Individual Assignment Task

Financial Charting Program

A C++ console program is required to chart and analyse the historical price data for a financial market. After analysing the market price data and calculating the output, the program should plot the corresponding candlestick chart first, with a volume bar-chart underneath and a financial indicator of your choice underneath that (e.g. Simple Moving Average (SMA), Weighted or Exponential Moving Average (EMA), Moving Average Convergence/Divergence (MACD), Relative Strength Index (RSI), Mayer Multiple – Price divided by Moving Average, other well-known financial indicator, etc.).

The program should read in the historical input data from a Comma Separated Value (.csv) plain text file. The input file will usually contain 3 months' worth of data points (about 90 days), but ideally your program will be able to cope with varying amounts of input data or invalid file formats.

Finally, the program should give the user the option of outputting the financial plots to a separate plain text file (.txt) and an option to re-run the program with a new input filename.

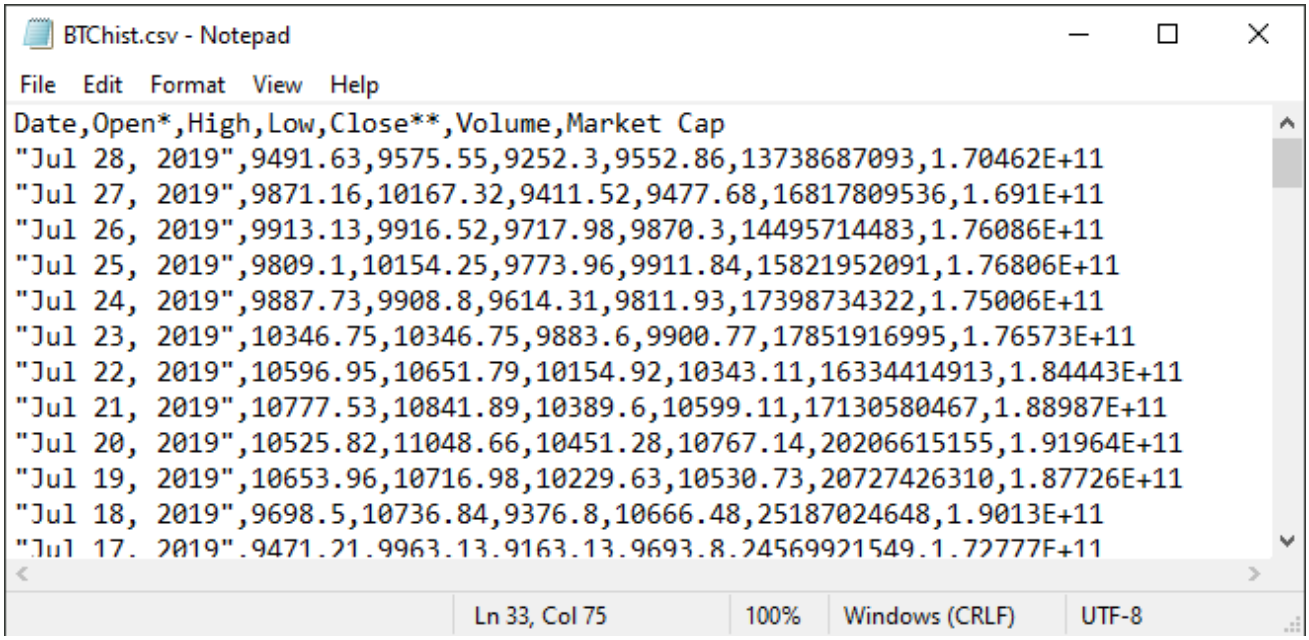


Figure 1: An example text file containing input values.

(Data pulled from: https://coinmarketcap.com/currencies/bitcoin/historical-data/ with currency in USD)

Figure 1 shows the **exact** format the input file data will be in. Be very careful to note where the commas appear to separate the columns. Note that with the first column for 'Date' a comma appears in the middle of the data itself and this should be part of the read data – not a new column.

You can find an example input file download along with this coursework task sheet on Moodle.

Each student should work **individually** on this coursework. In support of this, each student should solve the problem for a different financial indicator of choice or with significant changes to the calculation, output and implementation. Any cases of collusion, as in similar source code submissions, will be forwarded to Academic Conduct in the normal process.

