BASIC에 의한 컴퓨터 프로그램(7)

編 輯 室

5. TAB과 SPC

이들은 PRINT(또는 LPRINT)문 내에서 사용되며,

· TAB(n)은 "왼쪽에서 n번째 위치까지 건너뛰어라"

· SPC(n)은 "n자 만큼 공백을 출력하라"

라는 것을 나타낸다.

위치는 행의 왼쪽끝을 1로 한다(다른 시스템에서는 0으로 하는 것이 많으므로 주의할 것).

[프로그램5－1] TAB문(1)

```
110 PRINT "12345678901234567890"
120 A=753
130 PRINT A;TAB(12);A
140 PRINT TAB(3);A;TAB(12);A
150 PRINT TAB(5);A;TAB(12);A

run
12345678901234567890
 −753        −753
   −753      −753
     −753    −753
OK
```

[주의] TAB에 의하여 현재 위치보다 왼쪽으로 되돌아 올 수는 없다. 일부러 그러한 프로그램을 작성하려면 다음 행의 지정위치에 다음의 항목을 출력한다.

[프로그램5－2] TAB문(2)

```
110 PRINT "12345678901234567890"
120 A=−753
130 PRINT A;TAB(12);A;TAB(3);A
run
12345678901234567890
           −753
  −753
OK
```

6. 서식 지정(USING)

출력 서식을 상세하게 지정하려면

PRINT USING "지정서식";출력항목 열

의 형식으로 작성하며 서식은 다음과 같은 형으로 지정한다.

```
+ 부호
· 소수점 출력위치
# 숫자 출력위치
공백 공백위치
```

《예》 RPINT USING "+###.###";A;B

이것은 "A와 B를

· 부호는 수치의 앞에 붙이고, 양수는+, 음수는 −로 표시하고

· 정수부에 2자리 잡고

· 왼쪽에서 4번째 위치에 소수점을 찍고

· 소수부에 3자리 잡고
· 그 오른쪽에 1자분의 공백을 넣는다.
라는 형식을 출력하라"하는 것을 나타낸다.

(1) 처음 위치에 「#」을 쓰면 양수일 때의 부호표시가 「+」가 아니고 공백으로 된다.
(2) 부호를 오른쪽에 붙일 수도 있다. 이것은 「+」나 「−」를 오른쪽에 써서 지정하며, 「−」는 위치 (1)과 같고, 「+」는 생략의 의미이다.
《예》 RPINT USING "###.##+";X
(3) 정수값으로서 오른쪽으로 붙여서 표시하는 때는 「·」는 필요 없다.
《예》 RPINT USING "+###";N
(4) 3 자리마다 , 를 넣는 것도 가능하다.
《예》 RPINT USING "+#,###,###";KEUMAEK
(5) 지수부가 붙은 형태(부동 소수점 표시 : floating point)로 출력할 때는 오른쪽를 붙여서, 지수부는 E±XX형으로 출력되므로 항상 4자리가 필요하다.

[프로그램6−1] USING문(1)

```
110 A=123.56
120 PRINT USING "####.###";A
130 PRINT USING "####.##";A
140 PRINT USING "######.#";A
150 PRINT USING "######."A
160 PRINT USING "########";A
170 PRINT USING "##.#";A
180 PRINT USING "+####.#";A
190 PRINT USING "####.##+";A
run
 123.560
  123.56
   123.6
    124.
     124
1.2E+02
  +123.6
123.56+
OK
```

(6) 처음에 ₩₩이라고 기입하면 수치의 앞에 ₩가 붙는다.
(7) 처음에 **이라고 기입하면 수치의 왼쪽 여백에 **가 표시된다.
(8) 처음에 ₩**이라고 기입하면 앞에 ₩가 출력되고, ₩에서 수치의 앞까지 사이의 여백은 **로 채워진다.
(9) 서식 지정은 하나의 PRINT USING문에 하나만 써진다. 서식이 다른 항목을 같은 행을 출력하려면 몇개의 PRINT USING 문으로 나누어 써야 한다.
(10) & 공백 n개 &이라고 기입하면 n + 2 문자분의 표시위치가 확보되어 문자열 변수의 값(문자열)이 왼쪽부터 출력되며, 문자열쪽이 길면 앞의 n 문자가 출력된다.

