CM2005 Candlestick Assignment

Introduction

In this assignment, I was tasked to edit an existing program to be able to compute Candlestick data, as well as to make a text based plot of it. To that end, I have successfully updated the program to work with the dataset given by the school. I would like to note that my due to different hardware specifications, the program will take awhile to compute a large dataset.

Most of the information about this project can be found in the `README.md` file.

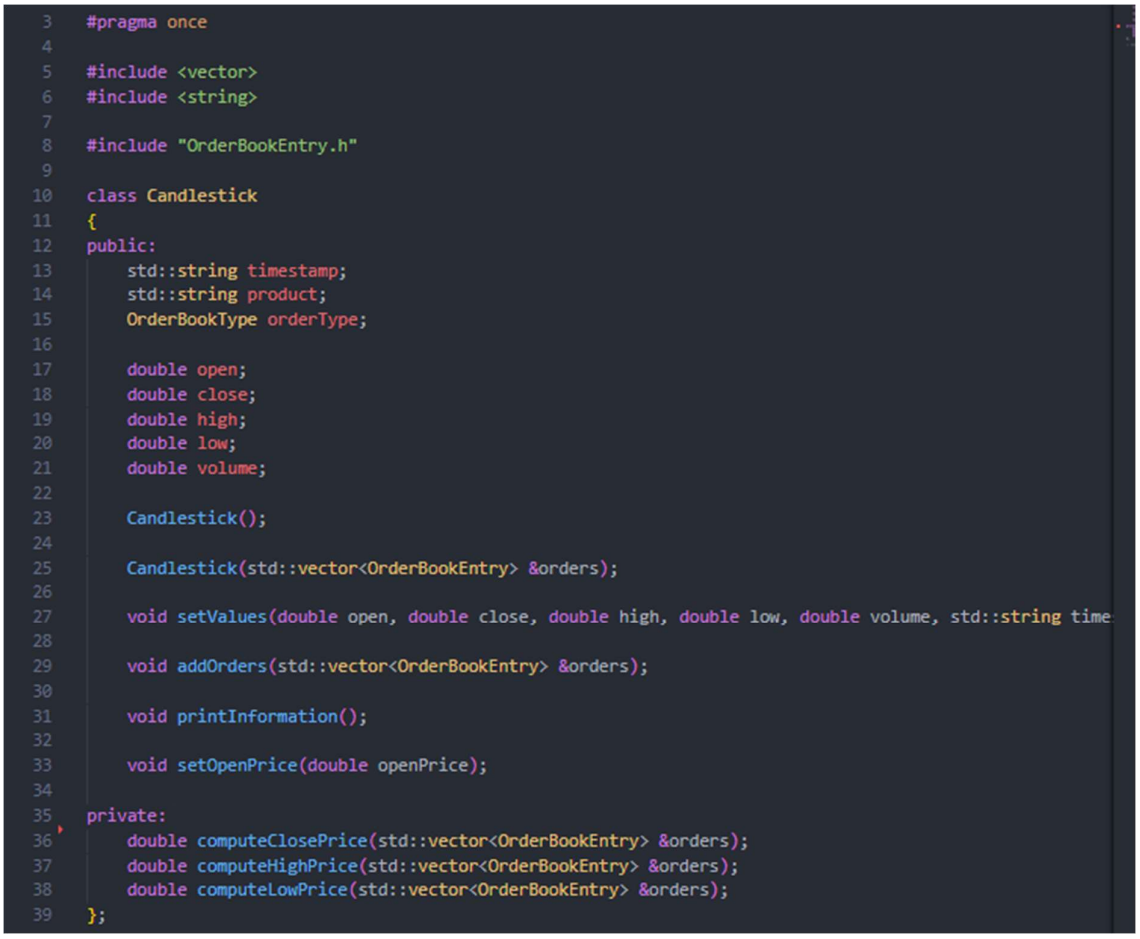
I will be writing this report mostly in the order of progress. The functions that I have written also come with comments as well as Code Documentation to help identify its purpose and flow.

Task 1: Compute Candlestick Data

In this section, I will detail about the changes and modifications I have made to the program, such that it was able to accomplish the task.

