CS351 Programming Assignment 4

Due: October 21, 2014

This assignment will help you in gaining better understanding to the concept of pointers and arrays in C. You are to write a C program to perform matrix operations including input, output, addition, and multiplication for matrices of <u>any dimensions</u>. In this assignment we consider matrices of integers only. A matrix in C can be declared as an array of arrays, such as:

```
int x[5][8];        /*  declare x as a 5x8 matrix  */
```

In this case, a matrix operation, such as print, can be declared as follows:

```
void print(int x[5][8]);
```

Note that the compiler will ignore the first dimension 5, and it needs the second dimension 8 to build the correct storage mapping function. (**WHY?**) In this way, the function can only be used to print matrices that have 8 columns. Such a function is not very useful for two reasons:

(1) The function doesn't know the first dimension.

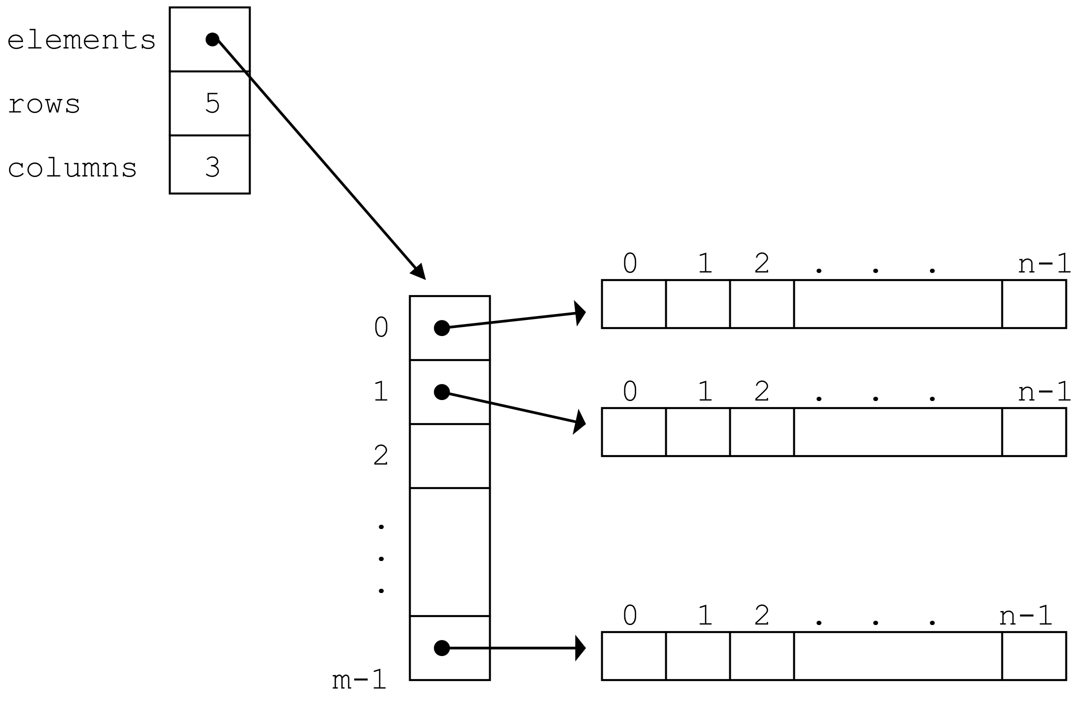
(2) The function cannot print any matrix whose column number is not 8.

In this assignment. we want to write matrix operations that can be used for matrices of any dimensions. To do so, we declare a matrix type as the following **structure**:

```
typedef struct
{
     int **elements; /* element is a pointer, what it points
                        to is a pointer that points to an int */
     int rows;       /* number of rows of the matrix        */
     int columns;    /* number of columns of the matrix     */

} matrix;   /*  matrix is the struct type name  */
```

Therefore a matrix variable contains 3 components:



Use this struct definition in your program. Your program should implement the following 5 matrix operations:

```
void read_matrix( matrix *a);       /*  Why "*" here?  */
     /* Read a matrix from the keyboard, this function should
        prompt the user to enter the dimensions, then the
        matrix entries.  Note that to practice parameter
        passing, we return the matrix through a parameter.  */

void print(matrix a);
     /*  output the matrix  */

matrix add(matrix a, matrix b);
     /*  return the sum of matrices a and b: a + b  */

matrix subtract (matrix a, matrix b);
     /*  return the subtraction of matrices a and b: a - b  */

matrix multiply(matrix a, matrix b);
     /*  return the product of matrices a and b: a * b  */
```

Note that

(1) ijth entry of the matrix "elements", which is a pointer to an int pointer, can be accessed by either

elements[i][j]

or ***(*(elements+i)+j)** /* **WHY?** */

(2) If the dimensions of a and b are different, the sum a+b and difference a - b can not be performed. If the number of columns of a and the number of rows of b are different, the product a * b can not be performed. In these case, the function should return a matrix with a **NULL** (not null as in Java) pointer assigned to elements, and 0 assigned to its rows and columns, indicating that the operation is not possible. In this case, the caller can check the elements pointer or the number of rows or columns to see whether the operation was successful.

(3) You must dynamically allocate storage space before you can put any value in the matrix. To allocate dynamic storage for such a matrix a, you may use the following piece of code:

```
a.elements = (int **) calloc(m, sizeof(int *));

for (i = 0; i < m; ++i)
     elements [i] = (int *) calloc(n, sizeof(int));
```

where m is the number of rows and n is the number of columns of the matrix.

calloc is a function declared **stdlib.h** header file that allows you to allocate a contiguous block of memory in the dynamic storage area.

(4) You should run your program with matrices of various dimensions

(5) You should run your program with matrices that operations are not possible, such as adding a 3 x 4 matrix with a 5 x 5 matrix, or multiplying a 3x4 matrix with another 3x4 matrix.

(6) You should use the notation **elements[i][j]** as well as ***(*(elements+i)+j)** to access the ith entry of a matrix in your program.

Recall that if A is a matrix of dimensions $m \times n$, and B is a matrix of dimensions $n \times p$, then $A * B$ is a matrix C of dimensions $m \times p$, and in this case,

$$C_{ij} = \sum_{k=1}^{p} A_{ik} * B_{kj}$$

where A_{ik} = ik-th entry of matrix A

B_{kj} = kj-th entry of matrix B

and C_{ij} = ij-th entry of matrix C

That is, C_{ij} = dot product of the ith row of A and the jth column of B.