第６章　事務計算アルゴリズム

指導目標

実務のプログラムを作成する上での基本的なアルゴリズムとそれを表現するプログラミング技法について学習することで、数値計算、表操作、文字列処理、整列操作などの基本的なデータ操作が理解できるようにする。

繰り返し処理では添字や指標の扱いが重要であり、プログラムの読みやすさに関わることを身につけさせ、何が適切なのかを考えさせる。
また、ＣＯＢＯＬは標準データ形式で記述できるが、さらに実際に使うコンピュータの内部データ表現形式を理解していれば、より効率的なプログラムが作成できることも指導する。

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
データチェック	COBOLの機能を利用したﾁｪｯｸについて説明する ・ﾆｭｰﾒﾘｯｸﾁｪｯｸ ・ﾘﾐｯﾄﾁｪｯｸ ・ｵｰﾊﾞｰﾌﾛｰﾁｪｯｸ	どのような場合に使用したら有効かを議論する		ﾌﾟﾛｸﾞﾗﾑを作成しﾃﾞｰﾀﾁｪｯｸの確認を行う
四捨五入	ROUNDEDの使用と注意点を説明する。	正と負の場合の四捨五入について議論する。		全ての算術文でROUNDEDの実習をする。
最大、最小、平均	最大値、最小値合計値、平均値の求め方とプログラムの注意点を説明する。			例題を実行し問題点を整理する。
時間の変換	算術文での余りの求め方と計算の順序について説明する。			
閏年の判定	閏年の判定の計算式を説明する			
文字列処理	文字列の連結操作を中心に、他の言語を比較して説明する。	INSPECTの使い方について議論する。	1文字単位の操作をｺｰﾃﾞｨﾝｸﾞして比較する。	
文字列検索	文字と文字列の関係、添字処理について説明する。	IF、指標名を使用した場合のﾌﾟﾛｸﾞﾗﾑについて議論する。		
基数変換	2進数→10進数 10進数→2進数 変換を説明する 8進数→10進数 10進数→8進数 変換を説明する 10進数→16進数 16進数→10進数 変換を説明する		例題以外の方法を考えさせる。	

内　　容	説　明	議　論	机上実習	計算機実習
内部形式ﾃﾞｰﾀ変換	ﾃﾞｰﾀの内部形式を意識した処理としてﾃﾞｰﾀ変換を説明する。			
配列処理	PERFORMの使い方と配列の領域について説明する。	FORTRANとCOBOLの配列の割付方を議論する。		添字や指標が表操作で使いこなせるようにする。
表探索	逐次、２分探索を使うための要件を説明する。	逐次、２分探索の比較回数について議論する。		表とSEARCHを使いこなせるようにする。
ハッシュ法	ハッシュの考え方を中心に応用事例を説明する	ｺﾝﾌﾘｸﾄの対策と問題点を議論する。	ﾊｯｼｭ関数を考える。	ﾊｯｼｭ関数を作成しﾊｯｼｭ値のｺﾝﾌﾘｸﾄを調査する。
内部整列法	基本交換法、基本選択法、基本挿入法を具体的に説明する。	内部整列法とSORT文の応用について議論する	ﾃﾞｰﾀを具体的に与えて整列法の動きを理解させる。	例題を実行して特徴を理解する。

第６章　事務計算アルゴリズム（基本）

６．１　データチェック

データに誤りがあり、それがチェックされずに処理が進行すると全体に大きな影響がする。ここでは、ＣＯＢＯＬプログラミングの観点からデータチェックについて解説する。

６．１．１　項目チェック

（１）ニューメリックチェック（numeric　check）
データ項目に数字（０～９）以外の文字が入っていないかどうかチェックする。

（２）アルファベティックチェック（alphabetic　check）
データ項目に英字（Ａ～Ｚ、空白）以外の文字が入っていないかどうかチェックする。

・字類条件を使用してチェックができる。
データの内容が数字であるか、英字であるかを判定する。

```
一意名　IS　[NOT]　{ NUMERIC
                    { ALPHABETIC
                    { ALPHABETIC-LOWER
                    { ALPHABETIC-UPPER
```

使用例は、次のとおり。

```
01　社員ﾚｺｰﾄﾞ.
    02　社員ｺｰﾄﾞ        PICTURE　9(7).
    02　氏名             PICTURE　X(20).
        :

  IF　社員ｺｰﾄﾞ　NOT　NUMERIC　THEN
      DISPLAY　"社員ｺｰﾄﾞに数字以外の文字があります"
  END-IF
```

字類条件とデータ項目の組合せは、次のとおり。

条件＼タイプ	英字項目	英数字項目	英数字編集項目	数字編集項目	数字項目
NUMERIC	×	○	○	○	○
ALPHABETIC ALPHABETIC-LOWER ALPHABETIC-UPPER	○	○	○	○	×

規格ＣＯＢＯＬにはないが漢字（全角文字）かどうかのデータチェックができる例もある。

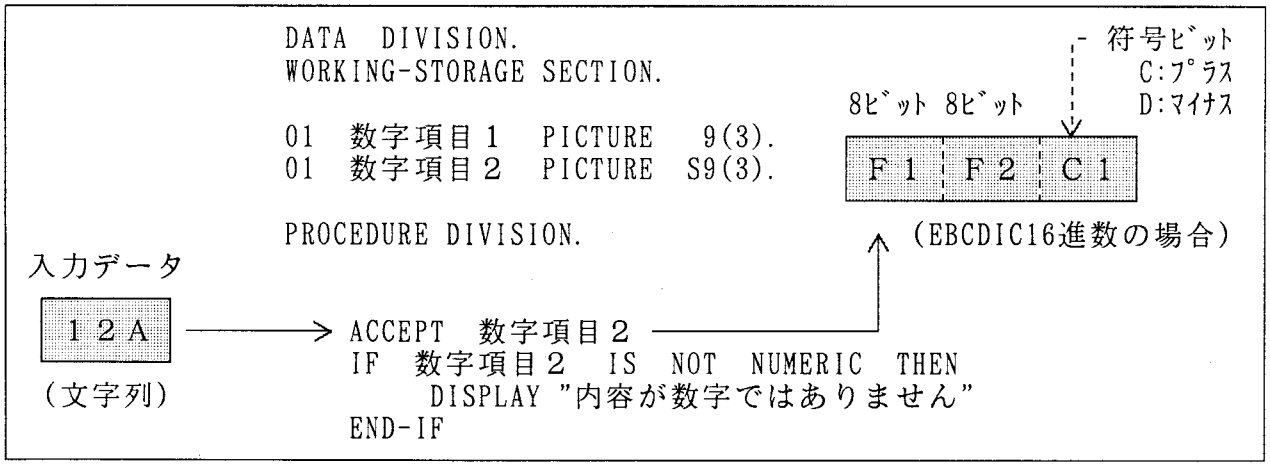
```
一意名　IS　[NOT]　KANJI
```

Ｃ言語にも文字チェック関数がある。（ただし、対象は１文字）

```
isdigit(c)　　isalpha(c)　　isupper(c)　　islower(c)
```

・符号の考慮

ニューメリックチェックの場合、データ項目が符号付きか、または符号なしかどうかで注意する必要がある。



上記の入力例では、「１２Ａ」は代数的に「＋１２１」と認識されエラーとならない。
このような場合には、データ記述項のＰＩＣＴＵＲＥ文字列で符号なし数字項目として定義する。

```
01  数字項目１  PICTURE  9(3) .
          :
    IF  数字項目１  IS  NOT  NUMERIC  THEN
        DISPLAY "内容が数字ではありません"
    END-IF
```

（３）リミットチェック（limit　check）

値がある範囲内にあるかどうかチェックする。

・条件名条件を使用してチェックができる。

ＣＯＢＯＬでは、レベル番号「８８」の条件名としてデータ部でデータ項目の値の範囲を定義できる。

```
88  条件名1 { VALUE  IS   } {定数1  [{THROUGH} 定数2] } ･･･
            { VALUES ARE  }          {THRU   }
```

条件名の使用例は、次のとおり。

```
01  学生        PICTURE  9(2).
    88  小学生  VALUE     7  THRU  12.
    88  中学生  VALUE    13  THRU  15.
    88  高校生  VALUE    16  THRU  18.
    88  大学生  VALUE    19  THRU  22.
          :
    IF  高校生  THEN   ～
          :
```

条件名を使用すると、手続きが容易に記述でき誤りが少なくて済む。また、条件の範囲の変更が生じてもデータ部の修正だけで済む場合もある。

```
IF  高校生  THEN
```

```
IF  学生 > 15  AND
    学生 < 19  THEN
```

（４）サインチェック（sign　check）

データ項目の値かどうかチェックする。

・正負条件を使用してチェックできる。算術式の代数値がゼロより大きい（POSITIVE）か、小さい（NEGATIVE）かまたは等しい（ZERO）か判定する。

```
算術式  IS  [NOT] { POSITIVE }
                  { NEGATIVE }
                  { ZERO     }
```

（５）オーバーフローチェック（overflow　check）

データの算術演算の結果、その値が指定の桁数よりあふれないかどうかチェックする。

```
算術文　ON　SIZE　ERROR
　　＜オーバーフロー処理の手続き＞
　　NOT　ON　SIZE　ERROR
　　＜正常処理の手続き＞
END-算術文
```

算術文は、COMPUTE、ADD、SUBTRACT、MULTIPLY、DIVIDEである。

ON　SIZE　ERRORは算術演算結果を格納する際、小数点の位置をそろえたとき、結果格納域の最大値（絶対値）を超えている場合に発生する。発生した場合には結果格納域は算術演算前のままである。ROUNDEDを一緒に指定すると、ROUNDED処理後にON　SIZE　ERRORのチェックを行う。

```
DATA　DIVISION.
WORKING-STORAGE　SECTION.
01　A　　PICTURE　S9(2)　VALUE　ZERO.　　<------ 加算すると１００になり
01　B　　PICTURE　S9(2)　VALUE　99.　　　　----- ２桁の領域には入らない
01　C　　PICTURE　S9(2)　VALUE　 1.　　　　-----

PROCEDURE　DIVISION.

    COMPUTE　A　=　B　+　C　ON　SIZE　ERROR
                DISPLAY　"ｵｰﾊﾞｰﾌﾛｰが発生しました"
    END-COMPUTE
```

ゼロの除算では常にON　SIZE　ERRORになる。

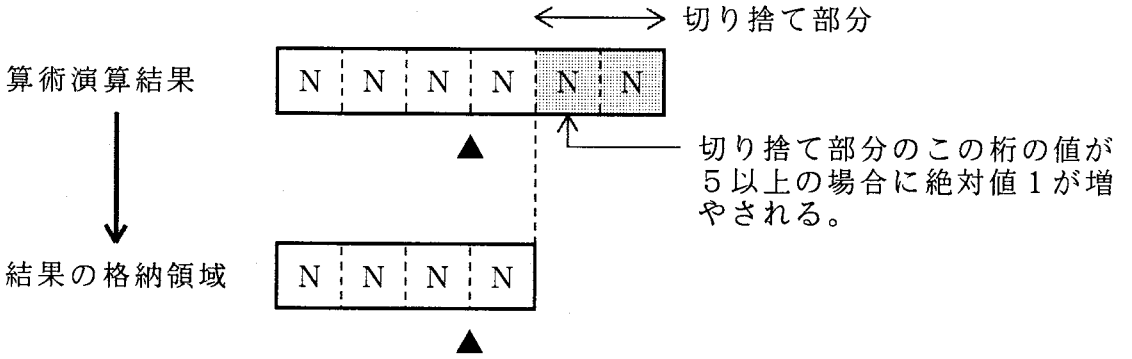
```
DATA　DIVISION.
WORKING-STORAGE　SECTION.
01　A　　PICTURE　S9(2)　VALUE　ZERO.
01　B　　PICTURE　S9(2)　VALUE　99.
01　C　　PICTURE　S9(2)　VALUE　 0. <-------- 除数がゼロ

PROCEDURE　DIVISION.
                                  |
                                  v
    COMPUTE　A　=　B　/　C　ON　SIZE　ERROR
                DISPLAY　"ｵｰﾊﾞｰﾌﾛｰが発生しました"
    END-COMPUTE
```

