第１０章　プログラム間連絡機能とプログラム構造

指導目標

本章では、プログラム間での制御の移行と、データの受け渡しについて指導する。

大規模システムでは１個のプログラム（実行単位）を開発するのに、複数の要員で分割して開発する。
分割手段にはいろいろあるが、本章では、「主プログラム」と「サブ・プログラム」の概念で原始プログラムを分割した場合の要点を学習する。

プログラム分割において、「共通処理」をサブ・プログラムにする場合などの注意点や問題点を指導し、効果的な使い方についても指導する。
また、新しく規定された「組込み関数」の利点がよく理解されるよう、要点を指導する。

本章で学習する命令文は、次のとおり。

- ＣＡＬＬ文
- ＥＸＩＴ　ＰＲＯＧＲＡＭ文
- ＣＡＮＣＥＬ文
- ＦＵＮＣＴＩＯＮ（組込み関数）

内容のあらまし

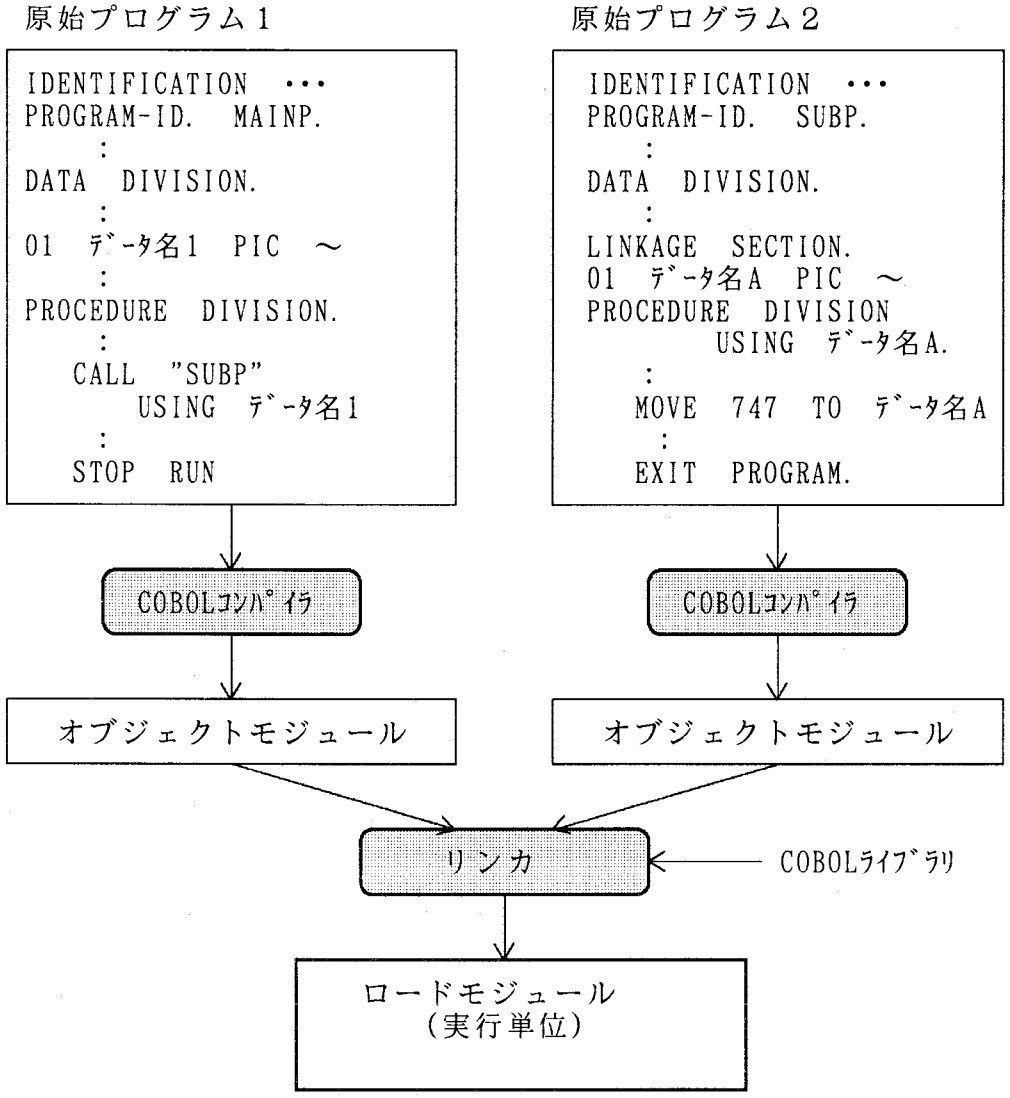
内　　容	説　明	議　論	机上実習	計算機実習
ﾌﾟﾛｸﾞﾗﾑ間連絡	制御の移行方法を説明する。ﾃﾞｰﾀの受け渡しの概念を説明する。	複数のﾌﾟﾛｸﾞﾗﾑを１個のﾓｼﾞｭｰﾙにするにはどんな情報が必要なのか議論する。	簡単な例題から複雑な例題までのﾌﾟﾛｸﾞﾗﾑ分割を想定してﾌﾟﾛｸﾞﾗﾑ構造をよく理解させる。	ｺｰﾃﾞｨﾝｸﾞだけでなく実際に実行して使いこなせるようにする。
CALL	ﾊﾟﾗﾒｰﾀの定義、受け渡しを重点に説明する。	PERFORMとの違いを議論する。		ｻﾌﾞﾌﾟﾛｸﾞﾗﾑの深さを3ﾚﾍﾞﾙ以上にしてﾊﾟﾗﾒｰﾀの受け渡しと制御の関係を実習させる。
EXIT　PROGRAM	制御の戻り方法について説明する。			
CANCEL	ﾌﾟﾛｸﾞﾗﾑ初期状態について説明する。	CANCEL機能の応用例を考えてもらい議論する。		CANCEL機能と主記憶装置上の関連を実習する。
ﾌﾟﾛｸﾞﾗﾑの入れ子	COBOL利用者語の有効範囲や制御構造について説明する。	入れ子の利点とは何かを議論する。	入れ子の深さを変えてﾃﾞｰﾀ名の有効範囲を考えさせる	ｺﾝﾊﾟｲﾙと実行でGLOBAL句やEXTERNAL句の機能を実習する。
組込み関数	組込み関数の特徴と機能について説明する。	組込み関数を使用した場合と自分でｺｰﾃﾞｨﾝｸﾞした場合の違いを議論する。	他の言語の組込み関数との機能比較を行う。	組込み関数を使いこなせるように実習を行う。

第１０章　プログラム間連絡機能とプログラム構造

ＣＯＢＯＬのプログラム間連絡機能とは、プログラムとプログラムで情報の連絡が行える機能であり、次の機能をいう。

①プログラムから別のプログラムへ制御を移す。
②プログラム間でデータの受け渡しができる。
　受け渡しの手段としては、次の方法がある。
　・ＣＡＬＬ文のパラメータ（引数）
　・データの共用（ＧＬＯＢＡＬ／ＥＸＴＥＲＮＡＬ句の指定）
　・ファイルの共用（ＧＬＯＢＡＬ／ＥＸＴＥＲＮＡＬ句の指定）

