

プログラミングII

第1回: ポインタ

2018年11月14日(水)

筑波大学 情報学群 情報メディア創成学類

三河 正彦

- 科目のページ

お知らせやテキストは, [manaba](#) (学習管理システム) で公開

- シラバス

<http://www.mast.tsukuba.ac.jp/lecture/syllabus/GC12301.html>

シラバス

- 第 1～5 週

- ♦ ポインタの基礎、ポインタと配列、ポインタと関数の受渡し、動的メモリ、ポインタと構造体、リンクトリスト(ポインタの応用)

- 第 6～10 週

- ♦ 構造体の復習他、リンクトリストの復習と探索他、リンクトリストとファイルの復習他、ポインタの復習他

今日やること

- プログラミングⅠの復習
- ポインタの基礎
- ポインタと関数

小テスト (プログラミング I の復習)

- 1) 0から整数 n の和を算出する関数`sum()`を作成せよ.
- 2) `for`文を使う.
- 3) `main`関数から関数`sum()`を呼び出し, 結果`ans`を表示せよ.

ポインタ

```
1:  /* pointer.c */
2:  #include <stdio.h>
3:
4:  int main(void)
5:  {
6:      int a;          /* int型変数 */
7:      int *b;         /* int型へのポインタ */
8:
9:      a = 3;
10:     printf("a の値\n");
11:     printf("a のアドレス\n");
12:     printf("\n");
13:
14:     b = &a;          /* aのアドレスをbに代入 */
15:     printf("b が指している場所の中身: %d\n", *b);
16:     printf("b の値\n");
17:     printf("\n");
18:
19:     /* b が指している場所の中身を5 に変更 */
20:     *b = 5;
21:
22:     printf("a の値\n");
23:     printf("b が指している場所の中身: %d\n", *b);
24: }
```

```
# cc pointer.c
# ./a.out
a の値          : 3
a のアドレス    : 0x7fff5fbff84c

b が指している場所の中身: 3
b の値          : 0x7fff5fbff84c

a の値          : 5
b が指している場所の中身: 5
#
```

