COP3330 Programming Assignment 5

Deadline: Friday Apr 5th, 2019

Objective: Upon completion of this program, you should have more programming experience with **copy constructors**, **assignment operator overloading**, and comprehensive understanding of implantation details for matrixes in C++.

Overview: **Matrix** is a two-dimensional data structure used universally in Science, Technology, Engineering, and Math (STEM). Given a matrix **M** with *x* rows and *y* columns, it is often denoted as M_{x*y} for brevity. For instance, a matrix $\mathbf{M_{2*3}}$ with 2 rows and 3 columns is shown below,

1 1 1
2 3 5

In this assignment, we assume each entry of a matrix maintains a non-negative integer, and it can be accessed given the row and column indices. For instance, the value at row 2, column 2 of the above matrix is 3.

Matrices are commonly stored as a one-dimensional array in memory. The **row-majored** layout is to store the matrix row-by-row, starting from row one. For example, the above matrix will be stored into an array in a row-majored format as **1 1 1 2 3 5**. In contrast, the **column-majored** layout is to store the matrix column-by-column, starting from column one. For example, the above matrix will be stored into an array in a column-majored format as **1 2 1 3 1 5**.

Details:

You are to build a class named ***Matrix*** that can deal with non-negative integer matrices. The **Matrix** class is specified in detail as follows,

- **Member Data (that needs to be private)**

1. The number of **rows** of a matrix, which is a non-negative integer;
2. The number of **columns** of a matrix, which is a mon-negative integer;
3. A one-dimensional array of non-negative integers maintaining all the values within a matrix **(NOTE: two dimensional arrays or STL vectors are NOT allowed in this assignment!)**
4. An indicator variable indicates if the matrix is stored in a row-major way or a column-major way (In the following discussion, you will see that these two internal storage formats can be transformed from one to the other)

- **Member functions**

1. **Default constructor:**

```
Matrix(unsigned int row=5, unsigned int column=5, unsigned int value=0);
```

A default constructor will create a row-majored matrix with 5 rows and 5 columns, and all the values within the matrix will be initialized to 0. User can also specify different values for rows, columns, and initial values in the matrix.

2. **Copy constructor:**

Note: The newly created matrix has the same storage layout at the matrix to be copied.

3. **Destructor;**

Hint: make sure proper clean-up work should be done here.

4. **Assignment operator overloading;**

Note: The matrix to be assigned has the same storage layout at the assigning matrix.

5. **`numofrows(), numofcols();`**

Return the number of rows/columns of a matrix;

6. **`int get(unsigned int r, unsigned int c)`**

Given the row index *r* and the column index *c*, return the corresponding value at the *r*-th row and the *c*-th column of the matrix. If either *r* or *c* is out of bound, simply return -1. **Hint**: Depending on if the matrix is in row-major storage format or column-major storage format, the implementation may be different;

7. **`bool set(unsigned int r, unsigned int c, unsigned int value)`**

Set **`value`** to the corresponding entry of the matrix, specified by the row index *r* and the column index *c*. If either *r* or *c* is out of bound, return false. **Hint**: Again, the implementation may vary depending on if the matrix is row-majored or column-majored;

8. **`Operator+ overloading (as a member function):`**

We can add up two matrices in order to generate the third matrix as output. Given two matrices A_{x*y} and B_{x*y}, the summation **A+B** will result in a new row-majored matrix with the same number of rows (x) and columns (y) as A and B, where each entry is equivalent to the summation of the corresponding entries of A and B, respectively. In mathematics, it is commonly denoted as $(A+B)[i, j] = A[i, j] + B[i, j]$, where $1 \le i \le x$, and $1 \le j \le y$. Example:

$$A = \begin{matrix} 1 & 1 & 1 \\ 2 & 3 & 5 \end{matrix} \qquad B = \begin{matrix} 2 & 3 & 5 \\ 1 & 1 & 1 \end{matrix} \qquad A + B = \begin{matrix} 3 & 4 & 6 \\ 3 & 4 & 6 \end{matrix}$$

However, if A and B have different row or column numbers, the summation becomes meaningless, and we simply return a row-majored matrix whose numbers of rows/columns equal those of A's (A is the calling object), with all the values initialized to zero;