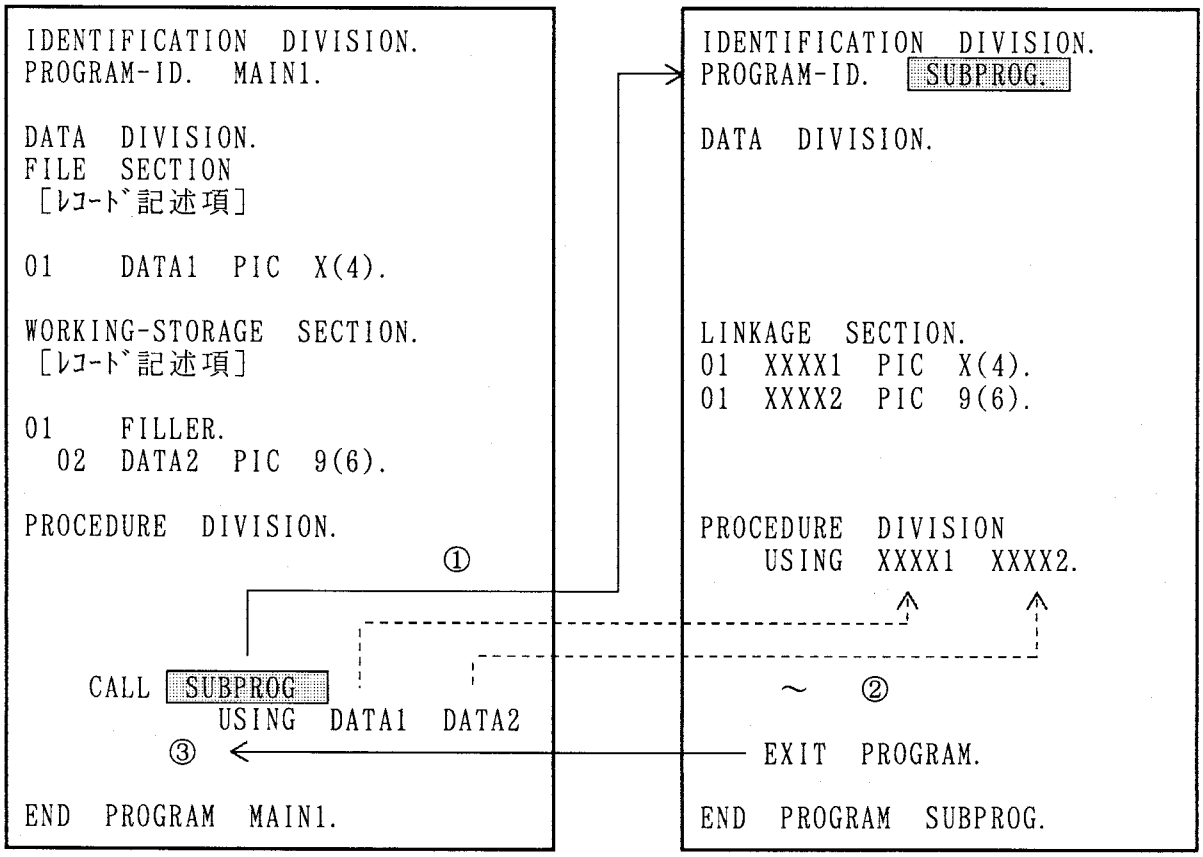
原始プログラムと実行単位の関係は、次のとおり。



オペレーティング・システムのタスク管理（ジョブ管理）機能には、複数のオブジェクトモジュールを１個のロードモジュールにしないで、個々のロードモジュールを作成して置き、実行時に主記憶装置にロードしてプログラム間連絡を行うこともできる。（例えば「ダイナミック・リンク」など）

１０．１　プログラム間連絡

プログラム間連絡機能は、一つのプログラムから他のプログラムへ制御を移し、プログラム間でデータの受け渡しができる機能である。



①ＣＡＬＬ文で別プログラムの名前を指定して制御を別プログラムへ移す。
　プログラム間で必要なデータはＵＳＩＮＧのパラメータ指定で受け渡す。
②制御が渡された別プログラムは、手続き部の最初の命令文から実行する。
　ＣＡＬＬのパラメータは、手続き部のＵＳＩＮＧで受け取る。その受け取るパラメータはデータ部のLINKAGE SECTIONに定義されている。
③ＣＡＬＬで呼ばれたプログラムが「EXIT PROGRAM」を実行すると、制御はＣＡＬＬの後の命令文に戻る。

パラメータの対応関係は、次のとおり。

```
呼ぶﾌﾟﾛｸﾞﾗﾑ
    CALL  "ﾌﾟﾛｸﾞﾗﾑ名"    USING  ﾃﾞｰﾀ名1  ﾃﾞｰﾀ名2  ﾃﾞｰﾀ名3  ・・・
                                  ◇       ◇       ◇
呼ばれるﾌﾟﾛｸﾞﾗﾑ
    PROCEDURE DIVISION USING  ﾃﾞｰﾀ名A  ﾃﾞｰﾀ名B  ﾃﾞｰﾀ名C  ・・・
```

１０．１．１　ＣＡＬＬ

ＣＡＬＬ文は、制御を別のプログラムに移す。
パラメータは、参照（REFERENCE）で受け渡す場合と、内容（CONTENT）を受け渡すことができる。

```
CALL { 一意名1 } [USING { BY REFERENCE {一意名2} ・・・ } ・・・]
     { 定数1   }       { BY CONTENT   {一意名2} ・・・ }
      [ON OVERFLOW 無条件文1]
 [END-CALL]

CALL { 一意名1 } [USING { BY REFERENCE {一意名2} ・・・ } ・・・]
     { 定数1   }       { BY CONTENT   {一意名2} ・・・ }
      [ON EXCEPTION 無条件文1]
      [NOT ON EXCEPTION 無条件文2]
 [END-CALL]
```

ＣＡＬＬ文で指定したプログラムが実行できない場合に、制御は「ON　OVERFLOW」または「ON　EXCEPTION」の無条件文1に移る。

呼ぶプログラム名を指定する定数1は文字定数であり、一意名1で指定する場合は、英数字項目として定義する。

```
DATA  DIVISION.
    :
01  NNNN    PICTURE  X(8).
    :
PROCEDURE  DIVISION.
    :
    CALL  "ﾌﾟﾛｸﾞﾗﾑ1"  USING  ～          ←定数指定のﾌﾟﾛｸﾞﾗﾑ名
    :
    IF  要求  =  "月次"  THEN
        MOVE  "ﾌﾟﾛｸﾞﾗﾑ2"  TO  NNNN
    ELSE
        MOVE  "ﾌﾟﾛｸﾞﾗﾑ3"  TO  NNNN
    END-IF
    CALL  NNNN  USING  ～               ←ﾃﾞｰﾀ名指定の場合は、
    :                                     実行時にﾌﾟﾛｸﾞﾗﾑ名が決まる
```

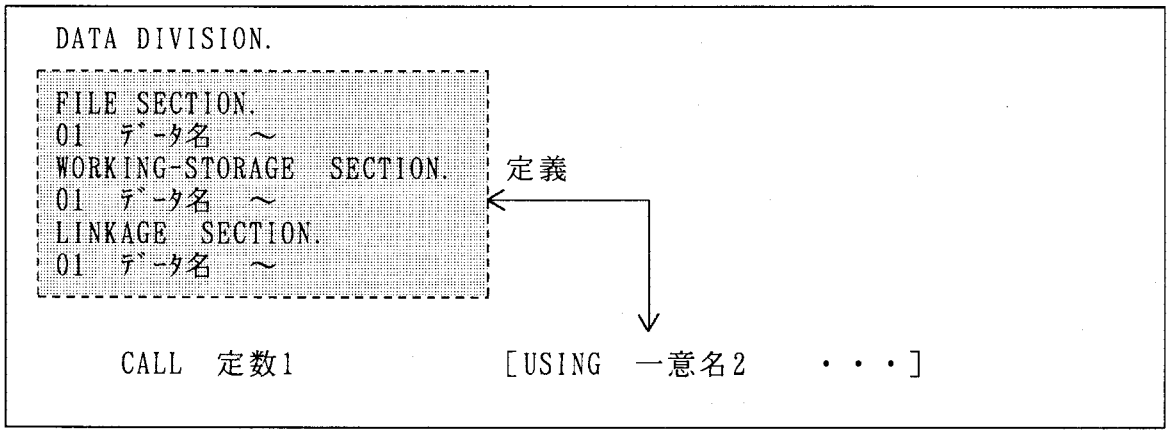
一意名2のパラメータに定数は指定できない。

```
ＣＯＢＯＬの場合
      CALL  "ﾌﾟﾛｸﾞﾗﾑ"  USING  "HELLOW"  12345.  ←  ﾊﾟﾗﾒｰﾀの誤り

Ｃ言語の場合
      関数名("HELLOW",  12345)                  ←  定数が指定できる
```

