ICT209 Assignment 2 2016

Objectives:

- Demonstrate that you can do Object Oriented design.
- Demonstrate that you can write Object Oriented programs using C++.
- Demonstrate that you can design and write programs using user defined data structures.
- Demonstrate that you can use data structures appropriately.
- Demonstrate that you can write test plans and show evidence of systematic testing.
- Demonstrate that you can draw UML diagrams.
- Acquire further experience with design and code maintenance.

You do not work in groups for this assignment as this is an individual assignment.

Worth:

30% of the unit

Due:

Midnight, end of Session 12 (last teaching week). See due date and time (and any amendments to these) in LMS. The due date and time depends on when is the last day of the last teaching week for the semester or trimester at the campus you are enrolled in.

How to submit:

Same way as assignment 1. The submission must contain the entire directory/folder in which the source was built. Other submission items are specified in this document.

Non-Murdoch Campus:
Into the assignment submission area for the unit in LMS.

Murdoch Campus Internal students:
Into the assignment submission area for the unit in LMS.

Externals:
Into the assignment submission area for the unit in LMS.

For submitting in LMS, zip up the entire folder. Make sure that you have included all needed files. Do not include temporary files or files not relevant to the assignment. After submitting the assignment, log out of LMS. Then log back in again. Check that the assignment is submitted.

Name the zip file with the unit code, Assignment number, your name, student number.
ICT209Asg2JoBlogs12345678.zip
or alternatively,

ICT209_Asg2_JoBlogs_12345678.zip

Textual submissions should be type-written. External documentation can only be in the following formats:
Text (.txt)
PDF (.pdf)
RTF (.rtf)
HTML (.html)
Image formats : PNG, GIF, JPG, TIFF, BMP. (BMP and TIFF cannot be used for Web documents)

Assignment cover sheet requirements are listed in the unit outline found in the Admin area. Also see the end of this document.

Mandatory Readings/work needed:

- Unit textbook. "*C++ Programming: Program Design Including Data Structures*" by D.S. Malik.
- Lecture notes.
- Complete lab exercises.
  - You need to complete all exercises up to laboratory session 10 to cover the data structures requirements of this assignment.
  - Complete laboratory session 11 before submission of this assignment.
- Question and Answer files (**QandA**) if these exist in the assignment 1 and assignment 2 area. This QandA file for assignment 2 can be updated on the day of the assignment submission if someone asks a question on that day and the question and answer is deemed suitable for sharing with everyone.
- C++ Coding Standards http://0-lib.myilibrary.com.prospero.murdoch.edu.au/Open.aspx?id=291948

Assignment Question:
Design an object-oriented solution and implement the solution in C++ to solve the problem described below. You must read the entire specification and work out what is needed before starting work on this assignment.

Like assignment 1, this assignment is not for actually buying and selling of shares. The assignment only deals with data from past share transactions (trades).

You need to read the whole assignment specification first. The solution needs to be worked out on paper before you start coding.

Extend assignment 1 to do the tasks described below. You need to also reflect on the design and code changes needed to convert your assignment 1 into assignment 2.

Start early. Design the menu system and test to make sure the menu works. Use code stubs (refer to your earlier units) to test the overall flow of execution. If you have forgotten what a stub is, see

http://en.wikipedia.org/wiki/Method_stub for a quick overview. You already have data file reading code, so make sure that you are able to read data files correctly. In the initial stages, read the files and print them out to check. Once this stage is completed, you start using the data structures after first unit testing the data structures.

Create your own test data files as given in these specifications below. You will be submitting these files.

Make sure you design a simple menu so that users do not get confused or annoyed – consider what it is like if you were the end-user.

Your program should be able to deal with course of sales data for multiple days and multiple stock exchange codes (*x-code*). Each day's trading data for each stock code (*x-code*) is still stored in CSV files. This means that there will be a number of data files for each stock code. For proper testing (for the various test conditions), you need to create your own data files. These test data files shouldn't be large. There should only be enough data to check the test conditions. This involves various *x-code* values and dates. You decide on these values and the decision is important as the data values are used to test your solution.