ＤＩＶＩＤＥ文には剰余をもとめる書き方もあり、商と剰余をいれる項目の両方でON　SIZE　ERRORが発生する可能性がある。発生した場合は商なのか剰余なのかの区別は利用者が行う。

```
DIVIDE　一意名1　INTO　一意名2　GIVING　　一意名3
                              REMAINDER　一意名4
                ON SIZE　ERROR
                  《一意名3なのか一意名4なのかは判らない》
END-DIVIDE
```

６．２　四捨五入

ＣＯＢＯＬでは、算術演算文に「ＲＯＵＮＤＥＤ」句を付けることで四捨五入ができる。ＲＯＵＮＤＥＤ句は、小数点位置をそろえたとき、算術演算結果の小数部の桁数が結果を入れる項目のそれより大きい場合で、切り捨て部分の最上位桁の値が５以上の場合に結果を入れる項目の最下位桁に絶対値の１が増やされる。四捨五入されても演算符号は変わらない。



四捨五入の使用例は、次のとおり。

```
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  A          PIC  S99V99     VALUE  7.45.
01  B          PIC  S99V9      VALUE  2.1.
01  計算結果   PIC  S999.
01  四捨五入   PIC  S999V9.

PROCEDURE  DIVISION.

    COMPUTE  計算結果            =  A  +  B.       ←結果は「９」
    COMPUTE  四捨五入  ROUNDED  =  A.             ←結果は「７．５」
```

整数部の四捨五入処理例として、１００円未満を四捨五入する場合は、PICTURE句で「Ｐ」を使用するとよい。

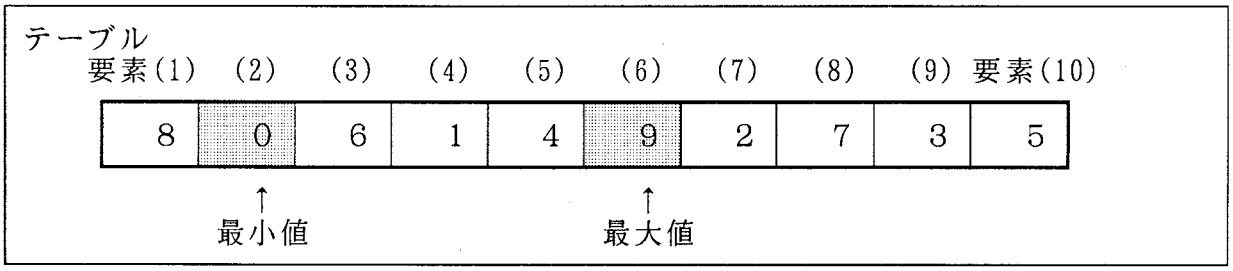
```
DATA  DIVISION.
WORKING-STORAGE SECTION.
01  A              PICTURE  S9999.
01  B              PICTURE  S9999.
01  １００円単位   PICTURE  S999PP.

PROCEDURE DIVISION.

    COMPUTE  １００円単位  ROUNDED  =  A  +  B
```

結果のデータ項目の整数部にPICTURE句の文字「Ｐ」が用いられていると、四捨五入や切り捨ては、実際に記憶領域を割り当てられている部分の最右端に対して行われる。

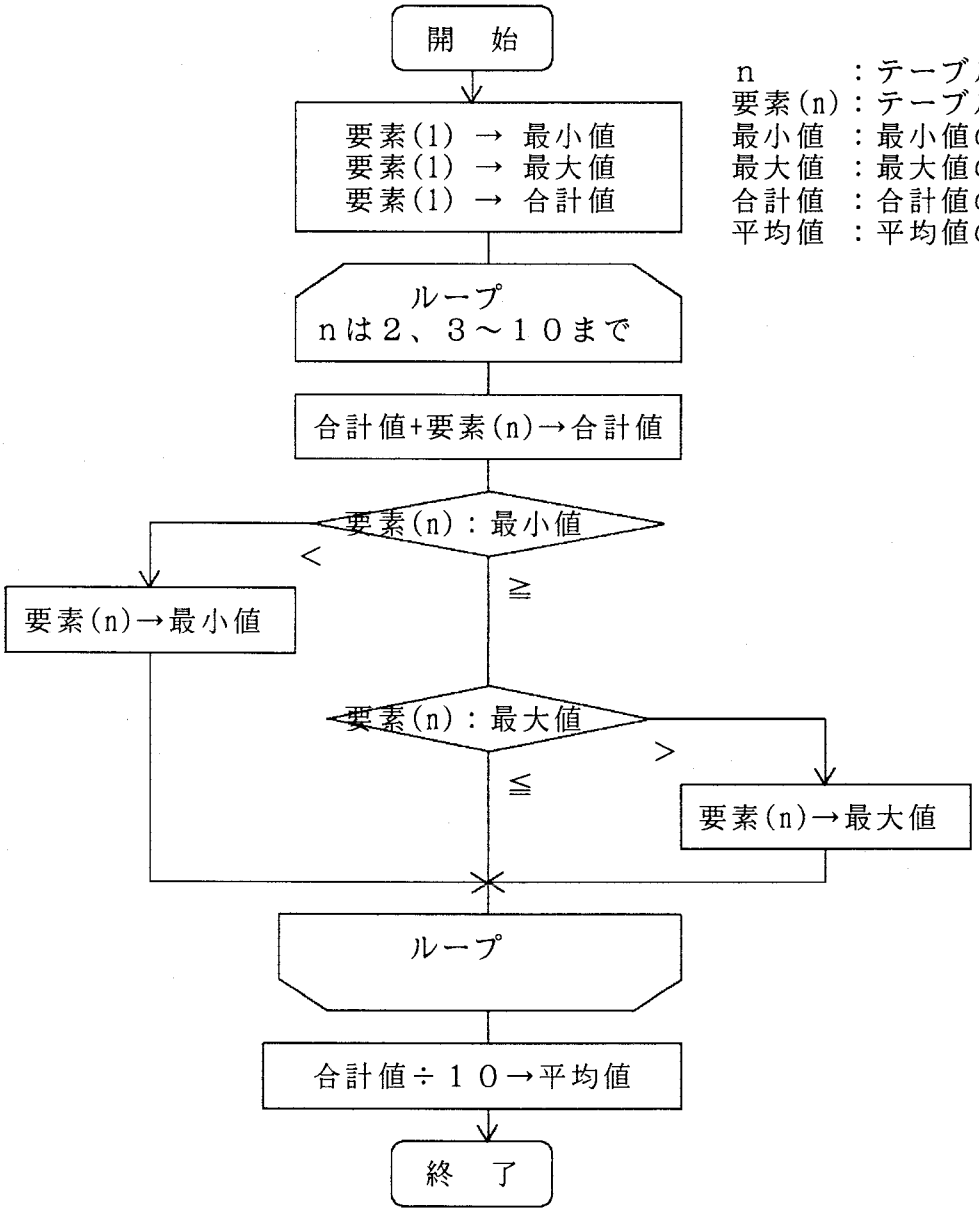
６．３　最大値、最小値、合計、平均

大きさ１０個のテーブル（表）から最大値、最小値、合計値、平均値を求める。

テーブル（表）とは、論理的に連続したデータ項目の集まりである。ＣＯＢＯＬではＯＣＣＵＲＳ句で定義する。テーブル要素（表要素）は、テーブル中の１個のデータ項目をいう。



流れ図は次のとおり。



n　　：テーブルを検索する添字
要素(n)：テーブル要素
最小値　：最小値のデータ項目名
最大値　：最大値のデータ項目名
合計値　：合計値のデータ項目名
平均値　：平均値のデータ項目名

・最大値、最小値、合計、平均のプログラム例

テーブルの定義の方法

```
01  テーブル.
    02  要素    PICTURE  99    OCCURS  10  TIMES.
```

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.     値.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  N       PICTURE  S9(4)  USAGE  BINARY.
LINKAGE  SECTION.
01  テーブル.
    02  要素       PICTURE  S99    OCCURS  10  TIMES.
01  最小値         PICTURE  S99.
01  最大値         PICTURE  S99.
01  合計値         PICTURE  S9999.
01  平均値         PICTURE  S99.
PROCEDURE  DIVISION  USING  テーブル  最小値  最大値  合計値  平均値.
開始.
    MOVE  要素(1)  TO  最小値.
    MOVE  要素(1)  TO  最大値.
    MOVE  要素(1)  TO  合計値.

    PERFORM  VARYING  N
             FROM  2  BY  1  UNTIL  N  >  10
        COMPUTE  合計値  =  合計値  +  要素(N)
        IF  要素(N)  <  最小値  THEN
            MOVE  要素(N)  TO  最小値
        ELSE  IF  要素(N)  >  最大値  THEN
            MOVE  要素(N)  TO  最大値
        END-IF
    END-PERFORM

    COMPUTE  平均値  =  合計値  /  10.
終了.
    EXIT  PROGRAM.
```

注意点は、最小値、最大値、合計値をゼロにして、テーブルに最初から終わりまでのループにした場合は、データの値がマイナスの時に正しく処理されない。

（例１）

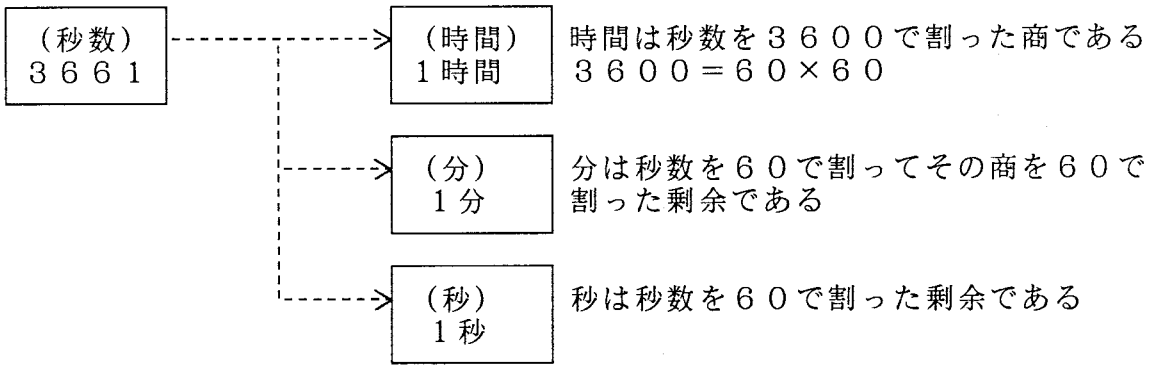
```
MOVE  要素(1)  TO 最小値.
MOVE  要素(1)  TO 最大値.
MOVE  要素(1)  TO 合計値.

PERFORM  VARYING  N
   FROM  2  BY 1 UNTIL N > 10
```

（例２）

```
MOVE  ZERO  TO  最小値.
MOVE  ZERO  TO  最大値.
MOVE  ZERO  TO  合計値.