（１）　パラメータの定義関係

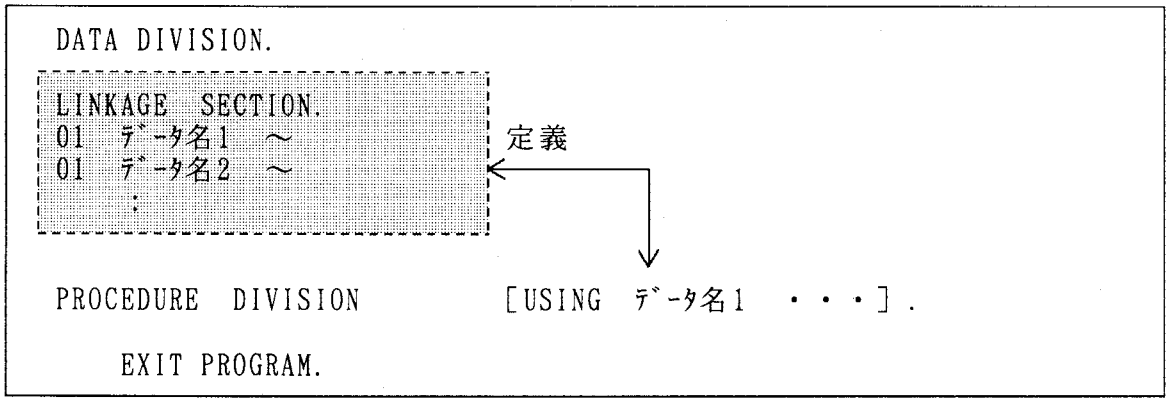
ＣＡＬＬのパラメータは、ファイル節、作業場所節、通信節、連絡節で定義された０１または、７７のレベル番号のデータ項目である。

①　呼ぶプログラムのパラメータの定義



呼び出されるプログラムで受け取るパラメータの属性はLINKAGE　SECTIONで定義する。

②　呼ばれるプログラムのパラメータ定義



（２）　連絡節（LINKAGE　SECTION）

連絡節は、呼ばれるプログラムに記述され、呼ぶプログラムと呼ばれるプログラムのデータの受け渡しに使用される。一般形式は、次のとおり。

```
LINKAGE  SECTION.
{ 独立データ記述項 }
{ レコード記述項   } ...
```

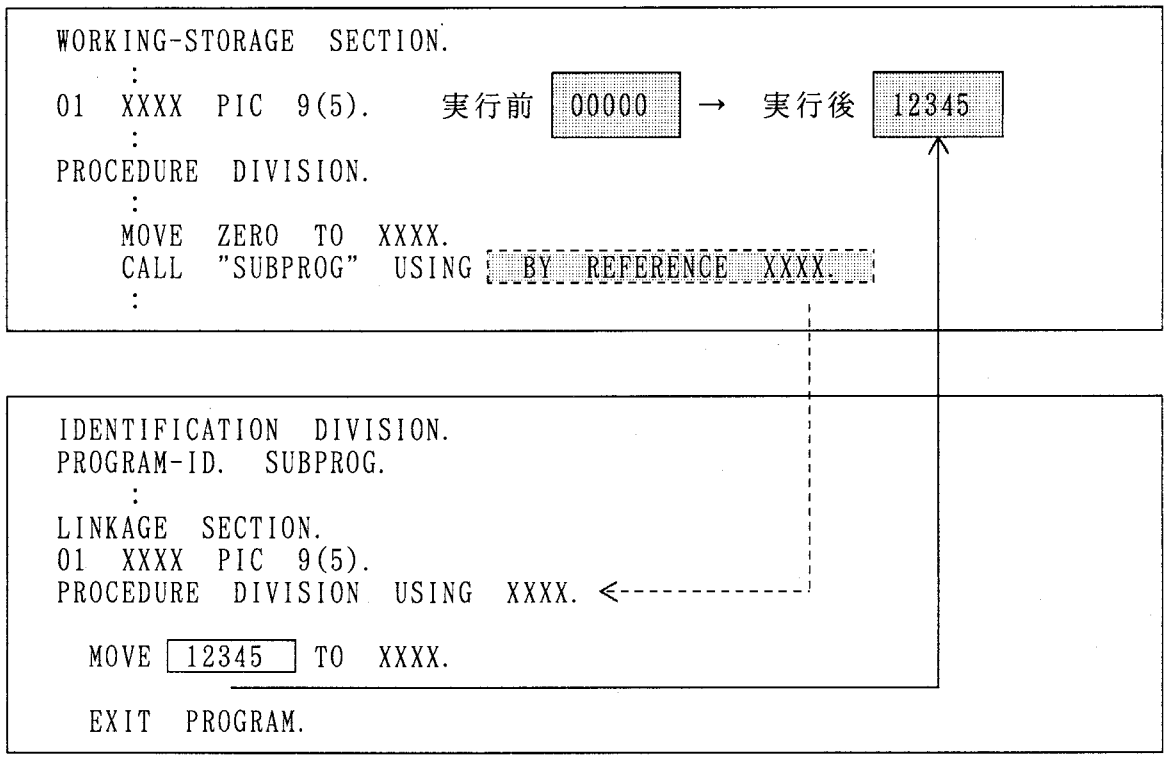
各データ記述項には、次のデータ句を記述する。
・レベル番号０１、７７
・データ名
・ＰＩＣＴＵＲＥ句またはＵＳＡＧＥ　ＩＳ　ＩＮＤＥＸ句
その他の句は必要に応じて記述する。

