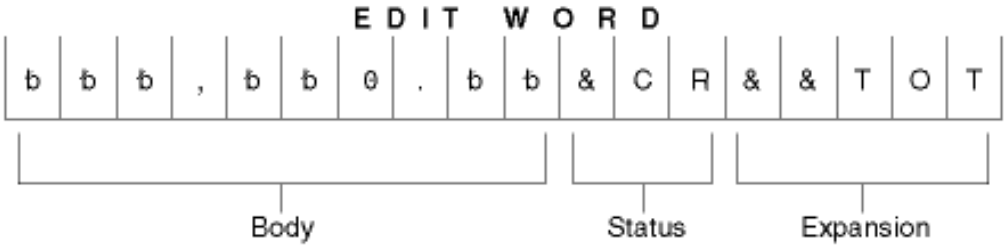
47〜51 出力レコード中のフィールドの終わり (右端) の位置を識別します。

53 〜 80 編集語: 最大 26 文字の長さにすることができますが、名前付き固定情報でない限り、アポストロフィで囲む必要があります。先行アポストロフィを入力するか、あるいは名前付き固定情報名を 53 桁目から始めてください。名前付き固定情報でない限り、編集語は 54 桁目から始めなければなりません。

演算仕様で編集語を使用して編集を行なうには、組み込み関数 %EDITW を使用し、編集される値を最初のパラメーターとして指定し、編集語を 2 番目のパラメーターとして指定します。

編集語の部分

編集語 (出力仕様の 53 - 80 桁目にコーディング) は、本体、状況部分、拡張部分という 3 つの部分からなります。以下は、編集語の 3 つの部分を示したものです。



本体は、ソース・データ・フィールドから出力レコードに転送される数字のためのスペースです。本体は、編集語の左端の位置から始まります。編集語本体のブランクの数 (それに 1 つのゼロまたはアスタリスクを加えたもの) は、編集されるソース・データ・フィールドの桁数に等しいか、それより大きくなければなりません。本体は、数字で置き換えることができる右端の文字で終わります。

状況部分は、負標識 (2 文字 CR か負符号 (–) のいずれか) のために確保するスペースを定義します。指定された負標識は、ソース・データが負である場合にだけ出力されます。編集語で最後の置き換え可能 (ブランク、ゼロ抑制文字) と負標識の間にあるすべての文字も、ソース・データが負である場合にだけ負標識と共に出力されます。ソース・データが正である場合は、これらの状況部分はブランクで置き換えられます。CR または – 標識のない編集語は、状況位置をもちません。

状況部分は、編集語中で最後のブランクの後に入力しなければなりません。最後のブランクの後に複数の CR が続いている場合には、最初の CR だけが状況部分とし

て扱われます。残りの CR は、固定情報として扱われます。状況部分と見なされる負符号の場合、編集語中の最後の文字でなければなりません。

拡張部分は、状況部分の後に入力される一連のアンパーサンドと固定情報文字です。アンパーサンドは、出力中でブランク・スペースで置き換えられます。固定情報は、そのまま出力されます。状況部分が指定されていない場合は、拡張部分は本体の後に続きます。

編集語の本体の形成

次の文字は、編集語の本体で使用されると特殊な意味をもちます。

アンパーサンド: 編集済みフィールドにブランクを入れます。下の例は、電話番号を編集するために使用されるものです。固定情報 AREAを印刷するには、最初の位置にゼロが必要であるということに注意してください。

編集語	ソース・データ	出力レコードに現われるデータ
'0AREA&bbb&NO.&bbb-bbbb;'	4165551212	bAREAb416bNO.b555-1212

アスタリスク: また編集語の本体の最初のアスタリスクは、ゼロ抑制を終わらせます。編集語に入れられた後続のアスタリスクは、固定情報として扱われます (下の「固定情報」を参照してください)。編集語中でこのアスタリスクに続くゼロも、すべて固定情報として扱われます。編集語中では 1 つのゼロ抑制終了文字しか使用することができず、その文字は、編集語中の最初のアスタリスクまたは最初のゼロです。

ゼロ抑制終了文字としてアスタリスクが使用される場合には、抑制されたすべての先行ゼロが出力中でアスタリスクで置き換えられます。そうでない場合には、アスタリスクは、上記で「ゼロ」について説明された場合と同様に先行ゼロを抑制します。

編集語	ソース・データ	出力レコードに現われるデータ
'*bbbbbb.bb'	000000123	*000001.23
'bbbbb*b.bb'	000000000	******0.00
'bbbbb*b.bb**'	000056342	****563.42**

アスタリスクの位置の後に現われる先行ゼロは、先行ゼロとして出力されるということに注意してください。アスタリスクの位置にあるものを含め、抑制された先行ゼロだけがアスタリスクで置き換えられます。

ブランク: ブランクは、出力仕様の 30 - 43 桁目のフィールド名によって指定されたソース・データ・フィールドの対応する位置の文字で置き換えられます。ブランク位置は、数字部分として参照されます。

固定情報: 編集語の本体に入力された他のすべての文字は、固定情報として扱われます。ソース・データが出力中で有効数字または先行ゼロを固定情報の左側におくようなものである場合には、固定情報が出力に現われます。そうでない場合には、固定情報は、出力中で抑制されます。コンマも、小数点も固定情報と同じ規則に従

います。以下の例では、ゼロ抑制終了文字の存在だけでなく、ソース・データ中の有効数字の数も固定情報の出力に影響するということに注意してください。

次の編集語は、小切手を印刷するために使用することができます。2 番目のアスタリスクは固定情報として扱われるということ、および 3 番目の例では、最初の有効数字の前の固定情報は出力されないということに注意してください。

編集語	ソース・データ	出力レコードに現われるデータ
'$bbbbbb**DOLLARS&bb&CTS'	000012345	$****123*DOLLARSb45bCTS
'$bbbbbb**DOLLARS&bb&CTS'	000000006	$********DOLLARSb06bCTS
'$bbbbbbb&DOLLARS&bb&CTS'	000000006	$bbbbbbbbbbbbbbbbb6bCTS

日付は、編集語を使用して印刷することができます。

編集語	ソース・データ	出力レコードに現われるデータ
'bb/bb/bb'	010388	b1/03/88
'0bb/bb/bb'	010389	b01/03/89

編集語の最初のオカレンスに続くゼロまたはアスタリスクは、固定情報として扱われるということに注意してください。 – および CR: の場合にも同じことが言えます。

編集語	ソース・データ	出力レコードに現われるデータ
'bb0.bb000'	01234	b12.34000
'bb*.bb000'	01234	*12.34000

通貨記号: 編集語中で最初のゼロ (ゼロ抑制終了文字) の直後にある通貨記号は浮動と呼ばれます。出力中ですべての先行ゼロは抑制され、通貨記号が出力中で最上位の有効数字のすぐ左側に現われます。

編集語	ソース・データ	出力レコードに現われるデータ
'bb,bbb,b$0.bb'	000000012	bbbbbbbbb$.12
'bb,bbb,b$0.bb'	000123456	bbbb$1,234.56

通貨記号が編集語の最初の位置におかれている場合には、通貨記号は、出力中で常にその位置に現われます。これは固定通貨記号と呼ばれています。

編集語	ソース・データ	出力レコードに現われるデータ
'$b,bbb,bb0.bb'	000123456	$bbbb1,234.56
'$bb,bbb,0b0.bb'	000000000	$bbbbbbbb00.00
'$b,bbb,*bb.bb'	000123456	$****1,234.56

編集語中のどこか他の場所にあり、ゼロ抑制終了文字の直後にない通貨記号は、固定情報として扱われます (上記の「固定情報」を参照してください)。

小数部分およびコンマ: 小数部分およびコンマは、編集語の最初の有効数字の左側に現われない限り、編集済み出力では、編集語にあった時と同じ相対位置に現われます。最初の有効数字の左側に現われる場合には、ブランクにされるか、アスタリスクで置き換えられます。

下の例では、すべての先行ゼロは抑制され (デフォルト)、小数点は、その左に有効数字がない限り、印刷されません。

編集語	ソース・データ	出力レコードに現われるデータ
'bbbbbbb'	0000072	bbbbb72
'bbbbbbb.bb'	000000012	bbbbbbbb12
'bbbbbbb.bb'	000000123	bbbbbb1.23

ゼロ: 編集語の本体の最初のゼロは、ゼロ抑制終了文字として解釈されます。このゼロは、ゼロ抑制が終わる所におかれます。編集語に入れられた後続のゼロは、固定情報として扱われます (上記の 「固定情報」を参照してください)。

ソース・データ中の先行ゼロは、ゼロ抑制終了文字まで (ゼロ抑制終了文字を含む) 抑制されます。ゼロ抑制終了文字の位置あるいはその左に現われることになる有効数字は出力されます。

編集語	ソース・データ	出力レコードに現われるデータ
'bbb0bbbbbb'	00000004	bbbb000004
'bbb0bbbbbb'	012345	bbbb012345
'bbb0bbbbbb'	012345678	bb12345678

先行ゼロまたはその右側への拡張部分にゼロ抑制終了文字が含まれている場合には、その位置はブランクで置き換えられます。これは、ソース・データ中にある時と同じ数の先行ゼロを現わしたい場合には、編集語本体がソース・データより広くなければならないということを意味します。

編集語	ソース・データ	出力レコードに現われるデータ
'0bbb'	0156	b156
'0bbbb'	0156	b0156

ゼロ抑制終了文字の右側におかれる固定情報 (コンマおよび小数点を含む) は、ソース・データがなくても出力されます。ゼロ抑制終了文字の左側の固定情報は、ソース・データにこうした固定情報の左側におかれる有効数字がある場合にのみ出力されます。

編集語	ソース・データ	出力レコードに現われるデータ
'bbbbbb0.bb'	000000001	bbbbbbb.01
'bbbbbb0.bb'	000000000	bbbbbbb.00
'bbb,b0b.bb'	00000012	bbbbbb0.12
'bbb,b0b.bb'	00000123	bbbbbb1.23
'b0b,bbb.bb'	00000123	bb0,001.23

編集語の状況部分の形成

次の文字は、編集語の状況部分で使用されると特殊な意味をもちます。

アンパーサンド: 編集済み出力フィールドにブランクを入れます。編集済み出力フィールドにアンパーサンドを入れることはできません。

CR またはマイナス記号: 編集済み出力の符号が正 (+) である場合には、これらの位置はブランクになります。編集済み出力の符号が負 (–) である場合には、これらの位置は未変更のままになります。

次の例では、負の値の指示が追加されています。負符号は、フィールドの値が負である時にのみ印刷されます。CR 記号は、負符号と同じ働きをします。

編集語	ソース・データ	出力レコードに現われるデータ
'bbbbbbb.bb–'	000000123–	bbbbbb1.23–
'bbbbbbb.bb–'	000000123	bbbbbb1.23b

最後の置き換え可能文字と – または CR 記号の間の固定情報は、フィールドが負である場合にのみ印刷されます。そうでない場合には、こうした位置にブランクが印刷されます。ブランクを表すのにアンパーサンドが使用されていることに注意してください。

編集語	ソース・データ	出力レコードに現われるデータ
'b,bbb,bb0.bb&30&DAY&CR'	000000123–	bbbbbbbbb1.23b30bDAYbCR
'b,bbb,bb0.bb&30&DAY&CR'	000000123	bbbbbbbbb1.23bbbbbbbbbb

編集語の拡張部分の形式設定

編集語の拡張部分の文字は、常に書き込まれます。拡張部分にブランクを入れることはできません。 編集済み出力フィールドにブランクが必要な場合には、編集語の本体にアンパーサンドを指定してください。

すべての行に印刷される固定情報を追加することができます。

編集語	ソース・データ	出力レコードに現われるデータ
'b,bb0.bb&CR&NET'	000123–	bbbb1.23bCRbNET
'b,bb0.bb&CR&NET'	000123	bbbb1.23bbbbNET

語の途中にある CR は負フィールド値の指示として検出される場合もあるということに注意してください。 SECRET のような語が必要な場合には、下の例のコーディングを使用してください。

編集語	ソース・データ	出力レコードに現われるデータ
'bb0.bb&SECRET'	12345–	123.45bSECRET
'bb0.bb&SECRET'	12345	123.45bbbbbET
'bb0.bb&CR&&SECRET'	12345	123.45bbbbbSECRET

編集語のコーディング上の規則の要約

編集語には次の規則が適用されます。

- 44 桁目 (編集コード) はブランクでなければなりません。
- 30 - 43 桁目 (フィールド名) には数値フィールドの名前が含まれていなければなりません。
- 編集語は、名前付き固定情報でないかぎり、アポストロフィで囲まなければなりません。先行アポストロフィを入力するか、あるいは名前付き固定情報名を 53 桁目から始めてください。編集語自体は、54 桁目から始めなければなりません。
- 編集語には、編集されるフィールドより多くの数字部分 (ブランク + 初期ゼロまたはアスタリスク) を含めることができますが、少なくすることはできません。編集語の数字部分が編集されるフィールドの桁数より多い場合には、編集前にフィールドに先行ゼロが追加されます。
- ソース・データからの先行ゼロが必要な場合には、編集語に編集されるフィールドより 1 桁多く含まれていなければならず、編集語の高位桁にゼロを入れなければなりません。
- 編集語の本体では、ブランクとゼロ抑制停止文字 (ゼロまたはアスタリスク) だけが桁数としてカウントされます。浮動通貨記号は、桁数としてカウントされません。
- 浮動通貨記号が使用される時には、編集語に含まれるブランクとゼロ抑制停止文字の合計 (桁数) が編集されるフィールドの桁数に等しいか、それより大きくなければなりません。
- 左端のゼロまたはアスタリスクに続くゼロまたはアスタリスクは、すべて固定情報として扱われます。これらは置き換え可能文字ではありません。
- 符号なし整数フィールドを編集する時には、DB および CR を使用することができますが、これらは常にブランクとして印刷されます。

外部記述ファイルの編集

リモート・ディスク・ファイル用に出力を編集するには、データ記述仕様 (DDS) に編集コードを指定しなければなりません。

注: 特殊ファイルの場合には、編集コードを使用することはできません。

第 14 章 データの初期化

この項では、データが初期化される方法について説明します。データの初期化は、以下の 3 パートで構成されています: 初期化サブルーチン (*INZSR)、CLEAR と RESET 命令コード、およびデータ初期化 (INZ キーワード) です。コンポーネントの初期化については、31 ページの『コンポーネントの初期化』 を参照してください。

初期化サブルーチン (*INZSR)

初期化サブルーチンで、演算仕様を処理することができます。これは、他のサブルーチンと同様に宣言されますが、演算項目 1 の *INZSR が付いています。RESET 演算命令以外の任意の演算命令をこのサブルーチンに入力することができます。また *INZSR を EXSR または CASxx 命令コードを使用して、明示的に呼び出すことができます。

CLEAR および RESET 命令コード

CLEAR 命令コードは、ウィンドウまたは構造の変数をデフォルト値に設定します。構造 (レコード様式、データ構造、または配列) を指定した場合には、その構造のすべてのフィールドが、宣言された順序で消去されます。

RESET 命令コードは、ウィンドウまたは構造の変数を初期値に設定します。データ構造初期化を使用して、初期値をサブフィールドに割り当て、プログラムの実行中にその値を変更することができ、また RESET 命令コードを使用して、フィールド値を初期値に設定し直すことができます。

データ初期化

データは、定義仕様上の INZ キーワードで初期化されます。初期値を INZ キーワード上のパラメーターとして指定する、あるいはパラメーターをもたないキーワードを指定してデフォルトの初期値を使用することができます。さまざまなデータ・タイプのデフォルト初期値が、『第 9 章 データ・タイプおよびデータ形式』 に説明されています。配列の初期化については、 『第 12 章 配列およびテーブルの使用』 を参照してください。

第 3 部 仕様

この項ではパーツの属性について説明します。

- キーワード構文および継続の規則など、いくつかの仕様に共通な情報が説明されています。
- それぞれの仕様は、それをプログラムに入力する必要がある順序で説明されています。それぞれの仕様記述には、その仕様のすべてのフィールドがリストされ、可能なすべての入力が説明されています。

第 15 章 VisualAge RPG 仕様について

VisualAge RPG 言語は、桁依存およびフリー・フォームのコードの混合から成っています。VisualAge RPG プログラムは、いろいろな仕様でコーディングされます。各仕様には特定の関数のセットがあります。

VisualAge RPG プログラムでコーディングされることのあるソース・レコードのグループには、メイン・ソース・セクション、サブプロシージャー・セクション、およびプログラム・データ・セクションの 3 つがあります。 **メイン・ソース・セクション**の構造は、結果としてのコンパイル・ターゲットがコンポーネント、NOMAIN DLL、または EXE かによって異なります。**メイン・ソース・セクション**には、モジュールのすべてのグローバル定義が含まれています。コンポーネント・ターゲットの場合には、このセクションには、アクションおよびユーザー・サブルーチンも含まれます。各コンパイル・ターゲットのメイン・ソース・セクションのレイアウトは256 ページの『定義の配置と有効範囲』に示されています。

サブプロシージャー・セクションには、モジュール内でコーディングされるすべてのサブプロシージャーを定義する仕様が含まれています。**プログラム・データ・セクション**には、コンパイル時に提供されるデータとともにソース・レコードが含まれています。

次の図は、各グループに入力されるソース・レコードのタイプとその順序を示したものです。

注: VisualAge RPG ソース・プログラムは、示された順序でシステムに入力しなければなりません。仕様タイプのどれかが無くても構いませんが、少なくとも 1 つは必要です。

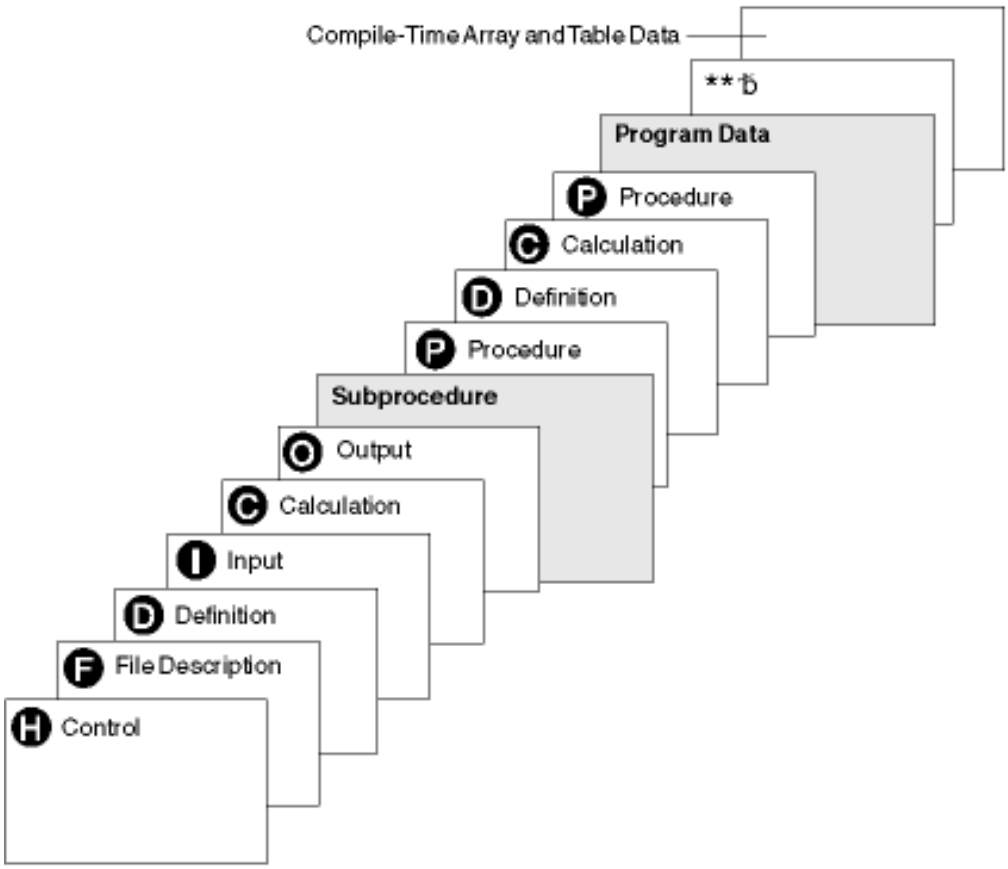


図 70. VisualAge RPG ソース・プログラムにおける仕様のタイプの順序

H 制御 (ヘッダー) 仕様は、プログラム生成およびコンパイル済みプログラムの実行に関する情報を提供します。 この仕様の記入項目の説明については、『第 16 章 制御仕様』 を参照してください。

F ファイル仕様はプログラム中のすべてのファイルを定義します。 この仕様の説明については、 『第 17 章 ファイル仕様』 を参照してください。

D 定義仕様は、ユーザーのプログラムの中で使用される項目を定義します。この仕様には、配列、テーブル、データ構造、サブフィールド、固定情報、独立型フィールド、イベント属性、プロトタイプとそのパラメーター、およびプロシージャー・インターフェースとそのパラメーターが定義されます。この仕様の説明については、 『第 18 章 定義仕様』 を参照してください。

I 入力仕様はレコードおよび入力ファイル中のフィールドについて記述し、レコードおよびフィールドがプログラムで使用される方法を指示します。この仕様の説明については、 『第 19 章 入力仕様』を参照してください。

C 演算仕様は、プログラムによって実行される演算について記述し、その実行順序を指示します。 演算仕様は一定の入出力操作を制御することができます。 コンポーネント・ターゲットについては、このセクションには、アクション・サブルーチンおよび独立型ユーザー・サブルーチンが含まれています。 NOMAIN DLL および EXE には、演算仕様セクションはありません。 この仕様の項目の説明については、『第 20 章 演算仕様』 を参照してください。『第 23 章 演算命令』 は、演算仕様でコード化された命令コードについて記述しています。

O 出力仕様はレコードおよびフィールドを記述し、それらがプログラムによって作成される時期を指示します。この仕様の説明については、 『第 21 章 出力仕様』 を参照してください。

サブプロシージャー仕様

P プロシージャー仕様はプロトタイプ化されたプログラムまたはプロシージャーのプロシージャー・インターフェース定義について記述しています。 この仕様の項目の説明については、『第 22 章 プロシージャー仕様』 を参照してください。

D 定義仕様は、プロトタイプ化されたプロシージャーで使用される項目を定義します。 プロシージャー・インターフェース定義、入力パラメーター、およびその他のローカル項目はこの仕様で定義されます。 この仕様の項目の説明については、『第 18 章 定義仕様』 を参照してください。

C 演算仕様はプロトタイプ化されたプロシージャーの論理を実行します。 この仕様の項目の説明については、『第 20 章 演算仕様』 を参照してください。

プログラム・データ

すべてのソース仕様の後には、プログラム・データをもつソース・レコードが続きます。データ・セクションの最初の行は ** で始まらなければなりません。必要な場合には、CTDATA キーワードを指定することによって、** の後に続くプログラム・データのタイプを指示することができます。プログラム・データをこのキーワードと関連付けることによって、プログラム・データのグループを任意の順序でソース・レコードの後に入れることができます。

各入力レコードの最初の記入項目は 1 桁目から始まらなければなりません。レコード全体を記入項目で満たす必要はありません。未使用の記入項目のある配列エレメントはデフォルト値で初期化されます。

コンパイル時配列レコードの入力の詳細については、 179 ページの『配列ソース・レコードの規則』を参照してください。

キーワードをサポートする仕様 (制御、ファイル、記述、定義、およびプロシージャー) によって、キーワード・フィールド内でフリー・フォームを使用することができます。演算仕様では、拡張演算項目 2 をサポートする命令コードをもつフリー・フォームを使用できます。そうでない場合には、記入項目は桁指定となります。これを表すために、VisualAge RPG コードの各イラストは、上部に目盛りが描かれたリスト形式になっています。

この解説には、個々の仕様の詳細説明が含まれています。各フィールドとその記入項目について説明されています。 345 ページの『第 23 章 演算命令』 は演算仕様にコード化されている命令コードについて記述しており、これは 311 ページの『第 20 章 演算仕様』 に記述されています。

共通記入項目

次の記入項目はすべての VisualAge RPG 仕様に対して共通です:

- 1 ～ 5 桁目はコメント用に使用できます。
- 6 桁目は仕様タイプを示します。次の文字コードを使用することができます:

項目	仕様タイプ
H	制御
F	ファイル記述
D	定義
I	入力
C	演算
O	出力
P	プロシージャー

- 注釈ステートメント
  - 7 桁目には アスタリスク (*) が入ります。これは、仕様上の他の記入項目に関係なく、行をコメント行として指示します。フリー・フォーム演算仕様では、コメントに // を使用できます。任意の固定形式仕様上にある // で始まる任意の行は、コンパイラーにはコメントであると見なされます。// は指定された任意の桁から開始でき、6 桁目から // 文字の間にはブランクが入ります。
  - 6 ～ 80 桁目はブランクです。
- 7 - 80 桁目はブランクで、6 桁目には有効な指定が入ります。 これはコメント行ではなく、有効な行であり、順序規則が適用されます。

キーワードの構文

キーワードには、パラメーターなし、任意パラメーター、あるいは必須パラメーターがあるものがあります。キーワードの構文は次の通りです。

```
Keyword(parameter1 : parameter2)
```

ここで:

- パラメーター (1 つまたは複数) は括弧 ( ) で囲みます。

  注: パラメーターがない場合は括弧を指定してはいけません。
- コロン (:) は複数のパラメーターを区切るのに使用します。

次の国際規則は、任意指定のパラメーターと必須のパラメーターを示すために使用されます。

- 中括弧 { } は、オプション・パラメーターまたはパラメーターのオプション・エレメントを示します。
- 省略記号 (...) は、パラメーターを繰り返し使用できることを示します。
- コロン (:) はパラメーター間を区切り、複数のパラメーターを指定できることを示します。 コロンで区切られたすべてのパラメーターは、それらが中括弧で囲まれていない限り必須です。
- 縦線 (|) は、キーワードに 1 つのパラメーターしか指定できないことを示します。

- キーワード・パラメーターを区切るブランクは、1 つまたは複数のパラメーターを指定できることを示します。

注: 中括弧、省略記号、および縦線はキーワード構文の一部ではなく、ソースの中に入力してはいけません。

表 24. キーワード表記の例

表記	使用される表記の例	説明	入力されたソースの例
中括弧 {}	PRTCTL (data_struct {:*COMPAT})	パラメーター data_struct は必須で、パラメーター *COMPAT は任意指定です。	PRTCTL (data_struct1)
中括弧 {}	TIME(format {separator})	パラメーター format{separator} は必須ですが、パラメーターの {separator} 部分は任意指定です。	TIME(*HMS&)
コロン (:)	RENAME(Ext_format :Int_format)	パラメーター Ext_format および Int_format は必須です。	RENAME (nameE: nameI)
省略記号 (...)	IGNORE(recformat {:recformat...})	パラメーター recformat は必須で、複数回指定できます。	IGNORE (recformat1: recformat2: recformat3)
縦線 (|)	FLTDIV{(*NO | *YES)}	*NO または *YES を指定するか、あるいはパラメーターを指定しないでください。	FLTDIV
ブランク	OPTIONS(*OMIT *VARSIZE *STRING *RIGHTADJ)	*OMIT、*VARSIZE、*STRING、または *RIGHTADJ の 1 つが必須で、複数のパラメーターを任意で指定できます。	OPTIONS (*OMIT: *VARSIZE: *STRING: *RIGHTADJ)

継続規則

続行できるフィールドは次の通りです:

- 制御仕様のキーワード・フィールド
- ファイル仕様のキーワード・フィールド
- 定義仕様のキーワード・フィールド
- 演算仕様の拡張演算項目 2 フィールド
- 出力仕様の固定情報/編集語フィールド
- 定義仕様またはプロシージャー仕様の名前フィールド

継続の一般的な規則は次の通りです:

- 継続行は継続される仕様にとって有効なものでなければなりません (6 桁目に H、F、D、C、または O)。
- リテラルまたは名前を分割する必要がある場合を除いて、複数行にまたがって仕様を継続する時は、特殊文字を使用してはいけません。 たとえば、次のペアは同等のものです。最初のペアでは、正符号 (+) が行の終わりにありますが、これは演算子です。 2 番目のペアでは、正符号 (+) は継続文字です。

```
C                   eval      x = a + b
C                   eval      x = a +
C                                 b

C                   eval      x = 'abc'
C                   eval      x = 'ab+
C                                  c'
```

- ブランク行、空の仕様行、またはコメント行だけは、継続される行の間で使用できます。
- 継続は完全なトークンの後で起こることがあります。 トークンは次の通りです:
  - 名前 (たとえば、キーワード、ファイル名、フィールド名)
  - 括弧
  - 区切り文字 (:)
  - 式の演算子
  - 組み込み関数
  - 特殊語
  - リテラル
- 継続はリテラル内でも起こります:
  - 文字、日付、時刻、およびタイム・スタンプ・リテラルの場合:
    - ハイフン (-) は、継続されるフィールドの最初の有効桁に継続があることを示します
    - プラス (+) は、継続されるフィールドの最初の桁またはその後の最初の非ブランク文字で継続することを示します
  - グラフィック・リテラルの場合:
    - 継続を示すために、ハイフン (-) または プラス (+) のいずれかを使用できます。
  - 数値リテラルの場合:
    - 継続文字は使用されません
    - 数値リテラルは、継続されるフィールドの継続行上の数字または小数点で継続します
  - 16 進数および UCS-2 リテラルの場合:
    - 継続を示すために、ハイフン (-) またはプラス (+) のいずれかを使用できます

- リテラルは次の行の最初の非ブランク文字で継続されます

- 継続はフリー・フォーム記入項目の名前の中で起こることもあります
  - 定義仕様およびプロシージャー仕様の名前記入項目の中。 名前記入項目で名前を継続する場合の詳細については、 221 ページの『定義およびプロシージャー仕様の名前フィールド』 を参照してください。
  - ファイル仕様および定義仕様のキーワード記入項目の中。
  - 演算仕様の拡張演算項目 2 記入項目の中。

  以下に示すように、修飾名はピリオドで分割できます。

```
C                   EVAL      dataStructureWithALongName.
C                               subfieldWithAnotherLongName = 5
```

  名前をピリオドで分割しない場合は、部分名の終わりに間にブランクを入れずに省略符号 (...) をコーディングします。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++++
D                                     Keywords-cont++++++++++++++++++++++++++++
* ロング・ネームの 10 文字のフィールドを定義します。
* 2 番目の定義は、ロング・ネームの変数のアドレスに合わせて初期化
* されるポインターです。
D QuiteLongFieldNameThatCannotAlwaysFitInOneLine...
D                 S             10A
D Ptr             S               *   inz(%addr(QuiteLongFieldName...
D                                             ThatCannotAlways...
D                                             FitInOneLine))
D ShorterName     S              5A

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++++
C                           Extended-factor2-++++++++++++++++++++++++++++++++
* 式の中でロング・ネームを使用します
* 必要な位置で名前を分割できることに注意してください。
C                   EVAL          QuiteLongFieldName...
C                                 ThatCannotAlwaysFitInOneLine = 'abc'

* この方法で名前を分割できます
C                   EVAL          P...
C                                 tr = %addr(Shorter...
C                                 Name)
```

図 71. 例

制御仕様キーワード・フィールド

制御仕様での継続規則は、仕様は次の制御仕様の 7 桁目以降で継続するということです。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DATFMT(
H       *MDY&
H       )
```

図 72. 制御仕様での継続

ファイル仕様キーワード・フィールド

ファイル仕様の継続規則は次の通りです:

- 仕様は次のファイル仕様の 44 桁目以降に継続します。
- 継続行の 7-43 桁目はブランクでなければなりません

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
FFilename++IT.A.FRlen+......A.Device+.Keywords+++++++++++++++++++++++++++++
F                                     RECNO
F                                     (
F                                     *INU1
F                                     )
```

図 73. ファイル仕様キーワード・フィールド

定義仕様キーワード・フィールド

定義仕様の継続規則は次の通りです:

- 指定された継続文字に応じて、仕様は次の定義仕様の 44 桁目以降に継続します
- 継続行の 7-43 桁目はブランクでなければなりません。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
D                                     Keywords-cont+++++++++++++++++++++++++++
 *
DMARY             C                   CONST(
D                                     'Mary had a little lamb, its -
D* ここには、コメントまたは完全なブランク行だけが使用できます
D                                     fleece was white as snow.'
D                                     )
D* 数値リテラルは、44 桁目以降の最初の非ブランクで継続します
D*
DNUMERIC          C                   12345
D                                       67
D* グラフィックスの名前付き固定情報
DGRAF             C                   G'AABBCCDD+
D                                      EEFFGG'
```

図 74. 定義仕様キーワード・フィールドの例

演算仕様の拡張演算項目 2

演算仕様の継続規則は次の通りです:

- 仕様は次の演算仕様の 36 桁目以降に継続します
- 継続行の 7-35 桁目はブランクでなければなりません。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++++
C                            Extended-factor2-++++++++++++++++++++++++++++++++
 *
C                   EVAL      MARY='Mary had a little lamb, its +
C* ここには、コメントまたは完全なブランク行だけが使用できます
C                               fleece was white as snow.'
C*
C*  継続文字ではなく、演算式の継続
C
C                   EVAL      A = (B*D)/ C +
C                             24
C*  この例の最初の '+' の使用は、連結演算子です。2 番目の使用は
C*  文字リテラルの継続です。
C                   EVAL      ERRMSG = NAME +
C                                      ' was not found +
C                                      in the file.'
```

図 75. 演算仕様の拡張演算項目 2 の例

フリー・フォーム演算仕様

フリー・フォーム演算仕様の継続規則は次のとおりです。

- フリー・フォーム行は、次の行に続けることが可能です。セミコロンが示されるまで、ステートメントが続行されます。

例

```
/FREE
          time = hours * num_employees
                   + overtime_saved;
/END-FREE
```

出力仕様の固定情報/編集語フィールド

出力仕様の継続規則は次の通りです:

- 仕様は次の出力仕様の 53 桁目以降に継続します
- 継続行の 7-52 桁目はブランクでなければなりません。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
 *
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat+++
O                                          Continue Constant/editword+++

O                                        80 'Mary had a little lamb, its-
O* ここには、コメントまたは完全なブランク行だけが使用できます
O                                         fleece was white as snow.'
```

図 76. 出力仕様の固定情報/編集語フィールドの例

定義およびプロシージャー仕様の名前フィールド

定義およびプロシージャー仕様における名前の継続規則は次の通りです:

- 15 文字より長い名前に対して継続規則が適用されます。 部分名の終わりに省略記号 (...) をコーディングすることによって、任意の名前 (15 文字またはそれ以下のものでも) を複数の行に継続することができます。
- 名前定義は以下のパーツから成り立っています:
  1. ゼロまたはそれ以上の継続名前行。継続名前行は、項目中の最後の非ブランク文字として省略記号をもつことによって識別されます。名前は 7 - 21 桁目で始まり、最大 77 桁目までの任意の位置で終わることができます (80 桁目で終わる省略記号を指定して)。 名前の始めと省略記号 (...) 文字の間にブランクがあってはいけません。 これらの条件のいずれかが適合しない場合には、その行はメイン定義行と見なされます。
  2. 名前、定義属性、およびキーワードが入っている 1 つのメイン定義行。継続名前行をコーディングする場合には、メイン定義行の名前記入項目をブランクにすることができます。
  3. ゼロまたはそれ以上のキーワード継続行。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D                                  Keywords-cont++++++++++++++++++++++++++
D* 継続名前行のないロング・ネーム:
D RatherLongName  S             10A
D* 1 つの継続名前行を使用したロング・ネーム:
D NameThatIsEvenLonger...
D                 C                   'This is the constant -
D                                     that the name represents.'
D* 1 つの継続名前行を使用したロング・ネーム:
D NameThatIsSoLongItMustBe...
D   Continued     S             10A
D* コンパイル時配列はロング・ネームをもつ場合があります :
D CompileTimeArrayContainingDataRepresentingTheNamesOfTheMonthsOf...
D  TheYearInGermanLanguage...
D                 S             20A   DIM(12) CTDATA PERRCD(1)
D* 3 つの継続名前行を使用するロング・ネーム:
D ThisNameIsSoMuchLongerThanThe...
D    PreviousNamesThatItMustBe...
D    ContinuedOnSeveralSpecs...
D                 PR            10A
D parm_1                        10A   VALUE
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C* 演算仕様で定義されたロング・ネーム:
C     LongTagName   TAG
C     *LIKE         DEFINE    RatherLongNameQuiteLongName    +5
PName++++++++++..B..................Keywords++++++++++++++++++++++++++++++
PContinuedName+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
P* プロシージャー仕様で指定されたロング・ネーム:
P ThisNameIsSoMuchLongerThanThe...
P    PreviousNamesThatItMustBe...
P    ContinuedOnSeveralSpecs...
P                 B
D ThisNameIsSoMuchLongerThanThe...
D    PreviousNamesThatItMustBe...
D    ContinuedOnSeveralSpecs...
D                 PI            10A
D parm_1                        10A   VALUE
D* プロシージャーの本体
D*
P ThisNameIsSoMuchLongerThanThe...
P    PreviousNamesThatItMustBe...
P    ContinuedOnSeveralSpecs...
P                 E
```

図 77. RPG でのロング・ネームの定義

第 16 章 制御仕様

6 桁目の H で識別される 制御仕様ステートメント は、プログラムの生成および実行に関する情報を提供します。この情報は、ユーザーのソースに含まれる制御仕様によってコンパイラーに提供されます。制御仕様を含めないと、制御仕様キーワードにそのデフォルト値が割り当てられます。

デフォルト値については、個々の項目の説明を参照してください。

制御仕様キーワードは、モジュラー・レベルで適用されます。これは、1 つのモジュール内に複数のプロシージャーをコーディングした場合に、制御仕様に指定された値がすべてのプロシージャーに適用されることを意味します。

制御仕様ステートメント

制御仕様は、キーワードだけから構成されます。キーワードは、7 - 80 桁目の任意の桁に入れることができます。81 - 100 桁目はコメントに使用できます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
 HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++Comments++++++++++++
```

図 78. 制御仕様のレイアウト

制御仕様の例は次のとおりです。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H CURSYM('\') DATEDIT(*MDY) DATFMT(*MDY/)
H DECEDIT('.') TIMFMT(*ISO)
```

6 桁目 (仕様のタイプ)

この行が制御仕様であることを示すために、6 桁目に H がなければなりません。

7-80 桁目 (キーワード)

制御仕様キーワードは、プログラムによる装置の取扱方法および特定情報タイプの表現方法を判別するために使用します。

キーワードの構文

制御仕様には、パラメーター、任意指定パラメーター、または必須パラメーターがないことがあります。キーワードの構文は次の通りです:

```
Keyword(parameter1 : parameter2)
```

ここで:

- パラメーター (1 つまたは複数) は括弧 ( ) で囲みます。

  注: パラメーターがない場合には括弧を指定しないでください。
- コロン (:) は複数のパラメーターを区切るのに使用します。

次の国際規則は、任意指定のパラメーターと必須のパラメーターを示すために使用されます。

- 中括弧 { } は、オプション・パラメーターまたはパラメーターのオプション・エレメントを示します。
- 省略記号 (...) は、パラメーターを繰り返し使用できることを示します。
- コロン (:) はパラメーター間を区切り、複数のパラメーターを指定できることを示します。 コロンで区切られたすべてのパラメーターは、それらが中括弧で囲まれていない限り必須です。
- 縦線 (|) は、キーワードに 1 つのパラメーターしか指定できないことを示します。
- キーワード・パラメーターを区切るブランクは、1 つまたは複数のパラメーターを指定できることを示します。

注: 中括弧、省略記号、および縦線はキーワード構文の一部ではなく、ソースの中に入力してはいけません。

制御仕様キーワードに追加のスペースが必要な場合には、キーワード・フィールドを次の行に継続できます。 223 ページの『制御仕様ステートメント』 および 218 ページの『制御仕様キーワード・フィールド』 を参照してください。

ALWNULL(*NO | *INPUTONLY | *USRCTL)

ALWNULL キーワードは、外部記述データベース・ファイルからのヌル値可能フィールドを含むレコードをどのように使用するかを指定します。

ALWNULL(*NO) は、外部記述ファイルからのヌル値フィールドのあるレコードを処理できないことを指定します。ヌル値を含むレコードを検索しようとした場合には、レコード中のデータはアクセスできず、データ・マッピング・エラーが起こります。

ALWNULL(*INPUTONLY) を指定すると、外部記述入力専用データベース・ファイルから、ヌル値を含むヌル値可能フィールドのあるレコードを正常に読み取ることができます。ヌル値を含むレコードを検索すると、データ・マッピング・エラーは起こらず、ヌル値のあるフィールドには、データベースのデフォルト値が入れられます。ただし、次のいずれも実行できません。

- ヌル値可能キー・フィールドの使用
- ヌル値可能フィールドを含むレコードの作成または更新
- プログラムの実行時にヌル値可能フィールドが実際にヌルであるかどうかの判別
- ヌル値可能フィールドのヌル値への設定

ALWNULL(*USRCTL) を指定すると、外部記述データベース・ファイルから、ヌル値のあるレコードを読み取り、書き込み、および更新できます。 NULL キーのレコードはキー付き演算命令を使用して検索することができます。ヌル値可能フィールドが実際にヌル値であるかどうかを判別でき、出力または更新用にヌル値可能フィールドをヌル値に設定できます。ヌル値を含むフィールドが正しく使用されるようにするのは、ユーザーの責任です。

ALWNULL キーワードを指定しない場合には、コマンド上に指定された値が使用されます。

詳細については、 145 ページの『データベースのヌル値サポート』 を参照してください。

CACHE(*YES | *NO)

CACHE キーワードは、ワークステーションのキャッシュ・フォルダーに保管されたリモート・ファイル記述がアプリケーションによって使用されることを指定します。最初に CACHE(*YES) オプションを使用した時に、リモート・ファイルの記述が作成されます。それ以降、コンパイル処理ではサーバー・ファイルをアクセスする代わりに、この情報をアクセスします。

CACHEREFRESH(*YES | *NO)

CACHEREFRESH キーワードは、キャッシュ・フォルダー中のリモート・ファイル記述がコンパイル処理の前に更新されることを指定します。CACHE(*NO) を指定すると、既存のリモート・ファイル記述が使用されます。

CCSID(*GRAPH : パラメーター | *UCS2 : 数値 | *MAPCP : 932)

このキーワードは、そのモジュールのデフォルトのグラフィック (*GRAPH) および UCS-2 (*UCS2) CCSID を設定します。これらのデフォルトは、コンパイル時データ、プログラム記述の入力と出力フィールド、およびCCSID がコーディングされていないデータ定義に使用されます。

CCSID(*GRAPH : *IGNORE | *SRC | 数値)

モジュールのデフォルト・グラフィック CCSID を設定します。指定できる値は次の通りです。

***IGNORE**

これがデフォルトです。モジュール内でグラフィックと UCS-2 フィールド間の変換は使用できません。%GRAPH 組み込み関数は使用できません。

***SRC**

ソース・ファイルの CCSID と関連したグラフィック CCSID が使用されます。

数値

グラフィック CCSID。

CCSID(*UCS2 : 数値)

モジュールのデフォルト UCS-2 CCSID を設定します。このキーワードを指定しないと、デフォルトの UCS-2 CCSID は 13488 です。

数値 は UCS-2 CCSID でなければなりません。有効な CCSID は 13844 および 17584 (これにはユーロが入っています)。

CCSID(*MAPCP : 932)

リモート・ファイル・オープンおよびプログラム呼び出しの場合は、日本語コード・ページ 932 ～ CCSID 943 をマップします。

CCSID(*GRAPH : *SRC) または CCSID(*GRAPH : 数値) を指定した場合には、

- 外部記述データ構造のグラフィックおよび UCS-2 フィールドは、外部ファイルの CCSID を使用します。
- プログラム記述のグラフィックまたは UCS-2 フィールドは、デフォルトとしてそれぞれそのモジュールのグラフィックまたは UCS-2 CCSID を使用します。この仕様は、そのフィールドの定義に CCSID(数値) キーワードを使用することによって上書きできます。(268 ページの『CCSID(number | *DFT)』 を参照してください。)
- プログラム記述のグラフィックまたは UCS-2 の入力または出力フィールドおよびキーは、そのモジュールのデフォルト CCSID をもつものと見なされます。

COPYNEST(数値)

COPYNEST キーワードは、/COPY ディレクティブで可能なネストの最大深さを指定します。深さは、1 以上で 2048 以下としなければなりません。デフォルトの深さは 32 です。

COPYRIGHT('著作権ストリング')

COPYRIGHT キーワードは、著作権情報を提供します。著作権ストリングは、最大長 256 の文字リテラルです。このリテラルは、継続仕様に継続できます。(継続行の使用規則については、216 ページの『継続規則』 を参照してください。)
COPYRIGHT キーワードを指定しないと、作成されるモジュールまたはプログラムに著作権情報が追加されません。

CURSYM('sym')

CURSYM キーワードは、編集で通貨記号として使用する文字を指定します。この記号は引用符で囲んだ単一文字でなければなりません。VisualAge RPG 文字セット中の任意の文字を使用できます。 (3 ページの『第 1 章 記号名と予約語』 を参照してください。) 次の文字は例外です。

0 (ゼロ)	* (アスタリスク)	, (コンマ)
& (アンパーサンド)	. (ピリオド)	– (負符号)
C (英字 C)	R (英字 R)	ブランク

このキーワードを指定しないと、円記号 (¥) が通貨記号のデフォルトになります。

CVTOEM(*YES | *NO)

CVTOEM キーワードは、ローカル・ファイルに対して入出力を実行する時に OEM 変換の使用が必要であることを指定します。CVTOEM(*NO) を指定すると、OEM 変換は行なわれません。

CVTOPT(*{NO}VARCHAR *{NO}VARGRAPHIC)

CVTOPT キーワードは、外部記述データベース・ファイルから検索した可変長データ・タイプを VARPG コンパイラーがどのように処理するかを決めるために使用します。

任意の順序で任意のデータ・タイプまたはすべてのデータ・タイプを指定できます。ただし、あるデータ・タイプを指定すると、同じデータ・タイプに対して *NOxxxx パラメーターは指定できず、その逆も同じです。たとえば、*VARCHAR を指定すると、 *NOVARCHAR は指定できず、その逆も同じです。パラメーター間はコロンで区切ります。パラメーターを 2 回指定することはできません。

注: キーワード CVTOPT に対からのメンバーが入っていない場合には、この特定のデータ・タイプに対してコマンド上に指定された値が使用されることになります。たとえば、制御仕様にキーワード CVTOPT(*NOVARCHAR) が指定されている場合には、対 (*VARGRAPHIC、*NOVARGRAPHIC) について、コマンド上に暗黙または明示的に指定された値が使用されることになります。

*VARCHAR を指定すると、可変長文字データ・タイプが固定長文字フィールドとして宣言されます。

*NOVARCHAR を指定すると、可変長文字データ・タイプは変換されません。

*VARGRAPHIC を指定すると、可変長 2 バイト文字セット (DBCS) グラフィック・データ・タイプが固定長文字フィールドとして宣言されます。

*NOVARGRAPHIC を指定すると、可変長 2 バイト文字セット (DBCS) グラフィック・データ・タイプは変換されません。

CVTOPT キーワードを指定しないと、コマンド上に指定された値が使用されます。

DATEDIT(fmt{区切り文字})

DATEDIT キーワードは、Y 編集コード使用時の数値フィールドの形式を指定します。区切り文字は任意指定です。値 (fmt) は *DMY、*MDY、または *YMD とすることができます。デフォルトの区切り文字は / です。区切り文字 & (アンパーサンド) を使用して、ブランクの区切り文字を指定できます。

DATFMT(fmt{区切り文字})

DATFMT キーワードは、日付リテラルの内部日付形式およびプログラム内の日付フィールドのデフォルト形式を指定します。そのフィールドの定義仕様で DATFMT キーワードに形式を指定することによって、特定のフィールドに別の内部日付形式を指定できます。

デフォルトは *ISO 形式です。内部形式の詳細については、 107 ページの『内部形式と外部形式』 を参照してください。

表 25 には、各種の日付形式およびその区切り文字の説明があります。

表 25. 日付データ・タイプの外部日付形式

RPG 名	説明	形式 (デフォルトの区切り文字)	有効な区切り文字	長さ	例
*MDY	月/日/年	mm/dd/yy	/ - . , '&'	8	01/15/91
*DMY	日/月/年	dd/mm/yy	/ - . , '&'	8	15/01/91
*YMD	年/月/日	yy/mm/dd	/ - . , '&'	8	91/01/15
*JUL	年間通算日	yy/ddd	/ - . , '&'	6	91/015
*ISO	国際標準化機構	yyyy-mm-dd	-	10	1991-01-15
*USA	IBM USA 標準規格	mm/dd/yyyy	/	10	01/15/1991
*EUR	IBM 欧州標準規格	dd.mm.yyyy	.	10	15.01.1991
*JIS	日本工業規格西暦	yyyy-mm-dd	-	10	1991-01-15

DEBUG{(*NO | *YES)}

DEBUG キーワードは、デバッグ情報を生成するかどうかを判別します。

このキーワードを指定しないか、あるいは *NO で指定すると、デバッグ情報は生成されません。

DECEDIT('値')

このキーワードは、編集済み 10 進数で小数点として使用する文字を指定します。その数の絶対値が 1 より小さい時には、先行ゼロが印刷されます。デフォルト値は '.' (ピリオド) です。

使用できる値は次の通りです:

'.' 小数点はピリオドで、先行ゼロは印刷されません (.123)

',' 小数点はコンマで、先行ゼロは印刷されません (,123)

'0.' 小数点はピリオドで、先行ゼロが印刷されます (0.123)

'0,' 小数点はコンマで、先行ゼロが印刷されます (0,123)

EXE

EXE キーワードは、これがメイン・プロシージャーおよびサブプロシージャーを構成するモジュールであることを指示します。すべてのサブルーチン (BEGSR) はプロシージャーに対してローカルでなければなりません。EXE には、名前がソース・ファイルの名前と一致するプロシージャーが入っていなければなりません。 これは EXE の主入り口点、すなわち、メイン・プロシージャーとなります。

EXE モジュールについては以下のことを考慮してください。

- ソースでは GUI 命令コードを使用することはできません。これには、START、STOP、SETATR、GETATR、%SETATR、%GETATR、SHOWWIN、CLSWIN、および READS が含まれます。DSPLY は使用できます。
- *INZSR および *TERMSR は許可されません。
- *ENTRY パラメーターは許可されません。

入力パラメーターがある場合には、これらはメイン・プロシージャーのパラメーター定義上に指定し、値によって渡す (すなわち、各パラメーターに VALUE キーワードを指定する) 必要があります。

- 開始 P 仕様では EXPORT キーワードは使用できません。
- メイン・プロシージャーの戻り値は、精度ゼロ (0) の 2 進数または整数として定義しなければなりません。

EXPROPTS(*MAXDIGITS | *RESDECPOS)

EXPROPTS (式オプション) キーワードは、プログラム全体に使用する精度規則のタイプを指定します。これを指定しないか、あるいは *MAXDIGITS を指定すると、デフォルトの精度規則が適用されます。*RESDECPOS つきで EXPROPTS を指定すると、″結果の小数点以下の桁数″ 精度規則が適用され、式の中間結果が結果よりも少ない小数点以下の桁数にならないよう強制されます。

注: 命令コード拡張 Rおよび M は、それぞれ EXPROPTS(*RESDECPOS) および EXPROPTS(*MAXDIGITS) と同じですが、これらは単一のフリー・フォームの式用です。

EXTBININT{(*NO | *YES)}

EXTBININT キーワードは、2 進数外部形式で小数点以下の桁数がゼロの外部記述フィールドを、外部整数形式をもっているかのように処理するために使用します。これを指定しないか、あるいは *NOを指定した場合には、外部記述 2 進数フィールドは外部 2 進数形式で処理されます。EXTBININT を指定すると (*YES の指定は任意)、外部記述フィールドは、次のように処理されます。

DDS 定義	RPG 外部様式
B(n,0) ただし 1 ≤ n ≤ 4	I(5)
B(n,0) ただし 5 ≤ n ≤ 9	I(10)

EXTBININT キーワードを指定することによって、ユーザーのプログラムは、DDS 2 進数値の完全な範囲を使用できるようになります。(DDS 2 進数値の範囲は、符号付き整数と同じです。5 桁のフィールドでは -32768 - 32767、10 桁のフィールドでは -2147483648 - 2147483647 です。)

注: キーワード EXTBININT を指定すると、小数点以下の桁数がゼロの 2 進数である外部記述サブフィールドは、 *内部* 整数形式で定義されます。

FLTDIV{(*NO | *YES)}

FLTDIV キーワードは、式内部のすべての除算が浮動小数点で計算され、浮動タイプの値が戻されることを指示します。これを指定しないか、あるいは *NO を指定した場合には、除算は (2 つのオペランドの 1 つがすでに浮動になっていないかぎり) パック 10 進数形式で実行されます。

FLTDIV を指定すると (*YES の指定は任意)、すべての除算は浮動形式で実行されます (結果が常に 15 桁の精度をもつことが保証されます)。

GENLVL(数値)

GENLVL キーワードは、オブジェクトの作成を制御します。オブジェクトは、コンパイル時に起こったすべてのエラーの重大度レベルが、指定された生成重大度レベル以下の場合に作成されます。値は 0 - 20 の範囲でなければなりません。重大度が 20 より大きいエラーの場合には、オブジェクトは作成されません。

GENLVL キーワードを指定しないと、コマンド上に指定された値が使用されます。

INDENT(*NONE | '文字値')

INDENT キーワードは、拡張読み取り機能のソース・リストで構造化演算を字下げするかどうかを指定します。また、構造化演算命令文節のマーク付けに使用する文字も指定します。

*NONE を指定すると、ソース・リスト中構造化演算命令は字下げされません。

文字値を指定すると、ソース・リストの構造化演算命令が字下げされます。ステートメントと文節の位置合わせに、選択した文字でマークが付けられます。長さが 2 文字以内の任意の文字リテラルを選択できます。

注: ソースにエラーがあると、字下げが予定どおりにならない場合があります。

INDENT キーワードを指定しないと、コマンド上に指定された値が使用されます。

INTPREC(10 | 20)

INTPREC キーワードは、式の 2 進算術演算中の整数および符号なし中間値の 10 進数精度を指定するために使用します。整数および符号なし中間値は常に、8 バイト形式に維持されています。このキーワードが影響するのは、2 進算術演算 (+、-、*、/)で使用した時に、整数および符号なし中間値が 10 進数形式に変換される方法だけです。

デフォルトの INTPREC(10) は、整数および符号なし演算に 10 桁の 10 進数精度を指示します。ただし、式の中の少なくとも 1 つのオペランドが 8 バイト整数または符号なしフィールドである場合には、式の結果は、INTPREC 値に関係なく 20 桁の 10 進数精度をもちます。

INTPREC(20) は、整数および符号なし演算の 10 進数精度が 20 桁であることを指示します。

LIBLIST('filename1 filename2 ... filenamen')

LIBLIST キーワードは、アプリケーションのリンク時に使用するライブラリー・ファイルのリストを指定します。各ファイル名をブランクで区切り、リストを単一引用符で囲む必要があります。ファイル名にブランクが入っている場合には、その名前を二重引用符で囲まなければなりません。

NOMAIN

NOMAIN キーワードは、モジュール中にアクションまたはスタンドアロン・ユーザー・サブルーチンがないことを指示します。 **NOMAIN モジュール** には、サブプロシージャーだけが入っています。結果のコンパイル・ターゲットは、他のアプリケーションで使用できる DLL です。

NOMAIN DLL については、次の事柄を考慮に入れなければなりません:

- DLL はプロシージャーだけで構成されていなければなりません。すべてのサブルーチン (BEGSR) はプロシージャーに対してローカルでなければなりません。
- ソースでは GUI 命令コードを使用できません。これには、START、STOP、SETATR、GETATR、%SETATR、%GETATR、SHOWWIN、CLSWIN、および READS が含まれます。 DSPLY は使用できます。しかし、それが入っているプロシージャーが VisualAge RPG DLL から呼び出された場合には、DSPLY 命令コードは何も実行しません。
- *INZSR; および *TERMSR; は許されません。
- *ENTRY; パラメーターは許されません。

複数プロシージャーのコーディングおよび呼び出しの詳細については、*VisualAge for RPG プログラミング* を参照してください。

OPTION(*{NO}XREF *{NO}GEN *{NO}SECLVL *{NO}SHOWCPY *{NO}EXPDDS *{NO}EXT *{NO}SHOWSKP *{NO}INHERITSIGNON)

OPTION キーワードは、ソース・メンバーのコンパイル時に使用するオプションを指定します。

任意の順序で任意のオプションまたはすべてのオプションを指定できます。ただし、あるコンパイル・オプションを指定すると、同じコンパイル・オプションに対して *NOxxxx パラメーターは指定できず、その逆も同じです。たとえば、*XREF を指定すると、*NOXREF は指定できず、その逆も同じです。オプション間はコロンで区切ります。オプションは複数回指定できません。

注: キーワード OPTION に対からのメンバーが入っていない場合には、この特定のオプションに対してコマンド上に指定された値が使用されることになります。たとえば、制御仕様にキーワード OPTION(*XREF : *NOGEN : *NOSECLVL : *SHOWCPY) が指定されている場合には、対 (*EXT、*NOEXT)、(*EXPDDS、*NOEXPDDS)、および (*SHOWSKP、*NOSHOWSKP) について、コマンド上に暗黙または明示的に指定された値が使用されることになります。

*XREF を指定すると、ソース・メンバーの相互参照リスト (該当する場合) が作成されます。 *NOXREF は、相互参照リストが作成されないことを指示します。

*GEN を指定すると、コンパイラーから戻された最高重大度レベルが GENLVL オプションに指定された重大度を超えない場合にプログラム・オブジェクトが作成されます。*NOGEN の場合には、オブジェクトは作成されません。

*SECLVL を指定すると、メッセージ要約セクションの第 1 レベル・メッセージ・テキストの次の行に第 2 レベル・メッセージ・テキストが印刷されます。*NOSECLVL の場合には、第 1 レベル・メッセージ・テキストの次の行に第 2 レベル・メッセージ・テキストが印刷されません。

*SHOWCPY を指定すると、/COPY コンパイラー指示によって組み込まれたメンバーのソース・レコードがコンパイラー・リストに表示されます。*NOSHOWCPY の場合には、/COPY コンパイラー指示によって組み込まれたメンバーのソース・レコードは表示されません。

*EXPDDS を指定すると、リストされている外部記述ファイルの拡張部分およびキー・フィールド情報が表示されます。*NOEXPDDS の場合は、リストされている外部記述ファイルの拡張部分またはキー・フィールド情報は表示されません。

*EXT を指定すると、コンパイル時に参照される外部プロシージャーおよびフィールドがリストに組み込まれます。*NOEXT の場合には、コンパイル時に参照される外部プロシージャーおよびフィールドはリストに組み込まれません。

*SHOWSKP を指定すると、コンパイラーがスキップしたかどうかに関係なく、リストのソース部分にすべてのステートメントが表示されます。*NOSHOWSKP の場合には、スキップされたステートメントはリストのソース部分に表示されません。コンパイラーは、/IF、/ELSEIF、または /ELSE 指示の結果としてステートメントをスキップします。

*INHERITSIGNON を指定すると、呼び出しアプリケーションのサーバー・サインオン情報が呼び出し先プログラムによって使用されます。これにより、データまたはプログラムがリモート・サーバー上でアクセスされる時に、ユーザー ID / パスワードのプロンプトが出されされません。

OPTION キーワードを指定しないと、コマンド上に指定された値が使用されます。

SQLBINDFILE('ファイル名')

SQLBINDFILE キーワードは、SQL バインド・ファイルの作成を指定します。ユーザーは、完全修飾バインド・ファイル名を単一引用符で囲んで任意に指定することができます。この名前は、最大 8 文字の長さにすることができます。

バインド・ファイルによって、アプリケーションはデータベースへのバインディングを後の時点まで据え置くことができるし、1 つのアプリケーションが多くのデータベースをアクセスすることができます。これは、アプリケーションの実行の前に、SQLBIND コマンドを使用して行なわれます。

SQLPACKAGENAME キーワードを指定しないかぎり、パッケージ・ファイルは生成されません。アプリケーションのビルドは、バインド使用可能 (すなわち SQLPACKAGENAME キーワードを指定) でも、バインド据え置き (パッケージ名なし) でも行なうことができます。バインド可能でビルドすると、パッケージ・ファイルが生成され、データベース中に保管されます。バインド据え置き状態でビルドすると、パッケージの作成に必要なデータをソース・ファイルから取り出し、この情報をバインド・ファイル中に格納します。

SQLDBBLOCKING(*YES | *NO)

SQLDBBLOCKING キーワードは、任意のカーソルに対してブロッキングを実行するかどうかを指定します。任意のカーソルに対してレコード・ブロック化を実行するには、SQLDBBLOCKING(*YES) を指定します。

レコードのブロック化を使用し、SQLISOLATIONLVL(*RR) (読み取り専用カーソル分離レベル) を指定すると、データベース・サーバー側のデータベース・マネージャーは、1 回のネットワーク伝送で、行のブロックをデータベース・クライアントに戻します。これらの行は、データベース・マネージャーが FETCH 要求を処理する時に、1 回に 1 つデータベース・クライアントから取り出されます。ブロック内のすべての行の取り出しが終わると、データベース・クライアント側のデータベース・マネージャーは、すべての出力行の取り出しが完了するまで、リモート・データベースに対して次の要求を送ります。

カーソル固定 (SQLISOLATIONLVL(*CS)) またはアンコミット読み取り (SQLISOLATIONLVL(*UR)) の分離レベルと組み合わせてレコードのブロック化を使用すると、データベースとは完全には整合しない結果となることがあります。カーソル固定およびアンコミット読み取りでは、アプリケーションによってブロックから取り出し中の行は、リモート・データベース側ではロックされません。したがって、ユーザーのアプリケーションがブロックから行を読み取っている間に、別のアプリケーションはデータベース中のその行を更新することができます。反復読み取りの分離レベルを指定すると、データベース内のアクセス済みのすべての行を、その作業単位が完了するまでロックし、別の処理による更新を制限します。

すべてのカーソルでブロック化しない場合には、SQLDBBLOCKING(*NO) を指定します。SELECT ステートメントが複数行を戻す場合には、アプリケーションでカーソルを宣言し、FETCH ステートメントを使用して、行を一度に 1 つ取り出さなければなりません。リモート・データベースを使用する場合に、このことは各要求および各応答がネットワークを経由することを意味します。大量の行を処理する場合には、これによりネットワーク・トラフィックがかなり増大することになります。

SQLDBNAME('Dbname')

SQLDBNAME キーワードは、アプリケーション中の組み込み SQL ステートメントで参照される DB2 データベースの名前を指定します。この名前は単一引用符で囲む必要があり、最大 8 文字の長さにすることができます。

SQLDTFMT(*EUR | *ISO | *USA | *JIS)

SQLDTFMT キーワードは、アプリケーションで使用する日付および時刻形式を指定します。使用できる値は次の通りです:

***EUR** IBM 欧州標準規格形式。

***ISO** 国際標準化機構形式。

***USA** IBM USA 標準規格形式。

***JIS** 日本工業規格西暦形式。

SQLISOLATIONLVL(*RR | *CS | *UR)

SQLISOLATIONLVL キーワードは、ユーザーのアプリケーションによる SQL データベース・レコードの読み取り方法を指定します。使用できる値は次の通りです:

***RR** 反復可能読み取りは、最終コミット点以後に、アプリケーションによってアクセスされるすべての行でのロックを保存します。アプリケーションが同じ行を再度読み取っても、値が変更されることはありません。*RR 分離レベルの影響は、1 つのアプリケーションがその他のアプリケーションまたはユーザーによるテーブルの変更を防止することができることです。結果として、全般的な並行性は低下します。

アプリケーションで行のロックが必要な場合にだけ、反復可能読み取りを指定し、そうでない場合には、カーソル固定を指定します。

***CS** カーソル固定 (*CS) は、カーソルがその行に位置指定されている間だけ行ロックを保持します。カーソルが他の行に移動すると、ロックが解放されます。ただし、データが変更されている場合には、そのデータがコミットされるまでロックが保留されていなければなりません。カーソル固定は読み取られたデータだけに適用されます。すべての変更データは、COMMIT または ROLLBACK のどちらかが処理されるまでロックされたままです。

特定の行をトランザクションの処理中に一度だけアクセスする場合には、カーソル固定を指定します。この方法であれば、並行のアプリケーションおよびユーザーにロックが与える影響はわずかです。

***UR** アンコミット読み取り *UR は、ロックを待たずに行を表示します。アンコミット読み取りは、FETCH および SELECT INTO 命令に適用されます。他の命令の場合に、*UR を選択すると、*CS (カーソル固定) と同じ機能が実行されます。このレベルを使用するアプリケーションは、すべての行を読

み取って、戻します。このことは、その他のアプリケーションによってなされたコミットされていない変更が行に含まれていても同様です。この分離レベルは同時ロックを待機しないので、全体的なパフォーマンスは上がります。

SQLPACKAGENAME('package.txt')

SQLPACKAGENAME キーワードは、実行可能 SQL ステートメントを含むパッケージ・ファイルが作成されることを指定します。ユーザーは、完全修飾パッケージ名を単一引用符で囲んで任意に指定することができます。この名前は、最大 8 文字の長さにすることができます。

データベース・マネージャー・アプリケーションは、アプリケーションのビルドに使用するすべてのビルド済みソース・ファイルに対して、1 つのパッケージ・ファイルを使用します。各パッケージは別個のものであり、同一または別のアプリケーションによって使用されるその他のパッケージとはなんの関係もありません。バインド使用可能でソース・ファイルに対してプリコンパイラーを実行するか、あるいは 1 つまたは複数の DB2 名に対してバインダー（SQLBIND コマンド）を実行することによって、パッケージが作成されます。

SQLPASSWORD('パスワード')

SQLPASSWORD キーワードは、SQL データベースにアクセスするユーザー ID のパスワードを指定します。 パスワードは単一引用符で囲む必要があります。

SQLUSERID('ユーザー ID')

SQLUSERID キーワードは、SQL データベースに接続するユーザー ID を指定します。ユーザー IDは単一引用符で囲む必要があります。

TIMFMT(fmt{区切り文字})

TIMFMT キーワードは、時刻リテラルの内部時刻形式およびプログラム内の時刻フィールドのデフォルト形式を指定します。そのフィールドの定義仕様で TIMFMT キーワードに形式を指定することによって、特定のフィールドに別の内部時刻形式を指定できます。

デフォルトは *ISO です。内部形式の詳細については、107 ページの『内部形式と外部形式』 を参照してください。

236 ページの表 26 は、サポートされている時刻形式およびその区切り文字を示します。

表 26. 時刻データ・タイプの外部時刻形式

RPG 名	説明	形式 (デフォルトの区切り文字)	有効な区切り文字	長さ	例
*HMS	時:分:秒	hh:mm:ss	: . , &	8	14:00:00
*ISO	国際標準化機構	hh.mm.ss	.	8	14.00.00
*USA	IBM USA 標準規格。AM および PM は、大文字小文字の混合とすることができます。	hh:mm AM または hh:mm PM	:	8	02:00 PM
*EUR	IBM 欧州標準規格	hh.mm.ss	.	8	14.00.00
*JIS	日本工業規格西暦	hh:mm:ss	:	8	14:00:00

TRUNCNBR(*YES | *NO)

TRUNCNBR キーワードは、切り捨てられた値を結果フィールドに移動するかどうか、あるいはオブジェクトの実行中に数値オーバーフローが起こった時にエラーを生成するかどうかを指定します。

注: TRUNCNBR オプションは、式内で実行される計算には適用されません。 (式は拡張演算項目 2 フィールドで見つかります。) これらの計算でオーバーフローが起こった場合には、常にエラーが起こります。

*YES を指定すると、数値オーバーフローが無視され、切り捨てられた値が結果フィールドに移動されます。

*NO を指定すると、数値オーバーフローが検出された時に実行時エラーが生成されます。

TRUNCNBR キーワードを指定しないと、コマンド上に指定された値が使用されます。

第 17 章 ファイル仕様

ファイル仕様は、プログラムによって使用される各ファイルを識別します。プログラム中の各ファイルには、対応するファイル仕様ステートメントが必要です。

ファイルは、プログラム記述または外部記述のいずれかにできます。プログラム記述ファイルでは、レコード記述およびフィールド記述は入出力仕様を使用してプログラム内部に組み込まれます。外部記述ファイルには、DDS コマンドまたは SQL/400™ コマンドを使用して iSeries サーバー上で外部から定義されたレコード記述およびフィールド記述が入っています。

各プログラムには以下の制限が適用されます。
- 使用できるファイルの最大数には制限がありません
- DISK ファイル:
  - DISK ファイルはリモートまたはローカルのいずれかにできます。
  - リモート・ファイルは外部記述でなければなりません。
  - ローカル・ファイルはプログラム記述でなければなりません。
- PRINTER ファイル:
  - 使用できる PRINTER ファイルは最大 8 つまで
  - PRINTER ファイルはプログラム記述でなければなりません。
- SPECIAL ファイル:
  - SPECIAL ファイルはプログラム記述でなければなりません。

ファイル仕様ステートメント

ファイル仕様の一般レイアウトは以下のとおりです。
- ファイル仕様タイプ (F) は 6 桁目に入力されます
- 仕様の非コメント部分は、以下のように 7 - 80 桁目に拡張されます。
  - 固定形式項目は 7 ～ 42 桁目に拡張されます。
  - キーワード記入項目は 44 ～ 80 桁目に拡張されます。
- 仕様のコメント・セクションは、81 - 100 桁目に拡張されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
FFilename++IT.A.FRlen+......A.Device+.Keywords++++++++++++++++++++++++++++++Comments++++++++++++
```

図 79. ファイル仕様レイアウト

ファイル記述キーワード継続行

キーワードのための追加のスペースが必要な場合には、次のように、キーワード・フィールドを後続の行に続けることができます :
- 継続行の 6 桁目には F が入っていなければなりません
- 継続行の 7 - 43 桁目はブランクでなければなりません
- 仕様は 44 桁目上またはそれを超えて継続します

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
F.....................................Keywords++++++++++++++++++++++++++++++Comments++++++++++++
```

図 80. ファイル記述キーワード継続行レイアウト

6 桁目 (仕様のタイプ)

F はこの位置に入力しなければなりません。

7-16 桁目 (ファイル名)

項目	説明
有効なファイル名	プログラムで使用される各ファイルには固有の名前が必要です。ファイル名は 1 - 10 文字の長さにすることができ、7 桁目から開始しなければなりません。

外部記述ファイルの場合には、ファイルはコンパイル時および実行時の両方に存在する必要があります。プログラム記述ファイルの場合には、ファイルの存在が必要であるのは実行時だけです。

実行時:

- EXTFILEキーワード、EXTMBR キーワード (リモート OS/400 ファイルの場合のみ）、あるいはその両方を使用した場合には、RPG は、これらのキーワードで指定されたファイルをオープンします。
- そうでない場合には、RPG は 7 桁目で指定されたファイルをオープンします。オープンの時点で、このファイル (または指定変更ファイル) が存在していなければなりません。
- リモート OS/400 ファイルの場合、RPG がオープンするファイルに OS/400 システム指定変更コマンドが使用されていると、その指定変更が効力をもち、この指定変更にしたがって実際のファイルがオープンされます。指定変更とこのキーワードがどのように相互作用するかについては、 247 ページの『EXTFILE(filename)』 を参照してください。

ファイルを実行時にオープンすると、ファイル仕様で指定された順序の逆順にオープンされます。装置名は、関連したファイル上で処理できる操作を定義します。

プログラム記述ファイル

プログラム記述ファイルでは、7 ～ 16 桁目に入力されたファイル名を以下に入力する必要もあります。

- 入力仕様
- 出力仕様または出力演算命令行 (ファイルが出力、更新、または入出力共用ファイルの場合、あるいはファイルの追加が以下のファイル仕様に指定されている場合)
- 定義仕様 (ファイルがテーブルまたは配列ファイルである場合)
- 演算仕様 (ファイル名が指定された操作コードに必要である場合)

外部記述ファイル

外部記述ファイルでは、7 - 16 桁目に入力されたファイル名はファイルのレコード記述を位置指定するために使用された名前です。外部記述ファイルには以下の規則が適用されます。

- 外部記述ファイルの入出力仕様はオプションです。これが必要であるのは、VisualAge RPG 機能 (たとえばレコード識別標識) を検索された外部記述に追加する場合だけです。
- 外部記述を検索すると、レコード定義は入力、出力、または演算仕様でそのレコード様式名によって参照できます。
- レコード様式名は固有な記号名でなければなりません。
- 外部記述論理ファイルでは 2 つのレコード様式を同じ名前にできません。

17 桁目 (ファイル・タイプ)

項目	説明
I	入力ファイルは、ローカル DISK ファイルまたはリモート DISK ファイルのいずれかにできます
O	出力ファイルは、ローカル DISK ファイルまたはリモート DISK ファイルのいずれかにできます
U	更新ファイルは、ローカル DISK ファイルまたはリモート DISK ファイルのいずれかにできます
C	共用 (入出力) ファイルは、リモート DISK ファイルでなければなりません

入力ファイル

プログラムは入力ファイルからの情報を読み取ります。入力ファイルにはデータ・レコード、配列、またはテーブルを入れることができます。

出力ファイル

出力ファイルは情報が書き込まれるファイルです。

更新ファイル

更新ファイルは、読み取りおよび更新できるレコードが入っている入力ファイルです。更新は、ファイルに入っているレコードの 1 つまたは複数のフィールドのデータを更新して、そのレコードを読みとったファイルに書き込みます。レコードを削除する場合には、そのファイルは更新ファイルとして指定する必要があります。

入出力共用ファイル

入出力共用ファイルは、入力ファイルおよび出力ファイルの両方です。入出力共用ファイルを処理すると、出力レコードには出力レコードのフィールドによって表示されたデータだけが入ります。これが更新ファイルとは異なります。更新ファイルでは、出力レコードには出力レコードのフィールドによって変更された入力レコードが入ります。

18 桁目に T (配列またはテーブル置き換えファイル) が入っている場合には、入出力共用ファイルは SPECIAL ファイルおよび DISK ファイルに有効です。

18 桁目 (ファイルの指定)

項目　説明

ブランク
　出力ファイル

T　配列またはテーブル・ファイル

F　全手順ファイル

配列またはテーブル・ファイル

18 桁目の T によって指定される配列ファイルおよびテーブル・ファイルは、プログラム初期設定時にロードされます。配列ファイルまたはテーブル・ファイルは、入力または結合できます。配列またはテーブル出力ファイルのこの項目はブランクのままにしてください。 SPECIAL は配列およびテーブル入力ファイルの装置として指定できません。外部記述ファイルは配列またはテーブル・ファイルとして指定できません。

T が 18 桁目で指定される場合には、 DISK ファイルに結合されたファイル・タイプ (17 桁目の C) を指定できます。結合されたファイル・タイプによって、配列ファイルまたはテーブル・ファイルは同じファイル (配列またはテーブル置き換えファイル) または別のファイルとの読み取りまたは書き込みができます。さらに、7 - 16 桁目のファイル名も定義仕様の TOFILE キーワードに対するパラメーターとして指定される必要があります。

全手順ファイル

全手順ファイルでは、入力が演算命令によって制御されます。ファイル命令コード (たとえば CHAIN または READ) が入力機能を実行するために使用されます。

19 桁目 (予約済み)

項目　説明

ブランク
　この項目はブランクでなければなりません。

20 桁目 (ファイルの追加)

20 桁目は、レコードが入力ファイルまたは更新ファイルに追加されるかどうかを示します。出力ファイルでは、この項目は無視されます。

項目　説明

ブランク
　レコードは入力ファイルまたは更新ファイル (17 桁目の I または U) に追加できません。

A　レコードは、ファイルの出力レコード仕様の 18 - 20 桁目に「ADD」が入っているとき、あるいは WRITE 命令コードが演算仕様で使用されるときに、入力ファイルまたは更新ファイルに追加されます。

ファイル仕様の 17 桁目および 20 桁目間の関係と、出力仕様の 18 - 20 桁目については、 241 ページの表 27を参照してください。

表 27. ファイルの機能の処理

	仕様		
	ファイル記述		出力
機能	17 桁目	20 桁目	18 - 20 桁目
新ファイル [1] を作成 または レコードを既存のファイルに追加	O O	ブランク A	ブランク ADD
ファイルの処理	I	ブランク	ブランク
ファイルを処理し、レコードを既存のファイルに追加	I	A	ADD
ファイルを処理し、レコードを更新 (更新または削除)	U	ブランク	ブランク
ファイルを処理し、新規レコードを既存のファイルに追加	U	A	ADD
ファイルを処理し、既存のレコードをファイルから削除	U	ブランク	DEL
: [1] RPG 内部では、*新ファイルの作成*という用語はレコードを新しく作成されたファイルに追加することを意味します。すなわち、このテーブルの最初の 2 つの項目は同一の機能を実行します。両方がリストされて、その機能を指定する 2 つの方法があることを表示します。			

21 桁目 (予約済み)

項目　説明

ブランク
この項目はブランクでなければなりません。

22 桁目 (ファイル形式)

項目　説明

F　プログラム記述ファイル

E　外部記述ファイル

22 桁目の F は、ファイルのレコードが入出力仕様 (配列 / テーブル・ファイルを除く) のプログラム内部に記述されることを示します。 PRINTER ファイルおよび SPECIAL ファイルはプログラム記述でなければなりません。ローカル DISK ファイルはプログラム記述でなければなりません。

22 桁目の E は、ファイルのレコード記述が VisualAge RPG ソース・プログラムの外部であることを示します。コンパイラーはこれらの記述をコンパイル時に入手し、ソース・プログラムに組み込みます。リモート DISKファイルは外部記述でなければなりません。

23 - 27 桁目 (レコード長)

23 - 27 桁目は、プログラム記述ファイルに入っている論理レコードの長さを示すために使用します。指定できる最大レコード・サイズは 32766 です。ただし、装置のレコード・サイズ制約がこの値をオーバーライドする場合があります。 PRINTER ファイルでは、プリンター出力の欄数を超えないレコード長を指定してください。外部記述ファイルのこの項目はブランクです。

28 桁目 (予約済み)

項目　説明

ブランク

この項目はブランクでなければなりません。

29 - 33 桁目 (予約済み)

項目　説明

ブランク

この項目はブランクでなければなりません。

34 桁目 (レコード・アドレスの種類)

項目　説明

ブランク

相対レコード番号を使用して、ファイルを処理します。レコードは連続的に読み取られます。

K　キー値を使用して、ファイルを処理します。この項目が有効であるのは外部記述ファイルの場合だけです。

ブランク = 非キー処理

ブランクは、ファイルがキーを使用せずに処理されることを示します。

キーなしで処理されるファイルは、相対レコード番号によって連続的またはランダムに処理できます。

相対レコード番号による入力処理は、34 桁目のブランクによって、および CHAIN 、 SETLL 、または SETGT 命令コードの使用によって決定されます。相対レコード番号による出力処理は、34 桁目のブランクによって、およびファイル仕様での RECNO キーワードの使用によって決定されます。

キー

K 項目は、アクセス・パスをキー値でビルドすることを前提として外部記述ファイルが処理されることを示します。処理がランダムである場合には、キー値はレコードを識別するために使用されます。

キー付きファイルでこの位置がブランクである場合には、レコードは到着順に検索されます。

35 桁目 (予約済み)

項目　説明

ブランク

この項目はブランクでなければなりません。

36 - 42 桁目 (装置)

項目	説明
PRINTER	ファイルはプリンター・ファイルであり、プリンターに送信できる制御文字が付いています。
DISK	ファイルはディスク・ファイルです。リモート・ファイルでは、順次およびランダムの読み取り / 書き込み処理が可能です。ローカル・ファイルでは、順次および相対レコード処理が可能です。
SPECIAL	これは特殊ファイルです。入力または出力は、 VisualAge RPG アプリケーションにリンクしているユーザー提供コードによってアクセスされる装置上にあります。ユーザー提供コード・モジュールの名前は、 PROCNAMEキーワードのパラメーターとして指定されなければなりません。パラメーター・リストはこのプログラムでの使用のために作成され、オプション・コード・パラメーターおよび状況コード・パラメーターが入っています。このファイルは固定非ブロック化形式でなければなりません。詳細については、 249 ページの『PLIST(Plist_name)』および 251 ページの『PROCNAME(proc_name)』を参照してください。

36 - 42 桁目は、このファイルと関連した装置名を指定するために使用します。装置名は、関連したファイルで実行できる機能を定義します。特定の機能は特定の装置名だけに有効です。

43 桁目 (予約済み)

43 桁目はブランクでなければなりません。

---

44 ～ 80 桁目 (キーワード)

44 - 80 桁目は、ファイル仕様キーワード用に提供されます。キーワードは、定義されるファイルについての追加情報を提供するために使用されます。

ファイル仕様キーワードには、パラメーター、オプション・パラメーター、または必須パラメーターがない場合があります。キーワードの構文は以下のとおりです。

```
Keyword(parameter1 : parameter2)
```

ここで:

- パラメーター (1 つまたは複数) は括弧 ( ) で囲みます。

  注: パラメーターがない場合には括弧を指定しないでください。
- コロン (:) は複数のパラメーターを区切るのに使用します。

次の国際規則を使用して、どのパラメーターが任意指定で、どれが必須かを示します:

- 中括弧 { } は、オプション・パラメーターまたはパラメーターのオプション・エレメントを示します。
- 省略記号 (...) は、パラメーターを繰り返し使用できることを示します。
- コロン (:) はパラメーター間を区切り、複数のパラメーターを指定できることを示します。 コロンで区切られたすべてのパラメーターは、それらが中括弧で囲まれていない限り必須です。
- 縦線 (|) は、キーワードに 1 つのパラメーターしか指定できないことを示します。
- キーワード・パラメーターを区切るブランクは、1 つまたは複数のパラメーターを指定できることを示します。

注: 中括弧、省略記号、および縦線はキーワード構文の一部ではなく、ソースの中に入力してはいけません。

キーワードに追加のスペースが必要である場合には、キーワード・フィールドは次行に続けることができます。 237 ページの『ファイル記述キーワード継続行』および 218 ページの『ファイル仕様キーワード・フィールド』を参照してください。

以下のテーブルは、外部記述ファイルに適用されるキーワードおよびプログラム記述ファイルに適用されるキーワードを要約しています。

キーワード	プログラム記述	外部記述
ブロック		Y
COMMIT{(rpg_name)}		Y
CVTHEX		Y
DATFMT(format{separator})	Y	Y
EOFMARK(*NONE)	Y	
EXTFILE(fname)	Y	
EXTMBR(membername)		Y
FORMLEN(number)	Y	
IGNORE(recformat{:recformat...})		Y
INCLUDE(recformat{:recformat...})		Y
INFDS(DSname)	Y	Y
INFSR(SUBRname)	Y	Y
PLIST(Plist_name)	Y	Y
PREFIX(prefix_name)		Y
PROCNAME(proc_name)	Y	
PRTCTL(data_struct{:*COMPAT})	Y	
PRTFMT(*SYS | *TEXT)	Y	
RCDLEN(fieldname)	Y	
RECNO(fieldname)	Y	Y
REMOTE		Y
RENAME(Ext_format:Int_format)		Y
TIMFMT(format{separator})	Y	Y

キーワード	プログラム記述	外部記述
USROPN	Y	Y

BLOCK(*YES|*NO)

BLOCK キーワードは、ファイルと関連したレコードのブロック化を制御します。このキーワードは、 DISK ファイルだけに有効です。

このキーワードを指定しない場合には、以下の条件に適合するときに VARPG コンパイラーは入力レコードの非ブロック化および出力レコードのブロック化を行なって、 DISK ファイルでの実行時パフォーマンスを改善します。

1. ファイルが外部記述であり、レコード様式を 1 つだけもつ。
2. RECNO キーワードがファイル仕様で使用されない。
3. 以下のいずれかが真である。
   a. ファイルが出力ファイルである。
   b. ファイルが入出力共用ファイルの場合には、配列ファイルまたはテーブル・ファイルである。
   c. ファイルが入力専用ファイルであり、 READE、 READPE、 SETGT、 SETLL、および CHAIN 操作がそのファイルで使用されない。 (READE または READPE 操作を使用すると、入力ファイルのレコード・ブロックは起こりません。 SETGT、 SETLL、または CHAIN 操作を使用すると、 BLOCK(*YES) キーワードが入力ファイルに指定されない限り、レコード・ブロックは起こりません。)

BLOCK(*YES) を指定すると、入力ファイルで SETGT、SETLL、および CHAIN 操作を使用してもブロック化がまだ起こる場合 (上記の条件 3c を参照) を除いて、レコード・ブロックは上記のように起こります。

レコードをブロック化しないようにするには、 BLOCK(*NO) を指定できます。レコード・ブロックはコンパイラーによって実行されません。

COMMIT{(rpg_name)}

COMMIT キーワードによって、コミットメント制御下でリモート・ファイルを処理するオプションが使用できます。オプション・パラメーター rpg_name が指定される場合があります。パラメーターは、タイプ標識のフィールド (つまり、長さ 1 の文字フィールド) として暗黙的に定義され、「0」に初期設定されます。

オプション・パラメーターを指定することにより、コミットメント制御を実行時に可能にするかどうかをプログラマーが制御できます。パラメーターに「1」が入る場合にはファイルは COMMIT をオンにしてオープンされ、そうでない場合にはファイルは COMMIT なしでオープンされます。パラメーターはファイルをオープンする前に設定しなければなりません。ファイルをプログラム初期化時にオープンすると、パラメーターはパラメーターを介して渡すことができます。ファイルを明示的にオープンする場合には、演算仕様の OPEN 操作を使用して OPEN 操作の前に設定できます。

このファイルおよび現在コミットメント制御下にある他のファイルに対するグループ変更には、 COMMIT および ROLBK 操作コードを使用してください。すると、変更はすべて同時に起こるか、あるいはまったく起こりません。

注: ファイルが共用オープン・データ・パスによってすでにオープンされている場合には、コミットメント制御の値は前の OPEN 操作の値と一致していなければなりません。

CVTHEX

CVTHEX は、CCSID 65535 のデータベース・フィールドが含まれる外部記述リモート・ディスク・ファイルの処理をサポートします。

CCSID 値 65535 は、フィールド・データへのアクセス時に変換が行われず、サーバー上のこれらのフィールドにある従来の EBCDIC データが、ANSI で稼動しているクライアント・ワークステーションで認識されないことがあることを意味します。

ファイルに CVTHEX を指定すると、ファイル内の 65535 CCSID の文字フィールドが、アプリケーションでの使用のための入出力操作でワークステーション CCSID に変換されます。 (クライアント・サイド変換プロセスは、フィールドの 65535 CCSID の代わりにサーバー接続ジョブの CCSID を使用して変換を実行します。)

注: CVTHEX は、JAVA へのコンパイル時にはサポートされません。

DATFMT(format{separator})

DATFMT キーワードによって、プログラム記述ファイルにあるすべての日付フィールドのデフォルト外部日付形式およびデフォルト区切り文字 (オプション) の仕様を使用できます。このキーワードが指定されているファイルが示されてキー・フィールドが日付である場合には、これはキー・フィールドのデフォルト形式も提供します。ファイルはリモートまたはローカルのいずれかにできます。

対応する入力仕様 (31 - 35 桁目) または出力仕様 (53 - 57 桁目) でフィールドの日付形式 / 区切り記号を指定することにより、ファイルにある個々の入力または出力日付フィールドに異なる外部形式を指定できます。

日付入力フィールドでは、これはデフォルトの外部日付形式 / 区切り記号 (入力仕様の 31 - 35 桁目) を指定します。

日付出力フィールドでは、これはデフォルトの外部日付形式 / 区切り記号 (出力仕様の 53 - 57 桁目) を指定します。

日付形式および区切り文字については、 227 ページの『DATFMT(fmt{区切り文字})』を参照してください。外部形式の詳細については、 107 ページの『内部形式と外部形式』 を参照してください。

EOFMARK(*NONE)

EOFMARK(*NONE) キーワードは、ローカル・ディスク・ファイルから「ファイルの終わり」マーカーを省略するために指定します。 *NONE パラメーターは必須です。

EXTFILE(filename)

EXTFILE キーワードによって、コンパイル時に名前を提供するのではなく、実行時にオープンされる実際のファイル名を指定できます。値は、リテラルまたは変数とすることができます。

注:

1. 変数名を使用する場合には、その変数名をファイルがオープンされる前に設定しなければなりません。プログラムの初期化のときに自動的にオープンされるファイルの場合、以下のいずれかの方法で変数を設定しなければなりません。
   - D 仕様に INZ キーワードを使用する
   - 入り口点パラメーターとして値を渡す

ローカル・ファイル

ファイルはローカルの DISK または PRINTER ファイルでなければなりません。EXTFILE キーワードと一緒に USROPN キーワードも指定する必要があります。

リモート OS/400 ファイル

以下の形式のいずれかで値を指定することができます。

```
filenamelibname/filename
*LIBL/filename
```

注:

1. ライブラリー名として *CURLIB を指定することはできません。
2. ライブラリー名なしにファイル名を指定した場合には、*LIBL が使用されます。
3. 名前は、大文字小文字を正しく指定しなければなりません。たとえば、EXTFILE(filename) を指定し、変数 filename に値 'qtemp/myfile' がある場合には、ファイルはオープンされません。代わりに、値 'QTEMP/MYFILE' が必要になります。
4. このキーワードは、コンパイル時に外部記述ファイルを見つけるのに使用されません。

RPG がオープンするファイルに指定変更を指定している場合には、その指定変更が効力をもちます。以下のコードでは、RPG プログラム内の **INPUT** という名前のファイルの場合、実行時にオープンされるファイルは、***filename*** フィールドの値によって決定されます。

```
Finput     if   e            disk    extfile(filename) remote
```

実行時に ***filename*** フィールドの値が MYLIB/MYFILE であると、RPG はファイル MYLIB/MYFILE をオープンします。コマンド OVRDBF MYFILE OTHERLIB/OTHERFILE が使用された場合には、オープンされる実際のファイルは OTHERLIB/OTHERFILE となります。名前 INPUT は RPG ソース・メンバー内で使用される唯一の名前であるので、名前 INPUT に対する指定変更はどれも無視されることに注意してください。

EXTMBR(membername)

EXTMBR キーワードは、オープンされるファイルのメンバーを指定します。メンバー名 '*ALL' または '*FIRST' を指定することができます。*ALL および *FIRST

は、メンバー「名」であり、 RPG特殊語でないので、引用符付きで囲んで指定しなければなりません。値は、リテラルまたは変数とすることができます。デフォルトは '*FIRST' です。

名前は、大文字小文字を正しく指定しなければなりません。たとえば EXTMBR(mbrname) を指定し、変数 mbrname に値 'mbr1' がある場合には、メンバーは見つかりません。代わりに、値 'MBR1' が必要になります。

変数名を使用する場合には、その変数名をファイルがオープンされる前に設定しなければなりません。プログラムの初期化のときに自動的にオープンされるファイルの場合、以下のいずれかの方法で変数を設定しなければなりません。
- D 仕様に INZ キーワードを使用する
- 入り口点パラメーターとして値を渡す

FORMLEN(number)

FORMLEN キーワードは、 PRINTER ファイルの用紙の長さを指定するために使用します。用紙の長さは、 1 以上 255 以下でなければなりません。パラメーターは、使用する形式またはページで使用できる正確な行数を指定します。行数が FORMLEN に一致するときには、自動用紙送りが挿入されます。

IGNORE(recformat{:recformat...})

IGNORE キーワードにより、外部記述ファイルからのレコード様式が無視されます。無視されるレコード様式の外部名は、パラメーター recformat として指定されます。 1 つまたは複数のレコード様式を、コロンで (:) で区切って指定できます。プログラムは、指定されたレコード様式が存在しないかのように実行されます。ファイルに入っている他のすべてのレコード様式は組み込まれます。

IGNORE キーワードをファイルに指定すると、 INCLUDE キーワードは指定できません。

INCLUDE(recformat{:recformat...})

INCLUDE キーワードは、組み込まれるレコード様式名を指定します。ファイルに入っている他のすべてのレコード様式は無視されます。複数のレコード様式を、コロン (:) で区切って指定できます。

INCLUDE キーワードをファイルに指定すると、 IGNORE キーワードは指定できません。

INFDS(DSname)

INFDS キーワードにより、ファイルと関連したフィードバック情報が入っているデータ構造を定義および名前付けできます。データ構造名は INFDS のパラメーターとして指定されます。 INFDS が複数のファイルに指定されている場合には、各関連データ構造には固有の名前が必要です。 INFDS を定義できるのはメイン・ソース・セクションだけです。

INFSR(SUBRname)

このキーワードにパラメーターとして指定されたファイル例外 / エラー・サブルーチンは、次のファイル例外 / エラーの制御を受け取る場合があります。サブルーチン名は *PSSR であり、これはユーザー定義のプログラム例外 / エラー・サブルーチンがこのファイルでエラーの制御を得ることを示します。

INFSR キーワードは、ファイルがサブプロシージャーによってアクセスされる場合には指定できません。

PLIST(Plist_name)

PLIST は、そのパラメーターとして SPECIAL ファイルのプログラムに渡されるパラメーター・リストの名前を提供します。プロシージャーは、PROCNAME(proc_name) キーワードを使用して指定されます。この項目が有効であるのは、ファイル記述行に指定された装置 (36 - 42 桁目) が SPECIAL のときだけです。この項目によって識別されるパラメーターは、プログラムによって渡されるパラメーター・リストの最後に追加されます。

PREFIX(prefix{:nbr_of_char_replaced})

PREFIX キーワードは、外部記述ファイルでフィールドを一部名前変更するために使用されます。指定された文字ストリングまたは文字リテラルが、7 - 16 桁目に指定されたファイルのすべてのレコードに定義されたすべてのフィールドの名前の前に付きます。さらに、任意で数値を指定して、置き換えられる既存の名前の文字数を指定することができます (存在する場合) 。「nbr_of_char_replaced」を指定しない場合には、ストリングは名前の先頭に付加されます。

「nbr_of_char_replaced」を指定する場合には、小数部のない 0 - 9 の値が入っている数値固定情報でなければなりません。たとえば、仕様 PREFIX(YE:3) はフィールド名「YTDTOTAL」を「YETOTAL」に変更します。ゼロの値を指定すると、「nbr_of_char_replaced」を指定しない場合とまったく同じになります。

規則:

- PREFIX キーワードがファイルに指定されている場合であっても、入力仕様のフィールドを明示的に名前変更できます。コンパイラーは、プログラムで最初に使用される名前を認識 (および必要と) します。たとえば、入力仕様のプレフィックス名を指定してそのフィールドを標識と関連づけてから、非プレフィックス名を参照しているフィールドを名前変更しようとする場合には、エラーが起こります。逆に、フィールドを最初にプレフィックス名でない名前に名前変更してから仕様のプレフィックス名を使用する場合には、コンパイル時にエラーが起こります。
- プレフィックス適用後の名前の全長は、 VisualAge RPG フィールド名の最大長を超えることはできません。
- プレフィックスされる名前の文字数は、「nbr_of_char_replaced」パラメーターによって表示される値以下にしないでください。つまり、プレフィックスを適用した結果の名前は、プレフィックス・ストリングと同じにすることができません。
- プレフィックスが文字リテラルの場合には、ピリオドで終わらなければなりません。
- プレフィックスが文字リテラルの場合には、英大文字でなければなりません。この場合には、フィールド名は、同じ修飾データ構造のすべてのサブフィールドでなければなりません。

例:

以下の例では、MYFILE 中のフィールドを修飾データ構造 MYDS のサブフィールドと関連付けるために、プレフィックス「MYDS.」が使用されています。

```
Fmyfile    if   e            disk    prefix('MYDS.') remote
D myds          e ds                   qualified extname(myfile)
```

次の例では、MYFILE 中のフィールドを修飾データ構造 MYDS のサブフィールドと関連付けるために、プレフィックス 'MYDS.F2':3 が使用されています。サブフィールド自体は、最初の 3 文字を 'F2' で置き換えることによってさらにプレフィックス付けされています。このファイルによって使用されるフィールドは、MYDS2.F2FLD1 および MYDS2.F2FLD2 となります。(データ構造 MYDS2 を同じプレフィックスで定義しなければなりません。しかしデータ構造名が含まれていないため、正確に同じではありません。)

```
A          R REC
A            ACRFLD1       10A
A            ACRFLD2        5S 0
```

```
Fmyfile2   if   e            disk    prefix('MYDS2.F2':3) remote
D myds2         e ds                 qualified extname(myfile)
D                                    prefix('F2':3)
```

PROCNAME(proc_name)

SPECIAL が装置項目 (36 - 42 桁目) であるときには、 PROCNAME に対するパラメーターとして指定されたユーザー提供のコード・モジュールは特殊入出力装置のサポートを処理します。詳細については、 243 ページの『36 - 42 桁目 (装置)』および 249 ページの『PLIST(Plist_name)』を参照してください。

PRTCTL(data_struct{:*COMPAT})

PRTCTL キーワードは動的印刷装置制御の使用を指定します。パラメーター data_struct として指定されるデータ構造は、形式制御情報および行カウント値を参照します。 PRTCTL キーワードは、プログラム記述ファイルだけに有効です。

オプション・パラメーター *COMPAT は、データ構造レイアウトが RPG III と互換性のあることを示します。 *COMPAT を指定しない場合には、拡張長さデータ構造が使用されます。

拡張長さ PRTCTL データ構造

このデータ構造には最小 15 バイトが必要です。 PRTCTL データ構造のレイアウトは以下のとおりです。

データ構造の位置	サブフィールドの内容
1 - 3	印刷前行送り値が入っている 3 桁の文字フィールド (ブランクまたは 0 - 255)
4 - 6	印刷後行送り値が入っている 3 桁の文字フィールド (ブランクまたは 0 - 255)
7 - 9	前スキップ値が入っている 3 桁の文字フィールド (有効な項目: ブランクまたは 1 - 255)
10 - 12	後スキップ値が入っている 3 桁の文字フィールド (ブランクまたは 1 - 255)
13 - 15	現在行カウント値が入っている小数点以下の桁がゼロである 3 桁の数値 (ゾーン 10 進数) フィールド

*COMPAT PRTCTL データ構造

データ構造の位置	サブフィールドの内容
1	印刷前行送り値が入っている 1 桁の文字フィールド (ブランクまたは 0 - 3)
2	印刷後行送りが入っている 1 桁の文字フィールド (有効な項目: ブランクまたは 0 - 3)
3 - 4	前スキップ値が入っている 2 桁の文字フィールド (ブランク、 1 - 99 、 100 - 109 の場合は A0 - A9 、 110 - 112の場合は B0 - B2)
5 - 6	後スキップ値が入っている 2 桁の文字フィールド (ブランク、 1 - 99 、 100 - 109 の場合は A0 - A9 、 110 - 112の場合は B0 - B2)
7 - 9	現在行カウント値が入っている小数点以下の桁がゼロである 3 桁の数値 (ゾーン 10 進数) フィールド

拡張長さデータ構造の最初の 4 つのサブフィールドにある値は、出力仕様の 40 ～ 51 桁目 (スペースおよびスキップ項目) で使用できる値と同じです。出力仕様のスペースおよびスキップ項目 (40 ～ 51 桁目) がブランクの場合、およびサブフィールド 1 ～ 4 もブランクである場合には、デフォルトは後に 1 つスペースを入れます。PRTCTL オプションが指定される場合には、40 ～ 51 桁目にブランクがある出力レコードだけに使用されます。プログラムの実行中にこれらのサブフィールドの値を変更することにより、 PRINTER ファイルのスペースおよびスキップ値 (サブフィールド 1 ～ 4) を制御できます。

サブフィールド 5 には、現在行カウント値が入っています。 VisualAge RPG コンパイラーは、最初の出力行が印刷されるまでサブフィールド 5 を初期化しません。ファイルに対する各出力操作の後で、 VisualAge RPG コンパイラーはサブフィールド 5 を変更します。

PRTFMT(*SYS | *TEXT)

PRTFMT キーワードとパラメーター *SYS をプリンター・ファイルで使用して、デフォルトのロー・テキスト出力ではなく、装置コンテキストと、オペレーティング・システムのグラフィック装置インターフェース呼び出しを介してアプリケーションがプリンターへの出力を実行することを指定できます。

プリンター・ファイルをオープンした後に、独自の Windows GDI 呼び出しを行うときにアプリケーションが参照できるように、アプリケーション装置コンテキスト・ハンドルがプリンター・ファイル INFDS の 81 から 84 桁目にコピーされます。

デフォルトは *TEXT です。この場合、アプリケーションのテキスト・データが直接出力されます。

注: PRTFMT は、JAVA へのコンパイル時には適用されません。

RCDLEN(fieldname)

RCDLEN キーワードは、ローカル DISK ファイルに使用できます。フィールド名パラメーターは、小数部がゼロの数字でなければなりません。入力ファイルでは、フィールド名には読み取ったレコードの長さが入っています。出力ファイルでは、フィールド名は書き込んだレコードの長さが指定されます。23 - 27 桁目に指定されるレコード長は、最大フィールド長を定義します。 RCDLEN はこのレコード長以下でなければなりません。書き込みできる最小のレコード長はゼロです。RECLEN で指定される値がゼロより少ない場合には、ゼロに切り上げられます。

RCDLEN キーワードが表示される場合には、ファイルは可変長レコードが入っているように扱われます。キーワードが表示されない場合には、ファイルは固定長レコードが入っているように扱われます。

注: RCDLEN フィールドが出力で設定される場合には、使用されるデータ構造の長さをオーバーライドします。

RECNO(fieldname)

このキーワードは、相対レコード番号によって処理される DISK ファイルのオプションです。 RECNO キーワードは相対レコード番号によって処理される出力ファイ

ル、ランダム WRITE 演算命令によって参照される出力ファイル、または出力仕様で ADD によって使用される出力ファイルに指定する必要があります。

注: RECNO キーワードを指定しない場合には、レコード・ブロックが起こります。

RECNO キーワードは入力 / 更新ファイルに指定できます。ファイル (たとえば READ 、 SETLL 、または OPEN) を位置変更するすべての操作では、検索されるレコードの相対レコード番号は「fieldname」に置かれます。これは小数点以下にゼロがある数字として定義しなければなりません。フィールド長は、ファイルの最長レコード番号が入る長さでなければなりません。

RECNO キーワードをファイルの追加がある入力ファイルまたは更新ファイルに指定するときには (20 桁目の 'A') 、正常に実行される出力操作では、フィールド名パラメーターの値は削除済みレコードの相対レコード番号を参照する必要があります。

注: RECNO キーワードは、ローカル・ファイルに書き込み (WRITE) を行なっている場合には無視されます。

REMOTE

REMOTE キーワードは、ディスク装置が iSeries サーバー上に常駐することを指定します。

RENAME(Ext_format:Int_format)

RENAME キーワードによって、外部記述ファイルのレコード形式を名前変更できます。名前変更されるレコード様式の外部名は、 Ext_format パラメーターとして入力されます。 Int_format パラメーターは、プログラムで使用されるときのレコードの名前です。外部名はプログラムでこの名前によって置き換えられます。

プレフィックスを追加することによりすべてのフィールドを名前変更するには、PREFIX キーワードを使用します。

TIMFMT(format{separator})

TIMFMT キーワードによって、プログラム記述フィールドですべての時刻フィールドのデフォルト外部時刻形式およびデフォルト区切り文字 (オプション) の仕様が使用できます。 このキーワードが指定されているファイルが示されてキー・フィールドが時刻である場合には、指定された時刻形式はキー・フィールドのデフォルト形式も提供します。ファイルはローカルまたはリモートのいずれかにできます。

対応する入力仕様 (31 - 35 桁目) または出力仕様 (53 - 57 桁目) でフィールドの時刻形式 / 区切り文字を指定することにより、ファイルにある個々の入力または出力時刻フィールドに異なる外部形式を指定できます。

有効な形式および区切り文字については、 144 ページの表 18を参照してください。
外部形式の詳細については、 107 ページの『内部形式と外部形式』を参照してください。

USROPN

USROPN キーワードにより、プログラム初期化時にファイルがオープンされなくなります。これにより、プログラマーはファイルを最初にオープンするための制御を得ます。ファイルは、演算仕様で OPEN 操作を使用して明示的にオープンされなければなりません。このキーワードは、テーブル・ファイルとして指定された入力ファイルには無効です。

USROPN キーワードは、最初にファイルをオープンするプログラマー制御に必要です。たとえば、ファイルをオープンして後で CLOSE 操作によってクローズする場合には、ファイル仕様で USROPN キーワードを指定することなく (OPEN 操作を使用して) そのファイルを再オープンできます。

ファイル・タイプおよび処理方式

以下のテーブルは、各種のファイル・タイプおよび処理方式に有効なファイル仕様の 28、 34、および 35 桁目の項目を表示しています。ディスク・ファイル処理の方式には以下があります。

- 相対レコード番号処理
- 連続処理
- キー順処理
- キーによるランダム処理

注: ローカル DISK ファイルを処理できるのは、順次または相対レコードによってだけです。

表 28. DISK ファイルの処理方式

アクセス	Method	命令コード	28 桁目	34 桁目	35 桁目	説明
ランダム	RRN	CHAIN	ブランク	ブランク	ブランク	レコードの物理順によるアクセス
順次	キー	READ READE READP READPE	ブランク	K	ブランク	キー順によるアクセス
順次	RRN	READ	ブランク	ブランク	ブランク	順次アクセス
ランダム	キー	CHAIN	ブランク	K	ブランク	キーによりランダムにアクセス

第 18 章 定義仕様

定義仕様は、データ構造、データ構造サブフィールド、プロトタイプ、プロシージャー・インターフェース、プロトタイプ・パラメーター、独立型フィールド、名前付き固定情報、および「メッセージ」ウィンドウを定義するために使用することができます。

定義する場所によって、定義できるものと、また、定義の有効範囲の両方に相違が生じます。24〜25 桁目に、定義のタイプを次のように指定してください。

項目　定義タイプ

ブランク
　データ構造サブフィールドまたはパラメーター定義

C　名前付き固定情報

DS　データ構造

PI　プロシージャー・インターフェース

PR　プロトタイプ

S　独立型フィールド

データ構造、プロトタイプ、およびプロシージャー・インターフェースの定義は、24-25 桁目がブランクでない最初の定義仕様、または定義仕様でない最初の仕様で終わります。

定義仕様は、モジュールまたはプログラム内の次の 2 つの場所に表すことができます: メイン・ソース・セクション内とサブプロシージャー内。メイン・ソース・セクションでは、すべてのグローバル定義を作成します。サブプロシージャーでは、プロトタイプの必要に応じてプロシージャー・インターフェースとそのパラメーターを定義します。また、プロトタイプ・プロシージャーが処理される時にそのプロシージャーに必要とされるすべてのローカル・データ項目も定義されます。プロトタイプ・プロシージャー内の定義は、すべてがローカルです。それらは、(メイン・プロシージャーを含む) 他のすべてのプロシージャーには認識されません。メイン・ソース・セクションの構造および定義の配置が有効範囲にどのように影響するかの詳細については、 256 ページの『定義の配置と有効範囲』 を参照してください。

定義仕様では、配列およびテーブルをデータ構造サブフィールドか独立型フィールドのいずれかとして定義することができます。配列およびテーブルの定義および使用法の追加情報については、 175 ページの『第 12 章 配列およびテーブルの使用』 を参照してください。

定義仕様のキーワード・フィールドには、組み込み関数 (BIF) をキーワードに対するパラメーターとして指定することができます。定義仕様上の組み込み関数は、すべての引き数の値がコンパイル時に通用する場合にのみ許されます。BIF のすべての引き数を定義仕様キーワードの DIM、OCCURS、OVERLAY、および PERRCD

として定義する場合には、仕様の前半で定義する必要があります。組み込み関数の使用法の詳細については、 399 ページの『第 25 章 組み込み関数』 を参照してください。

データ構造、固定情報、データ・タイプ、およびデータ形式の詳細については、 107 ページの『第 9 章 データ・タイプおよびデータ形式』、 165 ページの『第 11 章 データ構造』、および 157 ページの『第 10 章 リテラルおよび名前付き固定情報』 を参照してください。プロトタイプの詳細については、 73 ページの『プロトタイプおよびパラメーター』 を参照してください。

定義の配置と有効範囲

定義する場所によって、その有効範囲が異なることになります。**有効範囲**は、ソース行の名前が通用する範囲を指します。有効範囲には、グローバルとローカルの 2 つのタイプがあります。図 81 は、モジュール内の定義の配置が有効範囲とどのように関係するかを示しています。 257 ページの図 82 は、可能なコンパイル・ターゲットであるコンポーネント、NOMAIN DLL、または EXE のおのおのの場合のメイン・ソース・セクションのレイアウトを示しています。

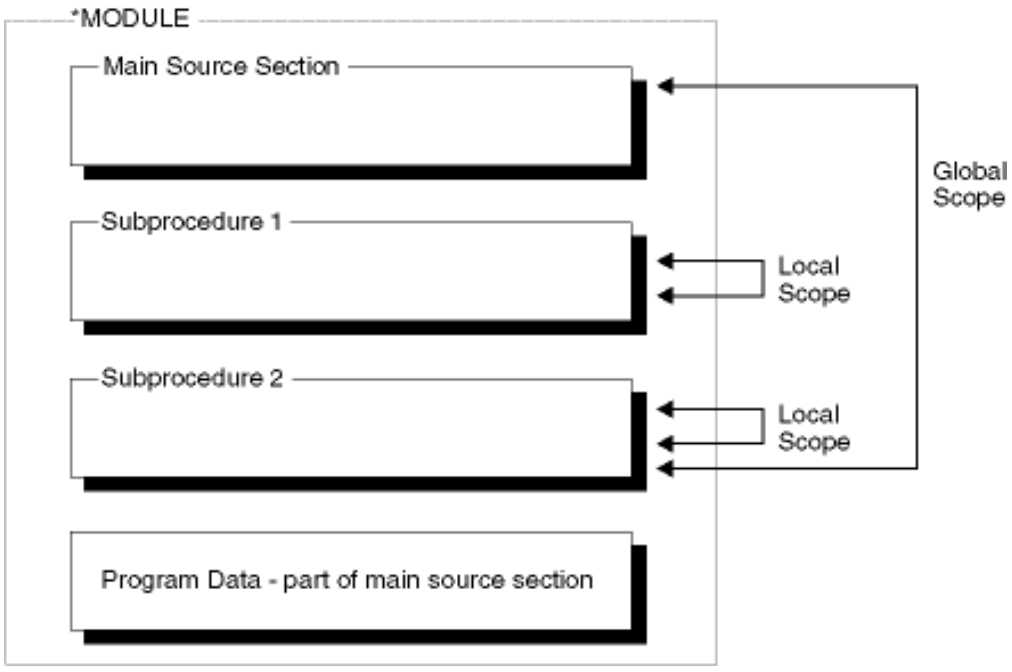


図 81. 定義の有効範囲

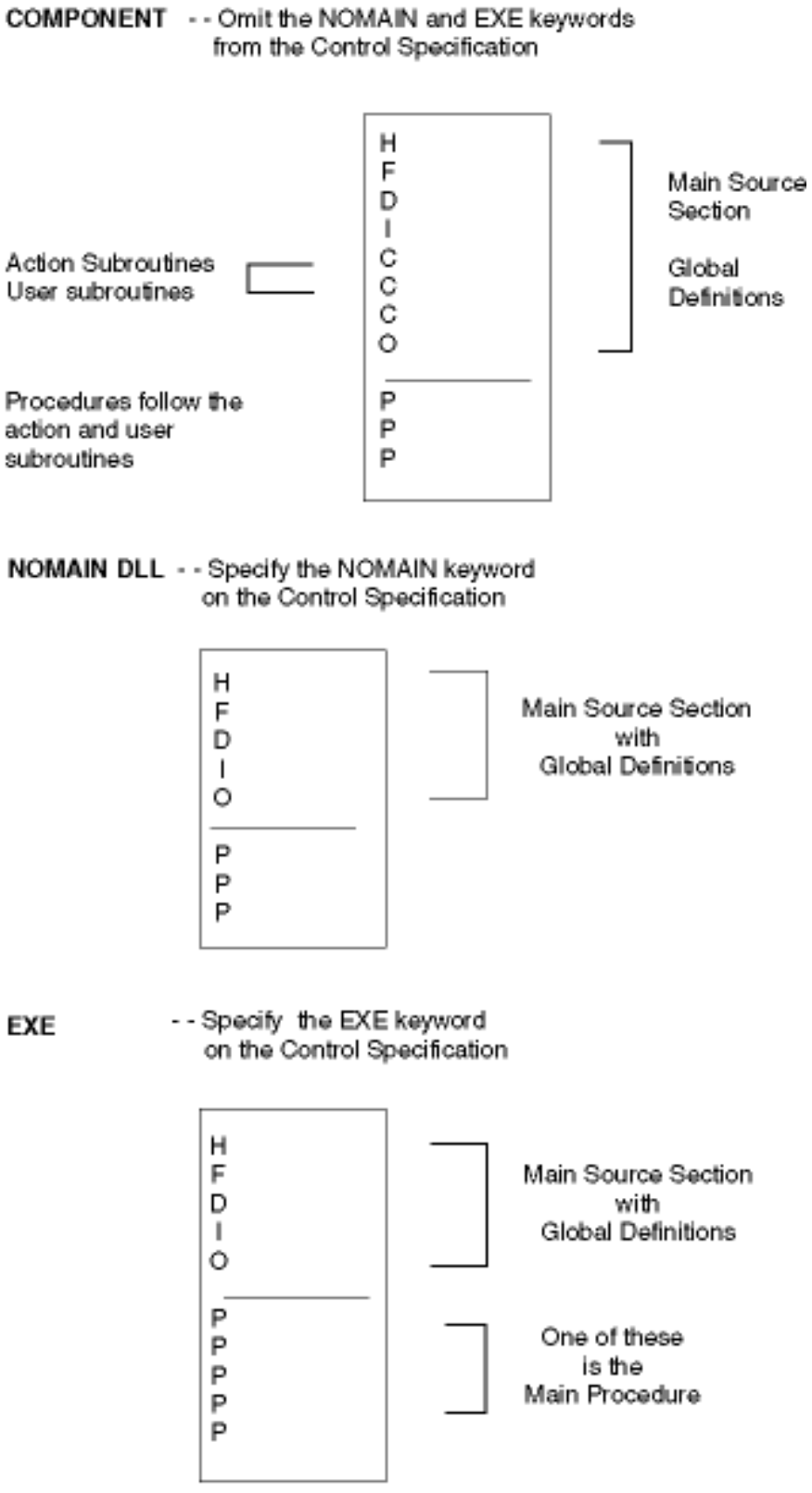


図 82. 各コンパイル・ターゲットごとのメイン・ソース・セクション

一般に、メイン・ソース・セクションに定義されたすべての項目は、グローバルであり、したがって、モジュール全体で通用します。**グローバル定義**とは、メイン・

プロシージャー内のステートメントとモジュール内のすべてのサブプロシージャーの両方によって使用することができる定義です。

他方、サブプロシージャー内の定義はローカルです。**ローカル定義**とは、サブプロシージャーの内部でのみ通用する定義です。グローバル項目と同じ名前の項目が定義された場合に、サブプロシージャーの内部でのその名前に対する参照にはローカル定義が使用されます。

しかし、次の例外に注意してください。
- サブルーチン名およびタグ名は、それらが定義されたプロシージャーにのみ通用します。これには、メイン・プロシージャーで定義されたサブルーチンまたはタグ名が含まれます。
- 入力および出力仕様に指定されているすべてのフィールドはグローバルです。 たとえば、サブプロシージャーがレコード様式を使用して WRITE 演算命令のような演算命令を実行した場合には、そのレコード様式フィールドと同じ名前のローカル定義があっても、グローバル・フィールドが使用されます。この規則は、ウィンドウの READ および WRITE にも適用されます。

場合によっては、グローバル定義とローカル定義が混合されることがあります。たとえば、KLIST および PLIST はグローバルまたはローカルとすることができます。*グローバル* KLISTおよび PLIST と関連したフィールドには、グローバル・フィールドのみが入れられます。 *ローカル* KLIST および PLIST と関連したフィールドには、グローバルとローカルの両方のフィールドを入れることができます。サブプロシージャー内部の KLIST および KFLD が示す動作の詳細については、68 ページの『定義の有効範囲』 を参照してください。

定義のストレージ

ローカル定義には自動ストレージが使用されます。**自動ストレージ**とは、プロシージャーを呼び出している期間だけ存在するストレージです。自動ストレージ内の変数では、値が呼び出しの間にまたがって保管されることはありません。

他方、グローバル定義には静的ストレージが使用されます。**静的ストレージ**とは、プログラムまたはプロシージャーのすべての呼び出しにわたってメモリー内に一定の位置をもつストレージです。その値は、呼び出しの間にまたがって保存されます。

ローカル・フィールド定義に静的ストレージを使用することを指示する場合は、STATIC キーワードを指定します。その場合には、プロシージャーに対する各呼び出し時に値が保存されます。キーワード STATIC が指定された場合には、モジュールの初期化時に項目が初期化されます。

自動ストレージを使用すれば、実行時にプログラムに必要なストレージの量が削減されます。自動ストレージはプロシージャーの実行中にだけ割り振られるので、ストレージが大幅に削減されます。他方、プログラムと関連したすべての静的ストレージは、静的ストレージを使用するプロシージャーが呼び出されない場合であっても、プログラムの始動時に割り振られます。

定義仕様ステートメント

定義仕様の一般レイアウトは、次の通りです。

- 定義仕様タイプ (D) が 6 桁目に入れられます。
- 仕様の非コメント部分は 7 ～ 80 桁目に拡張されます。
  - 固定形式項目は 7 ～ 42 桁目に拡張されます。
  - キーワード記入項目は 44 ～ 80 桁目に拡張されます。
- 仕様のコメント・セクションは 81 桁目から 100 桁目までです

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++Comments++++++++++++
```

図 83. 定義仕様のレイアウト

定義仕様のキーワード継続行

キーワードのための追加のスペースが必要な場合には、次のように、キーワード・フィールドを後続の行に続けることができます :

- 継続行の 6 桁目に D が含まれていなければなりません。
- 継続行の 7 から 43 桁目をブランクにします
- 仕様は 44 桁目以降に継続します

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
D.....................................Keywords+++++++++++++++++++++++++++++++Comments++++++++++++
```

図 84. 定義仕様のキーワード継続行のレイアウト

定義仕様の継続名前行

定義仕様の名前項目には、最高 15 文字までの長さの名前を継続させずに指定することができます。部分名の終わりに省略記号 (...) をコーディングすることによって、任意の名前 (15 文字またはそれ以下のものでも) を複数の行に継続することができます。名前定義は以下のパーツから成り立っています:

1. ゼロまたはそれ以上の継続名前行。継続名前行は、項目内の最後の非ブランク文字として省略記号が含まれるものとして識別されます。名前は、7-21 桁目で始まっている必要がありますが、77 桁目 (省略記号は 80 桁目で終わります) までの任意の位置で終わらせることができます。名前の始めと省略記号文字の間にブランクを入れることはできません。これらの条件のいずれかが適合しない場合には、その行はメイン定義行として解析されます。
2. 名前、定義属性、およびキーワードを含む 1 つのメイン定義行。継続名前行をコーディングする場合には、メイン定義行の名前項目はブランクのままとしておくことができます。
3. ゼロまたはそれ以上のキーワード継続行。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
DContinuedName++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++Comments++++++++++++
```

図 85. 定義仕様の継続名前行のレイアウト

6 桁目 (仕様のタイプ)

定義仕様の場合には、この位置に D を入力する必要があります。

7-21 桁目 (名前)

項目　説明

名前　定義するデータ構造、データ構造サブフィールド、独立型フィールド、名前付き固定情報、ローカル・プログラム、およびローカル・プログラム用のパラメーターの名前。

ブランク

データ構造サブフィールド定義の充てんフィールドまたはデータ構造定義の名前なしデータ構造を指定します。

定義されているデータ項目の名前を指定する場合は、7-21 桁目を使用します。名前は提供されたスペースの中の任意の桁から始まることができます。字下げを使用して、データ構造内のデータの形状を指示することができます。

継続名前行の場合には、名前は、継続名前行の 7-80 桁目およびメイン定義行の 7-21 桁目に指定されます。従来の名前の定義と同様に、文字が大文字か小文字かは重要ではありません。

外部記述サブフィールドの場合には、ここに指定された名前によって、EXTFLD キーワードに指定された外部サブフィールド名が置き換えられます。

プロトタイプ・パラメーター定義の場合には、名前項目は任意指定です。名前を指定した場合には、その名前は無視されます。(プロトタイプ・パラメーターは、24-25 桁目にブランクがあり、その後に PR 仕様または別のプロトタイプ・パラメーター定義が続く定義仕様です。)

プロトタイプを定義していて、7-21 桁目に指定した名前をプロシージャーの外部名として役立てることができない場合には、EXTPROC キーワードを使用して、有効な外部名を指定してください。たとえば、C で書かれたプロシージャーのためのプロトタイプを定義している場合には、外部名が小文字であることが必要とされる場合があります。

22 桁目 (外部記述)

この位置は、データ構造またはデータ構造サブフィールドを外部記述のものとして識別します。データ構造またはサブフィールドをこの仕様で定義しない場合には、このフィールドはブランクのままにしておく必要があります。

項目　データ構造の説明

E　データ構造を外部記述のものとして識別します。サブフィールド定義は外部

で作成されます。 EXTNAME キーワードが指定されていない場合には、7-21 桁目に、データ構造定義を含む外部記述ファイルの名前が入っていなければなりません。

ブランク
プログラム記述: この仕様の後に、このデータ構造のプログラム記述サブフィールド定義が続きます。

項目　サブフィールドの説明

E　データ構造サブフィールドを外部記述のものとして識別します。外部記述サブフィールドの仕様は、EXTFLD および INZ などのキーワードが必要とされる場合にのみ必要です。

ブランク
プログラム記述: この仕様行には、データ構造サブフィールドが定義されます。

23 桁目 (データ構造のタイプ)

この項目は、定義されているデータ構造のタイプを識別するために使用されます。データ構造を定義していない場合には、この項目はブランクのままにしておく必要があります。

項目　説明

ブランク
定義されているデータ構造は、プログラム状況またはデータ域データ構造ではありません。あるいはこの仕様ではデータ構造は定義されていません。

S　プログラム状況データ構造。プログラム状況データ構造として指定できるのは、1 つのデータ構造だけです。

U　データ域データ構造。データ域は、初期化の時点で検索され、プログラムの終わりに再書き込みされます。
- DTAARA キーワードが指定された場合には、その DTAARA キーワードに対するパラメーターが外部データ域の名前として使用されます。
- DTAARA キーワードが指定されない場合には、7-21 桁目の名前が外部データ域の名前として使用されます。

24-25 桁目 (定義のタイプ)

項目	説明
ブランク	この仕様は、プロトタイプまたはプロシージャー・インターフェース定義内のデータ構造サブフィールドまたはパラメーターを定義します。
C (24 桁目)	この仕様は固定情報を定義します。25 桁目はブランクでなければなりません。
DS	この仕様はデータ構造を定義します。
M (24 桁目)	この仕様は、DSPLY 命令コードで使用するための「メッセージ」ウィンドウを定義します。25 桁目はブランクでなければなりません。
PI	この仕様は、プロシージャー・インターフェース、および戻り値 (あった場合) を定義します。

PR
この仕様は、ローカル EXE、CMD、または BAT ファイルに対する呼び出しのプロトタイプを定義します。PR 仕様の後に、プログラムに必要なパラメーターの数とタイプを示す、ゼロ以上のパラメーター定義 (24-25 桁目はブランク) が続きます。プロトタイプ定義は、24-25 桁目がブランクでない最初の定義仕様、または定義仕様でない最初の仕様で終わります。

S (24 桁目)
この仕様は、独立型フィールド、配列、またはテーブルを定義します。独立型フィールドによって、データ構造の定義を必要とせずに、個々の作業ファイルを定義することができます。独立型フィールドの場合には、次のことが許されています。

- 独立型フィールドには明示可能な内部データ・タイプが含まれます。
- 独立型フィールドは配列、テーブル、またはフィールドとして定義することができます。
- 長さ表記だけを使用することができます。

データ構造、プロトタイプ、およびプロシージャー・インターフェースの定義は、24-25 桁目がブランクでない最初の定義仕様、または定義仕様でない最初の仕様で終わります。

名前付き固定情報および独立型フィールドの定義仕様は、データ構造およびそのサブフィールドのための定義仕様に組み込まなくてもかまいません。

定義のタイプに応じてグループ化される有効なキーワードのリストについては、295 ページの『定義仕様タイプに従った要約』を参照してください。

26-32 桁目 (始め位置)

26 ～ 32 桁目には、データ構造内のサブフィールドの位置が定義されている場合にだけ、項目を入れることができます。

項目　説明

ブランク
ブランクの「始め」位置は、「終わり/長さ」フィールドの値がサブフィールドの長さを指定すること、あるいはこの仕様行ではサブフィールドは定義されていないことを指示します。

nnnnnnn
データ構造内のサブフィールドの絶対開始桁。指定する値は、名前付きデータ構造の場合は 1 ～ 65535 (名前なしデータ構造の場合は 1 ～ 9999999) で、これらの桁の中で右寄せにする必要があります。

予約語　「始め - 終わり / 長さ」フィールド (26 ～ 39 桁目) では、プログラム状況データ構造またはファイル情報データ構造のための (左寄せした) 予約語が許されています。これらの特殊な予約語は、データ構造内のサブフィールドの位置を定義します。プログラム状況データ構造の予約語は、*STATUS、 *PROC、*PARM、および *ROUTINE です。ファイル情報データ構造 (INFDS) の予約語は、*FILE、*RECORD、*OPCODE、 *STATUS、および *ROUTINE です。

33 ～ 39 桁目 (終わり位置/長さ)

項目　説明

ブランク

33 ～ 39 桁目がブランクの場合:

- 名前付き固定情報がこの仕様行で定義されているか、または
- 独立型フィールドまたはサブフィールドが別のフィールドに似せて定義されているか、または
- 独立型フィールドまたはサブフィールドが長さの暗黙指定されるタイプであるか、または
- サブフィールドの属性が別の場所で定義されるか、または
- データ構造が定義されています。データ構造の長さは、サブフィールド「終わり位置」の最大値となります。

nnnnnnn

名前付きデータ構造の場合には、33 ～ 39 桁目に、次のようにして 1 から 65535 (名前なしデータ構造の場合は 1 から 9999999) の数値を (右寄せして) 入れることができます。

- 「始め」フィールド (26 ～ 32 桁目) に数値が入っている場合には、このフィールドの数値がデータ構造内のサブフィールドの絶対終了桁を指定します。
- 「始め」フィールドがブランクである場合には、このフィールドの数値は次のものを指定します。
  - データ構造全体の長さ、または
  - 独立型フィールドの長さ、または
  - パラメーターの長さ、または
  - サブフィールドの長さ。

    データ構造内では、このサブフィールドは、その開始桁がデータ構造で前に定義されたすべてのサブフィールドの最大終わり位置より大きくなるように位置決めされます。サブフィールドが基底ポインターまたはプロシージャー・ポインターのタイプによって定義された場合には、そのサブフィールドが正しく確実に位置合わせされるように、埋め込みが挿入されます。

注: グラフィックまたは UCS-2 フィールドの場合には、ここに指定された数値がグラフィックまたは UCS-2 文字の数となり、バイト数 (1 グラフィックまたは UCS-2 文字 = 2 バイト) とはなりません。数値フィールドの場合には、ここに指定された数値が桁数となります (パックおよびゾーン数値フィールドの場合: 1-30; 2 進数値フィールドの場合: 1-9; 整数および符号なし数値フィールドの場合: 3、5、10、または 20)。

+I-nnnnn

この項目は、LIKE キーワードを使用して定義される独立型フィールドまたはサブフィールドに有効です。この仕様行に定義される独立型フィールドまたはサブフィールドの長さは、これらの位置に入力された値を LIKE キーワードに対するパラメーターとして指定されたフィールドの長さに加算したり、あるいはその長さから減算して決定されます。

注: グラフィックまたは UCS-2 フィールドの場合には、ここに指定された数値がグラフィックまたは UCS-2 文字の数となり、バイト数 (1 グラフィックまたは UCS-2 文字 = 2 バイト) とはなりません。数値フィールドの場合には、ここに指定された数値が桁数となります。

予約語 26 ～ 32 桁目が特殊な予約語を入力するために使用される場合には、このフィールドが直前のフィールドの拡張部分となって、1 つの大きなフィールド (26 ～ 39 桁目) が作成されます。これによって、名前が 7 文字より長い予約語をこのフィールドに拡張させることができます。 262 ページの『26-32 桁目 (始め位置)』を参照してください。

40 桁目 (内部データ・タイプ)

この項目によって、独立型フィールドまたはデータ構造サブフィールドが内部的にどのように保管されるかを指定することができます。この項目は、データ項目が外部でどのように保管されたかとは無関係に (すなわち、それが外部で保管されたとしても)、定義されているデータ項目の内部表現と厳密に関係します。 可変長文字、グラフィック、または UCS-2 形式を定義する場合は、キーワード VARYING を指定しなければなりません。さもないと、形式は固定長となります。

項目　説明

ブランク

LIKE キーワードが指定されない場合: 小数点以下の桁数の項目がブランクであった場合には、項目は文字として定義されます。小数点以下の桁数がブランクでなく、項目が独立型フィールドであった場合はパック数値として定義され、項目がサブフィールドであった場合はゾーン数値として定義されます。

注: LIKE キーワードを指定した場合には、この項目はブランクでなければなりません。

A 文字(固定または可変長形式)

N 文字 (標識形式)

C UCS-2 (固定または可変長形式)

G グラフィック (固定または可変長形式)

T 時刻

D 日付

Z タイム・スタンプ

O オブジェクト

P 数値 (パック 10 進数形式)

B 数値 (2 進数形式)

I 数値 (整数形式)

S 数値 (ゾーン形式)

U 数値 (符号なし形式)

F 数値 (浮動形式)

O オブジェクト (Java™ アプリケーションの場合のみ)

*　　基底ポインターまたはプロシージャー・ポインター

41-42 桁目 (小数点以下の桁数)

41 ～ 42 桁目は、数値サブフィールドまたは独立型フィールド内の小数点以下の桁数の数値を指示するために使用されます。このフィールドが数値の場合には、常にこれらの桁数が入力されます。小数点以下の桁数がない場合には、0 を入力してください。

項目　　説明

ブランク
　　値は数値でないか、LIKE キーワードを使用して定義されたものです。

0 ～ 30
　　小数点以下の桁数: 数値フィールドの小数点の右側の桁数。

この項目は、「終わり/長さ」フィールドと組み合わせてのみ指定することができます。「終わり/長さ」フィールドがブランクである場合には、この項目の値はこのプログラム以外のどこかで (たとえば、外部記述データベース・ファイルを通じて) 定義されます。

43 桁目 (予約済み)

43 桁目はブランクでなければなりません。

44 ～ 80 桁目 (キーワード)

44 ～ 80 桁目は、定義仕様のキーワードのために提供されています。キーワードは、データとその属性を記述および定義するために使用されます。各キーワードの説明については、『定義仕様のキーワード』 を参照してください。

このエリアは、フィールドを完全に定義するために必要なすべてのキーワードを指定するために使用してください。

定義仕様のキーワード

定義仕様のキーワードには、パラメーターをもたせなかったり、任意指定パラメーターまたは必須パラメーターをもたせることができます。キーワードの構文は以下のとおりです。

```
Keyword(parameter1 : parameter2)
```

ここで:
- パラメーター (1 つまたは複数) は括弧 ( ) で囲みます。

  注: パラメーターがない場合には括弧を指定しないでください。
- コロン (:) は複数のパラメーターを区切るのに使用します。

次の国際規則は、任意指定のパラメーターと必須のパラメーターを示すために使用されます。
- 中括弧 { } は、オプション・パラメーターまたはパラメーターのオプション・エレメントを示します。
- 省略記号 (...) は、パラメーターを繰り返し使用できることを示します。

- コロン (:) はパラメーター間を区切り、複数のパラメーターを指定できることを示します。 コロンで区切られたすべてのパラメーターは、それらが中括弧で囲まれていない限り必須です。
- 縦線 (|) は、キーワードに 1 つのパラメーターしか指定できないことを示します。
- キーワード・パラメーターを区切るブランクは、1 つまたは複数のパラメーターを指定できることを示します。

注: 中括弧、省略記号、および縦線はキーワード構文の一部ではなく、ソースの中に入力してはいけません。

キーワードに追加のスペースが必要である場合には、キーワード・フィールドは次行に続けることができます。 259 ページの『定義仕様のキーワード継続行』 および 219 ページの『定義仕様キーワード・フィールド』 を参照してください。

ALIGN

ALIGN キーワードは、浮動、整数、符号なしサブフィールドを位置合わせするために使用されます。 ALIGN を指定した場合には、2 バイトのサブフィールドは 2 バイトの境界で位置合わせされ、4 バイトのサブフィールドは 4 バイトの境界で位置合わせされ、また、8 バイトのサブフィールドは 8 バイトの境界で位置合わせされます。浮動、整数、または符号なしサブフィールドにアクセスする場合には、パフォーマンスを向上させるために位置合わせの必要なことがあります。

ALIGN はデータ構造定義で指定します。しかし、ファイル情報データ構造 (INFDS) またはプログラム状況データ構造 (PSDS) については、どちらにも ALIGN を指定することはできません。

位置合わせは、長さ表記を指定し、キーワード OVERLAY は指定しないで定義されたデータ構造サブフィールドに対してのみ行なわれます。絶対表記によるかまたは OVERLAY キーワードを使用して定義されたサブフィールドが適切に位置合わせされなかった場合には、診断メッセージが出されます。

ポインター・サブフィールドは、ALIGN の指定の有無にかかわらず、常に 4 バイトの境界で位置合わせされます。

詳しくは、 167 ページの『データ構造サブフィールドの位置合わせ』を参照してください。

ALT(array_name)

ALT キーワードは、コンパイル時配列、事前実行時配列、またはテーブルが交互形式にあることを指示します。

ALT キーワードを指定して定義された配列は、交互配列であり、パラメーターとして指定された配列名がメイン配列となります。代替配列定義は、メイン配列定義の前または後に入れることができます。

メイン配列上のキーワードは、その両方の配列のロードを定義します。初期化データは、次のように、メイン配列で始まる交互順序です: メイン/代替/メイン/代替/...

