Natural for Windows

システム関数

バージョン 6.3.3

October 2008

Natural

This document applies to Natural バージョン 6.3.3 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

目次

1 システム関数

このドキュメントでは、特定のステートメントで使用できる Natural の各種 「組み込み」 関数について説明します。

Note: Natural for Windows/UNIX バージョン 6.2、Natural for OpenVMS 6.3、および Natural for Mainframes バージョン 4.2 以降、名前のコンフリクト（例えば、既存アプリケーションのユーザー定義変数とのコンフリクト）を避けるために、新しいシステム関数の名前はすべてアスタリスク（*）で始まっています。

このドキュメントは次の項目で構成されています。

	処理ループで使用する Natural システム関数	プログラムのループコンテキストで使用できる Natural システム関数について説明します。
	算術関数	算術処理ステートメントおよび論理条件基準でサポートされているシステム関数について説明します。
	その他の関数	他の各種システム関数について説明します。フィールド識別用のシステム関数、CALL ステートメントで呼び出された Natural 以外のプログラムからリターンコードを受け取るシステム関数、「不正にソートされた」 文字を変換するためのシステム関数、フィールドの最小値／最大値を取得するシステム関数、ステートメントの *operand1* を大文字／小文字に変換するためのシステム関数、*operand1*から先頭の空白または末尾の空白を削除するためのシステム関数が該当します。

以下の項目も参照してください。

- 『*プログラミングガイド*』の「*システム関数*」
- 『*プログラミングガイド*』の「*システム変数とシステム関数の例*」

2 処理ループで使用する Natural システム関数

このchapterでは、プログラムのループコンテキストで使用できる Natural システム関数について説明します。

次のトピックが含まれています。

処理ループでのシステム関数の使用

次のトピックについて説明します。

- 指定／評価
- SORT GIVE ステートメントでの使用
- AVER、NAVER、SUM、TOTAL での桁あふれ
- ステートメント参照 (*r*)

指定／評価

Natural システム関数は次のステートメントに指定できます。

- 割り当ておよび算術ステートメント：
  - MOVE
  - ASSIGN
  - COMPUTE
  - ADD
  - SUBTRACT
  - MULTIPLY
  - DIVIDE
- 入出力ステートメント：
  - DISPLAY
  - PRINT
  - WRITE

上記のステートメントを次のステートメントブロック内で使用する場合に、システム関数を指定できます。

- AT BREAK
- AT END OF DATA
- AT END OF PAGE

つまり、FIND、READ、HISTOGRAM、SORT、または READ WORK FILE のすべての処理ループに指定できます。

AT END OF PAGE ステートメントでシステム関数を使用する場合、対応する DISPLAY ステートメントに GIVE SYSTEM FUNCTIONS 節を指定する必要があります。

WHERE 節で排除されたレコードは、システム関数で評価されません。

FIND、READ、HISTOGRAM、SORT ステートメントで開始した、異なるレベルの処理ループのデータベースフィールドでシステム関数が評価される場合、その値は常にループ階層の位置に従って処理されます。 例えば、新しいデータ値が外側のループに対して取得されていた場合、値はそのループに対してだけ処理されます。

ユーザー定義変数でシステム関数が評価される場合、レポーティングモードの処理は、ユーザー定義変数が定義されたループ階層の位置に依存します。 処理ループの開始前に定義されているユーザー定義変数の場合、AT BREAK、AT END OF DATA、AT END OF PAGE ステートメントが定義されたループ内のシステム関数に対して評価されます。 ユーザー定義変数が処理ループ内で定義されている場合、処理中のデータベースフィールドと同様に処理されます。

ユーザー定義変数に対するシステム関数評価の参照を選択する場合、値を処理するループを示すためにユーザー定義変数にループ参照を付けて指定することをお勧めします。 ループ参照はステートメントラベルまたはソースコード行番号として指定できます。

SORT GIVE ステートメントでの使用

システム関数は、SORT ステートメントの GIVE 節で評価されたときにも参照できます。

SORT GIVE ステートメントで評価されたシステム関数を参照するには、システム関数の名前の先頭にアスタリスク（*）を付ける必要があります。

AVER、NAVER、SUM、TOTAL での桁あふれ

システム関数 AVER、NAVER、SUM、TOTAL を適用するフィールドには、桁あふれを防止するために十分な長さ（デフォルトまたはユーザー指定）が必要です。桁あふれが起きると、エラーメッセージが発行されます。

通常、長さはシステム関数を適用するフィールドの長さと同じです。この長さでは不十分の場合、SORT GIVE ステートメントの NL オプションを使用して、次のように出力長を拡張する必要があります。

```
SUM(field)(NL=nn)
```

この場合、出力長が拡張されるだけでなく、フィールドの内部長も拡張されます。

ステートメント参照 (*r*)

ステートメント参照はシステム関数に対しても使用できます。『*プログラミングガイド*』の「*ユーザー定義変数*」セクションの「*表記(r)を使用したデータベースフィールドの参照*」も参照してください。

ステートメントラベルまたはソースコード行番号 (*r*) を使用して、指定したフィールドに対してシステム関数を評価する処理ループを指定できます。

AVER(*r*)(*field*)

フォーマット／長さ：	フィールドと同じ。 例外：フィールドフォーマットが N の場合、AVER(*field*) のフォーマットは P（桁数はフィールドと同じ）。

AVER に指定されたフィールドのすべての値の平均値を持ちます。 AVER は、AVER 要求時の条件が真になるたびに更新されます。

COUNT(*r*)(*field*)

フォーマット／長さ：	P7

COUNT の置かれた処理ループを通過するたびに 1 ずつ増加します。 COUNT の増加には、COUNT に指定されたフィールドの値は関係しません。

MAX(*r*)(*field*)

フォーマット／長さ：	フィールドと同じ。

MAX に指定されたフィールドの最大値を持ちます。 MAX の置かれた処理ループが実行されるたびに（必要に応じて）更新されます。

MIN(*r*)(*field*)

フォーマット／長さ：	フィールドと同じ。

MIN に指定されたフィールドの最小値を持ちます。 MIN の置かれた処理ループが実行されるたびに（必要に応じて）更新されます。

NAVER(*r*)(*field*)

フォーマット／長さ：	フィールドと同じ。 例外：フィールドフォーマットがNの場合、NAVER*(field)*のフォーマットはP（桁数はフィールドと同じ）。

NAVER に指定されたフィールドのすべての値（空値を除く）の平均値を持ちます。 NAVER は、NAVER 要求時の条件が真になるたびに更新されます。

NCOUNT(*r*)(*field*)

