

ポインタ

三池 克明

ポインタとは変数の一種ではありますが、変数は値を書き込むメモ用紙であるのに対し、ポインタは他の変数の場所をメモすることができます。
これだけではわかりにくいと思いますが、まずは先に進んでみましょう。

—目 次—

1.	ポインタとは.....	1
1.1.	ポインタと変数.....	1
1.2.	ポインタと配列.....	6
2.	ポインタを使ったプログラム.....	14
2.1.	double 型配列の各数値を 10 倍する.....	14
2.2.	文字数を調べる.....	16
3.	演習問題.....	17

1. ポインタとは

1.1. ポインタと変数

まずは以下のソースプログラムを作成し、コンパイル・実行させてみましょう。
初めてみる記述がいくつかありますが、後で説明しますので今は深く考えないで下さい。

pointer1.cpp

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int    a, b;
6:     int    *p;
7:
8:     a = 5;
9:     b = 3;
10:
11:     p = &a;
12:     *p = 100;
13:     p = &b;
14:     *p = 200;
15:
16:     printf("a = %d, b = %d\n", a , b);
17: }
```

実行例

a = 100, b = 200

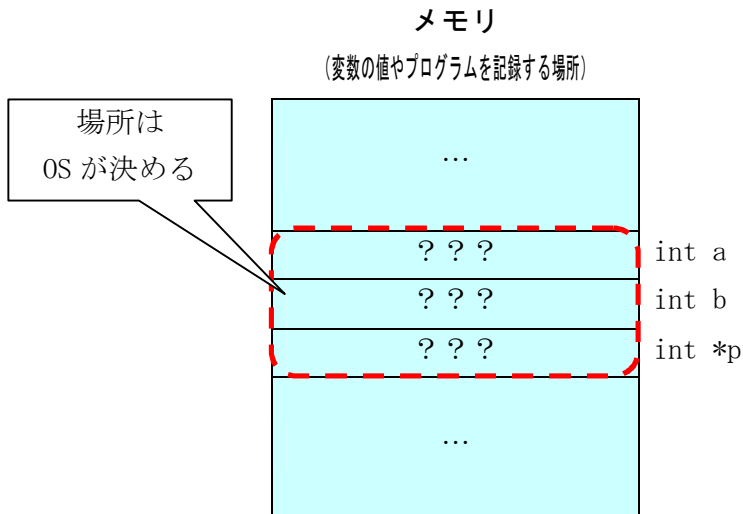
続いてプログラムの動きを追ってみましょう。

```
5:     int    a, b;
6:     int    *p;
```

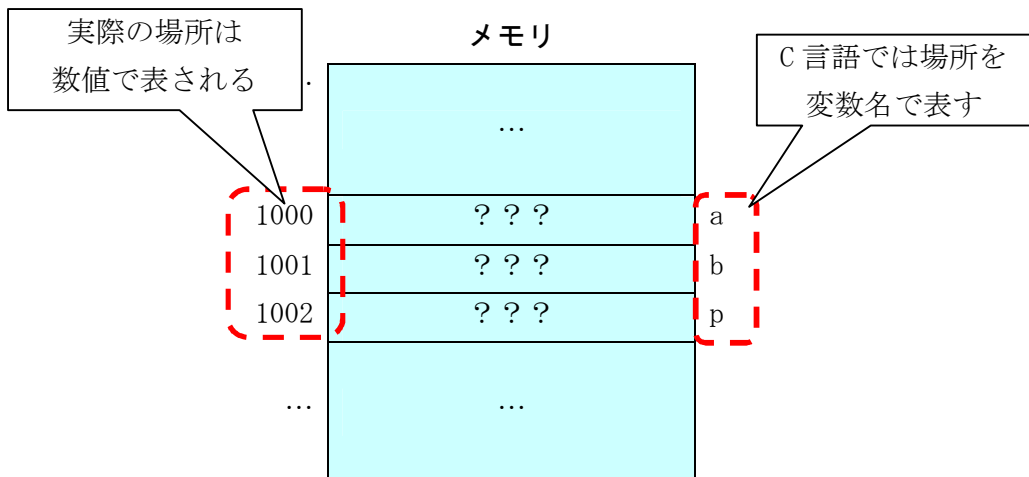
5行目は意味がわかると思います。
問題は6行目ですね。

これは「int 型へのポインタ」p を宣言しています。単に「int 型ポインタ」p と呼ぶこともあります。

また、コンピュータのメモリ内部は大抵は下図のように確保されます。



各変数がメモリのどの場所に確保されるかは OS が決めるのですが、ここでは説明のために下図の場所を確保したことにします。

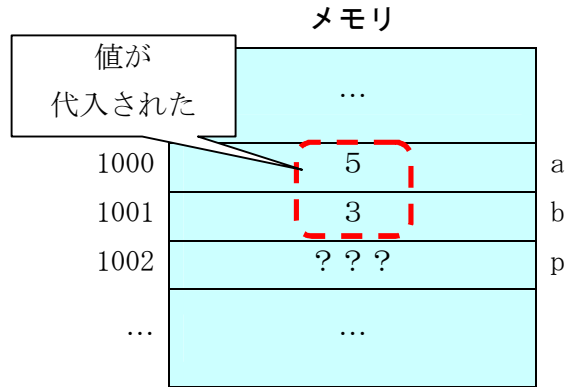


それでは次のステップに進みましょう。