PERFORM  VARYING  N
   FROM  1  BY 1 UNTIL N > 10
```

６．４　時間の変換

秒数を時、分、秒に変換する。



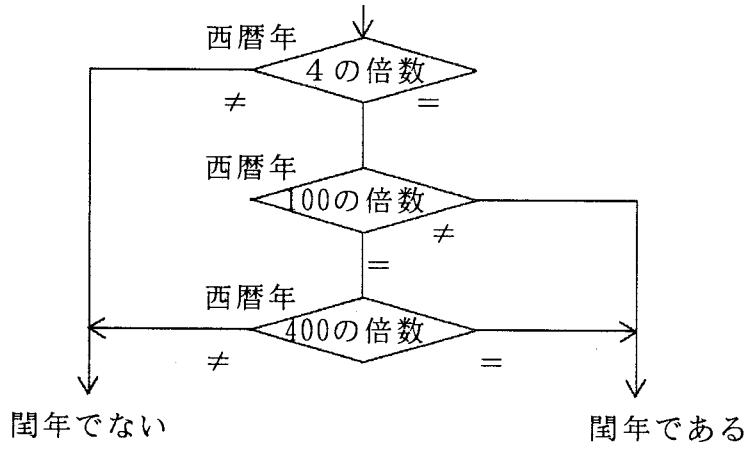
時間変換のプログラム例は、次のとおり。

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.   時間.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  TEMP     PIC  9(4).
LINKAGE  SECTION.
01  秒数     PIC  9(4).
01  時       PIC  9(2).
01  分       PIC  9(2).
01  秒       PIC  9(2).
PROCEDURE  DIVISION  USING  秒数,  時,  分,  秒.
開始.
    COMPUTE  時  =  秒数  /  3600.
    DIVIDE  60  INTO  秒数  GIVING  TEMP  REMAINDER  秒.
    DIVIDE  60  INTO  TEMP  GIVING  TEMP  REMAINDER  分.
終了.
    EXIT  PROGRAM.
```

６．５　閏年の判定

西暦年が閏年かどうか判定する。

閏年の判定は、次のとおり。
①西暦年が４の倍数でない場合は、閏年ではない。
②西暦年が４の倍数の場合は、閏年である。ただし、西暦年が１００の倍数であり４００の倍数でない場合は閏年ではない。



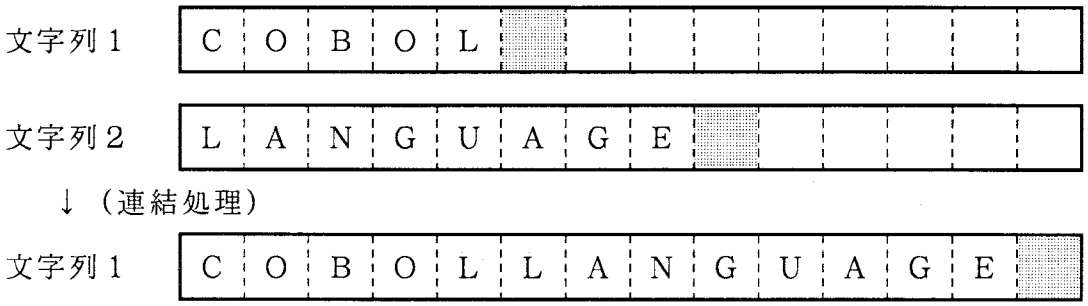
閏年の判定のプログラム例は、次のとおり。

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  閏年.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  TEMP     PIC  9(4).
01  余り     PIC  9(4).
LINKAGE  SECTION.
01  西暦     PIC  9(4).
01  判定     PIC  X(3).
PROCEDURE  DIVISION  USING  西暦,  判定.
開始.
    DIVIDE  4  INTO  西暦  GIVING  TEMP  REMAINDER  余り.
    IF  余り  NOT  ZERO  THEN
        MOVE  "NO"  TO  判定
    ELSE
        DIVIDE  100  INTO  西暦  GIVING  TEMP  REMAINDER  余り
        IF  余り  NOT  ZERO  THEN
            MOVE  "YES"  TO  判定
        ELSE
            DIVIDE  400  INTO  西暦  GIVING  TEMP  REMAINDER  余り
            IF  余り  =  ZERO  THEN
                MOVE  "YES"  TO  判定
            ELSE
                MOVE  "NO"  TO  判定
            END-IF
        END-IF
    END-IF.
終了.
    EXIT  PROGRAM.
```

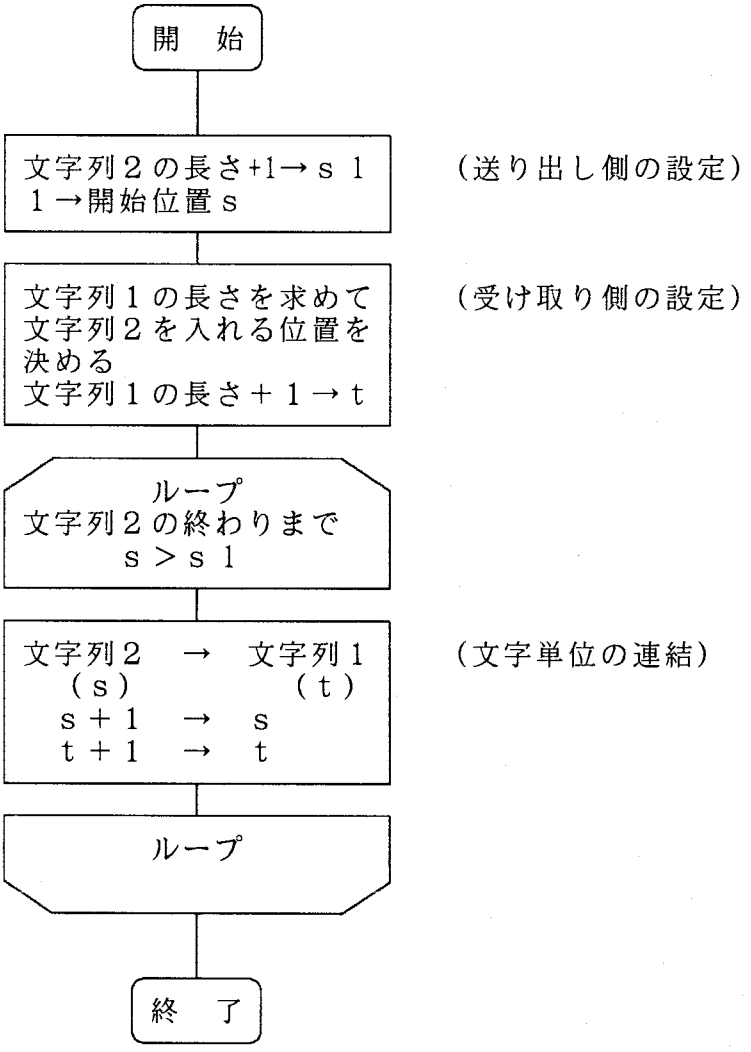
６．６　文字列処理

文字列の連結

文字列１の後に文字列２を連結する。文字列の終わりを示す文字は空白とする。



流れ図は次のとおり。



注：文字列１は文字列２を連結しても領域に納まることが前提である。

文字列の連結プログラム例は、次のとおり。

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.     連結.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  PTR          PICTURE  S9(4)  USAGE  BINARY.
LINKAGE  SECTION.
01  文字列１.
    02  文字１   PICTURE  X(1)    OCCURS  14  TIMES.
01  文字列２     PICTURE  X(14).
PROCEDURE  DIVISION  USING  文字列１  文字列２.
開始.
    MOVE  ZERO  TO  PTR
    INSPECT 文字列１  TALLYING  PTR
                      FOR  CHARACTERS  BEFORE  INITIAL  SPACE
    COMPUTE  PTR  =  PTR  +  1
    STRING  文字列２  DELIMITED  BY  SPACE
                      INTO  文字列１  WITH  POINTER  PTR
            ON  OVERFLOW        CONTINUE
            NOT  ON  OVERFLOW   MOVE  SPACE  TO  文字１(PTR)
    END-STRING.
終了.
    EXIT PROGRAM.
```

文字列の有効データの長さは、ＩＮＳＰＥＣＴ文で求める。区切り文字は、ＩＮＩＴＩＡＬ句で指定する。

Ｃ言語には、文字列の連結用の関数が標準に用意されている。

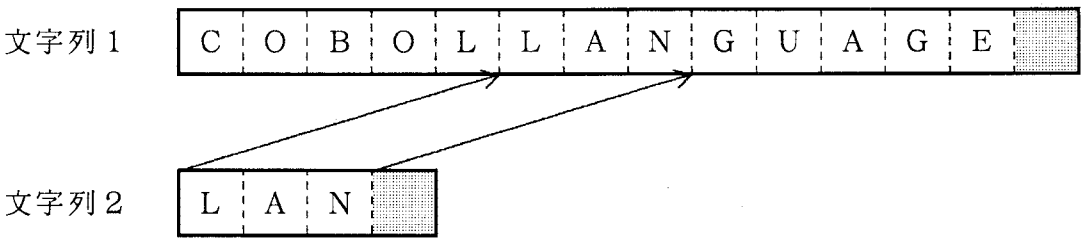
```
    char  string1[14];
    char  string2[14];
            :
          strcat(string1,string2);     ←string2をstring1に付け加える
```

ＦＯＲＴＲＡＮには、文字列演算子（character operator）がある。連結結果は、ｘ１の値の右側にｘ２の値を連結した値をもち、ｘ１とｘ２の長さの合計をもつ文字列である。

```
                ｘ１／／ｘ２          ←ｘ１とｘ２を連結する
```

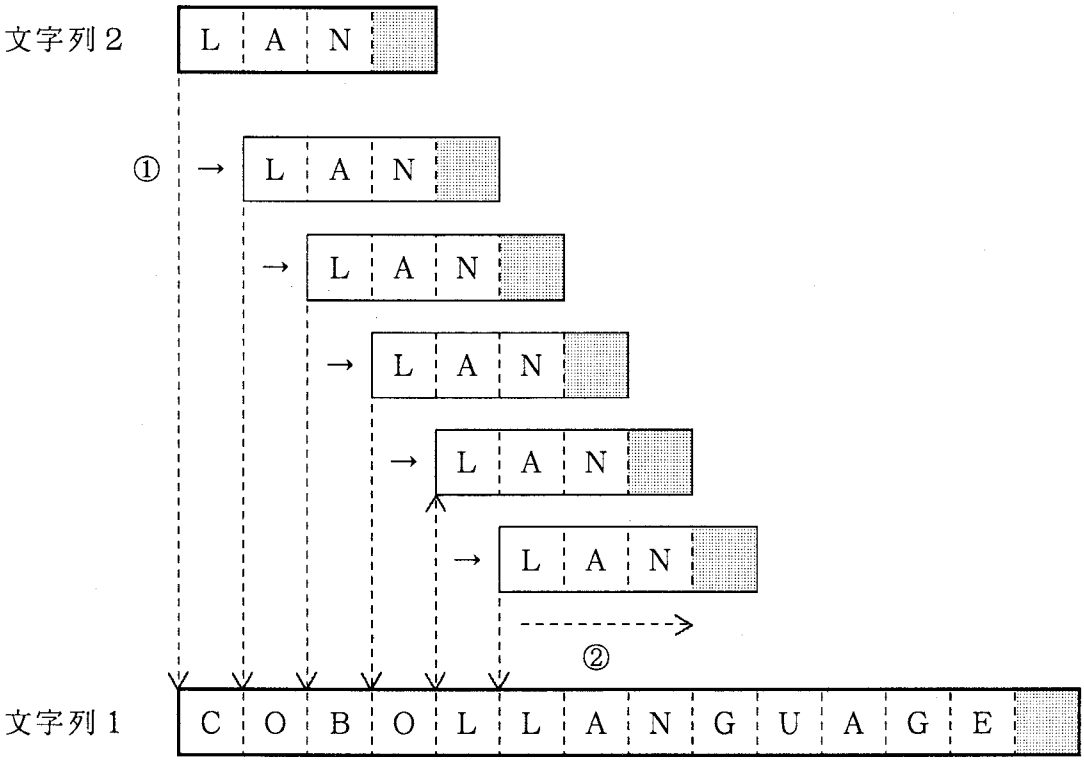
６．７　文字列の検索

文字列１に文字列２が存在するか検査する。



考え方は次のとおり。

文字列１の最初の文字から文字列２の最初の文字と一文字ずつ比較する。
①一致しない場合は、文字列１の位置をずらして文字列２の最初の文字と比較する。
②一致した場合は、それぞれの位置をずらして次の一文字を比較する。



