Problem Sheet 1

LM Advanced Mathematical Finance Module

Part B C++ Spring Term 2021

PROBLEM SHEET 1	SPRING TERM, PROGRAMMING IN C++ LECTURE 1
Lecturer	Dr Simon Hartley s.hartley@bham.ac.uk
Of	School of Mathematics, College of Engineering and Physical Sciences, University of Birmingham
Due	Tuesday 23rd March 2021, 23:59
Weighting	10%

INSTRUCTIONS FOR THE SUBMISSION

Submit your code on Canvas twice:

First as a single zip file containing all code for each problem, and then upload the individual source files.

Each problem sheet should be implemented in source code files that are named as follows: Each program contains a main_X.cpp. Here, X is the number of the problem. **For example,** main_P1.cpp.

The reason for this double-submission is that Canvas internally renames files. Therefore, the individual files cannot be compiled unless they are renamed back (which would create a lot of work for the marker). If a zip file is submitted, that zip file is renamed, but not the files within that zip file. However, comments on Canvas can only be given on text/word/pdf files, which is why you have to additionally submit the individual source files.

The source files should compile without errors or warnings on Visual Studio 2019 as installed on the clusters, and the resulting executable should run without problems. You do not have to submit the executable.

If you use another compiler other than Visual Studio, you should take your final code and compile it on Visual Studio on the remote clusters.

Unfortunately, different compilers follow slightly different rules, and even a small difference may cause the code to fail on Visual Studio. The source code should also be well formatted. This means that opening and closing brackets should be aligned in a readable way, and we will discuss such styles briefly in the lectures. A good guide would be:

Google C++ Style Guide

but Consistency is what will be checked. Also, it is crucial that you comment your code well. For this first problem sheet, this means that you should explain every single line of code with a comment. Later in this module, it will not be necessary to comment basic code (e.g. what each for loop is doing, as long as it is obvious). But for now, please comment everything. It is your responsibility to make sure the marker can understand what your code is doing. Also, please give variables meaningful names. This greatly increases readability of the code. Finally, the programs you write should run in a self-explanatory way. If you are asked that the program provides some input, it should first have some output on the command line telling the user what kind of input is required.

For instance, if you are supposed to read in the price of a interest rate, do the following:

```
double interest_rate;
std::cout << “ Please enter the interest rate: “ << std::endl;
std::cin >> interest_rate;
```

This way, a user of the program who does not know about the code knows what numbers one has to enter.

PLAGIARISM

Obviously, you are not allowed to plagiarise. This means that you are not allowed to directly copy the code from any online sources or any of your fellow students. Direct group work is also not allowed. While you are allowed (and even encouraged) to discuss the contents of this module (including problem sheets) with your fellow students, every one of you needs to write their own code.

PROBLEM 1

Write a function **almost_equal** that returns a bool value if the two doubles **a**,**b** are within a predefined epsilon (**c**). Your solution should include predefined tests for all qualitatively different possible return cases. Example tests should include:

- small numbers near zero
- numbers between 1-2
- large number against small numbers
- large numbers against large numbers
- negative numbers against positive numbers

the **main** function should automatically run these tests.

$$a \approx b \pm c$$

Output the results on the command line.

PROBLEM 2

Write a function void **clamp**(**const double** a, **const double** b, **double**& c); that takes three doubles **a**, **b**, **c** as an input and returns a **double** which is always between the two doubles **a**, **b**. That is the value double **c** will always be either **a** or **b** or between the two. Your **clamp** function should use your **almost_equal** function.

You should include tests in your code to test the **clamp** function.

$$a \leqslant c \leqslant b$$

In **main**, ask the user either to:

- input of three doubles and then **clamps** the first value between the last two

or

- runs your series of tests. Explaining which test is running and what its checks for with the **clamp** function

Output the results on the command line.

PROBLEM 3

Write the **std::string** function **reverse(s)** which reverses a **std::string s**, Do not use the built-in reverse function. The value returned is **std::string** which the reverse of **s** in the case.

Write the **bool** function **reverse(s,t)** which examines two **std::string**’s **s**, **t** to see if one of these is the reverse of the other. The value to be returned is **true** if this is the case and **false** if it is not.

In main, ask the user for the input of two strings and ask to either

- make the second string **t** the reverse of the first
- tests if the first string **s** is the reverse of the second **t**

Output the results on the command line.