```
8:      a = 5;
9:      b = 3;
```

これも意味はわかると思います。

この処理を行うとメモリは下図の状態になります。



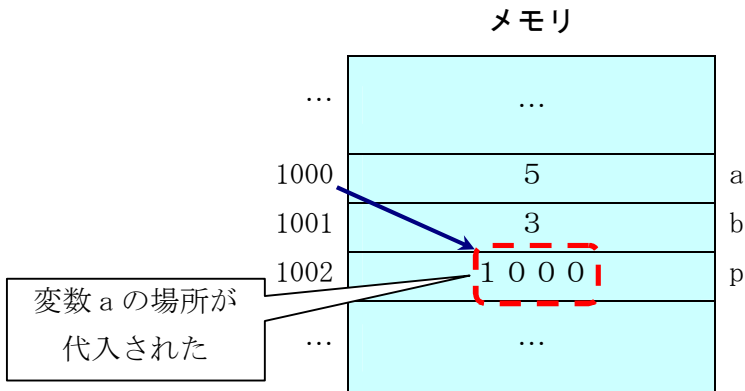
続いて 11 行目です。

```
11:     p = &a;
```

これは「変数 a の場所を p に代入する」という意味です。

“&[変数名]” という書式でその変数の場所を表します。

この処理を行うとメモリは下図の状態になります。



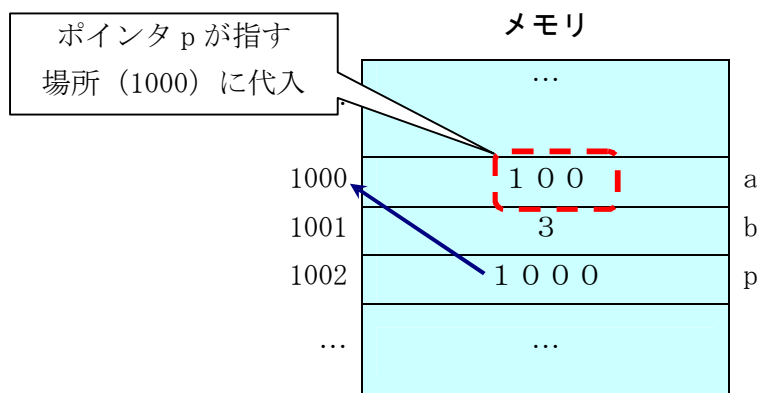
続いて 12 行目です。

