Functions and Arrays

Object Oriented Programming - Programming Coursework #6 (20 pts)

Due date : June 18, 2020 5pm

Please note the new exception handling requirements (Sec. 3.4) and output fine name convention.

1 Task

The objective of this coursework is to gain a deeper insight into defining and using functions and arrays. The task is to implement two programs. The first program (Program1) will sort a sequence of real-valued numbers in non-decreasing order (8 pts): The program will read values from an input data file ('Program1Input.txt') and generate an output file ('Program1Output*StudentID*.txt') that lists the input numbers arranged in non-decreasing order. The second porgram (Program2) will take two real-valued matrices, say A and B and generate the matrix multiplication AB of A and B (12 pts). It will read values from two input data files ('Program2InputA.txt' and 'Program2InputB.txt') and generate an out file ('Program2Output*StudentID*.txt').

2 Program instructions and specifications

All submitted codes must satisfy the execution requirements and compilation requirements described in Secs. 3.2 and 3.3, respectively. If the submitted codes do not meet these two sets of requirements, the corresponding codes will get 0 point. There are additional, source code requirements (see Sec. 3.1 for details). The programs also have to **handle exceptions** listed in (Sec. 3.4): We will assess the submitted codes with several exception cases, and each time the submitted codes do not handle the cases properly, the assigned points will decrease by 2.

2.1 Program1

Input file: 'Program1Input.txt' to be provided by the CW assessor. This contains real numbers each separated by a space.

Output file: 'Program1Output*StudentID*.txt' to be generated by Program1. This should contain a sequence of real numbers, each separated by a space: These are the numbers in 'Program1Input.txt', arranged in non-decreasing order. The precision of each number in 'Program1Output*StudentID*.txt' should be the same as corresponding entry in 'Program1Input.txt'.

In the output file name, *StudentID* should be replaced by your student ID. For example, if your student ID is 20201111, the output file name should be 'Program1Output20201111.txt'. This file name convention applies to all output files generated by the submitted codes. If the submitted codes do not observe this file name convention, the maximum possible points will be reduced to 50% of the total points.

2.2 Program2

Input file: 'Program2InputA.txt' and 'Program2InputB.txt' to be provided by the CW assessor. Each of 'Program2InputA.txt' and 'Program2InputB.txt' contains a matrix. The first line of ‘Program2InputA.txt’ contains two integers, namely R and C representing the number of rows and columns of a matrix A, respectively. From the next line, each line represents a row of A: The n-th line of ‘Program2InputA.txt’ contains C numbers each separated by a space, corresponding to the $(n-1)$-th row of A. Similarly, ‘Program2InputB.txt’ contains matrix B. The format of ‘Program2InputB.txt’ is the same as ‘Program2InputA.txt’: It will start with a line consisting of two integer numbers followed by multiple lines each representing a row of B.

Output file: ‘Program2Output*StudentID*.txt’ to be generated by Program2. ‘Program2Output*StudentID*.txt’ follows the same format as ‘Program2InputA.txt’ and ‘Program2InputB.txt’. It should present the multiplication AB of matrices A and B. The first line of ‘Program2Output*StudentID*.txt’ should contain two integer numbers, representing the numbers of rows and columns of AB, respectively. From the second line, each line should correspond to a row of AB: The n-th line of ‘Program2Output*StudentID*.txt’ should contain numbers representing the $(n-1)$-th row of AB. The numbers in ‘Program2Output*StudentID*.txt’ should be given to six digits after the decimal point (by rounding off).

Sample inputs and outputs.

‘Program1Input.txt’ file:

```
5 −10.2 40.5 10 10 −80 90
```

‘Program1Output*StudentID*.txt’ file:

```
−80 −10.2 5 10 10 40.5 90
```

‘Program2InputA.txt’ file:

```
2 3
0.4863     0.3110     0.4121
−0.2155    −0.6576    −0.9363
```

‘Program2InputB.txt’ file:

```
3 3
−0.4462     0.6469     0.9004
−0.9077     0.3897    −0.9311
−0.8057    −0.3658    −0.1225
```

‘Program2Output*StudentID*.txt’ file (note that the numbers are displayed to six digits after the decimal point):

```
2 3
−0.831311    0.285038     0.097810
1.447437    −0.053175    0.532952
```

2.3 Assessment

Program1 will be assessed based on five different instances of ‘Program1Input.txt’. This input file will be located at the same directory as the executable file of Program1. When executed with an instance of ‘Program1Input.txt’, Program1 will generate ‘Program1Output*StudentID*.txt’ file in the same directory. Similarly, Program2 will be assessed based on five different paris of ‘Program2InputA.txt’ and ‘Program2InputB.txt’ (in the same directory as the executable file of Program2) and it should generate the corresponding ‘Program2Output*StudentID*.txt’ file.

