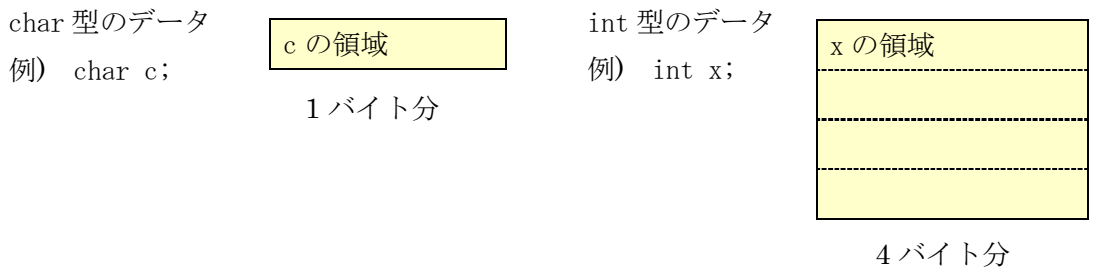
2013 年 6 月 13 日（木）実施

ポインタ

アドレス

C 言語のプログラムを実行する際，主記憶装置（メインメモリ）上にプログラムを配置して，中央処理装置（CPU）で解釈・実行することは，第１回の教材で述べたが，プログラムで利用される変数，定数，配列，関数もメインメモリ上でプログラムそのものとは別の場所（データ領域）に配置される。このとき，**メインメモリ上に格納されたプログラムやデータの場所を特定する**ための**アドレス**（番地）はメモリ空間をバイト単位で区切って番号付けされて表される。ここで，何番地にプログラムの先頭が配置されるかはそのときの状況により，一定していない。

C 言語では，データがメインメモリ上で占める領域が何バイト分となるかは，データ型ごとに定められている。例えば，char 型のデータは 1 バイト分の領域を占有する。int 型の場合には，何バイト分を占めるかは CPU に依存し，32 ビット CPU の場合には通常 4 バイト分を占める。



ポインタ変数，アドレス演算子，間接演算子

プログラミングの用語では，一般に**他のデータを指し示すデータ**を**ポインタ**と呼び，C 言語では他のデータを指し示すためにアドレスを利用する。C 言語で利用できるポインタには，次のようなものがある。

1）変数を指すポインタ
2）配列を指すポインタ
3）ポインタを指すポインタ
4）ポインタを指すポインタを指すポインタ
5）ポインタ配列を指すポインタ
6）構造体を指すポインタ
7）関数を指すポインタ
8）ファイルを指すポインタ

アドレスは数値として表されるため，C 言語ではその数値を格納する変数として**ポインタ変数**が利用できる。ポインタ変数を用いるには，先ず次のように変数名に * を付けて宣言を行う。

データ型 *変数名;

ここでのデータ型は，ポインタ変数が指している相手先データのデータ型である。(ポインタ変数そのものがメインメモリ上で占有するバイト数は，アドレスを表現するのに必要なバイト数である)

例）int *p;　　/* int 型のデータを指すポインタ変数 p */

宣言されたポインタ変数に，指し示すデータをセットするには，そのデータの格納場所を表すアドレスを代入する。指し示すデータが通常の変数の場合，**アドレス演算子** & を付けてアドレスを表す。次の例の代入が行われると，「p は x を指している」または「p は x へのポインタを保持している」と言う。

例）p = &x;

p が x を指しているとき，**間接演算子** * を p に付けて *p とすると，p が指している領域（x）を表す。このとき，*p への代入は x への代入，*p の参照は x の参照となる。

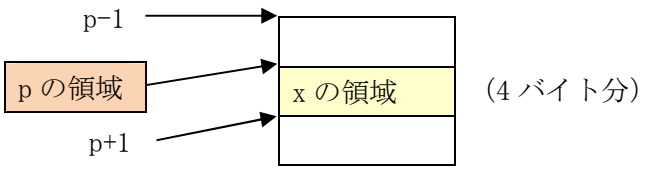
代入の例）*p = 100;　　/*　p 及び x が int 型の場合（x = 100; と同じ働き）　*/
参照の例）printf("p の指す先に格納されているデータは%d¥n", *p);

ポインタ演算と１次元配列

例題 2 で確認するが，32 ビット CPU に対応した C 言語のコンパイラの場合には，ポインタ変数はメインメモリ上で 4 バイト分の領域を占める。int 型変数の場合にもメインメモリ上で 4 バイト分の領域を占めるが，ポインタ変数に格納されているアドレスデータが int 型変数に格納されているデータと同一形式というわけではない。

