DO 文と 1 次元配列

山本昌志*

2003 年 10 月 14 日

1　はじめに

本日から、教科書の第 3 章に入ります。第 3 章には、

- DO 文によるループ
- 書式付出力 (FORMAT 文)
- 1 次元配列 (DIMENSION)

について書かれています。本日は、その中で DO 文と 1 次元配列について説明します。本日の説明の後、来週の授業では実際にプログラムをしてもらいます。

2　DO 文

コンピュータープログラム上で同じ処理を繰り返す、ただし変数などの値は異なっているようなものをループといいます。これは、以前学習した IF 文と GO TO 文で実現できたことを思い出してください。例えば、ループ構造を使って 1〜N までの和を計算するプログラムは、次のようになります。

```
      PROGRAM SUM N BY USING IF AND GO TO
      INTEGER N,I,S

      I=0
      S=0

      READ(5,*)N

1     I=I+1
      S=S+I                                                    10
      IF(I.LT.N)THEN
        GO TO 1
      ELSE
        WRITE(6,*)S
        STOP
```

*国立秋田工業高等専門学校　電気工学科

```
      ENDIFIF

      END
```

コンピューターは同じようなことを繰り返すのが非常に得意です。このループ構造は、いたるところで出現します。FORTRAN ではループを構造を作る専用の実行文があります。それが、ここで学習する DO 文というものです。先の IF 文と GO TO 文で作成した同じプログラムを DO 文で書くと、以下のようになります。先に比べて、プログラムは非常にすっきりしたものになります。

```
      PROGRAM SUM N BY USING DO
      INTEGER N,I,S

      I=0
      S=0

      READ(5,*)N

      DO 10 I=1,N,1
        S=S+I                                                    10
10    CONTINUE

      WRITE(6,*)S
      STOP

      END
```

それでは、新しい実行文 DO と CONTINUE について、説明します。まずは、先ほどのプログラムのループ構造の説明の図 1 を見てください。DO と CONTINUE は対になって表れ、その間がループになっていることが分かると思います。単純でしょう。

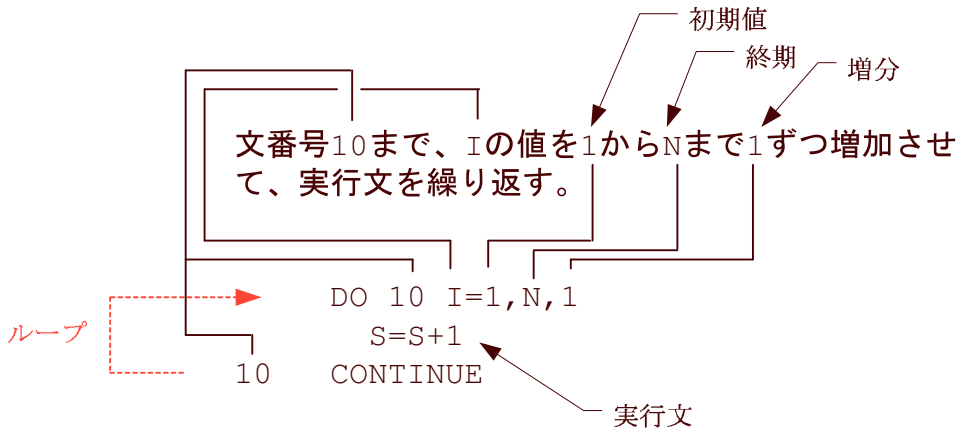

図 1: DO 文を使ったループ構造

ここまでで、DO 文を使ったループの概要が理解できたと思います。もう少し、詳しく、教科書の P.57〜60 で学習しましょう。

3　1 次元配列

3.1　配列とは

コンピューターの仕事は、データの処理です。データとは数字や文字、あるいは絵、音など様々な形が有ります。いずれにしても、どのようなデータでもコンピューターで扱うときには数字になります。とりあえず、データといえば数字しか扱いませんが、ほかもほとんど同じです。

この数字を扱うときに、コンピューターのプログラムはどのように書くのでしょうか?。まさか、電卓のようにいちいち数字を書き込むわけにも行きません。例えば、100 万個の数の和を求めるときに、電卓のようにいちいち全ての数字を書き出すプログラムを書くわけにも行きません。そこで、数は全て変数に格納されて、処理をすることになります。その変数の処理の仕方をプログラムで書くことになります。これが電卓と決定的に違うことです。いちようここまでのことはこれまでの学習の範囲です。理解していますね。