代替配列定義では、PERRCD、FROMFILE、TOFILE、および CTDATA キーワードは無効です。

ASCEND

ASCEND キーワードは、事前実行時またはコンパイル時にロードされる配列またはテーブル内のデータの順序を記述します。 270 ページの『DESCEND』を参照してください。

昇順は、配列またはテーブルの項目が (デフォルトの ASCII 照合に従って) 最低のデータ項目から始まり、最高のデータ項目へ進まなければならないことを意味します。等しい値をもつ項目が許されています。

事前実行時配列またはテーブルについては、配列またはテーブルにデータがロードされる時点で指定された順序であるかどうかが検査されます。配列またはテーブルの順序が間違っていた場合には、例外/エラー処理ルーチンに制御が渡されます。(入力または演算、あるいはその両方の仕様によってロードされる)実行時配列の順序は検査されません。

項目が検索引き数と比較して高いかまたは低いかを判別するために、LOOKUP 演算命令、%LOOKUPxx 組み込み関数、または %TLOOKUPxx 組み込み関数を使用して配列またはテーブルの項目を検索する場合には、順序 (昇順または降順) を指定する必要があります。

SORTA 命令コードが配列と一緒に使用され、順序が指定されなかった場合には、昇順と見なされます。

BASED(basing_pointer_name)

データ構造または独立型フィールドに BASED キーワードが指定された場合には、キーワード・パラメーターとして指定された名前を使用して、基底ポインターが作成されます。この基底ポインターでは、定義されている基底付きデータ構造または独立型フィールドのアドレス (記憶場所) が保持されます。言いかえると、7 - 21 桁目に指定された名前が、基底ポインターに含まれている場所で保管されているデータを参照するために使用されます。

注: 基底付きデータ構造または独立型フィールドを使用する前に、基底ポインターに有効なアドレスを割り当てておく必要があります。

配列を基底付き独立型フィールドとして定義する場合には、それが実行時配列でなければなりません。

基底付きフィールドがサブプロシージャーの中で定義される場合には、フィールドと基底ポインターの両方がローカルでなければなりません。

BUTTON(button1:button2....)

BUTTON キーワードは、DSPLY 命令コードの演算項目 2 で指定された「メッセージ」ウィンドウ上のボタンを定義します。1 つのキーワードにつき最大 3 つのボタン・パラメーターを指定することができます。 有効なボタンの組み合わせは次の通りです。

OK(*O)	*OK: *CANCEL	*RETRY: *CANCEL
*YESBUTTON: *NOBUTTON	*RETRY: *ABORT: *IGNORE	*YESBUTTON: *NOBUTTON: *CANCEL

MSGDATA、MSGNBR、または MSGTEXT キーワードを使用した場合には、このキーワードは使用できません。

CCSID(number | *DFT)

このキーワードは、グラフィックおよび UCS-2 定義の CCSID を設定します。

number は、0 から 65536 の範囲の整数でなければなりません。 これは、有効なグラフィックまたは UCS-2 CCSID 値であることが必要です。有効な UCS-2 CCSID は 13488 および 17584です。

プログラム記述フィールドの場合の CCSID(number) は、CCSID(*GRAPH: *SRC)、CCSID(*GRAPH: number)、または CCSID(*UCS2: number) キーワードによる制御仕様上のデフォルトのセットを上書きします。

CCSID(*DFT) は、モジュールにデフォルト CCSID を使用することを指示します。これは、それがなければ新規フィールドがソース・フィールドの CCSID を継承するので、LIKE キーワードが使用される場合に有用です。

このキーワードが指定されない場合には、モジュールのデフォルト・グラフィックまたは UCS-2 CCSID と見なされます。 (CCSID(*GRAPH : *IGNORE) が指定されるか、またはそのように見なされた場合には、このキーワードをグラフィック・フィールドに使用することはできません。)

このキーワードを指定しないで LIKE キーワードを指定した場合には、新規フィールドは LIKE フィールドと同じ CCSID をもつことになります。

CLASS(*JAVA:class_name)

このキーワードは、オブジェクト定義のクラスを示します。

オブジェクトを保管できるフィールドを宣言するには、D 仕様の 40 桁目に O を指定し、CLASS キーワードを使用して、そのオブジェクトのクラスを示してください。CLASS キーワードには次のように 2 つのパラメーターが必要です。

```
CLASS(*JAVA:class_name)
```

*JAVA は、オブジェクトを Java オブジェクトとして識別します。class_name は、オブジェクトのクラスを指定します。クラス名は、文字リテラルであり、Java クラスを完全に修飾するものでなければなりません。クラス名は大文字・小文字が区別されます。

タイプ O のフィールドは、データ構造のサブフィールドとして定義することはできません。タイプ O のフィールドからなる配列を使用することはできますが、タイプ O のテーブルは、実行時に事前ロードする必要があるので、使用することはできません。

次のキーワードは、CLASS キーワードとともに使用することはできません。

```
ALIGN, ALT, ASCEND, BASED, BUTTON, CLTPGM, CONST, CTDATA, DATFMT,
DESCEND, DTAARA, EXTFLD, EXTFMT, EXTNAME, FROMFILE, INZ, LINKAGE,
MSGDATA, MSGNBR, MSGTEXT, MSGTITLE, NOOPT, NOWAIT, OCCURS, OPTIONS,
OVERLAY, PACKEVEN, PERRCD, PREFIX, PROCPTR, STYLE, TIMFMT, TOFILE,
VALUE, VARYING
```

java メソッドの呼び出しおよびその例については、*VisualAge for RPG プログラミング* という資料を参照してください。

CLTPGM(program name)

CLTPGM キーワードは、CALLP 演算命令を使用して、VisualAge RPG プログラムによって呼び出されるローカル・プログラムの名前を指定するために使用されます。

呼び出されるプログラムは、EXE、PIF、COM、または BAT ファイルとすることができます。

デフォルト拡張子は EXE です。

注: 各パラメーターごとに定義仕様をコーディングすることが必要です。

CONST(constant)

CONST キーワードは、名前付き固定情報の値を指定するために使用されます。このキーワードは、任意指定 (固定情報値は、CONST キーワードがあってもなくても指定することができます) であり、名前付き固定情報定義 (24 桁目に C) の場合にのみ有効です。

このパラメーターは、リテラル、表意定数、または組み込み関数でなければなりません。固定情報は、適切な継続規則に従うことによって後続行に継続させることができます。 216 ページの『継続規則』を参照してください。

名前付き固定情報をキーワード DIM、OCCURS、PERRCD、または OVERLAY のパラメーターとして使用する場合には、その使用に先立って定義しておく必要があります。

読み取り専用参照パラメーターを指定している場合には、プロトタイプとプロシージャー・インターフェースの両方のパラメーター定義の定義仕様にキーワード CONSTを指定します。キーワードに対するパラメーターは許されません。

キーワード CONST が指定された場合には、コンパイラーがパラメーターを一時的な場所にコピーし、その一時的な場所のアドレスを渡すことがあります。この原因となる条件の一部として、次のものがあります : 渡されるパラメーターが式であるか、あるいは渡されるパラメーターに異なる形式が含まれている。

パラメーターが呼び出し先プログラムまたはプロシージャーによって変更されないことが確実でないかぎりは、このキーワードをプロトタイプ定義では使用しないようにしてください。

呼び出し先プログラムまたはプロシージャーが同じプロトタイプでプロシージャー・インターフェースを使用してコンパイルされる場合には、コンパイラーがこの点を検査することになるので、このような心配はありません。

固定情報値によってパラメーターを渡すことには、値によって渡すのと同じ利点があります。とくに、リテラルおよび式を渡すことが可能です。

CTDATA

CTDATA キーワードは、配列またはテーブルがコンパイル時データを使用してロードされることを指示します。データは、** または **CTDATA(array/table name) 仕様の後のプログラムの終わりに指定されます。

配列またはテーブルがコンパイル時にロードされた場合には、ソース・プログラムとともにコンパイルされて、プログラムに組み込まれます。このような配列またはテーブルは、プログラムが実行されるたびに別個にロードする必要はありません。

DATFMT(format{separator})

DATFMT キーワードは、「日付」フィールドの内部日付形式と、任意指定で区切り文字を指定します。このキーワードは、日付タイプの外部記述データ構造サブフィールドの場合には自動的に生成され、コンパイル時に判別されます。

このキーワードは、CALLP パラメーターの定義時に使用することができます。

227 ページの『DATFMT(fmt{区切り文字})』を参照してください。

日付配列またはフィールドの内部形式および区切り文字を判別する時に使用される階層は、次の通りです。

1. 定義仕様に指定された DATFMT キーワードから
2. 制御仕様に指定された DATFMT キーワードから
3. *ISO

DESCEND

DESCEND キーワードは、事前実行時またはコンパイル時にロードされる配列またはテーブル内のデータの順序を記述します。 267 ページの『ASCEND』を参照してください。

降順は、配列またはテーブルの項目が (照合順序に従って) 最高のデータ項目から始まり、最低のデータ項目へ進まなければならないことを意味します。等しい値をもつ項目が許されています。

事前実行時配列またはテーブルについては、配列またはテーブルにデータがロードされる時点で指定された順序であるかどうかが検査されます。配列またはテーブルの順序が間違っていた場合には、例外/エラー処理ルーチンに制御が渡されます。(入力または演算、あるいはその両方の仕様によってロードされる)実行時配列の順序は検査されません。

項目が検索引き数と比較して高いかまたは低いかを判別するために、LOOKUP 演算命令、%LOOKUPxx 組み込み関数、または %TLOOKUPxx 組み込み関数を使用して配列またはテーブルの項目を検索する場合には、順序 (昇順または降順) を指定する必要があります。

SORTA 命令コードが配列と一緒に使用され、順序が指定されなかった場合には、昇順と見なされます。

DIM(numeric_constant)

DIM キーワードは、配列またはテーブル内のエレメントの数を定義します。

数値固定情報の小数点以下の桁数はゼロ (0) でなければなりません。これは、リテラル、名前付き固定情報、または組み込み関数とすることができます。

キーワードの処理時には、固定情報値が通用するものである必要はありませんが、コンパイル時にはその値が通用するものでなければなりません。

このキーワードは、CALLP パラメーターの定義時に使用することができます。

DLL(name)

DLL キーワードは、LINKAGE キーワードと一緒に、Windows API を含む Windows® DLL の機能を呼び出すプロシージャーのプロトタイピングに使用されます。

次の例は、プロトタイプおよび Windows API GetCurrentDirectory に対する呼び出しのコーディング方法を示しています。

```
D GetCurDir       PR            10I 0 ExtProc('GetCurrentDirectoryA')

D                                     DLL('KERNEL32.DLL')
D                                     Linkage(*StdCall)
D                                10I 0 Value
D                                255A
D CurDir          S              255A
D CurDirSiz       S               10I 0 Inz(%Size(CurDir))
D RCLong          S               10I 0

C                   Eval      RCLong = GetCurDir(CurDirSiz:CurDir)
```

外部プロシージャー名 (GetCurrentDirectory**A**) の中の *A* は、DLL の 1 バイト・バージョンが呼び出されることを指示します。ユニコード・バージョンを呼び出す場合は、*W* を指定してください。

DTAARA{(data_area_name)}

DTAARA キーワードは、独立型フィールド、データ構造、データ構造サブフィールド、またはデータ域データ構造を外部データ域と関連づけるために使用されます。DTAARA キーワードには、*DTAARA DEFINE命令コードと同じ機能があります。530 ページの『データ域としてのフィールドの定義』を参照してください。

data_area_name が指定されない場合には、7-21 桁目に指定された名前もデータ域名となります。 data_area_name を指定する場合には、それが 1 つのデータ域名でなければなりません。

パラメーターを指定しない場合には、データ構造名の指定が必要です。

DTAARA キーワードを指定した場合には、データ域で IN、OUT、および UNLOCK 命令コードを使用することができます。

EXTFLD(field_name)

EXTFLD キーワードは、外部記述データ構造内のサブフィールドを名前変更するために使用されます。 field_name パラメーターは、サブフィールドの外部名です。使用するプログラムの名前は、「名前」フィールド (7-21 桁目) に指定されます。

キーワードは任意指定です。指定しない場合には、外部定義から抽出された名前がデータ構造サブフィールド名として使用されます。

PREFIX キーワードがデータ構造に指定された場合に、EXTFLD によって名前変更されたフィールドには接頭部は適用されません。

EXTFMT(code)

EXTFMT キーワードは、コンパイル時および事前実行時数値配列およびテーブルのための外部データ・タイプを指定します。外部データ・タイプは、ファイルのレコード内のデータの形式です。この項目は、プログラム内の配列またはテーブルの内部処理に使用される形式 (内部データ・タイプ) には影響しません。

パラメーターに可能な値は、次の通りです。

S 配列またはテーブルのデータはゾーン 10 進数形式です。

P 配列またはテーブルのデータはパック 10 進数形式です。

B 配列またはテーブルのデータは 2 進数形式です。

C 配列またはテーブルのデータは UCS-2 形式です。

I 配列またはテーブルのデータは整数形式です。

L 数値配列またはテーブル・エレメントのデータの前 (左側) に正または負符号が入れられます。

R 数値配列またはテーブル・エレメントのデータの後 (右側) に正または負符号が入れられます。

U 配列またはテーブルのデータは符号なし形式です。

F 配列またはテーブルのデータは浮動数値形式です。

注:

1. EXTFMT キーワードが指定されない場合には、外部形式はデフォルトによって、非浮動配列およびテーブルの場合は 'S' となり、浮動事前実行時配列およびテーブルの場合は外部表示浮動表現となります。
2. コンパイル時配列およびテーブルで、データ・タイプが浮動でない場合に許される値は、S、L、および Rだけであり、その場合には EXTFMT キーワードは使用できません。

3. 10 桁以上として定義された配列に EXTFMT(I) または EXTFMT(U) は使用できません。1 から 5 桁として定義された配列は、2 バイトを占めます。6 から 10 桁として定義された配列は、4 バイトを占めます。
4. EXTFMT(I) または EXTFMT(U) が使用された場合に、1 から 5 桁として定義された配列は、1 つのエレメントにつき 2 バイトを占めます。6 から 10 桁として定義された配列は、1 つのエレメントにつき 4 バイトを占めます。11 から 20 桁として定義された配列は、1 つのエレメントにつき 8 バイトを占めます。
5. UCS-2 配列の場合のデフォルトの外部形式は文字です。UCS-2 コンパイル時データに許されている文字の数は、UCS-2 配列内の 2 バイト文字の数です。
6. 配列またはテーブルのデータがワークステーション上にある場合には、EXTFMT キーワードを使用することはできません。

EXTNAME(file_name{:format_name})

EXTNAME キーワードは、定義されているデータ構造のサブフィールド記述として使用されるフィールド記述が入ったファイルの名前を指定します。

file_name パラメーターは必須です。任意選択で様式名を指定し、ファイル内の特定の様式をコンパイラーに指示することができます。 format_name パラメーターが指定されない場合には、最初のレコード様式が使用されます。

データ構造定義の 22 桁目に E が入っていて、EXTNAME キーワードが指定されない場合には、7-21 桁目に指定された名前が使用されます。

コンパイラーは、外部記述データ構造のすべてのフィールドについて次の定義仕様項目を生成します。

- サブフィールド名: キーワード EXTFLD によって名前変更されていないか、PREFIX キーワードを使用して接頭部が適用されていないかぎり、この名前は外部名と同じです。
- サブフィールドの長さ
- サブフィールドの内部データ・タイプ: この名前は外部タイプと同じですが、それは、CVTOPT制御仕様キーワードがタイプに対して指定されていない場合です。その場合には、データ・タイプは文字となります。

EXTNAME キーワードは、すべてのデータ構造キーワードとともに使用することができます。

EXTPROC({*JAVA:class-name:}name)

EXTPROC キーワード以下の 1 つの形式とすることができます。

EXTPROC(*JAVA:class-name:name)

Java で書かれるメソッド、または Java で呼び出される RPG ネイティブ・メソッドを指定します。最初のパラメーターは *JAVA です。2 番目のパラメーターは、メソッドのクラスを含む文字固定情報です。3 番目のパラメーターは、メソッド名を含む文字固定情報です。特殊なメソッド名の *CONSTRUCTOR は、メソッドがコンストラクターであることを意味します。このメソッドは、クラスのインスタンスを生成 (新規クラス・インスタンスを作成) するために使用できます。

Java プロシージャーの呼び出しの詳細については、*ILE RPG プログラマーの手引き* を参照してください。

EXTPROC(name)

外部プロシージャーを指定します。

EXTPROC キーワードは、そのプロトタイプが定義されているプロシージャーの外部名を指示します。この名前は、文字固定情報またはプロシージャー・ポインターにすることができます。EXTPROC を指定した場合には、CALLB または CALLP を使用してプロシージャーを呼び出すことが必要です。

EXTPROC が指定されない場合は、プロシージャーが定義されて、それに 7 ～ 21 桁目で見つかった外部名が割り当てられるとコンパイラーは見なします。

EXTPROC に指定される名前 (または、EXTPROC が指定されない場合はプロトタイプ名) が ″CEE″ または下線 ('_') で始まっている場合には、コンパイラーは、これをシステム組み込みとして取り扱います。それが実際にシステム組み込みでない場合には、リストに警告が表示されます。システム提供の API との混同を避けるために、ユーザー・プロシージャーの名前は ″CEE″ で始めないようにしてください。

プロシージャー・ポインターを指定した場合には、それを呼び出しで使用する前に、有効なアドレスを割り当てることが必要です。これは、戻り値およびパラメーターにプロトタイプ定義との整合性があるプロシージャーを示していなければなりません。

たとえば、プログラム QSQCLI の中にあるプロシージャー SQLAllocEnv のためのプロトタイプを定義する場合は、次のような定義仕様をコーディングすることができます。

```
D SQLEnv          PR                  EXTPROC('SQLAllocEnv')
```

275 ページの図 86 は、プロシージャー・ポインターをそのパラメーターとして指定した EXTPROC キーワードの例を示しています。

```
D* EXTPROC としてプロシージャー・ポインターを含む
D* プロシージャーを呼び出すとします。以下は、
D* プロトタイプがどのように定義されるかです。
D*
D DspMsg          PR            10A   EXTPROC(DspMsgPPtr)
D  Msg                       32767A
D  Length                        4B 0 VALUE
D*
D* 以下は、DspMsgPPtr の割り当てが可能なプロシージャー
D* のためのプロトタイプをどのように定義するかです。
D*
D MyDspMsg        PR                  LIKE(DspMsg)
D  Msg                       32767A
D  Length                        4B 0 VALUE
C*
C* DSPMSG を呼び出す前に、DSPMSGPPTR を
C* MyDspMsg の実際のプロシージャー名である
C* MYDSPMSG に割り当てます。
C*
C                   EVAL      DspMsgPPtr = %paddr('MYDSPMSG')
C                   EVAL      Reply = DspMsg(Msg, %size(Msg))
 ...
P MyDspMsg       B
```

図 86. プロトタイプ・プロシージャーの外部名の指定

VARPG Java アプリケーションから呼び出される Java メソッドをプロトタイピングするために、EXTPROC キーワードの拡張仕様を使用することができます。『Java メソッドのプロトタイピング』を参照してください。

Java メソッドのプロトタイピング

Java メソッドは、VARPG Java アプリケーションがそれを正しく呼び出せるようにプロトタイピングしなければなりません。コンパイラーは、メソッドの名前、それが属するクラス、パラメーターのデータ・タイプ、および戻り値があればそのデータ・タイプを知り、メソッドが静的メソッドであるかどうかを知る必要があります。

メソッドの名前およびそれが属するクラスを指定するには、EXTPROC キーワードの拡張仕様を使用します。EXTPROC キーワードの形式は次の通りです。

```
EXTPROC(*JAVA:class_name:method_name | *JAVARPG:class_name:method_name)
```

使用できるパラメーター値は次の通りです。

***JAVA:class_name:method_name**
メソッドを最初に Java で書かれたコードから生成された Java メソッドとして識別します。

***JAVARPG:class_name:method_name**
VARPG 生成 Java メソッドを識別します。

VARPG 生成のメソッドにより、通常では Java での参照で渡すことができない一定のデータ・タイプを参照により渡すことができます。これにより、ターゲットを Windows アプリケーションにする時、あるいは Java ソース・コードを生成する時に同じソース・コードを使用できるようになります。

クラス名およびメソッド名は文字リテラルでなければならず、大文字小文字を区別されます。クラス名は、完全修飾 Java クラス名でなければなりません。メソッド名は、呼び出されるメソッドの名前でなければなりません。

パラメーターのデータ・タイプおよびメソッドの戻り値のデータ・タイプは、サブプロシージャーをプロトタイピングする時と同様に指定します。しかし、コンパイラーは次のように VARPG データ・タイプを Java データ・タイプにマップするということに注意してください。

Java データ・タイプ	**VARPG データ・タイプ**
boolean	標識 (N)
byte[]	英字 (任意の長さの A)
byte	整数 (3I)
int	整数 (10I)
short	整数 (5I)
long	整数 (20I)
float	浮動 (4F)
double	浮動 (8F)
任意のオブジェクト	オブジェクト (O)

VARPG 生成メソッド (*JAVARPG) を呼び出す時には、パラメーターおよび戻り値のデータ・タイプとしてパック、ゾーン、2 進数、または符号なしのデータ・タイプを指定することができます。Java ソース・コードから生成されたメソッドは、パラメーターまたは戻り値のプロトタイプでこうしたデータ・タイプを使用することはできません。

メソッドを呼び出す時には、コンパイラーは、パラメーターが DIM キーワードを使用してプロトタイピングされていれば配列をパラメーターとして受け入れます。そうでない場合にはスカラー・フィールド、データ構造、またはテーブルだけが受け入れられます。

Java の **char** のデータ・タイプを必要とするか、あるいはこの値タイプを戻すメソッドを呼び出すことはできません。

メソッドの戻り値がオブジェクトである場合には、プロトタイプに CLASS キーワードを指定することによってそのオブジェクトのクラスを指定してください。指定されたクラス名は、戻されるオブジェクトのクラス名となります。

呼び出されるメソッドが静的メソッドである場合には、プロトタイプに STATIC キーワードを指定してください。

Java では、値によって次のデータ・タイプだけを渡すことができます。

```
   byteintshortlong   float
double
```

こうしたデータ・タイプのパラメーターの場合には、プロトタイプに VALUE キーワードを指定してください。こうしたデータ・タイプは参照によって渡すことができるので、VARPG 生成のメソッドを呼び出す時には、VALUE キーワードは必要ありません。

オブジェクトは、参照によってのみ渡すことができます。タイプ 'O' では、VALUE キーワードを指定することはできません。配列は Java によってオブジェクトとして見られるので、配列へのパラメーター・マッピングも参照によって渡さなければなりません。これにはバイト配列が含まれます。

java メソッドの呼び出しおよびその例については、*VisualAge for RPG プログラミング* という資料を参照してください。

FROMFILE(file_name)

FROMFILE キーワードは、定義されている事前実行時配列またはテーブルのための入力データを含むファイルを指定します。プログラムで使用されているすべての事前実行時配列またはテーブルについて FROMFILEキーワードを指定する必要があります。

294 ページの『TOFILE(file_name)』を参照してください。

INZ{(initial value)}

INZ キーワードは、独立型フィールド、データ構造、またはデータ構造サブフィールドのデータ・タイプをそのデフォルト値に、あるいは任意選択で括弧内に指定された固定情報に初期化します。

- プログラム記述データ構造の場合、INZ キーワードにパラメーターは使用できません。
- 外部記述データ構造の場合、*EXTDFT パラメーターしか使用できません。
- LIKEDS キーワードで定義されているデータ構造の場合、値 *LIKEDS は、サブフィールドが親データ構造と同じ方法で初期化されることを指定します。
- オブジェクトの場合、*NULL パラメーターしか指定できません。INZ(*NULL) の指定の有無にかかわらず、すべてのオブジェクトは *NULL に初期化されます。

指定する初期値は、初期化しようとするタイプと整合性のあるものでなければなりません。初期値はリテラル、名前付き固定情報、表意定数、組み込み関数とするか、特殊値 *EXTDFT、 *LIKEDS、または *NULL とすることができます。日付または時刻データ・タイプ・フィールド、あるいは日付または時刻値を含む名前付き固定情報を初期化する場合には、日付または時刻フィールドの実際の形式とは無関係に、リテラルの形式が制御仕様から派生した通りのデフォルト形式と整合している必要があります。

数値フィールドは、任意の数値リテラルのタイプを使用して初期化することができます。ただし、浮動リテラルを使用できるのは浮動フィールドだけです。いずれの数値フィールドも、16 桁以下の 16 進数リテラルを使用して初期化することができます。この場合は、16 進数リテラルが符号なし数値と見なされます。

INZ(*EXTDFT) を指定すると、外部記述データ構造サブフィールドが DDS の DFT キーワードからのデフォルト値によって初期化されます。DFT または固定情報値が指定された場合には、フィールド・タイプの DDS デフォルト値が使用されます。サブフィールド仕様でパラメーターを指定するか、あるいは指定しなくても、INZ キーワードを指定すると、DDS に指定された値を変更することができます。

外部データ構造定義で INZ(*EXTDFT) を指定すると、すべての外部記述サブフィールドがその DDS デフォルト値に初期化されます。外部記述データ構造に追加のプログラム記述サブフィールドがある場合には、それらは RPG デフォルト値に初期化されます。

INZ(*EXTDFT) キーワードを使用する時には、次のことに注意してください。
- 日付フィールドまたは時刻フィールドの DDS 値が RPG 内部形式でない場合には、その値はプログラムで有効な内部形式に変換されます。
- 外部記述は物理ファイルになければなりません。
- DDS 中でヌル値使用可能フィールドに *NULL が指定されている場合には、コンパイラーは、そのフィールドの DDS デフォルト値を初期値として使用します。
- 可変長フィールドに DFT('') が指定されている場合には、そのフィールドは、長さ 0 のストリングによって初期化されます。
- CVTOPT オプションが効力をもっている場合には、INZ(*EXTDFT) は使用できません。
- 日付、時刻、またはタイム・スタンプ・フィールドに初期値が指定されていないか、*NULL が指定されている場合には、フィールドの初期値は *LOVAL に設定されます。

INZ キーワードによって定義されたデータ構造、データ構造サブフィールド、または独立型フィールドを *ENTRY PLIST 上でパラメーターとして指定することはできません。

注: INZ パラメーターを指定しない場合には、次のようになります。
- 初期化済みデータ構造の静的独立型フィールドおよびサブフィールドは、そのデフォルトの初期値 (たとえば、文字の場合はブランク、数値の場合は 0) に初期化されます。
- 初期化されていない静的データ構造のサブフィールド (INZ がそのデータ構造の定義仕様に指定されない場合) は、(そのデータ・タイプとは無関係に) ブランクに初期化されます。
- 自動ストレージのフィールドは初期化されません。

このキーワードと BASED の組み合わせは無効です。

LIKE(RPG_name)

LIKE キーワードは、項目を既存の項目に似せて定義するために使用されます。LIKE キーワードを指定した場合には、定義されている項目はパラメーターとして指定された項目の長さとデータ形式を継承します。このキーワードを使用して、独立型フィールド、プロトタイプ、パラメーター、およびデータ構造サブフィールドを定義することができます。LIKE のパラメーターは、独立型フィールド、データ構造、データ構造サブフィールド、プロシージャー・インターフェース定義内のパラメーター、またはプロトタイプ名とすることができます。データ・タイプ項目 (40 桁目) はブランクでなければなりません。

このキーワードは、*LIKE DEFINE 命令コードと類似のものです (529 ページの『別のフィールドに基づくフィールドの定義』 を参照してください)。ただし、定義されたデータが長さだけでなく、データ形式および CCSID も継承する点で *LIKE DEFINE とは異なります。

注: NOOPT、ASCEND、CONST、およびヌル使用可能性などの属性は、LIKE のパラメーターから、定義された項目へは継承されません。継承されるのは、長さ、小数点以下の桁数、および CCSIDだけです。

LIKE のパラメーターがプロトタイプであった場合には、定義されている項目は、そのプロトタイプの戻り値と同じデータ・タイプとなります。戻り値がない場合には、エラー・メッセージが出されます。

このキーワードは、CALLP パラメーターの定義時に使用することができます。 529 ページの『別のフィールドに基づくフィールドの定義』を参照してください。

LIKE を使用して、文字フィールド、グラフィック・フィールド、グラフィック文字、数値フィールド、および配列を定義することができます。以下は、異なるデータ・タイプで LIKE キーワードを使用する場合のいくつかの考慮事項です。

- **文字フィールドの場合には**、「終わり/長さ」項目に指定された数値が増やす (または減らす) 文字の数となります。
- **グラフィックまたは UCS-2 フィールドの場合には**、「終わり/長さ」項目に指定された数値が増やす (または減らす) グラフィックまたは UCS-2 文字の数 (1 グラフィックまたは UCS-2文字 = 2 バイト) となります。
- **数値フィールドの場合には**、「終わり/長さ」項目に指定された数値が増やす (または減らす) 数字の数となります。整数または符号なしフィールドの場合には、調整値が フィールドの結果の桁数が 3、5、10、または 20 になるような値でなければなりません。浮動フィールドの場合には、長さの調整は許されません。
- **日付、時刻、タイム・スタンプ、基底ポインター、またはプロシージャー・ポインター・フィールド**の場合には、「終わり / 長さ」項目 (33 - 39 桁目) はブランクであることが必要です。

LIKE 使用して配列を定義する場合に、配列のディメンションの定義にはこれまで通り DIMキーワードが必要です。しかし、DIM(%elem(array)) を使用して、配列を別の配列に正確に似せて定義することができます。

同じサブフィールドをもつ別のデータ構造と似たデータ構造を定義するには、LIKEDS を使用します。

以下は、LIKE キーワードを使用してデータを定義する例です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
D*
D*  長さを 5 文字増やして別のフィールドに似せたフィールドを定義します。
D*
D  Name           S             20
D  Long_name      S             +5    LIKE(Name)
D*
D*  別のフィールドに似せて、DIM(20) によってデータ構造サブフィールド
D*  配列を定義し、各配列エレメントを値 *ALL'X' によって初期化します。
D*  また、最初のサブフィールドの直後にタイプ・ポインターの別の
D*  サブフィールドを宣言します。ポインターは、4 バイトの長さで
D*  暗黙的に定義されます。
D*
D  Struct         DS
D   Dim20                              LIKE(Name) DIM(20) INZ(*ALL'X')
D   Pointer                       *
```

図 87. 他のフィールドに似せたフィールドの定義

LIKE(object-name)

LIKE キーワードを使用すれば、1 つのオブジェクトが前に定義されたオブジェクトと同じクラスをもつことを指定できます。CLASS キーワードの値しか継承されません。

```
* 変数の MyString と OtherString は、ともに Java String オブジェクトです。
D MyString        S               0   CLASS(*JAVA
D                                        :'java.lang.String')
D OtherString     S                   LIKE(MyString)
 * Proc は、Java String オブジェクトを戻す Java メソッドです。
D Proc            PR                  EXTPROC(*JAVA:'MyClass':'meth')
D                                     LIKE(MyString)
```

図 88. 他のオブジェクトと類似したオブジェクトの定義

注: *LIKE DEFINE 演算命令を使用して、オブジェクトを定義することはできません。 LIKE キーワードを使用しなければなりません。

LIKEDS(data_structure_name)

LIKEDS キーワードは、他のデータ構造と似たデータ構造、プロトタイピングされた戻り値、プロトタイピングされたパラメーターを定義するために使用します。新規構目のサブフィールドは、他のデータ構造のサブフィールドと同じになります。

サブフィールドの名前は、新規のデータ構造名で修飾されます。*subfield* という未修飾のサブフィールド名または *dsname.subfield* という修飾サブフィールド名では、*newdsname.subfield* という名前の新規サブフィールドができます。名前のないサブフィールドは、新規データ構造中でも名前をもちません。

ALIGN キーワードの値は、新規データ構造によって継承されます。
OCCURS、NOOPT、および INZ の各キーワードの値は継承されません。親データ構造と同様にサブフィールドを初期化するには、INZ(*LIKEDS) を指定します。

```
D sysName         DS                  qualified
D   lib                         10A   inz('*LIBL')
D   obj                         10A
D userSpace       DS                  LIKEDS(sysName) INZ(*LIKEDS)
 // 変数 "userSpace" は *LIKEDS で初期化されていたので、最初の 'lib'
 // サブフィールドは *LIBL に初期化されました。2 番目の 'obj'
 // サブフィールドは、演算を使用して設定しなければなりません。
C                   eval      userSpace.obj = 'TEMPSPACE'
```

*図 89. Using INZ(*LIKEDS)*

```
P createSpace     B
D createSpace     PI
D  name                           LIKEDS(sysName)
/free
     if name.lib = *blanks;
          name.lib = '*LIBL';
     endif;
     QUSCRTUS (name : *blanks : 4096 : ' ' : '*USE' : *blanks);
 /end-free
P createSpace     E
```

図 90. サブプロシージャーでのデータ構造パラメーターの使用

LINKAGE(linkage_type)

CALL および START 演算命令で使用するプログラム名を定義する場合には、LINKAGE キーワードによって呼び出し先プログラムの位置が指定されます。CALL 演算命令の場合には、*SERVER パラメーター値をこのキーワードと一緒に使用してください。*SERVER パラメーターは、呼び出しているプログラムが iSeries サーバー上に存在していることを指定します。START 演算命令の場合には、 *CLIENT パラメーター値をこのキーワードと一緒に使用してください。

LINKAGE キーワードは、DLL キーワードと一緒に、DLL の機能を呼び出す時に使用されるリンケージ規約 (インターフェース) を指定します。指定するリンケージ規約は、受け入れられる外部 DLL のそれと一致しなければなりません。Windows システム API は、StdCall リンケージ規約を使用します。そこで、Windows システム API をプロトタイピングする時には、LINKAGE(*STDCALL) を指定してください。

DLL キーワードを使用して VARPG サブプロシージャーを NOMAIN アプリケーション中でプロトタイピングする場合には、LINKAGE キーワードを使用しないでください。コンパイラーは、デフォルトの __cdecl リンケージ規約を使用します。

自身の DLL をプロトタイピングする時には、__stdcall または __cdecl リンケージ規約を使用して作成してください。他のリンケージ規約を使用すると、予期しない結果となったり、実行時エラーとなることがあります。

MSGDATA(msgdata1:msgdata2....)

MSGDATA キーワードは、DSPLY 命令コードの演算項目 1 で使用される置換テキストを ユーザーがプログラムに定義したフィールド名のリストのフォームで定義します。 VisualAge RPGコンパイラーが、各置換変数を定義された対応するフィールドで置き換えます。たとえば、%1 は MSGDATA に定義された最初のフィールドで置き換えられ、%2 は MSGDATA に定義された 2 番目のフィールドで置き換えられ、以下同様です。 置換変数は、単一の数字 (1-9) が続くパーセント (%) 文字を入力することによって定義されます。1 つのキーワードにつき最大 3 つのパラメーターを指定することができます。

MSGDATA および MSGNBR キーワードは一緒に使用されます。

MSGNBR(*MSGnnnn または fieldname)

MSGNBR キーワードは、DSPLY 命令コードの演算項目 1 で使用されるメッセージ番号を定義します。メッセージ番号は、最大で 4 桁の長さとすることができます。次の 1 つを指定する必要があります。

- メッセージ ID (たとえば、*MSG0001)
- メッセージ番号 (たとえば *MSG0001) を含むフィールド

メッセージに置換テキストが含まれる場合には、MSGNBR および MSGDATA キーワードを一緒に使用してください。

MSGTEXT('message text')

MSGTEXT キーワードは、単一引用符 (') の中に入れられたメッセージ・テキストを定義します。このテキストは、DSPLY 命令コードの演算項目 1 で使用されます。 BUTTON、MSGDATA、MSGNBR、 MSGTITLE、または STYLE キーワードを使用した場合には、このキーワードは使用できません。

MSGTITLE('title text')

MSGTITLE キーワードは、「メッセージ」ウィンドウ (DSPLY 命令コードの演算項目 2) 用のタイトル・テキストを指定します。たとえば、'*MSG0001' のように単一引用符 (') で囲んだ 8 文字のメッセージ ID を入力するか、あるいは 4 桁のメッセージ番号を入力することができます。 メッセージ番号を使用した場合には、テキストがメッセージ・ファイルから検索されます。(メッセージ形式のタイトルを指定する場合は、GUI Designer のメッセージの定義オプションを使用してください。)

MSGDATA、MSGNBR、または MSGTEXT キーワードを使用した場合には、このキーワードは使用できません。

NOOPT

このキーワードが指定された独立型フィールド、パラメーター、またはデータ構造については、最適化は実行されません。これによって、データ項目の内容は最新の割り当て値であることが保証されます。これは、値が例外処理に使用されるフィールドに必要とされることがあります。

注: 最適化プログラムは、通常のプログラム実行中に、一部の値をレジスターの中に保存し、それらを事前定義された点でのみストレージに復元することがあり

ます。例外処理によってこの通常の実行順序が中断され、その結果、レジスターに含まれているプログラム変数がその割り当て済み記憶場所に戻されないことがあります。結果として、それらの変数が例外処理で使用された場合には、そこに最新の割り当て値が含まれない場合があります。NOOPT キーワードは、それらの最新性を保証するものです。

参照によって渡されるデータ項目を NOOPT キーワードとともに定義した場合には、すべてのプロトタイプまたはプロシージャー・インターフェース・パラメーター定義にも NOOPT キーワードを指定する必要があります。この要件は、値によって渡されるパラメーターには適用されません。

独立型フィールドの定義、パラメーター、またはデータ構造の定義に使用できるすべてのキーワードを NOOPT キーワードと一緒に使用することができます。

このキーワードは、CALLP パラメーターの定義時に使用することができます。

NOWAIT

NOWAIT キーワードによって、ワークステーション・ファイルを使用する OS/400 プログラムを呼び出すことができます。詳細については、499 ページの『ワークステーション・ファイルを使用する OS/400 プログラムの呼び出し』 を参照してください。

OCCURS(numeric_constant)

OCCURS キーワードは、複数オカレンス・データ構造のオカレンスの数を指定します。

numeric_constant パラメーターは、小数点以下の桁数のない 0 より大きい値であることが必要です。これは、数値リテラル、数値を戻す組み込み関数、または数値固定情報とすることができます。

キーワードの処理時には、固定情報値が通用するものである必要はありませんが、コンパイル時にはその値が通用するものでなければなりません。

このキーワードは、プログラム状況データ構造、ファイル情報データ構造、またはデータ域データ構造には無効です。

複数オカレンス・データ構造にポインター・サブフィールドが含まれている場合には、ポインターに対するシステム・ストレージの制限のために、オカレンス相互間の距離は正確に 4 の倍数でなければなりません。これは、オカレンス相互間の距離を個々のオカレンスの長さより大きくできることを意味します。

次の例は、ポインター・サブフィールドを含む複数オカレンス・データ構造のストレージ割り振りを示しています。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D DS1             DS                  OCCURS(2)
D   POINTER                      4*
D   FLD5                         5
D DS2             DS                  OCCURS(2)
D   CHAR16                       4
D   CHR5                         5
```

ストレージ内のフィールドの割り振り。DS1 のオカレンスが 8 バイト・パーツであるのに対して、DS2 のオカレンスは 9 バイト・パーツです。

DS1 OCCURRENCE 1			DS1 OCCURRENCE 2		
POINTER	FLD5	(fill)	POINTER	FLD5	(fill)

DS2 OCCURRENCE 1		DS2 OCCURRENCE 2	
CHAR4	CHR5	CHAR4	CHR5

図 91. ポインター・サブフィールドを含む複数オカレンス・データ構造のストレージ割り振り

OPTIONS(*OMIT *VARSIZE *STRING *RIGHTADJ)

OPTIONS キーワードは、次の指定に使用されます。

- 参照によって渡されるパラメーターの場合に、特殊値 *OMIT を渡すことができるかどうか。
- 参照によって渡されるパラメーターをプロトタイプに指定された長さより短くすることができるかどうか。
- 文字式を渡されるパラメーターとして指定できるように、呼び出し先プログラムまたはプロシージャーでヌル文字で終了するストリングに対するポインターが予期されているかどうか。

OPTIONS キーワードは、パラメーターなしで指定することはできません。1 つの定義仕様に複数のパラメーターを指定できますが、各パラメーターは別のものでなければなりません。

この値は、*OMIT、*STRING、*VARSIZE、または *RIGHTADJ とすることができます。

OPTIONS(*OMIT) を指定した場合には、そのパラメーターに値 *OMIT が許されます。*OMIT は、CONST パラメーターおよび参照によって渡されるパラメーターにのみ使用することができます。

OPTIONS(*VARSIZE) は、参照によって渡され、文字、グラフィック、または UCS-2 データ・タイプを含むか、または任意のタイプの配列を表すパラメーターのにみ有効です。

OPTIONS(*VARSIZE) を指定した場合には、渡されるパラメーターをプロトタイプに指定された長さより短くしたり長くしたりすることができます。これは、次に、呼び出し先プログラムまたはサブプロシージャーにまで及び、渡されたデータだけのアクセスが保証されます。渡された量のデータをやりとりするために、長さを含むエクストラ・パラメーターを渡すか、あるいはサブプロシージャーに対する操作記述子を使用することができます。可変長フィールドの場合には、%LEN 組み込み関数を使用して、渡されたパラメーターの現在の長さを判別することができます。

固定長フィールドについて OPTIONS(*VARSIZE) を省略した場合には、少なくともプロトタイプで必要とされるだけのデータは渡す必要があります。可変長フィールドの場合には、このパラメーターの宣言済み最大長が定義で指示されているものと同じでなければなりません。

注: オプション *OMIT および *VARSIZE を渡すパラメーターの場合には、それがプロシージャーのプログラマーにまで及び、これらのオプションが確実に処理されます。

値または固定情報の参照によって渡される、基底ポインター・パラメーターについて OPTIONS(*STRING)を指定した場合には、ポインターか文字式のいずれかを渡すことができます。文字式を渡す場合には、文字式の値に続いてヌル終了文字 (X'00') を含む一時値が作成されます。この一時値のアドレスが呼び出し先プログラムまたはプロシージャーに渡されます。

関数プロトタイプの中で CONST または VALUE パラメーターに OPTIONS(*RIGHTADJ) を指定した場合には、文字、グラフィック、または UCS-2 パラメーター値は右寄せされます。このキーワードは、プロシージャー・プロトタイプ内の可変長パラメーターには使用できません。可変長の値は、対応するパラメーターが OPTIONS(*RIGHTADJ) によって定義されるプロシージャー呼び出しでパラメーターとして渡すことができます。

次の例は、特殊値 *OMIT をパラメーターとして渡すことができることを指示するための、 OPTIONS(*OMIT) を使用したプロトタイプおよびプロシージャーのコーディング方法を示しています。

```
F*
FQSYSPRT   O    F   10        PRINTER USROPN
D*
D* 次のプロトタイプは、特殊値 *OMIT をパラメーターとして渡す
D* ことができるプロシージャーを記述します。
D* パラメーターが渡された場合に、エラーが起こればこれが '1'
D* に設定され、それ以外は '0' に設定されます。
D OpenFile        PR
D   Error                       1A   OPTIONS(*OMIT)
C*
C                   SETOFF                                       10
C* OpenFile に対する最初の呼び出しではエラーは起こらないことが前提
C* とされるので、エラー・コードに悩まされずに、*OMIT が渡されます。
C                   CALLP     OpenFile(*OMIT)
C*
C* OpenFile に対する 2 番目の呼び出しでは標識が渡されるので、
C* エラーが起こったかどうかを検査することができます。
C                   CALLP     OpenFile(*IN10)
C                   IF        *IN10
C                   ... an error occurred
C                   ENDIF
C                   RETURN
P*---------------------------------------------------------------
P* OpenFile
P* このプロシージャーはパラメーターの数を検査する必要があります。
P*---------------------------------------------------------------
P OpenFile        B
D OpenFile        PI
D   Error                       1A   OPTIONS(*OMIT)
D SaveIn01        S               1A
C* 標識 01 が別の場所で使用されている場合に、その現在の値を
C* 保管します。
C                   EVAL      SaveIn01 = *IN01
C* ファイルをオープンします。*IN01 はエラーが起こったかどうかを示します。
C                   OPEN      QSYSPRT                            01
C* Error パラメーターが渡された場合には、それを標識で更新します。
C                   IF        %ADDR(Error) <> *NULL
C                   EVAL      Error = *IN01
C                   ENDIF
C* *IN01 をその元の値に復元します。
C                   EVAL      *IN01 = SaveIn01
P OpenFile        E
```

*図 92. OPTIONS(*OMIT) の使用*

次の例は、OPTIONS(*VARSIZE) を使用した、可変長パラメーターを使用できるようにするプロトタイプおよびプロシージャーのコーディング方法を示しています。

```
D* 次のプロトタイプは、可変長配列と可変長文字の両方の
D* フィールドを渡すことができるプロシージャーを記述します。
D* 他のパラメーターは長さを指示します。
D Search          PR             5U 0
D   SearchIn                    50A   OPTIONS(*VARSIZE)
D                                     DIM(100) CONST
D   ArrayLen                     5U 0 VALUE
D   ArrayDim                     5U 0 VALUE
D   SearchFor                   50A   OPTIONS(*VARSIZE) CONST
D   FieldLen                     5U 0 VALUE
D*
D Arr1            S               1A   DIM(7) CTDATA PERRCD(7)
D Arr2            S              10A   DIM(3) CTDATA
D Elem            S               5U 0
C* 長さが 1 の検索引き数によって長さが 1 の 7 つのエレメントの配列を
C* 検索するために、Search を呼び出します。'*' は配列の 5 番目のエレメント
C* であるので、Elem は 5 の値をもちます。
C                   EVAL      Elem = Search(Arr1 :
C                                           %SIZE(Arr1) : %ELEM(Arr1) :
C                                           '*' : 1)
C* 長さが 4 の検索引き数によって長さが 10 の 3 つのエレメントの配列を
C* 検索するために、Search を呼び出します。'Pink' はどの配列にも
C* ないので、Elem は 0 の値をもちます。
C                   EVAL      Elem = Search(Arr2 :
C                                           %SIZE(Arr2) : %ELEM(Arr2) :
C                                           'Pink' : 4)
C                   RETURN
```

*図 93. OPTIONS(*VARSIZE) の使用 (1/2)*

```
P*-------------------------------------------------------------
P* Search:
P*   配列 SearchIn の中で SearchFor を検索します。値が見つかった
P*   場合にはそのエレメントを戻し、見つからなければ 0 を戻します。
P*   どちらにも OPTIONS(*VARSIZE) が指定されているので、文字パラメーター
P*   は任意の長さまたはディメンションとすることができます。
P*-------------------------------------------------------------
P Search          B
D Search          PI             5U 0
D   SearchIn                    50A   OPTIONS(*VARSIZE)
D                                     DIM(100) CONST
D   ArrayLen                     5U 0 VALUE
D   ArrayDim                     5U 0 VALUE
D   SearchFor                   50A   OPTIONS(*VARSIZE) CONST
D   FieldLen                     5U 0 VALUE
D I               S              5U 0
C* 配列の各エレメントが SearchFor と同じであるかどうかを確認するために
C* 検査されます。宣言済みディメンションではなく、パラメーターと
C* して渡されたディメンションを使用します。そのパラメーターには
C* して済みの長さが含まれていないことがあるので、長さパラメーター
C* を指定した %SUBST を使用します。
C     1           DO        ArrayDim      I                 5 0
C* このエレメントが SearchFor と一致した場合に、索引を戻します。
C                 IF        %SUBST(SearchIn(I) : 1 : ArrayLen)
C                           = %SUBST(SearchFor : 1 : FieldLen)
C                 RETURN    I
C                 ENDIF
C                 ENDDO
C* 一致したエレメントが見つかりませんでした。
C                 RETURN    0
P Search          E
     **CTDATA ARR1
     A2$@*jM
     **CTDATA ARR2
     Red
     Blue
     Yellow
```

*図 93. OPTIONS(*VARSIZE) の使用 (2/2)*

次の例は、ヌル文字で終了するストリング・パラメーターを使用するプロトタイプおよびプロシージャーをコーディングするための OPTIONS(*STRING) の使用法を示しています。

```
D* 次のプロトタイプは、ヌル文字で終了するストリング・
D* パラメーターを予期するプロシージャーを記述します。
D* これは、ストリングの長さを戻します。
D StringLen       PR             5U 0
D   Pointer                       *   VALUE OPTIONS(*STRING)
D P               S               *
D Len             S              5U 0
C*
C* 文字リテラルを使用して StringLen を呼び出します。リテラルは
C* 4 バイトの長さであるので、結果は 4 となります。
C                   EVAL      Len = StringLen('abcd')
C*
C* ストリングに対するポインターを指定して StringLen を呼び出します。
C* ポインターのストレージを取り出すために ALLOC を使用し、%STR を使用
C* して、そのストレージを 'My string¬' に初期化します。ここで、
C* '¬' はヌル終了文字 x'00' を表します。
C* 結果は、'My string' の長さである 9 となります。
C                   ALLOC     25            P
C                   EVAL      %STR(P:25) = 'My string'
C                   EVAL      Len = StringLen(P)
C* ストレージを解放します。
C                   DEALLOC                 P
C                   RETURN
P*---------------------------------------------------------------
P* StringLen:
P*   パラメーターが指し示しているストリングの長さを
P*   戻します。
P*---------------------------------------------------------------
P StringLen       B
D StringLen       PI             5U 0
D   Pointer                       *   VALUE OPTIONS(*STRING)
C                   RETURN    %LEN(%STR(Pointer))
P StringLen       E
```

*図 94. OPTIONS(*STRING) の使用*

OVERLAY(name{:pos | *NEXT})

OVERLAY キーワードは、1 つのサブフィールドのストレージを別のサブフィールドのストレージまたはデータ構造それ自体のストレージでオーバーレイします。このキーワードは、データ構造サブフィールドにのみ使用することができます。

「名前項目」サブフィールドは、pos パラメーターによって指定された位置で、name パラメーターによって指定されたストレージにオーバーレイします。pos を指定しない場合には、デフォルトの 1 となります。

注: pos パラメーターは、サブフィールドのタイプとは無関係に、バイト単位です。

OVERLAY(name:*NEXT) を指定すると、サブフィールドはオーバーレイされるフィールド内の次に使用可能な位置に位置決めされます。(これは、他のすべてのサブフィールドの後にあり、同じサブフィールドにオーバーレイするこのサブフィールドより前の最初のバイトになります。)

OVERLAY キーワードには、次の規則が適用されます。

- name パラメーターは、現行データ構造の中で前に定義されたサブフィールドの名前であるか、または現行データ構造の名前 でなければなりません。
- データ構造を修飾する場合には、OVERLAY キーワードの最初のパラメーターは、データ構造名の修飾なしに指定しなければなりません。以下の例では、サブフィールド MsgInfo.MsgPrefix がサブフィールド MsgInfo.MsgId をオーバーレイします。

```
D MsgInfo         DS                  QUALIFIED
D   MsgId                        7
D     MsgPrefix                  3    OVERLAY(MsgId)
```

- pos パラメーター (指定する場合) は、小数点以下の桁数のない 0 より大きい値でなければなりません。これは、数値リテラル、数値を戻す組み込み関数、または数値固定情報とすることができます。pos が名前付き固定情報である場合には、それがこの仕様より前に定義されている必要があります。
- 「始め位置」項目がブランクである場合には、OVERLAY キーワードは許されません。
- name パラメーターがサブフィールドである場合には、定義されているサブフィールドがその name パラメーターによって指定されたサブフィールドの中に完全に含まれていなければなりません。
- 定義されているサブフィールドは、name パラメーターによって指定されたサブフィールドの中に完全に含まれていることが必要です。
- OVERLAY キーワードを使用して定義されるサブフィールドの位置合わせは、手動で行なう必要があります。それらが正しく位置合わせされない場合には、警告メッセージが出されます。
- OVERLAY キーワードの最初のパラメーターとして指定されたサブフィールドが配列である場合には、 OVERLAY キーワードはその配列の各エレメントに適用されます。すなわち、定義中のフィールドは、同じ数のエレメントをもつ配列として定義されます。この配列の最初のエレメントは、オーバーレイされる配列の最初のエレメントにオーバーレイし、この配列の 2 番目のエレメントは、オーバーレイされる配列の 2 番目のエレメントにオーバーレイし、以下同様です。この状況では、OVERLAY キーワードをもつサブフィールドには配列キーワードは指定されないことがあります。 291 ページの図 95 および 676 ページの『SORTA (配列の分類)』 を参照してください。

OVERLAYキーワードの最初のパラメーターとして指定されたサブフィールド名が配列であり、そのエレメントの長さが定義中のサブフィールドの長さより長い場合には、定義中のサブフィールドの配列エレメントは連続して保管されません。このような配列は、PARM 演算命令の結果フィールド、あるいは MOVEA 演算命令の演算項目 2 または結果フィールドとしては使用できません。

- ALIGN キーワードがデータ構造について指定された場合に、OVERLAY(name:*NEXT) を指定して定義されたサブフィールドは、優先調整位置で位置合わせされます。ポインター・サブフィールドは、常に 4 バイトの境界で位置合わせされます。
- オーバーレイするサブフィールドを含むサブフィールドが他の方法では定義されない場合には、そのサブフィールドは次のように暗黙的に定義されます。
  - 開始桁は、データ構造内の最初に使用可能な位置となります。
  - 長さは、オーバーレイするすべてのサブフィールドを入れることができる最小長となります。サブフィールドが配列として定義された場合には、オーバーレイするすべてのサブフィールドを確実に正しく位置合わせするために、長さが増やされます。

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D DataStruct      DS
D   A                              10    DIM(5)
D     B                             5    OVERLAY(A)
D     C                             5    OVERLAY(A:6)
```

ストレージ内のフィールドの割り振り:

A(1)		A(2)		A(3)		A(4)		A(5)	
B(1)	C(1)	B(2)	C(2)	B(3)	C(3)	B(4)	C(4)	B(5)	C(5)

図 95. キーワード DIM および OVERLAY を使用したストレージ割り振り

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D DataStruct      DS
D   A                              5
D     B                            1    OVERLAY(A) DIM(4)
```

ストレージ内のフィールドの割り振り:

A				
B(1)	B(2)	B(3)	B(4)	

図 96. キーワード DIM および OVERLAY を使用したストレージ割り振り

次の例は、サブフィールドのオーバーレイ位置を定義する (name:pos) による明示的と (name:*NEXT) による暗黙的の 2 つの同等の方法を示しています。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
 * サブフィールドのオーバーレイ位置を明示的に定義します。
D DataStruct      DS
D   PartNumber                  10A
D     Family                     3A   OVERLAY(PartNumber)
D     Sequence                   6A   OVERLAY(PartNumber:4)
D     Language                   1A   OVERLAY(PartNumber:10)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
 * サブフィールドのオーバーレイ位置を *NEXT を使用して定義します。
D DataStruct      DS
D   PartNumber
D     Family                     3A   OVERLAY(PartNumber)
D     Sequence                   6A   OVERLAY(PartNumber:*NEXT)
D     Language                   1A   OVERLAY(PartNumber:*NEXT)
```

*図 97. *NEXT を使用したサブフィールドのオーバーレイ位置の定義*

PACKEVEN

PACKEVEN キーワードは、パック 10 進数フィールドまたは配列が偶数の桁数となることを指示します。このキーワードは、「始め/終わり」位置を使用して定義されたパック・プログラム記述データ構造サブフィールドにのみ有効です。長さが N のフィールドまたは配列エレメントで、PACKEVEN キーワードが指定されない場合には、桁数は 2N - 1 となり、PACKEVEN キーワードが指定された場合には、桁数は 2(N-1) となります。

PERRCD(numeric_constant)

PERRCD キーワードは、コンパイル時または事前実行時配列またはテーブルの 1 つのレコード当りのエレメントの数を指定します。PERRCD キーワードを指定しない場合には、1 つのレコード当りのエレメントの数はデフォルトの 1 となります。

numeric_constant パラメーターは、小数点以下の桁数のない 0 より大きい値であることが必要です。これは、数値リテラル、数値を戻す組み込み関数、または数値固定情報とすることができます。パラメーターが名前付き固定情報である場合には、それがこの仕様より前に定義されている必要はありません。

PERRCD キーワードは、キーワード FROMFILE、TOFILE、または CTDATA を指定した場合にのみ有効です。

PREFIX(prefix{:nbr_of_char_replaced})

PREFIX キーワードでは、定義中の外部記述データ構造のサブフィールド名の接頭部となる文字ストリングまたは文字リテラルを指定できます。さらに、任意で数値を指定して、置き換えられる既存の名前の文字数を指定することができます (存在する場合) 。パラメーター 'nbr_of_char_replaced' が指定されない場合には、ストリングは名前の先頭に付加されます。

'nbr_of_char_replaced' を指定する場合には、それが小数部のない 0 から 9 の範囲の数値を表している必要があります。ゼロの値を指定すると、

「nbr_of_char_replaced」を指定しない場合とまったく同じになります。たとえば、仕様 PREFIX(YE:3) はフィールド名「YTDTOTAL」を「YETOTAL」に変更します。

次の規則が適用されます。
- EXTFLD キーワードを使用して明示的に名前変更されたサブフィールドは、このキーワードを影響を受けません。
- 接頭部を適用した後の名前の全長が RPG フィールド名の最大長を超えていてはなりません。
- 接頭部を付ける名前の文字数が 'nbr_of_char_replaced' パラメーターによって表された値より小さいか等しい場合には、その名前全体が prefix_string で置き換えられます。
- 接頭部は、ピリオドで終わることはできません。
- 接頭部が文字リテラルの場合には、それは英大文字でなければなりません。

PROCPTR

PROCPTR キーワードは、項目をプロシージャー・ポインターとして定義します。「内部データ・タイプ」フィールド (40 桁目) に * が含まれていなければなりません。

QUALIFIED

QUALIFIED キーワードは、データ構造のサブフィールドがデータ構造名とその後にピリオドとサブフィールド名を続けて指定することによってアクセスされることを指定します。データ構造には名前が必要です。

サブフィールドは、プログラム中のどこか他の場所に使用されている名前であっても、有効な名前としてもつことができます。これは、以下の例に示しています。

```
 * この例では、FILE1 と FILE2 はファイルの名前です。FILE1 と
 * FILE2 は、修飾されたデータ構造 FILESTATUS のサブフィールド
 * でもあります。サブフィールド FILE1 と FILE2 はデータ構造名
 * で修飾されなければならないので (FILESTATUS.FILE1 と
 * FILESTATUS.FILE2)、これは有効です。
Ffile1     if   e            disk         remote
Ffile2     if   e            disk          remote

D fileStatus      ds                  qualified
D   file1                        N
D   file2                        N

C                   open(e)   file1
C                   eval      fileStatus.file1 = %error
```

STATIC

STATIC キーワードは、以下のために使用されます。
- ローカル変数が静的ストレージに保管されることを指定するため。
- Java メソッドが静的メソッドとして定義されていることを指定するため。

サブプロシージャーのローカル変数の場合 、STATIC キーワードは、データ項目は静的ストレージに保管され、それにより、そのデータ項目が定義されたプロシージャーに対する呼び出しにまたがって値が保持されることを指定します。このキーワードは、サブプロシージャーの中でのみ使用することができます。すべてのグローバル・フィールドが静的です。

データ項目は、それが入っているサブプロシージャーが最初に活動化された時に初期化されます。サブプロシージャーが再度呼び出されても、それが再び再初期化されることはありません 。

STATIC を指定しない場合には、ローカルに定義されたすべてのデータ項目が自動ストレージに保管されます。自動ストレージに保管されたデータは、どの呼び出しの開始時にも初期化されます。プロシージャーが再帰的に呼び出された場合には、個々の呼び出しごとにストレージの固有のコピーが取り出されます。

Java メソッドの場合 、STATIC キーワードは、メソッドが静的として定義されることを指定します。 STATIC を指定しない場合には、メソッドはインスタンス・メソッドと見なされます。

STYLE(style_type)

STYLE キーワードは、「メッセージ」ウィンドウ (DSPLY 命令コードの演算項目 2) に使用されるスタイル・タイプを指示します。スタイル・タイプは、次の表意定数の 1 つとすることができます。

- *INFO
- *WARN
- *HALT

MSGDATA、MSGNBR、または MSGTEXT キーワードを使用した場合には、このキーワードは使用できません。

TIMFMT(format{separator})

TIMFMT キーワードは、時刻タイプの次のいずれかの項目について内部時刻形式を指定し、また、任意選択で時刻区切り記号を指定します: 独立型フィールド、データ構造サブフィールド、プロトタイプ・パラメーター、あるいはプロトタイプまたはプロシージャー・インターフェース定義上の戻り値。このキーワードは、時刻タイプの外部記述データ構造サブフィールドの場合には自動的に生成されます。

TIMFMT が指定されない場合には、「時刻」フィールドは制御仕様の TIMFMT キーワードによって指定された通りの時刻形式および区切り文字 (あった場合) をもつことになります。制御仕様になにも指定されていない場合には、*ISO 形式となります。

有効な形式および区切り文字については、 144 ページの表 18 を参照してください。

時刻配列またはフィールドの内部形式および区切り文字を判別する時に使用される階層は、次の通りです。

1. 定義仕様に指定された TIMFMT キーワードから
2. 制御仕様に指定された TIMFMT キーワードから
3. *ISO

TOFILE(file_name)

TOFILE キーワードは、事前実行時またはコンパイル時配列またはテーブルの書き込み先のターゲット・ファイルを指定します。

配列またはテーブルを書き込む場合には、入出力共用ファイルのファイル名をキーワード・パラメーターとして指定してください。このファイルは、また、ファイル仕様にも定義されていることが必要です。配列またはテーブルは、1 つの出力装置にのみ書き込むことができます。

配列またはテーブルをそれが読み取られたのと同じファイルに書き込む場合には、FROMFILE パラメーターに指定したのと同じファイル名を TOFILE パラメーターとして指定する必要があります。このファイルは、入出力共用ファイル (ファイル仕様の 17 桁目に C) として定義されていることが必要です。

VALUE

VALUE キーワードは、パラメーターが参照によってではなく値によって渡されることを指示します。パラメーターは、それと関連したプロシージャーがプロシージャー呼び出しを使用して呼び出された時に、値によって渡すことができます。

CLTPGM キーワードを使用した場合には、パラメーターを値によって渡す必要があります。

値パラメーターとして呼び出し先プロシージャーに渡すことができるパラメーターに関する規則は、EVAL演算命令によって割り当てることができるパラメーターに関する規則と同じです。プロシージャーによって受け取られるパラメーターは式の左辺と対応し、渡されるパラメーターは式の右辺と対応します。詳しくは、 552 ページの『EVAL (式の評価)』を参照してください。

VARYING

VARYING キーワードは、定義仕様に定義された文字、グラフィック、または UCS-2フィールドが可変長形式でなければならないことを指示します。このキーワードを文字、グラフィック、 または UCS-2 フィールドに指定しない場合には、それらは固定長として定義されます。

詳細については、 118 ページの『可変長の文字形式、グラフィック形式、および UCS-2 形式』を参照してください。

定義仕様タイプに従った要約

296 ページの表 29 は、個々の定義仕様タイプに必須および許されている項目をリストしたものです。

296 ページの表 30 および 298 ページの表 31 は、各定義仕様タイプに許されているキーワードをリストしています。

これらの表で、それぞれの **R** は、その位置に項目が必須であることを示し、 **A** は、それらの位置に項目が許されていることを示しています。

表 29. 各定義仕様タイプに必須/許されている項目

タイプ	7 ～ 21 桁目 名前	22 桁目 外部	23 桁目 DS タイプ	24 ～ 25 桁目 定義タイプ	26 ～ 32 桁目 開始位置	33 ～ 39 桁目 終わり/長さ	40 桁目 データ・タイプ	41 ～ 42 桁目 小数点以下の桁数	44 ～ 80 桁目 キーワード
データ構造	A	A	A	R		A			A
データ構造サブフィールド	A				A	A	A	A	A
外部サブフィールド	A	R							A
独立型フィールド	R			R		A	A	A	A
名前付き固定情報	R			R					R
プロトタイプ	R			R		A	A	A	A
プロトタイプ・パラメーター	A					A	A	A	A
プロシージャー・インターフェース	A			R		A	A	A	A
プロシージャー・インターフェース・パラメーター	R					A	A	A	A

表 30. データ構造、独立型フィールド、名前付き固定情報、および「メッセージ」ウィンドウのキーワード

キーワード	データ構造	データ構造サブフィールド	外部サブフィールド	独立型フィールド	名前付き固定情報	「メッセージ」ウィンドウ
ALIGN	A					
ALT		A	A	A		
ASCEND		A	A	A		
BASED	A			A		
BUTTON						A
CCSID		A		A		
CLASS				A		
CONST (1.)					R	
CTDATA (2.)		A	A	A		
DATFMT		A		A		
DESCEND		A	A	A		
DIM		A	A	A		
DTAARA (2.)	A	A		A		
EXTFLD			A			

表 30. データ構造、独立型フィールド、名前付き固定情報、および「メッセージ」ウィンドウのキーワード (続き)

キーワード	データ構造	データ構造サブフィールド	外部サブフィールド	独立型フィールド	名前付き固定情報	「メッセージ」ウィンドウ
EXTFMT		A	A	A		
EXTNAME (4.)	A					
FROMFILE (2.)		A	A	A		
INZ	A	A	A	A		
LIKE		A		A		
LIKEDS [5]	A	A				
LINKAGE				R	R	
MSGDATA						A
MSGNBR						A
MSGTEXT						A
MSGTITLE						A
NOOPT	A			A		
OCCURS	A					
OVERLAY		A				
PACKEVEN		A				
PERRCD		A	A	A		
PREFIX (4.)	A					
PROCPTR		A		A		
QUALIFIED	A					
STATIC (3.)	A			A		
STYLE						A
TIMFMT		A		A		
TOFILE (2.)		A	A	A		
VARYING		A		A		

注:

1 名前付き固定情報の定義時には、キーワードは任意指定ですが、キーワードに対するパラメーターが必要です。たとえば、名前付き固定情報に値 '10' を割り当てる場合は、CONST('10') か '10' のいずれかを指定することができます。

2 このキーワードは、グローバル定義にのみ適用されます。

3 このキーワードは、ローカル定義にのみ適用されます。

4 このキーワードは、外部記述データ構造にのみ適用されます。

5 このキーワードは、プログラム記述データ構造にのみ適用されます。

表 31. プロトタイプ、プロシージャー・インターフェース、およびパラメーターのキーワード

キーワード	プロトタイプ (PR)	プロシージャー・インターフェース (PI)	PR または PI パラメーター
ASCEND			A
CCSID	A	A	A
CLASS	A	A	A
CLTPGM	A		
CONST			A
DATFMT	A	A	A
DESCEND			A
DIM	A	A	A
DLL	A		
EXTPROC	A		
LIKE	A	A	A
LIKEDS	A	A	A
LINKAGE	A		
NOOPT			A
OPTIONS			A
PROCPTR	A	A	A
STATIC	A	A	
TIMFMT	A	A	A
VALUE			A
VARYING	A	A	A

第 19 章 入力仕様

プログラム記述入力ファイルの場合には、入力仕様でファイル内のレコードのタイプ、レコード内のフィールド、フィールド内のデータ、およびフィールドの内容に基づいた標識を記述します。

外部記述ファイルの場合には、入力仕様は任意選択で、機能を外部記述に追加するために使用することができます。

入力仕様の詳細は以下に示されています。

- 300 ページの『プログラム記述ファイル』
- 308 ページの『外部記述ファイル』

入力仕様ステートメント

入力仕様の一般的なレイアウトは次のとおりです。
- 入力仕様のタイプ (I) は 6 桁目に入れます。
- 仕様の非コメント部分は 7 桁目から 80 桁目までです。
- 仕様のコメント・セクションは 81 桁目から 100 桁目までです

プログラム記述

プログラム記述ファイルの場合には、入力仕様の項目は次のカテゴリーに分けられます。
- 入力レコードとファイル内の他のレコードとの関係を記述するレコード識別項目 (7-46 桁目) 。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
IFilename++Sq..RiPos1+NCCPos2+NCCPos3+NCC.................................Comments++++++++++++
I.........And..RiPos1+NCCPos2+NCCPos3+NCC.................................Comments++++++++++++
```

図 98. プログラム記述レコード・レイアウト

- レコード内のフィールドを記述するフィールド記述項目 (31-74 桁目) 。各フィールドは、その対応するレコード識別項目の下に、別々の行に記述されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
I.......................Fmt+SPFrom+To+++DcField+++++++++....FrPlMnZr......Comments++++++++++++
```

図 99. プログラム記述フィールド・レイアウト

外部記述

外部記述ファイルの場合には、入力仕様の項目は次のカテゴリーに分けられます。
- VARPG 機能が追加されるレコード (外部記述レコード様式) を識別するレコード識別項目 (7 - 16 桁目と 21 - 22 桁目)。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
IRcdname+++....Ri.....................................................Comments++++++++++++
```

図 100. 外部記述レコード・レイアウト

- レコード内のフィールドに VARPG 機能が追加されるフィールド記述項目 (21-30 桁目、49-66 桁目、および 69-74 桁目) 。フィールド記述項目は、対応するレコード識別項目の後の行に書き込まれます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
I..............Ext-field+..................Field+++++++++......PlMnZr......Comments++++++++++++
```

図 101. 外部記述フィールド・レイアウト

プログラム記述ファイル

入力仕様では、プログラム記述ファイルに次の項目が含まれています。

6 桁目 (仕様のタイプ)

この行を入力仕様ステートメントとして識別するには、6 桁目に I がなければなりません。

レコード識別項目

プログラム記述ファイルのレコード識別項目 (7-46 桁目) は、入力レコードとファイル内の他のレコードとの関係を記述します。

7-16 桁目 (ファイル名)

項目　説明

有効なファイル名

これは、入力ファイルのファイル記述仕様に表示されるものと同じ名前です。

記述するファイルの名前をこれらの桁に入力してください。この名前は、ファイル記述仕様でこのファイルに定義した名前と同じでなければなりません。このファイルは入力ファイル、更新ファイル、または入出力共用ファイルでなければなりません。ファイル名は、それぞれのファイルの最初のレコード識別行に入力しなければならず、そのファイルの以後のレコード識別行に入力することができます。1 つの入力ファイルを記述するすべての項目は一緒に表示されなければならず、他のファイルの項目と混ぜることはできません。

16-18 桁目 (論理関係)

項目　説明

AND　4 つ以上の ID が使用されます。

OR　2 つまたはそれ以上のレコード・タイプが共通フィールドを持ちます。

AND/OR 行の数は制限なしに使用することができます。詳細については、 303 ページの『AND 関係』 および 303 ページの『OR 関係』 を参照してください。

17-18 桁目 (順序)

項目　説明

任意の英数字 2 文字
特別の順序検査は行なわれません。

19 桁目 (予約済み)

項目　説明

ブランク
レコード・タイプは特別の順序では検査されません (17 桁目と 18 桁目に英数字項目) 。

20 桁目 (オプション)

項目　説明

ブランク
この項目は、17 桁目と 18 桁目に英数字項目が入っている場合には、ブランクでなければなりません。

21-22 桁目 (レコード識別標識)

項目　説明

ブランク
標識は使用されません。

01 ～ 99
一般的な標識

LR　レコード識別標識に使用する制御レベル標識

これらの桁に指定された標識は、レコード ID (23-46 桁目) と一緒に使用されます。

標識

21-22 桁目は、標識をこの行で定義されたレコード・タイプと関連づけます。01-99 または LR を入力することができます。

処理のためにレコードが選択されてレコード ID で指示された条件が満たされると、該当するレコード識別標識がオンに設定されます。この標識は、演算および出力命令の条件づけに使用することができます。レコード識別標識は、プログラマーがオンまたはオフに設定することができます。

23-46 桁目 (レコード ID)

23-46 桁目の項目は、入力ファイルのそれぞれのレコード・タイプを識別します。それぞれの仕様行に 1-3 の ID を入力することができます。4 つ以上のレコード ID は、AND/OR 関係で追加の行に指定することができます。ファイルに 1 つのレコード・タイプしか入っていない場合には、 ID はブランクのままにすることができますが、レコード識別標識 (21-22 桁目) と順序 (17-18 桁目) には入力しなければなりません。

注: レコード ID は、グラフィック UCS-2 データ・タイプの処理には適用されません。レコード識別は単一バイトの桁のみで行なわれます。

23-46 桁目 (23-30 桁目、31-38 桁目、および 39-46 桁目) には、3 セットの項目を入力することができます。それぞれのセットは 4 つのグループに分かれます。桁、否定、コード部分、および文字です。

次のテーブルに、それぞれのセットでどのカテゴリーがどの桁を使用するかを示します。

カテゴリー	23-30	31-38	39-46
位置	23-27	31-35	39-43
否定	28	36	44
コード部分	29	37	45
文字	30	38	46

これらのセットの項目は順番にする必要はありません。たとえば、23 ～ 30 桁目に入力しなくても 31 ～ 38 桁目に入力することができます。ファイル内の入力レコードが同じタイプの場合には、レコード ID を入力する必要はありません。レコード ID が入っていない入力仕様はそのファイルの最後のレコード・タイプを定義するので、未定義のレコード・タイプを処理することができます。レコード ID が満たされない場合には、制御は VARPG 例外 / エラー処理ルーチンに渡されます。

23 ～ 27 桁目、31 ～ 35 桁目、および 39 ～ 43 桁目 (位置)

項目 **説明**

ブランク レコード ID は渡されません。

1 ～ 32766 これは、レコードにレコード ID が入っている位置です。コードが入っている位置は、そのファイルに指定されたレコード長の範囲内でなければなりません。この項目は右寄せされなければなりませんが、先行ゼロは省略することができます。

28 桁目、36 桁目、および 44 桁目 (否定)

項目 **説明**

ブランク
レコード ID が渡されなければなりません。

N この位置の N は、指定したレコード位置にレコード ID があってはならないことを意味します。

29 桁目、37 桁目、および 45 桁目 (コード部分)

この項目は、レコード ID のどの文字部分をテストするかを指定します。

項目	説明
C	文字全体
D	ディジット

文字 (C): C 項目は、文字の完全な構造 (ゾーンおよびディジット) をテストすることを示します。

ディジット (D): D 項目は、文字のディジット部分をテストすることを示します。文字の右端 4 ビットが位置項目で指定された文字と比較されます。

30 桁目、38 桁目、および 46 桁目 (文字)

この位置の項目は、入力レコードに指定された位置の文字と比較される識別文字を示します。

レコード・タイプの検査は、指定された最初のレコード・タイプから開始されます。レコードのデータが複数セットのレコード ID を満たす場合には、満たされた最初のレコード・タイプによってレコード・タイプが決定されます。

1 つのファイルに複数のレコード・タイプが指定されている場合には、それぞれの入力レコードが固有の ID のセットを持つようにレコード ID をコーディングしなければなりません。

AND 関係

AND 関係は、1 つのレコードを 4 つ以上のレコード ID で識別するときに使用します。

AND 関係を使用するには、最初の行に少なくとも 1 つのレコード ID を入れ、追加の各行の 16-18 桁目に AND をコーディングして、それらの行に残りのレコード ID を入力します。16-18 桁目に AND を持つ各行の 7-15 桁目、19-20 桁目、および 46-80 桁目はブランクでなければなりません。順序とレコード識別標識項目はグループの最初の行に入れなければならず、追加の行に指定することはできません。

入力仕様では、AND/OR 行の数は制限なしに使用することができます。

OR 関係

OR 関係は、2 つまたはそれ以上のレコード・タイプが共通フィールドを持つときに使用します。

OR 関係を使用するには、16-17 桁目に OR を入力します。7-15 桁目、18-20 桁目、および 46-80 桁目はブランクでなければなりません。レコード識別標識は 21-22 桁目に入れることができます。標識を入力して、OR 行のレコード ID が満たされると、その行の 21-22 桁目に指定された標識がオンに設定されます。標識を入力しないと、前の行の標識がオンに設定されます。

入力仕様では、AND/OR 行の数は制限なしに使用することができます。

フィールド記述項目

フィールド記述項目 (31-74 桁目) は、それぞれのファイルのレコード識別項目 (7-46 桁目) に続けられなければなりません。

6 桁目 (仕様のタイプ)

この行を入力仕様ステートメントとして識別するには、6 桁目に I がなければなりません。

7-30 桁目 (予約済み)

7-30 桁目はブランクでなければなりません。

31-34 桁目 (データ属性)

31-34 桁目は、日付、時刻、または可変長文字、グラフィック、あるいは UCS-2 フィールドの外部形式を指定します。

日付または時刻フィールドでこの項目がブランクの場合には、ファイルに (DATFMT または TIMFMT で) 指定された形式 / 区切り文字が使用されます。ファイルに外部日付または時刻形式が指定されていない場合には、エラー・メッセージが出されます。日付および時刻の形式については、 125 ページの『日付データ』 および 144 ページの『時刻データ』 を参照してください。

外部日付/時刻形式および日付と時刻フィールドの区切り記号を決定する時に使用される階層は次の通りです:

1. 31-35 桁目に指定された日付形式と区切り文字
2. 現行ファイルに指定されている DATFMT/TIMFMT キーワードから
3. 制御仕様に指定されている DATFMT/TIMFMT キーワードから
4. *ISO

日付および時刻フィールドは、上で決定された外部日付 / 時刻形式から日付 / 時刻フィールドの内部形式に変換されます。

文字、グラフィック、または UCS-2 データの場合には、可変長入力フィールドを指定するために *VAR データ属性が使用されます。文字、グラフィック、または UCS-2 データでこの項目がブランクの場合には、外部形式は固定長でなければなりません。プログラム内のどこかでこのフィールドが定義されている場合には、内部形式と外部形式が一致しなければなりません。可変長フィールドの詳細については、 118 ページの『可変長の文字形式、グラフィック形式、および UCS-2 形式』 を参照してください。

外部形式の詳細については、 107 ページの『内部形式と外部形式』 を参照してください。

35 桁目 (日付 / 時刻区切り文字)

35 桁目は、日付 / 時刻フィールドに使用する区切り文字を指定します。 & (アンパーサンド) は、ブランクの区切り文字を指定するために使用することができます。日付および時刻の形式とそれらのデフォルトの区切り文字については、 125 ページの『日付データ』 および 144 ページの『時刻データ』 を参照してください。

このフィールドに入力した場合には、31-34 桁目 (日付 / 時刻外部形式) にも入力しなければなりません。

36 桁目 (データ形式)

入力フィールドは次のとおりです。

項目　説明

ブランク
　ゾーン 10 進数形式また文字

A　文字フィールド (固定または可変長形式)

N　文字フィールド (標識形式)

G　グラフィック・フィールド (固定または可変長形式)

C　UCS-2 フィールド (固定または可変長形式)

B　2 進数形式

F　数値フィールド (浮動形式)

I　数字形式 (整数形式)

L　前 (左) にプラスまたはマイナス符号を持つ数値フィールド (ゾーン 10 進形式)

P　数値フィールド (パック 10 進数形式)

R　後 (右) にプラスまたはマイナス符号を持つ数値フィールド (ゾーン 10 進形式)

S　数値フィールド (ゾーン 10 進数フィールド)

U　数値フィールド (符号なし形式)

D　日付フィールド - 日付フィールドは 31-34 桁目に指定された外部形式またはデフォルトのファイル日付形式を持ちます。

T　時刻フィールド - 時刻フィールドは、31-34 桁目に指定された外部形式またはデフォルトのファイル時刻形式を持ちます。

Z　タイム・スタンプ・フィールド

36 桁目の項目はデータ・タイプを指定し、数字の場合にはプログラム記述ファイルのデータの外部データ形式を指定します。この項目は、プログラムで入力フィールドの内部処理に使用される形式には影響を与えません。

37 ～ 46 桁目 (フィールドの位置)

項目　説明

2 つの 1 ～ 5 桁の数字
　フィールドの始め (から) とフィールドの終わり (まで) 。

この項目は、入力レコードの各フィールドの位置とサイズを記述します。 37-41 桁目はフィールドの開始桁の位置を指定し、42-46 桁目はフィールドの終了桁の位置

を指定します。 1 桁のフィールドを定義するには、37-41 桁目と 42-46 桁目に同じ数字を入力します。数値項目は右寄せされなければなりませんが、先行ゼロは省略することができます。

フィールドのそれぞれのタイプの入力レコードの最大桁数は次のとおりです。

桁数	フィールドのタイプ
30	ゾーン 10 進数 (30 桁)
16	パック数値 (30 桁)
4	2 進数 (9 桁)
8	整数 (20 桁)
8	符号なし (20 桁)
8	浮動 (8 バイト)
31	先行または末尾符号付きの数字 (30 桁)
10	Date
8	時刻
26	タイム・スタンプ
32766	文字 (32766 文字)
32766	可変長文字 (32764 文字)
32766	グラフィックまたは UCS-2 (2 バイト文字で 16383 文字)
32766	可変長グラフィックまたは UCS-2 (2 バイト文字で 16382 文字)
32766	データ構造

プログラム記述入力フィールドとして指定された文字またはデータ構造フィールドの最大サイズは、ファイルの最大レコード長である 32766 です。

可変長文字、グラフィック、または UCS-2 入力フィールドを指定する場合には、長さに 2 バイトの長さの接頭部が含まれます。

配列の場合には、37-41 桁目に配列の開始位置を入れ、42-46 桁目に終了位置を入れます。配列の長さは、エレメントの長さの整数倍でなければなりません。開始・終了位置は、配列のすべてのエレメントを考慮する必要はありません。配列へのデータの配置は、最初のエレメントから始められます。

47-48 桁目 (小数点以下の桁数)

項目　説明

ブランク
文字、グラフィック、UCS-2、浮動、日付、時刻、またはタイム・スタンプ・フィールド。

0 ～ 30
数値フィールドの小数点以下の桁数。

この項目は、36 桁目のデータ形式項目と一緒に使用して、フィールドの形式を記述します。このフィールドの項目は、入力フィールドを (浮動数値以外の) 数値として識別します。フィールドが数値の場合には、入力する必要があります。数値フィールドに指定された小数点以下の桁数は、フィールドの長さを超えることができません。

49 ～ 62 桁目 (フィールド名)

項目　説明

記号名 フィールド名、データ構造名、データ構造サブフィールド名、配列名、配列エレメント、PAGE、PAGE1-PAGE7、*IN、*INxx、または *IN(xx) 。

これらの桁は、VARPG プログラムで使用される入力レコードのフィールドを指定します。この名前は、記号名の規則に従わなければなりません。

入力仕様の配列全体を参照するには、49-62 桁目に配列名を入力します。49-62 桁目に配列名を入力した場合には、フィールド標識 (67-68 桁目) はブランクでなければなりません。

配列のエレメントを参照するには、配列名に続けて括弧で囲んだ索引を指定します。この索引は、小数点以下の桁数がゼロの数値フィールドか、使用する配列エレメントの実際の数のいずれかです。索引の値は 1 から n までとすることができ、この場合の n は配列内のエレメント数になります。

63 ～ 64 桁目 (予約済み)

項目　説明

ブランク
この項目はブランクでなければなりません。

65 ～ 66 桁目 (予約済み)

項目　説明

ブランク
この項目はブランクでなければなりません。

67 ～ 68 桁目 (フィールドとレコードの関連)

項目　説明

ブランク
このフィールドはすべてのレコード・タイプに共通です。

01 ～ 99
一般標識。

フィールドとレコードの関連標識は、特定のレコード・タイプ内のフィールドを (そのレコード・タイプが OR 関係のいくつかのうちの 1 つである場合に) 関連づけるために使用します。この項目は、書かなければならない行数を減らします。

行に記述されるフィールドがレコードから抜き出されるのは、67 および 68 桁目にコーディングされている標識がオンであるか、あるいは 67 および 68 桁目がブランクの場合だけです。67 および 68 桁目がブランクの場合には、そのフィールドは OR 関係で定義されたすべてのレコード・タイプに共通です。

69 ～ 74 桁目 (フィールド標識)

項目　説明

ブランク
標識は指定されません

01 ～ 99
一般標識

69 ～ 74 桁目の項目は、プログラムに読み取られたフィールドまたは配列エレメントの状況をテストします。フィールド標識は、テストするフィールドと同じ行に指定されます。フィールドの状況 (プラス、マイナス、ゼロ、またはブランク) によって該当する標識がオンに設定されて、後の仕様の条件づけに使用することができます。2 つの位置に同じ標識を指定することができますが、3 つの位置全部に使用してはいけません。フィールド標識は、索引づけされていない配列に使用することはできません。

69 ～ 70 桁目 (プラス) と 71 ～ 72 桁目 (マイナス) は数値フィールドだけに有効です。73 ～ 74 桁目は、数値フィールドのゼロまたは文字、グラフィック、UCS-2 フィールドのブランクのテストに使用することができます。

フィールド標識は、レコードが読み取られたときにフィールドまたは配列エレメントが指定された条件に適合した場合にはオンに設定されます。それぞれのフィールド標識は 1 つのレコード・タイプに関連があるだけです。したがって、関連したレコードがもう一度読み取られるかまたは他の仕様で標識が定義されない限り、標識は (オンまたはオフに) 設定されません。

外部記述ファイル

外部記述ファイルには、入力仕様における以下の記入項目が組み込まれています。

6 桁目 (仕様のタイプ)

この行を入力仕様ステートメントとして識別するためには、6 桁目に I が表示されなければなりません。

レコード識別項目

外部記述ファイルの記述がリトリーブされると、レコード定義もリトリーブされます。レコード定義を参照するには、プログラムの入力、演算、および出力仕様にレコード様式名を指定してください。外部記述ファイルの入力仕様は次の場合に必要です。

- レコード識別標識が指定される場合。
- レコード内のフィールドがプログラムで名前変更される場合。
- フィールド標識が使用される場合。

フィールド記述仕様は、外部記述ファイルのレコード識別仕様の直後に続けられなければなりません。

外部記述ファイルのレコード行は、レコードの指定変更仕様の始めを定義します。レコード行の後のすべての仕様は、入力仕様の 7-16 桁目に別のレコード様式名またはファイル名が見つかるまで、レコード指定変更の一部となります。外部記述ファイルに属するすべてのレコード行は一緒に表示されなければなりません。他のファイルの項目と一緒にすることはできません。

7 ～ 16 桁目 (レコード名)

次の 1 つを入力してください。

- レコード様式の外部名。このファイル名は外部記述ファイルの場合には使用できません。
- 外部レコード様式が名前変更された場合に、ファイル記述仕様で RENAME キーワードによって指定された名前。レコード様式名は、プログラムの入力仕様の 7-16 桁目に一回しか表示できません。

17-20 桁目 (予約済み)

17-20 桁目はブランクでなければなりません。

21-22 桁目 (レコード識別標識)

これらの位置のレコード識別標識の指定はオプションですが、存在する場合には の規則に従います。プログラム記述ファイル。 300 ページの『プログラム記述ファイル』 を参照してください。

23-80 桁目 (予約済み)

23-80 桁目はブランクでなければなりません。

フィールド記述項目

外部記述ファイルのフィールド記述の指定は、プログラムのレコード内のフィールドの名前を変更し、あるいはフィールド標識機能を指定するために使用することができます。フィールド定義 (属性) は外部記述ファイルからリトリーブされ、プログラムによって変更することはできません。フィールドの属性が VARPG プログラムに有効でない (数値の長さが 30 桁を超えているなどの) 場合には、そのフィールドは使用できません。ソース・ステートメントの場合と同様に、外部レコード形式で入っているフィールドに対して診断検査が行なわれます。

通常、入力操作中に外部記述入力フィールドが読み取られるのは、そのフィールドが実際にプログラム内のどこかで使用されている場合だけです。

7-20 桁目 (予約済み)

7-20 桁目はブランクでなければなりません。

21-30 桁目 (外部フィールド名)

外部記述ファイルのレコード内のフィールドを名前変更する場合には、これらの位置にフィールドの外部名を入力してください。フィールドは、名前がプログラムで指定されたフィールド名と同じで、2 つの別の名前が必要なために名前変更しなければならない場合があります。

31-48 桁目 (予約済み)

31-48 桁目はブランクでなければなりません。

49 ～ 62 桁目 (フィールド名)

フィールド名は、それが外部記述に追加する制御レベルなどの機能に必要な場合にだけ入力します。フィールド名項目には、以下の 1 つが入ります。

- 外部レコード記述で定義されたフィールドの名前 (10 文字またはそれ以下)
- プログラムで使用するように指定された、21-30 桁目に指定された外部名と置き換える名前。

フィールド名は、記号名を使用するための規則に従わなければなりません。

標識を NULL 値可能にすることはできません。

63 ～ 64 桁目 (予約済み)

項目　説明

ブランク
この項目はブランクでなければなりません。

65 ～ 66 桁目 (予約済み)

項目　説明

ブランク
この項目はブランクでなければなりません。

67 ～ 68 桁目 (予約済み)

67 ～ 68 桁目はブランクでなければなりません。

69 ～ 74 桁目 (フィールド標識)

項目　説明

ブランク
標識は指定されません

01 ～ 99
一般標識

フィールド標識を NULL 値可能フィールドに使用できるのは、**ユーザー制御**オプションまたは **ALWNULL(*USRCTL)** キーワードが指定されている場合だけです。

NULL 値可能フィールドに NULL 値が入っている場合には、標識はオフに設定されます。

プログラム記述ファイルについては、 307 ページの『69 ～ 74 桁目 (フィールド標識)』 を参照してください。

75 ～ 80 桁目 (予約済み)

75 ～ 80 桁目はブランクでなければなりません。

第 20 章 演算仕様

演算仕様 は、プログラム中のデータに対して実行する演算命令を指示します。

演算仕様は 3 つの異なる形式で指定することができます。
- 『従来からの構文』
- 316 ページの『拡張演算項目 2 の構文』
- 318 ページの『フリー・フォーム構文』

個々の命令コードについて必要な演算仕様項目の指定方法の詳細については、481 ページの『第 26 章 命令コードの詳細』 を参照してください。

従来からの構文

演算仕様の一般レイアウトは次のとおりです。
- 演算仕様タイプ (C) を 6 桁目に入力します。
- 仕様の非コメント部分は、7 桁目から 80 桁目までです。これらの桁は、以下の項目を指定する 3 つの部分に分けられています。
  - 演算の実行時点: 7 - 11 桁目に指定する条件標識によって、その演算を実行する時点および条件が決まります。
  - 実行する演算の種類: 12 -70 桁目 (拡張演算項目を使用する命令については、12 - 80 桁目、316 ページの『拡張演算項目 2 の構文』および 383 ページの『第 24 章 式』 を参照) に指定する項目は、実行する演算の種類、命令を実行する対象データ (フィールドやファイルなど)、および演算の結果を入れるフィールドを指定します。
  - 演算命令の結果に対して実行するテスト: 71 - 76 桁目に指定する標識は、演算の結果のテストに使用され、続く演算のまたは出力操作の条件付けに使用できます。演算結果標識の桁には、命令コードによっていろいろな用途があります。
- 仕様のコメント・セクションは 81 桁目から 100 桁目までです

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....Comments++++++++++++
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++Comments++++++++++++
```

図 102. 演算仕様のレイアウト

演算仕様の拡張演算項目 2 継続行

拡張演算項目 2 フィールドは、次のように後続行に継続できます。
- 継続行の 6 桁目には、C を入れなければなりません。
- 継続行の 7 - 35 桁目はブランクでなければなりません。
- 仕様は、36 桁目以降に継続します。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
C.........................Extended-factor2-continuation+++++++++++++++++Comments++++++++++++
```

図 103. 演算仕様の拡張演算項目 2 継続行

6 桁目 (仕様のタイプ)

この行が演算仕様ステートメントであることを示すために、6 桁目に C がなければなりません。

7-8 桁目 (制御レベル)

項目	説明
ブランク	この演算命令は、9 - 11 桁目の標識によって許された場合に実行されます。あるいはこの演算はサブルーチンの一部です。また、ブランクは、宣言命令コードの場合にも使用されます。
SR	この演算命令は、サブルーチンの一部です。サブルーチンの一部である演算には、ブランク項目も有効です。
AN、OR	複数行の標識が演算を条件づけています。

サブルーチン ID

7 - 8 桁目の SR 項目は、任意でサブルーチン内の演算命令に使用しますが、これはドキュメンテーションのためです。命令コード BEGACT および ENDACT は、アクション・サブルーチンの区切り文字として使用します。命令コード BEGSR および ENDSRは、サブルーチンに使用します。

AND/OR 行 ID

7 - 8 桁目には、AN または OR を入れて、演算の追加標識 (9 - 11 桁目) を定義します。

AND/OR 行または AND/OR 行グループの直前の行の 7 - 8 桁目の項目によって、の処理時点が決まります。グループの最初の行の 7 - 8 桁目の項目が、グループ中のすべての AND/OR 行に適用されます。

9-11 桁目 (標識)

10 - 11 桁目には、特定の演算を処理するかどうかを決めるためにテストする標識を入れます。

項目	説明
ブランク	この演算命令はすべてのレコードに対して処理されます。
01 ～ 99	一般標識
LR	最終レコード標識。

9 桁目のブランクは、演算の実行にはその標識がオンでなければならないことを指定します。9 桁目の N は、演算の実行には関連した標識がオフでなければならないことを指定します。

12-25 桁目 (演算項目 1)

演算項目 1 には、演算命令の対象となるフィールドや実際のデータ (リテラル) を指定し、あるいは演算命令の実行方法に関する追加の情報を提供する VARPG 特殊語 (たとえば *LOCK) を入れます。この項目は、12 桁目から開始する必要があります。演算項目 1 の項目は、26 ～ 35 桁目に指定する命令コードによってかわります。特定の命令コードの場合の演算項目 1 の特定の入力項目については、 481 ページの『第 26 章 命令コードの詳細』 を参照してください。一部の命令コードでは、2 つのオペランドをコロンで区切って指定することがあります。

26-35 桁目 (演算命令および拡張機能)

26 ～ 35 桁目には、演算項目 1、演算項目 2、および結果フィールドの項目を使用して実行する命令の種類を指定します。命令コードは 26 桁目から開始しなければなりません。 命令コードの詳細については、 345 ページの『第 23 章 演算命令』および 481 ページの『第 26 章 命令コードの詳細』を参照してください。

命令拡張

項目　説明

ブランク
命令拡張を使用しません。

H　数値演算の結果を四捨五入し (丸め)、四捨五入を実行した後の結果フィールドの値にしたがって演算結果標識を設定します。

N　READ、READE、READP、READPE、または CHAIN 命令でディスク・ファイルの更新時にレコードは読み取られますが、ロックされません。

正常な DEALLOC の後で、ポインターが *NULL に設定されます。

P　結果フィールドが演算命令の結果よりも長い場合に、結果フィールドにブランクを埋め込みます。

バインド済み呼び出しでは、操作記述子をわたします。

D　日付フィールド。

T　時刻フィールド。

Z　タイム・スタンプ・フィールド

M　デフォルト精度規則。

R　″結果の小数点以下の桁数″ 精度規則。

E　エラー処理。

命令拡張は、付帯する命令に追加の属性を提供します。命令拡張は、演算仕様の 26-35 桁目に指定します。これらは、命令コードの右から開始し、括弧に入れる必要があります。読みやすくするためにブランクを使用できます。たとえば、次の項目が有効です: MULT(H), MULT (H), MULT (H)。

複数の命令拡張を指定できます。たとえば、EVAL 命令には、EVAL(HM) として四捨五入とデフォルト精度規則の両方を指定できます。

H は、結果フィールドの内容を四捨五入する (丸める) かどうかを指示します。演算結果標識は、四捨五入が行なわれた後で、結果フィールドの値に従って設定されます。

READ、READE、READP、READPE、または CHAIN 命令の N は、ディスク・ファイルの更新時にレコードは読み取られるが、ロックされないことを指示します。値を指定しないと、ロックのデフォルト・アクションが実行されます。

DEALLOC 命令の N は、正常な割り振り解除の後で、結果フィールドのポインターが *NULL に設定されることを指示します。

P は、結果フィールドが演算の結果より長い場合に、命令の実行後に結果フィールドが埋め込まれることを指示します。

D、T、および Z 拡張は、TEST 命令コードで使用し、日付、時刻、または時刻スタンプ・フィールドを指示します。

M および R は、単一のフリー・フォームの精度用に指定します。詳細については、392 ページの『数値演算の精度の規則』を参照してください。

M は、デフォルトの精度規則が使用されることを指示します。

R は、小数点以下の桁数が割り当て結果の小数点以下の桁数よりも小さくならないように、中間の10 進数が計算されることを指示します。

E は、演算命令関連のエラーが組み込み関数 %ERROR でチェックされることを指示します。

36-49 桁目 (演算項目 2)

演算項目 2 には、演算命令の対象となるフィールド、レコード様式、またはファイルや実際のデータを指定し、あるいは演算命令の実行方法に関する追加の情報を提供する特殊語 (たとえば *ALL) を入れます。この項目は、36 桁目から開始する必要があります。演算項目 2 に有効な項目は、26 ～ 35 桁目に指定する命令コードによってかわります。一部の命令コードでは、2 つのオペランドをコロンで区切って指定することがあります。特定の命令コードの場合の演算項目 2 の特定の入力項目については、481 ページの『第 26 章 命令コードの詳細』 を参照してください。

50-63 桁目 (結果フィールド)

結果フィールドには、26 ～ 35 桁目に指定された演算命令の結果を入れるフィールドまたはレコード様式を指定します。指定するフィールドは、変更可能なフィールドでなければなりません。たとえば、ユーザー日付フィールドを使用することはできません。一部の命令コードでは、2 つのオペランドをコロンで区切って指定することがあります。個々の命令コードの結果フィールドの規則については、481 ページの『第 26 章 命令コードの詳細』を参照してください。

64-68 桁目 (フィールド長)

項目	説明
1-30	数値フィールドの長さ。
1-65535	文字フィールドの長さ。
ブランク	結果フィールドが他のどこかで定義されているか、あるいはこの命令コードを使用してフィールドを定義できません。

64 - 68 桁目には、結果フィールドの長さを指定します。この項目は、任意指定ですが、プログラム中の他の個所で定義されていない数値または文字フィールドの定義に使用できます。フィールド項目のこれらの定義は、結果フィールドにフィールド名が入っている場合に使用できます。他のデータ・タイプは、定義仕様で定義するか、あるいは *LIKE DEFINE 命令を使用する演算仕様で定義する必要があります。

この項目は、結果フィールドに予約する桁数を指定します。項目は右寄せしなければなりません。数値フィールドについては、アンパック長さ (桁数) を指定する必要があります。

結果フィールドがプログラム中の他のどこかで定義されている場合には、長さについての入力は不要です。ただし、長さを指定し、その結果フィールドが他の個所で定義されている場合には、長さは前に定義された長さと同じでなければなりません。結果フィールド長が前に定義した長さと異なっている場合には、前に定義された値が使用されます。

69-70 桁目 (小数点以下の桁数)

項目　説明

ブランク
結果フィールドが文字データであるか、プログラム中の他の個所で定義されているか、あるいはフィールド長を指定しません。

0 ～ 30
数値の結果フィールドの小数点以下の桁数:
- 数値の結果フィールドに小数点以下の桁数がない場合には、'0' (ゼロ) を入力します。
- 指定する小数点以下の桁数は、フィールドの長さを超えることはできません。

69 - 70 桁目は、数値の結果フィールドの小数点の右側の桁数を指示します。

71-76 桁目 (演算結果標識)

これらの桁を使用して、命令完了後の結果フィールドの値をテストしたり、あるいはファイルの終わり、エラー、またはレコード不存在といった条件を指示できます。一部の命令では、3 つの演算結果標識のいろいろな組み合わせを指定することによって、命令の実行方法を制御できます (たとえば LOOKUP)。演算結果標識の桁には、指定する命令コードによっていろいろな用途があります。関連した結果標識の説明については、481 ページの『第 26 章 命令コードの詳細』の個々の命令コードを参照してください。算術演算の場合に、結果フィールドがテストされるのは、フィールドが切り捨てられ、四捨五入 (指定されている場合) が実行された後でだけです。標識の設定は、指定されたテストの結果に依存します。

項目　説明

ブランク
演算結果標識は指定されません。

01 ～ 99
一般標識

LR　最終レコード標識。

結果フィールドに指標のない配列を使用している時には、演算結果標識は使用できません。

複数の演算仕様で演算結果標識と同じ標識を使用した場合には、最も後に処理された仕様によってその標識の状況が決まります。

注: 演算命令の実行時には、指定された演算結果標識がオフに設定され、演算結果標識によって指定された条件が満たされると、その標識がオンに設定されます。

拡張演算項目 2 の構文

特定の命令コードでは、拡張演算項目 2 フィールドに式を使用できます。

7-8 桁目 (制御レベル)

312 ページの『7-8 桁目 (制御レベル)』 を参照してください。

9-11 桁目 (標識)

312 ページの『9-11 桁目 (標識)』 を参照してください。

12-25 桁目 (演算項目 1)

演算項目 1 はブランクでなければなりません。

26-35 桁目 (演算命令および拡張機能)

26 ～ 35 桁目には、拡張演算項目 2 フィールドの式を使用して実行する命令の種類を指定します。命令コードは 26 桁目から開始しなければなりません。 命令コードの詳細については、345 ページの『第 23 章 演算命令』および 481 ページの『第 26 章 命令コードの詳細』を参照してください。

プログラムは、演算仕様フォームに指定された順序で命令を処理します。

命令拡張

項目　説明

ブランク
　命令拡張を使用しません。

H　数値演算の結果を四捨五入し (丸め) ます。

M　デフォルト精度規則。

R　″結果の小数点以下の桁数″ 精度規則。

E　エラー処理。

演算命令 EVAL および RETURN で H 拡張を使用して、四捨五入を指定できます。

DOU、DOW、EVAL、IF、RETURN、および WHEN 命令で M または R 拡張を使用して、精度のタイプを指定できます。

’E’ 拡張機能を使用して、エラー処理を指定できます。

36-80 桁目 (拡張演算項目 2)

このフィールドではフリー・フォーム構文を使用します。これは、オペランドと演算子の組み合わせで構成され、任意に複数行にスパンできます。複数行にわたって指定するには、継続行の 7 - 35 桁目はブランクでなければなりません。

拡張演算項目 2 を使用する演算命令は次のとおりです。

- 503 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』
- 537 ページの『DOU (実行の終了)』
- 541 ページの『DOW (実行の時期)』
- 552 ページの『EVAL (式の評価)』
- 554 ページの『EVALR (式の評価、右寄せ)』
- 563 ページの『FOR (For)』
- 569 ページの『IF (判断)』
- 626 ページの『ON-ERROR (エラー処理)』
- 659 ページの『RETURN (呼び出し元への戻り)』
- 703 ページの『WHEN (真の場合に選択)』

詳細については、特定の命令コードを参照してください。継続行のコーディングの詳細については、216 ページの『継続規則』 を参照してください。

フリー・フォーム構文

フリー・フォーム演算グループを開始するには、7 ～ 11 桁目に /FREE を指定し、12 ～ 80 桁目はブランクのままにします。フリー・フォーム演算ブロックは、/END-FREE を指定すると終了します。

フリー・フォームステートメントでは、命令コードを 8 – 80 桁の間の特定の位置で始める必要はありません。どの拡張機能も、同じ行の命令コードの直後に括弧で囲んで表す必要があります。命令コードと拡張機能の間にブランクが埋め込まれていてはなりません。命令コードと拡張機能に続いて、演算項目 1、演算項目 2、および結果フィールド・オペランドをブランクで区切って指定します。これらのいずれかが演算命令で必要とされない場合は、省略することができます。ステートメントの残りの部分では、ブランクおよび継続行を自由に使用することができます。各ステートメントはセミコロンで終わっていることが必要です。セミコロンの後のレコードの残りの部分は、ブランクであるか、あるいは行の終わりコメントが含まれていなければなりません。

EVAL または CALLP 命令コードの場合には、命令コードを省略することができます。たとえば、次の 2 つのステートメントは同等です:

```
eval pos = %scan (',': name);
pos = %scan (',': name);
```

フリー・フォーム演算ブロック内の各レコードの場合には、6 ～ 7 桁目はブランクでなければなりません。

フリー・フォーム演算ブロック内にはコンパイラー指示を指定できますが、以下の制限があります:

- コンパイラー指示は行の最初の項目であることが必要です。この指示のコーディングは 7 桁目より前の任意の位置から初めてください。次の行に継続することはできません。
- コンパイラー指示はステートメント内では許可されません。この指示は、1 つのステートメントが終わり、次のステートメントが始まる前の新規行に表す必要があります。
- /COPY または /INCLUDE 指示によって組み込まれたステートメントは、すべて固定構文演算と見なされます。コピー・メンバーの中のすべてのフリー・フォームステートメントは、/FREE および /END-FREE 指示によって区切る必要があります。

フリー・フォームオペランドは 14 文字より長くすることができます。以下のことはサポートされていません:

- 数値リテラルの継続
- フィールド名の定義
- 結果標識。(命令コードを結果標識とともに使用する必要がある多くの場合に、代わりに同等の組み込み関数を使用することができます。)

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/free

        read  file;              // 次のレコードを取り出します
        dow not %eof(file);      // レコードがある間はループ
                                 // を保持します
            if %error;
                dsply 'The read failed';
                leave;
            else;
                chain(n) name database data;
                time = hours * num_employees
                         + overtime_saved;
                pos = %scan (',': name);
                name = %xlate(upper:lower:name);
                exsr handle_record;
                read file;
            endif;
        enddo;

     begsr handle_record;
        eval(h) time = time + total_hours_array (empno);
        temp_hours = total_hours - excess_hours;
        record_transaction();
     endsr;

/end-free
```

図 104. フリー・フォーム演算仕様の例

以下に示されているように、フリー・フォーム演算仕様と従来からの演算仕様を結合することができます:

```
C                   testb     OPEN_ALL      flags                   10
 /free
          if *in10;
            openAllFiles();
          endif;
 /end-free
```

図 105. 従来からの演算仕様とフリー・フォーム演算仕様を結合した例

8-80 桁目 (フリー・フォーム演算命令)

サポートされる演算命令をフリー・フォーム構文で入力します。命令コード (EVAL および CALLP は任意選択) に続けて、オペランドまたは式をコーディングします。演算命令は複数行に任意にスパンできます。新規継続文字は必要ありません。各ステートメントはセミコロン (;) で終わります。ただし、この場合も既存の継続規則が適用されます。

継続行のコーディングの詳細については、216 ページの『継続規則』を参照してください。

フリー・フォーム構文を使用できる命令コードは以下にリストされています。フリー・フォーム構文を使用できない演算命令については、481 ページの『第 26 章 命令コードの詳細』の詳細な説明を調べて、推奨されている置き換えがあるかどうかを確認してください。

- 492 ページの『BEGSR (ユーザー・サブルーチンの開始)』
- 503 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』
- 510 ページの『CHAIN (ファイルからのランダム検索)』
- 520 ページの『CLEAR (消去)』
- 523 ページの『CLOSE (ファイルのクローズ)』
- 525 ページの『COMMIT (コミット)』
- 527 ページの『DEALLOC (ストレージの解放)』
- 532 ページの『DELETE (レコードの削除)』
- 537 ページの『DOU (実行の終了)』
- 541 ページの『DOW (実行の時期)』
- 544 ページの『DSPLY (「メッセージ」ウィンドウの表示)』
- 546 ページの『ELSE (その他)』
- 547 ページの『ELSEIF (Else If)』
- 548 ページの『ENDyy (構造化グループの終了)』
- 551 ページの『ENDSR (ユーザー・サブルーチンの終了)』
- 552 ページの『EVAL (式の評価)』
- 554 ページの『EVALR (式の評価、右寄せ)』
- 556 ページの『EXCEPT (演算時の出力)』
- 558 ページの『EXSR (ユーザー・サブルーチンの起動)』
- 562 ページの『FEOD (データの終わりの強制)』
- 563 ページの『FOR (For)』
- 569 ページの『IF (判断)』
- 572 ページの『IN (データ域の検索)』
- 574 ページの『ITER (繰り返し)』

- 579 ページの『LEAVE (DO/FOR グループの終了)』
- 581 ページの『LEAVESR (サブルーチンの終了)』
- 585 ページの『MONITOR (モニター・グループの開始)』
- 626 ページの『ON-ERROR (エラー処理)』
- 627 ページの『OPEN (処理用ファイルのオープン)』
- 630 ページの『OTHER (その他の場合の選択)』
- 632 ページの『OUT (データ域の書き出し)』
- 638 ページの『POST (転記)』
- 640 ページの『READ (レコードの読み取り)』
- 643 ページの『READC (次の変更レコードの読み取り)』
- 645 ページの『READE (等しいキーのレコードの読み取り)』
- 648 ページの『READP (前のレコードの読み取り)』
- 650 ページの『READPE (前の等しいキーのレコードの読み取り)』
- 656 ページの『RESET (リセット)』
- 659 ページの『RETURN (呼び出し元への戻り)』
- 660 ページの『ROLBK (ロールバック)』
- 664 ページの『SELECT (選択グループの始め)』
- 668 ページの『SETGT (より大きいレコードへのセット)』
- 671 ページの『SETLL (下限のセット)』
- 676 ページの『SORTA (配列の分類)』
- 690 ページの『TEST (日付、時刻、タイム・スタンプのテスト)』
- 699 ページの『UNLOCK (データ域のロック解除またはレコードの解放)』
- 701 ページの『UPDATE (既存レコードの変更)』
- 703 ページの『WHEN (真の場合に選択)』
- 707 ページの『WRITE (新しいレコードの作成)』

第 21 章 出力仕様

出力仕様は、プログラム記述出力ファイルの中のレコードおよび、フィールドの形式、およびレコードを書き込む時点を記述します。出力仕様は外部記述ファイルの場合は任意指定です。制御仕様に NOMAIN がコーディングされている場合は、例外出力だけを行なうことができます。

出力仕様は次の 2 つのカテゴリーに分けることができます: レコードの識別と制御 (7 桁目から 51 桁目)、およびフィールドの記述と制御 (21 桁目から 80 桁目)。これらの指定は出力仕様に入力されます。

出力仕様の詳細については次を参照してください:
- 324 ページの『プログラム記述ファイル』
- 335 ページの『外部記述ファイル』

出力ファイルには次の規則が適用されます。
- DISK ファイル:
  - DISK ファイルはリモートまたはローカルのいずれかにできます。
  - リモート・ファイルは外部記述でなければなりません。
  - ローカル・ファイルはプログラム記述でなければなりません。
- PRINTER ファイル:
  - PRINTER ファイルはプログラム記述でなければなりません。
- SPECIAL ファイル:
  - SPECIAL ファイルはプログラム記述でなければなりません。

出力仕様ステートメント

出力仕様の一般的なレイアウトは次の通りです:
- 出力仕様タイプ (O) は 6 桁目に入力されます
- 仕様の非コメント部分は 7 桁目から 80 桁目までです
- 仕様のコメント・セクションは 81 桁目から 100 桁目までです

プログラム記述

プログラム記述ファイルの場合は、出力仕様の記入項目は次の 2 つのカテゴリーに分けることができます :
- レコードの識別と制御 (7 桁目から 51 桁目)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
OFilename++EF..N01N02N03Excnam++++B++A++Sb+Sa+.............................Comment++++++++++++++
OFilename++EAddN01N02N03Excnam++++.....................................Comment++++++++++++++
O.........And..N01N02N03Excnam++++.....................................Comment++++++++++++++
```

図 106. プログラム記述レコード・レイアウト

- フィールドの記述と制御 (21 桁目から 80 桁目)。各フィールドは、その対応するレコード識別項目の下に、別々の行に記述されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++Comment++++++++++++++
O.........................................Constant/editword-ContinutioComment++++++++++++++
```