[프로그램6−2] USING문(2)

```
110 DEFDBL A
120 A=100000#
130  C$="ABCDEFGHIJKLMNOPQRSTUV
WXYZ" 140 PRINT USING "###,###,###"
A
150 PRINT USING "$$#####,###,###";A
151 PRINT USING "₩₩#####,###,###";A
160 PRINT USING "******,###,###";A
170 PRINT USING "₩******,###,###";A
180 PRINT USING "##########.";A
190 PRINT USING "&          &";C$
run
        1,000,000
       $1,000,000
₩₩     1,000,000
*****  1,000,000
₩***1,000,000
        1,000,000
ABCDEFGHIJKLMN
OK
```

7. STRING$, SPACE$, LPOS

STRING$는 문자열형 함수이며, 일반적으로
STRING$(문자수, "문자")

의 형식으로 사용되며, 두번째 인수에 쓰여진 문자를 첫번째 인수로 지정한 수만큼 표시한다.

《예》 STRING$(3, "K")는 "KKK"와 같다.

일반적으로

STRING$(수치 변수명, 문자열 변수명)

도 가능하며 두번째 인수로 지정된 문자열이 2자 이상일 경우에는 첫문자만 유효하다.

STRING$는 줄을 긋거나 막대 그래프를 그릴 때 편리하며,

PRINT STRING$(5, CHR$ (13))

과 같이 사용할 수도 있다(ASCII 코드로 13은 행 바꾸기이므로 5행 건너뛴다)

SPACE$(n)은 "공백 n자의 문자열"을 나타낸다.

SPC는 PRINT 문이나 LPRINT문 내에서만 사용되며, SPACE$는 대입문이나 IF문, 문자열 식 내에서도 사용할 수 있다.

LPOS는 현재 프린터의 왼쪽에서 몇번째까지 인쇄했는가(정확하게 말하면 프린터에 출력하기 위한 버퍼의 몇자째까지 사용했는가)를 나타내는 함수이며,

LPOS(인수)

의 형식으로 사용한다(인수는 형식적이며 보통 0이나 1을 쓴다.)

《예》 단어를 하나씩 입력하여 그 문자수를 헤아려서 행의 오른쪽에 그 단어를 인쇄할 수 있는 여백이 있으면 인쇄하고, 여백이 없으면 줄을 바꾸어서 인쇄한다.

[프로그램7－1] LPOS의 사용

```
110 REM -------- LPOS의 例----
120 REM--- 單語의 組立----
130 W$=""---CLS
140 C$=INPUT$(1)
150 IF C$=" " THEN GOSUB 100:GOTO 30
160 IF C$="**" THEN LPRING ₩$:END
170 IF C$=CHR$(13) THEN LPRINT ₩$:GOTO
30
180 W$=₩$+C$
190 GOTO 40
200 REM --- 인쇄---
210 L=LEN(₩$)
220 IF LPOS(1)+L>39 THEN LPRINT
230 LPRINT ₩$;" ";
240 RETURN
```

8. 디스크 관계 커맨드, 명령, 함수

8－1 CHAIN

이어지는 프로그램을 디스크에서 로드(load)하여 실행한다. 이것은 연결작업(chain job)이라 하여 긴 프로그램을 일부분씩 기억장치(memory)에 입력하여 실행하기 위한 명령이다. "다음 프로그램을 로드하여 바로 실행하라"라는 지시만 하려면,

LOAD "장치명 : 화일명",R

또는

RUN "장치명 : 화일명"

이라는 방법도 가능하지만, CHAIN을 사용하면,

· 데이터의 전부 또는 일부를 다음 프로그램으로 연결시킬 수 있고

· 프로그램의 일부분을 남겨서 계속할 수 있다.

라는 장점이 있다. 일반형은

CHAIN "장치명 : 화일명"

이며, 추가로 다음과 같은 지정이 가능하다.

