Problem Sheet 3

LM Advanced Mathematical Finance Module

Part B C++ Spring Term 2021

PROBLEM SHEET 3	SPRING TERM, PROGRAMMING IN C++ LECTURE 3
Lecturer	Dr Simon Hartley s.hartley@bham.ac.uk
Of	School of Mathematics, College of Engineering and Physical Sciences, University of Birmingham
Due	Wednesday 12th MAY 2021, 23:59
Weighting	20% of Part B

INSTRUCTIONS FOR THE SUBMISSION

Submit your code on Canvas as zip:

Submit as a single zip file containing all code for the problem.

The source files should compile without errors or warnings on Visual Studio 2019 as installed on the clusters, and the resulting executables should run without problems. **Using debug and x64 build options**. You do not have to submit the executable. You should comment you code and describe what each function does.

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. 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. You may assume any user input will be correct and have no errors meeting the format requested.

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 your own code.

PROBLEM 1 Create Time Series Share monitoring class in C++

Problem

In the analysis of time series data, filtering, the task of identifying and removing values, often needs to be carried out in order to identify both short and long term trends. For instance, rolling average is commonly used with time series data to smooth out short-term fluctuations and highlight longer-term trends or cycles. The first step toward this is to implement this filtering in C++. In C++ transformations can straightforwardly be applied to the data when it is stored in container data structures. This is a very common task, and tools to do so are provided by the STL, which includes templated methods that execute common tasks, such as sorting and selection. The STL

algorithms for doing this can be applied to data stored in the STL containers such as std::vector, std::array and std::list. You will create a class, TimeSeriesTransformations, which implements a several time-series transformation operations. The class should be able to be used to perform common time-series transformations, such as adding or subtracting values to prices and removing undesired values. The algorithms necessary to complete this task can all be accessed in your C++ code by including the Correct STL header file. For example: . We do not expect you in this task to implement any algorithms yourself.

Your programme must be able to read in a CSV file (the full file can be found on Canvas) containing time series data, which contains a column with a timestamp stored in UNIX time (see Lecture 3), and a column of data (floating point doubles) representing a share price labeled X like the following:

```
TIMESTAMP,ShareX
1619120010,61.43814038
1619125010,58.74814841
1619130010,90.10017163
1619135010,100.5352112
1619140010,12.2706639
1619145010,50.17422982
1619150010,21.88813039
1619155010,91.20241001
1619160010,41.79297323
```

When you finish this problem, you should have:

- Implemented a C++ class TimeSeriesTransformations.
- Implemented a program that employs a TimeSeriesTransformations C++ class and functions.

TimeSeriesTransformations

Your implementation of the TimeSeriesTransformations class must:

- Have a constructor which can load a file given the filename (for e.g. “testdata.csv”) and store the time data and the share prices internally.
- Have a constructor which can load the data into the class, where the input is given by two std::vectors representing the time series and the share prices.
- Have functions `mean` and `standardDeviation` which compute the mean and standard deviations of the share prices for the column of data.
- Have functions `computeIncrementMean` and `computeIncrementStandardDeviation` which compute the mean and standard deviation of the increments (the difference between the share price on two consecutive timestamps).
- Have a function `appendPrice` that takes a date and a value, and adds that at the end of the data.
- Have a function `getTimestampsMatchingPrice` that allows the user to filter the data by a share price value and return all of the timestamps that the share price matches that.
- Have a function `printEntriesOnDay` that allows the user to filter the data by a calendar date and returns the timestamps and prices on that day.
- Have a function `removeEntryAtTime` that allows the user to remove an entry by providing the timestamp. Your solution should utilise the remove_if template.
- Have a function `removePricesEqualTo` that allows the user to remove all entries matching a given value.
- Have a function `removePricesGreaterThan` that allows the user to remove all entries above a given value (exclusive).
- Have a function `removePricesLessThan` that allows a user to remove all entries below a given value (exclusive).
- Have a function `printSharePricesOnDate` that allow us to print all share prices a given day, and a function `printIncrementsOnDate` that allows us to print all increments that occur on a given day.
- Have a function `saveData` that is able to write the data (including any new data) to a filename given by the user.

This list is not exclusive - you may need to implement other functions in order for your class to function correctly.

Program

To demonstrate that your class works you will need to implement a program which uses it, and which presents a menu to the user. Initially, the user should be asked to enter a filename which contains the data. Then, menus should be shown corresponding to those below (where appropriate):

Main Menu

```
Please Choose:
1. Print all share prices on a date
2. Print mean and standard deviation of prices
3. Print mean and standard deviation of price increments
4. Add a share price
5. Remove a share price
6. Filter share prices by date
7. Find greatest increment in prices
8. Save the data to file
```

** Expected Menu “Print all share prices on date” **

```
Please enter date in the format YYYY/MM/DD
```

** Expected Menu “Add a share price” **

```
Please enter date and time in the format YYYY/MM/DD-HH:MM and the corresponding share price.
```

** Expected Menu for “Remove a share price” **

```
Please enter either:
1) Remove value at timestamp
2) Remove all prices above share value
3) Remove all prices below share value
4) Remove all prices equal to share value
```

HINTS

There are lots of functions in the **<Algorithm>** header file which you can use:

```
  copy, copy_backward,
  for_each,
  find_if,
  count , count_if,
  transform,
  fill,
  reverse,
  sort,
  binary_search,
  min_element, max_element
```

In addition, other useful tools are found in the **<functional>** header file:

```
 bind,
 plus,
 minus,
greater_equal
```