Program input:
The data format in each course of sales file is the same as assignment 1. As there are multiple *x-codes*, the date files are arranged in directories or folders. The folder names are the *x-codes*. To illustrate, assume there are course of sales data for 3 companies with their own *x-codes*. Assume the *x-codes* are IAG, NAB, CBA. The data folder will have 3 folders called *"IAG"*, *"NAB"*, and *"CBA"*. This data folder will have an index file called *code_index.txt*. The contents of this index file would just be the *x-code* names whose folder exists in the data folder. The folder names match the *x-code* names.

So, for this example, *code_index.txt* will contain each *x-code* on a line:
IAG
NAB
CBA

The contents of *code_index.txt* indicate that there are these 3 sub-folders in the data folder.

Do not make any assumption that the *x-code* is sorted in *code_index.txt*.

The course of sales files (in CSV format, as in assignment 1) for each day and given *x-code* will found in the sub-folder of the data folder. As indicated earlier, the sub-folder is named by the *x-code*. Do not make any assumptions about the naming conventions used for the course of sales files. The names of the files in the *x-code* sub-folders are listed in a file called *sales_index.txt*. Each *x-code* subfolder will have a *sales_index.txt* file.

Assuming that the folder *IAG* has course of sales files *day1.csv*, *day2.csv*, *day3.csv*, the sales*_index.txt* file in folder IAG will contain the following lines.
day1.csv
day2.csv
day3.csv.

Do not assume that there is any sort order in *sales_index.txt*. The names of the CSV files do not indicate any chronological order. Arrange your data files to reflect this – change the order in which the files are listed and check.

So to read course of sales data into your program:
In the data folder, read the *code_index.txt* file.
For each code in *code_index.txt*
    Read code/*sales_index.txt*
    For each csv file in code/*sales_index.txt*
        Load the csv file data into your program.
    Endfor
Endfor

The program reads all files and loads the appropriate data structures before the menu is displayed to the end user.

Program output:
The program reads data from the data files and produces output according to the menu option selected by the user. As alerted in assignment 1, assignment 2 deals with multiple stock codes and course of sales for multiple days. Consequently, most of the menu options are variations of what was needed for assignment 1.

Date is shown as *dd/mm/yyyy*. (02/01/2014).

Time uses the 24-hour format.

The stock code is the *x-code*.

Read the requirements for each option, and design your menu system to be user friendly. Your design may have some of the menu options listed below as sub-menus.

Before any menu option is displayed, the program reads all data files and loads the appropriate data structures.

Menu option 1:
The highest share price and start time(s) of the highest share price during the day for a user specified stock code and date. This is printed on the screen in the following format:

Stock code: <the stock code>
Date: <the transaction date>
Highest price: <the highest price>
Start time(s):
<time when highest price transaction occurred>
<time when highest price transaction occurred>
...

The highest price could be reached more than once during the day. If so, list each time on a new row. Do not list duplicate time values.

Menu option 2:
The lowest share price and start time(s) of the lowest share price during the day for a user specified stock code and date. This is printed on the screen in the following format:

Stock code: <the stock code>
Date: <the transaction date>
Lowest price: <the lowest price>
Start time(s):
<time when lowest price transaction occurred>
<time when lowest price transaction occurred>
...

The lowest price could be reached more than once during the day. If so, list each time on a new row. Do not list duplicate time values.

A value of 0 for price should not be part of the output.

Menu option 3:
This option is for a user specified stock code and date.

The output goes to a file called *price-change-date.csv* where the fields (individual items of data) are separated by commas. The date part of the file name depends on the user specified date. The format is yyyy-mm-dd. As an example, an output file can be *price-change-2014-02-10.csv*. This is the output for the 10th of February 2014. This file is created in the specified *x-code* directory.

The output format is:

Stock code, Date, Start time, Price of share, Volume of shares traded, Total value of shares traded

Make sure you write the header row. This is the first row of the output file. This involves simply printing the format line above as a string.

The data file will have a record (row) for each time the share price changes. The output will have data arranged in increasing time order.