フォーマット／長さ：	P7

NCOUNT の置かれた処理ループを通過するたびに、NCOUNT に指定されたフィールドが空値でないときに 1 ずつ増加します。

NCOUNT の結果が配列になるかスカラー値になるかは、その引数（フィールド）によって決まります。 結果のオカレンスの数はフィールドと同じです。

NMIN(*r*)(*field*)

フォーマット／長さ：	フィールドと同じ。

NMIN に指定されたフィールドの最小値（空値を除く）を持ちます。 NMIN の置かれた処理ループが実行されるたびに（必要に応じて）更新されます。

OLD(*r*)(*field*)

フォーマット／長さ：	フィールドと同じ。

AT BREAK 条件で指定されたコントロールブレイクの前、またはエンドオブページ条件やエンドオブデータ条件の前に、OLD で指定されたフィールドの値を持ちます。

SUM(*r*)(*field*)

フォーマット／長さ：	フィールドと同じ。 例外：フィールドフォーマットが N の場合、SUM(*field*) のフォーマットは P（桁数はフィールドと同じ）。

SUM に指定されたフィールドのすべての値の合計を持ちます。 SUM の置かれた処理ループが実行されるたびに更新されます。 SUM を AT BREAK 条件に続けて指定すると、値のブレイクごとにリセットされます。 ブレイク間の値だけが加算されます。

TOTAL(*r*)(*field*)

フォーマット／長さ：	フィールドと同じ。 例外：フィールドフォーマットがNの場合、TOTAL*(field)*のフォーマットはP（桁数はフィールドと同じ）。

TOTAL の置かれているすべてのオープン処理ループで、TOTAL に指定されたフィールドのすべての値の合計を持ちます。

例

以降で、例を示します。

- 例 1 - Natural システム関数 OLD、MIN、AVER、MAX、SUM、COUNT を使用した AT BREAK ステートメント
- 例 2 - Natural システム関数 AVER を使用した AT BREAK ステートメント
- 例 3 - システム関数 MAX、MIN、AVER を使用した AT END OF DATA ステートメント
- 例 4 - システム関数 AVER を使用した AT END OF PAGE ステートメント

例 1 - Natural システム関数 OLD、MIN、AVER、MAX、SUM、COUNT を使用した AT BREAK ステートメント

```
** Example 'ATBEX3': AT BREAK  (with Natural system functions)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 3
READ EMPLOY-VIEW LOGICAL BY CITY = 'SALT LAKE CITY'
  DISPLAY NOTITLE CITY NAME 'SALARY' SALARY(1) 'CURRENCY' CURR-CODE(1)
  /*
  AT BREAK OF CITY
    WRITE /  OLD(CITY) (EM=X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X)
      31T '   MINIMUM:' MIN(SALARY(1))  CURR-CODE(1) /
      31T '   AVERAGE:' AVER(SALARY(1)) CURR-CODE(1) /
      31T '   MAXIMUM:' MAX(SALARY(1))  CURR-CODE(1) /
```

```
      31T '       SUM:' SUM(SALARY(1))  CURR-CODE(1) /
      35T COUNT(SALARY(1)) 'RECORDS FOUND' /
  END-BREAK
  /*
  AT END OF DATA
    WRITE 22T 'TOTAL (ALL RECORDS):'
              T*SALARY TOTAL(SALARY(1))  CURR-CODE(1)
  END-ENDDATA
END-READ
*
END
```

プログラム ATBEX3 の出力：

```
        CITY                   NAME            SALARY   CURRENCY
-------------------- -------------------- ---------- --------

SALT LAKE CITY       ANDERSON                    50000 USD
SALT LAKE CITY       SAMUELSON                   24000 USD

S A L T   L A K E   C I T Y      MINIMUM:        24000 USD
                                 AVERAGE:        37000 USD
                                 MAXIMUM:        50000 USD
                                     SUM:        74000 USD
                                         2 RECORDS FOUND

SAN DIEGO            GEE                         60000 USD

S A N   D I E G O                MINIMUM:        60000 USD
                                 AVERAGE:        60000 USD
                                 MAXIMUM:        60000 USD
                                     SUM:        60000 USD
                                         1 RECORDS FOUND

                     TOTAL (ALL RECORDS):       134000 USD
```

例 2 - Natural システム関数 AVER を使用した AT BREAK ステートメント

```
** Example 'ATBEX4': AT BREAK  (with Natural system functions)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 SALARY    (2)
*
1 #INC-SALARY (P11)
END-DEFINE
*
```

```
LIMIT 4
EMPL. READ EMPLOY-VIEW BY CITY STARTING FROM 'ALBU'
  COMPUTE #INC-SALARY = SALARY (1) + SALARY (2)
  DISPLAY NAME CITY SALARY (1:2) 'CUMULATIVE' #INC-SALARY
  SKIP 1
  /*
  AT BREAK CITY
    WRITE NOTITLE
      'AVERAGE:'            T*SALARY (1)  AVER(SALARY(1)) /
      'AVERAGE CUMULATIVE:' T*#INC-SALARY AVER(EMPL.) (#INC-SALARY)
  END-BREAK
END-READ
*
END
```

プログラム ATBEX4 の出力：

```
        NAME                 CITY           ANNUAL    CUMULATIVE
                                            SALARY
-------------------- -------------------- ---------- ------------

HAMMOND              ALBUQUERQUE               22000        42200
                                               20200

ROLLING              ALBUQUERQUE               34000        65200
                                               31200

FREEMAN              ALBUQUERQUE               34000        65200
                                               31200

LINCOLN              ALBUQUERQUE               41000        78700
                                               37700

AVERAGE:                                       32750
AVERAGE CUMULATIVE:                                         62825
```

例 3 - システム関数 MAX、MIN、AVER を使用した AT END OF DATA ステートメント

```
** Example 'AEDEX1S': AT END OF DATA
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
```

```
LIMIT 5
EMP. FIND EMPLOY-VIEW WITH CITY = 'STUTTGART'
  IF NO RECORDS FOUND
    ENTER
  END-NOREC
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
          SALARY (1) CURR-CODE (1)
  /*
  AT END OF DATA
    IF *COUNTER (EMP.) = 0
      WRITE 'NO RECORDS FOUND'
      ESCAPE BOTTOM
    END-IF
    WRITE NOTITLE / 'SALARY STATISTICS:'
                  / 7X 'MAXIMUM:' MAX(SALARY(1))  CURR-CODE (1)
                  / 7X 'MINIMUM:' MIN(SALARY(1))  CURR-CODE (1)
                  / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDDATA
  /*
END-FIND
*
END
```