もう一度言います。コンピューターでデータを処理するときには、データは全て変数に入れて処理します。変数とはデータを入れる箱のことでしたよね?。よく理解してください。その箱の中には、型 (REAL や INTEGER など) が指定されたデータがたった一つ入ります。

ここで問題が生じます。整数のデータが 1 万個あった場合、どうしますか?。一つの解決方法は、1 万個の整数の型の変数を用意することです。例えば、

INTEGER *AAA, AAB, AAC, AAD, AAE, AAF, AAG, AAH*
INTEGER *AAI, AAJ, AAK, AAL, AAM, AAN, AAO, AAP*

C まだまだ変数宣言をいっぱい書く

C 変数ごとの処理を書く

END

のように書きます。この方法は明らかに手間が必要で、現実的では有りません。問題の所在は、

- 10000 個の変数を 1 個ずつ宣言している。
- その 10000 個の変数の、個々について処理をしようとしている。

というところにあります。明らかにプログラムが大変です。

こういうときに配列と呼ばれる変数の入れ物を使います。それは、

```
INTEGER ABC(10000)
```

のように宣言します。すると、ABC(1), ABC(2), ABC(3), $\cdots$, ABC(9999), ABC(10000) のような、整数の入れ物が用意されます。そして、その入れ物の中のデータ (個々では整数の数値) を操作したい場合は、ABC(50) というようにすればよいのです。配列を使うと、大きなデータを扱う場合、非常に簡単になります。

ここで、ここで言葉の定義をしっかりしておきましょう。先ほどの例を使って、

- ABC を配列名といいます。
- 10000 のように配列に格納できるデータ数を配列の大きさと言います。

- ABC(1) や ABC(2)、ABC(7777) などの個々の配列のデータを配列の要素と言います。ここでは、配列の要素は 10000 個有ります。
- 要素を示す番号を添字と言います。

と、言葉は定義されます。

3.2 配列の宣言の方法

配列を使うときには宣言が必要です。以前学習した変数の宣言文と同様に、配列の宣言文を書かなくてはなりません。宣言文なので、実行文に先立って書く必要があります。例えば、

配列の型　配列名　配列の大きさ (下限:上限)

となります。配列の下限を省略した場合、それは 1 となります。実際には次のような宣言の方法があります。通常は、以下の宣言の方法を用います。

`REAL A(1:3)`	実数型の配列、A(1),A(2),A(3) を用意する。
`REAL B(-1:2)`	実数型の配列、B(-1),B(0),B(1),B(2) を用意する。
`REAL C(5:6)`	実数型の配列、C(5),C(6) を用意する。
`REAL D(3)`	これは、配列の下限を省略した場合で、それは 1 となります。従って用意される配列は、実数型の D(1), D(2), D(3) です。

宣言として、DIMENSION を使う方法もあります。その場合、変数宣言ができませんので暗黙の型宣言[1]に従います。あるいは、DIMENSION のあとに配列名の型宣言を行えば、型を決められます。ただし、先ほどの方法に比べて 2 行書く必要があり、手間がかかります。例えば、以下のように記述すると暗黙の型宣言が適用されます。

`DIMENSION A(1:3)`	実数型の配列、A(1),A(2),A(3) を用意する。
`DIMENSION KVALUE(-1:2)`	整数型の配列、KVALUE(-1),KVALUE(0),KVALUE(1),KVALUE(2) を用意する。
`DIMENSION RVALUE(3)`	実数型の配列 RVALUE(1),RVALUE(2),RVALUE(3) を用意する。

この暗黙の型宣言を避けるためには、以下のように記述します。

`DIMENSION KVALUE(-1:2)` `REAL KVALUE`	実数型の配列、KVALUE(-1), KVALUE(0), KVALUE(1), KVALUE(2) を用意する。
`DIMENSION RVALUE(3)` `INTEGER RVALUE`	整数型の配列 RVALUE(1), VALUE(2), RVALUE(3) を用意する。

[1]配列名の先頭が I,J,K,L,M,N で始まる場合、その配列の型は INTEGER(整数) になる。それ以外の場合、配列の型は REAL(実数) になる。

3.3　配列の使い方

配列の使い方については、教科書の P.60〜68 で学習しましょう。