

ポインタ変数

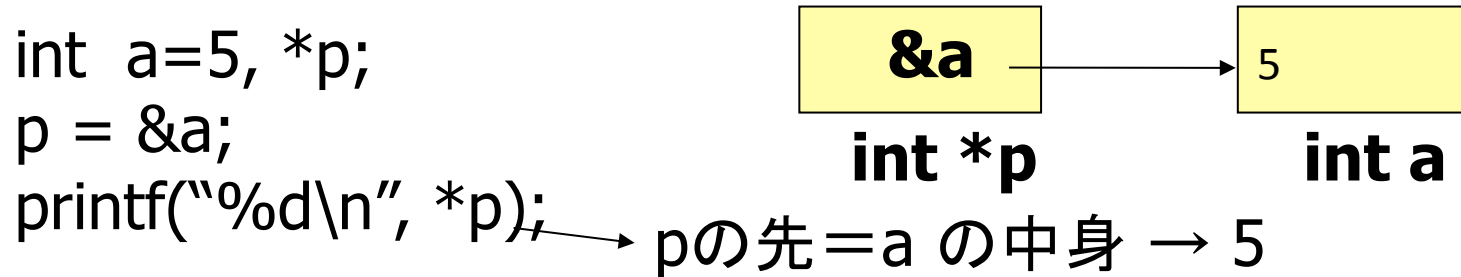
ポインタ変数とは

- データ(変数)の場所を指し示す
型 *変数名;
- 変数のアドレスを代入する
- &変数名 で変数のアドレスを参照する
- *変数名 でアドレス先の変数値を参照する

```
int a=5, *p;
```

```
p = &a;
```

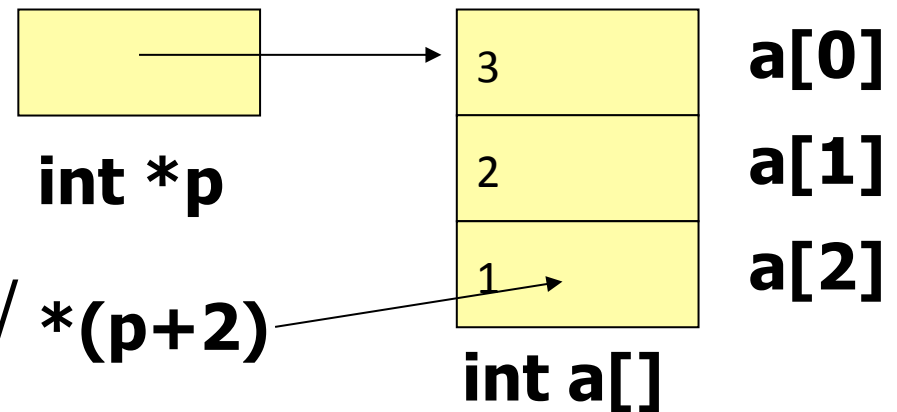
```
printf("%d\n", *p);
```



配列とポインタ

- 配列名は配列の先頭アドレスを表す
- ポインタをn増減すると、ポインタの先の配列をn個ずらす

```
int a[3], *p;  
a[0]=3; a[1]=2; a[2]=1;  
p = a; /* &(a[0]) */  
printf("%d\n", *p); /* 3 */  
printf("%d\n", *(p+2)); /* 1 */  
printf("%p\n", p); /* ??? */
```



ポインタ演算

- ポインタは加減すると適切な大きさを増減してくれる
 - char* なら1バイト : sizeof (char)
 - int* ならば4バイト : sizeof (int)

```
int a[3], *p;  
a[0]=3; a[1]=2; a[2]=1;  
p = a;  
printf("%p\n", p);  
printf("%p\n", p+1); /* 上の値よりも4だけ大きい */
```

配列とポインタ(相違点)

- 配列とポインタは同じように使える

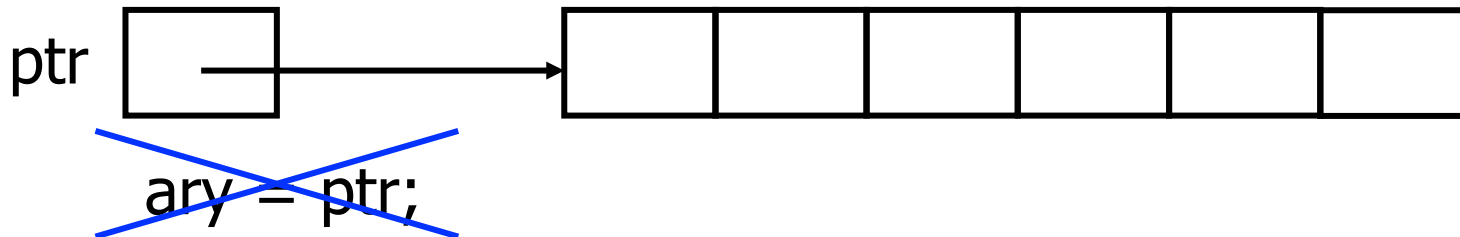
```
char ary[128];
```

```
*(ary+10) = 'E'; ary[10] = 'E'; /* *ary + 10 = はエラー */
```

