

BASIC 解説書 (B)

まえがき

5 BASIC ステートメント (その2)	1
5. 1 配 列	1
5. 1. 1 1次元配列	1
5. 1. 2 2次元配列	3
5. 1. 3 配列とDIM文	4
5. 1. 4 配列を使ったプログラム例	6
5. 2 その他の有用なステートメント	10
5. 2. 1 PRINT USING文	10
5. 2. 2 GOSUB RETURN文	14
5. 2. 3 ON GOTO文	15
5. 2. 4 論理式 (その2)	15
5. 2. 5 LINE INPUT文	16
5. 2. 6 DEFFN文	16
5. 2. 7 型宣言文	17
5. 2. 8 LOCATE文	17
5. 3 データファイルの使い方	18
5. 3. 1 シーケンシャルファイルとランダムファイル	20
5. 3. 2 ファイル名	21
5. 3. 3 OPEN文とCLOSE文	21
5. 3. 4 シーケンシャル・ファイルの入出力文	22
5. 3. 5 ランダム・ファイルの入出力文	25
5. 3. 6 FIELD文と変換関数	26
5. 3. 7 データファイルを使ったプログラム例	29
5. 4 グラフィクス (図形処理)	36

本解説書はN-BASIC REFERENCE MANUALに基づいて慶應義塾大学経営管理研究科助教授柳原一夫が編集・作成したものである。

まえがき

本解説書はBASIC解説書(A)の後編である。(A)はBASICの基本ステートメントを中心とし、初心者がコンピュータを使って、とにかく計算・演算を行なわせることができるようになる、このことに主眼を置いて作成されたものである。(A)で示されたステートメントだけでも、種々の目的にかなうプログラムが作成可能である。しかし、BASICの持つ能力を十分発揮し、ビジネスにおけるより多様なニーズに応えるためには、なお不便、不十分なので、そのために追加の機能を解説書(B)で補充しようとするものである。

本書は大量のデータを効率よく処理するための配列変数、および報告書や表の作成に便利なFORMAT付きの出力文、長いプログラムの作成時に便利なサブルーチン文、そしてデータ・ファイルの扱い方について主に解説するが、これらのステートメントはビジネスのアプリケーション開発にとって非常に有役なものである。

5 BASIC 応用ステートメント

5-1 配列 (Array)

データを1つ1つ個別に見ないで、同じ属性のデータを1つのグループとして、ひとまとめに見る方が理解が容易で便利な場合がある。例えば、ある会社における1976年から、1980年までの売上をSという記号を使って表現し、個別の年における売上については、Sと添字を使って示すことにするのである。例えばS₁は1976年の売上を示し、S₃は1978年の、またS₅は1980年の売上を表わすものとし、また単にSとすれば、1976年から1980年までの5年間のうち特に年度を指定しないで、売上一般を総称するものとするのである。同様に、同じ会社の1976年から、1980年までの各年における費用をCとすれば、利益Pは次式によって示すことができる。

$$P = S - C$$

または、 $P_i = S_i - C_i$ ただし、 $1 \leq i \leq 5$ …………… (1)

(1)式は具体的には次の事を意味している。

$$P_1 = S_1 - C_1, P_2 = S_2 - C_2, P_3 = S_3 - C_3$$

$$P_4 = S_4 - C_4, P_5 = S_5 - C_5 \dots\dots\dots (2)$$

(1)式の表現ですむものを(2)式の型式で表現するのは繁雑でかえって判りにくい。上の例ではデータが5年間であったが、もし10年間、20年間というようにデータ数が多くなれば、(2)式の表現は(1)に比べて、一層めんどうになる。

データを何らかの属性に従って1まとめにして扱うという上で述べた便利な方法がBASICでは用意されている。これが以下に説明する配列である。

5-1-1 1次元配列

BASICにおいて、配列変数に関しては普通の変数に適用された同じルールが適用される。すなわち、配列変数には整数型、単精度型、倍精度型および文字型変数が存在し、その使いわけは普通の変数と同様である。BASICが普通の変数と配列変数を識別できるのは、配列変数の後尾にカッコで囲まれた添字 (Subscript) があるからである。添字は文字でも、変数でもかまわないが、整数 (負は許されない) でなければならない。

配列変数の例をあげれば次の通りである。

$$A(10), Y\$(X), MIX\%(I), KEIO(J\%)$$

いずれの例も添字は1個だけなので、これらは1次元配列と呼ばれる。

BASICでは配列変数名だけを使った演算は用意されていない (高級なBASICでは出来るものもある)。従って、添字を省略した使い方はできない。以下に1次元配列の簡単な使い方を例で示すことにする。(なお、添字の大きさが10を超える場合は、5-1-3のDIM文を参

照のこと) 配列の演算は既に気づいていることと思うが、FOR…NEXT文を使って行なわれることが多い。

〔例1〕 配列と入出力文

```
100 FOR I=1 TO 5
110 READ A(I)
120 NEXT I
130 FOR I=1 TO 5
140 PRINT A(I);
150 NEXT I
160 DATA 22.8,34.5,42.3,55.6,60.7
```

```
RUN
22.8 34.5 42.3 55.6 60.7
```

〔例2〕 配列の演算

```
100 PRINT"sales volum and price"
110 FOR I=1 TO 5
120 INPUT V(I),P(I)
130 NEXT I
140 FOR I=1 TO 5
150 S(I)=V(I)*P(I)
160 NEXT I
170 FOR I=1 TO 5
180 PRINT S(I);
190 NEXT I
```

```
RUN
sales volum and price
? 10,20
? 15,35
? 22,33
? 56,47
? 34,26
200 525 726 2632 884
```

〔例3〕 データを読み込んだ逆の順番に出力せよ。

```
100 FOR I=1 TO 7
110 READ X(I)
120 NEXT I
130 DATA 1.1,1.2,1.3,1.4,1.5,1.6,1.7
140 FOR I=7 TO 1 STEP-1
150 PRINT X(I);
160 NEXT I
```

```
RUN
1.7 1.6 1.5 1.4 1.3 1.2 1.1
```

別解として

```
100 FOR I=1 TO 7
110 READ X(I)
120 NEXT I
130 DATA 1.1,1.2,1.3,1.4,1.5,1.6,1.7
140 FOR I=1 TO 3
150 SWAP X(I),X(8-I)
160 NEXT I
170 FOR I=1 TO 7
180 PRINT X(I);
190 NEXT I
```

```
RUN
1.7 1.6 1.5 1.4 1.3 1.2 1.1
```

5-1-2 2次元配列

データの中には単純に1つの属性だけでは表わせないものもある。例えば、ある会社における製品別、年別売上高などは、2つの属性を有している。こうしたデータを扱うためには、2つの添字が必要である。すなわち、製品を表わす添字と年を表わす添字である。

例えば、以下の表に示されるデータにおいて、

年 間 売 上 高 S

製品 \ 年	1976年	1977年	1978年	1979年	1980年
製品 A	161	172	183	194	205
製品 B	23	20	18	17	16
製品 C	108	201	194	166	185
製品 D	90	75	60	55	48

S(1, 1)は製品Aの1976年における売上高161を示している。また、同様にS(2, 4)は製品Bの1979年の売上高17を示している。

このように、2つの添字によって、1つのデータを表わす配列を2次元配列という。2次元という意味は上の表でも判るように、縦と横あるいは、行と列を指定することが必要だからである。S(2, 4)は2行4列という様に行が先で列が後(すなわち行列の順)で示されるのが一般的である。S(2, 4)とS(4, 2)とは全然異なるデータを指すので注意が必要である。

以下に2次元配列の簡単な使用例を示すことにする。2次元配列の演算では通常2つのFOR...NEXT文がネスト(入子)となって使われる場合が多い。

〔例1〕 2次元配列と入出力文

```
100 FOR I=1 TO 4
110 FOR J=1 TO 5
120 READ S(I,J)
130 NEXT J,I
140 DATA 161,172,183,194,205
150 DATA 23,20,18,17,16
160 DATA 108,201,194,166,185
170 DATA 90,75,60,55,48
180 FOR I=1 TO 4
190 FOR J=1 TO 5
200 LPRINT S(I,J);
210 NEXT J
220 LPRINT
230 NEXT I
```

```
RUN
 161  172  183  194  205
  23   20   18   17   16
 108  201  194  166  185
  90   75   60   55   48
```

120行でS(i,j)という使い方をしている。このことはiは行を表わす添字、jは列を表わす添字にするという意味になる。

次は100行110行のFOR文を見ると、FOR Jが入子になっている。従って、添字Jが早く回転するので、まずiを1にして、Jが1から5まで回転する。その結果5-1-2に示されるテーブルの製品Aの(つまり1行目の)売上データが5年間分(つまり5列目まで)配列Sに読み込まれる。次にi=2となり、Jが5回転するということによりテーブルと同様にデータが配列Sに読み込まれるのが判る。このように、配列の添字と2つのFOR文の前後関係によってDATA文のデータの順序(INPUT文の場合はキーボードから入力するデータの順序)が変わることを意味している。もし、100行と110行のFOR文を入れ替えれば(もちろん130行のNEXT文もNEXT i,jとなる)、添字iが入れ子の関係で早く回転し、従って行を早く読むことになる。この場合、DATA文は161, 23, 108, 90, 172, 20, 201, 75 ……………の順序になっていないと正しいデータの読み込みはされない。

さて、180行以下の2次元配列の出力についてであるが、なぜ220行にLPRINTが必要なのであろうか。これがないと、出力結果のようにテーブル形式の出力にはならないで、プリンターの中が許す限り、長い行の印刷となってしまふ。

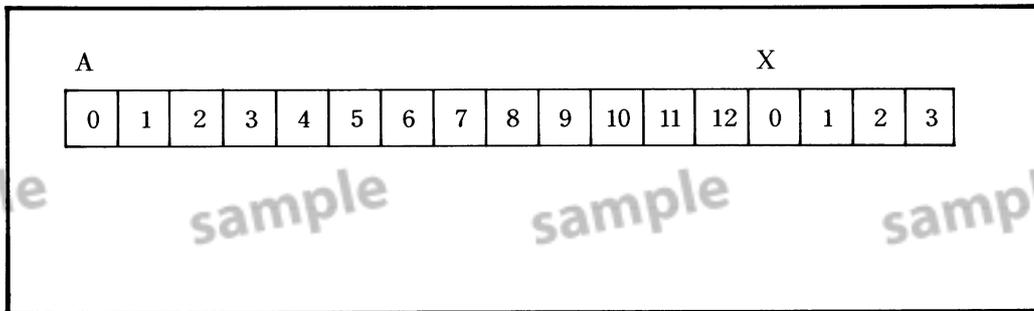
なお、入力と違って、出力では列が早回り(入れ子)でなければならない。なぜなら、プリンターは行送りはできて行戻りはできないからである。

もし、240 LPRINT S(2,3), S(3,2), S(4,4)とすれば、結果はどうなるか。もちろん、18, 201, 55である。

5-1-3 配列とDIM文

DIM文は配列の大きさ(添字がとり得る最大値)を宣言するためのステートメントである。例えば、DIM A(12), X(3)などはDIM文の使用例である。BASICがDIM文に出会うと、メモリーの中に配列変数の大きさに応じて下図のように記憶場所を確保し、データの読み出し、書き込みには変数名と添字によって、然るべき記憶場所が割当てられる。