アドレス演算子

```
: %d\n", a);
: %p\n", &a);
```

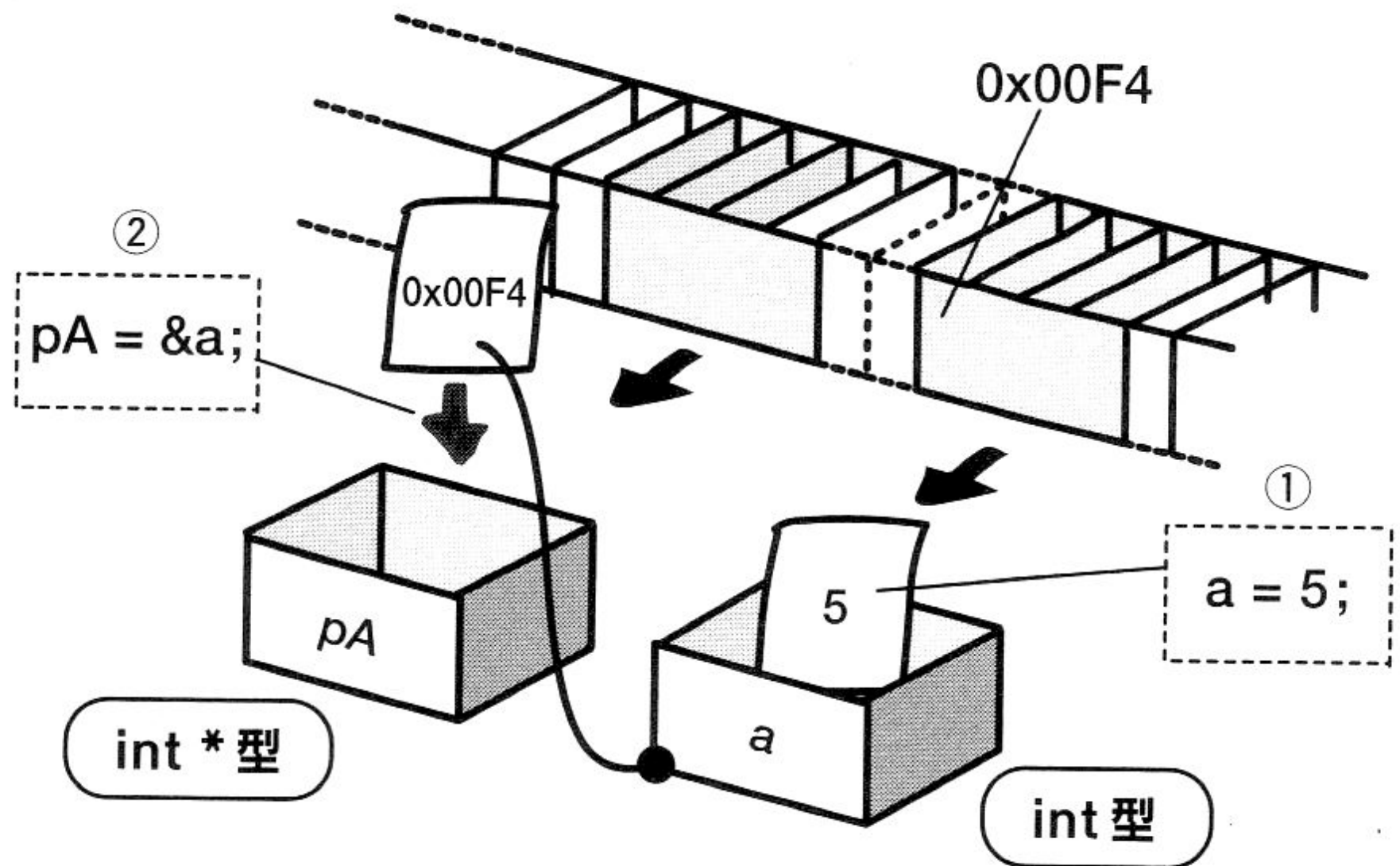
アドレスの表示

演算子

& : アドレス演算子

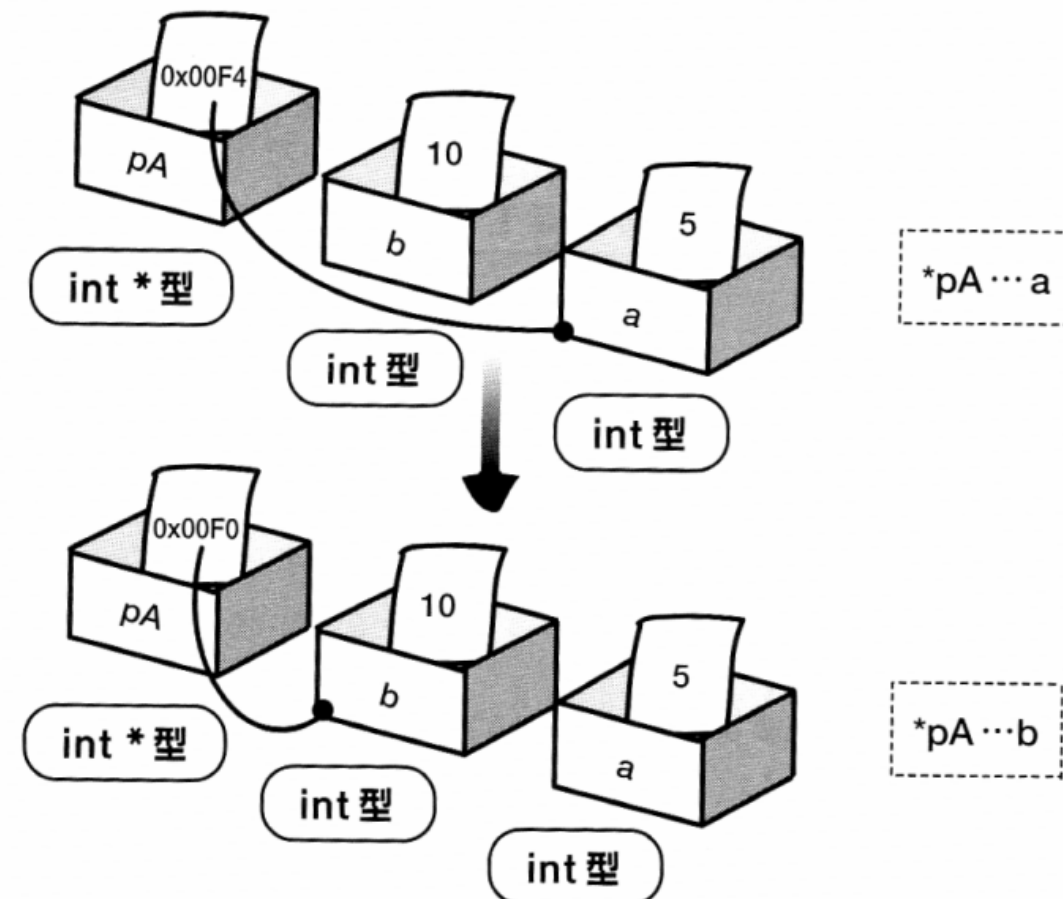
* : 間接参照演算子

```
int a;  
int *pA;  
pA = &a;
```



