13 配列 (続き)

課題 9. 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。

解答例 9.1. 関数 min_index() で最小値を取る要素を探すには、値が格納されている場所の先頭アドレスと、そこから幾つの要素を比較するかという個数とが必要である。

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。

```
/*   rand2a.c   2010-06-14   */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define VALUES1 10
#define VALUES2 20

int min_index( int *, int );

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

    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]);
    }
    printf("\n");
```

↗

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

    min_i = min_index(a,VALUES1);
    printf("minimum: a[%d]=%d\n", min_i, a[min_i]);

    min_i = min_index(b,VALUES2);
    printf("minimum: b[%d]=%d\n", min_i, b[min_i]);
}

int min_index(int *p,int n)
{
    int i, min_i=0;

    for( i=1; i<n; i++ )
    {
        if( p[i] < p[min_i] )
            min_i = i;
    }

    return min_i;
}
```

解答例 9.2. 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した。ここでも値の格納場所の先頭アドレスの他に、要素の個数を関数に渡す必要がある。

setvalues() で、アドレス演算子 & を用いていないのに、関数内で配列要素の値が書き変わっていることに注意。

```
/*   rand2b.c   2010-06-14   */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define VALUES1 10
#define VALUES2 20

void setvalues( int *, int );
void printarray( int *, int );
int min_index( int *, int );

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

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

    setvalues(a,VALUES1);
    printarray(a,VALUES1);
    setvalues(b,VALUES2);
    printarray(b,VALUES2);

    i=min_index(a,VALUES1);
    printf("minimum: a[%d]=%d\n", i, a[i]);
    i=min_index(b,VALUES2);
    printf("minimum: b[%d]=%d\n", i, b[i]);
}
```

```
void setvalues(int *p,int n)
{
    int i;

    for( i=0; i<n; i++ )
    {
        p[i] = rand();
    }

    return;
}

void printarray(int *p,int n)
{
    int i;

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

    return;
}
```

関数 min_index() の定義は同じなので省略。

解答例 9.3. 返値として添字を返すのではなく、結果を格納すべき変数のアドレスを渡して、関数の中でその変数に書込んでもらってくるという手もある。

関数は 1 つ以下の値しか返せないので、呼び出した側に 2 つ以上の値を返したい場合に有効。折角なので、ここでは添字と最小値とをもらってみた。

この場合も何か値を返しても良い。通常は値が返ってくる方が式中でも使えたりして便利。実行ステータス (実行に成功したか失敗したか) などを返させることも多い。

```
/*   rand2c.c   2010-06-14   */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define VALUES1 10
#define VALUES2 20

void setvalues( int *, int );
void printarray( int *, int );
void min_index( int *, int, int *, int *);

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

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

    setvalues(a,VALUES1);
    printarray(a,VALUES1);
    setvalues(b,VALUES2);
    printarray(b,VALUES2);

    min_index(a,VALUES1,&min_i,&min);
    printf("minimum: a[%d]=%d\n", min_i, min);
    min_index(b,VALUES2,&min_i,&min);
    printf("minimum: b[%d]=%d\n", min_i, min);
}
```

```
void min_index(int *p,int n,int *mi,int *m)
{
    int i;

    *mi = 0;
    *m = p[*mi];

    for( i=1; i<n; i++ )
    {
        if( p[i] < p[*mi] )
        {
            *mi = i;
            *m = p[i];
        }
    }

    return;
}
```

関数 `setvalues()`, `printarray()` の定義は同じなので省略。

解答例 9.4. 折衷案。添字は返値として返させる一方、最小値を格納すべき変数のアドレスを渡して、関数の中でついでに、その変数に最小値そのものを書込んでもらってみた。

```
/*   rand2d.c   2010-06-14   */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define VALUES1 10
#define VALUES2 20

void setvalues( int *, int );
void printarray( int *, int );
int min_index( int *, int, int * );

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

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

    setvalues(a,VALUES1);
    printarray(a,VALUES1);
    setvalues(b,VALUES2);
    printarray(b,VALUES2);

    min_i=min_index(a,VALUES1,&min);
    printf("minimum: a[%d]=%d\n", min_i, min);
    min_i=min_index(b,VALUES2,&min);
    printf("minimum: b[%d]=%d\n", min_i, min);
}
```

