C 言語 016 配列とポインタ

【配列とポインタ】

実は配列はポインタです。配列は「複数のアドレスを格納している物」になります。

```
int array[] = {1, 2, 3};
```

配列 array の先頭アドレスを仮に 100 番地、int 型を 4 バイトとすると、メモリの中身は下記の通り。

配列 array

添字	0	1	2
値	100 番地	104 番地	108 番地

実データ

アドレス	100 番地	104 番地	108 番地
値	1	2	3

配列は先頭アドレスを格納しています。

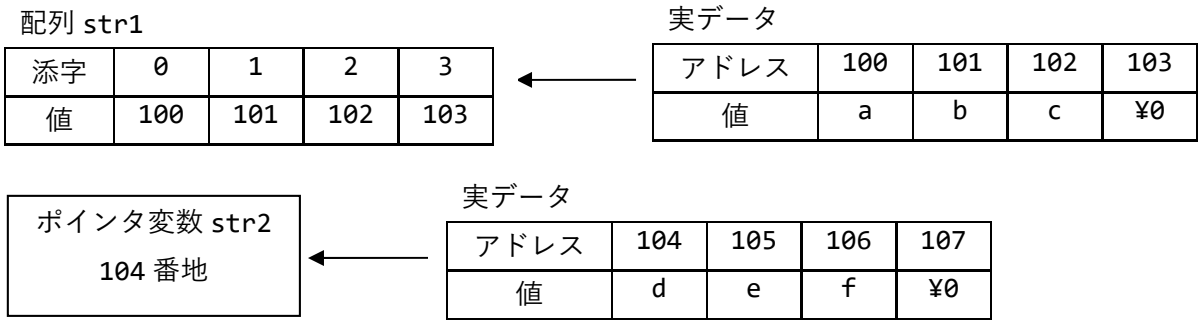
```
int array[] = {1, 2, 3};
int* p = array;      /* p には 100 番地が代入。配列はポインタなので「&」が不要 */
printf("%d", *p);    /* 100 番地の値を参照する。「1」が出力 */
```

配列への値の代入は下記のようになります。

```
int array[] = {1, 2, 3};
array[0] = 10;              /* 100 番地に 10 を代入 */
printf("%d", array[0]);     /* 100 番地の値を参照。「10」と出力 */
array[0] = array[1];        /* array[0]に「104 番地」を代入 */
printf("%d", array[0]);     /* 104 番地の値を参照。「2」と出力 */
```

＜文字配列とポインタ＞

```
char  str1[] = "abc";
char* str2 = "def";
printf("%s¥n", str1);    /* 「abc」と出力 */
printf("%s¥n", str2);    /* 「def」と出力 */
```



文字列は「先頭アドレスから NULL 文字まで」の領域を指します。
配列 str1 は 100 番地から 103 番地まで、ポインタ変数 str2 は 104 番地から 107 番地までが文字列になります。
文字列関連の関数で引数が「char*」なのは、この仕組みを利用しています。

＜同アドレス参照に注意＞
配列（文字配列）はポインタなので下記のような現象を発生します。

```
char str1[] = "abc";     /* str1 の先頭アドレスを 100 番地とする */
char* str2 = str1;       /* str2 に 100 番地が代入される */
strcpy(str1, "def");     /* str1（100 番地）の文字列に「def」 */
printf("%s¥n", str1);    /* str1（100 番地）は「def」 */
printf("%s¥n", str2);    /* str2（100 番地）は「def」 */
```

str1 も str2 も同じアドレスを参照しているため、同じ出力結果になってしまう。

【ポインタ変数のインクリメント】

下記のソースコードの実行結果は「12345」と出力されます。

```
int  array[] = {1, 2, 3, 4, 5};    /* array の先頭アドレスは 100 番地 */
int* p = array;                    /* ポインタ変数 p に 100 番地を代入 */
int  i;
for (i = 0; i < 5; i++) {
    printf("%d¥n", *p);            /* アドレスの参照先を出力する */
    p++;                           /* ポインタ変数をインクリメントする */
}
```

配列 array の先頭アドレスを仮に 100 番地、int 型を 4 バイトとすると、メモリの中身は下記の通り。

添字	0	1	2	3	4
アドレス	100	104	108	112	116
値	1	2	3	4	5

ポインタ変数 p は int 型。int 型は 4 バイト。ポインタ変数をインクリメント、デクリメントするとデータ型のバイト分、アドレスを増減させます。

ポインタ変数 p の中身

ループ回数	1	2	3	4	5
値	100	104	108	112	116

よって「12345」が出力されるのです。

【配列を関数の引数にする】

```
int  var;
char str[10];
scanf("%d", &var);
scanf("%s", str);
```

「scanf 関数」の引数は「書式指定子」と「アドレス」です。
配列はもともとアドレスを格納しているので「&」は不要です。引数はアドレスなので「参照渡し」になります。

