情報基礎BII

柳本･野津
TA 中島

入力してみよう

プログラム1：大きい配列が欲しい！

```
#include <stdio.h>

#define MAX 10000000
                    適当に大きくする
int main(void){
  int i;
  int a[MAX];

  for(i = 0; i < MAX; i++){
    a[i] = 1;
  }

  return(0);
}
```

プログラム2：配列の大きさを指定したい！

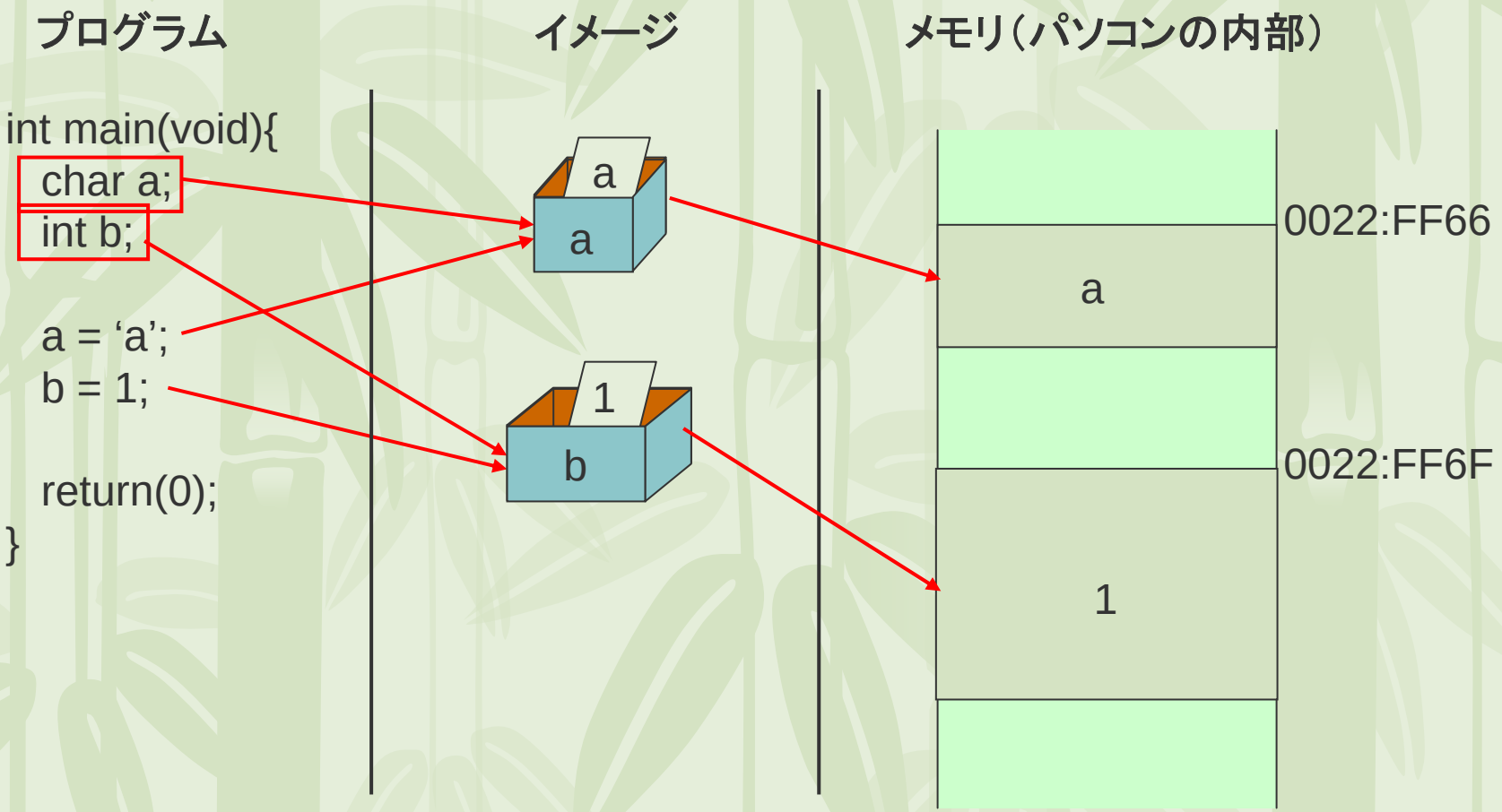
```
#include <stdio.h>

int main(void){
  int i;
  int a[i];

  printf(“How many array?: ”);
  scanf(“%d”, &i);

return(0);
}
```

