C プログラミング　入門

— ポインタ —

早稲田大学

本日の目標

ポインタを理解する

- メモリとアドレス
- swap 関数
- ポインタ型変数
- アドレス演算子「&」
- 間接参照演算子「＊」
- 配列とポインタ

メモリとアドレス

- メモリはバイト単位で扱う（１バイト毎にアドレスが割り振られている）
- 1byte は 8bit
- 変数を宣言すると，必要なメモリが自動的に確保される
  - char 型　：　1byte （8bit）
  - int 型　：　4byte（32bit）
  - double 型　：　8byte（64bit）
- 変数の読み書きは，メモリを確保した場所（アドレス）を指定することで行われる
- アドレスを値に持てる変数をポインタ型変数という

アドレス	メモリ内容	割り当て例
0x00000001	0x32	int a が割り当てられている
0x00000002	0x4a	
0x00000003	0x2f	
0x00000004	0xaf	
0x00000005	0xd4	double b が割り当てられている
0x00000006	0x29	
0x00000007	0x82	
0x00000008	0xcc	
0x00000009	0x3d	
0x0000000a	0x10	
0x0000000b	0x04	
0x0000000c	0x27	
0x0000000d	0x7d	char c
...	...	

メモリとアドレス

- メモリはバイト単位で扱う（１バイト毎にアドレスが割り振られている）
- 1byte は 8bit
- 変数を宣言すると，必要なメモリが自動的に確保される

  char 型　：　1byte （8bit）
  int 型　：　4byte（32bit）
  double 型　：　8byte（64bit）

- 変数の読み書きは，メモリを確保した場所（アドレス）を指定することで行われる
- アドレスを値に持てる変数をポインタ型変数という

アドレス	メモリ内容	割り当て例
0x00000001	0x32	int a が割り当てられている
0x00000002	0x4a	
0x00000003	0x2f	
0x00000004	0xaf	
0x00000005	0xd4	double b が割り当てられている
0x00000006	0x29	
0x00000007	0x82	
0x00000008	0xcc	
0x00000009	0x3d	
0x0000000a	0x10	
0x0000000b	0x04	
0x0000000c	0x27	
0x0000000d	0x7d	char c
...	...	

関数復習（ローカル変数のスコープ）

- 関数内で引数の値を変更しても，呼び出した引数の値は変更されない
- 別の関数内なら，同じ名前の変数を宣言しても別物

```
#include <stdio.h>
void func(int x){          /* 引数 x はこの関数内だけ */
  x=7;          /* x を変更しても下の x は影響を受けない */
}
int main(void){
  int x=3;
  func(x);                /* x=3 が上の x にコピーされる */
  printf("x is %d.¥n", x);              /*「x is 3.」*/
  return 0;
}
```

swap 関数

２つの変数の内容を交換する関数

```
void swap(int a, int b){
     int c;
     c = a;      a = b;       b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(x,y);          /*この関数で x, y を交換させたい*/
     :
}
```

- 上では，呼び出し元の引数は変更されない
  (値がコピーされて渡されるだけで，変数はそのまま)
  → 変更したい変数のアドレスが分かれば，変更できる
  → アドレスを扱いたい
- アドレス演算子「&」，ポインタ型変数，間接参照演算子「*」を使うと可能．(scanf のときに使ったものが実はアドレス演算子)

swap 関数

２つの変数の内容を交換する関数

```
void swap(int a, int b){
     int c;
     c = a;      a = b;       b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(x,y);          /*この関数で x, y を交換させたい*/
     :
}
```

- 上では，呼び出し元の引数は変更されない
  (値がコピーされて渡されるだけで，変数はそのまま)
  → 変更したい変数のアドレスが分かれば，変更できる
  → アドレスを扱いたい
- **アドレス演算子**「&」，ポインタ型変数，間接参照演算子「*」を使うと可能．(scanf のときに使ったものが実は**アドレス演算子**)

アドレス演算子

アドレス演算子「&」は，変数のアドレスを返す

&変数名

```
int a=50;
printf("a=%d, hex=%x, adress=%p ¥n",a ,a, &a);
```

- "%x" は整数を 16 進数で表示せよという変換指定文字
- "%p" はアドレスを表示せよという変換指定文字
- a のアドレスが表示される

アドレス	メモリ内容	
0x1234abc0 (a のアドレス)	0x32 (a の値)	int a が割り当て
0x1234abc4	...	
...	...	

&a : a のアドレス

ポインタ型変数

ポインタ型変数は，アドレス一つを記憶しておける

型 *変数名

```
int *p, a=50;
p = &a;
```

- 型は，ポインタが指し示す変数の型を指定する

アドレス	メモリ内容	
0x1234abc0 (a のアドレス)	0x32 (a の値)	int a が割り当て
0x1234abc4	0x1234abc0 (a のアドレス)	int *p が割り当て
0x1234abc8	...	
...	...	