記 憶 装 置



DIM文が省略された配列はBASICが大きさを10に設定する。すなわち、DIM文で10の配列宣言をしたのと同じである。従って、添字の大きさが10以内の配列でも、配列宣言をすればメモリーの節約が計れる。配列宣言の添字には定数はもちろん変数を使用することもできる。

[例]

```
100 INPUT"no.of data";N
110 DIM X(N)
120 FOR I=1 TO N
130 INPUT X(I)
140 NEXT I
```

```
RUN
no.of data? 5
? 2
? 3
? 6
? 8
? 4
```

5-1-4 配列を使ったプログラム例

次に、配列を使ったプログラム例を幾つか示すことにする。配列の演算をする場合、普通の演算と同様にコンピュータは配列要素を1つずつ計算・処理して行くわけで、決して、一度に配列全体を計算・処理するのではない。従って、配列の要素は計算・処理の全ての段階で特定化されていなければならない。

〔例題1〕 N個の乱数を配列Aに記憶し、それを1行に3個ずつ表示せよ。

```
100 REM ex-1, (1)
110 INPUT "N="; N
120 DIM A(N)
130 FOR I=1 TO N
140 A(I)=RND(1)
150 NEXT I
160 FOR I=1 TO N STEP 3
170 K=I+2
180 IF K<=N THEN 190 ELSE K=N
190 FOR J=I TO K
200 PRINT A(J);
210 NEXT J
220 PRINT
230 NEXT I
240 END
```

RUN

N=? 11

```
.591065 .207991 .550967
.635863 .10411 .806334
4.29073E-02 .953862 .354289
.556498 .445778
```

別 解

```
100 REM ex-1, (2)
110 INPUT "n'="; N
120 DIM A(N)
130 FOR I=1 TO N
140 A(I)=RND(1)
150 NEXT I
160 FOR I=1 TO N
170 PRINT A(I);
180 IF INT(I/3)=I/3 THEN PRINT
190 NEXT I
200 END
```

RUN

n'=? 11

```
.591065 .207991 .550967
.635863 .10411 .806334
4.29073E-02 .953862 .354289
.556498 .445778
```

〔例 2〕 N個のデータを入力順と逆に印字せよ。

```
100 INPUT "no.of data=";N
110 DIM X(N)
120 FOR I=1 TO N
130 PRINT I;:INPUT X(I)
140 NEXT I
150 FOR I=N TO 1 STEP-1
160 PRINT I; X(I)
170 NEXT I
```

RUN

```
no.of data=? 5
1 ? 2
2 ? 6
3 ? 3
4 ? 4
5 ? 1
5 1
4 4
3 3
2 6
1 2
```

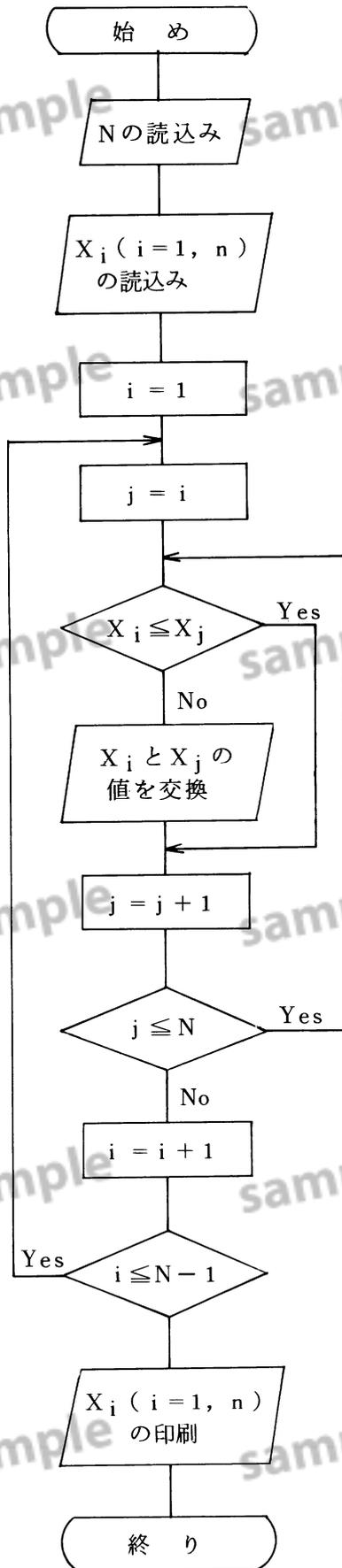
〔例 3〕 1番から 10番の社員番号を入力すれば、対応する社員名が出力されるプログラムを作成せよ。

```
100 REM シェイン ハンコウ
110 DIM A$(10)
120 FOR I=1 TO 10
130 READ A$(I)
140 NEXT I
150 DATA スズキ,タカハシ,イトウ,コハヤシ,ヒクチ
160 DATA カトウ,イシダ,ナカムラ,キムラ,フシウラ
170 INPUT "シェイン ハンコウ";N
180 IF N<=0 THEN 170
190 IF N>10 THEN 170
200 PRINT A$(N)
210 GO TO 170
```

RUN

```
シェイン ハンコウ? 3
イトウ
シェイン ハンコウ? 7
イシダ
```

[例4] N個 (≤ 100) のデータを読み込み、小さい順に並べ替え (ソート) して印刷せよ。



```

100 DIM X(100)
110 INPUT "data スワ="; N
120 FOR I=1 TO N
130 PRINT "x ("; I; ")=";
140 INPUT X(I)
150 NEXT I
160 FOR I=1 TO N-1
170 FOR J=I TO N
180 IF X(I)>X(J) THEN SWAP X(I),X(J)
190 NEXT J,I
200 FOR I=1 TO N
210 PRINT X(I);
220 NEXT I
230 END
  
```

```

RUN
data スワ=? 5
x ( 1 )=? 3.5
x ( 2 )=? 2.8
x ( 3 )=? 4.4
x ( 4 )=? 1.2
x ( 5 )=? 5.7
1.2 2.8 3.5 4.4 5.7
  
```

〔例 5〕 下図の様に、製品別、年度別の売上と費用の表がある。データを読んで売上－費用を計算することにより、製品別、年度別の利益を計算せよ。

売 上 表

製品 \ 年度	1978	1979	1980
A	125	135	144
B	86	78	95
C	31	65	88

費 用 表

製品 \ 年度	1978	1979	1980
A	88	102	112
B	78	82	86
C	24	28	32

```

100 DIM S(3,3),C(3,3),P(3,3)
110 FOR I=1 TO 3
120 FOR J=1 TO 3
130 READ S(I,J),C(I,J)
140 NEXT J,I
150 DATA 125,88,135,102,144,112
160 DATA 86,78,78,82,95,86
170 DATA 31,24,65,28,88,32
180 FOR I=1 TO 3
190 FOR J=1 TO 3
200 P(I,J)=S(I,J)-C(I,J)
210 NEXT J,I
220 FOR I=1 TO 3
230 FOR J=1 TO 3
240 PRINT P(I,J);
250 NEXT J
260 PRINT
270 NEXT I
280 END

```

```

RUN
37 33 32
8 -4 9
7 37 56

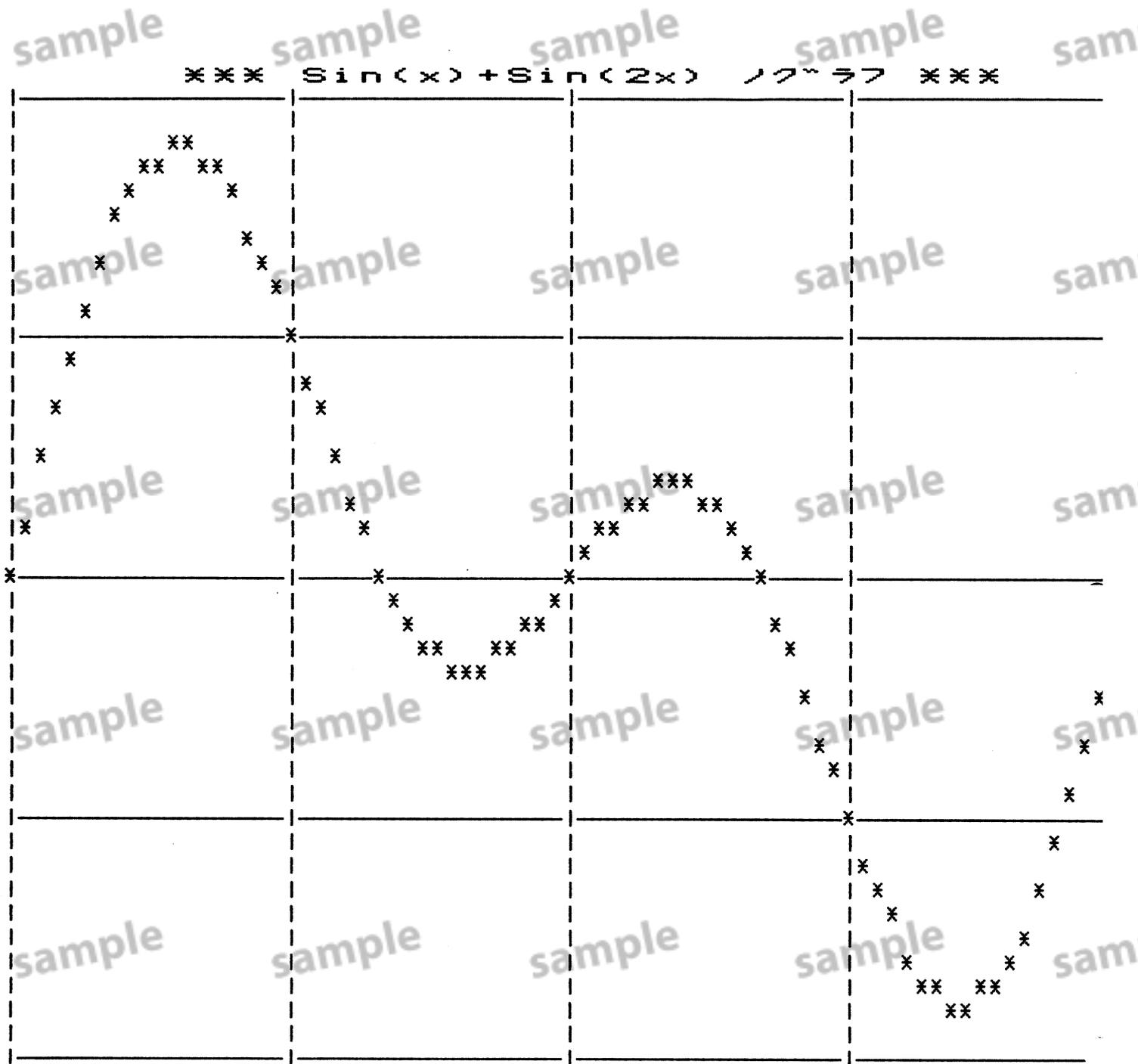
```