文字列の検索プログラム例は、次のとおり。

存在した場合は最初に現れた位置を返す。存在しない場合は、０を返す。
文字列の終わりを示す文字は空白とする。

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.     検索.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  長さ１          PICTURE  S9(4).
01  長さ２          PICTURE  S9(4).
01  位置１          PICTURE  S9(4).
01  位置２          PICTURE  S9(4).
01  X               PICTURE  S9(4).
01  終了SW          PICTURE  X      VALUE  "0".
    88  一致        VALUE  "1".
LINKAGE  SECTION.
01  文字列１.
    02  文字１       PICTURE  X(1)  OCCURS  100  TIMES.
01  文字列２.
    02  文字２       PICTURE  X(1)  OCCURS  100  TIMES.
01  一致位置         PICTURE  9(3).
PROCEDURE  DIVISION  USING  文字列１  文字列２  一致位置.
開始.
    MOVE  ZERO  TO  長さ１,  長さ２
    INSPECT 文字列１  TALLYING  長さ１
                      FOR  CHARACTERS  BEFORE  INITIAL  SPACE
    INSPECT 文字列２  TALLYING  長さ２
                      FOR  CHARACTERS  BEFORE  INITIAL  SPACE
    MOVE    0  TO  一致位置
    MOVE  "0"  TO  終了SW.
    MOVE    1  TO  位置１.
    PERFORM  UNTIL  一致  OR  長さ２  >  長さ１  -  位置１  +  1
        MOVE  1       TO  位置２
        MOVE  位置１  TO  X
        PERFORM  UNTIL  位置２  >  長さ２
                     OR  文字１(X)  NOT  =  文字２(位置２)
            COMPUTE  位置２  =  位置２  +  1
            COMPUTE  X       =  X       +  1
        END-PERFORM
        IF  位置２  >  長さ２  THEN
            SET  一致     TO  TRUE
            MOVE  位置１  TO  一致位置
        ELSE
            COMPUTE  位置１  =  位置１  +  1
        END-IF
    END-PERFORM.
終了.
    EXIT  PROGRAM.
```

Ｃ言語には、文字列の検索用の関数が標準に用意されている。

```
    char  string1[100];
    char  string2[100];
            :
          strstr(string1,string2);     ←string1をstring2で検索する
```

６．８　基数変換

（１）２進数→１０進数への変換

２進数の各桁ごとに数値と重みを掛けて得られる各数値を加算し、結果を得る。

例　２進数１０１．０１１を１０進数に変換する

$$\begin{array}{cccccccccc} 2^{2} & 2^{1} & 2^{0} & & 2^{-1} & 2^{-2} & 2^{-3} & \leftarrow \text{各桁の重み} \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \\ (1 & 0 & 1 & . & 0 & 1 & 1)_{2} & \end{array}$$

$$\begin{aligned} &= 1\times2^{2}+0\times2^{1}+1\times2^{0}+0\times2^{-1}+1\times2^{-2}+1\times2^{-3} \\ &= 1\times4+0\times2+1\times1+0\times1/2+1\times1/4+1\times1/8 \\ &= 4+0+1+0+1/4+1/8 \\ &= 4+1+0.25+0.125 \\ &= 5.375 \end{aligned}$$

プログラム例は、次のとおり。（但し、整数部の変換のみ）

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  BINDEC.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  ｎ乗数            PICTURE  9(4) USAGE BINARY.
LINKAGE SECTION.
01  ２進数.
    02  ２進数字      PICTURE  9(1)  OCCURS  10 TIMES
                                      INDEXED BY K.
01  １０進数          PICTURE  9(10).
PROCEDURE DIVISION USING  ２進数  １０進数.
開始.
    MOVE ZERO TO １０進数
    MOVE 1    TO ｎ乗数
    PERFORM  WITH TEST BEFORE VARYING K FROM 10 BY -1
                                        UNTIL  K = 0
        COMPUTE １０進数 = １０進数 + ２進数字(K) *  ｎ乗数
        COMPUTE ｎ乗数   = ｎ乗数 * 2
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

次のようにすれば「ｎ乗」は使用しないで済む。

```
PROCEDURE DIVISION USING  ２進数  １０進数.
開始.
    MOVE ZERO TO １０進数
    PERFORM  WITH TEST BEFORE VARYING K FROM 1 BY 1
                                        UNTIL  K > 10
        COMPUTE １０進数 = 2 * １０進数 + ２進数字(K)
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

（２）１０進数→２進数への変換

①整数部の変換

1)１０進数の整数部を２進数の基数である２で割り、商と余り（０または１）を求める。
2)上で得られた商をさらに２で割り、商と余りを求める。これを商が０になるまで繰り返す。
3)計算が終了したら、余りを得た順と逆順序に上位から並べる。
4)これが求める２進数である。

例　１０進数１９を２進数にする。

	（商）		余り		
2)	19				１９を２で割る
2)	9	・・・・・・・	1	↑	商９余り１，さらに９を２で割る
2)	4	・・・・・・・	1	↑	商４余り１，さらに４を２で割る
2)	2	・・・・・・・	0	↑	商２余り０，さらに２を２で割る
2)	1	・・・・・・・	0	↑	商１余り０，さらに１を２で割る
	0		1	↑	商０余り１．

求める２進数は、１００１１となる。

プログラム例は、次のとおり。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  DECBIN.
ENVIRONMENT DIVISION.
DATA DIVISION.
LINKAGE SECTION.
01  １０進数          PICTURE  9(10).
01  ２進数.
    02  ２進数字      PICTURE  9(1)  OCCURS  10 TIMES
                                     INDEXED BY K.
PROCEDURE DIVISION USING  １０進数  ２進数.
開始.
    MOVE ZERO TO ２進数
    PERFORM  WITH TEST BEFORE VARYING K FROM 10 BY -1
                              UNTIL  K = 0 OR １０進数 = 0
        DIVIDE  2  INTO １０進数 GIVING １０進数
                                 REMAINDER ２進数字 (K)
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

（３）８進数→１０進数への変換

８進数の各桁ごとに数値と重みを掛けて得られる各数値を加算し、結果を得る。

例　８進数２７５．１５を１０進数に変換する

$8^2\ 8^1\ 8^0\quad 8^{-1}8^{-2}$　　← 各桁の重み
↓　↓　↓　　↓　↓
$(2\quad 7\quad 5\quad .\ 1\quad 5)_8$
$= 2\times8^2+7\times8^1+5\times8^0+1\times8^{-1}+5\times8^{-2}$
$= 2\times64+7\times8+5\times1+1\times1/8+5\times1/64$
$= 128+56+5+0.125+0.078125$
$= 189.203125$

プログラム例は、次のとおり。（但し、整数部の変換のみ）

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  OCTDEC.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  ｎ乗数            PICTURE  9(4) USAGE BINARY.
LINKAGE SECTION.
01  ８進数.
    02  ８進数字      PICTURE  9(1)  OCCURS  10 TIMES
                                      INDEXED BY K.
01  １０進数          PICTURE  9(10).
PROCEDURE DIVISION USING  ８進数  １０進数.
開始.
    MOVE ZERO TO １０進数
    MOVE 1    TO ｎ乗数
    PERFORM  WITH TEST BEFORE VARYING K FROM 10 BY -1
                                        UNTIL  K = 0
        COMPUTE １０進数 = １０進数 + ８進数字(K) *  ｎ乗数
        COMPUTE ｎ乗数   = ｎ乗数 * 8
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

（４）１０進数→８進数への変換

①整数部の変換

1)１０進数の整数部を８進数の基数である８で割り、商と余りを求める。
2)上で得られた商をさらに８で割り、商と余りを求める。これを商が０になるまで繰り返す。
3)計算が終了したら、余りを得た順と逆順序に上位から並べる。
4)これが求める８進数である。

例　１０進数１４７を８進数にする。

```
          (商)          余り
８)  １４７
８)    １８   ･･･････   ３ ↑   １４７を８で割る
８)     ２   ･･･････   ２ ↑   商１８余り３，さらに１８を８で割る
           ０           ２ ↑   商２余り２，さらに２を８で割る
                               商０余り２．
```

求める８進数は、２２３となる。

プログラム例は、次のとおり。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  DECOCT.
ENVIRONMENT DIVISION.
DATA DIVISION.
LINKAGE SECTION.
01  １０進数          PICTURE  9(10).
01  ８進数.
    02  ８進数字      PICTURE  9(1)  OCCURS  10 TIMES
                                     INDEXED BY K.
PROCEDURE DIVISION USING  １０進数  ８進数.
開始.
    MOVE ZERO TO ８進数.
    PERFORM  WITH TEST BEFORE VARYING K FROM 10 BY -1
                              UNTIL  K = 0 OR １０進数 = 0
        DIVIDE  8  INTO １０進数 GIVING １０進数
                                REMAINDER ８進数字 (K)
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

（５）１６進数→１０進数への変換

１６進数の各桁ごとに数値と重みを掛けて得られる各数値を加算し、結果を得る。

例　１６進数Ｄ３Ａ．Ｃ５を１０進数に変換する

$$
\begin{array}{l}
16^{2}\ 16^{1}\ 16^{0}\quad 16^{-1}\quad 16^{-2} \quad \leftarrow \text{各桁の重み} \\
\quad\downarrow\quad\ \downarrow\quad\ \downarrow\quad\ \ \downarrow\quad\ \ \downarrow \\
(\ D\quad\ 3\quad\ A\ .\ C\quad\ 5)_{16} \\
=13\times16^{2}+3\times16^{1}+10\times16^{0}+12\times16^{-1}+5\times16^{-2} \\
=13\times256+3\times16+10\times1+12\times1/16+5\times1/256 \\
=3328\ +\ 48\ +\ 10\ +0.75\ +0.0195312 \\
=\ 3386.7695312
\end{array}
$$

プログラム例は、次のとおり。（但し、整数部の変換のみ）

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  HEXDEC.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  ｎ乗数            PICTURE  9(4) USAGE BINARY.
01  DEC               PICTURE  9(4) USAGE BINARY.
LINKAGE SECTION.
01  １６進数.
    02  １６進数字    PICTURE  X(1)  OCCURS  10 TIMES
                                      INDEXED BY K.
01  １０進数          PICTURE  9(10).
PROCEDURE DIVISION USING  １６進数  １０進数.
開始.
    MOVE ZERO TO １０進数
    MOVE 1    TO ｎ乗数
    PERFORM  WITH TEST BEFORE VARYING K FROM 10 BY -1
                                        UNTIL  K = 0
        EVALUATE １６進数字(K)
        WHEN "0"   MOVE 0  TO DEC
        WHEN "1"   MOVE 1  TO DEC
        WHEN "2"   MOVE 2  TO DEC
        WHEN "3"   MOVE 3  TO DEC
        WHEN "4"   MOVE 4  TO DEC
        WHEN "5"   MOVE 5  TO DEC
        WHEN "6"   MOVE 6  TO DEC
        WHEN "7"   MOVE 7  TO DEC
        WHEN "8"   MOVE 8  TO DEC
        WHEN "9"   MOVE 9  TO DEC
        WHEN "A"   MOVE 10 TO DEC
        WHEN "B"   MOVE 11 TO DEC
        WHEN "C"   MOVE 12 TO DEC
        WHEN "D"   MOVE 13 TO DEC
        WHEN "E"   MOVE 14 TO DEC
        WHEN "F"   MOVE 15 TO DEC
        WHEN OTHER MOVE 0  TO DEC
        END-EVALUATE
        COMPUTE １０進数 = １０進数 + DEC  * ｎ乗数
        COMPUTE ｎ乗数   = ｎ乗数 * 16
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

