ポインタと配列

1. 配列　　[復習を兼ねて]

配列とは、同種のデータ型を集めた変数であり、添え字により順序づけられている。

1.1 配列変数の宣言

配列の型宣言においては、データ型と要素数を指定する。

1次元配列の型宣言：　　データ型　配列名[要素数];

2次元配列の型宣言：　　データ型　配列名[要素数][要素数];

3次元配列の型宣言：　　データ型　配列名[要素数][要素数][要素数];

：　　：

このとき確保される変数は、添え字が0から(要素数-1) までの配列変数

例1）int dt[10];　　/* 整数型1次元配列dtの宣言 */

例2）float　a[2][4], c[2][4][5];　/* 実数型2次元配列a、実数型3次元配列cの宣言 */

1.2 配列のメモリ上の配置

例1）int　dt[10];
は、整数型のデータが10個からなる変数dt の型宣言文である。このとき確保される配列は、
dt[0] ～ dt[9] である。

注意：BASICでは、dt[0] ～ dt[10] までの11個が確保される。混同しないように！

int dt[10]; のメモリ上の配置

dt[0]
dt[1]
dt[2]
dt[3]
dt[4]
dt[5]
dt[6]
dt[7]
dt[8]
dt[9]

例２）float a[2][4]; は、配列長４の配列を２組用意した、ということになる。

float a[2][4]; の行列イメージ

a[0][0]	A[0][1]	a[0][2]	a[0][3]
a[1][0]	A[1][1]	a[1][2]	a[1][3]

float a[2][4]; のメモリ上の配置

行ごとに配置（右側の括弧[ ]内数字が先に変化するように順番づけられる）

a[0][0]	↑
a[0][1]	配列長４の配列
a[0][2]	
a[0][3]	↓
a[1][0]	↑
a[1][1]	配列長４の配列
a[1][2]	
a[1][3]	↓

注意：FORTRAN では、２次元配列 a(2,4) は列ごとに配置される（括弧内の左側の数字が先に変化するように順番づけられる）。混同しないように！

2. 変数の初期化

変数初期化の記述：

単純変数の場合は

```
データ型　変数名=定数;
```

１次元配列の場合は

```
データ型　配列名[要素数]={定数, 定数, ……, 定数};
```

２次元配列の場合は

```
データ型　配列名[要素数][要素数] = {
                {定数, 定数, ……, 定数},
                {定数, 定数, ……, 定数},
                    ……
                {定数, 定数, ……, 定数}
                };
```

特に文字型配列の場合、上記定数は文字定数（一重引用符 ' ' でくくられた１文字）に相当する。また文字列定数（二重引用符 " " でくくられた複数文字）も使用される。

なお文字列の最後には１６進コードの 00（終端コードあるいはヌルコードとも呼ぶ。特に文字コードの終端を明示するときには'¥0'を使う）を付加するという規則がある。

例１）int i, a=9999;　　/* 整数型変数の初期化 */
　　double pi=3.141592653589793;　　/* 倍長実数型変数の初期化 */

例２）int data[5]={10,11,12,13,14};　　/* 整数型１次元配列の初期化 */
　　int data[]={10,11,12,13,14};　　/* このように要素数を省略すると、上のように解釈される */

例３）float matrx[3][4] = {　　/* 実数型２次元配列の初期化 */
　　{ 1.0,　2.0,　3.0,　4.0 },
　　{ 5.0,　6.0,　7.0,　8.0 },
　　{ 9.0, 10.0,　11.0, 12.0}
　};

例４）文字列の配列（1 次元配列）
　　char str1[7] ={‘S’,‘T’,‘R’,‘I’,‘N’,‘G’,‘\0’}, /* 終端コード‘\0’を付加すること */
　　str2[ ] = “STRING“ ,　　/* 要素数を省略すると上の str1 と同じになる */
　　/* (末尾に‘\0’が付加され要素数７となる) */
　　str3[10]= “STRING“ ;　　/* str3[0]～str3[6]には str1 と同じ内容が入り、*/
　　/* str3[7]～str3[9] は空き領域となる */

例５）文字列の配列（2 次元配列）

```
char dayname[8][10]={
                    "0ther","Sunday","Monday","Tuesday",
                    "Wednesday","Thursday","Friday","Saturday"
                    };
```

上記の文字列定数を用いた配列の初期化は、下記の文字定数を用いた初期化と同じである。

```
char dayname[8][10]={
    {'0','t','h','e','r','\0'},
    {'S','u','n','d','a','y','\0'},
    {'M','o','n','d','a','y','\0'},
    {'T','u','e','s','d','a','y','\0'},
    {'W','e','d','n','e','s','d','a','y','\0'},
    {'T','h','u','r','s','d','a','y','\0'},
    {'F','r','i','d','a','y','\0'},
    {'S','a','t','u','r','d','a','y','\0'}
                    };
```