The output fields have the following meanings:

- Stock code: The *x-code.*
- Date: Date of trading.
- Start time: The time when the share price changed to the value indicated in the "Price of share" field. This time value is used to indicate that the share price changed to a new price value. The input data file may have multiple consecutive records (rows) where the share price is the same. This means that there may be multiple consecutive share transactions where the share price remained the same. In the output data file, *output.csv*, the Start time field records when the share price changed to a different value. The intention here is that this field records the time of initial change to a price in a series of consecutive same price transactions.
- Volume of shares traded: The number of shares traded since "Start time" at the value indicated in the "Price of share" field.
- Total value of shares traded: The total monetary (dollar) value of the shares given in the "Volume of shares" field.

Treat a value of zero (0) for the price field in the input data file the same way as other prices are treated.

Menu option 4:
This menu option is to be completed only when all other menu options are working as expected.

Proper completion of this option will give you a 15 marks bonus. **If you attempt this option when any of the other options are not working, no bonus marks would be awarded.**

It is theoretically possible to get 115 marks if all options are done perfectly. However the maximum mark that will awarded is 100.

Calculate a simple moving average (SMA) for the price of a user specified stock code on a given date. The moving average is over 5 transactions by default. The output goes to a file named *SMA-date.csv*. The date part of the file name depends on the user specified date. The format is yyyy-mm-dd. As an example, an output file can be *SMA-2014-02-10.csv*. This is the SMA for the 10th of February 2014. This file is created in the specified *x-code* directory.

The output format is:

Stock code, Date, Time, Price of share, simple moving average.

Make sure you write the header row. This is the first row of the output file. This involves simply printing the format line above as a string.

Stock code and Date have the same meaning as in option 3. Time is the recorded time in the course of sales file. Price of share is the price recorded in the course of sales file. The SMA is described at http://www.investopedia.com/terms/m/movingaverage.asp. Use the approach described at this site but instead of using daily closing prices, the actual transaction price at a given time recorded in the course of sales file is used.

Provide a sub-menu of this menu that enables the default of 5 transactions for the moving average to be changed to some other value.

Price values of 0 must not be averaged.

Menu option 5:
Exit the program

Data Structures requirement:
STL data structures and algorithms can be used in this assignment. You must use the Binary search tree data structure and the STL map (and/or multimap) data structure.

The data for the output in the various menu options must come from these data structures. You may use other data structures together with the STL map and tree.

You need to identify where these data structures can be used and provide a rationale for their use. There is a 30 marks penalty if you do not do this.

You may use std::string and string stream classes in your program instead of using C like strings. You may use iostream and file handling classes and objects in C++. See laboratory exercises.

When working on this assignment, think of the usability of your program from the point of view of the user. This is the first thing you have to do, even before you start designing the program. This will form the menu system for the running program. You should not make the program more complicated than what is specified in the assignment. Make sure you provide an option to exit the program. It is not a good idea to exit a program by killing the program or by re-booting the computer.

Any advice and further clarifications to these requirements would be found in the QandA file in the assignment 2 area. Please ask your questions early and do not wait till near the due date as you may not have time to incorporate any answers into your assignment.

Documentation: *(Printed versions apply only when there is a notification in LMS asking for hard copies. Ignore the advice below for printed copies if there is no notification in LMS)*

- UML diagram showing the design of your classes. (printed and soft copy)
- Reflection on the changes you had to make to cater for the requirements of assignment 2. You describe what parts of your assignment 1 design and code enabled you to complete the requirements of assignment 2 easily. What parts of your assignment 1 design and code were a problem? (printed and softcopy)
- Rationale for the use of the data structures in the program. (printed and softcopy)
- A high level algorithm for the solution. (printed and softcopy)
- Doxygen output which shows all information as was done in the practice for session 2. (soft copy only)
- Test plan and output of the test run(s). (soft copy only)

Do not print code. Code will only exist as soft copy. You need to submit the entire directory/folder where you built your program. All relevant data files need to be there and a way must be provided to build with either Microsoft Visual Studio or Code:Blocks on a Windows operating system.