```
12:      *p = 100;
```

これは「ポインタ p に書き込まれている場所に 100 を代入する」という意味です。

“*[ポインタ名]” という書式でそのポインタが指す場所を表します。

この処理を行うとメモリは下図の状態になります。

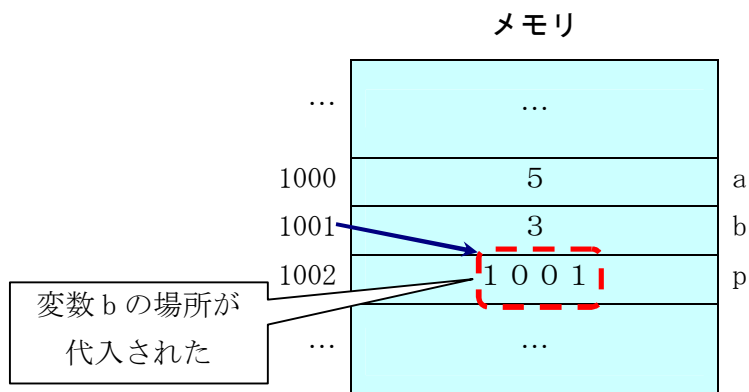


続いて 13 行目です。

```
13:      p = &b;
```

これは 11 行目と同じ考え方です。

つまり「変数 b の場所を p に代入する」になり、メモリは下図の状態になります。

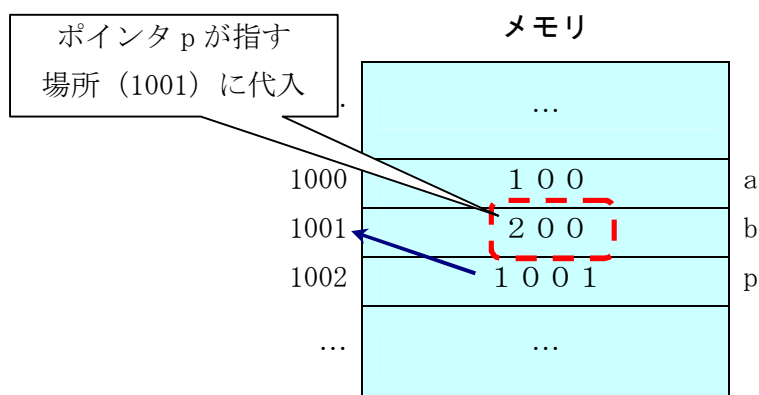


続いて 14 行目です。

```
14:      *p = 200;
```

これは 12 行目と同じですね。よって意味も「ポインタ p に書き込まれている場所に 100 を代入する」と同じです。

ただしポインタ p の内容は変数 b の場所を指していますので、処理の結果は下図のようになります。



このようにポインタは他の変数の場所をメモし、その指し示す場所の値などを操作することができます。

—ポインタのおさらい—

- ポインタの宣言は名前の前に “*” を付ける

```
int      *a;  
char     *c;
```

- ポインタに変数の場所を代入するときは、その変数名に “&” を付ける。

```
c = &moji;
```

- ポインタが指す場所の値を操作するときはポインタ名に “*” を付ける。

```
*c = 'm';
```

1.2. ポインタと配列

まずは以下のソースプログラムを入力し、コンパイル・実行して結果を確かめてみましょう。

pointer2.cpp

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int    array[] = {2, 4, 6, 1, 3};
6:     int    *p;
7:
8:     p = array;
9:     printf("%d\n", *p);
10:    p++;
11:    printf("%d\n", *p);
12:    p++;
13:    printf("%d\n", *p);
14:    p++;
15:    printf("%d\n", *p);
16:    p++;
17:    printf("%d\n", *p);
18: }
```