図 107. プログラム記述フィールド・レイアウト

外部記述

外部記述ファイルの場合は、出力仕様の記入項目は次のカテゴリーに分けられます:

- レコードの識別と制御 (7 桁目から 39 桁目)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
ORcdname+++E...N01N02N03Excnam++++.....................................Comment++++++++++++++
ORcdname+++EAddN01N02N03Excnam++++.....................................Comment++++++++++++++
O.........And..N01N02N03Excnam++++.....................................Comment++++++++++++++
```

図 108. 外部記述レコード・レイアウト

- フィールドの記述と制御 (21 桁目から 43 桁目、および 45 桁目)。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
O.............N01N02N03Field+++++++++.B.................................Comment++++++++++++++
```

図 109. 外部記述フィールド・レイアウト

プログラム記述ファイル

プログラム記述ファイルには、出力仕様における以下の記入項目が組み込まれています。

6 桁目 (仕様のタイプ)

この行を出力仕様ステートメントとして識別するためには、6 桁目に O が表示されていなければなりません。

レコードの識別および制御項目

7 桁目から 51 桁目までの記入項目は、ファイルを構成する出力レコードを識別し、印刷される報告上の正しいスペーシングを提供し、どんな条件でレコードが書き込まれるかを決定します。

7-16 桁目 (ファイル名)

項目　　説明

ファイル名

ファイルの出力レコードを定義する最初の行でファイル名を指定してください。指定するファイル名は、ファイル仕様で出力、更新、または入出力共用ファイルに割り当てられたファイル名と同じでなければなりません。ファイ

ルからのレコードが出力仕様上に散在している場合には、ファイルが変わるたびにファイル名を指定しなければなりません。

出力、更新 、 入出力共用 、または ADD での 入力 として指定されたファイルの場合は、結果フィールドに指定されているデータ構造名をもつ明示的なファイル命令コードが演算中で使用されていない限り、少なくとも 1つの出力仕様が必要です。たとえば、WRITE 演算には出力仕様は必要ありません。

16-18 桁目 (論理関係)

項目　説明

AND または OR

AND/OR は、出力標識の各行の間の関係を示します。AND/OR 行は出力レコードには有効ですが、フィールドには無効です。この関係を指定するためには、ファイル名を含む行の後に続く各追加行の 16 から 18桁目に AND/OR を入力してください。各 AND 行に少なくとも 1 つの標識を指定しなければなりません。出力仕様には AND/OR 行を無制限に指定することができます。

AND/OR を指定する時には、7 桁目から 15 桁目はブランクでなければなりません。

17 桁目 (タイプ - プログラム記述ファイル)

項目　説明

E　例外レコードは演算処理時に書き込まれます。例外レコードを指定できるのは、演算コード EXCEPT を使用する時だけです。EXCEPT演算コードの詳細については、 345 ページの『第 23 章 演算命令』を参照してください。

18-20 桁目 (レコードの追加／削除)

項目　説明

ADD　レコードを入力ファイル、出力ファイル、更新ファイル、またはサブファイルに追加します。ローカル・ファイルの場合は、すべてのレコードがファイルの終わりに追加されます。更新は現行レコードで行なわれます。

DEL　ファイルから読み取られた最後のレコードを削除します。削除されたレコードは検索できず、レコードはシステムから削除されます。

21-29 桁目 (ファイル・レコード ID 標識)

項目　説明

ブランク
出力用のレコードが検査されるたびに、行またはフィールドが出力されます。

01 ～ 99
演算結果標識、フィールド標識、またはレコード識別標識として使用される一般標識。

LR　最終レコード標識。

条件づけ標識は出力行には必要ありません。条件づけ標識を指定しない場合には、出力用のレコードが検査されるたびに行が出力されます。レコードまたはレコード内の特定のフィールドが書き込まれる時期を制御するために、1 つの仕様行に 3 つまでの標識を入力することができます。出力の条件づけをする標識は、22 および 23 桁目、25 および 26 桁目、そして 28 および 29 桁目にコーディングされます。 21、24、または 27 桁目に N が入力されている時は、書き込まれる行またはフィールドの対応する桁の標識をオフにしなければなりません。そうでない場合は、書き込まれる行またはフィールドの標識をオンにしなければなりません。出力標識が PAGE フィールドに及ぼす影響については、 330 ページの『PAGE, PAGE1-PAGE7』 を参照してください。

1 つの行に複数の標識が指定されている場合には、すべての標識が AND 関係にあると見なされます。

出力レコードを AND 関係で 4 つ以上の標識で条件づけなければならない場合には、後に続く行の 16 から 18 桁目に文字 AND を入力し、その行の 21 から 29 桁目に追加の標識を指定してください。

すべての AND 行の 40 から 51 桁目 (スペーシングおよびスキップ) はブランクでなければなりません。

2 つまたはそれ以上の条件のセットのいずれか 1 つが存在する時に出力レコードが書き込まれるようになっている場合 (OR 関係) には、後に続く仕様行の 16 - 18 桁目に文字 OR を入力し、その行に追加の OR 標識を指定してください。

印刷装置ファイルに OR 行が指定されている場合には、スキップおよびスペース記入項目 (40 から 51 桁目) はすべてブランクにすることができ、この場合には、先

行行のスペースおよびスキップ記入項目が使用されます。これらが先行行とは異なっている場合には、スペースおよびスキップ記入項目を OR 行に入力してください。

30 ～ 39 桁目 (EXCEPT 名)

レコード・タイプが例外レコード (17 桁目の E で示される) である場合には、レコード行のこれらの桁に名前を入れることができます。EXCEPT 演算は、出力されるレコードのグループに割り当てられる名前を指定することができます。この名前は EXCEPT 名と呼ばれます。EXCEPT 名は記号名の使用規則に従わなければなりません。任意の数の出力レコードのグループは同じ EXCEPT 名を使用することができ、レコードは連続したレコードである必要はありません。

EXCEPT 名なしで EXCEPT 演算が指定されている時には、条件づけ標識が満たされた場合に、EXCEPT 名のない例外レコードだけが検査され、書き込まれます。

EXCEPT 演算が EXCEPT 名を指定している時には、条件づけ標識が満たされた場合に、その名前をもつ例外レコードだけが検査され、書き込まれます。

EXCEPT 名がメイン・レコード行に指定され、すべての AND/OR 行に適用されます。

ファイル中のレコード・ロックを解放するためには、フィールドのない EXCEPT 演算を使用することができます。この目的には、UNLOCK 演算を使用することもできます。次の図では、ファイル RCDAの中のレコード・ロックが EXCEPT 演算によって解放されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C     KEY           CHAIN     RCDA
C                   EXCEPT    RELESE
ORcdname+++D...N01N02N03Excnam++++.....................................
O
O*
ORCDA      E            RELESE
O*                     (フィールドなし)
```

図 110. EXCEPT 演算によって解放されるファイル中のレコード・ロック

40 ～ 51 桁目 (スペースおよびスキップ)

印刷装置ファイルの行送りおよびスキップを指定するためには、40 から 51 桁目を使用してください。スペーシングとは一度に 1 行進むことであり、スキップは 1 つの印刷行から別の印刷行にジャンプすることです。

同じ行にスペーシングとスキップが指定されている場合には、スペーシング操作とスキップ操作は次の順序で処理されます:

1. 前スキップ
2. 印刷前行送り
3. 行を印刷

4. 後スキップ
5. 印刷後行送り

ファイル仕様に PRTCTL (印刷装置制御オプション) キーワードが指定されていない場合には、装置が PRINTER である時には、40 ～ 42 桁目 (印刷前行送り)、43 ～ 45 桁目 (印刷後行送り)、46 ～ 48 桁目 (前スキップ)、または 49 ～ 51 桁目 (後スキップ) のいずれかに入力しなければなりません。スペース / スキップ記入項目をブランクのままにすると、ブランク項目 (印刷前行送りまたは印刷後行送りなど) での特定の機能が実行されません。40 ～ 42 桁目 (印刷前行送り) または 46 ～ 51 桁目 (前スキップおよび後スキップ) に入力し、43 ～ 45 (印刷後行送り) に入力しない場合には、印刷後にスペースが入りません。PRTCTL が指定されている時は、それは 40 から 51 桁目に指定されているブランクのレコードでのみ使用されます。

新規ページで行の前スキップまたは後スキップが指定されていても、印刷装置がその行に位置している場合には、スキップは行なわれません。

40 ～ 42 桁目 (印刷前行送り)

項目 **説明**

0 またはブランク
スペーシングなし

1 ～ 255 スペーシング値

43 ～ 45 桁目 (印刷後行送り)

項目 **説明**

0 またはブランク
スペーシングなし

1 ～ 255 スペーシング値

46 ～ 48 桁目 (前スキップ)

項目 **説明**

ブランク
スキップは行なわれません。

1 ～ 255
スキップ値

49 ～ 51 桁目 (後スキップ)

項目 **説明**

ブランク
スキップは行なわれません。

1 ～ 255
スキップ値

フィールド記述および制御項目

各フィールドは別々の行で記述されます。これらの記入項目は、どんな条件で、どんな形式で、レコードのフィールドが書き込まれるかを決定します。フィールドのフィールド記述および制御情報は、レコード識別行の後に続く行から始まります。

21 ～ 29 桁目 (出力標識)

フィールド記述行に指定された標識は、PAGE 予約フィールドの場合を除いて、フィールドが出力レコードに組み込まれるかどうかを決定します。出力標識が PAGE フィールドに及ぼす影響については、 330 ページの『PAGE, PAGE1-PAGE7』 を参照してください。レコードを制御するために使用されるように、フィールドを制御するために同じタイプの標識を使用することができます。 326 ページの『21-29 桁目 (ファイル・レコード ID 標識)』を参照してください。フィールド記述行の条件づけをするために使用される標識は、AND/OR 関係には指定できません。

30 ～ 43 桁目 (フィールド名)

30 から 43 桁目で、次の項目の 1 つを使用して、書き出される各フィールドを指定してください:

- フィールド名
- 53 ～ 80 桁目に固定情報が指定されている場合はブランク
- テーブル名、配列名、または配列エレメント
- 名前付き固定情報
- 予約語 PAGE、PAGE1 から PAGE7、*PLACE、UDATE、*DATE、 UDAY、*DAY、UMONTH、*MONTH、UYEAR、*YEAR、*IN、*INxx、または *IN(xx)
- データ構造名またはデータ構造サブフィールド名

注: ポインター・フィールドは有効な出力フィールドではありません。すなわち、ポインター・フィールドには書き込みができません。

フィールド名、ブランク、テーブル、および配列

使用するフィールド名はプログラムに定義されていなければなりません。53 - 80 桁目に固定情報が使用されている場合には、フィールド名を入力しないでください。30 から 43 桁目にフィールド名を入力する場合には、7 から 20 桁目はブランクとしなければなりません。

フィールドは任意の順序で指定することができます。というのは、出力レコード上でのそれらの順序は、 47 から 51 桁目の記入項目によって決定されるからです。フィールドがオーバーラップしている場合には、指定された最後のフィールドだけが完全に書き込まれます。

索引の付いていない配列名が指定されると、その配列全体が書き込まれます。固定情報索引または可変索引が付いている配列名の場合は、1 つのエレメントが書き込まれます。テーブル名を指定すると、LOOKUP 演算で最後に検出されたエレメントが書き込まれます。LOOKUP 演算が失敗した場合には、テーブルの最初のエレメントが書き込まれます。

フィールドが書き出される前に、レコードおよびそれに含まれているフィールドの条件が満たされていなければなりません。

PAGE, PAGE1-PAGE7

自動ページ番号付けを使用するためには、30 から 43 桁目に出力フィールドの名前として PAGE を入力してください。21 から 29桁目に指定されている標識は、PAGE フィールドのリセットの条件づけをするのであって、印刷するかどうかの条件づけではありません。PAGE フィールドは常に 1 だけ増分されて印刷されます。条件づけ標識が満たされた場合には、このフィールドは 1 だけ増分されて印刷される前に、ゼロにリセットされます。複数の出力ファイルにページ番号 (または 1 つのファイル内で異なる番号) が必要な場合には、記入項目 PAGE1 から PAGE7 までを使用することができます。PAGE フィールドは、Z 編集コードによって自動的にゼロが禁止されます。

PAGE 予約語の詳細については、 5 ページの『特殊な機能を持つ語および予約語』を参照してください。

*PLACE

*PLACE は出力レコード内でデータを反復するために使用されます。前の仕様行に指定されているフィールドまたは固定情報は、新規の仕様行にフィールドおよび終了桁を指定しなくても、出力レコード内で反復することができます。30 から 43 桁目に *PLACE が入力されている時には、その出力レコード内のフィールドに前に指定されている最初の桁と最高位の終了桁との間のすべてのデータが、 *PLACE 仕様行の出力レコードに指定されている終了桁に達するまで反復されます。 *PLACE 仕様行に指定する終了桁は、少なくとも、複写されるフィールドのグループの最高位の終了桁の 2 倍でなければなりません。*PLACE は任意のタイプの出力で使用できます。後で消去 (45 桁目)、編集 (44 桁目、53 から 80 桁目)、データ形式 (52 桁目)、および相対終了桁は、*PLACE と一緒に使用できません。

ユーザー日付予約語

UDATE、*DATE、UDAY、*DAY、UMONTH、*MONTH、UYEAR、および *YEAR によって、ユーザーは実行時にプログラムの日付を提供することができます。ユーザー日付語の詳細については、 8 ページの『ユーザー日付の特殊語』 を参照してください。

*IN, *INxx, *IN(xx)

*IN、*INxx、および *IN(xx) によって、ユーザーは標識をデータとして参照したり操作することができます。

44 桁目 (編集コード)

項目	説明
ブランク	編集コードは使用されません。
1-4, A-D, J-Q, X, Y, Z	数値フィールドは編集語を使用しないで、事前定義されたパターンに応じてゼロが禁止され、句読点が付けられます。

44 桁目は、数値フィールドの中で先行ゼロの使用を禁止する編集コードを指定したり、あるいは編集語を使用しないで数値フィールドの句読点を付けるために使用されます。使用できる記入項目は 1 から 4、Aから D、J から Q、X、Y、Z、およびブランクです。

45 桁目 (後で消去)

項目　説明

ブランク
フィールドはリセットされません。この桁は、先読み、ユーザー日付予約語、*PLACE、名前付き固定情報、およびリテラルの場合はブランクでなければなりません。

B　30 から 43 桁目に指定されたフィールドは、出力操作の完了後に、ブランク、ゼロ、またはデフォルトの日付 / 時刻 / タイム・スタンプの値にリセットされます。

45 桁目は、数値フィールドをゼロにリセットするか、あるいは文字、グラフィック、または UCS-2 フィールドをブランクにリセットするために使用されます。日付、時刻、タイム・スタンプ・フィールドはそれらのデフォルト値にリセットされます。

フィールドが 21 から 29 桁目の標識によって条件づけされる場合には、「後で消去」も条件づけされます。

複数回書き込まれるフィールドに対して、「後で消去」 (45 桁目) が指定されている場合には、そのフィールドの出力を指定する最後の行に B を入力しなければなりません。そうしないと、「後で消去」を実行する行の後のすべての行について、指定されたフィールドが「後で消去」の値として印刷されます。

47 ～ 51 桁目 (終了桁)

項目	説明
1-n	終了桁

47 から 51 桁目はフィールドの終了桁または出力レコードの固定情報を定義します。

終了桁の有効な記入項目は、ブランク、+nnnn、–nnnn、および nnnnn です。これらの桁の中のすべての記入項目は 51 桁目で終了しなければなりません。フィールドまたは固定情報の右端の文字の桁を入力してください。終了桁がファイルのレコード長を超えてはいけません。

配列全体を書き込む場合には、配列の最後のエレメントの終了桁を 47 から 51 桁目に入力してください。配列を編集することにしている場合には、編集されたすべてのエレメントが十分な桁数に書き込まれるように、終了桁の指定時に注意してください。各エレメントは編集コードまたは編集語に応じて編集されます。

+nnnn または –nnnn 記入項目は、前のフィールドの終了桁に対応するフィールドまたは固定情報の配置を指定します。数値 (nnnn) は右寄せしなければなりませんが、先行ゼロは必要ありません。入力フィールド内の数値の左側の任意の場所に符号を入力してください。終了桁を計算するためには、次の数式を使用してください:

```
終了桁 = 前の終了桁 +nnnn + フィールド長
終了桁 = 前の終了桁 -nnnn + フィールド長
```

レコード中の最初のフィールド仕様の場合は、前の終了桁はゼロとなります。フィールド長は編集後のフィールドの長さであるか、あるいはこの仕様に指定されている固定情報の長さとなります。 +nnnn を使用することは、フィールド間に nnnn 桁を入れるのと同じことです。-nnnn を使用すると、フィールドが nnnn 桁だけオーバーラップします。たとえば、前の終了桁が 6 で、フィールド (nnnn) 間に入れる桁数が 5で、フィールド長が 10 である場合には、終了桁は 21 となります。

*PLACE を使用する時には、実際の終了桁を指定しなければなりません; これはブランクまたは変位ではいけません。

ブランクを入力すると +0000 が入力されたものと見なされます。桁によってフィールドが分離されません。

52 桁目 (データ形式)

項目　説明

ブランク

編集が指定されている場合は、この桁はブランクでなければなりません。
- 数値フィールドの場合は、データはゾーン 10 進形式で書き込まれます。
- 浮動数値フィールドの場合は、データは外部表示表現で書き込まれます。
- UCS-2 フィールドの場合は、データは UCS-2 形式で書き込まれます。
- 日付、時刻、タイム・スタンプ・フィールドの場合は、データは形式変換を実行しないで書き込まれます。
- 文字フィールドの場合は、データは保管されたままの状態で書き込まれます。

A　文字フィールドの場合にのみ有効。文字フィールドは、*VAR データ属性の有無によって、固定長または可変長形式のいずれかで書き込まれます。

N　文字フィールドは標識形式で書き込まれます。

C　UCS-2 フィールドは、*VAR データ属性の有無によって、固定長または可変長形式のいずれかで書き込まれます。

G　プログラム記述ファイル中のグラフィック・フィールドの場合にのみ有効。グラフィック・フィールドは、*VAR データ属性の有無によって、固定長または可変長形式のいずれかで書き込まれます。

B　数値フィールドは 2 進数形式で書き込まれます。

F　数値フィールドは、浮動形式で書き込まれます。

I　数値フィールドは、整数形式で書き出されます。

L　数値フィールドは、先行する (左の) 正符号または負符号と一緒に、ゾーン 10 進数形式で書き込まれます。

P　数値フィールドは、パック 10 進数形式で書き込まれます。

R　数値フィールドは、後書きの (右の) 正符号または負符号と一緒に、ゾーン 10 進数形式で書き込まれます。

S　数値フィールドは、ゾーン 10 進形式で書き出されます。

U　数値フィールドは、符号なし整数形式で書き出されます。

D　日付フィールド– 日付フィールドは、53 - 80 桁目に指定されている形式、またはデフォルトのファイル日付形式に変換されます。

T　時刻フィールド– 時刻フィールドは、53 - 80 桁目に指定されている形式、またはデフォルトのファイル時刻形式に変換されます。

Z　タイム・スタンプ・フィールドの場合にのみ有効。

52 桁目の記入項目は、ファイル中のレコードの中のデータの外部形式を指定します。この記入項目は、プログラム中の出力フィールドの内部処理に使用される形式には影響しません。

数値フィールドの場合は、出力レコードで必要なバイト数はこの形式によって異なります。たとえば、5桁の数値フィールドは次のバイト数が必要です:
- ゾーン形式で書き込まれる場合は 5 バイト

- パック形式で書き込まれる場合は 3 バイト
- L または R 形式で書き込まれる場合は 6 バイト
- 2 進数形式で書き込まれる場合は 4 バイト
- I または U 形式で書き込まれる場合は 2 バイト。値が 2 バイト整数フィールドまたは符号なしフィールドの最大値より大きい場合は、これによって実行時にエラーが起こることがあります。5 桁のフィールドの場合は、2 進数形式の方が良い場合があります。

  ブランクのデータ形式項目で書き出される浮動数値フィールドは、出力レコード中で 14 または 23 桁を占めます (それぞれ 4 バイトおよび 8 バイト浮動フィールドの場合)。

注: プログラム記述ファイル中のグラフィック・フィールドの場合は、'G' またはブランクを指定しなければなりません。

53 ～ 80 桁目 (固定情報、編集語、データ属性)

53 から 80 桁目は、固定情報、編集語、またはデータ属性 を指定するために使用されます。

固定情報

固定情報は、プログラムの 1 つの処理から次の処理に変更されない文字データ (リテラル) から成っています。固定情報は、データの位置を表す名前ではなく、出力レコード内で使用される実際のデータです。

固定情報は 53 から 80 桁目に入れることができます。固定情報は 54 桁目から始まり (53 桁目はアポストロフィ)、数字しか入っていない場合でも、アポストロフィで終わらなければなりません。固定情報内で使用するアポストロフィは 2 回入力しなければなりません。しかし、固定情報が書き出される時は 1 つのアポストロフィだけが表示されます。フィールド名 (30 から 43 桁目) はブランクでなければなりません。固定情報は継続することができます。(継続規則については、 216 ページの『継続規則』 を参照してください。)固定情報を入力する代わりに、名前付き固定情報を使用することができます。

グラフィック および UCS-2 リテラルまたは名前付き固定情報は、編集語としては使用できませんが、固定情報としては指定できます。

編集語

編集語は、数値フィールドの句読点を指定します (円記号、コンマ、ピリオド、および符号の状況の印刷を含む)。詳細については、 200 ページの『編集語の部分』 を参照してください。

編集語は文字リテラルまたは名前付き固定情報でなければなりません。グラフィック、 UCS-2 または16 進リテラルまたは名前付き固定情報は使用できません。

データ属性

データ属性は、日付、時刻、または可変長文字、グラフィック、または UCS-2 フィールドの外部形式を指定します。

日付および時刻データについては、日付または時刻形式が指定されていない場合には、ファイルに指定されている形式 / 区切り記号 (DATFMT または TIMFMT あるいは両方) が使用されます。ファイルに外部日付または時刻形式が指定されていない場合には、エラー・メッセージが出されます。日付および時刻形式については、227 ページの『DATFMT(fmt{区切り文字})』 および 235 ページの『TIMFMT(fmt{区切り文字})』 を参照してください。

外部日付/時刻形式および日付と時刻フィールドの区切り記号を決定する時に使用される階層は次の通りです:
1. 53 ～ 58 (または 53 ～ 57) 桁目に指定されている日付形式および区切り記号。
2. 現行ファイルに指定されている DATFMT/TIMFMT キーワードから
3. 制御仕様に指定されている DATFMT/TIMFMT キーワードから
4. *ISO

日付と時刻フィールドは、その内部日付/時刻形式から、上記で決定された外部形式に変換されます。

文字、グラフィック、および UCS-2 データの場合は、*VAR データ属性を使用して可変長出力フィールドを指定されます。文字、グラフィック、および UCS-2 データのこの記入項目がブランクである場合には、外部形式は固定長となります。可変長フィールドの詳細については、118 ページの『可変長の文字形式、グラフィック形式、および UCS-2 形式』 を参照してください。

注: 出力レコード内で占有されるバイト数は、指定された形式によって異なります。たとえば、*MDY 形式で書かれた日付は 8 バイトを必要とし、*ISO 形式で書かれた日付は 10 バイトを必要とします。

外部形式の詳細については、107 ページの『内部形式と外部形式』 を参照してください。

外部記述ファイル

外部記述ファイルには、入力仕様における以下の記入項目が組み込まれています。

6 桁目 (仕様のタイプ)

この行を出力仕様ステートメントとして識別するためには、6 桁目に O が表示されていなければなりません。

レコードの識別および制御項目

外部記述ファイルの出力仕様は任意指定です。 レコード識別行の 7 桁目から 39 桁目の記入項目は、レコード様式を識別し、どんな条件でレコードが書き込まれるかを決定します。

7 ～ 16 桁目 (レコード名)

項目	説明
有効なレコード様式名	外部記述ファイルにはレコード様式名を指定しなければなりません。

16 ～ 18 桁目 (外部論理関係)

項目	説明
AND または OR	AND/OR は、出力標識の行の間の関係を示します。 AND/OR 行は出力レコードには有効ですが、フィールドには無効です。

詳しくは、 325 ページの『16-18 桁目 (論理関係)』を参照してください。

17 桁目 (タイプ)

項目	説明
E	例外レコード。

17 桁目は、書き込まれるレコードのタイプを示します。

詳しくは、 325 ページの『17 桁目 (タイプ - プログラム記述ファイル)』を参照してください。

18 ～ 20 桁目 (レコードの追加)

項目	説明
ADD	レコードをファイルに追加します。
DEL	ファイルから既存のレコードを削除します。

21 ～ 29 桁目 (出力標識)

外部記述ファイルの出力標識は、プログラム記述ファイルの場合と同じ方法で指定します。出力標識の詳細については、 326 ページの『21-29 桁目 (ファイル・レコード ID 標識)』 を参照してください。

30 ～ 39 桁目 (EXCEPT 名)

例外レコード行の EXCEPT 名は、これらの桁に指定することができます。詳しくは、 327 ページの『30 ～ 39 桁目 (EXCEPT 名)』を参照してください。

フィールド記述および制御項目

外部記述ファイルの場合は、有効なフィールド記述は、出力標識 (21 から 29 桁目)、フィールド名 (30から 43 桁目)、そして後で消去 (45 桁目) だけです。

21 ～ 29 桁目 (出力標識)

フィールド記述行に指定された標識は、フィールドが出力レコードに組み込まれるかどうかを決定します。フィールドを制御するためには、レコードを制御するために使用されるものと同じタイプの標識を使用することができます。詳しくは、 326 ページの『21-29 桁目 (ファイル・レコード ID 標識)』を参照してください。

30 ～ 43 桁目 (フィールド名)

項目	説明
有効なフィールド名	外部記述ファイルに指定するフィールド名は、プログラムの外部名が名前変更されていない限り、外部記述の中に存在していなければなりません。
***ALL**	レコード中に含まれているすべてのフィールドを指定します。

外部記述ファイルの場合は、指定されたフィールドだけが出力レコードに入れられます。レコード中のすべてのフィールドを組み込むためには、 *ALL を指定することができます。 *ALL を指定する場合には、そのレコードの他のフィールド記述行を指定することはできません。特に、45 桁目に B (後で消去) を指定することはできません。

更新レコードの場合は、出力フィールド仕様に指定されているフィールドおよび出力標識によって指定された条件に適合するフィールドだけが、再書き込みされる出力レコードに入れられます。読み取られた値は、他のすべてのフィールドを再書き込みするために使用されます。

新規レコードの作成 (18 ～ 20 桁目に ADD を指定) の場合は、指定されたフィールドが出力レコードに入れられます。指定されていないフィールド、または出力標識によって指定された条件に適合しないフィールドは、外部記述に指定されているデータ形式に応じて、ゼロまたはブランクとして書き込まれます。

45 桁目 (後で消去)

項目	説明
ブランク	フィールドはリセットされません。
B	30 から 43 桁目に指定されたフィールドは、出力操作の完了後に、ブランク、ゼロ、またはデフォルトの日付 / 時刻 / タイム・スタンプの値にリセットされます。

45 桁目は、数値フィールドをゼロにリセットするか、あるいは文字、グラフィック、または UCS-2 フィールドをブランクにリセットするために使用されます。日付、時刻、タイム・スタンプ・フィールドはそれらのデフォルト値にリセットされます。

フィールドが 21 から 29 桁目の標識によって条件づけされる場合には、「後で消去」も条件づけされます。この桁は、先読み、ユーザー日付予約語、*PLACE、名前付き固定情報、およびリテラルの場合はブランクでなければなりません。

複数回書き込まれるフィールドに対して、「後で消去」 (45 桁目) が指定されている場合には、そのフィールドの出力を指定する最後の行に B を入力しなければなりません。そうしないと、「後で消去」を実行する行の後のすべての行について、指定されたフィールドが「後で消去」の値として印刷されます。

第 22 章 プロシージャー仕様

プロシージャー仕様は、メイン・ソース・セクションの後に指定されるプロトタイプ化されたプロシージャーを定義するために使用されます。そうでない場合は、サブプロシージャーとして知られています。

サブプロシージャーのプロトタイプは、サブプロシージャー定義が入っているモジュールのメイン・ソース・セクションに定義しなければなりません。サブプロシージャーには次のものがあります:

1. 開始プロシージャー仕様 (プロシージャー仕様の 24 桁目に B)
2. プロシージャー・インターフェース定義。これは戻り値およびパラメーターがある場合に指定します。サブプロシージャーが値を戻さず、また、渡されるパラメーターがない場合には、プロシージャー・インターフェース定義は任意指定となります。プロシージャー・インターフェースは対応するプロトタイプと一致していなければなりません。
3. サブプロシージャーに必要な変数、固定情報、およびプロトタイプのその他の定義仕様。これらの定義はローカル定義です。
4. プロシージャーのタスクを実行するのに必要な演算仕様。サブプロシージャー内に組み込まれているサブルーチンはローカルです。これらはサブプロシージャーの外側では使用できません。サブプロシージャーが値を戻す場合には、サブプロシージャーの中に RETURN演算がコーディングされていなければなりません。プロシージャーの終わりに達する前に RETURN 演算が実行されるようにしなければなりません。
5. 終了プロシージャー仕様 (プロシージャー仕様の 24 桁目に E)

定義仕様の中に入れられることのあるプロシージャー・インターフェース定義の場合を除いて、サブプロシージャーは上記の順序でコーディングされていなければなりません。

メイン・ソース・セクションの構造および定義の配置が有効範囲に及ぼす影響の詳細については、256 ページの『定義の配置と有効範囲』を参照してください。サブプロシージャーおよびプロトタイプ化については、65 ページの『第 6 章 サブプロシージャーおよびプロトタイプ』を参照してください。

プロシージャー仕様ステートメント

プロシージャー仕様の一般的なレイアウトは次の通りです:

- プロシージャー仕様タイプ (P) は 6 桁目に入力されます
- 仕様の非コメント部分は 7 桁目から 80 桁目までです:
  - 固定形式の記入項目は 7 桁目から 24 桁目です
  - キーワード記入項目は 44 ～ 80 桁目に拡張されます。
- 仕様のコメント・セクションは 81 桁目から 100 桁目までです

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10

PName++++++++++..B..................Keywords++++++++++++++++++++++++++++++Comments++++++++++++
```

図 111. プロシージャー仕様レイアウト

プロシージャー仕様キーワード継続行

キーワードのための追加のスペースが必要な場合には、次のように、キーワード・フィールドを後続の行に続けることができます :

- 継続行の 6 桁目に P を入れます
- 継続行の 7 から 43 桁目をブランクにします
- 仕様は 44 桁目以降に継続します

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
P.....................................Keywords++++++++++++++++++++++++++++++Comments++++++++++++
```

図 112. プロシージャー仕様キーワード継続行レイアウト

プロシージャー仕様の継続名前行

15 文字までの長さの名前を、継続の必要のないプロシージャー仕様の名前記入項目に指定することができます。 部分名の終わりに省略記号 (...) をコーディングすることによって、任意の名前 (15 文字またはそれ以下のものでも) を複数の行に継続することができます。

名前定義は以下のパーツから成り立っています:

1. ゼロまたはそれ以上の継続名前行。継続名前行は、項目中の最後の非ブランク文字として省略記号をもつことによって識別されます。名前は 7 - 21 桁目で始まり、最大 77 桁目までの任意の位置で終わることができます (80 桁目で終わる省略記号を指定して)。 名前の始めと省略記号 (...) 文字の間にブランクがあってはいけません。 これらの条件のいずれかが適合しない場合には、その行はメイン定義行として解析されます。
2. 名前、定義属性、およびキーワードが入っている 1 つのメイン定義行。継続名前行をコーディングする場合には、メイン定義行の名前記入項目をブランクにすることができます。
3. ゼロまたはそれ以上のキーワード継続行。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
DContinuedName++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++Comments++++++++++++
```

図 113. プロシージャー仕様の継続名前行

6 桁目 (仕様のタイプ)

プロシージャー仕様のこの桁に P を入力します。

7-21 桁目 (名前)

項目 **説明**

名前 定義するプロシージャーの名前。

定義されるサブプロシージャーの名前を指定するには、7-21 桁目を使用します。この名前が 15 文字より長い場合には、継続名前行の 7 - 80 桁目に名前が指定されます。 名前は提供されたスペースの中の任意の桁から始まることができます。

指定する名前は、プロシージャーを記述するプロトタイプの名前と同じでなければなりません。 24 桁目に E が入っている場合には、その名前は任意指定です。

24 桁目 (開始/終了プロシージャー)

項目 **説明**

B この仕様は、定義されるサブプロシージャーの先頭にマーク付けします。

E 仕様は、定義されるサブプロシージャーの終わりにマーク付けします。

サブプロシージャーのコーディングは、最小限でも、開始プロシージャー仕様と終了プロシージャー仕様から構成されます。 サブプロシージャーの他の定義および演算とともに、パラメーターと戻り値が、プロシージャー仕様の間に指定されます。

44 ～ 80 桁目 (キーワード)

44 から 80 桁目はプロシージャー仕様キーワード用に提供されています。 開始プロシージャー仕様 (24 桁目に B) だけがキーワード記入項目をもつことができます。

プロシージャー仕様キーワード

プロシージャー仕様は現在 『EXPORT』 が可能です。

EXPORT

EXPORT キーワードの指定によって、プロシージャーを NOMAIN DLL からエクスポートすることができます。 7-21 桁目の名前が大文字の形式でエクスポートされます。

EXPORT キーワードを指定しない場合には、プロシージャーはモジュール内からしか呼び出せません。

第 4 部 操作、式、および機能

この項では、データまたは装置の取扱方法を説明します。主なトピックには、次が含まれています。

- 345 ページの『第 23 章 演算命令』 は関数別にグループに分けられた命令コードの概要を提供します。
- 383 ページの『第 24 章 式』 は式とその管理規則を説明します。
- 399 ページの『第 25 章 組み込み関数』 は組み込み関数と定義でのその使用法および演算仕様を説明します。
- 481 ページの『第 26 章 命令コードの詳細』 はそれぞれの命令コードを詳細に説明します。

第 23 章 演算命令

VisualAge RPG プログラム言語によって、ユーザーのデータで多様な種類の演算を実行することができます。演算を実行するには、命令コードか組み込み関数のいずれかを使用します。

この章では、使用可能な命令コードおよび組み込み関数を要約してあります。また命令コードおよび組み込み関数をカテゴリー別に編成しています。

特定の命令コードまたは組み込み関数の詳細については、481 ページの『第 26 章 命令コードの詳細』 または 399 ページの『第 25 章 組み込み関数』 を参照してください。

命令コード

以下の表には、各命令コードのフリー・フォーム構文を示しています。

- 拡張
  - **(D)** 日付フィールド。
  - **(E)** エラー処理。
  - **(H)** 四捨五入し (数値演算の結果を丸め) ます。
  - **(M)** デフォルト精度規則。
  - **(N)** レコードをロックしません。
  - **(N)** 正常な DEALLOC の後で、ポインターが *NULL に設定されます。
  - **(P)** ブランクまたはゼロを結果に埋め込みます。
  - **(R)** ″結果の小数点以下の桁数″ 精度規則。
  - **(T)** 時刻フィールド。
  - **(Z)** タイム・スタンプ・フィールド

表 32. フリー・フォーム構文での命令コード

コード	フリー・フォーム構文
BEGACT	BEGACT *action-subroutine-name*
BEGSR	BEGSR *subroutine-name*
CALLP	{CALLP{(EMR)}} *name*({*parm1*{:*parm2*...}})
CHAIN	CHAIN{(EN)} *search-arg name* {*data-structure*}
CLEAR	CLEAR {*NOKEY} {*ALL} *name*
CLOSE	CLOSE{(E)} *file-name*
COMMIT	COMMIT{(E)}
DEALLOC	DEALLOC{(EN)} *pointer-name*
DELETE	DELETE{(E)} {*search-arg*} *name*
DOU	DOU{(MR)} *indicator-expression*
DOW	DOW{(MR)} *indicator-expression*
DSPLY	DSPLY{(E)} {*message* {*message-window-definition-name* {*response*}}}
ELSE	ELSE
ELSEIF	ELSEIF{(MR)} *indicator-expression*

表 32. フリー・フォーム構文での命令コード (続き)

コード	フリー・フォーム構文
ENDACT	ENDACT
ENDDO	ENDDO
ENDFOR	ENDFOR
ENDIF	ENDIF
ENDMON	ENDMON
ENDSL	ENDSL
ENDSR	ENDSR {*return-point*}
EVAL	{EVAL{(HMR)}} *result* = *expression*
EVALR	EVALR{(MR)} *result* = *expression*
EXCEPT	EXCEPT {*except-name*}
EXSR	EXSR *subroutine-name*
FEOD	FEOD{(E)} *file-name*
FOR	FOR{(MR)} *index* {= *start*} {BY *increment*} {TO|DOWNTO *limit*}
IF	IF{(MR)} *indicator-expression*
IN	IN{(E)} {*LOCK} *data-area-name*
ITER	ITER
LEAVE	LEAVE
LEAVESR	LEAVESR
MONITOR	MONITOR
ON-ERROR	ON-ERROR {*exception-id1* {:*exception-id2...*}}
OPEN	OPEN{(E)} *file-name*
OTHER	OTHER
OUT	OUT{(E)} {*LOCK} *data-area-name*
POST	POST{(E)} *file-name*
READ	READ{(EN)} *name* {*data-structure*}
READC	READC{(E)} *subfile-name*
READE	READE{(EN)} *search-arg*|*KEY *name* {*data-structure*}
READP	READP{(EN)} *name* {*data-structure*}
READPE	READPE{(EN)} *search-arg*|*KEY *name* {*data-structure*}
RESET	RESET{(E)} {*NOKEY} {*ALL} *name*
RETURN	RETURN{(HMR)} *expression*
ROLBK	ROLBK{(E)}
SELECT	SELECT
SETGT	SETGT{(E)} *search-arg name*
SETLL	SETLL{(E)} *search-arg name*
SORTA	SORTA *array-name*
TEST	TEST{(EDTZ)} {*dtz-format*} *field-name*
UNLOCK	UNLOCK{(E)} *name*
UPDATE	UPDATE{(E)} *name* {*data-straucture*}
WHEN	WHEN{(MR)} *indicator-expression*

表 32. フリー・フォーム構文での命令コード (続き)

コード	フリー・フォーム構文
WRITE	WRITE{(E)} *name* {*data-structure*}

次のテーブルは、通常構文による各命令コードの仕様の要約です。

- 空の桁は、フィールドをブランクにしなければならないことを示しています。
- 下線フィールドはすべて必須です。
- 下線スペースは、その位置に演算結果標識がないことを示しています。
- 記号:
  - **+** ＋
  - **–** -
- 拡張:
  - **(D)** 日付フィールド。
  - **(E)** エラー処理。
  - **(H)** 四捨五入し (数値演算の結果を丸め) ます。
  - **(M)** デフォルト精度規則。
  - **(N)** レコードをロックしません。
  - **(P)** ブランクまたはゼロを結果に埋め込みます。
  - **(R)** ″結果の小数点以下の桁数″ 精度規則。
  - **(T)** 時刻フィールド。
  - **(Z)** タイム・スタンプ・フィールド
- 演算結果標識の記号:
  - **BL** ブランク
  - **BN** ブランクの後ろに数字
  - **BOF** ファイルの先頭
  - **EOF** ファイルの終わり
  - **EQ** 等しい
  - **ER** エラー
  - **FD** 見つかりました
  - **HI** より大
  - **IN** 標識
  - **LO** より小
  - **LR** 最終レコード
  - **NR** レコードが見つかりませんでした
  - **NU** 数値
  - **OF** オフ
  - **ON** オン
  - **Z** ゼロ
  - **ZB** ゼロまたはブランク

表 33. 通常構文での命令コード

コード	演算項目 1	演算項目 2	結果フィールド	演算結果標識		
				71-72	73-74	75-76
ADD (H)	加数	加数	和	+	–	Z
ADDDUR (E)	日付 / 時刻	期間:期間コード	日付 / 時刻		ER	
ALLOC (E)		長さ	ポインター		ER	
ANDxx	被比較数	被比較数				

表 33. 通常構文での命令コード (続き)

コード	演算項目 1	演算項目 2	結果フィールド	演算結果標識		
				71-72	73-74	75-76
BEGACT	パーツ名	イベント名	ウィンドウ名			
BEGSR	サブルーチン名					
BITOFF		ビット番号	文字フィールド			
BITON		ビット番号	文字フィールド			
CABxx	被比較数	被比較数	Label	HI	LO	EQ
CALL (E)		プログラム名	PLIST 名		ER	
CALLB (E)		プロシージャー名またはプロシージャー・ポインター	PLIST 名		ER	
CALLP (M/R)		name{ (parm1 {:parm2...}) }				
CASxx	被比較数	被比較数	サブルーチン名	HI	LO	EQ
CAT (P)	ソース・ストリング 1	ソース・ストリング 2:ブランクの数	目的のストリング			
CHAIN (E N)	検索引き数	名前 (ファイルまたはレコード様式)	データ構造	NR[2]	ER	
CHECK (E)	比較ストリング	基本ストリング:開始	左端の位置		ER	FD[2]
CHECKR (E)	比較ストリング	基本ストリング:開始	右端の位置		ER	FD[2]
CLEAR	*NOKEY	*ALL	名前 (変数、レコード様式、またはウィンドウ)			
CLOSE (E)		ファイル名または *ALL			ER	
CLSWIN (E)		ウィンドウ名			ER	
COMMIT (E)					ER	
COMP[1]	被比較数	被比較数		HI	LO	EQ
DEALLOC (E/N)			ポインター名		ER	
DEFINE	*LIKE	参照フィールド	定義されるフィールド			
DEFINE	*DTAARA	外部データ域	内部フィールド			
DELETE (E)	検索引き数またはサブファイル索引	名前 (ファイルまたはレコード様式)		NR[2]	ER	
DIV (H)	被除数	除数	商	+	−	Z
DO	開始値	限界値	索引値			
DOU (M/R)		標識式				
DOUxx	被比較数	被比較数				
DOW (M/R)		標識式				
DOWxx	被比較数	被比較数				

表 33. 通常構文での命令コード (続き)

コード	演算項目 1	演算項目 2	結果フィールド	演算結果標識		
				71-72	73-74	75-76
DSPLY (E)	メッセージ	メッセージ・ウィンドウ定義名	応答		ER	
ELSE						
ELSEIF (M/R)		標識式				
END		増分値				
ENDACT		戻り点				
ENDCS						
ENDDO		増分値				
ENDOR						
ENDIF						
ENDMON						
ENDSL						
ENDSR	ラベル	戻りポイント				
EVAL (H M/R)		結果 = 式				
EVALR (M/R)		結果 = 式				
EXCEPT		例外レコード名				
EXSR		サブルーチン名				
EXTRCT (E)		日付 / 時刻:期間コード	ターゲット・フィールド		ER	
FEOD (E)		ファイル名			ER	
FOR		Index-name = start-value BY increment TO | DOWNTO limit				
GETATR (E)	パーツ名	属性名	フィールド名		ER	
GOTO		ラベル				
IF (M/R)		標識式				
IFxx	被比較数	被比較数				
IN (E)	*LOCK	データ域名			ER	
ITER						
KFLD			キー・フィールド			
KLIST	KLIST 名					
LEAVE						
LEAVESR						
LOOKUP[1] (配列)	検索引き数	配列名		HI	LO	EQ[6]
LOOKUP[1] (テーブル)	検索引き数	テーブル名	テーブル名	HI	LO	EQ[6]
MONITOR						
MOVE (P)	データ属性	ソース・フィールド	ターゲット・フィールド	+	–	ZB
MOVEA (P)		ソース	ターゲット	+	–	ZB

表 33. 通常構文での命令コード (続き)

コード	演算項目 1	演算項目 2	結果フィールド	演算結果標識		
				71-72	73-74	75-76
MOVEL (P)	データ属性	ソース・フィールド	ターゲット・フィールド	+	–	ZB
MULT (H)	被乗数	乗数	積	+	–	Z
MVR			剰余	+	–	Z
OCCUR (E)	オカレンス値	データ構造	オカレンス値		ER	
ON-ERROR		状況コード				
OPEN (E)		ファイル名			ER	
ORxx	被比較数	被比較数				
OTHER						
OUT (E)	*LOCK	データ域名			ER	
PARM	ターゲット・フィールド	ソース・フィールド	パラメーター			
PLIST	PLIST 名					
POST (E)[3]		ファイル名	INFDS 名		ER	
READ (E N)		名前 (ファイル、レコード様式、またはウィンドウ)	データ構造[4]		ER	EOF[5]
READC (E)		サブファイル名			ER	EOF[5]
READE (E N)	検索引き数	名前 (ファイルまたはレコード様式)	データ構造[4]		ER	EOF[5]
READP (E N)		名前 (ファイルまたはレコード様式)	データ構造[4]		ER	BOF[5]
READPE (E N)	検索引き数	名前 (ファイルまたはレコード様式)	データ構造[4]		ER	BOF[5]
READS (E)		サブファイル名			ER	EOF[5]
REALLOC (E)		長さ	ポインター		ER	
RESET (E)	*NOKEY	*ALL	名前 (変数、レコード様式、またはウィンドウ)		ER	
RETURN (H M/R)		式				
ROLBK (E)					ER	
SCAN (E)	比較ストリング:長さ	基本ストリング:開始	左端の位置		ER	FD[2]
SELECT						
SETATR (E)	パーツ名	属性値	属性		ER	
SETGT (E)	検索引き数	名前 (ファイルまたはレコード様式)		NR[2]	ER	
SETLL (E)[6]	検索引き数	名前 (ファイルまたはレコード様式)		NR[2]	ER	EQ
SETOFF[1]				OF	OF	OF
SETON[1]				ON	ON	ON
SHOWWIN (E)		ウィンドウ名			ER	

表 33. 通常構文での命令コード (続き)

コード	演算項目 1	演算項目 2	結果フィールド	演算結果標識		
				71-72	73-74	75-76
SORTA		配列名				
SQRT (H)		値	根			
START (E)		コンポーネント名またはフィールド名	PLIST 名		ER	
STOP (E)		コンポーネント名			ER	
SUB (H)	被減数	減数	差	+	−	Z
SUBDUR (E) (期間)	日付 / 時刻 / タイム・スタンプ	日付 / 時刻 / タイム・スタンプ	期間: 期間コード		ER	
SUBDUR (E) (新規日付)	日付 / 時刻 / タイム・スタンプ	期間:期間コード	日付 / 時刻 / タイム・スタンプ		ER	
SUBST (E P)	抽出する長さ	基本ストリング:開始	目的のストリング		ER	
TAG	ラベル					
TEST (E)[8]			日付 / 時刻または タイム・スタンプ・フィールド		ER	
TEST (D E) [8]	日付形式		文字または数字フィールド		ER	
TEST (E T)[8]	時刻形式		文字または数字フィールド		ER	
TEST (E Z)[8]	タイム・スタンプ形式		文字または数字フィールド		ER	
TESTB[1]		ビット番号	文字フィールド	OF	ON	EQ
TESTN[1]			文字フィールド	NU	BN	BL
TESTZ[1]			文字フィールド	+	−	
TIME	別名		ターゲット・フィールド			
UNLOCK (E)		名前 (ファイルまたはデータ域)			ER	
UPDATE (E)		名前 (ファイル、レコード様式、またはウィンドウ)	データ構造[4]		ER	
WHEN (M/R)		標識式				
WHENxx	被比較数	被比較数				
WRITE (E)		名前 (ファイル、レコード様式、サブファイル、またはウィンドウ)	データ構造[4]		ER	EOF[5]
XFOOT (H)		配列名	和	+	−	Z

表 33. 通常構文での命令コード (続き)

コード	演算項目 1	演算項目 2	結果フィールド	演算結果標識		
				71-72	73-74	75-76
XLATE (E P)	始め:終わり	ストリング:開始	目的のストリング		ER	
Z-ADD (H)		加数	和	+	–	Z
Z-SUB (H)		減数	差	+	–	Z

注:

1. 演算結果標識が少なくとも 1 つ必要です。
2. NR または FD 演算結果標識を指定する代わりとして、%FOUND 組み込み関数を使用することができます。
3. 結果フィールドに演算項目 2 を指定しなければなりません。入出力共用を指定することができます。
4. 演算項目 2 にプログラム記述ファイル名が含まれている時にのみ、結果フィールドでデータ構造を使用することができます。
5. EOF または BOF 演算結果標識を指定する代わりとして、%EOF 組み込み関数を使用することができます。
6. %EQUAL 組み込み関数を使用して、SETLL および LOOKUP 演算命令をテストすることができます。
7. 拡張 'E' をもつすべての命令コードでは、拡張 'E' か ER エラー標識のどちらかを指定することができますが、両方を指定することはできません。
8. TEST 演算には、拡張 'E' またはエラー標識を指定しなければなりません。

算術演算

以下の表には算術演算を示しています。

表 34. 算術演算

演算	従来の構文	フリー・フォーム構文
絶対値	404 ページの『%ABS (式の絶対値)』	
加算	481 ページの『ADD (加算)』	+ 演算子
除算	533 ページの『DIV (除算)』	/ 演算子または 420 ページの『%DIV (商の整数部分を戻す)』
除算の剰余	621 ページの『MVR (剰余の移動)』	451 ページの『%REM (整数剰余を戻す)』
乗算	620 ページの『MULT (乗算)』	* 演算子
平方根	678 ページの『SQRT (平方根)』	459 ページの『%SQRT (式の平方根)』
減算	682 ページの『SUB (減算)』	- 演算子
ゼロにして加算	712 ページの『Z-ADD (ゼロにして加算)』	(使用できません)
ゼロにして減算	713 ページの『Z-SUB (ゼロにして減算)』	(使用できません)

算術演算の例については、 355 ページの図 114 を参照してください。

算術演算を指定する場合には、次の規則が適用されます。

- 算術演算を実行できるのは以下の数値だけです。
  - 数値サブフィールド
  - 数値配列
  - 数値配列エレメント
  - 数値テーブル・エレメント

  - 数値名前付き固定情報
  - 数値表意定数
  - 数値リテラル
- 一般に、算術演算はパック 10 進数形式で実行されます。このことは、算術演算を行なう前にフィールドが最初にパック 10 進数形式に変換され、次に結果を結果フィールドに入れる前に (必要な場合) 指定された形式に戻されることを意味しています。しかし、次の例外に注意してください。
  - すべてのオペランドが符号なしの場合には、演算では符号なしの演算が使用されます。
  - すべてが整数または整数と符号なしの場合には、演算では整数の演算が使用されます。
  - いずれかのオペランドが浮動の場合には、残りのオペランドが浮動に変換されます。

  しかし、DIV 演算ではその演算にパック 10 進数または浮動形式のいずれかが使用されます。整数と符号なしの演算の詳細については、354 ページの『整数および符号なしの演算』 を参照してください。
- 小数点位置合わせはすべての算術演算で行なわれます。切り捨てが行なわれる場合であっても、結果フィールドの小数点の位置には影響しません。
- 算術演算で指定されるフィールドの長さは 30 桁を超えることができません。結果が 30 桁を超える場合には、小数点の位置によって、いずれかまたは両端の桁が除去されます。
- TRUNCNBR オプションは、数値オーバーフローまたは実行時エラーが生成された場合に左側で切り捨てを行なうかどうかを決定します。このオプションはビルド・ウィンドウで指定することができます。詳細については、*WebSphere Development Studio Client for iSeries ご使用に際して* を参照してください
- 算術演算では、演算項目 1 と演算項目 2 はそれらが結果フィールドと同じでない限り変更はされません。
- 算術演算の結果は、結果フィールドに入っていたデータを置き換えます。
- 四捨五入は、結果フィールドの最後に指定された小数点位置の 1 桁右側に 5 (フィールドが負の場合には -5) を加えて行なわれます。四捨五入が行なわれるのは算術演算の場合だけですが、MVR 演算または MVR 演算が続く DIV 演算では行なわれません。四捨五入が結果に影響を与えるのは、計算された結果が結果フィールドの小数点以下の桁数より大きい場合だけです。四捨五入は、演算の後で、結果が結果フィールドに入れられる前に行なわれます。演算結果標識は、四捨五入が行なわれた後で、結果フィールドの値に従って設定されます。
- DIV および MVR で条件標識を使用する場合には、DIV 演算が MVR 演算の直前に行なわれるようにするのはユーザーの責任です。 DIV の条件標識によって MVR 演算が DIV の直前に実行されると、好ましくない結果が起こることがあります。

3 つのフィールド全部を使用する算術演算の場合:
- 演算項目 1、演算項目 2、および結果フィールドは異なる 3 つのフィールドとすることができます。
- 演算項目 1、演算項目 2、および結果フィールドはすべて同じフィールドとすることができます。
- 演算項目 1 と演算項目 2 は同じで、結果フィールドは異なるフィールドとすることができます。

- 演算項目 1 または演算項目 2 のいずれかを結果フィールドと同じにすることができます。

算術演算で配列を使用する場合については、 186 ページの『演算での配列の指定』を参照してください。

パフォーマンスの考慮

算術演算でパフォーマンス時間が一番短いのは、すべてのオペランドが整数または符号なしの形式のときです。次にパフォーマンス時間が短いのは、パック 10 進数形式のときです。共通形式への変換が要らないためです。

整数および符号なしの演算

すべての算術演算 (式の演算を除く) で、演算項目 1、演算項目 2、および結果フィールドが符号なしの形式で定義されている場合には、演算は符号なしの形式で行なわれます。同様に、演算項目 1、演算項目 2、および結果フィールドが整数または符号なしのいずれかの形式として定義されている場合には、演算は整数の形式で行なわれます。いずれかのフィールドが整数または符号なしの形式でない場合には、演算はデフォルトの形式 (パック 10 進数) で行なわれます。

以下は、整数および符号なしの算術演算のみに適用されます。

- いずれかのフィールドが 4 バイト・フィールドとして定義されている場合には、演算が行なわれる前にすべてのフィールドが 4 バイトに変換されます。
- 整数と符号なしの値は 1 つの演算で一緒に使用することができます。しかし、演算項目 1、演算項目 2、または結果フィールドが符号付きの場合には、符号なしのすべての値が整数に変換されます。必要な場合には、数値オーバーフローが起こる機会を少なくするために、符号なしの 2 バイトの値が 4 バイトの整数値に変換されます。
- リテラルが小数点以下の桁数ゼロで 10 桁またはそれ以下で、整数と符号なしのフィールドに許容される範囲内に入っている場合には、それが負の値か正の値かによって、それぞれ整数形式または符号なしの形式でロードされます。

注: 整数または符号なしの演算ではパフォーマンスがよくなることがあります。しかし、いずれかのタイプの演算を使用したときに数値オーバーフローが起こる機会は大きくなります。

算術演算の例

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
C*
C*   以下の例ではフィールド初期値は以下の通りです。
C*
D A               s              3p 0  inz(1)
D B               s              3p 1  inz(10.0)
D C               s              2p 0  inz(32)
D D               s              2p 0 inz(-10)
D E               s              3p 0 inz(6)
D F               s              3p 0 inz(10)
D G               s              3p 2 inz(2.77)
D H               s              3p 0 inz(70)
D J               s              3p 1 inz(0.6)
D K               s              2p 0 inz(25)
D L               s              2p 1 dim(3)
D V               s              5p 2
D W               s              5p 1
D X               s              8p 4
D Y               s              6p 2
D Z               s              5p 3

 /FREE
    L(1) = 1.0;
    L(2) = 1.7;
    L(3) = -1.1;

    A = A + 1;               // A = 002
    V = B + C;               // V = 042.00
    V = B + D;               // V = -010.00
    V = C;                   // V = 032.00
    E = E - 1;               // E = 005
    W = C - B;               // W = 0022.0
    W = C - D;               // W = 0052.0
    W = - C;                 // W = -0032.0
    F = F * E;               // F = 060
    X = B * G;               // X = 0027.7000
    X = B * D;               // X = -0200.0000
    H = H / B;               // H = 007
    Y = C / J;               // Y = 0053.33
    eval(r) Z = %sqrt(K);    // Z = 05.000
    Z = %xfoot(L);           // Z = 01.600

    dump(a);
    *inlr = *on;
 /END-FREE
```

図 114. 算術演算の要約

配列の演算

以下の表には配列演算命令を示しています。

表 35. 配列演算命令

演算	従来の構文	フリー・フォーム構文
エレメントのルックアップ	582 ページの『LOOKUP (テーブルまたは配列エレメントの検索)』	441 ページの『%LOOKUPxx (配列エレメントの検索)』 または 471 ページの『%TLOOKUPxx (テーブル・エレメントの検索)』
エレメントの数	426 ページの『%ELEM (エレメント数の取り出し)』	
配列の移動	603 ページの『MOVEA (配列の転送)』	(使用できません)
配列の分類	676 ページの『SORTA (配列の分類)』	
配列のエレメントの合計	709 ページの『XFOOT (配列エレメントの合計)』	477 ページの『%XFOOT (配列式エレメントを合計)』

配列では多くの演算が使用されますが、それらの演算では特定の配列の機能が実行されます。その機能についての説明は、それぞれの演算を参照してください。

ビット演算

ビット演算命令は次のとおりです。

- 493 ページの『BITOFF (ビットのオフ設定)』
- 494 ページの『BITON (ビットのオン設定)』
- 693 ページの『TESTB (ビットのテスト)』.

バイト中のビットは左から右へと番号づけされています。左端のビットがビット番号 0 です。これらの演算では、演算項目 2 がビット・パターン (ビット番号) を指定し、結果フィールドが演算の行なわれる 1バイトの文字フィールドを指定します。演算項目 2 にビット番号を指定するためには、1 バイトの 16進リテラルまたは 1 バイトの文字フィールドが使用することができます。ビット番号は、そのリテラルまたはフィールドでオンにされたビットで示されます。また、演算項目 2 にはビット番号が入っている文字リテラルも指定することができます。

その機能についての説明は、それぞれの演算を参照してください。

分岐命令

以下の表には分岐演算命令を示しています。

表 36. 分岐演算命令

演算	従来の構文	フリー・フォーム構文
比較して分岐	496 ページの『CABxx (比較および分岐)』	(使用できません)
ジャンプ	568 ページの『GOTO (行先指定)』	(使用できません)
Iterate	574 ページの『ITER (繰り返し)』	
Leave	579 ページの『LEAVE (DO/FOR グループの終了)』	
タグ	689 ページの『TAG (タグ)』	(使用できません)

その機能についての説明は、それぞれの演算を参照してください。

呼び出し命令

以下の表には呼び出し演算命令を示しています。

表 37. 呼び出し演算命令

演算	従来の構文	フリー・フォーム構文
プログラムまたはプロシージャーの呼び出し	• 498 ページの『CALL (AS/400 プログラムの呼び出し)』 • 502 ページの『CALLB (機能の呼び出し)』 • 503 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』	503 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』
パラメーターの識別	• 633 ページの『PARM (パラメーターの識別)』 • 636 ページの『PLIST (パラメーター・リストの識別)』	PI または PR 定義仕様
戻り	659 ページの『RETURN (呼び出し元への戻り)』	
コンポーネントの開始	679 ページの『START (コンポーネント開始またはローカル・プログラム呼び出し)』	

その機能についての説明は、それぞれの演算を参照してください。

CALLP はプロトタイプ呼び出しのタイプの 1 つです。2 番目のタイプは式の中からの呼び出しです。 **プロトタイプ呼び出し**は、呼び出しインターフェースにプロトタイプが定義されている呼び出しです。

呼び出し命令によって、VisualAge RPG プロシージャーは制御権を他のプログラムまたはプロシージャーに移動することができます。しかし、プロトタイプ呼び出しは、フリー・フォームの構文を使用できる点で CALL および CALLB 命令とは異なります。

RETURN 命令は、制御権を呼び出し元プログラムまたはプロシージャーに返し、値 (もしあれば) を戻します。PLIST および PARM 命令は、CALL および CALLB 命令と一緒に使用して、呼び出しで渡すパラメーターを指示することができます。プロトタイプ呼び出しでは、呼び出しでパラメーターを渡します。

推奨されるプログラムまたはプロシージャー (いずれかの言語で書かれた) の呼び出し方法は、プロトタイプ呼び出しをコーディングする方法です。

プロトタイプ呼び出し

プロトタイプ呼び出しでは、以下を (同じ構文で) 呼び出すことができます。
- 実行時にシステム上にあるプログラム
- 同じプログラムにバインドされている他のモジュールにエクスポートされたプロシージャー
- 同じモジュールのサブプロシージャー

プロトタイプは、呼び出しを行なうプログラムまたはプロシージャーの定義仕様に含めなければなりません。これは、プログラムまたはプロシージャーを正しく呼び出して、呼び出し元が正しいパラメーターを渡すようにするために、コンパイラーによって使用されます。

プログラムまたはプロシージャーがプロトタイプの場合には、パラメーターの数とタイプだけで、プログラムまたはプロシージャーで使用するデータ項目の名前を知る必要はありません。

プロトタイプは、プログラムとプロシージャーの間の通信を改善します。プロトタイプ呼び出しを使用する利点のいくつかは次のとおりです。

- PARM または PLIST 命令が必要ないので、構文が単純化されます。
- 一部のパラメーターでは、リテラルおよび式を渡すことができます。
- コンパイラーは、呼び出しが正しくない場合はコンパイル時にエラーを出して、タイプ、形式、および長さが正しい十分な数のパラメーターが渡されるのを援助します。
- コンパイラーは、実行時に変換を実行して、一部のタイプのパラメーターに形式および長さが正しい十分な数のパラメーターが渡されるのを援助します。

図 115 は、プロトタイプ ProcName を使用し、3 つのパラメーターを渡す例を示します。プロトタイプ ProcName は、プログラムまたはプロシージャーのいずれも参照することができます。プロトタイプを定義するときにこのことを知っていることは重要ですが、これは、呼び出しを行なうときには重要ではありません。

```
/FREE
// 以下では、3 つのパラメーター CharField、7、および
// Field2 で ProcName を呼び出します。
     ProcName (CharField: 7: Field2);

// 命令拡張を指定する必要がある場合には、CALLP 命令コードも
// 指定しなければなりません。
     CALLP(e) ProcName (CharField: 7: Field2);
/END-FREE
```

図 115. CALLP 命令の例

式の中でプロシージャーを呼び出す場合には、指定された戻り値のデータ・タイプに合った方法でプロシージャー名を使用してください。たとえば、プロシージャーが数値を戻すように定義されている場合には、式の中でのプロシージャー呼び出しは数値が必要な場合でなければなりません。

プログラムおよびプロシージャーの呼び出しとパラメーターの受け渡しの詳細については、 *VisualAge for RPG プログラミング* を参照してください。プロトタイプおよびパラメーターの定義の詳細については、73 ページの『プロトタイプおよびパラメーター』 を参照してください。

呼び出しでのプログラム名の解析

プログラム名は CALL 命令の演算項目 2 に指定します。ライブラリー名を指定する場合には、すぐ後にスラッシュを続けて、その後にプログラム名を続けなければ

なりません (たとえば、'LIB/PROG') 。ライブラリーを指定しない場合には、ライブラリー・リストを使用してプログラムを見つけます。*CURLIB はサポートされません。

次の規則に注意してください。
- リテラルの全長 (スラッシュを含む) は 12 文字を超えることができません。
- フィールドまたは名前付き固定情報の非ブランク・データ (スラッシュを含む) の全長は 21 文字を超えることができません。
- プログラムまたはライブラリー名が 10 文字を超える場合には、10 文字で切り捨てられます。

プログラム名は、呼び出すプログラムを判別するために、リテラル、フィールド、名前付き固定情報、または配列エレメントで指定されているとおりにそのまま使用されます。具体的にいえば、次のようになります。
- 先行ブランクまたは末尾ブランクは無視されます。
- 項目の最初の文字がスラッシュの場合には、プログラムを見つけるためにライブラリー・リストが使用されます。
- 項目の最後の文字がスラッシュの場合には、コンパイル時メッセージが出されます。
- 小文字は大文字にシフトされません。
- 引用符で囲まれた名前 (たとえば、'″ABC″') には、呼び出すプログラムの名前の一部として常に引用符が含められます。

プログラム参照は、ターゲット・プログラムを解決するオーバーヘッドを避けるためにグループ化されます。特定のプログラムの参照はすべて名前付き固定情報やリテラルを使用してグループ化され、プログラムを一回解決するだけで、そのプログラムに対する (名前付き固定情報やリテラルのみによる) 以後のすべての参照で再び解決しなくても済むようになります。

プログラム参照は、プログラムとライブラリー名が両方とも同じ場合には、グループ化されます。変数名によるプログラム参照はすべてその変数名でグループ化されます。プログラム参照が変数で行なわれた場合には、その現在の値がその変数を使用している前のプログラム参照命令で使用された値と比較されます。値が変更されていなければ、解決は行なわれません。値が代わっている場合には、指定された新しいプログラムに対して解決が行なわれます。この規則が適用されるのは、変数名を使用した参照だけであることを注意してください。名前付き固定情報またはリテラルを使用した参照は再び解決されることはなく、変数によるプログラム参照が再度解決されるかどうかには影響を与えません。プログラム参照のグループ化を示します。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D Pgm_Ex_A        C                   'LIB1/PGM1'
D Pgm_Ex_B        C                   'PGM1'
D PGM_Ex_C        C                   'LIB/PGM2'
D*
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C                   CALL      Pgm_Ex_A
C*
C* 次の 2 つの呼び出しは、両方が同じプログラム名 (PGM1) と
C* 同じライブラリー名 (none) を持っているのでグループ化され
C* ます。これらは上の Pgm_Ex_A を使用している呼び出しとは
C* グループ化されません。Pgm_Ex_A が別のライブラリー名 (LIB1)
C* を指定しているためです。
C*
C                   CALL      'PGM1'
C                   CALL      Pgm_Ex_B
C*
C* 次の 2 つのプログラム参照は、両方が同じプログラム名 (PGM2)
C* と同じライブラリー名 (LIB) を持っているのでグループ化され
C* ます。
C*
C                   CALL      'LIB/PGM2'
C                   CALL      Pgm_Ex_C
C*
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 下の CALLV を使用しているプログラムの最初の呼び出しで、プログラム
C* PGM1 に対する変数 CALLV の解決が実行されることになります。
C* これは、そのプログラムですでに行なわれているリテラルまたは
C* 名前付き固定情報による PGM1 の呼び出しとは別です。CALLV を使用した
C* 2 番目の呼び出しでは、CALLV の値が変更されていないので、
C* PGM1 の解決は行なわれません。
C*
C                   MOVE     'PGM1'         CALLV         21
C                   CALL     CALLV
C                   CALL     CALLV
```

図 116. プログラム参照のグループ化の例

比較命令

以下の表には比較演算命令を示しています。

表 38. 比較演算命令

演算	従来の構文	フリー・フォーム構文
And	486 ページの『ANDxx (AND)』	AND 演算子
比較	526 ページの『COMP (比較)』	=、<、>、<=、>=、または <> 演算子
比較して分岐	496 ページの『CABxx (比較および分岐)』	(使用できません)
条件付きサブルーチン	505 ページの『CASxx (サブルーチンの条件付き起動)』	569 ページの『IF (判断)』 および 558 ページの『EXSR (ユーザー・サブルーチンの起動)』
Do Until	537 ページの『DOU (実行の終了)』 または 538 ページの『DOUxx (実行の終了)』	537 ページの『DOU (実行の終了)』
Do While	541 ページの『DOW (実行の時期)』 または 542 ページの『DOWxx (実行の時期)』	541 ページの『DOW (実行の時期)』
If	569 ページの『IF (判断)』 または 570 ページの『IFxx (判断)』	569 ページの『IF (判断)』
Or	629 ページの『ORxx (OR)』	OR 演算子
When	703 ページの『WHEN (真の場合に選択)』 または 704 ページの『WHENxx (真の場合に選択)』	703 ページの『WHEN (真の場合に選択)』

ANDxx、CABxx、CASxx、DOUxx、DOWxx、IFxx、ORxx、および WHENxx 命令の場合には、xx は以下のようになります。

xx	意味
GT	演算項目 1 は演算項目 2 より大きい。
LT	演算項目 1 は演算項目 2 より小さい。
EQ	演算項目 1 は演算項目 2 に等しい。
NE	演算項目 1 は演算項目 2 と等しくない。
GE	演算項目 1 は演算項目 2 と等しいかそれより大きい。
LE	演算項目 1 は演算項目 2 と等しいかそれより小さい。
ブランク	無条件の処理 (CASxx または CABxx) 。

比較命令は、フィールドの演算に指定された条件をテストします。これらの命令はフィールドの値を変更しません。COMP、CABXX、および CASXX の場合には、71-76 桁目に割り当てられた演算結果標識はその演算の結果によって設定されます。すべてのデータ・タイプを同じデータ・タイプのフィールドと比較することができます。

比較命令を使用する場合には、以下のことに留意してください。

- 数値フィールドが比較される場合には、長さの異なるフィールドは暗黙の小数点に位置合わせされます。フィールドには、比較のためにフィールドの長さと小数点以下の桁数が同じになるように、小数点の左側または右側、あるいはその両方にゼロが埋められます。

- すべての数値比較は代数によります。プラス (+) の値は常にマイナス (-) の値より大きくなります。
- すべてのグラフィックおよび UCS-2 の比較は、グラフィック文字の 16 進数表現を使用して行なわれます。
- 文字、グラフィック、または UCS-2 フィールドが比較される場合には、長さの異なるフィールドは左端文字に位置合わせされます。短いフィールドには、比較のためにフィールドの長さが同じになるように、長いフィールドの長さと等しくなるまでブランクが埋められます。
- 日付フィールドは、比較されるときに共通形式に変換されます。
- 時刻フィールドは、比較されるときに共通形式に変換されます。
- 基底ポインター・フィールドは、等号または不等号以外のすべてについて比較される場合には、結果はポインターが連続ストレージ内のアドレスをポイント (たとえば、すべてが同じデータ構造、配列、または独立型フィールド内の位置をポイント) していない限り予期できないもとのなります。
- プロシージャー・ポインター・フィールドが等号または不等号以外のすべてについて比較される場合には、結果は予期できないものとなります。
- 比較命令で配列名を指定することはできませんが、配列エレメントは指定できます。
- ANDxx および ORxx 命令は、次の DOUxx、DOWxx、IFxx、および WHENxx には使用することができません。
- 長さがゼロの文字、グラフィック、または UCS-2 リテラルをブランクが入っているフィールド (固定または可変) と比較すると、フィールドは等しいとして比較されます。値が長さ 0 であることをテストしたい場合には、%LEN 組み込み関数を使用してください。

変換操作

以下の組み込み関数は、変換操作を実行します。
- 408 ページの『%CHAR (文字データに変換)』
- 416 ページの『%DEC (パック 10 進数形式に変換)』
- 417 ページの『%DECH (四捨五入のあるパック 10 進数形式に変換)』
- 431 ページの『%FLOAT (浮動形式に変換する)』
- 435 ページの『%GRAPH (グラフィック値に変換)』
- 437 ページの『%INT (整数形式に変換)』
- 437 ページの『%INTH (四捨五入付き整数形式に変換)』
- 475 ページの『%UCS2 (UCS-2 値に変換する)』
- 421 ページの『%EDITC (編集コードを使用しての値の編集)』
- 424 ページの『%EDITFLT (浮動外部表現に変換)』
- 425 ページの『%EDITW (編集語を使用しての値の編集)』
- 476 ページの『%UNS (符号なし形式に変換)』
- 476 ページの『%UNSH (四捨五入付き符号なし形式に変換)』

これらの組み込み関数は、従来の構文でも、フリー・フォーム構文でも使用できます。

従来の MOVE および MOVEL 命令コードは、演算項目 2 と結果フィールドが異なるタイプであるときに変換を実行します。以下を参照してください。
- 587 ページの『MOVE (転送)』
- 610 ページの『MOVEL (左につめて転送)』

データ域命令

データ域命令は次のとおりです。
- 572 ページの『IN (データ域の検索)』
- 632 ページの『OUT (データ域の書き出し)』
- 699 ページの『UNLOCK (データ域のロック解除またはレコードの解放)』.

これらの演算命令は、従来の構文でも、フリー・フォーム構文でも使用できます。

アプリケーションが OS/400 データ域にアクセスする場合は、このデータ域の名前は OS/400 データ域の名前または「サーバー情報の定義」メニュー項目を使用して定義した一時変更名のいずれかにすることができます。GUI Designer を使用したサーバー情報の定義に関する詳細については、*VisualAge for RPG プログラミング* を参照してください。

次のロック状態が使用されます。

*LOCK を指定した IN 命令	データ域に排他読み取りロック許可状態が入れられます。
*LOCK を指定した OUT 命令	データ域は書き出し命令の後もロックされています。
ブランクの OUT 命令	データ域は更新後にロック解除されます。
UNLOCK	データ域はロック解除され、レコード・ロックが解除されて、データ域またはレコード、あるいはその両方が更新されません。

データがデータ域に移動され、データ域から出される場合には、システムはそのデータ域をロックします。複数のユーザーが同じデータ域で競合している場合には、ユーザーはそのデータ域が使用不可能であることを示すメッセージを受け取ることがあります。

データ域命令には次の規則が適用されます。
- データ域命令は、オペレーティング・システムに対して定義されていないデータ域で実行することはできません。
- データ域命令を実行する前に、そのデータ域は定義仕様または DEFINE 命令コードで指定されていなければなりません。詳細については、 529 ページの『DEFINE (フィールド定義)』 を参照してください。
- ロックされているデータ域を更新したり別のプログラムでロックすることはできません。
- データ域名を、複数回繰り返しデータ構造、入力レコード・フィールド、配列、配列エレメント、またはテーブルの名前にすることはできません。
- データ域を、複数回繰り返しデータ構造、データ域データ構造、プログラム状況データ構造、ファイル情報データ構造 (INFDS)、または *DTAARA DEFINE ステートメントに現われるデータ構造のサブフィールドにすることはできません。

日付の演算

以下の表には日付に関する演算命令を示しています。

表 39. 日付に関する演算命令

演算	従来の構文	フリー・フォーム構文
期間の加算	482 ページの『ADDDUR (期間の加算)』	+ 演算子
抽出	560 ページの『EXTRCT (日付 / 時刻 / タイム・スタンプの抽出)』	465 ページの『%SUBDT (日付、時刻、または時刻スタンプの部分の取り出し)』
期間の減算	683 ページの『SUBDUR (期間の減算)』	- 演算子または 419 ページの『%DIFF (2 つの日付、時刻、または時刻スタンプ値の間の差)』
テスト	690 ページの『TEST (日付、時刻、タイム・スタンプのテスト)』	
年数	479 ページの『%YEARS (年数)』	
月数	444 ページの『%MONTHS (月数)』	
日数	415 ページの『%DAYS (日数)』	
時間数	436 ページの『%HOURS (時間数)』	
分数	443 ページの『%MINUTES (分数)』	
秒数	455 ページの『%SECONDS (秒数)』	
マイクロ秒数	445 ページの『%MSECONDS (マイクロ秒数)』	

日付に関する演算命令によって、日付、時刻、およびタイム・スタンプのフィールドを処理したり、日付、時刻、およびタイム・スタンプを表す文字フィールドまたは数値フィールドを処理したりできます。以下のことを行うことができます。

- 年、月、日、時間、分、秒、またはマイクロ秒単位で期間を加算する
- 2 つの日付、時刻、またはタイム・スタンプとの間の期間を判別する
- 日付、時刻、またはタイム・スタンプの一部 (たとえば日) を抽出する
- 値が日付、時刻、またはタイム・スタンプとして有効であるかをテストする

期間を加算または減算するには、フリー・フォーム構文で + または - 演算子を使用するか、従来の構文で ADDDUR または SUBDUR 命令コードを使用できます。以下の表には、フリー・フォーム構文で使用する組み込み関数および従来の構文で使用する期間コードを示しています。

表 40. 組み込み関数および期間コード

単位	組み込み関数	期間コード
年	%YEARS	*YEARS または *Y
月	%MONTHS	*MONTHS または *M
日	%DAYS	*DAYS または *D
時間	%HOURS	*HOURS または *H
分	%MINUTES	*MINUTES または *MN
秒	%SECONDS	*SECONDS または *S
マイクロ秒	%MSECONDS	*MSECONDS または *MS

たとえば、以下のいずれかの方法で既存の日付に 23 日を加算することができます。

```
C                   ADDDUR    23:*D         DUEDATE

 /FREE
     newdate = duedate + %DAYS(23)
 /END-FREE
```

2 つの日付、時刻、またはタイム・スタンプの間の期間を計算するには、フリー・フォーム構文では %DIFF 組み込み関数を使用し、従来の構文では SUBDUR 命令コードを使用することができます。いずれの場合にも、 364 ページの表 40 に示した期間コードの 1 つを指定しなければなりません。

期間は、完全な単位で示され、剰余は廃棄されます。 59 分の期間は時間では 0、61 分の期間は時間では 1 となります。

以下の表には、SUBDUR 命令コードを使用する追加の例を示しています。%DIFF 組み込み関数でも同じ結果が得られます。

表 41. SUBDUR を使用した期間の結果

期間の単位	演算項目 1	演算項目 2	結果
月	1999-03-28	1999-02-28	1 月
	1999-03-14	1998-03-15	11 か月
	1999-03-15	1998-03-15	12 か月
年	1999-03-14	1998-03-15	0 年
	1999-03-15	1998-03-15	1 年
	1999-03-14-12.34.45.123456	1998-03-14-12.34.45.123457	0 年
時間	1990-03-14-23.00.00.000000	1990-03-14-22.00.00.000001	0 時間

予期しない結果

月は 28、29、30、または 31 日となります。年には 365 または 366 日となります。この不整合のために、以下の演算では予期しない結果が生じる可能性があります。

- 各月の 29 日、30 日、または 31 日の日付で月数を加算または減算する (あるいは月単位で期間を計算する)
- 年数を 2 月 29 日で加算または減算する (あるいは年単位で期間を計算する)

以下の規則が使用されます。

- 月または年が加算または減算されるときには、可能な限り日の部分は未変更のままになります。たとえば、2000-03-15 + %MONTHS(1) は 2000-04-15 になります。
- 加算または減算で存在しない日付 (たとえば 4 月 31 日) が生じた場合には、代わりにその月の最後の日が使用されます。
- 日の部分を変更する月または年の演算は、可逆で**ありません**。たとえば、2000-03-31 + %MONTHS(1) is 2000-04-30 は、31 から 30 に日を変更します。1 か月を減算しても、元の 2000-03-31 に戻すことはできません。

  演算 2000-03-31 + %MONTHS(1) - %MONTHS(1) は、2000-03-30 になってしまいます。

- 後の方の日付から 1 か月を引くと最初の日付になる場合には、2 つの日付の間の期間は 1 か月となります。たとえば、2000-04-30 - %MONTHS(1) は 2000-03-30 (2000-03-31 でなく) であるので、2000-03-31 と 2000-04-30 の間の月による期間 (端数切り捨て) は 0 となります。

宣言命令

以下の表には宣言の演算命令を示しています。

表 42. 宣言演算命令

演算	従来の構文	フリー・フォーム構文
フィールドの定義	529 ページの『DEFINE (フィールド定義)』	定義仕様における LIKE または DTAARA キーワード
キーの定義	• 576 ページの『KFLD (キーのパーツの定義)』 • 577 ページの『KLIST (複合キーの定義)』	(使用できません)
パラメーターの識別	• 633 ページの『PARM (パラメーターの識別)』 • 636 ページの『PLIST (パラメーター・リストの識別)』	PR 定義仕様
タグ	689 ページの『TAG (タグ)』	(使用できません)

宣言命令は、フィールドのプロパティーを宣言するか、またはプログラムの一部にマークづけするために使用します。制御レベル項目 (7 桁目と 8 桁目) は、ブランクにするか、またはプログラムの該当セクション内のステートメントをグループ化する項目を含めることができます。

エラー処理操作

例外処理命令コードは以下の通りです。
- 585 ページの『MONITOR (モニター・グループの開始)』
- 626 ページの『ON-ERROR (エラー処理)』
- ENDMON、 548 ページの『ENDyy (構造化グループの終了)』 で説明

これらの命令コードは、従来の構文でも、フリー・フォーム構文でも使用できます。

MONITOR、ON-ERROR、および ENDMON は、モニター・グループをコーディングするために使用されます。モニター・グループは、モニター・ブロック、その後に 1 つ以上のエラー処理ブロックが続き、さらにその後に ENDMON が続いたものからなります。

モニター・ブロックにはエラーを生成したと考えられるコードが入っています。エラー処理ブロックには、モニター・ブロック中で起こったエラーを処理するためのコードが入っています。

モニター・ブロックは、MONITOR 演算命令とその後にモニターされる演算命令が続いたものからなります。エラー処理ブロックは、ON-ERROR 演算命令、後に状況

コードのリストが続き、さらにその後に演算命令 (モニター・ブロック中のエラーのために、リストされた状況コードのいずれかが生成された場合に実行される演算命令) が続いたものからなります。

モニター・ブロック中でエラーが起こり、演算命令に (E) 拡張またはエラー標識があると、そのエラーは、(E) 拡張またはエラー標識によって処理されます。標識または拡張でエラーを処理できない場合には、制御がエラーの状況コードを含むエラー処理ブロックに移ります。エラー処理ブロックが完了すると、制御が ENDMON に移ります。エラーを処理するためのエラー処理ブロックがない場合には、制御は次のレベルの例外処理 (*PSSR または INFSR サブルーチンあるいはデフォルトのエラー・ハンドラー) に移ります。

```
/free
   MONITOR;                              _
       OPEN   FILE;                       |
       DOW    getNextRecord ();           |
         X = X + 1;                       +-- これがモニター・ブロックです
         nameList(X) = name;              |
       ENDDO;                             |
       CLOSE  FILE;                      _|
   ON-ERROR   1216;                      _
       DSPMSG                             |
          ('Error opening file FILE'      |
         : %status);                      +-- 最初のエラー処理ブロック
       RETURN;                           _|
   ON-ERROR   121;                       _
       DSPMSG                             |
          ('Array NAME is too small'      +-- 2 番目のエラー処理ブロック
         : %status);                      |
       RETURN;                           _|
   ON-ERROR   *ALL;                      _
       DSPMSG                             |
          ('Unexpected error'             +-- 最終の総括エラー処理ブロック
         : %status);                      |
       RETURN;                           _|
   ENDMON;                               --- MONITOR グループの終わり
/end-free
```

図 117. MONITOR および ON-ERROR ブロックの例

ファイル命令

ファイル命令コードは次のとおりです。

- 510 ページの『CHAIN (ファイルからのランダム検索)』
- 523 ページの『CLOSE (ファイルのクローズ)』
- 525 ページの『COMMIT (コミット)』
- 532 ページの『DELETE (レコードの削除)』
- 556 ページの『EXCEPT (演算時の出力)』
- 562 ページの『FEOD (データの終わりの強制)』
- 627 ページの『OPEN (処理用ファイルのオープン)』
- 638 ページの『POST (転記)』
- 640 ページの『READ (レコードの読み取り)』
- 643 ページの『READC (次の変更レコードの読み取り)』

- 645 ページの『READE (等しいキーのレコードの読み取り)』
- 648 ページの『READP (前のレコードの読み取り)』
- 650 ページの『READPE (前の等しいキーのレコードの読み取り)』
- 653 ページの『READS (選択されたレコードの読み取り)』
- 660 ページの『ROLBK (ロールバック)』
- 668 ページの『SETGT (より大きいレコードへのセット)』
- 671 ページの『SETLL (下限のセット)』
- 699 ページの『UNLOCK (データ域のロック解除またはレコードの解放)』
- 701 ページの『UPDATE (既存レコードの変更)』
- 707 ページの『WRITE (新しいレコードの作成)』

ファイル組み込み関数は以下の通りです。
- 427 ページの『%EOF (ファイル条件の終了または開始を戻す)』
- 429 ページの『%EQUAL (正確に一致した条件を戻す)』
- 432 ページの『%FOUND (検出された条件を戻す)』
- 448 ページの『%OPEN (ファイル・オープン条件を戻す)』
- 460 ページの『%STATUS (ファイルまたはプログラム状況を戻す)』

これらの演算命令は、従来の構文でも、フリー・フォーム構文でも使用できます。

ファイル命令を使用して、ローカル・ファイル、リモート・ファイル、またはウィンドウのパーツを処理することができます。

演算	OS/400 ファイル	ローカル・ファイル	サブファイル	静的テキスト	入力フィールド	特殊ファイル
CHAIN	Y	Y	Y			
CLOSE	Y	Y				Y
COMMIT	Y					
DELETE	Y	Y	Y			Y
EXCEPT	Y	Y				
FEOD	Y					Y
OPEN	Y	Y				Y
POST	Y	Y				
READ	Y	Y		Y	Y	Y
READC			Y			
READE	Y					
READP	Y	Y				
READPE	Y					
READS			Y			
SETGT	Y					
SETLL	Y					
UNLOCK	Y					
UPDATE	Y	Y	Y			Y
WRITE	Y	Y	Y	Y	Y	Y

外部記述ファイルで特定のファイル命令を使用する場合には、演算項目 2 にファイル名でなくレコード様式名を指定することができます。したがって、処理命令コードでは、使用する演算命令コードの規則によって、指定したタイプのレコード様式でファイルがリトリーブまたは位置づけされ、あるいはリトリーブして位置づけされます。

演算項目 2 にプログラム記述ファイル名を指定する WRITE および UPDATE 命令では、結果フィールドにデータ構造名が指定されていなければなりません。演算項目 2 にプログラム記述ファイル名を指定する CHAIN 、READ、および READP 命令では、結果フィールドにデータ構造名が指定されていてもかまいません。
CHAIN、READ、および READP 命令では、データはそのファイルの入力仕様を処理しないで直接ファイルとデータ構造の間で転送されます。したがって、データ構造への入力命令の結果としてレコード識別またはフィールド標識は設定されません。ファイルに対するすべての入力および出力命令で結果フィールドにデータ構造が指定されている場合には、入力仕様および出力仕様は必要ありません。

レコードが見つからなかったり、操作でエラーが起こったり、あるいは最後のレコードがすでにリトリーブされていたり (ファイルの終わり) して、入力命令 (CHAIN、READ、READC、READE、READP、READPE) でレコードがリトリーブされない場合には、データは抜き出されず、プログラムのすべてのフィールドが変更されないままとなります。

更新ディスク・ファイルに対する CHAIN、READ、READE、READP、または READPE 命令の命令拡張として N を指定した場合には、レコードはロックしないで読み取られます。命令拡張を指定しない場合には、ファイルが更新ディスク・ファイルであれば、レコードはロックされます。

ファイル命令の実行中に起こった例外およびエラーを処理するには、エラー標識またはファイル・エラー・サブルーチンを指定する必要があります。そうしないと、例外やエラーはデフォルトのエラー処理プログラムで処理されます。

注: 入力および出力仕様を含むサブプロシージャーの入力および出力操作では、同じ名前のローカル変数があっても、常にグローバル名が使用されます。たとえば、メイン・ソース・セクションとサブプロシージャーにフィールド名 TOTALS が定義されている場合には、サブプロシージャーの入力または出力操作ではメイン・ソース・セクションに定義されているフィールドが使用されます。

NULL 値可能フィールドのファイルの処理については、 145 ページの『データベースのヌル値サポート』 を参照してください。

標識設定命令

標識設定命令コードは、以下の通りです。

- 674 ページの『SETOFF (標識をオフにセット)』
- 674 ページの『SETON (標識をオンにセット)』

これらの命令コードは、従来の形式でしか使用できません。フリー・フォーム構文では、EVAL 演算命令を使用して *INxx の値を *ON または *OFF に設定できます。

以下の標識設定組み込み関数は、従来の構文でも、フリー・フォーム構文でも使用できます。
- 446 ページの『%NULLIND (ヌル標識の照会または設定)』

SETON および SETOFF 命令は、71 ～ 76 桁目に指定されている標識をそれぞれオンおよびオフに設定します。これらの桁には、少なくとも 1 つの演算結果標識を指定しなければなりません。

その機能についての説明は、それぞれの演算を参照してください。

情報命令

以下の表には情報演算命令を示しています。

表 43. 情報演算命令

演算	従来の構文	フリー・フォーム構文
時刻および日付の取り出し	697 ページの『TIME (時刻)』	• 414 ページの『%DATE (日付に変換)』 • 469 ページの『%TIME (時刻に変換)』 • 470 ページの『%TIMESTAMP (時刻スタンプに変換)』

TIME 命令によって、プログラムはプログラムの実行中にいつでもシステム時刻およびシステム日付にアクセスすることができます。日付は、ローカル・システムまたは iSeries サーバーからリトリーブすることができます。

初期設定命令

初期設定命令は次のとおりです。
- 520 ページの『CLEAR (消去)』
- 656 ページの『RESET (リセット)』

初期設定命令は、ウィンドウ (入力フィールドのパーツ)、構造 (レコード様式、データ構造、配列、またはテーブル)、または変数 (フィールド、サブフィールド、または標識) のすべてのエレメントの実行時消去およびリセットを行ないます。

これらの演算命令は、従来の構文でも、フリー・フォーム構文でも使用できます。

メモリー管理命令

以下の表にはメモリー管理演算命令を示しています。

表 44. メモリー管理演算命令

演算	従来の構文	フリー・フォーム構文
ストレージの割り振り	485 ページの『ALLOC (ストレージの割り振り)』	407 ページの『%ALLOC (ストレージの割り振り)』
ストレージの解放	527 ページの『DEALLOC (ストレージの解放)』	
ストレージの再割り振り	654 ページの『REALLOC (新規長さのストレージの再割り振り)』	450 ページの『%REALLOC (ストレージの再割り振り)』

表 44. メモリー管理演算命令 (続き)

演算	従来の構文	フリー・フォーム構文
変数のアドレスの取り出し	405 ページの『%ADDR (変数のアドレスの取り出し)』	
プロシージャーのアドレスの取り出し	449 ページの『%PADDR (プロシージャー・アドレスの取り出し)』	

ALLOC 命令は、ヒープ・ストレージを割り振って、結果フィールドのポインターをストレージをポイントするように設定します。ストレージはまだ初期設定されていません。

REALLOC 命令は、結果フィールドのポインターでポイントされてヒープ・ストレージの長さを変更します。新しいストレージが割り振られて、古いストレージの値に初期設定されます。新しいサイズが古いサイズより小さい場合には、データが切り捨てられます。新しいサイズが古いサイズより大きい場合には、コピーされたデータの後のストレージは初期設定されません。古いストレージが解放されます。結果フィールドのポインターは新しいストレージをポイントするように設定されます。

DEALLOC 命令は、結果フィールドのポインターが設定されているヒープ・ストレージを解除します。命令拡張 (N) が指定されている場合には、ポインターは、正常な割り振り解除の後で *NULL に設定されます。

ストレージは、活動化グループが終了すると、暗黙的に解放されます。LR をオンに設定すると、モジュールによって割り振られたヒープ・ストレージは解放されませんが、ヒープ・ストレージに対するポインターは失われます。

ヒープ・ストレージの誤用によって問題が起こることがあります。以下の例に、これを避けるためのシナリオを示してあります。

```
D Fld1         S                  25A    BASED(Ptr1)
D Fld2         S                   5A    BASED(Ptr2)
D Ptr1         S                     *
D Ptr2         S                     *
 ....
C                   ALLOC     25              Ptr1
C                   DEALLOC                   Ptr1
C* この点以後は、基底ポインター Ptr1 が割り振られたストレージを
C* ポイントしないので、Fld1 にアクセスしてはいけません。
C                   CALL      'SOMEPGM'

C* 前の 'SOMEPGM' 呼び出し中に、いくつかのストレージ割り振りが行なわれて
C* いる可能性があります。いずれの場合にも、25 バイトのストレージに 'a'
C* が入れられるので、次の割り当てを行なうことは非常に危険なことです。
C* 現在、そのストレージがどのような目的で使用されているかを知ることは
C* できません。
C                   EVAL      Fld1 = *ALL'a'
```

以下は、さらに問題のある状況です。

- 再割り振りまたは割り振り解除される前にポインターがコピーされると、同様なエラーが起こることがあります。割り振り済みのストレージにポインターをコピ

ーする場合には、そのストレージが割り振り解除または再割り振りされた後で使用されることのないように注意する必要があります。
- ヒープ・ストレージへのポインターがコピーされると、ストレージの割り解除または再割り振りにそのコピーを使用することができます。この場合には、元のポインターは、新しい値に設定されるまで使用してはいけません。
- ヒープ・ストレージへのポインターがパラメーターとして渡されると、被呼側はそのストレージを割り振り解除または再割り振りすることができます。呼び出しが戻された後で、ポインターを通してストレージにアクセスしようとすると問題が起こることになります。
- ヒープ・ストレージへのポインターが *INZSR で設定されると、後でのポインターの RESET で、ポインターがすでに割り振られていないストレージに設定される原因になります。
- 別のタイプの問題は、ヒープ・ストレージへのポインターが (たとえば、消去されるか、あるいは ALLOC 命令で新しいポインターに設定されることによって) 失われた場合に起こることがあります。ポインターが失われると、それがポイントしていたストレージを解放することができません。このストレージは使用不可能で、ポイントしていたストレージを解放することはできません。そのストレージがもはやアドレス指定可能でないことをシステムが認識していないので、このストレージは割り振ることができません。このストレージは、活動化グループが終了するまで解放されません。

メッセージ命令

メッセージ命令 DSPLY は「メッセージ」ウィンドウを表示します。詳細については、544 ページの『DSPLY (「メッセージ」ウィンドウの表示)』を参照してください。

この演算命令は、従来の構文でも、フリー・フォーム構文でも使用できます。

移動命令

以下の表には移動演算命令を示しています。

表 45. 移動演算命令

演算	従来の構文	フリー・フォーム構文
移動	587 ページの『MOVE (転送)』	554 ページの『EVALR (式の評価、右寄せ)』または組み込み関数
配列の移動	603 ページの『MOVEA (配列の転送)』	(使用できません)
左に移動	610 ページの『MOVEL (左につめて転送)』	552 ページの『EVAL (式の評価)』 または組み込み関数

移動命令は、演算項目 2 の全部または一部を結果フィールドに転送します。演算項目 2 は変更されません。

データがどのように移動されるかについては、それぞれの命令を参照してください。MOVE および MOVEL を使用した場合の日付・時刻データの移動方法については、374 ページの『日付・時刻データの移動』を参照してください。

移動命令のソースとターゲットは同じタイプまたは別のタイプとすることができますが、次のようないくつかの制約事項が適用されます。

- ポインターを移動する場合には、ソースとターゲットは同じタイプ (両方とも基底ポインターか、両方ともプロシージャー・ポインター) でなければなりません。
- MOVEA を使用する場合には、ソースとターゲットは同じタイプのものでなければなりません。
- MOVEA を日付、時刻、またはタイム・スタンプ・フィールドに使用することはできません。
- MOVE および MOVEL を浮動フィールドまたはリテラルに使用することはできません。

演算結果標識を指定できるのは、文字、グラフィック、UCS-2、および数値の結果フィールドの場合だけです。MOVE および MOVEL 命令では、結果フィールドが索引付き配列以外の場合には演算結果標識を使用することはできません。MOVEA では、索引付きかどうかに関係なく、結果フィールドが配列の場合には、演算結果標識を使用することはできません。

命令拡張 P を指定できるのは、結果フィールドが文字、グラフィック、UCS-2、または数値の場合だけです。

文字、グラフィック、UCS-2、および数値データの移動

文字フィールドが数値の結果フィールドに移動されると、それぞれの文字のディジット部分が対応する数字に変換されて、結果フィールドに移動されます。ブランクはゼロとして転送されます。MOVE 命令の場合には、右端文字のゾーン部分は対応する符号に変換されて、数値の結果フィールドの右端位置に移動されます。これは、フィールドの符号になります。MOVEL 命令の場合には、右端の文字が移動命令に含まれるかどうかに関係なく、演算項目 2 の右端の文字のゾーン部分が変換されて、結果フィールドの符号として使用されます (演算項目 2 が結果フィールドより短い場合を除く)。

移動命令が数値フィールドの間に指定されている場合には、演算項目 2 フィールドに指定された小数点以下の桁数は無視されます。たとえば、1.00 が小数点以下の桁数 1 桁の 3 桁の数値フィールドに移動されると、結果は 10.0 になります。

演算項目 2 には、文字または数値フィールドに移動するために表意定数 *ZEROS を含めることができます。グラフィック・フィールドの場合に同じ機能を達成するために、ユーザーは *ALLG'xx' をコーディングしなければなりません (ここで、'xx' はグラフィックのゼロを表す)。

文字ソースからグラフィック・フィールドへデータを移動する場合に、ソースが文字、リテラル、名前付き固定情報、または *ALLm の場合には、偶数長で最低 2 バイトでなければなりません。16 進リテラルまたは *ALLx からグラフィック・フィールドに移動する場合には、16 進リテラル (またはパターン) は偶数バイトでなければなりません。

文字フィールドにグラフィック・フィールドへの移動またはグラフィック・フィールドからの移動が含まれる場合には、ソース・フィールドは偶数長で最低 2 バイトでなければなりません。

移動命令が文字から UCS-2 または UCS-2 から文字へのデータの変換に使用される場合には、その文字データにグラフィック文字が含まれることも含まれないこともあるので、移動される文字数は変数になります。たとえば、 UCS-2 の 5 文字は次に変換されます。
- 単一バイトで 5 文字
- 2 バイト文字で 5 文字
- 単一バイトと 2 バイト文字の組み合わせ

結果のデータが長過ぎて結果フィールドに入らない場合には、データが切り捨てられることになります。結果が 1 バイト文字の場合には、結果に完全な文字が入るようにするのはユーザーの責任です。

移動命令に命令拡張 P を指定すると、結果フィールドは、MOVEL および MOVEA の場合には右から、 MOVE の場合には左から埋められます。埋め込み文字は次のとおりです。
- 文字の場合にはブランク
- グラフィックの場合には 2 バイトのブランク
- UCS-2 の場合には UCS-2 のブランク
- 数値の場合には 0 (ゼロ)
- 標識の場合には '0'

埋め込みは操作の後で行なわれます。MOVE または MOVEL を使用してフィールドを配列に移動する場合には、配列のそれぞれのエレメントが埋め込まれます。これらの操作で配列を配列に移動して、結果に演算項目 2の配列より多くのエレメントが入っている場合には、同じ埋め込みが行なわれますが、余分なエレメントには影響がありません。結果フィールドに配列名を持つ MOVEA 命令では、この命令によって影響を受ける最後のエレメントと以後のすべてのエレメントが埋め込まれます。

移動命令で演算結果標識が指定されている場合には、結果フィールドによってどの標識がオンに設定されるかが決まります。結果フィールドが 文字、グラフィック、または UCS-2 フィールドの場合には、75 ～ 76 桁目の演算結果標識しか指定することができません。この標識は、結果フィールドがすべてブランクの場合にオンに設定されます。結果フィールドが数値の場合には、3 つの標識位置すべてが使用されることがあります。これらの標識は以下のようにオンに設定されます。

高 (71 ～ 72)
結果フィールドが 0 より大きい場合にオンに設定されます。

低 (73 ～ 74)
結果フィールドが 0 より小さい場合にオンに設定されます。

等しい (75 ～ 76)
結果フィールドが 0 に等しい場合にオンに設定されます。

日付・時刻データの移動

MOVE および MOVEL 命令コードは、日付、時刻、およびタイム・スタンプ・データ・タイプ・フィールドの移動に使用することができます。

MOVE および MOVEL 命令のコードには次の組み合わせを使用することができます。

- 日付から日付、日付からタイム・スタンプ、日付から文字または数字
- 時刻から時刻、時刻から文字または数字、時刻からタイム・スタンプ
- タイム・スタンプからタイム・スタンプ、タイム・スタンプから日付、タイム・スタンプから時刻、タイム・スタンプから文字または数字
- 文字または数字から日付、文字または数字から時刻、文字または数字からタイム・スタンプ

移動のソースとターゲットの両方が日付、時刻、またはタイム・スタンプ・フィールドの場合には、演算項目 1 はブランクでなければなりません。演算項目 1 がブランクの場合には、日付、時刻、またはタイム・スタンプ・フィールドの形式が使用されます。

そうでない場合には、演算項目 1 に、この命令のソースまたはターゲットである文字または数字フィールドに対応する日付または時刻の形式が入ります。有効な任意の形式を指定することができます。 125 ページの『日付データ』、 144 ページの『時刻データ』、および 145 ページの『タイム・スタンプ・データ』を参照してください。

演算項目 1 を指定する場合には、以下のことに留意してください。
- 時刻形式 *USA は、時刻と数字フィールド間の移動には使用できません。
- 演算項目 1 には、形式 *LONGJUL、*CYMD、*CMDY、および *CDMY を使用することができます。(詳細については、 126 ページの表 15を参照してください。)
- 形式の終わりに指定されたゼロ (0) (たとえば、*MDY0) は、文字フィールドに区切り文字が入っていないことを示します。
- 2 桁の年の形式 (*MDY、*DMY、*YMD、および *JUL) は、1940 から 2039 までの範囲の日付しか表すことができません。3 桁の年の形式 (*CYMD、*CMDY、*CDMY) は、1900 から 2899 までの日付しか表すことができません。この範囲外の日付に 2 桁または 3 桁の年の形式への変換を要求すると、エラーが出されます。
- MOVE および MOVEL を文字または数値からタイム・スタンプへ (またはその逆) の移動に使用すると、その文字または数値にはタイム・スタンプが入っていると見なされます。

演算項目 2 は必須で、文字、数字、日付、時刻、またはタイム・スタンプの値でなければなりません。これには、変換するフィールド、配列、配列エレメント、テーブル名、リテラル、または名前付き固定情報が入ります。

演算項目 2 には次の規則が適用されます。
- 区切り文字は、指定された形式に有効でなければなりません。
- 演算項目 2 が有効な日付または時刻の表現でないか、またはその形式が演算項目 1 で指定された形式と一致しない場合には、エラーが生成されます。
- 演算項目 2 に UDATE または *DATE が入っている場合には、演算項目 1 は任意指定で、ヘッダー仕様の DATEDIT キーワードに対応します。
- 演算項目 2 に UDATE が入っていて、演算項目 1 がコーディングされている場合には、2 桁の年の日付形式でなければなりません。演算項目 2 に *DATE が入っていて演算項目 1 がコーディングされている場合には、4 桁の年の日付形式でなければなりません。

結果フィールドは日付、時刻、タイム・スタンプ、数字、または文字の変数でなければなりません。これは、フィールド、配列、配列エレメント、またはテーブル名とすることができます。日付または時刻は、定義されたその形式または演算項目 1 に指定された形式コードに従って、結果フィールドに入れられます。結果フィールドが数値の場合には、操作の前に区切り文字が除去されます。長さは、区切り文字を除去した後の長さになります。

日付からタイム・スタンプ・フィールドに移動する場合には、タイム・スタンプの時刻とマイクロ秒部分には影響はありませんが、タイム・スタンプ全体が検査されて、正しくなければエラー・メッセージが生成されます。

時刻からタイム・スタンプ・フィールドに移動する場合には、タイム・スタンプのマイクロ秒部分は 000000 に設定されます。日付部分には影響はありませんが、タイム・スタンプ全体が検査されて、正しくない場合にはエラーが生成されます。

文字または数字データが必要以上に長い場合には、左端のデータ (MOVE 命令の右端) だけが使用されます。演算項目 1 で移動するデータの長さが決まることに留意してください。たとえば、演算項目 1 の形式が数値データからの MOVE 命令で *MDY の場合には、演算項目 2 の右端の 6 桁だけが使用されます。

命令拡張 P を指定できるのは、結果が文字または数字の場合だけです。

文字フィールドから日付フィールドへの変換例

図 118 は、日付フィールド間または文字と日付フィールド間で 2 桁と 4 桁の年日付を定義し移動する方法の例を示すものです。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 * 2 つの 8 バイト文字フィールドを定義します。
D CHR_8a          s              8a   inz('95/05/21')
D CHR_8b          s              8a   inz('abcdefgh')
 *
 * 8 バイトの日付フィールドを定義します。デフォルトの 4 桁
 * (*ISO 形式の場合) ではなく 2 桁とするには、2 桁の年形式
 * *YMD で定義されます。D_8a の場合、区切り文字 (.) も指定します。
 * INZ キーワードで指定する日付リテラルの形式は * 制御仕様で指定
 * されたものと同じでなければならないということに注意してください。
 * この場合、何も指定されないので、デフォルトの *ISO となります。
 *
D D_8a            s               d   datfmt(*ymd.)
D D_8b            s               d   inz(d'1995-07-31') datfmt(*ymd)
 *
 * 10 バイトの日付フィールドを定義します。デフォルトでは *ISO 形式です。
D D_10            s               d   inz(d'1994-06-10')
 *
 *  D_10 の値はここで 1995-05-21 となります。
 *
 *  8 文字のフィールドを 10 文字の日付フィールド D_10 に移動します。
 *  これには CHR_8a が初期設定された日付が入りますが、4 桁の
 *  年と D_10 の形式、すなわち 1995-05-21 (*ISO 形式) に
 *  なります。
 *
 *  文字フィールドの形式を指示するには、組み込み関数 %DATE で
 *  形式を指定しなければならないことに注意してください。
 *
 /FREE
    D_10 = %DATE (CHR_8a: *YMD);
    //
    //  10 文字の日付を 8 文字フィールド CHR_8b に移動します。
    //  これには、D_10 に移動したばかりの日付が入りますが、2 桁の
    //  年と *YMD 形式で指示されたデフォルトの区切り文字になります。
    //
    CHR_8b = %CHAR (D_10: *YMD);
    //
    //  10 文字の日付を 8 文字の日付 D_8a に移動します。
    //  これには、* が D_10 に移動されたばかりの日付が入り
    //  ますが、D_8a が (*YMD.) 形式で定義されているので、
    //  2 桁の年と . の区切り文字になります。
    //
    D_8a = D_10;
    //
    //  8 文字の日付を 10 文字の日付 D_10 に移動します。
    //  これには、*  D_8b が初期設定された日付が入りますが、
    //  4 桁の年 (1995-07-31) になります。
    //
    D_10 = D_8b;
    //
    // 最後の移動の後で、フィールドには以下が入ります。
    //  CHR_8b:  95/05/21
    //  D_8a:    95.05.21
    //  D_10:    1995-07-31
    //
    *INLR = *ON;
 /END-FREE
```

図 118. 文字および日付データの移動

ストリング命令

表 46. ストリング演算命令

演算	従来の構文	フリー・フォーム構文
連結	507 ページの『CAT (2 つのストリングの連結)』	+ 演算子
検査	514 ページの『CHECK (文字の検査)』	410 ページの『%CHECK (文字の検査)』
反転検査	517 ページの『CHECKR (逆方向の検査)』	412 ページの『%CHECKR (逆方向に検査)』
Create	462 ページの『%STR (ヌル文字終了ストリングの取り出しまたは保管)』	
置き換え	452 ページの『%REPLACE (文字ストリングの置き換え)』	
スキャン	661 ページの『SCAN (文字ストリングの走査)』	454 ページの『%SCAN (文字のスキャン)』
サブストリング	686 ページの『SUBST (サブストリング)』	466 ページの『%SUBST (サブストリングの取り出し)』
変換	710 ページの『XLATE (変換)』	478 ページの『%XLATE (変換)』
ブランクのトリム	472 ページの『%TRIM (端のブランクをトリム)』、 473 ページの『%TRIML (先行ブランクをトリム)』、または 474 ページの『%TRIMR (末尾ブランクをトリム)』	

ストリング命令には、連結、スキャン、サブストリング、変換、および検査が含まれます。ストリング命令を使用できるのは、文字、グラフィック、または UCS-2 フィールドだけです。

注:

- ストリングは 1 桁目から索引づけされます。
- 演算項目 1、演算項目 2、または結果フィールドに表意定数を使用することはできません。
- 演算項目 1 と結果フィールドまたは演算項目 2 と結果フィールドにデータ構造のオーバーラップがあってはなりません。

グラフィック・フィールドでストリング命令を使用する場合には、演算項目 1、演算項目 2、および結果フィールドのデータはすべてグラフィックでなければなりません。長さ、開始位置、およびグラフィック文字のブランクの数に数値が指定されている場合には、その値は 2 バイト文字を表します。

UCS-2 フィールドでストリング命令を使用する場合には、演算項目 1、演算項目 2、および結果フィールドのデータはすべて UCS-2 でなければなりません。長さ、開始位置、および UCS-2 文字のブランクの数に数値が指定されている場合には、その値は 2 バイト文字を表します。

混合モード文字データのグラフィック部分でストリング命令を使用する場合には、開始位置、長さ、およびブランクの数は 1 バイト文字を表します。

注: データの保全性の保持はユーザーの責任です。

構造化プログラミング命令

以下の表には構造化プログラミング演算命令を示しています。

表 47. 構造化プログラミング演算命令

演算	従来の構文	フリー・フォーム構文
And	486 ページの『ANDxx (AND)』	AND 演算子
Do	535 ページの『DO (実行)』	563 ページの『FOR (For)』
Do Until	537 ページの『DOU (実行の終了)』 または 538 ページの『DOUxx (実行の終了)』	537 ページの『DOU (実行の終了)』
Do While	541 ページの『DOW (実行の時期)』 または 542 ページの『DOWxx (実行の時期)』	541 ページの『DOW (実行の時期)』
Else	546 ページの『ELSE (その他)』	
Else If	547 ページの『ELSEIF (Else If)』	
End	548 ページの『ENDyy (構造化グループの終了)』	
For	563 ページの『FOR (For)』	
If	569 ページの『IF (判断)』 または 570 ページの『IFxx (判断)』	569 ページの『IF (判断)』
Iterate	574 ページの『ITER (繰り返し)』	
Leave	579 ページの『LEAVE (DO/FOR グループの終了)』	
Or	629 ページの『ORxx (OR)』	OR 演算子
Otherwise	630 ページの『OTHER (その他の場合の選択)』	
Select	664 ページの『SELECT (選択グループの始め)』	
When	703 ページの『WHEN (真の場合に選択)』 または 704 ページの『WHENxx (真の場合に選択)』	703 ページの『WHEN (真の場合に選択)』

***制約事項:** Java アプリケーションでは、FOR および ENDFOR はサポートされていません。

ANDxx、DOUxx、DOWxx、IFxx、ORxx、および WHENxx 命令コードにおいて比較を行なう場合の規則は、 361 ページの『比較命令』のところで示したものと同じです。

ANDxx、DOUxx、DOWxx、IFxx、ORxx、および WHENxx 命令では、xx は以下のようにすることができます。

xx **意味**

GT 演算項目 1 は演算項目 2 より大きい。

LT 演算項目 1 は演算項目 2 より小さい。

EQ 演算項目 1 は演算項目 2 に等しい。

NE 演算項目 1 は演算項目 2 と等しくない。

GE 演算項目 1 は演算項目 2 と等しいかそれより大きい。

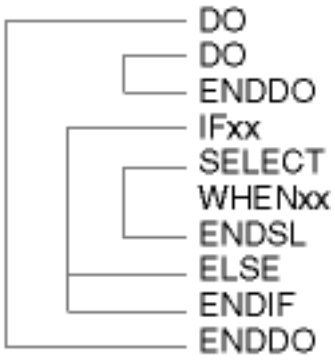
LE 演算項目 1 は演算項目 2 と等しいかそれより小さい。

ENDyy 命令では yy は以下のようにすることができます。

yy	意味
CS	CASxx 命令の終わり。
DO	DO、DOUxx、および DOWxx 命令の終わり。
FOR	FOR 命令の終わり。
IF	IFxx 命令の終わり。
SL	SELECT 命令の終わり。
ブランク	構造化命令の終わり。

注: ENDyy 命令の yy は任意指定です。

構造化グループ (この場合には do グループ) に別の完全な構造化グループが入っている場合には、それらは一緒に、ネストされた構造化グループを形成します。構造化グループは、最大 10 レベルの深さまでネストすることができます。以下は、深さが 3 レベルのネストされた構造化グループの例です。



構造化グループを指定する時には、以下に留意してください。

- ネストされたそれぞれの構造化グループは、外側の構造化グループ内に完全に組み込まれていなければなりません。
- それぞれの構造化グループには、DO、DOUxx、DOWxx、FOR、IFxx、または SELECT 命令のいずれかとそれに関連した ENDyy 命令が入っていなければなりません。
- 構造化グループの外側から構造化グループ内に分岐すると、好ましくない結果になることがあります。

サブルーチン命令

サブルーチン命令は次のとおりです。
- 488 ページの『BEGACT (アクション・サブルーチンの開始)』
- 550 ページの『ENDACT (アクション・サブルーチンの終了)』
- 492 ページの『BEGSR (ユーザー・サブルーチンの開始)』
- 551 ページの『ENDSR (ユーザー・サブルーチンの終了)』
- 558 ページの『EXSR (ユーザー・サブルーチンの起動)』
- 581 ページの『LEAVESR (サブルーチンの終了)』
- 505 ページの『CASxx (サブルーチンの条件付き起動)』 (従来の構文のみ)

CASxx を除くこれらすべての命令は、従来の構文でも、フリー・フォーム構文でも使用できます。

サブルーチンは、そのプログラムの中で数回処理することができるプログラム内の演算仕様のグループです。

サブルーチン仕様は、プログラムで処理できる他のすべての演算命令に従わなければなりませんが、PLIST、PARM、KLIST、KFLD、および DEFINE 命令は ENDSR 命令 (1 つのサブルーチンの終わり) と BEGSR 命令 (別のサブルーチンの始め) との間、またはすべてのサブルーチンの後に指定することができます。サブルーチンは、演算仕様の任意の場所で EXSR または CASxx 命令を使用して呼び出すことができます。サブルーチン行は、7 - 8 桁目の SR で識別することができます。サブルーチン行の 7 - 8 桁目に有効な項目は SR、AN、 OR、またはブランクだけです。

サブルーチンのコーディング方法については、 558 ページの『ユーザー・サブルーチンのコーディング』を参照してください。

テスト命令

テスト命令は次のとおりです。
- 690 ページの『TEST (日付、時刻、タイム・スタンプのテスト)』
- 693 ページの『TESTB (ビットのテスト)』
- 695 ページの『TESTN (数字のテスト)』
- 696 ページの『TESTZ (ゾーンのテスト)』

これらの命令の結果は、演算結果標識で示されます。

TEST は、従来の構文でも、フリー・フォーム構文でも使用できます。他の命令は、従来の形式でしか使用できません。

GUI 命令

VisualAge RPG 命令は次のとおりです。
- 488 ページの『BEGACT (アクション・サブルーチンの開始)』
- 524 ページの『CLSWIN (ウィンドウのクローズ)』
- 544 ページの『DSPLY (「メッセージ」ウィンドウの表示)』
- 550 ページの『ENDACT (アクション・サブルーチンの終了)』
- 566 ページの『GETATR (属性の検索)』

- 653 ページの『READS (選択されたレコードの読み取り)』
- 666 ページの『SETATR (属性の設定)』
- 675 ページの『SHOWWIN (ウィンドウの表示)』
- 679 ページの『START (コンポーネント開始またはローカル・プログラム呼び出し)』
- 681 ページの『STOP (コンポーネントの停止)』

VisualAge RPG は、アプリケーションのユーザー・インターフェース (たとえば、SHOWWIN) で機能するか、あるいは命令のコンポーネント (たとえば、STOP) で機能します。その機能についての説明は、それぞれの演算を参照してください。

修飾 GUI パーツ属性アクセス

次の修飾命名構文が、%GETATR および %SETATR 組み込み関数の代わりとして、式またはフリー・フォーム演算での GUI パーツ属性へのアクセスでサポートされます。

```
          <window-name>.<part-name>.<attribute-name>

eg.
     C                   eval      caltest.disparrows.checked =
     C                                caltest.calendar.montharrow
```

注:

1. %WINDOW と %PART はこの形式では無効です。
2. GUI パーツ属性へのこの修飾アクセスをサポートするために、コンポーネントの GUI 定義に定義されているウィンドウ名が、プロシージャー内でも、プログラム内のシンボル名 (フィールド名など) として予約されます。
3. この修飾属性アクセスは、パーツの対応するプログラム・フィールドには影響しません。属性値およびプログラム・フィールド内の値が同一であることを保証するには、必要に応じて割り当てを行います。

   これは、プログラム・フィールドがマップされた属性 (入力フィールド・パーツの TEXT 属性など) にのみ適用されます。

   ```
   C                   EVAL      ENT0000B = INVENTORY.ENT0000B.TEXT
   ```

   また、別の指示としては次があります。

   ```
    /FREE
         ENT0000B = *BLANKS;
         inventory.ent0000b.text = ENT0000B;
    /END-FREE
   ```

第 24 章 式

式とは、フリー・フォームの構文を使用してプログラム論理を表現する 1 つの方法です。これを使用して、固定形式のステートメントより簡潔な方法でプログラム・ステートメントを書き込むことができます。

式は、次の通りオペランドと演算命令の単純なグループです。

```
A+B*21
STRINGA + STRINGB
D = %ELEM(ARRAYNAME)
*IN01 OR (BALANCE > LIMIT)
SUM + TOTAL(ARRAY:%ELEM(ARRAY))
'The tax rate is ' + %editc(tax : 'A') + '%.'
```

式は、以下のステートメントにコーディングすることができます。

- 503 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』
- 537 ページの『DOU (実行の終了)』
- 541 ページの『DOW (実行の時期)』
- 552 ページの『EVAL (式の評価)』
- 554 ページの『EVALR (式の評価、右寄せ)』
- 563 ページの『FOR (For)』
- 569 ページの『IF (判断)』
- 659 ページの『RETURN (呼び出し元への戻り)』
- 703 ページの『WHEN (真の場合に選択)』

384 ページの図 119 は、式の幾つかの使用例を示しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* DOU グループ中の演算命令は、論理式が真になるまで繰り返えされます。
* すなわち、COUNTER が MAXITEMS より小さくなるまでか、あるいは
* 標識 03 がオンになるまでのいずれかです。
/FREE
    dou  counter < MAXITEMS or *in03;
        enddo;

    // DUEDATE (日付変数) が 1994 年 12 月 31 日より以前
    // の日付の場合には、IF 演算命令によって制御される演算
    // 命令が実行されます。
    if DueDate < D'12-31-94';
     endif;

    // この数式では、COUNTER には COUNTER プラス 1 の値
    // が割り当てられます。
    Counter = Counter + 1;

    // この数式は組み込み関数を使用して、配列 ARRAY の
    // エレメント数を変数 ARRAYSIZE に割り当てます。
    ArraySize = %elem (Array);

    // この式は利息を計算して、変数 INTEREST に入れられる
    // 結果を四捨五入します。
    eval(h) Interest = Balance * Rate;

    // この文字式は、連結を使用して名前および数値からセンテンスをビルド
    // します。組み込み関数の %CHAR、%EDITC、%EDITW、または %EDITFLT を
    // 使用して数値を文字データに変換できます。このステートメントは
    // 'Id number for John Smith is 231 364' を作成します。
    String = 'Id number for '
             + %trimr (First) + ' ' + %trimr (Last)
             + ' is ' + %editw (IdNum: '   &   ');

    // この式は、日付に 10 日の期間を加算します。
    DueDate = OriginalDate + %days(10);

    // この式は、2 つの時刻値の間の差 (秒数) を決定します。
    Seconds = %diff (CompleteTime: t'09:00:00': *seconds);

    // この式は、日付値と時刻値を結合して、タイム・スタンプ値
    // にします。
    TimeStamp = TransactionDate + TransactionTime;
/END-FREE
```

図 119. 式の例

式の一般規則

一般の次の規則がすべての式に適用されます。

- 式は、演算仕様の拡張演算項目 2 の項目に、またはフリー・フォーム演算の命令コードの後にコーディングします。
- 式は、複数の仕様に続行することができます。継続の指定では、使用可能な項目は 6 桁目および拡張演算項目 2 の項目の **C** だけです。

  式がリテラルまたは名前内で分割されていないかぎり、特別な継続文字は不要です。
- あいまいさを解決する場合にかぎり、ブランク (括弧と同様に) が必要となります。ただし、読みやすくするためには、これを使用することができます。

RPG は、式のそれぞれのトークンを解析する時に可能なかぎりの文字数を読み取ることに注意してください。たとえば、次のようになります。
  – X**DAY は、X の DAY 乗です。
  – X* *DAY は、X に *DAY を掛けたものです。
- 制御仕様の TRUNCNBR キーワードは式内で行なわれる演算に適用されます。式演算の時にオーバーフローが起こった場合には、常に例外が発行されます。

式オペランド

オペランドは、フィールド名、名前付き固定情報、リテラル、または値を戻すプロトタイプ・プロシージャーのいずれかとすることができます。さらに、どれかの演算命令の結果を、他の演算命令のオペランドとして使用することもできます。たとえば、A+B*21 の式では、B*21 の結果が加算演算命令に対するオペランドとなります。

式の演算子

式の演算子は次のいずれかとすることができます。

単項演算

単項演算は、演算命令の後に、1 つのオペランドを続けて指定してコーディングされます。単項演算は次の通りです。

+ 単項に演算命令をプラスすると、数字オペランドの値が維持されます。

– 単項から演算命令をマイナスすると、数字オペランドの値が否定されます。

NOT 標識オペランドの値が '0' の場合には論理否定演算命令は '1' を戻し、標識オペランドが '1' の場合には '0'を戻します。どの比較演算または演算 **AND** または **OR** の結果はタイプ標識の値となります。

2 項演算

2 項演算は、2 つのオペランドの間にそのオペランドを指定してコーディングされます。2 項演算は次の通りです。

+ この演算命令の意味はオペランドのタイプによって異なります。これは、次の場合に使用することができます。
- 2 つの数値の加算
- 期間を日付、時刻、またはタイム・スタンプに加算します。
- 2 つの文字、2 つのグラフィック、または 2 つの UCS-2 値を連結
- 数字オフセットの基底ポインターへの加算

– この演算命令の意味はオペランドのタイプによって異なります。これは、次の場合に使用することができます。
- 2 つの数値の減算
- 期間を日付、時刻、またはタイム・スタンプから減算します。
- 数字オフセットの基底ポインターからの減算
- 2 つのポインターの減算