しかしながら，ポインタ変数に int 型の値を加算または減算することができ，その結果は単にアドレスに整数値を加算または減算したものとは全く異なる意味合いを持つ。例えば，ポインタ変数 p に対して，p+1，p-1 等を**ポインタ演算**と呼び，p が指しているデータのデータ型によって演算結果が指すアドレスが異なる。

例）int x;
　　int *p;
　　p = &x;



の場合，p+1 が指す先は p が指す先の 4 バイト後ろのアドレスとなる。

また，ポインタ演算の結果がアドレスとなることから，演算結果をポインタ変数に代入することも可能である。これは例えば，ポインタ変数 p に対して，p++，p-- 等が可能であることを意味している。

但し，ポインタ演算は指しているデータの格納領域の範囲に注意して用いないと，大変危険で

ある。例えば，ポインタ演算のうち，減算によって変数の格納領域を超えてプログラムの格納領域を書き換えてしまったり，甚だしい場合には，システム領域にアクセスしてしまったりする場合がある。UNIX 系の OS であればユーザプログラムの一部が壊れて暴走しても，そのプロセスだけを切り捨てられるメモリ保護機能があるが，OS によってはメモリ保護が不完全なものがあり，ユーザプログラムの暴走はシステムダウンを引き起こす場合がある。

ポインタ演算が用いられる典型的な例としては，ポインタ変数が配列を指す場合が挙げられる。配列名が配列の 0 番の要素の先頭アドレスを表すことから，これをポインタに代入して，i 番の要素の領域を ポインタ変数名 + i で指すことで，for 文等に於いて便利な利用が可能となる。

例）
```
int a[10];
int *p;
p = a;        /* 配列名 a は&a[0]を表す。*/
```
⇒ p+i は a[i]を指す。

例題１（**ポインタとアドレスとの関係**）

次のソースプログラムをテキストエディタで入力して，prog9-1.c の名前を付けて保存する。それを翻訳・編集して実行形式のファイルを作成し，実行せよ。

```
/* prog9-1.c */
#include <stdio.h>

int main(void)
{
    char c;
    char *pc;
    int x;
    int *p1, *p2;

    pc=&c;
    *pc='z';

    p1 = &x;
    *p1 = 80;

    p2 = p1;

    printf(" c に格納された文字は%c¥n", c);
    printf(" c の値(文字コード)は%02x¥n", c);
    printf("&c の値は%#x¥n", &c);
    printf("pc の値は%p¥n", pc);
    printf("pc-1 の値は%p¥n", pc-1);
    printf("pc+1 の値は%p¥n", pc+1);
    printf("pc の指す先に格納された文字は%c¥n¥n", *pc);

    printf(" x の値は%d¥n", x);
    printf("&x の値は%#x¥n", &x);
    printf("p1 の値は%p¥n", p1);
    printf("p1-1 の値は%p¥n", p1-1);
```

```
    printf("p1+1 の値は%p¥n", p1+1);
    printf("p1 の指す先の値は%d¥n", *p1);
    printf("p2 の値は%p¥n", p2);
    printf("p2 の指す先の値は%d¥n", *p2);
    return 0;
}
```

【解説】

1. printf 中の変換指定 %#x はアドレスを 16 進数表記（0x から始まる）に変換する。なお，%p は void へのポインタ（あらゆる型へのポインタ）実引数の値を印字可能文字列に変換する。

2. ポインタ p1 の参照はアドレスを得るので，別のポインタ p2 への代入が可能である。&x は定数なので，これに他のアドレスを代入することはできない。

例題 2（**ポインタ演算と 1 次元配列の要素との関係**）

次のソースプログラムをテキストエディタで入力して，prog9-2.c の名前を付けて保存する。それを翻訳・編集して実行形式のファイルを作成し，実行せよ。

```
/* prog9-2.c */
#include <stdio.h>
#define  MAX  5

int main(void)
{
    int i, x[MAX];
    int *p;

    p = x;

    for (i=0; i<MAX; i++)
    {
        *(p+i) = i+1;
        printf("  x[%d]の値は%d¥n", i, x[i]);
        printf("*(p+%d)の値は%d¥n", i, *(p+i));
        printf(" &x[%d]の値は%p¥n", i, &x[i]);
        printf("  p+%d の値は%p¥n", i, p+i);
    }

    return 0;
}
```