（６）１０進数→１６進数への変換

①整数部の変換

1)１０進数の整数部を１６進数の基数である１６で割り、商と余りを求める。
2)上で得られた商をさらに１６で割り、商と余りを求める。これを商が０になるまで繰り返す。
3)計算が終了したら、余りを得た順と逆順序に上位から並べる。
4)これが求める１６進数である。

例　１０進数３５１を１６進数にする。

```
          (商)          余り
１６)  ３５１                      ３５１を１６で割る
１６)    ２１     ･･･  １５  ↑     商２１余り１５，さらに２１を１６で割る
１６)      １     ･･･    ５  ↑     商１余り５，さらに１を１６で割る
            ０           １  ↑     商０余り１．
```

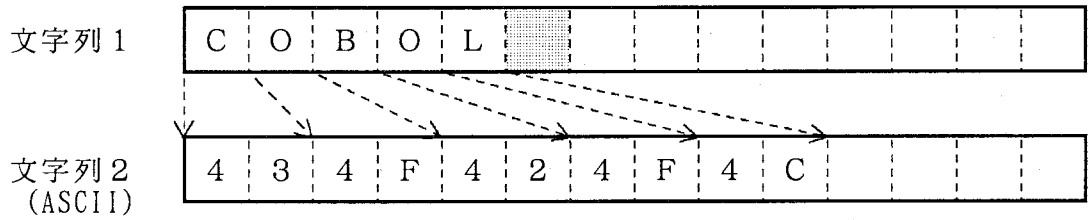
求める１６進数は、１５Ｆとなる。

プログラム例は、次のとおり。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  DECHEX.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  余り              PICTURE  9(1).
01  HEX               PICTURE  9(4)  USAGE BINARY.
01  １６進文字列      VALUE  "0123456789ABCDEF".
    02 １６進文字     PICTURE  X(1)  OCCURS 16 TIMES.

LINKAGE SECTION.
01  １０進数          PICTURE  9(10).
01  １６進数.
    02  １６進数字      PICTURE  X(1)  OCCURS  10 TIMES
                                       INDEXED BY K.
PROCEDURE DIVISION USING  １０進数  １６進数.
開始.
    MOVE ZERO TO １６進数.
    PERFORM  WITH TEST BEFORE VARYING K FROM 10 BY -1
                              UNTIL  K = 0 OR １０進数 = 0
        DIVIDE  16  INTO １０進数 GIVING １０進数
                                  REMAINDER HEX
        COMPUTE HEX = HEX + 1
        MOVE １６進文字(HEX) TO １６進数字(K)
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

（７）　内部形式データ変換

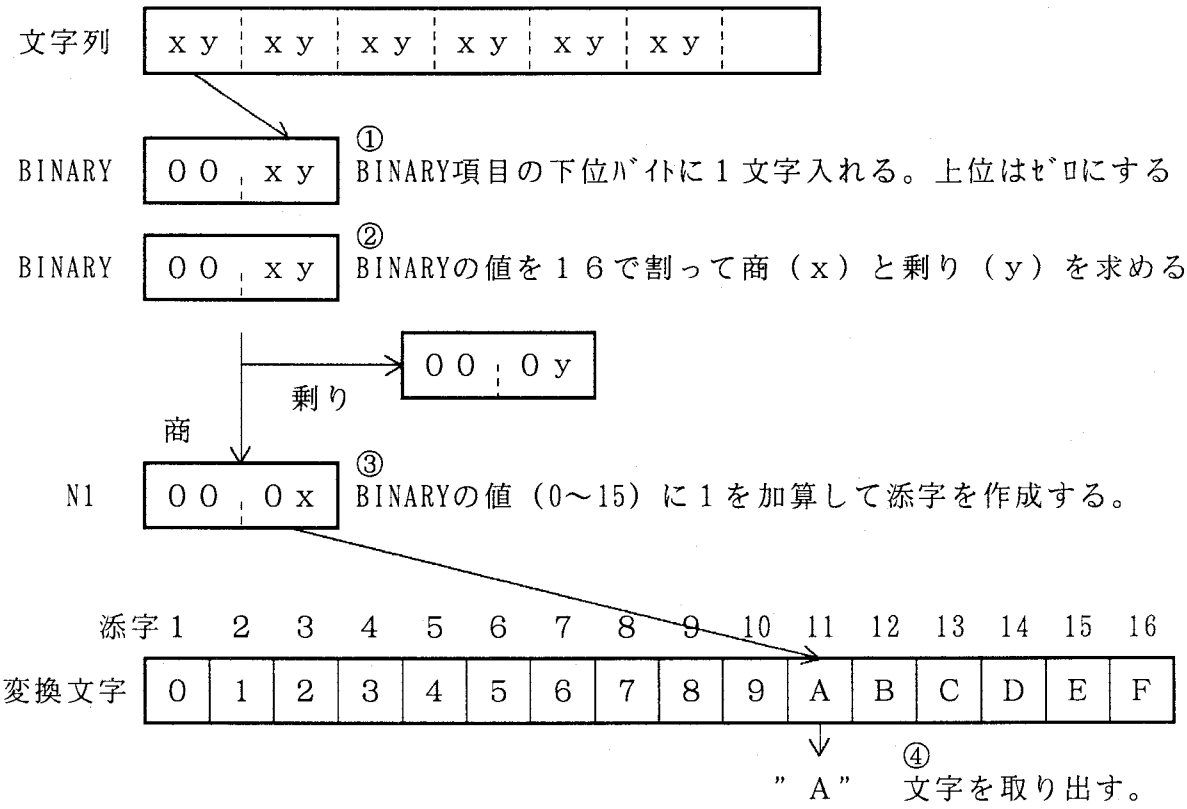
文字列（文字、数字等）のデータを１６進数の文字列に変換する。



１６進数表現（０～９、Ａ～Ｆ）

考え方は次のとおり。
　ＣＯＢＯＬのＢＩＮＡＲＹ項目とＤＩＳＰＬＡＹ項目を同じ領域に割り付けて、用途に応じてデータ項目名を使い分ける。

```
01  BIN             PICTURE  9(4)  USAGE  BINARY.
01  CHARS           REDEFINES  BIN.
    03  X1          PICTURE  X.
    03  X2          PICTURE  X.
```

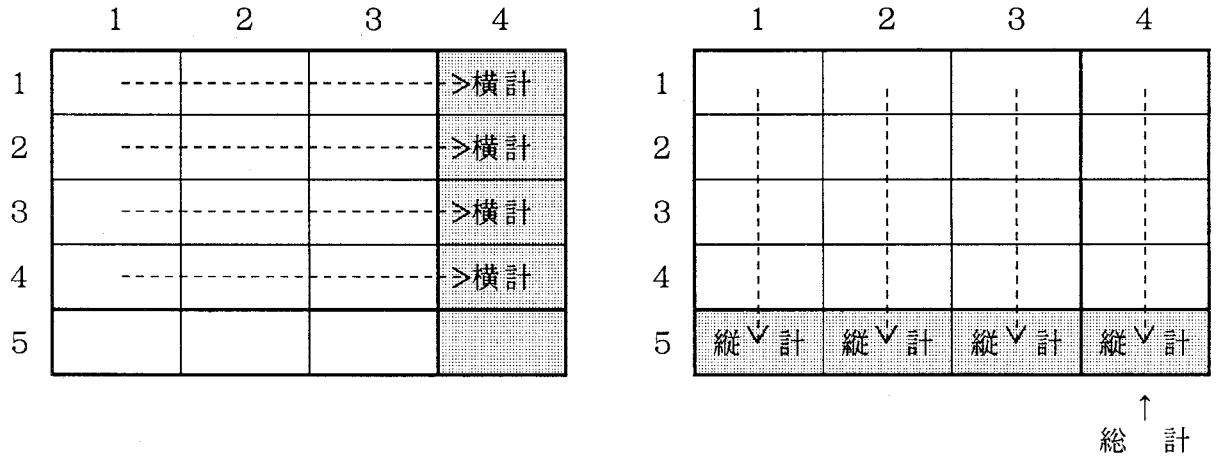


文字列データの１６進数文字列への変換プログラム例は、次のとおり。

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.   １６進数.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  変換テーブル  VALUE  "0123456789ABCDEF".
    03  変換文字  PICTURE  X  OCCURS  16  TIMES.
01  BIN           PICTURE  9(4)  USAGE  BINARY.
01  CHARS         REDEFINES  BIN.
    03  X1        PICTURE  X.
    03  X2        PICTURE  X.
01  N1            PICTURE  9(4)  USAGE  BINARY.
01  N2            PICTURE  9(4)  USAGE  BINARY.
01  S             PICTURE  9(4)  USAGE  BINARY.
01  T             PICTURE  9(4)  USAGE  BINARY.
LINKAGE  SECTION.
01  文字列１.
    03  文字１    PICTURE  X  OCCURS  100  TIMES.
01  文字列２.
    03  文字２    PICTURE  X  OCCURS  200  TIMES.
PROCEDURE DIVISION  USING  文字列１  文字列２.
開始.
    MOVE  1  TO  T
    PERFORM  VARYING  S  FROM  1  BY  1  UNTIL  文字１(S)  =  SPACE
        MOVE  ZERO  TO  BIN
        MOVE  文字１(S)  TO  X2
        DIVIDE  BIN  BY 16  GIVING N1  REMAINDER N2
        COMPUTE  N1  =  N1  +  1
        COMPUTE  N2  =  N2  +  1
        MOVE  変換文字(N1)  TO  文字２(T)
        COMPUTE  T  =  T  +  1
        MOVE  変換文字(N2)  TO  文字２(T)
        COMPUTE  T  =  T  +  1
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

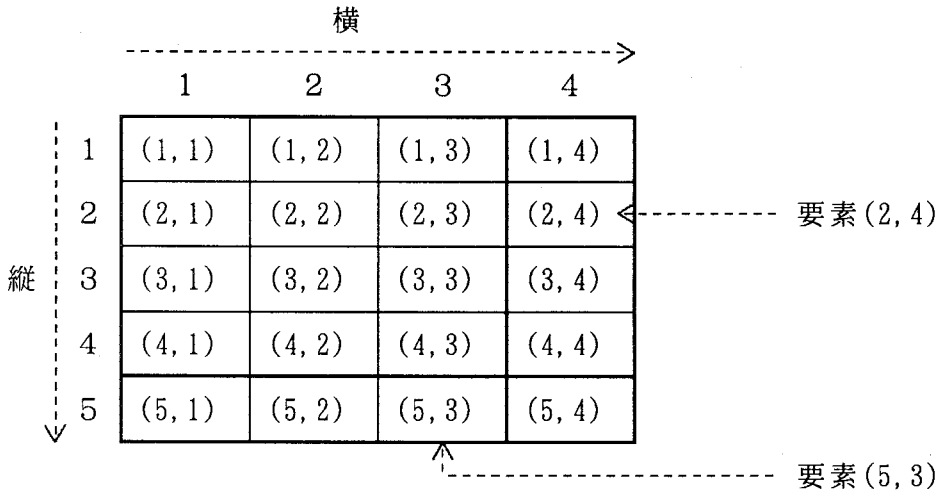
６．９　配列処理

配列の横の行、縦の行について各要素を集計（横計、縦計、総計）する。



ＣＯＢＯＬの配列の定義と割付関係は、次のとおり。

```
01  テーブル.
    02  縦  OCCURS  5  TIMES.
        03  横  OCCURS  4  TIMES
            04  要素  PIC  S9(4).
```



