プログラミングI
数理物理，総合理学等向け

2019年1月21日

海谷 治彦

目次

- ポインタ その2

- 11章[レ] 10章[明] ポインタ
  - C言語の最大難関といわれています・・・
- 関数の引数とポインタ

- まとめ

ポインタとは何か？ (まとめ)

1. コンピュータのメモリは，プログラムの全域から，アクセスすることのできる配列のようなものである．
2. この配列の一区画には，1 byte のデータが入る．
   - よって，int 等のデータは，通常，連続した四区画が必要である．
3. アドレスとは，このメモリを配列と見なした場合の添え字(index)であり，番地という単位で呼ばれる．
4. プログラム内のおおむね全ての変数は，メモリの上に配置されている．
   - 特に配列は，順番に配置されていることが保証されている．
   - メモリ上に配置されない変数は本授業で扱わない．
5. ポインタとは，メモリ上に配置されている変数の型の種類(intやchar)とアドレスを記録する変数である．

アドレス	内容
0028FEE2	
0028FEE3	
0028FEE4	5
0028FEE5	
0028FEE6	
0028FEE7	
0028FEE8	3
0028FEE9	
0028FEEA	
0028FEEB	
0028FEEC	4
0028FEED	
0028FEEE	
0028FEEF	

復習

Scopeに関するちょっとした例

```
// scope2.c
#include <stdio.h>

int main(void){
int a=10; // ココで定義した変数 a と
int *p;
  p=(&a); // 上記のaのアドレスをpに保存
  printf("%d at %p¥n", a, &a);
  if(a==10){
  int a=22; // ココで定義した a は,
    (*p)=12; // 直上のaではなく,
             // main直下のaを更新
    printf("%d at %p¥n", a, &a);
  }
  printf("%d at %p¥n", a, &a);
  // 別物であることがアドレスを見るとわかる
  return 0;
}
```

```
sh-3.1$ ./a.exe
10 at 0028FEE8
22 at 0028FEE4
12 at 0028FEE8
```

アドレス	値
0028FEE3	
0028FEE4	内側の a 22
0028FEE5	
0028FEE6	
0028FEE7	
0028FEE8	外側の a 10→12
0028FEE9	
0028FEEA	
0028FEEB	

異なるScopeの変数アクセスできる理由

- スコープが異なっても，全ての変数はメモリ上に配置されている.
- ポインタによって，その配置されているメモリのアドレスを，プログラム中の任意の場所で知ることができる.
- このポインタを使って，メモリ上の変数を直接にアクセス(読み出しや更新)することできてしまう.

ポインタの値としての NULL

- ポインタが，どのアドレスも指していない，未定義の場合，ポインタの値に NULL を設定する.
- 多くの，ポインタの値を返す関数は，計算できない場合等に，NULL を返すように設計されている.
- 未定義状態のポインタを用いて，変数にアクセスしようとすると，当然，エラーとなる.
  - 所謂，NULL Pointer Error (ぬるぽ)
- ポインタの宣言(定義)時点では，値が NULL であるとは限らない.
  - 不定(適当)な値が入っている場合がある.

エラーが起こるサンプル

```
// nullpo.c 意図的にちゃんと動かないプログラム
#include <stdio.h>

int main(void){
int a=12, *p=NULL; // 意図的にNULLに初期化した
  // p=&a; // 意図的にポインタpの設定をコメント
  printf("%d¥n", *p); // 結果として何も指してない値をアクセスしエラー
  return 0;
}
```

再掲載

ポインタで何が嬉しい？

- コンピュータのメモリを直接操作してる雰囲気を味わえて面白い(人もいるかもしれない).

- 関数の引数を関数内で更新することが可能となる.
- より一般的には，異なるスコープにある変数にアクセスするための手段を提供する.
  - メモリはプログラム全体で共有されている配列のようなもの.
- 配列を走査(scan)するのに便利.
- 変数を事前に宣言せずに，プログラムの実行中に適宜，確保するのに役立つ. (本授業では扱いません)

ポインタと関数引数

- 関数内から引数の値を更新することはできない.
- より一般的には関数内部から外部(関数の呼び出し側)の変数を更新することはできない.

