Object-Oriented Programming – SPRING 2014

(BS-IT-F13 Morning & Afternoon)

Assignment # 2

Submission Deadline: **Thursday, 1ST May, 2014 (till 6:30 PM)**

Submission Folders:

**\\printsrv\Teacher Data\Muhammad Farooq\OOP - Fall 2013\ Morning**

**\\printsrv\Teacher Data\Muhammad Farooq\OOP - Fall 2013\ Afternoon**

**Instructions**

* **This is an individual assignment. You are NOT allowed to work/submit in form of group. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating would result in an “F” grade in this course.**
* Do **NOT** copy even a single line of code from any other person or book or Internet or any other source.
* This assignment needs to be submitted in **soft form**. See **Submission Procedure** at the end.
* Create a **multi-file project** for this assignment.
* Clearly mention your **Name**, **Roll Number** and **Section** in comments at the top of each file in your project.

In this assignment you are required to develop a C++ class (**TwoDMatrix**) for storing and manipulating 2-dimensional matrices of integers. There will be three data members of this class:

* A pointer to pointer of type int (through it you will allocate the 2-D array of integers dynamically)
* An integer to store the number of rows of the matrix
* An integer to store the number of columns of the matrix

Your class should support the following operations (public member functions):

1. A default constructor for creating a 5x5 matrix of integers. Each element of the matrix should be initialized to zero.
2. An overloaded constructor for creating a 2-dimensional matrix of any dimensions. The size of the matrix (number of rows and columns) will be specified through arguments. Each element of the matrix should be initialized to zero.
3. Destructor
4. Copy constructor
5. Getter functions for retrieving the number of rows and number of columns of the matrix.
6. Overload the **assignment operator =** to assign one matrix to another.
7. Overload the **stream insertion operator <<** (as a friend function) to display a matrix on screen in a neat tabular format. Use **setw** to format the output properly.
8. Overload the **stream extraction operator >>** (as a friend function) to take a matrix from the input stream.
9. Overload the **function call operator ()** to retrieve or store an integer value at a particular location in the matrix. For example, if we want to display the value store at row 0, column 4 in the matrix **m1**, we will use the following statement:

**cout << m1(0,4);**

If we want to store the value 12 at row 5, column 2 in the matrix **m2**, we will use the following statement:

**m2(5,2) = 12;**

1. Overload the **! operator** to transpose the current matrix. In this function you may need to reallocate the 2-D array which is being used to store the elements of the matrix. This operator function will have the return type void, and it will be changing the current matrix object on which it has been called.
2. Overload the **+ operator** to add a matrix with another matrix and return the resulting matrix. For example, if we want to add two matrices m1 and m2 and store the resulting matrix in m3, we will use the following statement:

**m3 = m1 + m2;**

Keep in mind that two matrices can be added only if the dimensions of both the matrices are same. This function should display an appropriate error message if the two matrices cannot be added.

1. Overload the **\* operator** to multiply two matrices and return the resulting matrix. Again keep in mind that two matrices can be multiplied only if the number of columns in the first matrix is the same as the number of rows in the second matrix. This function should also display an appropriate error message if the two matrices cannot be multiplied.
2. Overload the **\*= operator** to multiple each element of a matrix with an integer value. The current matrix will be updated.

Now, you are required to write a menu-driven program which will illustrate the working of the **TwoDMatrix** class. The starting portion of the output of your program should look like this:

|  |
| --- |
| **Reading data from "input.txt"...**  **File opened successfully and 5 matrices have been read**  **Menu**  **1. Display a matrix**  **2. Display ALL matrices**  **3. Modify an element in a matrix**  **4. Modify a whole matrix**  **5. Take transpose of a matrix**  **6. Add two matrices**  **7. Multiply two matrices**  **8. Multiply a matrix with an integer**  **9. Quit**  **Enter your choice:** |

***VERY IMPORTANT: Your program should display the above 9 options in the exactly the same order. If you have not implemented a particular menu option, display “NOT IMPLEMENTED” on screen, if the user selects that option.***

The input file (**input.txt**) will contain several matrices in the following format:

First line of the input file will contain the number of matrices (lets say *N*). After that there will be *N* matrices present in the input file. For each matrix in the input file, its dimensions (# of rows and # of columns) will be specified on one line. If *R* is the number of rows and *C* is the number of columns, the next *R* lines will contain *C* integers each, indicating the elements of the matrix.