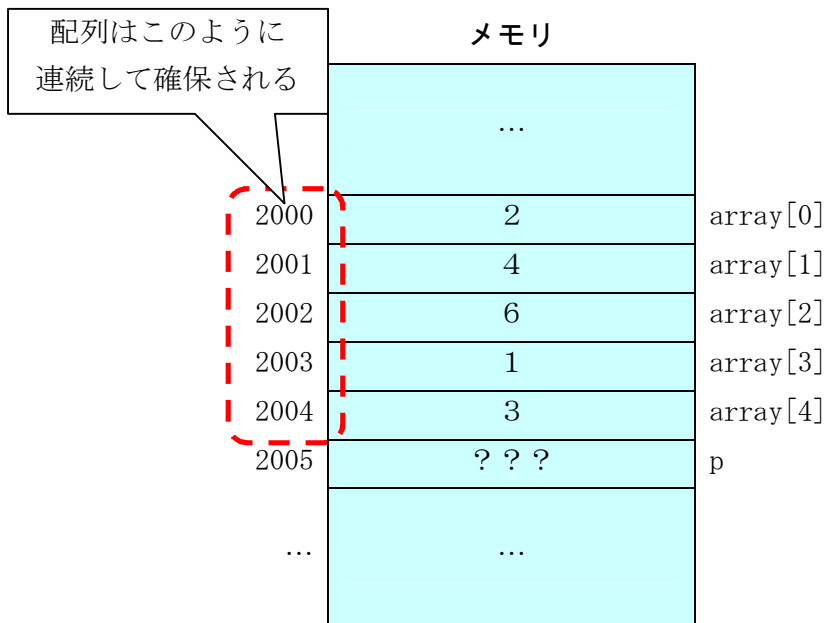
実行例

```
2
4
6
1
3
```

では、プログラムの動きを追ってみましょう。

```
5:      int    array[] = {2, 4, 6, 1, 3};
6:      int    *p;
```

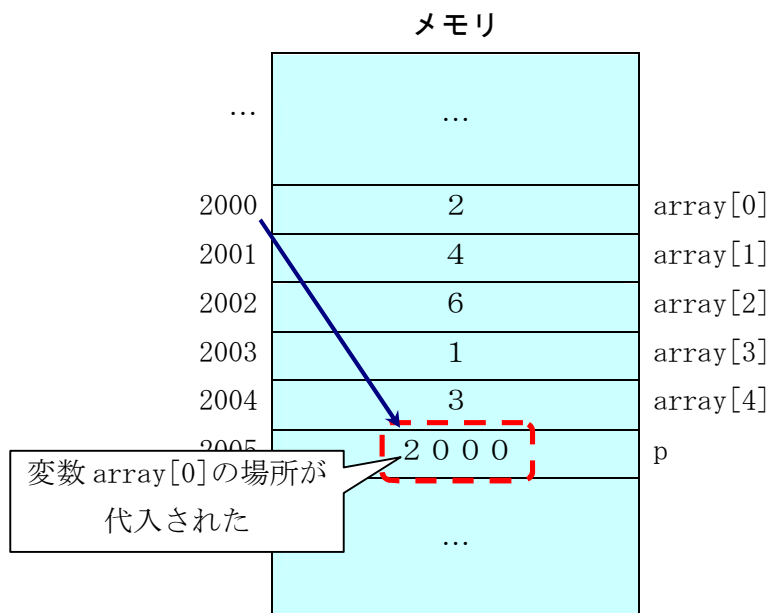
この処理によりメモリが確保されます。
なお場所を表す数値はあくまで一例です。



次に8行目です。

```
8:      p = array;
```

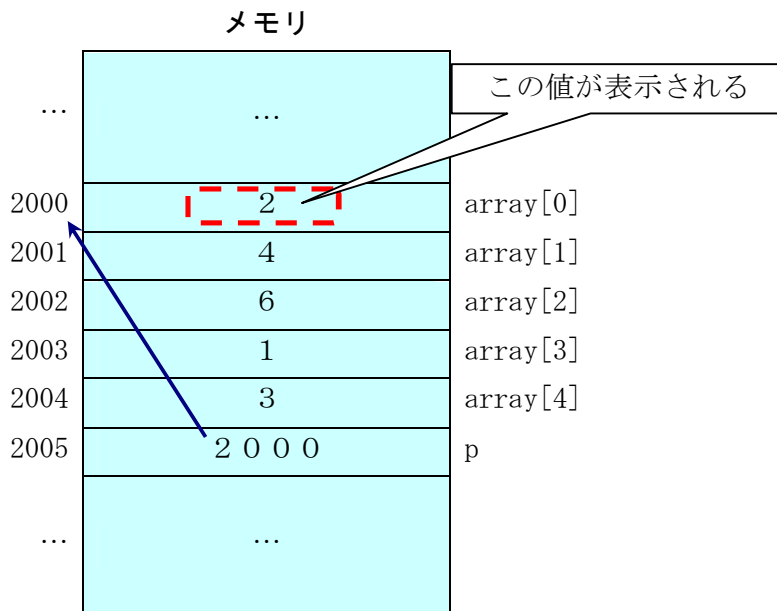
この場合の array は &array[0] と同じです。
よって array[0] の場所が p に代入されます。