縦計、横計、総計のプログラム例は、次のとおり。

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.   集計.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  X          PICTURE  S9(4)  USAGE  BINARY.
01  Y          PICTURE  S9(4)  USAGE  BINARY.
LINKAGE  SECTION.
01  テーブル.
    02  縦  OCCURS  5  TIMES.
        03  横  OCCURS  4  TIMES.
            04  要素     PICTURE  S9(4).
PROCEDURE  DIVISION  USING  テーブル.
開始.
    PERFORM  VARYING  Y  FROM  1  BY  1  UNTIL  Y  >  4
        PERFORM  VARYING  X  FROM  1  BY  1  UNTIL  X  >  3
            COMPUTE  要素(Y,4)  =  要素(Y,4)  +  要素(Y,X)
        END-PERFORM
    END-PERFORM
    PERFORM  VARYING  X  FROM  1  BY  1  UNTIL  X  >  4
        PERFORM  VARYING  Y  FROM  1  BY  1  UNTIL  Y  >  4
            COMPUTE  要素(5,X)  =  要素(5,X)  +  要素(Y,X)
        END-PERFORM
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

指標名を使う場合は、次のように修正する。

```
WORKING-STORAGE  SECTION.
01  X          PICTURE  S9(4)  USAGE  BINARY.     } 取り除く
01  Y          PICTURE  S9(4)  USAGE  BINARY.
LINKAGE  SECTION.
01  テーブル.
    02  縦  OCCURS  5  TIMES        INDEXED  BY  Y.
        03  横  OCCURS  4  TIMES    INDEXED  BY  X.
            04  要素     PICTURE  S9(4).
```

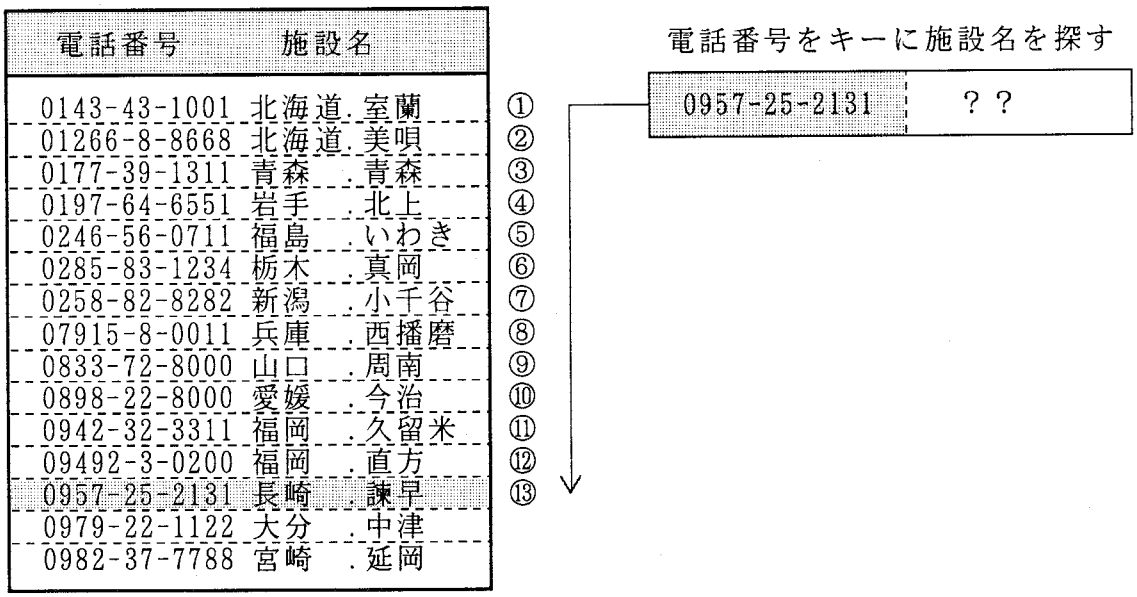
内ＰＥＲＦＯＲＭ（４章参照）ではＡＦＴＥＲ指定は使えない。

```
    PERFORM  VARYING  Y  FROM  1  BY  1  UNTIL  Y  >  4
        AFTER  X  FROM  1  BY  1  UNTIL  X  >  3    ←誤り
            COMPUTE  要素(Y,4)  =  要素(Y,4)  +  要素(Y,X)
        END-PERFORM
    END-PERFORM

          ↓   AFTER指定を使う場合は、手続き名の書き方で記述する

    PERFORM  手続き名 VARYING  Y  FROM  1  BY  1  UNTIL  Y  >  4
                      AFTER    X  FROM  1  BY  1  UNTIL  X  >  3
        :
手続き名.
        COMPUTE  要素(Y,4)  =  要素(Y,4)  +  要素(Y,X)
手続き名2.
```

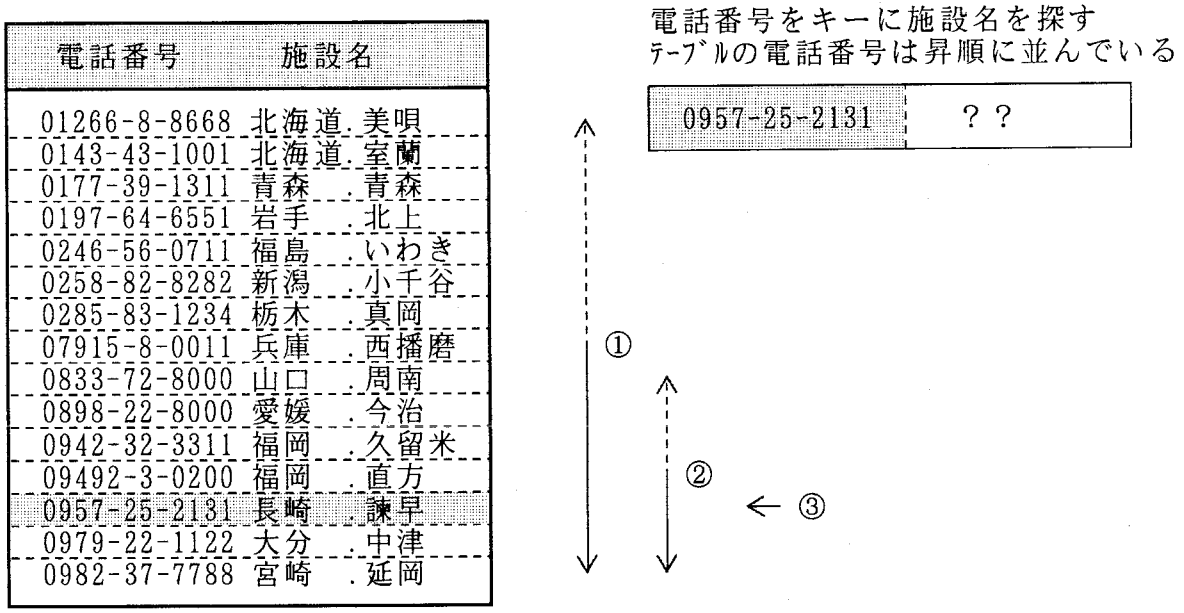
６．１０　表探索

探索は、テーブル等に格納されているデータの中から目的のデータを探し出すことをいう。

逐次探索（sequential　search）は、テーブルを最初から順番に探す方法である。



２分探索（binary　search）は、テーブルの目的のデータが昇順または、降順に並んでいる場合の探索方法である。
テーブルを半分に分けて目的のキーが前半にあるか後半にあるかを定めることを繰り返して探す方法である。



逐次探索のプログラム例は、次のとおり。

データ項目「でんわ」の順序は、どのような順番にならべられていてもよい。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   逐次探索.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  テーブル.
    02  カレッジ     OCCURS  15  TIMES  INDEXED  BY  指標.
        03  でんわ     PICTURE  X(12).
        03  なまえ     PICTURE  X(12).
LINKAGE  SECTION.
01  電話番号     PICTURE  X(12).
01  施設名       PICTURE  X(12).
PROCEDURE  DIVISION  USING  電話番号  施設名.
開始.
    SET  指標  TO  1.
    SEARCH  カレッジ  AT  END  MOVE  "？？？？？？"  TO  施設名
        WHEN  でんわ(指標)  =  電話番号
            MOVE  なまえ(指標)  TO  施設名
    END-SEARCH.
終了.
    EXIT  PROGRAM.
```

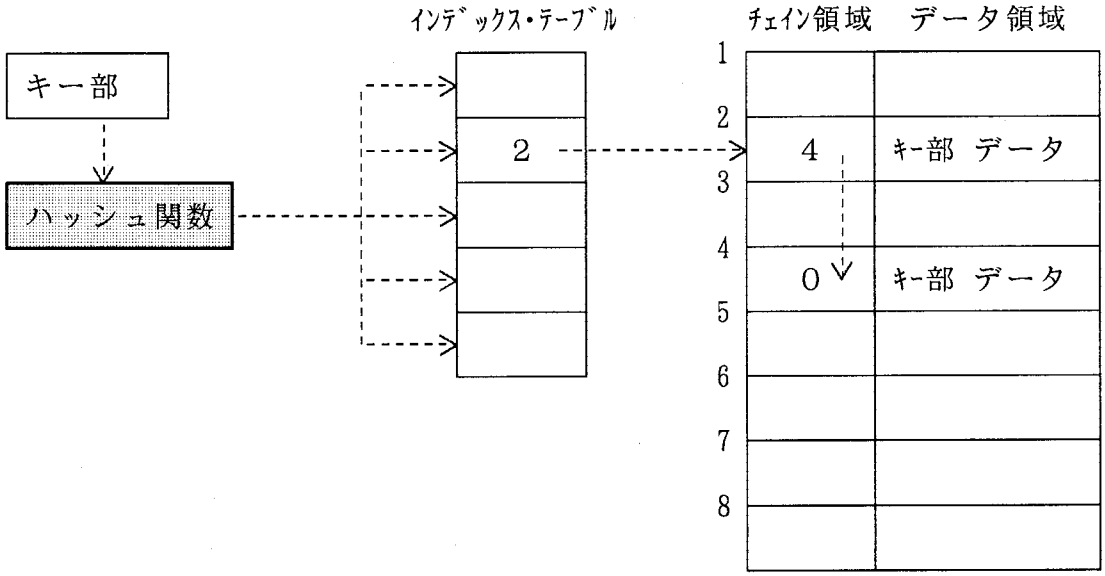
２分探索のプログラム例は、次のとおり。
データ項目「でんわ」の内容は、昇順（ascending）に並んでいること。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   ２分探索.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  テーブル.
    02  カレッジ     OCCURS  15  TIMES
                  ASCENDING  KEY  IS  でんわ  INDEXED  BY  指標.
        03  でんわ     PICTURE  X(12).
        03  なまえ     PICTURE  X(12).
LINKAGE  SECTION.
01  電話番号     PICTURE  X(12).
01  施設名       PICTURE  X(12).
PROCEDURE  DIVISION  USING  電話番号  施設名.
開始.
    SEARCH   ALL   カレッジ  AT  END  MOVE  "？？？？？？"  TO  施設名
        WHEN  でんわ(指標)  =  電話番号
            MOVE  なまえ(指標)  TO  施設名
    END-SEARCH.
終了.
    EXIT  PROGRAM.
```