```
int min_index(int *p,int n,int *m)
{
    int i, min_i=0;

    for( i=1; i<n; i++ )
    {
        if( p[i] < p[min_i] )
            min_i = i;
    }
    *m = p[min_i];

    return min_i;
}
```

関数 setvalues(), printarray() の定義は同じなので省略。

14　多次元配列

配列を使えばベクトルが自然に扱えるが、行列を扱うには添字を 2 つ取る必要がある。C 言語では、このような多次元配列を「配列の配列」として扱う。例えば、行列のような 2 次元配列は次のように宣言すれば良い。

> **型名　配列名**[**要素数** 1][**要素数** 2];

実習 14.1. 2 次元配列を用いて、「九九」の表を作ってみる。

```
/*   kuku.c   2010-06-21   */
#include <stdio.h>

#define BASE 10

int main(int argc, char** argv)
{
    int a[BASE][BASE], i, j;

    for( i=0; i<BASE; i++ )
        for( j=0; j<BASE; j++ )
        {
            a[i][j] = i*j;
        }

    for( i=0; i<BASE; i++ )
    {
        for( j=0; j<BASE; j++ )
        {
            printf("%3d",a[i][j]);
        }
        printf("\n");
    }
}
```

考察 14.1.1. 変数 a[i][j] はどのようにメモリ上に配置されているか。

実習 14.2. ちょっと手を入れて、十六進版の「九九」(「FF」？) の表を作ってみよ。

従って、冷静に考えれば新たな文法事項ではなく、前節の内容の応用・組合せである。

配列名[**要素数** 1, **要素数** 2] では**ない**。

一つ目の for loop では、loop の中身が

```
for( j=0; j<BASE; j++ )
{
    a[i][j] = i*j;
}
```

の一文のみなので、{ } で括るのを省略した。内側 (上記) の { } も省略可だが、この場合は 2 重の for loop で i, j を全て走らせて、その中身が

```
    a[i][j] = i*j;
```

という感じなので、左の形でも解り易いだろう。

十六進表示の printf 変換指定は x で、これを使う時は unsigned int (符号無し整数型) を用いるのが良い。

実習 14.3. 3×3 行列 a,b を 2 次元配列を用いて扱い、行列の掛け算を計算してみる。

```
/*   matrix.c   2010-06-21   */
#include <stdio.h>

#define SIZE 3

int main(int argc, char** argv)
{
    int a[SIZE][SIZE], b[SIZE][SIZE], c[SIZE][SIZE], i, j, k;

    printf("a=\n");
    for( i=0; i<SIZE; i++ )
    {
        for( j=0; j<SIZE; j++ )
        {
            a[i][j] = i+j;
            printf("%4d",a[i][j]);
        }
        printf("\n");
    }

    行列 b についても同様に成分の設定・表示

    行列 a, b の積を計算し、行列 c に代入

    printf("a*b=\n");
    for( i=0; i<SIZE; i++ )
    {
        for( j=0; j<SIZE; j++ )
        {
            printf("%4d",c[i][j]);
        }
        printf("\n");
    }
}
```

行列 a, b の成分は適当に与えよ。(ここでは行列 a の成分は例として

```
a[i][j] = i*j;
```

と与えてあるが、好きなもので良い。)

勿論、成分を入力する仕様にしても良い。入力すべき成分の個数が多いので、ここではデータファイルを用意して、入力リダイレクションなどを利用してプログラムに与えるのが良かろう。

表示はもっと工夫した方が見易いかも知れないが、出力リダイレクションなどでファイルに保存して、次の計算の入力データにすることも念頭に置いている。

さて、前節と同様に、ここでも当然、行列の基本的演算は関数化しておきたい所であるが、多次元配列の場合は通常の (1 次元の) 配列の時にはなかった問題がある。

C 言語の多次元配列はあくまでも「配列の配列」であって、例えば行列を扱う為に