初期化された dayname[8][10]の行列格納イメージ

	0	1	2	3	4	5	6	7	8	9
0	'0'	't'	'h'	'e'	'r'	'\0'				
1	'S'	'u'	'n'	'd'	'a'	'y'	'\0'			
2	'M'	'o'	'n'	'd'	'a'	'y'	'\0'			
3	'T'	'u'	'e'	's'	'd'	'a'	'y'	'\0'		
4	'W'	'e'	'd'	'n'	'e'	's'	'd'	'a'	'y'	'\0'
5	'T'	'h'	'u'	'r'	's'	'd'	'a'	'y'	'\0'	
6	'F'	'r'	'i'	'd'	'a'	'y'	'\0'			
7	'S'	'a'	't'	'u'	'r'	'd'	'a'	'y'	'\0'	

プログラム例）数字を曜日に変え、負の数が入力されると終了する。

```
#include<stdio.h>

void main(void)
{
      char dayname[8][10]={
                              "0ther","Sunday","Monday","Tuesday",
                              "Wednesday","Thursday","Friday","Saturday"
                              };
       int day;

       while(1){
          printf("number for day of week ==> ");
          scanf("%d",&day);

          if(day<0)
              break;
           else if(day>0 && day<8)
               printf("(%d)  %s\n",day,dayname[day]);
           else
               printf("(%d)  %s\n",day,dayname[0]);
        }
}
------------------------実行開始----------------------
number for day of week ==> 6
(6)  Friday
number for day of week ==> 1
(1)  Sunday
number for day of week ==> 10
(10)  0ther
number for day of week ==> -1
------------------------おしまい----------------------
```

3. ポインタ

3.1 アドレスとポインタ

アドレスとは、変数が蓄えられているメモリ上の場所のことである。変数のアドレスを求める時、**アドレス演算子**（&）を用いて

&変数名

と書けばよい。

例)

```
#include <stdio.h>
void main(void)
{
      int a=1;

      printf("a= %d ；そのアドレスは10進数で%d（16進数で%x）\n", a, &a, &a);
}
```

```
------------------------実行開始----------------------
a= 1 ；そのアドレスは10進数で312480（16進数で4c4a0）
------------------------おしまい----------------------
```

ポインタとはアドレスを情報として持つ変数のことであり、ポインタが指し示すデータの型によって制御される。ポインタ変数の宣言は、以下のように行う。

データ型　*ポインタ変数名

例）　char *point1;　　point1はchar型を指すポインタ
　　int *point2;　　point2はint型を指すポインタ
　　float *point3;　　point3はfloat型を指すポインタ

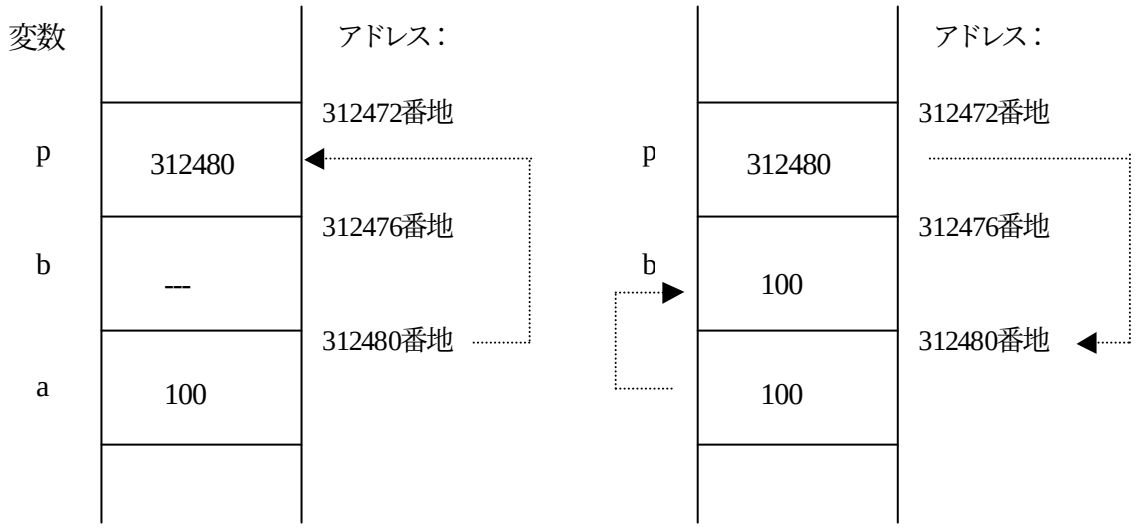
ポインタの示すアドレスに格納されているデータを表わすためには、**間接演算子**（*）を以下のように用いる。
　　*ポインタ変数名