3 What to hand in for assessment

Please submit the source codes for Program1 and Program2, and a brief report (a single report for both programs) via Black Board. Comments in the codes and the report should be written in English. The report should provide code comments explaining the algorithms. If the submission does not include a report, the maximum possible points will be kept at 40% of the full points. Please format the submission as a single zip file ‘StudentID_Name.zip’ that includes the files of the source codes and the report,

e.g.,
20201111_KwangInKim.zip
- Program1.cpp
- Program2.cpp
- Report.pdf

The submitted source codes should contain ‘Program1.cpp’ and ‘Program2.cpp’ files each including the id and name of the submitting student.

In “Program1.cpp”

```
//StudentId Name
#include <iostream>
Your code here...
```

3.1 Source code requirements

The submitted Program1 code should implement and call a function ‘mySwap’ that takes two real-valued variables as inputs, and swap the contents of these two in the calling function. For example, suppose that the calling function (e.g., main function) has two float variables A and B. After calling ‘mySwap(A,B)’, the values of A and B are exchanged in the calling function, e.g.,

```
int main()
{
        float A = 0.1, B=0.2;
        cout << A << `` '' << B <<endl;      //It will print out ``0.1 0.2''
        mySwap(A,B);                         //A and B values are swapped by mySwap
        cout << A << `` '' << B <<endl;      //It will print out ``0.2 0.1''
        return 0;
}
```

All submitted codes should implement the required operations from scratch: **Calling library functions that perform the required operations from any part of the submitted codes is not allowed**. If each submitted code does not meet the source code requirements, e.g.,. not properly defining and using ‘mySwap’ function in Program1, the maximum possible points for the respective program will be kept at 50% of the corresponding maximum points. Please note that this does not mean that such code will automatically get 50% of the maximum points: E.g., if Program1 uses external libraries that sorts numbers, it will get 0 point.

3.2 Execution requirements

For assessing Program1, ‘Program1Input.txt’ will be prepared such that it contains less than 50 numbers. For Program2, the maximum of the number of columns and rows for the two input matrices will be 10. Once executed for these cases, both Program1 and Program 2 should terminate within a second at UNI06 server.

3.3 Compilation requirements and instruction

Each submitted code needs to be compilable by the GNU C++ compiler and in the Linux environment. Specifically, they should be **compilable and executable in UNI06 server**. Students can verify the correct compilation and execution of their codes by following the instructions provided in ‘howtocompile.pdf’ document accompanying the CW1 announcement.

3.4 Exception handling requirements

Program1 and Program2 must handle the following exceptions.

Program1

1. If 'Program1Input.txt' does not exist in the same directory as the executable of Program1, Program1 should generate 'Program1Output*StudentID*.txt' file that only contains a string "Error#0".
2. If 'Program1Input.txt' contains anything else than numbers (and spaces), Program1 should generate 'Program1Output*StudentID*.txt' file only containing a string "Error#1".

Program2

1. If any of 'Program2InputA.txt' and 'Program2InputB.txt' do not exist in the same directory as the executable of Program2, Program2 should generate 'Program2Output*StudentID*.txt' file that only contains a string "Error#0".
2. If any of 'Program2InputA.txt' and 'Program2InputB.txt' contain anything else than numbers (plus spaces and newline characters), Program2 should generate 'Program2Output*StudentID*.txt' file containing only a string "Error#1".
3. If the multiplication AB is not defined for the matrices A and B provided in 'Program2InputA.txt' and 'Program2InputB.txt', respectively, e.g., the number of columns of A and the number of rows of B differ, Program2 should generate 'Program2Output*StudentID*.txt' file containing only a string "Error#2".

Your programs do not have to handle exceptions other than what are listed above. For instance, for Program2, if the first line of 'Program2InputA.txt' contains "5 2", the rest of 'Program2InputA.txt' will contain exactly 10 (5×2) entries.