12 ポインタ (続き)

課題 8. 本節冒頭の実習 12.1 を修正して、int 型変数 2 つの内容を交換する関数 swap() を正しく作って書換えよ。(動作を確認した上で、何故そういう動作になるか理解する。)

解答例 8.1. 内容を入れ替えたい変数のアドレスを渡す。

```
/*   swap.c   2010-06-07   */
#include <stdio.h>

void swap( int *, int * );

int main(int argc, char** argv)
{
    int x = 2010, y = 607;

    printf("x = %d, y = %d\n", x, y);

    swap(&x, &y);

    printf("x = %d, y = %d\n", x, y);
}

void swap( int *a, int *b )
{
    int c;

    c = *a;
    *a = *b;
    *b = c;

    return;
}
```

- 関数 swap() の引数として渡された値は何か。
- 関数 swap() 内のそれぞれの文で代入された値は何か。
- 変数 x, y の値が書き換えられたのは、どの文の実行時か。

Newton-Raphson 法の課題については、提出状況に鑑み、解説を無期限延期とする。既に提出した人も、前回指摘のチェックポイントを確かめて、必要なら再提出せよ。

ポインタの値は変数の場所を示すので、「ここととここととを入れ替えて」という感じ。

値の入替えは代入 3 回で出来る。
基本手筋として頭に留めておこう。

*&c は、定義により c と完全に同じ。

尚、&*p はあり得ません。念の為。

13 配列

13–1 配列の利用

大量のデータを扱いたい時など、同じ型の変数を添字付きで沢山使いたい時がある。C 言語 (をはじめ多くのプログラム言語) では、添字付きの変数 (**配列**) を利用する仕組みが備わっている。

配列: array
添字: index

配列は個々の変数の型と変数の個数 (配列の大きさ) とを指定して「**〜型変数○個の配列**」として宣言される。配列が宣言されると、プログラムはメモリ上に必要な分の**連続的な**領域を確保する。配列の宣言の書式は次の通り。

> **型名　配列名［要素数］;**

配列の扱い方に関する基本的な注意を幾つか。

- 配列の要素数は宣言時に指定しなければならない (プログラムの途中では変更できない)。前もって確定できない場合には余裕をみて多めに確保しておく必要がある場合もある。
- 配列の要素の添字は 0 から始まる。従って、添字として有効な値は 0 **から** (**要素数** − 1) **まで**。

  例 . 次の宣言
  `int a[10];`
  で、`int` 型変数 10 個 `a[0]`, `a[1]`, . . ., `a[9]` が宣言される。`a[10]` は不可。

- プログラム内でもし、宣言して確保した要素の上限を超えた要素を参照したとしても、コンパイラはチェックを行わ**ない**。実行形式は生成されるが、動作は保証されず、実行中に他の変数の領域を破壊してしまう恐れがある。十分に注意すべし。

実行時に変数の値が範囲内に収まっているかどうか、コンパイラには判らない。

ということは、配列で確保した変数の個数や添字の範囲については、使う側で管理しなくてはいけないということである。

実習 13.1. 20 個の要素を持つ int 型配列 a[] を宣言し、i 番目の要素 a[i] に i^2 を代入した後、変数 a[i] の値を表示しつつ総和を求める。

```
/*   array.c   2010-06-14   */
#include <stdio.h>

#define VALUES 20

int main(int argc, char** argv)
{
    int a[VALUES], i, sum=0;

    for( i=0; i<VALUES; i++ )
    {
        a[i] = i*i;
    }
    for( i=0; i<VALUES; i++ )
    {
        printf("%3d: %5d\n",i,a[i]);
        sum += a[i];
    }
    printf("sum = %d\n",sum);
}
```

考察 13.1.1. 配列 a[] の添字の範囲が正しい範囲に収まっていることを確認せよ。

考察 13.1.2. 20 個の int 型変数 a[0], a[1], ..., a[19] の為の変数領域が、メモリ上の**連続的な**領域に確保されていることを確かめよ。

考察 13.1.3. #define によるマクロ定義の効果について考えよ。

勿論、平方和を求めるだけなら配列に代入する必要はないが……、ここは配列の練習という事で。

配列の要素数は宣言時に指定しなければならないので、このプログラムで、実行中に上限の値を入力してその個数分の配列を用意するようには出来ない。予め大きめに確保しておいて、その範囲内の入力に対応することなら出来る。実行中に必要に応じて必要なだけ変数領域を確保する (動的確保という) 方法はあるが、それは追い追い (後期の情報処理 IV で採り上げる予定)。

実習 13.2. 10 個の (疑似) 乱数を発生させ、それを配列 a[] に格納した後、その値を表示する。更に、最小値を取る要素の添字 (およびそのときの値) を表示する。

```
/*   rand.c   2010-06-14   */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define VALUES 10

int main(int argc, char** argv)
{
    int a[VALUES], i, min_i = 0;
    time_t t;

    srand((unsigned int)time(&t));

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

    for( i=0; i<VALUES; i++ )
    {
        printf("%2d: %12d\n", i, a[i]);
    }

    最小値を取る要素の添字を求める

    printf("\nminimum: a[%d]=%d\n", min_i, a[min_i]);
}
```

考察 13.2.1. 比較するには値が 2 つ必要なので、最初の比較では例外処理が必要になることがある。最小値を取る要素の添字を覚えておく変数 (上の例では min_i) の値の初期化のしかたに注意して、スマートに処理せよ。