9. **[Extra Credit] `Operator* overloading (as a member function):`**

Given two matrices $\boldsymbol{A_{x*y}}$ and $\boldsymbol{B_{y*z}}$ (the matrix A is the calling object; note that the column of A is equal to the row of B), the multiplication of **A*B** will lead to a new row-majored matrix $\boldsymbol{C_{x*z}}$ with *x* rows and *z* columns, where each entry c_{ij} within *C* is computed as

$$\boldsymbol{c_{ij} = a_{i1} * b_{1j} + \cdots + a_{iy} * b_{yj} = \sum_{k=1}^{y} a_{ik} * b_{kj}}$$

Example:

$$\boldsymbol{A = \begin{matrix} 1 & 1 & 1 \\ 2 & 3 & 5 \end{matrix} \qquad B = \begin{matrix} 2 & 1 \\ 2 & 1 \\ 2 & 1 \end{matrix} \qquad A * B = \begin{matrix} 6 & 3 \\ 20 & 10 \end{matrix}}$$

However, if A's column is NOT equal to B's row, matrix multiplication becomes meaningless. In this case, return a row-majored matrix with A's row and column and all the entries are initialized as zero.

10. `void torowmajor();`

If the current storage format of the matrix is column-majored, it will be changed to row-majored. Note that the internal storage (the one-dimensional data structure, `array`) of the matrix has to be changed. However, if the current storage format is already row-majored, we do nothing.

11. `void tocolumnmajor();`

If the current storage format of the matrix is row-majored, it will be changed to column-majored. Note that the internal storage (the one-dimensional data structure, `array`) of the matrix has to be changed. However, if the current storage format is already column-majored, we do nothing.

12. `void printinternal();`

Output all the values of the one-dimensional data structure, `array`, of the matrix on the screen (using cout). Values are separated with TAB. This is a diagnose function to check the internal storage layout of the matrix, which can be either row-majored or column-majored (but NOT both).

13. `Matrix transpose();`

Given a matrix $\boldsymbol{A_{x*y}}$, we create another row-majored matrix as output $\boldsymbol{B_{y*x}}$ with *B[i, j] = A[j, i]*. Example:

$$\boldsymbol{A = \begin{matrix} 1 & 1 & 1 \\ 2 & 3 & 5 \end{matrix} \qquad B = \begin{matrix} 1 & 2 \\ 1 & 3 \\ 1 & 5 \end{matrix}}$$

14. [Extra Credit] `Matrix submatrix(unsigned int startrow, unsigned int endrow, unsigned int startcol, unsigned int endcol);`

Extract as output a row-majored submatrix from the given matrix A (the calling object) with the rows in the range of *[startrow, endrow]*, and with the columns in the range of *[startcol, endcol]*. If the ranges defined by startrow, endrow, startcol, endcol are meaningless (for instance, endrow < startrow, or endcol < startcol), or simply out of bound, return a row-majored matrix with the same number of rows/columns as A and all the values are initialized as zero. Example:

```
B = A.submatrix(1, 2, 3, 3);
```

$$A = \begin{matrix} 1 & 1 & 1 \\ 2 & 3 & 5 \end{matrix} \qquad B = \begin{matrix} 1 \\ 5 \end{matrix}$$

- **Friend functions**

1. Operator << overloading (as a friend function)
Output the matrix in a row-by-row fashion: each row is output in a separate line, and values of each row are output well separated by TAB. By overloading this function, we can print out a matrix easily.

Testing: Once the **Matrix** class is implemented, you can test your implementation using the sample **`proj5.cpp`** program. You should also write other test programs to do a thorough test of your implementation. Please note that the expect output is also provided in the file **`output.txt`**.

General Requirement:

1. Please implement the **Matrix** class **all by yourself**! You may use the istream, ostream, iostream in your implementation. DO NOT use any other C++ standard libraries, or any third-party libraries or API calls;
2. No global variables, other than constants!
3. All member data of your class must be private;
4. The const qualifier should be used on any member functions and parameters where it is appropriate;
5. You only need to do error-checking that is specified in the descriptions above;
6. User input and/or screen output should only be done where described (i.e. do not add in extraneous input/output);
7. When you write source code, it should be readable and well-documented.

Submission:

Program submissions should be done through Canvas. Do not send program submissions through e-mail -- e-mail attachments will not be accepted as valid submissions! For this assignment, submit a compressed package (named **proj5.tar.gz**) including the following files

`matrix.h`
`matrix.cpp`
`makefile` (the final executable is named **`proj5`**, and the file containing main() function is named **`proj5.cpp`**)
`readme` (optional)

Make sure your filenames are these exact names, and do not submit your proj5.cpp file (or any other main program you might create for testing purposes).