ポインタの型

- ポインタは、ある型と結び付けられている変数のアドレスを格納する。
 - ✦ 型によってメモリ上で占める大きさが違うので「int型変数へのポインタ」と表現することで、そのポインタが指している変数の(先頭)アドレスと、その変数に必要な大きさが分かる。
- ポインタには、指定した型の値のアドレスしか格納できない。



ポインタによる間接参照とは

```
#include <stdio.h>
```

```
int main(void)
{
```

```
    int a;
    int *pA;
```

```
    a = 5;
    pA = &a;
```

```
    printf("変数aの値は%dです。\\n", a);
```

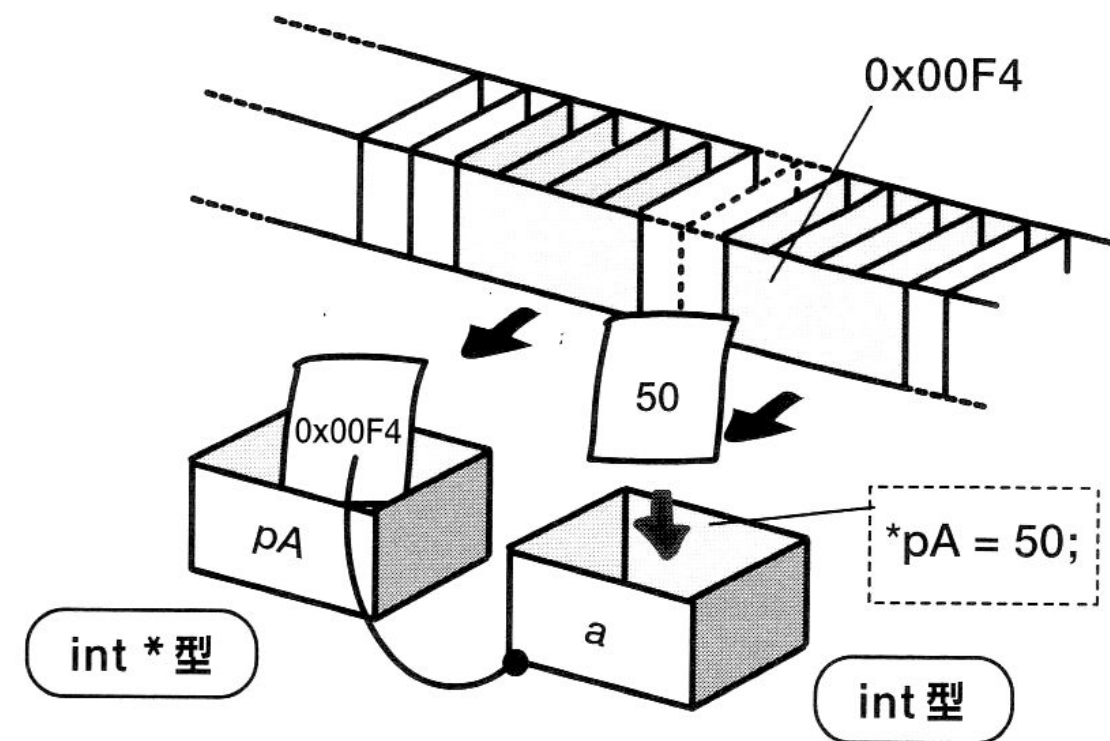
```
    *pA = 50; ● ————— *pA、つまり変数aに値を代入しました
```

```
    printf("*pAに50を代入しました。\\n");
    printf("変数aの値は%dです。\\n", a);
```

```
    return 0;
```

```
    ● ————— 変数aの値を出力してみると...
```

```
}
```



- 教科書 p.286
- aに対する操作を, pAを用いて行う.

関数とポインタ

```
/* add_pointer.c */
#include <stdio.h>

void add(int x, int y, int *z);

int main(void)
{
    int a = 15, b = 25;
    int ans;

    add(a, b, &ans);

    printf("add (%d, %d) in main() => %d\n",
           a, b, ans);
}

void add(int x, int y, int *z)
{
    *z = x + y;
    printf("add (%d, %d) in add() => %d\n",
           x, y, *z);
}
```

```
# cc add_pointer.c
# ./a.out
add (15, 25) in add() => 40
add (15, 25) in main() => 40
#
```

```
/* add_pointer_ng.c */
#include <stdio.h>

void add(int x, int y, int z);

int main(void)
{
    int a = 15, b = 25;
    int ans;

    add(a, b, ans);

    printf("add (%d, %d) in main() => %d\n",
           a, b, ans);
}

void add(int x, int y, int z)
{
    z = x + y;
    printf("add (%d, %d) in add() => %d\n",
           x, y, z);
}
```

```
# cc add_pointer_ng.c
# ./a.out
add (15, 25) in add() => 40
add (15, 25) in main() => 0
#
```