Minimum requirements:

You must provide all of the following; otherwise 50 marks will be deducted.

- UML diagram
- Reflection on the design and code conversion from assignment 1 to assignment 2.
- Written rationale for the data structures: tree, map/multimap and any others used.
- Doxygen output
- Program that builds (using Microsoft Visual C++ or code::blocks) and runs. Build mechanism must be provided.
- Source code. Doxygen style comments in header files. (soft copy only)
- Test plan and output of test run(s)
- Executable program with associated data files in a separate directory called "*executable*". Make sure that the executable runs on a machine which does not have a compiler. Data files needed to enable it run should be included here.

- A declaration indicating what works and what does not work in your program. This can form the summary of your test plan and output of test runs and should be provided as a separate document called "***evaluation.txt***". This is an unformatted text file and it is not the test plan or output of test runs.

Marking

UML diagram	10%
Design reflection and rationale for data structures	20%
Program (includes coding style and comments)	55 % + *15% bonus for menu 4*
Test plan and testing	15%

There is a 10 marks penalty if the cover sheet is not filled in properly and/or the requirements of the coversheet are not met. If the cover sheet is not provided, you get no marks. There is a separate cover sheet available.

If progress on this assignment is not demonstrated to your tutor in the weeks prior to the submission of the assignment, you may not get any marks for this assignment. We need to know that you have been working on the assignment. The assignment must not "magically" appear. Please check the unit guide for the requirements for passing the unit – section on "Determination of the final grade".

Penalty summary:

Specified data structures not used: -30 marks
Missing minimum requirements: -50 marks
Cover sheet problems: -10 marks (cover sheet is not required now for LMS submissions, but you should ensure that all requirements are met for the cover sheet and submission into LMS.)

The lowest total mark you can get is 0.

MURDOCH UNIVERSITY
PERTH, WESTERN AUSTRALIA

School of Engineering and Information Technology

ICT209 COVER SHEET used for Non-LMS submission

Given Names	Surname	Student Numbers	Mode (D/X)	Email

Name of tutor: _____ **Day & Time of tutorial:** _____

Assignment Number: __2___ **Due Date:** _____ **Date Submitted:** _____

If the given name by which your tutor knows you differs from your name on university records, you should indicate both names above. Tutor's name must be entered. Penalty is 10% of the total marks for the submission for not providing information asked for.

Your assignment should meet the following requirements. Please confirm this (by ticking boxes) before submitting your assignment that you have checked the declarations you need to make. ***Some items do <u>not</u> apply for online/electronic submission.***

- ☐ Except where I have indicated, the work I am submitting is my own work for the purpose of this assessment and has not been submitted for assessment in another unit.
- ☐ This submission complies with Murdoch University's academic integrity commitments. I am aware that information about plagiarism and associated penalties can be found at http://www.murdoch.edu.au/teach/plagiarism/. If I have any doubts or queries about this, I am further aware that I can contact my Unit Coordinator prior to submitting the assignment.
- ☐ I acknowledge and agree that the assessor of this assignment may, for the purpose of assessing this assignment:
  - Reproduce this assignment and provide a copy to another academic staff member; and/or
  - Submit a copy of this assignment to a plagiarism-checking service. This web-based service will retain a copy of this work for the sole purpose of subsequent plagiarism checking, but has a legal agreement with the University that it will not share or reproduce it in any form.
- ☐ I have retained a copy of <u>all</u> submitted work.
- ☐ I will retain a copy of the notification of receipt of this assignment. LMS submission would provide the required receipt. If you have not received a receipt within three days, please check with your Unit Coordinator.
- ☐ Assignment is presented on A4 size paper and is neatly collated. (Internal students)
- ☐ Assignment includes virus-free disk with machine-readable programs & files relevant only to this submission. CD or DVD only may be accepted. (Internal students)
- ☐ Instructions relating to answering of questions, formats used for electronic submission and LMS submission have been followed.
- ☐ Writing is clearly legible or has been printed.
- ☐ Pages have been firmly stapled. (Internal students)

_____
Signature for submission via the assignment box (internal students)