プログラム AEDEX1S の出力：

```
PERSONNEL         NAME              FIRST-NAME        ANNUAL   CURRENCY
   ID                                                 SALARY     CODE
--------- -------------------- -------------------- ---------- --------

11100328  BERGHAUS             ROSE                    70800 DM
11100329  BARTHEL              PETER                   42000 DM
11300313  AECKERLE             SUSANNE                 55200 DM
11300316  KANTE                GABRIELE                61200 DM
11500304  KLUGE                ELKE                    49200 DM

SALARY STATISTICS:
       MAXIMUM:      70800 DM
```

```
       MINIMUM:      42000 DM
       AVERAGE:      55680 DM
```

例 4 - システム関数 AVER を使用した AT END OF PAGE ステートメント

```
** Example 'AEPEX1S': AT END OF PAGE (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
FORMAT PS=10
LIMIT 10
READ EMPLOY-VIEW BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE GIVE SYSTEM FUNCTIONS
          NAME JOB-TITLE 'SALARY' SALARY(1) CURR-CODE (1)
  /*
  AT END OF PAGE
    WRITE / 28T 'AVERAGE SALARY: ...' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDPAGE
END-READ
*
END
```

プログラム AEPEX1S の出力：

```
        NAME                 CURRENT            SALARY   CURRENCY
                            POSITION                       CODE
-------------------- ------------------------ ---------- --------

CREMER               ANALYST                       34000 USD
MARKUSH              TRAINEE                       22000 USD
GEE                  MANAGER                       39500 USD
KUNEY                DBA                           40200 USD
NEEDHAM              PROGRAMMER                    32500 USD
JACKSON              PROGRAMMER                    33000 USD

                           AVERAGE SALARY: ...     33533 USD
```

3 算術関数

次の算術関数は、算術処理ステートメント（ADD、COMPUTE、DIVIDE、MULTIPLY、SUBTRACT）および論理条件基準でサポートされています。

関数	フォーマット／長さ	説明
ABS*(field)*	*field* と同じ	*field* の絶対値。
ATN*(field)*	F8（*）	*field* のアークタンジェント。
COS*(field)*	F8（*）	*field* のコサイン。
EXP*(field)*	F8（*）	基数e、指数 *field* による累乗（すなわち e^{field}、eはオイラー数）。
FRAC*(field)*	*field* と同じ	*field* の小数部分。
INT*(field)*	*field* と同じ	*field* の整数部分。
LOG*(field)*	F8（*）	*field* の自然対数。
SGN*(field)*	*field* と同じ	*field* の符号（-1、0、+1）。
SIN*(field)*	F8（*）	*field* のサイン（正弦）。
SQRT*(field)*	(*)	*field* の平方根。 負の値の引数フィールドは正の値として扱われます。
TAN*(field)*	F8（*）	*field* のタンジェント。
VAL*(field)*	ターゲットフィールドと同じ	英数字 *field* から数値を抽出します。 *field* の内容は、数値の英数字（コードページまたは Unicode）文字表現にする必要があります。 *field*の先頭または末尾の空白は無視されます。小数点や先行符号文字は処理されます。 ターゲットフィールドの長さが不足している場合、小数は切り捨てられます（『*プログラミングガイド*』の「*演算割り当てのルール*」セクションの「*フィールドの切り捨てと切り上げ*」も参照）。

* これらの関数は次のように評価されます。

- 引数は、フォーマット／長さが F8 に変換された後、計算のためにオペレーティングシステムに渡されます。
- オペレーティングシステムによって返される結果のフォーマット／長さは F8 で、その後ターゲットフォーマットに変換されます。

算術関数（VAL を除く）で使用できる *field* は、定数またはスカラーです。フォーマットは数値（N）、パック 10 進（P）、整数（I）、または浮動小数点（F）にする必要があります。

VAL 関数に使用できる *field* は、定数、スカラー、または配列です。フォーマットは英数字にする必要があります。

算術関数の例：

```
** Example 'MATHEX': Mathematical functions
************************************************************************
DEFINE DATA LOCAL
1 #A     (N2.1) INIT <10>
1 #B     (N2.1) INIT <-6.3>
1 #C     (N2.1) INIT <0>
1 #LOGA  (N2.6)
1 #SQRTA (N2.6)
1 #TANA  (N2.6)
1 #ABS   (N2.1)
1 #FRAC  (N2.1)
1 #INT   (N2.1)
1 #SGN   (N1)
END-DEFINE
*
COMPUTE #LOGA = LOG(#A)
WRITE NOTITLE '=' #A 5X 'LOG'         40T #LOGA
*
COMPUTE #SQRTA = SQRT(#A)
WRITE         '=' #A 5X 'SQUARE ROOT' 40T #SQRTA
*
COMPUTE #TANA  = TAN(#A)
WRITE         '=' #A 5X 'TANGENT'     40T #TANA
*
COMPUTE #ABS   = ABS(#B)
WRITE     //  '=' #B 5X 'ABSOLUTE'    40T #ABS
*
COMPUTE #FRAC  = FRAC(#B)
WRITE         '=' #B 5X 'FRACTIONAL'  40T #FRAC
*
COMPUTE #INT   = INT(#B)
WRITE         '=' #B 5X 'INTEGER'     40T #INT
*
COMPUTE #SGN   = SGN(#A)
WRITE      // '=' #A 5X 'SIGN'        40T #SGN
*
```

```
COMPUTE #SGN   = SGN(#B)
WRITE         '=' #B 5X 'SIGN'        40T #SGN
*
COMPUTE #SGN   = SGN(#C)
WRITE         '=' #C 5X 'SIGN'        40T #SGN
*
END
```

プログラム MATHEX の出力：

```
#A:  10.0     LOG                       2.302585
#A:  10.0     SQUARE ROOT               3.162277
#A:  10.0     TANGENT                   0.648360


#B:  -6.3     ABSOLUTE                   6.3
#B:  -6.3     FRACTIONAL                -0.3
#B:  -6.3     INTEGER                   -6.0


#A:  10.0     SIGN                       1
#B:  -6.3     SIGN                      -1
#C:   0.0     SIGN                       0
```