〔例 6〕 $\sin x + \sin 2x$ のグラフを作成せよ。

```

100 LPRINT CHR$(8);
110 LPRINT TAB(7);"*** Sin(x)+Sin(2x) / 7^7 ***"
120 LPRINT CHR$(8);
130 DIM A$(40,76)
140 FOR I=0 TO 40 STEP 10
150 FOR J=0 TO 76:A$(I,J)=" ":NEXT J
160 NEXT I
170 FOR J=0 TO 76 STEP 19
180 FOR I=0 TO 40:A$(I,J)="|":NEXT I
190 NEXT J
200 FOR J=0 TO 76
210 X=3.14159*J/38
220 Y=SIN(X)+SIN(2*X)
230 I=20-INT(10*Y+.5)
240 A$(I,J)="*"
250 NEXT J
260 FOR I=0 TO 40
270 FOR J=0 TO 76
280 LPRINT USING "!";A$(I,J);
290 NEXT J
300 LPRINT
310 NEXT I

```



5-2 その他の有用なステートメント

5-2-1 PRINT USING<フォーマット文>;<式のリスト>

目的：文または数値を指定したフォーマットでプリントする。

解説：<式のリスト>は、セミコロンで区切られたプリントされる文字または数式で構成される。<フォーマット文>は、引用符で囲まれた特別なフォーマット文字より構成される。これらのフォーマット文字（下記参照）が、プリントされる文字や数字の領域と書式を決定する。

文字領域：PRINT USING文が文字列をプリントするのに使われる場合は、次の2つのうちの1つのフォーマット文字を使って文字領域の書式を決める。

！ 与えられた文字列の最初の文字だけをプリントするよう指定する。

&n個の空白& 文字列の内、2+n個の文字がプリントされる。すなわち2つのアンド記号の間に空白がない場合には、2文字がプリントされ、空白が1つの場合には3文字が、など以下同様にプリントされる。もし与えられた文字列がこの領域よりも長い場合には、余分の文字は無視される。また、この領域の方が文字列よりも長い場合には、文字列は領域に左づめでプリントされ、あいた右側には空白が詰められる。次に文字領域フォーマット文字を使用したプログラム例を示す。

```
10 A$="LOOK"  
20 B$="OUT"  
30 PRINT USING "!" ; A$ ; B$  
40 PRINT USING "&□□&" ; A$ ; B$  
50 PRINT USING "&□□□&" ; A$ ; B$ ; "!!"  
  
RUN  
  
LO  
  
LOOKOUT  
  
LOOK OUT !!
```

数値領域：PRINT USING文を数値をプリントするのに使う場合には、以下に示す特別な文字によって数値領域の書式を指定する。

(ナンバ記号) このナンバ記号は各桁の位置を示すのに使う。この指定された桁は常に数字で満される。しかし指定した桁数よりもプリントされる数値の桁数の方が少ない場合は、その領域に右づめでプリントされ、左側に空いた部分には空白が詰められる。

。(小数点) 領域の任意の場所に小数点を挿入することができる。フォーマット文の指定に小数点に続く桁がある場合、その桁は(必要ならば0が)必ずプリントされる。

```
PRINT USING "###.##" ; 987.654  
987.65  
PRINT USING "##.##□□□" ; 10.2, 5.3, 66.789, .234  
10.20□□□□ 5.30 □□□□ 66.79□□□□ 0.23
```

最後の例では、プリントされる値を区切るために3個の空白がフォーマット文の最後に加えられている。

+ (プラス) フォーマット文の最初、または最後のプラス記号は、数値の前または後ろにその数値の符号(プラスまたはマイナス)をつける。

- (マイナス) フォーマット文の最後にマイナス記号を書くと、負の数値の後ろにマイナスがプリントされる。

```
PRINT USING "+##.##□□□" ; -68.95, 2.4, 55.6, -.9
```

-68.95 +2.40 +55.60 -0.90

PRINT USING "##.##-□□□"; -68.95, 22.449, -7.01
68.95 -□□□□22.45 □□□□ 7.01 -

** (2個のアスタリスク) フォーマット文の最初に2つのアスタリスクを書くと、数値領域のはじめの空白部分にアスタリスクがプリントされる。また**は、さらに2桁分の領域を確保する。

PRINT USING "***#.##□□□"; 12.39, -0.9, 765.1
* 12.4□□□* -0.9□□□765.1

¥¥ (2個の円記号) フォーマット文の最初に¥¥を使用すると、出力される数字の直前のスペースに円記号をプリントする。¥¥は数字2桁分の位置を確保するが、円記号自身ではこの場所のうち1個だけを使う。指数型式のときは¥¥符号を使用することはできない。

¥ (2個のアスタリスクと円記号) フォーマット文の最初に¥を使用すると、上記両方(**および¥¥)の機能を果す。上記条件はすべて適用されるが、異なる点は**¥は数字3桁分の位置を確保し、そのうちの1個は円符号であるという点である。

, (コンマ) フォーマット文の小数点の左側にコンマがある場合は、小数点の左側の数値の3桁ごとにコンマがプリントされる。フォーマット文の最後のコンマは文字列の一部としてそのままプリントされる。コンマは余分の領域を確保する。指数型式(^^^^)と共に使われた場合には無効となる。

PRINT USING "####, .##"; 1234.5
1, 234.50
PRINT USING "####.##, "; 1234.5
1234.50,

^^^ (4個の矢印) 桁指定文字の後ろの4つの上向き矢印を置くと、指数形式の書式を指定することができる。4つの上向き矢印は、E+nn, がプリントされる領域を確保する。任意の場所に小数点を指定することができる。有効桁は左づめでプリントされ、指数部はそれに合わせて調整される。前に+記号、または後ろに+か-記号がない限り、小数点の左側の1桁が空白または-記号に使われる。

```
90 INPUT A
100 PRINT USING ".####^^^~";A
Ok
run
? 234.56
.2346E+03
Ok
run
? 88888
.8889E+05
Ok
run
? -88888
.8889E+05-
Ok
```

% (パーセント) もし指定された数値領域の長さよりもプリントされる数値の桁数の方が長い場合には、数値の前にパーセント記号がプリントされる。また丸めが領域をはみ出す原因となった場合にも、丸めた数の前にパーセント記号がプリントされる。

```
PRINT USING "##.##" ; 111.22
```

```
%111.22
```

```
PRINT USING ".##" ; .999
```

```
%1.00
```

もし桁数の指定が 24 をこえると、"Illegal function call" エラーが起る。

追記：フォーマット文の前または後にフォーマット文字以外のキャラクタ（記号，英数字，カナ，グラフィック記号）を書くことができる。この場合には数値の前または後にそのキャラクタがプリントされる。

```
PRINT USING "ヘイキン ##.## 秒" ; 57.45
```

```
ヘイキン 57.5 秒
```

以下に PRINT USING文の簡単な例を示す。

[例 1]

```
-----
                        請求書
=====
秋山様
-----
                        請求金額=¥      12,345.
```

```
-----
                        請求書
=====
石田様
-----
                        請求金額=¥      25,874.
```

```
-----
                        請求書
=====
笹森様
-----
                        請求金額=¥      36,985.
```

```
-----
                        請求書
=====
竹田様
-----
                        請求金額=¥      65,478.
```

```
100 DATA 秋山
110 DATA 石田
120 DATA 笹森
130 DATA 竹田
140 FOR I=1 TO 4
150 READ A$
160 CLS
170 PRINT "請求先 ";A$;"様"
180 INPUT "請求金額=";X
190 PRINT:PRINT
200 LPRINT "-----"
210 LPRINT TAB(20);"請求書"
220 LPRINT "=====
230 LPRINT A$;"様"
240 LPRINT "-----"
250 LPRINT TAB(15); "請求金額=";
260 LPRINT USING"¥#####. ";X
270 LPRINT:LPRINT:LPRINT
280 NEXT I
290 LPRINT:LPRINT:LPRINT
300 END
```

〔例 2〕 製品コードと販売量を対にして読込んでいき、n 対の製品の販売量の大きい順にソート（並べかえ）しそれを棒グラフで示せ。

```
100 INPUT "products no.=";N
110 DIM P(N,2)
120 FOR I=1 TO N
130 INPUT "code &sales=";P(I,1),P(I,2)
140 NEXT I
150 REM sorting
160 FOR I=1 TO N-1
170 FOR J=I TO N
180 IF P(I,2)>=P(J,2) THEN 210
190 SWAP P(I,2),P(J,2)
200 SWAP P(I,1),P(J,1)
210 NEXT J,I
220 INPUT "max width <=60 ";W
230 T=P(1,2)/W
232 LPRINT " code";" vol."
240 FOR I=1 TO N
250 LPRINT USING "#####";P(I,1);
260 L%=P(I,2)/T
270 LPRINT "+";:IF L%=0 THEN 280
280 FOR J=1 TO L%:LPRINT "*";:NEXT J
290 LPRINT
300 NEXT I
```

5-2-2 GOSUB <行番号>

```
⋮
RETURN
```

目的：サブルーチンへ分岐し、戻る。

解説：<行番号>はサブルーチンの最初の行の行番号である。また、行番号の代わりに * × × × のようにラベルをつかうことができる。

一つのサブルーチンが、プログラムの中で何度呼ばれてもかまわないし、また他のサブルーチンによって呼ばれることもできる。このサブルーチンの多重化（ネスタイング）は、使えるメモリの容量にのみ制限される。

サブルーチンの RETURN 文を実行すると BASIC は、最も最近の GOSUB 文の次の文へ戻る。1 つのサブルーチンに、1 以上の RETURN 文があってもかまわないが、プログラムのロジックは正しくサブルーチンの中の正しい RETURN 文を指示するようになっていなくてはならない。サブルーチンはプログラムのどこに置かれてもかまわないが、一目でメインプログラムと区別できるように書くことを勧める。また、不用意にサブルーチンに入ってしまうことを防ぐためには、STOP, END, またはサブルーチンを飛びこすよう指示する GOTO 文をサブルーチンの前に置くようにする。

```
例：10 GOSUB 40
      20 PRINT "BACK FROM SUBROUTINE"
      30 END
```

```

40 PRINT "SUBROUTINE ";
50 PRINT "IN ";
60 PRINT "PROGRESS"
70 RETURN
RUN
SUBROUTINE IN PROGRESS
BACK FROM SUBROUTINE

```

5-2-3 ON<式> GOTO <行番号>[, <行番号>…]

ON<式> GOSUB<行番号>[, <行番号>…]

目 的 : <式>の評価により定まった値に基づき, 指定されたいくつかの行の1つに分岐する。

解 説 : <式>の値がリストのどの行番号の文に分岐するかを決定する。たとえば, <式>の値が3であったなら, 行番号のリストの3番目の行が分岐先となる。(もし値が整数にならない時は, 小数部分は切り捨てられる。)

ON~GOSUB文では, リストの各行番号はサブルーチンの最初の行の行番号でなければならない。

<式>の値が負になった場合には, "Illegal function call" エラーが起るが, 値が0またはリストの行番号の個数より大きくなった場合には, 次の行へプログラムの制御が移り, エラーは起らない。

例 : 100 L=4 : ON L-1 GOTO 150, 300, 320, 390

(この例では制御は320行へ移る)

5-2-4 論 理 式 (その2)

論理演算子として, 次のNOT, AND, ORが使える。以下にはこれらの論理演算の結果が示されている。

NOT (否定)

X	NOT X
1	0
0	1

AND (論理積)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR (論理和)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