(1) CHAIN의 다음에 MERGE라 쓰면, 앞 프로그램과 새로운 프로그램을 조합(merge : 통합해서 하나의 프로그램으로 한다)해 준다.

(2) "장치명 : 화일명" 다음에 「,」를 써서 행번호를 지정하면, 그 행번호에서 실행이 시작된다.

(3) 다음에 ALL이라 쓰면, 모든 변수가 다음의 프로그램에 인계된다(전부가 아니고 특정의 몇개를 남길 때는 COMMON 문으로 별도로 지정한다).

(4) 또,

· DELETE 행번호－행번호

를 붙이면 지정한 범위를 삭제하고 나서 조합된다.

《비고》 MERGE 또는 ALL을 지정하면 앞 프로그램의 선언(변수형 선언, 함수 정의, ON－－GOTO 등)은 계속된다(MERGE도 ALL도 지정하지 않으면 독립된 프로그램으로 되

어 선언은 계속되지 않는다.)

8－2 MERGE

현재 기억장치에 있는 프로그램을 지우지 않고 새로운 디스크에서 프로그램을 로드하여 2개를 합하여 하나의 프로그램으로 한다. 단, 같은 행번호가 있을 때는 기존의 행은 지워지고 새로운 쪽(디스크에 있는 프로그램)이 남게 된다.

일반형은

MERGE "화일명"

또는

MERGE "장치명 : 화일명"

《주의》 조합(merge)을 하기 위해서는 디스크에 프로그램을 기록(save)할때 A 추가기능(ASCII 형식을 지정)을 붙여서

SAVE "화일명", A

의 형식으로 기록해 두어야 한다. 이 시스템에서는 디스크의 용량이 충분하므로 프로그램을 기록할 때 언제나 A를 붙여 두는 것이 좋다.

8－3 COMMON

연결작업시에 공통변수를 정의한다.

COMMON 변수명

8－4 NAME

화일명을 변경한다.

NAME "장치명 : 기존 화일명" AS "장치명 : 화일명"

《주의》 이 시스템에서는 화일명의 뒷부분에「.BAS」라는 것이 붙어 있지 않으면 LOAD 문으로 로드하지 못한다. SAVE할 때는 하나 하나「.BAS」를 기록하지 않아도 시스템 쪽에서 자동적으로 붙여주지만, 화일명 변경시에는 기존 이름에「.BAS」가 붙어 있어도 새로운 이름에 자동적으로 붙여주지 않으므로 BASIC의 프로그램 화일명에는 반드시 뒷부분에「.BAS」를 붙여야 한다. 역으로, BASIC 프로그램을 문자열 데이터로 읽어서 처리할 때(부록 B. 8 참조) 화일명에는「.BAS」가 주어져 있으면 입력 못하므로 NAME 커맨드를 사용하여「.BAS」가 없는 화일명에는 바꾸어야 한다.

8－5 KILL

화일의 말소

KILL "장치명 : 화일명"

9. 비정상 사태에 있어서의 조치

비정상 사태(error, 예를 들면 0에 의한 나눗셈, memory 부족, 문법위반 등)가 발생하면 일반적으로 에러 메세지를 표시하고 정지하지만, 먼저

ON ERROR GOTO error 처리시작 행번호

라는 문을 실행하여 두면, 이 문에서 지정된 행으로 뛰어서 error 처리할 수 있다. error 처리가 끝난 뒤에는

RESUME NEXT

라고 쓰면 비정상 사태 발생장소 다음의 문에서 실행을 다시 시작할 수 있다.

RESUME 행번호

라고 쓰면, 지정한 행번호에서 실행을 다시 시작할 수 있다.

· error 처리 프로그램 내에 error의 발생 장소나 error의 종류를 알 수 있도록 ERR 및 ERL 이라는 특별한 변수가 준비되어 있으므로

ERR에는 error의 종류를 나타내는 번호

ERL에는 error를 일으킨 문의 행번호

가 들어가며, error 번호의 의미는 다음과 같다(각 message의 상세한 설명은 부록 C－4 참조).