4 その他の関数

次のトピックについて説明します。

- **POS -** フィールド **ID** 関数
- **RET -** リターンコード関数
- **SORTKEY** ソートキー関数
- ***MINVAL/*MAXVAL -** 最小値／最大値の評価
- ***TRANSLATE -** 小文字／大文字に変換
- ***TRIM -** 先頭または末尾の空白のいずれか、あるいは両方を削除

5 POS - フィールド ID 関数

フォーマット／長さ：	I4

システム関数 POS(*field-name*) の内容は、システム関数で指定される名前を持つフィールドの内部識別子になります。

POS(*field-name*) を使用すると、マップ内の位置と関係なく、特定のフィールドを特定できます。 これは、マップ内のフィールドのシーケンスと数が変更されても、POS(*field-name*) は引き続き同じフィールドを一意に識別できることを意味します。 これにより、例えば、プログラムロジックに依存しているとフィールドに MARK を付ける場合、必要なのは 1 つの REINPUT ステートメントのみになります。

例：

```
DECIDE ON FIRST VALUE OF ...
   VALUE ...
      COMPUTE #FIELDX = POS(FIELD1)
   VALUE ...
      COMPUTE #FIELDX = POS(FIELD2)
   ...
END-DECIDE
...
REINPUT ... MARK #FIELDX
```

POS で指定されたフィールドが配列の場合、POS(FIELDX(5)) のように特定のオカレンスを指定する必要があります。 POS を配列範囲に適用することはできません。

POS および *CURS-FIELD

システム関数 POS(*field-name*) は、カーソルが現在位置づけられているフィールドに応じて特定の機能を実行するために、Natural システム変数 *CURS-FIELD と組み合わせて使用できます。

*CURS-FIELDは、カーソルが現在位置づけられているフィールドの内部識別子を持ちます。これは単体では使用できず、POS(*field-name*) と組み合わせて使用する必要があります。 これらを使用して、現在カーソルが特定のフィールドに置かれているかどうかを確認し、その状況に応じて処理を実行できます。

例：

```
IF *CURS-FIELD = POS(FIELDX)
   MOVE *CURS-FIELD TO #FIELDY
END-IF
...
REINPUT ... MARK #FIELDY
```

Notes:

1. *CURS-FIELD および POS(*field-name*) の値は、フィールドの内部識別子としてのみ機能し、算術演算に使用することはできません。
2. 配列の次元のオカレンス数が EXPAND、RESIZE、または REDUCE ステートメントを使用して変更された後には、X-array（最低でも 1 次元の最低でも 1 つの境界が拡張可能として定義されている配列）のオカレンスに対して POS(*field-name*) から返される値は変わることがあります。
3. Natural RPC：*CURS-FIELD および POS(*field-name*) がコンテキスト変数を参照する場合、結果の情報は同じ会話内でのみ使用できます。

6 RET - リターンコード関数

フォーマット／長さ：	I4

システム関数 RET(*program-name*) を使用して、CALL ステートメントで呼び出された Natural 以外のプログラムからのリターンコードを受け取ることができます。

RET(*program-name*) は、IF ステートメントおよび算術ステートメント ADD、COMPUTE、DIVIDE、MULTIPLY、SUBTRACT で使用できます。

例：

```
DEFINE DATA LOCAL
1 #RETURN (I4)
...
END-DEFINE
...
...
CALL 'PROG1'
IF RET('PROG1') > #RETURN
   WRITE 'ERROR OCCURRED IN PROGRAM 1'
END-IF
...
```

7 SORTKEY ソートキー関数

SORTKEY (*character-string*)

このシステム関数は、「不正にソートされた」 文字（または文字の組み合わせ）を、ソートプログラムまたはデータベースシステムでアルファベット順に 「正しくソートされる」 他の文字（または文字の組み合わせ）に変換するために使用します。

フォーマット／長さ：	A253

国によっては、ソートプログラムまたはデータベースシステムによって正確なアルファベット順にソートされない文字（または文字の組み合わせ）が言語に含まれます。コンピュータで使用される文字セットの文字のシーケンスが必ずしもアルファベット順に相当するとは限らないためです。

例えば、スペイン語の "CH" は、ソートプログラムやデータベースシステムでは 2 文字の文字列とみなされ、"CG" と "CI" の間にソートされます。しかし、実際、スペイン語では、これは "C" と "D" の間に属する 1 文字となります。

または、要求に反して、小文字と大文字がソート順で同等に扱われないこともあります。文字は（数字の前にソートされることを希望しているにもかかわらず）数字の後にソートされることもあります。あるいは、特殊文字（例：ダブル名のハイフン）によって予期しないソート順が生じることもあります。

このような場合、システム関数 SORTKEY(*character-string*) を使用できます。 SORTKEY で計算された値はソート条件にしか使用できず、エンドユーザーとのやり取りには元の値が使用されます。

COMPUTE ステートメントおよび論理条件の算術演算オペランドとして SORTKEY 関数を使用できます。

character-string として、英数字の定数や変数、または英数字配列の単一オカレンスを指定できます。

Natural プログラムに SORTKEY 関数を指定すると、ユーザー出口 NATUSK*nn*（*nn*：現在の言語コード、つまりシステム変数 *LANGUAGE の現在の値）が呼び出されます。

このユーザー出口は、標準 CALL インターフェイスを提供する任意のプログラミング言語で記述できます。 SORTKEY に指定された *character-string* は、そのユーザー出口に渡されます。 ユーザー出口は、この文字列内の「不正にソートされた」文字を「正しくソートされた」文字に変換するようにプログラミングされている必要があります。 続けて、変換された文字列が Natural プログラムで使用され、さらに処理が実行されます。

外部プログラムの一般的な呼び出し規則については、CALL ステートメントの説明を参照してください。

ユーザー出口の呼び出し規則の詳細については、「ユーザー*出口*」を参照してください。

例：

```
DEFINE DATA LOCAL
1 CUST VIEW OF CUSTOMERFILE
  2 NAME
  2 SORTNAME
END-DEFINE
...
*LANGUAGE := 4
...
REPEAT
  INPUT NAME
  SORTNAME := SORTKEY(NAME)
  STORE CUST
  END TRANSACTION
  ...
END-REPEAT
...
READ CUST BY SORTNAME
  DISPLAY NAME
END-READ
...
```