- 相違点

- 配列は並びに対する名前なので、変数ではない。
ゆえに、ポインタのようにアドレス値を代入できない。

```
char *ptr = ary; char ary[128];
```



関数呼び出しとポインタ

- 関数の引数にポインタを使うと
 - 呼び出した先の関数から呼んだ方のデータを操作できる
 - 外部のデータを操作する必要がある時
 - 複数の値を処理結果として渡すとき
 - 文字列

入力関数と出力関数の違い

- 入力は与えられた変数の値を書き換える

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

アドレスの値を渡す



- 出力は与えられた変数の値を参照する

```
int x;  
printf ("%d", x);
```

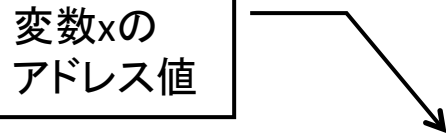
変数の値を渡す



ポインタ引数の使用例

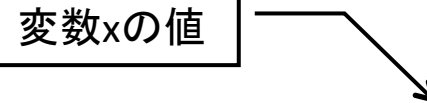
- 二つの変数の値を入れ替える

変数xの
アドレス値



```
void swap (int *a, int *b) {  
    int tmp;  
    tmp = *a;  
    *a = *b; /* x,yの値 */  
    *b = tmp;  
}  
int x, y;  
呼び出し方: swap (&x, &y);
```

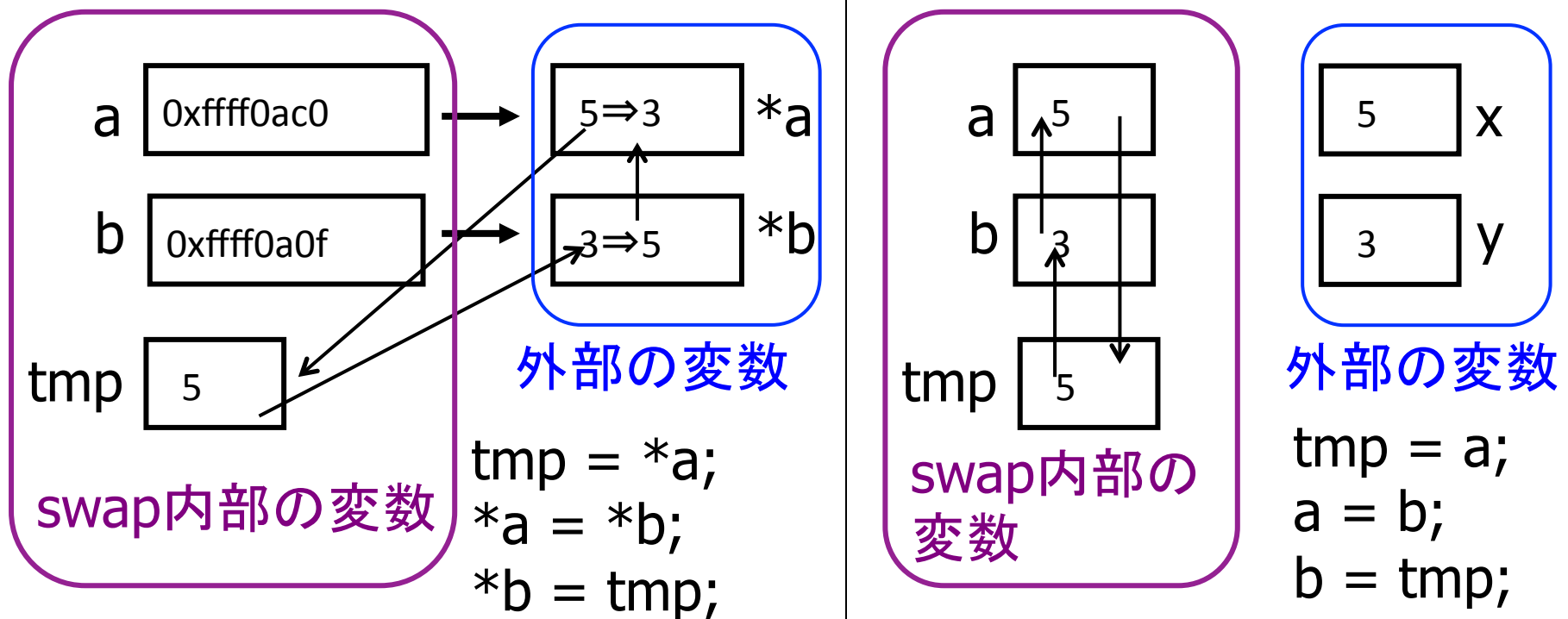
変数xの値



```
void swap (int a, int b) {  
    int tmp;  
    tmp = a;  
    a = b; /* a,bの値 */  
    b = tmp;  
}  
int x, y;  
呼び出し方: swap (x, y);
```


