パソコンの基礎的知識 (5)

Basic Knowledge for Personal Computers (5)

清水 悟*

東京女子医科大学衛生学教室

I. 分岐命令

BASIC プログラムは文番号を昇順(小さい方から大きい方へ)に実行してゆき、END 命令があれば実行を終了する。プログラム実行途中でプログラム実行順序を変更する様な場合に用いられるのが分岐命令である。

たとえば

```
10 PRINT "♥"
20 GOTO 10
30 END
```

というプログラムを実行させた場合、ディスプレイには "♥" を永遠に表示しつづけることになる。つまりこのプログラム中の行番号20で、行番号10への無条件分岐命令が存在するため、常に行番号10から20の間をくり返すことになる。この様な状態を"無限ループ"という。この様なプログラムを作成することはまず無いが、よく誤った GOTO 命令がプログラム中に存在すると、いつまでたってもプログラムの実行が終了しない"無限ループ"に入ってしまうことになる。

ここでこのプログラムに手を加えて

```
10 I = 0
20 PRINT "♥"
30 I = I + 1
40 IF I = 100 THEN GOTO 60
50 GOTO 20
60 END
```

という様にすれば、このプログラムはディスプレイ

に "♥" を100回表示して実行を終了する。このプログラムの場合、文番号30を実行するごとに、変数 I に 1 が加算され、文番号40のところで変数 I が100になった場合、つまり "♥" を100回ディスプレイ表示した場合、文番号60へ分岐して実行を終了する。

文番号40の命令 IF-THEN- を条件判断命令といい、IF-THEN GOTO- とすると条件命令という。つまり IF のあとの条件式を満たすならば THEN 以後を実行せよというのが本来の意味である。また THEN を ELSE に変ると IF に続く条件式が満たされない場合、ELSE 以後を実行せよという意味になる。さらに THEN と ELSE を一度に使用してもよい。上のプログラム例の文番号40と50を変更して

```
10 I = 0
20 PRINT "♥"
30 I = I + 1
40 IF I = 100 THEN GOTO 60
   ELSE GOTO 20
60 END
```

とすることも可能である。

ここでもうひとつ重要なことを指摘する必要がある。それは文番号10の演算命令である。この意味は変数 I に初期値 0 を与えることである。もしこの文番号10がない場合、プログラミング後初めて実行させた時は、このプログラムの意図するとおり、"♥" を100回ディスプレイに表示して実行を終了するが、2回目以後の実行はプログラムの意図どおりにはいかない。たぶん無限ループに入ってしまう。なぜならば、1回目の実行終了時では変数 I には100という値が保持されていて、2回目の実行命令を与えた時、"♥" をディスプレイに表示すると、つぎの文番号30のところで I の値は100+1 となり、それ以後文番号40での条件判断に

* Satoru SHIMIZU : Tokyo Women's Medical College, Department of Hygiene, 10 Kawada-cho, Ichigaya, Shinjuku-ku, Tokyo 162, Japan.

(昭和60年10月30日 受理)

において変数 I は常に >100 であって終了条件を満たさないことになる。文番号40をつぎの様に

```
40 IF I = > 100 THEN GOTO 60
      ELSE GOTO 20
```

と変更しても2回目以後の実行は、“♥”の表示を1度行なうだけである。この場合 “ $=>$ ” 等の組み合わせによる意味はつぎに示すとおりである。

〈条件式〉	〈意味〉
IF I=100	$I=100$
IF I EQ 100	$I=100$
IF I=>100	$I\geq 100$
IF I>=100	“ ”
IF I GE 100	“ ”
IF I>100	$I>100$
IF I GT 100	“ ”
IF I<100	$I<100$
IF I LT 100	“ ”
IF I<=100	$I\leq 100$
IF I <=100	“ ”
IF I LE 100	“ ”

このプログラム例に見られるような変数 I の使用法を“ループ・カウンタ”という。

この様に“ループ・カウンタ”として使用している変数に対しては、プログラムの冒頭で必ず初期値を与えるか、あるいは“CLEAR”という様な数値変数を0に、文字変数にスペースを初期設定する命令を加えておくが必要になる。ループやくり返しをプログラム中で行なわせる場合の注意点として、そのループから脱出する条件分岐命令を含むことを忘れてはならないことと、必ずそのループを終了する場合の条件が満たれることを確認しなければならない。

II. GOSUB 命令

もうひとつよく用いられる分岐命令を上げるが、このプログラムの別な文番号あるいはラベル (* SHORI) で指示されるところへ分岐したあと、その分岐した部分の命令を実行後、はじめの分岐命令のつぎの文番号へもどる命令である。

```
10 PRINT "♦"
20 GOSUB 40
30 END
40 PRINT "♣"
50 RETURN
60 END
```