上記の例では、INPUT ステートメントの繰り返しの実行で、"Sanchez"、"Sandino"、"Sancinto" の値が入力されるものと仮定しています。

SORTNAME への SORTKEY(NAME) の割り当てで、ユーザー出口 NATUSK04 が呼び出されます。 このユーザー出口は、最初にすべての小文字を大文字に変換し、次に文字の組み合わせ "CH" を "C*x*" に変換します。この場合、*x* は使用する文字セットの最終文字、つまり 16 進の H'FF'（この最終文字は出力不可能な文字とみなされる）に対応します。

「元」 の名前（NAME）は、希望するソート（SORTNAME）に使用する変換済みの名前とともに保存されます。 ファイルを読み込むには、SORTNAME を使用します。 DISPLAY ステートメントは、次のように正しいスペイン語のアルファベット順に名前を出力します。

```
Sancinto
Sanchez
Sandino
```

8 *MINVAL/*MAXVAL - 最小値／最大値の評価

```
{ *MINVAL } ([(IR=result-format/length)] operand,...)
{ *MAXVAL }
```

フォーマット／長さ：フォーマットおよび長さは、IR 節を使用して明示的に指定するか、後述のフォーマット／長さの変換規則表を使用して自動的に評価できます。

このchapterでは、次のトピックについて説明します。

関数

*MINVAL/*MAXVAL システム関数は、指定されたすべてのオペランド値の最小値／最大値を評価します。 結果は常にスカラー値です。 オペランドとして配列が指定された場合、すべての配列フィールドの最小／最大が評価されます。

*MINVAL/*MAXVAL システム関数は、*MINVAL/*MAXVAL が許可されるステートメントの任意の位置にオペランドとして指定できます。ただし、ターゲット変数が予期される位置に *MINVAL/*MAXVAL を使用しないでください。

*MINVAL/*MAXVAL は、システム関数でネストすることはできません。

英字データまたはバイナリデータを引数として使用するとき、データが同一であれば（例えば、*MINVAL('AB','AB')）、結果は最小／最大の長さを持つ引数になります。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット												ステートメント参照	ダイナミック定義
operand	C	S	A	G		A	U	N	P	I	F	B	D	T					
																		可	不可

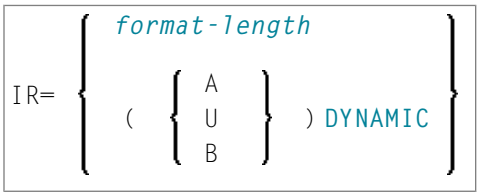
構文要素の説明：

***MINVAL**	指定されたすべてのオペランド値の最小値を評価します。
***MAXVAL**	指定されたすべてのオペランド値の最大値を評価します。
operand	*MINVAL/*MAXVAL システム関数で最小値／最大値が評価されるオペランドです。
result-format-length	結果のフォーマット／長さを明示的に指定するための中間結果節です。 下記の「*IR 節*」を参照してください。

IR 節

IR（中間結果）節は、*MINVAL/*MAXVAL システム関数全体の *result-format/length* を明示的に指定するために使用できます。

```
IR=result-format/length
```



結果のフォーマット／長さの有効な組み合わせについては、後述のフォーマット／長さの変換規則表を参照してください。

構文要素の説明：

format-length	コンパイラは関数全体の結果のフォーマット／長さを決定しようとします。 コンパイラが、精度に損失のないことが保証される方法でフォーマット／長さを決定できない場合、プログラマが IR オペランド拡張を使用して *format-length* を設定する必要があります。
A、U、または B	フォーマット：ダイナミック変数用の英数字、Unicode、またはバイナリです。
DYNAMIC	固定フォーマット／長さを指定する代わりに、ダイナミック長の英数字フォーマット、Unicode フォーマット、またはバイナリフォーマットを指定できます。

例：

```
DEFINE DATA LOCAL
1 #RESULTI       (I4)
1 #RESULTA       (A20)
1 #RESULTADYN    (A) DYNAMIC
1 #A(I4)         CONST <1234>
1 #B(A20)        CONST <H'30313233'> /* '0123' stored
1 #C(I2/1:3)     CONST <2000, 2100, 2200>
END-DEFINE
*
#RESULTA     := *MAXVAL((IR=A20)         #A, #B)     /*no error, I4->A20 is allowed!
#RESULTADYN  := *MAXVAL((IR=(A)DYNAMIC)  #A, #B)     /*result is (A) dynamic
/* #RESULTI  := *MAXVAL((IR=I4)         #A, #B)     /*compiler error, because conv.
A20->I4 is not allowed!
#RESULTI     := *MAXVAL((IR=I4)          #A, #C(*))  /*maximum of the array is
evaluated
```

```
DISPLAY #RESULTA #RESULTADYN (AL=10) #RESULTI
END
```

結果のフォーマット／長さの変換規則表

*MINVAL/*MAXVAL システム関数全体の結果のフォーマット／長さは2とおりの方法で定義できます。

- 結果のフォーマット／長さの明示的な指定
- 結果のフォーマット／長さの暗黙的な指定

結果のフォーマット／長さの明示的な指定

*MINVAL/*MAXVAL システム関数全体の結果のフォーマット／長さは、IR 節で指定できます。 指定されたすべてのオペランドは、精度が損失ないのであれば、この結果のフォーマット／長さに変換されます。 その後、変換されたすべてのオペランドの最小／最大が評価され、評価されたフォーマット／長さを持つ単一スカラー値がシステム関数全体の結果として設定されます。

結果のフォーマット／長さの暗黙的な指定

*MINVAL/*MAXVAL システム関数内で IR 節を使用しない場合、結果のフォーマット／長さは、*MINVAL/*MAXVAL システム関数内の引数として指定されたすべてのオペランドのフォーマット／長さに関連して評価されます。 各オペランドのフォーマット／長さが取得され、引数リストの次のオペランドのフォーマット／長さと結合されます。 そして、フォーマット／長さの変換規則表を使用して2つの単一オペランドの結果のフォーマット／長さが評価されます。

フォーマット／長さの変換規則表は、2つの異なる表に分割されます。下記の2つの表に示されていない組み合わせはすべて無効であり、*MINVAL/*MAXVAL システム関数の引数リスト内に適用しないでください。キーワード"FLF"は、他の方法では精度の損失が生じる可能性があるため、結果のフォーマット／長さを定義するために IR 節を使用する必要があることを示します。