Candlestick Class

I have created a Candlesticks Class to contain the Candlestick data required by Part 1 of the project. The candlesticks contain multiple variables to store the required data, as well as different functions to help compute certain elements.



```
#pragma once

#include <vector>
#include <string>

#include "OrderBookEntry.h"

class Candlestick
{
public:
    std::string timestamp;
    std::string product;
    OrderBookType orderType;

    double open;
    double close;
    double high;
    double low;
    double volume;

    Candlestick();

    Candlestick(std::vector<OrderBookEntry> &orders);

    void setValues(double open, double close, double high, double low, double volume, std::string time

    void addOrders(std::vector<OrderBookEntry> &orders);

    void printInformation();

    void setOpenPrice(double openPrice);

private:
    double computeClosePrice(std::vector<OrderBookEntry> &orders);
    double computeHighPrice(std::vector<OrderBookEntry> &orders);
    double computeLowPrice(std::vector<OrderBookEntry> &orders);
};
```

For Part 1 and 2 of the project, Candlesticks are created by passing in a set of orders. With that set of orders, the candlestick will start to compute the High, Low and Close prices. It will also start to populate other information based on the orders given to it. As the Open price depends on the previous' candlestick's Close price, the Open price should be appended to the candlestick afterwards.

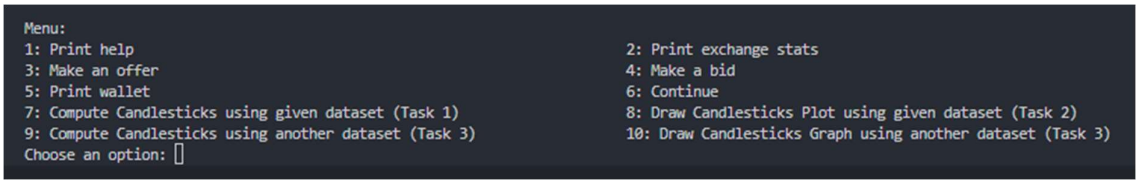
Candlestick objects are then stored in either a CandlestickBook (Task 3) object or the MerkelMain class.

MerkelMain Class

Most of the program runs from the MerkelMain class and can be considered as the main brain of the program. Certain modifications have been made to improve and adapt it to work with new classes and datasets.

Trimming Outputs

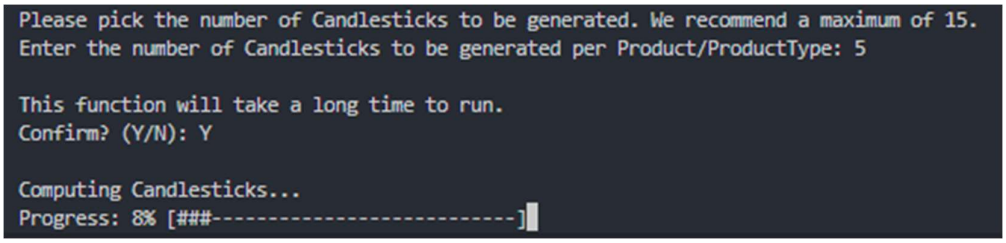
In this class, I have simplified many of the console outputs to make the console more streamlined, and not be littered with excessive outputs. With that in mind, I have also reduced the Menu to be printed into 2 columns as well. This would help cater for users who use a shorter terminal.



Computing Candlestick Data

As you might have noticed above, I have added new options to the menu as well, specifically options 7 to 9. I will be focusing on option for this segment.

Option 7 was created to handle Task 1 of the coursework assignment, whereby we were to compute the Candlestick data. To do so, I have created a function under MerkelMain, ComputeCandlestick(). The function will ask the user to choose the number of Candlesticks to generate, per type, per product, based on all the orders stored in the order book.



As shown in the example above, if the user chooses 5, the program will compile 5 ask candlesticks and 5 bid candlesticks per product. This is important for a later part, whereby the user selects option 8 in the menu.