なぜコンパイルできないのか？

- 変数の宣言で大きな配列を確保できない
- 配列の宣言で添え字に変数を利用できない

変数の仕組みを調べてみよう

コンピュータ内での変数



変数の宣言と同時にメモリ上に記憶場所が確保される

プログラムが動かない理由

- 変数の宣言で大きな配列を確保できない
  - 宣言と同時に記憶場所が決められるので、コンパイラが用意できる場所に限界がある
- 配列の宣言で添え字に変数を利用できない
  - 宣言と同時に記憶場所が決められるので、宣言と同時に大きさを指定しなくてはいけない

宣言と同時に記憶場所を用意しなければ解決できるのでは？

記憶場所を確保しないポインタ

- 宣言と同時に記憶場所を用意しない変数
  - ⇒ ポインタ
- ポインタの特徴
  - ポインタ変数を宣言してもすぐに利用できない
  - 自分で記憶場所を用意しなくてはいけない
    - 記憶場所を自分で指定できる
    - 記憶の大きさを自分で指定できる

ポインタについて

- 宣言方法
  - 変数の前に*を付ける

    *int *a;　　int型のポインタa*

    *char *b;　char型のポインタb*
- ポインタにはアドレスを代入する

  用意した記憶場所を指定する操作
- 格納した値は*を付けて読み出す

  ポインタに*を付ければ普通の変数と同じ

  *aや*cはint型の変数、char型の変数と同じ

アドレスについて

- アドレスとは

  変数などパソコンのメモリ上の場所

  ⇒ 変数にはすべてアドレスがある

- 他の変数のアドレスを取り出すには？

  変数の前に&を付ける

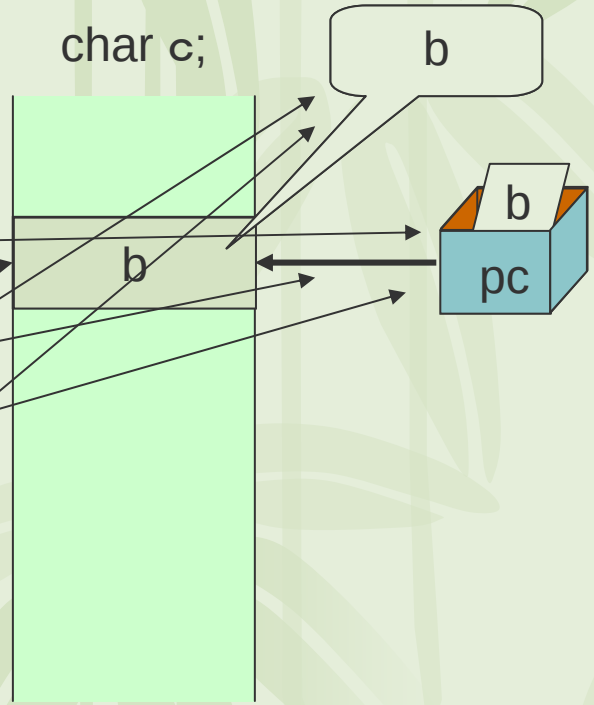
  &cは変数cのアドレスを表す

↓

&を使ってポインタにアドレスを設定する

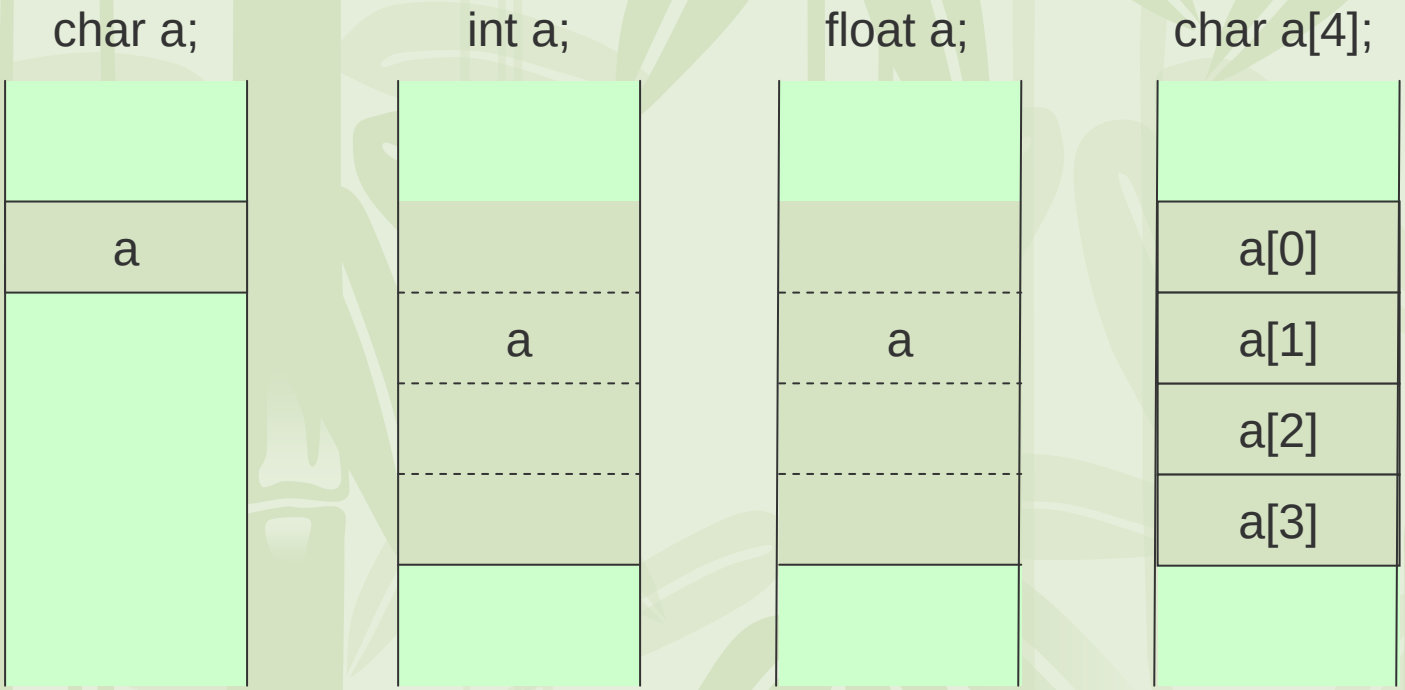
ポインタの使い方

```
#include <stdio.h>

int main(void){
  char c, *pc;

  c = ‘a’;
  pc = &c;
  printf(“%c¥n”, c);
  *pc = ‘b’;
  printf(“%c¥n”, c);

  return(0);
}
```