表 **1**

2つの異なるオペランドの数値の組み合わせをすべて示しています。

	第2オペランド					
	フォーマット／長さ	I1	I2	I4	P*a.b*、N*a.b*	F4、F8
第1オペ	I1	I1	I2	I4	Pmax(3,*a*)。*b*	F8
	I2	I2	I2	I4	Pmax(5,*a*)。*b*	F8
	I4	I4	I4	I4	Pmax(10,*a*)。*b*	F8

	第 2 オペランド					
	フォーマット／長さ	I1	I2	I4	P*a*.*b*、N*a*.*b*	F4、F8
ランド	P*x*.*y*、N*x*.*y*	Pmax(3,*x*)。*y*	Pmax(5,*x*)。*y*	Pmax(10,*x*)。*y*	max(*x*,*a*) + max(*y*,*b*) <= 29 の場合 Pmax(*x*,*a*).max(*y*,*b*) else FLF	*y*=0 および *x* <=15 の場合 F8 else FLF
	F4、F8	F8	F8	F8	*b*=0 および *a* <=15 の場合 F8 else FLF	F8

凡例：

FLF	フォーマット／長さ宣言が強制されます。結果のフォーマットは IR節を使用して指定する必要があります。
I*x*	フォーマット／長さは整数です。 *x* では、整数の値を保存するために使用されるバイト数を指定します。
F*x*	フォーマット／長さは浮動小数点です。 *x* では、浮動小数点の値を保存するために使用されるバイト数を指定します。
P*x*.*y* **P***a*,*b*	対応する小数点の前の桁数（*x*,*a*）および精度（*y*,*b*）を持つパックフォーマットです。
Nx.y **Na,b**	対応する小数点の前の桁数（*x*,*a*）および精度（*y*,*b*）を持つ数値フォーマットです。
Pmax(*c*,*d***).***e*	生成されるフォーマットはパックされています。長さは後続の情報で評価されます。小数点の前の桁数は、*c* および *d* の最大値です。 精度の値は、*e* です。
Pmax(*c*,*d***).max(***e*,*f***)**	生成されるフォーマットはパックされています。長さは後続の情報で評価されます。小数点の前の桁数は、*c* および *d* の最大値です。 精度の値は、*e* および *f* の最大値です。

表 **2**

*MINVAL/*MAXVAL システム関数オペランドに使用できるその他のフォーマットと長さをすべて示しています。

	第 2 オペランド					
	フォーマット／長さ	D	T	A*a*、A ダイナミック	B*a*、B ダイナミック	U*a*、U ダイナミック
第1オペランド	D	D	T	NA	NA	NA
	T	T	T	NA	NA	NA
	A*x*、A ダイナミック	NA	NA	A ダイナミック	B ダイナミック	U ダイナミック

	第 2 オペランド					
	フォーマット／長さ	D	T	A*a*、A ダイナミック	B*a*、B ダイナミック	U*a*、U ダイナミック
	B*x*、B ダイナミック	NA	NA	B ダイナミック	B ダイナミック	U ダイナミック
	U*x*、U ダイナミック	NA	NA	U ダイナミック	U ダイナミック	U ダイナミック

凡例：

NA	この組み合わせは許可されていません。
D	日付フォーマットです。
T	時刻フォーマットです。
B*x*、B*a*	長さ *x*，*a* のバイナリフォーマットです。
A*x*、A*a*	長さ *x*，*a* の英数字フォーマットです。
U*x*、U*a*	長さ *x*，*a* の Unicode フォーマットです。
B ダイナミック	ダイナミック長のバイナリフォーマットです。
A ダイナミック	ダイナミック長の英数字フォーマットです。
U ダイナミック	ダイナミック長の Unicode フォーマットです。

result-format-length の評価

コンパイラは、上記の規則に基づいて、オペランドのペアを考慮し、各ペアの中間結果を計算することで、ソースオペランドを処理できます。1 番目のペアは第 1 と第 2 オペランド、中間結果の 2 番目のペアと第 3 オペランド、などから構成されます。すべてのオペランドが処理された後、最後の結果は、最小／最大を評価するために全オペランドの比較に使用されるフォーマットと長さの比較を示します。 フォーマット／長さの評価にこの方法を使用するとき、オペランド *format-lengths* は任意の順序で示すことができます。

例：

```
DEFINE DATA LOCAL
1 A (I2)      INIT <34>
1 B (P4.2)    INIT <1234.56>
1 C (N4.4)    INIT <12.6789>
1 D (I1)      INIT <100>
1 E (I4/1:3)  INIT <32, 6745, 456>
1 #RES-MIN (P10.7)
1 #RES-MAX (P10.7)
END-DEFINE
*
MOVE *MINVAL(A, B, C, D, E(*)) TO #RES-MIN
MOVE *MAXVAL(A, B, C, D, E(*)) TO #RES-MAX
DISPLAY #RES-MIN #RES-MAX
END
```

出力：

```
         #RES-MIN                      #RES-MAX
--------------------------  --------------------------

                12.6789000                  6745.0000000
```

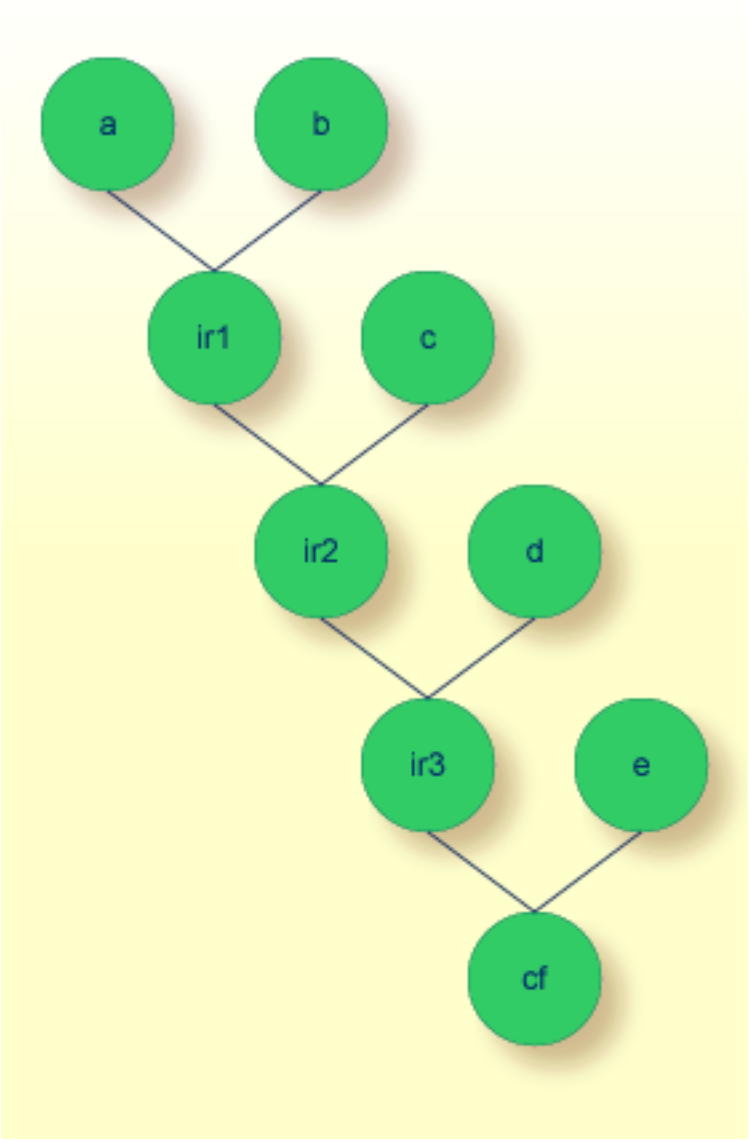
次の表は、例のフォーマット／長さを自動的に評価する単一ステップを示しています。 それは全ステップの中間結果（ir）および *result-format/length* として使用される比較フォーマット／長さ（cf）を示します。

