COMP2113 Programming Technologies / ENGG1340 Computer Programming II
Assignment 2
Deadline: 31 October (Thursday) 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 2.

Total marks: 100 marks

General Instructions

Read the instructions in this document carefully.

In this assignment you will solve 2 tasks and a tester would automatically test your submitted program. So if your submitted files and program outputs do not conform to our instructions given here, your programs cannot be evaluated and you will risk losing marks totally.

Sample test cases are provided with each task in this document. Note that the test cases may or may not cover all boundary cases for the problem. It is also part of the assessment whether you are able to design proper test cases to verify the correctness of your program. We will also use additional test cases when marking your assignment submission.

Input and output format

Your C++ programs should read from the **standard input**. Also, your answer should be printed through the **standard output**. If you failed to follow this guide, the tester may not able to give a score for your program. Additionally, you should strictly follow the sample output format (including space, line breaker, etc.), otherwise, your answer might be considered as wrong.

How to use the sample test cases for problems 1 and 2?

Sample test cases in text file formats are made available for you to check against your work. Here's how you may use the sample test cases. Take problem 1 test case 2 as an example. The sample input and the expected output are given in the files input1_2.txt and output1_2.txt, respectively. Suppose that your program is named "1", do the followings at the command prompt of the terminal to check if there is any difference between your output and the expected output.

```
./1 < input1_2.txt > myoutput.txt
diff myoutput.txt output1_2.txt
```

Testing against the sample test cases is important to avoid making formatting mistakes. The additional test cases for grading your work will be of the same formats as the sample test cases.

Coding environment

You must make sure that your C++ program can compile, execute and generate the required outputs on our standard environment, namely, the gcc C++11 environment we have on the CS Linux servers (academy*). Make sure the following compilation command is used to compile your programs:

```
g++ -pedantic-errors -std=c++11 [program_name].cpp -o [object_code_name]
```

The name after the "-o" option is the name of the object code. In the above example, you can run the program using the following command:

```
./[object_code_name]
```

As a programmer/developer, you should always ensure that your code can work perfectly as expected on a target (e.g., your client's) environment, not only on yours.

While you may develop your work on your own environment, you should always try your program (compile & execute & check results) on our standard environment before submission.

Submission

Name your C++ program files (and Makefile for Problem2) as the following table shows and put them together into one directory. Make sure that the folder contains only these source files (*.cpp / *h / Makefile) and no other files. **You need to use VPL to test your code.** Resubmission after the deadline is not allowed.

Filename	**Description**
main.cpp	Problem 1
process_scores.h, process_scores.cpp, main.cpp, Makefile	Problem 2

Late submission

If you submit within 2 days after the deadline, there will be 30% mark deduction. If you submit within 3 to 5 days after the deadline, there will be 50% mark deduction. After that, no mark will be given.

Evaluation

Your code will be auto-graded for technical correctness. In principle, we use test cases to benchmark your solution, and you may get zero marks for not being able to pass any of the test cases. Normally partial credits will not be given for incomplete solution, as in many cases the logic of the programs are not complete and an objective assessment could be difficult. However, your work may still be considered on a case-by-case basis during the rebuttal stage.

Academic dishonesty

We will be checking your code against other submissions in the class and from the Internet for logical redundancy. Please be reminded that no matter whether it is providing your work to others, assisting others to copy, or copying others will all be considered as committing plagiarism and we will follow the departmental policy to handle such cases. Please refer to the course information notes for details.

Getting help

You are not alone! If you find yourself stuck on something, post your questions to the course forum, send us emails or come to our support sessions. We want this assignment to be rewarding and instructional, not frustrating and demoralizing. But we don’t know when or how to help unless you ask.

Discussion forum

Please be careful not to post spoilers. Please don’t post any code that is directly related to the assignments. However, you are welcome and encouraged to discuss general ideas on the discussion forums. If you have any questions about this assignment, you should post them in the discussion forums.

Problem 1 [50%]:

A 2D geometric shape can be either a circle or a rectangle. The area and perimeter of each shape are determined by specific mathematical formulas. A circle is defined by its radius r and a rectangle is defined by its width w and height h. You are tasked with writing a C++ program that reads an integer N (the number of shapes) and then takes N shapes as input, where each shape is specified by its type and the necessary dimensions.
The program should then output:
1. The area and perimeter of each shape.
2. The total area and total perimeter of all N shapes.

The inputs will follow these rules:
- A circle is represented by “C r” where r is the radius.
- A rectangle is represented by “R w h” where w is the width and h is the height.

Assumptions and requirements:
1. Use $\pi = 3.14$ for all circle calculations.
2. Output all results rounded to two decimal places.
3. Inputs may also contain two decimal numbers.

Formulas for each shape:
1. Circle:
   Area: $= \pi * r * r$
   Perimeter: $= 2 * \pi * r$
2. Rectangle:
   Area: $= w * h$
   Perimeter: $= 2 * (w + h)$

Sample Test Cases
User inputs are shown in blue.