- ポインタは変数がある場所(アドレス)を渡すことができる.
- ポインタを関数に引数として渡すことで，関数内部から外部の変数を(間接的に)更新することができる.

更新できない例

```
// unswap.c
#include <stdio.h>

// aとbを入れ替えるつもりがかわらない orz
void unSwap(int a, int b){
int tmp;
  tmp=a;
  a=b;
  b=tmp;
}

int main(void){
int x=12, y=23;
  printf("%d %d¥n", x, y);
  unSwap(x, y);
  printf("%d %d¥n", x, y);

  return 0;
}
```

```
sh-3.1$ ./a.exe
12 23
12 23
sh-3.1$
```

値をコピーしてるんだから,
入れ替わらないのはあたりまえ.
以下とほぼ同じ.

```
a=x; b=y;

tmp=a;
a=b;
b=tmp;
```

上記でx,yが更新されるわけがない.

ちゃんと入れ替わる

```
// swap.c 教科書
// p.294とほぼ同じ
#include <stdio.h>

// ポインタを介して変数を更新
void swap(int *px, int *py){
int tmp;
  tmp=(*px);
  (*px)=(*py);
  (*py)=tmp;
}

int main(void){
int x=12, y=23;
  printf("%d %d¥n", x, y);
  swap(&x, &y);
  printf("%d %d¥n", x, y);

  return 0;
}
```

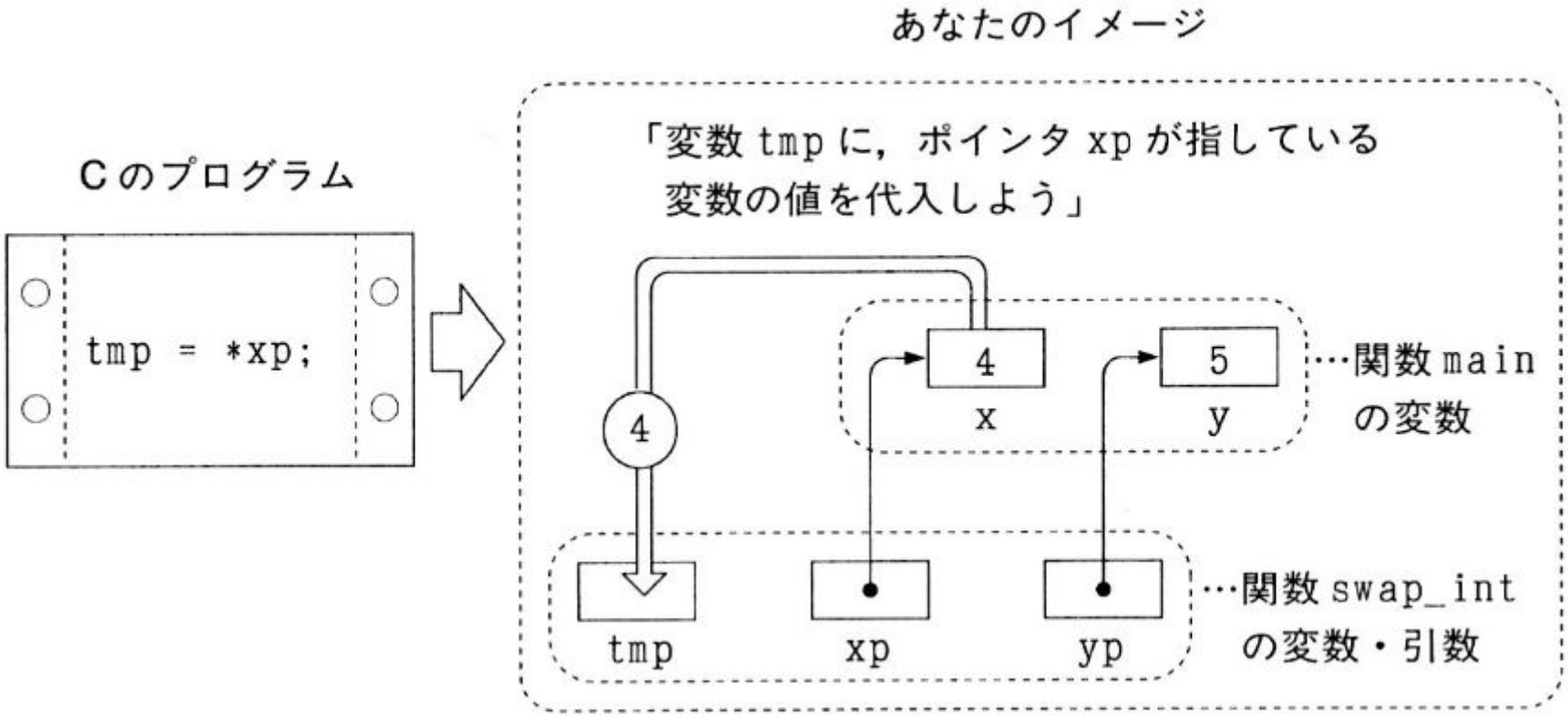
```
sh-3.1$ ./a.exe
12 23
23 12
sh-3.1$
```