【解説】*(p+i)はポインタ演算 p+i の結果が指す先，即ちこの例では配列 x の i 番の要素がメインメモリ上で占める領域の中身を表す。従って，*(p+i)=i+1; は x[i]=i+1; と同じ働きをする。

注）*p+i は*p を参照してその値に i を加算するので，この例では x[0]+i と同じ働きをする。

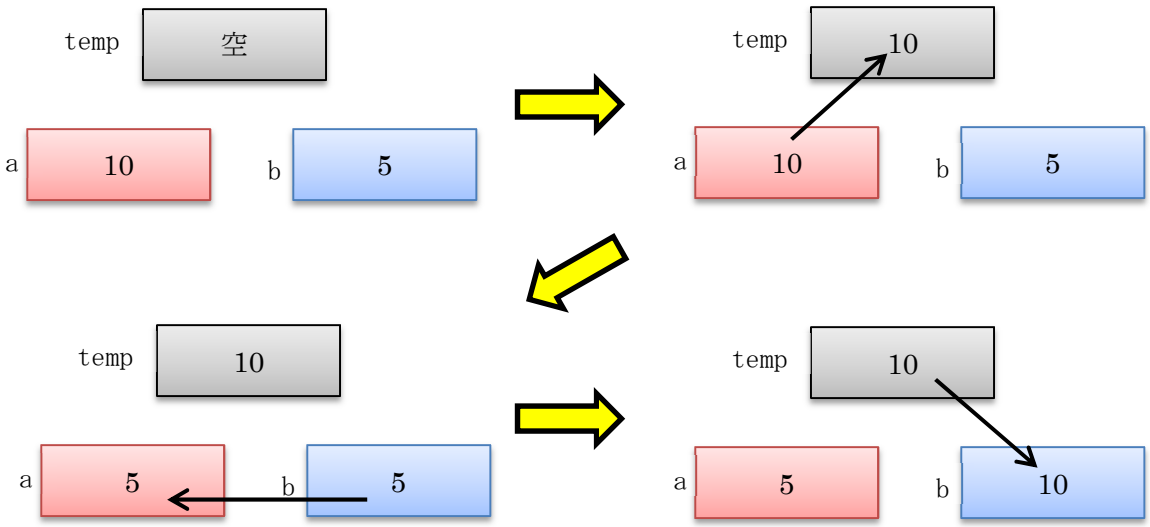
例題３（**ポインタを引数とするユーザ定義関数**）

次のソースプログラムをテキストエディタで入力して，prog9-3.c の名前を付けて保存する。それを翻訳・編集して実行形式のファイルを作成し，実行せよ。

```
/* prog9-3.c */
#include <stdio.h>

void swap1(int, int);
void swap2(int *, int *);

int main(void)
{
    int x, y;

    printf("x の値を整数で入力してください：");
    scanf("%d", &x);

    printf("y の値を整数で入力してください：");
    scanf("%d", &y);

    swap1(x, y);
    printf("swap1 を実行した結果，x の値は%d，y の値は%d¥n", x, y);

    swap2(&x, &y);
    printf("swap2 を実行した結果，x の値は%d，y の値は%d¥n", x, y);

    return 0;
}

void swap1(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}

void swap2(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}
```

【解説】関数の引数には変数の値が渡される。swap1 では呼び出し側の実引数 x，y の値を仮引数 a，b に受け渡した後，a，b の値を関数内で入れ替えるが，元の x，y の内容に変更は生じない。それに対して，swap2 では引数にポインタを用いていて，アドレスの値が渡されるため，ポインタが指す先の変数の領域を操作することが可能となる。

【入れ替え(swap)】



演習１

例題３で作成した swap2 を利用して，二つのデータを降順に並べ替える関数 sort0 を作成し，この sort0 を利用して 10 個のデータをソートする（並べ替える）プログラムの空欄 1)　　，2)　　，3)　　を埋めてソースプログラムを完成させ，テキストエディタで入力して，ex9-1.c の名前を付けて保存する。それを翻訳・編集して実行形式のファイルを作成し，実行せよ。

```
/* ex9-1.c */
#include <stdio.h>
#define 1)          10

void swap2(int *, int *);
void sort0(2)          );

int main(void)
{
    int i, j, data[KOSUU], *p1, *p2;

    for (i=0; i<KOSUU; i++)
    {
        printf("%2d 番目の整数データを入力してください： ", i+1);
        scanf("%d", &data[i]);
    }
```

```
    printf("入力したデータの並び¥n[");

    for (i=0; i<KOSUU-1; i++)
        printf("%d, ", data[i]);

    printf("%d]¥n", data[KOSUU-1]);

    for (i=0; i<KOSUU-1; i++)
        for (j=i+1; j<KOSUU; j++)
        {
            p1=&data[i];
            p2=&data[j];

            3)          (p1,p2);
        }

    printf("処理後のデータの並び¥n[");

    for (i=0; i<KOSUU-1; i++)
        printf("%d, ", data[i]);

    printf("%d]¥n", data[KOSUU-1]);

    return 0;
}

void swap2(int *a, int *b)
{
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;
}

void sort0(int *d1, int *d2)
{
    if (*d1 < *d2)
        swap2(d1, d2);
}
```