[例]

```
IF D>200 AND F<4 THEN 80
IF I>10 OR K<0 THEN 50
```

5-2-5 LINE INPUT [`<“プロンプト文”>`;]`<文字変数>`

目的：1行全体（255文字以内）を区切ることなく文字変数に入力する。

解説：プロンプト文は、入力を受け入れる前にディスプレイにプリントされる文章である。プロンプト文に書かない限り疑問符（?）はプリントされない。プロンプトからキャリッジターンまでに入力されたすべての入力が`<文字変数>`に代入される。LINE INPUT文の実行はコントロール-CまたはSTOPキーの入力によって中断することができる。その場合、N-BASICはコマンドレベルに戻り、Okとプリントする。CONTを入力するとLINE INPUT文より実行は継続される。

5-2-6 DEFFN`<名前>`[(`<パラメータリスト>`)]=`<関数の定義>`

目的：ユーザーによって書かれた関数を定義し、名前をつける。

解説：`<名前>`は正しい変数名でなければならない。このFNに引き続く名前が関数の名前となる。`<パラメータのリスト>`はダミー変数であり、関数の定義式に使われている変数名のリストである。リスト内の各変数名はコンマで区切る。`<関数の定義>`はその関数の演算内容を記述する式で、1行の範囲に限られる。この式に現われる変数名は単に関数を定義するだけで、したがって同じ名前をもつプログラム中の変数には何の影響も及ぼさない。また、関数の定義式に使われた変数名はパラメータリスト中にあってもなくてもかまわない。もしあれば、そのパラメータの値は関数が呼ばれたときに与えられる。そうでなければその変数の現在の値が使われる。

パラメータリストに変数は、関数の呼び出しの際に与えられる変数、または値に1対1に対応する。

このユーザ定義関数は数値関数であっても文字関数であってもかまわない。関数名によって型が指定されたなら、式の値は関数名により指定された型とアーギュメント（引き数）との型が合わせられなければ“Type mismatch”のエラーが発生する（一方が数値、他方が文字の場合）。

DEFFN文はそれが定義する関数が呼ばれる前に実行されなければならない。もし定義される前に関数が呼ばれた場合は、“Undefined user function”（未定義ユーザ関数）エラーが発生する。また、ダイレクトモードではDEFFN文は使えない。

例：410 DEFFNB(X, Y)=X³/Y²
420 T=FNB(I, J)

5-2-7 型 宣 言 文

DEFINT/SNG/DBL/STR

書 式：DEF<型><文字の範囲>

ただし、<型>はINT, SNG, DBL, またはSTR。

目 的：変数の型を整数，単精度，倍精度，または文字に指定する。

解 説：DEF<型>文は、<文字の範囲>で指定されたある文字ではじまる変数名の変数をその型の変数に定義する。ただし、変数の型の指定においては、型宣言文字が常にDEF型文に優先する。

もし型宣言文がなければ、BASICは型宣言文を伴わない変数はすべて単精度変数とみなす。

例：10 DEFDBL L-P L, M, N, O そしてPではじまる変数はすべて倍精度変数になる。

10 DEFSTR A 文字Aではじまるすべての変数は文字変数になる。

5-2-8 LOCATE<水平位置>,<垂直位置>[, <カーソルスイッチ>]

目 的：CRT画面上の任意の位置にカーソルを移動する。

解 説：水平位置，垂直位置で示された画面位置にカーソルを移動する。なお画面左上が0, 0の位置となっている。

また、通常カーソルは点滅しているが、<カーソルスイッチ>を0に指定することでカーソルを消すことができる。

例：LOCATE 10, 5, 0 : PRINT "ASCII"

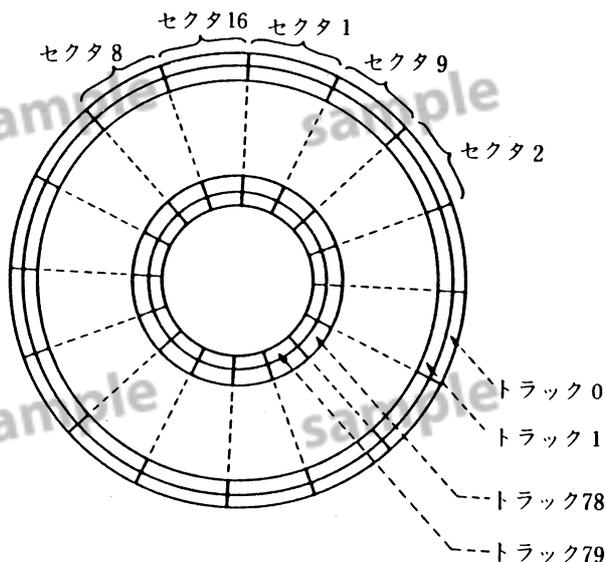
5-3 データファイルの使い方

BASICの変数に記憶されたデータは電源を切ると同時に失われてしまうので、再度の利用のためには、大変不便である。すなわち、キーボードから再度データを入力するのは手間がかかり、ミスを犯しやすい。DATA文からの入力、その点すぐれているが、しかし、大量のデータを扱うには不向きであるし、その上、DATA文はプログラムの一部であるため、他のプログラムによるデータの利用には向いていない。しかしながらビジネスにおいては、大量のデータを複数のプログラムが繰返し利用するということが多いのである。そこで、こうした利用に適したデータの記憶、保存装置がパーソナルコンピュータにおいて用意されている。そうしたもののなかで、最も経済的なのがカセットテープレコーダで、カセット・テープを記憶媒体とするものである。市販されているオーディオ用のものが、そのまま利用可能である。

カセット・テープへの記録はオーディオにおける曲の録音と同様、テープの頭から順次記録し、再生もまた頭から順次行なり方式である。テープの中間に録音された曲の取出しが不便なのは、データの記録の場合も全く同様である。まして、曲の中の特定の小節から正しく再生するのは不可能であるのと同様、データ・ファイルの中から、ある順番のデータだけを取り出すことはできない。

データの記録としてより便利な装置としては、ミニフロッピーと呼ばれるドーナツ状の円盤を記録媒体にしたディスク装置がある。フロッピーディスクもカセットテープも共に磁気によりビット信号を記録するという点では同じ原理を利用したものであるが、データにアクセスする方法が異なるため、データの入出力に要する時間はおおいに異なり、有用性も違ったものになる。すなわち下図に示されるように、フロッピー・ディスクはデータの書込み場所がセクター別に定められている上、どのセクターのデータでもダイレクトに読取り、書込みできるので、データへのアクセス・タイム（読み書き時間）はほぼ一定で、かつ短時間に行なうことができる。この様な特徴は、多くのデータ（ファイル）の中から、一部のデータを読取る（検索する）

フロッピーディスクと記憶容量



5インチ両面倍密度倍トラックフロッピーディスク

データ記憶容量

ユニット当り	655. 360バイト
1ドライブ当り	327. 680バイト
1ドライブ当りのトラック数	80トラック
1トラック当りのセクタ数	16セクタ
1セクタ当りのデータ数	256バイト

場合や、一部のデータを修正する場合に大変有用である。なお、特定のデータ（レコード）を直接読み書きできることをランダム・アクセスという。フロッピーディスクはランダム・アクセス方式でデータを記録できるだけでなく、カセット・テープのような順次にデータを記録す

することもできる。

5-3-1 シーケンシャル・ファイルとランダム・ファイル

意味のある一群のデータの集まりをファイルと呼ぶが、このファイルには、通常データ・ファイルとプログラム・ファイルの2つがある。記録・保存という観点からは、データとプログラムとの違いはないのである。

データ・ファイルは構造的にシーケンシャル・ファイル（順編成ファイル）とランダム・ファイルに分けられる。シーケンシャル・ファイルはデータがファイルの頭から順次書き込まれているもので、読み取りもまた、頭から順次行なう方式のファイルである。一部のデータだけがが必要な場合にも、ファイルの頭から順次読み取りしなければならないので、ファイルが大きくなると、余分な時間がかかり、データがファイルのどの部分に記憶されているかで、読み取りの時間が大いに異なるという欠点をもつ。しかしながら、シーケンシャル・ファイルは構造が簡単なので、全てのデータの読み取り、あるいは書き込みを行なう場合には、効率的である。例えば、配列の値を記憶するには、配列の次元に関係なくシーケンシャル・ファイルが便利である。

ランダム・ファイルはファイル全体のデータを取扱うというよりは、むしろ一部のデータを読み書きする場合に便利でかつ効率の良いファイルである。情報検索、一部データの更新、新データの追加あるいは削除といった処理はランダム・ファイルの方がシーケンシャル・ファイルに比べてはるかに短時間で処理することができる。ランダム・ファイルでは記憶されている特定のデータ（レコード）がどこに記憶されているかを何らかの方法で知らなければ、データの読み書きができない。そのために幾つかの方法が考え出されている。以下はその中の代表的なものである。

- (1) レコードの中の一部の属性、例えばキー・コードからレコードが記憶されている記憶場所（アドレス）を直接算出する。そのためにはアドレスがコードの関数として定められていなければならない。こうしたファイルはダイレクト・アクセス・ファイルと呼ばれる。
- (2) キーコードと対応するアドレス・テーブルを別ファイルで持って、最初にテーブルを参照し、次いでデータの読み書きを行なうという2段で処理を行なう。こうしたテーブルのことをインデックス・ファイルと呼ぶ。インデックス・ファイルは通常シーケンシャル・ファイルであるが、キーコードからアドレスを読み取るための方式は種々考えられている。
- (3) 最初のレコードのアドレスはディレクトリから読み取るが、次のレコードのアドレスは先行のレコードに記録されているという方式のファイルである。この方式ではレコードが次ぎつぎに関連づけられ、チェーン構造で結びつけられている。このように先行するレコードが後続するレコードのアドレスをデータの一部として保有するやり方をチェーンニングと呼ぶ。チェーン方式のファイルでは、レコードの更新、追加・削除が極めて容易に行なえる。

一方、プログラムを記憶するためのファイル（プログラム・ファイル）はその性質上、記録

媒体がカセット・テープであれ、フロッピー・ディスクであっても、こうした媒体に関係なく、シーケンシャル・ファイルに記録される。プログラムは全てのデータが順序正しく読み込まれなければ意味がないからである。

5-3-2 ファイル名

N-BASICはファイルを256バイトの長さのレコード(ディスク/セクターに相当)単位で取扱うが、実際の入出力に当っては8レコードよりなる「クラスタ」を単位として行なうので、高速な入出力が可能になっている。

一つのファイルが扱うことのできる最大のレコード数は544レコード(139Kバイト)である。また、1枚のディスクには最大68個のファイルが収容できるが、ただし、全体で139Kバイトの容量を越えることはできない。

ファイルの管理はコンピュータが自動的に行なうので、ユーザはディスクのどこにファイルが収容されているかを知る必要は通常ない。

ファイル名は次の書式を取る。

<ファイル名> [. <ファイルタイプ>]

ファイル名は6文字で、小数点に続く<ファイルタイプ>はもしあれば3文字までの文字列で構成される。もし文字数がそれ以下の場合は不足分に空白が埋められ、それ以上の場合は超加分が無視される。

5-3-3 OPEN文とCLOSE文

ディスク上に新たにデータ・ファイルを作成するか、あるいはデータを追加するか、またはデータ・ファイルからデータを読み取る場合には、その旨を宣言する必要がある。これがOPEN文である。また、ファイルの使用が終了したならば、CLOSE文でそれを知らせる。