コレが py に渡る

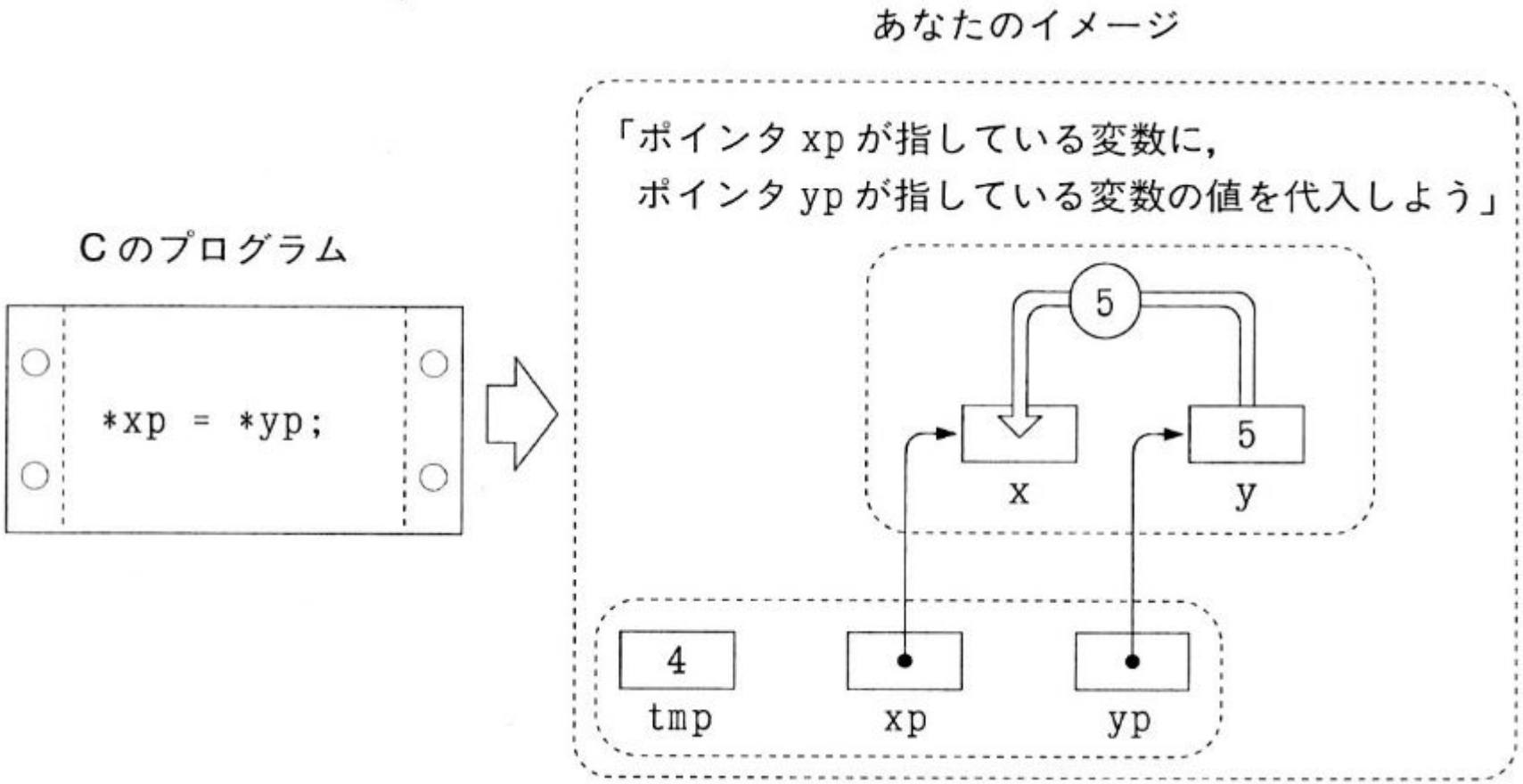
コレが px に渡る

アドレス	値
0028FEE7	
0028FEE8	y =23
0028FEE9	
0028FEEA	
0028FEEB	
0028FEEC	x =12
0028FEED	
0028FEEE	
0028FEEF	

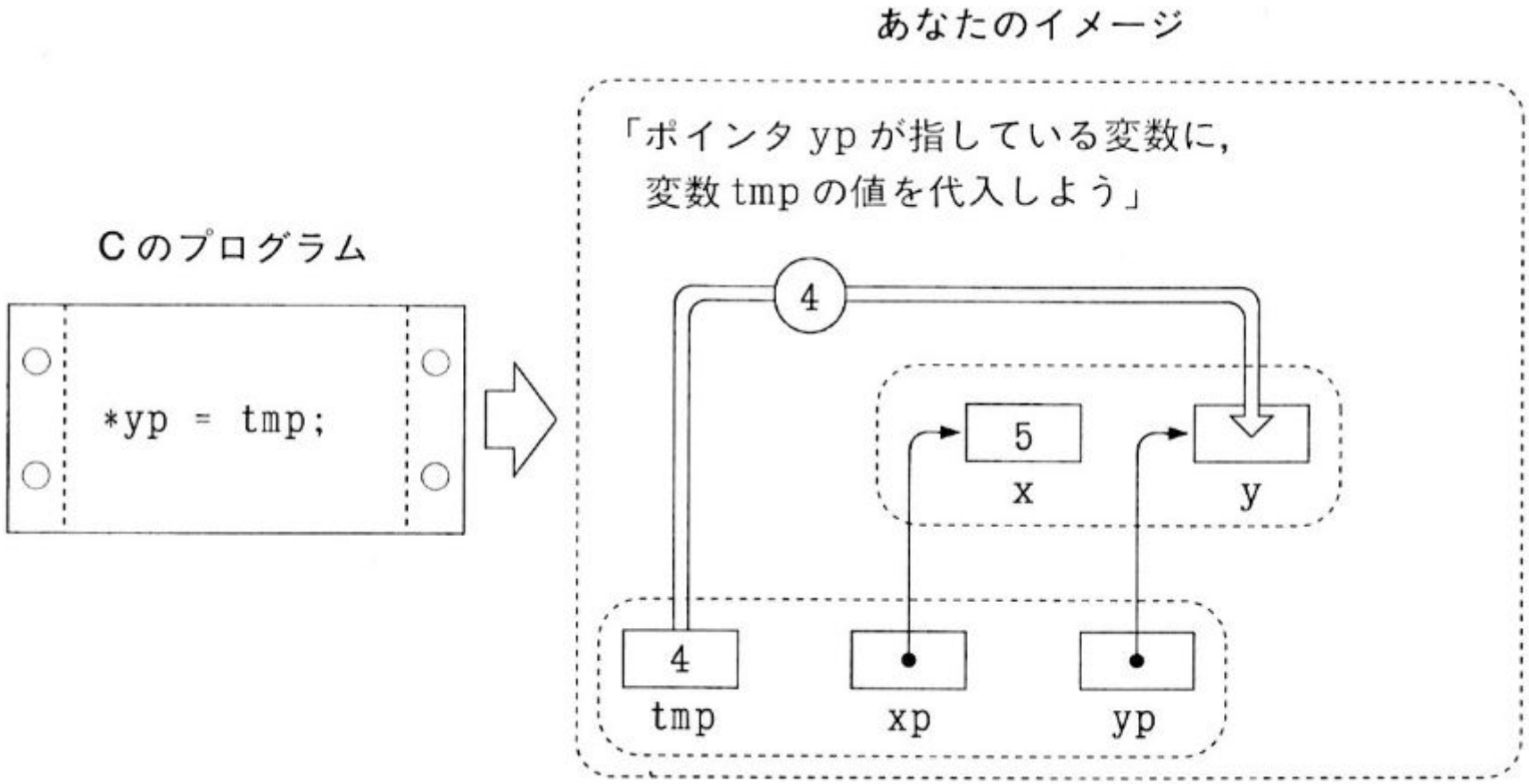
前頁の概念説明図 1/3


あなたのイメージ
Cのプログラム
tmp = *xp;
「変数 tmp に，ポインタ xp が指している
変数の値を代入しよう」
4
4
x
5
y
…関数 main
の変数
tmp
xp
yp
…関数 swap_int
の変数・引数

前頁の概念説明図 2/3


あなたのイメージ
「ポインタ xp が指している変数に，
ポインタ yp が指している変数の値を代入しよう」
Cのプログラム
*xp = *yp;
5
5
x
y
4
tmp
xp
yp

前頁の概念説明図 3/3


あなたのイメージ
「ポインタ yp が指している変数に,
変数 tmp の値を代入しよう」
Cのプログラム
*yp = tmp;
4
5
x
y
4
tmp
xp
yp

再掲載

ポインタで何が嬉しい？

- コンピュータのメモリを直接操作してる雰囲気を味わえて面白い(人もいるかもしれない).

- 関数の引数を関数内で更新することが可能となる.
- より一般的には，異なるスコープにある変数にアクセスするための手段を提供する.
  - メモリはプログラム全体で共有されている配列のようなもの.
- 配列を走査(scan)するのに便利.
- 変数を事前に宣言せずに，プログラムの実行中に適宜，確保するのに役立つ. (本授業では扱いません)

