COSC 1430 HW3: Sparse Matrix Addition

1 - Objective

This homework assignment will give you a better understanding of how to parse an input file and use structs to combine several variables in one object.

2 - Problem

For this assignment, you will create a C++ program, named **sparseAdd.cpp** and **you must submit in your home folder on the server**. This program will add two sparse matrices in an efficient manner. Sparse matrices have many zero entries, but such entries are omitted in the input file and corresponding data structures to save memory and reduce processing time. You will read in 2 files and output 1 file; you must ask the user for the names of the files in the console.

Note: Functions are a good way to increase the modularity of your code.

2.1 Matrices and Matrix Addition:

Subscripts in input and output files are matrices that are based on a standard mathematical definition, indexed from 1. Matrices of size m × n start on 1,1 and go up to m,n. Therefore, the input/output matrices should <u>always</u> have subscripts starting at 1,1.

2.2 Matrix Addition:

The addition of 2 matrices is done by adding each element of the matrices that are the same size. Assuming a matrix M1 and M2 are of size m x n, the sum would be $M3 = M1_{rc} + M2_{rc} \ \forall r,c$ where $1 \le r \le m$, $1 \le c \le n$. M_{rc} represents the value of the matrix at row r and column c, and M1 and M2 are the input matrices.

2.2 Storage

You must store each input matrix as a single 1-dimensional dynamic array of structs.

```
struct element {
        int r;
        int c;
        double value;
}
element* m1=new element[size1];  // you must determine the size dynamically
element* m2=new element[size2]; // you must determine the size dynamically
```

The input file will have an input of *triples (r, c, v) that indicate the non-zero entries in the matrix*. Here, the r and c are integers that represent the entry (row, column) of the matrix, and v (double) is the value of the entry. You cannot store zero entries in your arrays.

2.2.1 Matrix Example 1

For example, a 3x2 sparse matrix will look like:

$$\begin{bmatrix} 1.2 & 0 \\ 0 & 0 \\ 2.3 & -4.5 \end{bmatrix}$$

And the resulting input file will read:

```
#Matrix A
3 2
1 1 1.2          ← Row 1 Column 1 value 1
3 1 2.3          ← Row 3 Column 1 value 2
3 2 -4.5         ← Row 3 Column 2 value 4
```

2.2.2 Matrix Example 2

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 2 & 0 \end{bmatrix}$$

And the resulting input file will read:

```
#Matrix A
3 2
1 1 1            ← Row 1 Column 1 value 1
3 1 2            ← Row 3 Column 1 value 2
```

3 - Sample Input

3.1 Input files specifics

The input are two text files, with one matrix per file. Prior to any data, a line containing the dimensions of the matrix will be present. There will be ONE matrix entry per line in the file and each line will have a triplet of numbers r, c, v. Subscripts r and c are positive integers and the value v is a real number (possibly negative).

Comments have a # at the beginning of the line. Your program must ignore lines that begin with the '#' symbol. These can be placed anywhere in the input file. Your program should not get stuck due to this format in the input file.

Values v can be positive or negative, where negatives start with '-' and one digit (numbers like -.2, or -0.31416e+10 will not be provided).

Your program **must reject matrices** whose sizes are incompatible for addition (i.e., having different dimensions) and produce an empty file as result with an error printed on the console. Incompatible matrices is the only case where your program should stop with something printed on the console; otherwise, nothing is printed on the console.

Note that matrices are not necessarily squared and they can even have only one non-zero entry.

Examples of input and result file are below.

3.2 Input files example 1

Contents of File 1:

```
#Matrix A
1 1
1 1 10.11
```

Contents of File 2:

```
#Matrix B
1 1
1 1 2.2
```

3.3 Input files example 2

Contents of File 1:

```
#Matrix A,
2 2
1 1 2.2
2 2 3
```

Contents of File 2:

```
2 2
#Matrix B
#Aliens are real
1 1 1.44
1 2 4.52
```

3.4 Input files example 3

Contents of File 1:

```
#A
3 3
3 1 1
3 2 1
#Never gonna give you up. Never gonna let you down
3 3 1
```

Contents of File 2:

```
3 3
1 1 1
2 1 1
#Never gonna run around and desert you. Never gonna make you cry
3 1 1
####### 9 + 10 = 21 #########
3 3 21
#Never gonna say goodbye. Never gonna tell a lie and hurt you
```

3.5 Input files example 4

Contents of File 1:

```
#Matrix A
1020 12340
1 1 -10
100 100 100
200 200 200
300 300 300
400 400 400
500 500 500
600 600 -600
800 800 800
1020 12340 5.23
```