６．１１　ハッシュ法

テーブルを検索するキー部を、インデックス・テーブルの限られた範囲の番号に対応させる。

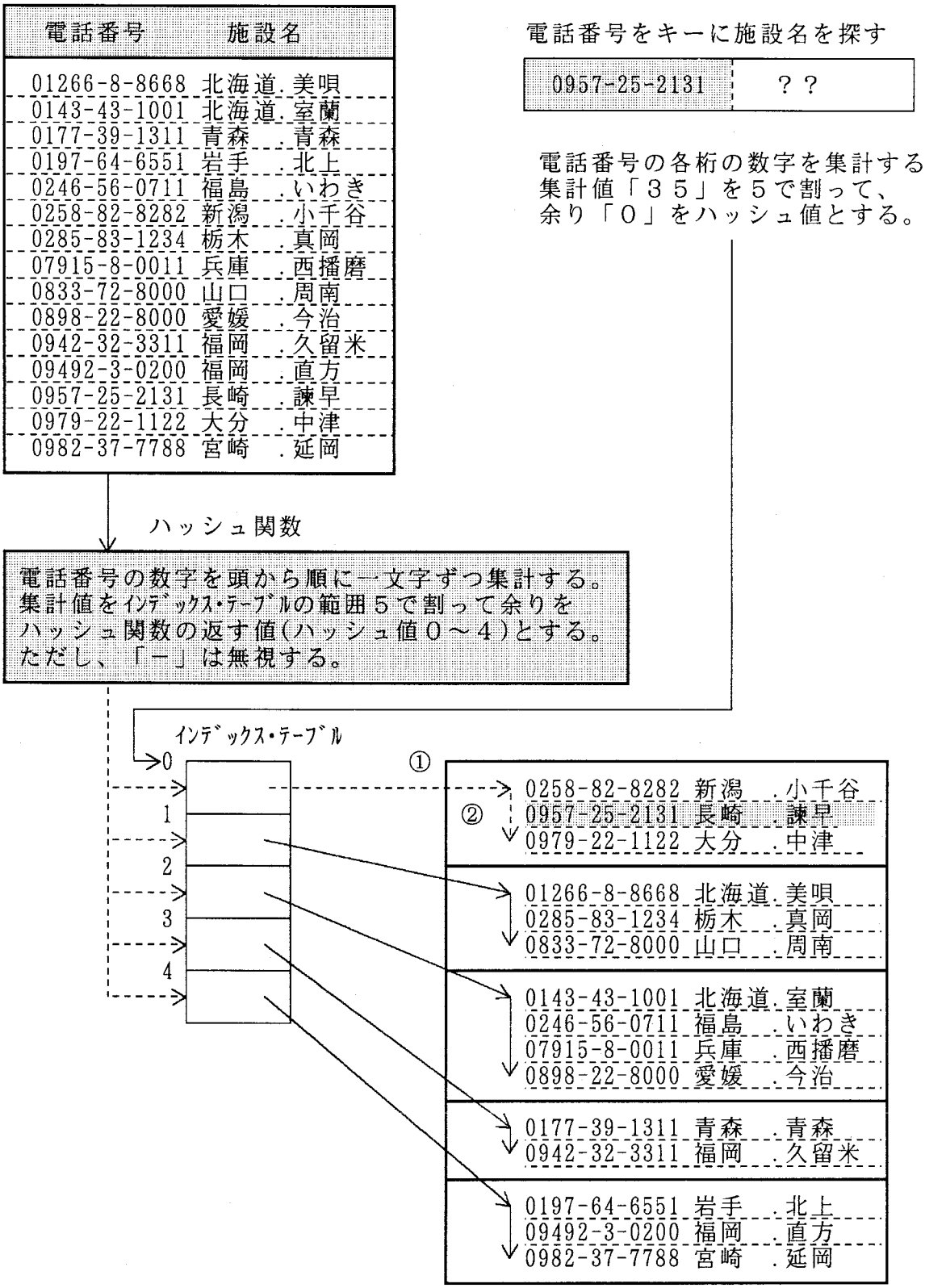
キー部をインデックス・テーブルの範囲に対応させる計算処理を「ハッシュ関数」という。ハッシュ関数が返す値をハッシュ値という。



衝突（コンフリクト）

キー部の値が違っていても同じハッシュ値になってしまうことがある。

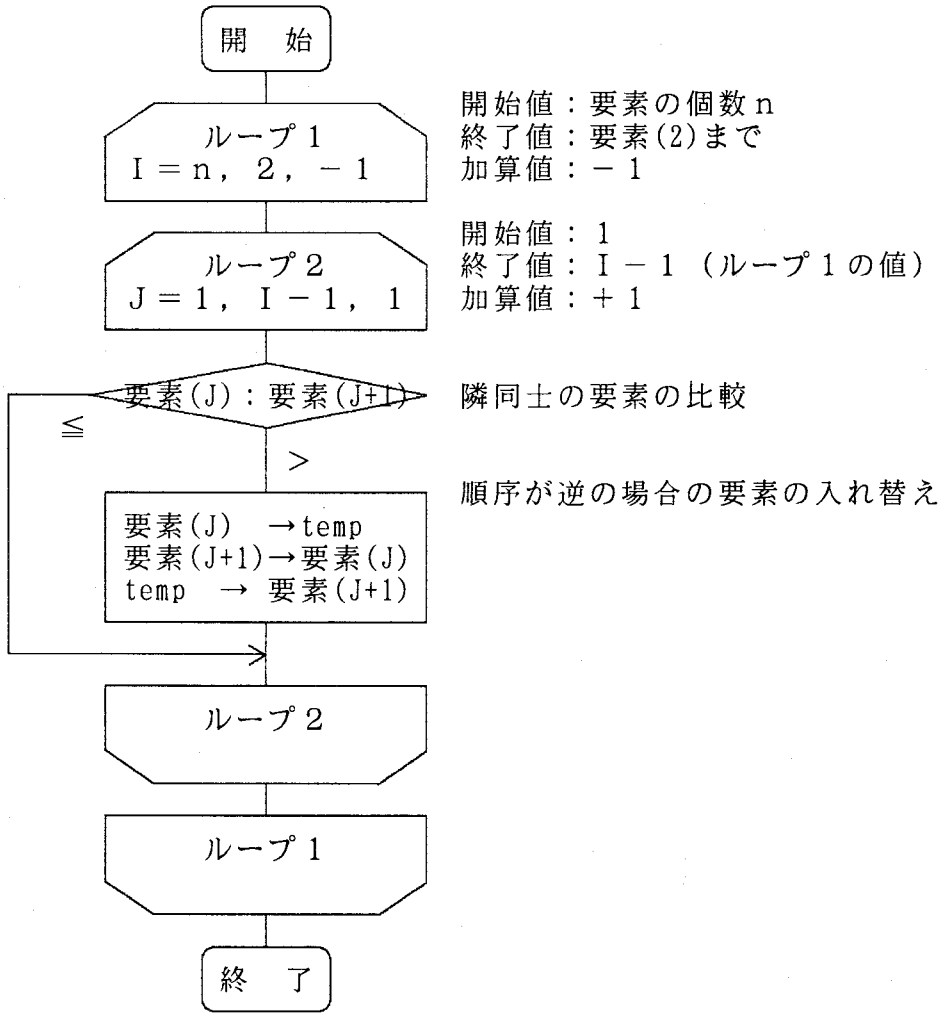
（１）ハッシュ法の登録と探索



６．１２　内部整列法

６．１２．１　基本交換法（バブルソート）

テーブルを先頭から最後まで、隣合う２個の要素を比較して、順序（昇順、降順）が逆であれば入れ替える。次の処理は昇順の場合の例である。



テーブルの内容を昇順に並べ替える。

	9	8	7	6	5	4	3	2	1	0

←実行前

①	8	7	6	5	4	3	2	1	0	9
②	7	6	5	4	3	2	1	0	8	9
③	6	5	4	3	2	1	0	7	8	9
④	5	4	3	2	1	0	6	7	8	9
⑤	4	3	2	1	0	5	6	7	8	9
⑥	3	2	1	0	4	5	6	7	8	9
⑦	2	1	0	3	4	5	6	7	8	9
⑧	1	0	2	3	4	5	6	7	8	9

←隣どうしの入れ替えで最後になる

	0	1	2	3	4	5	6	7	8	9

←実行後

基本交換法のプログラム例は、次のとおり。

１０個の要素のテーブルを昇順に並べかえる

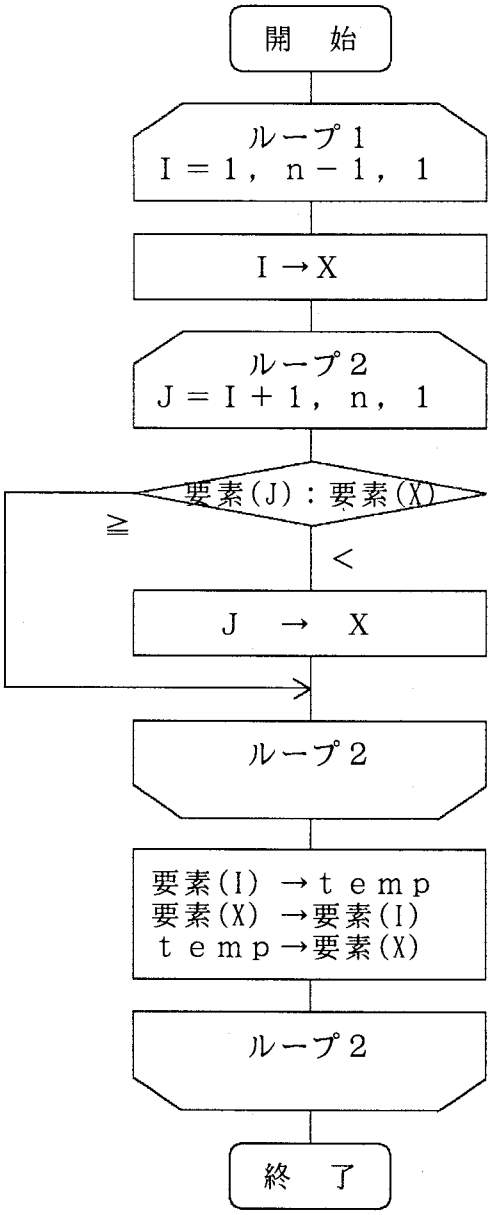
```
IDENTIFICATION  DIVISION.
PROGRAM-ID.     基本交換.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  TEMP      PICTURE  9(3).
LINKAGE  SECTION.
01  テーブル.
    03  要素     PICTURE  9(3)  OCCURS  10  TIMES
                 INDEXED  BY  I,  J.
PROCEDURE  DIVISION  USING  テーブル.
開始.
    PERFORM  VARYING  I  FROM  10  BY  -1  UNTIL  I  <  2
        PERFORM  VARYING  J  FROM  1  BY  1  UNTIL  J  >  I  -  1
            IF  要素(J)  >  要素(J  +  1)  THEN
                MOVE  要素(J)  TO  TEMP
                MOVE  要素(J  +  1)  TO  要素(J)
                MOVE  TEMP     TO  要素(J  +  1)
            END-IF
        END-PERFORM
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

降順（descending）に並べかえる場合は、次のように修正する。