*p : a のアドレスを記憶するための変数
p=&a : a のアドレスをポインタ型変数 p に記憶

ポインタ型変数

- **ポインタ型変数を複数まとめて宣言するときは，すべてに「*」をつける**

```
int *p, *q;
```

- **型を特に指定したくない場合は，**void **とかく**

```
void *p;
```

- **ポインタが** NULL **のときは，何も指し示していない．と解釈される**

```
int *p;
p = ... ;
if (p==NULL) ...;
```

間接参照演算子

- 間接参照演算子「*」はポインタ型変数が示す元の変数にアクセスできる
  - * ポインタ型変数
- ポインタを使用し，値を代入したり読み出したりできる

アドレス	メモリ内容	
0x1234abc0 (a のアドレス)	0x32 (a の値)	int a が割り当て
0x1234abc4	0x1234abc0 (a のアドレス)	int *p が割り当て
0x1234abc8	...	
...	...	

```
int a;     /*変数の宣言*/
int *p;    /*ポインタ型変数の宣言*/
p = &a;    /*ポインタに a のアドレスを記憶*/
*p = 3;    /*p は a を指しているので a=3; と同じ*/
```

例題 1

次のプログラムの出力結果を予想し，実際に表示し確かめよ．

```
#include<stdio.h>

int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
    return 0;
}
```

- **プログラム名は** pointer.c **とせよ**
- **実行し，表示を確かめよ**

例題 1

次のプログラムの出力結果を予想し，実際に表示し確かめよ．

```
int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
}
```

【出力結果】

```
a=5, b=10, *p=5
a=10, b=10, *p=10
```

例題 1

次のプログラムの出力結果を予想し，実際に表示し確かめよ．

```
int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
}
```

【出力結果】

```
a=5, b=10, *p=5
a=10, b=10, *p=10
```

例題 1

次のプログラムの出力結果を予想し，実際に表示し確かめよ．

```
int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
}
```

【出力結果】

```
a=5, b=10, *p=5
a=10, b=10, *p=10
```

例題 1

次のプログラムの出力結果を予想し，実際に表示し確かめよ．

```
int main(void){
    int a=5,b=10;
    int *p;
    p=&a;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);

    *p=b;
    printf("a=%d, b=%d, *p = %d¥n",a,b,*p);
}
```

【出力結果】

```
a=5, b=10, *p=5
a=10, b=10, *p=10
```

swap 関数

２つの変数の内容を交換する関数

```
void swap(int *a, int *b){    /* 交換したい変数のアドレスを受け取る*/
     int c;
     c = *a;     *a =*b;       *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);              /* アドレスを引数として渡してやる */
     :
}
```

	アドレス	メモリ内容
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

参考：

```
int a;   /*変数の用意*/
int *p;  /*ポインタ型変数の用意*/
p = &a;  /*ポインタに a のアドレスを記憶*/
*p= 3;   /* p は a を指しているので a=3 と同意*/
```

swap 関数

２つの変数の内容を交換する関数

```
void swap(int *a, int *b){    /* 交換したい変数のアドレスを受け取る*/
     int c;
     c = *a;     *a =*b;       *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);              /* アドレスを引数として渡してやる */
     :
}
```

	アドレス	メモリ内容
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

参考：

```
int a;   /*変数の用意*/
int *p;  /*ポインタ型変数の用意*/
p = &a;  /*ポインタに a のアドレスを記憶*/
*p= 3;   /* p は a を指しているので a=3 と同意*/
```

swap 関数

２つの変数の内容を交換する関数

```
void swap(int *a, int *b){    /* 交換したい変数のアドレスを受け取る*/
     int c;
     c = *a;     *a =*b;       *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);              /* アドレスを引数として渡してやる */
     :
}
```

	アドレス	メモリ内容
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

参考：

```
int a;   /*変数の用意*/
int *p;  /*ポインタ型変数の用意*/
p = &a;  /*ポインタに a のアドレスを記憶*/
*p= 3;   /* p は a を指しているので a=3 と同意*/
```

swap 関数

２つの変数の内容を交換する関数

```
void swap(int *a, int *b){    /* 交換したい変数のアドレスを受け取る*/
     int c;
     c = *a;     *a =*b;       *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);              /* アドレスを引数として渡してやる */
     :
}
```