Figure 2: A fully-featured candle stick chart with volume, MA and RSI plotted underneath.
(Chart data pulled from: https://uk.tradingview.com/chart/Ur8Pxvgg/)

Figure 2 shows an example of a fully-featured browser-based charting system. We will not be building something this complex, but hopefully it helps to illustrate our programming task.

- The first chart is called a candlestick chart, which shows the daily price action.
- The second chart shows the volume of orders completed on that day as a bargraph.
- The third chart shows two simple moving averages calculated from the price data, one MA is over a 9-day period and the other MA is over an 18-day period.
- Finally, the fourth chart shows the Relative Strength Index (RSI) calculated from the price data.

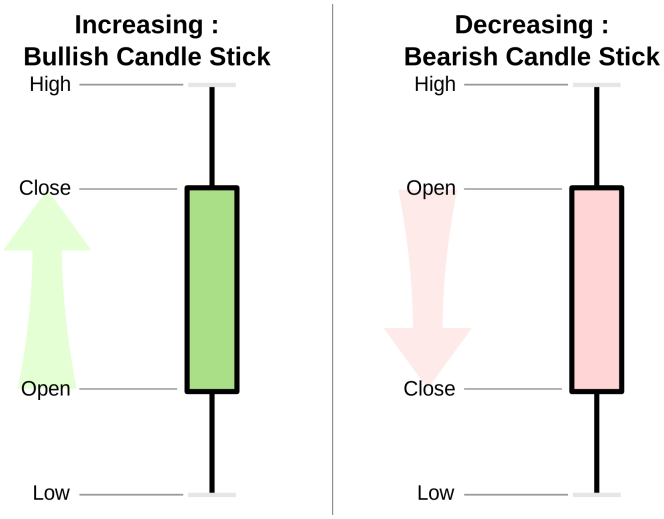


Figure 3: A representation of a candle stick.
(Candle stick image pulled from: https://en.wikipedia.org/wiki/Candlestick_chart)

Figure 3 illustrates how candlesticks are drawn according to market price data over a specific timeframe (e.g. daily). There are 4 parts to a candlestick: Open, High, Low and Close price. When the Close price is greater-than Open price then this is referred to as a Bullish candlestick. When the Close price is less-than the Open price then this is referred to as a Bearish candlestick.

ASCII art should be used to draw the candlestick chart for the console program. There is some flexibility in how to do this, but Figure 4 is given as an example of what your console program might look like when running.

The ASCII art candlestick chart could be plotted in a in a vertical instead of horizontal manner if you choose to.

The solid body of the candlesticks can be represented using the extended ASCII characters, e.g. char(176), char(177), char(178), char(219) to represent the green and red bodies of bullish and bearish candles. The axis can be drawn also using the extended ASCII characters, e.g. char(192), char(194) and char(196).

After plotting to the console, your program should prompt the user if they want to save the plot to a text file of their choice.

Finally, your program should have a re-run option at the end to allow the user to repeat the whole program and select another input file as required.

Figure 4 is given as a starting point in your analysis of the problem and shows an example of what your program should look like when running.