評価順序	第1オペランド名	第1オペランドまたは中間結果のフォーマット／長さ	第2オペランド名	第2オペランドまたは中間結果のフォーマット／長さ	中間結果（ir）のフォーマット／長さ
1.	A	I2	B	P4.2	ir1 = P5.2
2.	ir1	P5.2	C	N4.4	ir2 = P5.4
3.	ir2	P5.4	D	I1	ir3 = P5.4
4.	ir3	P5.4	E	I4	**cf = P10.4**

ランタイムに、すべてのオペランドは cf のフォーマット／長さに変換されます。 そして、変換されたすべての値が比較され、対応する最小／最大が評価されます。

フォーマット／長さの評価順序

下図は、フォーマットと長さが評価される順序を表しています。



凡例：

ir1、ir2、ir3	中間結果 1、2、3 です。
cf	フォーマットと長さの比較です。

9 *TRANSLATE - 小文字／大文字に変換

```
*TRANSLATE (operand [ ,{ LOWER } ] )
                       { UPPER }
```

フォーマット／長さ：*operand* と同じ。

このchapterでは、次のトピックについて説明します。

関数

Natural システム関数 *TRANSLATE は、英字またはバイナリのオペランドの文字を大文字または小文字に変換します。 *operand* の内容は、修正されていません。

*TRANSLATE は、フォーマット A または B の *operand* が許可されるステートメントの任意の位置に *operand* として指定できます。 ただし、ターゲット変数が予期される位置に *TRANSLATE を使用しないでください。 *TRANSLATE は、システム関数でネストすることはできません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット										ステートメント参照	ダイナミック定義
operand	C	S	A			A	U	B								可	不可

構文要素の説明：

LOWER	2 番目の引数としてキーワード LOWER を使用すると、*operand* を小文字に変換します。 `*TRANSLATE (..., LOWER)`
UPPER	2 番目の引数としてキーワード UPPER を使用すると、*operand* を大文字に変換します。 `*TRANSLATE (..., UPPER)`

例

```
DEFINE DATA LOCAL
1 #SRC  (A)DYNAMIC INIT <'aBcDeFg !§$%&/()=?'>
1 #DEST (A)DYNAMIC
END-DEFINE
*
PRINT 'Source string to be translated:.....' #SRC
*
MOVE *TRANSLATE(#SRC, UPPER) TO #DEST
PRINT 'Source string translated into upper case:' #DEST
*
MOVE *TRANSLATE(#SRC, LOWER) TO #DEST
PRINT 'Source string translated into lower case:' #DEST
END
```

出力：

```
Source string to be translated:..... aBcDeFg !§$%&/()=?

Source string translated into upper case: ABCDEFG !§$%&/()=?

Source string translated into lower case: abcdefg !§$%&/()=?
```

10 *TRIM - 先頭または末尾の空白のいずれか、あるいは両方を削除

```
*TRIM(operand [ ,{ LEADING  } ] )
                 { TRAILING }
```

フォーマット／長さ：*operand*（A または B）／DYNAMIC と同じ。

このchapterでは、次のトピックについて説明します。

関数

Natural システム関数 *TRIM は、英数字またはバイナリの文字列から先頭または末尾の空白（あるいは両方）をすべて削除します。 *operand* の内容は、修正されていません。 ダイナミック変数を *operand* として使用すると、この変数の長さは結果に従って調整されます。

*TRIM システム関数は、フォーマット A または B の *operand* が許可されるステートメントの任意の位置に *operand* として指定できます。

ただし、ターゲット変数が予期される位置に *TRIM を使用しないでください。 *TRIM は、システム関数でネストすることはできません。

構文説明

オペランド定義テーブル：

オペランド	構文要素					フォーマット										ステートメント参照	ダイナミック定義
operand	C	S	A			A	U	B								可	不可

構文要素の説明：

LEADING	2 番目の引数としてキーワード LEADING を使用すると、*operand* から先頭の空白を削除します。 `*TRIM(..., LEADING)`

TRAILING	2 番目の引数としてキーワード TRAILING を使用すると、*operand* から末尾の空白を削除します。 `*TRIM(..., TRAILING)`
operand	オペランドの後にキーワードを指定しない場合： 2 番目の引数としてキーワードを使用しないとき、TRIM は *operand* から先頭および末尾の空白を削除します。 `*TRIM(...)` **注意:** 末尾の空白を削除するとき、スタティック変数またはダイナミック変数として使用しているかに応じて、*operand* の動作が異なります。 *operand* がスタティック変数であれば、*TRIM を使用して末尾の空白を削除することはできません。 これは、スタティック変数の場合、変数メモリの残りの末尾位置はスペース文字で満たされるためです。ただし、ダイナミック変数を使用しているときには、末尾の空白の削除は可能です。