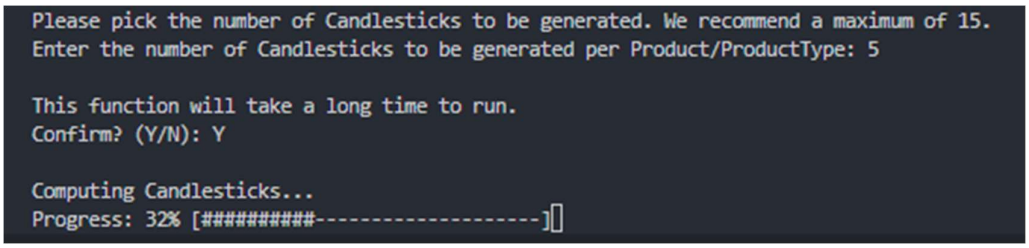
As the default dataset provided by the school is extremely huge, depending on hardware specifications, it could take over 5 minutes to compute. This function is an optional function as other parts of the application can be executed without computing the candlesticks. As a result, I have opted not to run this at the start of the program, and only at the user's request.

I have also added a confirmation dialog for this program to warn the user that the program could take a long time to execute. To prevent users from thinking that the program has gone unresponsive, I have also added a progress bar to this function, and I will explain it in the next section.

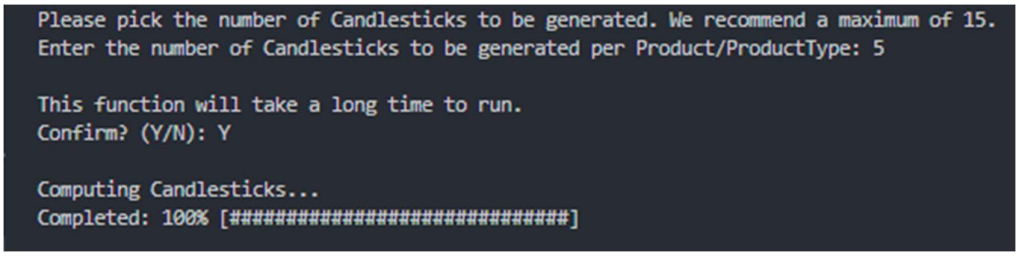
Progress Bar (Enhancement)

As mentioned earlier, the program will take awhile to compute a large dataset. The computation of the dataset is done in the menu option 7. If the program takes a long time to compute, the program might look like it has gone unresponsive, and the user might prematurely terminate the program.

To avoid this, I have added a Progress Bar to the computation of the dataset, such that it will dynamically update itself over time.



With a progress bar, the user can tell that the program is responsive and still running and might be more willing to wait for the program to finish. The program will continue to run until it hits 100%.



As you can see from the above few screen captures, the progress bar will progressively fill up the bar, and update the completion rate at the left. It will not remove any logs above it by excessively flooding a progress counter either.

I felt that this is an appropriate enhancement for my project as it dynamically updates, instead of simply printing a completion rate over and over. It also lets the user know that the program is running without flooding the terminal.

Task 2: Create a Text-Based Plot of the Candlestick data

As required by the project, I am required to create a text-based plot of the Candlestick data computed in Task 1. To achieve this, I created 2 new functions in MerkelMain class, and I will be explaining these functions.

Making a request for a Plot

In the menu provided by MerkelMain, you might have noticed the option 8, whereby the program will draw the Candlestick Plot. This is not as simple as it sounds. Instead of directly drawing the plot, the program instead calls a new function under MerkelMain, requestCandlesticks().

requestCandlestick()'s purpose is to ensure that the data is valid, and that we are able to properly draw a candlestick plot, before proceeding to do so. It will check whether there are valid candlesticks to be drawn. If there are none, it will prompt the user to compute the candlesticks, and redirect the user to it if the user agrees.

```
Choose an option: 8

No Candlesticks to display!
Compute Candlesticks first? (Y/N): Y

Please pick the number of Candlesticks to be generated. We recommend a maximum of 15.
Enter the number of Candlesticks to be generated per Product/ProductType:
```

requestCandlesticks() will also prompt the user to select a product to draw candlesticks for. Upon doing so, it will then call drawCandlesticks() to draw the text-based plot. This is explained further below.

```
Choose an option: 8

Enter the product to display Candlesticks:
1: BTC/USDT
2: DOGE/BTC
3: DOGE/USDT
4: ETH/BTC
5: ETH/USDT
```

Drawing the Plot

Drawing the plot is done with the new function drawCandlesticks() under the MerkelMain class. It is called by requestCandlesticks() as explained above. I will be explaining more about drawCandlesticks() in this portion.