At the start, your program should open the input file and **dynamically allocate an array of matrices**. Then, it should use the **overloaded steam extraction operator >>** to read each matrix from the file and store it at its appropriate location in the array. After that it should display the above menu and then perform appropriate operations depending upon the option(s) selected by the user.

Following tasks must be performed in each menu item:

1. **Display a matrix**

If the user selects this option, your program should ask the user which matrix he/she wants to display. The user will provide the **matrix number** as input. For example, if the user enters 0, your program should display matrix 0 on screen. Perform input validation so that if the user enters an invalid matrix number, display an error message and prompt the user again.

1. **Display ALL matrices**

Here your program should display all matrices that have been read from the file.

1. **Modify an element in a matrix**

If this option is selected by the user, your program should firstly ask the user which matrix he/she wants so modify (user will provide the **matrix number**) and then ask for the **row number** and **column number** in that matrix. Once input has been validated, your program should ask the user which **element** he/she wants to store at that location.

1. **Modify a whole matrix**

Here your program will ask the user **which matrix** he/she wants to modify. After that it will ask the user to enter elements of **each row** in the matrix i.e. firstly it will ask for the elements of row 0, then row 1 and so on.

1. **Take transpose of a matrix**

Here your program will take the **matrix number** from the user, display the matrix, transpose it, and then display the modified matrix on screen.

1. **Add two matrices**

Here your program will prompt the user to enter the **matrix # of 1st matrix** (left operand) and **matrix # of 2nd matrix** (right operand) that he/she wants to add. If the two matrices can be added, your program should ask the user the **matrix # of the destination matrix** (in which he/she wants to store the resulting matrix). If the two matrices specified by the user can not be added, your program should display an appropriate error message and ask the user for matrix #’s once again.

1. **Multiply two matrices**

Here your program will prompt the user to enter the **matrix # of 1st matrix** (left operand) and **matrix # of 2nd matrix** (right operand) that he/she wants to multiply. If the two matrices can be multiplied, your program should ask the user for the **matrix # of the destination matrix** (in which he/she wants to store the resulting matrix). If the two matrices specified by the user can not be multiplied, your program should display an appropriate error message and ask the user for matrix #’s once again.

1. **Multiply a matrix with an integer**

If the user selects this option, your program should prompt the user to enter the **matrix #** and the **integer value** that he/she wants to multiply the matrix with. After that your program should display the matrix, multiply the specified matrix with the given value, and then display the updated matrix on screen.

1. **Quit**

This option will allow the user to quit the program ☺

**Submission Procedure**

You are required to submit this assignment in soft copy format. Make sure that you follow these steps to submit your code:

1. Delete the “Debug” and “ipch” folders from your project folder.
2. Rename your project folder to your complete roll number (like BSEF11M234).
3. Compress the folder in the form of a **.RAR** file.
4. Copy the **.RAR file** (e.g. **BSEF11A567.rar**) in the submission folder **of your section**.

***Late Submission:*** You can late-submit this assignment according to the following rules. **Note that if you submit more than one copy of your assignment, only the latest submission will be marked. All previous submissions will be deleted.**

* Submission within 24 hours of the deadline will result in 25% deduction
* Submission within 48 hours of the deadline will result in 50% deduction
* Submission after 48 hours of the deadline will not be considered

|  |
| --- |
| **Please note:**   * Do NOT use any **global variables** in your programs. However, you can use global named constants. * These *good programming practices* will also have their (significant) weightage in the marking of this assignment: * Comment your code intelligently. **Uncommented code *will* not be given any credit.** * Indent your code properly. * Use meaningful variable and function names. * Use meaningful prompt lines/labels for input/output. * **Make sure that all memory is properly allocated and de-allocated. There should NOT be any dangling pointers, memory leaks, or any other type of run-time error in your program.** * If your submitted program gives an error message at the time of compilation, you will get a **ZERO** in this assignment. |

**☺ GOOD LUCK! ☺**

*Remember: Honesty always gives fruit (no matter how frightening is the consequence); and Dishonesty is always harmful (no matter how helping it may seem in a certain situation)!*