メモリ上の配列の配置

- メモリ上には配列は順番に並んでいることが保証されている.
  - 配列以外は保証されない.
- 右記のようにcharなら1個毎，intなら4個毎に.

- 配列の先頭のアドレスは,

&配列名[0]

で取得できる. 右記なら &c[0]

- アドレスはポインタに記録できる.
- ポインタは+と-の演算子を適用できる.

アドレス	値	変数名
0028FEDD		
0028FEDE	H	c[0]
0028FEDF	e	c[1]
0028FEE0	l	c[2]
0028FEE1	l	c[3]
0028FEE2	o	c[4]
0028FEE3	¥0	c[5]
0028FEE4		
002		
002		
002		
0028FEE8		

char c[]="Hello";
等宣言した場合.

配列の走査 (scan)

- 配列は添え字を使って順にアクセス(走査，scan)できるが，ポインタを使っても走査できる．

```
// scanstr1.c 添え字を使う
略
char c[]="Hello";
int i=0;
  while(i<5) {printf("%c", c[i]); i++;}
  printf("¥n");
略
```

```
// scanstr2.c ポインタを使う
略
char c[]="Hello";
char *p= &c[0];
  while(*p) {printf("%c", *p); p++;}
  printf("¥n");
略
```

&c[0] は c に同じ

- 配列cの最初の要素 c[0] のアドレスは，&c[0] である.
- このアドレスは，配列全体の先頭アドレスでもある.
- Cの文法上，&c[0] を 単に c と書いてよい.

```
// scanstr3.c ポインタを使う
略
char c[]="Hello";
char *p=c;
  while(*p){printf("%c", *p); p++;}
  printf("¥n");
略
```

配列を関数の引数に用いる

- 配列も関数の引数に用いることができます.
- 使い方は次頁のサンプル(先週の演習のバリエーション)を見てください.

- 復習になりますが，関数の中で引数を更新しても，呼び出し側で与えた引数の値は変わりません.
- この点において，引数が配列の場合，上記の性質が成り立ちません.
  - 関数中の配列要素の更新が，呼び出し側にも影響する！

サンプルコード

```
#include <stdio.h>

double average(double a[5]){
  double sum=0;
  int i=0;
    while(i<5) {sum += a[i]; i++;}
    sum /= i;
    return sum;
}

double variance(double a[5]){ // variance(double a[]) でもよい
  double sum=0, av;
  int i=0;
    av=average(a);
       while(i<5) {sum += (a[i]-av)*(a[i]-av); i++;}
    sum /= i;
    return sum;
}

int main(void){
  // 入力面倒なので初期化した
  double x[5]={3.0, 1.0, 4.0, 1.0, 5.0};
  printf("%f¥n", variance(x));
  return 0;
} // variance2.c
```