***** 乗算演算は、2 つの数値を乗算するために使用されます。

/ 除算演算は、2 つの数値を除算するために使用されます。

****** 指数演算は、別の累乗に数値を乗算Tするために使用されます。

= 2 つのオペランドが等しい場合には、等価演算命令は '1' を戻し、そうでない場合には '0' を戻します。

<> 2 つのオペランドが等しい場合には、比較演算命令は '0' を戻し、そうでない場合には '1' を戻します。

> 最初のオペランドが 2 番目のオペランドより大きい場合には、より大演算命令は '1' を戻します。

>= 最初のオペランドが 2 番目のオペランドと等しいかより大きい場合には、より大か等しい演算命令は '1' を戻します。

< 最初のオペランドが 2 番目より小さい場合には、より小演算命令は '1' を戻します。

<= (より小か等しい
最初のオペランドが 2 番目と等しいかより小さい場合には、より小か等しい演算命令は '1' を戻します。

AND 両方のオペランドが標識 '1' の値をもつ場合には、論理 AND 演算命令は '1' を戻します。

OR いずれかのオペランドが標識 '1' の値をもつ場合には、論理 OR 演算命令は '1' を戻します。

組み込み関数

組み込み関数は 399 ページの『第 25 章 組み込み関数』 に説明されています。

ユーザー定義関数

値を戻すどのプロトタイプ・プロシージャーも、式の中で使用することができます。プロシージャーの呼び出しは 、プロシージャーの戻り値と同じタイプの値が使用される任意の位置に入れることができます。たとえば、プロシージャー **MYFUNC** は文字値を戻すものとします。次の例は、**MYFUNC** に対する 3 つの呼び出しを示しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
     if  MyFunc (string1) = %trim (MyFunc (string2));
         %subst(X(3))= MyFunc('abc');
     endif;
/END-FREE
```

図 120. 式でのプロトタイピングされたプロシージャーの使用

ユーザー定義関数の詳細については、65 ページの『第 6 章 サブプロシージャーおよびプロトタイプ』 を参照してください。

演算命令の優先順位

演算命令の優先順位は、式の中で演算命令を実行する順序を決定します。優先順位の高い演算命令は、優先順位の低い演算命令より前に実行されます。

括弧は最高位の優先順位をもつので、括弧内の演算命令は常に最初に実行されます。

同じ優先順位の演算命令は左から右の順序に評価されますが、** は例外で、これは右から左に評価されます。

式は左から右に評価されますが、これは、そのオペランドも左から右に評価されることを意味するものではないことに注意してください。詳細な考慮事項については、398 ページの『評価の順序』 を参照してください。

このリストは、演算子の優先順位を最高位から最下位までを示しています。

1. ()
2. 組み込み関数
3. 単項 +、単項 –、NOT
4. **
5. *,/
6. 2 項 +、2 項 –
7. =,>=,>,<=,<,<>
8. AND
9. OR

図 121 では、優先順位でどのようになるかを示しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* 以下の 2 つの演算命令は、オペランドの順序と演算子が
* 同じですが、異なる結果を生みます。想定は以下の通りです。
* PRICE = 100、DISCOUNT = 10、および TAXRATE = 0.15。
* 最初の EVAL の結果、TAX は 98.5 となります。
* 乗算は減算より優先順位が高いので、最初に実行される
* 演算命令は DISCOUNT * TAXRATE となります。その演算命令の
* 結果 (1.5) から PRICE が引かれます。
/FREE
    TAX = PRICE - DISCOUNT * TAXRATE;

    // 2 番目の EVAL の結果、TAX は 13.50 となります。
    // 括弧は最高位の優先順位となるので、括弧内の演算命令
    // が最初に実行されて、その演算命令の結果 (90) が
    // TAXRATE に乗算されます。

    TAX = (PRICE - DISCOUNT) * TAXRATE;
/END-FREE
```

図 121. 優先順位の例

データ・タイプ

式の中では、すべてのデータ・タイプを使用することができます。ただし、特定の演算命令だけが特定のデータ・タイプをオペランドとしてサポートしています。たとえば、 *演算命令だけがオペランドとして数値を使用することができます。リレーショナル演算および論理演算はタイプ標識の値を戻し、これは、文字データの特

殊なタイプです。結果として、どれかのリレーショナルまたは論理の結果を、文字オペランドが必要となるどれかの演算命令に対するオペランドとして使用することができます。

式オペランドによってサポートされるデータ・タイプ

次の表は、式オペランドをサポートするデータ・タイプを要約しています。

- 表 48 は、それぞれの単項演算子で許可されるオペランドのタイプと、その結果のタイプを記述しています。
- 表 49 は、それぞれの 2 項演算子で許可されるオペランドのタイプと、その結果のタイプを記述しています。
- 389 ページの表 50 は、それぞれの組み込み関数で許可されるオペランドのタイプと、その結果のタイプを記述しています。プロトタイプ・プロシージャーは、プロトタイプ定義に定義されたどんなデータ・タイプもサポートします。

表 48. 単項演算でサポートされるタイプ

演算	オペランド・タイプ	結果のタイプ
– (否定)	数値	数値
+	数値	数値
NOT	標識	標識

表 49. 2 項演算でサポートされるオペランド

演算子	オペランド 1 タイプ	オペランド 2 タイプ	結果のタイプ
+ (加算)	数値	数値	数値
+ (加算)	日付	期間	日付
+ (加算)	時刻	期間	時刻
+ (加算)	タイム・スタンプ	期間	タイム・スタンプ
– (減算)	数値	数値	数値
- (減算)	日付	期間	日付
- (減算)	時刻	期間	時刻
- (減算)	タイム・スタンプ	期間	タイム・スタンプ
* (乗算)	数値	数値	数値
/ (除算)	数値	数値	数値
** (指数)	数値	数値	数値
+ (連結)	文字	文字	文字
+ (連結)	グラフィック	グラフィック	グラフィック
+ (連結)	UCS-2	UCS-2	UCS-2
+ (ポインターからのオフセットの加算)	基底ポインター	数値	基底ポインター
- (減算ポインター)	基底ポインター	基底ポインター	数値
- (ポインターからのオフセットの減算)	基底ポインター	数値	基底ポインター
注: 次の演算命令では、オペランドは任意のタイプのものにすることができますが、2 つのオペランドは同じタイプのものでなければなりません。			
= (等しい)	不特定	不特定	標識

表 49. 2 項演算でサポートされるオペランド (続き)

演算子	オペランド 1 タイプ	オペランド 2 タイプ	結果のタイプ
>= (より大か等しい)	不特定	不特定	標識
> (より大)	不特定	不特定	標識
<= (より小か等しい)	不特定	不特定	標識
< (より小)	不特定	不特定	標識
<> (等しくない)	不特定	不特定	標識
AND (論理 AND)	標識	標識	標識
OR (論理 OR)	標識	標識	標識

表 50. 組み込み関数でサポートされるタイプ

演算	オペランド	結果のタイプ
%ABS	数値	数値
%ALLOC	数値	ポインター
%CHAR	グラフィック、数値、UCS-2、日付、時刻またはタイム・スタンプ	文字
%CHECK	文字、グラフィック、または UCS-2 {: 数値}	数値
%CHECKR	文字、グラフィック、または UCS-2 {: 数値}	数値
%DATE	{文字、数値、またはタイム・スタンプ {: 日付形式}}	日付
%DAYS	数字	数値 (期間)
%DEC	数値 {: 数値固定情報 : 数値固定情報}	数値 (パック)
%DECH	数値 : 数値固定情報 : 数値固定情報	数値 (パック)
%DECPOS	数字	数値 (符号なし)
%DIFF	日付、時刻、またはタイム・スタンプ: 日付、時刻、またはタイム・スタンプ: 単位	数値 (期間) (両方と互換性あり)
%DIV	数値 : 数値	数値
%EDITC	非浮動数値 : 長さ 1 の文字固定情報 {: *CURSYM | *ASTFILL | 文字通貨記号}	文字 (固定長)
%EDITFLT	数字	文字 (固定長)
%EDITW	非浮動数字 : 文字固定情報	文字 (固定長)
%EOF	{ファイル名}	標識
%EQUAL	{ファイル名}	標識
%ERROR		標識
%FLOAT	数字	数値 (浮動)
%FOUND	{ファイル名}	標識
%GETATR	文字 : 文字 : 文字	ポインターを除いた不特定タイプ
%GRAPH	文字、グラフィック、または UCS-2 {: ccsid}	グラフィック
%HOURS	数字	数値 (期間)

表 50. 組み込み関数でサポートされるタイプ (続き)

演算	オペランド	結果のタイプ
%INT	数字	数値 (整数)
%INTH	数字	数値 (整数)
%LEN	不特定	数値 (符号なし)
%LOOKUPxx	不特定 : 不特定配列 {: 数値 {: 数値}}	数値
%MINUTES	数字	数値 (期間)
%MONTHS	数字	数値 (期間)
%MSECONDS	数字	数値 (期間)
%OCCUR	複数回繰り返しデータ構造	複数回繰り返しデータ構造
%OPEN	ファイル名	標識
%REALLOC	ポインター : 数値	ポインター
%REM	数値 : 数値	数値
%REPLACE	文字 : 文字 {: 数値 {: 数値}}	文字
%REPLACE	グラフィック : グラフィック {: 数値 {: 数値}}	グラフィック
%REPLACE	UCS-2 : UCS-2 {: 数値 {: 数値}}	UCS-2
%SCAN	文字 : 文字 {: 数値}	数値 (符号なし)
%SCAN	グラフィック : グラフィック {: 数値}	数値 (符号なし)
%SCAN	UCS-2 : UCS-2 {: 数値}	数値 (符号なし)
%SECONDS	数字	数値 (期間)
%SQRT	数字	数値
%SETATR	文字 : 文字 : 文字	
%STATUS	{ファイル名}	数値 (ゾーン 10 進数)
%STR	基底ポインター {: 数値}	文字
注: %STR が式の左側が現れると、2 番目のオペランドが必要です。		
%SUBDT	D日付、時刻、またはタイム・スタンプ: 単位	数値
%SUBST	文字 : 数値 {: 数値}	文字
%SUBST	グラフィック : 数値 {: 数値}	グラフィック
%SUBST	UCS-2 : 数値 {: 数値}	UCS-2
%THIS		オブジェクト
%TIME	{文字、数値、またはタイム・スタンプ {: 時刻形式}}	時刻
%TIMESTAMP	{文字、数値、またはタイム・スタンプ {: タイム・スタンプ形式}}	タイム・スタンプ
%TLOOKUPxx	不特定テーブル: 不特定テーブル {: 不特定}	標識
%TRIM	文字	文字
%TRIM	グラフィック	グラフィック
%TRIM	UCS-2	UCS-2
%TRIML	文字	文字

表 50. 組み込み関数でサポートされるタイプ (続き)

演算	オペランド	結果のタイプ
%TRIML	グラフィック	グラフィック
%TRIML	UCS-2	UCS-2
%TRIMR	文字	文字
%TRIMR	グラフィック	グラフィック
%TRIMR	UCS-2	UCS-2
%UCS2	文字またはグラフィック{:ccsid}	可変長の UCS-2 値
%UNS	数字	数値 (符号なし)
%UNSH	数字	数値 (符号なし)
注: 次の組み込み関数では、引き数はリテラル、名前付き固定情報または変数でなければなりません。		
%XFOOT	数字	数値
%XLATE	文字、グラフィック、または UCS-2 : 文字、グラフィック、または UCS-2 : 文字、グラフィック、または UCS-2{: 数値}	文字、グラフィック、または UCS-2
%YEARS	数字	数値 (期間)
%PADDR	文字	プロシージャーまたはプロトタイプ・ポインター
%SIZE	不特定 {: *ALL}	数値 (符号なし)
注: 次の組み込み関数では、引き数は変数でなければなりません。ただし、配列指数が指定された場合には、これは、有効などれかの数式とすることができます。		
%ADDR	不特定	基底ポインター
%ELEM	不特定	数値 (符号なし)
%NULLIND	不特定	標識

数字中間結果の形式

数値フィールドを含む 2 項演算では、中間結果の形式は、そのオペランドの形式によって異なります。

演算子 +, -, および * の場合:

- 少なくとも 1 つのオペランドが浮動形式の場合には、その結果は浮動形式となります。
- そうでない場合、少なくとも 1 つのオペランドがパック 10 進数、ゾーン 10 進数、または 2 進数形式であれば、その結果はパック 10 進数形式となります。
- そうでない場合、少なくとも 1 つのオペランドが整数形式であれば、結果は整数形式となります。
- そうでない場合には、符号なし形式となります。
- 浮動形式でない数値リテラルの場合:
  - リテラルが符号なし整数の範囲内にある場合には、リテラルは符号なし整数と見なされます。
  - そうでない場合、リテラルが整数の範囲内にあると、リテラルは整数と見なされます。
  - そうでない場合には、リテラルはパック 10 進数と見なされます。

/ 演算子の場合:

1 つのオペランドが浮動である場合、あるいは制御仕様に FLTDIV キーワードが指定されている場合には、/ 演算子の結果は浮動になります。そうでない場合には、その結果はパック 10 進数となります。

** 演算子の場合:

結果は浮動形式で表現されます。

パフォーマンスおよび 8 バイト演算

デフォルトによって、コンパイラーは 4 バイトの演算を実行します。8 バイトの演算が行なわれるのは、少なくとも 1 つのオペランドが 8 バイトの整数の場合だけです。パフォーマンスの面から見れば、8 バイト演算は費用がかかるので、避けるようにしてください。

数値演算の精度の規則

個々の演算結果を常に指定する必要がある固定形式の演算コードとは異なり、RPG では、式の中の各演算の結果の形式および精度を決定する必要があります。

演算に浮動、整数、または符号なし形式の結果がある場合には、その精度はその形式の最大サイズとなります。整数および符号なしの演算では 4 バイトの値が出され、浮動演算では 8 バイトの値が出されます。

ただし、演算がパック 10 進数、ゾーン 10 進数、または 2 進数形式の場合には、その結果の精度はそのオペランドの精度によって異なります。

比較的単純な式が予想外の結果になる可能性があるので、精度の規則には注意する必要があります。たとえば、乗算の 2 つのオペランドが十分な大きさの場合には、乗算の結果の小数部はゼロとなります。20 桁の 2つの数字を乗算している場合には、その乗算で起こりうるすべての結果を入れるには、40 桁の結果が必要です。ただし、RPG は 30 桁までの数値しかサポートしないので、その結果の行末が調整されます。この場合には、10 桁の数値が結果から除去されます。

中間値のサイズの制御に使用できる精度の規則は次の通り 2 つあります。

1. デフォルトの規則では、数値のオーバーフローの可能性を最小限にするために、可能なかぎり大きな中間結果を指定します。特定の場合には、その結果がかなり大きい場合には、小数点以下の桁数ゼロの結果が出されることがあります。
2. 「結果の小数点以下の桁数」の精度の規則はデフォルトの規則と同じになりますが、ステートメントに数値変数に対する割り当て、または特定の小数点以下の桁数に対する変換が含まれている場合には、中間結果の小数点以下の桁数はその結果に必要な小数点以下の桁数以下になることはない、という点が異なります。

   実際に、数式のコーディングの時にコンパイル・リストを試験する場合、正確な精度について考慮する必要はありません。診断メッセージには、中間結果の小数点以下の桁数が除去されることが示されます。式に割り当てが含まれる場合には、小数点以下の桁数は、演算コード拡張 **(R)** のコーディングによりステートメントの「結果小数点以下の桁数」の精度の規則を使用して維持されるようにすることができます。

「結果小数点以下の桁数」の精度の規則を使用できない (たとえば、関係式の中) 場合には、組み込み関数 **%DEC** を使用して、副次式の結果を、小数点以下の桁数が失われないようにできる低い精度に変換することができます。

デフォルト精度の規則の使用

デフォルトの精度の規則を使用して、数字のオーバーフローの可能性を最小限にするために、式の小数部の中間精度が計算されます。ただし、大きな 10 進数の式に幾つかの演算が含まれている場合には、その中間で小数点以下の桁数がゼロとなる場合があります。(特に、その式に 2 つ以上のネストされた除算がある場合。) これは、割り当ての中では特に、プログラマーが予測したものとは異なるものである可能性があります。

10 進数の中間の精度を決定する時には、次の 2 つのステップが実行されます。

1. 結果に必要な精度または「自然な」精度が計算されます。
2. 自然な精度が 30 桁以上の場合には、30 に収まるように精度が行末調整されます。これには、まず初めに小数点以下の桁数を減らし、必要な場合には、中間の桁数を減らす、というようにします。

このやり方がデフォルトであり、これは、モジュール全体 (制御仕様キーワード EXPROPTS(*MAXDIGITS)を使用) または単純なフリー・フォームの式 (演算コード拡張 M を使用) に対して指定することができます。

中間結果の精度

表 51 には、デフォルトの精度の規則が詳細に記述されています。

表 51. 中間結果の精度

演算	結果精度
注: 次の演算は数字の結果を作成します。Ln はオペランドの長さ (桁数) であり、ここの n は、結果の場合には r であり、あるいはオペランドを示す数字のいずれかです。Dn は小数点以下の桁数であり、ここの n は、結果の場合には r であり、あるいはオペランドを示す数字のいずれかです。T は一時的な値です。 どれかのオペランドに浮動小数点表示がある (たとえば、これが指数演算子の結果である) 場合には、その結果も浮動小数点の値となり、精度の規則は適用されません。浮動小数点の値には、倍精度浮動小数点表示から使用できる精度があります。	
N1+N2	T=min (max (L1-D1, L2-D2)+1, 30) Dr=min (max (D1,D2), 30-t) Lr=t+Dr
N1-N2	T=min (max (L1-D1, L2-D2)+1, 30) Dr=min (max (D1,D2), 30-t) Lr=t+Dr
N1*N2	Lr=min (L1+L2, 30) Dr=min (D1+D2, 30-min ((L1-D1)+(L2-D2), 30))

表 51. 中間結果の精度 (続き)

演算	結果精度
N1/N2	Lr=30 Dr=max (30-((L1-D1)+D2), 0)
N1**N2	Double float
注: 次の演算は文字の結果を作成します。Ln はオペランドの長さを文字数で表します。	
C1+C2	Lr=min(L1+L2,65535)
注: 次の演算は DBCS の結果を作成します。Ln はオペランドの長さを DBCS 文字数で表します。	
D1+D2	Lr=min(L1+L2,16383)
注: 次の演算はサブタイプ標識のあるタイプ文字の結果を作成します。その結果は常に標識値 (1 文字) です。	
V1=V2	1 (標識)
V1>=V2	1 (標識)
V1>V2	1 (標識)
V1<=V2	1 (標識)
V1<V2	1 (標識)
V1<>V2	1 (標識)
V1 AND V2	1 (標識)
V1 OR V2	1 (標識)

デフォルトの精度規則の例

この例は、デフォルトの精度規則の機能を示します。

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D FLD1            S             15P 4
D FLD2            S             15P 2
D FLD3            S              5P 2
D FLD4            S              9P 4
D FLD5            S              9P 4
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C                   EVAL      FLD1 = FLD2/(((FLD3/100)*FLD4)+FLD5)
                                         (    1   )
                                        (         2    )
                                       (              3    )
                                  (     4                    )
```

図 122. 中間結果の精度

上記の演算仕様が処理されると、FLD1 に割り当てられる結果の値は、必要な小数点以下の桁数 4 桁ではなく、小数部がゼロの精度になります。その理由は、最後の評価 (上記の例では、 **4**) でこれを得ると、係数の位取りに対する数値は負となるからです。理由を調べるためには、その式の評価方法を調べてください。

1 Evaluate FLD3/100

規則:

```
Lr = 30
Dr = max(30-((L1-D1)+D2),0)
   = max(30-((5-2)+0),0)
   = max(30-3,0)
   = 27
```

2 Evaluate (Result of 1 * FLD4)

規則:

```
Lr = min(L1+L2,30)
   = min(30+9,30)
   = 30
Dr = min(D1+D2,30-min((L1-D1)+(L2-D2),30))
   = min(27+4,30-min((30-27)+(9-4),30))
   = min(31,30-min(3+5,30)
   = min(31,30-8)
   = 22
```

3 Evaluate (Result of 2 + FLD5)

規則:

```
T  = min(max(L1-D1,L2-D2)+1,30)
   = min(max(30-22,9-4)+1,30)
   = min(max(8,5)+1,30)
   = min(9,30)
   = 9
Dr = min(max(D1,D2),30-T)
   = min(max(22,4),30-9)
   = min(22,21)
   = 21
Lr = T + Dr
   = 9 + 21 = 30
```

4 Evaluate FLD2/Result of 3

規則:

```
Lr = 30
Dr = max(30-((L1-D1)+D2),0)
   = max(30-((15-2)+ 21),0)
   = max(30-(13+21),0)
   = max(-4,0)     ****  NEGATIVE NUMBER TO WHICH FACTOR IS SCALED   ****
   = 0
```

この問題を避けるためには、最初の評価が除算ではなくて乗算となるように (すなわち、FLD3 * 0.01) するか、あるいは %DEC 組み込み関数を使用して副次式 FLD3/100: %DEC(FLD3/100 : 15 : 4)を設定するか、あるいは命令拡張 (R) を使用して、小数点以下の桁数が 4 以下にならないようにするため、上記の式を変更することができます。

「結果の小数点以下の桁数」精度の規則の使用

「結果の小数点以下の桁数」精度規則では、小数点以下の桁数が割り当ての結果の小数点以下の桁数より小さくならないように、10 進数の中間の精度が計算されることを意味します。これは、次に指定されます。

1. 制御仕様の **EXPROPTS(*RESDECPOS)** これを使用して、モジュール全体に対してこのやり方を指定します。
2. フリー・フォームの演算に指定される演算コード拡張 **R**

結果の小数点以下の桁数の規則は次の環境で適用されます。

1. 結果の小数点以下の桁数の精度規則が適用されるのは、パック 10 進数の中間結果に対してだけです。この動作は、整数、符号なし、または浮動の結果をもつ演算の中間結果には適用されません。
2. 結果の小数点以下の桁数の精度規則が適用されるのは、10 進数ターゲット (パック、ゾーン、または 2 進数) に対する割り当て (明示または暗黙のいずれか) がある場合だけです。

   次の状態では、これが起こることがあります。
   a. **EVAL** ステートメントでは、最小の小数点以下の桁数が割り当てのターゲットの小数点以下の桁数によって提供されて、その割り当ての右側で式に適用されます。また、そのステートメントに四捨五入も適用される場合には、その最小の小数点以下の桁数 (最小が 30 桁以下である場合) に余分の 1 桁が追加されます。
   b. **RETURN** ステートメントでは、最小の小数点以下の桁数は、プロシージャーの **PI** 仕様に定義された戻り値の小数点以下の桁数によって指定されます。また、そのステートメントに四捨五入も適用される場合には、その最小の小数点以下の桁数 (最小が 30 桁以下である場合) に余分の 1 桁が追加されます。
   c. **VALUE** または **CONST** パラメーターでは、最小の小数点以下の桁数は、仮パラメーター (プロシージャー・プロトタイプに指定される) の小数点以下の桁数によって指定され、渡されるパラメーターとして指定された式に適用されます。
   d. 組み込み関数 **%DEC** および明示的な長さおよび小数点以下の桁数の指定された **%DECH** では、最小の小数点以下の桁数は、組み込み関数の 3 番目のパラメーターによって指定され、最初のパラメーターとして指定された式に適用されます。

   小数点以下の桁数の最小値は、上記の演算の他のものによって上書きされないかぎり、その副次式全体に適用されます。四捨五入が指定された場合 (**H** 演算コード拡張として、あるいは組み込み関数 **%DECH** によって) には、中間結果の小数点以下の桁数は *N+1* 以下にはなりません。ここの *N* はその結果の小数点以下の桁数です。
3. 結果の小数点以下の桁数の規則は、対応する結果がない条件式には適用されません。(特定の精度に対して比較を行なう必要がある場合には、その 2 つの引き数で **%DEC** または **%DECH**を使用する必要があります。)

   また、最小の小数点以下の桁数が指定された式の中に条件式を (上記の手法の 1 つを使用して) 組み込む場合には、結果の小数点以下の桁数の規則が適用されます。

「結果の小数点以下の桁数」精度規則の例

次の例は、「結果の小数点以下の桁数」精度の規則を示しています。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 * この例は、2 つの精度の規則を使用して中間値の精度を示しています。
D p1              s              13p 2
D p2              s              13p 2
D p3              s              13p 2
D p4              s              15p 9
D s1              s              13s 2
D s2              s              13s 2
D i1              s              10i 0
D f1              s               8f
D proc            pr              8p 3
D   parm1                        20p 5 value

 * 次の例では、各副次式に対して、2 つの
 * 精度が示されます。最初に、自然な精度、
 * 次に行末調整される精度です。
 * 例 1:

 /FREE
    eval    p1 = p1 * p2 * p3;
    // p1*p2        -> P(26,4); P(26,4)
    // p1*p2*p3     -> P(39,6); P(30,0) (小数点以下の桁数は切り捨てられます)

    eval(r) p1 = p1 * p2 * p3;
    // p1*p2        -> P(26,4); P(26,4)
    // p1*p2*p3     -> P(39,6); P(30,2) (小数点以下の桁数は、ターゲットの小数点
    //                 以下の桁数以下には切り捨てられません)

    eval(rh)p1 = p1 * p2 * p3;
    // p1*p2        -> P(26,4); P(26,5)
    // p1*p2*p3     -> P(39,6); P(30,3) (小数点以下の桁数は、ターゲットの小数点
    //                 以下の桁数 + 1 以下には切り捨てられません)

    // 例 2:

    eval    p4 = p1 * p2 * proc (s1*s2*p4);
    // p1*p2        -> P(26,4); P(26,4)
    // s1*s2        -> P(26,4); P(26,4)
    // s1*s2*p4     -> P(41,13); P(30,2) (小数点以下の桁数は切り捨てられます)
    // p1*p2*proc() -> P(34,7); P(30,3)  (小数点以下の桁数は切り捨てられます)

    eval(r) p4 = p1 * p2 * proc (s1*s2*p4);
    // p1*p2        -> P(26,4); P(26,4)
    // s1*s2        -> P(26,4); P(26,4)
    // s1*s2*p4     -> P(41,13); P(30,5)
    // p1*p2*proc() -> P(34,7); P(30,7)  (すでにターゲットの小数点以下の桁数以下
    //                   になっているので、すべての桁が維持されます)
 /END-FREE
```

図 123. 精度の規則の例

短絡評価

リレーショナル演算 AND および OR は左から右の方向に評価されます。ただし、その値が通用すると同時に、式の評価が停止して、値が戻されます。結果として、式のすべてのオペランドを評価する必要はありません。

演算 AND では、最初の演算が偽の場合には、2 番目のオペランドは評価されません。同様に、演算 OR では、最初の演算が真の場合には、2 番目のオペランドは評価されません。

この動作には、2 つの言外の意味があります。まず初めに、同じ式の中で配列指数をテストして使用することができます。たとえば以下の式があります。

```
I<=%ELEM(ARRAY) AND I>0 AND ARRAY(I)>10
```

これは、配列の指標付け例外となることはありません。

2 番目の言外の意味は、2 番目のオペランドがユーザー定義関数に対する呼び出しである場合には、その関数が呼び出されない、ことを示します。これは、その関数がパラメーターまたはグローバル変数の値を変更する場合には重要です。

評価の順序

式の中のオペランドの評価の順序は保証されません。したがって、式の中のどこかで変数が 2 回使用され、その副次作用の可能性がある場合には、予期しない結果になることがあります。

たとえば、図 124 に示されたソース (A は変数であり、FN は A を変更するプロシージャーです) を考えてみましょう。2 番目の EVAL 演算命令の式部分には A の 2 つのオカレンスがあります。***加算の演算の左側 (オペランド 1) が最初に評価される場合*** には、X には値 17, (5 + FN(5) = 5 + 12 = 17) が割り当てられます。 ***加算演算の右側 (オペランド 2) が最初に評価される場合*** には、Xには値 18, (6 + FN(5) = 6 + 12 = 18) が割り当てられます。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*   A は変数です。FN は A を変更するプロシージャーです。
 /free
     a = 5;
     x = a + fn(a);
 /end-free

P fn              B
D fn              PI             5P 0
D   parm     5P 0
 /free
     parm = parm + 1;
     return  2 * parm;
 /end-free
P fn              E
```

図 124. 副次作用のある呼び出しのサンプル・コーディング

第 25 章 組み込み関数

組み込み関数は、ユーザーが指定するデータで演算命令を実行するために、演算コードと類似したものです。組み込み関数は式の中で使用することができます。さらに、固定情報値をもつ組み込み関数を名前付き固定情報の中で使用することができます。これらの名前付き固定情報は、任意の仕様で使用することができます。

すべての組み込み関数には、その先頭文字としてパーセント記号 (%) があります。組み込み関数の構文は次の通りです:

```
function-name{(argument{:argument...})} です。
```

この関数の引き数は変数、固定情報、式、プロトタイプ・プロシージャー、またはその他の組み込み関数とすることができます。式の引き数には、組み込み関数を組み込むことができます。これを示す例は次の通りです。

```
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C*
C* この例は、複数のネストされた組み込み関数をもつ複雑な式
C* を示しています。
C*
C* %TRIM は、その引き数としてストリングを取ります。この例では、
C* 引き数は、ストリング A と %SUBST 組込み関数によって戻された
C* ストリングを連結したものです。%SUBST は、11 桁目から始まり、
C* %SIZE によって戻された長さから 20 を引いた長さとなる
C* ストリング B のサブストリングを戻します。%SIZE は、ストリング
C* B の長さを戻します。
C*
C* A がストリング '     Toronto,' で、B がストリング
C* '  Ontario, Canada            ' である場合には、%TRIM
C* の引き数は '     Toronto, Canada            ' となり、RES は
C* 'Toronto, Canada' という値になります。
C*
C                   EVAL      RES = %TRIM(A + %SUBST(B:11:%SIZE(B) - 20))
```

図 125. 組み込み関数の引き数の例

使用できる引き数の詳細については、個々の組み込み関数の説明を参照してください。

演算コードと違い、組み込み関数は結果フィールドに値を入れるのではなく、値を戻します。この相違を示す例は次の通りです。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の例では、CITY にはストリング 'Toronto, Ontario' が入ります。
C* SCAN 演算命令は、この例では 9 桁目の分離ブランクを見付けるために
C* 使用されます。SUBST は、ストリング 'Ontario' フィールド
C* TCNTRE に入れます。
C*
C* 次に、TCNTRE はリテラル 'Ontario' と比較されて、
C* 1 が CITYCNT に加算されます。
C*
C     ' '           SCAN      CITY          C
C                   ADD       1             C
C                   SUBST     CITY:C        TCNTRE
C    'Ontario'      IFEQ      TCNTRE
C                   ADD       1             CITYCNT
C                   ENDIF
C*
C* この例では、CITY には同じ値が入っていますが、
C* 変数 TCNTRE は、%SUBST 組み込み関数が適切な
C* 値を戻すので不要です。さらに、C に 1 を加算する
C* 中間ステップは、%SUBST が式を引き数として
C* 受け入れるので単純化されます。
C*
C     ' '           SCAN      CITY          C
C                   IF        %SUBST(CITY:C+1) = 'Ontario'
C                   EVAL      CITYCNT = CITYCNT+1
C                   ENDIF
```

図 126. 組み込み関数の引き数の例

この例で使用される引き数 (変数 CITY および式 C+1) は、SUBST 演算の係数値と類似しています。その関数自身の戻り値はその結果と類似したものです。一般に、組み込み関数の引き数は演算コードの演算項目 1と演算項目 2 のフィールドと類似したものです。

組み込み関数のもう 1 つの便利な機能は、定義仕様で使用する時にユーザー・コードの保守を単純化できることです。次の例は、この機能を例示したものです。

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D*
D* この例では、CUSTNAME は外部記述データ構造
D* CUSTOMER 中のフィールドです。
D* CUSTNAME の長さが変わると、TEMPNAME と NAMEARRAY の両方
D* の属性の変更は再コンパイルによってのみ行われます。
D* %SIZE 組み込み関数を使用すると、ユーザー・コード
D* に対する変更が不要となることを意味します。
D*
D CUSTOMER      E DS
D                 DS
D TEMPNAME                          LIKE(CUSTNAME)
D NAMEARRAY                   1     OVERLAY(TEMPNAME)
D                                   DIM(%SIZE(TEMPNAME))
```

図 127. 組み込み関数によって単純化された保守

組み込み関数は、拡張演算項目 2 の演算仕様の式で、また、定義仕様のキーワードとともに使用することができます。定義仕様のキーワードとともに使用する時には、組み込み関数の値はコンパイル時に通知する必要があり、引き数を式にすることはできません。

次の表は、組み込み関数、その引き数、および戻される値をリストしています。

名前	引き数	戻される値
%ABS	数値表現	式の絶対値
%ADDR	変数名	変数のアドレス
%ALLOC	割り振るバイト数	割り振られたストレージに対するポインター
%CHAR	グラフィック、UCS-2、数値、日付、時刻、または時刻スタンプ式 {: 日付、時刻、または時刻スタンプ形式}	文字形式の値
%CHECK	比較ストリング:検査するストリング{:開始桁}	比較ストリングの中にない文字の最初の桁、あるいは見つからなければゼロ
%CHECKR	比較ストリング:検査するストリング{:開始桁}	比較ストリングの中にない文字の最後の桁、あるいは見つからなければゼロ
%DATE	{値 {: 日付形式}}	指定された*値*に対応する日付、あるいは指定されていなければ現行システム日付
%DAYS	日数	期間としての日数
%DEC	数値表現{:数字:小数点以下の桁数}	パック数字形式の値
%DECH	数値表現:数字:小数点以下の桁数	パック数字形式の四捨五入された値
%DECPOS	数値表現	10 進数の数値
%DIFF	日付または時刻式: 日付または時刻式:単位	2 つの日付、時刻、または時刻スタンプの間の指定された単位での差
%DIV	被除数: 除数	2 つの引き数の除算からの商
%EDITC	非浮動数値表現:編集コード {:*CURSYM|*ASTFILL|通貨記号}	編集された値を表現するストリング
%EDITFLT	数値表現	浮動の文字外部表示表現
%EDITW	非浮動数値表現:編集語	編集された値を表現するストリング
%ELEM	配列、テーブル、または複数回繰り返しデータ構造名	エレメント数またはオカレンス数
%EOF	{ファイル名}	最新のファイル入力演算命令、またはファイルの終わり条件またはファイルの始め条件で終了したサブファイルへの書き込み (特定のファイルで、指定された場合) の場合には '1'
		そうでない場合は '0'
%EQUAL	{ファイル名}	最新の SETLL (特定のファイルで、指定された場合) または LOOKUP 演算命令で正確な一致が検出された場合には '1'
		そうでない場合は '0'
%ERROR		拡張 'E' が指定された最新の演算コードで結果がエラーとなった場合には '1'
		そうでない場合は '0'
%FLOAT	数値表現	浮動形式の値

名前	引き数	戻される値
%FOUND	{ファイル名}	最新の関係のある演算命令 (特定のファイルで、指定された場合) でレコード (CHAIN、DELETE、SETGT、SETLL)、エレメント (LOOKUP)、または一致 (CHECK、CHECKR、および SCAN) が検出された場合には '1'
		そうでない場合は '0'
%GETATR	ウィンドウ名、パーツ名、属性名、%PART、%WINDOW	属性の値
%GRAPH	文字、グラフィック、または UCS-2 式	グラフィック形式の値
%HOURS	時間数	期間としての時間数
%INT	数値表現	整数形式の値
%INTH	数値表現	整数形式の四捨五入された値
%LEN	いずれかの式	数字または文字の長さ
%LOOKUPxx	引き数: 配列{:開始指標 {:エレメントの数}}	突き合わせエレメントの配列指標
%MINUTES	分数	期間としての分数
%MONTHS	月数	期間としての月数
%MSECONDS	マイクロ秒数	期間としてのマイクロ秒数
%NULLIND	ヌル値可能なフィールド名	ヌル値可能なフィールドのヌル標識設定を表現する標識形式の値
%OCCUR	複数オカレンス・データ構造名	複数オカレンス・データ構造の現在のオカレンス
%OPEN	ファイル名	指定されたファイルがオープンの場合には '1'
		指定されたファイルがクローズの場合には '0'
%PADDR	プロシージャーまたはプロトタイプ名	プロシージャーまたはプロトタイプのアドレス
%REALLOC	ポインター: 数値表現	割り振られたストレージに対するポインター
%REM	被除数: 除数	2 つの引き数の除算からの剰余
%REPLACE	置き換えストリング: ソース・ストリング{:開始位置 {:置換するソースの長さ}}	置き換えストリングをソース・ストリングに挿入して作成されたストリングで、開始桁で開始されて、指定された文字数が置き換えられます。
%SCAN	検索引き数:検索するストリング{:開始位置}	ストリングの検索引き数の最初の位置、または検索されない場合にはゼロ
%SECONDS	秒数	期間としての秒数
%SETATR	ウィンドウ名、パーツ名、属性名、%PART、%WINDOW	なし
%SIZE	変数、配列、またはリテラル{:*ALL}	変数またはリテラルのサイズ
%SQRT	数値	数値の平方根
%STATUS	{ファイル名}	拡張 'E' の最新の演算コードが指定されて以降にプログラムまたはファイル・エラーが起こらなかった場合には 0
		エラーが起こった場合には、いずれかのプログラムまたはファイル状況の最新の選択値群
		ファイルを指定した場合には、戻される値はそのファイルの最新状況

名前	引き数	戻される値
%STR	ポインター{:最大長}	ポインター argumentup によってアドレス指定される文字で、最初の x'00' は含まれない
%SUBDT	日付または時刻式: 単位	指定された日付または時刻値の部分を含む符号なし数値
%SUBST	ストリング:開始{:長さ}	サブストリング
%THIS		ネイティブ・メソッドに対するクラス・インスタンス
%TIME	{値 {: 時刻形式}}	指定された*値*に対応する時刻、あるいは指定されていなければ現行システム時刻
%TIMESTAMP	{(値 {: 時刻スタンプ形式})}	指定された*値*に対応する時刻スタンプ、あるいは指定されていなければ現行システム時刻スタンプ
%TLOOKUPxx	引き数: 検索テーブル {: 代替テーブル}	一致すれば '*ON'
		そうでなければ '*OFF'
%TRIM	ストリング	左または右のブランクがトリムされたストリング
%TRIML	ストリング	左のブランクがトリムされたストリング
%TRIMR	ストリング	右のブランクがトリムされたストリング
%UCS2	文字またはグラフィック式	UCS-2 形式の値
%UNS	数値表現	符号なし形式の値
%UNSH	数値表現	符号なし形式の四捨五入された値
%XFOOT	配列式	エレメントの合計
%XLATE	開始文字: 終了文字: ストリング {: 開始桁}	開始文字が終了文字で置き換えられたストリング
%YEARS	年数	期間としての年数

組み込み関数の使用の詳細については、下記を参照してください。

- 255 ページの『第 18 章 定義仕様』
- 316 ページの『拡張演算項目 2 の構文』
- 383 ページの『第 24 章 式』
- 537 ページの『DOU (実行の終了)』
- 541 ページの『DOW (実行の時期)』
- 552 ページの『EVAL (式の評価)』
- 569 ページの『IF (判断)』
- 659 ページの『RETURN (呼び出し元への戻り)』
- 703 ページの『WHEN (真の場合に選択)』

組み込み関数 (アルファベット順)

次のセクションでは、組み込み関数について説明します。

%ABS (式の絶対値)

%ABS

%ABS はパラメーターとして指定された数値表現の絶対値を戻します。数値表現の値が非負数の場合には、値は未変更で戻されます。値が負数の場合には、戻される値はその式の値ですが、負数の記号が除去されます。

%ABS は、式で使用するか、あるいはキーワードに対するパラメーターとして使用することもできます。キーワードとともに使用する場合には、オペランドは数値リテラル、数値を表現する固定情報名、またはコンパイル時に通用する数値付きの組み込み関数でなければなりません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name +++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D f8         s              8f   inz (-1)
D i10        s             10i 0 inz (-123)
D p7         s              7p 3 inz (-1234.567)

 /FREE
    f8  = %abs (f8);        // "f8" は 1 になります。
    i10 = %abs (i10 - 321); // "i10" は 444 になります。
    p7  = %abs (p7);        // "p7" は 1234.567 になります。
 /END-FREE
```

図 128. %ABS の例

%ADDR (変数のアドレスの取り出し)

```
%ADDR(変数)
%ADDR(変数(索引))
%ADDR(変数(式))
```

%ADDR は基底ポインター・タイプの値を戻します。この値は指定された変数のアドレスです。これを比較して割り当てることができるのは、基底ポインター・タイプの項目に対してだけです。

配列指数パラメーターのある %ADDR を、定義仕様キーワード INZ または CONST のパラメーターとして指定する場合には、その配列指数はコンパイル時に通用していなければなりません。この指数は数値リテラルまたは数値固定情報のいずれかでなければなりません。

割り当ての結果が指数なしの配列である EVAL 演算命令では、代入演算子の右側の %ADDR は、 %ADDR の引き数によって異なった意味となります。%ADDR の引き数が指数なしの配列名であり、その結果が配列名である場合には、結果の配列の各エレメントには、その引き数配列の始めのアドレスが入っています。%ADDR の引き数が (*) の指数のある配列名である場合には、結果の配列の各エレメントには、その引き数配列の対応するエレメントのアドレスが入っています。これは、406 ページの図 129に示されています。

パラメーターとして指定した変数がテーブル、複数回繰り返しデータ構造、または複数回繰り返しデータ構造のサブフィールドである場合には、このアドレスは現行のテーブル索引またはオカレンス番号のアドレスです。

変数を基本にしている場合には、%ADDR はその変数の基底ポインターの値を戻します。変数が基本のデータ構造のサブフィールドである場合には、%ADDR の値は基底ポインターの値に、サブフィールドのオフセットをプラスしたものとなります。

この変数が *ENTRY PLIST の PARM として指定された場合には、%ADDR は呼び出し側によってプログラムに渡されるアドレスを戻します。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
 * 次の定義のセットは、配列指標にコンパイル時の値がある
 * ために有効です。
 *
D   ARRAY         S             20A   DIM (100)
 * 配列の 7 番目のエレメントのアドレスを指すポインターを設定します。
D   PTR           S               *   INZ (%ADDR(ARRAY(SEVEN)))
D   SEVEN         C                   CONST (7)
 *
D DS1             DS                  OCCURS (100)
D                                20A
D   SUBF                         10A
D                                30A
D CHAR10          S              10A   BASED (P)
D PARRAY          S                *   DIM(100)


 /FREE
    %OCCUR(DS1) = 23;
    SUBF = *ALL'abcd';
    P = %ADDR (SUBF);
    IF CHAR10 = SUBF;
       // この条件は真です。
     ENDIF;
    IF %ADDR (CHAR10) = %ADDR (SUBF);
       // この条件も真です。
     ENDIF;
    // 次のステートメントは SUBF の値も変更します。
    CHAR10 = *ALL'efgh';
    IF CHAR10 = SUBF;
       // この条件はまだ真です。
     ENDIF;
    //---------------------------------------------------------------
    %OCCUR(DS1) = 24;
    IF CHAR10 = SUBF;
      // この条件は真ではなくなりました。
     ENDIF;
    //---------------------------------------------------------------
    // 配列エレメントのアドレスは、配列指標として式を使用して
    // 取り出されます。
    P = %ADDR (ARRAY (X + 10));
    //---------------------------------------------------------------
    // 配列 PARRAY の各エレメントには、配列  ARRAY の最初のエレメントの
    // アドレスが入っています。
    PARRAY = %ADDR(ARRAY);
   // 配列 PARRAY の各エレメントには、配列 ARRAY の対応するエレメント
   // のアドレスが入っています。
    PARRAY = %ADDR(ARRAY(*));
 /END-FREE
```

図 129. %ADDR の例

%ALLOC (ストレージの割り振り)

```
%ALLOC(num)
```

%ALLOC は、指定された長さの、新規に割り振られたヒープ・ストレージを指すポインターを戻します。新規に割り振られたストレージはまだ初期化されていません。

パラメーターは、小数点以下の桁数がゼロの非浮動数値でなければなりません。指定する長さは 1 ～ 16776704 の間になければなりません。

詳細については、 370 ページの『メモリー管理命令』を参照してください。

演算命令を正常に完了できない場合には、例外 00425 または 00426 が出されます。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
  // 200 バイトのエリアを割り振ります
  pointer = %ALLOC(200);
/END-FREE
```

図 130. %ALLOC の例

%CHAR (文字データに変換)

```
%CHAR(式{:形式})
```

%CHAR は、式の値をグラフィック、UCS-2、数値、日付、時刻、または時刻スタンプから文字タイプに変換します。変換された値は未変更のままとなりますが、文字データと互換性のある形式で戻されます。

パラメーターが固定情報の場合には、変換はコンパイル時に行われます。

UCS-2 変換の結果が置換文字となる場合、そのパラメーターが固定情報の場合には、コンパイラー・リストに警告メッセージが示されます。そうでない場合には、状況 00050 が実行時に設定されますが、エラー・メッセージは提供されません。

グラフィック・データでは、戻される値はそれぞれのグラフィック・フィールドで 2 バイトです。たとえば、5 文字のグラフィック・フィールドが変換された場合には、戻される値は 10 文字 (10 バイトの・データ) です。式の値が可変長の場合には、戻される値は可変長形式となります。

日付、時刻、または時刻スタンプ・データの場合には、2 番目のパラメーターに、戻された文字データの変換先の日付、時刻、または時刻スタンプ形式が含まれます。戻される値には、指定された形式の後にゼロがなければ、区切り文字が組み込まれます。

数値データでは、式の値が浮動であると、結果は浮動形式 (たとえば、'+1.125000000000000E+020') になります。それ以外の場合は、値が負であれば結果は先行負符号付きで、先行ゼロのない 10 進数形式になります。小数点に使用される文字は、制御仕様の DECEDIT キーワードによって指定された文字 (デフォルトは '.') になります。たとえば、packed(7,3) 式の %CHAR は値 '-1.234' を戻します。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D Name            S             20G   VARYING INZ(G'XXYYZZ')
D date            S               D   INZ(D'1997/02/03')
D time            S               T   INZ(T'12:23:34')
D result          S            100A   VARYING
D points          S             10i 0 INZ(234)

 *---------------------------------------------------------------
 * 時刻および日付をデフォルト形式で形式設定するには、以下を使用します:
 *---------------------------------------------------------------
 /FREE
   result = 'It is ' + %CHAR(time) + ' on ' + %CHAR(date);
   // デフォルト形式がどちらも *USA である場合
   // result = '02/03/1997 の 12:23 PM'

   //---------------------------------------------------------------
   // 時刻および日付を特定の形式で形式設定するには、以下を使用します:
   //---------------------------------------------------------------
   result = 'It is ' + %CHAR(time : *hms:)
            + ' on ' + %CHAR(date : *iso);
   // result = '1997-02-03 の 12:23:34'
   //

   //---------------------------------------------------------------
   // 結果のパーツのみが必要な場合には、%char の結果とともに
   // %subst を使用できます
   //---------------------------------------------------------------
   result = 'The time is now ' + %SUBST (%CHAR(time):1:5) + '.';
   // result = '時刻は 12:23 になります。'

   //---------------------------------------------------------------
   // 文字値と連結できるようにグラフィック値を文字に変換するには、
   // %CHAR を使用します。
   //---------------------------------------------------------------
   result = 'The customer''s name is ' + %CHAR(Name) + '.';
   // result = 'customer の名前は XXYYZZ です。'

   //-----------------------------------------------------
   // 数値を文字形式に変換するには、%CHAR を使用します:
   //-----------------------------------------------------
   result = 'You have ' + %char(points) + ' points.';
   // result = '234 ポイントあります。'
   //
 /END-FREE
```

図 131. %CHAR の例

%CHECK (文字の検査)

```
%CHECK(比較 : 基本 {: 開始})
```

%CHECK は、*比較*ストリングに現れていない文字を含む*基本*ストリングの最初の位置を戻します。*基本*のすべての文字が*比較*にも現れている場合には、この関数は 0 を戻します。

検査は、開始桁で始まり、比較ストリングに含まれていない文字が見つかるまで右側へ続けられます。開始桁のデフォルトは 1 です。

最初のパラメーターは、文字、グラフィック、または UCS-2 の固定長または可変長のタイプでなければなりません。2 番目のパラメーターは最初のパラメーターと同じタイプでなければなりません。3 番目のパラメーター (指定する場合) は、小数点以下の桁数がゼロの非浮動数値でなければなりません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 *----------------------------------------------------
 * ストリングにはブランクおよび/またはコンマで区切られた
 * 一連の数値が含まれます。
 * %CHECK を使用して数値を取り出します
 *----------------------------------------------------
D string          s             50a   varying
D                                     inz('12, 233 17, 1, 234')
D delimiters      C                   ' ,'
D digits          C                   '0123456789'
D num             S             50a   varying
D pos             S             10i 0
D len             S             10i 0
D token           s             50a   varying

 /free

    // ストリングが区切り文字で終わっていることを確認します
    string = string + delimiters;

    dou string = '';

      // 数字のグループの先頭を見つけます
      pos = %check (delimiters : string);
      if (pos = 0);
         leave;
      endif;

      // 前の区切り文字をスキップします
      string = %subst(string : pos);

      // 数字のグループの長さを見つけます
      len = %check (digits : string) - 1;

      // 数字のグループを取り出します
      token = %subst(string : 1 : len);
      dsply ' ' ' ' token;

      // 前の数字をスキップします
      if (len < %len(string));
        string = %subst (string : len + 1);
      endif;

    enddo;

 /end-free
```

図 132. %CHECK の例

413 ページの図 134も参照してください。

%CHECKR (逆方向に検査)

```
%CHECKR(比較 : 基本 {: 開始})
```

%CHECKR は、*比較*ストリングに現れていない文字を含む*基本*ストリングの最後の位置を戻します。*基本*のすべての文字が*比較*にも現れている場合には、この関数は 0 を戻します。

検査は、開始桁で始まり、比較ストリングに含まれていない文字が見つかるまで左側へ続けられます。開始桁のデフォルトとしてストリングの終わりが使用されます。

最初のパラメーターは、文字、グラフィック、または UCS-2 の固定長または可変長のタイプでなければなりません。 2 番目のパラメーターは最初のパラメーターと同じタイプでなければなりません。3 番目のパラメーター (指定する場合) は、小数点以下の桁数がゼロの非浮動数値でなければなりません。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*-------------------------------------------
* ストリングの終わりにブランク以外の文字が
* 埋め込まれた場合には、%TRIM を使用して
* 文字を除去することはできません。
* %CHECKR は、"埋め込み文字" のリスト中に
* ないストリング内の最後の文字を検索する
* ことにより、このために使用できます。
*-------------------------------------------
D string1         s             50a   varying
D                                     inz('My *dog* Spot.* @ * @ *')
D string2         s             50a   varying
D                                     inz('someone@somewhere.com')
D padChars        C                   ' *@'

 /free

    %len(string1) = %checkr(padChars:string1);
    //  %len(string1) は 14 ("padChars" の中にない最後の文字の
    //  位置) に設定されます。
    //  string1 = 'My *dog* Spot.'

    %len(string2) = %checkr(padChars:string2);
    //  %len(string2) は 21 ("padChars" の中にない最後の文字の
    //  位置) に設定されます。
    //  string2 = 'someone@somewhere.com'  (ストリングは変更されません)

 /end-free
```

図 133. %CHECKR の例

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 *------------------------------------------------------
 * ストリングには数値が含まれるが、その数値は
 * ブランクおよびアスタリスクで囲まれていたり、
 * 前に通貨記号がある可能性があります。
 *------------------------------------------------------
D string          s             50a   varying inz('$****12.345***  ')

 /free
    // ' $*' の 1 つでない最初の文字の位置を見つけます
    numStart = %CHECK (' $*' : string);
    //  = 6

    // ' *' の 1 つでない最後の文字の位置を見つけます
    numEnd = %CHECKR (' *' : string);
    //  = 11

    // 数値ストリングを取り出します
    string = %SUBST(string : numStart : numEnd - numStart + 1);
    //  = '12.345'

 /end-free
```

図 134. %CHECK および %CHECKR の例

%DATE (日付に変換)

```
%DATE{(式{:日付形式})}
```

%DATE は、式の値を文字、数値、または時刻スタンプ・データから日付タイプに変換します。変換された値は未変更のまま残されますが、日付として戻されます。

最初のパラメーターは変換する値です。値を指定しない場合には、%DATE は現行システム日付を戻します。

2 番目のパラメーターは、文字または数値入力のための日付形式です。入力形式とは無関係に、出力は *ISO 形式で戻されます。

使用できる入力形式については、 125 ページの『日付データ』を参照してください。文字または数値入力のための日付形式が指定されない場合に、デフォルト値は DATFMT 制御仕様キーワードに指定された形式か *ISO のどちらかになります。詳細については、 227 ページの『DATFMT(fmt{区切り文字})』を参照してください。

最初のパラメーターが時刻スタンプ、*DATE、または UDATE である場合には、2 番目のパラメーターは指定しないようにしてください。これらの場合には、システムが入力の形式を認識しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE

  string = '040596';
  date =  %date(string:*MDY0);
  // 日付には現在 d'1996-04-05' が含まれています
/END-FREE
```

図 135. %DATE の例

%DAYS (日数)

```
%DAYS(数値)
```

%DAYS は、数値を日付または時刻スタンプ値に追加できる期間に変換します。

加算または減算演算命令では、%DAYS は右側の値にのみすることができます。左側の値は日付または時刻スタンプでなければなりません。結果は、該当する日数が加算または減算された日付または時刻スタンプ値となります。日付の場合の結果の値は *ISO 形式です。

日付および時刻算術演算の例については、 444 ページの図 158を参照してください。

%DEC (パック 10 進数形式に変換)

```
%DEC(数値表現{:精度:小数点以下の桁数})
```

%DEC は数値表現の値を、精度および小数点のある10 進数 (パック) 形式に変換します。精度および小数点は、数値リテラルまたはコンパイル時に通用する数字の組み込み関数を表現する数値リテラルの名前付き固定情報でなければなりません。

パラメーターの精度および小数点は、数値表現のタイプが浮動でない場合には省略することができます。これらのパラメーターが省略された場合には、精度および小数点は数値表現の属性から取り出されます。

417 ページの図 136は、%DEC 組み込み関数の例を示しています。

%DECH (四捨五入のあるパック 10 進数形式に変換)

%DECH(数値表現:精度:小数点以下の桁数)

%DECH は %DEC と同じですが、数値表現が 10 進数または浮動値の場合に、必要な精度に変換する時に数値表現の値に四捨五入が適用される点が異なります。四捨五入が実行できない場合には、メッセージは出されません。

%DEC とは異なり、3 つのすべてのパラメーターが必須です。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D p7              s               7p 3 inz (1234.567)
D s9              s               9s 5 inz (73.73442)
D f8              s               8f   inz (123.456789)
D result1         s              15p 5
D result2         s              15p 5
D result3         s              15p 5

 /FREE
    result1 = %dec (p7) + 0.011;  // "result1" は 1234.57800 になります
    result2 = %dec (s9 : 5: 0);   // "result2" は 73.00000 になります
    result3 = %dech (f8: 5: 2);   // "result3" は 123.46000 になります
 /END-FREE
```

図 136. %DEC および %DECH の例

%DECPOS (小数点以下の桁数の取り出し)

%DECPOS(数値表現)

%DECPOS は数値変数または式の小数点以下の桁数を戻します。戻される値は固定情報ですので、固定情報結合に関与することができます。

数値表現は浮動変数または式としてはいけません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D p7              s               7p 3 inz (8236.567)
D s9              s               9s 5 inz (23.73442)
D result1         s               5i 0
D result2         s               5i 0
D result3         s               5i 0

 /FREE
   result1 = %decpos (p7);     //  "result1" は 3 になります。
   result2 = %decpos (s9);     //  "result2" は 5 になります。
   result3 = %decpos (p7 * s9);//  "result3" は 8 になります。
 /END-FREE
```

図 137. %DECPOS の例

%LEN を指定した %DECPOS の例については、 440 ページの図 156を参照してください。

%DIFF (2 つの日付、時刻、または時刻スタンプ値の間の差)

```
%DIFF(op1:op2:*MSECONDS|*SECONDS|*MINUTES|*HOURS|*DAYS|*MONTHS|*YEARS)
%DIFF(op1:op2:*MS|*S|*MN|*H|*D|*M|*Y)
```

%DIFF は、2 つの日付または時刻値の間の差 (期間) を生成します。最初と 2 番目のパラメーターは、同じであるかまたは互換性のあるタイプでなければなりません。 次の組み合わせを使用することができます:

- 日付と時刻
- 時刻と時刻
- 時刻スタンプと時刻スタンプ
- 日付と時刻スタンプ (時刻スタンプの日付部分のみが考慮されます)
- 時刻と時刻スタンプ (時刻スタンプの時刻部分のみが考慮されます)

3 番目のパラメーターは単位を指定します。次の単位が有効です:

- 2 つの日付または日付と時刻スタンプの場合: *DAYS、*MONTHS、および *YEARS
- 2 つの時刻または時刻と時刻スタンプの場合: *SECONDS、*MINUTES、および *HOURS
- 2 つの時刻スタンプの場合: *MSECONDS、*SECONDS、*MINUTES、*HOURS、*DAYS、*MONTHS、および *YEARS

結果の端数は切り捨てられます。たとえば、61 分は 1 時間と等しく、59 分は 0 時間と等しくなります。

この関数によって戻される値には、数値タイプと期間タイプの両方との互換性があります。結果を数値 (数値タイプ) あるいは日付、時刻、または時刻スタンプ (期間タイプ) に追加することができます。

32 年 9 か月以上離れた 2 つのタイム・スタンプの間のマイクロ秒による差を求めた場合には、15 桁の期間値の限界を超えることになります。この結果はエラーまたは切り捨てとなります。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE

  // 2 つの日付の間の日数を判別します
  num_days = %DIFF (loandate: duedate: *DAYS);

  // 開始と終了の間の分数を加算します
  time = time + %DIFF (start: end: *minutes);
/END-FREE
```

図 138. %DIFF の例

%DIV (商の整数部分を戻す)

```
%DIV(n:m)
```

%DIV は、オペランド **n** を **m** で除算した結果の商の整数部分を戻します。2 つのオペランドは小数点以下の桁数がゼロの数値でなければなりません。オペランドのいずれかがパック、ゾーン、または 2 進数の数字の場合には、その結果はパック数値となります。オペランドのいずれかが整数の数字の場合には、その結果は整数となります。そうでない場合には、その結果は符号なしの数字となります。浮動数字オペランドは使用できません。(また、 451 ページの『%REM (整数剰余を戻す)』も参照してください。)

オペランドが 8 バイト整数または符号なしのフィールドに適合できる固定情報の場合には、固定情報結合がその組み込み関数に適用されます。この場合には、%DIV 組み込み関数は定義仕様にコーディングすることができます。

この関数は、 451 ページの図 164 に示されています。

%EDITC (編集コードを使用しての値の編集)

```
%EDITC(数値 : 編集コード {: *ASTFILL | *CURSYM | 通貨記号})
```

この関数は、編集コードに従って編集された数値を表現した文字の結果を戻します。一般に、この数値および編集コードの規則は、出力仕様で数値を編集する場合の規則と同じです。3 番目のパラメーターは任意指定であり、指定する場合には、次の 1 つでなければなりません。

***ASTFILL**
アスタリスク保護が使用されることを示します。これは、戻り値で先行ゼロがアスタリスクで置き換えられることを意味します。たとえば、%EDITC(-0012.5 : 'K' : *ASTFILL) は '**12.5–' を戻します。

***CURSYM**
浮動通貨記号が使用されることを示します。実際の記号は CURSYM キーワードの制御仕様に指定されたものとなり、デフォルトは '\' です。*CURSYM が指定された場合には、通貨記号は最初の有効数字の直前の結果に入れられます。たとえば、%EDITC(0012.5 : 'K' : *CURSYM) は ' \12.5 ' の結果を戻します。

通貨記号
浮動通貨記号が提供された通貨記号とともに使用されることを示します。これは 1 バイトの文字固定情報 (リテラル、名前付き固定情報、またはコンパイル時に評価できる式) でなければなりません。たとえば、%EDITC(0012.5 : 'K' : 'X') は ' X12.5 'を戻します。

%EDITC の結果は常に同じ長さとなり、先行ブランクまたは末尾ブランクが含まれることがあります。たとえば、%EDITC(NUM : 'A' : '\') は NUM のある値で '\1,234.56CR' を戻し、別の値では ' \4.56 ' を戻します。

浮動式は最初のパラメーターには使用できません (浮動を編集可能な形式に変換するためには %DEC を使用できます)。編集コードは文字固定情報として指定されます。サポートされる編集コードは 'A' – 'D'、'J' – 'Q'、'X' – 'Z'、'1' – '9' です。この固定情報はリテラル、名前付き固定情報、またはコンパイル時に判別できる値の式とすることができます。

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D msg             S            100A
D salary          S              9P 2 INZ(1000)
 * salary の値が 1000 の場合には、salary * 12 の値は 12000.00 です。
 * 浮動通貨記号をもつ A 編集コードを使用した場合の salary * 12 の
 * 編集済みバージョンは ' \12,000.00 ' です。
 * msg の値は 'The annual salary is \12,000.00' となります。
CL0N01Factor1+++++++Opcode&ExtExtended-factor2++++++++++++++++++++++++++++
C                   EVAL      msg = 'The annual salary is '
C                               + %trim(%editc(salary * 12
C                                             :'A': *CURSYM))
 * 次の例では、msg の値は 'The annual salary is &12,000.00' となります。
C                   EVAL      msg = 'The annual salary is '
C                               + %trim(%editc(salary * 12
C                                             :'A': '&'))

 * 次の例では、msg の値は 'Salary is $*****12,000.00' となります。
 * この '\' は編集コードからではなく、テキストからのものであることに注意してください。
C                   EVAL      msg = 'Salary is \'
C                               + %trim(%editc(salary * 12
C                                              :'B': *ASTFILL))

 * 次の例では、msg の値は 'The date is 1/14/1999' となります。
C                   EVAL      msg = 'The date is '
C                               + %trim(%editc(*date : 'Y'))
```

図 139. %EDITC の例 1

共通の要件はフィールドを編集することであり、次の通りです。

- 先行ゼロは抑制される
- 値が負数の場合には、その値を括弧で囲む

これは、サブプロシージャーの %EDITC を使用して次の通り行われます。

```
D neg             S              5P 2      inz(-12.3)
D pos             S              5P 2      inz(54.32)
D editparens      PR            50A
D    val                        30P 2      value
D editedVal       S             10A

C                   EVAL      editedVal = editparens(neg)

C* 現在 editedVal は値 '(12.30)   ' をもちます

C                   EVAL      editedVal = editparens(pos)

C* 現在 editedVal は値 ' 54.32    ' をもちます

 *---------------------------------------------------------------
 * サブプロシージャー EDITPARENS
 *---------------------------------------------------------------

P editparens      B
D editparens      PI            50A
D    val                        30P 2      value
D lparen          S              1A        inz(' ')
D rparen          S              1A        inz(' ')
D res             S             50A

C* 値が負数の場合には、括弧を使用します

C                   IF        val < 0
C                   EVAL      lparen = '('
C                   EVAL      rparen = ')'
C                   ENDIF

C* 編集値を戻します
C* '1' 編集コードには符号が含まれていないので、絶対値を
C* 計算する必要はありません。
C                   RETURN    lparen             +
C                             %editc(val : '1')  +
C                             rparen

P editparens      E
```

図 140. %EDITC の例 2

%EDITFLT (浮動外部表現に変換)

%EDITFLT(数値表現)

%EDITFLT は、数値表現の値を浮動の文字外部表示表現に変換します。その結果は 14 文字または 23文字のいずれとなります。引き数が 4 バイトの浮動フィールドの場合には、その結果は 14 文字です。そうでない場合には、これは 23 文字です。

定義仕様キーワードに対するパラメーターとして指定された場合には、そのパラメーターは数値リテラル、浮動リテラル、または数値固定情報名、または組み込み関数でなければなりません。式の中に指定する場合、数値表現に固定情報値がある場合には、固定情報結合が適用されます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D f8              s              8f   inz (50000)
D string          s             40a   varying

 /FREE
    string = 'Float value is ' + %editflt (f8 - 4E4) + '.';
    // "string" の値は '浮動値は +1.000000000000000E+004 です。'
 /END-FREE
```

図 141. %EDITFLT の例

%EDITW (編集語を使用しての値の編集)

%EDITW(数値 : 編集語)

この関数は、編集語に従って編集された数値を表現した文字の結果を戻します。この数値および編集語の規則は、出力仕様で数値を編集する場合の規則と同じです。

最初のパラメーターには、浮動式を使用することはできません。浮動を編集可能な形式に変換するためには %DEC を使用してください。

編集語は文字固定情報でなければなりません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D amount          S             30A
D salary          S              9P 2
D editwd          C                    '$ ,   ,  **Dollars&  &Cents'

 * 給与の値が 2451.53 の場合には、(給与 * 12) の編集済み
 * バージョンは '$***29,418*ドル 36 セント' となります。金額
 * の値は '年間給与は $***29,418*ドル 36 セント' です。
 /FREE
   amount = 'The annual salary is '
                + %editw(salary * 12 : editwd);
 /END-FREE
```

図 142. %EDITW の例

%ELEM (エレメント数の取り出し)

```
%ELEM(テーブル名)
%ELEM(配列名)
%ELEM(複数回繰り返しデータ構造名)
```

%ELEM は指定された配列、テーブル、または複数回繰り返しデータ構造中のエレメント数を戻します。これは、定義仕様で数値固定情報が許可される任意の位置に、あるいは拡張演算項目 2 フィールドの式に指定することができます。

パラメーターは配列、テーブル、または複数回繰り返しデータ構造の名前でなければなりません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D arr1d           S             20    DIM(10)
D table           S             10    DIM(20) ctdata
D mds             DS            20    occurs(30)
D num             S              5p 0

 * like_array は 10 のディメンションにより定義されます。
 * array_dims は 10 の値により定義されます。
D like_array      S                   like(arr1d) dim(%elem(arr1d))
D array_dims      C                   const (%elem (arr1d))

 /FREE
    num = %elem (arr1d);  // num は 10 になります
    num = %elem (table);  // num は 20 になります
    num = %elem (mds);    // num は 30 になります
 /END-FREE
```

図 143. %ELEM の例

%EOF (ファイル条件の終了または開始を戻す)

```
%EOF{(ファイル名)}
```

最新の読み取り演算またはサブファイルへの書き込みがファイルの終わりまたはファイル条件の始めで終了した場合には、%EOF は '1' を戻します。そうでない場合には、'0' を戻します。

%EOF を設定する演算命令は次の通りです。
- 640 ページの『READ (レコードの読み取り)』
- 643 ページの『READC (次の変更レコードの読み取り)』
- 645 ページの『READE (等しいキーのレコードの読み取り)』
- 648 ページの『READP (前のレコードの読み取り)』
- 650 ページの『READPE (前の等しいキーのレコードの読み取り)』
- 707 ページの『WRITE (新しいレコードの作成)』 (サブファイルのみ)

次の演算命令が正常に行われると、%EOF(ファイル名) はオフに設定されます。演算命令が正常に行われなければ、 %EOF(ファイル名) は変更されません。パラメーターなしの %EOF は、これらの演算命令によって変更されません。
- 510 ページの『CHAIN (ファイルからのランダム検索)』
- 627 ページの『OPEN (処理用ファイルのオープン)』
- 668 ページの『SETGT (より大きいレコードへのセット)』
- 671 ページの『SETLL (下限のセット)』

全手順ファイルを指定すると、指定されたファイルでは、上記のリストの直前の演算命令がファイルの終わりまたはファイル条件の始めの結果となった場合には、この関数は '1' を戻します。1 次または 2 次ファイルでは、%EOF が使用可能となるのは、ファイル名を指定した場合だけです。*GETIN処理中の最新の入力演算がファイルの終わりまたはファイル条件の始めで終了した場合には、これは '1' に設定されます。そうでない場合には、これは '0' を戻します。

この関数は入力、更新、レコード・アドレス・ファイルで使用することができます。表示装置ファイルでは、サブファイル・レコードに対する WRITE が許可されます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * ファイル INFILE のレコード様式は INREC です
FINFILE    IF   E            DISK        remote

 /FREE
    READ INREC;  // レコードの読み取り
    IF  %EOF;
                 // ファイルの終わりの処理
     ENDIF;
 /END-FREE
```

図 144. ファイル名パラメーターを指定しない %EOF

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 * このプログラムは 2 つのファイルを比較します

F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
FFILE1     IF   E             DISK       remote
FFILE2     IF   E             DISK       remote

 * FILE1 か FILE2 がファイルの終わりに達するまでループします
 /FREE
    DOU %EOF(FILE1) OR %EOF(FILE2);
       // 各ファイルからレコードを読み取り、レコードを比較します

       READ REC1;
       READ REC2;
       IF %EOF(FILE1) AND %EOF(FILE2);
          // 両方のファイルがファイルの終わりに達しています
          EXSR EndCompare;

       ELSEIF %EOF(FILE1);
          // FILE1 は FILE2 より短くなっています
          EXSR F1Short;

       ELSEIF %EOF(FILE2);
          // FILE2 は FILE1 より短くなっています
          EXSR F2Short;

    ELSE;
          // 両方のファイルにはまだ比較するレコードがあります
          EXSR CompareRecs;
     ENDIF;
    ENDDO;
  // ...
 /END-FREE
```

図 145. ファイル名パラメーターを指定した %EOF

%EQUAL (正確に一致した条件を戻す)

%EQUAL{(ファイル名)}

最新の関係のある演算命令が正確に一致したものを検出した場合には、%EQUAL は '1' を戻します。そうでない場合には、これは '0' を戻します。

%EQUAL を設定する演算命令は次の通りです。
- 671 ページの『SETLL (下限のセット)』
- 582 ページの『LOOKUP (テーブルまたは配列エレメントの検索)』

オプショナルのファイル名パラメーターを指定せずに %EQUAL を使用した場合には、これは、最新の関係のある演算命令の値セットを戻します。

SETLL 演算命令では、レコードがあり、そのキーまたは相対レコード番号が検索引き数と等しい場合には、この関数は '1' を戻します。

EQ 標識が指定された LOOKUP 演算命令では、検索引き数と正確に一致するエレメントが検出された場合には、この関数は '1' を戻します。

ファイル名を指定した場合には、この関数は、指定されたファイルの最新の SETLL 演算命令に適用されます。この関数が使用可能となるのは、SETLL 命令コードが使用可能なファイルの場合だけです。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * ファイル CUSTS のレコード様式は CUSTREC です
FCUSTSIF   E           K DISK       remote

 /FREE
     // ファイルに Cust と一致するキーをもつレコードが含まれているかどうかを検査します
    setll Cust CustRec;
    if %equal;
    // ファイル内で完全な一致が見つかりました
      endif;
 /END-FREE
```

図 146. SETLL を指定した %EQUAL の例

```
     DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
     D TabNames        S             10A   DIM(5) CTDATA ASCEND
     D SearchName      S             10A
      * テーブルを SearchName かまたはその近くに位置決めします
      * ここには、SearchName の異なる値に対するこのプログラムの
      * 結果があります:
      *   SearchName   |   DSPLY
      *   -------------+------------------------------
      *   'Catherine ' |   '次に大きい     Martha'
      *   'Andrea    ' |   '完全に一致     Andrea'
      *   'Thomas    ' |   '見つからない   Thomas'
     C..N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
     C     SearchName    LOOKUP    TabNames                          10  10
     C                   SELECT
     C                   WHEN      %EQUAL
      * 完全な一致は見つかりません
     C                   WHEN      %FOUND
      * SearchName より大きい名前が見つかりました
     C                   OTHER
      * 見つかりません。SearchName はテーブル内にすべての名前より大きくなっています
     C                   ENDSL
     C                   RETURN
**CTDATA TabNames
Alexander
Andrea
Bohdan
Martha
Samuel
```

図 147. LOOKUP を指定した %EQUAL および %FOUND の例

%ERROR (エラー条件を戻す)

拡張 'E' が指定された最新の演算で結果がエラーとなった場合には、%ERROR は '1' を戻します。これは、演算命令にエラー標識が設定されたのと同じです。拡張 'E' が指定された演算が開始する前に、%ERRORは '0' を戻すように設定されて、エラーが起こらない場合には、その演算命令の後で未変更のままとなります。また、エラー標識を許可するすべての演算命令では %ERROR 組み込み関数を設定することもできます。

%ERROR 組み込み関数の例については、460 ページの図 170 および 461 ページの図 171を参照してください。

%FLOAT (浮動形式に変換する)

```
%FLOAT(数値表現)
```

%FLOAT は、数値表現の値を浮動形式に変換します。この組み込み関数を使用できるのは、式の中だけです。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D p1              s              15p 0 inz (1)
D p2              s              25p13 inz (3)
D result1         s              15p 5
D result2         s              15p 5
D result3         s              15p 5

 /FREE
    result1 = p1 / p2;          //  "result1" は 0.33000 になります。
    result2 = %float (p1) / p2; //  "result2" は 0.33333 になります。
    result3 = %float (p1 / p2); //  "result3" は 0.33333 になります。
 /END-FREE
```

図 148. %FLOAT の例

%FOUND (検出された条件を戻す)

%FOUND{(ファイル名)}

最新の関係のあるファイル演算命令がレコードを検出したか、ストリング演算命令が一致を検出したか、あるいは検索演算命令がエレメントを検出した場合には、%FOUND は '1' を戻します。そうでない場合には、この関数は '0' を戻します。

オプショナルのファイル名パラメーターを指定せずに %FOUND を使用した場合には、これは、最新の関係のある演算命令の値セットを戻します。ファイル名が指定された場合には、これは、そのファイルの最新の関係のある演算命令に適用されます。

ファイル演算命令では、%FOUND は ″no record found NR″ 標識に対する関数と逆になります。

ストリング演算命令では、%FOUND は関数で ″found FD″ 標識と同じになります。

LOOKUP 演算命令では、その演算命令が検索条件を満たすエレメントを検出した場合には、%FOUND は '1' を戻します。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * ファイル CUSTS のレコード様式は CUSTREC です
FCUSTS     IF   E           K DISK       remote

 /FREE
    // カスタマーがファイル中にあるかどうかを検査します
    chain Cust CustRec;
    if %found;
       exsr HandleCustomer;
      endif;
 /END-FREE
```

図 149. パラメーターを指定せずにファイル演算命令のテストに使用する %FOUND

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * ファイル MASTER にはすべてのカスタマーが含まれます
 * ファイル GOLD には "特権" カスタマーのみが含まれます
FMASTER    IF   E           K DISK         remote
FGOLD      IF   E           K DISK         remote

 /FREE
   // カスタマーが存在するかどうかを検査しますが、特権カスタマーは検査しません
   chain Cust MastRec;
   chain Cust GoldRec;

   // %FOUND に使用されるのはレコード名ではなく、ファイル名であることに注意してください
   if  %found (Master)  and not  %found (Gold);
   //
      endif;
 /END-FREE
```

図 150. パラメーターを指定してファイル演算命令のテストに使用する %FOUND

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D Numbers         C                   '0123456789'
D Position        S              5I 0
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 * 名前の実際の位置が必要でない場合には、SCAN 演算命令の結果の
 * テストには %FOUND だけを使用してください。
 * Name が値 'Barbara' で、Line が値 'in the city of Toronto.      '
 * の場合には、%FOUND は '0' を戻します。
 * Line が値 'the city of Toronto where Barbara lives ' の
 * 場合には、%FOUND は '1' を戻します。
C     Name          SCAN      Line
C                   IF        %FOUND
C                   EXSR      PutLine
C                   ENDIF
 * Value に値「12345.67」が入っている場合には、Position は 6 に
 * 設定されて、%FOUND は値「1」を戻します。
 * Value に値「10203040」が入っている場合には、Position は 0 に
 * 設定されて、%FOUND は値「0」を戻します。
C     Numbers       CHECK     Value         Position
C                   IF        %FOUND
C                   EXSR      HandleNonNum
C                   ENDIF
```

図 151. ストリング演算命令のテストに使用される %FOUND

%GETATR (属性の検索)

%GETATR(ウィンドウ名:パーツ名:属性名)

%GETATR はウィンドウ上のあるパーツの属性値を戻します。最初のパラメーターと 2 番目のパラメーターの両方を %WINDOW または %PART とすることができます。

パーツ属性にアクセスするための別の形式については、382 ページの『修飾 GUI パーツ属性アクセス』を参照してください。

注:

1. %GETATR 組み込み関数は複数のパーツの対応するプログラム・フィールドには影響しません。そのパーツの対応するプログラム・フィールドに、入力フィールド現在の値を含めたい場合には、それを %GETATR 組み込みのターゲットなどにしてください。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....Comments++++++
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++Comments++++++
C                   EVAL      ENT0000B = %GETATR('INVENTORY':'ENT0000B':'TEXT')
```

図 152. %GETATR の例

2. %GETATR 組み込み関数は 1 バイトおよび 8 バイトの符号付きおよび符号なし整数値をサポートしていません。

%GRAPH (グラフィック値に変換)

```
%GRAPH(文字式 | グラフィック式 | UCS-2 式 { : CCSID })
```

%GRAPH は文字、グラフィック、または UCS-2 から式の値を変換して、グラフィック値を戻します。パラメーターが可変長の場合には、その結果は可変長となります。

2 番目のパラメーターの *ccsid* は任意指定であり、結果の式の CCSIDを指示します。CCSID のデフォルトとして、ワークステーション CCSID と関連した CCSID を使用します。 CCSID(*GRAPH : *IGNORE) が制御仕様に指定されるか、あるいはモジュールで想定された場合には、%GRAPH組み込みは使用できません。

パラメーターが固定情報の場合には、変換はコンパイル時に行われます。この場合、CCSID はソース・ファイルの CCSID と関連した CCSID です。

変換の結果が置換文字となった場合には、コンパイル時に警告メッセージが出されます。実行時に、状況 00050 が設定されて、エラー・メッセージは出されません。

注: 2 つのユニコード CCSID 相互間の変換はサポートされません。サポートされる CCSID 値のリストについては、 725 ページの『付録 C. サポートされる CCSID 値』 を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
H*Keywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H ccsid (*graph: 942)

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D  char           S               5A   inz('abcde')
 * %GRAPH 組み込み関数はグラフィック・フィールドの初期化に使用されます
D  graph          S              10G   inz (%graph ('AABBCCDDEE'))
D  ufield         S               2C   inz (%ucs2 ('FFGG'))
D  graph2         S               2G   ccsid (951) inz (*hival)
D  isEqual        S               1N
D  proc           PR
D    gparm                        2G   ccsid (951) value

 /FREE
    graph = %graph (char) + %graph (ufield);
    // graph は現在 7 つのグラフィック文字 AABBCCDDEEFFGG です。

    isEqual = graph = %graph (graph2 : 942);
    // %GRAPH 組み込み関数の結果は graph2 の値となり、
    // CCSID  951 から CCSID 942 に変換されます。

    graph2 = graph;
    // graph の値は CCSID 942 から CCSID 951 に変換されて、
    // graph2 に保管されます。
    // この変換はコンパイラーによって暗黙的に実行されます。

    proc (graph);
   // graph の値は、そのパラメーターを値ごとに渡す一部として
   // CCSID 942 から CCSID 951 に暗黙的に変換されます。 /END-FREE
```

図 153. %GRAPH の例

%HOURS (時間数)

```
%HOURS(数値)
```

%HOURS は、数値を時刻または時刻スタンプ値に追加できる期間に変換します。

加算または減算演算命令では、%HOURS は右側の値にのみすることができます。左側の値は時刻または時刻スタンプでなければなりません。結果は、該当する時間数が加算または減算された時刻または時刻スタンプ値となります。時刻の場合の結果の値は *ISO 形式です。

日付および時刻算術演算の例については、 444 ページの図 158を参照してください。

%INT (整数形式に変換)

%INT(数値表現)

%INT は、数値表現の値を整数に変換します。10 進数字のどれかの切り捨てが行われます。この組み込み関数を使用できるのは、式の中だけです。%INT を使用して浮動または 10 進数の小数点以下の桁数を切り捨てて、それを配列指数として使用することができます。

図 154は、%INT 組み込み関数の例を示しています。

%INTH (四捨五入付き整数形式に変換)

%INTH(数値表現)

%INTH は %INT と同じですが、数値表現が 10 進数または浮動値の場合に、整数タイプに変換する時に数値表現の値に四捨五入が適用される点が異なります。四捨五入が実行できない場合には、メッセージは出されません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D p7              s              7p 3 inz (1234.567)
D s9              s              9s 5 inz (73.73442)
D f8              s              8f   inz (123.789)
D result1         s             15p 5
D result2         s             15p 5
D result3         s             15p 5
D array           s              1a   dim (200)
D a               s              1a

 /FREE
    result1 = %int (p7) + 0.011;  // "result1" は 1234.01100 になります。
    result2 = %int (s9);          // "result2" は   73.00000 になります。
    result3 = %inth (f8);         // "result3" は  124.00000 になります。

    // %INT および %INTH は配列指標として使用することができます
    a = array (%inth (f8));
 /END-FREE
```

図 154. %INT および %INTH の例

%LEN (長さの取り出しまたは設定)

```
%LEN(式)
```

%LEN を使用して、変数式の長さを取り出すか、あるいは可変長フィールドの現行の長さを設定することができます。

このパラメーターを表意定数とすることはできません。

その値で使用される %LEN

式の右側で使用する時には、この関数はその変数式の桁数または文字数を戻します。

数値表現では、戻される値はその式の精度を表し、必ずしも有効数字の実際の桁数ではありません。浮動の変数または式では、戻される値は 4 または 8 です。このパラメーターが数値リテラルの時には、戻される長さはそのリテラルの桁数です。

文字、グラフィック、または UCS-2 の式では、戻される値はその式の値の文字数です。組み込み関数または可変長フィールドから戻された値などの可変長の値では、%LEN によって戻される値は、その文字、グラフィック、または UCS-2 値の現行の長さです。

そのパラメーターがコンパイル時に計算できる値をもつ組み込み関数または式である場合には、戻される値は、その式によって戻すことができた最大可能値ではなくて、その固定情報値の実際の桁数です。

この他のすべてのデータ・タイプでは、戻される値はその値のバイト数です。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D num1            S              7P 2
D num2            S              5S 1
D num3            S              5I 0 inz(2)
D chr1            S             10A   inz('Toronto   ')
D chr2            S             10A   inz('Munich    ')
D ptr             S               *

 * 数値表現:
 /FREE
   num1 = %len(num1);                 //  7
   num1 = %decpos(num2);              //  1
   num1 = %len(num1*num2);            // 12
   num1 = %decpos(num1*num2);         //  3

   // 文字表現:
   num1 = %len(chr1);                 // 10
   num1 = %len(chr1+chr2);            // 20
   num1 = %len(%trim(chr1));          //  7
   num1 = %len(%subst(chr1:1:num3) + ' ' + %trim(chr2));//  9

   // %len および %decpos は他の組み込み関数とともに有効に使用できます:

   // この除算は浮動で行われますが、その結果は eval の結果と
   // 同じ精度に変換されます:
   num1 = 27 + %dec (%float(num1)/num3 : %len(num1) : %decpos(num1));

   // 連結の結果 (に末尾ヌル文字の余分なバイトを加えたもの) を
   // 保留するための十分なスペースを割り振ります:
   num3 = %len (chr1 + chr2) + 1;
   ptr = %alloc (num3);
   %str (ptr: num3) = chr1 + chr2;
 /END-FREE
```

図 155. %DECPOS および %LEN の例

可変長フィールドの長さの設定に使用される %LEN

式の左側で使用される時に、この関数は可変長フィールドの現行の長さを設定します。設定された長さが現行の長さより大きい場合には、旧の長さと新しい長さの間のフィールド中の文字はブランクに設定されます。

注: %LEN を使用できるのは、そのパラメーターが可変長の時の式の左側でだけです。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
D city            S             40A   varying inz('North York')
D n1              S              5i 0

 * 可変長フィールドの現行の長さを取り出すために使用される %LEN:
 /FREE
    n1 = %len(city);
    // 現行の長さ n1 = 10

    // 可変長フィールドの現行の長さを設定するために使用される %LEN:
    %len (city) = 5;
    // city = 'North' (長さは 5)

    %len (city) = 15;
    // city = 'North          '  (長さは 15)
 /END-FREE
```

図 156. 可変長フィールドのある %LEN の例

%LOOKUPxx (配列エレメントの検索)

```
%LOOKUP(arg : 配列 {: startindex {: numelems}})
%LOOKUPLT(arg : 配列 {: startindex {: numelems}})
%LOOKUPGE(arg : 配列 {: startindex {: numelems}})
%LOOKUPGT(arg : 配列 {: startindex {: numelems}})
%LOOKUPLE(arg : 配列 {: startindex {: numelems}})
```

以下の関数は、*arg* と一致した*配列*内の項目の配列指標を次のように戻します:

%LOOKUP 完全な一致。

%LOOKUPLT *arg* に最も近いが、*arg* より小さい値。

%LOOKUPLE 完全な一致、あるいは *arg* に最も近いが、*arg* より小さい値。

%LOOKUPGT *arg* に最も近いが、*arg* より大きい値。

%LOOKUPGE 完全な一致、あるいは *arg* に最も近いが、*arg* より大きい値。

指定された条件と一致する値がない場合には、ゼロが戻されます。

検索は指標 *startindex* で始まり、*numelems* エレメントまで続行されます。デフォルトでは、配列全体が検索されます。

最初の 2 つのパラメーターは任意のタイプにできますが、同じタイプでなければなりません。それらが同じ長さまたは小数点以下の桁数である必要はありません。3 番目と 4 番目のパラメーターは、小数点以下の桁数がゼロの非浮動数値であることが必要です。

%LOOKUPLT、%LOOKUPLE、%LOOKUPGT、および %LOOKUPGE の場合には、配列はキーワード ASCEND または DESCEND とともに定義しなければなりません。

組み込み関数 %FOUND および %EQUAL は、%LOOKUP 演算命令に続いて設定されることはありません。

注: LOOKUP 命令コードと異なり、%LOOKUP は配列にのみ適用されます。テーブル内の値を検索するには、%TLOOKUP 組み込み関数を使用してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
  arr(1) = 'Cornwall';
  arr(2) = 'Kingston';
  arr(3) = 'London';
  arr(4) = 'Paris';
  arr(5) = 'Scarborough';
  arr(6) = 'York';

  n = %LOOKUP('Paris':arr);
  // n = 4

  n = %LOOKUP('Thunder Bay':arr);
  // n = 0 (見つかりません)

  n = %LOOKUP('Kingston':arr:3);
  // n = 0 (開始指標の後に見つかりません)

  n = %LOOKUPLE('Paris':arr);
  // n = 4

  n = %LOOKUPLE('Milton':arr);
  // n = 3

  n = %LOOKGT('Sudbury':arr);
  // n = 6

  n = %LOOKGT('Yorks':arr:2:4);
  // n = 0 (エレメントの 2 と 5 の間に見つかりません)
/END-FREE
```

図 157. %LOOKUPxx の例

%MINUTES (分数)

```
%MINUTES(数値)
```

%MINUTES は、数値を時刻または時刻スタンプ値に追加できる期間に変換します。

加算または減算演算命令では、%MINUTES は右側の値にのみすることができます。左側の値は時刻または時刻スタンプでなければなりません。結果は、該当する分数が加算または減算された時刻または時刻スタンプ値となります。時刻の場合の結果の値は *ISO 形式です。

日付および時刻算術演算の例については、 444 ページの図 158を参照してください。

%MONTHS (月数)

%MONTHS(数値)

%MONTHS は、数値を日付または時刻スタンプ値に追加できる期間に変換します。

加算または減算演算命令では、%MONTHS は右側の値にのみすることができます。左側の値は日付または時刻スタンプでなければなりません。結果は、該当する月数が加算または減算された日付または時刻スタンプ値となります。日付の場合の結果の値は *ISO 形式です。

多くの場合、一定の月数の加算または減算の結果は明らかです。たとえば、2000-03-15 + %MONTHS(1) は 2000-04-15 です。加算または減算よって存在しない日付 (たとえば、2 月 30 日) が生成された場合には、代わりにその月の最終日が使用されます。

月の 29、30、または 31 日目で 1 か月を加算または減算すると、逆戻りできないことがあります。たとえば、2000-03-31 + %MONTHS(1) - %MONTHS(1) は 2000-03-30 になります。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE

    // 3 年目の日付を判別します
    newdate = date + %YEARS(3);

    // 6 か月前の日付を判別します
    loandate = duedate - %MONTHS(6);

    // 日付と時刻から時刻スタンプを構成します
    duestamp = duedate + t'12.00.00';
/END-FREE
```

図 158. %MONTHS および %YEARS の例

%MSECONDS (マイクロ秒数)

```
%MSECONDS(数値)
```

%MSECONDS は、数値を時刻または時刻スタンプ値に追加できる期間に変換します。

加算または減算演算命令では、%MSECONDS は右側の値にのみすることができます。左側の値は時刻または時刻スタンプでなければなりません。結果は、該当するマイクロ秒数が加算または減算された時刻または時刻スタンプ値となります。時刻の場合の結果の値は *ISO 形式です。

日付および時刻算術演算の例については、 444 ページの図 158を参照してください。

%NULLIND (ヌル標識の照会または設定)

%NULLIND(フィールド名)

%NULLIND 組み込み関数を使用して、ヌルが使用可能なフィールドのヌル標識を照会または設定することができます。この組み込み関数が使用できるのは、ユーザー制御コンパイル・オプションまたは ALWNULL(*USRCTL) キーワードを指定した場合だけです。フィールド名はヌル値可能な配列エレメント、データ構造、独立フィールド、サブフィールド、または複数回繰り返しデータ構造とすることができます。

%NULLIND を使用できるのは、拡張演算項目 2 の式の中だけです。

式の右側で使用する時には、この関数はそのヌル値可能フィールドのヌル標識の設定を戻します。この設定は *ON または *OFF とすることができます。

式の左側で使用する時には、この関数を使用して、そのヌル値可能フィールドのヌル標識を *ON または *OFFに設定することができます。ヌル値可能フィールドの内容は未変更のままです。

ヌル値可能フィールドおよびキーをもつレコードの取り扱いについては、 145 ページの『データベースのヌル値サポート』 を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* ヌル値可能フィールドのヌル標識をテストします。
/FREE
   if  %nullind (fieldname1);
       // フィールドはヌルです
     endif;

   // ヌル値可能フィールドのヌル標識を設定します。
   %nullind(fieldname1) = *ON;
   %nullind (fieldname2) = *OFF;
/END-FREE
```

図 159. %NULLIND の例

%OCCUR (データ構造のオカレンスの設定/取り出し)

```
%OCCUR(dsn 名)
```

%OCCUR は、複数オカレンス・データ構造の現在位置を取り出しまたは設定します。

この関数がその値について評価されると、指定されたデータ構造の現在のオカレンス番号を戻します。これは符号なし数値です。

この関数が EVAL ステートメントの左側に指定された場合には、指定された番号が現在のオカレンス番号になります。これは、小数点以下の桁数がゼロの非浮動数値でなければなりません。値が 1 より小さいかまたはオカレンスの総数より大きい場合には、例外 00122 が出されます。

複数オカレンス・データ構造および OCCUR 命令コードの詳細については、622 ページの『OCCUR (データ構造のオカレンスの設定 / 取り出し)』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D mds             DS                  OCCURS(10)

 /FREE
   n = %OCCUR(mds);
   // n = 1

   %OCCUR(mds) = 7;

   n = %OCCUR(mds);
   // n = 7
 /END-FREE
```

図 160. %OCCUR の例

%OPEN (ファイル・オープン条件を戻す)

%OPEN(ファイル名)

指定されたファイルがオープンの場合には、%OPEN は '1' を戻します。ファイルが初期化中に RPG プログラムによって、または OPEN 演算命令によってオープンされ、その後クローズされていない場合には、そのファイルは "オープン"と見なされます。ファイルが外部標識によって条件付けされて、その外部標識がプログラム初期化時にオフであった場合には、そのファイルはクローズされたと見なされて、%OPEN は '0' を戻します。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * 印刷装置ファイルは演算仕様でオープンされます
FQSYSPRT   O    F  132        PRINTER USROPN

 /FREE
    // ファイルがまだオープンされていなければオープンします
    if not %open (QSYSPRT);
       open QSYSPRT;
     endif;
 /END-FREE
```

図 161. %OPEN の例

%PADDR (プロシージャー・アドレスの取り出し)

%PADDR(ストリング)

%PADDR はプロシージャー・ポインターのタイプの値を戻します。この値は、引き数によって識別されたエントリー・ポイントのアドレスです。

%PADDR を比較して割り当てることができるのは、プロシージャー・ポインターのタイプの項目に対してだけです。

%PADDR に対するパラメーターは文字または 16 進数リテラル、あるいは文字または 16 進数リテラルを表現する固定情報名でなければなりません。文字ストリングによって指定されたエントリー・ポイント名は、プログラム・バインド時に検出されて、正しい大文字小文字でなければなりません。

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D
D PROC            S               *   PROCPTR
D                                     INZ (%PADDR ('FIRSTPROG'))
D PROC1           S               *   PROCPTR
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C*
C*  次のステートメントはプロシージャー 'FIRSTPROG' を呼び出します。
C*
C                   CALLB     PROC
 *----------------------------------------------------------------
C* 次のステートメントはプロシージャー 'NextProg' を呼び出します。
C* これは C プロシージャーであり、大文字小文字混合です。
C* プロシージャー名は大文字・小文字が区別されることに注意してください。
C*
C                   EVAL      PROC1 = %PADDR ('NextProg')
C                   CALLB     PROC1
```

図 162. %PADDR の例

%REALLOC (ストレージの再割り振り)

```
%REALLOC(ptr:num)
```

%REALLOC は、最初のパラメーターによって指示されたヒープ・ストレージを 2 番目のパラメーターに指定された長さに変更します。新規に割り振られたストレージはまだ初期化されていません。

最初のパラメーターは基底ポインター値でなければなりません。2 番目のパラメーターは、小数点以下の桁数がゼロの非浮動数値でなければなりません。指定する長さは 1 ～ 16776704 の間になければなりません。

この関数は、割り振られたストレージを指すポインターを戻します。これは *ptr* と同じであることも、異なることもあります。

詳細については、370 ページの『メモリー管理命令』を参照してください。

演算命令を正常に完了できない場合には、例外 00425 または 00426 が出されます。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
  // 200 バイトのエリアを割り振ります
  pointer = %ALLOC(200);
  // エリアのサイズを 500 バイトに変更します
  pointer = %REALLOC(pointer:500);
/END-FREE
```

図 163. %REALLOC の例

%REM (整数剰余を戻す)

```
%REM(n:m)
```

%REM は、オペランド **n** を **m** で除算した結果の剰余を戻します。2 つのオペランドは小数点以下の桁数がゼロの数値でなければなりません。オペランドのいずれかがパック、ゾーン、または 2 進数の数字の場合には、その結果はパック数値となります。オペランドのいずれかが整数の数字の場合には、その結果は整数となります。そうでない場合には、その結果は符号なしの数字となります。浮動数字オペランドは使用できません。結果は被除数と同じ符号をもちます。(また、 420 ページの『%DIV (商の整数部分を戻す)』 を参照してください。)

%REM および %DIV は次の関係となります。

```
%REM(A:B) = A - (%DIV(A:B) * B)
```

オペランドが 8 バイト整数または符号なしのフィールドに適合できる固定情報の場合には、固定情報結合がその組み込み関数に適用されます。この場合には、%REM 組み込み関数は定義仕様にコーディングすることができます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D A               S             10I 0 INZ(123)
D B               S             10I 0 INZ(27)
D DIV             S             10I 0
D REM             S             10I 0
D E               S             10I 0

 /FREE
    DIV = %DIV(A:B);  // DIV は 4 になります
    REM = %REM(A:B);  // REM は 15 になります
    E = DIV*B + REM;  // E は 123 になります
 /END-FREE
```

図 164. %DIV および %REM の例

%REPLACE (文字ストリングの置き換え)

```
%REPLACE(置き換えストリング: ソース・ストリング{:開始位置 :置き換えるソースの長さ})
```

%REPLACE は、置き換えストリングをソース・ストリングに挿入して作成された文字ストリングを戻し、開始桁で開始されて、指定された文字数が置き換えられます。

最初と 2 番目のパラメーターが文字、グラフィック、または UCS-2 のタイプのものでなければなりません。固定長または可変長のいずれかの形式にすることができます。 2 番目のパラメーターは最初のものと同じタイプでなければなりません。

3 番目のパラメーターは置き換えストリングの開始桁 (文字数で測る) を示します。これが指定されない場合には、開始桁はソース・ストリングの先頭となります。この値は 1 から、ソース・ストリングの現行の長さに 1 を加えた範囲とすることができます。

4 番目のパラメーターは置き換えられるソース・ストリングの文字数を表します。ゼロを指定した場合には、置き換えストリングは指定された開始桁の前に挿入されます。このパラメーターが指定されない場合には、置き換えられる文字数は置き換えストリングの長さと同じになります。この値は、ゼロより大か等しく、ソース・ストリングの現行の長さと等しいかこれより小さくなければなりません。

開始桁および長さは小数点以下の桁数のない数値または数値表現とすることができます。

ソース・ストリングまたは置き換えストリングが可変長の場合、あるいは開始桁または置き換えるソース長が変数の場合には、戻される値は可変長です。そうでない場合には、その結果は固定長となります。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D var1            S             30A   INZ('Windsor') VARYING
D var2            S             30A   INZ('Ontario') VARYING
D var3            S             30A   INZ('Canada') VARYING
D fixed1          S             15A   INZ('California')
D date            S               D   INZ(D'1997-02-03')
D result          S            100A   VARYING

 /FREE
    result = var1 + ', ' + 'ON';
  // result = 'Windsor, ON'

  // ストリングの先頭にあるテキストを置き換えるための 2 つのパラメーターのある %REPLACE:
    result = %replace ('Toronto': result);
  // result = 'Toronto, ON'

  // 指定された位置にあるテキストを置き換えるための 3 つのパラメーターのある %REPLACE:
    result = %replace (var3: result: %scan(',': result) + 2);
  // result = 'Toronto, Canada'

  // テキストを挿入するための 4 つのパラメーターのある %REPLACE:
    result = %replace (', ' + var2: result: %scan (',': result): 0);
  // result = 'Toronto, Ontario, Canada'

  // 異なる長さのストリングを置き換えるための 4 つのパラメーターのある %REPLACE
    result = %replace ('Scarborough': result:
     1: %scan (',': result) - 1);
  // result = 'Scarborough, Ontario, Canada'

  // テキストを削除するための 4 つのパラメーターのある %REPLACE:
    result = %replace ('': result: 1: %scan (',': result) + 1);
  // result = 'Ontario, Canada'

  // ストリングの終わりにテキストを追加するための 4 つのパラメーターのある %REPLACE:
    result = %replace (', ' + %char(date): result:
   %len (result) + 1: 0);
  // result = 'Ontario, Canada, 1997-02-03'

  // 指定された位置の固定長テキストを置き換えるための 3 つのパラメーターの
  // ある %REPLACE:  (fixed1 は 15 文字の固定長)
    result = %replace (fixed1: result: %scan (',': result) + 2);
  // result = 'Ontario, California     -03'

  // 先頭のテキストに接頭部を付けるための 4 つのパラメーターのある %REPLACE:
    result = %replace ('Somewhere else: ': result: 1: 0);
  // result = 'Somewhere else: Ontario, California     -03'
 /END-FREE
```

図 165. %REPLACE の例

%SCAN (文字のスキャン)

```
%SCAN(検索引き数 : ソース・ストリング {: 開始})
```

%SCAN はソース・ストリングの検索引き数の最初の桁を戻します。これが検出されなかった場合には、0 を戻します。開始桁を指定した場合には、検索は開始桁で開始されます。開始桁を指定した場合でも、その結果は常にソース・ストリング中の桁となります。開始桁のデフォルトは 1 です。

最初のパラメーターは文字、または UCS-2 のタイプのものでなければなりません。2 番目のパラメーターは最初のパラメーターと同じタイプでなければなりません。3 番目のパラメーター (指定された場合) は、小数点以下の桁数がゼロの数字でなければなりません。

パラメーターのいずれかが可変長の場合には、他のパラメーターの値は最大長ではなく、現行の長さに対してチェックされます。

戻り値のタイプは符号なし整数です。この組み込み関数は、符号なし整数の式が有効となるどの位置でも使用することができます。

注: SCAN 演算コードとは異なり、%SCAN は検索文字列のすべてのオカレンスを含む配列を戻すことはできないし、%FOUND 組み込み関数を使用してその結果をテストすることもできません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D source          S             15A   inz ('Dr. Doolittle')
D pos             S              5U 0

 /FREE
    pos = %scan ('oo' : source);
    // EVAL の後、'oo' が 'Dr. Doolittle' の 6 桁目で始まるので
    // pos = 6 となります。

    pos = %scan ('D' : source : 2);
    // EVAL の後、2 桁目から始まって検出された最初の 'D' は 5 桁目
    // であるので、 pos = 5 となります。

    pos = %scan ('abc' : source);
    // EVAL の後、'abc' が 'Dr. Doolittle' で検出されないので、
    // pos = 0 となります。

    pos = %scan ('Dr.' : source : 2);
    // EVAL の後、検索が 2 桁目で始まる場合には、'Dr.' が
    // 'Dr. Doolittle' で検出されないので、pos = 0 となります。
 /END-FREE
```

図 166. %SCAN の例

%SECONDS (秒数)

```
%SECONDS(数値)
```

%SECONDS は、数値を時刻または時刻スタンプ値に追加できる期間に変換します。

加算または減算演算命令では、%SECONDS は右側の値にのみすることができます。左側の値は時刻または時刻スタンプでなければなりません。結果は、該当する秒数が加算または減算された時刻または時刻スタンプ値となります。時刻の場合の結果の値は *ISO 形式です。

日付および時刻算術演算の例については、 444 ページの図 158を参照してください。

%SETATR (属性の設定)

%SETATR(ウィンドウ名:パーツ名:属性名)

%SETATR はウィンドウ上のあるパーツの属性値を設定します。最初のパラメーターと 2 番目のパラメーターの両方を %WINDOW または %PART とすることができます。

パーツ属性にアクセスするための別の形式については、382 ページの『修飾 GUI パーツ属性アクセス』を参照してください。

注:

1. %SETATR 組み込み関数は複数のパーツの対応するプログラム・フィールドには影響しません。 属性値およびプログラム・フィールド中の値が同一であることを保証するには、属性値の設定時に、このプログラム・フィールドを使用してください。これは、それらにマップされるプログラム・フィールド (TEXT 属性が指定されている入力フィールドなど) がある属性に適用されます。
2. %SETATR 組み込み関数は 1 バイトおよび 8 バイトの符号付きおよび符号なしの整数値、およびユニコード値をサポートしていません。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
     ENT0000B = *BLANKS;
     %setatr('inventory':'ent0000b':'text') = ENT0000B;
/END-FREE
```

図 167. %SETATR の例

%SIZE (固定情報またはフィールドのサイズ)

```
%SIZE(変数)
%SIZE(リテラル)
%SIZE(配列{:*ALL})
%SIZE(テーブル{:*ALL})
%SIZE(複数回繰り返しデータ構造{:*ALL})
```

%SIZE は固定情報またはフィールドによって占有されているバイト数を戻します。引き数はリテラル、名前付き固定情報、データ構造、データ構造サブフィールド、フィールド、配列、またはテーブル名とすることができます。これに式を含めることはできませんが、一部の固定情報値組み込み関数および固定情報式は受け入れられます。戻される値は符号なしの整数の形式 (タイプ U) です。

グラフィック・リテラルでは、このサイズはそのグラフィック文字によって占有されたバイト数です。 16 進数または UCS-2 リテラルでは、戻されるサイズはそのリテラルの 16 進数桁数の半分です。

可変長フィールドでは、%SIZE はそのフィールドによって占有された合計バイト数 (宣言された最大長より 2 バイト短い) を戻します。

引き数が配列名、テーブル名、または複数回繰り返しデータ構造名の場合には、戻される値は 1 つのエレメントまたはオカレンスのサイズです。 %SIZE の2 番目のパラメーターとして *ALL が指定された場合には、戻される値はすべてのエレメントまたはオカレンスが占有するストレージです。ポインター・サブフィールドを含む複数回繰り返しデータ構造では、このサイズは、1 つのオカレンスのサイズにオカレンス数を乗算したものより大きくなることがあります。システムでは、ポインターはストレージ内で 16 で割りきれるアドレスに入れる必要があるために、これが起こる場合があります。これは、ポインター・サブフィールドがすべてのオカレンスで正しくストレージ内に位置指定されるように、各オカレンスの長さは 16 の倍数の長さにするために増大する必要があることを意味します。

%SIZE は、定義仕様で数値固定情報が許可される任意の位置に、また、演算仕様の拡張演算項目 2フィールドの式に指定することができます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
D arr1            S             10    DIM(4)
D table1          S              5    DIM(20)
D field1          S             10
D field2          S              9B 0
D field3          S              5P 2
D num             S              5P 0
D mds             DS            20    occurs(10)
D mds_size        C                   const (%size (mds: *all))
D mds_ptr         DS            20    OCCURS(10)
D   pointer                       *
D vCity           S             40A   VARYING INZ('North York')
D fCity           S             40A           INZ('North York')

 /FREE
     num = %SIZE(field1);        //  10
     num = %SIZE('HH');          //   2
     num = %SIZE(123.4);         //   4
     num = %SIZE(-03.00);        //   4
     num = %SIZE(arr1);          //  10
     num = %SIZE(arr1:*ALL);     //  40
     num = %SIZE(table1);        //   5
     num = %SIZE(table1:*ALL);   // 100
     num = %SIZE(mds);           //  20
     num = %SIZE(mds:*ALL);      // 200
     num = %SIZE(mds_ptr);       //  20
     num = %SIZE(mds_ptr:*ALL);  // 320
     num = %SIZE(field2);        //   4
     num = %SIZE(field3);        //   3
     n1 = %SIZE(vCity);          //  42
     n2 = %SIZE(fCity);          //  40
 /END-FREE
```

図 168. %SIZE の例

%SQRT (式の平方根)

%SQRT(数値表現)

%SQRT は、指定された数値表現の平方根を戻します。オペランドのタイプが浮動である場合には、結果は浮動タイプとなり、それ以外の場合には、結果はパック 10 進数になります。パラメーターの値がゼロより小さい場合には、例外 00101 が出されます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D n               S             10I 0
D p               S              9P 2
D f               S              4F

 /FREE

   n = %SQRT(239874);
   // n = 489

   p = %SQRT(239874);
   // p = 489.76

   f = %SQRT(239874);
   // f = 489.7693
 /END-FREE
```

図 169. %SQRT の例

%STATUS (ファイルまたはプログラム状況を戻す)

```
%STATUS{(ファイル名)}
```

%STATUS はプログラムまたはファイル状況の最新の値セットを戻します。プログラム状況またはいずれかのファイル状況が変更された時 (通常、エラーが起こった時) には常に %STATUS が設定されます。

%STATUS が任意選択のファイル名パラメーターなしで使用された場合には、これは、最後に変更されたプログラムまたはファイル状況を戻します。ファイルを指定した場合には、指定されたファイルの INFDS *STATUS フィールドに含まれた値が戻されます。そのファイルに INFDS を指定する必要はありません。

%STATUS は 00000 の戻り値で開始して、指定された 'E' 拡張機能でどれかの演算命令を開始する前に 00000に 設定されます。

%STATUS は、'E' 拡張機能の演算命令または指定されたエラー標識、あるいは INFSR または *PSSR サブルーチンの開始の直後にチェックされます。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* 'E' 拡張機能は、エラーが起こった場合に、エラー標識がコーディング
* されたかのようにそのエラーを取り扱います。
* そして、演算命令の成功は %ERROR 組み込み関数を使用して検査する
* ことができます。そのエラーと関連した状況は %STATUS 組み込み関数
* を使用して検査することができます。
/FREE
   read(e) InFile;
            if %error;
      exsr CheckError;
     endif;

 //---------------------------------------------------------------------
 // CheckError: ファイル入出力エラーを処理するためのサブルーチン
 //---------------------------------------------------------------------
   begsr CheckError;
      select;
      when  %status < 01000;

         // エラーは起こっていません
      when  %status = 01211;
         // オープンされていないファイルを読み取ろうとしました
         exsr InternalError;

      when  %status = 01331;
         // READ 演算命令の待ち時間を超えました
         exsr TimeOut;

      when  %status = 01251;
         // 永続入出力エラー
         exsr PermError;

      other;
         // 他の何らかのエラーが起こりました
         exsr FileError;
      endsl;
     endsr;
/END-FREE
```

図 170. 'E' 拡張機能のある %STATUS および %ERROR

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D Zero            S               5P 0 INZ(0)
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 * %STATUS は 0 の値で始まります
 *
 * 開始桁に 0 の値があるので、次の SCAN 演算命令で *PSSR に
 * 分岐することになります。
C     'A'           SCAN      'ABC':Zero    Pos
C     BAD_SCAN      TAG
 * 次の EXFMT 演算命令には 'E' 拡張機能があるので、%STATUS は
 * 演算命令を開始する前に 0 に設定されます。したがって、%STATUS
 * の検査はその演算命令の後で有効となります。
 * 'E' 拡張機能がコーディングされているので、%ERROR を使用して、
 * エラーが起こったかどうかを検査することもできます。
C                   READ(E)  REC1
C                   IF        %ERROR
C                   SELECT
C                   WHEN      %STATUS = 01211
C ...
C                   WHEN      %STATUS = 01299
C ...
 * 次のスキャン演算命令にはエラー標識があります。演算命令の開始の
 * 前に %STATUS は 0 に設定されませんが、エラー標識がオンの場合に
 * は、%STATUS を正当に検査することができます。
C     'A'           SCAN      'ABC':Zero    Pos                   10
C                   IF        *IN10 AND %STATUS = 00100
C                     ...