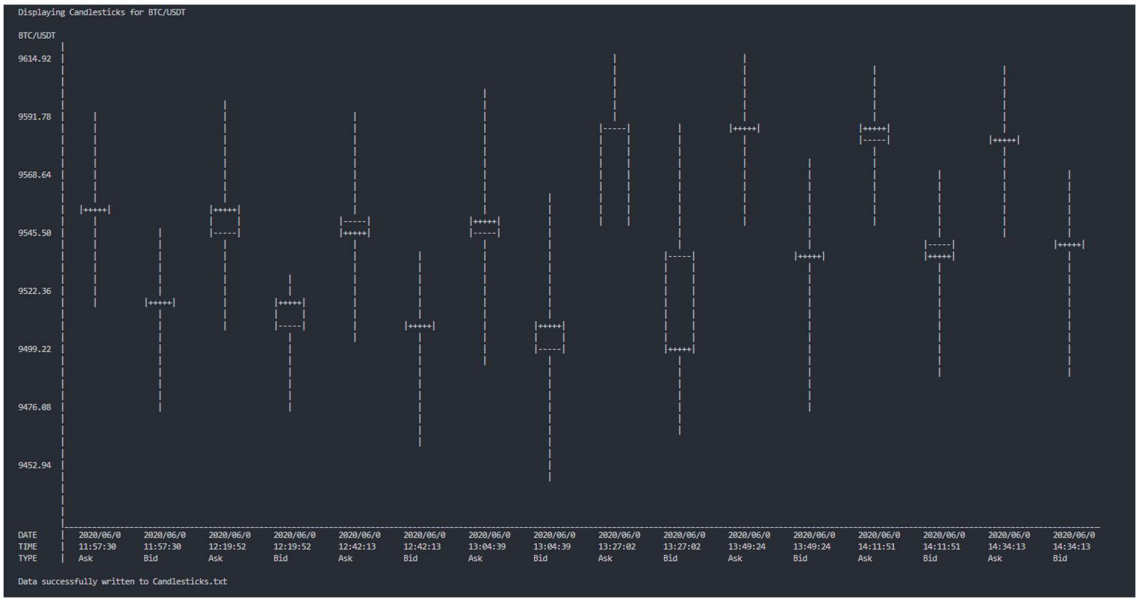
I will try to simply explain the way the function works. Note that all the plotting done in the following is stored into a vector of strings, to be printed in one shot at the end of the function.

1. When the function starts, given a vector of candlesticks, it will identify the highest and lowest value of all the candlesticks, and draw the Y axis accordingly. It will also plot the y axis according to the highest and lowest values, such that all values within the candlesticks provided can be plotted.
2. With the Y axis drawn, it will then draw the candlesticks across the X axis. For each candlestick in the vector, it will draw a candlestick using symbols like `| + -`, such that it