参考: Mersenne Twister (http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html)

乱数とはでたらめな数列のこと。疑似乱数とは或る規則に基づいて発生させた乱数っぽい数列のこと (でたらめさ・乱数っぽさは数学的に定式化されている)。

srand() は疑似乱数の初期値 (種, seed) の設定。同じ値を種とすると同じ数列を生成するので、ここではシステムの時計を利用して毎回違う値を種に取るようにしている。rand() は呼ばれる度に疑似乱数を生成して値を返す。値は 0 以上 RAND_MAX 以下の整数値で、センターの環境では RAND_MAX $= 2^{16} - 1 = 32767$。stdlib.h は srand(), rand() を使うのに必要。time.h はシステムから (暦) 時間を取る関数 time() を使うのに必要。

(疑似) 乱数の発生法は色々な場面で利用できるだろう (ゲームとか)。

本当はここで用意されている関数 rand() よりも性質の良い (よりでたらめな) 疑似乱数の生成法がある。シミュレーション実験などでは良い疑似乱数を用いてデータの偏りを防がなければならない。
← 参考: 東京大学の松本眞さんの Mersenne Twister のページ

13–2　配列名を引数として関数に渡す

配列名は変数名ではないので、値を代入することは出来ないが、式として用いることが出来る。配列名を式中で用いると、確保されたメモリ領域の先頭アドレスを値とするポインタ型の式として評価される。例えば、int 型配列 a[] については、配列名 a は int へのポインタ型の式であって、

a == 配列 a[] の先頭アドレス == int 型変数 a[0] の先頭アドレス == &a[0]

という関係が成り立っている。

さて、配列そのもの (全体) を引数として関数に渡す場合は、実際には配列名を引数として渡すことで実現する。配列名は (式としては) 配列の先頭アドレスを値とするポインタ型の式なので、関数側ではポインタ変数でそれを受ける。関数内ではそのポインタ変数に [] で添字を付けて使えば良い。

ポインタ変数に [] で添字を付けちゃって良い理由は次回に。

この時に注意すべきは、その値を受け取ったポインタ変数は呼び出した側の配列と同じアドレスを指している (つまり要するにポインタ渡しをしている) ということで、従って、受け取った関数側で配列の要素の値を変更すると、呼び出した側の元の配列の値を変更していることになる。ポインタ型変数を使った覚えも、アドレス演算子 & を使った覚えもなくても、実質的には関数の引数としてポインタ渡しをしていることに、特に注意しておこう。

配列の全要素をコピーして渡すのは効率が良くないので、こういう仕様にしたのであろう。

従って、関数の引数として配列名を渡した場合、関数の中で配列の要素の値を変更すると、呼出側でも渡した配列の要素の値が変更されてしまうことに、特に注意せよ。

注． ポインタ型の式には整数を加減することが出来る。この演算は、その型の大きさ sizeof(*type*) を単位として行なわれる。例えば、p をポインタとするとき、p+k は p の指している領域からその型の変数 k 個分先を指すポインタであり、番地の計算としては「(p の指すアドレス) + sizeof(int) $*$ k」となる。

一方、配列 a[] の添字 k の要素 a[k] の先頭アドレスは「(配列 a[] の先頭アドレス) + sizeof(*type*) $*$ k」なので、配列名 a がポインタ型の式であることから、

a + k == &a[k]

となり、そこの内容を間接演算子 * で取ると、

*(a + k) == a[k]

となる。実は C 言語では、配列の記号で書いたものは、ポインタの形に置き換えられてコンパイルされている。

とは言え (と言うか、だからこそ)、配列らしい所は配列の記号で判り易く書いておいてよろしい。

実習 13.3. rand.c で、乱数による配列の要素の値の設定の部分や、配列の要素の値の表示の部分を、関数化してみよ。

課題 9 (〆切 6/20(日)). rand.c の「最小値を取る要素の添字を求める」部分を min_index() として関数化し、それを大きさの異なる 2 つの配列に適用して、最小値を取る要素の添字とその値とを表示せよ。(出来れば、乱数による配列の要素の値の設定の部分や、配列の要素の値の表示の部分をも関数化せよ。)

関数プロトタイプの設計からが問題。

```
/*   rand2.c   2010-06-14   */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define VALUES1 10
#define VALUES2 20


    関数 min_index() のプロトタイプ宣言


int main(int argc, char** argv)
{
    int a[VALUES1], b[VALUES2], i;
    time_t t;

    必要なら、最小値を取る要素の添字を覚えておく変数などの宣言


    srand((unsigned int)time(&t));

    for( i=0; i<VALUES1; i++ )
    {
        a[i] = rand();
    }
    for( i=0; i<VALUES1; i++ )
    {
        printf("%2d: %12d\n", i, a[i]);
    }

    for( i=0; i<VALUES2; i++ )
    {
        b[i] = rand();
    }
    for( i=0; i<VALUES2; i++ )
    {
        printf("%2d: %12d\n", i, b[i]);
    }

    最小値を取る要素の添字を求め、その添字と要素の値とを表示

}

    関数 min_index() の本体
```

演習 3. 上で作った関数 min_index() を用いて、配列の要素を昇順 (小さい順) に並べ替える関数 arraysort() を作り、動作例のプログラムを書いて動作を確認せよ。