ポインタ引数の使用例

- 二つの変数の値を入れ替える



動的な領域の確保：malloc

- 配列⇒実行時に領域が固定
- ポインタ⇒実行時に領域を指定できる
- ポインタを動的な配列として使用する

型変換演算子

```
int *p, size, i;  
scanf ("%d", &size);  
p = (int *) malloc (sizeof (int) * size);  
p[0] = p[1] = 1;  
for (i = 2; i < size; i++) {  
    p[i] = p[i-1] + p[i-2]; /* フィボナッチ数列 */  
} /* p[size] とすると暴走... */
```

確保した領域の開放 : free

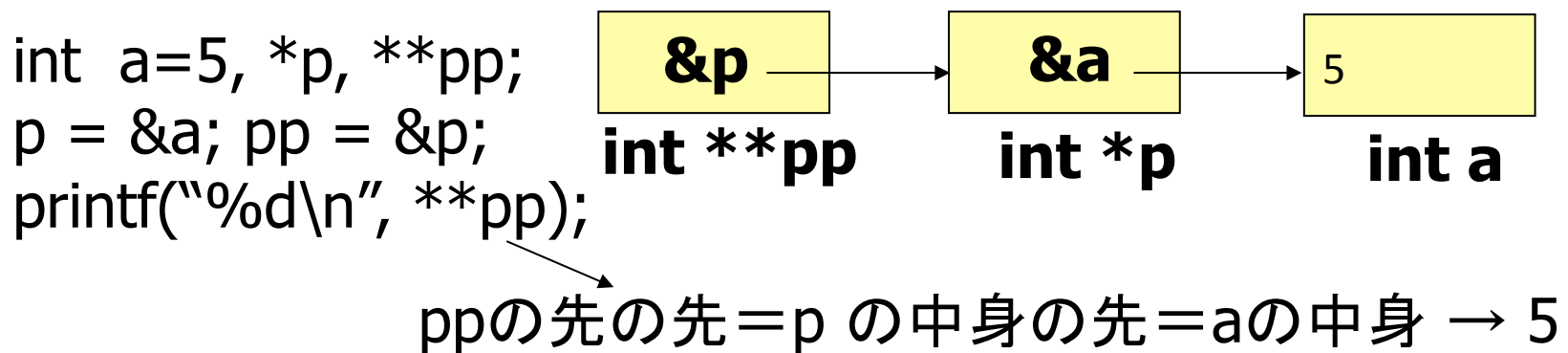
```
int *p, size, i;  
scanf ("%d", &size);  
p = (int *) malloc (sizeof (int) * size);
```

p を用いた処理;