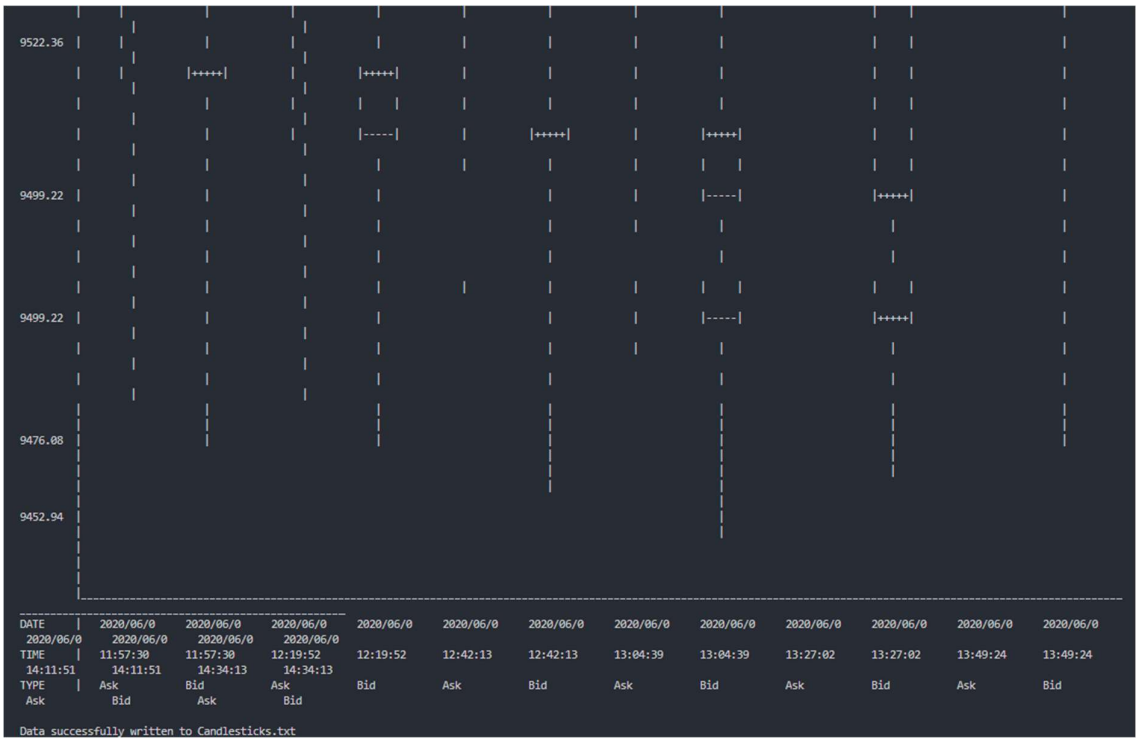
would look like a candlestick plot. The Open and Close are drawn using + and – on the plot respectively.

3. The function will then append the date, time and type at the bottom of the plot.

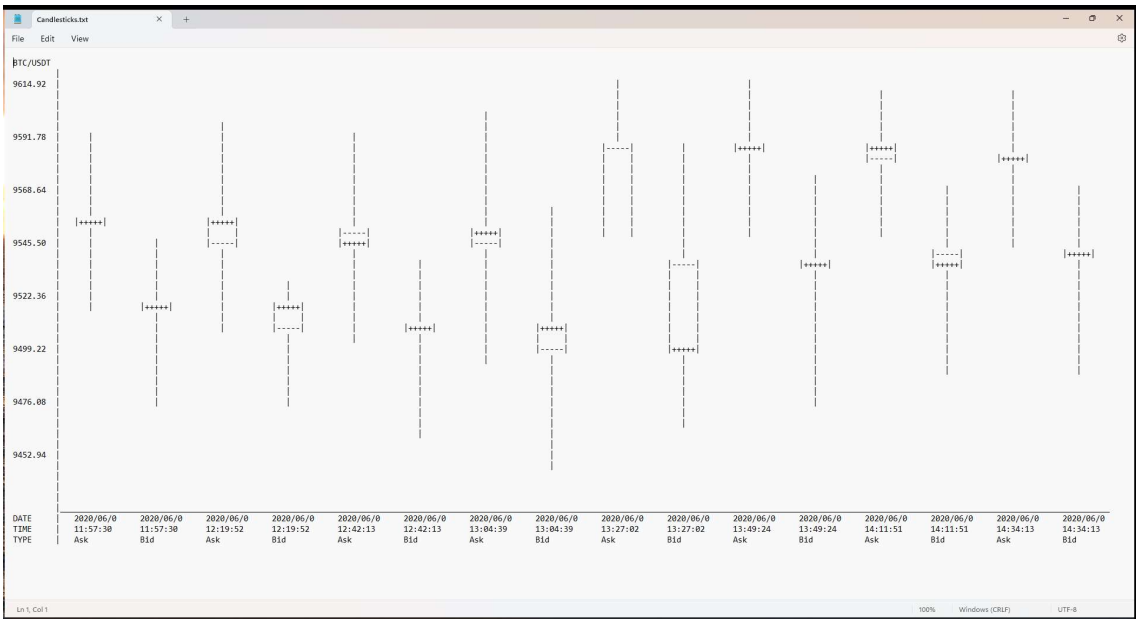
With that done, the function will print out the entire vector of strings into the console, such that it can be viewed by the user.