 * 次のスキャン演算命令ではエラーにはなりません。
 * 'E' 拡張機能がないので、%STATUS は 0 に設定されずに、
 * 前のエラーから 00100 の値が戻されます。
 * したがって、前の演算命令と関連した値が確かでない
 * ので、エラー標識または 'E' 拡張機能がコーディング
 * されていない演算命令の後に %STATUS を使用するのは
 * お勧めできません。
C     'A'           SCAN      'ABC'         Pos
C ...
C     *PSSR         BEGSR
 * エラーが起こっているはずなので、*PSSR で %STATUS を使用することができます。
C                   IF        %STATUS = 00100
C                   GOTO      BAD_SCAN
C ...
```

*図 171. 'E' 拡張機能、エラー標識および *PSSR のある %STATUS および %ERROR*

%STR (ヌル文字終了ストリングの取り出しまたは保管)

```
%STR(基底ポインター{: 最大長})(右側)
%STR(基底ポインター : 最大長)(左側)
```

%STR を使用して、ヌル終了文字ストリングを作成または使用し、これは、C および C++ アプリケーションでは通常に使用されます。

最初のパラメーターは基底ポインター変数でなければなりません。2 番目のパラメーター (指定された場合) は、小数点以下の桁数がゼロの数値でなければなりません。指定されない場合には、このデフォルトとして 65535 が使用されます。

最初のパラメーターは、少なくとも 2 番目のパラメーターによって指定される長さであるかぎり、ストレージを指示していなければなりません。

エラー条件:

1. 長さパラメーターが 1 - 65535 の範囲でない場合には、状況 00100 のエラーが起こります。
2. ポインターが設定されない場合には、状況コード 00222 のエラーが起こります。
3. ポインターによってアドレス指定されたストレージが長さパラメーターによって指示されたものより短い場合には、次のいずれかとなります。
   a. 状況コード 00222 のエラーが起こります。
   b. データ汚損が起こります。

ヌル終了ストリングを取り出すために使用される %STR

式の右側で使用する時には、この関数は、最初のパラメーターで指示されたデータまで (だが、指定された長さ内で最初に検出されたヌル文字 (x'00') は含まれない) を戻します。この組み込み関数は、文字式が有効となるどの位置でも使用することができます。指定された長さ内でヌル文字が検出されない場合には、実行時にエラーは示されません。この場合に、結果の値の長さは指定された長さと同じになります。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D String1         S               *
D Fld1            S             10A

 /FREE
     Fld1 = '<' + %str(String1) + '>';
    // String1 が '123¬' を指し、'¬' はヌル文字を表現すると、
    // EVAL の後、Fld1 = '<123>     ' となります。
 /END-FREE
```

図 172. %STR (右側) の例 1

以下は、2 番目のパラメーターが指定された %STR の例です。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D String1         S               *
D Fld1            S             10A

 /FREE
     Fld1 = '<' + %str(String1 : 2) + '>';
  // String1 が '123¬' を指し、'¬' はヌル文字を表現すると、
  // EVAL の後、Fld1 = '<12>      ' となります。
  // 演算命令によって読み取られた最大長は 2 であったので、「3」および
  // '¬' は考慮されませんでした。
 /END-FREE
```

図 173. %STR (右側) の例 2

この例では、指定された最大長の中でヌル終止符が検出されます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D String1         S               *
D Fld1            S             10A

 /FREE
    Fld1 = '<' + %str(String1 : 5) + '>';
    // String1 が '123¬' を指し、'¬' はヌル文字を表現すると、
    // EVAL の後、Fld1 = '<123>     ' となります。
    // 演算命令によって読み取られた最大長は 5 であったので、
    // 4 桁目のヌル終止符が検出されて、そのヌル文字までのすべて
    // のデータが使用されました。
 /END-FREE
```

図 174. %STR (右側) の例 3

ヌル終了ストリングを保管するために使用される %STR

式の左側で使用される場合には、%STR(ptr:length) はその式の右側の値を、ポインターで示されたストレージに割り当てて、その終わりにヌル終了バイトを追加します。指定できる最大長は 65535です。これは、終わりのヌル終了用に 1 バイトを未使用にする必要があるので、右側の多くても 65534 バイトしか使用できないことを意味します。

この長さは、そのポインターが示すストレージの容量を示します。この長さは、右側に入れる最大長より大きくなければなりません。このポインターは、少なくとも長さパラメーターのストレージを指すように設定しなければなりません。式の右側の長さが指定の長さより長い場合には、右側の値が切り捨てられます。

注: 次が真の場合には、データ汚損が起こります。

1. 長さパラメーターがポインターでアドレス指定される実際のデータの長さより大きい場合
2. 右側の長さがポインターでアドレス指定される実際のデータの長さよりより大か等しい場合

%STR で使用するためにストレージを動的に割り振っている場合には、割り振った長さに注意していなければなりません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D String1         S               *
D Fld1            S             10A

 /FREE
     %str(String1(25))= 'abcdef';
     // String1 によって現在指示されているストレージには 'abcdef¬' が含まれています
     // ヌル終止符より後のバイト 8 ～ 25 は未変更です。

     %str (String1: 4) = 'abcdef';
     // String1 によって現在指示されているストレージには 'abc¬' が含まれています
 /END-FREE
```

図 175. %STR (左側) の例

%SUBDT (日付、時刻、または時刻スタンプの部分の取り出し)

```
%SUBDT(値:*MSECONDS|*SECONDS|*MINUTES|*HOURS|*DAYS|*MONTHS|*YEARS)
%SUBDT(値:*MS|*S|*MN|*H|*D|*M|*Y)
```

%SUBDT は、日付、時刻、または時刻スタンプ値の情報の部分を取り出します。これは符号なし数値を戻します。

最初のパラメーターは、日付、時刻、または時刻スタンプ値です。

2 番目のパラメーターは、取り出したい部分です。次の値が有効です:

- 日付の場合: *DAYS、*MONTHS、および *YEARS
- 時刻の場合: *SECONDS、*MINUTES、および *HOURS
- 時刻スタンプの場合: *MSECONDS、*SECONDS、*MINUTES、*HOURS、*DAYS、*MONTHS、および *YEARS

この関数の場合に、*DAYS は (年間通算日形式を使用している場合であっても) 常に年の日ではなく月の日を参照します。たとえば、2 月 10 日の日部分は 41 ではなく 10 です。

日付形式に 2 桁の年が含まれる場合であっても、この関数は常に 4 桁の年を戻します。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE

    date = d'1999-02-17';
    time = t'01.23.45';

    num = %subdt(date:*YEARS);
    // num = 1999

    num = %subdt(time:*MN);
    // num = 23
/END-FREE
```

図 176. %SUBDT の例

%SUBST (サブストリングの取り出し)

```
%SUBST(ストリング:開始{:長さ})
```

%SUBST 引き数ストリングの一部を戻します。また、引き数の結果として EVAL 演算コードとともにこれを使用することもできます。

start パラメーターはサブストリングの開始桁を示します。

length パラメーターはサブストリングの長さを示します。これが指定されない場合には、この長さは、string パラメーターから、開始値に 1 を加えたものを差し引いた長さとなります。

このストリングは 文字、グラフィック、または UCS-2 データでなければなりません。開始桁および長さは小数点以下の桁数がゼロの数値または数値表現とすることができます。開始桁はゼロより大きくなければなりません。長さはゼロより大か等しくすることができます。

ストリング・パラメーターが可変長の場合には、他のパラメーターの値は最大長ではなく、現行の長さに対してチェックされます。

定義仕様キーワードのパラメーターとして指定した場合には、このパラメーターはリテラルまたはリテラルを表現する名前付き固定情報でなければなりません。フリー・フォームの演算仕様に指定する場合には、パラメーターは任意の式とすることができます。

その値で使用される %SUBST

%SUBST は指定されたストリングの内容からサブストリングを戻します。このストリングは任意の 文字、グラフィック、または UCS-2 フィールドまたは式とすることができます。ストリング、開始、および長さでは、指数なしの配列を使用することができます。このサブストリングはストリングに指定された開始桁で始まり、指定された長さにわたり続行します。長さが指定されない場合には、サブストリングはそのストリングの終わりまで続行します。たとえば:

```
%subst('Hello World': 5+2) の値は  'World'  です。
%subst('Hello World':5+2:10-7) の値は 'Wor' です。
%subst('abcd' + 'efgh':4:3) の値 'def' です。
```

グラフィックまたは UCS-2 文字では、開始桁長さは 2 バイト文字長と整合します (3 桁目は 3 番目の 2 バイト文字であり、長さ 3 は 3 個の 2 バイト文字が演算されることを示す)。

割り当ての結果として使用される %SUBST

割り当ての結果として使用される時には、この組み込み関数は引き数ストリングの特定の位置を示します。指標なしの配列は開始および長さに使用することはできません。

この結果は変数に指定された開始桁で始まり、指定された長さにわたり続行します。長さが指定されないか、あるいはそれがストリングの終わりを超えて文字を示す場合には、そのストリングはその終わりを示します。

%SUBST が割り当ての結果として使用される場合には、最初のパラメーターは記憶場所を示していなければなりません。すなわち、%SUBST 演算命令の最初のパラメーターは次の 1 つでなければなりません。

- フィールド
- データ構造
- データ構造サブフィールド
- 配列名
- 配列エレメント
- テーブル・エレメント

%SUBST が EVAL 演算での割り当ての結果として表示される時には、その 2 番目および 3 番目のパラメーターに有効な任意の式を使用することができます。

```
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C*
C* この例では、CITY には 'Toronto, Ontario' が含まれます。
C* %SUBST は値 'Ontario' を戻します。
C*
C     ' '           SCAN      CITY          C
C                   IF        %SUBST(CITY:C+1) = 'Ontario'
C                   EVAL      CITYCNT = CITYCNT+1
C                   ENDIF
C*
C* EVAL の前に、 A には値 'abcdefghijklmno' があります。
C* EVAL の後に、 A には値 'ab****ghijklmno' があります。
C*
C                   EVAL      %SUBST(A:3:4) = '****'
```

図 177. %SUBST の例

%THIS (ネイティブ・メソッドに対するクラス・インスタンスの戻し)

```
%THIS
```

%THIS は、それに代わってネイティブ・メソッドが呼び出されているクラス・インスタンスに対する参照を含むオブジェクト値を戻します。%THIS は非静的ネイティブ・メソッドでのみ有効です。この組み込み関数は、クラス・インスタンスに対する非静的ネイティブ・メソッド・アクセスを提供します。

非静的ネイティブ・メソッドは、そのクラスの特定のインスタンスに作用します。このオブジェクトは、実際には、Java によってパラメーターとしてネイティブ・メソッドに渡されますが、ネイティブ・メソッド用のプロトタイプまたはプロシージャー・インターフェースには現れません。Java メソッドにおいては、オブジェクト・インスタンスは Java の予約語である ***this*** によって参照が指示されます。RPG ネイティブ・メソッドにおいては、オブジェクト・インスタンスは %THIS 組み込み関数によって参照が指示されます。

```
 * メソッド 「vacationDays」はクラス 「Employee」内のメソッドです
D vacationDays    PR            10I 0 EXTPROC(*JAVA
D                                     : 'Employee'
D                                     : 'vacationDays')

 * メソッド「getId」はクラス「Employee」内のもう 1 つのメソッドです
D getId           PR            10I 0 EXTPROC(*JAVA
D                                     : 'Employee'
D                                     : 'getId')
...
 * 「vacationDays」は RPG ネイティブ・メソッドです。STATIC キーワード
 * が使用されていないので、これはインスタンス・メソッドです。
P vacationDays    B
D vacationDays    PI            10I 0

D id_num          S             10I 0

 * Employee の ID 番号を取り出すには、別の Employee メソッドが必要です。
 * このメソッドはクラス Employee のオブジェクトを必要とします。
 * ここでのネイティブ・メソッド「vacationDays」が作用するオブジェクトの ID 番号
 * を取り出すために、オブジェクト・パラメーターとして %THIS を使用します。
C                   eval      id_num = getId(%THIS)
C     id_num        chain     EMPFILE
C                   if        %found
C                   return    VACDAYS
C                   else
C                   return    -1
C                   endif

P vacationDays    E
```

図 178. %THIS の例

%TIME (時刻に変換)

```
%TIME{(式{:時刻形式})}
```

%TIME は、式の値を文字、数値、または時刻スタンプ・データから時刻タイプに変換します。変換された値は未変更のまま残されますが、時刻として戻されます。

最初のパラメーターは変換する値です。値を指定しない場合には、%TIME は現行システム時刻を戻します。

2 番目のパラメーターは、数値または文字入力のための時刻形式です。入力形式とは無関係に、出力は *ISO 形式で戻されます。

使用できる入力形式については、 144 ページの『時刻データ』を参照してください。数値または文字入力のための時刻形式が指定されない場合に、デフォルト値は TIMFMT 制御仕様キーワードに指定された形式か *ISO のどちらかになります。詳細については、 235 ページの『TIMFMT(fmt{区切り文字})』を参照してください。

最初のパラメーターが時刻スタンプである場合には、2 番目のパラメーターは指定しないようにしてください。この場合には、システムが入力の形式を認識しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE

   string = '12:34 PM';
   time =  %time(string:*USA);
   // time = t'12.34.00'
/END-FREE
```

図 179. %TIME の例

%TIMESTAMP (時刻スタンプに変換)

```
%TIMESTAMP{(式{:*ISO|*ISO0})}
```

%TIMESTAMP は、式の値を文字、数値、または日付データから時刻スタンプ・タイプに変換します。変換された値は時刻スタンプとして戻されます。

最初のパラメーターは変換する値です。値を指定しない場合には、%TIMESTAMP は現行システム時刻スタンプを戻します。

2 番目のパラメーターは、文字入力のための時刻スタンプ形式です。入力形式とは無関係に、出力は *ISO 形式で戻されます。*ISO (デフォルト) か *ISO0 のどちらかを指定することができます。詳細については、 145 ページの『タイム・スタンプ・データ』を参照してください。

最初のパラメーターが数値である場合には、2 番目のパラメーターを指定する必要はありません。許可されている値は *ISO (デフォルト) のみです。

最初のパラメーターが日付である場合には、2 番目のパラメーターは指定しないようにしてください。システムは、日付をその現行形式から *ISO 形式に変換して、00.00.00.0000 を追加します。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE

  string = '1960-09-29-12.34.56.000000';
  timest =  %timestamp(string);
  // timest には現在 t'1960-09-29-12.34.56.000000' が含まれています
/END-FREE
```

図 180. %TIMESTAMP の例

%TLOOKUPxx (テーブル・エレメントの検索)

```
%TLOOKUP(arg : 検索テーブル {: alt-table})
%TLOOKUPLT(arg : 検索テーブル {: alt-table})
%TLOOKUPGE(arg : 検索テーブル {: alt-table})
%TLOOKUPGT(arg : 検索テーブル {: alt-table})
%TLOOKUPLE(arg : 検索テーブル {: alt-table})
```

以下の関数は、*arg* と一致した値の検索テーブルを次のように検索します:

%TLOOKUP 完全な一致。

%TLOOKUPLT
arg に最も近いが、*arg* より小さい値。

%TLOOKUPLE
完全な一致、あるいは *arg* に最も近いが、*arg* より小さい値。

%TLOOKUPGT
arg に最も近いが、*arg* より大きい値。

%TLOOKUPGE
完全な一致、あるいは *arg* に最も近いが、*arg* より大きい値。

値が指定された条件を満たしている場合には、検索テーブルの現行テーブル・エレメントは条件を満たすエレメントに設定され、代替テーブルの現行テーブル・エレメントは同じエレメントに設定され、また、関数は値 *ON を戻します。

指定された条件と一致する値がない場合には、*OFF が戻されます。

最初の 2 つのパラメーターは任意のタイプにできますが、同じタイプでなければなりません。それらが同じ長さまたは小数点以下の桁数である必要はありません。

組み込み関数 %FOUND および %EQUAL は、%LOOKUP 演算命令に続いて設定されることはありません。

注: LOOKUP 命令コードと異なり、%TLOOKUP はテーブルにのみ適用されます。配列内の値を検索するには、%LOOKUP 組み込み関数を使用してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
  *IN01 = %TLOOKUP('Paris':tab1);
  IF %TLOOKUP('Thunder Bay':tab1:tab2);
  // Thunder Bay を取り扱うためのコード
    ENDIF;
/END-FREE
```

図 181. %TLOOKUPxx の例

%TRIM (端のブランクをトリム)

%TRIM(ストリング)

%TRIM は先行および末尾ブランクをすべて除去して指定されたストリングを戻します。

このストリングは文字、グラフィック、または UCS-2 データとすることができます。

定義仕様キーワードのパラメーターとして指定する場合には、このストリング・パラメーターは固定情報でなければなりません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D Location        S             16A
D FirstName       S             10A   inz ('   Chris')
D LastName        S             10A   inz ('   Smith')
D Name            S             20A

 * LOCATION は値 'Toronto, Ontario' をもつことになります。
 /FREE
     Location = %trim ('  Toronto, Ontario  ');

     // Name は値 'Chris Smith' をもつことになります。
     Name = %trim (FirstName) + ' ' + %trim (LastName);
 /END-FREE
```

図 182. %TRIM の例

%TRIML (先行ブランクをトリム)

%TRIML(ストリング)

%TRIML は先行ブランクをすべて除去して指定されたストリングを戻します。

このストリングは文字、グラフィック、または UCS-2 データとすることができます。

定義仕様キーワードのパラメーターとして指定する場合には、このストリング・パラメーターは固定情報でなければなりません。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* LOCATION は値 'Toronto, Ontario' をもつことになります。
/FREE
   Location = %triml('  Toronto, Ontario  ');
/END-FREE
```

図 183. %TRIML の例

%TRIMR (末尾ブランクをトリム)

%TRIMR(ストリング)

%TRIMR は末尾ブランクをすべて除去して指定されたストリングを戻します。

このストリングは文字、グラフィック、または UCS-2 データとすることができます。

定義仕様キーワードのパラメーターとして指定する場合には、このストリング・パラメーターは固定情報でなければなりません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D Location        S             16A   varying
D FirstName       S             10A   inz ('Chris')
D LastName        S             10A   inz ('Smith')
D Name            S             20A   varying

 * LOCATION は値 '  Toronto, Ontario' をもつことになります。
 /FREE
     Location = %trim ('  Toronto, Ontario  ');

     // Name は値 'Chris Smith:' をもつことになります。
     Name = %trimr (FirstName) + ' ' + %trimr (LastName) + ':';
 /END-FREE
```

図 184. %TRIMR の例

%UCS2 (UCS-2 値に変換する)

%UCS2(文字式 | グラフィック式)

%UCS2 は文字またはグラフィックから式の値を変換して、UCS-2 値を戻します。パラメーターが可変長の場合、あるいはパラメーターが 1 バイト文字の場合には、その結果は可変長となります。

2 番目のパラメーターの *ccsid* は任意指定であり、結果の式の CCSIDを指示します。 CCSID のデフォルトとして 13488 が使用されます。

パラメーターが固定情報の場合には、変換はコンパイル時に行われます。

変換の結果が置換文字となった場合には、コンパイル時に警告メッセージが出されます。実行時に、状況 00050 が設定されて、エラー・メッセージは出されません。

```
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H CCSID(*UCS2 : 13488)
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D  char           S               5A   INZ('abcde')
D  graph          S               2G   INZ(G'oAABBi')
 * %UCS2 組み込み関数は UCS-2 フィールドの初期化に使用されます。
D  ufield         S              10C   INZ(%UCS2('abcdefghij'))
D  ufield2        S               1C   CCSID(61952) INZ(*LOVAL)
D  isLess                         1N
D  proc           PR
D    uparm                        2G   CCSID(13488) CONST
CSRN01Factor1+++++++Opcode&ExtExtended-factor2++++++++++++++++++++++++++
C                   EVAL      ufield = %UCS2(char) +  %UCS2(graph)
 * ufield は現在 'a.b.c.d.e.AABB' を表す 7 桁の UCS-2 文字をもち、
 * ここで 'x.' は UCS-2 形式の 'x' を表します。
C                   EVAL      isLess = ufield < %UCS2(ufield2:13488)
 * %UCS2 組み込み関数の結果は ufield2 となり、
 * 比較のために CCSID 61952 から CCSID 13488 に変換されます。
C                   EVAL      ufield = ufield2
 * ufield2 の値は CCSID 61952 から CCSID 13488
 * に変換されて、ufield に保管されます。
 * この変換はコンパイラーによって暗黙的に取り扱われます。

C                   CALLP     proc(ufield2)
 * ufield2 の値は、固定情報参照によってパラメーターを渡す
 * 一部として暗黙的に CCSID 13488 に変換されます。
```

図 185. %UCS2 の例

%UNS (符号なし形式に変換)

%UNS(数値表現)

%UNS は、数値表現の値を符号なし形式に変換します。10 進数字のどれかの切り捨てが行われます。 %UNS を使用して浮動または 10 進数の小数点以下の桁数を切り捨てて、それを配列指数として使用することができます。

図 186は、%UNS 組み込み関数の例を示しています。

%UNSH (四捨五入付き符号なし形式に変換)

%UNSH(数値表現)

%UNSH は %UNS と同じですが、数値表現が 10 進数または浮動値の場合に、符号なしタイプに変換する時に数値表現の値に四捨五入が適用される点が異なります。四捨五入が実行できない場合には、メッセージは出されません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D p7              s              7p 3 inz (8236.567)
D s9              s              9s 5 inz (23.73442)
D f8              s              8f   inz (173.789)
D result1         s             15p 5
D result2         s             15p 5
D result3         s             15p 5
D array           s              1a   dim (200)
D a               s              1a

 /FREE
    result1 = %uns (p7) + 0.1234; // "result1" は 8236.12340 になります
    result2 = %uns (s9);          // "result2" は 23.00000 になります
    result3 = %unsh (f8);         // "result3" は 174.00000 になります

    // %UNS および %UNSH は配列指標として使用することができます
    a = array (%unsh (f8));
 /END-FREE
```

図 186. %UNS および %UNSH の例

%XFOOT (配列式エレメントを合計)

```
%XFOOT(配列式)
```

%XFOOT は、指定された数字配列式のすべてのエレメントの合計の結果となります。

結果の精度は、すべての配列エレメントを加算した結果 (最大 30 桁まで) を入れることができる最小です。結果の小数点以下の桁数は配列式の小数点以下の桁数と同じです。

たとえば、ARR が精度 (17,4) の 500 個のエレメントの配列の場合には、%XFOOT(ARR) の結果は (20,4) となります。

X が精度 (m,n) である %XFOOT(X) では、次のテーブルは X のエレメント数に基づいた結果の精度を示します。

X のエレメント数	%XFOOT(X) の制度
1	(m,n)
2-10	(m+1,n)
11-100	(m+2,n)
101-1000	(m+3,n)
1001-10000	(m+4,n)
10001-32767	(m+5,n)

配列式の通常の規則が適用されます。たとえば、ARR1 に 10 個のエレメントがあり、ARR2 に 20 個のエレメントがある場合には、%XFOOT(ARR1+ARR2) の結果は ARR1+ARR2 の最初の 10 個のエレメントの合計となります。

この組み込み関数は XFOOT 演算命令と類似していますが、浮動配列が他のすべてのタイプのように、索引 1 から開始して合計される点が相違しています。

%XLATE (変換)

```
%XLATE(from:to:string{:startpos})
```

%XLATE は、*from*、*to*、および *startpos*の値に従って *string* を変換します。

最初のパラメーターには、置き換えるべき文字のリストが含まれ、2 番目のパラメーターには、その置き換え文字が含まれます。たとえば、ストリングに *from* の 3 番目の文字が含まれている場合には、その文字のすべてのオカレンスが *to* の 3 番目の文字で置き換えられます。

3 番目のパラメーターは、変換するストリングです。4 番目のパラメーターは、変換の開始桁です。デフォルトでは、変換は 1 桁目で始まります。

最初の 3 つのパラメーターのタイプは、文字、グラフィック、または UCS-2 とすることができます。3 つのすべてが同じタイプでなければなりません。戻される値は *string* と同じタイプおよび長さです。

4 番目のパラメーターは、小数点以下の桁数がゼロの非浮動数値でなければなりません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D up              C                   'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
D lo              C                   'abcdefghijklmnopqrstuvwxyz'
D string          S             10A   inz('rpg dept')

 /FREE

   string = %XLATE(lo:up:'rpg dept');
   // string には現在 'RPG DEPT' が含まれています

   string = %XLATE(up:lo:'rpg dept':6);
   // string には現在 'RPG Dept' が含まれています
 /END-FREE
```

図 187. %XLATE の例

%YEARS (年数)

```
%YEARS(数値)
```

%YEARS は、数値を日付または時刻スタンプ値に追加できる期間に変換します。

加算または減算演算命令では、%YEARS は右側の値にのみすることができます。左側の値は日付または時刻スタンプでなければなりません。結果は、該当する年数が加算または減算された日付または時刻スタンプ値となります。日付の場合の結果の値は *ISO 形式です。

左側の値が 2 月 29 日であり、結果の年がうるう年でない場合には、代わりに 2 月 28 日が使用されます。2 月 29 日に年数を加算または減算すると、逆戻りできないことがあります。たとえば、2000-02-29 + %YEARS(1) - %YEARS(1) は 2000-02-28 になります。

%YEARS 組み込み関数については、 444 ページの図 158を参照してください。

第 26 章 命令コードの詳細

以下の項は、個々の命令コードを詳細に説明しています。

ADD (加算)

フリー・フォーム構文	(使用できません。+ 演算子を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ADD (H)	加数	加数	和	+	–	Z

演算項目 1 を指定した場合には、ADD 演算命令はそれを演算項目 2 に加算し、和を結果フィールドに入れます。演算項目 1 を指定しない場合には、演算項目 2 の内容が結果フィールドに加算され、その和が結果フィールドに入れられます。

演算項目 1 および演算項目 2 は数字でなければならず、配列、配列エレメント、固定情報、フィールド名、リテラル、サブフィールド、またはテーブル名のいずれかを入れることができます。

352 ページの『算術演算』には、算術演算を指定するための一般規則が説明されています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 値 1 が RECNO に加算されます。
C                   ADD       1             RECNO
C* EHWRK の内容が CURHRS に加算されます。
C                   ADD       EHWRK         CURHRS
C* OVRTM と REGHRS の内容が加算され、
C* TOTPAY に入れられます。
C     OVRTM         ADD       REGHRS        TOTPAY
```

図 188. ADD 演算命令

ADDDUR (期間の加算)

フリー・フォーム構文	(使用できません。%YEARS や %MONTHS などの期間関数とともに + 演算子を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ADDDUR (E)	日付 / 時刻	期間:期間コード	日付 / 時刻	_	ER	_

ADDDUR 演算命令は、演算項目 2 に指定された期間を日付または時刻に加算し、結果の日付、時刻、またはタイム・スタンプを結果フィールドに入れます。

演算項目 1 を指定する場合には、日付、時刻、またはタイム・スタンプ・フィールド、サブフィールド、配列、配列エレメント、リテラル、あるいは固定情報が含まれていなければなりません。

演算項目 1 にはフィールド名、配列、または配列エレメントが含まれ、この場合に、そのデータ・タイプは結果フィールドに指定されるフィールドと同じデータ・タイプでなければなりません。演算項目 1 を指定しない場合には、期間が結果フィールドに指定されたフィールドに加算されます。

演算項目 2 には、2 つの副演算項目が含まれていなければなりません。最初の副演算項目は期間で、小数点以下の桁数がゼロの数値フィールド、配列エレメント、または固定情報であることが必要です。期間が負の場合には、それが日付から減算されます。2 番目の副演算項目は、期間のタイプを指示する有効な期間コードであることが必要です。期間コードは、結果フィールドのデータ・タイプと整合性がなければなりません。年、月、または日を日付フィールドに加算することができます。分の期間を日付フィールドに加算することはできません。 363 ページの『日付の演算』は、期間コードについて説明しています。

結果フィールドは、日付、時刻、またはタイム・スタンプ・データ・タイプ・フィールド、配列、あるいは配列エレメントでなければなりません。演算項目 1 がブランクの場合には、期間が結果フィールドの値に加算されます。結果フィールドが配列である場合は、演算項目 2 の値がその配列の各エレメントに加算されます。結果フィールドが時刻フィールドである場合には、結果は常に有効な時刻となります。たとえば、59 分を 23:59:59 に加算すると、24:58:59 になります。この時刻は有効でないので、コンパイラーがそれを 00:59:59 に調整します。

月による期間を日付に追加する場合の一般規則では、月部分が期間中の月数だけ増やされ、日部分は変更されません。この例外は、結果の日の部分が結果の月の実際の日数を超える場合です。その場合には、結果の日の部分を調整して、実際の月の終わりの日付に合わせます。 次の例 (*YMD 形式とします) は、この点を示しています。

```
'98/05/30' ADDDUR 1:*MONTH の結果は '98/06/30'
```

結果の月の部分を 1 だけ増やして、日の部分は変更なしです。

```
'98/05/31' ADDDUR 1:*MONTH の結果は '98/06/30'
```

結果の月部分は 1 だけ増やされ、6 月は 30 日だけであるので、結果の日部分が調整されています。

年の期間を加算する場合も、同様の結果となります。たとえば、'92/02/29' に 1 年を加えると、結果の年は閏年ではないので '93/02/28' (調整した値) になります。

詳細については、 370 ページの『メモリー管理命令』を参照してください。

次のいずれかが起こった場合には、エラー状態が生じます。
- 演算項目 1 の日付、時刻、またはタイム・スタンプの値が無効である
- 演算項目 1 がブランクであり、演算命令の前に結果フィールドの値が無効である
- オーバーフローまたはアンダーフローが起こった (すなわち、結果の値が *HIVAL より大きいかまたは *LOVAL より小さい)

エラー状態では、次のようになります。
- エラー (状況コード 112 または 113) の信号が出されます。
- エラー標識 (73 ～ 74 桁目) は指定されている場合はオンに設定され、%ERROR 組み込み関数は「E」拡張が指定されている場合は「1」を戻すように設定されます。
- 結果フィールドの値は変更されないままです。

プログラム状況コードが 112 または 113 である例外を処理するには、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することできますが、両方は指定できません。エラー処理の詳細については、 52 ページの『プログラム例外およびエラー』を参照してください。

システムでは、期間は 15 桁に制限されます。15 桁より多い有効数字の期間を加算すると、エラーまたは切り捨ての原因となります。これは、演算項目 2 の最初の副演算項目を 15 桁に制限することで避けることができます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H TIMFMT(*USA) DATFMT(*MDY&)
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D*
DDateconst        C                   CONST(D'12 31 92')
D*
D* 日付フィールドを定義し、次の初期化をします。
D*
DLoandate         S               D   DATFMT(*EUR) INZ(D'12 31 92')
DDuedate          S               D   DATFMT(*ISO)
Dtimestamp        S               Z
Danswer           S               T

CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....

C* LOANDATE より xx 年、yy 月、zz 日後の DUEDATE を
C* 判別します。
C     LOANDATE      ADDDUR    XX:*YEARS     DUEDATE
C                   ADDDUR    YY:*MONTHS    DUEDATE
C                   ADDDUR    ZZ:*DAYS      DUEDATE
C* 23 日後の日付を判別します。
C*
C                   ADDDUR    23:*D         DUEDATE

C* 1234 マイクロ秒をタイム・スタンプに加算します。
C*
C                   ADDDUR    1234:*MS      timestamp

C* 12 時間 16 分を真夜中に加算します。
C*
C     T'00:00 am'   ADDDUR    12:*Hours     answer
C                   ADDDUR    16:*Minutes   answer

C* ローン期限から 30 日を減算します。
C*
C                   ADDDUR    -30:*D        LOANDUE
```

図 189. ADDDUR 演算命令

ALLOC (ストレージの割り振り)

フリー・フォーム構文	(使用できません。%ALLOC 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ALLOC (E)		長さ	ポインター	_	ER	_

ALLOC 演算命令は、演算項目 2 に指定された長さのデフォルト・ヒープのストレージを割り振ります。結果フィールドのポインターは、新規ヒープ・ストレージを指し示すように設定されます。このストレージは初期化されていません。

演算項目 2 は、小数点以下の桁数がゼロの数字でなければなりません。これは、リテラル、固定情報、独立型フィールド、サブフィールド、テーブル名、または配列エレメントとすることができます。この値は、1 ～ 16776704 の範囲内でなければなりません。値が実行時にこの範囲外であった場合には、状況 00425 のエラーが起こります。ストレージを割り振ることができなかった場合には、状況 426 のエラーが起こります。これらのエラーが起こった場合には、結果フィールドのポインターは変更されないままです。

結果フィールドは、基底ポインター・スカラー変数 (独立型フィールド、データ構造サブフィールド、テーブル名、または配列エレメント) でなければなりません。

プログラム状況コード 425 または 426 での例外を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

詳細については、370 ページの『メモリー管理命令』を参照してください。

```
D Ptr1            S               *
D Ptr2            S               *

C                   ALLOC     7                 Ptr1
 * これで、Ptr1 は 7 バイトのストレージを指し示します

C                   ALLOC (E) 12345678          Ptr2

 * これは大量のストレージであるので、使用不可能になることが
 * あります。ストレージを割り振ることができなかった場合には、
 * %ERROR は '1' を戻し、状況は 00426 に設定されます。また、
 * %STATUS は 00426 を戻してきます。
```

図 190. ALLOC 演算命令

ANDxx (AND)

フリー・フォーム構文	(使用できません。AND 演算子を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ANDxx	被比較数	被比較数				

ANDxx 演算命令は、次の 1 つの演算命令の直後になければなりません。
- ANDxx
- DOUxx
- DOWxx
- IFxx
- ORxx
- WHENxx

ANDxx を使用すれば、DOUxx、DOWxx、IFxx、および WHENxx 演算命令について複合条件を指定することができます。ANDxx 演算命令は、ORxx 演算命令より優先します。例については、 487 ページの図 192 を参照してください。

演算項目 1 と演算項目 2 には、リテラル、名前付き固定情報、表意定数、テーブル名、配列エレメント、データ構造名、またはフィールド名が入っていなければなりません。演算項目 1 と演算項目 2 のタイプは同じでなければなりません。たとえば、文字フィールドを数字と比較することはできません。演算項目 1 と演算項目 2 の比較は、比較演算命令に示されているものと同じ規則に従います。

361 ページの『比較命令』および 379 ページの『構造化プログラミング命令』は、ANDxx 演算命令を指定する場合の規則を説明しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* ACODE が A と等しく、標識 50 がオンであれば、
C* MOVE および WRITE 演算命令が処理されます。
C     ACODE         IFEQ      'A'
C     *IN50         ANDEQ     *ON
C                   MOVE      'A'           ACREC
C                   WRITE     RCRSN
C* 前の条件は満たされていないが、ACODE が A と等しく、
C* 標識 50 がオフで、ACREC が D と等しければ、次の
C* MOVE 演算命令が処理されます。
C                   ELSE
C     ACODE         IFEQ      'A'
C     *IN50         ANDEQ     *OFF
C     ACREC         ANDEQ     'D'
C                   MOVE      'A'           ACREC
C                   ENDIF
C                   ENDIF
```

図 191. ANDxx 演算命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の例では、最初の 2 つの条件が「真」であるか、または 3 番目の
C* 条件が「真」である場合にのみ標識 25 がオンに設定されます。
C*
C* これは、式として次のように書かれます:
C* EVAL *IN25 = ((FIELDA > FIELDB) AND (FIELDA >= FIELDC)) OR (FIELDA < FIELDD)
C*
C*
C     FIELDA        IFGT      FIELDB
C     FIELDA        ANDGE     FIELDC
C     FIELDA        ORLT      FIELDD
C                   SETON                                        25
C                   ELSE
C                   SETOFF                                       25
C                   ENDIF
```

図 192. AND/OR プロシージャーの例

BEGACT (アクション・サブルーチンの開始)

フリー・フォーム構文	BEGACT *action-subroutine-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
BEGACT	パーツ名	イベント名	ウィンドウ名			

BEGACT 演算命令は、アクション・サブルーチンの開始を定義します。パーツに対するイベントが起こった時に、アクション・サブルーチンが呼び出されます。

GUI Designer では、パーツのイベントを既存のアクション・サブルーチンにリンクしたり、定義されていない場合にイベントの新規アクション・サブルーチンを作成したりすることができます。後者の場合、関連するウィンドウ、パーツ、およびイベント名から作成された名前を持つスケルトン・アクション・サブルーチンがソース・コードに挿入されます。

スケルトン・アクション・サブルーチンの追加場所がフリー・フォーム・セクション内であるとき (つまり、演算仕様の後で、出力またはプロシージャー・コンパイル時データ仕様の前に '/END-FREE' ディレクティブがあるとき) そのサブルーチンはフリー・フォーム・スタイルでプログラム・ソース・コードに追加されます。

従来からの構文のアクション・サブルーチン名

GUI Designer が従来からの構文演算セクションに新規アクション・サブルーチンを挿入するとき、BEGACT 演算の演算項目 1 にパーツ名、演算項目 2 にイベント名、および結果フィールドにウィンドウ名 (パーツ名を含む) が指定されます。アクション・サブルーチン名はこれらの値から派生し、パーツ・イベントにリンクされています。

アクション・サブルーチン名は、演算項目 1、演算項目 2、および結果フィールドを使用してビルドされます。個々の項目はプラス (+) 文字で区切られます。次の表は、GUI Designer を使用して作成されるリンクの例を示しています。

表 52. 単一リンクおよび複数リンクのアクション・サブルーチン

ウィンドウ	パーツ	イベント	アクション・サブルーチン
INVENTORY	PSB0001	PRESS	PSB0001+PRESS+INVENTORY
INVENTORY	PSB0004	PRESS	SETCOLORS
INVENTORY	PSB0005	PRESS	PSB0005+PRESS+INVENTORY
ADDPART	PSB0008	PRESS	SETCOLORS
INVENTORY	PSB0002	PRESS	PSB0002++INVENTORY
INVENTORY	PSB0002	MOUSEMOVE	PSB0002++INVENTORY
ADDPART	PSB0009	MOUSEMOVE	PSB0009+MOUSEMOVE

次の例は、表 52 で説明されている情報を使用してアクション・サブルーチン名がどのようにビルドされるかを示しています。

演算項目 1 および演算項目 2 を使用したアクション・サブルーチン名

演算項目 1 に PSB0009 が含まれ、演算項目 2 に MOUSEMOVE が含まれ、結果フィールドには項目が含まれていない場合は、アクション・サブルーチン名は PSB0009+MOUSEMOVE になります。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++
C     PSB0009       BEGACT    MOUSEMOVE
```

図 193. アクション・サブルーチン名 - 演算項目 1 および演算項目 2

演算項目 1 および結果フィールドを使用したアクション・サブルーチン名

演算項目 1 に PSB0002 が含まれ、演算項目 2 には項目が含まれず、結果フィールドに INVENTORY が含まれている場合には、アクション・サブルーチン名は PSB0002++INVENTORY となります。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++
C     PSB0002       BEGACT                  INVENTORY
```

図 194. アクション・サブルーチン名 - 演算項目 1 および結果フィールド

演算項目 1、演算項目 2、および結果フィールドを使用したアクション・サブルーチン名

演算項目 1 に PSB0001 が含まれ、演算項目 2 に PRESS が含まれ、結果フィールドに INVENTORYが含まれている場合には、アクション・サブルーチン名は PSB0001+PRESS+INVENTORY となります。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++
C     PSB0001       BEGACT    PRESS         INVENTORY
C     PSB0005       BEGACT    PRESS         INVENTORY
```

図 195. アクション・サブルーチン名 - 演算項目 1、演算項目 2、および結果フィールド

演算項目 1 を使用したアクション・サブルーチン名

演算項目 1 に SETCOLORS が含まれ、演算項目 2 と結果フィールドの両方に項目が含まれていない場合には、アクション・サブルーチン名は SETCOLORS となります。この名前は、このアクション・サブルーチン SETCOLORS にリンクされる 1

つまたは複数のウィンドウ、パーツ、およびイベントに関する情報を検索するために使用されます。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++
C     SETCOLORS     BEGACT
```

図 196. アクション・サブルーチン名 - 演算項目 1

フリー・フォーム構文のアクション・サブルーチン名

フリー・フォーム・セクションに挿入されるアクション・サブルーチンの名前は、ウィンドウ、パーツ、イベント名を下線で区切って結合することによって生成される値です。

フリー・フォーム構文のアクション・サブルーチンの例:

```
/free

 //*********************************************************************
 //
 // Window . . :  DATEDIALOG
 // Part . . . :  PSBOK
 // Event  . . :  CREATE
 //
 // Description:
 //
 //*********************************************************************

 BEGACT DATEDIALOG_PSBOK_CREATE;

 ENDACT;
/end-free
*****END OF SOURCE***
```

単一リンクおよび複数リンクのアクション・サブルーチン

1 つのウィンドウ / パーツ / イベントの組み合わせにのみリンクされるアクション・サブルーチンは、単一リンクのアクション・サブルーチンと呼ばれます。

複数のウィンドウ / パーツ / イベントの組み合わせにリンクされるアクション・サブルーチンは、複数リンクのアクション・サブルーチンと呼ばれます。

注: ユーザー・サブルーチンは、すべてが複数リンクのアクション・サブルーチンと見なされます。実行時には、ユーザー・サブルーチンのためのデフォルト・ウィンドウまたはイベントが、そのユーザー・サブルーチンを直接にかまたは他のユーザー・サブルーチンを通じて呼び出すアクション・サブルーチンのデフォルト・ウィンドウまたはイベントとなります。

488 ページの表 52 は、単一リンクおよび複数リンクのアクション・サブルーチンを例示しています。たとえば、項目 1、3、および 7 は単一リンクのアクション・サブルーチンです。項目 2 と 4 および項目 5と 6 は複数リンクのアクション・サブルーチンです。

アクション・サブルーチンを処理する場合は、次のガイドラインを使用してください。

- 重複したアクション・サブルーチン名は許されません。ユーザー・プログラムには、重複したアクション・サブルーチン名を含めることはできません。演算項目 1 が BEGACT 演算命令に対する唯一の項目である場合には、ユーザー・プログラム内のどのフィールド名、ユーザー・サブルーチン名、または他の構成における名前とも同じにすることはできません。
- 関連したイベントのないアクション・サブルーチンが実行されることはありません。これは、GUI Designer を使用してアクション・リンクを除去した場合に起こる可能性があります。

アクション・サブルーチンを作成し、各アクション・サブルーチンを少なくとも 1 つのウィンドウ /パーツ / イベントの組み合わせにリンクさせるには、GUI Designer を使用します。アクション・サブルーチンがコンパイルされる時に、コンパイラーが、GUI Designer を使用して作成されたリンクを参照します。 GUI設計機能によって作成されたアクション・サブルーチン名を使用するか、あるいはユーザーの固有の名前でそのアクション・サブルーチン名を置き換えることができます。アクション・サブルーチンを作成してリンクさせるための GUI Designer の詳細については、*WebSphere Development Studio Client for iSeries ご使用に際して* を参照してください。

BEGSR (ユーザー・サブルーチンの開始)

フリー・フォーム構文	BEGSR *subroutine-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
BEGSR	サブルーチン名					

BEGSR 演算命令は、ユーザー・サブルーチンの開始を識別します。

Subroutine-name には、固有の記号名を指定するか、キーワード *TERMSR、*PSSR、または *INZSRのうち 1 つを指定しなければなりません。名前を指定した場合には、そのサブルーチンを参照する EXSR 演算命令に同じ名前を指定するか、そのサブルーチンを参照する CASxx 演算命令の結果フィールドに同じ名前を指定する必要があります。

キーワードを指定する場合には、次のキーワードによって 1 つのサブルーチンのみを指定することができます。

- *TERMSR は、正常終了時に実行されるサブルーチンを指定します。
- *PSSR は、これがプログラムで検出された例外 / エラーを処理するプログラム例外 /エラー・サブルーチンであることを指定します。
- *INZSR は、初期化時に実行されるサブルーチンを指定します。

558 ページの『EXSR (ユーザー・サブルーチンの起動)』 は、サブルーチンを起動する方法を説明しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                             Extended-factor2++++++++++++++++++++++++++++++
C*
C     *TERMSR       BEGSR
C                     .
C                     .
C                     .
C                   ENDSR
```

図 197. ユーザー・サブルーチンの開始演算命令

注: サブルーチン内のパーツを参照する場合には、次の点を考慮してください。すべてのユーザー・サブルーチンは、複数リンクのアクション・サブルーチンと見なされます。実行時には、ユーザー・サブルーチンのためのデフォルト・ウィンドウまたはイベントが、そのユーザー・サブルーチンを直接にかまたは他のユーザー・サブルーチンを通じて呼び出すアクション・サブルーチンのデフォルト・ウィンドウまたはイベントとなります。

BITOFF (ビットのオフ設定)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
BITOFF		ビット番号	文字フィールド			

BITOFF 演算命令によって、演算項目 2 で識別されたビットが結果フィールドでオフに設定 (0 に設定) されます。演算項目 2 で識別されないビットは、変更されないままです。BITOFF を文字の形式設定に使用する時は、BITON と BITOFF の両方を使用する必要があります。 BITON はビットをオンに設定 (1 に設定) することを指定し、BITOFF はビットをオフに設定 (0 に設定) することを指定します。文字中のすべてのビットを明示的にオンまたはオフに設定しなければ、必要な文字は得られないことがあります。

演算項目 2 には、次を入れることができます。

- *ビット番号 0 ～ 7:* 1 ～ 8 ビットを演算命令ごとにオフに設定することができます。それらは、ビット番号 0 ～ 7 によって識別されます (0 が左端ビットです)。 ビット番号はアポストロフィで囲みます。たとえば、ビット 0、2、および 5 をオフに設定する場合は、 '025' を演算項目 2 に入力します。
- *フィールド名:* 1 桁の文字フィールド、テーブル・エレメント、または配列エレメントを演算項目 2 に指定します。フィールド、テーブル・エレメント、または配列エレメントの中でオンになっているビットは、結果フィールドではオフに設定されます。オフになっているビットは、結果に影響しません。
- *16 進数リテラルまたは名前付き固定情報:* 1 バイトの 16 進数リテラルまたは 16 進数名前付き固定情報を指定します。演算項目 2 でオンになっているビットは結果フィールドではオフに設定されます。オフになっているビットに影響はありません。
- *名前付き固定情報:* オフに設定されるビット番号を含む最高 8桁までの長さの文字の名前付き固定情報を指定します。

結果フィールドには、1 桁の文字フィールドを指定します。配列内の各エレメントが 1 桁の文字フィールドである場合には、これを配列エレメントとすることができます。

BITOFF および BITON 演算命令の例については、 495 ページの図 198 を参照してください。

特定のビット・パターンを文字フィールドに割り当てたい場合には、演算項目 2 に 16 進数リテラルが含まれる MOVE 演算命令を使用してください。

BITON (ビットのオン設定)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
BITON		ビット番号	文字フィールド			

BITON 演算命令によって、演算項目 2 で識別されたビットが結果フィールドでオンに設定 (1 に設定)されます。演算項目 2 で識別されないビットは、変更されないままです。BITON を文字の形式設定に使用する場合には、BITON と BITOFF の両方を使用する必要があります: BITONはビットをオンに設定 (1 に設定) することを指定し、BITOFF はビットをオフに設定 (0 に設定) することを指定します。文字中のすべてのビットを明示的にオンまたはオフに設定しなければ、必要な文字は得られないことがあります。

演算項目 2 には、次を入れることができます。

- *ビット番号 0 ～ 7:* 1 ～ 8 ビットを演算命令ごとにオンに設定することができます。それらは、ビット番号 0 ～ 7 によって識別されます (0 が左端ビットです)。 ビット番号はアポストロフィで囲みます。たとえば、ビット 0、2、および 5 をオンに設定する場合には、'025'を演算項目 2 に入力します。
- *フィールド名:* 1 桁の文字フィールド、テーブル・エレメント、または配列エレメントを演算項目 2 に指定することができます。フィールド、テーブル・エレメント、または配列エレメントの中でオンになっているビットは、結果フィールドではオンに設定されます。オフになっているビットに影響はありません。
- *16 進数リテラルまたは名前付き固定情報:* 1 バイトの 16 進数リテラルを指定することができます。演算項目 2 でオンになっているビットは、結果フィールドではオンに設定されます。オフになっているビットは、結果に影響しません。
- *名前付き固定情報:* オンに設定されるビット番号を含む最高 8桁までの長さの文字の名前付き固定情報を指定することができます。

結果フィールドには、1 桁の文字フィールドを指定します。配列内の各エレメントが 1 桁の文字フィールドである場合には、これを配列エレメントとすることができます。

BITOFF および BITON 演算命令の例については、 495 ページの図 198 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D BITNC           C                   '01234567'
D HEXNC           C                   X'0F'
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  ビットの設定値は次の通りです。
C*          演算命令の前:  演算命令の後:

C*          FieldA = 00000000    FieldA = 10001111
C*          FieldB = 00000000    FieldB = 00010000
C*          FieldC = 11111111    FieldC = 11111111
C*          FieldD = 11000000    FieldD = 11010000
C*          FieldE = 11000000    FieldE = 11000001
C*          FieldG = 11111111    FieldG = 01111111
C*          FieldH = 00000000    FieldH = 00001110
C*          FieldI = 11001010    FieldI = 00001111
C*
C                   BITON     '04567'       FieldA
C                   BITON     '3'           FieldB
C                   BITON     '3'           FieldC
C                   BITON     '3'           FieldD
C                   BITON     '01'          FieldH
C                   BITOFF    '0'           FieldG
C                   BITOFF    BITNC         FieldI
C                   BITON     HEXNC         FieldI
```

図 198. BITON および BITOFF 演算命令

特定のビット・パターンを文字フィールドに割り当てたい場合には、演算項目 2 に 16 進数リテラルが含まれる MOVE 演算命令を使用してください。

CABxx (比較および分岐)

フリー・フォーム構文	(使用できません。LEAVE、ITER、RETURN などの他の命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CABxx	被比較数	被比較数	Label	HI	LO	EQ

CABxx 演算命令は、演算項目 1 を演算項目 2 と比較します。xx によって指定された条件が「真」であった場合に、プログラムは、結果フィールドに指定されたラベルと関連した TAG または ENDSR 演算命令に分岐します。それ以外の場合には、プログラムは順序内の次の演算命令から続行されます。結果フィールドが指定されない場合には、それに応じて演算結果標識が設定され、プログラムは順序内の次の演算命令から続行されます。

361 ページの『比較命令』は、xx の種々の値について説明しています。

演算項目 1 と演算項目 2 には、リテラル、名前付き固定情報、表意定数、テーブル名、配列エレメント、データ構造名、またはフィールド名が入っていなければなりません。演算項目 1 と演算項目 2 のタイプは同じでなければなりません。

メイン・プロシージャー内の CABxx 演算命令では、前または後続の仕様行への分岐を指定することができます。サブプロシージャー内の CABxx 演算命令では、次のような分岐を指定することができます。

- サブプロシージャーの本体の 1 つの行からそのサブプロシージャーの本体の別の行へ
- サブルーチンの 1 つの行から同じサブルーチンの別の行へ
- サブルーチンの 1 つの行からサブプロシージャーの本体の 1 つの行へ

CABxx 演算命令では、サブルーチンの外側からそのサブルーチンの中の TAG または ENDSR 演算命令への分岐を指定することはできません。結果フィールドに指定されるラベルは、固有の TAG 演算命令と関連づけられた固有の記号名であることが必要です。

演算結果標識は任意指定です。指定した場合には、比較演算命令の結果を反映するようにそれらが設定されます。たとえば:

- 演算項目 1 が演算項目 2 より大きい場合は HI が設定されます。
- 演算項目 1 が演算項目 2 より小さい場合は LO が設定されます。
- 演算項目 1 と演算項目 2 が等しい場合は EQ が設定されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*          フィールドの値は次の通りです:
C*          FieldA = 100.00
C*          FieldB = 105.00
C*          FieldC = ABC
C*          FieldD = ABCDE
C*
C*           TAGX に分岐します。
C     FieldA        CABLT     FieldB        TAGX
C*
C*           TAGX に分岐します。
C     FieldA        CABLE     FieldB        TAGX
C*
C*           TAGX に分岐し、標識 16 はオフになります。
C     FieldA        CABLE     FieldB        TAGX                      16
C*
C*           TAGX に分岐し、標識 17 はオフになり、標識 18 はオンになります。
C     FieldA        CAB       FieldB        TAGX                  1718
C*
C*           TAGX に分岐し、標識 19 はオンになります。
C     FieldA        CAB       FieldA        TAGX                      19
C*
C*           分岐は起こりません。
C     FieldA        CABEQ     FieldB        TAGX
C*
C*           分岐は起こらず、標識 20 はオンになります。
C     FieldA        CABEQ     FieldB        TAGX                    20
C*
C*           分岐は起こらず、標識 21 はオフになります。
C     FieldC        CABEQ     FieldD        TAGX                21
C                   :
C     TAGX          TAG
```

図 199. CABxx 演算命令

CALL (AS/400 プログラムの呼び出し)

フリー・フォーム構文	(使用できません。CALLP 命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CALL (E)		プログラム名	PLIST 名	_	ER	_

CALL 演算命令は、演算項目 2 に指定されたプログラム名により表される AS/400 プログラムに制御を渡します。

演算項目 2 は、呼び出されるプログラムの名前を定義する定義仕様の名前であることが必要です。プログラム名は、OS/400 名 (任意選択でライブラリーにより修飾したもの) または「サーバー情報の定義」メニュー項目を使用して定義された一時変更名のいずれかにすることができます。GUI Designer を使用したサーバー情報の定義に関する詳細については、*VisualAge for RPG プログラミング* を参照してください。

結果フィールドを指定する場合には、呼び出し元プログラムと呼び出し先プログラムの間で値をやりとりするための PLIST の名前がそこに含まれている必要があります。呼び出し先プログラムがパラメーターにアクセスしない場合、あるいは CALL 演算命令の直後に PARM ステートメントが続いている場合には、結果フィールドをブランクとすることができます。

OS/400 プログラムに対する CALL と関連したパラメーターには、次の制限があります。

- パラメーターにポインターを含めることはできません。パラメーターにポインターが含まれていた場合には、コンパイル時にコンパイラーがエラー・メッセージを生成します。
- データ構造を文字以外のフィールドにオーバーラップさせることはできません。オーバーラップするフィールドは双方とも文字でなければなりません。
- 値 *HIVAL (X'FF') を文字またはグラフィック・パラメーターとして渡すと、予測のできない結果の原因となることがあります。
- リモート呼び出しが EBCDIC に変換できない文字フィールドの中で渡されるプログラムの場合には、変換が停止される原因となります。通常、これは数値フィールドが文字フィールドをオーバーレイした場合に起こる可能性があります。
- 最大 25 個のパラメーターを指定することができます。
- パラメーターに割り振られた合計バイト数が 32K を超えることはできません。

演算結果標識が 73 - 74 桁目に指定された場合には、CALL 演算命令の実行中にエラーが起こった時にオンに設定されます。

CALL の例外 (プログラム状況コード 202、211、または 231) を処理するには、命令コード拡張 'E'かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Functions+++++++++++++++++++++++++
D                                   Functions-cont++++++++++++++++++++
D
D*名前付き固定情報
D Remote1         C                 CONST('PROG1')
D                                   LINKAGE(*SERVER) NOWAIT
D*
D*独立型フィールド
D Remote2         S             13A   INZ('MYLIB/REMPROG')
D                                   LINKAGE(*SERVER)
D parm1           S              8P 2
D parm2           DS
D name                    1     20A
D   first                 1      8A
D   last                  9     20A
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq
C* リモート・プログラムに対する CALL
C*
C                   CALL      Remote1                               90
C                   PARM                    parm1
C                   PARM                    parm2
C*
C* リモート呼び出し
C*
C                   CALL      Remote2       PLIST1                  90
C*
C*
C*
C     PLIST1        PLIST
C                   PARM                    Fld1            10 2
C                   PARM                    Charfld         50
```

図 200. CALL 演算命令

ワークステーション・ファイルを使用する OS/400 プログラムの呼び出し

ワークステーション・ファイルを使用する OS/400 プログラムを呼び出す VisualAge RPG プログラムを使用するには、次のようにしてください。

- NOWAIT キーワードを定義仕様に指定します。
- OS/400 ワークステーション・ファイルをサーバー上に作成する時に、CRTDSPF コマンドに次を指定します。
  - 表示装置の値: 表示装置ファイルを表示するセッションの名前。
  - 装置の最大数: 1 より大きい任意の値。
- リモート AS/400 プログラムでは、表示装置を獲得するのに ACQ 演算命令は**使用しないようにしてください**。これを実行すると、結果としてエラーとなる競合の原因となります。

注: このメソッドを使用している場合には、パラメーターをリモート・プログラムに渡すことができます。ただし、リモート・プログラムがパラメーターを戻すことはできません。

表示装置ファイルを使用するホスト・プログラムの呼び出し

VARPG コンパイラーが表示装置ファイルを使用する OS/400 ホスト・プログラムを呼び出す時には、使用できる有効なセッション装置の判別が必要です。ホスト・プログラムで使用できる有効なセッション装置を判別するには、AS/400 ホスト上で CL プログラムを使用し、有効なセッションを見つけることができます。

次の例は、このような CL プログラムを示しています。そこでは、ユーザーが 5250 との SNA プロトコルを使用しているか、あるいはクライアント・アクセスでグラフィカル・アクセス・エミュレーションが実行されていることが前提とされます。

```
PGM  PARM(&SESS)
/*-----------------------------------------------------------*/
/*                                                           */
/* 作業変数の宣言                                            */
/*                                                           */
/*-----------------------------------------------------------*/
 DCL  VAR(&JOBN) TYPE(*CHAR) LEN(10)
 DCL  VAR(&SESS) TYPE(*CHAR) LEN(10)
 DCL  VAR(&SUB)  TYPE(*CHAR) LEN(2)
 DCL  VAR(&STS)  TYPE(*DEC) LEN(5 0)
 DCL  &ITLEN     TYPE(*DEC)  VALUE(2)
 DCL  &ITPTR     TYPE(*DEC)  LEN(5 0)
 DCL  VAR(&SUBFIX) TYPE(*CHAR) LEN(40) +
      VALUE('A B C D E F G H I J G0G1G2G3G4G5G6G7G8G9')

 RTVJOBA    JOB(&JOBN)
/*-----------------------------------------------------------*/
/* 可能な装置名全体を通じてのループおよびオンになって    */
/* いる 1 つのサインオン表示装置があるかどうかの検査     */
/*-----------------------------------------------------------*/
 CHGVAR    &ITPTR 1

LOOP1:
      IF (&ITPTR *GT 40) THEN(DO)
         CHGVAR &SESS VALUE('INVALID   ')
         GOTO END
      ENDDO

      CHGVAR     VAR(&SUB)  VALUE(%SST(&SUBFIX &ITPTR &ITLEN))
      CHGVAR     VAR(&SESS) VALUE(&JOBN *TCAT &SUB)
      RTVCFGSTS  CFGD(&SESS) CFGTYPE(*DEV) STSCDE(&STS)
      MONMSG     MSGID(CPF9801)
      IF  (&STS = 50) THEN(GOTO END)
      CHGVAR    &ITPTR (&ITPTR + &ITLEN)
      GOTO      LOOP1
END:  ENDPGM
```

CL コマンドの呼び出し

VisualAge RPG プログラムで CL コマンドを呼び出す場合は、次のようにしてください。

- CL コマンドで OS/400 ファイルに対するコマンドを出す場合は、QCMDDDM に対する CALL を指定します。
- CL コマンドで OS/400 プログラムまたはデータ域、あるいはその両方に対するコマンドを出す場合は、QCMDEXC に対する CALL を指定します。

CALLB (機能の呼び出し)

フリー・フォーム構文	(使用できません。CALLP 命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CALLB (DE)		プロシージャー名またはプロシージャー・ポインター	PLIST 名	_	ER	_

CALLB は、Windows 機能を呼び出すために使用します。これらの機能は、VisualAge RPGアプリケーションのコンパイル時にそのアプリケーションにリンクされたダイナミック・リンク・ライブラリー (DLL) からエクスポート済みの名前です。Windows C 機能を呼び出すアプリケーションのコンパイル方法については、*WebSphere Development Studio Client for iSeries ご使用に際して* を参照してください。

演算項目 2 には、プロシージャー名、または呼び出す機能のアドレスが入っているプロシージャー・ポインターが含まれていることが必要です。

プロシージャー名では大文字・小文字が区別されます。これは、演算項目 2 に入力された名前が呼び出される機能の大文字・小文字と一致していなければならないことを意味します。プロシージャー名は 255文字以下でなければなりません。255 より長い名前が入力された場合には、255 までに切り捨てられます。

演算項目 2 にプロシージャー・ポインターが含まれる場合には、PSDS の *ROUTINE は消去されて、ブランクが埋め込まれます。演算項目 2 にリテラルまたは名前付き固定情報が含まれる場合には、 PSDSの *ROUTINE にプロシージャー名の最初の 8 文字が入れられます。

結果フィールドを指定する場合には、PLIST 名が入っていなければなりません。

演算結果標識が 73-74 桁目に指定された場合には、CALLB 演算命令の実行中にエラーが起こった時にオンに設定されます。

CALLB の例外 (プログラム状況コード 202、211、または 231) を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

呼び出し先機能では、リンケージ規約 __cdecl が使用されている必要があります。

注: VisualAge RPG コンパイラーは、このリンケージ規約を VARPG サブプロシージャーに使用します。

CALLB 演算命令の使用法の例については、*VisualAge for RPG プログラミング* を参照してください。

CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)

フリー・フォーム構文	{CALLP{(EMR)}} *name*({*parm1*{:*parm2*...}})

コード	演算項目 1	演算項目 2
CALLP (M/R)		name{ (Parm1 {:Parm2...}) }

CALLP 演算命令は、ワークステーション上のプロトタイプ・プロシージャーまたはローカル・プログラムを呼び出すために使用されます。これは、お勧めしているプログラム呼び出しの方法です。

CALLP では、EXE、BAT、COM、または DOS EXE を呼び出すことができます。UCS-2 パラメーターは使用できません。

他の呼び出し演算命令と異なり、CALLP にはフリー・フォーム構文が使用されます。*name* オペランドを使用して、渡したいパラメーターだけでなく、呼び出し先のプログラムまたはプロシージャーのプロトタイプの名前を指定します。(これは、組み込み関数の呼び出しと類似しています。) プログラム呼び出しの場合は最大 255 個のパラメーターが許され、プロシージャー呼び出しの場合は最大 399 個のパラメーターが許されています。

フリー・フォーム演算仕様では、拡張が必要なければ、命令コード名を除外できます。

コンパイラーは、次に、このプロトタイプ名を使用して、必要な場合に呼び出しの外部名を入手します。プロトタイプにキーワード CLTPGM が指定されていれば、呼び出しは動的外部呼び出しとなり、それ以外の場合には、プロシージャー呼び出しとなります。

呼び出し先のプログラムまたはプロシージャーのプロトタイプは、定義仕様の中で CALLP の前に組み込まれていることが必要です。

値を戻すプロシージャーを呼び出すために CALLP を使用した場合には、呼び出し元ではその値を使用できなくなることに注意してください。値が必要な場合には、プロトタイプ・プロシージャーを式の中から呼び出してください。プロトタイプで CLTPGM キーワードが使用された場合には、戻り値を得ることができないので、パラメーターを値によって渡す必要があります。

ローカル・プログラムの定義方法、およびパラメーターを渡す場合の規則については、 255 ページの『第 18 章 定義仕様』を参照してください。プロシージャー、サブプロシージャー、およびプロトタイピングについては、 65 ページの『第 6 章 サブプロシージャーおよびプロトタイプ』を参照してください。プログラムの呼び出しおよび複数プロシージャーの使用については、*VisualAge for RPG プログラミング* を参照してください。

命令拡張 M および R の使用法については、 392 ページの『数値演算の精度の規則』を参照してください。

注: CALLP を使用して呼び出されたプログラムの実行は、CALLP が実行された後に、他のステートメントより前に完了します。

次の例では、パラメーター fld1 がプログラム pgm1 に渡されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++
D                                   Functions-cont++++++++++++++++++++
D pgm1            PR                CLTPGM('testprog')
D fld1                           20A VALUE
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq
C*
C                   CALLP     pgm1(fld1)                          90
```

図 201. CALLP 演算命令

CASxx (サブルーチンの条件付き起動)

フリー・フォーム構文	(使用できません。IF および EXSR 命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CASxx	被比較数	被比較数	サブルーチン名	HI	LO	EQ

CASxx 演算命令は、処理するサブルーチンを条件付きで選択するために使用されます。選択は、xx によって指定される演算項目 1 と演算項目 2 の関係に基づいて行なわれます。xx によって指示された関係が演算項目 1 と演算項目 2 の間に存在していた場合に、結果フィールドに指定されたサブルーチンが処理されます。xx によって指示された関係が存在していない場合は、プログラムは CAS グループ内の次の CASxx 演算命令から続行されます。xx 値のリストについては、361 ページの『比較命令』 を参照してください。

CAS グループには、1 つの CASxx 演算命令だけを含めることができます。最後の CASxx 演算命令の次に ENDCS 演算命令がなければなりません。サブルーチンが処理された後は、そのサブルーチンで制御が別の演算命令に渡されないかぎり、プログラムは ENDCS 演算命令の後の次の演算命令から続行されます。

演算項目 1 および演算項目 2 を指定する場合は、それらにリテラル、名前付き固定情報、表意定数、フィールド名、テーブル名、配列エレメント、データ構造名、またはブランクを含めることができます。演算項目 1 と演算項目 2 の両方が同じデータ・タイプでなければなりません。ブランクは、xx がブランクであり、演算結果標識が指定されていない場合にのみ有効です。

結果フィールドには、ユーザー・サブルーチンの名前、あるいはキーワード *TERMSR、 *PSSR、または *INZSR のうち 1 つが含まれていることが必要です。
- *TERMSR は、正常終了時に実行されるサブルーチンを指定します。
- *PSSR は、これがプログラムで検出された例外 / エラーを処理するプログラム例外 /エラー・サブルーチンであることを指定します。
- *INZSR は、初期化時に実行されるサブルーチンを指定します。

CASxx 演算命令の場合には条件付け標識を指定することができますが、CAS グループのための ENDCS 演算命令に条件付け標識を指定することはできません。

CASbb 演算命令で、演算項目 1 および演算項目 2 は、71 - 76 桁目に演算結果標識が指定された場合にのみ必要です。71 - 76 桁目に演算結果標識が指定されない CASbb 演算命令は、その CASbb 演算命令の結果フィールドに名前の指定されたサブルーチンが無条件で実行されるので、機能的には EXSR 演算命令と同一です。同じ CAS グループ内の無条件 CASbb 演算命令の後の CASxx 演算命令は、いずれもテストされることがありません。したがって、無条件 CASbb 演算命令の通常の配置は、CAS グループ内の他のすべての CASxx演算命令の後となります。

演算結果標識が指定された場合には、それらが次のように設定されます。
- 高: (71 ～ 72) 演算項目 1 が演算項目 2 より大きい場合。
- 低: (73 ～ 74) 演算項目 1 が演算項目 2 より小さい場合。

- 等しい: (75 ～ 76) 演算項目 1 が演算項目 2 と等しい場合。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  CASGE 演算命令は FieldA を FieldB と比較します。FieldA が
C*  FieldB より大きいか等しい場合に、Subr01 が処理され、プログラム
C*  は ENDCS 演算命令の後の演算命令から続行されます。
C*
C     FieldA        CASGE     FieldB        Subr01
C*
C*  FieldA が FieldB より大きくないか等しくない場合には、プログラム
C*  は次に FieldA を FieldC と比較します。FieldA が FieldC と等しい
C*  場合に、SUBR02 が処理され、プログラムは ENDCS 演算命令の後の
C*  演算命令から続行されます。
C*
C     FieldA        CASEQ     FieldC        Subr02
C*
C*  FieldA が FieldC と等しくない場合には、CAS 演算命令によって
C*  Subr03 が処理された後に、プログラムは ENDCS 演算命令の後の
C*  演算命令から続行されます。
C*  前の CASxx 演算命令がいずれも満たされなかった場合には、
C*  CAS ステートメントを使用してサブルーチンが指定されます。
C*
C                   CAS                     Subr03
C*
C*  ENDCS 演算命令は CAS グループの終わりを指示します。
C*
C                   ENDCS
```

図 202. CASxx 演算命令

CAT (2 つのストリングの連結)

フリー・フォーム構文	(使用できません。+ 演算子を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CAT (P)	ソース・ストリング 1	ソース・ストリング 2: ブランクの数	目的のストリング			

CAT 演算命令は、演算項目 2 に指定されたストリングを演算項目 1 に指定されたストリングの終わりに連結し、それを結果フィールドに入れます。ソース・ストリングと目的のストリングは、すべてが同じタイプ、すなわち、すべてが文字、すべてがグラフィック、またはすべてが UCS-2 のいずれかでなければなりません。

演算項目 1 を指定する場合には、フィールド名、配列エレメント、名前付き固定情報、データ構造名、テーブル名、またはリテラルとすることができるストリングがそこに含まれていなければなりません。演算項目 1を指定しない場合には、演算項目 2 が結果フィールドのストリングの終わりに連結されます。

注: 以下の CAT 演算命令の説明において、演算項目 1 が指定されない場合には、演算項目 1 に対する参照は結果フィールドに適用されます。

演算項目 2 には、ストリングが含まれていなければなりませんが、連結されたストリングの間に挿入するブランクの数を含めることもできます。その形式は、ストリングの後にコロンが続き、その後にブランクの数が続きます。ブランクはデータの形式となります。たとえば、文字データの場合のブランクは x'20' であるのに対して、UCS-2 データの場合のブランクは x'0020' です。グラフィック・ストリングを連結する場合には、ブランクは 2 バイト・ブランクとなります。このストリング部分には、フィールド名、配列エレメント、名前付き固定情報、データ構造名、テーブル名、リテラル、またはデータ構造サブフィールド名を含めることができます。ブランクの数は、小数点以下の桁数がゼロの数字でなければなりませんが、名前付き固定情報、配列エレメント、リテラル、テーブル名、またはフィールド名を含めることができます。

コロンを指定した場合には、ブランクの数を指定しなければなりません。コロンが指定されない場合には、演算項目 1 に末尾ブランクがあればその末尾ブランクとの連結が行なわれ、演算項目 1 が指定されていなければ結果フィールドとの連結が行なわれます。

ブランクの数 (N) が指定された場合には、演算項目 1 は左寄せして結果フィールドにコピーされます。演算項目 1 が指定されない場合には、結果フィールドのストリングが使用されます。次に、その最後のブランク以外の文字の後に N 個のブランクが追加されます。それから、演算項目 2 がこの結果に付加されます。 N 個のブランクが結果に追加される場合に、演算項目 2 の中の先行ブランクはカウントされません。それらは、単に演算項目 2 のパーツと見なされます。ブランクの数が指定されない場合には、演算項目 1 と演算項目 2 の末尾および先行ブランクが結果に組み込まれます。

結果フィールドは、ストリングでなければなりませんが、フィールド名、配列エレメント、データ構造名、またはテーブル名を含めることができます。その長さは、演算項目 1 と演算項目 2 を結合し、それに中間のブランクの数を加えた長さでなければなりません。そうでない場合には、右側から切り捨てが行なわれます。

P 命令拡張は、連結が行なわれた後に結果フィールドが演算命令の結果より長かった場合に、結果フィールドの右側にブランクを埋め込む必要があることを指示します。埋め込みが指定されない場合には、フィールドの左端パーツにのみ影響します。

実行時に、ブランクの数がゼロより小さくなった場合には、コンパイラーがデフォルトでブランクの数をゼロにします。

演算項目 1、演算項目 2、または結果フィールドに表意定数を使用することはできません。演算項目 1と結果フィールド、または演算項目 2 と結果フィールドについてデータ構造のオーバーラップは許されません。

378 ページの『ストリング命令』は、ストリング演算命令を指定する場合の一般規則について説明しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* CAT は LAST を NAME に連結し、演算項目 2 に指定された通りに 1 つ
C* のブランクを挿入します。TEMP には 'Mr.bSmith' が入っています。
C                   MOVE      'Mr.   '      NAME              6
C                   MOVE      'Smith '      LAST              6
C     NAME          CAT       LAST:1        TEMP              9
C*
C* CAT は 'OS' を STRING に連結し、'OSXX' を TEMP に入れます。
C                   MOVE     'XX'         STRING             2
C     'OS'          CAT      STRING         TEMP             4
C*
C* 次の例は、TEMP が 10 バイト・フィールドとして定義されること
C* を除いて、前の例と同じです。P 命令拡張は、連結結果の 'OSXX'
C* が埋め込まれない結果フィールドの右端位置にブランクが使用
C* されることを指示します。結果として、連結の後に、TEMP には
C* 'OSXXbbbbbb' が含まれます。
C*
C                   MOVE      *ALL'*'       TEMP             10
C                   MOVE      'XX'          STRING            2
C     'OS'          CAT(P)    STRING        TEMP
C*
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の例は、演算項目 2 の先行ブランクを示しています。CAT の後に、
C* RESULT には 'MR.bSMITH' が含まれます。
C*
C                   MOVE      'MR.'         NAME              3
C                   MOVE      ' SMITH'      FIRST             6
C     NAME          CAT       FIRST         RESULT            9
C*
C*  次の例は、演算項目 1 のない CAT の使用を示しています。
C*  FLD2 は 9 文字のストリングです。連結の前に、そこには
C*  'ABCbbbbbb' が含まれています。FLD1 には 'XYZ' が含まれて
C*  います。連結の後に、FLD2 には 'ABCbbXYZb' が含まれます。
C*
C                   MOVEL(P) 'ABC'          FLD2              9
C                   MOVE     'XYZ'          FLD1              3
C                   CAT      FLD1:2         FLD2
```

図 203. CAT 演算命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 *
 * 次の例は、グラフィック・ストリングの使用を示しています。
 *
DName++++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++
 *       Graffld の値は 'AACCBBGG' です。
 *       CAT の後の Graffld2 の値は 'aa    AACCBBGG      ' です。
 *       CAT の後の Graffld3 の値は CAT 'AABBCCDDEEFFGGHHAACC' です。
 *
D Graffld                        4G   INZ(G'AACCBBGG')
D Graffld2                      10G   INZ
D Graffld3                      10G   INZ(G'AABBCCDDEEFFGGHH')
 CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
 * 値 2 は区切り文字としての 2 つのグラフィック・ブランクを表します。
C     G'aa'         cat       Graffld:2     Graffld2
C                   cat       Graffld       Graffld3
```

図 204. グラフィック・データによる CAT 演算命令

CHAIN (ファイルからのランダム検索)

フリー・フォーム構文	CHAIN{(EN)} *search-arg name* {*data-structure*}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CHAIN (EN)	検索引き数	名前 (ファイルまたはレコード様式)	データ構造	NR	ER	_

CHAIN 演算命令は、全手順ファイル (ファイル仕様の 18 桁目に F) またはサブファイルからレコードを検索し、レコード識別標識をオンに設定し (入力仕様に指定されている場合)、そのレコードからのデータを入力フィールドに入れます。

ファイルまたはレコード様式からのデータの検索

ファイルはファイル仕様に指定されていることが必要です。これは、リモート・サーバー・ファイルまたはローカル・ファイルにすることができます。

検索引き数 *search-arg* は、レコードの検索に使用されるキー、相対レコード番号、または KLIST 名でなければなりません。

- キーによるアクセスの場合には、*name* オペランドはリモート OS/400 ファイルでなければなりません。検索引き数は、フィールド名、名前付き固定情報、表意定数、またはリテラルとすることができます。このファイルが外部記述ファイルである場合には、検索引き数を KLIST 名にすることができます。
- 相対レコード番号によるアクセスの場合には、検索引き数は、整数リテラルまたは小数点以下の桁数がゼロの数値フィールドを指定しなければなりません。
- グラフィックおよび UCS-2 キー・フィールドの CCSID は、ファイル内のキーと同じでなければなりません。

name オペランドは、読み取るファイルまたはレコード様式名を指定します。

- *name* がファイル名の場合には、検索引き数と一致する最初のレコードが検索されます。
- *name* が OS/400 ファイル名であり、*MBR ALL が指定された場合には、現行のオープン・ファイル・メンバーだけが処理されます。
- *name* がローカル・ディスク・ファイルである場合には、それがプログラム記述でなければなりません。
- *name* がレコード様式名である場合は、ファイルを外部記述にすることができます。
- *name* がレコード様式名であり、キーによるアクセスの場合には、検索引き数と一致するキーをもつ指定レコード・タイプの最初のレコードが検索されます。

注: OS/400 リモート・ファイルの場合には、レコード・ロックがサポートされます。レコード・ロックは、ローカル・ファイルではサポートされません。

data-structure オペランドにデータ構造名を指定できるのは、*name* に指定したファイルがプログラム記述ファイル (ファイル仕様の 22 桁目の F によって識別) である場合だけです。*data-structure* オペランドにデータ構造名を指定した場合には、

CHAIN 演算命令で、レコード ID が検索引き数と一致した最初のレコードが検索されて、データ構造に入れられます。ファイルとデータ構造の間のデータの転送については、 367 ページの『ファイル命令』を参照してください。

name に指定したファイルが OS/400 入力 DISK ファイルである場合は、命令拡張は使用できません。ロックしないですべてのレコードが読み取られます。

name に指定したファイルが OS/400 UPDATE ファイルであり、命令拡張 N を指定しない場合は、CHAIN 演算命令でレコードがロックされます。レコードは、以下の時点までロックされています。
- レコードが更新される
- レコードが削除される
- 別のレコードが入力または更新のためにファイルから読み取られる
- SETLL または SETGT がファイルに実行される
- UNLOCK 演算命令がファイルに実行される
- 出力仕様によって定義されたフィールド名のない出力演算命令がファイルに実行される

レコードをファイルに追加する出力演算命令では、レコード・ロックは解放されません。

71 ～ 72 桁目には、検索引き数と一致するレコードがファイルにない場合にオンに設定される標識を指定することができます。この情報は、レコードが見つからないと '0' を、レコードが見つかると '1' を戻す %FOUND 組み込み関数から取得することもできます。

CHAIN の例外 (1000 より大きいファイル状況コード) を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、 41 ページの『ファイルの例外 / エラー』を参照してください。

75 ～ 76 桁目はブランクでなければなりません。

CHAIN 演算命令が正常に行なわれた場合には、後続の読み取り演算命令で、ここで検索されたレコードの論理的に後または前にあるレコードを検索できるように、ファイルが位置決めされます。CHAIN 演算命令が正常に完了しなかった時には、プログラム内のフィールドは変更されないままであるので、ファイルを再位置決めしなければ、それ以降はそのファイルに読み取り演算命令を実行できません。

CHAIN 演算命令が正常に実行された直後にファイルを更新した場合には、最後に検索されたレコードが更新されます。

レコードが見つからない場合、CHAIN 演算命令の実行中にエラーが起こった場合、または最後のレコードがすでに検索されている (ファイルの終わり) の場合には、データは検索されず、すべてのフィールドは変更されないままです。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*
*  CHAIN 演算命令は、ファイル FILEX から検索引き数 KEY
*  (演算項目 1) と同じ値のキー・フィールドをもつ最初の
*  レコードを検索します。
/FREE
    CHAIN  KEY  FILEX;

 //  キー値が検索引き数と等しいレコードが見つからなかった
 //  場合には、%FOUND は '0' を戻し、EXSR 演算命令が処理され
 //  ます。キー値が検索引き数と等しいレコードが見つかった
 //  場合には、プログラムは EXSR 演算命令の後の演算命令から
 //  続行されます。

    IF  NOT %FOUND;
       EXSR  Not_Found;
    ENDIF;
/END-FREE
```

図 205. ファイル名を指定した CHAIN 演算命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  CHAIN 演算命令は、検索引き数 KEY に含まれている値を使用して、
C*  レコード・タイプが REC1 のレコードを外部記述ファイルから検索
C*  します。キー・フィールドが検索引き数と等しい、指定された
C*  タイプのレコードが見つからなかった場合には、標識 72 がオンに
C*  設定されます。複合キーをもつファイルからのレコードの検索には、
C*  KLIST を使用した複合キーが使用されます。
C*  キー・フィールドが検索引き数と等しい、指定されたタイプの
C*  レコードが見つかった場合には、標識 72 がオフに設定され、
C*  したがって、UPDATE 演算命令が処理されます。
C*
C     KEY           CHAIN     REC1                              72
C     KEY           KLIST
C                   KFLD                Field1
C                   KFLD                Field2
C                   IF        NOT *IN72
C*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  UPDATE 演算命令は REC1 レコード内のすべてのフィールドを変更します。
C*
C                   UPDATE    REC1
C                   ENDIF
C*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  次の例は、ロックを指定しない CHAIN を示しています。
C*
C                   MOVE 3                Rec_No
C     Rec_No        CHAIN (N) INPUT                             99
```

図 206. レコード様式名を指定し、ロックは指定しない CHAIN 演算命令

サブファイル・パーツからのレコードの検索

name がサブファイル・パーツである場合には、検索引き数は索引でなければなりません。 CHAIN演算命令では、索引を使用して、サブファイルからレコードが読み取られます。

サブファイル・パーツのレコードを更新または削除する前に、そのサブファイルがレコードに位置決めされている必要があります。*START および *END をサブファイル・パーツで使用することはできません。サブファイル・パーツからのフィールド値は、サブファイル・フィールドについて対応するプログラム値に割り当てられます。これらの値は、プログラムによって変更することができます。

CHECK (文字の検査)

フリー・フォーム構文	(使用できません。%CHECK 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CHECK (E)	比較ストリング	基本ストリング:開始	左側の桁	_	ER	FD

CHECK 演算命令は、基本ストリング (演算項目 2) 中の各文字が比較ストリング (演算項目 1) に示されている文字の中にあるかどうかを検査します。基本ストリングと比較ストリングは、同じタイプ、すなわち、両方が文字、両方がグラフィック、または両方が UCS-2 のいずれかでなければなりません。 (グラフィックおよび UCS-2 タイプは同じ CCSID 値をもっている必要があります。) 検査は、演算項目 2 の左端文字から始まり、左から右へ文字ごとに続けられます。基本ストリングの個々の文字が演算項目 1 の文字と比較されます。 演算項目 1 の中に演算項目 2の文字と一致するものが存在していた場合に、次の基本ストリング文字が検査されます。一致が見つからなかった場合には、正しくない文字の位置を示す整数値が結果フィールドに入れられます。

最初の正しくない文字が見つかった時、あるいは基本ストリングの終わりに達した時に、この演算命令による検査が停止されます。正しくない文字が見つからなかった場合には、結果フィールドはゼロに設定されます。

演算項目 1 は、ストリングでなければなりませんが、フィールド名、配列エレメント、名前付き固定情報、データ構造名、データ構造サブフィールド、リテラル、またはテーブル名を含めることができます。

演算項目 2 には、基本ストリングか、基本ストリングに続いてコロン、および開始桁のいずれかが含まれている必要があります。基本ストリングには、フィールド名、配列エレメント、名前付き固定情報、データ構造名、リテラル、またはテーブル名が含まれていなければなりません。開始桁は、小数点以下の桁数のない数字でなければなりませんが、名前付き固定情報、配列エレメント、フィールド名、リテラル、またはテーブル名とすることができます。開始桁が指定されない場合には、1 の値が使用されます。開始桁が 1 より大きい場合には、結果フィールドの値は、その開始桁とは無関係に、基本ストリングの左端桁との相対関係で決まります。

結果フィールドを指定する場合には、それを数値変数、数値配列エレメント、数値テーブル名、または数値配列とすることができます。結果フィールドを指定しない場合には、75 - 76 桁目に検出標識を指定する必要があります。

結果フィールドで小数点以下の桁数は使用しないようにしてください。

結果フィールドが配列である場合には、その配列内にあるエレメントと同じ数のオカレンスについて最初の正しくない文字が見つかった後に、この演算命令による検査が続行されます。配列エレメントが正しくない文字の数より多かった場合には、残りのすべてのエレメントがゼロに設定されます。グラフィックまたは UCS-2 データが使用された場合には、結果フィールドにグラフィック桁数が入れられます (すなわち、3 桁目にある 3 番目のグラフィック文字は 5 桁目となります)。

CHECK の例外 (プログラム状況コード 100) を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、 52 ページの『プログラム例外およびエラー』を参照してください。

75 ～ 76 桁目には、誤った文字が見つかった場合にオンに設定される標識を指定することができます。この情報は、また、誤った文字が見つかった場合に '1' を戻す %FOUND 組み込み関数から入手することもできます。

演算項目 1、演算項目 2、または結果フィールドに表意定数を使用することはできません。演算項目 1と結果フィールド、または演算項目 2 と結果フィールドについてデータ構造のオーバーラップは許されません。

378 ページの『ストリング命令』は、ストリング演算命令を指定する場合の一般規則について説明しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D* 次の例の後に、N=6 で検出標識 90 がオンになります。開始桁は 2 で
D* あるので、見つかった最初の非数値文字は '.' です。
D*
D*
D
D Digits          C                   '0123456789'
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C
C                   MOVE      '$2000.'      Salary
C     Digits        CHECK     Salary:2      N                       90
C*
C
C                   MOVE      '$2000.'      Salary
C     Digits        CHECK     Salary:2      N
C                   IF        %FOUND
C                   EXSR      NonNumeric
C                   ENDIF
C*
C*
C* 演算項目 1 はブランクであるので、CHECK は最初の非ブランク文字の
C* 位置を示します。STRING に 'bbbthe' が含まれていた場合には、
C* NUM には値 4 が入れられます。
C*
C
C           ' '     CHECK     String        Num               2 0
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
.
.
.
```

図 207. CHECK 演算命令 (1/2)

```
.
.
.
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D* 次の例は、FIELD に文字 A ～ J だけが入っているかどうかを検査します。
D* 結果として、CHECK 演算命令の後は ARRAY=(136000) になります。
D* 標識 90 はオンになります。
D*
D
D Letter          C                   'ABCDEFGHIJ'
D
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C
C                   MOVE      '1A=BC*'      Field             6
C     Letter        CHECK     Field         Array                    90
C
C*
C* 次の例では、FIELD には文字 A-J だけが入っているので、
C* ARRAY=(000000) となります。標識 90 はオフになります。
C*
C
C                   MOVE      'FGFGFG'      Field             6
C     Letter        CHECK     Field         Array                    90
C
C
```

図 207. CHECK 演算命令 (2/2)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D
 * 次の例は、フィールドのグラフィック 2 桁目で始まるグラフィック
 * 文字有効であるかどうか DBCS フィールドを検査します。
D
 *       Graffld の値は 'DDBBCCDD' です。
 *       これはストリングに含まれていない最初の文字が 'DD' である
 *       ので、CHECK の後の num の値は 4 になります。
D
D Graffld                       4G   INZ(G'DDBBCCDD')
D Num                           5 0
D
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
C
C
C     G'AABBCC'     check     Graffld:2     Num
```

図 208. グラフィック・データによる CHECK 演算命令

CHECKR (逆方向の検査)

フリー・フォーム構文	(使用できません。%CHECKR 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CHECKR (E)	比較ストリング	基本ストリング:開始	右側の桁	_	ER	FD

CHECKR 演算命令は、基本ストリング中の各文字が比較ストリングに示されている文字の中にあるかどうかを検査します。基本ストリングと比較ストリングは、同じタイプ、すなわち、両方が文字、両方がグラフィック、または両方が UCS-2 のいずれかでなければなりません。(グラフィックおよび UCS-2 タイプは同じ CCSID 値をもっている必要があります。). 検査は、演算項目 2 の右端文字から始まり、右から左へ文字ごとに続けられます。基本ストリングの個々の文字が演算項目 1 の文字と比較されます。 演算項目 1 の中に演算項目 2 の文字と一致するものが存在していた場合に、次のソース文字が検査されます。一致が見つからなかった場合には、正しくない文字の位置を示す整数値が結果フィールドに入れられます。 検査は右側から実行されますが、結果フィールドには、左側からの相対関係で位置が入れられます。

演算項目 1 は、ストリングでなければなりませんが、フィールド名、配列エレメント、名前付き固定情報、データ構造名、データ構造サブフィールド、リテラル、またはテーブル名を含めることができます。

演算項目 2 には、基本ストリングか、基本ストリングに続いてコロン、および開始桁のいずれかが含まれている必要があります。基本ストリングには、フィールド名、配列エレメント、名前付き固定情報、データ構造名、データ構造サブフィールド名、リテラル、またはテーブル名が含まれていなければなりません。開始桁は、小数点以下の桁数のない数字でなければなりませんが、名前付き固定情報、配列エレメント、フィールド名、リテラル、またはテーブル名とすることができます。開始桁が指定されない場合には、ストリングの長さが使用されます。

結果フィールドを指定する場合には、それを数値変数、数値配列エレメント、数値テーブル名、または数値配列とすることができます。結果フィールドを指定しない場合には、75 - 76 桁目に検出標識を指定する必要があります。結果フィールドの値は、その開始桁とは無関係に、ソース・ストリングの左端桁との相対関係で決まります。

結果フィールドで小数点以下の桁数は使用しないようにしてください。

結果フィールドが配列である場合には、その配列内にあるエレメントと同じ数のオカレンスについて最初の正しくない文字が見つかった後に、この演算命令による検査が続行されます。配列エレメントが正しくない文字の数より多かった場合には、残りのすべてのエレメントがゼロに設定されます。結果フィールドが配列でない場合には、最初の正しくない文字が見つかった時、あるいは基本ストリングの終わりに達した時に、この演算命令による検査が停止されます。正しくない文字が見つからなかった場合には、結果フィールドはゼロに設定されます。

グラフィックまたは UCS-2 データが使用された場合には、結果フィールドにグラフィック桁数が入れられます (すなわち、3 桁目にある 3 番目のグラフィック文字は 5 桁目となります)。

CHECKR の例外 (プログラム状況コード 100) を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

75 ～ 76 桁目には、誤った文字が見つかった場合にオンに設定される標識を指定することができます。この情報は、また、誤った文字が見つかった場合に '1' を戻す %FOUND 組み込み関数から入手することもできます。

演算項目 1、演算項目 2、または結果フィールドに表意定数を使用することはできません。演算項目 1と結果フィールド、または演算項目 2 と結果フィールドについてデータ構造のオーバーラップは許されません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 演算項目 1 はブランク文字であるので、CHECKR は最初の非ブランク
C* 文字の位置を示します。この CHECKR の使用によって、ストリングの
C* 長さを判別することができます。STRING に 'ABCDEF   ' が含まれ
C* ている場合には、NUM に値 6 が入れられます。
C* エラーが起こった場合には、%ERROR は '1' を戻すように設定
C* され、%STATUS は状況コード 00100 を戻すように設定されます。
C*
C
C     ' '           CHECKR(E) String        Num
C
C                   SELECT
C                   WHEN      %ERROR
C ... an error occurred
C                   WHEN      %FOUND
C ... NUM is less than the full length of the string
C                   ENDIF
C*
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
D*
D* 次の例の後に、N=1 で検出標識 90 がオンになります。開始桁は
D* 5 であるので、この演算命令は右端の 0 で始まり、見つかった
D* 最初の非数値は '$' です。
D*
D Digits          C                   '0123456789'
D
D*
```

図 209. CHECKR 演算命令 (1/2)

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   C                   MOVE      '$2000.'      Salary          6
C     Digits        CHECKR    Salary:5      N                         90
C                   .
.
.
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*
D* 次の例は、FIELD に文字 A-J だけが入っているかどうかを検査します。
D* 結果として、CHECKR 演算命令の後は ARRAY=(876310) となります。
D* 標識 90 はオンになります。
D
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D Array           S              1    DIM(6)
D Letter          C                   'ABCDEFGHIJ'
D
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   C                   MOVE      '1A=BC***'    Field           8
C     Letter        CHECKR    Field         Array                     90
C
```

図 209. CHECKR 演算命令 (2/2)

CLEAR (消去)

フリー・フォーム構文	CLEAR {*NOKEY} {*ALL} *name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CLEAR	*NOKEY	*ALL	name (変数またはレコード様式)			
CLEAR			name (ウィンドウまたはサブファイル)			

CLEAR 演算命令は、フィールド・タイプ (数字、文字、グラフィック、UCS-2、標識、ポインター、または日付/時刻/タイム・スタンプ) に応じて、以下のものをそのデフォルトの初期設定値に設定します。

- 構造内のエレメント (レコード様式、データ構造、配列、テーブル)
- 変数 (フィールド、サブフィールド、配列エレメント、標識)
- ウィンドウ上の入力フィールド・パーツ
- サブファイル

データ・タイプのデフォルトの初期設定値については、107 ページの『第 9 章 データ・タイプおよびデータ形式』 を参照してください。消去しようとする構造または変数が可変長である場合には、その長さが 0 に変更されます。CLEAR 演算命令によって、実行時にエレメント単位だけでなく、グローバル単位で構造を消去できるようになります。

ウィンドウまたはサブファイルの場合には、*ALL、*NOKEY は指定できません。

変数の消去

*NOKEY は、指定できません。

*ALL は任意指定です。*ALL を指定し、*name* オペランドが複数回繰り返しデータ構造またはテーブル名である場合には、すべてのオカレンスまたはテーブル・エレメントが消去され、オカレンス・レベルまたはテーブル索引は 1 に設定されます。

name オペランドは、消去される変数を指定します。 *name* オペランドの一定の項目は、以下のように消去アクションを決定します。

単一オカレンス・データ構造

すべてのフィールドは、構造内で宣言されている順序で消去されます。

複数回繰り返しデータ構造

*ALL を指定しないと、*現行* オカレンス内のすべてのフィールドが消去されます。*ALL を指定すると、すべての オカレンスのすべてのフィールドが消去されます。

テーブル名
*ALL を指定しないと、*現行* テーブル・エレメントが消去されます。*ALL を指定すると、すべてのテーブル・エレメントが消去されます。

配列名 配列全体が消去されます。

配列エレメント (標識を含む)
指定されたエレメントだけが消去されます。

レコード様式の消去

name オペランドが DISK レコード様式名を指定する場合には、*NOKEY を指定して、キー・フィールドを除くすべてのフィールドを消去できます。

*ALL は任意指定です。*ALL を指定したが、*NOKEY を指定しない場合には、レコード様式内のすべてのフィールドが消去されます。*ALL を指定しない場合には、レコード様式内の出力フィールドだけが影響を受けます。*NOKEY を指定した場合には、*ALL を指定しても、キー・フィールドは消去されません。

name オペランドは、消去されるレコード様式です。フィールドは、レコード様式内で定義されている順に消去されます。

DISK または PRINTER ファイルのレコード様式内のフィールドは、プログラム中でそのレコード様式が出力となっている場合にだけ影響を受けます。入力専用フィールドは、*ALL を指定した場合を除いて RESET 演算命令の影響を受けません。

注: 論理ファイルの入力専用フィールドは、出力仕様に現れますが、実際にはファイルに書き込まれません。*NOKEY の指定のない CLEAR または RESET がこうしたフィールドを含むレコードに対して実行されると、これらのフィールドは、出力仕様に現れるので、消去またはリセットされます。

ウィンドウ上の入力フィールドの消去

name オペランドがウィンドウを指定する場合には、そのウィンドウに入力フィールドが含まれていなければなりません。

ウィンドウ上のすべての入力フィールドは、次のように消去してデフォルト値とすることができます。
- 数値フィールドはゼロで消去されます。
- 文字フィールドはブランクで消去されます。

対応するプログラムのフィールドも、そのタイプに応じてゼロまたはブランクに設定されます。たとえば、ウィンドウ INVENTORY に文字入力フィールド ENT0000B と数値入力フィールド ENT0000C が含まれている場合には、CLEAR 演算命令では次と同じようなことが実行されます。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
                    EVAL      ENT0000B = *BLANKS
                    EVAL      ENT0000C = *ZERO
                    EVAL      %setatr('inventory':'ent0000b':'text') = ENT0000B
                    EVAL      %setatr('inventory':'ent0000c':'text') = ent0000c
```

図 210. ウィンドウの消去

サブファイルの消去

name オペランドがサブファイルを指定する場合には、そのサブファイルのすべての項目が消去されて、そのカウント属性はゼロに設定されます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D DS1             DS
D  Num                     2      5  0
D  Char                   20     30A
D
D MODS            DS                   OCCURS(2)
D  Fld1                    1      5
D  Fld2                    6     10  0

 *  以下の例では、CLEAR によってデータ構造 DS1 の中のすべてのサブフィールド
 *  はそのデフォルトに、CHAR はブランクに、NUM はゼロに設定されます。
 /FREE
   CLEAR DS1;

  //  以下の例では、CLEAR によって複数回繰り返しデータ構造 MODS の
  //  すべてのオカレンスはそのデフォルト値に、Fld1 はブランクに、
  //  Fld2 はゼロに設定されます。
   CLEAR *ALL MODS;
 /END-FREE
```

図 211. CLEAR 演算命令

CLOSE (ファイルのクローズ)

フリー・フォーム構文	CLOSE{(E)} *file-name**ALL

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CLOSE (E)		file-name または *ALL		_	ER	_

CLOSE 演算命令は、1 つまたは複数のファイルをクローズします。そのファイルに OPEN 演算命令を指定しないかぎり、ファイルを再使用することはできません。ファイルは、ローカル・ファイルまたはリモート・ファイルにすることができます。

すでにクローズされているファイルに対する CLOSE 演算命令では、エラーが生成されます。

file-name は、クローズされるファイルの名前を指定します。キーワード *ALL を指定して、一度にすべてのファイルをクローズすることができます。配列またはテーブル・ファイル (ファイル仕様の 18 桁目の T で指定) を指定することはできません。

CLOSE の例外 (1000 より大きいファイル状況コード) を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

71、72、75、および 76 桁目はブランクでなければなりません。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*  明示 CLOSE 演算命令は FILEB をクローズします。

/FREE
   CLOSE FILEB;

 //  CLOSE *ALL 演算命令は、プログラム内のすべてのファイル
 //  をクローズします。再度使用したいファイルには明示
 //  OPEN を指定する必要があります。CLOSE 演算命令が
 //  正常に完了しないと、%ERROR は '1' を戻します。

   CLOSE(E) *ALL;

/END-FREE
```

図 212. CLOSE 演算命令

CLSWIN (ウィンドウのクローズ)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CLSWIN (E)		ウィンドウ名		_	ER	_

CLSWIN 演算命令は、ウィンドウをクローズし、それを表示から除去します。ウィンドウに対して破棄イベントが生成されます。このウィンドウは、アプリケーションの中で定義されている必要があります。

演算項目 2 には、クローズするウィンドウの名前が入れられます。

CHECKR の例外を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、 52 ページの『プログラム例外およびエラー』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                            Extended-factor2++++++++++++++++++++++++++++++
C*
C* UPDCUST という名前のウィンドウがクローズされます。
C                   CLSWIN    'UPDCUST'
```

図 213. CLSWIN 演算命令

COMMIT (コミット)

フリー・フォーム構文	COMMIT{(E)}

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
COMMIT (E)				_	ER	_

COMMIT 演算命令は、データベース変更のグループを 1 単位として処理します。この単位と関連した変更は、ROLBK 演算命令を使用してロールバックすることができます。

COMMIT 演算命令を使用できるのは、OS/400 ファイルだけです。これは、ローカル・ファイルで使用することはできません。

OS/400 データベース・ファイルをコミットメント制御のためにオープンする場合は、ファイル仕様に COMMIT を指定してください。COMMIT を出したコンポーネントとは無関係に、コミットメント制御下でオープンされたファイルだけが、この COMMIT 演算命令の影響を受けます。

COMMIT 演算命令では、ファイルの位置は変更されません。コミットメント制御下にあるファイルのすべてのレコード・ロックが解放されます。

コミットメント制御環境は、1 つのサーバーだけに開始できます。これらのファイルは他のサーバー上で使用することができますが、コミットメント制御下で操作することはできません。

コミットメント制御は、アプリケーションが終了した時に終了します。OS/400 データベース上で明示的にコミットまたはロールバックされていない変更が保留中である場合は、それらの変更はアプリケーションの終了時にロールバックされます。アプリケーションをコミットメント制御環境で実行する前に、GUI設計機能を使用して、コミットメント・レベルを定義する必要があります。GUI Designer を使用したサーバー情報の定義に関する詳細については、*VisualAge for RPG プログラミング* を参照してください。

COMMIT の例外 (プログラム状況コード 802 ～ 805) を処理するには、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方は指定できません。たとえば、コミットメント制御がアクティブでない場合には、エラーが起こります。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

COMP (比較)

フリー・フォーム構文	(使用できません。=、<、>、<=、>=、または <> 演算子を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
COMP	被比較数	被比較数		HI	LO	EQ

COMP 演算命令は、演算項目 1 を演算項目 2 と比較します。

演算項目 1 および演算項目 2 には、リテラル、名前付き固定情報、フィールド名、テーブル名、配列エレメント、データ構造、または表意定数が含まれていなければなりません。演算項目 1 と演算項目 2 は同じデータ・タイプでなければなりません。3 つのすべての条件に同じ標識を指定しないようにしてください。演算結果標識を指定した場合には、それが比較の結果を表すようにオンまたはオフに設定されます。

比較の結果として、標識が次のようにオンに設定されます。
- 高: (71 〜 72) 演算項目 1 が演算項目 2 より大きい場合。
- 低: (73 〜 74) 演算項目 1 が演算項目 2 より小さい場合。
- 等しい: (75 〜 76) 演算項目 1 が演算項目 2 と等しい場合。

361 ページの『比較命令』には、比較演算命令を指定するための一般規則が説明されています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  初期フィールド値は次の通りです:
C*                    FLDA = 100.00
C*                    FLDB = 105.00
C*                    FLDC = 100.00
C*                    FLDD = ABC
C*                    FLDE = ABCDE
C*
C*  標識 12 はオンに設定され、標識 11 および 13 はオフに設定されます。
C     FLDA          COMP      FLDB                                 111213
C*
C*  標識 15 はオンに設定され、標識 14 はオフに設定されます。
C     FLDA          COMP      FLDB                                 141515
C*
C*  標識 18 はオンに設定され、標識 17 はオフに設定されます。
C     FLDA          COMP      FLDC                                 171718
C*
C*  標識 21 はオンに設定され、標識 20 および 22 はオフに設定されます。
C     FLDD          COMP      FLDE                                 202122
```

図 214. COMP 演算命令

DEALLOC (ストレージの解放)

フリー・フォーム構文	DEALLOC{(EN)} *pointer-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DEALLOC (E/N)			ポインター名	_	ER	_

DEALLOC 演算命令は、1 つ前のヒープ・ストレージの割り振りを解放します。*pointer-name*は、ヒープ・ストレージ割り振り操作 (RPG における ALLOC 演算命令かその他なんらかのヒープ・ストレージ割り振り機構) によって前に設定された値でなければならないポインターです。単にヒープ・ストレージを指しているだけでは十分ではありません。ポインターは割り振りの先頭に設定されていなければなりません。

ポインターによって指し示されたストレージは、このプログラムまたは活動化グループ内の他のプログラムによるこれ以降の割り振りのために解放されます。

命令拡張 N が指定された場合には、ポインターは正常な割り振り解除の後に *NULL に設定されます。

DEALLOC の例外 (プログラム状況コード 426) を処理するには、命令コード拡張 'E' かエラー標識 ERのいずれかを指定することができますが、両方は指定できません。*pointer-name*は、'N' が指定されていても、エラーが起こった場合に変更されません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

pointer-name は、基底ポインター・スカラー変数 (独立型フィールド、データ構造サブフィールド、テーブル名、または配列エレメント) でなければなりません。

ポインターがすでに *NULL になっていれば、実行時にエラーは示されません。

詳細については、370 ページの『メモリー管理命令』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
D Ptr1            S               *
D Fld1            S              1A
D BasedFld        S              7A   BASED(Ptr1)

 /FREE
    // 7 バイトのストレージがヒープから割り振られ、Ptr1 がそれを指し示す
    // ように設定されます。
    Ptr1 = %alloc (7);

    // DEALLOC はストレージを解放します。ここでこのストレージは、この
    // プログラムまたは活動化グループ内の他のプログラムによって割り振り
    // に使用できるようになります。(次の割り振りでは再び同じストレージが
    // 得られることもあれば、得られないこともありることに注意してください。)
    dealloc Ptr1;

    // Ptr1 は割り振り解除されたストレージをまだ指し示していますが、この
    // ポインターをその現在の値とともに使用しないようにしてください。
    // Ptr1 に基づく BasedFld へのアクセスの試みは無効です。
    Ptr1 = %addr (Fld1);

    // ポインターがプログラム・ストレージのアドレスに設定されている
    // ので、DEALLOC は無効です。%ERROR は '1' を戻すように設定され、
    // プログラム状況は 00426 に設定され (%STATUS が 00426 を戻し
    // ます) 、ポインターは変更されません。
    dealloc(e) Ptr1;

    // ストレージを割り振りおよび割り振り解除します。命令拡張 N
    // が指定されているので、Ptr1 は DEALLOC の後に値 *NULL を
    // をもちます。
    Ptr1 = %alloc (7);
    dealloc(n) Ptr1;
 /END-FREE
```

図 215. DEALLOC 演算命令

DEFINE (フィールド定義)

フリー・フォーム構文	(使用できません。定義仕様で LIKE または DTAARA キーワードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DEFINE	*LIKE	参照フィールド	定義されるフィールド			
DEFINE	*DTAARA	外部データ域	内部フィールド			

DEFINE 演算命令は、フィールドを別のフィールドの属性 (長さおよび小数点以下の桁数) に基づいて定義したり、あるいはフィールドを OS/400 データ域によって定義するために使用します。

条件付け標識 (9 ～ 11 桁目) は許可されていません。

別のフィールドに基づくフィールドの定義

演算項目 1 には *LIKE が含まれていなければなりません。

演算項目 2 には、参照されているフィールドが含まれていなければなりません。このフィールドは、プログラム記述または外部記述フィールドとすることができます。演算項目 2 のフィールドの属性は、結果フィールドで定義されるフィールドに使用されます。このフィールドは、プログラム記述または外部記述フィールドとすることができます。演算項目 2 をリテラル、名前付き固定情報、浮動数値フィールド、またはオブジェクトとすることはできません。演算項目 2 が配列、配列エレメント、またはテーブル名である場合には、配列またはテーブルのエレメントの属性を使用して、フィールドが定義されます。

結果フィールドには、定義されているフィールドの名前が入れられます。それを配列、配列エレメント、データ構造、またはテーブル名とすることはできません。

64 - 68 桁目 (フィールド長) を使用して、結果フィールドの項目を演算項目 2 の項目より長くしたりまたは短くすることができます。64 桁目には、フィールド長を増やすことを示す正符号 (+)かフィールド長を減らすことを示す負符号 (–) のいずれかを入れることができます。65 - 68 桁目には、長さの増分または減分を (右寄せして) 入れたり、あるいはブランクとすることができます。フィールド長項目は、グラフィック、UCS-2、数値、および文字フィールドの場合にだけ許されます。グラフィックまたは UCS-2 フィールドの場合には、フィールド長の差が 2 バイト文字で計算されます。

64 - 68 桁目がブランクであった場合に、結果フィールドの項目は演算項目 2 の項目と同じ長さであるものとして定義されます。

注: 定義しているフィールドの小数点以下の桁数の数値を変更することはできません。

演算項目 2 がグラフィックまたは UCS-2 フィールドである場合には、結果フィールドは同じタイプとして、すなわちグラフィックまたは UCS-2 として、定義されます。新規フィールドは、デフォルトのグラフィックまたは UCS-2 CCSID をもちます。新規フィールドの CCSID を演算項目 2 のフィールドと同じにしたい場合には、定義仕様で LIKE キーワードを使用してください。長さの調整は 2 バイトで表されます。

*LIKE DEFINE の例については、 531 ページの図 216 を参照してください。

データ域としてのフィールドの定義

演算項目 1 には *DTAARA が含まれていなければなりません。

演算項目 2 を指定する場合には、参照される OS/400 データ域をそこに入れる必要があります。演算項目 2 を指定しない場合には、結果フィールドがデータ域名として使用されます。

データ域名は、OS/400 データ域名か、「サーバー情報の定義」メニュー項目を使用して定義された一時変更名のいずれかにできます。GUI Designer を使用したサーバー情報の定義に関する詳細については、*VisualAge for RPG プログラミング* を参照してください。

結果フィールドには、フィールド、データ構造、データ構造サブフィールド、またはデータ域データ構造が含まれていなければなりません。これは、演算項目 2 に指定されたデータ域から読み取られたり、また、そのデータ域に書き込まれるデータを検索するための IN および OUT 演算命令で使用されるのと同じ名前です。データ域データ構造が結果フィールドに指定された場合には、VisualAge RPG アプリケーションはプログラムの開始時にそのデータ域からデータを検索し、プログラムの終了時にそのデータ域にデータを書き込みます。

結果フィールドに、次のものを含めることはできません。

- プログラム状況データ構造の名前またはプログラム状況データ構造のサブフィールドの名前
- ファイル情報データ構造またはファイル情報データ構造のサブフィールドの名前
- データ域データ構造のサブフィールドの名前
- 複数オカレンス・データ構造または複数オカレンス・データ構造のサブフィールドの名前
- 別の *DTAARA DEFINE ステートメントに現れるデータ構造
- 定義仕様の DTAARA キーワード上のデータ域名
- 入力レコード・フィールド
- 配列
- 配列エレメント
- テーブル

注: 結果フィールドがパック 10 進数サブフィールドを含むデータ域データ構造である場合は、OS/400 データ域に初期化済みの有効なパック 10 進数値が入っていなければなりません。

数値データ域の場合の最大長は、小数点以下が 9 桁の 24 桁です。小数点以下が 9 桁より小さくても、小数点の左側は最大で 15 桁であることに注意してください。

64 ～ 70 桁目を使用して、結果フィールドの長さと小数点以下の桁数を定義することができます。これは、演算項目 2 に指定したデータ域の外部記述と一致している必要があります。

*DTAARA DEFINE の例については、図 216 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
*
C*  FLDA は 7 桁の文字フィールドです。
C*  FLDB は小数点以下が 2 桁の 5 桁のフィールドです。
C*
C*
C*  FLDP は 7 桁の文字フィールドです。
C     *LIKE         DEFINE    FLDA          FLDP
C*
C*  FLDQ は 9 桁の文字フィールドです。
C     *LIKE         DEFINE    FLDA          FLDQ             +2
C*
C*  FLDR は 6 桁の文字フィールドです。
C     *LIKE         DEFINE    FLDA          FLDR             - 1
C*
C*  FLDS は小数点以下が 2 桁の 5 桁の数値フィールドです。
C     *LIKE         DEFINE    FLDB          FLDS
C*
C*  FLDT は小数点以下が 2 桁の 6 桁の数値フィールドです。
C     *LIKE         DEFINE    FLDB          FLDT           +  1
C*
C*  FLDU は小数点以下が 2 桁の 3 桁の数値フィールドです。
C     *LIKE         DEFINE    FLDB          FLDU             - 2
C*
C*  FLDX は小数点以下が 2 桁の 3 桁の数値フィールドです。
C     *LIKE         DEFINE    FLDU          FLDX
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  データ域 (TOTGRS) の属性 (長さと小数点以下の桁数) は、
C*  外部データ域の場合の属性と同じでなければなりません。
C*
C
C     *DTAARA       DEFINE                  TOTGRS           10 2
C
C*
C*  結果フィールドの項目 (TOTNET) は、VRPG プログラムの中で
C*  使用されるデータ域の名前です。演算項目 2 の項目 (TOTAL) は、
C*  システムに対して定義された通りのデータ域の名前です。
C
C     *DTAARA       DEFINE    TOTAL         TOTNET
C
```

図 216. DEFINE 演算命令

DELETE (レコードの削除)

フリー・フォーム構文	DELETE{(E)} {*search-arg*} *name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DELETE (E)	search-arg	name (ファイル、レコード様式、またはサブファイル)		NR	ER	_

DELETE 演算命令は、レコードを削除します。レコードが削除されてしまうと、それを検索することはできません。

検索引き数 (*search-arg*) を指定しない場合には、DELETE 演算命令によって現行レコードが削除されます。現行レコードは、検索される最後のレコードです。このレコードは、CHAIN または READ のような前の入力演算命令によってロックされているはずです。

検索引き数 (*search-arg*) を指定する場合には、削除されるレコードを識別するキー、相対レコード番号、またはサブファイル索引番号がその引き数に含まれていなければなりません。

- キーによるアクセスの場合には、*name* オペランドはリモート・ファイルでなければなりません。検索引き数はフィールド名、名前付き固定情報、またはリテラルでなければなりません。キーについて重複したレコードが存在していた場合には、重複したレコードの最初のレコードだけがファイルから削除されます。このファイルが外部記述である場合には、検索引き数を KLIST とすることができます。
- 相対レコード番号またはサブファイル索引番号によるアクセスの場合には、検索引き数は、小数点以下の桁数がゼロの数値固定情報または変数でなければなりません。
- グラフィックおよび UCS-2 キー・フィールドの CCSID は、ファイル内のキーと同じでなければなりません。

name オペランドは、レコードが削除されるファイルの名前またはそのファイル内のレコード様式の名前でなければなりません。

- ファイルは、OS/400 ファイルまたはローカル・ファイルにすることができます。
- レコード様式名を使用できるのは、外部記述 OS/400 ファイルだけです。検索引き数を指定しない場合には、レコード様式名はファイルから読み取られる最後のレコードの名前でなければなりません。さもないと、エラーが起こります。

検索引き数を指定する場合は、削除するレコードがファイルに見つからないとオンに設定される標識を 71 ～ 72 桁目に入れることができます。検索引き数を指定しない場合には、これらの桁はブランクにしておいてください。この情報は、レコードが見つからないと '0' を、レコードが見つかると '1' を戻す %FOUND 組み込み関数から取得することもできます。

DELETE の例外 (1000 より大きいファイル状況コード) を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

75 ～ 76 桁目はブランクのままにしてください。

レコードを削除する場合は、次の点に注意してください。
- サブファイル・パーツからレコードを削除すると、そのサブファイル中の残りのレコードの間のサブファイル・レコード索引番号のシフトの原因となります。
- ファイルに対する DELETE 演算命令が正常に行なわれた後に、そのファイルで順次読み取り操作が実行された場合には、削除済みレコードの後の次のレコードが入手されます。

DIV (除算)

フリー・フォーム構文	(使用できません。/ 演算子または %DIV 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DIV (H)	被除数	除数	商	+	–	Z

演算項目 1 を指定した場合には、DIV 演算命令によって演算項目 1 が演算項目 2 で除算されます。これを指定しない場合には、結果フィールドが演算項目 2 で除算されます。商は結果フィールドに入れられます。演算項目 1 が 0 の場合には、この演算命令の結果は 0 です。演算項目 2 をゼロにすることはできません。演算項目 2 をゼロにした場合には、VARPG 例外 / エラー処理ルーチンが制御を受け取ります。演算項目 1 と演算項目 2 は数字でなければならず、それぞれに次の 1 つを入れることができます: 配列、配列エレメント、フィールド、表意定数、リテラル、名前付き固定情報、サブフィールド、またはテーブル名。

演算項目 2 を 0 にすることはできません。演算項目 2 を 0 にした場合には、VRPG クライアント例外 /エラー処理ルーチンが制御を受け取ります。演算項目 2 は、数字でなければなりませんが、配列、配列エレメント、フィールド、表意定数、リテラル、名前付き固定情報、サブフィールド、またはテーブル名を含めることができます。

剰余の移動 (MVR) 演算命令が次の演算命令として指定されていない場合には、除算演算命令から生じた剰余は失われます。条件付け標識を使用する場合には、DIV 演算命令を MVR 演算命令の直前に指定する必要があります。

MVR 演算命令が DIV 演算命令より後に指定された場合には、除算演算命令の結果を四捨五入する (丸める) ことはできません。

注: DIV 演算命令に浮動形式のオペランドがある場合には、MVR 演算命令を DIV 演算命令の後に続けることはできません。しかし、浮動変数は、命令コード MVR の結果として指定することができます。

352 ページの『算術演算』には、算術演算を指定するための一般規則が説明されています。

355 ページの図 114 は、DIV 演算命令の例を示しています。

DO (実行)

フリー・フォーム構文	(使用できません。FOR 命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
DO	開始値	限界値	索引値			

DO 演算命令は、1 つのグループの演算命令を開始し、そのグループが処理される回数を指示します。演算命令のグループが処理される回数を指定する場合は、索引フィールド、開始値、および限界値を指定します。関連した ENDDO ステートメントがグループの終わりをマークします。DO グループの詳細については、 379 ページの『構造化プログラミング命令』を参照してください。

演算項目 1 を指定する場合には、そこに数値リテラル、名前付き固定情報、またはフィールド名を入れる必要があります。演算項目 1 を指定しない場合には、開始値は 1 です。

演算項目 2 を指定する場合には、そこに数値フィールド名、リテラル、または名前付き固定情報を入れる必要があります。演算項目 2 は小数点以下の桁数をゼロとして指定しなければならず、演算項目 2 を指定しない場合には、限界値は 1 です。

結果フィールドを指定する場合には、限界値とその増分を十分に入れられる大きさの数値フィールド名でなければなりません。結果フィールドのすべての値は、DO 演算命令が開始された時に演算項目 1 で置き換えられます。

ENDDO 演算命令と関連した演算項目 2 は、索引フィールド (DO 演算命令の結果フィールド) に追加される値を指定します。これは、小数点以下の桁数のない数値リテラルまたは数値フィールドでなければなりません。これを指定しない場合には、索引フィールドに 1 が追加されます。

DO グループは、DO 演算命令それ自体に加えて、DO および ENDDO ステートメント上の条件付け標識によって制御されます。DO ステートメント上の条件付け標識は、DO 演算命令を開始するかどうかを制御します。これらの標識は、DO ループの開始時に一度だけ検査されます。関連した ENDDO ステートメント上の条件付け標識は、DO グループをもう 1 回に繰り返すかどうかを制御します。これらの標識は、各ループの終わりで検査されます。

DO 演算命令は、次の 7 つのステップに従っています。

1. DO ステートメント行上の条件付け標識が満たされた場合に、DO 演算命令が処理されます (ステップ 2)。標識が満たされない場合には、関連した ENDDO ステートメントの後の、次に処理される演算命令に制御が渡されます (ステップ 7)。
2. DO 演算命令が開始された時に、開始値 (演算項目 1) が索引フィールド (結果フィールド) に移されます。
3. 索引値が限界値より大きい場合には、関連した ENDDO ステートメントの後の演算命令に制御が渡されます (ステップ 7)。それ以外の場合には、制御は DO ステートメントの後の最初の演算命令に渡されます (ステップ 4)。
4. DO グループ内の個々の演算命令が処理されます。

5. ENDDO ステートメント上の条件付け標識が満たされない場合には、関連した ENDDO ステートメントの後の演算命令に制御が渡されます (ステップ 7)。それ以外の場合には、ENDDO 演算命令が処理されます (ステップ 6)。
6. 索引フィールドに増分を追加することによって、ENDDO 演算命令が処理されます。制御はステップ 3に渡されます。(制御がステップ 3 に渡された時に、DO ステートメント上の条件付け標識は再度テスト (ステップ 1) されないことに注意してください。)
7. ENDDO ステートメントの後のステートメントは、DO または ENDDO ステートメント上の条件付け標識が満たされない場合 (ステップ 1 または 5)、あるいは索引値が限界値より大きい場合 (ステップ 3) に処理されます。

注: ループの中で索引、増分、限界値、および標識を変更して、DO グループの終わりに影響させることができます。

これらの演算命令が DO 演算命令にどのように影響するかの説明については、 579 ページの『LEAVE (DO/FOR グループの終了)』および 574 ページの『ITER (繰り返し)』 を参照してください。

初期値、増分値、および限界値に対する**フリー・フォームの式**を使用した反復ループの実行については、 563 ページの『FOR (For)』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  DO グループは、標識 17 がオンの時に 10 回処理されます。
C*  それは、結果フィールドであるフィールド X の索引値が
C*  演算項目 2 の限界値 (10) より大きくなった時に停止します。
C*  DO グループの実行が停止した時には、制御が ENDDO 演算命令
C*  の直後の演算命令に渡されます。DO 演算命令の演算項目 1 は
C*  指定されていないので、開始値は 1 です。
C*  ENDDO 演算命令の演算項目 2 は指定されていないので、
C*  増分値は 1 です。
C
C   17              DO        10            X                3 0
C                   :
C                   ENDDO
C*
C*  DO グループは 10 回処理することができます。DO グループの
C*  実行は、フィールド X の索引値が演算項目 2 の限界値 (20)
C*  より大きいか、ENDDO 演算命令に達した時に標識 50 がオンに
C*  なっていない場合に停止します。標識 50 がオンでない場合に
C*  は、ENDDO 演算命令は処理されません。したがって、制御は
C*  ENDDO 演算命令の後の演算命令に渡されます。
C*  DO 演算命令の演算項目 1 には 2 の開始値が指定され、
C*  ENDDO 演算命令の演算項目 2 には 2 の増分値が指定され
C*  ています。
C*
C     2             DO        20            X                3 0
C                   :
C                   :
C                   :
C   50              ENDDO     2
```

図 217. DO 演算命令

DOU (実行の終了)

フリー・フォーム構文	DOU{(MR)} *indicator-expression*

コード	演算項目 1	拡張演算項目 2
DOU (M/R)		indicator-expression

DOU 演算命令は、DOUxx 演算命令と類似のものです。DOU 演算命令は、少なくとも 1 回、可能であれば複数回実行したい演算命令のグループより前に置かれます。関連した ENDDO ステートメントがグループの終わりをマークします。これは、標識によって値が与えられる式 (*indicator-expression*) で論理条件が表される点が異なります。

DOU 演算命令によって制御される演算命令は、 (*indicator-expression*) の式が「真」になるまで実行されます。命令拡張 M および R の使用法については、 392 ページの『数値演算の精度の規則』を参照してください。

固定形式構文の場合、レベル標識および条件標識が有効です。演算項目 1 はブランクでなければなりません。拡張演算項目 2 には、評価する式が入れられます。

383 ページの『第 24 章 式』は、式の指定方法を説明しています。

361 ページの『比較命令』は、比較演算命令を指定する場合の規則を説明しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
   // 以下の例では、F3 キーが押されるまで DO ループが繰り返され
   // ます。
   dou *inkc;
      do_something();
   enddo;

   // 以下の DO ループは、*In01 がオンになるかまたは FIELD2 が
   // FIELD3 より大きくなるまで繰り返されます。
   dou *in01 or (Field2 > Field3);
      do_something_else ();
   enddo;

   // 以下のループは、X が配列内のエレメントの数より大きくなるまで
   // 繰り返されます。
   dou X > %elem (Array);
      Total = Total + Array(x);
      X = X + 1;
   enddo;
/END-FREE
```

図 218. DOU 演算命令

DOUxx (実行の終了)

フリー・フォーム構文	(使用できません。DOU 命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DOUxx	被比較数	被比較数				

DOUxx 演算命令は、少なくとも 1 回、可能であれば複数回実行したい演算命令のグループより前に置かれます。関連した ENDDO ステートメントがグループの終わりをマークします。DO グループおよび xx の意味の詳細については、 379 ページの『構造化プログラミング命令』を参照してください。

演算項目 1 および演算項目 2 には、リテラル、名前付き固定情報、フィールド名、テーブル名、配列エレメント、表意定数、またはデータ構造名が含まれていなければなりません。演算項目 1 と演算項目 2 は同じデータ・タイプでなければなりません。

DOUxx ステートメントには、関係 xx を指示します。より複雑な条件を指定する場合は、DOUxx ステートメントの直後に ANDxx または ORxx ステートメントを続けます。DO グループの中の演算命令は、1 回だけ処理され、演算項目 1 と演算項目 2 の間の関係が存在する間、あるいは結合された DOUxx、ANDxx、または ORxx演算命令によって指定された条件が存在する間はそのグループが繰り返されます。グループは、そのグループの開始時に条件が「真」でなくとも、常に少なくとも 1 回は処理されます。

DO グループは、DOUxx 演算命令それ自体に加えて、DOUxx および ENDDO ステートメント上の条件付け標識によって制御されます。DOUxx ステートメント上の条件付け標識は、DOUxx 演算命令を開始するかどうかを制御します。関連した ENDDO ステートメント上の条件付け標識によって、DO ループを途中で終了させることができます。

DOUxx 演算命令は、次のステップに従っています。

1. DOUxx ステートメント行上の条件付け標識が満たされた場合に、DOUxx 演算命令が処理されます (ステップ 2)。標識が満たされない場合には、関連した ENDDO ステートメントの後の、次に処理が可能な演算命令に制御が渡されます (ステップ 6)。
2. DOUxx 演算命令は、処理が可能な次の演算命令に制御を渡すことによって処理されます (ステップ 3)。 DOUxx 演算命令は、この時点では演算項目 1 と演算項目 2 の比較または指定された条件のテストは実行しません。
3. DO グループ内の個々の演算命令が処理されます。
4. ENDDO ステートメント上の条件付け標識が満たされない場合には、関連した ENDDO ステートメントの後の次の演算命令に制御が渡されます (ステップ 6)。それ以外の場合には、ENDDO 演算命令が処理されます (ステップ 5)。
5. ENDDO 演算命令は、DOUxx 演算命令の演算項目 1 と演算項目 2 を比較し、結合された演算命令によって指定された条件をテストすることによって処理されます。演算項目 1 と演算項目 2 の間の関係 xx が存在しているか、指定された条件が存在している場合には、DO グループは終了され、ENDDO ステートメン

トの後の次の演算命令に制御が渡されます (ステップ 6)。演算項目 1 と演算項目 2 の間の関係 xx が存在していないか、指定された条件が存在していない場合には、DO グループの中の演算命令が繰り返されます (ステップ 3)。

6. ENDDO ステートメントの後のステートメントは、DOUxx または ENDDO ステートメント上の条件付け標識が満たされない場合 (ステップ 1 または 4)、あるいは演算項目 1 と演算項目 2 の間の関係 xx または指定された条件がステップ 5 で存在していた場合に処理されます。

これらの演算命令が DOUxx 演算命令にどのように影響するかについては、579 ページの『LEAVE (DO/FOR グループの終了)』 および 574 ページの『ITER (繰り返し)』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  DOUEQ 演算命令は、DO グループの中の演算命令を少なくとも 1 回は
C*  実行します。
C
C     FLDA          DOUEQ     FLDB
C
C*
C*  ENDDO 演算命令では、FLDA が FLDB と等しいかどうかを判別するため
C*  のテストが実行されます。FLDA が FLDB と等しくない場合には、
C*  ENDDO 演算命令の前の演算命令が再度処理されます。このループの
C*  処理は、FLDA が FLDB と等しくなるまで続けられます。FLDA が
C*  FLDB と等しくなった時に、プログラムは ENDDO 演算命令の直後の
C*  演算命令に分岐します。
C
C                   SUB       1             FLDA
C                   ENDDO
C
C*
C*  結合された DOUEQ ANDEQ OREQ 演算命令は、DO グループの中の
C*  演算命令を少なくとも 1 回は処理します。
C
C     FLDA          DOUEQ     FLDB
C     FLDC          ANDEQ     FLDD
C     FLDE          OREQ      100
C
C*
C*  ENDDO 演算命令では、指定された条件 (FLDA が FLDB に等しく、
C*  FLDC が FLDD に等しい) が存在するかどうかを判別するための
C*  テストが処理されます。その条件が存在する場合は、プログラムは
C*  演算命令の直後の演算命令に分岐します。OREQ 条件を
C*  テストする必要はありません。DOUEQ および ANDEQ 条件が適合
C*  する場合は、FLDE は 100 に等しくなります。指定された条件が
C*  存在しない場合は、OREQ 条件がテストされます。OREQ 条件が
C*  適合する場合は、プログラムは ENDDO の直後の演算命令に分岐
C*  します。適合しない場合は、OREQ 演算命令の次の演算命令が処理
C*  されて、その後でプログラムは 2 番目の DOUEQ 演算命令から
C*  始まる条件テストを処理します。
C*  DOUEQ 条件も ANDEQ 条件も、OREQ 条件も適合しない場合には、
C*  OREQ 演算命令の次の演算命令が
C*  もう一度処理されます。
C
C                   SUB       1             FLDA
C                   ADD       1             FLDC
C                   ADD       5             FLDE
C                   ENDDO
```

図 219. DOUxx 演算命令

DOW (実行の時期)

フリー・フォーム構文	DOW{(MR)} *indicator-expression*

コード	演算項目 1	拡張演算項目 2
DOW (M/R)		indicator-expression

DOW 命令コードは、指定の条件が存在している場合に処理したい演算命令のグループより前に置かれます。関連した ENDDO ステートメントがグループの終わりをマークします。その機能は、DOWxx 命令コードの機能と類似しています。これは、標識によって値が与えられる式 (*indicator-expression*) で論理条件が表される点が異なります。DOW 演算命令によって制御される演算命令は、 *indicator-expression* の式が「真」になっている間実行されます。命令拡張 M および R の使用法については、 392 ページの『数値演算の精度の規則』を参照してください。

固定形式構文の場合、レベル標識および条件標識が有効です。演算項目 1 はブランクでなければなりません。拡張演算項目 2 には、評価する式が入れられます。

361 ページの『比較命令』は、比較演算命令を指定する場合の規則を説明しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* 以下の例では、条件が「偽」になるまで DO ループが繰り返されます。条件が「偽」に
* なるのは、A > 5 または B+C がゼロと等しくない時です。
/FREE
   dow (a <= 5) and (b + c = 0);
      do_something (a:b:c);
   enddo;
/END-FREE
```

図 220. DOW 演算命令

DOWxx (実行の時期)

フリー・フォーム構文	(使用できません。DOW 命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DOWxx	被比較数	被比較数				

DOWxx 命令コードは、指定の条件が存在している場合に処理したい演算命令のグループより前に置かれます。より複雑な条件を指定する場合は、DOWxx ステートメントの直後に ANDxx または ORxx ステートメントを続けます。関連した ENDDO ステートメントがグループの終わりをマークします。DO グループおよび xx の意味の詳細については、379 ページの『構造化プログラミング命令』を参照してください。

演算項目 1 および演算項目 2 には、リテラル、名前付き固定情報、表意定数、フィールド名、テーブル名、配列エレメント、またはデータ構造名が含まれていなければなりません。演算項目 1 と演算項目 2 は同じデータ・タイプでなければなりません。演算項目 1 と演算項目 2 の比較は、比較演算命令に示されているものと同じ規則に従います。361 ページの『比較命令』 を参照してください。

DO グループは、DOWxx 演算命令それ自体に加えて、DOWxx および ENDDO ステートメント上の条件付け標識によって制御されます。DOWxx ステートメント上の条件付け標識は、DOWxx 演算命令を開始するかどうかを制御します。関連した ENDDO ステートメント上の条件付け標識は、DOW グループをもう 1 回繰り返すかどうかを制御します。

DOWxx 演算命令は、次のステップに従っています。

1. DOWxx ステートメント行上の条件付け標識が満たされた場合に、DOWxx 演算命令が処理されます (ステップ 2)。標識が満たされない場合には、関連した ENDDO ステートメントの後の、次に処理される演算命令に制御が渡されます (ステップ 6)。
2. DOWxx 演算命令は、DOWxx 演算命令の演算項目 1 と演算項目 2 を比較するか、あるいは結合された DOWxx、ANDxx、または ORxx 演算命令によって指定された条件をテストすることによって処理されます。演算項目 1 と演算項目 2 の間の関係 xx または結合された演算命令によって指定された条件が存在していない場合には、DOグループは終了され、ENDDO ステートメントの後の次の演算命令に制御が渡されます (ステップ 6)。演算項目 1 と演算項目 2 の間の関係 xx または結合された演算命令によって指定された条件が存在している場合には、DO グループの中の演算命令が繰り返されます (ステップ 3)。
3. DO グループ内の個々の演算命令が処理されます。
4. ENDDO ステートメント上の条件付け標識が満たされない場合には、関連した ENDDO ステートメントの後に実行する次の演算命令に制御が渡されます (ステップ 6)。それ以外の場合には、ENDDO 演算命令が処理されます (ステップ 5)。
5. ENDDO 演算命令は、DOWxx 演算命令に制御を渡すことによって処理されます (ステップ 2)。 (DOWxxステートメント上の条件付け標識は、ステップ 1 で再度テストされないことに注意してください。)

6. ENDDO ステートメントの後のステートメントは、DOWxx または ENDDO ステートメント上の条件付け標識が満たされない場合 (ステップ 1 または 4)、あるいは演算項目 1 と演算項目 2 の間の関係 xx または指定された条件がステップ 2 で存在していない場合に処理されます。

これらの演算命令が DOWxx 演算命令にどのように影響するかについては、579 ページの『LEAVE (DO/FOR グループの終了)』 および 574 ページの『ITER (繰り返し)』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  DOWLT 演算命令によって、FLDA が FLDB より小さい場合にのみ
C*  DO グループ内の演算命令を処理することができます。FLDA が
C*  FLDB より小さくない場合には、プログラムは ENDDO 演算命令
C*  の直後の演算命令に分岐します。FLDA が FLDB より小さい場合
C*  には、DO グループ内の演算命令が処理されます。
C
C     FLDA          DOWLT     FLDB
C
C*
C*  ENDDO 演算命令によって、プログラムは FLDA が FLDB より小さ
C*  いかどうかを判別するためのテストが行なわれた最初の DOWLT
C*  演算命令に分岐します。このループの処理は、FLDA が FLDB
C*  より大きいか等しくなるまで続けられます。次に、プログラム
C*  は ENDDO 演算命令の直後の演算命令に分岐します。
C
C                   MULT      2.08          FLDA
C                   ENDDO
C
C*  次の例では、複数の条件がテストされます。結合された DOWLT ORLT
C*  演算命令によって、DO グループ内の演算命令を FLDA が FLDB また
C*  は FLDC より小さい間だけ処理することができます。指定された
C*  どちらの条件も存在しない場合には、プログラムは ENDDO 演算命令
C*  の直後の演算命令に分岐します。指定された条件のいずれかが
C*  存在している場合には、ORLT 演算命令の後の演算命令が処理され
C*  ます。
C
C     FLDA          DOWLT     FLDB
C     FLDA          ORLT      FLDC
C
C*  ENDDO 演算命令によって、プログラムは指定された条件が存在する
C*  かどうかをテストして判別する 2 番目の DOWLT 演算命令に分岐
C*  します。このループは、FLDA が FLDB および FLDC より大きいか
C*  等しくなるまで続けられます。次に、プログラムは ENDDO 演算命令
C*  の直後の演算命令に分岐します。
C
C                   MULT      2.08          FLDA
C                   ENDDO
```

図 221. DOWxx 演算命令

DSPLY (「メッセージ」ウィンドウの表示)

フリー・フォーム構文	DSPLY{(E)} {*message* {*message-window- definition-name* {*response*}}}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DSPLY (E)	メッセージ	message-window-definition-name	応答	_	ER	_

DSPLY 演算命令は、「メッセージ」ウィンドウを表示します。プログラムは停止し、「メッセージ」ウィンドウを表示して、応答を待ちます。

message オペランドは、以下の 1 つでなければなりません。
- フィールド名
- MSGNBR キーワードで定義されたフィールド名
- 文字リテラル、数値リテラル、16 進数リテラル、DBCS リテラル、日付リテラル、時刻リテラル、またはタイム・スタンプ・リテラル
- 定義仕様名
- イベント属性
- メッセージ ID

message オペランドはイベント属性とすることができますが、それは、該当するイベントのためのアクション・サブルーチンの中に DSPLY 演算命令がある (すなわち、イベントにイベント属性が指定されている) か、DSPLY 演算命令がイベントのためのアクション・サブルーチンによって実行されるユーザー・サブルーチンである場合です。制御仕様に EXE または NOMAIN キーワードが指定されている場合には、メッセージ・ボックス記述またはメッセージ ID をフィールド名として使用することはできません。

ポインター・フィールドは許されません。メッセージ番号を除き、*message* のすべてのデータが表示される前に文字に変換されます。

message-window-definition-name を指定する場合には、スタイルを定義する定義仕様名を含める必要があります。以下の場合には、*message-window-definition-name* は任意指定です。
- *message* がメッセージ ID である (*MSGnnnn)
- *message* が定義仕様名で、参照される定義仕様に MSGNBR キーワードが含まれている。 MSGNBR は、メッセージ番号か、メッセージ番号を含むフィールドのいずれかとすることができます。