LINKAGE　SECTIONには「VALUE」句で初期値を指定できない。（ﾚﾍﾞﾙ88は除く）

（３）　パラメータの受け渡し

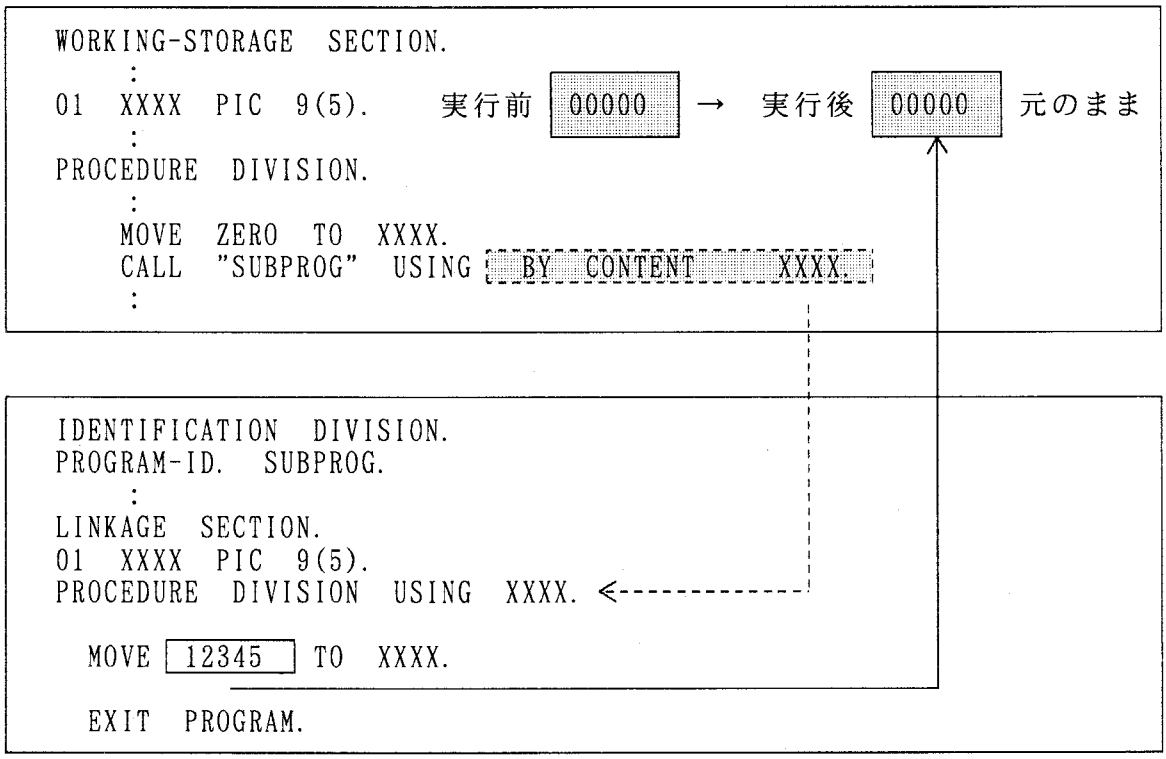
①　ＢＹ　ＲＥＦＥＲＥＮＣＥ指定

呼ばれるプログラムでパラメータの値を変更すると、ＣＡＬＬ文側のパラメータも変更される。



②　ＢＹ　ＣＯＮＴＥＮＴ指定

呼ばれるプログラムでパラメータの値を変更しても、ＣＡＬＬ文側のパラメータは変更されない。



１０．１．２　ＥＸＩＴ　ＰＲＯＧＲＡＭ

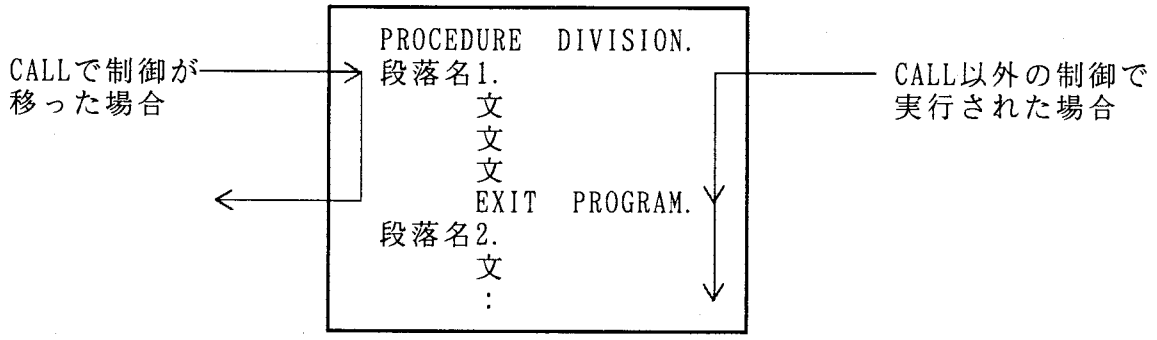
呼ばれるプログラムの論理的な終わりを示し、制御を呼出しプログラムへ戻す。

```
EXIT  PROGRAM
```

完結文の一連の文中に記述する場合は、最後の文として記述する。

```
段落名1.
        文
        文
        EXIT  PROGRAM          ←誤りである
        文                  .     最後の文ではない
段落名2.
        文
        文
        文
        EXIT  PROGRAM          .
段落名3.
```

ＣＡＬＬ文で呼ばれないプログラムでEXIT　PROGRAMが実行されると制御は、次の実行文を続ける。



１０．１．３　ＣＡＮＣＥＬ

ＣＡＮＣＥＬ（取消し）文は、指定したプログラムが再度呼び出された場合、そのプログラムが初期状態にあることを保証する。

```
CANCEL { 一意名1 } ...
       { 定数1  }
```

ＣＯＢＯＬでの「プログラム初期状態」とは、実行単位中で最初に呼び出された状態をいう。具体的には、作業場所節（WORKING-STORAGE）のデータ項目に記述したVALUE句の値で初期化されている状態である。
次の場合が初期状態である。

①そのプログラムが実行単位で最初に呼び出されたとき。
②ＣＡＮＣＥＬ文の実行後、そのプログラムが最初に呼び出されたとき。
③初期化属性をもつプログラムが呼び出されたとき。

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名  IS  INITIAL  PROGRAM.
```

「PROG」が実行される度にﾃﾞｰﾀ名「ｶｳﾝﾄ」の値が増加していく。

```
CALL  "PROG".
```

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  PROG.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
  :
01  ｶｳﾝﾄ  PIC  9(5)  VALUE  ZERO.
  :
PROCEDURE  DIVISION.
  :
  COMPUTE  ｶｳﾝﾄ  =  ｶｳﾝﾄ + 1
  :
  EXIT  PROGRAM.
```

「PROG」が何回も実行されてもﾃﾞｰﾀ名「ｶｳﾝﾄ」の値は初期値に設定されてから実行することになる。

```
CALL  "PROG"
CANCEL  "PROG".
```

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  PROG.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
  :
01  ｶｳﾝﾄ  PIC  9(5)  VALUE  ZERO.
  :
PROCEDURE  DIVISION.
  :
  COMPUTE  ｶｳﾝﾄ  =  ｶｳﾝﾄ + 1
  :
  EXIT  PROGRAM.
```

１０．２　プログラムの入れ子

ＣＯＢＯＬ原始プログラムは、別のＣＯＢＯＬ原始プログラムを含むことができる。
含まれるプログラムは、それを含むプログラムの中で定義したデータ名やファイル名が利用できる。

基本構造

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名.
ENVIRONMENT  DIVISION.
  環境部記述項
DATA  DIVISION.
  データ部記述項
PROCEDURE  DIVISION.
      :
     文
      :
END  PROGRAM  ﾌﾟﾛｸﾞﾗﾑ名.
```

入れ子構造

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名1.
ENVIRONMENT  DIVISION.
  環境部記述項
DATA  DIVISION.
  データ部記述項
PROCEDURE  DIVISION.

      :

    IDENTIFICATION  DIVISION.
    PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名2.
    ENVIRONMENT  DIVISION.
      環境部記述項
    DATA  DIVISION.
      データ部記述項
    PROCEDURE  DIVISION.

    END  PROGRAM  ﾌﾟﾛｸﾞﾗﾑ名2.

    IDENTIFICATION  DIVISION.
    PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名3.

        IDENTIFICATION  DIVISION.
        PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名4.

        END  PROGRAM  ﾌﾟﾛｸﾞﾗﾑ名4.

    END  PROGRAM  ﾌﾟﾛｸﾞﾗﾑ名3.

END  PROGRAM  ﾌﾟﾛｸﾞﾗﾑ名1.
```

１０．２．１　プログラム終わり見出し

プログラム終わり見出しは、指定したＣＯＢＯＬ原始プログラムの終わりを表す。

```
END  PROGRAM  ﾌﾟﾛｸﾞﾗﾑ名.
```

プログラム名は、先行するＰＲＯＧＲＡＭ－ＩＤで指定した名前と同じものを指定する。

原始プログラムの入れ子の一般形式は、次のとおり。

```
見出し部
[環境部]
[データ部]
[手続き部]
[原始プログラム] ・・・
[プログラム終わり見出し]
```

１０．３　共通プログラムと初期化プログラム

実行単位（ﾛｰﾄﾞﾓｼﾞｭｰﾙ）を構成する全てのCOBOLプログラムは、属性として「共通属性」と「初期化属性」をもつことができる。
共通プログラムとは、プログラムに直接、または間接に含まれるプログラムからも呼び出せるプログラムをいう。共通属性は、PROGRAM-IDで「ＣＯＭＭＯＮ」句を指定する。
初期化プログラムとは、呼び出される度にそのプログラムの状態が初期化されるプログラムをいう。初期化属性は、PROGRAM-IDで「ＩＮＩＴＩＡＬ」句を指定する。

```
PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名
          [IS { | COMMON  | } PROGRAM] .
              { | INITIAL | }
```

初期化プログラムは、そのプログラムに実行が移ると最初にデータ部を初期化する。
データ項目にVALUE句が指定されているとその初期値で初期化される。VALUE句が指定されていないと定義されていない値で初期化されることになる。

```
    CALL  "REENT"  USING  ﾃﾞｰﾀ名.