関数の配列引数とポインタ

- 配列を引数とした場合，実は関数には，配列の先頭のアドレスしか渡っていない．
- よって，

1. 関数内からでも外部の配列を更新できる．
2. 実は配列の長さは関数内から識別できてない．

の二点の重要な性質がある．

- 上記から関数の配列引数はポインタとして宣言しても意味はかわらない．すなわち，
- func(int a[10])
- func(int a[])
- func(int *a)

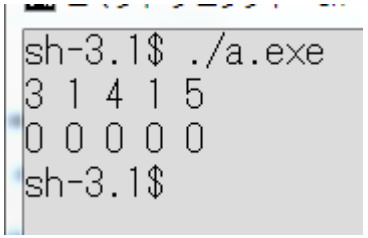
は同じ意味．

サンプルコード

```
// arrayarg1.c とりあえず正統的に
#include <stdio.h>

// 全てゼロにする
void updateZero(int a[5], int n){
int i=0;
  while(i<n) {a[i]=0; i++;}
}

int main(void){
int pi[5]={3, 1, 4, 1, 5}, i=0; // 適当に宣言
  while(i<5) {printf("%d ", pi[i]); i++;}
  printf("¥n");
  updateZero(pi, 5);
  i=0;
  while(i<5) {printf("%d ", pi[i]); i++;}
  printf("¥n");
  return 0;
}
```



以下でも結果は全て同じ

```
void updateZero(int a[], int n){
int i=0; // arrayarg2.c
  while(i<n) {a[i]=0; i++;}
}
```

```
sh-3.1$ ./a.exe
3 1 4 1 5
0 0 0 0 0
sh-3.1$
```

```
void updateZero(int *a, int n){
int i=0; // arrayarg3.c
  while(i<n) {*(a+i)=0; i++;}
}
```

```
int main(void){
int pi[]={3, 1, 4, 1, 5}, i=0; // 適当に宣言
  while(i<5) {printf("%d ", pi[i]); i++;}
  printf("¥n");
  updateZero(&pi[0], 5);
  i=0;
  while(i<5) {printf("%d ", pi[i]); i++;}
  printf("¥n");
  return 0;
}
```

構造体とポインタ

- 構造体が配置されているメモリもポインタに記憶できる. (構造体もポインタで指せる)
- 構造体は配列よりも普通の型(int等)に近い性質を持つ.
  - 丸ごと代入できる.
  - 関数の引数に渡しても, 関数側から変更ができない.
- 上記をかんがみて, 特に関数引数に渡す場合は, 構造体へのポインタを渡す場合がある.
- 尚, ポインタを介して, メンバーにアクセスする場合, ちょっとした略記法がある.
  - `(*ptr).member` は `ptr->member` と書いてよい.

サンプル

```
// struct3.c
中略

typedef struct person {
  char name[10];
  char bt; // ABの人ゴメン
  int weight;
  int height;
} PERSON;

// 構造体が配置されているアドレスを
// ポインタで渡すので更新できる
void update(PERSON *pp){
  strcpy((*pp).name, "NONE");
  (*pp).height=10;
}

int main(void){
PERSON ike={"Ike", 'A', 65, 175};
  printf("%s %c %d %d¥n", ike.name, ike.bt, ike.weight, ike.height);
  update(&ike);
  printf("%s %c %d %d¥n", ike.name, ike.bt, ike.weight, ike.height);

  return 0;
}
```

以下では更新されない struct2.c

```
void unUpdate(PERSON p){
  strcpy(p.name, "NONE");
  p.height=10;
}

中略

 unUpdate(ike);
```

以下の書き方もOK struct4.c

```
void update(PERSON *pp){
  strcpy(pp->name, "NONE");
  pp->height=10;
}
```