制御仕様に EXE または NOMAIN キーワードが指定されている場合には、*message-window-definition-name* は無視されます。

response オペランドには、「メッセージ」ウィンドウでユーザーが押したボタンを表す値が入れられます。この値は、メッセージ・ボックスに表示されるボタン (たとえば、*RETRY) を定義するために使用された表意定数の 1 つと対応します。これらの固定情報は、ユーザーがどのボタンを押したかを検査するために使用される必要があります。*response* オペランドは、小数点以下の桁数のない 9 桁の長さの数値フィールドでなければなりません。

EXE または NOMAIN キーワードを指定した場合には、*response* オペランドを精度を 9,0 の数値または文字とすることができます。「メッセージ」ウィンドウの応答フィールドは、数値および文字フィールドと異なる動作を示します。数値応答フィールドは、次のような動作を示します。

- このフィールドで「実行」または「改行」キーを押すと、値 0 が戻されます。
- フィールドは 9 桁だけを受け入れます。9 桁を超えて入力した場合には、無視されます。
- 小数点を含む文字を入力すると、実行時エラーの原因となり、プログラムは終了されます。

文字フィールドは、次のような動作を示します。

- 「実行」または「改行」キーを押すと、フィールドにブランクが埋め込まれます。
- フィールドに入力された余分な文字は無視されます。フィールドが受け入れることができるのは 1 つまたは複数の語です。

NOMAIN キーワードを使用した場合に、Windows GUI アプリケーションによって呼び出されたプロシージャーは、以下のような動作を示します。

- 文字または DBCS フィールドの場合には、メッセージは表示されず、応答フィールドにはブランクが埋め込まれます。
- 数値フィールドには値 0 が埋め込まれます。

DSPLY の例外を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。演算命令でエラーが起こった場合には、指定されたメソッドによって例外が処理されます。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

「メッセージ」ウィンドウを定義するために、定義仕様で各種のキーワードが使用されます。 BUTTON、MSGTITLE、および STYLE キーワードは、ウィンドウ・スタイルを定義します。 MSGDATA、MSGNBR、および MSGTEXT キーワードは、ウィンドウに表示されるメッセージ・テキストを定義します。 265 ページの『定義仕様のキーワード』を参照してください

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D Box1            M                   STYLE(*WARN)  BUTTON(*RETRY:*ABORT:*IGNORE)
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C     *MSG9999      DSPLY     BOX1          REPLY             9 0
C                   IF        reply = *RETRY
 *   「再試行」ボタンが押されました。
C                   .....
C                   ELSE
C                   IF        reply = *ABORT
 *   「打ち切り」ボタンが押されました。
C                   .....
C                   ELSE
 *   「無視」ボタンが押されました。
C                   .....
C                   ENDIF
C                   ENDIF
```

図 222. DSPLY 演算命令

ELSE (その他)

フリー・フォーム構文	ELSE

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ELSE						

ELSE 演算命令は、IFxx および IF 演算命令の任意指定パーツです。IFxx の比較が満たされた場合には、ELSE の前の演算が処理されます。それ以外の場合には、ELSE の後の演算が処理されます。

条件付け標識の入力 (9 - 11 桁目) は許可されていません。

IFxx/ELSE グループをクローズする場合は、ENDIF 演算命令を使用してください。

571 ページの図 235 は、IFxx 演算命令と一緒の ELSE 演算命令の例を示しています。

ELSEIF (Else If)

フリー・フォーム構文	ELSEIF{(MR)} *indicator-expression*

コード	演算項目 1	拡張演算項目 2
ELSEIF (M/R)	ブランク	indicator-expression

ELSEIF 演算命令は、ELSE 演算命令と IF 演算命令の組み合わせです。これによりネストのレベルの追加が必要でなくなります。

IF 命令コードによって、条件が満たされた場合に一連の命令コードを処理できるようになります。その機能は、IFxx 命令コードの機能と類似しています。これは、標識によって値が与えられる式 (*indicator-expression*) で論理条件が表される点が異なります。ELSEIF 演算命令によって制御される演算命令は、*indicator-expression* オペランドの式が真である (さらに前の IF または ELSEIF ステートメントが偽であった) ときに実行されます。

命令拡張 M および R の使用法については、392 ページの『数値演算の精度の規則』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/free

   IF state = 0;
      dosomething();
   ELSEIF state = 1;
      return;
   ELSEIF state = 2;
      report(state);
   ELSE;
      signalError ('Bad state');
   ENDIF;

/end-free
```

図 223. ELSEIF 演算命令

ENDyy (構造化グループの終了)

フリー・フォーム構文	ENDDO
	ENDFOR
	ENDIF
	ENDMON
	ENDSL
	(END および ENDCS は使用できません)

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
END		**increment-value**				
ENDCS						
ENDDO		increment-value				
ENDFOR						
ENDIF						
ENDMON						
ENDSL						

ENDyy 演算命令は CASxx、DO、DOU、DOW、DOUxx、DOWxx、FOR、IF、IFxx、MONITOR、または SELECT 演算命令グループを終了します。

ENDyy 演算命令は CASxx、DO、DOU、DOW、DOUxx、DOWxx、FOR、IF、IFxx、または SELECT 演算命令グループを終了します。

ENDyy 演算命令は、以下にリストされています。

END CASxx、DO、DOU、DOUxx、DOW、DOWxx、FOR、IF、IFxx、または SELECT グループを終了します。

ENDCS CASxx グループの終了

ENDDO DO、DOU、DOUxx、DOW、または DOWxx グループの終了

ENDFOR FOR グループの終了

***制約事項:** Java アプリケーションでは、ENDFOR はサポートされていません。

ENDIF IF または IFxx グループの終了

ENDMON MONITOR グループを終了します。

ENDSL SELECT グループの終了

increment-value オペランドは、DO グループを区切る ENDyy 演算命令でのみ使用できます。これには、DO グループの増分値が入れられます。これは、正または負とすることができますが、小数点以下の桁数はゼロでなければなりません。また、配列エレメント、テーブル名、データ構造、フィールド、名前付き固定情報、または数値リテラルとすることができます。*increment-value* を指定しない場合には、増分はデフォルトで 1 になります。 *increment-value* が負数である場合には、DO グループは終了することがなくなります。

ENDDO または ENDFOR 演算命令には条件付け標識を指定することができます。これらは ENDCS、ENDIF、ENDMON、および ENDSL には使用できません。

演算結果標識は許されません。ENDCS、ENDIF、ENDMON、ENDSL に使用できるオペランドはありません。

1 つの ENDyy フォームを別の演算命令グループと一緒に (たとえば、ENDIF を構造化グループと一緒に) 使用した場合には、コンパイル時にエラーの結果となります。

詳細については、ENDyy 演算命令を使用する例について次を参照してください。

- 505 ページの『CASxx (サブルーチンの条件付き起動)』
- 535 ページの『DO (実行)』
- 538 ページの『DOUxx (実行の終了)』
- 542 ページの『DOWxx (実行の時期)』
- 570 ページの『IFxx (判断)』
- 537 ページの『DOU (実行の終了)』
- 541 ページの『DOW (実行の時期)』
- 563 ページの『FOR (For)』
- 569 ページの『IF (判断)』
- 585 ページの『MONITOR (モニター・グループの開始)』
- 664 ページの『SELECT (選択グループの始め)』

ENDACT (アクション・サブルーチンの終了)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ENDACT		戻り点				

ENDACT 演算命令は、アクション・サブルーチンの終了を定義します。ENDACT は、アクション・サブルーチン内の最後の演算命令でなければなりません。

演算項目 2 を指定する場合には、次の 1 つを入れる必要があります。

***DEFAULT**
現行アクション・サブルーチンは終了し、現行イベントのためのデフォルトの処理が実行されます。

***NODEFAULT**
現行アクション・サブルーチンは終了し、現行イベントのためのデフォルトの処理は実行されません。

フィールド名
フィールド名は、12 文字でなければなりませんが、*DEFAULT または *NODEFAULTを入れることができます。フィールドに無効な値が含まれていた場合には、デフォルト・エラー処理プログラムが制御を受け取ります。

演算項目 2 が指定されない場合には、現行アクション・サブルーチンは終了し、現行イベントのためのデフォルトの処理が実行されます。

処理が ENDACT 演算命令に達した時に、LR がオンになっていれば、コンポーネントが終了されます。 *DEFAULT および *NODEFAULT は無視されます。アクション・サブルーチンがネストされていた場合には、LR は検査されず、*DEFAULT および *NODEFAULT は無視されます。

条件付け標識の入力は許されません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                            Extended-factor2++++++++++++++++++++++++++++
C*
C                   ENDACT    '*DEFAULT'
```

図 224. ENDACT 演算命令

ENDSR (ユーザー・サブルーチンの終了)

フリー・フォーム構文	ENDSR {*return-point*}

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
ENDSR	ラベル	戻りポイント				

ENDSR 演算命令は、ユーザー・サブルーチンの終了を定義します。これによって、EXSR 演算命令の次のステートメントに戻ります。ENDSR は、サブルーチン内の最後の演算命令でなければなりません。

固定形式では、*label* オペランドは、サブルーチン内での GOTO 演算命令の分岐先として指定できます。(フリー・フォーム構文では *label* は指定できません。)

(*return-point*) オペランドは、*PSSR または *INFSR サブルーチンの終わりでのみ指定することができます。これには、次の 1 つが含まれていなければなりません。

***CANCL**
エラーが起こった時に実行されていたアクション・サブルーチンは終了し、コンポーネントは異常終了します。

***ENDCOMP**
エラーが起こった時に実行されていたアクション・サブルーチンは終了し、コンポーネントは異常終了します。

***DEFAULT**
現行アクション・サブルーチンから制御が戻され、現行イベントのためのデフォルトの処理が実行されます。LR がオンである場合には、コンポーネントは正常に終了します。LR がオンでない場合には、アクション・サブルーチンは終了し、イベントのためのすべてのデフォルト・アクションが実行されます。

***NODEFAULT**
現行アクション・サブルーチンから制御が戻され、現行イベントのためのデフォルトの処理は実行されません。LR がオンである場合には、コンポーネントは正常に終了します。LR がオンでない場合には、アクション・サブルーチンは終了し、イベントのためのどのデフォルト・アクションも実行されません。

***ENDAPPL**
エラーが起こった時に実行されていたアクション・サブルーチンは終了し、現在アクティブになっているすべてのコンポーネントが低い階層から順に終了します。エラーが起こった時にアクティブであったコンポーネントは、正常に終了し、他のすべてのコンポーネントも正常に終了します。

フィールド名
フィールド名には、*CANCL、*ENDCOMP、*DEFAULT、*NODEFAULT、または *ENDAPPL を入れることができます。フィールドに無効な値が含まれていた場合には、デフォルト・エラー処理プログラムが制御を受け取ります。

条件付け標識は許されません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                             Extended-factor2++++++++++++++++++++++++++++
C*
C     Label         BEGSR
C                     .
C                     .
C                     .
C                   ENDSR     '*ENDCOMP'
```

図 225. ENDSR 演算命令

EVAL (式の評価)

フリー・フォーム構文	{EVAL{(HMR)}} *result = expression*

コード	演算項目 1	拡張演算項目 2
EVAL (H/M/R)		割り当てステートメント

EVAL 命令コードは、**結果 = 式**の形式の割り当てステートメントを評価します。式が評価され、その結果が **result** に入れられます。したがって、**result** をリテラルまたは固定情報とすることはできませんが、フィールド名、配列名、配列エレメント、データ構造、データ構造サブフィールド、あるいは %SUBST 組み込み関数を使用したストリングでなければなりません。

式はどんな RPG データ・タイプでも生じます。式のタイプは結果のタイプと同じでなければなりません。文字、グラフィック、または UCS-2 の結果は、左寄せされ、必要に応じてブランクが埋め込まれるかまたは切り捨てられます。**result** が可変長フィールドである場合には、その長さは、式の結果の長さに設定されます。

結果が索引付きでない配列または配列 (*) として指定された配列を表している場合には、 186 ページの『演算での配列の指定』で説明されている規則に従って、式の値が結果の各エレメントに割り当てられます。それ以外の場合には、式は 1 回だけ評価され、値が配列または副配列の各エレメントに入れられます。数式の場合には、四捨五入命令コード拡張が許されています。四捨五入の規則は、算術演算の場合の規則と同等です。

フリー・フォーム演算仕様では、拡張が必要なければ、命令コード名を除外できます。

式の一般情報については、 383 ページの『第 24 章 式』を参照してください。数式の精度規則については、 392 ページの『数値演算の精度の規則』を参照してください。これは、式に除算演算命令が含まれている場合、または EVALにいずれかの命令拡張が使用される場合にとくに重要です。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*                   Assume FIELD1 = 10
*                          FIELD2 =  9
*                          FIELD3 =  8
*                          FIELD4 =  7
*                          ARR は DIM(10) で定義されます。
*                          *IN01 = *ON
*                          A = 'abcdefghijklmno' (15 桁の長さとして定義)
*                          CHARFIELD1 = 'There'  (5 桁の長さとして定義)

/FREE
  //  演算命令の後の RESULT の内容は 20 となります。
  eval RESULT=FIELD1 + FIELD2+(FIELD3-FIELD4);

  //  標識 *IN03 が *ON に設定されます。
  *IN03 = *IN01 OR (FIELD2 > FIELD3);

  //  配列 ARR の各エレメントに値 72 が割り当てられます。
  ARR(*) = FIELD2 * FIELD3;

  //  演算命令の後は、A = 'Hello There   ' の内容となります。
  A = 'Hello ' + CHARFIELD1;

  //  演算命令の後は A = 'HelloThere    ' の内容となります。
  A = %TRIMR('Hello ') + %TRIML(CHARFIELD1);

  //  割り当ての日付
  ISODATE = DMYDATE;

  //  関係式
  //  演算命令の後の値は *IN03 = *ON となります。
  *IN03 = FIELD3 < FIELD2;

  // 関係式の日付
  // 演算命令の後、Date1 が Date2 の日付より後の日付を表している
  // 場合には、*IN05 は *ON に設定されます。
  *IN05 = Date1 > Date2;
  // EVAL の後の A の元の値には 'ab****ghijklmno' が含まれます。
  %SUBST(A(3:4))= '****';

  // EVAL の後の PTR には変数 CHARFIELD1 のアドレスが含まれます。
  PTR = %ADDR(CHARFIELD1);

  //  論理式の結果に文字データ・タイプとの互換性があることを
  //  示す例。
  //  以下の EVAL ステートメントは、3 つの論理式から構成され、
  //  その結果は '+' 演算子を使用して連結されます。
  //  文字フィールド RES の結果の値は '010' となります。
  RES = (FIELD1<10) + *in01 + (field2 >= 17);

  // EVAL を使用してユーザー定義関数を呼び出す例。
  // プロシージャー FormatDate は、日付フィールドを文字
  // ストリングに変換し、そのストリングを戻します。この
  //  EVAL ステートメントでは、フィールド DateStrng1 に
  // formatdate の出力が割り当てられています。
  DateStrng1 = FormatDate(Date1);

/END-FREE
```

図 226. EVAL 演算命令

EVALR (式の評価、右寄せ)

フリー・フォーム構文	EVALR{(MR)} *result* = *expression*

コード	演算項目 1	拡張演算項目 2
EVALR (M/R)		割り当てステートメント

EVALR 命令コードは、結果 = 式のフォームの割り当てステートメントを評価します。式が評価され、その結果が右寄せして「結果」に入れられます。したがって、この結果をリテラルまたは固定情報とすることはできませんが、固定長文字、グラフィック、または UCS-2 フィールド名、配列名、配列エレメント、データ構造、データ構造サブフィールド、あるいは %SUBST 組み込み関数を使用したストリングでなければなりません。式のタイプは結果のタイプと同じでなければなりません。結果は右寄せされ、必要に応じて左側にブランクが埋め込まれるか、または左側が切り捨てられます。

注:

1. EVAL 演算命令と異なり、EVALR の結果は文字、グラフィック、または UCS-2 のタイプにのみすることができます。さらに、%SUBST 組み込み関数が式の左辺を構成する場合はその組み込み関数に可変長フィールドを含めることができますが、固定長の結果フィールドだけが許されています。
2. %SETATR または %GETATR 組み込みで使用される EVALR は、EVAL 演算命令のように動作します。設定時または検索時には、属性値の右そろえは行われません。

結果が索引付きでない配列または配列 (*) として指定された配列を表している場合には、 186 ページの『演算での配列の指定』で説明されている規則に従って、式の値が結果の各エレメントに割り当てられます。それ以外の場合には、式は 1 回だけ評価され、値が配列または副配列の各エレメントに入れられます。

式の一般情報については、 383 ページの『第 24 章 式』を参照してください。数式の精度規則については、 392 ページの『数値演算の精度の規則』を参照してください。これは、式に除算演算命令が含まれている場合、または EVALRにいずれかの命令拡張が使用される場合にとくに重要です。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D Name            S             20A

 /FREE
    eval  Name = 'Kurt Weill';
    // 名前はここで 'Kurt Weill          ' になります
    evalr Name = 'Johann Strauss';
    // 名前はここで '      Johann Strauss' になります
    evalr %SUBST(Name:1:12) = 'Richard';
    // 名前はここで '     Richard Strauss' になります
    eval  Name = 'Wolfgang Amadeus Mozart';
    // 名前はここで 'Wolfgang Amadeus Moz' になります
    evalr Name = 'Wolfgang Amadeus Mozart';
    // 名前はここで 'fgang Amadeus Mozart' になります
 /END-FREE
```

図 227. EVALR 演算命令

EXCEPT (演算時の出力)

フリー・フォーム構文	EXCEPT {*except-name*}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
EXCEPT		例外レコード名				

EXCEPT 演算命令によって、演算時に 1 つまたは複数の例外レコードを書き込むことができます。レコードが書き込まれるファイルは、ローカル・ファイルまたは OS/400 ファイルにすることができます。

演算時に書き込まれる例外レコードは、出力仕様の (17 桁目) の E によって指示されます。*except-name* オペランドは、例外レコードの出力仕様 (30 ～ 39 桁目) の EXCEPT 名と同じ名前でなければなりません。

except-name オペランドが指定された場合には、それと同じ EXCEPT 名をもつ例外レコードだけ検査され、条件付け標識が満たされた場合に書き込まれます。*except-name* が指定されない場合は、出力仕様 (30 ～ 39桁目) にある例外レコードだけが検査され、条件付け標識が満たされた場合に書き込まれます。

フィールドを含まない形式となるように例外出力が指定された場合には、次のことが起こります。

- 出力ファイルが指定された場合は、レコードにデフォルト値が書き込まれます。
- レコードがロックされている場合には、システムはその演算命令をレコードをロック解除するための要求として取り扱います。これは、ロック解除を要求する代替的なフォームです。望ましいメソッドは、UNLOCK 演算命令を使用することです。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  演算項目 2 に HDG を指定した EXCEPT 演算命令が処理された場合に
C*  は、HDG の EXCEPT 名をもつすべての例外レコードが書き込まれます。
C*  次の例では、UDATE および PAGE が印刷された後に、印刷装置は
C*  2 行のスペースを送ることになります。
C*  2 番目の HDG レコードではドットの行が印刷された後に、印刷装置
C*  は 3 行のスペースを送ることになります。
C                   EXCEPT    HDG
C*
C*  演算項目 2 に指定のない EXCEPT 演算命令が処理されると、条件
C*  標識が満たされた場合に、30 から 39 桁目に EXCEPT 名が指定さ
C*  れていないすべての例外レコードが書き込まれます。条件標識が
C*  なく、EXCEPT 名のないどの例外レコードも、常に演算項目 2 に
C*  指定のない EXCEPT 演算によって書き込まれます。この例では、
C*  標識 10 がオンであると、TITLE および AUTH が印刷され、
C*  プリンターは 1 行行送りされます。
C                   EXCEPT
O*
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++
O          E    10                     1
O                       TITLE
O                       AUTH
O          E            HDG            2
O                       UDATE
O                       PAGE
O          E            HDG            3
O                                              '................'
O                                              '................'
O          E            DETAIL         1
O                       AUTH
O                       VERSNO
```

図 228. EXCEPT 演算命令

EXSR (ユーザー・サブルーチンの起動)

フリー・フォーム構文	EXSR *subroutine-name*

コード	演算項目 1	拡張演算項目 2
EXSR		subroutine-name

EXSR 演算命令は、*subroutine-name* オペランドに名前を指定されたユーザー・サブルーチンが処理されるようにします。ユーザー・サブルーチン名は、固有の記号名でなければならず、BEGSR 演算命令の *subroutine-name* オペランドとして現れている必要があります。EXSR 演算命令は、演算仕様の任意の位置に表すことができます。 ユーザー・サブルーチンが ENDSR 演算命令の *return-point* オペランドに項目がある例外 / エラー・サブルーチンである場合には、EXSR の後のステートメントは処理されません。

subroutine-name オペランドは、固有の記号名、あるいはキーワード *TERMSR、*PSSR、または *INZSR でなければなりません。
- *TERMSR は、正常終了サブルーチンを処理することを指定します。
- *PSSR は、プログラム例外 / エラー・サブルーチンを処理することを指定します。
- *INZSR は、初期化サブルーチンを処理することを指定します。

アクション・サブルーチンを処理するために EXSR 演算命令を使用することはできません。

ユーザー・サブルーチンのコーディング

ユーザー・サブルーチンは、演算命令のどこの点でも処理することができます。すべての演算命令をユーザー・サブルーチン内および固定形式で処理できる場合には、それらの演算命令は、9 ～ 11 桁目の任意の有効な標識によって条件付けすることができます。7 - 8 桁目には、SR またはブランクを表すことができます。ユーザー・サブルーチン内の AND/OR 行は、7 - 8 桁目で指示することができます。

ユーザー・サブルーチンで使用されるフィールドは、ユーザー・サブルーチンかプログラムの別のパーツのいずれかで定義することができます。いずれの場合にも、これらのフィールドをメイン・プログラムとユーザー・サブルーチンの両方で使用することができます。

ユーザー・サブルーチンに別のユーザー・サブルーチンを含めることはできません。1 つのユーザー・サブルーチンで別のユーザー・サブルーチンを呼び出すことができます。すなわち、サブルーチンの中に EXSRまたは CASxx 演算命令を含めることができます。ただし、ユーザー・サブルーチンの中の EXSR または CASxx 演算命令それ自体を直接呼び出すことはできません。予測のつかない結果が起こるので、別のサブルーチンを通じてのそのサブルーチン自体の呼び出しは実行しないようにしてください。同じサブルーチン内の別の点に分岐したい場合には、GOTO および TAG 命令コードを使用してください。

サブルーチンは、それらが使用される順に指定する必要はありません。各サブルーチンは、固有の記号名をもち、BEGSR および ENDSR 演算命令を含んでいることが必要です。

サブルーチンの中では GOTO 演算命令の使用が許されています。GOTO では、そのサブルーチンと関連した ENDSR 演算命令のラベルを指定することができますが、BEGSR 演算命令の名前を指定することはできません。 GOTO が TAG または ENDSR と同じサブルーチンの中になければ、その同じサブルーチン内の TAG または ENDSR に対して GOTO を出すことはできません。LEAVESR 演算命令を使用して、サブルーチン内の任意の点でサブルーチンを終了することができます。制御は、そのサブルーチンのための ENDSR演算命令に渡されます。 LEAVESR は、サブルーチンの中からのみ使用してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*
C                         :
C                         :
C                   EXSR      SUBRTB
C                         :
C                         :
C                   EXSR      SUBRTA
C                         :
C                         :
C     SUBRTA        BEGSR
C                         :
C                         :
C*
C*  1 つのサブルーチンで別のサブルーチンを呼び出すことができます。
C*
C                   EXSR      SUBRTC
C                         :
C                         :
C                   ENDSR
C     SUBRTB        BEGSR
C                         :
C                         :
C*
```

図 229. ユーザー・サブルーチンのコーディングの例 - BEGSR および EXSR の使用

EXTRCT (日付 / 時刻 / タイム・スタンプの抽出)

フリー・フォーム構文	(使用できません。%SUBDT 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
EXTRCT (E)		日付 / 時刻: 期間コード	ターゲット	_	ER	_

EXTRCT 演算命令は、次の 1 つを結果フィールドに戻します。
- 日付またはタイム・スタンプ・フィールドの年、月、または日パーツ
- 時刻またはタイム・スタンプ・フィールドの時、分、または秒パーツ
- タイム・スタンプ・フィールドのマイクロ秒パーツ

演算項目 2 は、フィールド、サブフィールド、テーブル・エレメント、または配列エレメントでなければなりません。日付、時刻、またはタイム・スタンプに続けて期間コードを指定する必要があります。期間コードのリストについては、363 ページの『日付の演算』を参照してください。

演算項目 1 はブランクでなければなりません。

結果フィールドは、数値または文字フィールド、サブフィールド、テーブル・エレメント、あるいは配列エレメントでなければなりません。文字データは、結果フィールドの中で左寄せされます。

EXTRCT 演算命令をユリウス日付 (形式 *JUL) と一緒に使用した場合に、*D の期間コードを指定すると月のうちの日が戻され、*M を指定すると年のうちの月が戻されます。日と月を 3 桁の形式とすることが必要な場合には、基底ポインターを使用してそれを入手することができます。

演算結果標識が 73 - 74 桁目に指定された場合には、EXTRCT 演算命令の実行中にエラーが起こった時にオンに設定されます。

EXTRCT の例外 (プログラム状況コード 112) を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C* タイム・スタンプ・フィールドの月を、月の名前を含む文字配列の
C* 索引として使用される 2 桁のフィールドに抽出します。次に、
C* タイム・スタンプの日を、EVAL 連結式で使用して、たとえば、
C* "March 13" を含むストリングを形成できる 2 バイトの文字
C* フィールドに抽出します。
C*
C
C                   EXTRCT    LOGONTIME:*M  LOGMONTH          2 0
C                   EXTRCT    LOGONTIME:*D  LOGDAY            2
C                   EVAL      DATE_STR = %TRIMR(MONTHS(LOGMONTH)
C                                        + ' ' + LOGDAY
```

図 230. EXTRCT 演算命令

FEOD (データの終わりの強制)

フリー・フォーム構文	FEOD{(E)} *file-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
FEOD (E)		ファイル名		_	ER	_

FEOD 演算命令は、OS/400 ファイルのデータの論理的な終わりを示す信号を出します。このファイルは、 OPEN 演算命令を指定しないで後続のファイル演算命令に再び使用することができます。ファイルは、プログラムに接続されたままです。これは、ファイルがプログラムから切断されて、そのファイルを再び使用したい場合には OPEN を指定する必要がある CLOSE 演算命令とは異なります。詳細については、 523 ページの『CLOSE (ファイルのクローズ)』を参照してください。

FEOD 演算命令を使用できるのは、OS/400 ファイルだけです。

file-name オペランドは、FEOD が指定されるファイルの名前を指定します。

FEOD の例外 (1000 より大きいファイル状況コード) を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、 41 ページの『ファイルの例外 / エラー』を参照してください。

FEOD 演算命令の後にさらに順次演算命令 (たとえば、READ または READP の使用) を処理する場合は、ファイルの位置変更が必要です。

FEOD 演算命令では、バッファーに入れられたすべてのデータが出力ファイル用にフラッシュされます。データは、ディスクまたは印刷装置に書き込まれます。

FOR (For)

フリー・フォーム構文	FOR{(MR)} *index-name* {= *start-value*} {BY *increment*} {TO|DOWNTO *limit*}

コード	演算項目 1	拡張演算項目 2
FOR		index-name = start-value BY increment TO ¦ DOWNTO limit

***制約事項:** Java アプリケーションでは、FOR 演算命令はサポートされていません。

FOR 演算命令は、1 つのグループの演算命令を開始し、そのグループが処理される回数を制御します。演算命令のグループが処理される回数を指示する場合は、索引名、開始値、増分値、および限界値を指定します。任意指定の開始、増分、および限界値は、フリー・フォームの式とすることができます。関連した END または ENDFOR ステートメントがグループの終わりをマークします。FOR グループの詳細については、379 ページの『構造化プログラミング命令』を参照してください。

FOR 演算命令の構文は、次の通りです。

```
FOR         index-name { = starting-value }
            { BY increment-value }
            { TO | DOWNTO limit-value }
   { loop body }
ENDFOR | END
```

index-name は、小数点以下の桁数がゼロのスカラー数値変数の名前でなければなりません。これを索引付き配列とすることはできません。
starting-value、increment-value、および limit-value は、小数点以下の桁数がゼロの数値または式とすることができます。増分値を指定する場合には、ゼロにすることはできません。

BY および TO (または DOWNTO) 文節は、どちらを先に指定してもかまいません。″BY 2 TO 10″ と ″TO 10 BY 2″ の両方を使用することができます。

FOR グループは、FOR 演算命令それ自体に加えて、FOR および ENDFOR (または END) ステートメント上の条件付け標識によって制御されます。FOR ステートメント上の条件付け標識は、FOR 演算命令を開始するかどうかを制御します。これらの標識は、FOR ループの開始時に一度だけ検査されます。関連した ENDまたは ENDFOR ステートメント上の条件付け標識は、FOR グループをもう 1 回繰り返すかどうかを制御します。これらの標識は、各ループの終わりで検査されます。

FOR 演算命令は、次のように実行されます。

1. FOR ステートメント行上の条件付け標識が満たされた場合に、FOR 演算命令が処理されます (ステップ 2)。標識が満たされない場合には、関連した END または ENDFOR ステートメントの後の、次に処理される演算命令に制御が渡されます (ステップ 8)。
2. 初期値を指定した場合には、それが索引名に割り当てられます。指定しない場合には、索引名はループを開始する前と同じ値のままです。
3. 限界値を指定した場合には、それが評価され、索引名と比較されます。限界値が指定されない場合には、ループを終了する (LEAVE または GOTO など) か、あ

るいはプログラムまたはプロシージャーを終了する (RETURN など) ステートメントが見つかるまで、ループは無限に反復されます。

TO 文節が指定され、索引名の値が限界値より大きい場合には、ENDFOR ステートメントの後の最初のステートメントに制御が渡されます。DOWNTO が指定され、索引名が限界値より小さい場合には、ENDFOR の後の最初のステートメントに制御が渡されます。

4. FOR グループの中の演算命令が処理されます。
5. END または ENDFOR ステートメント上の条件付け標識が満たされない場合には、関連した END または ENDFOR ステートメントの後のステートメントに制御が渡され、ループは終了します。
6. 増分値を指定した場合には、それが評価されます。指定しない場合には、デフォルトの 1 となります。
7. 増分値は、索引名に加算されるか (TO の場合)、索引名から減算されます (DOWNTO の場合)。制御はステップ 3 に渡されます。(制御がステップ 3 に渡された時に、FOR ステートメント上の条件付け標識は再度テスト (ステップ 1) されないことに注意してください。)
8. END または ENDFOR ステートメントの後のステートメントは、FOR、END、または ENDFOR ステートメント上の条件付け標識が満たされない場合 (ステップ 1 または 5)、索引値が限界値より大きいか (TO の場合)または小さい (DOWNTO の場合) 場合 (ステップ 3)、あるいは索引値がオーバーフローした場合に処理されます。

注: FOR ループが *n* 回実行された場合には、限界値は *n+1* 回として評価され、増分値は *n* 回として評価されます。これは、限界値または増分値が複雑で評価に時間がかかる場合、あるいは限界値または増分値に副次効果をもつサブプロシージャーに対する呼び出しが含まれていた場合には、重要となる可能性があります。限界または増分の複数回の評価が必要でない場合には、FOR ループの前に値を一時的に計算し、FORループの中ではその一時的な値を使用してください。

FOR 演算命令を指定する場合には、次の点に留意してください。

- 索引名を FOR 演算命令で宣言することはできません。D 仕様の中で変数を宣言する必要があります。
- 索引付き配列エレメントを FOR 演算命令の中で索引フィールドとすることは許されません。

これらの演算命令が FOR 演算命令にどのように影響するかについては、579 ページの『LEAVE (DO/FOR グループの終了)』および 574 ページの『ITER (繰り返し)』 を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/free
   // 例 1
   // n! を計算します。

   factorial = 1;
   for i = 1 to n;
      factorial = factorial * i;
   endfor;

   // 例 2
   // フィールド中の最後の非ブランク文字を検索します。
   // フィールドが全桁ブランクの場合には、"i" はゼロになります。
   // そうでない場合には、"i" は非ブランクの位置をなります。

   for i = %len (field) downto 1;
      if %subst(field: i: 1) <> ' ';
         leave;
      endif;
   endfor;

   // 例 3
   // ブランクで区切られたすべての語を文から抽出します。

   WordCnt = 0;
   for i = 1 by WordIncr to %len (Sentence);
      // ブランクがあるかどうか ?
      if %subst(Sentence: i: 1) = ' ';
         WordIncr = 1;
         iter;
      endif;

      // 語が見つかりました。 その長さを判別してください。
      for j = i+1 to %len(Sentence);
         if %subst (Sentence: j: 1) = ' ';
            leave;
         endif;
      endfor;

      // 語を保管します。
      WordIncr = j - i;
      WordCnt = WordCnt + 1;
      Word (WordCnt) = %subst (Sentence: i: WordIncr);
   endfor;

/end-free
```

図 231. FOR 演算命令の例

GETATR (属性の検索)

フリー・フォーム構文	(使用できません。%GETATR 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
GETATR (E)	パーツ名	属性	フィールド名	_	ER	_

GETATR 演算命令は、パーツの属性の値を検索します。親ウィンドウの名前がデフォルトのウィンドウ名となります。パーツの属性は、そのパーツが作成されている場合にのみ検索することができます。

注:

1. GETATR 演算命令は、複数リンクのアクション・サブルーチンに使用することができます。複数リンク・アクション・サブルーチンの説明については、 488 ページの『BEGACT (アクション・サブルーチンの開始)』 を参照してください。親ウィンドウ以外のウィンドウ上のパーツの属性を検索する場合は、%GETATR 組み込み関数を使用してください。 %GETATR 組み込み関数については、 434 ページの『%GETATR (属性の検索)』 を参照してください。
2. GETATR は、1 バイトおよび 8 バイトの符号付きおよび符号なし整数値とユニコード値はサポートしません。

演算項目 1 には、パーツの名前 (文字リテラルでなければなりません) またはパーツの名前を含むフィールド名 (文字でなければなりません) が含まれている必要があります。

演算項目 2 には、検索される属性の名前 (文字リテラルでなければなりません) または属性の名前を含むフィールド名 (文字でなければなりません) が含まれている必要があります。

演算項目 1 および演算項目 2 にグラフィック文字を含めることはできません。

結果フィールドには、検索された属性の値を入れるフィールドの名前が含まれていることが必要です。結果フィールドのタイプは、属性タイプと同じでなければなりません。

GETATR の例外を処理するには、命令コード拡張 'E' かエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、 52 ページの『プログラム例外およびエラー』を参照してください。

注: GETATR 演算命令は、パーツと対応するプログラム・フィールドには影響しません。パーツと対応するプログラム・フィールドに入力フィールドの現在の値を入れたい場合には、そのフィールドを演算命令のターゲットとしてください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                            Extended-factor2++++++++++++++++++++++++++++
C*
C* MLE01 と呼ばれるパーツのパーツ・タイプを検索します。
C*
C     'MLE01'       GETATR    'PartType'    Mle
```

図 232. GETATR 命令コード

GOTO (行先指定)

フリー・フォーム構文	(使用できません。LEAVE、ITER、RETURN などの他の命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
GOTO		ラベル				

GOTO 演算命令により、プログラム内の指定されたラベルに進む (または分岐する) ようにプログラムに指示することによって、演算命令をスキップすることができます。GOTO 演算命令の宛先の名前は、TAG 演算命令で指定されます。TAG は、GOTO の前または後に入れることができます。

メイン・プロシージャーでサブルーチン内で GOTO は、同一サブルーチン内の TAG に対して発行することができます。サブプロシージャーでサブルーチン内で GOTO は、同一サブルーチン内、あるいはサブプロシージャーの本体内の TAG に対して発行することができます。

演算項目 2 には、プログラムの分岐先のラベルが含まれていることが必要です。このラベルは、TAG または ENDSR 演算命令の演算項目 1 に入力されます。ラベルは固有の記号名でなければなりません。

TAG 演算命令の説明については、689 ページの『TAG (タグ)』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  標識 10、15、または 20 がオンである場合に、プログラムは GOTO 演算命令
C*  に指定された TAG ラベルに分岐します。
C*
C   10              GOTO      RTN1
C*
C*
C   15              GOTO      RTN2
C*
C     RTN1          TAG
C*
C                     :
C                     :
C   20              GOTO      END
C*
C                     :
C                     :
C     END           TAG
```

図 233. GOTO および TAG 演算命令

IF (判断)

フリー・フォーム構文	IF{(MR)} *indicator-expression*

コード	演算項目 1	拡張演算項目 2
IF (M/R)	ブランク	indicator-expression

IF 演算命令によって、条件が満たされた場合に一連の命令コードを処理することができます。その機能は、IFxx 命令コードの機能と類似しています。これは、標識によって値が与えられる式 (*indicator-expression*) で論理条件が表される点が異なります。IF 演算命令によって制御される演算命令は、 *indicator-expression* の式が「真」になると実行されます。

命令拡張 M および R の使用法については、 392 ページの『数値演算の精度の規則』を参照してください。 361 ページの『比較命令』は、比較演算命令を指定する場合の規則を説明しています。

```
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++..
C                           Extended-factor2-continuation+++++++++++++++++
C* IF 演算命令によって制御される演算命令は、式が「真」である場合
C* に実行されます。すなわち、次の A が 10 より大きく、標識 20 が
C* オンになっている場合です。
C
C                   IF      A>10 AND *IN(20)
C                   :
C                   ENDIF
C*
C* IF 演算命令によって制御される演算命令は、Date1 が
C* Date2 より後の日付を表している場合に実行されます。
C
C                   IF      Date1 > Date2
C                   :
C                   ENDIF
C*
```

図 234. IF 演算命令

IFxx (判断)

フリー・フォーム構文	(使用できません。IF 命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
IFxx	被比較数	被比較数				

IFxx 命令コードにより、xx によって指定された特定の関係が演算項目 1 と演算項目 2 の間に存在している場合に演算のグループを処理することができます。IFxx と一緒に ANDxx および ORxx 演算命令を使用した場合には、結合された演算命令によって指定された条件が存在している場合に演算のグループが実行されます。(xx の意味については、 379 ページの『構造化プログラミング命令』を参照してください。)

演算項目 1 と演算項目 2 には、リテラル、名前付き固定情報、表意定数、テーブル名、配列エレメント、データ構造名、またはフィールド名が入っていなければなりません。演算項目 1 と演算項目 2 の両方の項目が同じデータ・タイプでなければなりません。

IFxx および関連した任意の ANDxx または ORxx 演算命令によって指定された関係が存在していない場合には、関連した ENDIF 演算命令の直後の演算命令に制御が渡されます。ELSE 演算命令も指定されている場合には、ELSE 演算命令の後の処理が可能な最初の演算命令に制御が渡されます。

IFxx と関連した ENDIF 演算命令上の条件付け標識項目はブランクでなければなりません。

IFxx グループのクローズには、ENDIF ステートメントを使用することが必要です。IFxx ステートメントの後に ELSE ステートメントが続いている場合には、ELSE ステートメントの後に ENDIF ステートメントが必要ですが、IFxx ステートメントの後には不要です。

コンパイラー・リストの中で読みやすくするために、DO ステートメント、IF-ELSE 文節、および SELECT-WHENxx-OTHER 文節を字下げするオプションがあります。コンパイラー・オプションの説明については、GUI Designer の「プロジェクト」>「ビルド・オプション」ダイアログに関するオンライン・ヘルプを参照してください。

361 ページの『比較命令』は、比較演算命令を指定する場合の規則を説明しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  FLDA が FLDB と等しい場合に、IFEQ 演算命令の後の演算が処理
C*  されます。FLDA が FLDB と等しくない場合には、プログラムは
C*  ENDIF の直後の演算命令に分岐します。
C
C     FLDA          IFEQ      FLDB
C                       :
C                       :
C                   ENDIF
C
C*  FLDA が FLDB と等しい場合には、IFEQ 演算命令の後の演算が処理
C*  され、ENDIF 演算命令の直後の演算命令に制御が渡されます。
C*  FLDA が FLDB と等しくない場合には、ELSE ステートメントに制御
C*  が渡され、その直後の演算が処理されます。
C*
C
C     FLDA          IFEQ      FLDB
C                       :
C                       :
C                   ELSE
C                       :
C                       :
C                   ENDIF
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  FLDA が FLDB と等しく、FLDC より大きいか、FLDD が FLDE
C*  と等しく、FLDF より大きい場合に、ANDGT 演算命令の後の
C*  演算が処理されます。指定されたどちらの条件も存在して
C*  いない場合には、プログラムは ENDIF ステートメントの
C*  直後の演算命令に分岐します。
C
C     FLDA          IFEQ      FLDB
C     FLDA          ANDGT     FLDC
C     FLDD          OREQ      FLDE
C     FLDD          ANDGT     FLDF
C                       :
C                       :
C                   ENDIF
```

図 235. IFxx/ENDIF および IFxx/ELSE/ENDIF 演算命令

IN (データ域の検索)

フリー・フォーム構文	IN{(E)} {*LOCK} *data-area-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
IN (E)	*LOCK	データ域名		_	ER	_

IN 演算命令は、データ域を検索します。

*LOCK オプションを指定して、以下の時点までデータ域を別のプログラムが更新またはロックできないことを示すことができます。
- UNLOCK 演算命令が実行される。
- *data-area-name* オペランドがない OUT 演算命令が実行される。
- プログラムの終了時に、プログラムがデータ域のロックを明示的に解除する。

データ域がロックされた場合は、他のプログラムがそれを読み取ることはできますが、更新することはできません。

*data-area-name*は、DTAARA キーワードによって定義された定義の名前、*DTAARA DEFINE 演算命令の結果フィールド、または予約語 *DTAARA でなければなりません。 *DTAARA が指定されていると、プログラム中に定義されたすべてのデータ域が検索されます。

IN の例外 (プログラム状況コード 401 ～ 421、431、または 432) を処理するには、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方は指定できません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

固定形式演算では、71 ～ 72 および 75 ～ 76 桁目はブランクでなければなりません。

一般規則の説明については、363 ページの『データ域命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  TOTAMT、TOTGRS、および TOTNET がデータ域として定義されています。
C*  IN 演算命令は、プログラムの中で定義されているすべてのデータ域を
C*  検索し、それらをロックします。プログラムは、演算を処理し、次に、
C*  すべてのデータ域に書き込んで、ロックを解除します。
C*  その後、そのデータ域を他のプログラムで使用することができます。
C*
C     *LOCK         IN        *DTAARA
C                   ADD       AMOUNT        TOTAMT
C                   ADD       GROSS         TOTGRS
C                   ADD       NET           TOTNET
C
C                   OUT       *DTAARA
C
C*
C* データ域の定義
C*
C     *DTAARA       DEFINE                  TOTAMT            8 2
C     *DTAARA       DEFINE                  TOTGRS           10 2
C     *DTAARA       DEFINE                  TOTNET           10 2
```

図 236. IN および OUT 演算命令

ITER (繰り返し)

フリー・フォーム構文	ITER

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ITER						

ITER 演算命令は、DO または FOR グループの中から DO グループの ENDDO または ENDFOR ステートメントに制御を転送します。これを DO、DOU、DOUxx、DOW、DOWxx、および FOR ループの中で使用して、ループの ENDDO または ENDFOR ステートメントに制御を即時に転送することができます。これによって、ループの次の繰り返しが即時に実行されます。ITER はループ中の最深部に影響します。

制御を渡す先の ENDDO または ENDFOR ステートメントに条件付け標識が指定されていて、条件が満たされない場合には、その ENDDO または ENDFOR 演算命令の後のステートメントから処理が続行されます。

LEAVE 演算命令は、ITER 演算命令と類似しています。しかし、LEAVEでは、制御が ENDDO または ENDFOR 演算命令の ***後の*** ステートメントに転送されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の例は、DOW ループを含む DOU ループを使用します。
C* IF ステートメントが標識 01 を検査します。標識 01 がオンである
C* 場合には、LEAVE 演算命令が実行され、制御が最深部の DOW ループ
C* から Z-ADD 命令に転送されます。標識 01 がオンでない場合には、
C* サブルーチン PROC1 が処理されます。次に標識 12 が検査されます。
C* 標識 12 がオフである場合には、ITER は制御を最深部の ENDDO に
C* 転送するので、DOW の条件が再度評価されます。標識 12 がオンで
C* ある場合には、サブルーチン PROC2 が処理されます。
C
C                   DOU       FLDA = FLDB
C                   :
C     NUM           DOWLT     10
C                   IF        *IN01
C                   LEAVE
C                   ENDIF
C                   EXSR      PROC1
C     *IN12         IFEQ      *OFF
C                   ITER
C                   ENDIF
C                   EXSR      PROC2
C                   ENDDO
C                   Z-ADD     20            RSLT              2 0
C                   :
C                   ENDDO
C                   :
```

図 237. ITER 演算命令 (1/2)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の例は、DOW ループを含む DOU ループを使用します。
C* IF ステートメントが標識 1 を検査します。標識 01 がオンである
C* 場合には、MOVE 演算命令が実行され、続いて LEAVE 演算命令が
C* 実行されて、制御は最深部の DOW ループから Z-ADD 命令に転送
C* されます。標識 01 がオンでない場合には、ITER は制御を最深部
C* の ENDDO に転送するので、DOW の条件が再度評価されます。
C*
C                   :
C     FLDA          DOUEQ     FLDB
C                   :
C     NUM           DOWLT     10
C     *IN01         IFEQ      *ON
C                   MOVE      'UPDATE'      FIELD           20
C                   LEAVE
C                   ELSE
C                   ITER
C                   ENDIF
C                   ENDDO
C                   Z-ADD     20            RSLT             2 0
C                   :
C                   ENDDO
C                   :
```

図 237. ITER 演算命令 (2/2)

KFLD (キーのパーツの定義)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
KFLD			キー・フィールド			

KFLD 演算命令は、フィールドが KLIST 名によって識別される検索引き数のパーツであることを指示します。

KFLD 演算命令は、演算の任意の位置に指定することができますが、KLIST または KFLD 演算命令の後に続いていなければなりません。条件付け標識の入力 (9 - 11 桁目) は許可されていません。

KFLD は、グローバルまたはローカルとすることができます。メイン・プロシージャーの中の KLIST には、それと関連したグローバル KFLD だけが可能です。サブルーチンの中の KLIST には、ローカルおよびグローバル KFLD が可能です。

演算項目 2 には、**ユーザー制御**オプションまたは **ALWNULL(*USRCTL)** キーワードが指定された場合のヌル使用可能キー・フィールドの標識を含めることができます。

この標識がオンの場合には、ヌル値をもつキー・フィールドが選択されます。この標識がオフであるか指定されない場合には、ヌル値をもつキー・フィールドは選択されません。ヌル使用可能キーのアクセス方法については、149 ページの『キーによる演算命令』を参照してください。

結果フィールドには、検索引き数のパーツとなるフィールドの名前が含まれていることが必要です。結果フィールドに配列名を含めることはできません。各 KFLD フィールドは、長さ、データ・タイプ、および小数点以下の桁数の点でレコードまたはファイルの複合キーの中の対応するフィールドと一致していなければなりません。しかし、レコードに可変長 KFLD フィールドが含まれていた場合には、複合キーの中の対応するフィールドを変更することが必要になりますが、同じ長さである必要はありません。個々の KFLD フィールドが複合キーの中の対応するフィールドと同じ名前をもっている必要はありません。 KLIST に指定された KFLD フィールドの順序によって、どの KFLD が複合キーの中の特定のフィールドと関連づけられるかが決まります。たとえば、KLIST 演算命令の後の最初の KFLD フィールドは、複合キーの左端 (高位) フィールドと関連づけられます。

グラフィックおよび UCS-2 キー・フィールドの CCSID は、ファイル内のキーと同じでなければなりません。

578 ページの図 238 は、KFLD 演算命令と一緒の KLIST 演算命令の例を示しています。

KLIST (複合キーの定義)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
KLIST	KLIST 名					

KLIST 演算命令は、KFLD のリストに名前を与えます。このリストは、複合キーをもつ外部記述ファイルからレコードを検索するための検索引き数として使用されます。複合キーは、キー・フィールドのリストを含むキーです。これは、左から右へ組み込まれます。指定される最初の KFLD は、複合キーの左端 (高位) フィールドです。

KLIST の直後に、少なくとも 1 つの KFLD がなければなりません。KLIST は、KFLD 以外の演算命令に達した時に終了されます。検索引き数が複数のフィールド (複合キー) から構成される場合には、複数の KFLDをもつ KLIST を指定する必要があります。同じ KLIST 名を複数のファイルに対する検索引き数として使用したり、あるいは同じファイルに対する検索引き数として複数回使用することができます。

演算項目 1 には、固有の名前が含まれていなければなりません。この名前は、CHAIN、DELETE、READE、 READPE、SETGT、または SETLL 演算命令の演算項目 1 に表すことができます。

条件付け標識の入力 (9 - 11 桁目) は許可されていません。

メイン・プロシージャーの中の KLIST には、それと関連したローカル KFLD だけが可能です。サブルーチンの中の KLIST には、ローカルおよびグローバル KFLD が可能です。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
A*  DDS ソース
A          R RECORD
A            FLDA           4
A            SHIFT          1  0
A            FLDB          10
A            CLOCK#         5  0
A            FLDC          10
A            DEPT           4
A            FLDD           8
A          K DEPT
A          K SHIFT
A          K CLOCK#
A*
A*  DDS ソースの終わり
A*
A******************************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  KLIST 演算命令は、検索引き数の指定に使用できる名前 FILEKY
C*  を指示します。
C     FILEKY        KLIST
C                   KFLD                    DEPT
C                   KFLD                    SHIFT
C                   KFLD                    CLOCK#
```

図 238. KLIST および KFLD 演算命令

LEAVE (DO/FOR グループの終了)

フリー・フォーム構文	LEAVE

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
LEAVE						

LEAVE 演算命令は、DO または FOR グループの中から ENDDO または ENDFOR 演算命令の後のステートメントに制御を転送します。

LEAVE を DO、DOU、DOUxx、DOW、DOWxx、または FOR ループの中で使用して、最深部のループから最深部のループの ENDDO または ENDFOR 演算命令の直後のステートメントに制御を転送することができます。LEAVE を使用して DO または FORグループを終了すると、索引は増分されません。

ネストされたループでは、LEAVE によって制御が 1 レベルだけですが外側に転送されることになります。 DO または FOR グループの外側での LEAVE は許されません。

ITER 演算命令は、LEAVE 演算命令と類似しています。しかし、ITER では、制御が ENDDO または ENDFOR ステートメント へ 転送されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の例は無限ループを使用します。ユーザーが「q」を入力すると、
C* 制御が LEAVE 演算命令に転送されます。次に、制御はループから
C* Z-ADD 演算命令に転送されます。
C*
C     2             DOWNE     1
C                   :
C                   IF        ANSWER = 'q'
C                   LEAVE
C                   ENDIF
C                   :
C                   ENDDO
C                   Z-ADD     A             B
C*
C* 次の例は DOWxx を含む DOUxx ループを使用します。
C* IF ステートメントが標識 1 を検査します。これがオンである
C* 場合には、標識 99 がオンになり、制御は LEAVE 演算命令に
C* 渡され、内部の DOWxx ループから外に渡されます。
C*
C* 標識 99 がオンであるので、2 番目の LEAVE 命令が実行され、
C* 次に、制御は DOUxx ループから外へ転送されます。
C*
C                   :
C     FLDA          DOUEQ     FLDB
C     NUM           DOWLT     10
C     *IN01         IFEQ      *ON
C                   SETON                                        99
C                   LEAVE
C                   :
C                   ENDIF
C                   ENDDO
C  99               LEAVE
C                   :
C                   ENDDO
C                   :
```

図 239. LEAVE 演算命令

LEAVESR (サブルーチンの終了)

フリー・フォーム構文	LEAVESR

コード	演算項目 1	演算項目 2	結果フィールド	標識		
LEAVESR						

LEAVESR 演算命令は、サブルーチンの中の任意の点からサブルーチンを終了します。制御は、そのサブルーチンのための ENDSR演算命令に渡されます。 LEAVESR は、サブルーチンの中からのみ使用することができます。

制御レベルの項目 (7 - 8 桁目) は SR またはブランクとすることができます。条件付け標識の項目 (9 - 11桁目) を指定することができます。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
 *
C     CheckCustName BEGSR
C     Name          CHAIN     CustFile
 *
 * 名前 ID が有効なカストマーであるかどうかを検査します。
 *
C                   IF        not %found(CustFile)
C                   EVAL      Result = CustNotFound
C                   LEAVESR
C                   ENDIF
 *
 * カスタマーが割引プログラムを修飾しているかどうかを検査します。
C                   IF        Qualified = *OFF
C                   EVAL      Result = CustNotQualified
C                   LEAVESR
C                   ENDIF
 *
 * ここで、カスタマーが割引プログラムを使用することができます。
C                   EVAL      Result = CustOK
C                   ENDSR
```

図 240. LEAVESR 演算命令

LOOKUP (テーブルまたは配列エレメントの検索)

フリー・フォーム構文	(使用できません。%LOOKUPxx または %TLOOKUPxx 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
LOOKUP						
(配列)	検索引き数	配列名		HI	LO	EQ
(テーブル)	検索引き数	テーブル名	テーブル名	HI	LO	EQ

LOOKUP 演算命令は、配列またはテーブルのエレメントを検索します。検索引き数とテーブルまたは配列は、同じタイプおよび長さでなければなりません (時刻および日付フィールドは例外であって、異なる長さとすることができます)。配列またはテーブルが固定長文字、グラフィック、または UCS-2である場合には、検索引き数も固定長であることが必要です。可変長の場合には、検索引き数の長さを配列またはテーブルとは異なる長さとすることができます。テーブルまたは配列の順序は、ASCEND または DESCEND キーワードを使用して定義仕様に指定されていなければなりません。

演算項目 1 は、リテラル、フィールド名、配列エレメント、テーブル名、名前付き固定情報、または表意定数でなければなりません。比較の性質は、次のようにデータ・タイプによって異なります。

グラフィックおよび UCS-2 データ
比較は 16 進数です。

数値データ
小数点位置合わせは処理されません。

その他のデータ・タイプ
その他のタイプには、 361 ページの『比較命令』で説明されている比較の考慮事項が適用されます。

テーブル LOOKUP の場合には、検索引き数が LOOKUP 演算命令で最後に選択されたテーブルのエレメントとなります。最後の LOOKUP 演算命令が処理されていない場合には、テーブルの最初のエレメントが検索引き数として使用されます。テーブル LOOKUP について結果フィールドを指定する場合には、2 番目のテーブルの名前を含める必要があります。2 番目のテーブルのエレメントの位置は、最初のテーブルのエレメントの位置と対応します。 LOOKUP演算命令は、2 番目のテーブルからエレメントを検索します。

配列 LOOKUP の場合には、索引を使用することができます。LOOKUP は、索引によって指定されたエレメントから開始されます。索引値は、見つかったエレメントの位置番号に設定されます。索引がゼロと等しいか、または検索が開始された時の配列内のエレメントの数より大きい場合には、エラーが起こります。検索が正常に行なわれなかった場合には、索引は 1 に設定されます。索引が名前付き固定情報である場合には、索引値は変更されません。

実行する検索を決定し、次に、検索の結果を表すために、演算結果標識を指定する必要があります。また、テーブルまたは配列の順序が ASCEND または DESCEND キーワードを使用して定義仕様に指定されていなければなりません。指定されたどの標識も、検索が正常に行なわれた場合にのみオンに設定されます。使用できるのは、2 つ以下の標識です。演算結果標識は、EQ および HI または EQ および LO に割り当てることができます。プログラムが EQ が優先されるいずれかの条件を満たす項目を検索した場合、すなわち、等しい項目が見つからない場合には、その上下で最も近い項目が選択されます。

標識が 75 - 76 桁目に指定された場合に、検索引き数と正確に一致したエレメントが見つかれば、%EQUAL 組み込み関数が '1' を戻してきます。%FOUND 組み込み関数は、指定されたすべての検索が正常に行なわれた場合に '1' を戻します。

演算結果標識は、EQ および LO または EQ および HI に割り当てることができます。HI と LO を同じ LOOKUP 演算命令で指定することはできません。LOOKUP 演算命令に HI または LO 標識が指定された場合には、コンパイラーは配列またはテーブルが分類済みで順序づけられているものと見なします。LOOKUP 演算命令では、EQ が優先される LO/EQ または HI/EQ 条件を満たす項目を検索します。

- *HI (71-72):* 検索引き数に最も近いがまだそれより順序が高い項目を検出するようにプログラムに指示します。最初に見つかった、より高い項目によって、HIに割り当てられた標識がオンに設定されます。
- *LO (73-74):* 検索引き数に最も近いがまだそれより順序が低い項目を検出するようにプログラムに指示します。このようにして最初に見つかった項目によって、LO に割り当てられた標識がオンに設定されます。
- *EQ (75-76):* 検索引き数と等しい項目を検出するようにプログラムに指示します。最初に見つかった、等しい項目によって、EQ に割り当てられた標識がオンに設定されます。

EQ 標識が指定された唯一の標識であった場合には、配列またはテーブル全体が検索されます。テーブルまたは配列が昇順になっていて、EQ 比較が必要な場合には、HI 標識を指定してください。テーブルまたは配列全体の検索は行なわれません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  次の例は、プログラマーが LOOKUP 演算命令で突き止められるのは
C*  ARY のどのエレメントであるかを知りたい場合です。Z-ADD 演算命令は、
C*  フィールド X を 1 に設定します。LOOKUP は、フィールド X に
C*  よって指示されたエレメント ARY で始まり、SRCHWD と等しい最初のエレメントを
C*  見つけるまでその実行が続けられます。索引値 X は、突き止められ
C*  たエレメントの位置番号に設定されます。
C
C                   Z-ADD     1             X               3 0
C     SRCHWD        LOOKUP    ARY(X)                                26
C
C*  次の例は、プログラマーが SRCHWD と等しいエレメントが突き止められた
C*  かどうかを知りたい場合です。LOOKUP では、SRCHWD と等しい最初
C*  のエレメントを見つけるまで ARY が検索されます。これが行なわれると、
C*  標識 26 はオンに設定され、%EQUAL は '1' を戻すように設定されます。
C
C     SRCHWD        LOOKUP    ARY                                   26
C                   C*  LOOKUP は、フィールド X によって指定された可変索引番号
C*  で始まります。LOOKUP 演算命令の前にフィールド X を 1 に
C*  設定する必要はありません。LOOKUP で ARY 中の SRCHWD と
C*  等しいエレメントが突き止められた時に、標識 26 はオンに設定され、
C*  %EQUAL は '1' を戻すように設定されます。索引値 X は、
C*  突き止められたエレメントの位置番号に設定されます。
C*
C
C     SRCHWD        LOOKUP    ARY(X)                                26
```

図 241. 配列による LOOKUP 演算命令

```
 * この例では、カスタマー情報の配列は、実際には幾つかの副配列
 * から構成されています。メイン配列を検索するか、メイン配列を
 * オーバーレイする副配列のいずれかを検索できます。
D custInfo        DS
D  cust                             DIM(100)
D   name                      30A   OVERLAY(cust : *NEXT)
D   id_number                 10I 0 OVERLAY(cust : *NEXT)
D   amount                    15P 3 OVERLAY(cust : *NEXT)

 * "cust" 配列で検索を行うことによって、カスタマー情報の
 * 特定セットを検索できます。
C     custData      LOOKUP    cust(i)                               10

 * オーバーレイ配列の 1 つで検索を行うことによって、
 * カスタマー情報の特定フィールドを検索できます。
C     custName      LOOKUP    name(i)                               11
 * 検索後、配列索引は、オーバーレイするどの配列でも
 * 使用できます。name(i) での検索が成功すると、該当
 * するカスタマーの id_number と金額が id_number(i) と
 * amount(i) で使用可能になります。
```

図 242. 副配列による LOOKUP 演算命令

MONITOR (モニター・グループの開始)

フリー・フォーム構文	MONITOR

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
MONITOR						

モニター・グループは、状況コードに基づいて条件エラー処理を実行します。これは以下のものからなります。
- MONITOR ステートメント
- 1 つ以上の ON-ERROR グループ
- ENDMON ステートメント

MONITOR ステートメントの後、制御は次のステートメントに渡されます。モニター・ブロックは、MONITOR ステートメントから最初の ON-ERROR ステートメントまでのすべてのステートメントからなります。モニター・ブロックの処理時にエラーが起こった場合には、制御が適切な ON-ERROR グループに渡されます。

MONITOR ブロック内のすべてのステートメントがエラーなしに処理されると、制御は、ENDMON ステートメントの後のステートメントに渡されます。

モニター・グループは、演算のどこでも指定することができます。また IF、DO、SELECT または他のモニター・グループ内でネストすることができます。IF、DO、および SELECT グループをモニター・グループ内にネストすることができます。

モニター・グループが別のモニター・グループ内でネストされている場合、エラーが起こると、まず一番内側のグループが考慮されます。このモニター・グループがエラー条件を処理できない場合には、次のグループが考慮されます。

MONITOR ステートメントで条件標識を使用することができます。その標識が満たされない場合には、制御はモニター・グループの ENDMON 演算命令の後のステートメントに即時に渡されます。条件標識は、ON-ERROR 演算命令で個別に使用することはできません。

モニター・ブロックにサブプロシージャーに対する呼び出しが含まれていて、そのサブプロシージャーにエラーがある場合には、そのサブプロシージャーのエラー処理が優先されます。たとえば、サブプロシージャーに *PSSR サブルーチンがある場合には、そのサブルーチンが呼び出されます。サブプロシージャーがエラーの処理に失敗し、呼び出しがエラー呼び出し状況コード 00202 で失敗した場合には、この呼び出しを含む MONITOR グループだけが考慮されます。

モニター・グループは、サブルーチン内で起こったエラーを処理します。サブルーチンに自身のモニター・グループが含まれている場合には、そのグループが最初に考慮されます。

モニター・ブロック内では分岐演算命令は使用できませんが、ON-ERROR ブロック内では使用できます。

モニター・ブロック内の LEAVE または ITER 演算命令は、そのモニター・ブロックを含むどのアクティブ DO グループにも適用されます。モニター・ブロック内の LEAVESR または RETURN 演算命令は、そのモニター・ブロックを含むどのサブルーチン、サブプロシージャー、またはプロシージャーにも適用されます。

```
 * MONITOR ブロックは、READ ステートメントと IF グループから
 * 構成されます。
 * - 最初の ON-ERROR ブロックは、ファイルがオープンされない
 *   場合に READ 演算命令で出される状況 1211を処理します。
 * - 2 番目の ON-ERROR ブロックは、他のすべてのファイル・
 *   エラーを処理します。
 * - 3 番目の ON-ERROR ブロックは、ストリング演算命令の
 *   状況コード 00100 および配列索引状況コード 00121 を
 *   処理します。
 * - 4 番目の ON-ERROR ブロック (演算項目 2 が *ALL である
 *   場合があります) は、特定の ON-ERROR 演算命令で処理
 *   されなかったエラーを処理します。
 *
 * MONITOR ブロックでエラーが起こらなかった場合には、制御は
 * ENDIF から ENDMON に渡されます。
C                   MONITOR
C                   READ      FILE1
C                   IF        NOT %EOF
C                   EVAL      Line = %SUBST(Line(i) :
C                                       %SCAN('***': Line(i)) + 1)
C                   ENDIF
C                   ON-ERROR   1211
C                    ... handle file-not-open
C                   ON-ERROR   *FILE
C                    ... handle other file errors
C                   ON-ERROR   00100 : 00121
C                    ... handle string error and array-index error
C                   ON-ERROR
C                    ... handle all other errors
C                   ENDMON
```

図 243. MONITOR 演算命令

MOVE (転送)

フリー・フォーム構文	(使用できません。EVALR 命令コードを使用するか、%DATE、%TIME、%TIMESTAMP、%CHAR、%UCS2、%GRAPH などの組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MOVE (P)	データ属性	ソース・フィールド	ターゲット・フィールド	+	–	ZB

MOVE 演算命令は、演算項目 2 から結果フィールドに文字を転送します。移動は、演算項目 2 の右端の文字から開始されます。

日付、時刻、またはタイム・スタンプ・フィールドの移動時には、ソースまたはターゲットのいずれかが文字または数字フィールドでない限り、演算項目 1 はブランクでなければなりません。

そうでない場合には、演算項目 1 には、この演算命令のソースである文字または数字フィールドと互換の日付形式または時刻形式が入ります。使用できる形式については、 125 ページの『日付データ』、 144 ページの『時刻データ』、および 145 ページの『タイム・スタンプ・データ』を参照してください。

ソースまたはターゲットが文字フィールドの場合には、任意選択で演算項目 1 の形式の後に区切り文字を指示することができます。使用できるのはその形式に有効な区切り文字だけです。

演算項目 2 が *DATE または UDATE で、結果が日付フィールドの場合には、演算項目 1 は必須でありません。演算項目 1 に日付形式が入っている場合には、制御仕様で DATEDIT キーワードによって指定されている *DATE または UDATE の形式と互換になっていなければなりません。

文字、グラフィック、UCS-2、または数字日付の移動時には、演算項目 2 が結果フィールドより長い場合には、長さを超えている演算項目 2 の左端の文字または数字は移動されません。結果フィールドが演算項目 2 より長い場合には、埋め込みが指定されていない限り、長さを超えている結果フィールドの左端の文字または数字は未変更になっています。

結果フィールドが配列の場合には、演算結果標識をしてすることはできません。配列エレメントまたは配列以外のフィールドの場合には、それを指定することができます。

演算項目 2 の長さが結果フィールドより短い場合には、命令拡張桁に P が指定されていると、結果フィールドが移動後に左が埋め込まれます。

浮動数字フィールドおよびリテラルは、演算項目 2 または結果フィールドの項目として使用できません。

モジュールのために CCSID(*GRAPH : IGNORE) が指定されているか想定されている場合には、UCS-2 データとグラフィック・データの間で MOVE 演算命令は使用できません。

可変長文字、グラフィック、または UCS-2 データの移動時に、可変長フィールドは、現行の長さと同じ固定長フィールドと全く同様に処理されます。たとえば、250 - 255 を参照してください。

例の後に示されているテーブル (589 ページの『MOVE の例 (パート 1)』 を参照) は、データが演算項目 2 から結果フィールドにどのように移動されるかを示しています。MOVE 演算命令の詳細については、 372 ページの『移動命令』を参照してください。

MOVE の例 (パート 1)

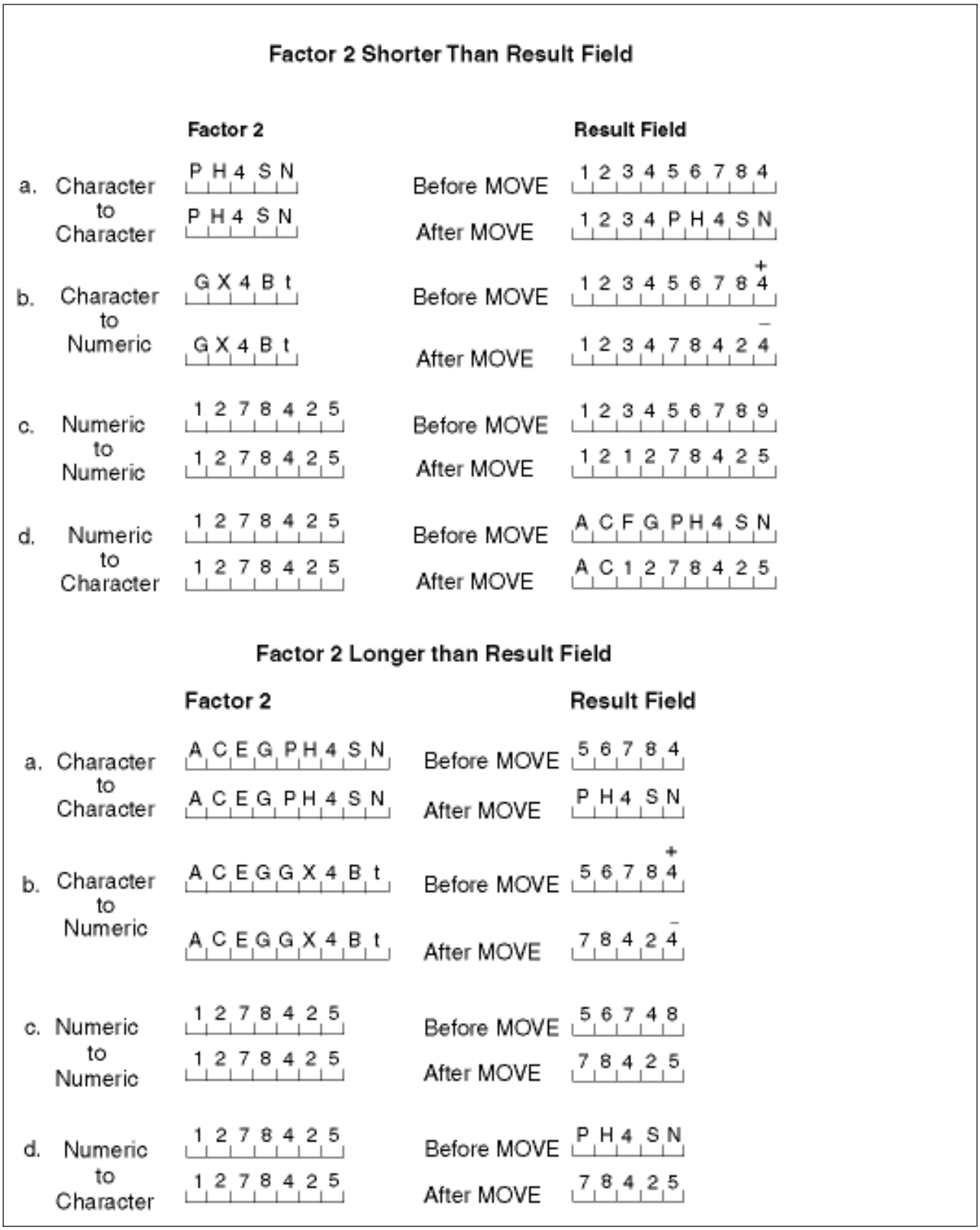


図 244. MOVE 演算命令 (1/2)

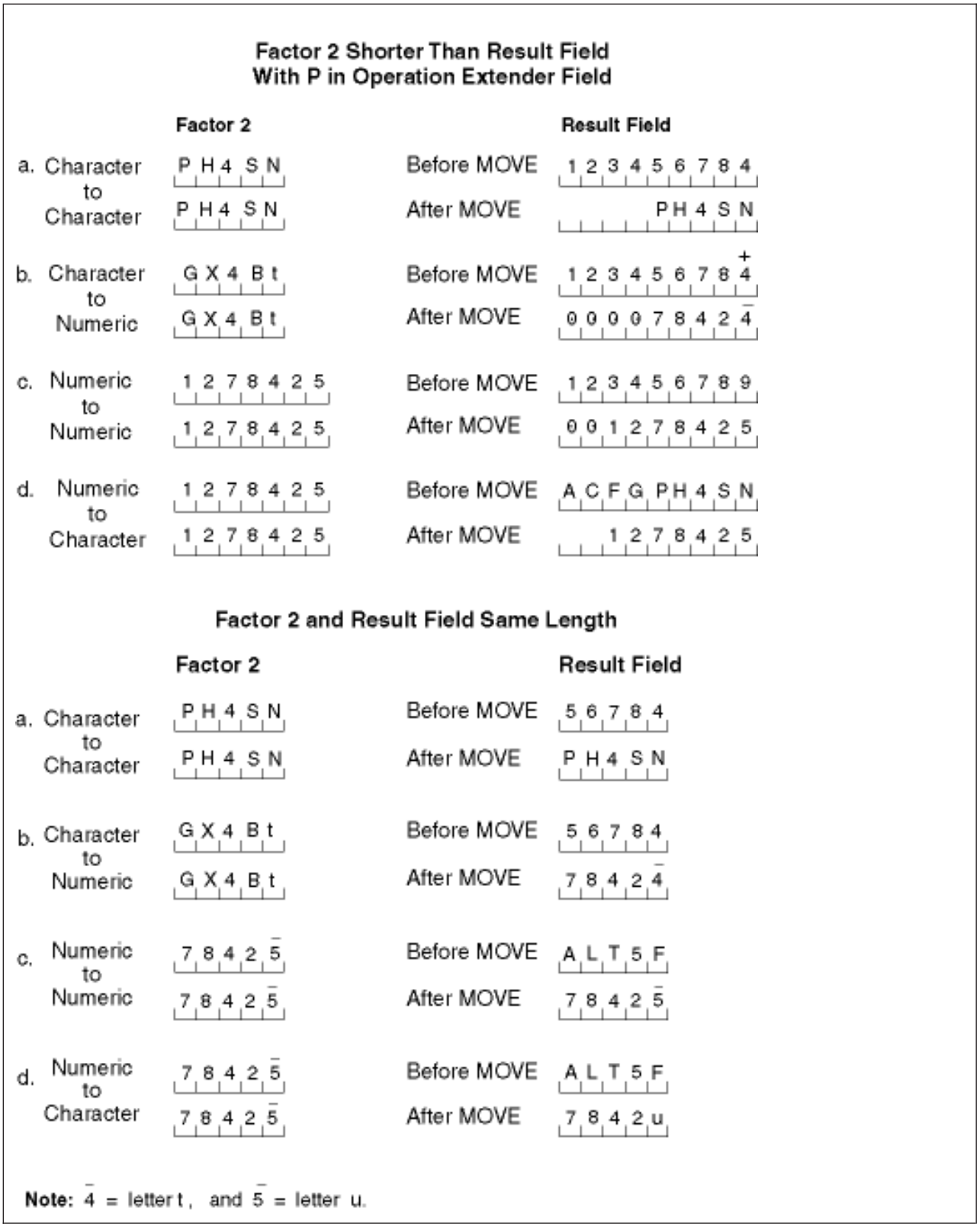


図 244. MOVE 演算命令 (2/2)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
H*  制御仕様日付形式
H*
H DATFMT(*ISO)
     H
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D*
D DATE_ISO       S               D
D DATE_YMD       S               D   DATFMT(*YMD)
D                                    INZ(D'1992-03-24')
D DATE_EUR       S               D   DATFMT(*EUR)
D                                    INZ(D'2197-08-26')
D DATE_JIS       S               D   DATFMT(*JIS)
D NUM_DATE1      S              6P 0 INZ(210991)
D NUM_DATE2      S              7P 0
D CHAR_DATE      S              8    INZ('02/01/53')
D CHAR_LONGJUL   S              8A   INZ('2039/166')
D DATE_USA       S               D   DATFMT(*USA)
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+H1LoEq..
C*
C* 日付フィールド間の移動。DATE_EUR には 24.03.1992 が入ります。
C*
C                   MOVE      DATE_YMD      DATE_EUR
C*
C* ddmmyy 形式の数字を *ISO 日付に変換します。
C* 2 つの各移動後に、DATE_ISO には 1991-09-21 が入ります。
C*
C     *DMY          MOVE      210991        DATE_ISO
C     *DMY          MOVE      NUM_DATE      DATE_ISO
C*
C* *MDY 日付を表している文字値を *JIS 日付に移動します。
C* 2 つの各移動後に、DATE_JIS には 1953-02-01 が入ります。
C*
C     *MDY/         MOVE      '02/01/53'    DATE_JIS
C     *MDY/         MOVE      CHAR_DATE     DATE_JIS
```

図 245. 日付での移動演算命令

```
C*
C* DATE_USA には 12-31-9999 が入ります
C*
C                   MOVE      *HIVAL        DATE_USA
C*
C*  エラー・コード 114 になる実行エラー。年が 1940-2039 の
C*  日付範囲になっていません、DATE_YMD は未変更です。
C*
C                   MOVE      DATE_USA      DATE_YMD
C*
C* *CYMD 日付を表している文字値を *USA 日付に移動します。
C* この移動後に、DATE_USA には 08/07/1961 が入ります。
C* *CYMD の 0 は、文字値に区切り文字が含まれていないことを
C* 指示します。
C*    *CYMD0        MOVE      CHAR_NO_SEP   DATE_USA
C*
C* *EUR 日付フィールドを数字フィールド (*CMDY 日付を表し
C* ている) に移動します。NUM_DATE2 には移動後に 2082697  が
C* 入ります。
C    *CMDY          MOVE      DATE_EUR      NUM_DATE2
C*
C* *LONGJUL 日付を表している文字値を *YMD 日付に移動
C* します。移動後に DATE_YMD は 39/06/15 になります。
C    *LONGJUL       MOVE      CHAR_LONGJUL  DATE_YMD
```

図 246. 日付での移動演算命令 (続き)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
H* 日付フィールドのデフォルト形式を指定します
H DATEFMT(*ISO)
H*
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D date_USA       S               D   DATFMT(*USA)
D datefield      S               D
D timefield      S               T   INZ(T'14.23.10')
D chr_dateA      S              6    INZ('041596')
D chr_dateB      S              7    INZ('0610807')
D chr_time       S              6
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* *MDY 日付を表している文字値を D(日付) 値に移動します。
C* *MDY0 は、演算項目 2 の文字日付には区切り文字が含まれていないことを
C* 指示します。
C* 移動後に、datefld には 1996-04-15 が入ります。
C     *MDY          MOVE      chr_dateA     datefld
C* T(時刻) 値が入っているフィールドを文字値に *EUR 形式で移動
C* します。*EUR0 は結果フィールドに区切り文字を入れないことを
C* 指示します。
C* 移動後に、chr_time には '142310' が入ります。
C     *EUR0         MOVE      timefld       chr_time
C*
C* *CYMD 日付を表している文字値を *USA 日付に移動します。
C* 移動後に Date_USA には 08/07/1961 が入ります。
C* *CYMD の 0 は、文字値に区切り文字が含まれていないことを
C* 指示します。
C*
C     *CYMD0        MOVE      chr_dateB     date_USA
```

図 247. 区切り文字なしの日時での MOVE 演算命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
H*  制御仕様 DATEDIT 形式
H*
H DATEDIT(*MDY)
H*
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++
D Jobstart        S               Z
D Datestart       S               D
D Timestart       S               T
D Timebegin       S               T   inz(T'05.02.23')
D Datebegin       S               D   inz(D'1991-09-24')
D TmStamp         S               Z   inz
D*
C* タイム・スタンプ Jobstart をジョブ開始日時で設定します。
C *
C * DATEDIT(*MDY) は *DATE が MMDDYYYY として様式化されるよ
C * うに指示しているので、MOVE *DATE (*USA = MMDDYYYY) の演
C * 算項目 1 は、制御仕様で DATEDIT キーワードに指定された
C * 値と整合性があります。
C *
C* 注: 必ずしも演算項目 1 は *DATE または
C*     UDATE で指定する必要はありません。
C*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C     *USA          MOVE      *DATE         Datestart
C                   TIME                    StrTime           6 0
C     *HMS          MOVE      StrTime       Timestart
C                   MOVE      Datestart     Jobstart
C                   MOVE      Timestart     Jobstart
C*
C* 次の C 仕様が実行された後で、フィールド stampchar には
C* stampchar には「1991-10-24-05.17.23.000000」が入ります。
C*
C* 最初に、タイム・スタンプに与えられた時刻 +15 分と与えられた日付 +30
C* 日の値を割り当てます。tmstamp を文字フィールドに移動します。
C* stampchar には「1991-10-24-05.17.23.000000」が入ります。
C*
C                   ADDDUR    15:*minutes   Timebegin
C                   ADDDUR    30:*days      Datebegin
C                   MOVE      Timebegin     TmStamp
C                   MOVE      Datebegin     TmStamp
C                   MOVE      TmStamp       stampchar        26
C* タイム・スタンプを区切り文字なしで文字フィールドに移動します。
C* 移動後に STAMPCHAR には '      19911024051723000000' が入ります。
C     *ISO0         MOVE(P)   TMSTAMP       STAMPCHAR0
```

図 248. タイム・スタンプでの MOVE 演算命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* グラフィックス・フィールドと文字フィールドの間の MOVE の例
D*
D char_fld1      S              8A   inz('K1K2K3  ')
D dbcs_fld1      S              4G
D char_fld2      S              8A   inz(*ALL'Z')
D dbcs_fld2      S              3G   inz(G'K1K2K3')
D*
C*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C* MOVE 演算命令後の dbcs_fld1 の値は 'K1K2K3  ' です
C* MOVE 演算命令後の char_fld2 の値は 'ZZK1K2K3' です
C*
C                   MOVE      char_fld1     dbcs_fld1
C                   MOVE      dbcs_fld2     char_fld2
```

図 249. 文字フィールドとグラフィックス・フィールドの間の MOVE

MOVE 例 (パート 2): 可変長フィールドおよび固定長フィールド

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 文字フィールドの可変長から可変長への
D* MOVEL(P) の例
D*
D var5a          S              5A   INZ('ABCDE') VARYING
D var5b          S              5A   INZ('ABCDE') VARYING
D var5c          S              5A   INZ('ABCDE') VARYING
D var10a         S             10A   INZ('0123456789') VARYING
D var10b         S             10A   INZ('ZXCVBNM') VARYING
D var15a         S             15A   INZ('FGH') VARYING
D var15b         S             15A   INZ('FGH') VARYING
D var15c         S             15A   INZ('QWERTYUIOPAS') VARYING
C*
C*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVE      var15a        var5a
C* var5a = 'ABFGH' (長さ =5)
C                   MOVE      var10a        var5b
C* var5b = '56789' (長さ =5)
C                   MOVE      var5c         var15a
C* var15a = 'CDE' (長さ =3)
C                   MOVE      var10b        var15b
C* var15b = 'BNM' (長さ =3)
C                   MOVE      var15c        var10b
C* var10b = 'YUIOPAS' (長さ =7)
```

図 250. 可変長フィールドから可変長フィールドへの MOVE

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 文字フィールドの可変長から固定長への
D* MOVEL(P) の例
D*
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGH') VARYING
D fix5a          S              5A   INZ('MNOPQ')
D fix5b          S              5A   INZ('MNOPQ')
D fix5c          S              5A   INZ('MNOPQ')
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVE      var5          fix5a
C* fix5a = 'ABCDE'
C                   MOVE      var10         fix5b
C* fix5b = '56789'
C                   MOVE      var15         fix5c
C* fix5c = 'MNFGH'
```

図 251. 可変長フィールドから固定長フィールドへの MOVE

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 文字フィールドの固定長から可変長への
D* MOVEL(P) の例
D*
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGHIJKL') VARYING
D fix5           S              5A   INZ('.....')
D fix10          S             10A   INZ('PQRSTUVWXY')
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVE      fix10         var5
C* var5 = 'UVWXY' (長さ =5)
C                   MOVE      fix5          var10
C* var10 = '01234.....' (長さ =10)
C                   MOVE      fix10         var15
C* var15 = 'STUVWXY' (長さ =7)
```

図 252. 固定長フィールドから可変長フィールドへの MOVE

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 文字フィールドの可変長から可変長への
D* MOVEL(P) の例
D*
D var5a          S              5A   INZ('ABCDE') VARYING
D var5b          S              5A   INZ('ABCDE') VARYING
D var5c          S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGH') VARYING
D var15b         S             15A   INZ('FGH') VARYING
D var15c         S             15A   INZ('FGH') VARYING
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVE(P)   var15a        var5a
C* var5a = 'FGH  ' (長さ =5)
C                   MOVE(P)   var10         var5b
C* var5b = '56789' (長さ =5)
C                   MOVE(P)   var5c         var15b
C* var15b = 'CDE' (長さ =3)
C                   MOVE(P)   var10         var15c
C* var15c = '789' (長さ =3)
```

図 253. 可変長フィールドから可変長フィールドへの MOVE(P)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 文字フィールドの可変長から固定長への
D* MOVEL(P) の例
D*
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGH') VARYING
D fix5a          S              5A   INZ('MNOPQ')
D fix5b          S              5A   INZ('MNOPQ')
D fix5c          S              5A   INZ('MNOPQ')
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVE(P)   var5          fix5a
C* fix5a = 'ABCDE'
C                   MOVE(P)   var10         fix5b
C* fix5b = '56789'
C                   MOVE(P)   var15         fix5c
C* fix5c = 'FGH  '
```

図 254. 可変長から固定長フィールドへの MOVE(P)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 文字フィールドの固定長から可変長への
D* MOVEL(P) の例
D*
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGHIJKLMNOPQR') VARYING
D var15b         S             15A   INZ('FGHIJ') VARYING
D fix5           S              5A   INZ('')
D fix10          S             10A   INZ('PQRSTUVWXY')
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVE(P)   fix10         var5
C* var5 = 'UVWXY' (前後に長さ =5)
C                   MOVE(P)   fix10         var10
C* var10 = 'PQRSTUVWXY' (前後に長さ =10)
C                   MOVE(P)   fix10         var15a
C* var15a = '   PQRSTUVWXY' (前後に長さ =13)
C                   MOVE(P)   fix10         var15b
C* var15b = 'UVWXY' (前後に長さ =5)
```

図 255. 固定長フィールドから可変長フィールドへの MOVE(P)

MOVE 例 (パート 3)

表 53. 文字フィールドから日付時刻フィールドへの移動。演算項目 1 は演算項目 2 項目の形式を指定します。

		結果フィールド	
演算項目 1	演算項目 2 (文字)	Value	DTZ タイプ
*MDY	11-19-75	75/323	D(*JUL)
*JUL	92/114	23/04/92	D(*DMY)
*YMD	14/01/28	01/28/2014	D*(USA)
*YMD0	140128	01/28/2014	D(*USA)
*USA	12/31/9999	31.12.9999	D(*EUR)
*ISO	2036-05-21	21/05/36	D(*DMY)
*JUL	45/333	11/29/1945	D(*USA)
*MDY/	03/05/33	03.05.33	D(*MDY.)
*CYMD&	121 07 08	08.07.2021	D(*EUR)
*CYMD0	1210708	07,08,21	D(*MDY,)
*CMDY.	107.08.21	21-07-08	D(*YMD-)
*CDMY0	1080721	07/08/2021	D(*USA)
*LONGJUL-	2021-189	08/07/2021	D(*EUR)
*HMS&	23 12 56	23.12.56	T(*ISO)
*USA	1:00 PM	13:00.00	T(*EUR)
*EUR	11.10.07	11:10:07	T(*JIS)
*JIS	14:16:18	14.16.18	T(*HMS.)
*ISO	24:00.00	12:00 AM	*T(*USA)
ブランク	1991-09-14-13.12.56.123456	1991-09-14-13.12.56.123456	Z(*ISO)
*ISO	1991-09-14-13.12.56.123456	1991-09-14-13.12.56.123456	Z(*ISO)

表 54. 数字フィールドから日付時刻フィールドへの移動。演算項目 1 は演算項目 2 項目の形式を指定します。

		結果フィールド	
演算項目 1	演算項目 2 (数字)	Value	DTZ タイプ
*MDY	111975	75/323	D(*JUL)
*JUL	92114	23/04/92	D(*DMY)
*YMD	140128	01/28/2014	D(*USA)
*USA (注 1 を参照。)	12319999	31.12.9999	D(*EUR)
*ISO	20360521	21/05/36	D(*DMY
*JUL	45333	11/29/1945	D(*USA)
*MDY	030533	03.05.33	D(*MDY.)
*CYMD	1210708	08.07.2021	D(*EUR)
*CMDY	1070821	21-07-08	D(*YMD-)
*CDMY	1080721	07/08/2021	D(*USA)
*LONGJUL	2021189	08/07/2021	D(*EUR)

表 54. 数字フィールドから日付時刻フィールドへの移動。演算項目 1 は演算項目 2 項目の形式を指定します。 (続き)

*USA	*DATE (092195) (注 3 を参照。)	1995-09-21	D(*JIS)
ブランク	*DATE (092195) (注 3 を参照。)	1995-09-21	D(*JIS)
*MDY	UDATE (092195) (注 3 を参照。)	21.09.1995	D(*EUR)
*HMS	231256	23.12.56	T(*ISO)
*EUR	111007	11:10:07	T(*JIS)
*JIS	141618	14.16.18	T(*HMS.)
*ISO	240000	12:00 AM	T(*USA)
ブランク (注 4 を参照。)	19910914131256123456	1991-09-14-13.12.56.123456	Z(*ISO)

注:

1 時刻形式 *USA は、時刻クラスと数字クラスの間の移動には使用できません。

2 ゼロ (0) の区切り文字は、日付、時刻、またはタイム・スタンプと数字クラスの間の移動のために演算項目 1 では使用できません。

3 *DATE および UDATE の場合は、ジョブ記述のジョブ日付が *MDY 形式になっていて、092195が入っているものとします。演算項目 1 は任意指定で、デフォルトとして正しい形式が使用されます。演算項目 2 が *DATE で、演算項目 1 がコーディングされている場合には、4 桁の年の日付形式でなければなりません。演算項目 2 が UDATE で、演算項目 1 がコーディングされている場合には、2 桁の年の日付形式でなければなりません。

4 タイム・スタンプ・フィールドの移動の場合は、演算項目 1 は任意指定です。コーディングする場合には、*ISO または *ISO0 としなければなりません。

MOVE 例 (パート 4)

表 55. 日付時刻フィールドから文字フィールドへの移動

	演算項目 2		
演算項目 1 項目	Value	DTZ タイプ	結果フィールド (文字)
*JUL	11-19-75	D(*MDY-)	75/323
*DMY-	92/114	D(*JUL)	23-04-92
*USA	14/01/28	D(*YMD)	01/28/2014
*EUR	12/31/9999	D(*USA)	31.12.9999
*DMY,	2036-05-21	D(*ISO)	20,05,36
*USA	45/333	D(*JUL)	11/29/1945
*USA0	45/333	D(*JUL)	11291945
*MDY&	03/05/33	D(*MDY)	03 05 33
*CYMD,	03 07 08	D(*DMY)	1080721
*CMDY	21-07-08	D(*YMD-)	107/08/21
*CDMY-	07/08/2021	D(*USA)	108-07-21
*LONGJUL&	08/07/2021	D(*EUR)	2021 189
*ISO	23 12 56	T(*HMS&)	23.12.56
*EUR	11:00 AM	T(*USA)	11.00.00
*JIS	11.10.07	T(*EUR)	11:10:07
*HMS,	14:16:18	T(*JIS)	14,16,18
*USA	24.00.00	T(*ISO)	12:00 AM
ブランク	2045-10-27-23.34.59.123456	Z(*ISO)	2045-10-27-23.34.59.123456

表 56. 日付時刻フィールドから数字フィールドへの移動

	演算項目 2		
演算項目 1 項目	Value	DTZ タイプ	結果フィールド (数字)
*JUL	11-19-75	D(*MDY-)	75323
*DMY-	92/114	D(*JUL)	230492
*USA	14/01/28	D(*YMD)	01282014
*EUR	12/31/9999	D(*USA)	31129999
*DMY	2036-05-21	D(*ISO)	210536
*USA	45/333	D(*JUL)	11291945
*MDY&	03/05/33	D(*MDY)	030533
*CYMD,	03 07 08	D(*MDY&)	1080307
*CMDY	21-07-08	D(*YMD-)	1070821
*CDMY-	07/08/2021	D(*USA)	1080721
*LONGJUL&	08/07/2021	D(*EUR)	2021189
*ISO	231256	T(*HMS&)	231256
*EUR	11:00 AM	T(*USA)	110000
*JIS	11.10.07	T(*EUR)	111007

表 56. 日付時刻フィールドから数字フィールドへの移動 (続き)

*HMS,	14:16:18	T(*JIS)	141618
*ISO	2045-10-27-23.34.59.123456	Z(*ISO)	20451027233459123456

次のテーブルには、日付時刻フィールドから日付時刻フィールドへの移動の例が示されています。タイム・スタンプの初期値は *1985-12-03-14.23.34.123456* とします。

表 57. 日付時刻フィールドから日付時刻フィールドへの移動

	演算項目 2		結果フィールド	
演算項目 1	Value	DTZ タイプ	Value	DTZ タイプ
N/A	1986-06-24	D(*ISO)	86/06/24	D(*YMD)
N/A	23 07 12	D(*DMY&)	23.07.2012	D(*EUR)
N/A	11:53 PM	T(USA)	23.53.00	T(*EUR)
N/A	19.59.59	T(*HMS)	19:59:59	T(*JIS)
N/A	1985-12-03-14.23.34.123456	Z(*ISO.)	1985-12-03-14.23.34.123456	Z(*ISO)
N/A	75.06.30	D(*YMD)	1975-06-30-14.23.34.123456	Z(*ISO)
N/A	09/23/2234	D(*USA)	2234-09-23-14.23.34.123456	Z(*ISO)
N/A	18,45,59	T(*HMS,)	1985-12-03-18.45.59.000000	Z(*ISO)
N/A	2:00 PM	T(*USA)	1985-12-03-14.00.00.000000	Z(*ISO)
N/A	1985-12-03-14.23.34.123456	Z(*ISO.)	12/03/85	D(*MDY)
N/A	1985-12-03-14.23.34.123456	Z(*ISO.)	12/03/1985	D(*USA)
N/A	1985-12-03-14.23.34.123456	Z(*ISO.)	14:23:34	T(*HMS)
N/A	1985-12-03-14.23.34.123456	Z(*ISO)	02:23 PM	T(*USA)

MOVE 例 (パート 5)

次のテーブルには、日付フィールドから文字フィールドへの移動の例が示されています。結果フィールドは演算項目 2 より大きくなっています。演算項目 1 には *ISO が入っていて、結果フィールドが次のように定義されているものとします。

```
D     Result_Fld        20S   INZ('ABCDEFGHIJabcdefghij')
```

表 58. 日付フィールドから文字フィールドへの移動

	演算項目 2		移動演算命令後の結果フィールドの値
命令コード	Value	DTZ タイプ	
MOVE	11 19 75	D(*MDY&)	'ABCDEFGHIJ1975-11-19'
MOVE(P)	11 19 75	D(*MDY&)	'1975-11-19 '
MOVEL	11 19 75	D(*MDY&)	'1975-11-19abcdefghij'
MOVEL(P)	11 19 75	D(MDY&)	'1975-11-19 '

次のテーブルには、時刻フィールドから数字フィールドへの移動の例が示されています。結果フィールドは演算項目 2 より大きくなっています。演算項目 1 には *ISO が入っていて、結果フィールドが次のように定義されているものとします。

```
D     Result_Fld        20S   INZ(1111111111111111111)
```

表 59. 時刻フィールドから数字フィールドへの移動

	演算項目 2		
命令コード	Value	DTZ タイプ	移動演算命令後の結果フィールドの値
MOVE	9:42 PM	T(*USA)	11111111111111214200
MOVE(P)	9:42 PM	T(*USA)	00000000000000214200
MOVEL	9:42 PM	T(*USA)	21420011111111111111
MOVEL(P)	9:42 PM	T(*USA)	21420000000000000000

表 60. 数字フィールドから時刻フィールドへの移動。演算項目 2 は、次のように結果フィールドより大きくなっています。強調表示されている部分は、移動される演算項目 2 フィールドの部分を示しています。

		結果フィールド	
命令コード	演算項目 2	DTZ タイプ	Value
MOVE	11:**12:13:14**	T(*EUR)	12.13.14
MOVEL	**11:12:13**:14	T(*EUR)	11.12.13

表 61. 数字フィールドからタイム・スタンプ・フィールドへの移動。演算項目 2 は、次のように結果フィールドより大きくなっています。強調表示されている部分は、移動される演算項目 2 フィールドの部分を示しています。

		結果フィールド	
命令コード	演算項目 2	DTZ タイプ	Value
MOVE	123406**18230323123420123456**	Z(*ISO)	1823-03-23-12.34.21.123456
MOVEL	**12340618230323123420**123456	Z(*ISO)	1234-06-18-23-.03.23.123420

MOVEA (配列の転送)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MOVEA (P)		ソース	ターゲット	+	–	ZB

MOVEA 演算命令は、文字、グラフィック、UCS-2、または数値を演算項目 2 から結果フィールドに転送します。(数値の移動時には、ある制限が適用される場合があります。)

MOVEA 演算命令は、次の目的で使用することができます。
- 隣接する複数の配列エレメントを単一フィールドに移動
- 単一フィールドを隣接する複数の配列エレメントに移動
- 隣接する配列エレメントを別の配列の隣接するエレメントに移動。

データの移動は、配列が索引付きでない場合には配列の最初のエレメントから、あるいは配列が索引付きの場合には指定されたエレメントから開始されます。データの移動は、最後の配列エレメントが移動されるか埋め込まれると終了します。結果フィールドに標識配列が入っていると、相互参照リストには、MOVEA 演算命令の影響を受けるすべての標識が入ります。

演算項目 2 または結果フィールドには配列が入っていなければなりません。この配列は、パック、2 進数、ゾーン、グラフィック、または文字の配列とすることができます。配列が索引付きでも、演算項目 2 と結果フィールドに同一配列を指定することはできません。

注: 文字、グラフィック、UCS-2、および数字 MOVEA 演算命令の場合は、結果を右から埋め込むために、P 命令拡張を指定することができます。

文字、グラフィック、および UCS-2 MOVEA 演算命令

演算項目 2 と結果フィールドは両方とも、文字、グラフィック、または UCS-2 として定義されていなければなりません。グラフィック・フィールドの場合に、制御仕様で CCSID(*GRAPH: *IGNORE) が指定されていない限り、グラフィックまたは UCS-2 の CCSID は同じでなければなりません。データの移動は、移動された文字数が演算項目 2 および結果フィールドによって指定されたフィールドの短い方の長さと等しくなると停止します。

MOVEA 演算命令は、配列エレメントの途中で終了する可能性があります。

可変長配列は使用できません。

数字 MOVEA 演算命令

フィールドと配列エレメントの間で移動されるデータの長さは同じでなければなりません。演算項目 2 および結果フィールドには、数字フィールド、数字配列エレメント、または数字配列が入っていなければなりません。少なくとも 1 つが配列または配列エレメントでなければなりません。数字タイプは、2 進数、パック 10 進

数、またはゾーン 10 進数とすることができます。数字タイプは、演算項目 2 と結果フィールドの間で同じである必要はありません。

結果フィールドに数字配列または数字配列エレメントが入っている場合には、演算項目 2 には数値リテラルを入れることができます。
- 数値リテラルに小数点を入れることはできません。
- 数値リテラルの長さは、結果フィールドに指定された配列エレメントのエレメントの長さより大きくすることができません。

移動時には小数点以下の桁数が無視されます。数値は、定義済みの小数点以下の桁数を考慮して変換されるわけではありません。

結果フィールドに数字配列が入っている場合には、演算項目 2 に表意定数 *BLANK、 *ALL、*ON、および *OFF を入れることはできません。

ゾーン 10 進数 MOVEA 演算命令

ゾーン 10 進形式配列を移動する場合は、次のとおりです。
- 数字配列をデータ構造のサブフィールドとして定義します。
- データ構造の数字配列を文字配列として再定義します。

その後で、MOVEA を文字配列の移動の場合と同様に使用することができます。

MOVEA での表意定数の指定

表意定数を移動するには、その表意定数の長さは指定されている配列の部分と等しくなっていなければなりません。数字配列の表意定数の場合は、各エレメント配列に入れられる符号を除き、エレメントの境界は無視されます。たとえば:
- MOVEA *BLANK ARR(X)

  エレメント X から始め、ARR の残りにはブランクが入ります。
- MOVEA *ALL'XYZ'ARRR(X)

  ARR は、4 バイトの文字エレメントをもちます。エレメント境界は無視されます。エレメント X から始め、配列の残りには 'XYZXYZXYZYXZ...' が入ります。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX          ARRY
C*  配列間の移動。索引付けなし。長さが異なる配列。
C*  エレメントの長さは同じ。
```

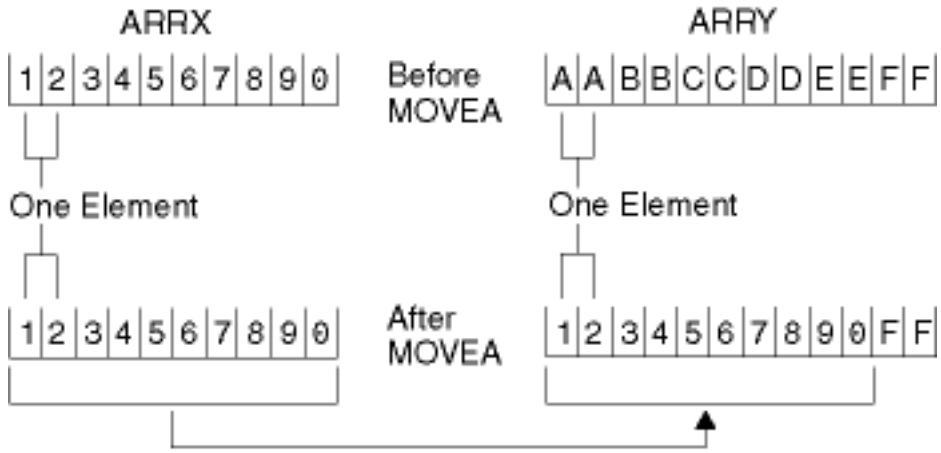


図 256. 配列間 - 異なる配列長、同一エレメント長

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX          ARRY(3)
C*  索引結果フィールドでの配列間の移動。
```

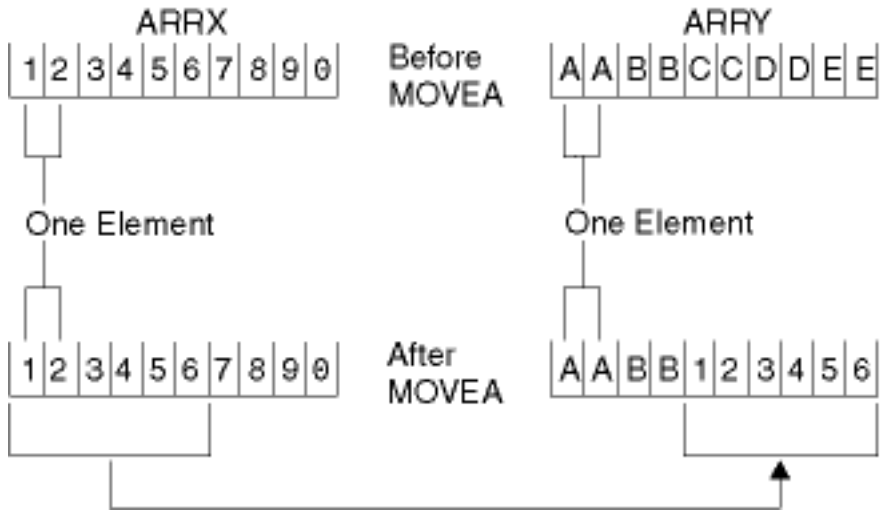


図 257. 配列間 - 索引付き結果フィールド

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX          ARRY
C*  配列間の移動。索引付けなしで長さが異なる配列
C*  索引付け。
```

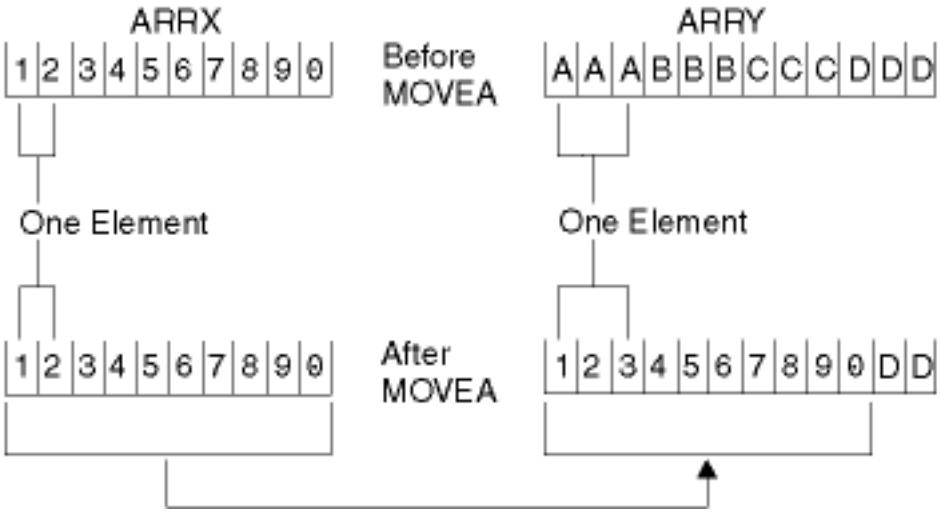


図 258. 配列間 - 長さが異なる配列エレメント

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX(4)       ARRY
C*  配列間の移動。配列エレメントの長さが異なる演算項目 2 で
C*  索引付け。
```

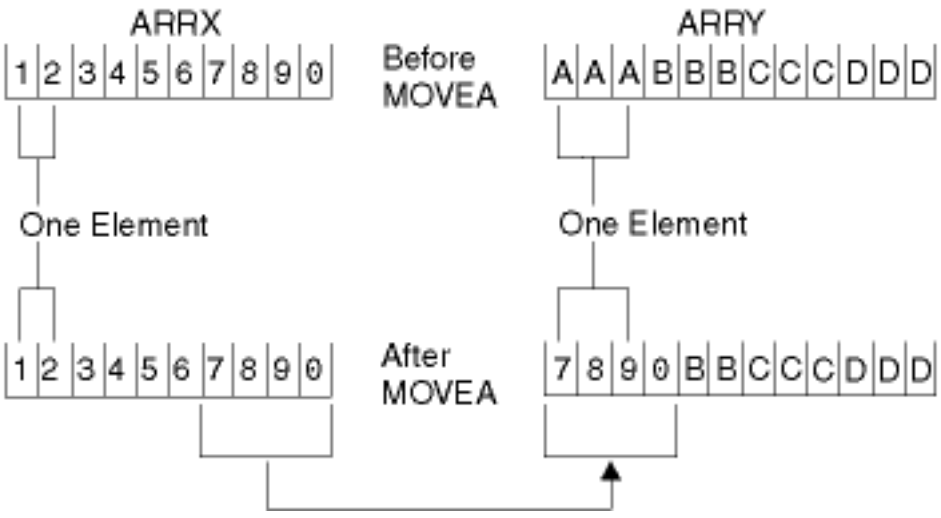


図 259. 配列間 - 索引付き演算項目 2

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     FIELDA         ARRY
C*   フィールドと配列間の移動。配列での索引付けなし。
```

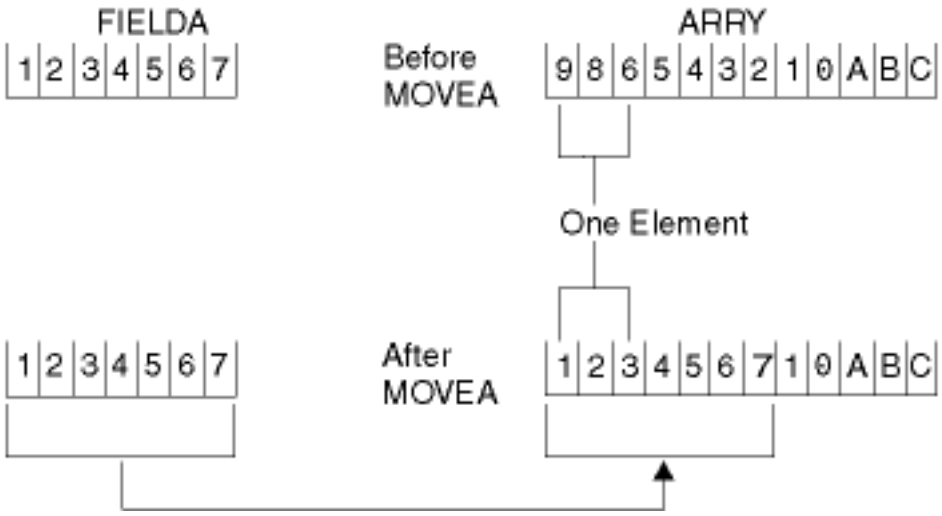


図 260. 配列間 - 索引付けなし

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の例では、N=3。変数索引付けされた配列とフィールド間の
C* 移動。
C                   MOVEA     ARRX(N)        FIELD
C*
```

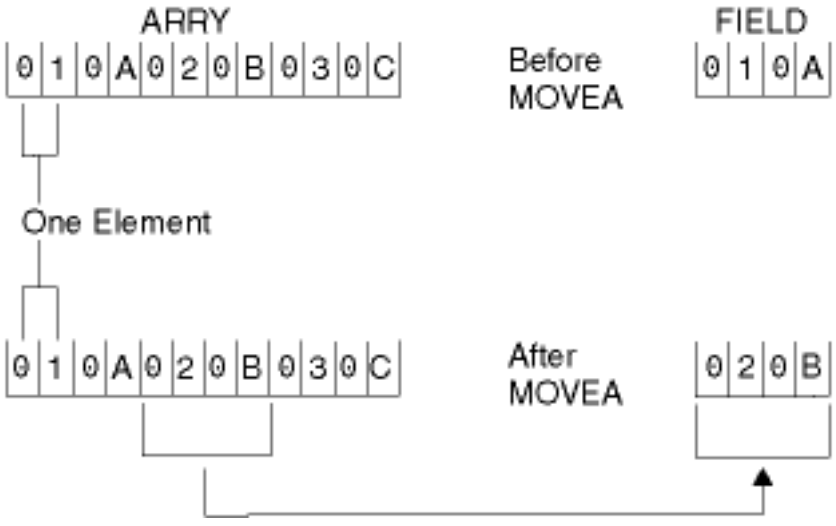


図 261. 配列とフィールド間 - 変数索引付け

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRB          ARRZ
C*
C* 数字エレメントが示されている配列間の移動。
```

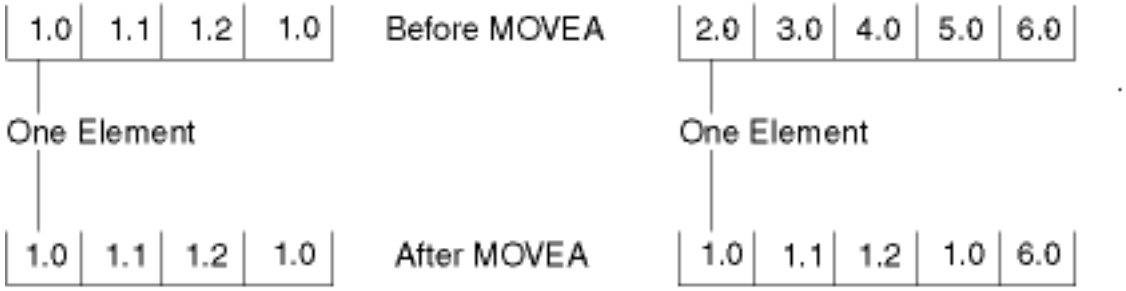


図 262. 配列間 - 数字エレメント

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA(P)  ARRX          ARRY
C*  埋め込みがある配列間の移動。索引付けなし。エレメントの長さが
C*  同じ異なる長さの配列。
```

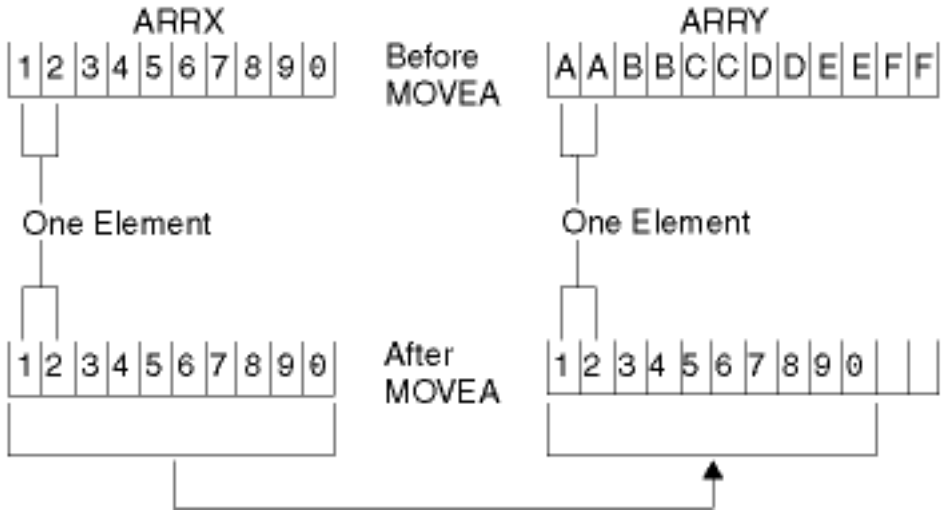


図 263. 配列間 - 埋め込みあり

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA(P)  ARRB          ARRZ
C*
C* 埋め込みありの数字エレメントが示されている配列間の移動。
```

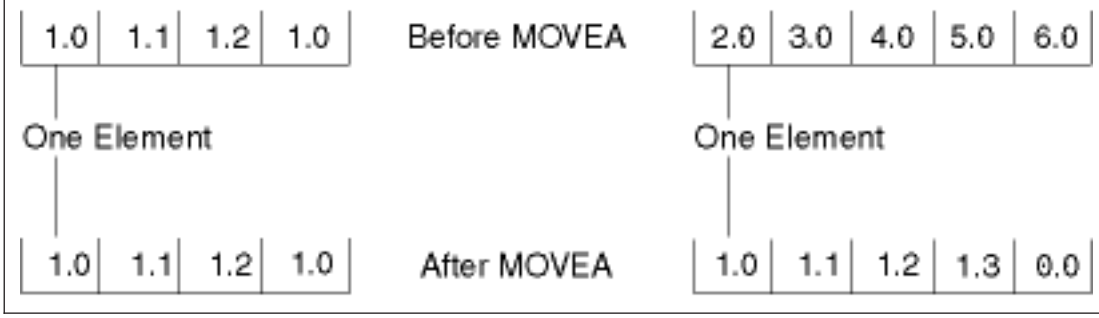


図 264. 配列間 - 埋め込みありの数字エレメント

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA(P)  ARRX(3)       ARRY
C*  埋め込みがある配列間の移動。索引付けなし。エレメントの長さが
C*  異なり長さが異なる配列。
```

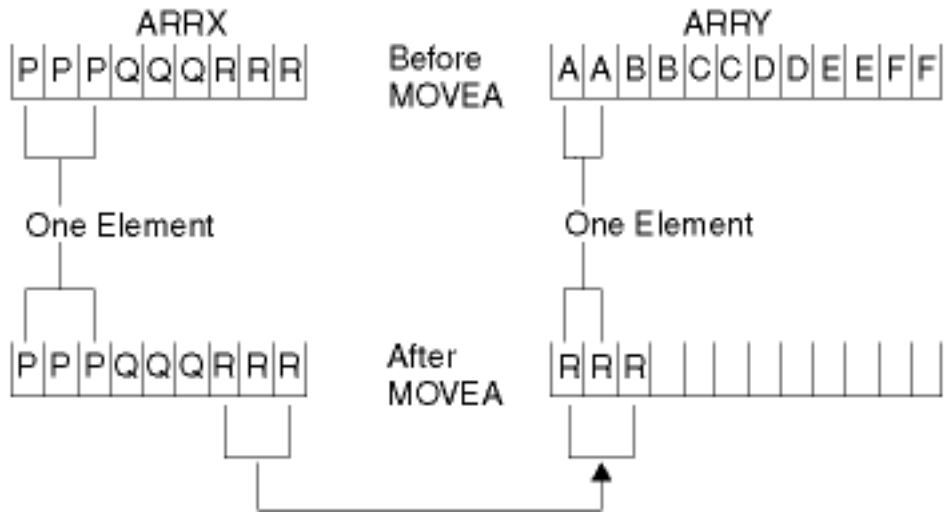


図 265. 配列間 - 埋め込みあり、索引付けなし

MOVEL (左につめて転送)

フリー・フォーム構文	(使用できません。EVAL 命令コードを使用するか、%DATE、%TIME、%TIMESTAMP、%CHAR、%UCS2、%GRAPH などの組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MOVEL (P)	データ属性	ソース・フィールド	ターゲット・フィールド	+	–	ZB

MOVEL 演算命令は、演算項目 2 から結果フィールドに文字を転送します。移動は、演算項目 2の左端の文字から開始されます。

演算項目 1 を指定する場合には、日付形式または時刻形式が入っていなければなりません。これは、演算命令のソースまたはターゲットである文字または数字フィールドの形式を指定します。

結果フィールドが配列である場合には、演算結果標識を指定することができません。結果フィールドが配列エレメントまたは非配列フィールドである場合には、それを指定することができます。

ソースまたはターゲットが文字フィールドの場合には、任意選択で演算項目 1 の形式の後に区切り文字を指示することができます。使用できるのはその形式に有効な区切り文字だけです。

演算項目 2 が *DATE または UDATE で、結果が日付フィールドの場合には、演算項目 1 は必須でありません。演算項目 1 に日付形式が入っている場合には、制御仕様で DATEDIT キーワードによって指定されている *DATE または UDATE の形式と互換になっていなければなりません。

演算項目 2 が結果フィールドより長い場合には、長さを超えている演算項目 2 の右端の文字は移動されません。結果フィールドが演算項目 2 より長い場合には、埋め込みが指定されていない限り、長さを超えている結果フィールドの右端の文字は未変更になっています。

浮動数字フィールドおよびリテラルは、演算項目 2 として使用できません。

演算項目 2 が UCS-2 で、結果フィールドが文字であるか、あるいは演算項目 2 が文字で、結果フィールドが UCS-2 である場合には、移動される文字数は可変です。たとえば、 UCS-2 の 5 文字は次に変換されます。

- 単一バイトで 5 文字
- 2 バイト文字で 5 文字
- 単一バイトと 2 バイト文字の組み合わせ

注: データを数字フィールドに移動するときに、演算項目 2 の長さが結果フィールドの長さ以上である場合を除き、結果フィールドの符号 (+ または –) は保存されます。この場合には、演算項目 2 の符号が結果フィールドの符号として使用されます。

次の節には、演算項目 2 および結果フィールドの長さを基にした MOVEL 演算命令の規則が要約されています。

演算項目 2 の長さは結果フィールドと同じ

演算項目 2 と結果フィールドの長さは次のように同じです。

- 演算項目 2 と結果フィールドが両方とも数字である場合には、符号は右端の桁に移動されます。
- 演算項目 2 と結果フィールドが両方とも文字である場合には、すべての文字が移動されます。
- 演算項目 2 が数字で、結果フィールドが文字である場合には、符号は右端の桁に移動されます。
- 演算項目 2 が文字で、結果フィールドが数字である場合には、演算項目 2 の右端の桁のゾーンが負ゾーンであれば、負ゾーンが結果フィールドの右端の桁に移動されます。しかし、演算項目 2 の右端の桁のゾーンが負ゾーンでない場合には、正ゾーンが結果フィールドの右端の桁に移動されます。数字部分はそれと対応する数字に変換されます。数字部分が有効数字でない場合には、データ例外エラーが起こります。
- 演算項目 2 と結果フィールドが両方ともグラフィックまたは UCS-2 である場合には、すべてのグラフィックまたは UCS-2 文字が移動されます。
- 演算項目 2 がグラフィックで、結果フィールドが文字である場合には、すべてのグラフィック文字が移動されます。
- 演算項目 2 が文字で、結果フィールドがグラフィックである場合には、すべての文字が移動されます。

演算項目 2 が結果フィールドより長い

演算項目 2 が結果フィールドより長い:

- 演算項目 2 と結果フィールドが両方とも数字である場合には、演算項目 2 の右端の符号が結果フィールドの右端の桁に移動されます。
- 演算項目 2 が数字で、結果フィールドが文字である場合には、結果フィールドには数字しか入りません。移動されるのは、結果フィールドを埋めるために必要な文字数だけです。
- 演算項目 2 が文字で、結果フィールドが数字である場合には、演算項目 2 の右端の桁のゾーンが負ゾーンであれば、負ゾーンが結果フィールドの右端の桁に移動されます。しかし、演算項目 2 の右端の桁のゾーンが負ゾーンでない場合には、正ゾーンが結果フィールドの右端の桁に移動されます。結果フィールドのその他の桁には数字しか入りません。
- 演算項目 2 と結果フィールドが両方ともグラフィックまたは UCS-2 である場合には、結果フィールドを埋めるために必要なグラフィックまたは UCS-2 の文字の数しか移動されません。
- 演算項目 2 がグラフィックで、結果フィールドが文字である場合には、グラフィック・データが切り捨てられます。
- 演算項目 2 が文字で、結果がグラフィックである場合には、文字データが切り捨てられます。

注: 長さを超えている演算項目 2 の右端の文字は移動されません。結果フィールドが演算項目 2 より長い場合には、埋め込みが指定されていない限り、長さを超えている結果フィールドの右端の文字は未変更になっています。

演算項目 2 が結果フィールドより短い

演算項目 2 が結果フィールドより短い:

- 演算項目 2 が数字または文字のいずれか一方で、結果フィールドが数字である場合には、演算項目 2の数字部分が結果フィールドの左端の桁の内容を置き換えます。結果フィールドの右端の桁の符号は変更されません。
- 演算項目 2 が数字または文字のいずれか一方で、結果フィールドが文字データである場合には、演算項目 2 の文字が結果フィールドの左端の桁の同等の数字を置き換えます。結果フィールドの右端の桁のゾーンは変更されません。

演算項目 2 は結果フィールドより短く、P が指定されている

演算項目 2 が結果フィールドより短くなっていて、P が指定されている場合には、次のとおりです。

- 移動は『演算項目 2 が結果フィールドより短い』の説明通りに実行されます
- 結果フィールドは右から埋め込まれます。

可変長文字、グラフィック、または UCS-2 データの移動時に、可変長フィールドは、現行の長さと同じ固定長フィールドと全く同様に処理されます。たとえば、270 - 275 を参照してください。

