CS 101
Spring 2016
Project 3

A sparse matrix is a matrix in which most of the elements are zero. It is not wise to use a two dimensional array to store a sparse matrix, especially a large one because of the large number of zeros in the matrix. As a result, a variety of data structures and algorithms have been proposed to deal with large sparse matrices.

In this project, you are asked to write a class named **SparseMatrix** that satisfies the following requirements. (The class will be defined and implemented in *SparseMatrix.h* and *SparseMatrix.cpp* respectively.)

- Define a data structure to efficiently represent a sparse matrix. You shall represent the sparse matrix internally using a linked structure. Do not use a 2D array representation, because some row and column numbers will be so large that a 2D array representation would be too inefficient in terms of both running time and memory usage. Matrix elements with value 0 are not permitted to appear within the sparse matrix representation.
- The default constructor will create a zero matrix (a matrix with all the elements being zero).
- Write a destructor to release all the dynamically allocated memory.
- Overload the + operator to get the sum of two sparse matrices.
- Overload the - operator to get the difference of two sparse matrices.
- Overload the * operator to get the product of two sparse matrices. Please see https://en.wikipedia.org/wiki/Matrix_(mathematics) for the definition of matrix multiplication.
- Overload the << operator to write a sparse matrix to an output stream. A file written by the output stream via << must be in the correct format to be readable by a future >> operation. The file format for a sparse matrix is to be described later.
- Overload the >> operator to read a sparse matrix from an input stream.

You can use the following *project3.cpp* to test the class you have implemented.

```
int main(int argc, char **argv) {
    // check if argc==5
    SparseMatrix m1, m2, m;
    ifstream infile1, infile2;

    infile1.open(argv[2]); // add error checking
    infile2.open(argv[3]); // add error checking

    infile1 >> m1;
    infile2 >> m2;
```

```
    infile1.close();
    infile2.close();

    string op(argv[1]);
    if (op=="add") {
        m=m1+m2;
    }
    else if (op=="sub") {
        m=m1-m2;
    }
    else if (op=="mul") {
        m=m1*m2;
    }
    else {
        cerr << "No such operation" << endl;
        return 1;
    }

    ofstream outfile;
    outfile.open(argv[4]); // add error checking

    outfile << m;

    outfile.close();

    return 0;
}
```

For example, you can use the following command to subtract the matrix stored in *b.txt* from the matrix stored in *a.txt*, and the resulting matrix will be saved into the file named *c.txt*.

```
$ ./project3 sub a.txt b.txt c.txt
```

A sparse matrix is stored in a text file with each line corresponding to a non-zero element. There are three numbers in each line separated by spaces: row, column, and value. The row or column number is non-negative integers and zero-based. The value is a double. The value of 0 is not permitted except in the last line. Having three 0's, the last line does not represent an matrix element. It just indicates the end of matrix input. The dimension of the matrix is not explicitly specified, but it can be derived from the maximum row or column number. Since the zero elements are not stored, a matrix can be expanded with additional zeros to satisfy the requirements of matrix addition, subtraction, and multiplication. The order of the non-zero elements from an input stream is unknown whereas you have to output the non-zero elements in row-major order to an output stream.

Programs that do not compile or run according to the given specifications will earn a failing grade, so be careful to follow all instructions precisely. In addition, please carefully follow the following directions for submission.

- You must have at least four files: *SparseMatrix.h*, *SparseMatrix.cpp*, *project3.cpp* and *makefile*.
- Your *makefile* must support the commands "make" and "make clean". Also, “make project3” should build your program executable named *project3*.
- You must use -Wall -std=c++11 options with g++.
- Your input and output files must be in the format as specified above.
- You must submit your project in a zipfile named LASTNAME_FIRSTNAME.zip. As previous project in this course, if you unzip your zipfile you should get only the contents of the directory and not the directory itself.
- You must submit the project on Blackboard by the due date. Late submissions will incur a 10% deduction per day late. No submissions will be accepted after 3 days late.
- You must do your own work, you must not share code.