1_1:
```
3 C 5 R 3 4 C 6
78.50
31.40
12.00
14.00
113.04
37.68
203.54
83.08
```

1_2:
```
2 C 7 R 2 10
153.86
43.96
20.00
```

```
24.00
173.86
67.96
```

Example explanation:

Consider the input:

```
2 C 5 R 3 4
```

“2” indicates there would be two shapes to calcualte. The first input “C 5” represents a circle with radius 5. The second input “R 3 4” represents a rectangle with width 3 and height 4. For each shape, the program immediately outputs the area and perimeter on separate lines. After all shapes are processed, the total area and total perimeter are output on two final lines. For the input above, the output will be:

```
78.50
31.40
12.00
14.00
90.50
45.40
```

The first two lines are the area and perimeter of the circle. The next two lines are the area and perimeter of the rectangle. The last two lines are the total area and total perimeter of all shapes combined.

Problem 2 [50%]:

You are required to write a C++ program that processes student score data from multiple files. Each file contains a list of student names and their corresponding scores.
Your program should calculate the following for each file:

1. The average score of all students across the file.
2. The student with the highest score and their name.
3. The student with the lowest score and their name.
4. The total number of students who passed (score ≥ 60).

To allow other programs to reuse our program, let us put the code in a separate file.
You need to implement these four functions in `process_scores.cpp`, write the corresponding head file. Then, you need to finish `main.cpp` to handle inputs. Finally, you will finish `Makefile` to compile your program.

Input:

- The program accepts command line arguments, where the first argument is the program name, followed by one or more filenames. Each file contains student data in the format:
  ```
  <student_name_1> <score_1>
  <student_name_2> <score_2>
  ```
  - Each student's name is a single word, and the score is an integer between 0 and 100.
  - The number of students in each file can vary.
  - If a file cannot be opened, print an error message for that file.
- Command line arguments in C/C++ are implemented by having input parameters for the `main()` function:
  `int main(int argc, char* argv[])`
- The main program **accepts command line argument as input**. The command line of the program is given by: ./`<program_name> <file1> <file2> <file3>`
  E.g., if you compile your program using `g++ -pedantic-errors -std=c++11 main.cpp process_scores.cpp -o student_scores`, then the following command at the command prompt
  `./student_scores students1.txt students2.txt students3.txt`
  will ask your object program `student_scores` to calculate in files `students1.txt`, `students2.txt`, `students3.txt`. You can use this to quickly check your answer, but this is not the final result for submission.
- Please remember finally you should have a `Makefile` to generate the following targets:
  - `Process_scores.o`: which depends on `process_scores.cpp` and `process_scores.h`
  - `main.o`: which depends on `main.cpp` and `process_scores.h`
  - `main`: which depends on `process_scores.o` and `main.o`
  - `clean`: for cleaning `main`, `main.o` and `process_scores.o`
  - You should assume that there can be a file named "clean" in the folder and you need to make sure that running `make clean` will not cause any conflict. If not you will get an error like `Error: Your Makefile does not define a 'clean' target.` on VPL.
  - Upon `Makefile` is created appropriately, you should be able to compile all programs by issuing the command `make main` and cleaning all object files by issuing the command `make clean`.
  - For the final submission, you should name your executable file as `student_scores`, and our VPL will test it. Otherwise, you would get an error like `Compilation failed. Cannot find stduent_scores. Exiting...` on VPL.

Error Handling:

You must handle the following error cases:

1. File cannot be opened: If a file cannot be accessed, output an error message.
2. Invalid file format: If a line in the file does not follow the format <name> <score>, skip the line and print an error message.
3. Invalid score values: If a score is not a valid integer or is outside the range [0, 100], skip the line and print an error message.
4. Empty or invalid files: If a file contains no valid student data, print a warning.

Output:

The program should output:

1. The overall average score of all students (rounded to two decimal places).
2. The student with the highest score and their score.
3. The student with the lowest score and their score.
4. The total number of students who passed (score ≥ 60).

or if there is any error accessing the file ():

`<filename>: error processing file`

Additional Clarifications:

- All scores are integers between 0 and 100.
- Names are single words (no spaces).
- Ensure that the output is formatted exactly as shown, especially rounding the average score to two decimal places.
- Handle all errors with a unified error message: `<filename>: error processing file`
- Skip the invalid line but continue processing the rest of the file.

Sample Test Cases

Sample text files `students1.txt`, `students2.txt`, `students3.txt` are provided.

students1.txt	**students2.txt**	**students3.txt**
Alice 85 Bob 78 Charlie 92 Diana 61	Frank 50 Grace 71 Heidi 95	Kate 45 Liam Mason 77 Olivia -64 Sophia 89

Commands entered by the users at the command line prompt “>” to run your program are shown in blue.

Note that since there is no user input from the standard input, here's how you should test your output against the sample output (e.g., for test case 2_1) to check for correctness:

```
./student_scores students1.txt students2.txt > myoutput.txt
diff myoutput.txt output2_1.txt
```

2_1:

```
./student_scores students1.txt students2.txt
Average score: 76.00
Highest score: Heidi 95
Lowest score: Frank 50
```

```
Number of students passed: 6
```

2_2:

```
./student_scores students1.txt students3.txt
students3.txt: error processing file
students3.txt: error processing file
Average score: 75.29
Highest score: Charlie 92
Lowest score: Kate 45
Number of students passed: 6
```

The first two lines show there are errors in two lines of students3.txt: One where the score is missing (Liam). One where the score is out of range (Olivia with -64). The last four lines give the results after processing all valid data from both files: The overall average score. The student with the highest score. The student with the lowest score. The total number of students who passed.

If you face any problems, **please feel free to contact us or join our STA support session! We are very happy to help you!**
We wish you enjoy this assignment! ☺