書 式：OPEN <ファイル名> [FOR <モード>] A\$ [#] <ファイル番号>

ただし<モード>は次のいずれか。

INPUT

OUTPUT

APPEND

解 説：<モード>は単にファイルのポインタの最初の位置を決定し、もしそのファイルが存在しない場合の対処の方法を決めるものである。それぞれのモードでの動作は次のとおりである。

INPUT ポインタの最初の位置はファイルの始め、ファイルが見つからない場合はエラーになる。

OUTPUT 最初の位置はファイルの始め、常に新しいファイルを作る。

APPEND 最初の位置はファイルの終り、ファイルが見つからない場合はエラーになる。

FOR<モード>節が省略された場合には、ポインタの最初の位置はファイルの始めになる。
もしファイルが見つからない場合は、その名前のファイルが作られる。

1レコードの長さは256バイトで変更することはできない。

あるファイル名のファイルに複数のファイル番号を割りあてて開くことができる。このことにより、違った目的のために異なる属性を使用することができる。また、プログラムをわかり易くするために、アクセスの方法ごとに異なるファイル番号を使うこともできる。それぞれのファイル番号は独立したバッファを持つので、そのレコードがディスクットに書きこまれるまではある番号の下でのファイルの変更は他の番号で参照することはできない。

ランダム・ファイルに対してはFOR節を省略するのが普通である。

CLOSE文については次の通りである。

書 式：CLOSE [[#]<ファイル番号>[, [#]<ファイル番号>…]]

目 的：ディスクファイルへの出力を終了する。

解 説：<ファイル番号>を伴わないCLOSEは、全てのオープン状態にあるファイルをクローズする。

CLOSEが実行されると、特定のファイルとファイル番号の間のつながりはなくなる。したがって、そのファイルは同じ、または異なったファイル番号によって再びOPENすることができる。同様にそのファイル番号は他のファイルをOPENするために使用できる。

シーケンシャル出力ファイルに対するCLOSEの実行は、最後の出力バッファを書き込む。

END文とNEWコマンドは、常に自動的に全てのファイルをCLOSEする。
(STOPはディスクファイルのCLOSEは行なわない。)

[例]

```
100 OPEN "DFILE" FOR INPUT A$#1
110 INPUT #1, A, B, C
.....
150 CLOSE #1
.....
```

なお、ファイル番号は最大15までであるが、この数字はシステムのイニシャライズの際のHow many FILES?に回答して入力した数字を超えることはできない。

5-3-4 シーケンシャル・ファイルの入出力文

(1) INPUT# (ディスク)

書 式：INPUT#<ファイル番号>, <変数名>[, <変数名>…]

目 的：ディスクのシーケンシャルデータファイルからデータを読み込み、プログラムの変数に代入する。<ファイル番号>が-1のときはカセットからの入力を意味す

る。

解説：〈ファイル番号〉は、OPEN文によってそのファイルを入力としてオープンしたときに使った番号である。〈変数のリスト〉はファイルのデータを割りあてる変数名のリストである。（変数の型は変数名によって指定されたものに一致してはならない。）INPUT#文ではINPUT文のように疑問符がプリントされることはない。

INPUT#を実行するともしバッファが空で“バッファ変更フラグ”がセットされていればそれをディスクに書き出す。ついで次のレコードをバッファに読み込む。

ファイル中のデータは、INPUT文に対して入力するデータと同じようになっていなくてはならない。数値の場合は、先に続く空白、キャリッジリターン、そしてラインフードなどは無視される。つまり、空白、キャリッジリターン、またはラインフード以外の最初の文字が数値データの始まりとされているわけである。数値は、空白、キャリッジリターン、ラインフード、またはコンマによって区切られる。

もしN-BASICがシーケンシャルデータファイルの文字列データを読み込む場合も、先に続く空白、キャリッジリターン、ラインフードは無視される。つまり、空白、キャリッジリターン、ラインフード以外の最初の文字を文字列データの始まりとみなす。もしこの最初の文字が引用符（"）の場合は、文字列データは最初の引用符から2番目の引用符のあいだに読み込まれたすべての文字で構成される。したがって、引用符で囲まれた文字列は、引用符を文字として含むことはできない。文字列の最初の文字が引用符でない場合には、その文字列は引用符で囲まれない文字列として扱われ、コンマ、キャリッジリターン、またはラインフード（、または255文字が読み込まれた後）で区切られる。数値または文字列データを読んでいる途中でファイルの終り（EOF）に達した場合は、そのデータ項目はそこで区切られる。

(ii) LINE INPUT#

書式：LINE INPUT#〈ファイル番号〉, 〈文字変数〉

目的：1行全体（254文字以内）を区切ることなくシーケンシャルディスクファイルより文字変数に読み込む。

解説：〈ファイル番号〉は、OPEN文によってそのファイルを入力としてオープンしたときに使った番号である。LINE INPUT#文は、シーケンシャルファイルより、キャリッジリターンまでの、すべての文字を読む。次のLINE INPUT#はそしてそのキャリッジリターンと続くラインフードの組を飛ばして、次のキャリッジリターンに至るまでのすべての文字を読む。LINE INPUT#文は、データファイルのそれぞれの行がいくつかの欄に分割されているときや、アスキー

モードでセーブしたN-BASICのプログラムを他のプログラムでデータとして読み込むときなどに特に便利である。

```
例： 10 OPEN "LIST" AS #1
      20 LINE INPUT #1, C$
      30 PRINT #1, C$
      40 CLOSE 1
```

(iii) PRINT# 及び PRINT# USING

書式：PRINT# <ファイル番号>, [USING <フォーマット文>;]
[<式の列>]

目的：シーケンシャルファイルにデータを書く。 <ファイル番号> が -1 のときはカセットへの出力を意味する。

解説： <ファイル番号> はそのファイルが出力としてオープンされたときに指定された番号である。 <フォーマット文> は 5-2-1 で説明したフォーマット文字によって構成されている。式のリストは、ファイルに書き込まれる数値式または文字式あるいはその両方である。

PRINT# を実行すると "バッファ変更フラグ" をセットする。もしバッファが一杯になったら、ディスクに書き出し、ファイルの最後に達していなければ次のレコードの読み込む。

PRINT# 文はディスクのデータを圧縮することは行なわない。すなわち、PRINT 文によってディスプレイに表示されるものと全く同じイメージがディスクに書き込まれる。従って、これらのデータがディスクから正しく入力できるように、ディスクに書き込まれたデータは適切に区切られるよう注意しなければならない。 <式のリスト> の中の数値式はセミコロンで区切るようにしなければならない。たとえば、

```
PRINT# 1, A ; B ; C ; X ; Y ; Z
```

(もし区切り記号にコンマを使うと、プリント領域の間に挿入される余分な空白もディスクに書き込んでしまう。)

リストの中の文字表記もセミコロンで区切る必要がある。ディスク上に文字表記を正しく書き込むためには、式のリストの中に独立した区切り記号を入れる必要がある。

例をあげると、A\$ = "CAMERA" そして、B\$ = "93604-1" のとき次の文、
PRINT# 1, A\$; B\$

はディスクに CAMERA 93604-1 と書き込む。これは区切り記号がないため 2 つの別の文字列として入力することはできない。この問題を解決するには、次のように PRINT# 文中に独立に区切り記号を入れることが必要である。

```
PRINT# 1, A$ ; " , " ; B$
```

これによってディスクに書き込まれるイメージは、

```
CAMERA, 93604 -1
```

となり、2つの文字変数として再び読み込むことができる。

もし文字列それ自身がデータとしてコンマ、セミコロン、意味のあるはじめの空白、キャリッジターン、またはラインフードなどを含む場合は、別の引用符、CHR\$(34)によって囲んでディスクに書く必要がある。

例をあげると、A\$="CAMERA, AUTOMATIC" また、B\$=" 93604 -1" のとき、次の文

```
PRINT#1, A$, B$
```

は次のようなイメージをディスクに書く。

```
CAMERA, AUTOMATIC 93604 -1
```

そして次の文、

```
INPUT#1, A$, B$
```

は、“CAMERA”をA\$に、そして“AUTOMATIC 93604 -1”をB\$に読み込む。これらの文字をディスク上で正しくわかるには、CHR\$(34)により引用符をディスクのイメージに書き込む必要がある。次の文

```
PRINT#1, CHR$(34); A$; CHR$(34); CHR$(34); B$; CHR$(34)
```

は次のようなイメージをディスクに書き込む。

```
"CAMERA, AUTOMATIC" " 93604 -1"
```

すると次の文、

```
INPUT#1, A$, B$
```

は“CAMERA, AUTOMATIC”をA\$に、“ 93604 -1”をB\$に読み込む。

PRINT#文は、USINGと共に使ってディスクファイルのフォーマットを制御することができる。

(iv) WRITE

書式：WRITE #<ファイル番号> [, 式の列]

目的：シーケンシャル・ファイルにデータを書く。

解説：PRINT#とほとんど同じ機能をもっているが、WRITE#文の方は、不要な空白桁は詰められ、それぞれの式の値の間は必ずコンマで区切られる。また、文字列はダブルクォーツ(")で囲まれて出力される。

5-3-5 ランダム・ファイルの入出力文

ランダム・ファイルではコンピュータとディスクとのデータのやりとりが、256バイトを単位とするランダム・ファイル・バッファ（メモリ）を介しておこなわれる。バッファから変数へのデータの書き込みは次のFIELD文を参照のこと。

(i) GET

書式：GET[#]<ファイル番号>[, <レコード番号>]

目的：ランダムディスクファイルからランダムバッファに1レコード読みこむ。

解説：<ファイル番号>は、そのファイルをOPEN文によって開いた時に指定した番号である。<レコード番号>が省略された場合は、（最後のGET文の後の）次のレコードがバッファに読み込まれる。レコード番号に使える最大の数は544である。

(ii) PUT

書式：PUT[#]<ファイル番号>[, <レコード番号>]

目的：ランダムディスクファイルにランダムバッファから1レコードを書き込む。

解説：<ファイル番号>はそのファイルを開くときに指定した番号である。<レコード番号>が省略された場合は、（最後にPUT文により書いた）次の番号が割り当てられる。レコードの番号の上限は、544である。

PUT文を実行するとまずバッファの内容を指定されたレコードに書き出し、“シーケンシャル入出力無効”フラグをGET動作が実行されるまでセットする。

5-3-6 FIELD文と変換関数

ランダム・ファイルの1レコードの大きさは256バイトに決められている。すなわち、文字ならば1レコード256文字まで書き込むことができるし、単精度の数値データならば、4バイトで記憶されるので64個のデータが収容できる。整数型のデータならば、128個が、また倍精度のデータならば32個が収容できる。文字型や精度の異なるデータを1レコード内に混ぜて記憶することもできる。

FIELD文は1レコード256バイトをどのように区分して利用するかを明示するためのステートメントである。レコードといっても実際はランダムファイルバッファにおける256バイトの利用区分を明らかにするものである。FIELD文は宣言文であって、バッファにデータの書き込みを行なうわけではない。これの実行はLSET文又はRSET文によって行なう。