Understandably, the user might have a short or a thin terminal and might not be able to view everything displayed. This would result in the terminal displayed like the following:



To help with this, I have saved the output into a text file, namely `Candlestick.txt`, such that a user can view it separately in a notepad instead, without resizing their terminal.



Task 3: Plot a Text Graph of some other trading data

Dataset from Yahoo Finance

As a requirement for this task, I was to get another set of trading data, and to plot a text-based graph.

To do so, I have obtained the trading data of Wrapped EOS USD (WEOS-USD) from Yahoo Finance. Yahoo Finance has provided me with 6 months of data, starting from 11th May 2023 to 28th December 2023, containing a total of 229 entries. This data was also obtained on 29th December 2023.

CandlestickBook Class

To assist my program with completing part 3 of the project, I have created a CandlestickBook class. It was created to mimic the function of the OrderBook class, whereby I would be able to store the entries into the CandlestickBook.

```
3   #pragma once
4
5   #include <string>
6   #include <vector>
7   #include <set>
8
9   #include "Candlestick.h"
10
11  class CandlestickBook
12  {
13  public:
14      CandlestickBook(std::string filename);
15
16      /** return vector of all know products in the dataset*/
17      std::vector<std::string> getKnownProducts();
18
19      /** returns the earliest time in the orderbook*/
20      std::string getEarliestTime();
21
22      int getTotalTimestampCount();
23
24      void ReduceCandlesticks(int count);
25
26      std::vector<Candlestick> candlesticks;
27      int totalTimestampCount;
28      std::set<std::string> allTimestamps;
29
30      private:
31  };
```

As the dataset I have acquired from Yahoo Finance directly provided a large number of Candlestick data, it would be incompatible with the OrderBookEntry and OrderBook class. As such, I have opted to create a similar class CandlestickBook.

The CandlestickBook class also comes with some extra functions, such as reduceCandlesticks(), whereby the function would merge multiple candlestick data together,

without removing information from the sum total. This is important for a later portion of my project.

MerkelMain Class

Just like in Task 1, I have modified the MerkelMain class to work with my new classes and new dataset. I have also created 2 similar functions to help facilitate the new classes.

Computing the new Candlesticks

I have created a new function computeCustomCandlesticks() to handle the new candlesticks I have provided. Despite the similar names with computeCandlesticks() function, this function is specially catered to the new candlestick data I have provided.

The function will ask the user as to how many candlesticks it should generate from the new dataset. Given an input from the user, the function will call the CandlestickBook's reduceCandlestick() function accordingly.

```
Choose an option: 9

Please pick the number of Candlesticks to be generated. We recommend a maximum of 15.
Enter the number of Candlesticks to be generated: 5
Candlesticks computed successfully! Total of 5 Candlesticks generated.
```

As the user might change his/her mind at any point in time, and might want more/less candlesticks later, the function saves a backup of the original candlesticks vector, and restores it before execution each time it is called.

Drawing the New Candlesticks

Similar to the above implementation, this portion relies on drawCandlestick() to draw the candlestick plot.

However, note that it would take some time for drawCandlestick to run if it were to plot many candlesticks. If it was successful, it would also appear to be too wide and hard to understand. As such, I have added a limit of 30 candlesticks to draw, such that the function would not execute at all if there were more than 30 candlesticks.

```
Choose an option: 10

Too many candlesticks to display!
Re-compute Candlesticks first? (Y/N): Y
```

To help with this, I have also added a cap to computeCustomCandlestick(), such that it will only accept a value between 1 and 30.

Conclusion

This project was quite a challenge to me, as I initially started with lots of concerns and worries about my project. Plotting a text-based candlestick graph by itself sounded challenging, let alone coding it. I managed to complete it step by step, while ensuring