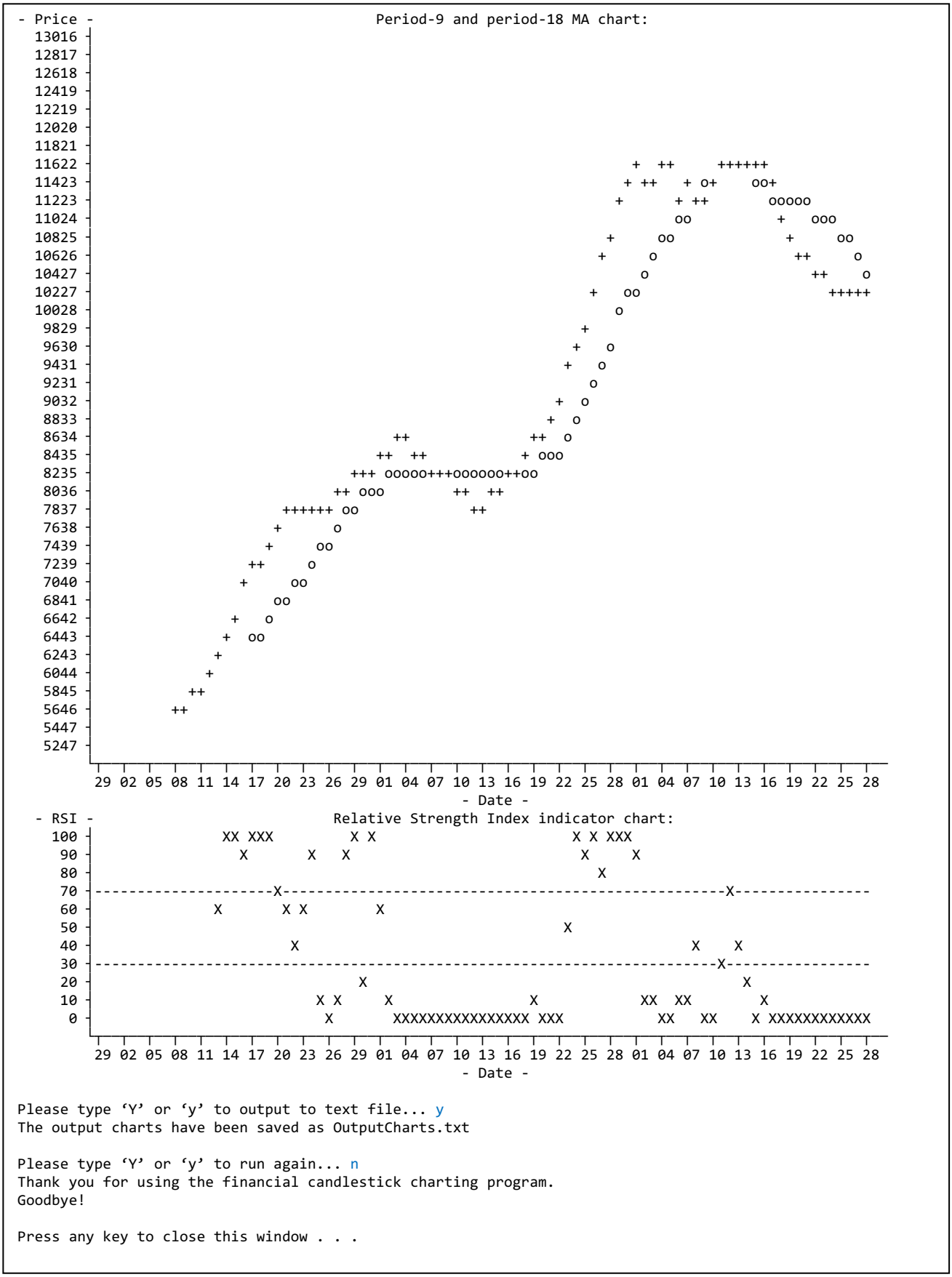
```
Provide an appropriate title.
Display your name and SID number.

Please enter the input filename:
BTChist.csv
File BTChist.csv read successfully...
```


```
MA or RSI or another financial indicator of your choice:

Continued next page...
```

Figure 4 – continued next page…




```
Please type 'Y' or 'y' to output to text file... y
The output charts have been saved as OutputCharts.txt

Please type 'Y' or 'y' to run again... n
Thank you for using the financial candlestick charting program.
Goodbye!

Press any key to close this window . . .
```

Figure 4: A console diagram showing an example of the expected console output.

Before writing the program, a methodical analysis and design must be completed including:

1. UML activity diagram at high-level to show an overview of the interaction between the user and the program (use swimlanes to show the user and system interaction).
2. UML activity diagram at low-level to show the details of how to "calculate the axis values for the candlestick chart".
3. UML activity diagram at low-level to show the details of how to "plot a row of the candlestick chart to the console".
4. UML class diagram to show the class design, attributes, methods, and any class relationships.

After writing your program you must create a screencast video clip, max. 3 minutes in length, where you demonstrate your program running and briefly go through and explain which features you have implemented successfully. You do **not** need to explain your source code in the screencast.