(i) FIELD

書式：FIELD[#]<ファイル番号>, <フィールド幅>AS<文字変数>…

目的：ランダムファイルバッファの中に変数用の領域を割り当てる。

解説：プログラムがランダムディスクとファイルの間で、レコードをPUTまたはGET文で転送する前に、ランダムディスクバッファを各変数のデータをたくわえる「フィールド」に分割しなくてはならない。（1レコードは256バイトである）<ファイル番号>はそのファイルを開いたときに指定した番号である。<フィー

ルド幅>は<文字変数>に割り当てる文字数です。

例をあげると、

```
FIELD 1, 20 AS N$, 10 AS ID$, 40 AS ADD$
```

は、ランダムファイルバッファの最初の 20 桁 (20 バイト) を文字変数 N\$ に、次の 10 桁を ID\$ に、そしてさらに次の 40 桁を ADD\$ に割り当てる。FIELD 文はランダムファイルバッファに実際にデータを書き込むことはしない。(LSET/RSET 及び GET の項を参照せよ。)

1 つの FIELD 文で 256 バイト以上を割り当てようとするとき “Field overflow” エラーが起こる。同じファイルに対していくつ FIELD 文が実行されてもかまわない。

注 意：FIELD 文で割り当てられた変数名を INPUT 又は LET 文で使用してはいけない。一度変数名が FIELD 文によって割り当てられると、その変数のポインタはランダムディスクバッファ内の相当する位置を示すが、しかし、その後に INPUT または LET 文でその変数名が使われると、そのポインタは文字変数領域に移されてしまう。そして次に、その FIELD 文によって割り当てられた変数を使うランダムディスクファイルに対する入出力を試みると、正しい動作が行なわれない。この例からも明らかのように、FIELD 文は専有するバイト数と位置を文字型変数に割り当てることによって、データの区分を行なうので、数値データも一度文字型に変換する必要がある。こうした役割をするのが変換関数である。

多数のフィールドに分割し、変数を割当てる場合には、次のように配列変数を使うのが便利である。

```
100 DIM A$(14)
110 OPEN "cfile" AS #1
120 S%=0
130 FOR II=1 TO 14
140 READ FF%
150 FIELD 1, S% AS DUMMY$, FF% AS A$(II)
160 S%=S%+FF% : NEXT II
170 DATA 2, 40, 6, 10, 2, 6, 2, 2, 2, 4, 4, 5, 8, 8
```

(ii) LSET 及び RSET

書 式：LSET <文字変数> = <文字表記>

RSET <文字変数> = <文字表記>

目 的：文字領域をメモリからランダムデータバッファに移す (PUT 文の準備として)。

解 説：LSET, RSET の実行に先立って FIELD 文が実行されていなくてはならない。もし <文字表記> が、FIELD 文によって <文字変数> に割り当てた数よりも少ないバイト数しか必要としない場合には、LSET 文はそのフィールドに左づめで RSET は右づめでフィールドを満していく。(余分のフィールドには空白が詰め

られる。)もし文字列がそのフィールドよりも長い場合には、右側から文字が失なわれる。数値はLSETまたはRSET文によって移動する前に文字列に変換しなくてはならない。

(iii) CVI, CVS, CVD

書 式: CVI (<2バイトの文字列>)

CVS (<4バイトの文字列>)

CVD (<8バイトの文字列>)

動 作: 文字で表現された値を実際の数値データに変換する。ランダムディスクファイルから読み出した数値は文字から数に変換しなくてはならない。CVIは2バイトの文字列を整数に, CVSは4バイトの文字列を単精度形式数値に, CVDは8バイトの文字列を倍精度形式の数値にそれぞれ変換する。

例: 70 FIELD #1, 4 AS N\$, 12 AS B\$...

80 GET #1

90 Y=CVS(N\$)

(iv) MKI\$, MKS\$, MKD\$ (ディスク)

書 式: MKI\$ (<整数表記>)

MKS\$ (<単精度表記>)

MKD\$ (<倍精度表記>)

動 作: 数値を文字に変換する。LSETまたはRSET文によりランダムファイルバッファに入れる数値はすべて文字に変換しなくてはならない。MKI\$は整数を2バイトの文字列に, MKS\$は単精度形式の数値を4バイトの文字列に, MKD\$は倍精度形式の数値を8バイトの文字列にそれぞれ変換する。

例: 90 AMI=(K+T)

100 FIELD #1, 8 AS D\$, 20 AS N\$, ...

110 LSET D\$=MKS\$(AMI)

120 LSET N\$=A\$

:

160 PUT #1

以上、述べてきた事を要約し、ランダムファイルの書き込み、読み取りの手順を例をあげて示すと次の通りである。

(1) レコードの設計: 256バイトをどのように利用するかを決定する。例えば、次のように設定する。

顧客コード: 5桁(4バイト) (変数はCODE)

顧客名: 20文字(20バイト)(変数はNAME\$)

住所: 40文字(40バイト)(変数はADD\$)

売上金額: 5桁(2バイト) (変数はSALES%)

売上年計：7桁（4バイト）（変数はYSAL）

未利用分 256 - 70 = 186バイト

(2) ファイル名をCUSTOMとする。

(3) ファイル番号を1番とする。

(4) OPEN文はOPEN "CUSTOM" AS #1となる。

(5) FIELD文は次のようにする。

```
FIELD #1, 4 AS AI$, 20 AS A2$, 40 AS A3$, 4 AS A4$, 4 AS A5$
```

(6) コード，顧客名，住所，金額をバッファに書き込むためにLSETを用いる。

```
LSET AI$=MKS$(CODE)
```

```
LSET A2$=NAME$
```

```
LSET A3$=ADD$
```

```
LSET A4$=MKI$(SALES%)
```

```
LSET A5$=MKS$(YSAL)
```

もちろんこの段階では変数CODE，NAME\$，ADD\$，SALES%，YSALにはあらかじめ値がセットされていなければ意味がない。（A2\$やA3\$の代わりに，FIELD文で直接NAME\$，ADD\$を使うことはできない。）

(7) レコード番号の決定，例えば10番とする。

(8) バッファ1番の内容をディスク上のファイル名CUSTOMの中のレコード番号10番に記録する。

```
PUT #1, 10
```

(9) 次に8番のレコード番号のデータをディスクから読むことにする。

```
GET #1, 8
```

(10) フィールドの文字を数値に変換する。

```
CODE=CVS(A1$)
```

```
SALE%=CVI(A4$)
```

```
YSAL=(A5$)
```

FIELD文は変更するまでは前の宣言が生きている。また，文字型変数は無変換で良い。

```
NAME$=A2$
```

```
ADD$=A3$
```

文字型の場合にはこのように変数に代入しなくても，いきなりPRINT A2\$ A3\$とすることもできる。

(11) 最後にファイルを閉じる。

```
CLOSE #1
```

FIELD文を使って，同一のバッファに何度でも変数を割り当てることができる。次の例はFOR文を使って，繰返し，変数の割り当てを行なっている。

```

100 OPEN "DATA" AS #1
110 FOR K%=1 TO 5
120 FOR i=1 TO 20
130 FIELD #1, (i-1)*4 AS DUMMY$, 4 AS X$
140 INPUT X
150 LSET X$=MKS$(X)
160 NEXT i
170 PUT #1, K%
180 NEXT K%

```

これによって、1レコードに20個の単精度データが記録される。レコード数は5になる。DUMMY\$はバッファに既にかかれたデータを壊さないために、その領域をスキップするためのものである。(変数名はDUMMY\$でなくても、普通の文字型変数でも良い)

5-4-7 データ・ファイルを使ったプログラム

(i) シーケンシャル・ファイルの使用例

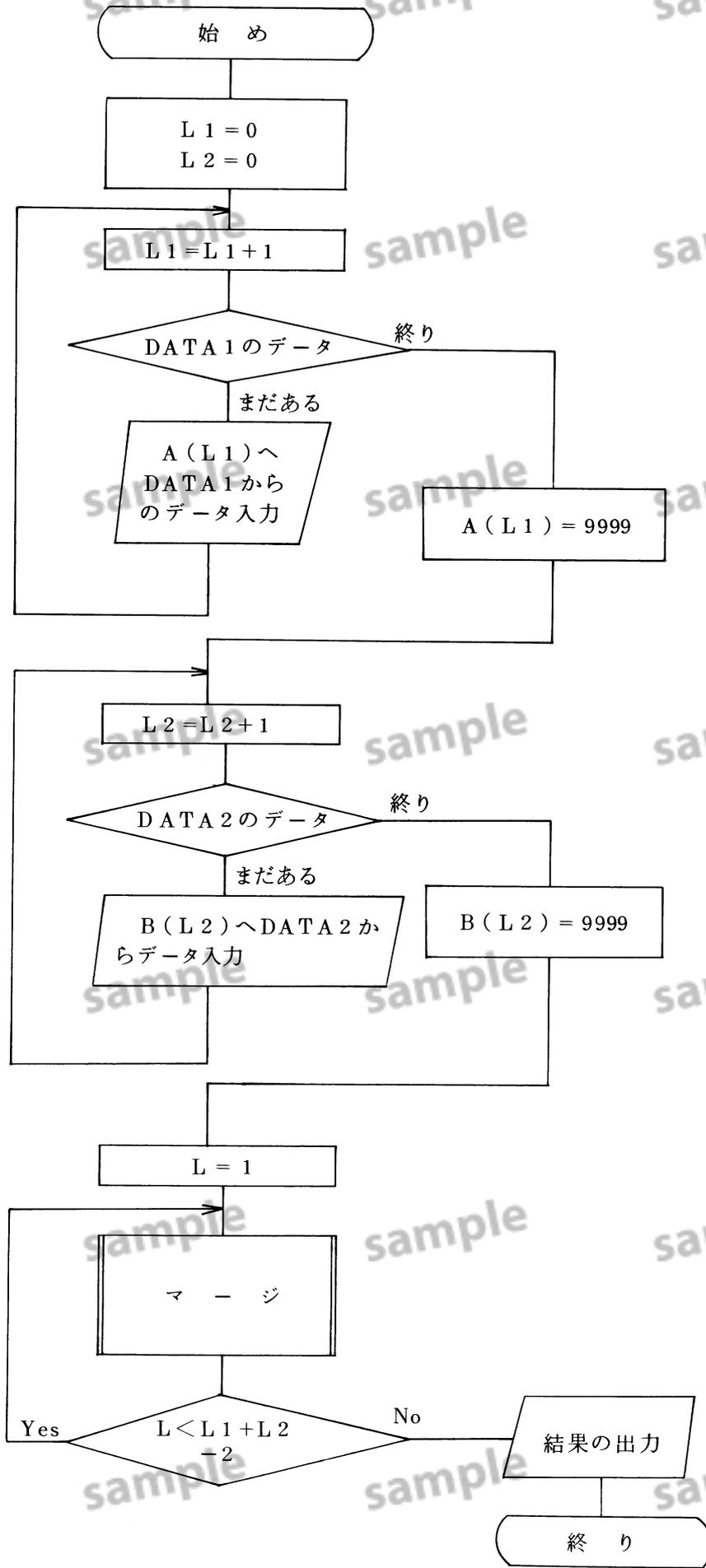
[例1] n個の乱数 ($0 \leq R < 1$) を小さい順にソートして、これをディスク・ファイル名 "DATA1" にまた同様に、m個の乱数 ($0 \leq R < 1$) を小さい順にソートして、これをディスク・ファイル名 "DATA2" に書き込むプログラムを作成せよ。

```

10 INPUT "inp n=" ; N
20 INPUT "inp m=" ; M
30 IF N >= M THEN Q=N ELSE Q=M
40 DIM A(N), B(M), R(Q)
50 FOR I=1 TO N
60 R(I)=RND(1):NEXT
70 H=N:GOSUB 200
80 FOR I=1 TO N:A(I)=R(I):NEXT I
90 FOR J=1 TO M
100 R(J)=RND(1):NEXT
110 H=M:GOSUB 200
120 FOR J=1 TO M:B(J)=R(J):NEXT J
130 OPEN "data1" AS#1
140 FOR I=1 TO N
150 PRINT#1,A(I):NEXT I
155 OPEN "data2" AS#2
160 FOR J=1 TO M
170 PRINT#2,B(J):NEXT J
180 CLOSE 1,2
190 END
200 FOR K=1 TO H-1
210 FOR L=K TO H
220 IF R(K) > R(L) THEN SWAP R(K), R(L)
230 NEXT L,K
240 RETURN