9040 NEXT III

9050 RETURN

이것을 호출하기 위하여 프로그램의 앞부분에 다음의 2행을 추가한다.

ON KEY (1) GOSUB 9000

KEY(1) ON

10－1 KEY

기능키의 내용을 변경한다.

KEY 번호, 문자열

《예》 KEY 1, "?I;J;K"＋CHR$(13)

라 하면 기능키의 1번은

"I, J, K 값을 표시"

라는 기능으로 된다. 이 시스템에서는 16자까지의

문자열을 등록할 수 있다.

10－2 KEY LIST

기능키의 내용 일람표를 표시한다.

《실험 예》 앞에서의 KEY문에 의한 변경되지 않은 상태는 다음과 같은 표준 설정으로 되어 있다.

```
key list
F1 LIST
F2 RUN←
F3 LOAD"
F4 SAVE"
F5 CONT←
F6, "LPTI:"←
F7 TRON←
F8 TROFF←
F9 KEY
F10 SCREEN 0,0,0←
OK
```

10－3 KEY ON

각 기능에 등록되어 있는 문자열의 앞부분 6문자를 화면 아래부분에 표시한다.

10－4 KEY OFF

KEY ON에 의한 표시를 지운다.

《주의》 이 시스템에서 키의 내용을 표시해 두면 화면표시가 매우 늦어져서 프로그램 표시나 결과의 표시에 시간이 걸리므로 일반적으로 표시를 지워 두는 것이 좋다.

11. 난수 RND와 응용

함수 RND는 0에서 1까지 사이의 일양분포 난수를 발생하며 인수는 쓰지 않아도 되고 RDN 라고만 쓰면 된다.

실제로 어떤 수가 나오는지 실행해 보자.

[프로그램11－1] 난수 발생(1)

```
110 FOR I=1 TO 10
120 U=RND
130 PRINT U
140 NEXT I
run
 · 1213501
 · 651861
 · 8788611
 · 7297625
 · 7.3698053－02
 · 4903128
 · 4545189
 · 1072496
 · 9505102
OK
```

용도에 따라서는 [0,1] 이외의 구간에 있는 난수가 필요하게 되는 경우가 있는데 이때는 변수를 변환하면 된다. 예를 들면 0에서 99까지의 정수 난수를 만들려면

```
INT(RND **100)
```

으로 하면 된다.

[프로그램11－2] 난수 발생(2)

```
110 FOR I=1 T0 10
120 U=INT(100**RND)
140 NEXT I
run
 12 65 86 72 79 7 49 45 10 95
OK
```

또 6배 해서 정수를 취하면 1에서 6까지의 수가 나오게 된다.

[프로그램11－3] 난수 발생(3)

```
110 FOR I=1 TO 10
120 U=INT(6**RND)+1
130 PRINT U;
140 NEXT 1
run
 1 4 6 5 5 1 3 3 1 6
OK
```

정규분포에 따른 난수가 필요할 때는 여러 가지 방법이 있는데 이 시스템에서 sin x나 cos x는 비교적 시간이 걸리지만 RND는 그 보다 빠르므로 [0, 1]상의 난수를 12개 더해서 6을 빼면 거의 N(0,1)에 따르면 난수가 얻어진다는 성질을 사용하

면 된다. 프로그램A－11은 이것을 더욱 변환해서 주어진 평균값 μ와 표준편차σ의 정규난수를 만들게 한 프로그램의 한 예이다.