```
int a[ROWS][COLUMNS];
```

と宣言した場合、

a[i] は「int 型 COLUMNS 個の配列」の名前

であり、

a は「「int 型 COLUMNS 個の配列」ROWS 個の配列」の名前

である。a[i] の先頭アドレスは、a の先頭アドレスから数えて「int 型 COLUMNS 個の配列」i 個分先、即ち、int 型 ($i *$ COLUMNS) 個分先になる。従って、a[i][j] の先頭アドレスは、a の先頭アドレスから数えて int 型 ($i *$ COLUMNS $+ j$) 個分先になる。i, j のみでなく、行列の列数 COLUMNS にも依っていることに注意。

1 次元配列では、配列全体の先頭アドレス (それは式としての配列名の評価値であった) と個々のデータ (要素) の型 (大きさ) とが判れば、(配列のサイズを知らずとも) i 番目の要素の先頭アドレスが定まったが、多次元配列では、(i, j) 番目の要素 a[i][j] の先頭アドレスは、配列全体の先頭アドレスと個々のデータ (要素) の型 (大きさ) とだけでは判らず、(最も左の添字以外の) 配列のサイズに依存する。

int a[ROWS][COLUMNS]; と宣言された行列を関数に渡す時に、呼出側では *functionname*(a) と配列名で渡したとして、関数のプロトタイプとしてどんな型で受ければ良いかと考える。配列名 a は「int 型 COLUMNS 個の配列」の先頭アドレスなので、受けるべき型は「「int 型 COLUMNS 個の配列」へのポインタ型」となる。即ち、プロトタイプを書く時に COLUMNS の値が必要なのである。

1 次元配列では配列のサイズを引数として渡すことにより、異なる大きさ (要素数) の配列に対応できる関数を書くことが出来たが、多次元配列では (最も左の添字以外は) コンパイル時に要素数が確定していなければならない。

という訳で、いろいろ厄介なので、ここでは異なるサイズの行列に対応できる関数を書くのはあきらめて、サイズを固定して関数化することにしよう。それなら簡単である。

課題 10 (〆切 6/27(日)). 3×3 行列に関する基本的な関数 (表示・演算など) を作成して実習 14.3 のプログラムを書き直した次頁のプログラム matrix2.c を完成させよ。(補足事項は次頁傍注に。)

行 (横の並び): row
列 (縦の並び): column

実際、内部的にはこのようなアドレス計算をしている。a[i][j] はコンパイル時に*(*(a+i)+j) に置き換えられるが、*(a+i) は、a の先頭アドレスから数えて「int 型 COLUMNS 個の配列」i 個分先のアドレス (を値とする int * 型の式) であって、コンパイル時に第 2 添字の個数 COLUMNS を知っている必要がある。そうでないと実際に何 byte 先なのか判らない。

実際には「「int 型 COLUMNS 個の配列」の配列」の形でプロトタイプを書くことも出来る。次頁の例を参照。

```
/*   matrix2.c   2010-06-21   */
#include <stdio.h>

#define SIZE 3

void printmatrix(int [SIZE][SIZE]);
void multmatrix(int [SIZE][SIZE],int [SIZE][SIZE],int [SIZE][SIZE]);

int main(int argc, char** argv)
{
    int a[SIZE][SIZE], b[SIZE][SIZE], c[SIZE][SIZE];
    int i, j, k;

    for( i=0; i<SIZE; i++ )
        for( j=0; j<SIZE; j++ )
        {
            a[i][j] = i+j;
        }
    printf("a=\n"); printmatrix(a);

    行列 b についても同様に成分の設定・表示

    multmatrix(a,b,c);

    printf("a*b=\n"); printmatrix(c);
}

void printmatrix(int x[SIZE][SIZE])
{

    行列 x の内容を表示する関数 printmatrix() の定義の中身

}

void multmatrix(int x[SIZE][SIZE],int y[SIZE][SIZE],int z[SIZE][SIZE])
{

    行列 x, y の積を計算して z に格納する関数 multmatrix() の定義の中身

}
```

本課題に関する補足:

数学的に面白い例を求む。

そのために、

- 行列の成分は左記の例にこだわらず適当に決めて良し。入力を受付ける形にしても良し。
- 演算内容も積だけにこだわらず適当に増やして良し。加減算だけでは不十分。
- 行列のサイズも適当に決めて良し。但し 3 以上であること。
- int 型でなくて double 型でも良し。
- 期限内に提出した後の機能拡張版の提出については、期限にこだわらず試みられたい。