CS 2413 – Data Structures – Spring 2019 – **Project Two**
Due: 11:59 PM, February 25, 2019

For **project 2** additions from project 1: please **scroll to the very bottom** of this document to the addendum.

Description: A rectangular array of numbers is called a *matrix*. Sometimes a matrix is referred as a table, 2-dimensional array, or array of arrays. Consider the following matrix below. It has 5 rows (denoted *n*) and 8 columns (denoted *m*). As you can see the row and column numbers start with a 0.

	0	1	2	3	4	5	6	7
0	100			900		500		
1					200			300
2		400					800	
3			200					
4	1600				700			

The empty cells have the same *common value* (for example, it can be 0). The above matrix is said to be *sparse* because the total number of common values is significantly large in comparison to other values in the matrix. In the example above, we have a total of 40 values in the matrix (8 × 5), 10 non-common values, and 30 common values.

A sparse matrix can be represented using sparse matrix representation. In a table format, the sparse matrix representation for the above matrix looks like the following

	Row#	Column#	Value
0	0	0	100
1	0	3	900
2	0	5	500
3	1	4	200
4	1	7	300
5	2	1	400
6	2	6	800
7	3	2	200
8	4	0	1600
9	4	4	700

In this project, you will create appropriate C++ classes (see below) to create the sparse matrix representation data structure along with matrix operations on the sparse matrix.

Your Project Implementation: As part of this project you will create at two classes as described below. The first class that you will create will be a SparseRow class. The second class that you will create will be the SparseMatrix class. A partial (you are responsible to complete this along with writing all the methods) class definition for both the classes is given below.

```
template < class DT>
class ExceptionAdd{};

template < class DT>
class ExceptionMultiply{};

template < class DT>
class ExceptionCV{};

template <class DT>
class SparseRow {
    protected:
        int row; //Row#
        int col; //Column#
        DT value; //We will assume that all our values will be integers
    public:
        SparseRow (); //default constructor; row=-1;col=-1;value=0
        SparseRow(int r, int c, DT& v);
        virtual ~SparseRow(); //Destructor
        void display(); // print Row#, Column#, value
        //other methods as you deem fit
};


template <class DT>
class SparseMatrix {
    protected:
        int noRows; //Number of rows of the original matrix
        int noCols; //Number of columns of the original matrix
        int commonValue; //read from input
        int noNonSparseValues;
        vector<SparseRow<DT> >* myMatrix; //Array – should be dynamic
    public:
        SparseMatrix ();
        SparseMatrix (int n, int m, int cv);
        virtual ~SparseMatrix(); //Destructor
        void setSparseRow (int pos, int r, int c, DT& v);

        SparseMatrix<DT> operator*(SparseMatrix<DT> M); //for multiplication
and similar for addition and transpose

        void display();//Display the sparse matrix
        void displayMatrix (); //Display the matrix in its original format

        //other methods as you deem fit
};

template <class DT>
SparseMatrix::SparseMatrix (int n, int m, int cv) {
      int noRows = n;
      int noCols = m;
      int commonValue = cv;
      // noNonSparseValues = noNSV;
      // myMatrix = new SparseRow[noNSV];
}
```

Your main program will have the following structure (changes may be required).

```
int main () {

    int n, m, cv;
    SparseMatrix<int>* temp;
    int v;
    int k;

    // cout << endl << "First Matrix" << endl;
    cin >> n >> m >> cv;
    SparseMatrix<int>* firstOne = new SparseMatrix<int>(n, m, cv);


    //Write the Statements to read in the first matrix


    cout << "First one in sparse matrix format" << endl;
    cout << firstOne;

    cout << "First one in normal matrix format" << endl;
    (*firstOne).displayMatrix();


    cin >> n >> m >> cv;
    SparseMatrix<int>* secondOne = new SparseMatrix<int>(n, m, cv);


    //Write the Statements to read in the second matrix


    cout << "Second one in sparse matrix format" << endl;
    cout << secondOne;

    cout << "Second one in normal matrix format" << endl;
    (*secondOne).displayMatrix();


    temp = !firstOne;
    cout << "After Transpose first one in normal format" << endl;
    (*temp).displayMatrix();

    temp = !secondOne;
    cout << "After Transpose second one in normal format" << endl;
    (*temp).displayMatrix();


    cout << "Multiplication of matrices in sparse matrix form:" << endl;
    temp = secondOne * firstOne;
    cout << temp;

    cout << "Addition of matrices in sparse matrix form:" << endl;
    temp = secondOne + firstOne;
    cout << temp;

}
```

You must ensure that your program outputs the correct results. Make sure that you know matrix transpose, addition, and multiplication. Write those methods. Your input for each matrix will have the following structure. Note that in the first line, we have 5 rows, 8 columns, 0 being the common value and 10 being the number of non-sparse values.

5 8 0 10
100 0 0 900 0 500 0 0
0 0 0 0 200 0 0 300
0 400 0 0 0 0 800 0
0 0 200 0 0 0 0 0
1600 0 0 0 700 0 0 0

After the first input matrix you will read the second input matrix which will have the similar format.

Example inputs and outputs for the above given program have been posted on our website csgrader.ou.edu under the project 1 tab or you can also find them on canvas.

Redirected Input: Redirected input provides you a way to send a file to the standard input of a program without typing it using the keyboard. To use redirected input in Visual Studio environment, follow these steps: After you have opened or created a new project, on the menu go to project, project properties, expand configuration properties until you see Debugging, on the right you will see a set of options, and in the command arguments type <"**input filename**". The < sign is for redirected input and the **input filename** is the name of the input file (including the path if not in the working directory). A simple program that reads a matrix can be found below.

```
#include <iostream>

using namespace std;

int main () {

      int r,c,cv,nsv;
      int val;

      cin >> r >> c >> cv >> nsv;
      for (int i=0; i < r; i++) {
         for (int j=0; j < c; j++) {
            cin >> value;
            cout << value << " ";
         }
         endl;
      }
      return 0;
}
```

Constraints

1. In this project, the only headers you will use are #include<iostream> and #include<vector>.
2. None of the projects is a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.

Addendum(Feb 6th 2019)

1. **Exceptions now should be thrown for three things:**
    a. **If the rows and the columns of both the matrices don't match for addition.**
    b. **If the columns of the first matrix does not match the rows of the second matrix for multiplication.**
    c. **The common values of both the matrices don't match.**
2. **Both the classes must be templated with class DT. The code above has been updated.**
3. **The new input files will not have the noNSV variable anymore. So you can now use the stl vector library to maintain the myMatrix array. Refer to the field definition of myMatrix in class SparseMatrix now.**
4. **Write destructors for every class and delete every object created.**
5. **Overload the following operators(refer to the main function given above to see how they will be implemented):**
    a. **Ostream operator (<<) – this is for displaying the matrix in sparse row format**
    b. **Multiplication operator (*) – when multiplying two SparseMatrix objects.**
    c. **Addition operator (+) - when adding two SparseMatrix objects.**
    d. **Transpose operator (!) – for transposing a SparseMatrix object.**