IDENTIFICATION  DIVISION.
PROGRAM-ID.  REENT  IS  INITIAL  PROGRAM .
DATA DIVISION.
WORKING-STORAGE  SECTION.
01  INIT1  PICTURE  S9(2)  VALUE  77.
01  INIT2  PICTURE  X(2).
LINKAGE  SECTION.
01  PARAM  PICTURE  S9(2).
PROCEDURE  DIVISION.
INI.
    COMPUTE  PARAM  =  PARAM  -  INIT1.
    MOVE  PARAM  TO  INIT1.
    MOVE  "**"   TO  INIT2.
      :
EXT.  EXIT  PROGRAM.
```

←何回呼ばれてもデータ部は最初の状態に戻される。

考え方は、Ｃ言語の自動変数の初期化と同じである。

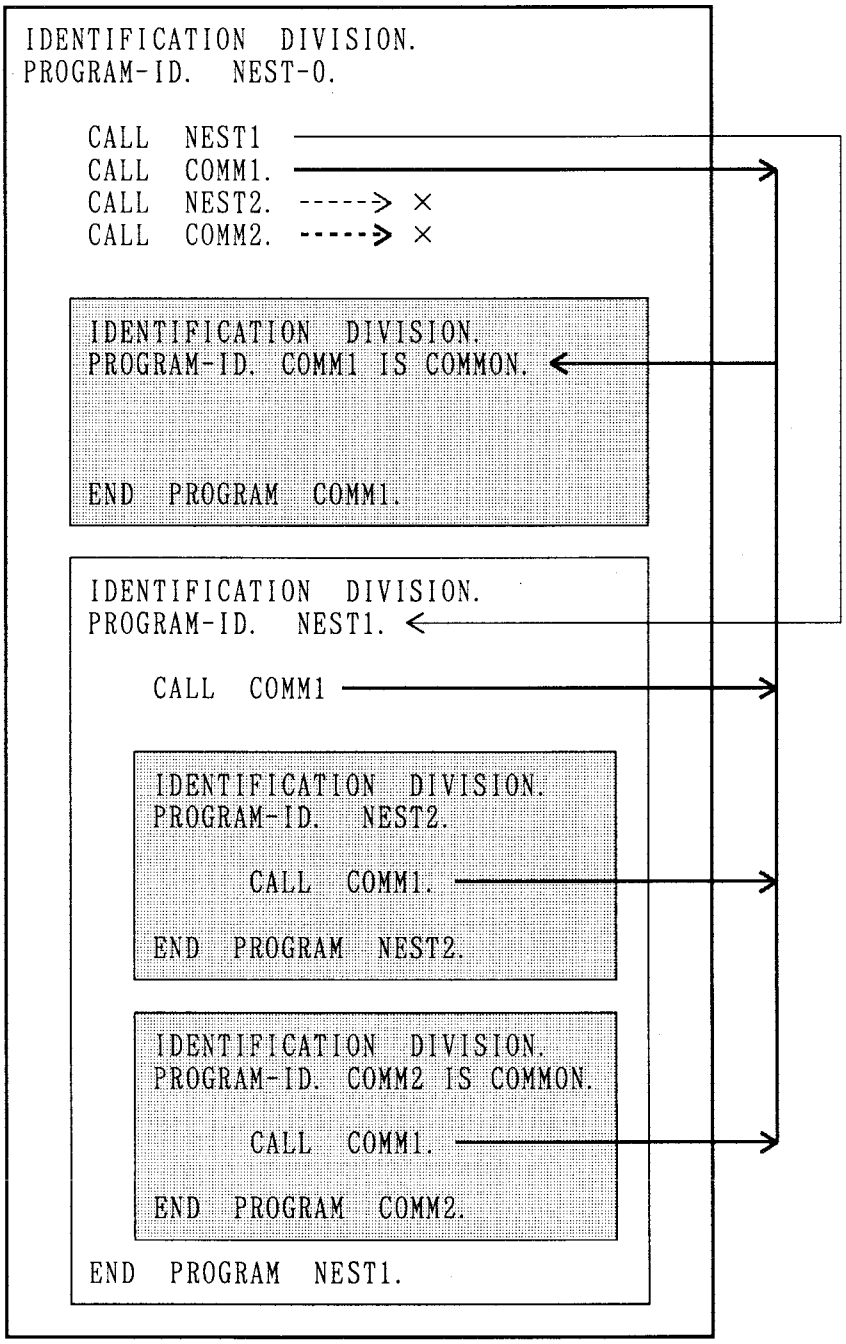
```
        :
      reent(&変数名);
}
reent(param)
int  *param;
{
auto  int  init1  =  77;
auto  char init2[2];

      *param = *param - init1;
      init1  = *param;
      strncpy(init2,"**",2);
        :
}
```

```
auto  int   init1;
auto  char  init2[2];
      init  =  77;
```

共通プログラムは、プログラム中に含まれる全てのプログラムから呼び出すことができる。また、共通プログラムにも初期化属性のＩＮＩＴＩＡＬ句が指定できる。

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名  IS  COMMON  PROGRAM.
```

共通プログラムの例は、次のとおり。



１０．４　ＧＬＯＢＡＬ句

ＧＬＯＢＡＬ句はデータ名、ファイル名、報告書名に指定できる。
ＧＬＯＢＡＬが指定されたデータ名やファイル名は、記述されたプログラムに含まれる全てのプログラムで使用できる。

```
IS  GLOBAL
```

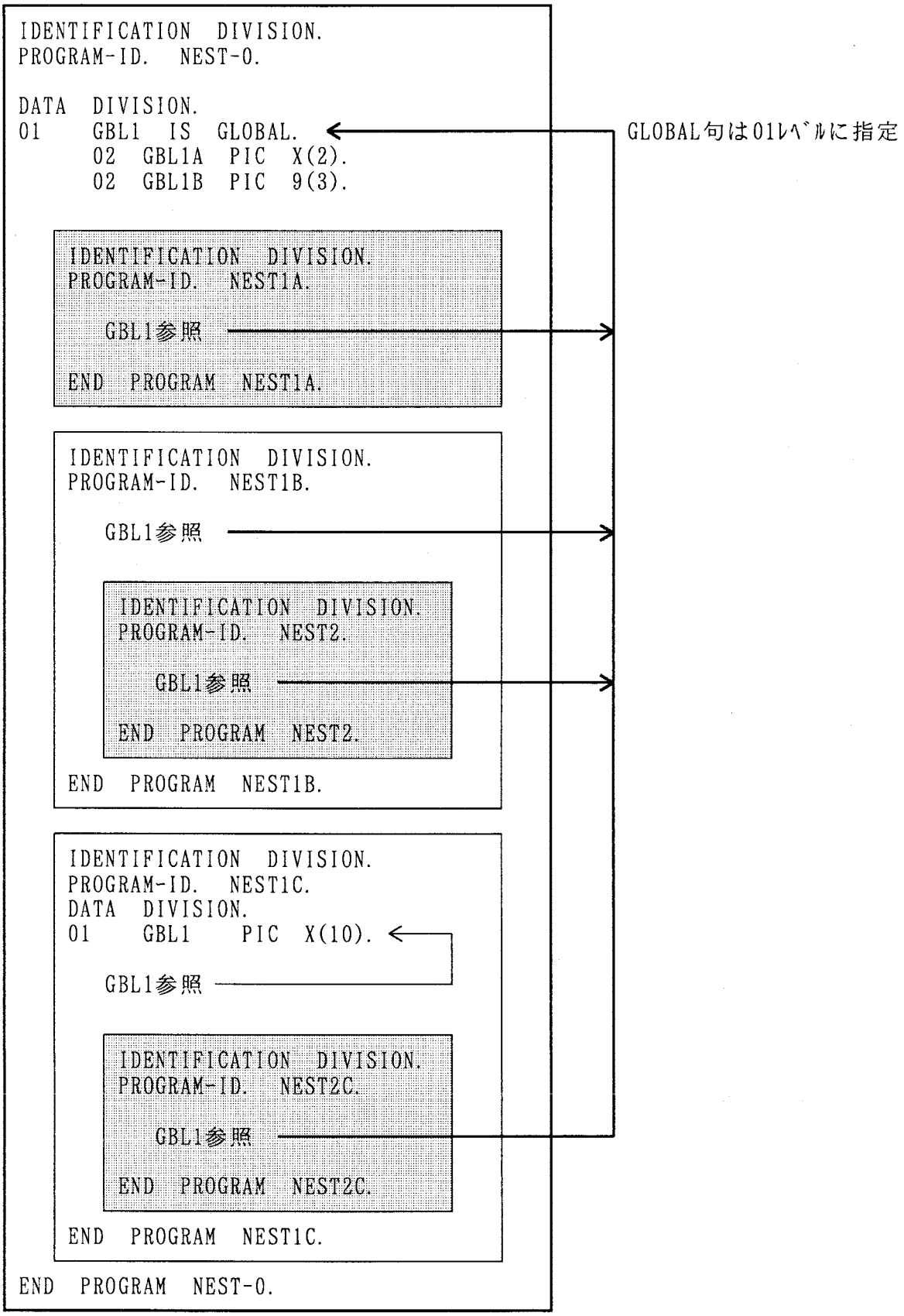
ＧＬＯＢＡＬは、ファイル節（FILE）や作業場所節（WORKING-STORAGE）のレベル番号が０１のデータ記述項に指定できる。

```
DATA  DIVISION.
FILE  SECTION.
FD    ﾌｧｲﾙ名  IS GLOBAL
      BLOCK  CONTAINS  整数  RECORDS.

      :

WORKING-STORAGE  SECTION.
01  ﾃﾞｰﾀ名  IS GLOBAL
      PICTURE  文字列  VALUE  定数.
```