pcとcは同じものを示している
pcを操作するとcの中身も変化する

メモリ上の変数

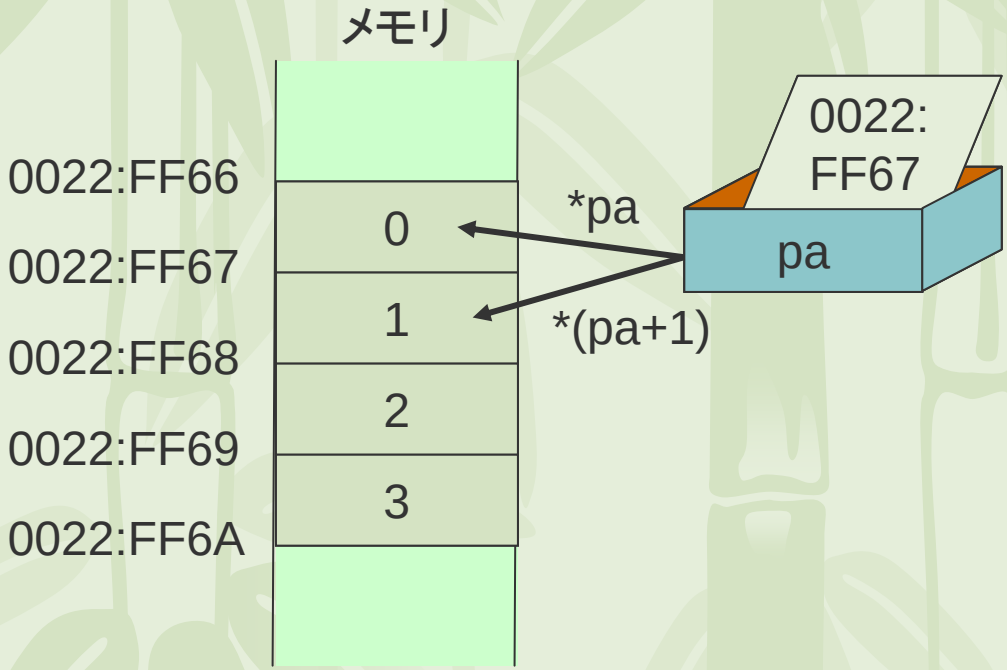


変数は型によって必要とするメモリの大きさが異なる

⇒ ポインタはアドレス以外に変数の型（大きさ）を指定しなくてはならない

メモリと配列

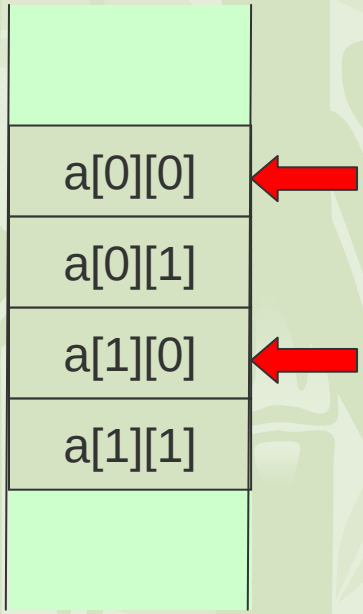
```
#include <stdio.h>

int main(void){
  char a[4], *pa;
  int i;

  for(i = 0; i < 4; i++){
    a[i] = i;
  }

  pa = a;
  printf("%d¥n", *pa);
  printf("%d¥n", *(pa+1));

  return(0);
}
```



- 配列の先頭アドレスは添字（[ ]）を取った配列名より得られる
- ポインタの演算はアドレスの変更となる

二次元配列とメモリ

char a[2][2];