```
    PERFORM  VARYING  I  FROM  10  BY  -1  UNTIL  I  <  2
        PERFORM  VARYING  J  FROM  1  BY  1  UNTIL  J  >  I  -  1
            IF  要素(J)  <  要素(J  +  1)  THEN
                MOVE  要素(J)  TO  TEMP
                MOVE  要素(J  +  1)  TO  要素(J)
                MOVE  TEMP     TO  要素(J  +  1)
            END-IF
        END-PERFORM
    END-PERFORM.
```

６．１２．２　基本選択法

基本選択法は、テーブル中の最小値（昇順）または、最大値（降順）を選び出して、順次、左端の値と交換する。



開始値：１　　（要素の最初から）
終了値：ｎ－１（要素の最後－１まで）
加算値：＋１

開始値：Ｉ＋１
終了値：ｎ（要素の最後まで）
加算値：＋１

順序が逆の場合の入れ替え位置の保管

テーブルの内容を昇順に並べ替える。

	9	8	7	6	5	4	3	2	1	0

←実行前

①	0	8	7	6	5	4	3	2	1	9
②	0	1	5	6	5	4	3	2	8	9
③	0	1	2	6	5	4	3	7	8	9
④	0	1	2	3	1	4	6	7	8	9
⑤	0	1	2	3	4	5	6	7	8	9
⑥	0	1	2	3	4	5	6	7	8	9
⑦	0	1	2	3	4	5	6	7	8	9
⑧	0	1	2	3	4	5	6	7	8	9

←最小値を見つけて左端と入れ換える。そして、左端をずらす。

0	1	2	3	4	5	6	7	8	9

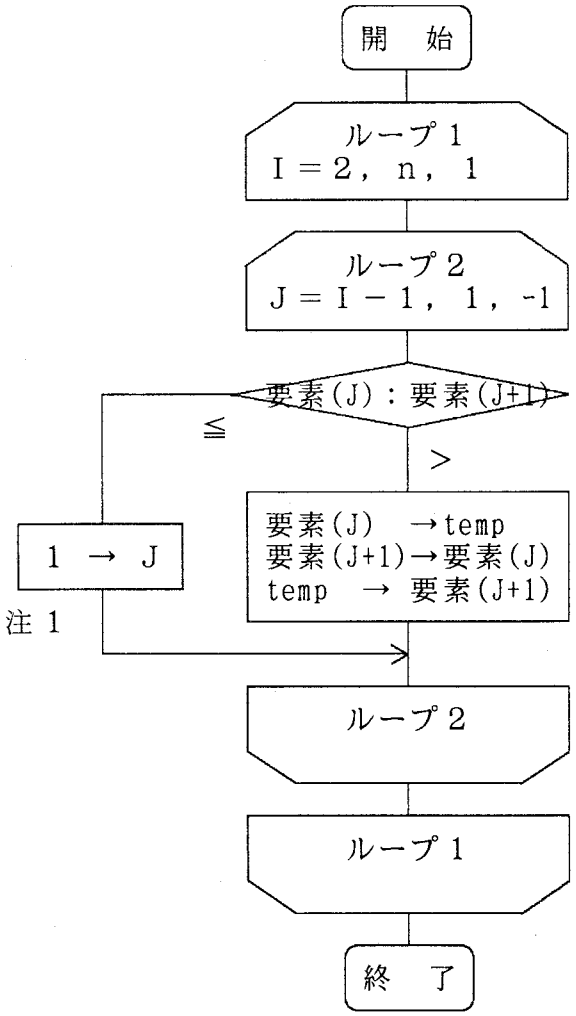
←実行後

基本選択法のプログラム例は、次のとおり。

１０個の要素のテーブルを昇順に並べかえる

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.     基本選択.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  I          PICTURE  S9(4)  USAGE  BINARY.
01  J          PICTURE  S9(4)  USAGE  BINARY.
01  K          PICTURE  S9(4)  USAGE  BINARY.
01  X          PICTURE  S9(4)  USAGE  BINARY.
01  TEMP       PICTURE  9(3).
LINKAGE  SECTION.
01  テーブル.
    03  要素     PICTURE  9(3)  OCCURS  10  TIMES.
PROCEDURE  DIVISION  USING  テーブル.
開始.
    PERFORM  VARYING  I  FROM  1  BY  1  UNTIL  I  >  9
        MOVE   I  TO  X
        COMPUTE  K  =  I  +  1
        PERFORM  VARYING  J  FROM  K  BY  1  UNTIL  J  >  10
            IF  要素(J)  <  要素(X)  THEN
                MOVE  J  TO  X
            END-IF
        END-PERFORM
        MOVE  要素(I)  TO  TEMP
        MOVE  要素(X)  TO  要素(I)
        MOVE  TEMP     TO  要素(X)
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

６．１２．３　基本挿入法



開始値：２　（ループ２
終了値：ｎ　（要素の最後まで）
加算値：＋１

開始値：Ｉ－１
終了値：１　（要素の最初まで）
加算値：－１

比較して順序が合っていたらループ１を直ちに抜けるでる

順序が逆の場合は隣どうしを入れ替える

注１：ループ２を抜けでるため

テーブルの内容を昇順に並べ替える。

9	8	7	6	5	4	3	2	1	0

←実行前

8	9	7	6	5	4	3	2	1	0
7	8	9	6	5	4	3	2	1	0
6	7	8	9	5	4	3	2	1	0
5	6	7	8	9	4	3	2	1	0
4	5	6	7	8	9	3	2	1	0
3	4	5	6	7	8	9	2	1	0
2	3	4	5	6	7	8	9	1	0
1	2	3	4	5	6	7	8	9	0

0	1	2	3	4	5	6	7	8	9

←実行後

基本挿入法のプログラム例は、次のとおり。

１０個の要素のテーブルを昇順に並べかえる

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.     基本挿入.
DATA  DIVISION.
WORKING-STORAGE  SECTION.
01  I          PICTURE  S9(4)  USAGE  BINARY.
01  J          PICTURE  S9(4)  USAGE  BINARY.
01  K          PICTURE  S9(4)  USAGE  BINARY.
01  TEMP       PICTURE  9(3).
LINKAGE  SECTION.
01  テーブル.
    03  要素     PICTURE  9(3)  OCCURS  10  TIMES.
PROCEDURE  DIVISION  USING  テーブル.
開始.
    PERFORM  VARYING  I  FROM  2  BY  1  UNTIL  I  >  10
        COMPUTE  K  =  I  -  1
        PERFORM  VARYING  J  FROM  K  BY  -1  UNTIL  J  <  1
            COMPUTE  K  =  J  +  1
            IF  要素(J)  >  要素(K)  THEN
                MOVE  要素(J)  TO  TEMP
                MOVE  要素(K)  TO  要素(J)
                MOVE  TEMP     TO  要素(K)
            ELSE
                MOVE  1  TO  J
            END-IF
        END-PERFORM
    END-PERFORM.
終了.
    EXIT PROGRAM.
```

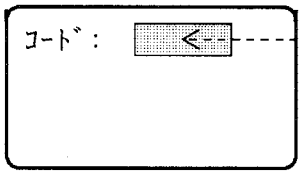
指導上の留意点

◎データチェック
企業などの業務をコンピュータで処理する場合のアプリケーションプログラムは、まずデータ入力の段階からデータの妥当性を確認する処理から始まる。
データを誤りをチェックする方法には次のような種類がある。

・データ発生時点チェック
　サイトチェック、ベリファイチェック
・データ入力時点チェック
　ニューメリックチェック、アルファベティックチェック、リミットチェック、
　シーケンスチェック、フォーマットチェック、カウントチェック
・データ処理時点チェック
　サインチェック、オーバーフローチェック、ダブルレコードチェック
・データ出力時点チェック
　バランスチェック、クロスフットチェック、バッチトータルチェック
・その他のチェック
　パリティチェック、チェックディジットチェック、リダーンダンシーチェック

これらのデータチェックはＣＯＢＯＬなどのプログラミングで行えるが、プログラミングの生産性、保守性、拡張性などを考慮してハードウェアとソフトウェアの持っている機能を充分に利用して行うことに注意する必要がる。

端末の画面



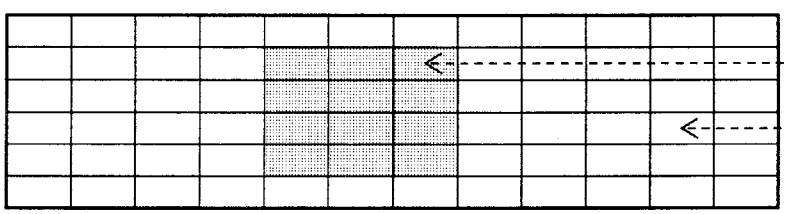
画面のｺｰﾄﾞ・ﾌｨｰﾙﾄﾞには数字のみ入力可とする場合これらは端末画面の定義ソフトウェアに宣言することで数字のチェックができる。

ハードウェアやソフトウェアは常に機能、性能アップが行われており日頃からこれらの情報の収集に心がけることも重要である。

◎データの構造
データチェックをプログラミングで行う場合、データの内部形式とハードウェア機能をよく理解しておくことも非常に重要である。

◎文字列処理
データを端末画面から入力する場合、「数字データ」や「文字データ」であろうと最初の処理はデータを文字列として扱うことが多い。
文字列処理は、１文字単位で区切りなどのチェックを行い、複数文字を結合して論理的なデータにする。
これらの文字列処理をＣＯＢＯＬで行う場合、１文字単位にＩＦやＭＯＶＥで処理できるが、ＩＮＳＰＥＣＴやＵＮＳＴＲＩＮＧ、ＳＴＲＩＮＧを使うことで比較などの分岐処理を少なくし誤りの発生を少なくすることができる。

◎配列処理
データ集計は配列を使用して行う。パソコンで使用されている表計算ソフト（EXCEL、lotus1-2-3など）も全体を大きな配列として処理している。
表計算ソフトでは、配列の要素に各種の複雑な計算式が記述できる。

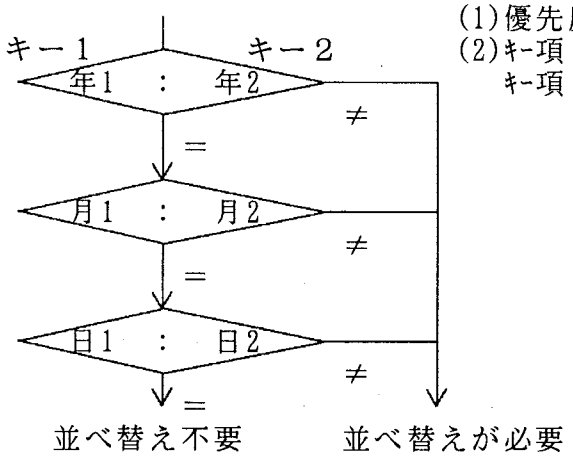


◎内部整列法

データを端末画面に表示する場合など、データを特定の順序に並べ替えてから表示することが多い。
データを並べ替える手段として、汎用の整列プログラム（ソート・マージ）があるが、これは大量データを一括して処理する場合に最も効率的であるが、少量（１画面分程度）のデータの整列にはあまり効率的でない。このように少量のデータを整列する場合は本章に記述した整列法の一つを利用するとよい。
なお、これらの整列法は少量データの整列にとどまらずデータを格納している配列を拡大すれば大量のデータも処理できるが、効率を重視する場合には、大量データの整列には向かないものもある。

また、整列のキー項目が複数の場合もあり、これらのキーの比較方法についても理解しておく必要がある。

３個のキー項目「年、月、日」を並べ替える場合のキー比較の方法は、次のとおり。



(1)優先度の高いｷｰ項目から比較する。
(2)ｷｰ項目が同じ場合のみ次の優先度のｷｰ項目を比較する。