続いて9行目です。

```
9:      printf("%d\n", *p);
```

これはポインタ p が指し示す場所、つまり array[0] の値を表示します。
これは分かりますよね？



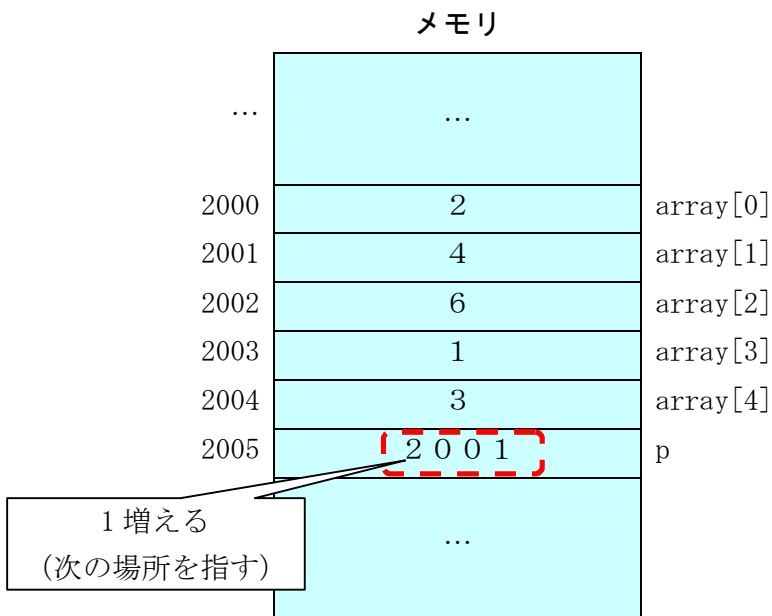
続いて 10 行目です。

```
10:      p++;
```

ポインタ p の値（場所）が 1 増えます。

これによりポインタ p は次の場所である array[1] を指し示します。

これは分かりますよね？



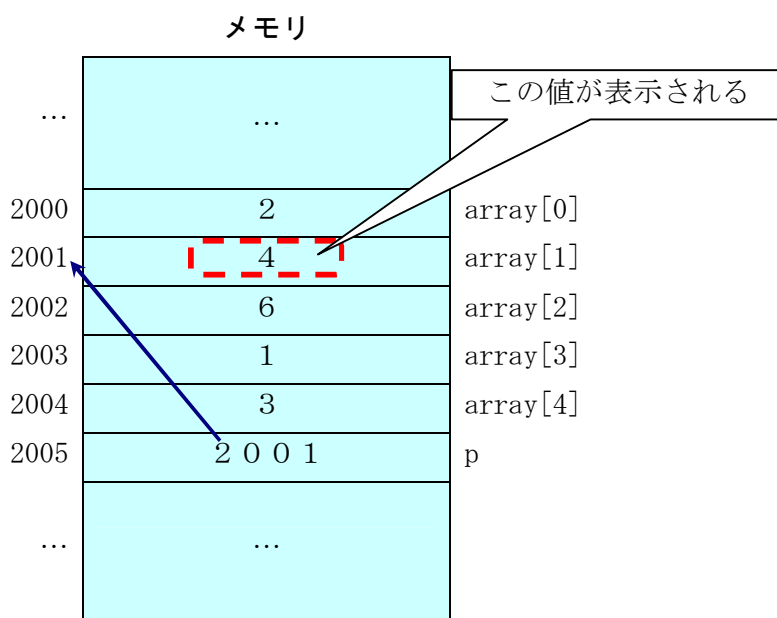
続いて 11 行目です。

```
11:      printf("%d\n", *p);
```

これは 9 行目同様ポインタ p が指し示す場所の値を表示します。

つまり array[1] の値を表示します。

分かりますよね？



この先の処理は 10~11 行目の繰り返しなので説明の必要はないと思います。

先ほどの pointer2.cpp を for 文を使ってコンパクトにまとめました
このソースプログラムを入力し、コンパイル・実行して結果を確かめてみましょ
う。

pointer3.cpp

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int    array[] = {2, 4, 6, 1, 3};
6:     int    *p, i;
7:
8:     p = array;
9:     for(i = 0; i < 5; i++){
10:         printf("%d¥n", *p);
11:         p++;
12:     }
13: }
```

実行例