二次元配列は要素の順番に気を付ければ
普通の配列と同じ

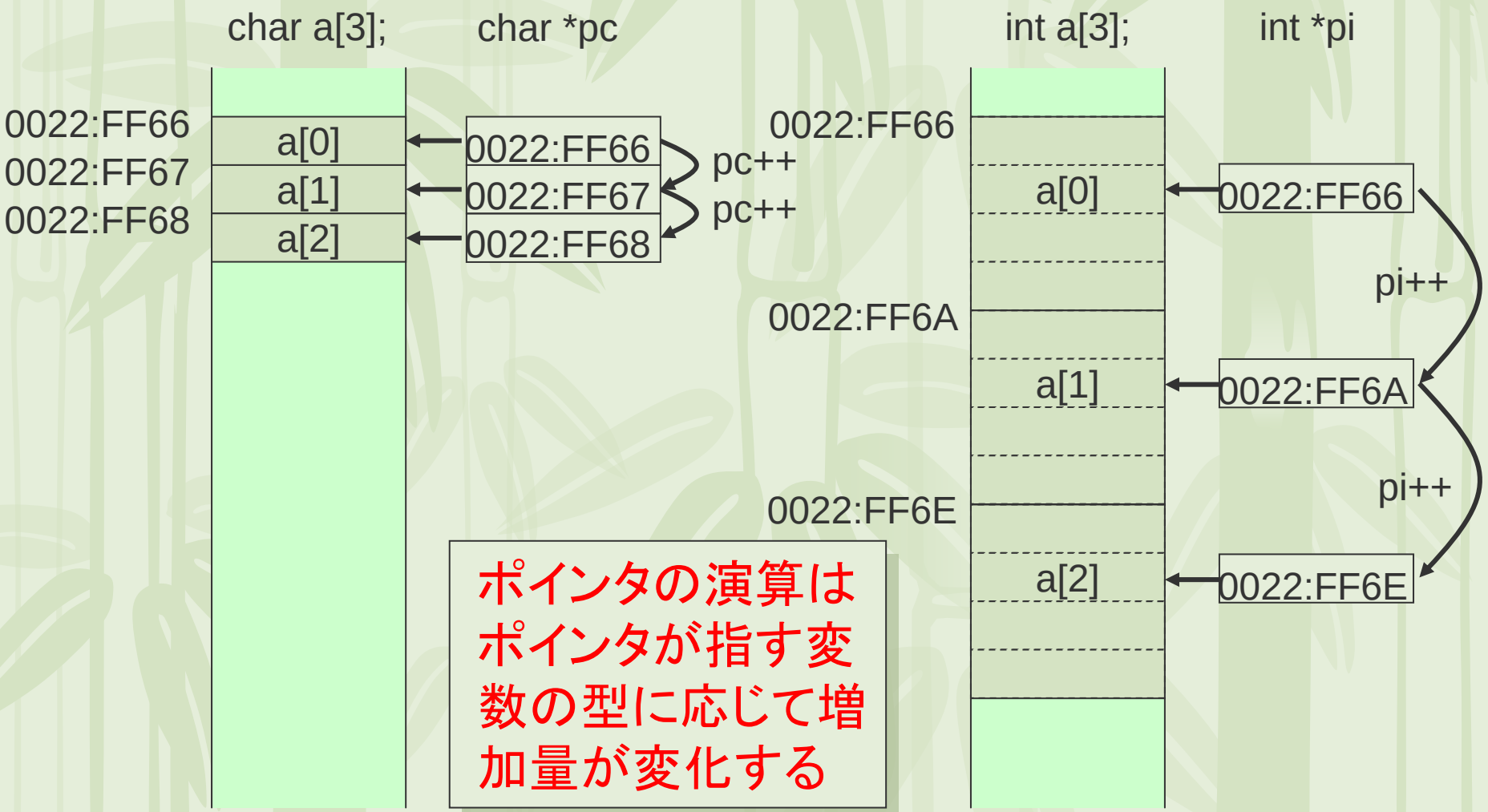
先頭のアドレス

a
a[0] ⇐ 同じアドレスを表す
&a[0][0]

その他のアドレス

a[1] ⇒ &a[1][0]を表す

ポインタ演算


char a[3];
char *pc
int a[3];
int *pi
0022:FF66
0022:FF67
0022:FF68
a[0]
a[1]
a[2]
0022:FF66
0022:FF67
0022:FF68
pc++
pc++
0022:FF66
0022:FF6A
0022:FF6E
a[0]
a[1]
a[2]
0022:FF66
0022:FF6A
0022:FF6E
pi++
pi++
ポインタの演算はポインタが指す変数の型に応じて増加量が変化する

ポインタ演算

ポインタ演算は配列の添え字と同じ考えで扱える

配列		ポインタ
a[1]	→	*(a+1)
a[2]	→	*(a+2)
a[i]	→	*(a+i)

プログラム中の配列をすべて
ポインタに変更することが可能

サンプルプログラム

```
#include <stdio.h>

int main(void){
  char a[10];
  int b[10];

  printf("&a[0]:%p¥n", &a[0]);
  printf("&a[1]:%p¥n", &a[1]);
  printf("&b[0]:%p¥n", &b[0]);
  printf("&b[1]:%p¥n", &b[1]);

  printf("a:%p¥n", a);
  printf("a+1:%p¥n", a+1);
  printf("b:%p¥n", b);
  printf("b+1:%p¥n", b+1);

  return(0);
}
```

charの配列はa+1でアドレスが1増える

intの配列はa+1でアドレスが4増える

ポインタ演算を行っても配列の場合と同じアドレスになっている
変数の型の大きさに応じてアドレスの増分が変わっている