[프로그램11－4] 난수 발생(4)

```
110 DIM H(12)
120 REM --- 100회 실행 ---
130 FOR K=1 TO 100
140       REM --- 정규난수 작성---
150       Y=－6
160       FOR J=1 TO 12
170         Y=Y+RND
180       NEXT J
190       REM--- 계산 ---
200       I=INT(Y**2+6.5)
210       IF K<1 THEN 140
220       IF 1>12 THEN 140
230       H(I)=H(I)+1
240 NEXT K
250 REM -- 히스토그램 ---
260 FOR I=1 TO 12
270         IL=(I－4)/2
280       IR=(I－3)/2
290       PRINT USING "##.#";IL;
300       PRINT "<×<";
310       PRINT USING "##.#":IR;
320       PRINT TAB(18):H(I);TAB(25);
330       PRINT STRING$(H(I),"**")
340 NEXT I
350 END
run
－1.5<×<－1.0   0
－1.0<×<－0.5   2    **
－0.5<×< 0.0    4    ****
  0.0<×< 0.5    12   ************
  0.5<×< 1.0    23   **********
  1.0<×< 1.5    17   **********
  1.5<×< 2.0    19   **********
  2.0<×< 2.5    10   *******
  2.5<×< 3.0    7    *****
  3.0<×< 3.5    3    ***
  3.5<×< 3.5    2    **
  40.<×< 4.5    0
OK
```

모의실험(simulation)에서 지수 난수가 필요할 경우가 있다. 이것은 간단하게 RND의 －LDG를 구해서 인수 μ를 곱하면 된다.

[프로그램11－5] 난수 발생(5)

```
110 DIM K(20
120 INPUT"mu=";MU
130 FOR I=1 TO 100
140       X=－LOG(RND)**MU
150       J=INT(2**X)
160       IF J<21 THEN K(J)=K(J)+1
170 NEXT I
180 FOR J=1 TO 20
190       PRINT J/2,K(J), STRING$(J),"**"
200 NEXT J
run
mu=? 2
 .5      22     **********
 1       17     **********
 1.5     10     *******
 2       8      ******
 2.5     4      ****
 3       4      ****
 3.5     3      ***
 4       5      ******
 4.5     1      *
 5       3      ***
 5.5     2      **
 6       0
 6.5     0
 7       0
 7.5     0
 8       0
 8.5     0
 9       1
 9.5     0
 10      0
OK
```

11-1 난수열의 변경

이 시스템의 RND는 매회(RUN할 때마다) 같은 난수를 발생한다. 이것은 실험의 재현성이라는 점에서는 좋으나 항상 같은 난수에서는 곤란한 경우도 있다. 이런 경우에 RANDOMIZE라는 명령을 사용하면 좋다. 이것은 카드를 섞거나 마작의 파이를 뒤섞이는 것에 상당하는 명령이다. 단 여기서도 실제로는 지정이 필요하며 난수열 발생의 값으로 −32768이상, 32767이하의 정수값을 하나 주어두어야 한다.

이것은

RANDOMIZE 난수발생값

이라는 형으로 쓰며, 난수발생 값을 쓰지 않고 RANDOMIZE 라고 하면,

Random number seed(−32768 to 32767)?

라는 메세지가 나오므로 키보드에서 난수발생값을 입력한다. 이렇게 해서 만들어진 난수는 어떻게 되는가 출현빈도를 살펴보자.

[프로그램11-6] 난수 발생 빈도

```
10 DIM K(6)
20 FOR I=1 TO 100
30   U=INT(6*RND)+1
40   K(U)=K(U)+1
50 NEXT I
60 FOR J=1 TO 6
70   PRINT STRING$(K(J), "*")
80 NEXT J

run
**********
**********
**********
**********
**********
**********
OK
```

일양분포에 따른 확률변수합의 분포는 이론적으로 구해지지만, 그것을 실험해 보는 것도 재미있는 일이다. 프로그램 A · 14는 그 프로그램 예와 실행예이다.

[프로그램11-7] 확률 변수의 분포

```
10 DIM K(12)
20 FOR 1=1 TO 100
30 UI=INT(6*RND)+1
35   U2=INT(6*RND)+1
37   R=U1+U2
40   K(R)=K(R)+1
50 NEXT I
60 FOR J=1 TO 12
70   PRINT J, K(J), STRING$(K(J), "*")
80NEXT J
run
1       0
2       1       *
3       4       ***
4       12      **********
5       12      **********
6       15      ***************
7       11      *******
8       11      ********
9       15      ***************
10      9       ******
11      6       ******
12      4       ***
OK
```