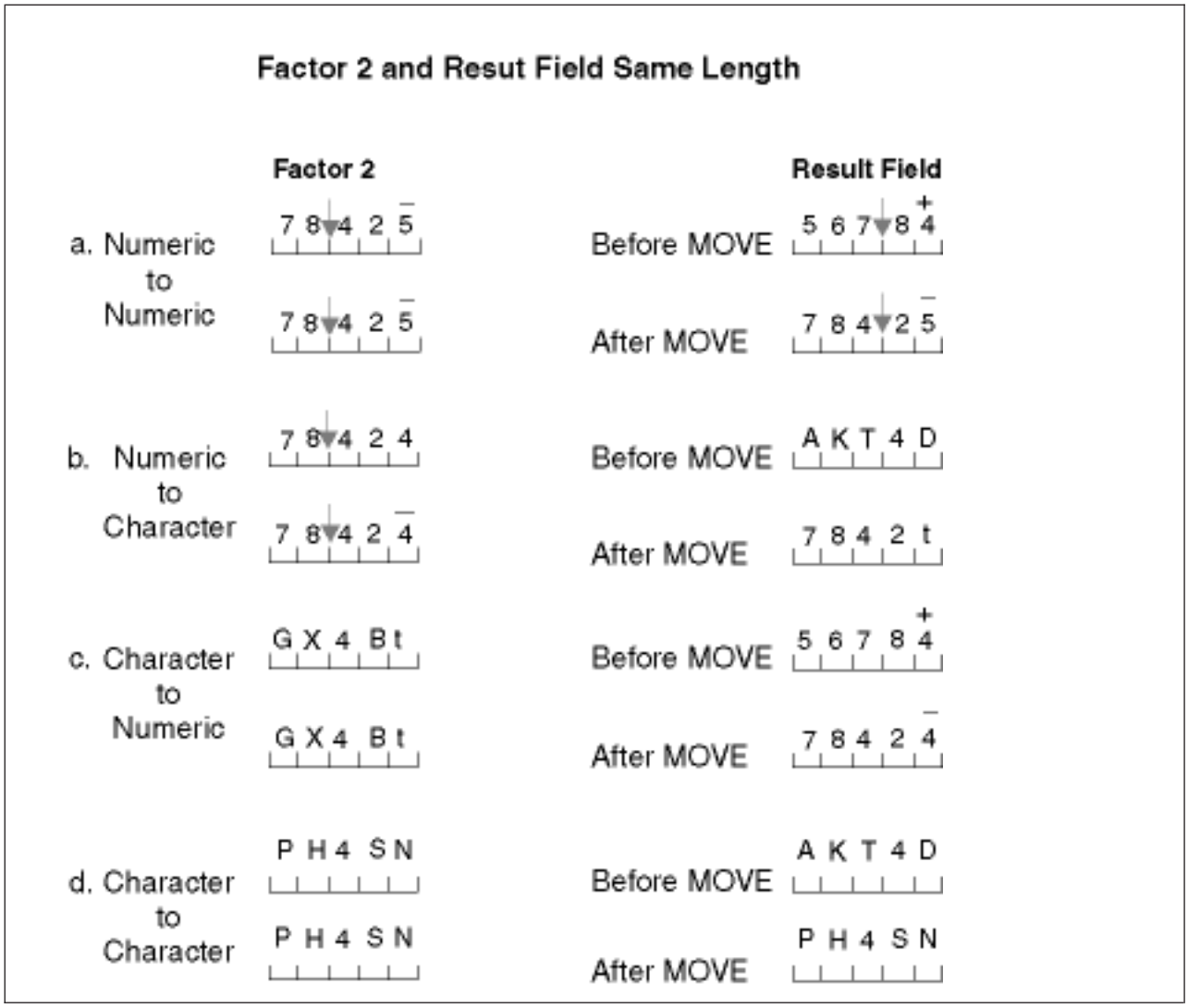


図 266. 演算項目 2 が結果フィールドの長さと同じ

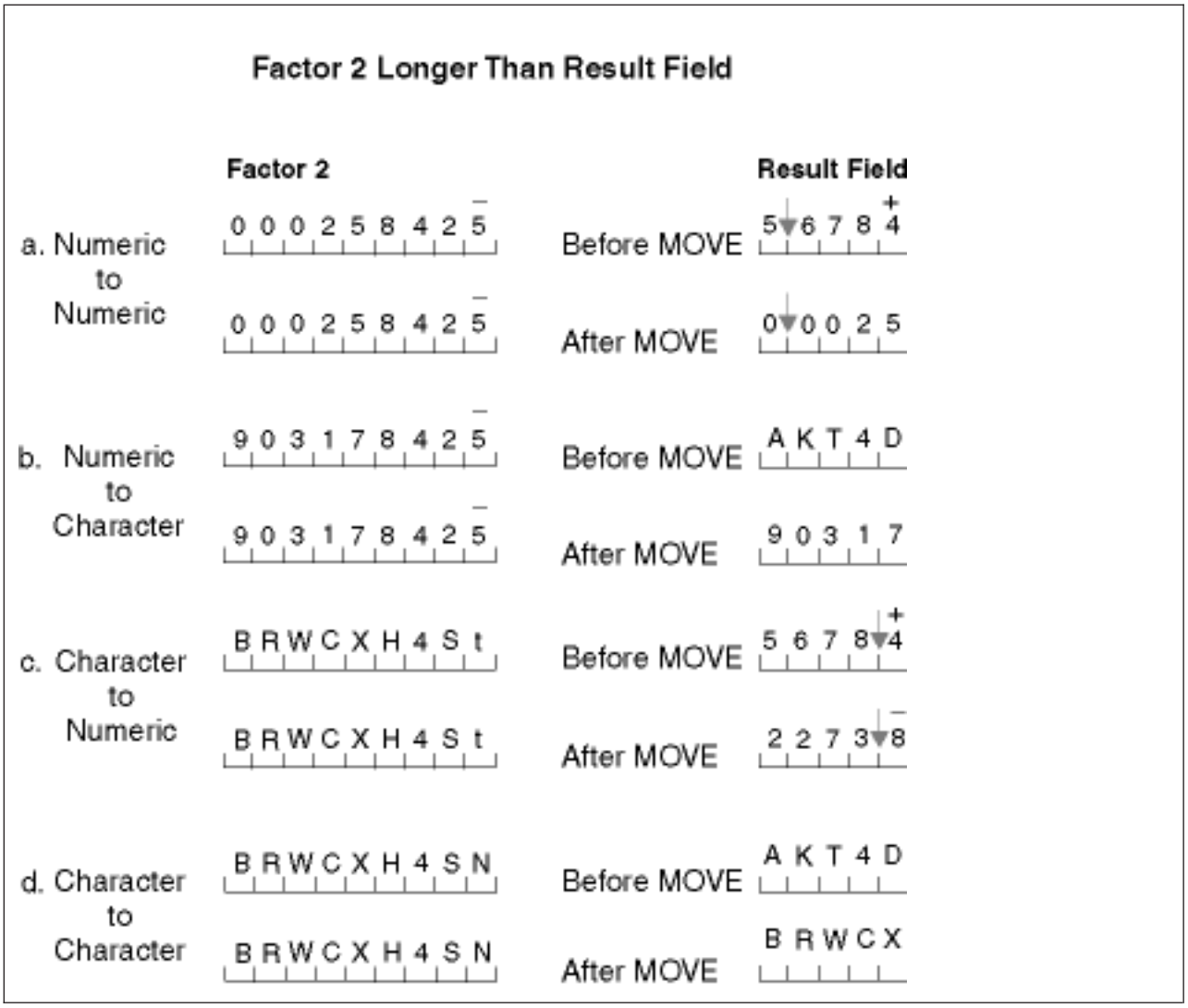


図 267. 演算項目 2 が結果フィールドより長い

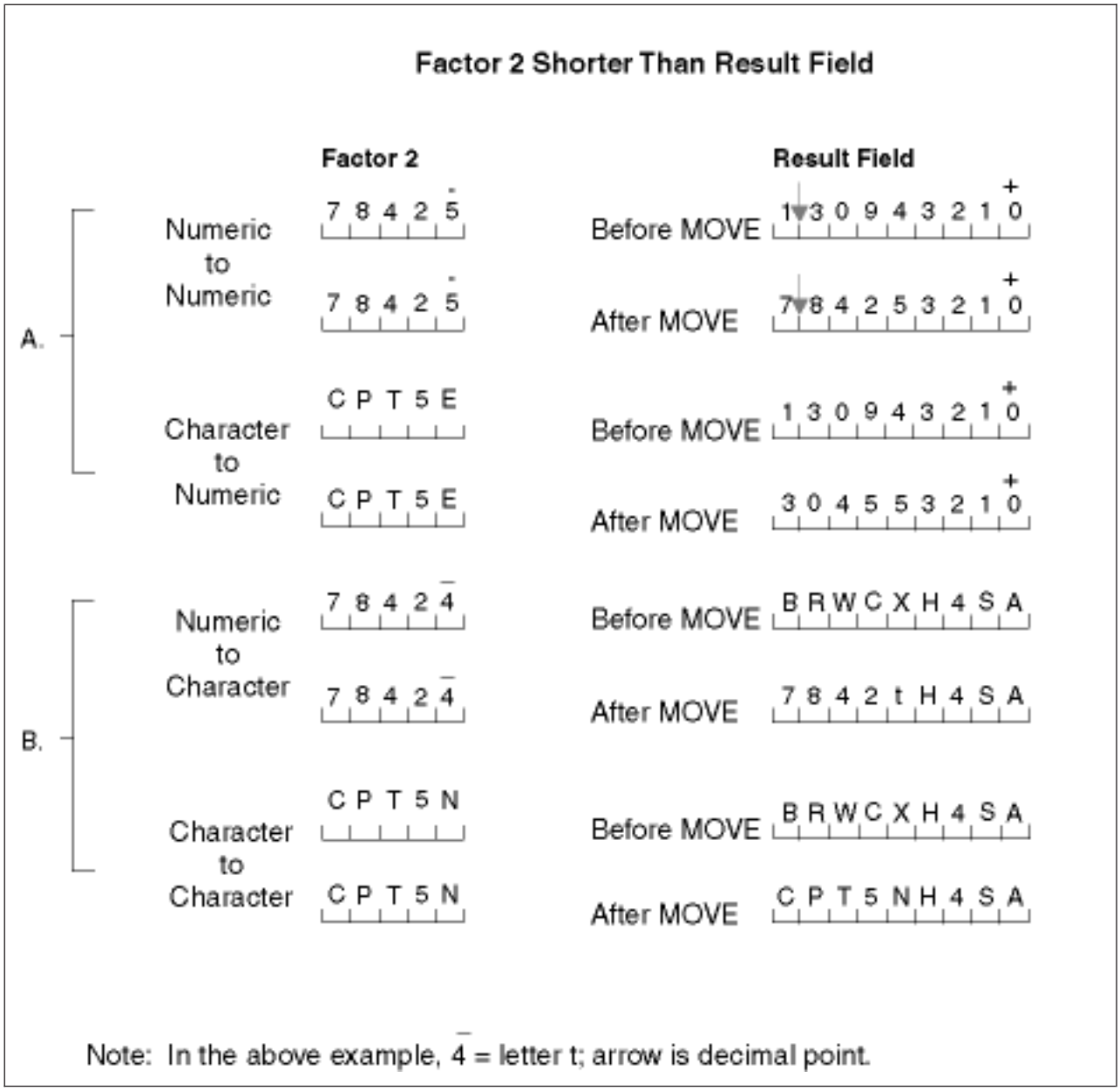


図 268. 演算項目 2 が結果フィールドより短い

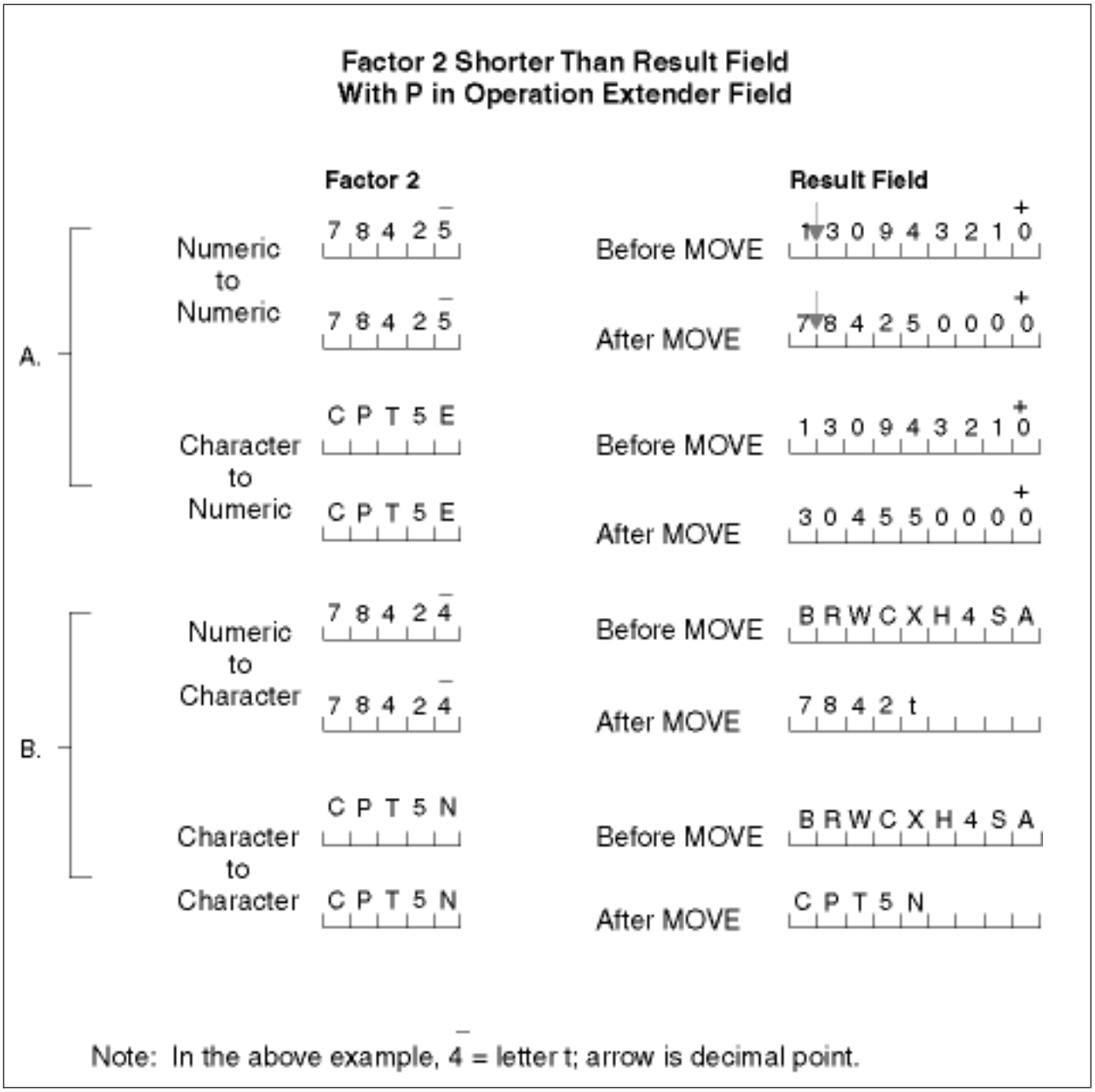


図 269. 演算項目 2 が結果フィールドより短く、P が指定されている

MOVEL 例: 可変長 / 固定長の移動

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 可変長から可変長への文字フィールドの
D* MOVEL(P) の例
D*
D var5a          S              5A   INZ('ABCDE') VARYING
D var5b          S              5A   INZ('ABCDE') VARYING
D var5c          S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGH') VARYING
D var15b         S             15A   INZ('FGH') VARYING
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVEL     var15a        var5a
C* var5a = 'FGHDE' (長さ =5)
C                   MOVEL     var10         var5b
C* var5b = '01234' (長さ =5)
C                   MOVEL     var5c         var15a
C* var15a = 'ABC' (長さ =3)
C                   MOVEL     var10         var15b
C* var15b = '012' (長さ =3)
```

図 270. MOVEL: 可変長フィールドから可変長フィールドへ

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 可変長から固定長への文字フィールドの
D* MOVEL(P) の例
D*
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGH') VARYING
D fix5a          S              5A   INZ('MNOPQ')
D fix5b          S              5A   INZ('MNOPQ')
D fix5c          S              5A   INZ('MNOPQ')
D fix10          S             10A   INZ('')
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVEL     var5          fix5a
C* fix5a = 'ABCDE'
C                   MOVEL     var10         fix5b
C* fix5b = '01234'
C                   MOVEL     var15         fix5c
C* fix5c = 'FGHPQ'
```

図 271. MOVEL: 可変長フィールドから固定長フィールドへ

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 固定長から可変長への文字フィールドの
D* MOVEL(P) の例
D*
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGHIJKLMNOPQR') VARYING
D var15b         S             15A   INZ('WXYZ') VARYING
D fix10          S             10A   INZ('PQRSTUVWXY')
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVEL     fix10         var5
C* var5 = 'PQRST' (長さ =5)
C                   MOVEL     fix10         var10
C* var10 = 'PQRSTUVWXY' (長さ =10)
C                   MOVEL     fix10         var15a
C* var15a = 'PQRSTUVWXYPQR' (長さ =13)
C                   MOVEL     fix10         var15b
C* var15b = 'PQRS' (長さ =4)
```

図 272. MOVEL: 固定長フィールドから可変長フィールドへ

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 可変長から可変長への文字フィールドの
D* MOVEL(P) の例
D*
D var5a          S              5A   INZ('ABCDE') VARYING
D var5b          S              5A   INZ('ABCDE') VARYING
D var5c          S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGH') VARYING
D var15b         S             15A   INZ('FGH') VARYING
D var15c         S             15A   INZ('FGHIJKLMN') VARYING
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVEL(P)  var15a        var5a
C* var5a = 'FGH  ' (長さ =5)
C                   MOVEL(P)  var10         var5b
C* var5b = '01234' (長さ =5)
C                   MOVEL(P)  var5c         var15b
C* var15b = 'ABC' (長さ =3)
C                   MOVEL(P)  var15a        var15c
C* var15c = 'FGH      ' (長さ =9)
```

図 273. MOVEL(P): 可変長フィールドから可変長フィールドへ

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 可変長から固定長への文字フィールドの
D* MOVEL(P) の例
D*
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGH') VARYING
D fix5a          S              5A   INZ('MNOPQ')
D fix5b          S              5A   INZ('MNOPQ')
D fix5c          S              5A   INZ('MNOPQ')
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVEL(P)  var5          fix5a
C* fix5a = 'ABCDE'
C                   MOVEL(P)  var10         fix5b
C* fix5b = '01234'
C                   MOVEL(P)  var15         fix5c
C* fix5c = 'FGH  '
```

図 274. MOVEL(P): 可変長フィールドから固定長フィールドへ

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
D*
D* 固定長から可変長への文字フィールドの
D* MOVEL(P) の例
D*
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGHIJKLMNOPQR') VARYING
D var15b         S             15A   INZ('FGH') VARYING
D fix5           S             10A   INZ('.....')
D fix10          S             10A   INZ('PQRSTUVWXY')
D*
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C                   MOVEL(P)  fix10         var5
C* var5 = 'PQRST' (長さ =5)
C                   MOVEL(P)  fix5          var10
C* var10 = '.....     ' (長さ =10)
C                   MOVEL(P)  fix10         var15a
 * var15a = 'PQRSTUVWXY   ' (長さ =13)
C                   MOVEL(P)  fix10         var15b
C* var15b = 'PQR' (長さ =3)
```

図 275. MOVEL(P): 固定長フィールドから可変長フィールドへ

MULT (乗算)

フリー・フォーム構文	(使用できません。* 演算子を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MULT (H)	被乗数	乗数	積	+	–	Z

演算項目 1 が指定されていると、演算項目 1 が演算項目 2 によって乗算されて、積が結果フィールドに入れられます。演算項目 1 が指定されていない場合には、演算項目 2 が結果フィールドによって乗算されて、積が結果フィールドに入れられます。

演算項目 1 と演算項目 2 は数字でなければならず、それぞれには配列、配列エレメント、フィールド、表意定数、リテラル、名前付き固定情報、サブフィールド、またはテーブル名を入れることができます。

結果フィールドは、積を保持するのに十分な大きさになっていなければなりません。次の規則は、結果フィールドの最大長を判別するために使用します。結果フィールドは、演算項目 1 の長さに演算項目 2 の長さを加算した長さと等しくなります。結果フィールドは数字でなければなりませんが、名前付き固定情報またはリテラルとすることはできません。結果を丸めるために四捨五入を指定することができます。

352 ページの『算術演算』には、算術演算を指定するための一般規則が説明されています。

355 ページの図 114 には、MULT 演算命令の例が示されています。

MVR (剰余の移動)

フリー・フォーム構文	(使用できません。%REM 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MVR			剰余	+	–	Z

MVR 演算命令は、直前の DIV 演算命令からの剰余を結果フィールドに指定された別個のフィールドに移動します。MVR 演算命令のすぐ前には DIV 演算命令がなければなりません。条件標識を使用する場合には、MVR 演算命令が DIV 演算命令の直後に指定されていなければなりません。結果フィールドは数字でなければならず、配列、配列エレメント、サブフィールド、またはテーブルを入れることができます。

DIV 演算命令で小数点以下の桁数がある演算項目を使用する場合には、結果フィールドに十分な余地を残しておいてください。小数点以下の有効桁数は、次より大きくなります。

- 直前の除算演算命令の演算項目 1 の小数点以下の桁数
- 直前の除算演算命令の演算項目 2 の小数点以下の桁数と結果フィールドの小数点以下の桁数の和。

剰余の整数桁の最大数は、直前の除算演算命令の演算項目 2 の整数桁数と等しくなります。

剰余の符号 (+ または -) は被除数 (演算項目 1) と同じです。

直後に MVR 演算命令が続いている DIV 演算命令に四捨五入を指定することはできません。直前の除算演算命令の結果フィールドに配列が指定されている場合には、MVR 演算命令を指定することはできません。また、直前の DIV 演算命令に浮動小数点オペランドが少なくとも 1 個指定されている場合にも、MVR 演算命令を使用することができません。

352 ページの『算術演算』には、算術演算を指定するための一般規則が説明されています。

355 ページの図 114 には、MVR 演算命令の例が示されています。

OCCUR (データ構造のオカレンスの設定 / 取り出し)

フリー・フォーム構文	(使用できません。%OCCUR 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
OCCUR (E)	オカレンス値	データ構造	オカレンス値	_	ER	_

OCCUR 演算命令は、プログラム内で次に使用されるデータ構造のオカレンスを指定します。

複数オカレンスがあるデータ構造またはそのデータ構造のサブフィールドが演算命令で指定されている場合には、OCCUR 演算命令が指定されるまで、データ構造の最初のオカレンスが使用されます。 OCCUR 演算命令が指定された後で、OCCUR 演算命令によって設定されたデータ構造が使用されます。

演算項目 1 を指定する場合には、数字、ゼロの小数点以下の桁数のリテラル、フィールド名、名前付き固定情報、またはデータ構造名が入っていなければなりません。演算項目 1 は、演算項目 2 に指定されたデータ構造のオカレンス設定をします。演算項目 1 が指定されていない場合には、演算項目 2 のデータ構造の現行オカレンスの値が、OCCUR 演算命令の実行時に結果フィールドに入れられます。

演算項目 1 がデータ構造名である場合には、複数回繰り返しデータ構造でなければなりません。演算項目 2 のデータ構造のオカレンスを設定するために、演算項目 1 のデータ構造の現行オカレンスが使用されます。

演算項目 2 は、複数回繰り返しデータ構造の名前でなければなりません。

結果フィールドを指定する場合には、小数点以下の桁数なしの数字フィールド名でなければなりません。オプショナルで演算項目 1 に指定されている任意の値またはデータ構造によって設定された後で、演算項目 2 に指定されたデータ構造の現行オカレンスの値が結果フィールドに入れられます。

注: 演算項目 1 または結果フィールドの少なくとも 1 つが指定されていなければなりません。

オカレンスがデータ構造に設定された有効範囲外になっている場合には、エラーが起こり、演算項目 2 のデータ構造のオカレンスは、OCCUR 演算命令が処理される前と同じままになっています。

OCCUR 例外 (プログラム状況コード 122) を処理するために、命令コード拡張 'E' またはエラー標識 ERのいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

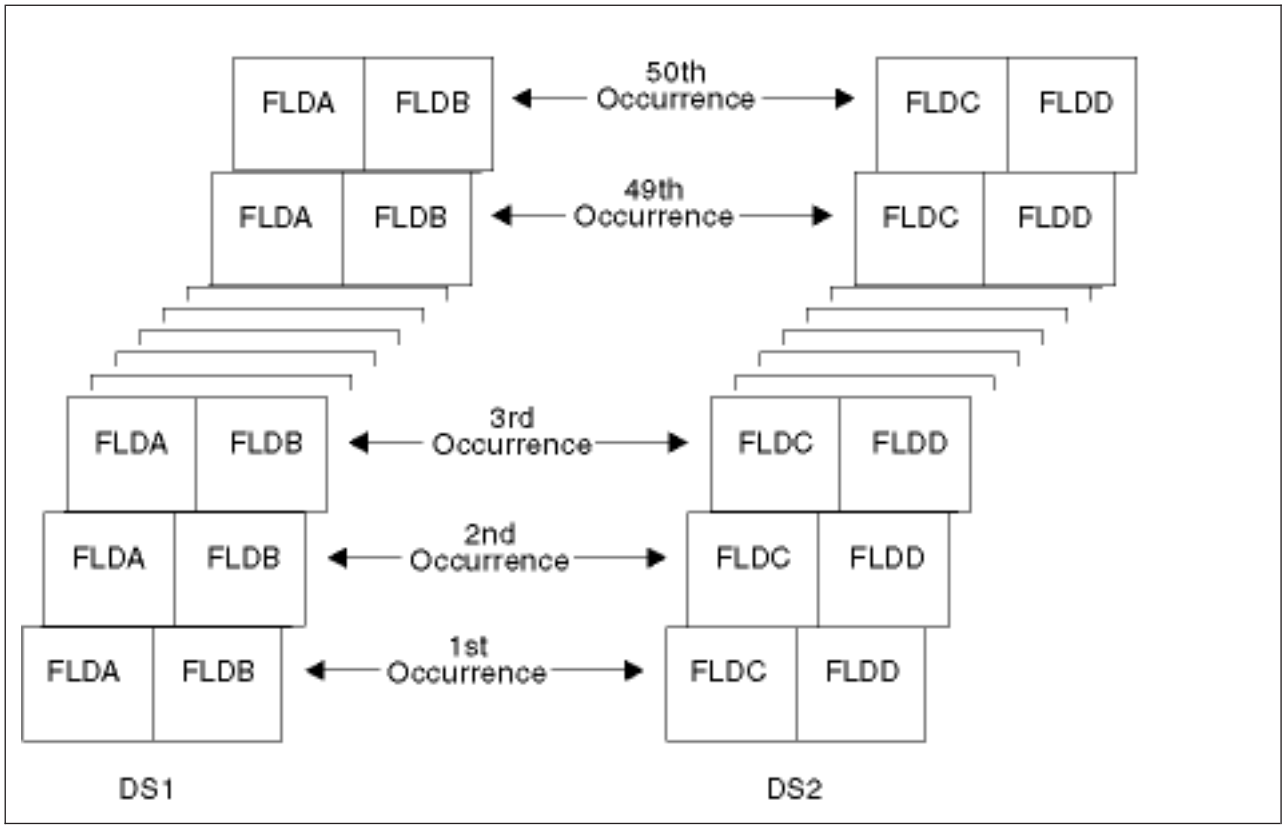


図 276. OCCUR 演算命令の例

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D*
D*  DS1 および DS2 は複数回繰り返しデータ構造です。
D*  各データ構造には 50 個のオカレンスがあります。
D DS1             DS                  OCCURS(50)
D  FLDA                    1      5
D  FLDB                    6     80
D*
D DS2             DS                  OCCURS(50)
D  FLDC                    1      6
D  FLDD                    7     11
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*  DS1 は 3 番目のオカレンスに設定されます。3 番目のオカレンスの
C*  サブフィールド FLDA および FLDB がこれで使用できます。MOVE お
C*  よび Z-ADD 演算命令はそれぞれ DS1 の 3 番目のオカレンスの FLDA
C*  および FLDB の内容を変更します。
C
C     3             OCCUR     DS1
C                   MOVE      'ABCDE'       FLDA
C                   Z-ADD     22            FLDB
C*
C*  DS1 は 4 番目のオカレンスに設定されます。DS1 の 4 番目の
C*  オカレンスの FLDA および FLDB の値を使用して、MOVE 演算
C*  命令は FLDA の内容を結果フィールド FLDX に入れて、
C*  Z-ADD 演算命令は FLDB の内容を結果フィールド FLDY に
C*  入れます。
C
C     4             OCCUR     DS1
C                   MOVE      FLDA          FLDX
C                   Z-ADD     FLDB          FLDY
C*
C*  DS1 はフィールド X に指定されたオカレンスに設定されます。
C*  たとえば、X = 10 の場合には、DS1 は 10 番目のオカレンスに設定
C*  されます。
C     X             OCCUR     DS1
C*
C*  DS1 は DS2 の現行オカレンスに設定されます。たとえば、DS2
C*  の現行オカレンスが 12 番目のである場合には、DSI は 12 番
C*  目のオカレンスに設定されます。
C     DS2           OCCUR     DS1
```

図 277. OCCUR 演算命令の例

```
C*
C*  DS1 の現行オカレンスの値が結果フィールド Z に入れられ
C*  ます。フィールド Z は小数点以下の桁数がゼロの数字でなけ
C*  ればなりません。たとえば、DS1 の現行オカレンスが 15 で
C*  ある場合には、フィールド Z には値 15 が入ります。
C                   OCCUR     DS1           Z
C
C*  DS1 は DS2 の現行オカレンスに設定されます。その後で、DS1
C*  の現行オカレンスの値が結果フィールド Z に移動されます。
C*  たとえば、DS2 の現行オカレンスが 5 番目のオカレンスであ
C*  る場合には、DS1 は 5 番目のオカレンスに設定されます。結
C*  果のフィールド Z には値 5 が入ります。
C
C     DS2           OCCUR     DS1           Z
C*
C*  DS1 が X の現行オカレンスに設定されます。たとえば、X = 15
C*  の場合には、DS1 は 15 番目のオカレンスに設定されます。
C*  X が 1 より小か、50 より大の場合には、エラーが起こって
C*  %ERROR は '1' を戻すように設定されます。
C*  %ERROR が '1' を戻すと、LR 標識はオンにセットされます。
C
C     X             OCCUR (E) DS1
C                   IF        %ERROR
C                   SETON                                        LR
C                   ENDIF
```

図 278. OCCUR 演算命令の例

ON-ERROR (エラー処理)

フリー・フォーム構文	ON-ERROR {*exception-id1* {:*exception-id2*...}}

コード	演算項目 1	拡張演算項目 2
ON-ERROR		例外 ID のリスト

どのエラーが例外 ID のリストでの (*exception-id1*:*exception-id2*...) エラー処理ブロック処理を条件付けるかを指定します。コロンで区切りながら以下のものをどのように組み合わせても使用できます。

nnnnn 状況コード

***PROGRAM** 00100 ～ 00999 のすべてのプログラム・エラー状況コードを処理します

***FILE** 01000 ～ 09999 のすべてのファイル・エラー状況コードを処理します

***ALL** 00100 ～ 09999 プログラム・エラーとファイル・エラーの両方のコードを処理します。これがデフォルトです。

0100 ～ 09999 の範囲外の状況コード (たとえば 0 ～ 99) は、モニターされません。エラー処理グループにこれらの値を指定することはできません。また使用中のコンパイラーの特定バージョンに有効でない状況コードを指定することもできません。

同じ状況コードが複数のエラー処理グループに関係している場合には、最初のものだけが使用されます。この理由で、特定の状況コードの後に *ALL などの特殊値を指定してください。

エラー処理グループ内で起こったエラーは、どれもモニター・グループによって処理されることはありません。エラーを処理するには、エラー処理グループ内でモニター・グループを指定できます。

エラー処理ブロック内のすべてのステートメントが処理されると、制御は、ENDMON ステートメントの後のステートメントに渡されます。

ON-ERROR ステートメントの例については、 585 ページの『MONITOR (モニター・グループの開始)』 を参照してください。

OPEN (処理用ファイルのオープン)

フリー・フォーム構文	OPEN{(E)} *file-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
OPEN (E)		ファイル名		_	ER	_

明示 OPEN 演算命令は、*file-name* オペランドに指定されたファイルをオープンします。ファイルは、ローカル・ファイルまたは OS/400 ファイルにすることができます。ファイルが、ローカル・ファイルとして定義されている場合および OPEN 演算命令の実行時に存在していない場合には、ローカル・ファイルが作成されます。リモート・ファイルは、OPEN 演算命令の実行時に存在していなければならず、そうでない場合には、作成されません。

ファイルは、テーブル・ファイルとすることができません。ユーザー・プログラムがファイルのオープン時点を制御できるようにするには、ファイル仕様で USROPN キーワードを指定してください。USROPN キーワードについて詳しくは、237 ページの『第 17 章 ファイル仕様』を参照してください。

ファイルがオープンされてから、CLOSE 演算命令によってクローズされると、そのファイルは、OPEN 演算命令で再オープンすることができます。ファイル仕様で USROPN キーワードは必須ではありません。 USROPNキーワードがファイル仕様に指定されていない場合には、そのファイルはプログラム初期化時にオープンされます。OPEN 演算命令がすでにオープンされているファイルに指定されると、エラーが起こります。

OPEN 演算命令の前にファイルがクローズされている限り、同一ファイルに対して複数の OPEN 演算命令が有効です。

固定形式構文の演算の 73 ～ 74 桁目に結果の標識が指定されている場合には、その標識は、OPEN 演算命令時にエラーが起こるとオンにセットされます。

OPEN 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

```
 *...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IT.A.FRlen+......A.Device+.Keywords++++++++++++++++++++++++++++++
FEXCEPTN   O    E             DISK    REMOTE USROPN
FFILEX     IF   E             DISK    REMOTE
 *...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  標識 97 がオンで、標識 98 がオフの場合には、明示の OPEN
C*  演算命令は EXCEPTN ファイルを処理のためにオープンします。
C*  ファイル仕様で EXCEPTN ファイルに指定された USROPN キー
C*  ワードがあることに注意してください。
C*  OPEN 演算命令が失敗すると、%ERROR は '1' を戻すように設定されます。
C*
C                   IF        *in97 and not *in98
C                   OPEN(E)   EXCEPTN
C                   IF        not %ERROR
C                   WRITE     ERREC
C                   ENDIF
C                   ENDIF
C*
C*  FILEX はプログラム初期化時にオープンされます。明示の CLOSE
C*  演算命令は、制御が RTNX に渡される前に FILEX をクローズしま
C*  戻るときに、OPEN 演算命令はファイルを再オープンします。
C*  USROPN キーワードが FILEX に指定されていないので、ファイルは
C*  プログラム初期化時にオープンされます。
C*
C                   CLOSE     FILEX
C                   CALL      'RTNX'
C                   OPEN      FILEX
```

図 279. CLOSE 演算命令での OPEN 演算命令

ORxx (OR)

フリー・フォーム構文	(使用できません。OR 演算子を使用してください)

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
ORxx	被比較数	被比較数				

ORxx 演算命令は、DOUxx、DOWxx、IFxx、WHENxx、および ANDxx 演算命令で任意指定です。ORxxは、DOUxx、DOWxx、IFxx、WHENxx、ANDxx、または ORxx ステートメントのすぐ後に指定されます。 ORxx は、DOUxx、DOWxx、IFxx、および WHENxx 演算命令にさらに多くの複合条件を指定するために使用します。条件付け標識項目 (9 - 11 桁目) は使用できません。

演算項目 1 と演算項目 2 には、リテラル、名前付き固定情報、表意定数、テーブル名、配列エレメント、データ構造名、またはフィールド名が入っていなければなりません。演算項目 1 と演算項目 2 のタイプは同じでなければなりません。演算項目 1 と演算項目 2 の比較は、比較演算命令に示されているものと同じ規則に従います。

361 ページの『比較命令』には、比較演算命令を指定するための一般規則が説明されています。

540 ページの図 219 には、DOUxx 演算命令での ORxx および ANDxx 演算命令の例が示されています。

OTHER (その他の場合の選択)

フリー・フォーム構文	OTHER

コード	演算項目 1	演算項目 2	結果フィールド	標識		
OTHER						

OTHER 演算命令は、WHENxx または WHEN 条件が SELECT グループ内で満たされない場合に処理される演算命令の順序で始まります。この順序は ENDSL または END 演算命令で終了します。

OTHER 演算命令を使用する場合に覚えておく規則は、次のとおりです。
- OTHER 演算命令は、SELECT グループでは任意指定です。
- SELECT グループで指定できる OTHER 演算命令は 1 つだけです。
- 同一 SELECT グループ内の OTHER 演算命令の後には WHENxx または WHEN 演算命令を指定することができません。
- OTHER グループ内の演算命令の順序はヌルであってもかまいません。効果は、OTHER ステートメントを指定しないことと同じです。
- 条件付け標識項目 (9 - 11 桁目) は使用できません。

選択グループの詳細については、 664 ページの『SELECT (選択グループの始め)』 および 704 ページの『WHENxx (真の場合に選択)』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* WHENxx および OTHER が指定された SELECT グループの例。X が
C* 1 と等しい場合には、順序 1 の演算命令が実行されます。X が
C* 1 と等しくなく、Y が 2 と等しい場合には、順序 2 の演算命令
C* が実行されます。どちらの条件も真でない場合には、順序 3 の演算命令
C* が実行されます。
C*
C                   SELECT
C     X             WHENEQ    1
C*
C*  順序 1
C*
C                      :
C                      :
C     Y             WHENEQ    2
C*
C*  順序 2
C*
C                      :
C                      :
C                   OTHER
C*
C*  順序 3
C*
C                      :
C                      :
C                   ENDSL
```

図 280. OTHER 演算命令

OUT (データ域の書き出し)

フリー・フォーム構文	OUT{(E)} {*LOCK} *data-area-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
OUT (E)	*LOCK	データ域名		_	ER	_

OUT 演算命令は、*data-area-name* オペランドに指定されたデータ域を更新します。データ域を OUT 演算命令の *data-area-name* オペランドとして指定するには、以下の 2 つのことを確認しなければなりません。

- データ域は、*DTAARA DEFINE ステートメントの結果フィールドにも指定されているか、あるいは定義仕様の DTAARA キーワードを使用しても定義されていなければなりません。
- データ域は *LOCK IN ステートメントによって前にロックされていなければならないか、あるいは定義仕様の 23 桁目の U でデータ域データ構造として指定されていなければなりません。(RPG は、プログラムの初期化時に暗黙にデータ域データ構造を検索して、ロックします。)

オプションの予約語 *LOCK を指定できます。*LOCK を指定すると、データ域は、更新後もロックされたままになります。*LOCK を指定しないと、データ域は、更新後にロック解除されます。

データ域がロックされた場合は、他のプログラムがそれを読み取ることはできますが、更新することはできません。

data-area-name オペランドは、データ域の名前または予約語 *DTAARA のいずれかでなければなりません。*DTAARA を指定すると、プログラム中で定義されているすべてのデータ域が更新されます。1 つ以上のデータ域の更新時にエラーが起こった場合には (たとえばプログラムによってロックされていないデータ域に対して OUT 演算命令を指定した場合には)、OUT 演算命令でエラーが起こり、RPG 例外/エラー処理ルーチンが制御を受け取ります。メッセージが出された場合には、そのメッセージがエラー状態のデータ域を示します。

演算結果標識が 73 および 74 桁目に指定されている場合には、OUT 演算命令時にエラーが起こるとオンにセットされます。

OUT 例外 (プログラム状況コード 401 - 421、431、または 432) を処理するために、命令コード拡張 'E'またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

71 ～ 72 および 75 ～ 76 桁目はブランクでなければなりません。

一般規則の説明については、363 ページの『データ域命令』を参照してください。
OUT 演算命令の例については、573 ページの図 236 を参照してください。

PARM (パラメーターの識別)

フリー・フォーム構文	(使用できません。PR 定義仕様を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
PARM	ターゲット・フィールド	ソース・フィールド	パラメーター			

PARM 演算命令は、パラメーター・リスト (PLIST) を構成するパラメーターを定義します。PARM 演算命令は、それを参照する PLIST、CALL、CALLB、または START 演算命令の直後に続いていなければなりません。 PARM ステートメントは、呼び出し先プログラムまたは機能で必要とされる順序になっていなければなりません。指定できるパラメーターの最大数は、次のとおりです。

- CALL 演算命令の場合は、255 個までのパラメーターを指定することができます
- CALLB、START (コンポーネント開始)、PLIST 演算命令の場合は、399 個までのパラメーターを指定することができます。

637 ページの図 281 は、PARM 演算命令の説明です。

注: ローカル・プログラムを呼び出すために CALLP を使用している場合には、パラメーターは、定義仕様でプロトタイプを指定することによって定義されます。 261 ページの『24-25 桁目 (定義のタイプ)』 および 284 ページの『OPTIONS(*OMIT *VARSIZE *STRING *RIGHTADJ)』 には、CALLP 演算命令にパラメーターを指定する方法が説明されています。

演算項目 1 を指定する場合には、結果フィールドと同じタイプにしなければなりません。ターゲット・フィールドが可変長である場合には、その長さは、ソース・フィールドの値の長さに設定されます。リテラルまたは名前付き固定情報とすることはできません。結果フィールドに、複数オカレンス・データ構造の名前が入っている場合には、これをブランクにすることができます。

演算項目 2 を指定する場合には、結果フィールドと同じタイプにしなければなりません。結果フィールドに、複数オカレンス・データ構造の名前が入っている場合には、これをブランクにすることができます。

パラメーター・タイプ検査がアプリケーションにとって重要な場合には、PLIST および PARM 演算命令を使用するよりむしろ、呼び出しインターフェースのプロトタイプおよびプロシージャー・インターフェース定義を定義する必要があります。

結果フィールドには、フィールド、データ構造、または配列の名前が入っていなければなりません。

- 配列を指定すると、その配列用に定義された領域が呼び出し先プログラムまたはプロシージャーに渡されます。
- 複数オカレンスがあるデータ構造を呼び出し先プログラムに渡すと、そのデータ構造のすべてのオカレンスが単一フィールドとして渡されます。しかし、結果フィールドに複数回繰り返しデータ構造のサブフィールドが指定されている場合には、そのサブフィールドの現行オカレンスのみが呼び出し先プログラムまたはプロシージャーに渡されます。

ホスト・プログラムが呼び出される場合でない限り、結果フィールドに UCS-2 パラメーターを入れることはできません。

*ENTRY PLIST PARM 演算命令以外の場合は、結果フィールドに、配列エレメントまたは *OMIT (CALLB の場合のみ) の名前を入れることができます。*OMIT を指定する場合には、演算項目 1 と演算項目 2 はブランクでなければなりません。

*ENTRY PLIST PARM 演算命令の場合は、結果フィールドに次を入れることはできません。

- *IN、*INxx、*IN(xx)、*OMIT
- ラベル、リテラル、または名前付き固定情報
- データ域名またはデータ域データ構造名名
- グローバルに初期化されたデータ構造、サブフィールドが初期化されているデータ構造、またはサブフィールドとしてコンパイル時配列をもつデータ構造
- テーブル名
- フィールドまたはキーワード BASED を指定して定義されたデータ構造
- 配列エレメント
- データ構造サブフィールド名
- コンパイル時配列の名前
- プログラム状況またはファイル情報データ構造 (INFDS) の名前
- UCS-2 パラメーターは使用できません。

注: フィールド名は *ENTRY PLIST で一度しか指定することができません。

条件付け標識項目 (9 - 11 桁目) は使用できません。

パラメーターに関する一般規則

各パラメーター・フィールドの記憶場所は、呼び出し元プログラムまたはプロシージャー内です。 PARM 演算命令で結果フィールドの記憶場所のアドレスが呼び出し先プログラムに渡されます。呼び出し先プログラムまたはプロシージャーがパラメーターの値を変更すると、記憶場所のデータが変更されます。制御が呼び出し元プログラムまたはプロシージャーに戻されると、呼び出し元プログラムまたはプロシージャーのパラメーター (すなわち、結果フィールド) が変更されています。呼び出し先プログラムまたはプロシージャーが、パラメーターの値を変更した後でエラー終了したとしても、変更された値は呼び出し元プログラムまたはプロシージャーに存在しています。呼び出し先プログラムまたはプロシージャーに渡された情報を後から使用するために保存するには、呼び出し先プログラムまたはプロシージャーに渡したい情報が入っているフィールドの名前を演算項目 2 に指定してください。演算項目 2 が結果フィールドにコピーされて、結果フィールドのストレージのアドレスが呼び出し先プログラムまたはプロシージャーに渡されます。

パラメーター・フィールドはフィールド名ではなくアドレスによって渡されるので、呼び出し元パラメーターおよび呼び出し先パラメーターは、渡されるフィールドに同一のフィールド名を使用する必要はありません。呼び出し元および呼び出し先のプログラムまたはプロシージャー内で対応しているパラメーター・フィールドは同じものになります。そうでない場合には、望ましくない結果が起こる場合があります。

CALL、CALLB、および START でのパラメーターの渡し

CALL、CALLB、または START (コンポーネント開始) 演算命令の実行時には、次が行なわれます。

1. 呼び出し元プログラムで、PARM 演算命令の内容は、同一 PARM 演算命令の結果フィールドにコピーされます。CALLB の結果フィールドが *OMIT の場合には、ヌル・アドレスが呼び出し先プロシージャーに渡されます。
2. 呼び出し先プログラムが制御を受け取った後および任意の通常プログラム初期化後に、PARM 演算命令の結果フィールドの内容が同一 PARM 演算命令の演算項目 1 フィールドにコピーされます。
3. 制御が呼び出し元プログラムに戻されるときに、PARM 演算命令の演算項目 2 の内容が同一 PARM演算命令の結果フィールドにコピーされます。呼び出し先プログラムが異常終了すると、この移動は実行されません。
4. START 演算命令の場合、ターゲット・コンポーネントが初期化されるとすぐに (*INZSR 処理の完了後に)、制御は呼び出し元プログラムに戻されます。ターゲット・コンポーネントの存続時間の残りの間、パラメーターはアクセス可能で、ソース・コンポーネントとターゲット・コンポーネントの両方によって変更されている可能性があります。
5. 呼び出し元プログラムに戻るときに、PARM 演算命令の結果フィールドの内容が同一 PARM 演算命令の演算項目 1 フィールドにコピーされます。呼び出し先プログラムが異常終了するか、あるいは呼び出し演算命令でエラーが起こると、この移動は実行されません。

PLIST (パラメーター・リストの識別)

フリー・フォーム構文	(使用できません。PR 定義仕様を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
PLIST	PLIST 名					

PLIST 演算命令は、CALL、CALLB、CALLP、または START 演算命令で指定されるパラメーター・リストの固有の記号名を定義します。PLIST 演算命令のすぐ後には、少なくとも 1 つの PARM 演算命令が続いていなければなりません。

演算項目 1 には、パラメーター・リストの名前が入っていなければなりません。パラメーター・リストが入り口点パラメーター・リストの場合には、演算項目 1 には *ENTRY が入っていなければなりません。プログラムまたは呼び出し先機能で指定できる *ENTRY パラメーター・リストは 1 つだけです。パラメーター・リストは、PARM 以外の演算命令が検出されると終了します。

パラメーター・タイプ検査がアプリケーションにとって重要な場合には、PLIST および PARM 演算命令を使用するよりむしろ、呼び出しインターフェースのプロトタイプおよびプロシージャー・インターフェース定義を定義する必要があります。

条件付け標識項目 (9 - 11 桁目) は使用できません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  呼び出し元プログラムでは、CALL 演算命令は PROG1 を呼び出して、
C*  PROG1 はパラメーター・リスト・フィールド中のデータにアクセ
C*  スすることができます。
C                   CALL      'PROG1'         PLIST1
C*
C*  2 番目の PARM ステートメントで、CALL が処理されると、演
C*  算項目 2 (*IN27) の内容が結果フィールド BYTE に入れら
C*  れます。PROG1 が制御を戻すときに、結果フィールド BYTE
C*  の内容が演算項目 1 (*IN30) に入れられます。PARM の演算
C*  項目 1 と演算項目 2 の項目は任意指定です。
C*
C     PLIST1        PLIST
C                   PARM                    Amount            5 2
C     *IN30         PARM      *IN27         Byte              1
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   CALLB     'PROG2'
.
.
.
C*  この例では、PARM 演算命令の直後には PLIST 演算命令の代りに
C*  CALLB 演算命令が続いています。
C                   PARM                    Amount            5 2
C     *IN30         PARM      *IN27         Byte              1
.
.
.
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*  呼び出し先機能 PROG2 で、PLIST ステートメントの演算項目 1
C*  の *ENTRY はそれを項目パラメーター・リストとして識別します。
C*  制御が PROG2 に転送されると、パラメーター・リストの結果フィー
C*  ルド (FieldC および FieldG) の内容が演算項目 1 フィールド
C*  (FieldA および FieldD) に入れられます。呼び出し先機能から
C*  戻るときに、パラメーター・リストの演算項目 2 フィールド
C*  (FieldB および FieldE) の内容が結果フィールド (FieldC
C*  および FieldG) に入れられます。フィールドのすべては呼び
C*  出し先機能で他の方法で定義されています。
C     *ENTRY        PLIST
C     FieldA        PARM      FieldB        FieldC
C     FieldD        PARM      FieldE        FieldG
```

図 281. PLIST/PARM 演算命令

POST (転記)

フリー・フォーム構文	POST{(E)} *file-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
POST (E)		file-name	INFDS 名	_	ER	_

POST 演算命令は、情報をファイル情報データ構造 (INFDS) に書き込みます。

リモート・ファイルの場合は、この構造体には次の情報が入ります。
- ファイル・フィードバック情報
- オープン・フィードバック情報
- 入出力フィードバック情報および装置依存フィードバック情報

ローカル・ファイルの場合には、この構造体にはファイル・フィードバック情報が入ります。

ファイルの名前を *file-name* オペランドに指定します。このファイルの情報は、このファイルに関連した INFDS にポストされます。

フリー・フォーム構文では、*file-name* を指定しなければなりませんが、INFDS 名を指定することはできません。従来の構文では *file-name*、INFDS 名、あるいはその両方を指定できます。
- INFDS 名を指定しない場合には、ファイル仕様の INFDS キーワードを使用してこのファイルに関連付けられた INFDS が使用されます。
- 従来の構文で INFDS 名を指定しない場合には、ファイル仕様の INFDS キーワードに使用されているデータ構造名を結果フィールドに指定しなければなりません。これにより、ファイル仕様中の関連ファイルからの情報がポストされます。

file-name オペランドを指定する場合には、ローカル・ファイルまたは OS/400 ファイルのいずれかとすることができます。このファイルは、POST 演算命令の前にオープンされていなければなりません。このファイルのための情報は、関連した INFDS にポストされます。

ファイルが複数メンバー処理用にオープンされている場合には、
READ、READP、READE、または READPE などの入力演算命令によって新規メンバーがオープンされるときに、オープン・フィードバック情報が更新されます。

入力レコードがブロック化されていて、POST 演算命令がアプリケーションにない場合には、現行キーおよび相対レコード番号が入出力フィードバック情報にコピーされます。入力レコードがブロック化されていて、POST 演算命令がアプリケーションにある場合には、入出力フィードバック情報がブロック内の現行レコードのキーおよび相対レコード番号で更新されます。

演算結果標識が 73 および 74 桁目に指定されている場合には、POST 演算命令時にエラーが起こるとオンにセットされます。

POST 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

READ (レコードの読み取り)

フリー・フォーム構文	READ{(EN)} *name* {*data-structure*}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READ (EN)		ファイル名	データ構造	_	ER	EOF
READ (EN)		レコード名		_	ER	EOF
READ (E)		ウィンドウ名		_	ER	_

READ 演算命令は、データをファイル、レコード様式、またはウィンドウから読み取ります。ファイルは、リモート OS/400 ファイルまたはローカル・ファイルにすることができます。

ファイルからの読み取り

name オペランドは必須で、全手順ファイルまたはレコード様式の名前でなければなりません。

レコード様式名を使用できるのは、外部記述ファイル (ファイル仕様の 22 桁目が E) の場合だけです。様式名による READ 演算命令で *name* オペランドによって指定したものとは異なる様式を受け取るのがその場合です。その場合には、READ 演算命令はエラー終了します。

指定されるファイルがプログラム記述されている場合には、*data-structure* オペランドを使用してデータ構造の名前を指定できます。レコードは、ファイルの入力仕様を処理しないで、このデータ構造に読み取られます。プログラム記述ファイルは、ファイル仕様の 22 桁目の F によって識別されます。ファイルとデータ構造の間でデータを転送する方法については、367 ページの『ファイル命令』を参照してください。

READ 演算命令が成功すると、ファイルは、読み取りを満たしている次のレコードに位置付けられます。エラーまたはファイルの終わり条件がある場合には、そのファイルは (CHAIN、SETLL、または SETGT 演算命令を使用して) 再位置決めしなければなりません。

ファイルが更新ディスク・ファイルである場合には、レコードの読み取り時にそのレコードをロックしないことを指示するために、命令拡張 N を指定することができます。

注: ローカル・ファイルのロックはサポートされません。

READ 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

READ 演算命令でファイルの終わりが起こったかどうかのシグナルを送るために、75 - 76 桁目に標識を指定することができます。標識は READ 演算命令が実行されるたびにオン (EOF 条件) またはオフのいずれかにセットされます。この情報は、EOF 条件が起こると '1' を、そうでない場合には '0' を戻す %EOF 組み込み関数から取得することもできます。ファイルに対する正常な順次操作 (たとえば、READ または READP) をさらに処理するためには、ファイルは EOF 条件後に再位置決めされていなければなりません。

ヌル可能フィールドがあるレコードの読み取りについては、145 ページの『データベースのヌル値サポート』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  READ は、次レコードを全手順ファイルでなければならないファ
C*  イル FILEA から検索します。
C*
C*  READ でファイルの終わりが起こる場合あるいはファイルの終わりが
C*  以前に起こっていて、ファイルが再位置決めされていない場合に、
C*  %EOF は '1' を戻すように設定されます。%EOF が '1' を戻すと、
C*  プログラムはループを終了します。
C*
C                   DOW       '1'
C                   READ      FILEA
C                   IF        %EOF
C                   LEAVE
C                   ENDIF
C*
C*  READ は、タイプ REC1 (演算項目 2) の次レコードを外部記述
C*  ファイルから検索します。(REC1 はレコード様式名です。)
C*  標識 64 は READ でファイルの終わりが起こる場合、あるいはファ
C*  イルの終わりが以前に起こっていて、ファイルが再位置決めされ
C*  ている場合にオンにセットされます。標識 64 がオンにセット
C*  されていると、プログラムはループを終了します。N 命令
C*  コード拡張は、レコードがロックされていないことを示します。
C*
C                   READ(N)   REC1                                64
C   64              LEAVE
C                   ENDDO
```

図 282. ファイルの場合の READ 演算命令

ウィンドウからの読み取り

ウィンドウは、外部記述ファイルとして処理されます。ウィンドウ名はレコード様式名として取り扱われます。

name オペランドでウィンドウ名を指定すると、READ 演算命令は、ウィンドウ上の組み合わせボックス、チェック・ボックス、入力フィールド、ラジオ・ボタン、および静的テキスト・パーツの属性を取り出します。項目パーツの属性は TEXT です。静的テキスト・パーツの属性は LABELです。

ウィンドウの読み取り時に、属性取り出し演算命令がすべての静的テキストおよび入力フィールド・パーツに対して実行されます。値は対応しているフィールドに保管されます。READ 演算命令後に、フィールド中に保管された値は、画面に表示される値と一致しています。多くの静的テキストおよび入力フィールドがある場合には、複数の GETATR よりもむしろ READ 演算命令を使用してください。たとえば、ウィンドウ INVENTORY に入力フィールド・パーツ ENT0000B および ENT0000C が含まれている場合には、ウィンドウの READ で次と同等の処理が実行されます。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....Comments++++++
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++Comments++++++
C                   EVAL      ENT0000B = %GETATR('INVENTORY':'ENT0000B':'TEXT')
C                   EVAL      ENT0000C = %GETATR('INVENTORY':'ENT0000C':'TEXT')
```

図 283. ウィンドウの場合の READ 演算命令

READC (次の変更レコードの読み取り)

フリー・フォーム構文	READC{(E)} *subfile-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READC (E)		サブファイル名		_	ER	EOF

READC 演算命令は、サブファイル・パーツで次に変更されたレコードを取得します。

subfile-name オペランドはサブファイル・パーツの名前でなければなりません。

結果フィールドを指定する場合には、小数点以下の桁数なしの数字フィールド名でなければなりません。検索されたレコードの相対レコード番号は結果フィールドに入れられます。

READC 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

サブファイル中にそれ以上の変更済みレコードがない時にオンにセットされる標識は、75 ～ 76 桁目に指定することができます。この情報は、サブファイル中にそれ以上の変更済みレコードがないと '1' を戻し、そうでない場合には '0' を戻す %EOF 組み込み関数からも取得することができます。

ファイルの終わり標識 (EOF) が指定されていると、サブファイル中にそれ以上の変更済みレコードがない場合にオンにセットされます。演算命令が成功しなかった場合には、プログラム中のフィールドは未変更のままです。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FFilename++IT.A.FRlen+......A.Device+.Keywords++++++++++++++++++++++++++++++
F* SUBCUST は、CUSINFO ファイルからレコードのリストを表示す
F* るサブファイル・パーツです。
F*
FCUSINFO   UF   E             DISK    REMOTE
F
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C* サブファイル SUBCUST は CUSINFO ファイルからレコードを使用し
C* てロードされています。このサブファイルに表示されたレコードの
C* いずれかでなんらかの変更がある場合には、READC 演算命令は、do
C* while ループで 1 件ずつ変更済みレコードを読み取ります。
C* CUSINFO ファイル中の対応するレコードは、CHAIN 演算命令で検索
C* されて、変更済みフィールドで更新されます。
C*
C* SCUSNO、SCUSNAM、SCUSADR、および SCUSTEL は、サブファイル中
C* に定義されたフィールドです。CUSNAM、CUSADR、および CUSTEL は
C* ファイル CUSINFO に定義されたレコード CUSREC 中に定義されています。
C*
C                   READC     SUBCUST
C                   DOW       %EOF = *OFF
C     SCUSNO        CHAIN (E) CUSINFO
C* ファイルがファイル中に見つかる場合だけそのレコードを更新します。
C                   :
C                   IF        NOT %ERROR
C                   EVAL      CUSNAM = SCUSNAM
C                   EVAL      CUSADR = SCUSADR
C                   EVAL      CUSTEL = SCUSTEL
C                   UPDATE    CUSREC
C                   ENDIF
C                   READC (E) SUBCUST
C                   ENDDO
```

図 284. READC の例

READE (等しいキーのレコードの読み取り)

フリー・フォーム構文	READE{(EN)} *search-arg*|*KEY *name* {*data-structure*}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READE (EN)	search-arg	name (ファイルまたはレコード様式)		_	ER	EOF

READE 演算命令は、レコードのキーが検索引き数と一致する場合には、次の前順次レコードを全手順ファイル (ファイル仕様の 18 桁目の F によって識別される) から検索します。レコードのキーが検索引き数と一致しない場合には、EOF 条件が起こり、レコードはプログラムに戻され ません。ファイルの終わりが起こると、EOF 条件も適用されます。

READE 演算命令を使用できるのは、OS/400 ファイルだけです。

検索引き数 *search-arg* は、検索されるレコードを識別します。*search-arg*オペランドは従来の構文では任意指定ですが、フリー・フォーム構文では必須です。*search-arg* は、以下のものとすることができます。

- フィールド名、リテラル、名前付き固定情報、または表意定数。
- 外部記述ファイルの KLIST 名。
- *KEY または (従来の構文のみ) 値なし。次のレコードのキー全体が現行レコードのものと等しい場合には、ファイル中の次のレコードが検索されます。キー全体は，レコード様式または *name* に指定したファイルで定義します。

グラフィックおよび UCS-2 キーは、同じ CCSID を必要とします。

読み取り中のファイルが更新として定義されている場合には、次レコードに対する一時ロックが要求されて、検索引き数がそのレコードのキーと比較されます。レコードがすでにロックされている場合には、プログラムは、一時ロックを取得して比較を行なう前に、レコードが使用可能になるまで待たなければなりません。比較が等しくない場合には、EOF 条件が起こり、一時レコード・ロックは除去されます。非ロック ('N' 命令拡張) が指定されている場合には、一時ロックは要求されません。

name オペランドは、検索されるファイルまたはレコード様式の名前でなければなりません。レコード様式名を使用できるのは、外部記述ファイル (ファイル仕様の 22 桁目の E によって識別) の場合だけです。

data-structure オペランドは、*name* に指定されたファイルが、プログラム記述ファイル (ファイル仕様の 22 桁目の F によって識別) の場合だけレコードが読み取られるデータ構造の名前とすることができます。ファイルとデータ構造の間でデータを転送する方法の説明については、 367 ページの『ファイル命令』を参照してください。

READE 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方と

も指定することはできません。エラー処理の詳細については、 41 ページの『ファイルの例外 / エラー』を参照してください。

EOF 条件が起こった場合 (すなわち、検索引き数と等しいキーのレコードが見つからない場合あるいはファイルの終わりが検出された場合) にオンにセットされる標識は、75 - 76 桁目に指定することができます。この情報は、EOF 条件が起こると '1' を、そうでない場合には '0' を戻す %EOF 組み込み関数から取得することもできます。

READE 演算命令が成功しないと、プログラム内のフィールドは未変更のままで、ファイルは (たとえば、CHAIN、SETLL、または SETGT を使用して) 再位置付けされていなければなりません。*START および *END は、ファイルを位置決めするために使用することができます。ファイルの位置決めの詳細については、 6 ページの『ファイルの位置決め』を参照してください。

OPEN 演算命令または EOF 条件のすぐ後に続き、*search-arg* が指定された READE は、レコードのキーが検索引き数と一致する場合にファイル中の最初のレコードを検索します。OPEN 演算命令または EOF 条件のすぐ後に続いているが、*search-arg* が指定されていない READE はエラーになります。 73 - 74 桁目のエラー標識は、指定されている場合には、オンにセットされます。そうでない場合には、%ERROR で検査される 'E' 拡張は、指定されている場合には、オンにセットされます。ファイルがクローズされて再オープンされるまで、それ以上の I/O 演算命令を発行することはできません。

ヌル可能フィールドがあるレコードの読み取りについては、 145 ページの『データベースのヌル値サポート』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  演算項目 1 が指定されている場合...
C*
C*  READE 演算命令は、次レコードをファイル FILEA から検索して、
C*  そのキーを検索引き数 KEYFLD と比較します。
C*  KEYFLD が読み取られたレコードのキーと等しくない場合、あ
C*  るいはファイルの終わりが検出されると %EOF 組み込み関数は '1'
C*  を戻すように設定されます。
C*
C     KEYFLD        READE     FILEA
C*
C*  READE 演算命令は、タイプ REC1 の次レコードを外部記述ファイル
C*  から検索して、読み取られたレコードのキーと検索引き数 KEYFLD
C*  を比較します。(REC1 はレコード様式名です。)
C*  KEYFLD が読み取られたレコードのキーと等しくない場合、あるいは
C*  ファイルの終わりが検出されると、標識 56 がオンにセットされます。
C*
C     KEYFLD        READE     REC1                                   56
C*
C*  演算項目 1 が指定されていない場合...
C*
C*  READE 演算命令は、キー値が現在位置のレコードのキー値と等しい
C*  と等しい場合に、アクセス・パス中の次レコードをファイル FILEA
C*  から検索します。
C*  キー値が等しくないと、%EOF が '1' を戻すように設定されます。
C                   READE     FILEA
C*
C*  READE 演算命令は、キー値が現在位置のレコードのキー値と等しい
C*  場合に、アクセス・パス中の次レコードをファイル FILEA から検索
C*  します。REC1 はレコード様式名です。
C*  キー値が等しくない場合には、標識 56 がオンにセットされます。
C*  N は、レコードがロックされていないことを示します。
C                   READE(N)  REC1                                   56
```

図 285. READE 演算命令

READP (前のレコードの読み取り)

フリー・フォーム構文	READP{(EN)} *name* {*data-structure*}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READP (EN)		ファイル名	データ構造	_	ER	BOF
READP (EN)		レコード名		_	ER	BOF

READP 演算命令は、前のレコードを全手順ファイル (ファイル仕様の 18 桁目の F によって識別される)から読み取ります。

name オペランドは、読み取られるファイルまたはレコード様式の名前でなければなりません。レコード様式名を使用できるのは、外部記述ファイルの場合だけです。レコード様式名を指定した場合には、検索されるレコードは指定されたタイプの最初の前レコードです。介在しているレコードはバイパスされます。

data-structure オペランドは、*name* に指定されたファイルが、プログラム記述ファイル (ファイル仕様の 22 桁目の F によって識別) の場合だけ、レコードが読み取られるデータ構造の名前とすることができます。ファイルとデータ構造の間でデータを転送する方法については、367 ページの『ファイル命令』を参照してください。

READP 演算命令が成功すると、ファイルは、読み取りを満たしている直前のレコードに位置付けられます。

読み取り中のファイルが更新ディスク・ファイルである場合には、レコードの読み取り時にそのレコードをロックしないことを指示するために、命令拡張 N を指定することができます。

READP 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

ファイル中に前レコードが存在していない場合にオンにセットされる標識は、75 - 76 桁目に指定することができます。この情報は、BOF 条件が起こると '1' を、そうでない場合には '0' を戻す %EOF 組み込み関数から取得することもできます。

ファイルに対する正常な順次操作 (たとえば、READ または READP) をさらに処理するためには、ファイルはエラーまたは BOF 条件後に (たとえば、CHAIN、SETLL、または SETGT 演算命令を使用して) 再位置決めされていなければなりません。*START および *END は、ファイルを位置決めするために使用することができます。ファイル位置決めについて詳しくは、6 ページの『ファイルの位置決め』を参照してください。

ヌル可能フィールドがあるレコードの読み取りについては、 145 ページの『データベースのヌル値サポート』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  READP 演算命令は前レコードを FILEA から読み取ります。
C*  ファイルの始めが検出されると、%EOF 組み込み関数は '1' を
C*  戻すように設定されます。%EOF が '1' を戻すし、プログラムは
C*  GOTO 演算命令に指定されたラベル BOF に分岐します。
C*
C                   READP     FILEA
C                   IF        %EOF
C                   GOTO      BOF
C                   ENDIF
C*
C*  READP 演算命令はタイプ REC1 の次の前レコードを外部記述ファイ
C*  ルから読み取ります。(REC1 はレコード様式名です。)
C*  READP 演算命令の処理中にファイルの始めが検出されると、標識
C*  72 がオンにセットされます。標識 72 がオンにセットされている
C*  と、プログラムは GOTO 演算命令に指定されたラベル BOF に分岐
C*  します。
C                   READP     PREC1                                  72
C   72              GOTO      BOF
C*
C     BOF           TAG
```

図 286. READP 演算命令

READPE (前の等しいキーのレコードの読み取り)

フリー・フォーム構文	READPE{(EN)} *search-arg*|*KEY *name* {*data-structure*}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READPE (EN)	search-arg	ファイル名	データ構造	_	ER	BOF
READPE (EN)	search-arg	レコード名		_	ER	BOF

READPE 演算命令は、レコードのキーが検索引き数と一致する場合には、次の前順次レコードを全手順ファイル (ファイル仕様の 18 桁目の F によって識別される) から読み取ります。レコードのキーが検索引き数と一致しない場合には、BOF 条件が起こり、レコードはプログラムに戻され ません。ファイルの始めが起こると、BOF 条件も適用されます。

READPE 演算命令を使用できるのは、OS/400 ファイルだけです。

検索引き数 *search-arg* は、検索されるレコードを識別します。*search-arg*オペランドは従来の構文では任意指定ですが、フリー・フォーム構文では必須です。*search-arg* は、以下のものとすることができます。

- フィールド名、リテラル、名前付き固定情報、または表意定数。
- 外部記述ファイルの KLIST 名。
- *KEY または (従来の構文のみ) 値なし。次の前レコードのキー全体が現行レコードのものと等しい場合には、ファイル中の次の前レコードが検索されます。キー全体は、演算項目 2 で使用されるレコード様式またはファイルで定義します。

グラフィックおよび UCS-2 キーは、同じ CCSID を必要とします。

name オペランドは、検索されるファイルまたはレコード様式の名前でなければなりません。レコード様式名を使用できるのは、外部記述ファイル (ファイル仕様の 22 桁目の E によって識別) の場合だけです。

data-structure オペランドは、*name* に指定されたファイルが、プログラム記述ファイル (ファイル仕様の 22 桁目の F によって識別される) の場合だけ、レコードが読み取られるデータ構造の名前とすることができます。ファイルとデータ構造の間でデータを転送する方法の説明については、367 ページの『ファイル命令』を参照してください。

読み取り中のファイルが更新ディスク・ファイルである場合には、レコードの読み取り時にそのレコードをロックしないことを指示するために、命令拡張 N を指定することができます。

READPE 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方と

も指定することはできません。エラー処理の詳細については、 41 ページの『ファイルの例外 / エラー』を参照してください。

BOF (ファイルの始め) 条件が起こった場合 (すなわち、検索引き数と等しいキーのレコードが見つからない場合あるいはファイルの始めが検出された場合) にオンにセットされる標識は、75 - 76桁目に指定することができます。この情報は、BOF 条件が起こると '1' を、そうでない場合には '0' を戻す %EOF 組み込み関数から取得することもできます。

READPE 演算命令が成功しない場合には、そのファイルは、たとえば、CHAIN、SETGT、または SETLL 演算命令を使用して、再位置決めしなければなりません。

注: 読み取り中のファイルが更新として定義されている場合には、前レコードに対する一時ロックが要求されて、検索引き数がそのレコードのキーと比較されます。レコードがすでにロックされている場合には、プログラムは、一時ロックを取得して比較を行なう前に、レコードが使用可能になるまで待たなければなりません。比較が等しくない場合には、BOF 条件が起こり、一時レコード・ロックは除去されます。非ロック ('N' 命令拡張) が指定されている場合には、一時ロックは要求されません。

OPEN 演算命令または BOF 条件のすぐ後に続き、*search-arg* オペランドが指定されている READPE は BOF を戻します。OPEN 演算命令または BOF 条件のすぐ後に続くが、 *search-arg* が指定されて**いない** READPE はエラー条件になります。73 - 74 桁目のエラー標識は、指定されている場合には、オンにセットされます。そうでない場合には、%ERROR で検査される 'E' 拡張は、指定されている場合には、オンにセットされます。演算項目 1 がブランクになっている READPE を発行する前に、ファイルは、*search-arg* が指定されている CHAIN、SETLL、READ、READE、または READP を使用して再位置決めされていなければなりません 。SETGT 命令コードは、(*search-arg* が指定されていない) READPE を発行する前にファイルを位置決めするために使用してはいけません。これは、(SETGT が発行された後には、現行レコードの直前のレコードが現行レコードと同じキーになることは決してないので)「該当レコードなし」条件になるためです。*search-arg* が両方の命令コードに同一キーで指定されている場合には、このエラー条件は起こりません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  演算項目 1 が指定されている場合...
C*
C* 直前のレコードが読み取られて、キーが FieldA と比較されました。
C* レコードのキーが FieldA と一致しないと、標識 99 はオンにセッ
C* トされます。
C     FieldA        READPE    FileA                                  99
C*
C* 直前のレコードがレコード様式 RecA から読み取られて、キーが
C* FieldC と比較されました。演算命令が正常に完了しないと、標識
C* 88 がオンにセットされて、レコード・キーが FieldC と一致しな
C* いと、99 がオンにセットされます。
C     FieldC        READPE    RecA                                 8899
C*
C*  演算項目 1 が指定されていない場合...
C*
C* キー値が現行レコードのキー値と等しい場合には、アクセス・パス
C* 中の直前のレコードが検索されます。
C* キー値が等しくない場合には、標識 99 がオンにセットされます。
C                   READPE    FileA                                  99
C*
C* キー値がアクセス・パス中の現行レコードのキー値と一致する場合に
C* は、直前のレコードがレコード様式 RecA から検索されます。
C* 演算命令が成功しない場合には、標識 88 がオンにセットされて、
C* キー値が等しくない場合には、99 がオンにセットされます。
C                   READPE    RecA                                 8899
```

図 287. READPE 演算命令

READS (選択されたレコードの読み取り)

フリー・フォーム構文	READS{(E)} *subfile-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READS (E)		サブファイル名		_	ER	EOF

READS 演算命令は、サブファイル・パーツから選択されたレコードを検索します。サブファイル・パーツから選択される最初のレコードが読み取られます。

サブファイルの選択スタイルが拡張または複数である場合には、そのレコードは選択解除されます。サブファイルの選択スタイルが単一である場合には、そのレコードは選択状態のままになります。次の READSは同一レコードを再び読み取ります。

subfile-name は、サブファイル・パーツの名前を指定します。

結果フィールドを指定する場合には、小数点以下の桁数なしの数字フィールドでなければなりません。検索されたレコードのサブファイル索引番号が結果フィールドに入れられます。

READS 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

EOF 条件が起こった場合 (すなわち、選択されたレコードがサブファイル中にない場合) にオンにセットされる標識は、75 - 76 桁目に指定することができます。この情報は、EOF 条件が起こると '1' を、そうでない場合には '0' を戻す %EOF 組み込み関数から取得することもできます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* SUBCUST は、レコードのリストを表示するサブファイル・パーツです。
C* READS 演算命令は、表示されたサブファイル中で選択されたレコード
C* を do while ループ内で 1 件ずつ読み取ります。SCUSNO および
C* SCUSNAM は、サブファイル中に定義されたフィールドです。
C*
C                   READS     SUBCUST                                27
C                   DOW       *IN27 = *OFF
C*
C* ここで、読み取られている選択済みのレコードを処理するために、
C* フィールド SCUSNO、SCUSNAM を使用することができます。
C*
C                   READS     SUBCUST                                27
C                   ENDDO
C*
```

図 288. READS 演算命令

REALLOC (新規長さのストレージの再割り振り)

フリー・フォーム構文	(使用できません。%REALLOC 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
REALLOC (E)		長さ	ポインター	_	ER	_

REALLOC 演算命令は、結果フィールド・ポインターによって指されたヒープ・ストレージの長さを演算項目 2に指定された長さに変更します。REALLOC の結果フィールドには基底ポインター変数が入ります。結果フィールド・ポインターには、ヒープ・ストレージ割り振り演算命令 (RPG では、ALLOC または REALLOC 演算命令、あるいはその他のヒープ・ストレージ関数のどれか) によって以前に設定された値が入っています。単にヒープ・ストレージを指しているだけでは十分ではありません。ポインターは割り振りの先頭に設定されていなければなりません。

指定されたサイズの新規ストレージが割り振られて、旧ストレージの内容がその新規ストレージにコピーされます。その後で、旧ストレージが割り振り解除されます。新規長さの方が短い場合には、値が右方で切り捨てられます。新規長さの方が長い場合には、コピーされたデータの右方の新規ストレージは未初期化状態になっています。

結果フィールド・ポインターは、新規ストレージを指すように設定されます。

この演算命令が成功しないと、エラー条件が起こりますが、結果フィールド・ポインターは変更されません。元のポインターが有効で、使用可能な新規ストレージが不足していた (状況 425) ために演算命令が失敗した場合には、元のストレージは割り振り解除されないので、結果フィールド・ポインターはその元の値のまま依然として有効です。

ポインターは有効だが、割り振り解除できるストレージを指していない場合には、状況 00426 (記憶管理操作でのエラー) が設定されます。

プログラム状況コード 425 または 426 での例外を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

演算項目 2 には、割り振られるストレージの新規サイズ (バイト数) を指示する数値変数または固定情報が入ります。演算項目 2 は、小数点以下の桁数がゼロの数字でなければなりません。この値は、1 〜 16776704 の範囲内でなければなりません。

```
D Ptr1            S               *
D Fld             S          32767A     BASED(Ptr1)
D*
C* ALLOC 演算命令は 7 バイトをポインター Ptr1 に割り振ります。
C* ALLOC 演算命令後には、変数 Fld の先頭から 7 バイトだけを使用
** することができます。
C                   ALLOC     7                Ptr1
C                   EVAL      %SUBST(Fld : 1 : 7) = '1234567'
C*
C                   REALLOC   10               Ptr1
C* これで Fld の 10 バイトを使用することができます。
C                   EVAL      %SUBST(Fld : 1 : 10) = '123456789A'
```

図 289. REALLOC 演算命令

詳細については、 370 ページの『メモリー管理命令』を参照してください。

RESET (リセット)

フリー・フォーム構文	RESET{(E)} {*NOKEY} {*ALL} *name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
RESET (E)		*ALL	変数	_	ER	_
RESET (E)	*NOKEY	*ALL	構造体	_	ER	_
RESET (E)			ウィンドウまたは サブファイル	_	ER	_

RESET 演算命令は、次をその初期値に設定します。
- 構造体 (レコード様式、データ構造、配列、テーブル) のエレメント
- 変数 (フィールド、サブフィールド、標識)
- ウィンドウの静的テキストおよび入力フィールド

RESET 例外 (プログラム状況コード 123) を処理するために、命令コード拡張 'E' またはエラー標識 ERのいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

RESET 演算命令は、ストレージがリセットされるすべての変数、構造体、またはウィンドウが倍増するので、プログラムが必要とするストレージの容量を増やします。複数回繰り返しデータ構造、テーブル、および配列の場合は、すべてのオカレンスまたはエレメントの初期値が保管されます。

初期化ルーチンの実行中に RESET を使用しないでください。演算命令 (GOTO など) が、初期値が保管される前に初期化サブルーチンを残しておくために使用されると、プログラム内で実行しようとするすべての RESET 演算命令でエラーが起こります。

ウィンドウの入力フィールドおよび静的テキストのリセット

結果フィールドがウィンドウ名である場合には、演算項目 1 と演算項目 2 はブランクでなければなりません。

ウィンドウがリセットされると、そのウィンドウの入力フィールド・パーツおよび静的テキスト・パーツはその初期値にリセットされます。このパーツは対応するプログラム・フィールドの初期値にリセットされ、GUI Designer の使用中に指定された初期値にリセットされるわけではありません。対応するフィールドの初期値は、プログラム初期化の終了時の値です。この値は、GUI Designer を使用して定義仕様に、あるいは初期化サブルーチンを使用して設定されます。初期化サブルーチン (*INZSR) で指定された値によって、定義仕様に指定された値が指定変更されて、定義仕様の値によって、GUI Designer に指定された値が指定変更されます。

たとえば、次のテーブルは、GUI Designer で指定される各種入力フィールド・パーツの値、ならびに定義仕様および初期化サブルーチン (*INZSR) のフィールドの値が RESET 演算命令からどんな影響を受けるかを示しています。

GUI Designer	定義仕様	*INZSR	RESET 後の値
ENT0000B=22.5	ENT0000B=30.5		ENT0000B=30.5
ENT0000A=abc	ENT0000A=xyz	ENT0000A=pqr	ENT0000A=pqr
		ENT0000C= 名前	ENT0000C= 名前
			ENT0000D が文字の場合には、RESET はブランクにリセットします。 ENT0000D が数字の場合には、RESET はゼロにリセットします。

注: RESET 演算命令後に、プログラム・フィールド中に保管される値は、画面に表示される値と一致しています。

構造体のエレメントおよび変数のリセット

変数または構造体の初期値は、プログラム初期化の終了時の値です。この値は、定義仕様で INZ キーワードを使用して、あるいは初期値を割り当てる初期化サブルーチンを使用して設定することができます。この初期値は RESET 演算命令によって使用されます。初期化サブルーチン (*INZSR) で指定された値によって、定義仕様で指定された値が指定変更されます。

結果フィールドには、レコード様式名、データ構造名、配列名、テーブル名、フィールド名、サブフィールド、配列エレメント、または標識が次のように入っていなければなりません。

- レコード様式または単一オカレンス・データ構造をリセット中の場合には、すべてのフィールドが、構造体内に宣言されている順序にリセットされます。

  演算項目 1 を指定する場合には、キー・フィールドをその初期値にリセットしないように指示する *NOKEY が入っていなければなりません。

  演算項目 2 を指定する場合には、レコード様式のすべてのフィールドをリセットするように指示する *ALL が入っていなければなりません。演算項目 2 が指定されていない場合には、レコード様式中の出力フィールドだけが影響を受けます。レコード様式のすべてのフィールド条件付け標識が影響を受けます。入力専用フィールドは RESET による影響を受けません。
- 複数回繰り返しデータ構造をリセットしている場合には、現行オカレンスのすべてのフィールドがリセットされます。
- 配列をリセットしている場合には、配列全体がリセットされます。
- テーブルをリセットしている場合には、現行テーブル・エレメントがリセットされます。
- 配列エレメント (標識を含む) をリセットしている場合には、指定されたエレメントしかリセットされません。

注: RESET は基底付き変数には使用できません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FEXTFILE   O    E             DISK    REMOTE
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D
D* ファイル EXTFILE には、文字フィールド CHAR1 および CHAR2、な
D* らびに数字フィールド NUM1および NUM2 が含まれている 1 つのレ
D* コード様式 RECFMT が入っています。
D
D DS1             DS
D  DAY1                    1      8    INZ('MONDAY')
D  DAY2                    9     16    INZ('THURSDAY')
D  JDATE                  17     22
D
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の演算命令は DAY1、DAY2、および JDATE をブランクに設定します。
C
C                   CLEAR                   DS1
C
C* 次の演算命令は、DAY1、DAY2、および JDATE をそれぞれその初期値
C*  'MONDAY'、'THURSDAY'、および UDATE に設定します。
C* JDATE の初期値 UDATE は *INZSR で設定されます。
C
C                   RESET                   DS1
C
C* 次の演算命令は、CHAR1 および CHAR2 をブランクに、NUM1 および
C* NUM2 をゼロに設定します。
C                   CLEAR                   RECFMT
C* 次の演算命令は、CHAR1、CHAR2、NUM1、および NUM2 をそれぞれその
C* 初期値 'NAME'、'ADDRESS'、 1、および 2 に設定します。
C* これらの初期値は *INZSR で設定されます。
C*
C                   RESET                   RECFMT
C
C     *INZSR        BEGSR
C                   MOVEL     UDATE         JDATE
C                   MOVEL     'NAME    '    CHAR1
C                   MOVEL     'ADDRESS '    CHAR2
C                   Z-ADD     1             NUM1
C                   Z-ADD     2             NUM2
C                   ENDSR
C* 次の演算命令は、レコード様式内のキー・フィールド以外のすべて
C* のフィールドをブランクに設定します。
C*
C     *NOKEY        RESET    *ALL           DBRECFMT
```

図 290. RESET 演算命令

RETURN (呼び出し元への戻り)

フリー・フォーム構文	RETURN{(HMR)} *expression*

コード	演算項目 1	拡張演算項目 2
RETURN (H M/R)		expression

RETURN 演算命令によって、次のように呼び出し元に戻ります。

- LR がオンの場合には、プログラムは正常終了して、コンポーネントは終了します。*TERMSR が実行されます。ロックされたデータ域構造体、配列、およびテーブルのすべてが書き込まれます。外部標識はリセットされます。複数のサブルーチンが呼び出されている場合には、RETURN によって、直前のアクション・サブルーチン呼び出しへ戻る原因になります。
- LR がオンでない場合には、RETURN がネストされたサブルーチン内にあるか、あるいは初期化または終了が実行中の場合を除き、現行イベントと関連したデフォルト処理が実行されます。

注: 最後のアクション・サブルーチン呼び出しから戻るまで、LR は何の効果もありません。

サブプロシージャーから戻るときに、戻り値が、呼び出し先プログラムまたはプロシージャーのプロトタイプで指定されていると、呼び出し元に渡されます。他の何かが自動的に行なわれることはありません。すべてのファイルおよびデータ域は手作業でクローズする必要があります。標識はオンにセットできますが、これでプログラム終了処理が行なわれるわけではありません。命令拡張 H、M、および R の使用法については、392 ページの『数値演算の精度の規則』を参照してください。

値を戻すサブプロシージャーでは、RETURN 演算命令がそのサブプロシージャー内にコーディングされていなければなりません。実際に戻される値は EVAL 式の左辺と同じ役割をもち、一方 RETURN 演算命令の *expression* オペランドは右辺と同じ役割をもちます。配列が戻されることがあるのは、プロトタイプで戻り値が配列として定義されている場合だけです。

値を戻すサブプロシージャーでは、プロシージャーの終わりに達する前に、RETURN 演算命令が実行されなければなりません。RETURN 演算命令が検出されないままサブプロシージャーが終了すると、例外のシグナルが呼び出し元に送られます。

ROLBK (ロールバック)

フリー・フォーム構文	ROLBK{(E)}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ROLBK (E)				_	ER	_

ROLBK 演算命令は、直前のコミットまたはロールバック演算命令以降、あるいは直前の COMMIT または ROLBK がない場合はコミットメント制御下での演算命令の開始以降に、コミットメント制御用にオープンされた OS/400 データベース・ファイルに対するすべての変更を除去します。

ROLBK 演算命令を使用できるのは、OS/400 ファイルだけです。これは、ローカル・ファイルで使用することはできません。

特定サーバーのコミットメント制御下にあるファイルのすべてのレコード・ロックが、ROLBK を出したコンポーネントとは無関係に解放されます。

注: ROLBK を発行しているコンポーネントは、すべてのファイルをコミットメント制御下にする必要がありません。

ROLBK 例外 (プログラム状況コード 802 ～ 805) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

SCAN (文字ストリングの走査)

フリー・フォーム構文	(使用できません。%SCAN 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SCAN (E)	ストリングの比較:長さ	基本ストリング:開始	左端位置	_	ER	FD

SCAN 演算命令は、基本ストリングから比較ストリングを走査します。SCAN は、演算項目 2 (開始位置によって指定されたとおり) の基本ストリングの左端の文字から開始されて、演算項目 2の文字と演算項目 1 の文字の比較が左から右へ 1 文字ずつ続行されます。ストリングは 1 桁目から索引付きです。

注:

1. 比較ストリングと基本ストリングのタイプは両方とも同じ (文字、グラフィック、または UCS-2) でなければなりません。
2. 比較ストリングに指定された先行、末尾、または組み込みブランクは、SCAN 演算命令に含まれます。
3. SCAN 演算命令では、大文字・小文字が区別されます。小文字で指定された比較ストリングは、英大文字で指定された基本ストリング中には見つからないことになります。
4. 演算項目 1、演算項目 2、または結果フィールドに表意定数を使用することはできません。
5. 演算項目 1 と結果フィールドまたは演算項目 2 と結果フィールドのデータ構造内でオーバーラップは許されません。

演算項目 1 には、比較ストリング、あるいは比較ストリングとそれにコロンと長さが続いているものが入っていなければなりません。比較ストリングには、フィールド名、配列エレメント、名前付き固定情報、データ構造名、リテラル、またはテーブル名が入っていなければなりません。長さは小数点以下の桁数なしの数字でなければならず、名前付き固定情報、配列エレメント、フィールド名、リテラル、またはテーブル名が入っていなければなりません。長さが指定されていない場合には、比較ストリングが使用されます。

演算項目 2 には、基本ストリング、あるいは基本ストリングとそれにコロンと開始位置が続いているものが入っていなければなりません。基本ストリングには、フィールド名、配列エレメント、名前付き固定情報、データ構造名、リテラル、またはテーブル名が入っていなければなりません。開始位置は小数点以下の桁数なしの数字でなければならず、名前付き固定情報、配列エレメント、フィールド名、リテラル、またはテーブル名でなければなりません。開始位置が指定されていない場合には、デフォルト値は 1 です。

注: 開始は長さより大きくすることができません。

グラフィックまたは UCS-2 ストリングを使用する場合は、開始桁および長さが 2バイト単位で測られます。

開始桁が 1 より大きい場合には、結果フィールドには、開始位置に対する相対位置ではなく、ソース・ストリングの先頭に対する比較ストリングの相対位置が入ります。

結果フィールドには、見つかれば、基本ストリング中の比較ストリングの左端桁の値が入ります。これは小数点以下の桁数なしの数字でなければならず、フィールド名、配列エレメント、配列名、またはテーブル名が入っていなければなりません。結果フィールドが指定されていない場合には、75 - 76 桁目に演算結果標識が指定されていなければなりません。結果フィールドは、ストリングが見つからない場合に 0 に設定されます。

結果フィールドに配列が入っている場合には、比較ストリングの各オカレンスがエレメント 1 の右端のオカレンスとして配列に入れられます。右端のオカレンスが入っているエレメントの次の配列エレメントはすべてゼロです。結果の配列は、演算項目 2 に指定された基本ストリングのフィールド長だけの大きさになっている必要があります。

結果フィールドが数字配列である場合には、配列中にある数だけのオカレンスが注目されます。オカレンスが見つからない場合には、結果フィールドはゼロに設定されます。

SCAN 例外 (プログラム状況コード 100) を処理するために、命令コード拡張 'E' またはエラー標識 ERのいずれかを指定することができますが、両方とも指定することはできません。開始桁が演算項目 2 の長さより大きい場合、あるいは演算項目 1 の値が大きすぎる場合には、エラーが起こります。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

走査中のストリングが見つかるとオンにセットされる標識は 75 - 76 桁目に指定することができます。この情報は、一致が見つかると '1' を戻す %FOUND 組み込み関数から取得することもできます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* SCAN 演算命令は、演算項目 2 の 3 桁目から始まっているサブスト
C* リング 'ABC' を検索します。3 が結果フィールドに入れられます。
C* ストリングが見つからないと、標識 90 がオンにセットされます。
C* 開始桁が指定されていないので、デフォルトの 1 が使用されます。
C     'ABC'         SCAN      'XCABCD'      RESULT                  90
C*
C* この SCAN 演算命令は、演算項目 1 の 3 桁目から始まっているス
C* トリングのオカレンスを演算項目 2 のストリングから走査します。
C* 走査演算命令は 3 桁目から開始されるので、基本ストリングの 1
C* 桁目の 'Y' は無視されます。
C* この演算命令で、値 5 および 6 が配列の 1 番目および 2 番目の
C* エレメントに入れられます。標識 90 はオンにセットされます。
C
C                   MOVE      'YARRYY'      FIELD1            6
C                   MOVE      'Y'           FIELD2            1
C     FIELD2        SCAN      FIELD1:3      ARRAY                   90
C                   C* この SCAN 演算命令は、演算項目 2 の 2 桁目から始まっているス
C* トリングから、演算項目 1 の長さ 4 のストリングのオカレンスを
C* 走査します。'TOOL' が FIELD1 には見つからないので、INT はゼロ
C* に設定されて、標識 90 はオフにセットされます。
C
C                   MOVE      'TESTING'     FIELD1            7
C                   Z-ADD     2             X                 1 0
C                   MOVEL     'TOOL'        FIELD2            5
C     FIELD2:4      SCAN      FIELD1:X      INT90            20      90
C
C*
C* この SCAN 演算命令は名前を検索中です。名前が見つかると、%FOUND
C* は '1' を戻すので、HandleLine が呼び出されます。
C*
C     SrchName      SCAN      Line
C                   IF        %FOUND
C                   EXSR      HandleLine
C                   ENDIF
```

図 291. SCAN 演算命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++
D*
D*       グラフィック SCAN の例
D*
D*       Graffld の値はグラフィック 'AACCBBGG' です。
D*       走査後の Number の値は、3 番目のグラフィック文字が演算項目 1 の
D*       値と一致すると 3 です。
D
D Graffld                       4G   inz(G'AACCBBGG')
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* SCAN 演算命令は、演算項目 2 のグラフィック・ストリングから演算項目 1 の
C* グラフィック・リテラルのオカレンスを走査します。これはグラフィック演算命令なので
C* SCAN は一度に 2 バイトに対して作動します。
C
C     G'BB'         SCAN      Graffld:2     Number            5 0    90
C
```

図 292. グラフィックを使用する SCAN 演算命令

SELECT (選択グループの始め)

フリー・フォーム構文	SELECT

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
SELECT						

選択グループは、幾つかの演算命令の代替順序のいずれかを条件付きで処理します。これは、次から成っています。
- SELECT ステートメント
- ゼロまたは 1 つ以上の WHENxx または WHEN グループ
- 任意選択の OTHER
- ENDSL または END ステートメント

SELECT 演算命令の後で、満たされている最初の WHENxx 条件の次のステートメントに制御が渡されます。その後で、すべてのステートメントが、次の WHENxx 演算命令まで実行されます。制御は ENDSL ステートメントに渡されます (ただ 1 つの WHENxx が実行されます)。WHENxx 条件が満たされていないで、OTHER演算命令が指定されている場合には、制御は OTHER 演算命令の次のステートメントに渡されます。WHENxx条件が満たされていないで、OTHER 演算命令が指定されていない場合には、制御は選択グループの ENDSL 演算命令の次のステートメントに転送されます。

条件付け標識は SELECT 演算命令で使用することができます。その標識が満たされない場合には、制御は選択グループの ENDSL 演算命令の次のステートメントに即時に渡されます。条件標識は、WHENxx、WHEN、 OTHER、および ENDSL 演算命令で個々に使用することはできません。

選択グループは、IF、DO、またはその他の選択グループ内でネストすることができます。IF グループおよび DO グループは、選択グループ内でネストすることができます。

SELECT 演算命令が選択グループの内側に指定されている場合には、ENDSL が指定されるまで、WHENxx 演算命令および OTHER 演算命令は新規選択グループに適用されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の例では、X が 1 と等しければ、順序 1 の演算命令が実行され
C* ます (次の WHENxx の前に END が必要ないことに注意してくださ
C* い)。X が 1 と等しくない場合、および Y=2 かつ X<10 の場合
C* には、順序 2 の演算命令が実行されます。どちらの条件も真でな
C* ければ、順序 3 の演算命令が実行されます。
C*
C                   SELECT
C                   WHEN      X = 1
C                   Z-ADD     A             B
C                   MOVE      C             D
C*  順序 1
C                   :
C                   WHEN      ((Y = 2) AND (X < 10))
C*  順序 2
C                   :
C                   OTHER
C*  順序 3
C                   :
C                   ENDSL
C*
C* 次の例は、条件付け標識が指定された選択グループを示しています。
C* CHAIN 演算命令後に、標識 10 がオンになっていると、制御は ADD
C* 演算命令に渡されます。標識 10 がオフになっていると、選択グルー
C* プが処理されます。
C*
C     KEY           CHAIN     FILE                              10
C  N10              SELECT
C                   WHEN      X = 1
C*  順序 1
C                   :
C                   WHEN      Y = 2
C*  順序 2
C                   :
C                   ENDSL
C                   ADD       1              N
```

図 293. SELECT 演算命令

SETATR (属性の設定)

フリー・フォーム構文	(使用できません。%SETATR 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SETATR (E)	パーツ名	属性値	属性	_	ER	_

SETATR 演算命令はパーツの属性を設定します。パーツの属性を設定できるのは、そのパーツが作成されている場合だけです。

注:

1. SETATR 演算命令は、複数リンク・アクション・サブルーチンに使用することができます。複数リンク・アクション・サブルーチンの説明については、 488 ページの『BEGACT (アクション・サブルーチンの開始)』 を参照してください。親ウィンドウ以外のウィンドウのパーツの属性を設定するには、%SETATR 組み込み関数を使用してください。 %SETATR 組み込み関数の説明については、 456 ページの『%SETATR (属性の設定)』 を参照してください。
2. SETATR 演算命令は、1 バイトおよび 8 バイトの符号付き整数値および符号なし整数値とユニコード値をサポートしません。

演算項目 1 を指定する場合には、パーツの名前、または属性を設定しようとしているパーツの名前が入っているフィールドが入っていなければなりません。

演算項目 2 には、属性の新規値が入っていなければなりません。これは、リテラル、名前付き固定情報、表意定数、または属性の新規値が入っているフィールドでなければなりません。

結果フィールドには、属性名または属性の名前が入っているフィールドが入っていなければなりません。

SETATR 例外 (プログラム状況コード 1400、1402 - 1404) を処理するために、命令コード拡張 'E'またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、 52 ページの『プログラム例外およびエラー』を参照してください。

演算結果標識が指定されている場合には、SETATR 演算命令が正常に完了しないとオンにセットされます。

注: %SETATR 組み込み関数は複数のパーツの対応するプログラム・フィールドには影響しません。 属性値およびプログラム・フィールド中の値が同一であることを保証するには、属性値の設定時に、このプログラム・フィールドを使用してください。これは、それらにマップされるプログラム・フィールド (TEXT 属性が指定されている入力フィールドなど) がある属性に適用されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                            Extended-factor2++++++++++++++++++++++++++++
C*
C* BUTTON1 と呼ばれるボタンのラベルを変更します。
C*
C     'BUTTON1'     SETATR    Cancel        'LABEL'
```

図 294. SETATR 演算命令

SETGT (より大きいレコードへのセット)

フリー・フォーム構文	SETGT{(E)} *search-arg name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SETGT (E)	検索引き数	name (ファイルまたはレコード様式)		NR	ER	_

SETGT 演算命令は、キーまたは相対レコード番号が検索引き数より大きい次レコードにファイルを位置決めします。ファイルは、全手順ファイル (ファイル仕様の 18 桁目の F によって識別される) でなければなりません。

SETGT 演算命令を使用できるのは、OS/400 ファイルだけです。

検索引き数オペランド (*search-arg*) は必須です。ファイルがキーによってアクセスされる場合には、*search-arg* は、ファイルの位置決めで検索引き数として使用されるフィールド名、名前付き固定情報、表意定数、またはリテラルとすることができます。キーにより位置決めされる外部記述ファイルの場合、*search-arg* に KLIST 名を指定することもできます。ファイルが相対レコード番号によってアクセスされる場合には、*search-arg* は、整数リテラル、名前付き固定情報、またはフィールドでなければなりません。グラフィックおよび UCS-2 キー・フィールドの CCSID は、ファイル内のキーと同じでなければなりません。

name オペランドは必須で、ファイル名またはレコード様式名のいずれかでなければなりません。レコード様式名を使用できるのは、外部記述ファイルの場合だけです。MBR(*ALL) が指定されていると、SETGT は最初のオープン・ファイル・メンバーしか処理しません。

指定された検索引き数 (*search-arg*) より大きいキーまたは相対レコード番号をもつレコードが見つからない場合にオンに設定される標識を 71 ～ 72 桁目に指定することができます。この情報は、レコードが見つからないと '0' を、レコードが見つかると '1' を戻す %FOUND 組み込み関数から取得することもできます。

SETGT 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

SETGT 演算命令が成功しない (「該当レコードなし」条件) 場合には、ファイルがそのファイルの終わりに位置付けられます。

SETGT 演算命令が実行されると、ファイルは、キーまたは相対レコード番号が指定された検索引き数 (*search-arg*) より大きい最初のレコードの直前になるように位置付けられます。このレコードは、ファイルを読み取ることによって取り出すことができます。*START および *END は、ファイルの位置決めに使用することができます。*search-arg*に *STARTまたは *END のいずれかを指定する場合には、以下のことに注意してください。

- *name* はファイル名でなければなりません。

- *HIVAL または *LOVAL を *search-arg* として使用できます。
- NR 標識を指定することはできません。

詳細については、 6 ページの『ファイルの位置決め』 を参照してください。

ヌル可能フィールドがあるレコードの読み取りについては、 145 ページの『データベースのヌル値サポート』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*  この例は、READ が次レコードを読み取るようにファイルを位置決
C*  めする方法を示しています。SETGT 演算命令に指定された検索引き
C*  数 KEY の値は 98 です。したがって、SETGT は、キー値が 98 よ
C*  り大きいファイル様式 FILEA の最初のレコードの前にファイルを
C*  位置決めします。このファイルはキー値が 100 の最初のレコード
C*  の前に位置決めされます。READ 演算命令は、そのキー・フィール
C*  ドの値が 100 のレコードを読み取ります。
C*
C
C     KEY           SETGT     FILEA
C                   READ      FILEA                                  64
C*
C*  この例は、キー値と様式が同じレコードのグループの最後のレコー
C*  ドをプログラム記述ファイルから読み取る方法を示しています。
C*  SETGT 演算命令に指定された検索引き数 KEY によって、ファイル
C*  FILEB は、キー・フィールド値が 70 より大きいファイルの最初の
C*  レコードの前に位置決めされます。
C*  このファイルは、キー値が 80 の最初のレコードの前に位置決めさ
C*  れます。READP 演算命令は、キー・フィールドのキー値が 70 の最
C*  後のレコードを読み取ります。
C
C     KEY           SETGT     FILEB
C                   READP     FILEB                                  64
```

図 295. SETGT 演算命令

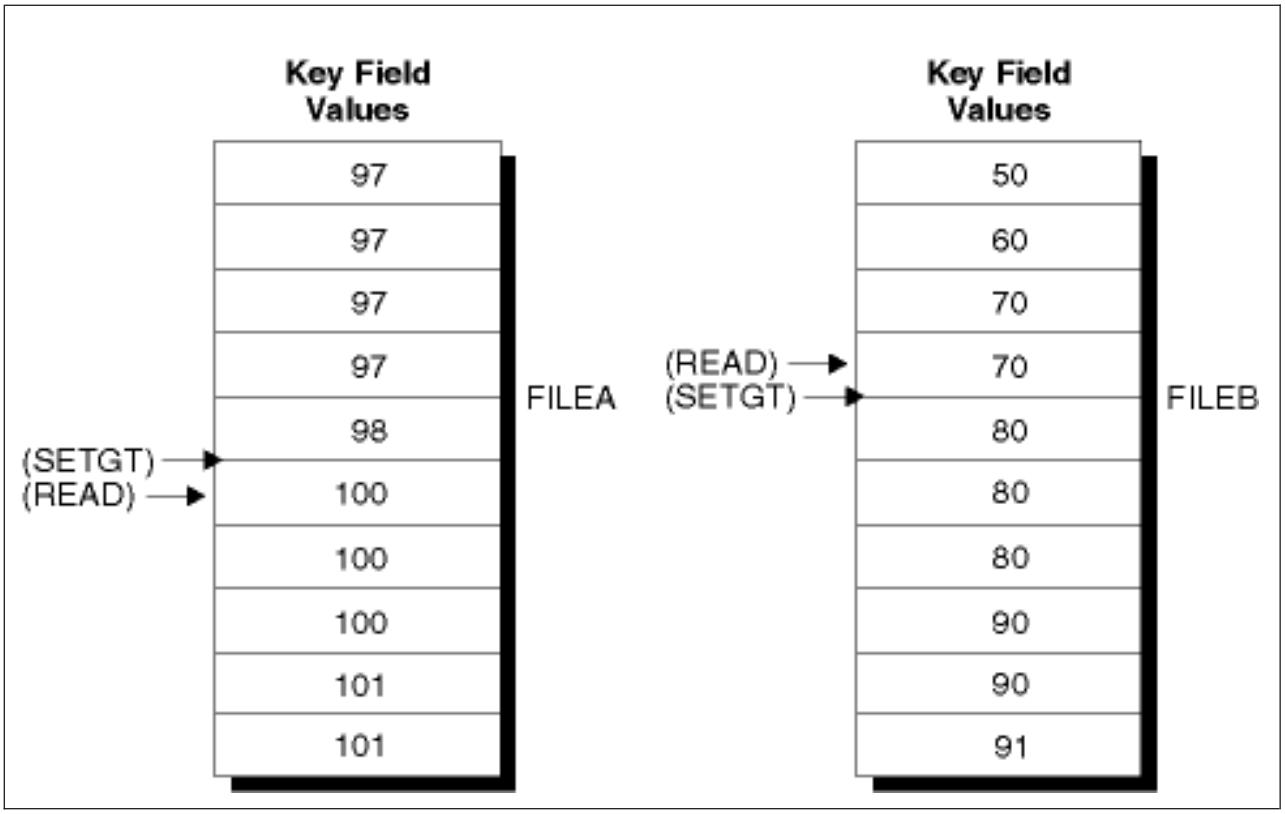


図 296. SETGT を使用したファイルの位置決め

SETLL (下限のセット)

フリー・フォーム構文	SETLL{(E)} *search-arg name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SETLL (E)	検索引き数	name (ファイルまたはレコード様式)		NR	ER	EQ

SETLL 演算命令は、キーまたは相対レコード番号が検索引き数より大か等しい次レコードにファイルを位置決めします。ファイルは、全手順ファイル (ファイル仕様の 18 桁目の F によって識別される) でなければなりません。

SETLL 演算命令を使用できるのは、OS/400 ファイルだけです。これは、ローカル・ファイルで使用することはできません。

検索引き数オペランド (*search-arg*) は必須です。ファイルがキーによってアクセスされる場合には、*search-arg* は、ファイルの位置決めで検索引き数として使用されるフィールド名、名前付き固定情報、表意定数、またはリテラルとすることができます。キーにより位置決めされる外部記述ファイルの場合、*search-arg* に KLIST 名を指定することもできます。ファイルが相対レコード番号によってアクセスされる場合には、*search-arg*には、整数リテラル、名前付き固定情報、または小数点以下の桁数なしの数字フィールドが入っていなければなりません。グラフィックおよび UCS-2 キー・フィールドの CCSID は、ファイル内のキーと同じでなければなりません。

name オペランドは必須で、ファイル名またはレコード様式名のいずれかを入れることができます。レコード様式名を使用できるのは、外部記述ファイルの場合だけです。 MBR(*ALL) が指定されていると、SETLL は最初のオープン・ファイル・メンバーしか処理しません。

演算結果標識は、演算命令の状況を反映します。検索引き数がファイル中の最大キーまたは相対レコード番号より大きい場合にオンに説とされる標識は、71 - 72 桁目に指定することができます。この情報は、レコードが見つからないと '0' を、レコードが見つかると '1' を戻す %FOUND 組み込み関数から取得することもできます。

SETLL 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

キーまたは相対レコード番号が検索引き数と等しいレコードが存在している場合にオンにセットされる標識は、75 - 76 桁目に指定することができます。この情報は、正確な一致が見つかると '1'を戻す %EQUAL 組み込み関数から取得することもできます。

name が下限を設定する対象のファイル名である場合には、そのファイルは、キーまたは相対レコード番号が検索引き数以上の最初のレコードに位置決めされます。

name が下限を設定する対象のレコード様式名である場合には、そのファイルは、キーが検索引き数以上の指定タイプの最初のレコードに位置決めされます。

SETLL によって処理中のファイルでファイルの終わりに達すると、そのファイルを再位置決めするために別の SETLL を発行することができます。SETLL 演算命令がファイルをレコードに正常に位置決めした後で、レコードはそれを読み取ることによって検索することができます。*START および *END は、ファイルの位置決めに使用することができます。*search-arg*に *STARTまたは *END のいずれかを指定する場合には、以下のことに注意してください。

- *name* はファイル名でなければなりません。
- *HIVAL または *LOVAL を *search-arg* に使用することはできません。
- NR または EQ 標識を指定することはできません。
- エラー標識 (73 - 74 桁目) または 'E' 拡張を指定することができます。

*START および *END について詳しくは、6 ページの『ファイルの位置決め』を参照してください。

ユーザー・アプリケーションがこのファイルを読み取る前に、別のアプリケーションがファイルからレコードを削除する場合があります。必要としているレコードを検索してはいけません。 %EQUAL 組み込み関数もオンにセットされているか、あるいは一致しているレコードが見つかっていることを示すために 75 - 76 桁目の演算結果標識がオンにセットされていても、そのレコードを取り出してはいけません。

SETLL は、データ・レコードにアクセスしません。キーが存在していることを検査するには、CHAIN 演算命令よりむしろ、等しい標識 (75 - 76 桁目) を指定した SETLL または %EQUAL 組み込み関数を使用してください。ファイルが、キーが散在している複数様式論理ファイルである場合には、CHAIN の方が SETLLより高速のソリューションとなる可能性があります。

ヌル可能フィールドがあるレコードの読み取りについては、145 ページの『データベースのヌル値サポート』 を参照してください。

次の例では、ファイル ORDFIL にはオーダー・レコードが入っています。キー・フィールドは、オーダー番号 (ORDER) フィールドです。レコードはオーダーごとに複数あります。演算仕様で ORDFILは、次のようになります。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  ORDFIL 中の 101 レコードのすべてが印刷されます。値 101 は、
C*  以前に ORDER に入れられています。SETLL 演算命令は、キー値が
C*  101 の最初のレコードにファイルを位置決めします。
C*  %EQUAL は '1' を戻します。
C
C     ORDER         SETLL     ORDFIL
C
C*  次の DO ループは、キー値が同じすべてのレコードを処理します。
C*
C*
C                   IF        %EQUAL
C                   DOU       %EOF
C     ORDER         READE     ORDFIL
C                   IF        NOT %EOF
C                   EXCEPT    DETAIL
C                   ENDIF
C                   ENDDO
C                   ENDIF
C
C*
C*  READE 演算命令は、最初の 101 レコードを読み取ったのと同じ方
C*  法で 2 番目、3 番目、および 4 番目の 101 レコードを読み取り
C*  ます。4 番目の 101 レコードの読み取り後に、READE 演算命令が
C*  試みられます。102 レコードは同一グループに属していないので、
C*  %EOF は '1' を戻し、EXCEPT 演算命令がバイパスされて、
C*  DOU ループは終了します。
```

図 297. SETLL 演算命令

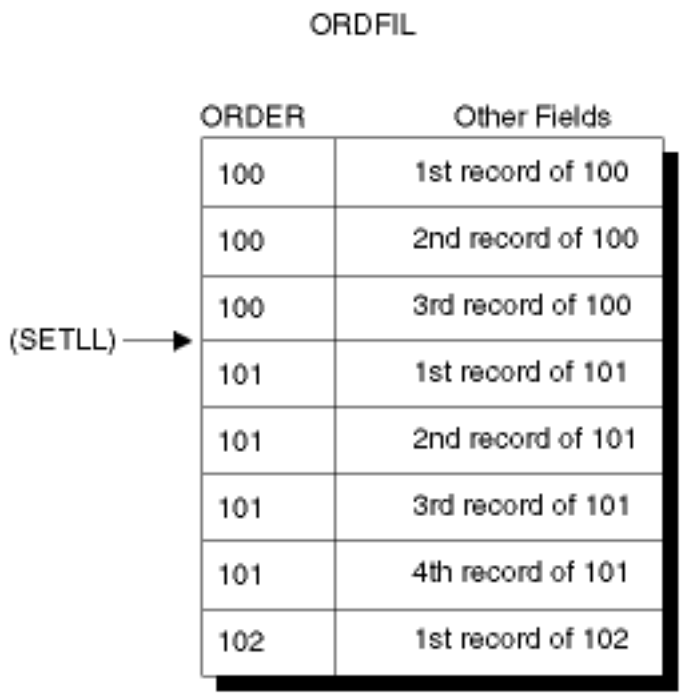


図 298. SETLL を使用したファイルの位置決め

SETOFF (標識をオフにセット)

フリー・フォーム構文	(使用できません。EVAL *INxx = *OFF を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SETOFF				OF	OF	OF

SETOFF 演算命令は、71 - 76 桁目に指定されたすべての標識をオフにセットします。71 - 76 桁目には、演算結果標識が少なくとも 1 つは指定されていなければなりません。

図 299 には、SETOFF 演算命令が説明されています。

SETON (標識をオンにセット)

フリー・フォーム構文	(使用できません - EVAL *INxx = *ON を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SETON				ON	ON	ON

SETON 演算命令は、71 - 76 桁目に指定されたすべての標識をオンにセットします。71 - 76 桁目には、演算結果標識が少なくとも 1 つは指定されていなければなりません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  SETON および SETOFF 演算命令は、71 - 76 桁目に指定された 1
C*  - 3 個の標識をオンおよびオフにセットします。
C*  SETON 演算命令は、標識 17 をオンにセットします。
C
C                   SETON                                      17
C
C*  SETON 演算命令は標識 17 および 18 をオンにセットします。
C
C                   SETON                                      1718
C
C*  SETOFF 演算命令は標識 21 をオフにセットします。
C
C                   SETOFF                                     21
```

図 299. SETON および SETOFF 演算命令

SHOWWIN (ウィンドウの表示)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SHOWWIN (E)		ウィンドウ名		_	ER	_

SHOWWIN 演算命令は、ウィンドウをメモリーにロードします。「可視」属性は、ウィンドウを表示するかどうかを制御します。

注: ウィンドウの属性は、SHOWWIN 演算命令の前に設定するか、あるいは参照することができません。ウィンドウ上のパーツは、そのウィンドウのための SHOWWIN 演算命令の前に参照することができません。

演算項目 2 には、表示するウィンドウの名前が入ります。

SHOWWIN 例外 (プログラム状況コード 1400 - 1420) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

SHOWWIN および「即時オープン」属性を使用しないでください。使用すると、演算結果標識がオンにセットされます。これは、ウィンドウがすでにロードされているという意味です。この標識がオンにセットされている場合には、そのウィンドウの「可視」属性を設定することができます。

SHOWWIN 演算命令は、ウィンドウを「即時オープン」に設定するよりもむしろ頻繁に表示されないウィンドウを表示するために使用してください。プライマリー・ウィンドウ (アプリケーションが表示する最初のウィンドウ) の場合は、ウィンドウには SHOWWIN よりもむしろ「即時オープン」設定を使用してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                            Extended-factor2++++++++++++++++++++++++++++++
C*
C* UPDCUST という名前のウィンドウが表示されます。
C                   SHOWWIN   'UPDCUST'
```

図 300. SHOWWIN 演算命令

SORTA (配列の分類)

フリー・フォーム構文	SORTA *array-name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SORTA		配列名				

SORTA 演算命令は、*array-name* で指定された配列を昇順または降順に分類します。この順序は、定義仕様で指定されます。この順序が指定されていない場合には、配列は昇順にされます。

array-name オペランドは、分類する配列の名前です。*IN を指定することはできません。配列が交互形式のデータがあるコンパイル時配列または実行時前配列として定義されている場合には、交互配列は分類されません。

配列が OVERLAY(name{:pos}) キーワードで定義されている場合には、基本配列は OVERLAY 配列によって定義された順序でソートされます。

グラフィック配列は、配列エレメントの 16 進値によって、定義仕様に指定された順序で分類されます。

配列をソートすると以前の順序はどれも保存されません。たとえば、異なるオーバーレイ配列を使用して配列が 2 度分類されると、最終順序が最終分類の順序になります。ソート・シーケンスは等しいが、16 進値が異なっているエレメント (たとえば、順序を判別するためにオーバーレイ配列を使用するため) は、ソート後は以前の順序と同じ順序にならない場合があります。

基底ポインターの配列の分類時に、配列中のすべての値は、必ず同一スペース内でアドレス指定されているようにしなければなりません。そうでない場合には、矛盾する結果が起こる場合があります。詳しくは、361 ページの『比較命令』を参照してください。

ヌル可能配列が分類されると、その分類ではヌル・フラグの設定は考慮に入れられません。

定義済みのすべてのエレメントが割り振られていない動的割り振り配列を分類すると、エラーが起こる原因になる場合があります。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DARRY             S              1A   DIM(8) ASCEND
D
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  SORTA 演算命令は、ASCEND キーワードが指定されているので、
C*  ARRY を昇順に分類します。
C*  未分類の ARRY の内容が GT1BA2L0 だった場合には、分類済み ARRY
C*  の内容は 012ABGLT になります。
C*  ASCII 分類順序が使用されることに注意してください。
C
C                   SORTA     ARRY
```

図 301. SORTA 演算命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D* この例では、基本配列の値は aa44 bb33 cc22 dd11 なので、
D* オーバーレイされた配列 ARRO の値は 44 33 22 11 です。
D                 DS
D ARR                             4    DIM(4) ASCEND
D ARRO                            2    OVERLAY(ARR:3)
D
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C* SORTA 演算命令後の基本配列の値は
C* 11dd 22cc 33bb 44aa です
C
C                   SORTA     ARRO
```

図 302. OVERLAY が指定された SORTA 演算命令

SQRT (平方根)

フリー・フォーム構文	(使用できません。%SQRT 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SQRT (H)		値	根			

SQRT 演算命令は、演算項目 2 に指定されたフィールドの平方根を導きます。演算項目 2 の平方根が結果フィールドに入れられます。

演算項目 2 は、数字でなければなりませんが、配列、配列エレメント、フィールド、表意定数、リテラル、名前付き固定情報、サブフィールド、またはテーブル名を含めることができます。演算項目 2 フィールドの値がゼロになっていると、結果フィールド値もゼロになります。演算項目 2 の値が負になっていると、VRPGクライアント例外 / エラー処理ルーチンが制御を受け取ります。

結果フィールドは数字でなければならず、配列、配列エレメント、サブフィールド、またはテーブル・エレメントを入れることができます。

演算項目 2 および結果フィールドに配列名が入っている場合には、SQRT 演算命令で配列全体を使用することができます。結果フィールドの小数点以下の桁数は、演算項目 2 の小数点以下の桁数より小さくても大きくてもかまいません。しかし、結果フィールドの小数点以下の桁数が演算項目 2 の小数点以下の桁数の半分未満としてはいけません。

352 ページの『算術演算』には、算術演算を指定するための一般規則が説明されています。

355 ページの図 114 には、SQRT 演算命令の例が示されています。

START (コンポーネント開始またはローカル・プログラム呼び出し)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
START (E)		コンポーネント名またはフィールド名	PLIST 名	_	ER	_

START 演算命令は、アプリケーションで新コンポーネントを開始するか、あるいはローカル・プログラムを呼び出すかのいずれかのために使用することができます。新コンポーネントを開始するすべての START 演算命令の場合は、STOP 演算命令があってもかまいません。

START 例外 (プログラム状況コード 1410) を処理するために、命令コード拡張 'E' またはエラー標識 ERのいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

コンポーネントの開始

START は非同期動作なので、アプリケーション内の他のすべてのアクティブ・コンポーネントの他に、呼び出し先 (ターゲット) コンポーネントと呼び出し元 (ソース) コンポーネントは両方とも、すべてのアプリケーションのコンポーネントによって現在使用可能になっているパーツについてのイベントを受け取ることができます。

演算項目 2 を指定する場合には、コンポーネントの名前である文字リテラルまたはコンポーネント名が入っている変数が入っていなければなりません。

結果フィールドを指定する場合には、PLIST 名が入っていなければなりません。結果フィールドを指定しない場合には、START 演算命令の後に PARM 演算命令を続けることができます。パラメーターはアドレスによって渡されます。これは、ソースおよびターゲット・コンポーネントがパラメーター・フィールドにアクセスすることができるという意味です。20 個までのパラメーターを指定することができます。開始済みコンポーネントが可変長フィールドを必要としている場合には、可変長フィールドをパラメーターとして使用しなければなりません。

START はコンポーネントを初期化し、その *INZSR を実行してから、ソース・コンポーネントに戻ります。ソース・コンポーネントでは、PARM 演算命令の演算項目 2 がその PARM 演算命令の結果フィールドにコピーされます。制御がソース・コンポーネントに戻されるときに、結果フィールドが演算項目 1 にコピーされます。ターゲット・コンポーネントでは、結果フィールドが演算項目 1 にコピーされます。制御がソース・コンポーネントに戻されるときに、*INZSR が成功すると、演算項目 2 が結果フィールドにコピーされます。

START 演算命令がターゲット・コンポーネントの初期化を完了すると、ソース・コンポーネント内のアクション・サブルーチンは実行を続行して、イベントを受け取るためにそのアクション・サブルーチンが使用可能になっているターゲット・コンポーネントはアクティブのままです。

ローカル・プログラムの呼び出し

ローカル・プログラムを呼び出すために START が使用されると、呼び出し元プログラムがローカル・プログラムに対する呼び出しを行なってから続行されます。呼び出し先プログラムは呼び出し元プログラムとは独立に実行されます。

演算項目 2 を指定する場合には、固定情報またはフィールド定義の定義仕様名が入っていなければなりません。LINKAGE(*CLIENT) が定義仕様になければなりません。詳細については、 281 ページの『LINKAGE(linkage_type)』を参照してください。

結果フィールドを指定する場合には、PLIST 名が入っていなければなりません。結果フィールドを指定しない場合には、START 演算命令の後に PARM 演算命令を続けることができます。パラメーターは参照によって渡されます。パラメーターの渡しについての詳細は、 633 ページの『PARM (パラメーターの識別)』および 636 ページの『PLIST (パラメーター・リストの識別)』を参照してください。

注: ポインターおよびプロシージャー・ポインターは、パラメーターとして使用できません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++Comments++++++++++++
Dtest2            C                   'testprog'   LINKAGE (*CLIENT)
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                   START     test2
```

図 303. START 演算命令

STOP (コンポーネントの停止)

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
STOP (E)		コンポーネント名			ER	

STOP 演算命令は、アプリケーション内の 1 つ以上のコンポーネントを停止します。終了中のコンポーネントによって開始されている場合があるすべての子コンポーネントが最初に終了されます。

コンポーネントの終了時には、次のとおりです。
1. 終了中のコンポーネントまたはコンポーネントの子孫によって開始されているコンポーネントは、逆の階層順序で終了されます。
2. *TERMSR サブルーチンは、*TERMSR が定義されている終了中のコンポーネントごとに呼び出されます。
3. *STATUS コードは PSDS に入れられます。

演算項目 2 を指定する場合には、コンポーネントの名前である文字リテラルまたはコンポーネント名が入っている変数が入っていなければなりません。演算項目 2 が指定されていない場合あるいは演算項目 2 に現在実行中のコンポーネント (STOP 演算命令が含まれているコンポーネント) と同じコンポーネント名が入っている場合には、そのコンポーネントが終了します。

現在実行中のコンポーネントに影響を与える STOP が実行されると、その STOP の後の演算命令は実行されません。たとえば、COMPA は COMPB を開始します。COMPB が現在実行中のコンポーネントである場合および COMPBが COMPA のために STOP を発行する場合には、COMPB が最初に終了してから、COMPA が終了します。STOPの後の演算命令は実行されません。