【配列を関数の戻り値にする】

配列は関数の戻り値にはできません。

関数の戻り値を「アドレス」にすれば同様の処理が行えますが、ローカル変数による「空き番地」問題があるので、やはり「参照渡し」で代用するのが一般的です。

```
int main(void) {
    int* func(void);
    int* p = func();       /* 100 番地を取得する */
    printf("%d¥n", *p);    /* 100 番地の値を出力する */
    p++;                   /* ポインタ変数のインクリメント。104 番地になる */
    printf("%d¥n", *p);    /* 104 番地の値を出力する */
    p++;                   /* 108 番地になる */
    printf("%d¥n", *p);    /* 108 番地の値を出力する */
    return 0;
}

int* func(void) {
    static int array[] = {10, 20, 30};    /* static にして領域を保持する */
    return array;                         /* array のアドレスを仮に 100 番地として返す */
}
```

＜実行結果＞

```
10
20
30
```

配列 array

添字	0	1	2
アドレス	100	104	108
値	10	20	30

【配列とポインタ変数のサイズの違い】

下記のソースコードでは x も y も同じアドレスを記録しています。よって x[0]も y[0]も値は「1」になります。

しかし、配列とポインタ変数で異なるのが sizeof で取得できるサイズです。

配列 x の場合は int 型（4 バイト）が 5 個なので 20 バイトになりますが、ポインタ変数 y はアドレス 1 個分（32bit 環境は 4 バイト、64bit 環境は 8 バイト）になっています。

```
int x[] = {1, 2, 3, 4, 5};
int* y = x;
printf("%p¥n", x);            /* x と y は同じアドレスを格納している */
printf("%p¥n", y);
printf("%d¥n", x[0]);         /* 「1」と出力 */
printf("%d¥n", y[0]);         /* 「1」と出力*/
printf("%d¥n", sizeof(x));    /* 「20」と出力*/
printf("%d¥n", sizeof(y));    /* 「4」と出力*/
```

＜配列を引数にした場合の注意点＞

配列を引数にする場合はサイズの取得に注意してください。

下記のソースコードでは、実引数の x は配列なので 20 バイトですが、仮引数 y はポインタ変数扱いになるので 4 バイトになります。

func 関数側では配列の要素数を取得することはできないので、要素数を取得したい場合は別な引数にして渡すことになります。

```
int main(void) {
    void func(int[]);
    int x[] = {1, 2, 3, 4, 5};
    printf("%d¥n", sizeof(x));    /* 「20」と出力 */
    func(x);
    return 0;
}

void func(int y[]) {
    printf("%d¥n", sizeof(y));    /* 「4」と出力*/
}
```

【ポインタ変数の配列】

```
int array[5];       /* この配列は int 型の値を入れる物 */
int var = 10;
array[0] = &var;    /* アドレスを代入すると警告になる */
```

確かに、配列はアドレスで管理してはいますが、int 型の配列はあくまでも int 型の値を記録する物であって、アドレスを記録する物ではありません。
そこで、ポインタ変数を配列化してみます。

```
int  array[5];    /* int 型の値を 5 個格納できる配列 */
int* array[5];    /* int 型のアドレスを 5 個格納できる配列 */
```

＜例文 1＞

```
int  var = 10;          /* 仮にこの変数のアドレスを 100 番地とする */
int* p[5];              /* int 型のアドレスを 5 個格納できる配列 */
p[0] = &var;            /* p の 0 番目に 100 番地を代入 */
printf("%d", *p[0]);    /* 100 番地の値を出力する */
```

変数名：var
アドレス：100 番地
値：10

配列 p

添字	0	1	2	3	4
値	100 番地				

＜例文 2＞

```
/* 文字配列 */
char str1[] = "abc";
char str2[] = "xyz";
char str3[] = "いろは";

/* ポインタ変数の配列 */
char* list[] = {str1, str2, str3};

/* 出力してみると */
printf("%s¥n", list[0]);
printf("%s¥n", list[1]);
printf("%s¥n", list[2]);
```

＜実行結果＞

abc

xyz

いろは

＜例文 2 の解説＞

4 つの配列を表にしてみます。

char 型は 1 バイト、最初のアドレスは仮に 100 番地とします。

配列 str1

添字	0	1	2	3
アドレス	100	101	102	103
値	a	b	c	¥0

配列 str2

添字	0	1	2	3
アドレス	104	105	106	107
値	x	y	z	¥0

配列 str3

添字	0		2		4		6
アドレス	108	109	110	111	112	113	114
値	い		ろ		は		¥0

配列 list

添字	0	1	2
アドレス	115	116	117
値	100 番地	104 番地	108 番地

文字列は「先頭アドレスから NULL 文字まで」なので

printf("%s¥n", list[0]);

で、100 番地の値から¥0 までの文字列、つまり「abc」が出力されます。

int main(int argc, char* argv[]) {

メイン関数の引数「cahr* argv[] 」はこの仕組みになっています。