```

[例2] 例17で作成されたデータ・ファイルには小さい順にソートされているデータがディスク・ファイルの "DATA1" と "DATA2" に記憶されている。2つのデータ・ファイルからデータを読み込んで、これらのデータを統合(マージ)し全体として小さい順になるように並べて、この結果を以刷するプログラムを作成せよ。



```

100 DIM A(100),B(100),C(200)
110 OPEN"data1"AS#1
120 OPEN"data2"AS#2
130 FOR L1=1 TO 100
140 IF EOF(1) THEN A(L1)=999 :GOTO 160
150 INPUT#1,A(L1):NEXT L1:PRINT "mem out"
160 FOR L2=1 TO 100
170 IF EOF(2) THEN B(L2)=999:GOTO 190
180 INPUT#2,B(L2):NEXT L2:PRINT "mem out"
190 GOSUB 290
220 FOR I=1 TO L
230 LPRINT USING"###.###";C(I),
240 NEXT I
250 CLOSE 1,2
260 LPRINT
270 LPRINT"data su=";L
280 END
290 K1=L1-1:K2=L2-1
300 J1=1:J2=1
310 IF A(J1)>=B(J2) THEN L=L+1:C(L)=B(J2):J2=J2+1:GOTO 330
320 L=L+1:C(L)=A(J1):J1=J1+1
330 IF J1>K1 AND J2>K2 THEN RETURN
340 GOTO 310

```

RUNの結果

```

0.0038 0.0090 0.0137 0.0192 0.0354 0.0445 0.0450 0.0546 0.0574 0.0972
0.1051 0.1178 0.1259 0.1341 0.1487 0.1544 0.2170 0.2191 0.2875 0.2893
0.3023 0.3387 0.3525 0.3657 0.4232 0.4275 0.4289 0.4334 0.4688 0.4847
0.5104 0.5401 0.5913 0.6287 0.6327 0.6378 0.6574 0.6650 0.6783 0.7106
0.7255 0.7632 0.7657 0.7670 0.8034 0.8617 0.9078 0.9535 0.9820 0.9933
data su= 50
0.0288 0.0496 0.0895 0.1005 0.1312 0.1603 0.2575 0.2713 0.2902 0.2977
0.3167 0.3262 0.3589 0.3831 0.4217 0.4537 0.5048 0.5366 0.5660 0.5697
0.6126 0.6635 0.6816 0.7933 0.8466 0.8696 0.9418 0.9536 0.9592 0.9676
data su= 30
0.0038 0.0090 0.0137 0.0192 0.0288 0.0354 0.0445 0.0450 0.0496 0.0546
0.0574 0.0895 0.0972 0.1005 0.1051 0.1178 0.1259 0.1312 0.1341 0.1487
0.1544 0.1603 0.2170 0.2191 0.2575 0.2713 0.2875 0.2893 0.2902 0.2977
0.3023 0.3167 0.3262 0.3387 0.3525 0.3589 0.3659 0.3831 0.4217 0.4232
0.4275 0.4289 0.4334 0.4537 0.4688 0.4847 0.5048 0.5104 0.5366 0.5401
0.5660 0.5697 0.5913 0.6126 0.6287 0.6327 0.6378 0.6574 0.6635 0.6650
0.6783 0.6816 0.7106 0.7255 0.7632 0.7657 0.7670 0.7933 0.8034 0.8466
0.8617 0.8696 0.9078 0.9418 0.9535 0.9536 0.9592 0.9676 0.9820 0.9933
data su= 80

```

(ii) ランダムファイルの使用例

〔例1〕 住所録を作成せよ。ただし、レコードの項目および桁数は以下の通りとする。

名 前	住 所	電 話	内 線	あ ま り
20	50	20	5	161

```

90 REM ch is index if ch=0 that is map s not changed
100 REM directory program name is direct.bas
110 CLEAR 1000
120 DIM MAP(100),MAP$(100)
130 OPEN"address"AS#1
140 FIELD #1,20 AS N$,50 AS A$,20 AS T$,5 AS R$
150 /
160 /read map
170 /
180 OPEN "map" FOR INPUT AS#2
190 IF EOF(2) THEN230
200 I=I+1
210 INPUT #2,MAP(I):LINEINPUT #2,MAP$(I)
220 GOTO 190
230 SIZE=I:CLOSE #2
240 WR=LOF(1) / record already written
250 PRINT :PRINT : INPUT "read(r),write(w),modifi(m) or
end(e)";P$
260 IF P$="r" THEN 300
270 IF P$="w" THEN 460
280 IF P$="m" THEN 600 ELSE IF P$="e" THEN 850
ELSE PRINT "error": GOTO 250
290 /
300 /read
310 /
320 LINEINPUT"ナリ--";B$
330 IF SIZE=0 THEN 370
340 FOR I= 1 TO SIZE
350 IF B$=MAP$(I) THEN R= MAP(I):GOTO 380
360 NEXT I
370 PRINT"no such files ":GOTO 250
380 GET #1,R
385 PRINT CHR$(12)
390 PRINT"ナリ:";N$

```

```

400 PRINT "シューショ:";A$
410 PRINT "テック:";T$
420 PRINT "ナイセン:";R$
430 GOTO 250
440 /
450 /write
460 /
465 PRINT CHR$(12)
470 LINEINPUT "ナミ?";I$
480 IF SIZE>99 THEN PRINT "buffer full":GOTO 250
490 SIZE=SIZE+1 :WR=WR+1:CH=CH+1
500 MAP$(SIZE)=I$
510 MAP(SIZE)=WR
520 LINEINPUT "シューショ?";IA$
530 LINEINPUT "テック?";IT$
540 LINEINPUT "ナイセン?";IR$
550 LSET N$=I$
560 LSET A$=IA$
570 LSET T$=IT$
580 LSET R$=IR$
590 PUT #1,WR :GOTO 250
600 /
610 /modify
620 /
625 PRINT CHR$(12)
630 LINEINPUT "ナミ?";B$
640 IF SIZE=0 THEN 370
650 FOR I=1 TO SIZE
660 IF B$=MAP$(I) THEN R=MAP(I):GOTO 69
670 NEXT I
680 GOTO 370
690 GET #1,R
700 PRINT "old ナミ:";N$
710 PRINT "old シューショ:";A$
720 PRINT "old テック:";T$
730 PRINT "old ナイセン:";R$
740 LINEINPUT "new ナミ:";I$
750 IF I$(">)" THEN LSET N$=I$:MAP$(R)=I$:CH=CH+1
760 LINEINPUT "new シューショ?";IA$
770 IF IA$(">)" THEN LSET A$=IA$
780 LINEINPUT "new テック?";IT$
790 IF IT$(">)" THEN LSET T$=IT$
800 LINEINPUT "new ナイセン?";IR$
810 IF IR$(">)" THEN LSET R$=IR$
820 PUT #1,R
830 GOTO 250
840 /
850 /end
860 /
870 IF SIZE=0 THEN END
880 IF CH=0 THEN END
885 OPEN "map" FOR OUTPUT AS #2
890 FOR I=1 TO SIZE
900 PRINT #2,MAP(I)
910 PRINT #2,MAP$(I)
920 NEXT I
930 CLOSE
940 END

```

〔例 2〕 例 19 において、キーコードの氏名を姓または名だけでも検索できるように改良せよ。

```
361 L=LEN(B#)
362 FOR I=1 TO SIZE
363 FOR J=1 TO 20-L
364 S#=MAP$(I)
365 IF B#=MID$(S#,J,L) THEN PRINT MAP$(I):GOTO 367
366 NEXT J
367 NEXT I
```

以上のステートメントを例 19 のプログラムに挿入する。

ただし、このプログラムで使われている BASIC 関数の機能は次の通りである。

(1) LOC (<ファイル番号>)

動作：ランダムディスクファイルの場合は、GET または PUT 文（レコード番号を伴わない）が実行された時に使用される次のレコード番号を与える。シーケンシャルファイルの場合には、そのファイルが開かれていて以来読み出された、または書き込まれたセクター（256 バイトのブロック）数を与える。

例：100 PRINT LOC(1)

(2) LOF (<ファイル番号>)

動作：ランダムディスクファイルの PUT または GET 文によりアクセスされた最大のレコード番号を与える。

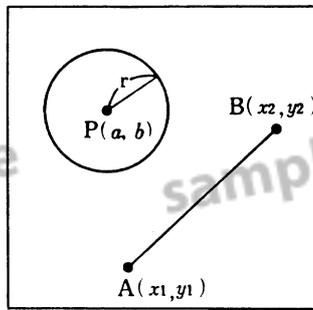
例：110 IF REC% > LOF(1) THEN PRINT "ニューリョク アヤマリ"

5.4. グラフィックス (図形処理)

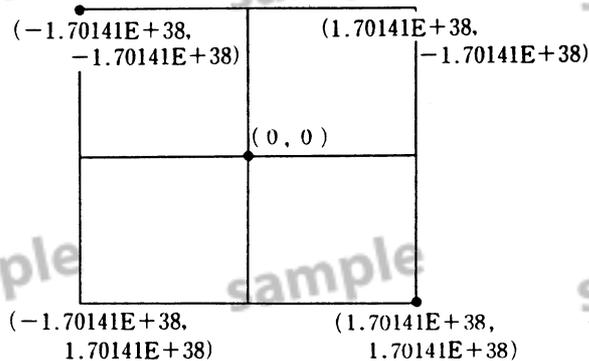
N 88 - BASICは複雑な図形を描いたり，図形の移動，拡大，縮小の操作を容易に行なうため，図形を表現する上で，使いやすい座標系が用意されている。すなわち，N 88 - BASICでは，(1)ワールド座標系，(2)オリジナルスクリーン座標系，(3)スクリーン座標系の3つの座標系がこれである。

(1) ワールド座標系

- ・グラフィックスで使用するプログラミング上の座標系を“ワールド座標系”とよぶ。例えば点 $P(a, b)$ を中心とする半径 r の円を描くとか，2点 $A(x_1, y_1)$ ， $B(x_2, y_2)$ を結ぶ線分 AB を描くには，論理的なワールド座標系を使用する。
- ・ワールド座標系は，N 88 - BASIC 中の論理的な座標系で，実数型数値で表現できる大きさの座標をもっている。