Contents of File 2:

```
1020 12340
#B
234 123 3
700 700 -700.43
800 800 800
1020 12340 244.44
```

3.6 Input files example 5

Contents of File 1:

```
#Matrix A,
2 2
#Climate Change is FAKE NEWS
1 1 2.2
```

Contents of File 2:

```
#Matrix B
2 2
1 1 1.44
1 2 4.52
```

3.7 Input files example 6 - These matrices are incompatible

Contents of File 1:

```
#Matrix A
10 10
1 1 10
10 10 6
```

Contents of File 2:

```
#B
21 9
3 3 2.44
21 9 1.0
```

4 - Sample Output

4.1- Output File Specifics

The output matrix should be written in sorted order by row and column in sparse format (one entry per line) as well. The first line must have the dimensions of the matrix.

You should write output numbers only with 2 decimals.

4.2 Output example 1

```
#Matrix C
1 1
1 1 12.31
```

4.3 Output example 2

```
#Matrix C
2 2
1 1 3.64
1 2 4.52
2 2 3.00
```

4.4 Output example 3

```
#Matrix C
3 3
1 1 1.00
2 1 1.00
3 1 2.00
3 2 1.00
3 3 22.00
```

4.5 Output example 4

```
#Matrix C
1020 12340
1 1 -10
100 100 100.00
200 200 200.00
234 123 3.00
400 400 400.00
500 500 500.00
600 600 -600.00
700 700 -700.43
800 800 1600.00
1020 12340 249.67
```

4.6 Output example 5

```
#Matrix C
2 2
1 1 3.64
1 2 4.52
```

4.7 Output example 6 - This will be an empty file

5 - Useful Hints

5.1 Accessing a certain character of a string

You can access an element of a string as you do in arrays in C++.

```
string str= "Hello";
char str2 = str[1]; // the value of str2 is 'e'
char str4 = str[0]; // the value of str3 is 'H'
```

5.2 Reading an entire line from a file

Make sure you have *#include<string>*. The getline() function will read an entire line of a file and will store it into a string defined by the user.

The following code snippet prints out every line of an input file *ex.txt*.

```
ifstream inputfile;
inputfile.open("ex.txt");
string line = "";
int num = 0;
while(getline(inputfile,line)){
   cout<<line<<endl;
}
inputfile.close();
```

6 - Grading

6.1 Memory Leaks

A memory leak in programming is when memory that is no longer needed is not released. As you will be dynamically allocating memory for your array, you will have to manage memory yourselves.

Your program will be tested for dynamic memory allocation and memory leaks (new, delete operators correctly programmed).

Use "`valgrind --leak-check=yes ./executableName`" to detect for memory leaks. `executableName` is the name you gave your program when compiling. For instance, if you compiled your program by: "`g++ -std=c++11 -o sparse sparseAdd.cpp`", then `executableName` would be replaced with `sparse`.

6.2 Testing

Your programs will be tested with sparse matrices whose theoretical size that may go up to 10,000 × 10,000 (i.e. they will have large subscripts in some entries).

6.3 Grading

A submitted program that **does not compile** is only worth **at most 10 points.**
A submitted program that contains **an infinite loop** is only worth **at most 20 points.**
A submitted program that contains **no dynamic arrays** is only worth **at most 20 points.**
A submitted program that **has memory leaks** will have 25 **points deducted.**

Note: An Additional 10 points will be deducted if your file is not named correctly.
Note: You cannot use simple 2-dimensional arrays to store the matrix (e.g. double A[10][10]); otherwise, your score will be **at most 20.**

Your program will have to pass 10 test cases. For each test case failed, you will lose 10 points. The test cases will be provided after the grading period is over.

7 - Bonus (5 pts)

Check for consistency of the inputs by ignoring lines in the input whose entries are out of bounds. For instance, if a matrix's dimensions are 3 by 2, your program should ignore input entries like 1 5 4, because there are not 5 columns in the matrix.

Note: You should have comment in the beginning of your program stating that you are attempting the bonus.

7.1 Example Input

Contents of File 1:

```
#Matrix A,
2 2
#Kanye2020
1 1 2.2
7 3 2.4      ← your program should ignore this line and you shouldn't store it in your array
2 1 4.32
```

Contents of File 2:

```
#Matrix B
2 2
1 1 1.44
#The Earth is flat
1 2 4.52
```

7.2 Example output

```
#Matrix C
2 2
1 1 3.64
1 2 4.52
2 1 4.32
```