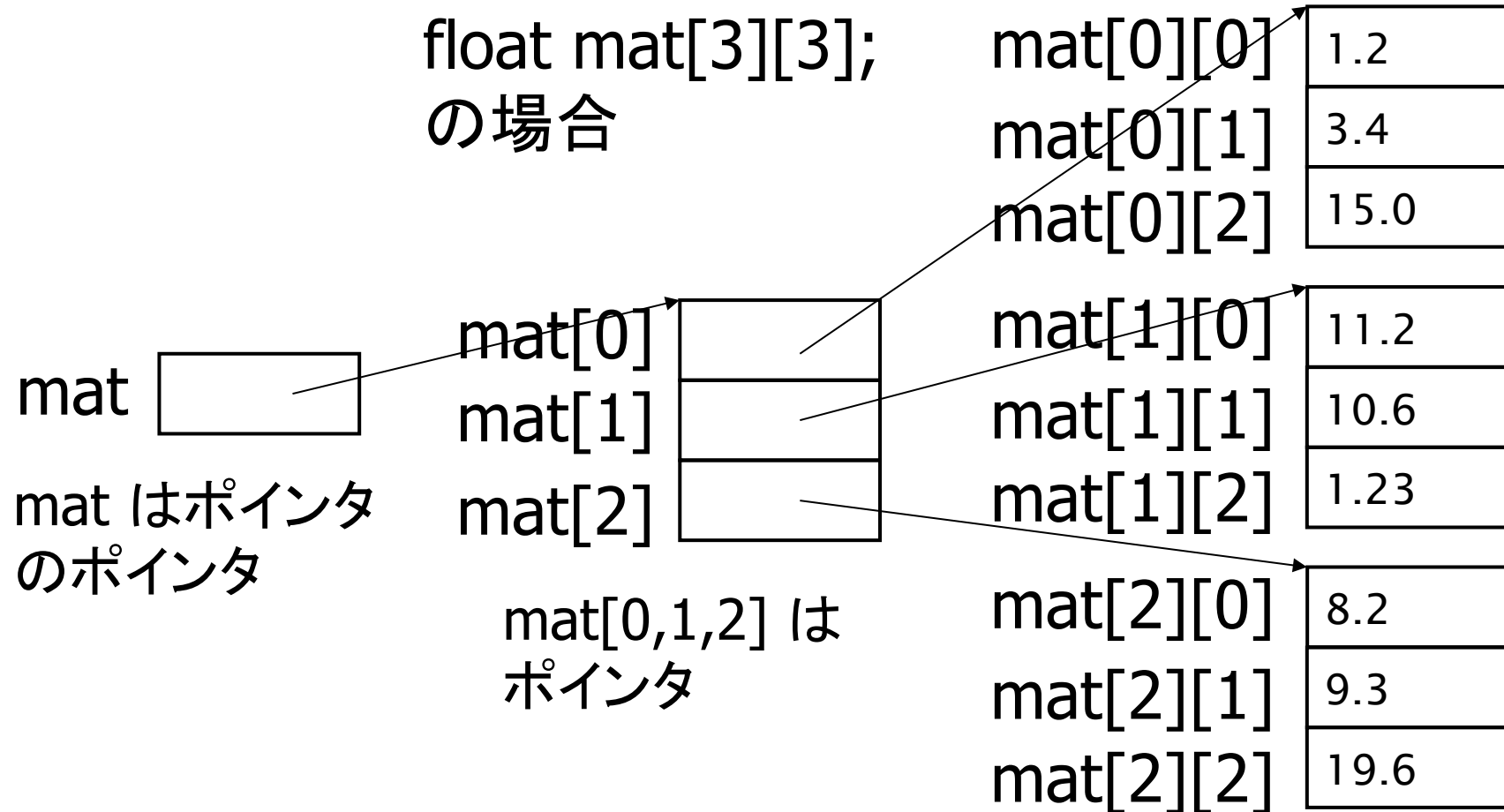
```
free (p); /* 最後に p の指す領域を開放する */  
return;
```

ポインタのポインタとは

- ポインタ変数の場所を指し示す
型 **変数名;
- *変数名 でアドレス先のポインタ変数値を参照
- **変数名 でアドレス先のポインタ変数のアドレス
先の変数値を参照する



2次元配列におけるポインタ



2次元配列の動的な領域確保

- 2次元配列 \Rightarrow 配列の配列と考える
- ポインタのポインタ(配列)を用いる

```
int **mat, row, column, i;  
scanf ("%d%d", &row, &column);  
/* ポインタ変数をrow個格納する領域の確保 */  
mat = (int **) malloc (sizeof (int *) * row);  
/* column個のint領域を確保した先頭アドレスの代入 */  
for (i = 0; i < row; i++) {  
    mat[i] = (int *) malloc (sizeof (int) * column);  
}  
mat[row - 1][column - 1] = 1; /* 配列のように扱える */
```

2次元配列領域の開放

```
void freeMatrix (int **mat, int row) {  
    int i;  
    for (i = 0; i < row; i++) {  
        free (mat[i]);  
    }  
    free (mat);  
}
```

関数へのポインタ

ポインタ変数で関数を呼び出す

- 関数のポインタの宣言

```
void (*funcPtr) (float v);
```

- funcPtr は、float型の変数を引数に取り、返り値を持たない関数が、メモリ上に配置されている先頭アドレスの値を保持する変数.

同じファイル内に関数

void func (float v) { ... } が作成されているとして、

funcPtr = func; で、

関数 func のアドレスの値が funcPtr に渡される。

利用ケース：繰返しの抽象化

```
void iterator (void (*func) (float), int n, float v) {  
    int i;  
    for (int i = 0; i < n; i++)  
        func (v);  
}  
  
void myFunc (float x) { ... } /* 関数本体の宣言*/  
/* 関数myFunc(12.3) を30回繰り返す */  
iterator (myFunc, 30, 12.3);
```