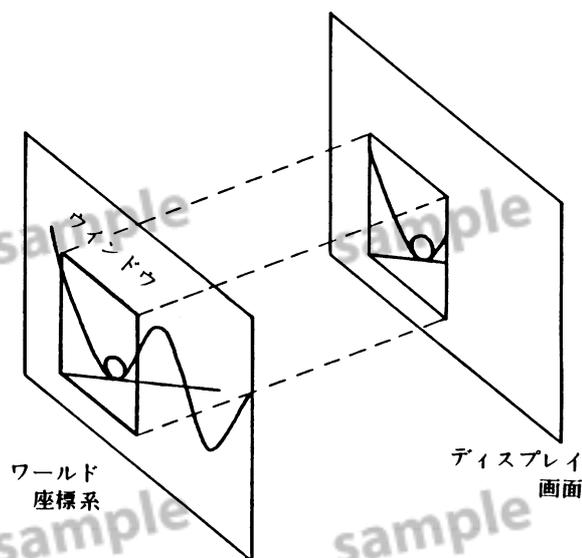
この例では CIRCLE 文，LINE 文を使用する。なお，ワールド座標系の範囲は次の通りである。



- ・ワールド座標系で表わしている図形の中から，実際にディスプレイに写し出す領域のことを“ウィンドウ (のぞきまど)”とよぶ。

ウィンドウを通して表示する領域は，その時点で必要な部分だけを取りだし，この部分が画面へ投影する対象になる。

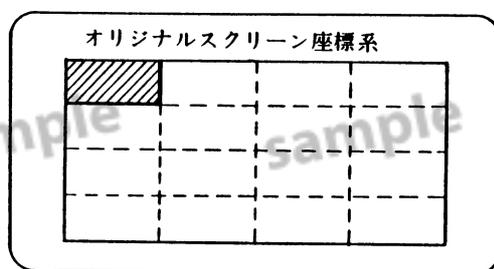
この領域を window 文で定義する。



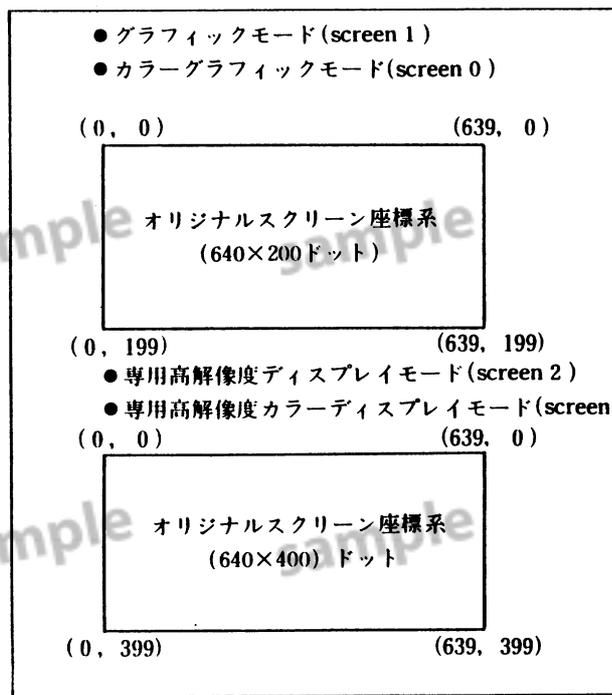
(2) オリジナルスクリーン座標系

実際に、ディスプレイ装置のブラウン管で表示される図形は、ドットで表わされている。このドットを単位としたディスプレイに、直接結びついた座標系を“オリジナルスクリーン座標系”とよぶ。最も身近な座標系で、物理座標である。

たとえば、次図のように、ディスプレイの左上、 $1/16$ の領域に図形を表示するような場合に、このオリジナルスクリーン座標系を使って表わさなければならない。

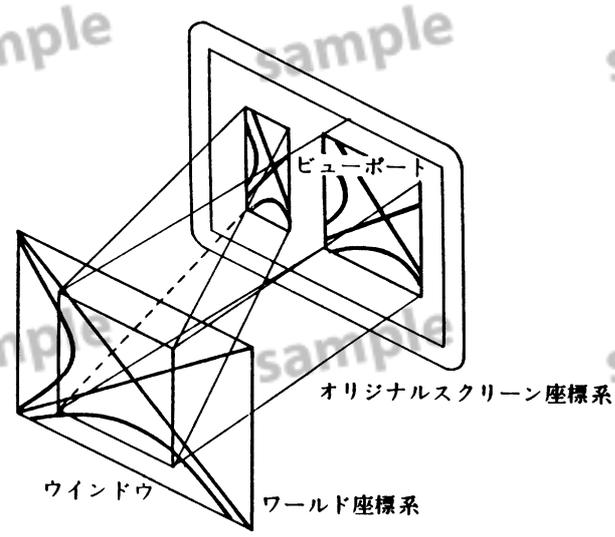


オリジナルスクリーン座標系の大きさは画面モードによって決まる。もちろん実際に表示されるか、どうかは、この画面モードとディスプレイ装置の分解能との対応が正しいものでなくてはならない(下図参照)。



・ワールド座標系の中の図形で、ウィンドウによってとりだした領域を、オリジナルスクリーン座標系に写し出す。オリジナルスクリーン座標系の中で、これから写し出される領域のことを“ビューポート”とよぶ。ビューポートを定めるためには、オリジナルスクリーン座標系で表わされる唯一の view 文を使う。

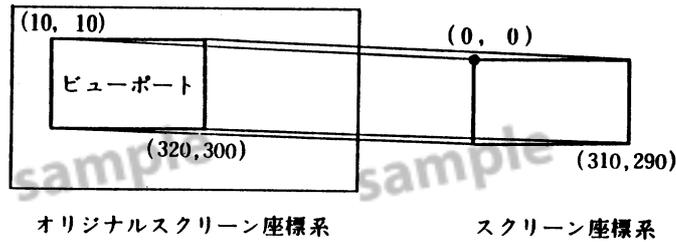
view 文の指定で、ビューポートを定めると、ビューポートの長方形の領域が、指定した領域色で塗りつぶされ、境界は指定した境界色で囲いが引かれる。



(3) スクリーン座標系

・オリジナルスクリーン座標系で表わされるディスプレイ画面の中で、これからの操作の対象になる領域がビューポートである。ビューポート以外の画面は以前の状態が持続している。この活動中の領域を、新しい座標系で見直しますと、これからの操作がしやすくなる。ビューポートの左上を基点 (0, 0) をして、ドットを単位に座標をとりなおしたのが、

スクリーン座標系である。



・スクリーン座標系は、ビューポートの操作をしやすくする。

たとえば、ビューポート内の図形をドットパターンでメモリ上の配列にセーブする場合（GET@文を使う）、この図形の領域を指定するのに、基点がビューポートの左上にあると、セーブする配列とビューポート内の領域との対応が容易につく。移動座標よりも、正規化されている方が処理しやすいだろう。

GET@文の逆のput @もスクリーン座標系で表わす。画面の指定したドットの色を知るためのPOINT関数も、この座標系を使う。

(4) 座標系の関係

ワールド座標系、オリジナルスクリーン系、スクリーン座標系の3つの座標系の関係、ウィンドウとビューポートの関係を、次にまとめておきます。

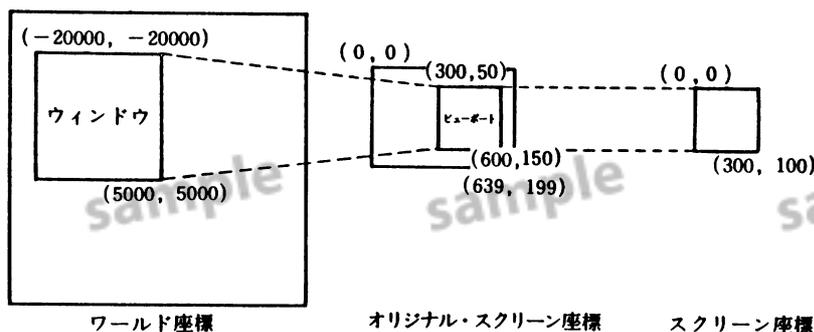
ワールド座標系で、複雑な図形をプログラミングするために、LINE文、CIRCLE文、POINT文等を使用する。

複雑な図形の中から、ディスプレイに表示する領域をウィンドウとしてとりだす。

オリジナルスクリーン座標系で、ディスプレイする画面の領域をビューポートとして定める。

ウィンドウ内の図形がビューポートに写しだされる。

ビューポート内の左上を基点としたスクリーン座標系で、ビューポート内の図形を見直す。ディスプレイ上の図形のドットイメージをメモリ上の配列に格納したり、メモリ上の配列に格納されている図形のドットイメージをディスプレイに写しだしたりする。



(5) 各座標系のもとで使用できるN88-BASIC文

詳しくは「BASICリファレンス・マニュアル」を参照せよ。

座 標 系	ワールド座標系	オリジナルスクリーン座標系	スクリーン座標系
使用できる文	CIRCLE DRAW LINE PAINT POINT PRESET PSET WINDOW	VIEW	GET@ POINT(関数) PUT@
座標の表 わし方	(Wx _i , Wy _i)	(Sx _i , Sy _i)	(Sx _i , Sy _i)

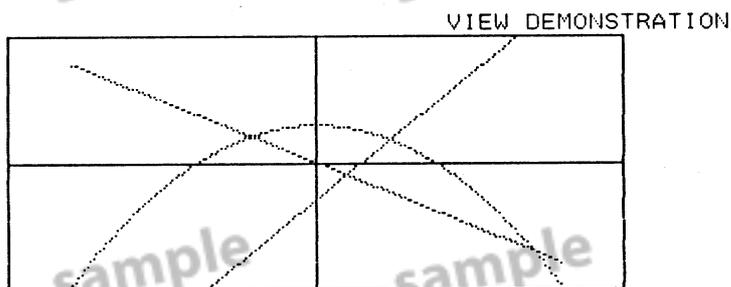
(6) ウィンドウとビューポートの初期値

screen文で画面モードを指定したときのウィンドウとビューポートはディスプレイのドット数の大きさに初期化される。

window文, view文によって変更するまで, この値が使われる。

以下に簡単な例を示す。

〔例〕



```

100 '--- VIEW SAMPLE PROGRAM ---
110 '
120 DEFINT A-Z
130 WIDTH 80,25:CONSOLE 0,25,0,1
140 SCREEN 0,0:COLOR 7,0:CLS 3
150 LOCATE 29,0:PRINT "VIEW DEMONSTRATION"
160 WINDOW(-100,-100)-(100,100)
170 '
180 VIEW(10,10)-(320,105),0,3
190 GOSUB *GRAPH
200 COPY 5
210 SCREEN 0
220 END
230 '--- GRAPH DISPLAY ROUTINE ---
240 *GRAPH
250 LINE(-100,0)-(100,0),7
260 LINE(0,-100)-(0,100),7
270 FOR X=-80 TO 80
280 PSET (X,X),1
290 PSET (X,-2*X+30),2
300 PSET (X,X*X/50-30),4
310 NEXT X
320 RETURN
    
```

不 許 複 製

慶應義塾大学ビジネス・スクール

Contents Works Inc.