このプログラムにおいて、文番号10のプリント命令実行後、文番号20の“GOSUB”命令によってプログラムの実行は文番号40へ分岐する。その後文番号50の位置から文番号30へと実行を移し、そこで終了する。

この場合の文番号40~50をサブルーチンといい、サブルーチンの終了（出口）には必ず RETURN 文を必要とする。反復あるいはくり返して同様な処理をする場合は、このようにサブルーチン化しておくでプログラムがすっきりして見易くなる。またこの様なプログラムをブロック構造プログラムと呼ぶ。

〈例〉

```
10 PRINT "PROGRAM START"
20 GOSUB * NYURYOKU
30 GOSUB * KEISAN
40 GOSUB * INSATSU
50 END
60 /
100 * NYURYOKU
110 INPUT "X=" ; X
120 INPUT "Y=" ; Y
130 RETURN
140 /
200 * KEISAN
210 Z=X ** 2 + 30 * Y
220 RETURN
230 /
300 * INSATSU
310 PRINT "X=" ; X ;
320 PRINT "Y=" ; Y
330 PRINT "X ** 2 + 30 * Y=" ; Z
340 RETURN
350 /
```

III. FÖR-NEXT ループ

すでに述べた様な同じ処理をくり返すということは、パソコンあるいはコンピュータの最も得意とする“技”である。さらに同じ処理をプログラム上で何度も“書く”という手間もはぶけることになる。やはりプログラミングの“コツ”は他の人が見てもわかりやすく、しかも平易なプログラムであることが必要不可欠である。長いプログラムの修正や誤った箇所を探すのは大変な労力をついやすことになる。

ここでもくり返し処理によく使う命令として FÖR-NEXT 命令について説明する。つぎの様なプログラムにおいて

```
10 FÖR I = 1 TO 100 STEP 1
20 PRINT "♠"
30 NEXT I
40 END
```

この文番号10と30がひとつのループを形成している。FÖR 文から NEXT 文までの間にある命令を反復して実行することになる。この FÖR 文と NEXT 文にある変数 I がループカウンタと呼ばれループの実行回数につれ+1づつ値が増加し、この変数が100まで達するまで実行をくり返し NEXT 文のつぎの文番号へ実行を移すのである。文番号10の最後にある“STEP 1”とはループカウンタである変数 I の増分値を表し、省略すると増分値は“1”と見なされる。また増分値の指定は小数を含む実数値でもよいが、“TO~”で指定した値を超えた場合にループの終了と見なされる。また FÖR ループの終了位置には必ずループカウンタに指定された変数を含む NEXT 文がなければならない。

FÖR-NEXT ループの中に別な FÖR-NEXT ループを含むことができる。

たとえば

```
10 DIM Z (10, 20)
15 II = 0
20 FÖR I = 1 TO 10
30 FÖR J = 1 TO 20
40 II = II + 1
```

```
50 Z (I, J) = II
60 NEXT J
70 NEXT I
80 END
```

このプログラムでは配列宣言をした変数 Z の10×20個の配列要素に順番に1~200までの数値を代入している。

変数 Z の配列要素に代入された値

	I = 1	2	3	4	-----	20
J = 1	1	2	3	4		20
2	21	22	23			40
3	41	42	43			60
					Z (I, J)	
10	191	192	193	194		200

この様な2次元配列の各要素への数値あるいは文字列の代入・格納の場合には FÖR-NEXT ループを2重に使用することができる。この2つのループカウンタ変数 I と変数 J が各配列要素を示す添字として使用されることに注目する必要がある。さらに多重ループの場合には必ず内側のループの終了位置を表わす NEXT 文が外側のループの NEXT 文よりも前になければならない。

正	誤
<pre>FOR I FOR J ... NEXT J NEXT I</pre>	<pre>FOR I FOR J ... NEXT I NEXT J</pre>

IV. 外部記憶装置からの入出力

フロッピー・ディスクあるいはハード・ディスクといった外部記憶装置に対する入出力はデータの保存などが必要な場合に、プログラム中にデータの出力（書

き込み)あるいは入力(読み込み)命令を実行することで可能となる。BASICで扱う外部記憶装置はすべてファイルという概念に置きかえられる。ひとつの外部記憶装置には複数のファイルが存在することができる。

ファイルとは項目(アイテム)の集まりをレコード

といい、レコードの集まりをファイルと呼ぶ。外部記憶装置には BASIC プログラムを SAVE 命令などで保存したプログラムファイルとプログラムで読み書きするデータを入れるデータファイルの2種がある。

図1. ファイルの構造

さらにデータファイルには次の2種類がある。

〈シーケンシャルファイル〉

順編成ファイルとも呼ばれ、レコードを順番にならべて記憶されたファイル。10番目にあるレコードだけを読み取りたいときでも、先頭のレコードから順番に読んでいかなければならない。特定のレコードを削除、変更する場合は記憶されているファイルを読み込み削除・変更してから、新しいファイルに書き出すということによって可能である。