参考 構造体のメモリレイアウト

- 構造体も基本的にはメモリ上に順番に配置されている.
- しかし，場合によってはメンバー間に「詰め物」がはいっており，ぴったりくっついてない場合がある.

```
// struct.c
// 構造体のメモリレイアウト
#include <stdio.h>

typedef struct person {
  char name[10];
  char bt; // ABの人ゴメン
  int weight;
  int height;
} PERSON;
```

```
/
int main(void){
PERSON ike, sato, *sp;
int i;
  sp=(&ike);

  printf("%p %p, %p %p %p %p¥n",
    &ike, sp,
    &(ike.name), &(ike.bt),
    &(ike.weight), &(ike.height));

  return 0;
}
```

前頁の結果とレイアウト

```
sh-3.1$ ./a.exe
0028FED0 0028FED0, 0028FED0 0028FEDA 0028FEDC 0028FEE0
```

```
// struct.c
// 構造体のメモリレイアウト
#include <stdio.h>

typedef struct person {
  char name[10];
  char bt; // ABの人ゴメン
  int weight;
  int height;
} PERSON;
```

ココが
つめもの

```
0028FED0 name
0028FED1
0028FED2
0028FED3
0028FED4
0028FED5
0028FED6
0028FED7
0028FED8
0028FED9
0028FEDA bt
0028FEDB
0028FEDC weight
0028FEDD
0028FEDE
0028FEDF
0028FEE0 height
0028FEE1
0028FEE2
0028FEE3
0028FEE4
0028FEE5
```