STOP 例外を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                            Extended-factor2++++++++++++++++++++++++++++++
C*
C                   STOP     'COMPX'
```

図 304. STOP 演算命令

SUB (減算)

フリー・フォーム構文	(使用できません。- 演算子を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SUB (H)	被減数	減数	差	+	–	Z

演算項目 1 が指定されていると、演算項目 2 が演算項目 1 から減算されて、差が結果フィールドに入れられます。演算項目 1 が指定されていない場合には、演算項目 2 が結果フィールドから減算されます。

演算項目 1 および演算項目 2 は数字でなければならず、それぞれは配列、配列エレメント、フィールド、表意定数、リテラル、名前付き固定情報、サブフィールド、またはテーブル名とすることができます。

結果フィールドは数字でなければならず、配列、配列エレメント、サブフィールド、またはテーブルの名前を入れることができます。

352 ページの『算術演算』には、算術演算を指定するための一般規則が説明されています。

355 ページの図 114 には、SUB 演算命令の例が示されています。

SUBDUR (期間の減算)

フリー・フォーム構文	(使用できません。- 演算子または %DIFF 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SUBDUR (E) (期間)	日付 / 時刻 / タイム・スタンプ	日付 / 時刻 / タイム・スタンプ	期間: 期間コード	_	ER	_
SUBDUR (E) (新規日付)	日付 / 時刻 / タイム・スタンプ	期間:期間コード	日付 / 時刻 / タイム・スタンプ	_	ER	_

SUBDUR 演算命令は、次の目的で使用することができます。
- 新規日付、時刻、またはタイム・スタンプを設定するために期間を減算します。
- 期間を計算します。

685 ページの図 305 には、SUBDUR 演算命令が説明されています。

期間の減算

SUBDUR 演算命令は、演算項目 2 に指定された期間を演算項目 1 に指定されたフィールドまたは固定情報から減算して、得られる日付、時刻、またはタイム・スタンプを結果フィールドに指定されたフィールドに入れるために使用することができます。

演算項目 1 は任意指定で、日付、時刻、またはタイム・スタンプ・フィールド、配列、配列エレメント、リテラル、あるいは固定情報を入れることができます。演算項目 1 にフィールド名、配列、または配列エレメントが入っている場合には、そのデータ・タイプは結果フィールドに指定されたフィールドと同じタイプでなければなりません。演算項目 1 が指定されていない場合には、期間が結果フィールドに指定されたフィールドから減算されます。

演算項目 2 は必須で、2 つの副演算項目が入ります。最初は、数字フィールド、配列、または小数点以下の桁数がゼロの固定情報でなければならない期間です。期間は 15 桁以下でなければなりません。期間フィールドが負になっていると、期間がこのフィールドに加算されます。

2 番目の副演算項目は、期間のタイプを示している有効な期間コードでなければなりません。期間コードは、結果フィールドのデータ・タイプと整合性がなければなりません。たとえば、年、月、または日期間を減算することができますが、期間を日付フィールドから減算することはできません。期間コードおよびその簡易フォームのリストについては、363 ページの『日付の演算』 を参照してください。

結果フィールドは、日付、時刻、またはタイム・スタンプデータ・タイプ・フィールド、配列、または配列エレメントでなければなりません。演算項目 1 がブランクのままになっていると、期間は結果フィールドの値から減算されます。結果フィールドが配列の場合には、演算項目 2 の値がその配列中の各エレメントから減算されます。結果が時刻有効の場合には、結果は常に有効な時刻になります。たとえば、

00:58:59 から 59 分を減算すると、-00:00:01 が得られます。この時刻が無効なので、コンパイラーは 23:59:59 に調整します。

月数の期間を日付から減算する場合の一般規則は、月部分が期間の月数だけ減らされて、日部分は未変更であるというものです。この例外は、結果の日の部分が結果の月の実際の日数を超える場合です。その場合には、結果の日の部分を調整して、実際の月の終わりの日付に合わせます。 次の例では、*YMD 形式を想定して、この点を説明します。

'95/05/30' SUBDUR 1:*MONTH は '95/04/30' になります

得られる月部分が 1 だけ減らされて、日部分は未変更です。

'95/05/31' SUBDUR 1:*MONTH は '95/04/30' になります

得られる月部分が 1 だけ減らされて、4 月は 30 日までしかないので、日部分が調整されます。

年期間の減算時に同様の結果が起こります。たとえば、'92/02/29' から 1 年を減算すると、得られる年がうるう年ではないので、調整された '91/02/28' が得られます。

プログラム状況コード 103、112、または 113 での例外を処理するために、命令コード拡張 'E'またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

期間の計算

SUBDUR 演算命令は、次の間の期間を計算するために使用することができます。

- 2 つの日付
- 日付とタイム・スタンプ
- 2 つの時刻
- 時刻とタイム・スタンプ
- 2 つのタイム・スタンプ

演算項目 1 は必須で、日付、時刻、またはタイム・スタンプ・フィールド、サブフィールド、配列、配列エレメント、固定情報、またはリテラルが入っていなければなりません。

演算項目 2 は必須で、これにも日付、時刻、またはタイム・スタンプ・フィールド、配列、配列エレメント、リテラル、または固定情報が入っていなければなりません。

以下の期間コードが有効です。

- 2 つの日付あるいは日付とタイム・スタンプの場合: *DAYS (*D)、*MONTHS (*M)、および *YEARS (*Y)
- 2 つの時刻あるいは時刻とタイム・スタンプの場合: *SECONDS (*S)、*MINUTES (*MN)、および *HOURS (*H)
- 2 つのタイム・スタンプの場合: *MSECONDS (*MS)、*SECONDS (*S)、*MINUTES (*MN)、*HOURS(*H)、*DAYS (*D)、*MONTHS (*M)、および *YEARS (*Y).

結果は、剰余が廃棄された整数単位数です。たとえば、61 分は 1 時間に、59 分は 0 になります。

結果フィールドは、2 つの副演算項目で構成されます。最初は、演算命令の結果が入れられる小数部がゼロの数字フィールド、配列、または配列エレメントの名前です。2 番目の副演算項目には、期間のタイプを示している期間コードが入ります。期間コードは、演算項目 1 および演算項目 2 と整合性がなければなりません。演算項目 1 の日付が演算項目 2 の日付より早い場合には、結果フィールドは負になります。

日付時刻フィールドの処理について詳しくは、363 ページの『日付の演算』を参照してください。

マイクロ秒期間 (*mseconds) の計算は、期間のシステム限界である 15 桁を超える可能性があります。これは、エラーまたは切り捨ての原因になります。この状態が起こるのは、演算項目 1 と演算項目 2 の間の差が 32 年および 9 か月より多い場合です。

プログラム状況コード 103、112、または 113 での例外を処理するために、命令コード拡張 'E'またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
C* DUEDATE の前に、xx 年、yy 月、zz 日である LOANDATE を
C* 判別します。
 *
C     DUEDATE       SUBDUR    XX:*YEARS     LOANDATE
C                   SUBDUR    YY:*MONTHS    LOANDATE
C                   SUBDUR    ZZ:*DAYS      LOANDATE
 *
C* ローン返済期日に 30 日を加算します
C*
C                   SUBDUR    -30:*D        LOANDUE
 *
C* LOANDATE と DUEDATE の間の数値または日数を計算します。
 *
C     LOANDATE      SUBDUR    DUEDATE       NUM_DAYS:*D       5 0
 *
C* LOANDATE と DUEDATE の間の秒数を判別します。
 *
C     LOANDATE      SUBDUR    DUEDATE       NUM_SECS:*S       5 0
```

図 305. SUBDUR 演算命令

SUBST (サブストリング)

フリー・フォーム構文	(使用できません。%SUBST 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SUBST (EP)	抽出する長さ	基本ストリング:開始	目的のストリング	_	ER	_

SUBST 演算命令は、演算項目 2 に指定された位置から始まり、演算項目 1 に指定された長さのサブストリングを戻します。このサブストリングが結果フィールドに入れられます。演算項目 1 が指定されていない場合には、開始桁からのストリングの長さが使用されます。グラフィックまたは UCS-2 ストリングの場合は、開始桁が 2 バイト単位で測られます。 基本ストリングと目的のストリングのタイプは両方とも同じ (文字、グラフィック、または UCS-2) でなければなりません。

演算項目 1 を指定する場合には、抽出されるストリングの長さが入っていなければなりません。これは小数点以下の桁数なしの数字でなければならず、フィールド名、配列エレメント、テーブル名、リテラル、または名前付き固定情報を入れることができます。長さが指定されていない場合には、基本ストリングの残り (開始位置から始まっている) が戻されます。

演算項目 2 には、基本ストリング、あるいは基本ストリングとそれにコロンと開始位置が続いているものが入っていなければなりません。基本ストリングには、フィールド名、配列エレメント、名前付き固定情報、データ構造名、テーブル名、またはリテラルが入っていなければなりません。開始桁は小数点以下の桁数なしの数字でなければならず、フィールド名、配列エレメント、テーブル名、リテラル、または名前付き固定情報を入れることができます。開始桁が指定されていない場合には、SUBST は基本ストリングの 1 桁目から開始されます。グラフィックまたは UCS-2 ストリングの場合は、開始桁が 2 バイト単位で測られます。

注:

- 抽出されるサブストリングの開始桁および長さは、正整数でなければなりません。
- 開始桁は長さより大としてはいけません。
- 長さは、基本ストリングの開始位置からの長さより大としてはいけません。

結果フィールドは、文字、グラフィック、または UCS-2 でなければならず、フィールド名、配列エレメント、データ構造、またはテーブル名を入れることができます。結果は左寄せされます。結果フィールドの長さは、少なくとも演算項目 1 に指定されたフィールドの大きさになっている必要があります。サブストリングが結果フィールドに指定されたフィールドより長い場合には、サブストリングは右から切り捨てられます。結果フィールドが可変長である場合にも、その長さは変わりません。命令拡張として P が指定されている場合には、演算命令後に、結果フィールドは右からブランクで埋め込まれます。

注:

- 表意定数は、演算項目 1、演算項目 2、または結果フィールドで使用することはできません。
- 演算項目 1 と結果フィールドのオーバーラップが許されます。
- 演算項目 2 と結果フィールドのオーバーラップが許されます。

SUBST 例外 (プログラム状況コード 100) を処理するために、命令コード拡張 'E' またはエラー標識 ERのいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* SUBST 演算命令は、演算項目 2 の 3 桁目から始まっている長さ 2
C* のサブストリングを抽出します。値 'CD' が結果フィールド TARGET
C* に入れられます。エラーが起こっていないので、標識 90 はオンに
C* セットされません。
C
C                   Z-ADD     3             T                2 0
C                   MOVEL     'ABCDEF'      String          10
C     2             SUBST     String:T      Target                90
C*
C* この SUBST 演算命令では、長さはストリングの長さ - 開始桁 + 1
C* より大です。結局、標識 90 はオンにセットされて、結果のフィー
C* ルドは変更されません。
C
C                   MOVE      'ABCDEF'      String           6
C                   Z-ADD     4             T                1 0
C     5             SUBST     String:T      Result                90
C                   C* この SUBST 演算命令では、基本ストリングの 5 桁目から始まって
C* いる 3 文字がサブストリング化されます。P が指定されていないの
C* で、TARGET の最初の 3 文字のみが変更されます。
C* TARGET には '123XXXXX' が入ります。
C
C                   Z-ADD     3             Length           2 0
C                   Z-ADD     5             T                2 0
C                   MOVE      'TEST123'     String           8
C                   MOVE      *ALL'X'       Target
C     Length        SUBST     String:T      Target           8
```

図 306. SUBST 演算命令

```
C*
C* この例は、P が指定されていることを除き、直前の例と同じで、
C* 結果はブランクで埋め込まれます。
C* TARGET は '123bbbbb' に等しくなります。
C
C                   Z-ADD     3             Length            2 0
C                   Z-ADD5                  T                 2 0
C                   MOVE      'TEST123'     String            8
C                   MOVE      *ALL'X'       Target
C     Length        SUBST(P)  String:T      Target            8
C
C
C*
C* 次の例では、CITY にはストリング 'Toronto, Ontario' が入ります。
C* SCAN 演算命令は、この例では 9 桁目の分離ブランクを見付けるために
C* 使用されます。SUBST は、ストリング 'Ontario' フィールド
C* 桁目から始まり、ストリングの長さだけ継続しているストリングを
C* フィールド TCNTRE に入れます。
C* TCNTRE には 'Ontario' が入ります。
C     ' '           SCAN      City          C
C                   ADD       1             C
C                   SUBST     City:C        TCntre
C*
C* 演算命令の前 STRING='bbbJohnbbbbbb'。
C* RESULT は、'ABCDEFGHIJ' が入る 10 文字のフィールドです。
C* CHECK 演算命令は、最初の非ブランク文字を検索して、このような
C* 文字が存在していると、標識 10 をオンにセットします。*IN10 が
C* オンの場合には、SUBST 演算命令は STRING の最初の非ブランク文
C* 字から始めて STRING の終わりまでサブストリング化します。埋め込
C* みは、結果フィールドの以前の内容が何も残らないようにするた
C* めに使用されます。STRING に値 '   HELLO ' が入っていると、RESULT
C* には値 'HELLO     ' が SUBST(P) 演算命令後に入ります。
C* 演算命令の後 RESULT='Johnbbbbbb'。
C
C     ' '           CHECK     STRING        ST                      10
C   10              SUBST(P)  STRING:ST     RESULT
```

図 307. 命令拡張 P を使用する SUBST 演算命令

TAG (タグ)

フリー・フォーム構文	(使用できません。LEAVE、ITER、RETURN などの他の命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TAG	ラベル					

宣言 TAG 演算命令は、GOTO または CABXX 演算命令の宛先を識別するラベルに名前を付けます。これは、演算のどこでも指定することができます。

メイン・プロシージャーでサブルーチン内で GOTO は、同一サブルーチン内の TAG に対して発行することができます。サブプロシージャーでサブルーチン内で GOTO は、同一サブルーチン内、あるいはサブプロシージャーの本体内の TAG に対して発行することができます。

演算項目 1 には、GOTO または CABxx 演算命令の宛先の名前が入っていなければなりません。この名前は、GOTO 演算命令の演算項目 2 または CABxx 演算命令の結果フィールドに指定されている固有の記号名でなければなりません。この名前は、複数の GOTO または CABxx 演算命令の共通点として使用することができます。

条件付け標識項目 (9 - 11 桁目) は使用できません。

VRPG アプリケーションの別のパーツから TAG への分岐は、永久ループになる場合があります。

TAG 演算命令の例については、 568 ページの図 233を参照してください。

TEST (日付、時刻、タイム・スタンプのテスト)

フリー・フォーム構文	TEST{(EDTZ)} {*dtz-format*} *field-name*

コード	演算項目 1 *(dtz-format)*	演算項目 2	結果フィールド *(field-name)*	標識		
TEST (E)			日付 / 時刻または タイム・スタンプ・フィールド	_	ER	_
TEST (D E)	日付形式		文字または数字フィールド	_	ER	_
TEST (E T)	時刻形式		文字または数字フィールド	_	ER	_
TEST (E Z)	タイム・スタンプ形式		文字または数字フィールド	_	ER	_

TEST 演算命令によって、日付、時刻、またはタイム・スタンプ・フィールドを使用する前に、それらの妥当性をテストすることができます。

field-name オペランドが日付、時刻、またはタイム・スタンプである場合には、*dtz-format* オペランドは指定できません。使用できる形式については、 125 ページの『日付データ』、 144 ページの『時刻データ』、および 145 ページの『タイム・スタンプ・データ』を参照してください。

field-name が日付、時刻、またはタイム・スタンプとして宣言されているフィールドである場合には、*dtz-format* オペランドは指定できず、命令コード拡張の 'D'、'T'、および 'Z' は使用できません。

field-name が文字または数字として宣言されているフィールドである場合には、命令コード拡張の 'D' 、'T'、または 'Z' の 1 つを指定しなければなりません。

注: *field-name* が区切り文字なしの文字フィールドである場合には、*dtz-format* オペランドは後にゼロが続く日付、時刻、またはタイム・スタンプを指定しなければなりません。

- 命令コード拡張に 'D' (日付のテスト) が含まれている場合には、次のとおりです。
  - *dtz-format* は任意指定であり、日付形式のいずれかとすることができます。125 ページの『日付データ』 を参照してください。
  - *dtz-format* が指定されていない場合には、DATFMT キーワードで制御仕様に指定された形式と見なされます。このキーワードが指定されていない場合には、*ISO が想定されます。

- 命令コード拡張に 'T' (時刻のテスト) が含まれている場合には、次のとおりです。
  - *dtz-format* は任意指定であり、有効な時刻形式のいずれかを入れることができます。 144 ページの『時刻データ』 を参照してください。
  - *dtz-format* が指定されていない場合には、TIMFMT キーワードで制御仕様に指定された形式がデフォルトとなります。このキーワードが指定されていない場合には、*ISO が想定されます。

    注: *USA 日付形式は、命令コード拡張 (T) と一緒には使用できません。*USA 日付形式には、数字結果フィールドを使用するときに数字に変換できない、AM/PM の制限があります。
- 命令コード拡張に 'Z' (タイム・スタンプのテスト) が含まれている場合には、*dtz-format* は任意指定で、*ISO または *ISO0 を入れることができます。 145 ページの『タイム・スタンプ・データ』を参照してください。

数字フィールドは、日付、時刻、またはタイム・スタンプ値の有効数字部分のためにテストされます。文字フィールドは、有効数字と区切り文字の利用法のためにテストされます。

テスト操作のためには、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。*field-name* オペランドの内容が無効の場合には、プログラム状況コード 112 のシグナルが送られます。その後で、指定されたエラー処理方式に応じて、エラー標識がオンにセットされるか、あるいは %ERROR 組み込み関数が '1' を戻すように設定されます。エラー処理の詳細については、 52 ページの『プログラム例外およびエラー』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D*
D Datefield       S               D   DATFMT(*JIS)
D Num_Date        S              6P 0 INZ(910921)
D Char_Time       S              8    INZ('13:05 PM')
D Char_Date       S              6    INZ('041596')
D Char_Tstmp      S             20    INZ('19960723140856834000')
D Char_Date2      S              9A   INZ('402/10/66')
D Char_Date3      S              8A   INZ('2120/115')
D*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  文字フィールドが有効な区切り文字なしの *ISO タイム・スタンプ・
C*  フィールドなので、標識 18 はオンにセットされません。
C*
C     *ISO0         TEST (Z)                Char_Tstmp             18
C*
C*  文字フィールドが有効な区切り文字なしの *MDY 日付なので、標識
C*  19 はオンにセットされません。
C*
C     *MDY0         TEST (D)                Char_Date              19
C*
C*  Num_Date が *DMY でないので、%ERROR は '1' を戻します。
C*
C     *DMY          TEST (DE)               Num_Date
C*
C*  結果 D データ・タイプ・フィールドなので演算項目 1 なし。
C*  フィールドには有効な日付が入っているので、%ERROR は '0'
C*  を戻します。
C*
C                   TEST (E)                Datefield
C*
C* 次のテストでは、%ERROR は '1' を戻します。
C* 時刻フィールドに有効な USA 時刻が入っていないためです。
C*
C     *USA          TEST (ET)               Char_Time
C*
C*  次のテストでは、文字フィールドは有効な *CMDY ですが、区
C*  切り文字があるので、標識 20 はオンにセットされます。
C*
C     *CMDY0        TEST (D)                char_date2             20
C*
C*  次のテストでは、文字フィールドが有効な *LONGJUL 日付なので、
C*  %ERROR は '0' を戻します。
C*
C     *LONGJUL      TEST (DE)               char_date3
```

図 308. TEST (D/T/Z) の例

TESTB (ビットのテスト)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TESTB		ビット番号	文字フィールド	OF	ON	EQ

TESTB 演算命令は、演算項目 2 に指定されたビットと、結果フィールドとして指定されたフィールド中の対応するビットを比較します。結果フィールドは、1 桁の文字フィールドでなければなりません。 71 - 76 桁目の演算結果標識は、結果フィールド・ビットの状況を反映します。演算項目 2 は、常に結果フィールドのビットのソースです。

演算項目 2 には、次を入れることができます。

- ビット番号 0 - 7: 1 - 8 ビットは演算命令のたびにテストすることができます。テストするビットは、番号 0 - 7 によって識別されます。(0 が左端のビットです。) ビット番号はアポストロフィで囲まれていなければなりません。たとえば、ビット 0、2、および 5 をテストするには、演算項目 2 に '025' と入力してください。
- フィールド名: 1 桁の文字フィールド、テーブル名、または配列エレメントを演算項目 2 に指定することができます。フィールド、テーブル名、または配列エレメントでオンになっているビットが、結果フィールドで対応しているビットと比較されます。オフになっているビットは考慮に入れられません。結果フィールドに指定されるフィールドは、配列の各エレメントが 1 桁の文字フィールドであれば、配列エレメントとすることができます。
- 16 進数リテラルまたは名前付き固定情報: 1 バイトの 16 進数リテラルまたは 16 進数名前付き固定情報を指定することができます。演算項目 2 でオンになっているビットが、結果フィールドで対応しているビットと比較されます。オフになっているビットは考慮に入れられません。

694 ページの図 309 には、TESTB 演算命令の使用について説明されています。

71 - 76 桁目に割り当てられた標識は、結果フィールド・ビットの状況を反映します。少なくとも 1つの標識が割り当てられていなければならず、1 つの演算命令に多くとも 3 つを割り当てることができます。 TESTB 演算命令の場合は、演算結果標識は、次のようにオンにセットされます。

- 71 - 72 桁目: これらの桁の標識は、演算項目 2 に指定されたビット番号または演算項目 2 フィールドでオンになっている各ビットが結果フィールドでオフになっているとオンにセットされます。指定されたビットのすべてがオフと等しくなります。
- 73 - 74 桁目: これらの桁の標識は、演算項目 2 に指定されたビット番号または演算項目 2 フィールドでオンになっている各ビットが混合状況 (あるビットはオンであるビットはオフ) になっているとオンにセットされます。指定されたビットの少なくとも 1 つはオンです。

注: 1 ビットしかテストしない場合には、これらの桁はブランクでなければなりません。フィールド名が演算項目 2 に指定されていて、オンになっているビットが 1 つしかない場合には、73 - 74 桁目の標識はオンにセットされません。

- 75 - 76 桁目: これらの桁の標識は、演算項目 2 に指定されたビット番号または演算項目 2 フィールドでオンになっている各ビットが結果フィールドでオンになっているとオンにセットされます。指定されたビットのすべてがオフと等しくなります。

  注: 演算項目 2 のフィールドにオンになっているビットがない場合には、標識はオンにセットされません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  フィールド・ビット設定は FieldF = 00000001 および FieldG = 11110001
C*  です
C*
C*  FieldF のビット 3 がオフ (0) なので標識 16 はオンにセットされます。
C*  標識 17 はオフにセットされます。
C                   TESTB     '3'           FieldF               16  17
C*
C*  FieldF のビット 3 および 6 がオフ (0) なので標識 16 はオンに
C*  セットされます。標識 17 および 18 はオフにセットされます。
C                   TESTB     '36'          FieldF               161718
C*
C*  FieldF のビット 3 がオフ (0) でビット 7 がオン (1) なので、
C*  標識 17 はオンにセットされます。標識 16 および 18 はオフに
C*  セットされます。
C                   TESTB     '37'          FieldF               161718
C*
C*  FieldF のビット 7 がオン (1) なので標識 17 はオンにセットされます。
C*  標識 16 はオフにセットされます。
C                   TESTB     '7'            FieldF              16  17
C*
C*  ビット 0、1、2、および 3 がオフ (0) でビット 7 がオン (1) な
C*  ので、標識 17 はオンにセットされます。標識 16 および 18 はオ
C*  フにセットされます。
C                   TESTB     FieldG         FieldF              161718
C*
C*  演算項目 2 で 16 進数リテラル X'88' (10001000) が使用されます。
C*  少なくとも 1 ビット (ビット 0) がオンなので標識 17 はオンに
C*  セットされます。標識 16 および 18 はオフにセットされます。
C                   TESTB     X'88'         FieldG               161718
```

図 309. TESTB 演算命令

TESTN (数字のテスト)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TESTN			文字フィールド	NU	BN	BL

TESTN 演算命令は、文字結果フィールドでゾーン 10 進数の数字とブランクの存在をテストします。

結果フィールドは、文字フィールドでなければなりません。数字と見なすには、フィールド中の右端の文字を除く各文字には、16 進数 3 のゾーンと数字 (0 - 9) が入っていなければなりません。右端の文字に 16 進数 0 -9、または A - F のゾーン、および数字 (0 - 9) が入っていると、それは数字です。テストの結果として、演算結果標識は次のようにオンにセットされます。

- 71 および 72 桁目: 結果フィールドには数字が入ります。
- 73 および 74 桁目: 結果フィールドには数字と少なくとも 1 つの先行ブランクの両方が入ります。たとえば、値 b123 または bb123 によってこの標識がオンにセットされます。しかし、組み込みブランクが原因で、値 b1b23 は有効な数字フィールドではないので、この値でこの標識はオンにセットされません。

  注: 文字フィールドには少なくとも 1 つの数字と 1 つの先行ブランクが入っていなければならないので、長さ 1 のフィールドをテストする場合には、標識を指定することができません。
- 75 および 76 桁目: 結果フィールドにはすべてブランクが入ります。

複数の条件のために同一標識を使用することができます。条件のいずれかが存在している場合には、標識がオンにセットされます。

TESTN 演算命令は、フィールドを使用すると望ましくない結果または例外の原因になる演算命令で使用する前に、そのフィールドを妥当性検査するために使用することができます。

TESTZ (ゾーンのテスト)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TESTZ			文字フィールド	+	−	

TESTZ 演算命令は、結果フィールドの左端の文字のゾーンをテストします。結果フィールドは、文字フィールドでなければなりません。

演算結果標識は、テストの結果に従ってオンにセットされます。71 - 74 桁目には、演算結果標識が少なくとも 1 つは指定されていなければなりません。ゾーンが正の任意の文字 (16 進数 0、1、2、3、8、9、A、B)で 71 および 72 桁目の標識がオンにセットされます。ゾーンが負の任意の文字 (16 進数 4、5、6、7、C、D、E、F) で 73 および 74 桁目の標識がオンにセットされます。

注: 正 / 負のゾーンは、2 番目のビットの値によって異なります。値 3 および 7 が符号に優先使用される値ですが、リストされているその他の値を使用することができます。

TIME (時刻)

フリー・フォーム構文	(使用できません。%DATE、%TIME、または %TIMESTAMP 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TIME	別名		ターゲット・フィールド			

TIME 演算命令は、プログラム処理中の任意の時点で、システム時刻およびシステム日付 (指定されている場合) にアクセスします。システム時刻およびシステム日付は、iSeries サーバーまたはワークステーションからもリトリーブできます。システム時刻は、24 時間クロックが基になっています。

演算項目 1 を指定する場合には、サーバーの別名であるリテラルが入っていなければなりません。演算項目 1 が指定されていない場合には、ワークステーションから時刻が検索されます。

結果フィールドには 6 桁、12 桁、または 14 桁の数字フィールド (小数点以下の桁数なし) を指定しなければなりません。時刻または時刻とシステム日付が結果フィールドに書き込まれます。

結果フィールド	戻される値	形式
6 桁の数字	時刻	hhmmss
12 桁の数字	時刻および日付	hhmmssDDDDDD
14 桁の数字	時刻および日付	hhmmssDDDDDDDD
時刻	時刻	結果の形式
Date	Date	結果の形式
タイム・スタンプ	タイム・スタンプ	*ISO
注: 結果フィールドが数字フィールドで、システム日付が含まれている場合には、それが結果フィールドの 7 ～ 12 桁目に入れられます。日付形式は *YMD です。		

時刻にしかアクセスしないためには、結果フィールドを 6 桁の数字フィールドとして指定してください。時刻とシステム日付の両方にアクセスするには、結果フィールドを 12 桁 (2 桁の年部分)または 14桁 (4 桁の年部分) の数字フィールドを指定してください。時刻は、常に、結果フィールドの最初の 6 桁に次の形式で入れられます。

hhmmss (hh= 時、mm= 分、および ss= 秒)

日付形式は、日付形式ジョブ属性 DATFMT によって異なり、mmddyy、ddmmyy、yymmdd、またはユリウスとすることができます。2 桁の年部分のユリウス形式には、7 - 8 桁目に年が、9 - 11 桁目に日 (右寄せして未使用の高位桁にゼロが入れられた 1 - 366) が、12 桁目に 0 が入ります。4 桁の年部分の場合は、7 - 10 桁目に年が、11 - 13 桁目に日 (右寄せして未使用の高位桁にゼロが入れられた 1 - 366) が、14 桁目に 0 が入ります。

結果フィールドがタイム・スタンプ・フィールドである場合には、マイクロ秒部分の最後の 3 桁は常に 000です。

特殊フィールド UDATE および *DATE にはジョブ日付が入ります。これらの値は、深夜の通過時、あるいはプログラムの実行中にジョブ日付の変更時には更新されません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  TIME 演算命令が (6 桁の数字フィールドを指定して) 処理される
C*  と、現在時刻が (hhmmss という形式で) 結果フィールド CLOCK
C*  に入れられます。TIME 演算命令は 24 時間クロック (たとえば、
C*  132710) が基になっています。(12 時間時刻システムでは、132710
C*  は 1:27:10 p.m. です。) この場合、CLOCK は出力仕
C*
C                   TIME                    Clock             6 0
C*  TIME 演算命令が (12 桁の数字フィールドを指定して) 処理される
C*  と、現在時刻と日が結果フィールド TIMSTP に入れられます。
C*  最初の 6 桁は時刻で、最後の 6 桁は日付です。たとえば、
C*  093315121579 は 1979 年 12 月 15 日 9:33:15 a.m.
C*  です。この場合、TIMSTP は出力仕様に指定することができます。
C*
C                   TIME                    TimStp           12 0
C                   MOVEL     TimStp        Time              6 0
C                   MOVE      TimStp        SysDat            6 0
C*  この例は、上記の 12 桁の例と重複しますが、14 桁のフィールド
C*  を使用します。最初の 6 桁は時刻で、最後の 8 桁は日付です。
C*  たとえば、13120001101992 は 1992 年 1 月 10 日 1:12:00 p.m.
C*  です。
C*  その後で、TIMSTP は出力仕様に指定することができます。
C                   TIME                    TimStp           14 0
C                   MOVEL     TimStp        Time              6 0
C                   MOVE      TimStp        SysDat            8 0
```

図 310. TIME 演算命令

UNLOCK (データ域のロック解除またはレコードの解放)

フリー・フォーム構文	UNLOCK{(E)} *name*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
UNLOCK (E)		name (ファイルまたはデータ域)		_	ER	_

UNLOCK 演算命令は、データ域をロック解除してレコード・ロックを解放します。

UNLOCK 例外 (プログラム状況コード 401 - 421、431、および 432) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

71、72、75、および 76 桁目はブランクでなければなりません。

データ域のロック解除

name オペランドは、ロック解除されるデータ域の名前または予約語の *DTAARA でなければなりません。

*DTAARA が指定されていると、プログラム中でロックされているすべてのデータ域がロック解除されます。

データ域は、*DTAARA DEFINE ステートメントの結果フィールドに、あるいは定義仕様の DTAARA キーワードですでに指定されていなければなりません。すでにロック解除されているデータ域に UNLOCK 演算命令が指定されると、エラーは起こりません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  TOTAMT、TOTGRS、および TOTNET はシステム内でデータ域として定
C*  義されます。IN 演算命令はプログラム内に定義されているすべて
C*  のデータ域を検索します。プログラムは演算を処理してから、デー
C*  タ域をアンロックします。その後で、データ域はプログラムで使用
C*  することができます。
C*
C     *LOCK         IN        *DTAARA
C                   :
C                   :
C                   UNLOCK    *DTAARA
C     *DTAARA       DEFINE                TOTAMT            8 2
C     *DTAARA       DEFINE                TOTGRS           10 2
C     *DTAARA       DEFINE                TOTNET           10 2
```

図 311. データ域のロック解除演算命令

レコード・ロックの解放

UNLOCK 演算命令は、更新ディスク・ファイルで最後にロックされたレコードをロック解除します。

name オペランドは、UPDATE ディスク・ファイルの名前でなければなりません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FFilename++IT.A.FRlen+......A.Device+.Keywords++++++++++++++++++++++++++++++
F*
FUPDATA    UF   E             DISK    REMOTE
F*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  ファイル UPDATA にはレコード様式 VENDOR が入っているものと
C*  します。
C*  レコードは UPDATA から読み取られます。ファイルが更新ファイル
C*  なので、レコードはロックされます。*IN50 は、UPDATE を実行す
C*  るかどうかを制御するために、プログラム内の他のどこかの場所で
C*  設定されています。
C*  そうでない場合には、レコードは UNLOCK 演算命令を使用してロック
C*  解除されます。
C*  UNLOCK 演算命令の演算項目 2 がファイル名であることに注意して
C*  ください。
C*  レコード様式 VENDOR ではなく UPDATA です。
C*
C                   READ      VENDOR                                  12
C                   :
C     *IN50         IFEQ      *ON
C                   UPDATE    VENDOR
C                   ELSE
C                   UNLOCK    UPDATA                                99
C                   ENDIF
```

図 312. レコードのロック解除演算命令

UPDATE (既存レコードの変更)

フリー・フォーム構文	UPDATE{(E)} *name* {*data-structure*}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
UPDATE (E)		name (ファイル、レコード様式、またはサブファイル)	データ構造	_	ER	_

UPDATE 演算命令は、更新ディスク・ファイルまたはサブファイルから処理のために検索されて最後にロックされたレコードを変更します。その他の演算命令は、レコードを検索してロックした入力演算命令と UPDATE 演算命令の間でファイルに対して実行されます。

READ、READC、READE、READP、READPE、および CHAIN などの演算命令は、レコードを検索してロックします。これらの入力演算命令が成功しない場合には、レコードはロックされず、UPDATE は発行することができません。レコードは、ロック要求を指定するために、デフォルトのブランク命令拡張で再び読み取らなければなりません。

UPDATE 演算命令の成功後に、次の順次入力演算命令で、更新済みのレコードの後のレコードが検索されます。

同一のファイルまたはレコードに対する連続する UPDATE 演算命令は無効です。更新するレコードを位置決めしてロックするためには、間に成功する読み取り演算命令が発行されていなければなりません。

name オペランドは、更新されるファイル、サブファイル、またはレコード様式の名前でなければなりません。ファイル名を指定する場合には、それはプログラム記述ファイルでなければなりません。レコード様式名を指定する場合には、それは外部記述ファイルでなければなりません。レコード様式名は、ファイルから最後に読み取られたレコードの名前でなければなりません。そうでない場合には、エラーが起こります。

name がファイル名 (サブファイルでありません) である場合には、*data-structure* オペランドにデータ構造名を指定しなければなりません。更新済みレコードは、直接データ構造からファイルに書き込まれます。*name* がレコード様式名である場合には、*data-structure* オペランドを指定することはできません。

UPDATE 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

注: レコード中のすべてではなく一部のフィールドを更新する場合には、出力仕様を使用して UPDATE 演算命令は使用しないでください。

ヌル可能フィールドがあるレコードの読み取りについては、145 ページの『データベースのヌル値サポート』 を参照してください。

WHEN (真の場合に選択)

フリー・フォーム構文	WHEN{(MR)} *indicator-expression*

コード	演算項目 1	拡張演算項目 2				
WHEN (M/R)		indicator-expression				

WHEN 命令コードは、SELECT 演算命令で行の処理を制御する WHENxx 命令コードに似ています。これは、条件が *indicator-expression* オペランドの論理式によって指定される点が違っています。WHEN 演算命令によって制御される演算命令は、*indicator-expression* の式が「真」になると実行されます。式の詳細については、383 ページの『第 24 章 式』を参照してください。命令拡張 M および R の使用法については、392 ページの『数値演算の精度の規則』を参照してください。

361 ページの『比較命令』には、比較演算命令を指定するための一般規則が説明されています。

```
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++..
C*
C                   SELECT
C                   WHEN    *INKA
C                   :
C                   :
C                   :
C                   WHEN    NOT(*IN01) AND (DAY = 'FRIDAY')
C                   :
C                   :
C                   :
C                   WHEN    %SUBST(A:(X+4):3) = 'ABC'
C                   :
C                   :
C                   :
C                   OTHER
C                   :
C                   :
C                   :
C                   ENDSL
```

図 313. WHEN 演算命令

WHENxx (真の場合に選択)

フリー・フォーム構文	(使用できません。WHEN 命令コードを使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
WHENxx	被比較数	被比較数				

選択グループの WHENxx 演算命令は、SELECT 演算命令の処理後に制御を渡すかどうかを判別します。

演算項目 1 と演算項目 2 が xx によって指定された関係がある場合には、WHENxx 条件付き演算命令は真です。この条件が真の場合には、WHENxx に続く演算命令が次の WHENxx、OTHER、ENDSL、または END 演算命令まで処理されます。

WHENxx 演算命令の実行時には、次の点に留意してください。
- 演算命令グループの処理後に、制御は ENDSL 演算命令の後のステートメントに渡されます。
- 複雑な WHENxx 条件は、ANDxx および ORxx を使用してコーディングすることができます。結合された WHENxx、ANDxx、および ORxx 演算命令によって指定された条件が真になると、演算が処理されます。
- WHENxx グループはヌルでもかまいません。

xx 値については、361 ページの『比較命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次の例は、ネストされた SELECT グループを示しています。従業員
C* タイプは、臨時の 'C'、退職の 'T'、正規の 'R'、および学生の 'S'
C* のいずれかとすることができます。従業員タイプ (EmpTyp) に応じ
C* て、年当たりの休日数 (Days) が異なります。
C*
C                   SELECT
C     EmpTyp        WHENEQ    'C'
C     EmpTyp        OREQ      'T'
C                   Z-ADD     0             Days
C     EmpTyp        WHENEQ    'R'
C*
C* 従業員タイプが 'R' の場合は、休日数は勤続年数によっても異なり
C* ます。基本日数は 14 です。
C* 2 年未満の場合は、エクストラ日数は加算されません。2 ～ 5 年の
C* 場合には、エクストラ日数 5 が加算されます。6 ～ 10 年の場合に
C* は、エクストラ日数 10 が加算されて、10 年を超えると、は、エク
C*
C                   Z-ADD     14            Days
C*
C* ネストされた選択グループ
C                   SELECT
C     Years         WHENLT    2
C     Years         WHENLE    5
C                   ADD       5             Days
C     Years         WHENLE    10
C                   ADD       10            Days
C                   OTHER
C                   ADD       20            Days
C                   ENDSL
C* ネストされた選択グループの終わり
C*
C     EmpTyp        WHENEQ    'S'
C                   Z-ADD     5             Days
C                   ENDSL
```

図 314. WHENxx 演算命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C* 複雑な WHENxx 式が使用される SELECT グループの例。レコードおよ
C* びアクション・コードはユーザーが入力しているものとします。
C* 次のいずれかを選択してください。
C*   * F3 が押されたら、サブルーチン QUIT を実行します。
C*   * アクション・コード (Acode) A (追加) が入力されて、レコード
C*     が存在していない (*IN50=1) 場合は、レコードを書き込みます。
C*   * アクション・コード A が入力されて、レコードが存在していて、
C*     そのレコードのアクティブ・レコード・コードが D (削除済み)
C*     だと、そのレコードをアクティブ・レコード・コード =A で更新
C*     します。アクション・コード D が入力されて、レコードが存在
C*     していて、レコード (AcRec) 中のアクション・コードが A の場
C*     合には、そのレコードに削除済みのマークを付けます。
C*   * アクション・コードが C (変更) で、レコードが存在していて、
C*     レコード (AcRec) 中のアクション・コードが A の場合には、そ
C*     のレコードを更新します。
C*   * そうでない場合には、エラー処理を実行します。
C*
C     RSCDE         CHAIN     FILE                                50
C                   SELECT
C     *INKC         WHENEQ    *ON
C                   EXSR      QUIT
C     Acode         WHENEQ    'A'
C     *IN50         ANDEQ     *ON
C                   WRITE     REC
C     Acode         WHENEQ    'A'
C     *IN50         ANDEQ     *OFF
C     AcRec         ANDEQ     'D'
C     Acode         OREQ      'D'
C     *IN50         ANDEQ     *OFF
C     AcRec         ANDEQ     'A'
C                   MOVE      Acode         AcRec
C                   UPDATE    REC
C     Acode         WHENEQ    'C'
C     *IN50         ANDEQ     *OFF
C     AcRec         ANDEQ     'A'
C                   UPDATE    REC
C                   OTHER
C                   EXSR      ERROR
C                   ENDSL
```

図 315. WHENxx 演算命令

WRITE (新しいレコードの作成)

フリー・フォーム構文	WRITE{(E)} *name* {*data-structure*}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
WRITE (E)		name (ファイル、レコード様式、サブファイル、またはウィンドウ)	データ構造	_	ER	EOF

WRITE 演算命令は、データをファイルまたはウィンドウに書き込みます。

ファイルへの書き込み

name オペランドは、ファイルまたはレコード様式の名前でなければなりません。このファイルは、ファイル仕様の 18 桁目の F によって識別されます。これは、全手順ファイルでなければなりません。

name で指定されるファイルがプログラム記述されている場合には、*data-structure* オペランドにデータ構造の名前を指定しなければなりません。レコードは、データ構造からこのファイルに書き込まれます。外部記述ファイルは、ファイル仕様の 22 桁目の E によって識別されます。

name がレコード様式名である場合には、プログラム内のレコード定義のすべてのフィールドの現在の値がレコードを構成するために使用されます。

相対レコード番号を使用するレコードがファイルに書き込まれるときに、RECNO ファイル仕様キーワード (相対レコード番号) に指定されたフィールド名は、書き込まれるレコードの相対レコード番号が入るように更新されていなければなりません。

ローカル・ファイルの場合は、レコードはそのファイルの終わりに書き込まれます。RECNO キーワードは無視されます。

レコードをリモート DISK ファイルに追加するには、ファイル仕様の 20 桁目に A が指定されていなければなりません。ローカル・ファイルの場合は、レコードはそのファイルの終わりに追加されます。詳しくは、240 ページの『20 桁目 (ファイルの追加)』を参照してください。

WRITE 例外 (1000 より大きいファイル状況コード) を処理するために、命令コード拡張 'E' またはエラー標識 ER のいずれかを指定することができますが、両方とも指定することはできません。エラーが起こるのは、オーバーフローが外部記述印刷ファイルに達して、オーバーフロー標識がファイル仕様に指定されていない場合です。エラー処理の詳細については、41 ページの『ファイルの例外 / エラー』を参照してください。

WRITE 演算命令でファイルの終わりが起こった (サブファイルがあふれている) かどうかのシグナルを送るために、 75 - 76 桁目に標識を指定することができます。標識は WRITE 演算命令が実行されるたびにオン (EOF 条件) またはオフにセット

されます。この情報は、EOF 条件が起こると '1' を、そうでない場合には '0' を戻す %EOF 組み込み関数から取得することもできます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  WRITE 演算命令は、データ構造 DS1 中のフィールドをファイル
C*  FILE1 に書き込みます。
C*
C                   WRITE     FILE1         DS1
```

図 316. ファイルの場合の WRITE 演算命令

ウィンドウへの書き込み

name オペランドがウィンドウ名である場合には、WRITE 演算命令はウィンドウ上の静的テキストとフィールド・パーツの属性を設定します。項目パーツの属性は TEXT です。静的テキスト・パーツの属性は LABELです。

data-structure オペランドおよび命令拡張は、指定できません。

ウィンドウに書き込む場合には、使用される対応しているフィールドに保管された値が、入力フィールド・パーツおよび静的テキスト・パーツの属性を設定するために使用されます。WRITE 演算命令後に、フィールド中に保管された値は、画面に表示される値と一致しています。多くの静的テキストおよび入力フィールドがある場合には、複数の SETATR よりむしろ WRITE 演算命令を使用してください。たとえば、ウィンドウ INVENTORY に入力フィールド・パーツ ENT0000B および ENT0000C が含まれている場合には、ウィンドウの WRITE で次と同等の処理が実行されます。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....Comments++++++
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++Comments++++++
C                   EVAL      %setatr('inventory':'ent0000B':'text') = ENT0000B
C                   EVAL      %setatr('inventory':'ent0000C':'text') = ENTd000C
```

図 317. ウィンドウの場合の WRITE 演算命令

サブファイルへの書き込み

name オペランドがサブファイル・パーツ名である場合には、 WRITE 演算命令は新規レコードをサブファイルに追加します。サブファイル・パーツに書き込まれるレコードは、サブファイルの終わりに追加されます。

data-structure オペランドは指定できません。

ヌル可能フィールドがあるレコードの読み取りについては、 145 ページの『データベースのヌル値サポート』 を参照してください。

XFOOT (配列エレメントの合計)

フリー・フォーム構文	(使用できません。%XFOOT 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
XFOOT (H)		配列名	和	+	–	Z

XFOOT は、配列エレメントをまとめて加算して、和を結果フィールドとして指定されたフィールドに入れます。演算項目 2 には配列の名前を入れます。

四捨五入が指定されていると、すべてのエレメントが合計された後、および結果が結果フィールドに移動される前に丸めが起こります。結果フィールドが演算項目 2 に指定された配列のエレメントの 1 つである場合には、XFOOT 演算命令前のエレメントの値が配列の合計を計算するために使用されます。

配列が浮動小数点である場合には、XFOOT は次のように実行されます。配列が降順になっている場合には、エレメントは逆順に損後に加算されます。そうでない場合には、エレメントは配列の最初のエレメントから始めて相互に加算されます。

352 ページの『算術演算』には、算術演算を指定するための一般規則が説明されています。

355 ページの図 114 には、XFOOT 演算命令の例が含まれています。

XLATE (変換)

フリー・フォーム構文	(使用できません。%XLATE 組み込み関数を使用してください)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
XLATE (EP)	始め:終わり	ソース・ストリング : 開始	目的のストリング	_	ER	_

XLATE 演算命令は、ソース・ストリング (演算項目 2) の転送元ストリングと一致する文字が転送先ストリング中の対応している文字に変換されます。XLATE は、ソースの変換を演算項目 2 に指定された位置から開始して、左から右へ 1 文字ずつ続行します。ソース・ストリング中の文字が転送元ストリング中に存在している場合には、転送先ストリング中で対応している文字が結果フィールドに入れられます。開始桁の前のソース・フィールド中の文字は未変更のまま結果フィールドに入れられます。

注:

- 転送元、転送先、ソース、および目的のストリングのタイプはすべて同じ (すべて文字、すべてグラフィック、またはすべて UCS-2 のいずれか) でなければなりません。グラフィック・フィールドの場合に、制御仕様で CCSID(*GRAPH: *IGNORE) が指定されていない限り、それらの CCSID は同じでなければなりません。
- 表意定数は、演算項目 1、演算項目 2、または結果フィールドで使用することができません。演算項目 1 と結果フィールドまたは演算項目 2 と結果フィールドにデータ構造のオーバーラップがあってはなりません。

演算項目 1 には、変換元ストリングと、それに続くコロンと変換先ストリングが入っていなければなりません。変換元ストリングと変換先ストリングには、フィールド名、配列エレメント、名前付き固定情報、データ構造名、リテラル、またはテーブル名を入れることができます。変換元ストリング中の文字が重複している場合には、最初のオカレンス (左端) が使用されます。

演算項目 2 には、ソース・ストリング、またはソース・ストリングとそれに続くコロンと開始桁が入っていなければなりません。演算項目 2 のソース・ストリング部分には、フィールド名、配列エレメント、名前付き固定情報、データ構造名、データ構造サブフィールド、リテラル、またはテーブル名を入れることができます。演算命令でグラフィックまたは UCS-2 データを使用する場合には、開始桁は 2 バイト文字を指しています。演算項目 2 の開始桁は小数点以下の桁数なしの数字でなければならず、名前付き固定情報とすることができます。開始桁が指定されていない場合には、デフォルト値は 1 です。

結果フィールドは、フィールド、配列エレメント、データ構造、またはテーブルとすることができます。結果フィールドの長さは、演算項目 2 に指定されたソース・ストリングの大きさになっている必要があります。結果フィールドがソース・ストリングより大きい場合には、結果は左寄せされます。結果フィールドがソース・ストリングより大きくて、命令拡張 P が指定されている場合には、結果は変換後にブ

ランクで右方が埋め込まれます。結果フィールドがソース・ストリングより短い場合には、結果フィールドには変換後のソースの左端部分が入ります。

注: 結果フィールドがグラフィックで、命令拡張 P が指定されている場合には、グラフィック・ブランクが使用されます。結果フィールドが UCS-2 で P が指定されている場合には、 UCS-2 ブランクが使用されます。

XLATE 例外 (プログラム状況コード 100) を処理するために、命令コード拡張 'E' またはエラー標識 ERのいずれかを指定することができますが、両方とも指定することはできません。エラー処理の詳細については、52 ページの『プログラム例外およびエラー』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* 次では、NUMBER のブランクが '-' に変換されます。
C* RESULT の結果は '999-9999' になります。
C*
C                   MOVE      '999 9999'    Number            8
C     ' ':'-'       XLATE     Number        Result            8
```

図 318. XLATE 演算命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
D*
D* 次の例では、STRING のすべての値が大文字に変換されます。結局、
D* RESULT='RPG DEP'。
D*
D Up              C                   'ABCDEFGHIJKLMNOPQRS-
D                                     'TUVWXYZ'
D Lo              C                   'abcdefghijklmnopqrs-
D                                     'tuvwxyz'
C*
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C
C                   MOVE      'rpg dep'     String            7
C     Lo:Up         XLATE     String        Result               90
C*
C* 次の例では、ストリング中のすべての値が小文字に変換されます。
C* 結局、RESULT='rpg dep'。
C*
C                   MOVE      'RPG DEP'     String            7
C     Up:Lo         XLATE     String        Result               90
```

図 319. 名前付き固定情報を使用する XLATE 演算命令

Z-ADD (ゼロにして加算)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
Z-ADD (H)		加数	和	+	–	Z

演算項目 2 がゼロのフィールドに加算されます。和が結果フィールドに入れられます。演算項目 2 は、数字でなければなりませんが、配列、配列エレメント、フィールド、表意定数、リテラル、名前付き固定情報、サブフィールド、またはテーブル名を含めることができます。

結果フィールドは数字でなければならず、配列、配列エレメント、サブフィールド、またはテーブルを入れることができます。

四捨五入を指定することができます。

352 ページの『算術演算』には、算術演算を指定するための一般規則が説明されています。

355 ページの図 114 には、Z-ADD 演算命令の例が示されています。

Z-SUB (ゼロにして減算)

フリー・フォーム構文	(使用できません)

コード	演算項目 1	演算項目 2	結果 フィールド	標識		
Z-SUB (H)		減数	差	+	–	Z

演算項目 2 がゼロのフィールドから減算されます。演算項目 2 の負数である差が結果フィールドに入れられます。

演算項目 2 は、数字でなければなりませんが、配列、配列エレメント、フィールド、表意定数、リテラル、名前付き固定情報、サブフィールド、またはテーブル名を含めることができます。

結果フィールドは数字でなければならず、配列、配列エレメント、サブフィールド、またはテーブルを入れることができます。

四捨五入を指定することができます。

352 ページの『算術演算』には、算術演算を指定するための一般規則が説明されています。

355 ページの図 114 には、Z-SUB 演算命令の例が示されています。

第 5 部 付録

付録 A. 制約事項

機能	制約事項
コンパイル時間用の配列 / テーブルの入力レコード長	最大長は 100
文字フィールドの長さ	最大長は 65535 バイト
グラフィックまたは UCS-2 フィールドの長さ	最大長は 32766 バイト
データ構造 (名前付き) の長さ	最大 65535
データ構造 (名前なし) の長さ	最大 9999999
データ構造のオカレンス (数)	データ構造ごとに最大 32767
編集語	最大長は、リテラルで 24 または名前付き固定情報で 115
配置 / テーブルのエレメント (定義仕様上の DIM キーワード)	配置 / テーブルごとに最大 32767
構造化されたグループでのネスティングのレベル	最大 100
名前付き固定情報またはリテラル	文字、16 進数、グラフィック、または UCS-2 リテラルで最大長 1024 バイト、および数値リテラルで小数点以下 30 桁をもつ最大長 30 桁。
リモート OS/400 プログラムに渡されるパラメーター (CALL)	最大 25
リモート OS/400 プログラムに渡されるすべてのパラメーター (CALL) の合計サイズ	最大 32767 バイト
呼び出し先の機能に渡されるパラメーター (CALLB)	最大 399
ローカル EXE、CMD、COMS、および BAT に渡されるパラメーター (CALLP)	最大 20 または合計 1024 バイト。
印刷装置ファイル	プログラムごとに最大 8
1 ページ当りの印刷行数	最小 2、最大 255
プログラム状況データ構造	プログラムにつき 1 つだけ
レコード長	最大長は 99999.[1]
注: [1]装置レコード・サイズの制約事項は、この値をオーバーライドします。	

付録 B. 照合順序

以下のテーブルには、EBCDIC と ASCII 照合順序の両方がリストされています。

EBCDIC 照合順序

表 62. EBCDIC 照合順序

序数	記号	意味	10 進数表記	16 進数表記
65	b	スペース	64	40
. . .				
75	¢	セント記号	74	4A
76	.	ピリオド、小数点	75	4B
77	<	不等号 (より小)	76	4C
78	(	左括弧	77	4D
79	+	正符号	78	4E
80	v	縦線、論理 OR	79	4F
81	&	アンパーサンド	80	50
. . .				
91	!	感嘆符	90	5A
92	$	ドル記号	91	5B
93	*	アスタリスク	92	5C
94	)	右括弧	93	5D
95	;	セミコロン	94	5E
96	¬	論理否定	95	5F
97	–	負記号、ハイフン	96	60
98	/	スラッシュ	97	61
. . .				
107	¦	分割垂直バー	106	6A
108	,	コンマ	107	6B
109	%	% 記号	108	6C
110	_	下線	109	6D
111	>	不等号 (より大)	110	6E
112	?	疑問符	111	6F
. . .				
122	`	抑音符号	121	79
123	:	コロン	122	7A
124	#	番号記号、ポンド記号	123	7B
125	@	アットマーク	124	7C
126	’	アポストロフィ、プライム符号	125	7D

表 62. EBCDIC 照合順序 (続き)

序数	記号	意味	10 進数表記	16 進数表記
127	=	等号	126	7E
128	"	引用符	127	7F
. . .				
130	a		129	81
131	b		130	82
132	c		131	83
133	d		132	84
134	e		133	85
135	f		134	86
136	g		135	87
137	h		136	88
138	i		137	89
. . .				
146	j		145	91
147	k		146	92
148	l		147	93
149	m		148	94
150	n		149	95
151	o		150	96
152	p		151	97
153	q		152	98
154	r		153	99
. . .				
162	~	ティルド	161	A1
163	s		162	A2
164	t		163	A3
165	u		164	A4
166	v		165	A5
167	w		166	A6
168	x		167	A7
169	y		168	A8
170	z		169	A9
. . .				
193	{	左中括弧	192	C0
194	A		193	C1
195	B		194	C2
196	C		195	C3
197	D		196	C4
198	E		197	C5
199	F		198	C6

表 62. EBCDIC 照合順序 (続き)

序数	記号	意味	10 進数表記	16 進数表記
200	G		199	C7
201	H		200	C8
202	I		201	C9
. . .				
209	}	右中括弧	208	D0
210	J		209	D1
211	K		210	D2
212	L		211	D3
213	M		212	D4
214	N		213	D5
215	O		214	D6
216	P		215	D7
217	Q		216	D8
218	R		217	D9
. . .				
225	\	左スラッシュ	224	E0
. . .				
227	S		226	E2
228	T		227	E3
229	U		228	E4
230	V		229	E5
231	W		230	E6
232	X		231	E7
233	Y		232	E8
234	Z		233	E9
. . .				
241	0		240	F0
242	1		241	F1
243	2		242	F2
244	3		243	F3
245	4		244	F4
246	5		245	F5
247	6		246	F6
248	7		247	F7
249	8		248	F8
250	9		249	F9

ASCII 照合順序

表 63. ASCII 照合順序

序数	記号	意味	10 進数表記	16 進数表記
1		NULL	0	0
. . .				
33	b	スペース	32	20
34	!	感嘆符	33	21
35	"	引用符	34	22
36	#	番号記号	35	23
37	$	ドル記号	36	24
38	%	% 記号	37	25
39	&	アンパーサンド	38	26
40	’	アポストロフィ、プライム符号	39	27
41	(	左括弧	40	28
42	)	右小括弧	41	29
43	*	アスタリスク	42	2A
44	+	正符号	43	2B
45	,	コンマ	44	2C
46	–	ハイフン、負符号	45	2D
47	.	ピリオド、小数点	46	2E
48	/	斜線	47	2F
49	0		48	30
50	1		49	31
51	2		50	32
52	3		51	33
53	4		52	34
54	5		53	35
55	6		54	36
56	7		55	37
57	8		56	38
58	9		57	39
59	:	コロン	58	3A
60	;	セミコロン	59	3B
61	<	不等号 (より小)	60	3C
62	=	等号	61	3D
63	>	不等号 (より大)	62	3E
64	?	疑問符	63	3F
65	@	アットマーク	64	40

表 63. ASCII 照合順序 (続き)

序数	記号	意味	10 進数表記	16 進数表記
66	A		65	41
67	B		66	42
68	C		67	43
69	D		68	44
70	E		69	45
71	F		70	46
72	G		71	47
73	H		72	48
74	I		73	49
75	J		74	4A
76	K		75	4B
77	L		76	4C
78	M		77	4D
79	N		78	4E
80	O		79	4F
81	P		80	50
82	Q		81	51
83	R		82	52
84	S		83	53
85	T		84	54
86	U		85	55
87	V		86	56
88	W		87	57
89	X		88	58
90	Y		89	59
91	Z		90	5A
92	[	左大括弧	91	5B
93	\	逆斜線	92	5C
94	]	右大括弧	93	5D
95	^	脱字記号 (^)	94	5E
96	_	下線	95	5F
97	`	抑音符号	96	60
98	a		97	61
99	b		98	62
100	c		99	63
101	d		100	64
102	e		101	65
103	f		102	66
104	g		103	67
105	h		104	68

表 63. ASCII 照合順序 (続き)

序数	記号	意味	10 進数表記	16 進数表記
106	i		105	69
107	j		106	6A
108	k		107	6B
109	l		108	6C
110	m		109	6D
111	n		110	6E
112	o		111	6F
113	p		112	70
114	q		113	71
115	r		114	72
116	s		115	73
117	t		116	74
118	u		117	75
119	v		118	76
120	w		119	77
121	x		120	78
122	y		121	79
123	z		122	7A
124	{	左中括弧	123	7B
125	¦	分割垂直バー	124	7C
126	}	右中括弧	125	7D
127	~	ティルド	126	7E

付録 C. サポートされる CCSID 値

次のリストには、UCS-2 値との間での変換がサポートされている CCSID 値が入っています。ユニコード CCSID 間の変換はサポートされていません。

注: CCSID 932、936、および 949 は次のように変換されます。

CCSID	マップ先
932	943
936	1386
949	1363

037	395	870	939	1018	1141	4951
256	420	871	941	1019	1142	4952
259	423	874	942	1025	1143	4960
273	424	875	943	1026	1144	5037
274	437	880	946	1027	1145	5039
277	500	891	947	1028	1146	5048
278	813	895	948	1038	1147	5049
280	819	896	950	1040	1148	5067
282	829	897	951	1041	1149	5142
284	833	903	971	1042	1250	5346
285	834	904	952	1043	1251	5347
287	835	905	955	1046	1252	5348
290	836	907	960	1047	1253	5349
293	837	909	961	1050	1254	5350
297	838	910	963	1051	1255	5351
300	850	912	964	1088	1256	5352
301	851	913	933	1089	1257	5353
361	852	914	949	1092	1275	5354
363	855	915	970	1097	1276	5478
367	856	916	1004	1098	1277	8612
382	857	918	1006	1112	1350	9030
383	858	919	1008	1114	1351	9056
385	860	920	1009	1115	1363	9066
386	861	921	1010	1116	1364	9145
387	862	922	1011	1117	1380	28709
388	863	923	1012	1118	1381	33722
389	864	924	1013	1119	1382	
391	865	927	1014	1122	1383	
392	866	930	1015	1123	1386	
393	868	935	1016	1124	1388	
394	869	937	1017	1140	4948	

付録 D. RPG コンパイラーの比較

VisualAge RPG 言語は、RPG IV 言語に基づいています。これは、クライアント / サーバー環境でグラフィカル・ユーザー・インターフェースを使用してアプリケーションを開発して、実行できるように拡張されています。

VisualAge RPG では一定の機能がサポートされていないような場合があります。たとえば、VisualAge RPG には RPG サイクルがありません。RPG サイクルがサポートされていないので、制御の切れ目や突き合わせフィールドなど、これに関連した機能もサポートされていません。

ワークステーション・アプリケーション開発環境の利点を生かすために、VisualAge RPG に機能が追加されているか (たとえば SHOWWIN などの命令コードはアプリケーションのウィンドウを表示するために使用されます)、あるいは既存の ILE RPG for AS/400 機能が変更されています (たとえば、/COPY コンパイラー指示を使用して OS/400 ファイルまたはワークステーション・ファイルをコピーします)。

この付録では、ILE RPG と VisualAge RPG の相違を要約します。

RPG サイクル

VisualAge RPG では RPG サイクルがサポートされていないので、サイクルに関連した標識はサポートされていません。RPG サイクルに関連した仕様上の項目もサポートされていません。

VisualAge RPG 標識

VisualAge RPG では以下の標識がサポートされています。VisualAge RPG によってサポートされていない標識のリストについては、『サポートされない標識』を参照してください。

レコード識別標識
01 - 99、LR

フィールド標識
01 - 99

結果の標識
01 - 99、LR

サポートされない標識

以下の標識を使用することはサポートされていません。

オーバーフロー標識
*INOA - *INOG, *INOV, *IN01 - *IN99, 1P

レコード識別標識
H1 - H9、L1 - L9、U1 - U8、RT

制御レベル標識

L1 - L9

フィールド標識

H1 - H9、U1 - U8、RT

結果の標識

H1 - H9, OA - OG, OV, L1 - L9, U1 - U8,KA - KN、KP - KY, RT

ファイル条件付け

VisualAge RPG ファイル仕様では EXTIND キーワードはサポートされていません。これは、ファイル条件付けに標識を使用できないということを意味します。

サポートされないワード

以下では特殊な機能をもつワードおよび VisualAge RPG でサポートされていない予約語について説明します。

- *ALTSEQ, *EQUATE, *FILE,
- ユーザー日付: VisualAge RPG プログラムはバッチで実行できません。したがって、ユーザー日付およびバッチ・プログラムに関する規則は、どれも VisualAge RPG プログラムには適用されません。

VisualAge RPG のワードについては、3 ページの『第 1 章 記号名と予約語』 を参照してください。

コンパイラー指示

/COPY コンパイラー指示には別のファイルからのレコードが含まれます。このファイルはワークステーションまたは iSeries サーバー上に存在することができます。レコードは、/COPY ステートメントが現れる場所に挿入され、プログラムとともにコンパイルされます。詳細については、11 ページの『/COPY または /INCLUDE)』を参照してください。

エラーおよび例外処理

VisualAge RPG アプリケーションでのエラーおよび例外の処理には、処理コンポーネントおよびアプリケーションのグラフィカル・ユーザー・インターフェースに対するサポートが含まれます。詳細については、31 ページの『第 4 章 コンポーネントの処理』 および 60 ページの『イベント・エラー処理』を参照してください。

Data

データ・タイプおよびデータ形式

以下では、ILE RPG for AS/400 と VisualAge RPG との間のデータ・タイプおよび形式の相違を要約しています。VisualAge RPG でサポートされるデータ・タイプおよび形式については、107 ページの『第 9 章 データ・タイプおよびデータ形式』を参照してください。

2 進形式

2 進データは、サーバーとクライアントの間で使用される時には、並べ替えられます。

基底ポインターのデータ・タイプ

ILE RPG for AS/400 の基底ポインター・フィールドは 16 バイトの長さであり、16 バイト境界で位置合わせされなければなりません。VisualAge RPG の基底ポインター・フィールドは 4 バイトの長さであり、4 バイト境界で位置合わせされなければなりません。

パック 10 進形式

ILE RPG for AS/400 では、正数に 16 進数 F、負数に 16 進数 D が使用されます。VisualAge RPG では、正数に 16 進数 C、負数に 16 進数 D が使用されます。VisualAge RPG は、正数に 16 進数 A、E、および F、負数に 16 進数 B もサポートします。

プロシージャー・ポインターのデータ・タイプ

ILE RPG for AS/400 のプロシージャー・ポインター・フィールドは 16 バイトの長さであり、16 バイト境界で位置合わせされなければなりません。VisualAge RPG のプロシージャー・ポインター・フィールドは 4 バイトの長さであり、4 バイト境界で位置合わせされなければなりません。

ゾーン 10 進数形式

ILE RPG for AS/400 では、正数に 16 進数 F、負数に 16 進数 D が使用されます。VisualAge RPG では、正数に 16 進数 3、負数に 16 進数 7 が使用されます。さらに VisualAge RPG は正数に 16 進数 0、1、2、8、9、A、および B、負数に 16 進数 4、5、6、C、D、E、および F もサポートします。

リテラルおよび名前付き固定情報

以下では、ILE RPG for AS/400 と VisualAge RPG との間のリテラルおよび名前付き固定情報の相違を要約しています。VisualAge RPG でサポートされるデータ・タイプおよび形式については、157 ページの『第 10 章 リテラルおよび名前付き固定情報』 を参照してください。

グラフィック・リテラル

ILE RPG for AS/400 グラフィック文字の形式は G'oK1K2i' です。ここで**o** と **i** シフトアウト文字とシフトイン文字です。シフトアウト文字とシフトイン文字は、VisualAge RPG のグラフィック文字では使用されません。形式は G'K1K2' です。

表意定数

ILE RPG for AS/400 の表意定数では、シフトアウト文字とシフトイン文字を使用できます (たとえば ALLG'oK1K2i')。シフトアウト文字とシフトイン文字は、VisualAge RPG の表意定数では使用されません。

以下は、VisualAge RPG 固有の表意定数です。詳細については、161 ページの『表意定数』を参照してください。

*ABORT	*BLACK	*BLUE	*BROWN
*CANCEL	*CYAN	*DARKBLUE	*DARKCYAN
*DARKGREEN	*DARKGRAY	*DARKPINK	*DARKRED
*END	*GREEN	*HALT	*IGNORE
*INFO	*NOBUTTON	*PALEGRAY	*PINK
*RED	*RETRY	*START	*YELLOW
*YESBUTTON	*WARN	*WHITE	

データ域

ローカル・データ域およびプログラム初期設定パラメーター・データ域は、サポートされていません。以下のものは使用できません。

- 演算項目 2 に *LDA または *PDA、結果フィールドに名前のある *DTAARA DEFINE 演算命令
- 定義仕様での DTAARA(*LDA) または DTAARA(*PDA)

VisualAge RPG でのデータ域のサポートの詳細については、165 ページの『第 11 章 データ構造』 を参照してください。

配列およびテーブル

VisualAge RPG は、配列およびテーブルでは次の命令コードをサポートしていません。MLLZO、MHHZO、MLHZO、MHLZO

AS/400 システムは EBCDIC 体系で、OS/2 システムは ASCII 体系です。VisualAge RPG は ASCII 照合順序を使用します。詳細については、175 ページの『第 12 章 配列およびテーブルの使用』を参照してください。

注: 配列の場合、グラフィック・データおよび ALTSEQ キーワードはサポートされていません。

VisualAge RPG の配列およびテーブルの詳細については、175 ページの『第 12 章 配列およびテーブルの使用』 を参照してください。

編集コード

ユーザー定義編集コードはサポートされていません。VisualAge RPG がサポートする編集コードおよび編集語については、 193 ページの『第 13 章 数値フィールドの編集』 を参照してください。GUI パーツの編集については、*VisualAge for RPG プログラミング*, SC88-5607-05 を参照してください。

ファイル

VisualAge RPG では、ファイルの装置固有入出力フィードバック域の内容が INFDS の装置固有フィードバック・セクションにコピーされるのは、ファイルの POST 後だけです。詳細については、 638 ページの『POST (転記)』 を参照してください。

仕様

以下のレコードは、VisualAge RPG ではサポートされていません。

- ファイル変換レコード
- 代替照合順序レコード

制御仕様

VisualAge RPG の制御仕様の詳細については、 223 ページの『第 16 章 制御仕様』を参照してください。

データ域

制御仕様でアプリケーションの生成および実行についての情報を指定しなかった場合には、ILE RPG はライブラリー・リスト (*LIBL) 中の RPGLEHSPEC という名前のデータ域を検索します。見つからない場合には、ILE RPG は QRPGLE 中の DFTLEHSPEC という名前のデータ域を検索します。VisualAge RPG は、データ域を *LIBL または QRPGLE から検索しません。

キーワード

以下のキーワードは、VisualAge RPG 制御仕様ではサポートされていません。

- ACTGRP
- ALTSEQ
- BNDDIR
- DFTACTGRP
- DFTNAME
- ENBPFRCOL
- FIXNBR
- FORMSALIGN
- FTRANS
- LANGID
- OPENOPT
- OPTIMIZE
- PRFDTA
- SRTSEQ
- TEXT
- THREAD

- USRPRF

OPTION キーワードの *{NO}SRCSTMT および *{NO}DEBUGIO 値はサポートされていません。

CCSID キーワードへのパラメーターは、ワークステーション CCSID でなければなりません。

以下に示すものは、制御仕様で VisualAge RPG 固有のキーワードです。
- CACHE
- CACHEREFRESH
- CVTOEM
- LIBLIST
- SQLBINDFILE
- SQLDBBLOCKING
- SQLDBNAME
- SQLDTFMT
- SQLISOLATIONLVL
- SQLPACKAGENAME
- SQLPASSWORD
- SQLUSERID

VisualAge RPG の制御仕様でサポートされるキーワードについては、 223 ページの『7-80 桁目 (キーワード)』 を参照してください。

ファイル仕様

VisualAge RPG ファイル仕様の詳細については、 237 ページの『第 17 章 ファイル仕様』 を参照してください。

ファイル・サポート

VisualAge RPG は、ILE RPG がサポートするほど多くのファイルをサポートしていません。以下では VisualAge RPG によってサポートされていないファイルをファイル仕様で影響のある位置とともにリストしてあります。

1 次ファイル、2 次ファイル、レコード・アドレス・ファイル

ILE RPG は、1 次ファイル、2 次ファイル、およびレコード・アドレス・ファイルのファイルの指定 (18 桁目) をサポートします。VisualAge RPG は、これらのファイルをサポートしていません。

レコード・アドレス・ファイルおよび索引付きファイル
- VisualAge RPG は、キーの長さまたはレコード・アドレス (29-33 桁目) だけでブランクをサポートします。
- VisualAge RPG プログラムのレコード・アドレスの種類 (34 桁目) には A (文字キー), P (パック 10 進数キー), G (グラフィック・キー), D (日付キー), T (時刻キー)、または Z (タイム・スタンプ・キー) を入れることはできません。
- VisualAge RPG プログラムのファイル編成の指定 (35 桁目) には I (索引付きファイル) または T (レコード・アドレス・ファイル) を入れることはできません。

WORKSTN ファイル

- ILE RPG は入出力共用のファイル・タイプ (17 桁目) をサポートしますが、このタイプは SPECIAL ファイルおよび WORKSTN ファイルで有効です。VisualAge RPG は WORKSTN ファイルをサポートしていないので、入出力共用のファイル・タイプを指定することは SPECIAL ファイルだけに適用されます。

ディスク・ファイルの処理方式

限界内順次アクセスは VisualAge RPG によってサポートされていません。

RPG サイクルに関連した項目

RPG サイクルは VisualAge RPG によってサポートされていないので、以下の項目はサポートされていません。

- ファイルの終わり (E)
- 順序 (A および D)
- 限界値範囲内処理 (L)
- オーバーフロー処理 (OA-OG, OV, または 01-99)

キーワード

VisualAge RPG プログラムのファイル仕様では以下のキーワードはサポートされていません。

DEVID	EXTIND	FORMOFL	INDDS
KEYLOC	MAXDEV	OFLIND	PASS
PGMNAME	RAFDATA	SAVEDS	SAVEIND
SFILE	SLN		

以下に示すものは、VisualAge RPG 固有のファイル仕様キーワードです。

EXTFILE(fname)

EXTFILE キーワードによって、コンパイル時に名前を提供するのではなく、実行時に実際のローカル・ファイル名を指定できます。

PROCNAME (procname)

装置の項目 (42 桁目) に SPECIAL を入力した場合には、procname として指定するモジュールが装置のサポートを扱います。

RCDLEN(fieldname)

RCDLEN キーワードは、ローカル DISK ファイルに使用できます。

REMOTE

DISK (36-42 桁目) を入力した場合には、 REMOTE キーワードは、ディスク装置が AS/400 サーバー上にあるということを指定します。

VisualAge RPG のファイル仕様でサポートされるキーワードについては、 243 ページの『44 ～ 80 桁目 (キーワード)』 を参照してください。

定義仕様

VisualAge RPG はメッセージ・ウィンドウをサポートします。メッセージ・ウィンドウは、定義仕様の 24 桁目に M、25 桁目にブランクを入力して指定します。メッセージ・ウィンドウの詳細については、 255 ページの『第 18 章 定義仕様』 を参照してください。

キーワード

以下では、ILE RPG と VARPG の間の定義仕様の相違について説明します。

ASCEND および DESCEND

ILE RPG は EBCDIC 照合順序を使用します。VisualAge RPG は ASCII 照合順序を使用します。

VisualAge RPG は ALTSEQ をサポートしないので、VisualAge RPG アプリケーションは、コンパイル時配列またはテーブルの順序を検査するのに代替照合順序を使用することはできません。

DTAARA

VisualAge RPG は、DTAARA キーワードによってローカル・データ域 (*LDA) をサポートしていません。

VisualAge RPG プログラムの定義仕様では以下のキーワードはサポートされていません。

- EXPORT
- EXTPGM
- IMPORT
- OPDESC
- OPTIONS(*NOPASS)

以下に示すものは、定義仕様で VisualAge RPG 固有のキーワードです。:

BUTTON(button1:button2...)

メッセージ・ウィンドウ上のボタンを定義するには、BUTTON キーワードを使用します。以下のパラメーターを指定できます (最大 3 つ)。
*OK、*CANCEL、*RETRY、*ABORT、*IGNORE、*ENTER、
*NOBUTTON、*YESBUTTON.

CLTPGM(program name)

CLTPGM キーワードは、CALLP 演算命令を使用して VARPG プログラムによって呼び出されるローカル・プログラムの名前を指定します。

DLL(name)

DLL キーワードは、LINKAGE キーワードと一緒に、Windows API を含む Windows DLL の機能を呼び出すプロシージャーのプロトタイピングに使用されます。

LINKAGE(linkage_type)

CALL 命令コードで使用されるプログラム名が AS/400 サーバーにあることを指定するには、パラメーター *SERVER のある LINKAGE キーワードを使用します。 LINKAGE キーワードと DLL キーワードを一緒に使用すれば、Windows API を含む Windows DLL で機能を呼び出すプロシージャーのプロトタイプを作成できます。

MSGDATA(msgdata1:msgdata2...)

MSGDATA キーワードは、置換テキストを定義するのに使用します。パラメーター (msgdata1:msgdata2...) はフィールド名です。VisualAge RPG は、メッセージを表示する前に、すべてのデータを文字形式に変換します。

MSGNBR(*MSGnnnn または fieldname)

MSGNBR キーワードは、メッセージ番号を定義します。メッセージ番号は、最大で 4 桁の長さとすることができます。

MSGTEXT('message text')

MSGTEXT キーワードは、メッセージ・テキストを定義します。メッセージ・テキストは、単一引用符 (') の間にあります。

MSGTITLE('title text')

MSGTITLE キーワードは、メッセージ・ウィンドウのタイトル・テキストを指定します。単一引用符 (') の間にストリングまたはメッセージ番号を入力できます。

NOWAIT

NOWAIT キーワードは、表示装置ファイルを使用するリモート AS/400 プログラムを呼び出すために使用されます。

STYLE(style_type)

STYLE キーワードは、メッセージ・ウィンドウに現れるアイコンを定義するためにメッセージ・ウィンドウに使用されます。STYLE キーワードには *INFO、*WARN、または *HALT のうちのいずれかを指定できます。

VisualAge RPG 定義仕様でサポートされるキーワードについては、 265 ページの『定義仕様のキーワード』 を参照してください。

入力仕様

VisualAge RPG ファイル仕様の詳細については、 237 ページの『第 17 章 ファイル仕様』 を参照してください。

以下の項目はサポートされていません。

- 順序 (17-18 桁目) の場合、2 桁の数字は入力することはできません。ILE RPG はこのオプションをサポートします。このオプションは、入力レコードの順序を検査するために使用できます。VisualAge RPG ではこのサポートは使用できません。
- 数値 (19 桁目) の場合、1 (順序付けられたグループにこのタイプのレコードが 1 つしかあってはならないことを示します) も N (順序付けられたグループの中にこのタイプのレコードが 1 つまたは複数あってよいことを示します) も入力できません。
- オプション (20 桁目) の場合、O (順序検査を指定すると、レコード・タイプが任意選択であることを示します) を入力することはできません。
- コード・パート (29、37、45 桁目) の場合、Z (文字のゾーン部分を示します) を入力することはできません。

組み込み関数

%GETATR および %SETATR は VARPG 固有の組み込み関数です。詳細については、 434 ページの『%GETATR (属性の検索)』 および 456 ページの『%SETATR (属性の設定)』 を参照してください。

注: VARPG 固有の組み込み関数は、1 バイト値、8 バイト値、およびユニコード値をサポートしていません。

命令コード

このセクションでは、ILE RPG 命令コードと VisualAge RPG 命令コードとを比較します。 VisualAge RPG 命令コードの詳細については、 481 ページの『第 26 章 命令コードの詳細』 を参照してください。

類似の命令コード

以下の命令コードは、ILE RPG と VisualAge RPG の両方で存在します。しかし、これらの命令コードをコーディングする方法には相違があり、またこれらの命令コードを含むアプリケーションを実行すると異なる結果が得られます。命令コードが VisualAge RPG でどのように機能するかについては、命令コードがリストされた見出しを参照してください。

- BEGSR (492 ページの『BEGSR (ユーザー・サブルーチンの開始)』)
- CALL (498 ページの『CALL (AS/400 プログラムの呼び出し)』)
- CALLB (502 ページの『CALLB (機能の呼び出し)』)
- CALLP (503 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』)
- CHAIN (510 ページの『CHAIN (ファイルからのランダム検索)』)
- CLEAR (520 ページの『CLEAR (消去)』)
- CLOSE (523 ページの『CLOSE (ファイルのクローズ)』)
- COMMIT (525 ページの『COMMIT (コミット)』)
- DEFINE (529 ページの『DEFINE (フィールド定義)』)
- DELETE (532 ページの『DELETE (レコードの削除)』)
- DSPLY (544 ページの『DSPLY (「メッセージ」ウィンドウの表示)』)
- FEOD (562 ページの『FEOD (データの終わりの強制)』)
- READ (640 ページの『READ (レコードの読み取り)』)
- WRITE (707 ページの『WRITE (新しいレコードの作成)』)

サポートされない命令コード

VisualAge RPG では以下の命令コードはサポートされていません。

- ACQ
- DUMP
- EXFMT
- FORCE
- MHHZO
- MHLZO
- MLHZO
- MLLZO
- NEXT
- REL
- SHTDN

VisualAge RPG 固有の命令コード

以下の命令コードは、VisualAge RPG 言語に固有なものです。

- BEGACT/ENDACT (アクション・サブルーチン開始, アクション・サブルーチン終了)
- CLSWIN (ウィンドウのクローズ)

- DSPLY (「メッセージ」ウィンドウの表示)
- GETATR/SETATR (属性検索、属性設定)
- READS (選択読み取り)
- SHOWWIN (ウィンドウのロード)
- START/STOP (コンポーネント開始、コンポーネント停止)

これらの命令コードの詳細については、 481 ページの『第 26 章 命令コードの詳細』 を参照してください。

注: DSPLY を除いて、VARPG 固有の命令コードは 1 バイトと 8 バイトの符号付きと符号なしの整数値、およびユニコード値をサポートしません。

CCSID 間の変換

VARPG コンパイラーは、1 バイト文字とグラフィック・データの間の変換をサポートしません。以下の間でのみ変換がサポートされます。:

- グラフィックから UCS-2 または UCS-2 からグラフィック
- 文字から UCS-2 または UCS-2 から文字
- グラフィックからグラフィック (その CCSID が異なる場合)

用語集

この用語集には、次のものから引用した用語とその定義が含まれています。

- The *American National Dictionary for Information Systems* ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). 米国規格協会 American National Standards Institute, 1430 Broadway, New York, New York, 10018 から購入することができます。これらの定義は定義の後のシンボル (A) によって明示されています。
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Committee (ISO/IEC JTC1/SC1). この用語集の公開済み部分の定義は、定義の後のシンボル (I) によって示されています。国際標準草案、委員会草案、および ISO/IEC JTC1/SC1 により開発中の作業文書から引用した定義は、定義の後のシンボル (T) によって示され、これは SC1 の関係国際本部間でいまだ最終合意に至っていないことを示します。
- *IBM Dictionary of Computing* , New York: McGraw-Hill, 1994.
- *Object-Oriented Interface Design IBM Common User Interface Guidelines*, SC34-4399-00, Carmel, IN: Que Corporation, 1992.

［ア行］

アイコン. オブジェクトのグラフィカル表現であり、イメージ、イメージの背景、およびラベルで構成されます。

アイコン・ビュー. コンテナーに含まれている各オブジェクトがアイコンとして表示される標準的な目次ビュー。

アクション. (1) アクション・サブルーチンの同義語。 (2) プロジェクトのパーツを操作するために使用したり、あるいはビルドにかかわるために使用する実行可能なプログラムまたはコマンド・ファイル。

アクション・サブルーチン. 特定のイベントに応答するために作成するロジック。

アクティブ・ウィンドウ. 現在ユーザーが対話しているウィンドウ。 これがキーボード入力を受け取るウィンドウです。

アニメーション制御パーツ. Windows で AVI という拡張子を持つビデオ・ファイルの再生、または Java アプリケーションでアニメーション GIF シーケンスの再生を可能にするパーツ。

アプリケーション. コンピューター上で特定のユーザー・タスクを実行するために使用するソフトウェア・コンポーネントの集合。

アプリケーション・プログラミング・インターフェース (API). オペレーティング・システムによって提供される機能インターフェースまたは個別に注文するライセンス・プログラムであり、高水準言語で書かれたアプリケーション・プログラムがオペレーティング・システムまたはライセンス・プログラムの特定のデータまたは機能を使用できるようにするのもの。

アプレット. Java で書かれ、Java 互換ブラウザーまたは appletviewer の内部で実行されるプログラム。

アンカー. 他のパーツを調整、サイズ決め、およびスペーシングする場合の参照点として使用する任意のパーツ。

移行. (1) 変更された操作環境に移動することであり、通常は新しいリリースまたはバージョンのシステムに移動します。 (2) ある階層のストレージから別の階層のストレージにデータを移動すること。

イベント. パーツの状態に対する変更の結果として生成されるシグナル。たとえば、ボタンを押すと *Press* イベントが生成されます。

イメージ・パーツ. ウィンドウ上で、BMP または ICO ファイルからピクチャーを表示するのに使用するパーツ。

インポート. AS/400 表示装置ファイル・オブジェクトを該当する VARPG パーツに変換する機能。エクスポートと対比。

ウィンドウ・パーツ. 可視境界を持つ領域で、オブジェクトのビューを表したり、あるいはこれを使用してユーザーはコンピューター・システムでダイアログを管理します。パーツ・パレットまたはパーツ・カタログ内のウィンドウ・パーツを指してクリックし、それをプロジェクト・ウィンドウ上に置くことができます。

ウェーブ・ファイル. ウェーブ形式装置のオーディオ・サウンド用に使用するファイル。

エクスポート. アプリケーションの外側で使用するために内部ファイルを何らかの標準ファイル形式に変換する機能。 *インポート*と対比。

エラー・ログ. エラー・ログにエラーのトラックを保存します。エディターを使用すると、ソース内でエラーが起こった場所を表示できます。

オブジェクト. (1) それ自身について記述し、場合によってはデータについて記述する一組のプロパティーから成る名前の付いた記憶スペース。 オブジェクトはストレージの中に存在し、そのスペースを占有するもので、それについて操作を実行することができます。 オブジェクトの例としては、プログラム、ファイル、ライブラリー、およびフォルダーなどがあります。 (2) ユーザー・インターフェースのビジュアルなコンポーネントであり、ユーザーはタスクを実行するためにこれで作業することができます。 オブジェクトはテキストまたはアイコンとして表示されることがあります。

オブジェクト指向プログラミング. 他のオブジェクトと対話することのできるオブジェクトのデータおよび操作について記述するための階層的に編成されたクラスとして、プログラムを構造化するためのメソッド。(T)

オブジェクト-アクション・パラダイム. 対話の 1 つのパターンであり、このパターンでユーザーはオブジェクトを選択し、そのオブジェクトに適用するアクションを選択します。

オブジェクト・プログラム. 実行に適したターゲット・プログラム。 オブジェクト・プログラムは、リンクを必要とする場合と、そうでない場合があります。(T)

オペレーティング・システム. コンピューター・システムの全体の操作を制御するシステム・プログラムの集合。

［カ行］

カーソル. ユーザーとキーボードの対話によって現れる可視位置の指示。

カラー・パレット. アプリケーションの GUI の中の任意のパーツのカラーを変更するために使用するカラーのセット。

カレンダー・パーツ. カレンダーを追加するパーツであり、ユーザーはテキスト、カラー、およびその他の属性を組み込むためにこれを変更することができます。

簡略記号. 選択項目のテキスト内の 1 つの文字で、その文字の下に下線を引いて示されます。 *略号の選択*も参照してください。

キャンバス付きウィンドウ・パーツ. ウィンドウ・パーツとキャンバス・パーツの組み合わせ。ウィンドウ・パーツおよびキャンバス・パーツも参照してください。

キャンバス・パーツ. 他のいろいろなパーツを指示してクリックし、それらを位置指定して、グラフィカル・ユーザー・インターフェースを作成するようにそれらを編成するパーツ。キャンバス・パーツは、ウィンドウ・パーツまたは

ノートブック・ページ・パーツのいずれかのクライアント域を占めます。キャンバス・パーツ付きノートブック・ページおよびキャンバス・パーツ付きウィンドウも参照してください。

キャンバス・パーツ付きノートブック・ページ. ノートブック・ページ・パーツとキャンバス・ページ・パーツの組み合わせ。ノートブック、キャンバス・パーツも参照してください。

強調. カラー変更を強調表示したり、あるいはオブジェクトまたは選択項目と対話するユーザーの能力に影響するそれらのオブジェクトまたは選択項目に関連する条件の可視な指示を強調表示すること。 また、強調により、選択項目またはオブジェクトの状態についての追加情報がユーザーに与えられます。

共通ユーザー・アクセス・アーキテクチャー (CUA アーキテクチャー). 人間とワークステーションまたは端末装置の間のダイアログの指針。

共用コンポーネント. 複数のプロジェクトがアクセスできるコンポーネント。

組み合わせボックス. 入力フィールドとリスト・ボックスの機能を結合する制御。組み合わせボックスには、入力フィールドを完成させるためにユーザーがスクロールして選択することのできるオブジェクトのリストが入っています。これがない場合は、ユーザーはテキストを直接入力フィールドに入力することができます。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内の組み合わせボックス・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

クライアント. (1) サーバーにデータを提供するために、サーバーに依存しているシステム。 (2) VARPG アプリケーションが実行される PWS 。*DDE クライアント*も参照してください。

クライアント域. ユーザーのワークスペースであるウィンドウ内の部分であり、ここでユーザーは情報を入力したり、選択フィールドから選択項目を選択します。プライマリー・ウィンドウにおいて、ユーザーが作業するオブジェクトをアプリケーション・プログラマーが提供する領域。

クライアント／サーバー. 分散データ処理における相互作用の形式であり、一方のサイトのプログラムが別のサイトのプログラムに要求を送信し、応答を待機します。 要求側のプログラムをクライアントと呼び、応答側のプログラムをサーバーと呼びます。クライアント、サーバー、*DDE クライアント*、*DDE サーバー*も参照してください。

グラフィカル・ユーザー・インターフェース (GUI). 高解像度グラフィックスを利用したユーザー・インターフェースのタイプ。グラフィカル・ユーザー・インターフェースには、グラフィックスの組み合わせ、オブジェクト - アクション・パラダイム、ポインティング・デバイスやメニュー・バーおよびその他のメニューの使用、オーバーラップ・ウィンドウ、およびアイコンが組み込まれています。

グラフィック・プッシュボタン・パーツ. グラフィックのラベルが付いたプッシュボタンで、ユーザーがこれを選択した時に開始されるアクションを表します。プッシュボタン・パーツと対比。

グラフ・パーツ. ユーザーが GUI にグラフを追加するために使用できるパーツ。使用可能なグラフ・スタイルは折れ線グラフ、棒グラフ、折れ線と棒グラフ、または円グラフです。

クリック. 選択項目またはオブジェクトからポインターを外さないでマウス・ボタンを押して放すこと。 *ダブルクリック*も参照してください。

クリップボード. データを一時的に保持するためにシステムによって提供された記憶領域。クリップボード中のデータは他のアプリケーションにも使用可能です。

グループ・ボックス・パーツ. 制御グループの回りを取り囲む四角形のフレームであり、これらの制御が関連していることを示し、そのグループに対して任意指定のラベルを提供します。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内のグループ・ボックス・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

グループ・マーカー. パーツをグループ内の最初のものであるとして認識するマーク。 パーツのグループ内でカーソルを移動している時に、最後のパーツに達すると、カーソルはグループ内の最初のパーツに戻ります。

コールド・リンク会話. DDE において、クライアント・プログラムからサーバー・プログラムに対して行われる明示的な要求。サーバー・プログラムはこの要求に応答します。*ホット・リンク会話*と対比。

構成. 情報処理システムのハードウェアおよびソフトウェアが編成され、相互接続される方法 (T)。

構文検査. ソースの編集中に、各行の構文が正しいことを確認します。確認することによって、コンパイル・エラーを回避することができます。このオプションは、オンまたはオフに設定することができます。一定の仕様タイプ (C 仕様など) または特定のストリングを持つ行だけを表示することができます。

項目. 動的データ交換におけるデータの単位。 たとえば、スプレッドシートにおける 左上のセルの位置は、行 1、列 1 です。このセル位置は、項目 R1C1 と呼ばれることもあります。

コンテナー・パーツ. 関連レコードを保管するパーツであり、それらを詳細、アイコン、またはツリー・ビューで表示します。

コンパイル. ソース・プログラムを実行可能プログラム (オブジェクト・プログラム) に変換すること。

コンポーネント. プロジェクト内の関連ファイルの機能上のソート。 コンポーネントは、制御仕様書に NOMAIN および EXE キーワードがない時に作成されます。

コンポーネント参照パーツ. 1 つのコンポーネントが VARPG アプリケーション中の別のコンポーネントと通信できるようにするパーツ。

［サ行］

サーバー. ネットワーク中にあり、他のシステム (クライアントと呼ばれる) の要求を処理するシステム。

サーバーの別名. サーバー名の代わりに使用することができるユーザー定義の名前。

最小化ボタン. タイトル・バーの右端のボタンの隣にあるボタンで、ウィンドウを最小サイズに縮小します。 *最大化ボタンおよび[隠す] ボタン*と対比。

最大化ボタン. タイトル・バーの右端にあるボタンで、ウィンドウをその最大サイズに拡大するためにクリックします。 *最小化ボタン*、 *[隠す] ボタン*と対比。

索引. リスト・ボックスまたは組み合わせボックスなどの VARPG パーツ中の項目の ID 。

サブファイル・パーツ. レコードのリストを表示するために使用するパーツであり、沢山のフィールドから成っています。 このパーツは AS/400 サブファイルに類似しています。*サブファイル・フィールド*も参照してください。

サブファイル・フィールド. サブファイル・パーツ内のフィールドを定義するために使用するフィールド。サブファイル・パーツも参照してください。

サブプロシージャー. サブプロシージャーはメイン・ソース・セクションの後に指定されるプロシージャーです。メイン・ソース・セクションの定義仕様の中に対応するプロトタイプが入っていなければなりません。

サブメニュー. 別のメニューの中のカスケード選択項目から表示され、それに関連した選択項目が入っているメニュー。サブメニューはプルダウン・メニューまたはポップアップ・メニューの長さを縮小するために使用します。 サブメニュー・パーツも参照してください。

サブメニュー・パーツ. メニュー項目または既存のメニューからサブメニューを開始したり、あるいはメニュー・バー上のメニュー項目からプルダウン・メニューを開始するために使用するパーツ。サブメニューおよびメニュー*項目*・パーツも参照してください。

参照解除. パーツと AS/400 データベース・フィールドとの間の関連を除去するアクション。

参照フィールド. AS/400 のデータベース・フィールドであり、入力フィールド・パーツがそのプロパティーを継承することができます。

状況バー. ウィンドウのパーツであり、現行のビューまたはオブジェクトの状態を示す情報を表示します。 *情報域*も参照してください。

状況バー・パーツ. ウィンドウ上にあり、そのウィンドウの処理またはアクションについての追加情報を表示できるパーツ。

詳細ビュー. オブジェクトについての説明情報を提供するテキストが小アイコンに結合されている標準目次ビュー。

情報域. カーソルが置かれているオブジェクトまたは選択項目についての情報が表示されるウィンドウのパーツ。 情報域には、処理の通常の完了についてのメッセージも表示されます。 *状況バー*も参照してください。

情報表示機能 (IPF). プログラマブル・ワークステーション上でオンライン・ヘルプを作成するのに使用するツール。

情報表示機能 (IPF) ファイル. アプリケーションのヘルプ・ソースが保管されるファイル。

進行状況バー・パーツ. ファイルのコピー、データベースのロード、などの処理の進行をグラフィカルに示すために使用できるパーツ。

進行状況表示. 処理の進行状況をユーザーに知らせるために使用される 1 つまたは複数の制御。

水平方向のスクロール・バー・パーツ. ウィンドウに水平方向のスクロール・バーを追加するパーツ。このパーツによって、ユーザーは、情報のペインを左から右または右から左にスクロールできるようになります。

スクロール・バー. 特定の方向にまだ情報があることをユーザーに示し、マウスまたはページ・キーを使用して移動しその情報を表示できることを示すパーツ。

スピン・ボタン・パーツ. 入力フィールドの 1 つのタイプであり、関連があるが互いに排他的な選択項目のリングを表示し、ユーザーはこれをスクロールして 1 つの選択項目を選択することができます。 ユーザーは入力フィールドに正しい選択を入力することもできます。 パーツ・パレットまたはパーツ・カタログ内のスピン・ボタン・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

スライダー・アーム. スライダー内の可視標識であり、ユーザーは数値を変更するためにこれを移動することができます。

スライダー・パーツ. ユーザー・インターフェースのビジュアル・コンポーネントであり、数量を表し、その数量に使用できる値の範囲に対するリレーションシップを表します。ユーザーはこの数量の値を変更することもできます。パーツ・パレットまたはパーツ・カタログ内のスライダー・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

スレッド. プロセス中で実行される操作の最小の単位。

静的テキスト・パーツ. 他のパーツのラベルとして使用されるパーツ。入力フィールド・パーツに対するプロンプトなど。

セカンダリー・ウィンドウ. プライマリー・ウィンドウの中の情報に依存した情報が入っているウィンドウで、プライマリー・ウィンドウ中の対話を補足するために使用されます。プライマリー・ウィンドウも参照してください。ポップアップ・ウィンドウの同義語。

設計ウィンドウ. GUI Designer の中のウィンドウであり、ユーザー・インターフェースを作成するためのパーツが置かれています。

選択ボタン. *左マウス・ボタン* を参照してください。

選択枠. VARPG パーツまたはオーダーメイド・パーツの回りに現れる可視枠で、マウスまたはキーボードを使用して移動できます。

ソース・ディレクトリー. VARPG アプリケーションのすべてのソース・ファイルが保管されているディレクトリー。

ソース・パーツ. ソース・パーツの状態が変更された時には常にターゲット・パーツに通知できるパーツ。 ソース・パーツは複数のターゲットを持つことができます。

操作ボタン. *右マウス・ボタン* を参照してください。

外枠パーツ. パーツのグループの回りを囲む長方形ボックスのパーツで、その中のすべてのパーツが関連していることを示します。

［タ行］

ターゲット・ディレクトリー. ビルド後にコンパイル済みの VARPG アプリケーションが保管されるディレクトリー。 *ターゲット・フォルダー*と対比。

ターゲット・パーツ. ソース・パーツの状態が変更された時には常に、ソース・パーツからリンク・イベントを受信するパーツ。

ターゲット・フォルダー. VARPG アプリケーションを表すアイコンが入っているオブジェクト。

ターゲット・プログラム. プロジェクトによってビルドされるオブジェクト。ダイナミック・リンク・ライブラリー (DLL) など。

タイトル・バー. 各ウィンドウの最上部にある領域で、システム・メニュー・シンボルが入っています。

ダイナミック・リンク・ライブラリー (DLL). リンク時ではなくロード時または実行時にプログラムにバインドされる実行可能コードおよびデータが入っているファイル。 ダイナミック・リンク・ライブラリー中のコードおよびデータは複数のアプリケーションで同時に共用することができます。

タイマー・パーツ. 2 つのイベント間の時間間隔を追跡し、その間隔が経過した時に 2 番目のイベントを起動するために使用するパーツ。

縦方向のスクロール・バー・パーツ. ウィンドウに縦方向のスクロール･バーを追加するパーツ。このパーツによって、ユーザーは、情報のペインを縦方向にスクロールできるようになります。

ダブルクリック. マウス・ボタンを素早く 2 回押すこと。

タブ・ストップ. ユーザーがインターフェースの中を移動するためにタブ・キーを使用する時に、パーツにフォーカスできるように、そのパーツのタブ・ストップを設定するために使用する属性。

チェック・ボックス・パーツ. 選択項目を表している関連テキストを持つ正方形のボックス。ユーザーが選択項目を選択すると、その選択項目が選択されたことを示す標識がチェック・ボックスの中に現れます。 その選択項目を再び選択することによって、チェック・ボックスを取り消すことができます。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内のチェック・ボックス・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

直接編集. ユーザーがオブジェクトをマウスでドラッグしたり、オブジェクトのポップアップ・メニューと対話することによって、そのオブジェクトを処理することのできる手法を使用すること。

ツールバー. ユーザーがマウスを使用して実行することのできるアクションを表す 1 つまたは複数のグラフィカル選択項目が入っているメニュー。

ツリー・ビュー. オブジェクトの目次を階層的に表示する方法。

データベース. (1) 複数ユーザーのためのデータをオンデマンドで受け入れ、保管し、および提供するための指定の構造体を持つデータの集合。(T) (2) システム中に保管されているすべてのデータ・ファイル。

データ・オブジェクト. テキスト、グラフィックス、オーディオ、またはビデオなどの情報を運ぶオブジェクト。

デフォルト. ユーザーによって値が指定されていない時に、システムまたはプログラムによって自動的に供給または想定される値。デフォルト値はプッシュボタンまたはグラフィック・プッシュボタンに割り当てることができます。

デフォルト・アクション. 何らかのアクション (たとえば Enter キーを押す) が取られた時に実行されるアクション。

トークン強調表示. コードの読みやすさを向上させます。異なるカラーまたはフォントを使用して、異なる言語コンポーネントの強調表示を構成して、プログラム構造を識別することができます。トークンの強調表示はオンにしたりオフにしたりすることができます。

動的データ交換 (DDE). プログラム間またはプログラムとデータ・ファイル・オブジェクトの間のデータの交換。 1 つのプログラムまたはセッションの中の情報に対して行われた変更が、他のプログラムによって作成された同じデータに適用されます。 *DDE 会話*、*DDE クライアント*、*DDE サーバー*も参照してください。

トピック. 動的データ交換 (DDE) において、DDE 会話のサブジェクトとなるデータのセット。

ドラッグ. マウスを使用してオブジェクトを移動またはコピーすること。たとえば、マウス・ボタンを押したままで移動することにより、ウィンドウ・ボーダーをドラッグしてそのウィンドウを大きくすることができます。 *ドラッグ・アンド・ドロップ* も参照してください。

ドラッグ・アンド・ドロップ. マウスを使用してオブジェクトを直接移動し、別の場所に置くこと。

ドロップダウン組み合わせボックス. 組み合わせボックスの変形であり、ユーザーが明示的にアクションを実行して可視にするまでは、リスト・ボックスは隠されています。

ドロップダウン・リスト. 現在の選択項目だけが可視になっている単一の選択フィールド。他の選択項目が入っているリスト・ボックスを表示するためのアクションを明示的に実行するまでは、他の選択項目は隠されたままです。

[ナ行]

ナビゲーション・パネル. サブファイル中のレコードの可視選択を制御するために使用するボタンのグループ。

入出力 (I/O). コンピューターに提供されるデータ、またはコンピューター処理の結果としてのデータ。

入力フィールド・パーツ. ユーザーが情報を入力できる画面上の領域 (そのフィールドが読み取り専用でない場合)。通常、入力フィールドの境界が指示されています。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内の入力フィールド・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

入力フォーカス. キーボードまたはマウスのどちらからでもユーザーの対話が可能なウィンドウの領域。

ノートブック・パーツ. ノートブックのグラフィカル表現。 ノートブック・ページをノートブック・パーツに追加し、その後でそれらのページをタブ付きの区切り記号によって分離されたセクションにグループ化することができます。 Windows では、ノートブックのことを Windows タブ制御と呼ぶことがあります。*ノートブック・ページ・パーツ*、*キャンバス・パーツ付きノートブック・ページ*も参照してください。

ノートブック・ページ・パーツ. *ノートブック・パーツ*にページを追加するために使用するパーツ。*ノートブック*も参照してください。

[ハ行]

パーツ. VARPG アプリケーションの GUI を構成するオブジェクト。

パーツ・カタログ. VARPG アプリケーション用のグラフィカル・ユーザー・インターフェースを作成するために使用するすべてのパーツのための記憶スペース。

パーツ・パレット. アプリケーション用の現在のグラフィカル・ユーザー・インターフェースをビルドするのに最も適しているパーツの集合。 1 つの GUI を終了したら、パレットをきれいにふき取り、次のアプリケーションに必要なパーツをパーツ・カタログから追加することができます。

パッケージ. VARPG アプリケーションのすべてのパーツを配布用に収集するために使用する機能。

非活動ウィンドウ. 指定の瞬間にキーボード入力を受け入れることができないウィンドウ。

左マウス・ボタン. デフォルトとして、選択のために使用するマウス上の左のボタン。

ビルド. VARPG アプリケーションのコンポーネントを構成するいろいろなソース・コードが、実行可能なバージョンのアプリケーションを作成するためにコンパイルされ、リンクされるプロセス。

ファイル. 関連データの集合であり、割り当てられた名前で保管されたり検索されます。ファイルには、プログラム (プログラム・ファイル・オブジェクト) を開始する情報、テキストまたはグラフィック (データ・ファイル・オブジェクト) を含む情報、あるいは一連のコマンド (バッチ・ファイル) を処理する情報を組み込むことができます。

フィールド. (1) ウィンドウ内の識別可能な領域であり、たとえば、ユーザーがテキストを入力する入力フィールドなど。 (2) 名前や容量などの関連したバイトのグループであり、レコード中の単位として扱われます。

フォーカス. *入力フォーカス*の同義語。

フォント・パレット. アプリケーション GUI の中のパーツのフォントを変更するために使用できるフォントのセット。

復元ボタン. ウィンドウが最大化された後に、タイトル・バーの右端の隅に現れるボタン。復元ボタンを選択すると、ウィンドウは最大化される前のサイズと位置に戻ります。*最大化ボタン*も参照してください。

複数行編集 (MLE) パーツ. ユーザーが複数行のテキストを入力することのできる入力フィールドを表すパーツ。

プッシュボタン・パーツ. ユーザーがプッシュボタンを選択した時に開始されるアクションを表すテキストがラベルとして付いているボタン。 パーツ・パレットまたはパーツ・カタログ内のプッシュボタン・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。「グラフィック・プッシュボタン・パーツ」も参照してください。

プライマリー・ウィンドウ. ユーザーとアプリケーションの間の主たる会話が行われるウィンドウ。メイン・ウィンドウ同義。

プラグイン. ユーザーまたは外部のベンダーによって作成された機能で、 VARPG プログラムで使用することができます。

プルダウン・メニュー. メニュー・バーの選択した項目またはシステム・メニュー・シンボルから拡張されるメニュー。 プルダウン・メニューの中の選択項目はある意味で互いに関連しています。

プログラマブル・ワークステーション (PWS). ある程度の処理能力を持っているワークステーションで、ユーザーがその機能を変更することができるワークステーション。

プロシージャー. プロシージャーは、CALLP 命令コードで呼び出すことのできるコードの任意の部分です。

プロシージャー・インターフェース定義. プロシージャー・インターフェース定義は、プロシージャーの定義の中のプロトタイプ情報の反復です。 これは、プロシージャーの入力パラメーターを宣言したり、プロシージャーの内部定義が外部定義 (プロトタイプ) と矛盾しないことを保証するために使用します。

プロジェクト. ダイナミック・リンク・ライブラリー (DLL) または実行可能ファイル (EXE) などの単一のターゲットをビルドするのに必要なデータおよびアクションの完全なセット。

プロトタイプ. プロトタイプは、呼び出しインターフェースの定義です。これには次のような情報が組み込まれています : 呼び出しがバインド (プロシージャー) であるかダイナミック (プログラム) であるか ; 外部名 ; パラメーターの数および特長 ; どのパラメーターを渡さなければならないか ; 戻り値のデータ・タイプ (プロシージャーの場合) 。

プロパティー・ノートブック. タブ区分ページで複数のセクションに分離されたページを含むバウンド・ノートブックに類似したグラフィカル表現。 1 つのセクションから別のセクションに移動するためには、ノートブックのタブを選択してください。

プロンプト. (1) ユーザーの応答を要求するために、プログラムによって送信される視覚または音によるメッセージ。(T) (2) ユーザーからの入力を要求したり、操作情報を提供する表示記号またはメッセージ。ユーザーは続行するためには、このプロンプトに応答しなければなりません。

ポイント／クリック. (1) パーツをパーツ・パレットまたはパーツ・カタログから GUI 設計ウィンドウ、アイコン・ビュー、またはツリー・ビューにコピーするために使用される選択方式。 (2) パーツを希望するいずれかのビューに入れるためには、そのパーツを指示してクリックし、次にカーソルを選択したウィンドウに移動して、そのカーソルを指示して、パーツを表示したい位置でクリックします。アイコンおよびツリー・ビューでは、パーツは親パーツ上に置かれるので、ユーザーはそれを設計ウィンドウ中の希望する位置に移動しなければなりません。

ぼかし表示. ユーザーがそのパーツを選択したり直接操作することができないことを示すために、明度が減光されて表示されているもの。

ボタン. (1) アクションを要求したり開始するために使用するマウスなどのポインティング・デバイス上のメカニズム。 (2) ウィンドウ内のグラフィカル・メカニズムであり、これを選択するとアクションが実行されます。 ボタンの 1 つの例に [OK] プッシュボタンがあり、これを選択するとアクションが開始されます。

ホット・リンク会話. DDE において、サーバー上でデータが変更された時の、サーバー・プログラムによるクライアント・プログラムの自動更新。 *コールド・リンク会話*と対比。

ポップアップ・ウィンドウ. 固定サイズの移動可能なウィンドウであり、アプリケーションがユーザーの要求の処理を続行できるように、アプリケーションに必要な情報をユーザーがこのウィンドウに提供します。 *セカンダリー・ウィンドウ*と同義。

ポップアップ・メニュー. オブジェクトに関連するメニューを要求した時に、そのオブジェクトの横に表示されるメニュー。 これには、オブジェクトの現在のコンテキストに適した選択項目が含まれています。

ポップアップ・メニュー・パーツ. オブジェクトに関して要求した時に、そのオブジェクトの横に表示されるパーツ (そのパーツがユーザー・インターフェース上のオブジェクトに追加されている場合)。パーツ・パレットまたはパーツ・カタログ内のポップアップ・メニュー・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

[マ行]

マウス. キーボードを使用しないで画面上のポインターを位置付けるために使用し、1 つまたは複数のプッシュボタンを持つ装置。実行する選択項目または機能を選択するために使用したり、画面上で操作を実行するために使用します。たとえば、ある場所から別の場所に線をドラッグしたり描いたりします。

マウス・ポインター. *カーソル*の同義語。

マウス・ボタン. 選択項目を選択したり、アクションを開始したり、あるいはポインターでオブジェクトを操作するために使用するマウス上の機構。 *左マウス・ボタン* および*右マウス・ボタン* も参照してください。

右マウス・ボタン. デフォルトとして、操作のために使用するマウス上の右のボタン。

メイン・ウィンドウ. *プライマリー・ウィンドウ*を参照してください。

メイン・ソース・セクション. VARPG プログラムでは、メイン・ソース・セクションには、モジュールのすべてのグローバル定義が含まれています。コンポーネントの場合には、このセクションには、アクションおよびユーザー・サブルーチンも含まれます。

メイン・プロシージャー. メイン・プロシージャーは、プログラムの入り口プロシージャーとして指定でき、最初に呼び出された時に制御を受け取るサブプロシージャーです。 メイン・プロシージャーは EXE の作成時にしか作成されません。 *EXE モジュール*を参照してください。

メッセージ. (1) ユーザーによって要求された情報ではなく、予期しないイベントに応答してプロダクトによって表示されるか、あるいは何か不都合なことが起こった時に表示される情報。 (2) ある人またはプログラムから別の人またはプログラムに対して送信される通信。

メッセージ・サブファイル・パーツ. プログラム・ロジックに提供されている事前定義メッセージまたはテキストを表示できるパーツ。

メッセージ・ファイル. アプリケーション・メッセージが入っているファイル。このファイルは、ビルド・プロセス時にメッセージ・ソース・ファイルから作成されます。*ビルド*も参照してください。

メディア・パーツ. プログラムに、サウンド・ファイルおよびビデオ・ファイルを処理する機能を与えるパーツ。

メディア・パネル・パーツ. ユーザーに他のパーツ全般の制御を与えるために使用するパーツ。たとえば、メディア・パネル・パーツは、メディア・パーツのボリュームを制御するために使用することができます。

メニュー. オブジェクトに適用することのできる選択項目のリスト。 メニューには、一定のコンテキストの下では選択できない選択項目も含まれることがあります。 これらの選択項目はぼかし表示されます。

メニュー項目パーツ. メニュー上のグラフィカルまたはテキスト項目であるパーツ。ユーザーは何らかの方法でオブジェクトを処理するためのメニュー項目を選択します。

メニュー・バー・パーツ. ウィンドウの上部近くで、タイトル・バーの下で、ウィンドウの他の部分より上にある領域で、他のメニューにアクセスできる選択項目が含まれています。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内のメニュー・バー・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

［ヤ行］

ユーザー定義パーツ. ユーザーがカスタマイズした 1 つまたは複数のパーツから成るパーツであり、再使用のためにこれをパーツ・パレットまたはパーツ・カタログに保管します。パレットまたはカタログの中にある時には、他の VARPG パーツと同様に、このパーツをポイントおよびクリックして設計ウィンドウに入れることができます。

ユーティリティー DLL. *NOMAIN モジュール* を参照してください。

［ラ行］

ラジオ・ボタン・パーツ. 横にテキストが付いている円。 ラジオ・ボタンは、固定した選択項目のセットから 1 つの選択項目を選択できる組み合わせです。選択項目が選択されると、円の一部が塗りつぶされます。 パーツ・パレットまたはパーツ・カタログ内のラジオ・ボタン・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

リスト・ボックス・パーツ. ユーザーが選択できるスクロール可能な選択項目が入っている制御。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内のリスト・ボックス・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

略号の選択. 選択項目の簡略記号を入力することにより、その選択項目を選択する選択手法。

リンク・イベント. ソース・パーツの状態が変更された時に、常にターゲット・パーツが受け取るイベント。

例外. (1) プログラム言語において、実行時に起こる可能性のある異常な状態、通常の実行順序から逸脱する原因となる可能性のある異常な状態であり、それを定義したり、引き起こしたり、認識、無視、および処理するための機能がプログラム言語中に存在しています。(I) (2) VisualAge RPG において、ユーザーが要求したアクションが予定通りに完了するのを妨げたり、妨げる可能性のあるイベントまたは状態。 例外は、プロダクトがユーザーの入力を解釈できない時に起こります。

［ワ行］

ワークエリア. ユーザーのタスクにしたがってオブジェクトを編成するために使用する領域。ユーザーがワークエリアをクローズすると、このワークエリアにあるオブジェクトからオープンされたすべてのウィンドウがワークプレースから除去されます。

ワークステーション. ユーザーが作業することのできる装置。*プログラマブル・ワークステーション*も参照してください。

ワークプレース. 画面全体を埋める領域で、ユーザー・インターフェースを構成するすべてのオブジェクトを保持する領域。

枠のサイズ変更. マウスまたはキーボードを使用してパーツ (またはパーツのセット) をサイズ変更するために選択するパーツ (またはパーツのセット) の回りの枠またはフレーム。

［数字］

1 バイト文字セット (SBCS). 各文字が 1 バイトのコードで表される文字セット。 *2 バイト文字セット (DBCS)* と対比。

2 バイト文字セット (DBCS). 各文字が 2 バイトで表される文字のセット。 256 個のコード・ポイントでは表しきれない記号が入っている、日本語、中国語、および韓国語、などの言語は 2 バイト文字セットを必要とします。各文字が 2 バイトを必要とするので、DBCS 文字の入力、表示、および印刷では、 DBCS をサポートするハードウェアおよびソフトウェアが必要です。システムは次の 4 つの 2 バイト文字セットをサポートします: 日本語、韓国語、中国語 (簡体字)、および中国語 (繁体字)。 *1 バイト文字セット (SBCS)* と対比。

A

activeX パーツ. プロジェクトに ActiveX 制御オブジェクトを追加するパーツ。これにより、VARPG アプリケーションは、その属性にアクセスして、イベントをモニターすることができます。

API. アプリケーション・プログラミング・インターフェース。

ASCII (情報交換用米国標準コード). 7 ビット (パリティー検査を組み込む場合は 8 ビット) のコード化文字から成るコード化文字セットを使用する標準コードであり、データ処理システム、データ通信システム、および関連機器の間で情報交換用に使用されます。 ASCII セットは制御文字とグラフィック文字から成ります。(A)

B

BMP. ビットマップ・ファイルのファイル拡張子。

C

CONFIG.SYS. DOS、OS/2、または Windows オペレーティング・システムのブート・ドライブのルート・ディレクトリーにある構成ファイル。 この中には、ハードウェアおよびソフトウェアをインストールし実行するために必要な情報が入っています。

CUA アーキテクチャー. 共通ユーザー・アクセス・アーキテクチャー。

D

DBCS. 2 バイト文字セット。

DDE. 動的データ交換。

DDE 会話. DDE クライアントと DDE サーバーの間のデータの交換。 *コールド・リンク会話*および*ホット・リンク会話*も参照してください。

DDE クライアント. DDE 会話を開始するアプリケーション。 *DDE サーバー*と対比。*DDE クライアント・パーツ*、*DDE 会話*も参照してください。

DDE クライアント・パーツ. 動的データ交換 (DDE) プロトコルをサポートする他のアプリケーション (スプレッドシート・アプリケーションなど) とデータを交換するために使用するパーツ。

DDE サーバー. データを別の DDE 可能アプリケーションに提供するアプリケーション。 *DDE クライアント*と対比。*DDE 会話*も参照してください。

DLL. ダイナミック・リンク・ライブラリー。

E

EBCDIC. 拡張 2 進化 10 進コード。 256 個の 8 ビット文字のコード化文字セット。

EXE. 実行可能ファイルの拡張子。

EXE モジュール. EXE モジュールはメイン・プロシージャーとサブプロシージャーから成ります。これは、 制御仕様に EXE キーワードがある場合に作成されます。すべてのサブルーチン (BEGSR) はプロシージャーに対してローカルでなければなりません。 EXE には、名前がソース・ファイルの名前と一致するプロシージャーが入っていなければなりません。これが EXE のメイン入り口点であり、すなわちメイン・プロシージャーとなります。

G

GUI Designer. パーツをパーツ・パレットから設計ウィンドウにドラッグ／ドロップすることによってインターフェースを作成するのに使用する一組のツール。

I

ICO. アイコン・ファイルのファイル拡張子。

INI. OS/2 または Windows オペレーティング・システムにおいて、アプリケーションから別のアプリケーションを呼び出すために保持する必要のあるアプリケーション固有の情報が入っているファイルのファイル拡張子。

IPF. 情報表示機能

J

JAR ファイル (.jar). Java では、Java ARchive の省略形。多くのファイルを 1 つのファイルに集めるために使用されるファイル形式。

Java. リモート・オブジェクト間の対話をサポートする移植可能解釈コードのためのオブジェクト指向プログラミング言語。Java は、Sun Microsystems, Incorporated によって開発され、明示されたものです。

Java 2 Software Development Kit (J2SDK). Sun Microsystems 社が Java 開発者のために配布するソフトウェア。このソフトウェアには、Java インタープリター、Java クラス、および Java 開発ツールが組み込まれています。開発ツールには、コンパイラー、デバッガー、ディスアセンブラー、AppletViewer、スタブ・ファイル生成プログラム、および文書生成プログラムが組み込まれています。

Java Bean パーツ. VARPG アプリケーションが Sun Microsystem の JavaBeans にアクセスできるようにするパーツ。

Java 仮想マシン (JVM). Java 実行時環境 (JRE) の一部で、Java バイトコードの解釈を受け持ちます。

Java 実行時環境 (JRE). JRE を再配布したいエンド・ユーザーおよび開発者のための Java Developer Kit のサブセット。JRE は、Java 仮想マシン、Java コア・クラス、およびサポート・ファイルからなります。

Java データベース接続性 (JDBC). Java と広範囲のデータベースの間のデータベースから独立した接続のための業界標準。JDBC は、SQL 基本データベース・アクセスのための呼び出しレベルのアプリケーション・プログラミング・インターフェース (API) を提供します。

Java ネイティブ・インターフェース (JNI). Java 仮想マシン (JVM) の内部で実行する Java コードが他のプログラム言語で書かれた機能と相互に関連し合って機能できるようにするプログラミング・インターフェース。

JavaBeans. Java では、移植可能で、プラットフォームから独立した再使用可能コンポーネント・モデルです。

M

MID. MIDI ファイルのファイル拡張子。

MIDI ファイル. 電子楽器ディジタル・インターフェース・ファイル。

N

NOMAIN モジュール. サブプロシージャーだけが入っているモジュール。 この中には、アクションまたはスタンドアロン・ユーザー・サブルーチンはありません。 NOMAIN モジュールは、制御仕様の中に NOMAIN キーワードがある場合に作成されます。

O

odbc/jdbc パーツ. VAPRG アプリケーションが Windows ODBC API または Sun Microsystem 社の JDBC API をサポートするデータベース・ファイルにアクセスして、それを処理できるようにするパーツ。

P

PWS. プログラマブル・ワークステーション。

S

SBCS. 1 バイト文字セット。

secure sockets layer (SSL). Netscape Communications Corp. および RSA Data Security, Inc によって開発された一般的なセキュリティー機構。SSL によって、クライアントはサーバーを認証できるようになり、すべてのデータおよび要求が暗号化できるようになります。SSL によって保護されたセキュア・サーバーの URL は、http ではなく、https で始まっています。

SSL. Secure sockets layer。

W

WAV. ウェーブ・ファイルのファイル拡張子。

［特殊文字］

***component パーツ.** コンポーネントの『パーツ表現』であるパーツ。 各コンポーネントごとに 1 つの *component パーツが自動的に作成され、それは目には見えません。

[隠す] ボタン. タイトル・バーにあるボタンで、ユーザーがウィンドウをクローズしないでワークプレースからそのウィンドウを除去する場合にクリックします。 ウィンドウが隠されると、ウィンドウの状態 (ウィンドウ・リストに表示される) が変わります。*最大化ボタン*および*最小化ボタン*と対比。

参考文献

WebSphere Development Studio Client と関連したトピックの追加情報については、以下の IBM 資料を参照してください。

WebSphere Development Studio Client マニュアル:

- *WebSphere Development Studio Client for iSeries ご使用に際して*, SD88-5054-03。WebSphere Development Studio Client for iSeries についての情報を提供し、各種コンポーネントの概要、それらがどのように関連して機能するか、およびそれを使用することによる業務上の利点を示します。

VisualAge RPG マニュアル:

- *VisualAge for RPG プログラミング*, SC88-5607-05 には、VisualAge RPG を使用してアプリケーションを作成する場合の具体的な情報が記載されています。これには、アプリケーション開発サイクルのすべての段階で実行する必要があるステップが、設計からパッケージおよび分配まで説明されています。VARPG アプリケーション開発の概念と工程を明らかにするために、プログラミング例が収録されています。
- *ADTS/CS VisualAge for RPG パーツ解説書*, SD88-5040-03 には、VARPG の各パーツ、パーツ属性、パーツ・イベント、およびイベント属性が説明されています。本書は、VisualAge RPG を使用してアプリケーションを開発しているすべての方々の参照資料です。
- *VisualAge for RPG WINDOWS 版 言語解説書*, SC88-5608-04 には、VARPG 言語およびコンパイラーに関する参照情報が提供されます。
- *Java for RPG プログラマー*。Java 言語 (および RPG IV) を RPG 言語と比較しながら紹介しています。これは、Java を使って様々なことを経験するための最初のステップとして格好なものです。これには、Java および VisualAge for Java に関する対話式 CD チュートリアルも MINDQ 別に含まれています。
- *Experience RPG IV Tutorial*。RPG IV および ILE について肩の凝らない段階的方法によって学習するための対話式 CD チュートリアルです。この資料は、この興味深い RPG の新規バージョンでの実技経験を得る上で役立つ質問および演習が記載されたハンドブックです。
- VisualAge RPG ユーザーの関心を引く IBM 以外の資料に *VisualAge for RPG by Example* があります。

インターネットにアクセスできる場合には、以下の Web サイトから iSeries および AS/400e についての最新の情報および資料を入手できます。

`http://www.ibm.com/eserver/iseries/infocenter`

Application Development Manager マニュアル:

- *AS/400 V3 適用業務開発ツール (ADT) セット /400 適用業務開発管理 /400 入門および計画の手引き*, GC88-5245-00 には、Application Development Manager 機能を効率的に使用するために必要な基本概念および計画が説明されています。
- *DTS/ Application Development Manager User's Guide*, SC09-2133-02 には、Application Development Manager 機能に対して定義されるプロジェクトを作成し管理する方法が説明されています。

- *AS/400 アドバンスト・シリーズ V3.6 適用業務開発ツール (ADT) セット OS/400 用 適用業務開発管理 OS/400 用 自習書*, SC88-5417-00。Application Development Manager 機能を使用する実践的な体験を得ることができます。この手引書には、ステップ順に練習することによって、Application Development Manager 機能の使用法が説明されています。
- *ADTS/400: Application Development Manager API Reference*, SC09-2180-00。アプリケーション・プログラマーが Application Development Manager 機能に対する固有のインターフェースを作成する方法が説明されています。

情報表示機能マニュアル:

- *Information Presentation Facility Programming Guide* G25H-7110 には、情報表示機能 (IPF) を構成するエレメントが説明されています。IPF は、オンライン文書機能およびオンライン・ヘルプ機能の設計および開発をサポートするツールです。

SQL マニュアル:

- *IBM SQL Reference Version 2* (SC26-8416) Volume 2 には、以下に示すものの機能比較があります。
  - DB2
  - SQL/DS™
  - DB2/400™
  - DB2/6000™
  - IBM SQL
  - ISO-ANSI (SQL92E)
  - X/Open™ (XPG4-SQL).
- *DB2 Universal Database Administration Guide* S10J-8157 には、DB2 製品の使用および管理に必要な情報が示されています。
- *DB2 Universal Database Embedded SQL Programming Guide* S10J-8158 には、DB2 クライアント／サーバーのファミリー・サーバー (DB2 または DB2/400) にアクセスするアプリケーション・プログラムの設計およびコーディングの方法が説明されています。これには、構造化照会言語 (SQL) の使用およびアプリケーションでの API 呼び出しに関する詳細情報が説明されています。

Printed in Japan

SC88-5608-04

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