	アドレス	メモリ内容
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

参考：

```
int a;   /*変数の用意*/
int *p;  /*ポインタ型変数の用意*/
p = &a;  /*ポインタに a のアドレスを記憶*/
*p= 3;   /* p は a を指しているので a=3 と同意*/
```

swap 関数

２つの変数の内容を交換する関数

```
void swap(int *a, int *b){    /* 交換したい変数のアドレスを受け取る*/
     int c;
     c = *a;     *a =*b;       *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);              /* アドレスを引数として渡してやる */
     :
}
```

	アドレス	メモリ内容
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

参考：

```
int a;   /*変数の用意*/
int *p;  /*ポインタ型変数の用意*/
p = &a;  /*ポインタに a のアドレスを記憶*/
*p= 3;   /* p は a を指しているので a=3 と同意*/
```

swap 関数

２つの変数の内容を交換する関数

```
void swap(int *a, int *b){    /* 交換したい変数のアドレスを受け取る*/
     int c;
     c = *a;     *a =*b;       *b = c;
}
int main(void){
     int x=3, y=7;
     :
     swap(&x,&y);              /* アドレスを引数として渡してやる */
     :
}
```

	アドレス	メモリ内容
x	(&x)	3 → 7
y	(&y)	7 → 3
a		&x
b		&y
c		3

参考：

```
int a;   /*変数の用意*/
int *p;  /*ポインタ型変数の用意*/
p = &a;  /*ポインタに a のアドレスを記憶*/
*p= 3;   /* p は a を指しているので a=3 と同意*/
```

例題 2

int 型のポインタを２つ受け取る order という関数をつくり，１番目のポインタが示す int 型変数の値が２番目のそれより大きい場合は，値を入れ替えよ（値の入れ替えには前の swap 関数を使うこと）

- ファイル名は「order.c」とする
- main 関数内で $x = 7, y = 3$ として，アドレス・値を表示
- 次に，order(&x,&y); として，$x \leq y$ となるようにせよ
- もう一度 x のアドレス・値，y のアドレス・値を表示
- 表示は以下のようにする．

```
x:  address = 0xbffff944, value = 7,
y:  address = 0xbffff940, value = 3,
x:  address = 0xbffff944, value = 3,
y:  address = 0xbffff940, value = 7,
```

配列とポインタ

- **配列はメモリ上に連続して確保される（右図，int 型配列）**
- **この場合，A は A[0] のアドレスを表すポインタのようなもの**
- **先頭のアドレスが分かれば，int 型は 4byte なので，A[1],A[2] などのアドレスも計算可**
- ***A としても A[0] と同じ意味**
- **配列の i 番目のアドレスを求めるには，変数の場合と同様，&をつける（`&A[i]`）**

アドレス	メモリ内容	割り当て例
0x00000001	0x32	A[0]
0x00000002	0x4a	
0x00000003	0x2f	
0x00000004	0xaf	
0x00000005	0xd4	A[1]
0x00000006	0x29	
0x00000007	0x82	
0x00000008	0xcc	
0x00000009	0x3d	A[2]
0x0000000a	0x10	
0x0000000b	0x04	
0x0000000c	0x27	
...	...	

配列とポインタ

- 配列はメモリ上に連続して確保される（右図，int 型配列）
- この場合，A は A[0] のアドレスを表すポインタのようなもの
- 先頭のアドレスが分かれば，int 型は 4byte なので，A[1],A[2] などのアドレスも計算可
- *A としても A[0] と同じ意味
- 配列の i 番目のアドレスを求めるには，変数の場合と同様，&をつける（&A[i]）

アドレス	メモリ内容	割り当て例
0x00000001	0x32	A[0]
0x00000002	0x4a	
0x00000003	0x2f	
0x00000004	0xaf	
0x00000005	0xd4	A[1]
0x00000006	0x29	
0x00000007	0x82	
0x00000008	0xcc	
0x00000009	0x3d	A[2]
0x0000000a	0x10	
0x0000000b	0x04	
0x0000000c	0x27	
...	...	

配列を引数で渡す

配列を引数で渡す時は，受け側ではポインタ型を用いる

要素数 size **の** int **型配列をすべて０にする**

```
void setZero( int *a, int size){
    int i;
    for(i=0; i<size; i++)
        a[i]=0;
}
int main(void){
    int A[10];
    setZero(A,10);                    /*ここで&A としない*/
    ⋮
```

まとめ

- **ポインタの取り扱い**
  宣言：型　*変数名;
  &a：a のアドレス（アドレス演算子）
  ***p：a のアドレスを記憶するための変数（ポインタ型変数）**
  p=&a：a のアドレスをポインタ型変数 p に記憶

- **参照渡し**
  関数の呼び出し側の変数を変更する関数を作るには，引数をポインタにする．

  整数型 a に値を代入したい（a の変数を変更する）

```
int a;
scanf("%d", &a); /*scanf("%d", a); ではダメ*/
```

- **配列とポインタ**