例

以降で、例を示します。

- 例 1 - 英数字引数の使用
- 例 2 - バイナリ引数の使用

例 1 - 英数字引数の使用

```
DEFINE DATA LOCAL
/*********************************
/* STATIC VARIABLE DEFINITIONS
/*********************************
1 #SRC  (A15) INIT <'  ab  CD  '>
1 #DEST (A15)

/* FOR PRINT OUT WITH DELIMITERS
1 #SRC-PRN  (A20)
1 #DEST-PRN (A20)

/*********************************
/* DYNAMIC VARIABLE DEFINITIONS
/*********************************
1 #DYN-SRC (A)DYNAMIC INIT <'  ab  CD  '>
1 #DYN-DEST (A)DYNAMIC

/* FOR PRINT OUT WITH DELIMITERS
```

```
1 #DYN-SRC-PRN  (A)DYNAMIC
1 #DYN-DEST-PRN (A)DYNAMIC

END-DEFINE

PRINT 'static variable definition:'
PRINT '---------------------------'
COMPRESS FULL ':' #SRC ':' TO #SRC-PRN LEAVING NO SPACE
PRINT ' '
PRINT ' 123456789012345      123456789012345'

MOVE *TRIM(#SRC, LEADING) TO #DEST
COMPRESS FULL ':' #DEST ':' TO #DEST-PRN LEAVING NO SPACE
DISPLAY #SRC-PRN #DEST-PRN '*TRIM(#SRC, LEADING)'

MOVE *TRIM(#SRC, TRAILING) TO #DEST
COMPRESS FULL ':' #DEST ':' TO #DEST-PRN LEAVING NO SPACE
DISPLAY #SRC-PRN #DEST-PRN '*TRIM(#SRC, TRAILING)'

MOVE *TRIM(#SRC) TO #DEST
COMPRESS FULL ':' #DEST ':' TO #DEST-PRN LEAVING NO SPACE
DISPLAY #SRC-PRN #DEST-PRN '*TRIM(#SRC)'

PRINT ' '
PRINT 'dynamic variable definition:'
PRINT '----------------------------'
COMPRESS FULL ':' #DYN-SRC ':' TO #DYN-SRC-PRN LEAVING NO SPACE
PRINT ' '
PRINT ' 1234567890           12345678'

MOVE *TRIM(#DYN-SRC, LEADING) TO #DYN-DEST
COMPRESS FULL ':' #DYN-DEST ':' TO #DYN-DEST-PRN LEAVING NO SPACE
DISPLAY (AL=20) #DYN-SRC-PRN #DYN-DEST-PRN '*TRIM(#SRC, LEADING)'

MOVE *TRIM(#DYN-SRC, TRAILING) TO #DYN-DEST
COMPRESS FULL ':' #DYN-DEST ':' TO #DYN-DEST-PRN LEAVING NO SPACE
DISPLAY (AL=20) #DYN-SRC-PRN #DYN-DEST-PRN '*TRIM(#SRC, TRAILING)'

MOVE *TRIM(#DYN-SRC) TO #DYN-DEST
COMPRESS FULL ':' #DYN-DEST ':' TO #DYN-DEST-PRN LEAVING NO SPACE
DISPLAY (AL=20) #DYN-SRC-PRN #DYN-DEST-PRN '*TRIM(#SRC)'

PRINT ' '
PRINT '":" := delimiter character to show the start and ending of a string!'
END
```

例 1 の出力：

```
      #SRC-PRN             #DEST-PRN
-------------------- ----------------------

static variable definition:
---------------------------

 123456789012345          123456789012345
:  ab  CD       :       :ab  CD         :     *TRIM(#SRC, LEADING)
:  ab  CD       :       :  ab  CD       :     *TRIM(#SRC, TRAILING)
:  ab  CD       :       :ab  CD         :     *TRIM(#SRC)

dynamic variable definition:
---------------------------

 1234567890               12345678
:  ab  CD  :            :ab  CD  :            *TRIM(#SRC, LEADING)
:  ab  CD  :            :  ab  CD:            *TRIM(#SRC, TRAILING)
:  ab  CD  :            :ab  CD:              *TRIM(#SRC)

':' := delimiter character to show the start and ending of a string!
```

例 2 - バイナリ引数の使用

```
DEFINE DATA LOCAL
/*******************************
/* STATIC VARIABLE DEFINITIONS
/*******************************
1 #SRC  (B10) INIT <H'2020FFFF2020FFFF2020'>
1 #DEST (B10)

/*******************************
/* DYNAMIC VARIABLE DEFINITIONS
/*******************************
1 #DYN-SRC (B)DYNAMIC INIT <H'2020FFFF2020FFFF2020'>
1 #DYN-DEST (B)DYNAMIC
END-DEFINE

FORMAT LS=100

PRINT 'static variable definition:
PRINT '---------------------------'
MOVE *TRIM(#SRC, LEADING) TO #DEST
PRINT #SRC #DEST '*TRIM(#SRC, LEADING)'

MOVE *TRIM(#SRC, TRAILING) TO #DEST
PRINT #SRC #DEST '*TRIM(#SRC, TRAILING)'

MOVE *TRIM(#SRC) TO #DEST
PRINT #SRC #DEST '*TRIM(#SRC)'
```

```
PRINT ' '
PRINT 'dynamic variable definition:'
PRINT '----------------------------'

MOVE *TRIM(#DYN-SRC, LEADING) TO #DYN-DEST
PRINT #DYN-SRC #DYN-DEST '    *TRIM(#SRC, LEADING)'

MOVE *TRIM(#DYN-SRC, TRAILING) TO #DYN-DEST
PRINT #DYN-SRC #DYN-DEST '    *TRIM(#SRC, TRAILING)'

MOVE *TRIM(#DYN-SRC) TO #DYN-DEST
PRINT #DYN-SRC #DYN-DEST '        *TRIM(#SRC)'

PRINT ' '

PRINT 'hex."20" := space character'
END
```

例 2 の出力：

```
static variable definition:
----------------------------

2020FFFF2020FFFF2020 0000FFFF2020FFFF2020     *TRIM(#src, leading)
2020FFFF2020FFFF2020 00002020FFFF2020FFFF     *TRIM(#src, trailing)
2020FFFF2020FFFF2020 00000000FFFF2020FFFF     *TRIM(#src)

dynamic variable definition:
----------------------------

2020FFFF2020FFFF2020 FFFF2020FFFF2020         *TRIM(#src, leading)
2020FFFF2020FFFF2020 2020FFFF2020FFFF         *TRIM(#src, trailing)
2020FFFF2020FFFF2020 FFFF2020FFFF             *TRIM(#src)

hex.'20' := space character
```

索引

シンボル

A

C

E

F

I

L

M

N

O

P

R

S

T

V

さ

し