例）アドレス演算子と間接演算子の働き

```
#include <stdio.h>
void main(void)
{
      int a=100, b;    int *p;    /* 型宣言により、変数を蓄える領域がメモリ上に確保される */

      printf("a,b,p のアドレスは、10進数表示で %d %d %d\n", &a,&b,&p);
      p=&a;
      printf("a=%d p=%d\n", a,p);
      b=*p;
      printf("a=%d b=%d\n", a,b);
}
------------------------実行開始----------------------
a,b,p のアドレスは、10進数表示で 312480 312476 312472
a=100 p=312480
a=100 b=100
------------------------おしまい----------------------
```

p = &a ； の働き　　b = *p ；の働き



3.2 ポインタと配列

配列名はその配列の先頭アドレスを示すポインタである。つまり、

```
int data[5];
int *p;
```

とデータ型宣言をして、配列 data の先頭アドレスをポインタ変数 p に代入する時

```
p = data;
```

とすればよい（ p = &data ではないことに注意 ）。ただし、data[0] のように配列要素を指定すれば、もはやポインタではないので、

```
p = &data[0];
```

は p = data; と同じことをしていることになる。

またポインタはポインタが指し示すデータの型によって制御される。例えば、

```
char str[5];
float fdt[5];
```

のように配列宣言がなされている場合、char 型のデータ長が１バイト（８ビット）、float 型のデータ長が４バイト（３２ビット）であるとき、

・str のポインタに１を加えるとその内容は char 型データ１個分（1 バイト）増える。
・fdt のポインタに１を加えるとその内容は float 型データ１個分（４バイト）増える。

例）上記の事柄を確認する。

```
#include <stdio.h>

void main(void)
{
    int i;
    char  str[]="Good", *ps;
    float fdt[5]={0.1, 0.2, 0.3, 0.4, 0.5}, *pf;

    ps=str;
    printf("文字列 str= %s ；その先頭アドレスは%06x\n",str,ps);
    for(i=0; i<5; i++){
    printf("str[%d]=%c ; そのアドレスは%06x  ポインタ演算でも%06x\n", i,str[i],&str[i],ps);
    ps++;                                   /* str のポインタ ps に１を加える */
    }

    pf=fdt;
    printf("\n配列 ddt の先頭アドレスは%06x\n",fdt);
    for(i=0; i<5; i++){
    printf("fdt[%d]=%5.2f ；そのアドレスは%06x  ポインタ演算でも%06x\n", i,fdt[i],&fdt[i],pf);
    pf++;                                   /* fdt のポインタ pf に１を加える */
    }

    printf("\n以下のことを確認しましたか？\n");
    printf("ポインタを１加算 --> その内容はポインタが指し示すデータ１個分増加\n");
}
```

----------実行開始----------
文字列 str= Good ；その先頭アドレスは 04c69c
str[0]=G；そのアドレスは 04c69c　ポインタ演算でも 04c69c
str[1]=o；そのアドレスは 04c69d　ポインタ演算でも 04c69d
str[2]=o；そのアドレスは 04c69e　ポインタ演算でも 04c69e
str[3]=d；そのアドレスは 04c69f　ポインタ演算でも 04c69f
str[4]= ；そのアドレスは 04c6a0　ポインタ演算でも 04c6a0

配列 ddt の先頭アドレスは 04c688
fdt[0]= 0.10 ；そのアドレスは 04c688　ポインタ演算でも 04c688
fdt[1]= 0.20 ；そのアドレスは 04c68c　ポインタ演算でも 04c68c
fdt[2]= 0.30 ；そのアドレスは 04c690　ポインタ演算でも 04c690
fdt[3]= 0.40 ；そのアドレスは 04c694　ポインタ演算でも 04c694
fdt[4]= 0.50 ；そのアドレスは 04c698　ポインタ演算でも 04c698

以下のことを確認しましたか？
ポインタを１加算 --> その内容はポインタが指し示すデータ１個分増加
----------おしまい----------

文字列配列 str

要素	内容	アドレス：配列の添え字が１増える毎にアドレスは１バイトづつ増加
str[0]	‘G’	04c69c
str[1]	‘o’	04c69d
str[2]	‘o’	04c69e
str[3]	‘d’	04c69f
str[4]	‘¥0’	04c6a0

実数型配列 fdt

要素	内容	アドレス：配列の添え字が１増える毎にアドレスは４バイトづつ増加
fdt[0]	0.10	04c688
fdt[1]	0.20	04c68c
fdt[2]	0.30	04c690
fdt[3]	0.40	04c694
fdt[4]	0.50	04c698

問）次のプログラムを実行させたところ、

```
------------------------実行開始----------------------
1) short int--> 04c498    double--> 04c470
2) short int--> ******    double--> @@@@@@
------------------------おしまい----------------------
```

と出力された。**** と @@@@@@ にはどのような１６進数が出力されたか？**

```
#include <stdio.h>
void main(void)
{
      short int data[5], *ps;
      double ddata[5], *pd;

      ps=data;
      pd=ddata;
      printf("1) short int--> %06x    double--> %06x \n",ps,pd);
      ps++;
      pd++;
      printf("2) short int--> %06x    double--> %06x \n",ps,pd);
}
```