考え方は、Ｃ言語の外部データ定義の機能であり、関数定義の外側に変数を定義して、そのプログラムに含まれる全ての関数から使用できる機能である。

```
#include <stdio.h>
static  FILE  *file1;                    <------
static  int    gbl1  =  2001;                    参照可能

main()
{
    file1  =  fopen("ﾌｧｲﾙ名","r");   --------->
    --gbl1;                          --------->
      :
}
func1()
{
    fgets(buffer,n,file1);   ----------------->
      :
}
func2()
{
    ++gbl1;   ---------------------------------->

}
```

ＧＬＯＢＡＬ句の例は、次のとおり。



１０．５　ＥＸＴＥＲＮＡＬ句

ＥＸＴＥＲＮＡＬ句は、データ項目やファイル（ファイル結合子）が外部属性をもつことを指定する。外部属性をもつデータ項目は実行単位中の全てのプログラムで使用できる。

```
IS  EXTERNAL
```

ＥＸＴＥＲＮＡＬ句は、作業場所節（WORKING-STORAGE）のレベル番号が０１のデータ記述項にだけ指定できる。

```
DATA  DIVISION.
FILE  SECTION.
FD    ﾌｧｲﾙ名  IS  EXTERNAL
      BLOCK  CONTAINS  整数  RECORDS.

      :

WORKING-STORAGE  SECTION.
01  ﾃﾞｰﾀ名  IS  EXTERNAL
      PICTURE  文字列  VALUE  定数.
```

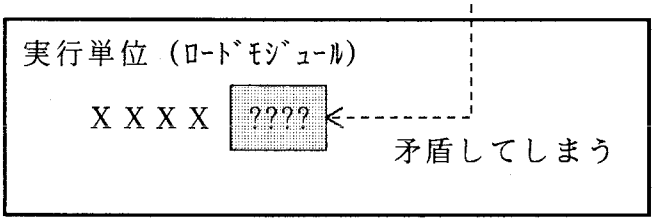
ＥＸＴＥＲＮＡＬ句を指定したデータ項目には、初期値を設定するＶＡＬＵＥ句が指定できない。

主プログラム

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  MAIN.
    :
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  ＸＸＸＸ  IS  EXTERNAL
    PIC  X(4)  VALUE  "ABCD"
    :
```

サブプログラム

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  SUB.
    :
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  ＸＸＸＸ  IS  EXTERNAL
    PIC  X(4)  VALUE  "1234"
    :
```



ＣＯＢＯＬの「ファイル結合子」とは、ファイルについての情報をもつ記憶領域であり、ファイルとレコード領域に関する情報をもつ。

ＥＸＴＥＲＮＡＬ句の例は、次のとおり。

```
ＦＤ      ファイル名   IS  EXTERNAL

ﾚﾍﾞﾙ番号  データ名     IS  EXTERNAL
```

原始プログラム「MAIN1」

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  MAIN1.

DATA  DIVISION.
FILE  SECTION.

FD    EXTFILE  IS  EXTERNAL
01    EXTREC   ～

WORKING-STORAGE  SECTION.

01    EXTDATA  IS  EXTERNAL.
      02  EXT1  PIC  9(4).
      02  EXT2  PIC  X(4).

PROCEDURE  DIVISION.

    OPEN  INPUT  EXTFILE
    CALL  SUB1.
    CALL  SUB2.
    CLOSE  EXTFILE。

END  PROGRAM  MAIN1.
```

原始プログラム「SUB1」

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  SUB1.

DATA  DIVISION.

01    EXTDATA  IS  EXTERNAL.
      02  EXT1  PIC  9(4).
      02  EXT2  PIC  X(4).




END  PROGRAM  SUB1.
```

原始プログラム「SUB2」

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  SUB2.

DATA  DIVISION.
FILE  SECTION.

FD  EXTFILE  IS  EXTERNAL
01  EXTREC   ～

PROCEDURE  DIVISION.

    READ  EXTFILE  ～

END  PROGRAM  SUB2.
```

１０．６　関数

ＣＯＢＯＬにも、組込み関数機能があり、実行時に引用すると自動的に値が得られるデータ項目を参照する機能である。
関数は、手続き部で「関数一意名」として引用する。