```
2
4
6
1
3
```

さらに改造したのが以下の pointer4. cpp です
このソースプログラムを入力し、コンパイル・実行して結果を確かめてみましょう。

pointer4. cpp

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int    array[] = {2, 4, 6, 1, 3};
6:     int    *p, i;
7:
8:     p = array;
9:     for(i = 0; i < 5; i++){
10:         printf("%d¥n", *(p + i));
11:     }
12: }
```

実行例

```
2
4
6
1
3
```

以上のようにポインタの値を加減すればその前後の値を得ることができます。
またポインタそのものを加減せず $*(p + i)$ のように、加減算の式で表すことも可能です。

—ポインタのおさらい 2—

- ポインタは加減算することで指し示す先をずらすことができる
 - $*(p + 4)$ … ポインタ p が指し示す場所から 4 つ先の値
 - $p = p - 3$ … ポインタ p の指し示す場所を 3 つだけ前に戻す。

2. ポインタを使ったプログラム

2.1. double 型配列の各数値を 10 倍する

これは「array7.cpp」のプログラムをポインタを使って処理するように改造したプログラムです。

このソースプログラムを入力し、コンパイル・実行してみましょう。

pointer5.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: main()
6: {
7:     double  a[SIZE], b[SIZE], *pa, *pb;
8:     int     i;
9:
10:    printf("%d 個の実数を入力して下さい。¥n", SIZE);
11:    for(pa = a, i = 0; i < SIZE; i++, pa++){
12:        printf("a[%d] = ", i);
13:        scanf("%lf", pa);
14:    }
15:
16:    for(pa = a, pb = b, i = 0; i < SIZE; i++, pa++, pb++){
17:        *pb = *pa * 10.0;
18:    }
19:
20:    printf("a を 10 倍した b は以下のとおりです。¥n");
21:    for(pb = b, i = 0; i < SIZE; i++){
22:        printf("b[%d] = %g¥n", i, *(pb + i));
23:    }
24: }
```

5 個の実数を入力して下さい。

a[0] = 3.2

a[1] = 45.1

a[2] = -2.5

a[3] = -0.42

a[4] = 3.14

a を 10 倍した b は以下のとおりです。

b[0] = 32

b[1] = 451

b[2] = -25

b[3] = -4.2

b[4] = 31.4

ポインタを使って配列を操作するとき、

1. ポインタそのものをインクリメントして参照する

```
p++;
```

```
*p = ~;
```

2. ポインタにインクリメントした値を足して参照する

```
i++;
```

```
*(p + i) = ~;
```

の 2 種類があります。

どちらも一長一短ありますので、どちらを使えばよいのかは一概に言えません。

ですので状況に応じて、どちらがよりわかりやすいプログラムになるかを考えて選べばよいでしょう。

2.2. 文字数を調べる

これは「array8.cpp」をポインタを使って処理するように改造したプログラムです。

このソースプログラムを入力し、コンパイル・実行してみましょう。

pointer6.cpp

```
1:  /*文字数を調べる*/
2:
3:  #include <stdio.h>
4:
5:  #define SIZE    128
6:
7:  main()
8:  {
9:      char    a[SIZE], *p;
10:     int     len;
11:
12:     printf("文字列 (最大%d 文字) を入力して下さい。:", SIZE - 1);
13:     scanf("%s", a);
14:
15:     for(p = a, len = 0; *p != '\0'; len++, p++) {
16:     }
17:
18:     printf("入力した文字列:%s、文字数:%d\n", a, len);
19: }
```

実行例

文字列 (最大 127 文字) を入力して下さい。:abc123

入力した文字列:abc123、文字数:6

3. 演習問題

- 12-1. 「pointer5.cpp」を改造して、各数値を半分にするプログラム「ensyu12-1.cpp」を作りなさい。
- 12-2. 「ensyu12-1.cpp」を改造して、奇数なら2倍、偶数なら半分にするプログラム「ensyu12-3.cpp」を作りなさい。なおそのデータは整数であることを前提とする。

ヒント：

奇数・偶数は整数であることが前提の話です。

ですから double 型の変数やポインタは int 型に修正しておきましょう。