関数の戻り値

```
1 /* add_return.c */
2 #include <stdio.h>
3
4 int add(int x, int y);
5
6 int main(void)
7 {
8     int a = 15, b = 25;
9     int ans;
10
11     ans = add(a, b);
12
13     printf("add (%d, %d) => %d\n", a, b, ans);
14 }
15
16 int add(int x, int y)
17 {
18     return(x + y);
19 }
```

swap.cの作成

```
1:  /* swap.c */
2:  #include <stdio.h>
3:
4:  void swap(int ??, int ??);
5:
6:  int main(void)
7:  {
8:      int a = 5;
9:      int b = 10;
10:
11:      printf("a = %d\n", a);
12:      printf("b = %d\n", b);
13:
14:      swap(??, ??);
15:
16:      printf("a = %d\n", a);
17:      printf("b = %d\n", b);
18:  }
19:
20: void swap(int ??, int ??)
21: {
22:     /* ここは自分で考える */
23: }
```

swap.cの一例

```
/* swap.c */
#include <stdio.h>

void swap(int *x, int *y);

int main(void)
{
    int a = 5;
    int b = 10;

    printf("a = %d\n", a);
    printf("b = %d\n", b);

    swap(&a, &b);

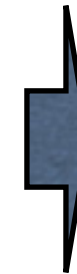
    printf("a = %d\n", a);
    printf("b = %d\n", b);
}

void swap(int *x, int *y)
{
    int tmp;

    tmp = *x;    /* xの値をtmpに一旦格納 */
    *x = *y;     /* yの値をxに代入 */
    *y = tmp;    /* tmp(元のx)の値をyに代入 */
}
```

```
void swap(int *x, int *y)
{
    int *tmp;

    *tmp = *x;
    *x = *y;
    *y = *tmp;
}
```



実行エラー

```
void swap(int *x, int *y)
{
    int *tmp;

    tmp = *x;
    *x = *y;
    *y = tmp;
}
```



コンパイルエラー

tmpはアドレスの
格納専用

sort.cの作成

- swap.cを改造し, 2つの整数型変数を大きい順に並べ替えるプログラムを以下の条件で考えよ.
 - ♦ main()では2つの整数型変数を大きい順に並べ替える関数sort()を呼び出す.
 - ♦ sort()は2つの整数型変数のアドレスを受け取る. 関数内では受け取った2つの整数の大小を比較し, 小さい方が前ならば関数swap()を呼び出して2つの変数の中身を入れ替える.

sort.cの一例

```
/* sort.c */
#include <stdio.h>

void swap(int *x, int *y);
void sort(int *x, int *y);


int main(void)
{
    int a = 5;
    int b = 10;

    printf("a = %d, b = %d\n", a, b);
    sort(&a, &b);
    printf("a = %d, b = %d\n", a, b);
}

void swap(int *x, int *y)
{
    int tmp;
    tmp = *x;    /* xの値をtmpに一旦格納 */
    *x = *y;     /* yの値をxに代入 */
    *y = tmp;    /* tmp(元のx)の値をyに代入 */
}

void sort(int *x, int *y)
{
    if (*x < *y) swap(x, y);
}
```

swap()には
アドレスを渡す.



swapary.cの作成

- swap.cに要素数5の配列を2つ宣言し， 値を代入する.
main()内でswap()を繰り返し呼ぶことによって， 2つの配列の要素を全て交換するプログラムを考えよ.

swapary.cの一例

```
/* swapary.c */
#include <stdio.h>

#define NUM 5
void swap(int *x, int *y);

int main(void)
{
    int i;
    int a[NUM]={0, 1, 1, 2, 3};
    int b[NUM]={3, 5, 8, 13, 21};

    /* 入れ換え前の値を表示 */
    printf("before swapping\n");
    for (i=0; i< NUM; i++) {
        printf("a[%d] = %3d, b[%d] = %3d\n",
               i, a[i], i, b[i]);
    }

    /* 値の入れ換え */
    for (i=0; i< NUM; i++) {
        swap(&a[i], &b[i]);
    }

    /* 入れ換え後の値を表示 */
    printf("after swapping\n");
    for (i=0; i< NUM; i++) {
        printf("a[%d] = %3d, b[%d] = %3d\n",
               i, a[i], i, b[i]);
    }
}
```

```
void swap(int *x, int *y)
{
    int tmp;

    tmp = *x; /* xの値をtmpに格納 */
    *x = *y;   /* yの値をxに代入 */
    *y = tmp; /* tmpの値をyに代入 */
}
```

おわり