```
FUNCTION　関数名　［（｛引数1｝･･･）］　［部分参照］
```

関数が返すデータ値を「関数値」という。
引数の値が規則に従っていない場合の関数値は規定されていない。

１０．６．１　関数の型

関数値のデータ項目は基本項目として、英数字、数字または整数の値を返す。
関数は受取り側には使用できない。

関数の型	字類・項類	解説
英数字関数	英数字	用途（USAGE）は暗黙的にDISPLAY。 データ項目のサイズは関数に規定されている。 MOVE　FUNCTION LOWER-CASE("ABC") TO　ﾃﾞｰﾀ名
数字関数	数字	関数値は符号付きであり、算術式の中に記述。 整数の作用対象が要求されているところで数字関数は指定できない。 COMPUTE　ﾃﾞｰﾀ名1 = FUNCTION REM(ﾃﾞｰﾀ名2 ﾃﾞｰﾀ名3) + 1
整数関数	数字	関数値は符号付きであり、算術式の中に記述。 符号付き整数が要求されているところで指定する。 COMPUTE　ﾃﾞｰﾀ名1 = FUNCTION INTERGER(ﾃﾞｰﾀ名2)

１０．６．２　引数

引数には、一意名、算術式や定数が指定できる。引数の個数は関数の定義に規定されているが、個数が可変の引数を指定する関数もある。
引数の型は、次のとおり。

引数の型	解　説
数　字	算術式を指定する。符号を含めた値を引数として使用する。 COMPUTE　最大値1　=　FUNCTION　MAX(ﾃﾞｰﾀ名1　1.23　ﾃﾞｰﾀ名2)
英　字	英字の基本項目または英字だけを含む文字定数を指定する。 データ項目や文字定数のサイズが関数値を定めるのに用いられることもある。 MOVE FUNCTION　MAX("DEF"　ﾃﾞｰﾀ名1　ﾃﾞｰﾀ名2)　TO　最大値2
英数字	英字または英数字の基本項目や文字定数を指定する。 データ項目や文字定数のサイズが関数値を定めるのに用いられることもある。 MOVE FUNCTION　MAX("12CH"　ﾃﾞｰﾀ名1　ﾃﾞｰﾀ名2)　TO　最大値3
整　数	値が常に整数値となる算術式を指定する。 COMPUTE　最大値4　=　FUNCTION　MAX(ﾃﾞｰﾀ名1　123　ﾃﾞｰﾀ名2)

引数に指定できる「ＣＯＢＯＬの算術式」は、次のとおり。

①数字項目の一意名、数字定数や表意定数ＺＥＲＯ
②上記の項目を算術演算子でつないだもの
③２個の算術式を算術演算子でつないだもの
④算術式を括弧で囲んだもの
⑤算術式の前に単項演算子を付けたもの

関数の引数が不定の回数繰り返すことを許す場合、そこに表を指定することができる。
表要素はデータ名と添字を記述するが、添字に「ＡＬＬ」を書くことができる。
「ＡＬＬ」は、その添字の関連する表要素をすべて指定したかのように扱われる。

データ名　（{ＡＬＬ}･･･）

１０．６．３　関数一覧

引数の型、関数の型の意味は、次のとおり。
　Ａ：英字　Ｉ：整数　Ｎ：数字　Ｘ：英数字

関　数　名	引　数	関数の型	関数値
FUNCTION　ACOS	N1	N	N1の逆余弦
FUNCTION　ANNUITY	N1　I2	N	元金1、利率N1、期間I2の均等払い額
FUNCTION　ASIN	N1	N	N1の逆正弦
FUNCTION　ATAN	N1	N	N1の逆正接
FUNCTION　CHAR	I1	X	文字の大小順序における順序位置I1の文字
FUNCTION　COS	N1	N	N1の余弦
FUNCTION　CURRENT-DATE	なし	X	現在の日付、時刻、ｸﾞﾘﾆｯｼﾞ平均時からの時差
FUNCTION　DATE-OF-INTEGER	I1	I	ｸﾞﾚｺﾞﾘｵ通日I1に対応する標準形式yyyymmddの日付
FUNCTION　DAY-OF-INTEGER	I1	I	ｸﾞﾚｺﾞﾘｵ通日I1に対応する年月yyyyddd
FUNCTION　FACTORIAL	I1	I	I1の階乗
FUNCTION　INTEGER	N1	I	N1を超えない最大整数
FUNCTION　INTEGER-OF-DATE	I1	I	標準形式yyyymmddの日付I1に対応するｸﾞﾚｺﾞﾘｵ通日
FUNCTION　INTEGER-OF-DAY	I1	I	年月yyyydddのI1に対応するｸﾞﾚｺﾞﾘｵ通日
FUNCTION　INTEGER-PART	N1	I	N1の整数部
FUNCTION　LENGTH	A1または N1または X1	I	引数の長さ
FUNCTION　LOG	N1	N	N1のｅを底とする自然対数
FUNCTION　LOG10	N1	N	N1の10を底とする常用対数
FUNCTION　LOWER-CASE	A1またはX1	X	引数中の大文字を全て小文字にしたもの
FUNCTION　MAX	A1‥または I1‥または N1‥または X1	引数の型	最大の引数の値

関　数　名	引　数	関数の型	関数値
FUNCTION　MEAN	N1･･･	N	引数の算術平均値
FUNCTION　MEDIAN	N1･･･	N	引数の中央値
FUNCTION　MIDRANGE	N1･･･	N	引数中の最小値と最大値の平均値
FUNCTION　MIN	A1･･または I1･･または N1･･または X1	引数の型	最小の引数の値
FUNCTION　MOD	I1、I2	I	I2を法とするI1
FUNCTION　NUMVAL	X1	N	数字定数の形をした数字列の数値
FUNCTION　NUMVAL-C	X1、X2	N	ｺﾝﾏや通貨記号のある数字列の数値
FUNCTION　ORD	A1またはX1	I	文字の大小順序における順序位置
FUNCTION　ORD-MAX	A1･･または N1･･または X1	I	引数中での最大値の順序位値
FUNCTION　ORD-MIN	A1･･または N1･･または X1	I	引数中での最小値の順序位値
FUNCTION　PRESENT-VALUE	N1、N2･･･	N	減価率N1による数列N2の和と現在価値
FUNCTION　RANDOM	I1	N	乱数
FUNCTION　RANGE	I1･･または N1･･	引数の型	引数中の最大値と最小値の差
FUNCTION　REM	N1、N2	N	N1÷N2の余り
FUNCTION　REVERSE	A1またはX1	X	引数の文字列の順序を逆にしたもの
FUNCTION　SIN	N1	N	N1の正弦
FUNCTION　SQRT	N1	N	N1の平方根
FUNCTION　STANDARD-DEVIATION	N1･･･	N	引数の標準偏差
FUNCTION　SUM	I1･･または N1	引数の型	引数の和

関　数　名	引　数	関数の型	関数値
FUNCTION　TAN	N1	N	N1の正接
FUNCTION　UPPER-CASE	A1またはX1	X	引数中の小文字を全て大文字にしたもの
FUNCTION　VARIANCE	N1・・・	N	引数の分散
FUNCTION　WHEN-COMPILED	なし	X	ﾌﾟﾛｸﾞﾗﾑが翻訳された日時

注１：　日付変換関数では、ｸﾞﾚｺﾞﾘｵ暦を用いる。1601年1月1日（月曜日）を、ｸﾞﾚｺﾞﾘｵ通日の第一日とする。

注２：　ＣＯＢＯＬの予約語としては「ＦＵＮＣＴＩＯＮ」のみであり、関数名は予約語に含まれない。

指導上の留意点

◎プログラム構造
プログラムの構造をコンパイル単位である原始プログラムからみると、次のとおり。

COBOL

```
IDENTIFICATION  ～
PROGRAM-ID.   MAIN.
   :
PROCEDURE   ～
   :
     CALL   SUBP   ～
   :
```

```
IDENTIFICATION  ～
PROGRAM-ID.   SUBP.
   :
PROCEDURE   ～   USING
   :
  EXIT   PROGRAM.
```

FORTRAN

```
PROGRAM   MAIN
   :
   CALL   SUBP(p1)
   :
   I=FUNC(p2)
   :
END
-----------------------
SUBROUTINE   SUB1
   :
END
```

```
SUBROUTINE   SUBP(p1)
   :
END
-----------------------
FUNCTION   FUNC1(p2)
END
```

```
FUNCTION   FUNC(p1)
   :
END
-----------------------
SUBROUTINE   SUB1(p3)
END
```

C

```
main()
{
   :
   func1(p1);
   func2(p2);
   :
}
-----------------------
func1(p1)
{
   :
   return ;
}
```

```
func2(p2)
{
   :
   func3(p3);
   :
}
-----------------------
func3(p3)
{
   :
}
```

◎プログラム間連絡
プログラムから別のプログラムに制御を移す方法には、サブルーチン方式と関数方式がある。

COBOLの場合

```
      CALL   "ﾌﾟﾛｸﾞﾗﾑ名"     USING   ﾃﾞｰﾀ名1   ﾃﾞｰﾀ名2   ﾃﾞｰﾀ名3   ～

      PROCEDURE   DIVISION   USING   ﾃﾞｰﾀ名1   ﾃﾞｰﾀ名2   ﾃﾞｰﾀ名3   ～
```

FORTRANの場合

```
(1)ｻﾌﾞﾙｰﾁﾝ副ﾌﾟﾛｸﾞﾗﾑ
      CALL   ﾌﾟﾛｸﾞﾗﾑ名          (実引数1,   実引数2,   変数名3,   ～)