For a pass your UML, program and video should meet and demonstrate the following requirements to a suitable standard:

1. All UML diagrams are completed.
2. The program builds and runs without errors.
3. The basic program functionality is achieved using at least a single object of a single user defined class (e.g. Candlestick).
4. The program output is a close match to the console diagram and displays your name and SID clearly at the top.
5. A plain text CSV filename can be entered by the user at runtime and the data inside the file can be read and analysed successfully.
6. The output answer is calculated correctly for any given input CSV text file (in the correct format) and the corresponding plot is accurate.
7. A plain text filename can be entered by the user at runtime and the output charts can be saved in the plain text file.
8. The program can be re-run multiple times without errors.
9. Neatly written program code with consideration to the use of tabbing, suitable naming conventions, structured appropriately, use of private attributes and public methods.

For a first your UML, program and video demonstration should additionally meet the following requirements to a high standard:

1. Output from the program is fully formatted and correct; making use of the <iomanip>.
2. Fully implements all the features, correct calculations for axis increments, handles differing input data, implements some of the more advanced financial charts e.g. RSI or MACD etc.
3. Good use of arrays or library data structures (e.g. std::vector) for storing the data before calculating and plotting. May use a vector of objects.
4. Use of multiple user defined classes or types. Data is passed between objects as appropriate and the class diagram displays the correct relationships.
5. Efficient code design, code reuse being used instead of repeating code.
6. Good use of C++ standard libraries e.g. cmath, fstream, string, stringstream, vector, limits, algorithm, etc.
7. More thorough use of error checking on input from keyboard and from input files, e.g. incorrectly formatted inputs, etc.
8. Appropriate use of references and/or pointers such as 'passing by reference'.
9. Passing objects between functions/methods.
10. An attempt at proper code documentation, no linting errors under static code analysis, and good practices being used e.g. private attributes, meaningful identifiers, verb or verb phrases for functions and methods, nouns for class names and variable identifiers.

Assessment

The individual assignment is a summative assessment worth 50% of the module mark.

Submission

1. Submission via Coventry Uni. GitHub: https://github.coventry.ac.uk/205SE-1920SEPJAN
2. Ensure your repository is named as such: 205SE-1920SEPJAN/<your username>-<your project name> e.g.: 'github.coventry.ac.uk/205SE-1920SEPJAN/aa6164-Cw-FinChart'.
3. Your source code repository **must** be part of our GitHub class: 205SE-1920SEPJAN.
4. Your repository must be created from Visual Studio such that there is a ready-to-run Visual Studio Solution and Project files in the repository along with the submitted source code.
5. Ensure your repository is private.
6. Make regular commits before the deadline due date. Commits after the due date will not be marked.
7. Include your full Visual Studio project, source code files (.cpp), and header files (.h).
8. Upload your UML diagrams to Moodle.
9. Upload a link to your GitHub source code repository on Moodle.
10. Upload a link to your video evidence on Moodle

You can submit any time before the deadline.
Please note that any source file may be scanned by plagiarism detecting software.
Late submission will incur a mark of zero.

Programming Notes

You must format the results from the program to produce an output similar to that shown in the example console diagram. This will require both the <iostream> and the <iomanip> standard libraries. Figure 5 shows example usage of library statements to output the text "77" into a width of 10 characters wide with 'x' characters as padding.

```
cout << setfill('x');
cout << setw(10);
cout << "77" << endl;
```

```
output:
```

```
xxxxxxxx77
```

Figure 5: Output manipulation using set fill and set width library functions.

This is a particularly useful function for getting correct spacing when creating the plot. Be aware that the DOS console screen is 80 characters wide. The same manipulators can be used when writing to text file. Writing to a text file output will require the use of the <fstream> library.

For more information on how to use standard C++ libraries please refer to online documentation:
https://msdn.microsoft.com/en-us/library/ydd54a6t.aspx
http://en.cppreference.com/w/cpp/header/iomanip
http://www.cplusplus.com/reference/iomanip/

Before you can do any calculations, your program must read data values from a text file. In C++ this is almost identical to reading in values from the keyboard using cin. We must use an input file object (type ifstream) instead of using the cin object.