scanfの意味

`int x; scanf("%d", &x)`

- 関数引数から整数xを変更しないといけないので，&を使い，xのアドレスをscanfに知らせる．

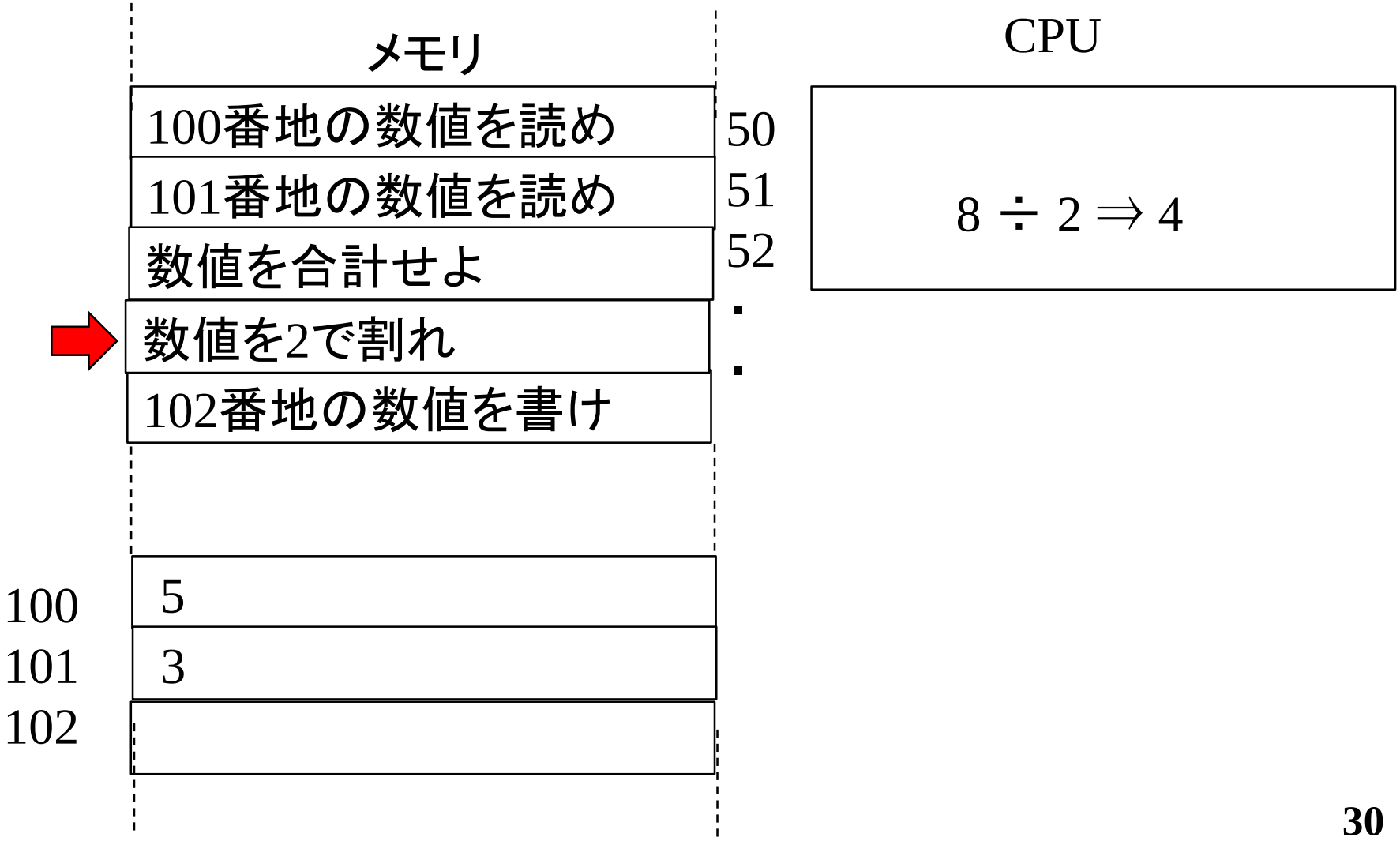
`char s[11]; scanf("%10s", s)`

- 関数引数から文字配列sを変更するので，配列sの先頭のアドレスをscanfに知らせる必要がある．
- 配列sの先頭のアドレスは `&s[0]`
- `&s[0]` は，単に `s` と書いてよい．

エピローグ

復習

簡易な例題 ～ 二値の平均



ノイマン型コンピュータ

- 前述のような，メモリにプログラムとデータの両方を記憶し，計算を進めて，結果をメモリ等に書き戻す計算方式のコンピュータ.
- ノイマンは人の名前.
  - 現在広く使われているコンピュータを考え出したアメリカ人.
  - 原爆開発プロジェクトでも大活躍した.

- ノイマン型コンピュータが広く使われる限り，C言語が廃れることは無い！
  - 量子コンピュータとかバイオコンピュータとかがデルやHPの格安パソコンで使われるようになれば，Cは寿命を終えるだろう.

基本的な概念

- 変数 (含む，配列や構造体)
- 逐次処理 (命令は書いてある順番に実行)
- 条件分岐 (if文)
- 繰り返し (for, while等)
- 関数

- 一部，尖った言語を除き多くの言語の原型となっている.
  - Java, C#, PHP, Perl, JavaScript, Ruby, Python, VB

簡単なCくらい知ってよう！

- 数学一筋，物理一筋を目指す人もいるかもしれません.
- しかし，今日では，数学，物理分野でも，プログラミング，そして，プログラミング的な考え方は必須といえます.
  - 今の科学は手作業で進めるには複雑すぎる.

- 本授業を良い機会だと思って，プログラミングの考え方を理解しよう.

演習14

- 整数列の最大値と最小値を計算する以下のような関数を作成せよ.

```
void getMinMax(int a[], int n, int *minp, int *maxp);
```

- aが入力となる数列，nは数列の長さ，*minp に最小値の結果を保存，*maxp に最大値の結果を保存.
- 数列の長さは，必ず1以上であるものとみなしてよい．すなわち，n > 0
- 数列には負の数も含まれている.
- 次頁のプログラム末に関数を追加する形で提出せよ．テストデータは変更してもよい.

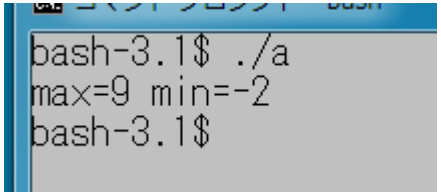
以下のような動作が期待される

```
// maxmin.c
#include <stdio.h>

// データa中の最大値 *maxp と，最小値 *minp を求める関数
// a データ n データの数 n>0 と想定してよい
// minp 最小値の変数へのポインタ maxp 最大値の変数へのポインタ
void getMinMax(int a[], int n, int *minp, int *maxp);

int main(void){
int data[]={3, 1, 4, 1, 5, 9, -2}; // テストデータ
int max, min;
        getMinMax(data, 7, &min, &max);
        printf("max=%d min=%d¥n", max, min);
}

// 以下に自分で getMinMax の中身を定義してね．
```



おしまい