〈連続ファイル〉

ランダムファイルとも呼ぶ。各レコードは固有のレコード番号を持って記憶されていて、レコード番号を指定することによってどのレコードでも、直接読み書きすることができる。連続ファイルは入出力以前にファイルの項目を RECORDSIZ 文、あるいは機種によっては FIELD 文で定義しておかなければならない。

ファイルは入出力する以前に OPEN 命令によってチャンネル番号(ファイル番号)と結合されなければならない。この場合のチャンネルとは入出力のためのルートである。チャンネルは普通パソコンの場合 0~8 を持ち、ファイルごとに 1 チャンネルを割りあてる。

ファイルとチャンネルの結合は OPEN 命令によってなされる。

```
OPEN "1:ファイル名" FOR OUTPUT AS
  デバイス名          <チャンネル番号>
```

図2. ファイルとチャンネルの概念

ここでデバイス名とは外部記憶装置に対して与えられている固有の番号である。たとえば上の例でフロッピー・ディスク装置が2台ついている場合、デバイス名として1と2が与えられる。つまり、フロッピー・ディスクの1に入っているファイルを指定していることになる。よく BASIC を始めたばかりの人で、デバイス名として用いられる番号とチャンネル番号を混同する人がいるので注意すること。

OPEN 命令でファイルとチャンネル番号を結合することによってファイルに対する読み書きが可能となる。前述の例では出力用のファイルとして OPEN したことになる。OPEN 命令実行後はすべてチャンネル番号によってファイルに対する読み書きを行なう。

```
PRINT # <チャンネル番号> ; A ; B ; C $
INPUT # <チャンネル番号> ; X ; Y ; Z $
```

プログラムの中で外部記憶装置に対する読み書きが終了した時は CLOSÉ 命令によってファイル名とチャンネル番号を切り離す。

CLOSÉ 〈チャンネル番号〉

以上は順編成ファイル（シーケンシャルファイル）に関して必要なコマンド（命令）の概略であるが、次に注意しなければならない点を述べる。

まず書き込み（PRINT）と読み込み（INPUT）時の変数の型と並びの順序を同一にすることである。

たとえば

```
PRINT # 3 ; A ; B $ ; C
```

という様に数値変数 A と C、文字変数 B をファイルに書き込みながら、今度は読み込みのプログラムで

```
INPUT # 1 ; X ; Y ; Z $
```

とした場合では、読み込み時に入力エラーとなる。

なぜならば、書き込み時の変数の数、数値変数 2、文字変数 1 と読み込み時の変数の数は同じでも、数値変数と文字変数の並び順序が異っているからである。

また文字変数を用いてファイルに読み書きをする場合、文字変数に含まれる文字数が同じになるように、DIM 文を用いて定義しておかなければならない。

順編成ファイル使用時の注意としてデータファイルの終りを確認する場合の問題がある。

たとえば

```
OPEN "TEXT" FÖR INPUT AS # 3
```

```
  : FÖR I = 1 TÖ 100
```

```
INPUT      # 3 ; X ; Y ; Z $
```

```
  :
```

```
  NEXT I
```

```
  :
```

この様なプログラムの場合、たとえデータファイルのレコード数が10件しかない場合でも読み込み命令を

100回実行することになる。レコード数が不明な場合はよくあることで、この場合どの様にデータファイルの終りを確認して、つぎの作業あるいはプログラムを実行させるかという問題である。NEC の N88 BASIC の場合では、入出力関数として、指定した番号の順編成ファイルの終りを示す "EOF" という命令が用意されている。それを用いた場合上の例では、

```
10 ÖPEN "TEXT" FÖR INPUT AS # 3
```

```
50 FÖR I = 1 TÖ 100
```

```
60 IF EÖF ( 3 ) THEN GÖTÖ 100
```

```
70 INPUT # 3 ; X ; Y ; Z $
```

```
  :
```

```
90 NEXT I
```

```
100 CLOSÉ # 3
```

```
  :
```

というようにプログラムを修正すれば、100レコード以下の場合、データファイルの終りが確認されると、つぎに CLOSÉ 処理をすることができる。

この様な入出力関係を持たない BASIC の場合は、データの終了したファイルに対して入力命令（読み込み命令）を実行すると、自動的にエラー番号を ERR という変数に格納するので、この変数がファイルの終りを示すエラー番号の時に CLOSÉ 処理を行なう様にプログラムをすればよい。

```
  :
```

```
INPUT # 3 ; X ; Y ; Z $
```

```
IF ERR=11 THEN GÖTÖ * OWARI
```

```
  :
```

という様にして、ファイルの終了処理を行なうことができる。

(以下次号へ続く)

*

*

*