      SUBROUTINE   ﾌﾟﾛｸﾞﾗﾑ名   (仮引数1,   仮引数2,   仮引数3,   ～)

(2)関数副ﾌﾟﾛｸﾞﾗﾑ
      関数名                (実引数1,   実引数2,   変数名3,   ～)

      FUNCTION   関数名   (仮引数1,   仮引数2,   仮引数3,   ～)
```

◎プログラムの入れ子
プログラムの入れ子の利点は、プログラム上に記述されている資源（ﾃﾞｰﾀ名等）を有効に使用できることにある。
プログラムの入れ子構造が可能な言語は、次のとおり。

ＣＯＢＯＬ

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名1.
      :
PROCEDURE  DIVISION.
      :
      IDENTIFICATION  DIVISION.
      PROGRAM-ID.  ﾌﾟﾛｸﾞﾗﾑ名2.
        :
      PROCEDURE  DIVISION.
        :
      END  PROGRAM  ﾌﾟﾛｸﾞﾗﾑ名2.
      :
END  PROGRAM  ﾌﾟﾛｸﾞﾗﾑ名1.
```

制御はCALL文で移行する。

ＰＬ／Ｉ

```
名前1:  PROCEDURE  OPTIONS(MAIN);
          :
          名前2:  PROCEDURE;
                    :
                  END  名前2;
          :
          名前3:  BEGIN;
                    :
                  END  名前3;
        END  名前1;
```

制御はCALL文、関数参照で移行する。

制御は通常の流れある。

Ｃ　（変数の定義が入れ子にできる）

```
main()
{
    宣言
    :
    文
    :
    {
        宣言
        文
    }
    文
}
```

実行制御がCOBOL,PL/Iと異なる

制御は複文の実行である。

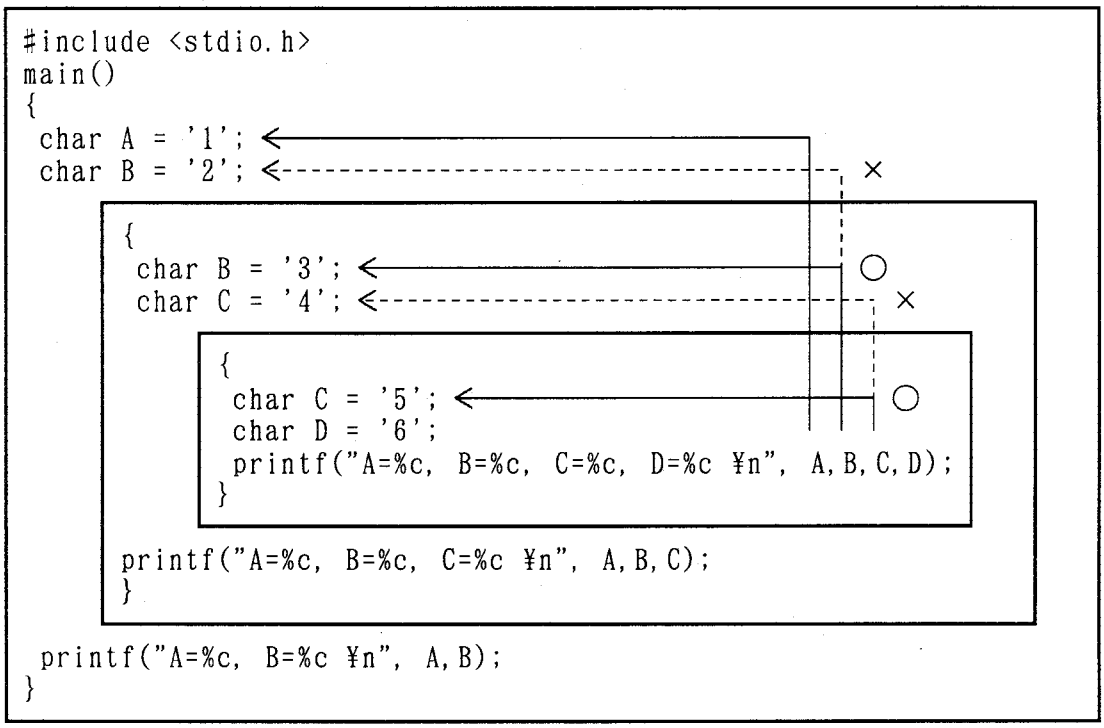
◎有効範囲について

ＣＯＢＯＬでは、データ名とファイル名の有効範囲が「大域」と「局所」に分類して次のように説明している。

大域名は、その大域名が宣言されているプログラム中から、または、その大域名を宣言するプログラムに含まれている他の任意のプログラム中から使用することができる。

局所名は、その局所名が宣言されたプログラム中からその局所名が表す対象を参照するためにだけ使用する事ができる。

次の例は、Ｃ言語の変数名の有効範囲を確認する原始ﾌﾟﾛｸﾞﾗﾑと実行結果である。



実行結果

```
A=1, B=3, C=5, D=6
A=1, B=3, C=4
A=1, B=2
```

有効範囲を実行時の記憶域管理からみると、スタック領域あるいは自動記憶域は、そのブロックに実行が移ったときに確保され、実行が終わるとその領域は開放される。
全部の自動変数領域を最初に確保して実行する場合よりも効率的である。

コーディング面では、ブロック内で定義しない変数名を参照しても外側のブロックに定義されていればエラーとならないので、利用には注意が必要である。

◎外部属性

ＣＯＢＯＬでは、「内部対象」と「外部対象」があり対象はデータ項目とファイル（結合子）である。
内部対象は、その対象を記述するプログラムにだけ属する場合をいう。
外部対象は、その対象を記述する実行単位のいかなるプログラムからも参照できる。

ＣＯＢＯＬでは、外部対象をEXTERNAL句で宣言する。

```
FD  ﾌｧｲﾙ名    [IS  EXTERNAL]

01  ﾃﾞｰﾀ名    [IS  EXTERNAL]
```

◎関数（function）

１９９２年の規格ＣＯＢＯＬから「組込み関数（intrinsic function）機能単位」が追加された。水準１のみで構成される。

プログラミング言語には、サブルーチンと関数の考え方がありサブルーチンはＣＡＬＬで参照する。
関数には、いくつかの種類がある。ＦＯＲＴＲＡＮを例にとると次のように分類されている。

組込み関数	処理系があらかじめ用意し、特別な意味（機能）をもっている。
文関数	算術代入文、文字代入文などと同様な形式をもつ単一の文で定義する 定義の形　　文関数名（［仮引数［, 仮引数］･･･］）＝式 Ｃ言語では、#define文で実現できる。 #define ADD(x,y) (x+y)
外部関数	引用するプログラム単位の外部で関数副プログラムとして定義する。 定義の形　　［ﾀｲﾌﾟ］FUNCTION 外部関数名（［仮引数］･･･）

関数は、式のなかで引用される。代入文の左辺では通常使用できない。

```
変数名  ＝  式  ←式の中で関数が使用できる。
```

ただし、ＰＬ／Ｉでは同じ関数名でも代入文の左辺と右辺に記述できるものがある。

```
SUBSTR(文字列,開始位置,長さ)  =  SUBSTR(文字列,開始位置,長さ);
   擬似変数と呼ばれている          組込み関数
```

◎関数の仕組み

関数は、実行時に引用すると定義された機能の手続きが実行されて値を返すデータ項目であり、仕組みは次のとおり。

ＦＯＲＴＲＡＮ	Ｃ
引用例 変数名=関数名(10,20)*100	引用例 変数名=関数名(10,20)*100;

定義例
関数名　FUNCTION (I,J)
関数名 = I+J
END
	定義例
	関数名(i,j)
	{
	int i, j;
	return(i+j);
	}

ＦＯＲＴＲＡＮでは、サブルーチンと関数を区別して利用者が作成できる。