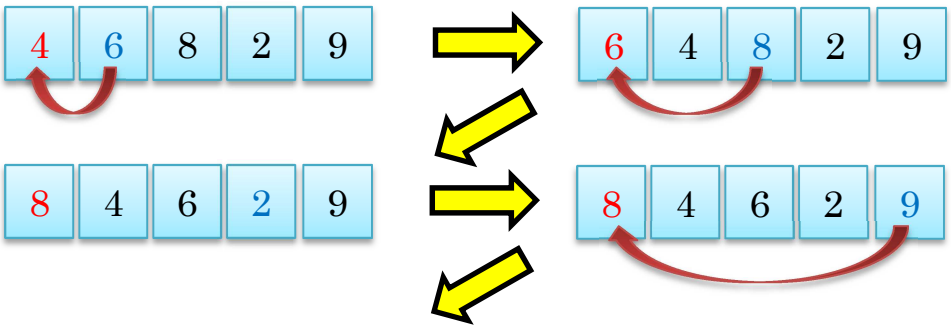
提出物：

1) 例題 1，2，3 の出力結果をコピーして貼り付けたテキストファイル res9.txt

2) 演習 1 のソースプログラムのファイル ex9-1.c の完成版

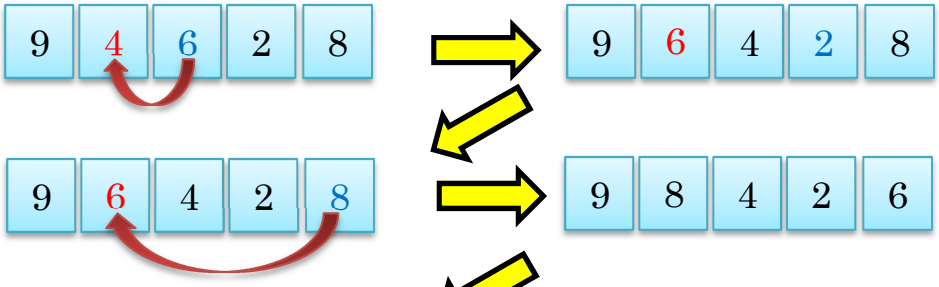
【並べ替え(sort)　データが５個の場合】

ｉの値：0

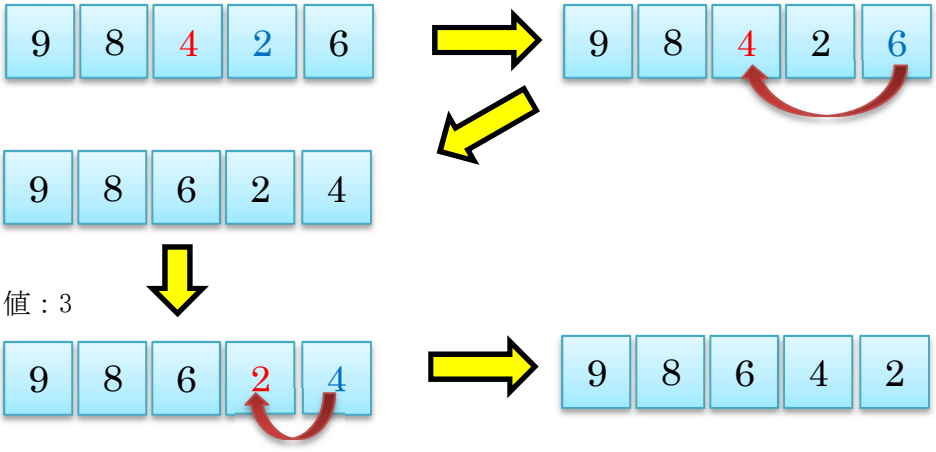


外側の for ループで i が 0 の時，内側の for ループを一通り実行し終わると，最大のものが最初に来る。

ｉの値：1



ｉの値：2



ｉの値：3