Note for higher marks you want to be avoiding code duplication using clever class design and method usage – note that both 'cout' and 'ofsteam' are both sub-classes of the 'ostream' type.

Possible class design:

- 'FinanceProgram' or 'FinanceDirector' – the controlling class – called directly from the main function and controls all the high-level aspects and steps of your program for example high-level interaction with the User. Interfaces with the 'CandlestickChart' object, for example when needing to do something related to the charting data.
- 'CandlestickChart' or 'CandlestickManager' – Responsible for managing a data structure containing multiple candlestick objects making up a full candlestick chart or a full CSV data file. Responsible for looping through the input CSV file and storing data into multiple candlestick objects in a data structure. Responsible for drawing the candlestick chart and maybe other charts which operate on the same data.
- 'Candle' or 'Candlestick' – Responsible for encapsulating and managing the data for a single candlestick object, i.e. date, open, high, low, close, volume, marketCap. Behaviour could include parsing a line from the CSV file from a CSV string to the candlestick data variables. Could also be made responsible for plotting a section of the candlestick.

The limits library is useful for accessing the largest and smallest possible values that you can store in a primitive type e.g. a double:

```
#include <limits>
std::numeric_limits<double>::min();
std::numeric_limits<double>::max();
```

The algorithm library is useful for finding the min or max between two numbers e.g.:

```
#include <algorithm>
std::max(1, 2) == 2 // true
std::max(2, 1) == 2 // true
std::max(0, -1) == 0 // true
```

The string stream library is useful for constructing strings, parsing through CSV formatted strings or converting from a string to an int or double.

```
#include <string>
#include <sstream>
stringstream ss(fromCSV);
string field;
getline(ss, field, ','); // parse CSV formatted string
stringstream fs(field);
fs >> open; // try to convert from string to a double
```

As you are developing this program you will run into programming errors. Throughout development start with program which compiles first before making changes and then make sure it compiles again before adding any more. Use cout to output to the console regularly to check values, use debugging features in VS, and step through the lines with the IDE to find the errors.

Steps to Get Started on the Coursework

First clone the starting repository into your own new private repository for the coursework.

Starting repository:
https://github.coventry.ac.uk/205SE-1920SEPJAN/CW-STARTING-POINT

Steps to clone the starting repository and use of CU GitHub with Visual Studio 2019:

1. Open CU GitHub.
2. Sign-in using your normal university login to CU GitHub and navigate to our 205SE class page here: https://github.coventry.ac.uk/205SE-1920SEPJAN
3. Create a new remote repository under 205SE-1920SEPJAN named as such: 205SE-1920SEPJAN/<your username>-coursework e.g.: 'github.coventry.ac.uk/205SE-1920SEPJAN/aa6164-coursework'.
4. Your source code repository **must** be part of our GitHub class: 205SE-1920SEPJAN.
5. Your source code repository **must** be private.

6. Open another tab and navigate to: https://github.coventry.ac.uk/205SE-1920SEPJAN/CW-STARTING-POINT
7. Click clone or download and copy the URL.

8. Open Visual Studio 2019.
9. Go to the Team Explorer tab.
10. Click the green plug 'connect' icon.
11. Click clone.
12. Paste the URL we copied.
13. Open the repository we cloned from Team Explorer by double clicking it.
14. Click Settings → Repository Settings.
15. Remove the remote repository.
16. Click home icon → Sync.
17. Under 'Push to **Remote** Repository', click the 'Publish Git Repo' button.
18. Copy your new empty repository URL and paste here under 'Publish Git Repo' and click 'Publish'.

19. Click Home icon → Solutions → Double click the solution 'CandlestickCharting.sln'.
20. Open the 'Solution Explorer' tab.
21. Open the project dropdown to view the 'Source Files'.
22. Open 'CandlestickCharting.cpp'.
23. Find the 'Author' and 'SID' in both the comments at the top and in the source code for title and update them to your own.
24. Go back to the Team Explorer tab → Changes.
25. There should be a new change to commit, add a commit message and commit.
26. Home → Sync → Push to send the commit to your remote repository.
27. Repeat from step 23 to make further changes to your source code, make regular commits and pushes to save your work and practice proper source control.