CM2005 Object Oriented Programming mid term assignment

Nikolaos Kalatzoglou

July 2023

1 Introduction

The implementation of my work has the format of the code extension given in the zip file "merklerex_end_topic_5".

2 Task 1

2.1 Approach

In Task 1, I created two new files for the new Candlestick class, Candlestick.h (header file) and Candlestick.cpp (implementation file). My approach around the calculation of candlesticks is synchronous with the timeframe the program is in during its execution, i.e. the current time displayed each time below the function menu. In my opinion, this method would help to count a few candlesticks per time. Later, this will help in task 2, where the user will see the printed candlesticks from the beginning to the time interval he is in (the program).

So, I created a function that calculates the candlestick of all products and stores them in a vector of candlesticks, which is a member of the MerkelMain class, so that it is always accessible from that class. That's why I added a call of this function to each of the already implemented init() and gotoNextTimeframe() functions of the MerkelMain class. As a result, during initialization and every time the program changes time period the new candlesticks are added to the vector 'candleSticks'.

2.2 Candlestick class description

The Candlestick class consists of two constructors, default and initialization. The default constructor is used to initialize data structures of the class where we do not initially know the objects to be inserted, for example the vector 'candleSticks'. On the other hand, the initialization constructor is called each time to calculate a candlestick that will then be inserted into 'candleSticks'. Also, in addition to assigning the fields of the object, it calculates the close price of the object. In addition, it has 7 getter functions for its 7 fields.

2.3 calculateCandlesticks() function

The calculateCandlesticks() function takes two definitions, the current timeframe and the exact previous one. First, for each option we collect the bid and ask orders in two vectors of the current timeframe using the implemented function 'getOrders' of the OrderBook class. Then, we check whether we are in the first timeframe of the dataset or in some other timeframe. This affects the price that will be the open price of the object that will be created.

2.3.1 First timeframe

When we are in the first timeframe we call the initialization constructor twice, once for the bid orders and once for the ask orders, with arguments the orders (bid or ask), lowest price, highest price, and the price of the first order with these characteristics as an assumption for the open price. To calculate the highest and lowest value we use the implemented functions of the class OrderBook, getHighPrice() and getLowPrice(), while the orders are used to calculate the close price in the constructor.

2.3.2 Any other timeframe

The procedure for creating the two candlesticks in every other time frame is similar to the above but the only difference is that we need to retrieve the close price of the just previous candlestick with the same filters to assign it as open price to the candlestick we create. For this reason, I created the helper function findCandleStick() that takes as arguments a time interval, order type and product and returns the matching object and thus retrieves the close price of the previous candlestick. In each of the above cases the objects that are created are stored in a temporary vector until the procedure for all products is finished, where the temporary vector is returned.

3 Task 2

3.1 Approach

For task 2, I added one more option to the functions menu that appears to the user so that when the plot creation option is selected the user is given the ability to choose the product and order type they want the plot to contain. Based on the user's choices, a search is performed on the vector 'candleSticks' using the auxiliary function findCandlestick() described in subsection 2.3.2 and all candlesticks matching these choices are returned in a time interval from the first to the interval where the program is currently located. After that, plot_data() is called which will include each of the candlesticks found in the final plot and print it to the user.

3.2 plot_data() function

The plot_data() is a function that takes as argument a vector of Candlestick class counterparts, places them on the final chart and prints the final results to the user. For this reason, it returns nothing to the MerkelMain class, from which it is called.

First, it seaches for the lowest and highest value of all the candlesticks, so that the y-axis that has the numerical values is correctly formed. Note that we always take a higher value as the upper bound and a lower value as the lower bound respectively with the above values so that all candlesticks are within bounds and not on them. Then divide the difference between the two limits by the lines on the diagram to get the numerical interval of the y-axis and store it in a table which is practically the y-axis.

3.3 toText() method

Before, we move on to placing each of the candlesticks in the final plot, I should describe my reasoning behind the toText() method I added to the Candlestick class. I thought the text-based representation of each candlestick is an iterative process and applies to each object individually. Therefore I decided to create a method of the candlestick class that converts the candlestick data into an ascii representation.

This method takes as argument the y-axis we talked about before and returns a double char pointer with the ascii representation of the candlestick. First, it allocates the subchart that will later contain the candlestick and initializes each of its positions with the empty ’ ’ character. Then, with an iterative process, we find in which interval of the y-axis each of the low, high, open, close values is located. Also, it is checked and at more than two limits of this space it is closer, so that this limit is saved which will later help in the representation.

As a next step, it is checked whether the open price is greater than the close price or not. In the first case, the inside of the candlestick is filled with the character ’v’, indicating that on average the price of the product fell. In the other case, the inside is filled with the ’̂’ character, indicating that on average the price of the product increased. In each case, however, the walls of the candle are represented by the character ’|’. At the end, the time period the candlestick belongs to is placed below and centered about it and the total result is returned in the form of a double char pointer.

3.4 Approach: Form final plot

Continuing the explanation of the plot_data(), we are led to an iterative process in which 5 candles are placed on each chart until the vector of candles to be printed is empty. By printing only 5 candles on each chart, we keep it sparse and informative. If the candlesticks to be printed are more than 5, correspondingly more charts are printed, each of which is a continuation of the previous one. In short, each iteration creates a plot of five or fewer candlesticks.

During the iteration, it is checked if the remaining candlesticks are less than 5, so we know how many candlesticks will be printed on this plot. So, with a nested iteration, the toText() method I mentioned above is called and the ascii representation of each candlestick is placed in the final plot in each of the 5 positions while still freeing the memory that the method allocated. Also, some of the y-axis values are added to the left of the chart and the title centered and at the top to keep the chart informative. Finally, the overall plot is printed and the table is re-initialized to be used in the next plot if there is one.

4 Task 3

In task 3, I chose to display the trading volume of a product, according to the user's choice, every time interval in a bar plot and I followed the formula I used in task 2.

4.1 Approach

As trading volume I considered the total amount of all orders, either bid or ask, of a product in a timeframe. Similar to task 2, the user selects the name of the product he wants to be informed about. Next, the plot_volume() function, which I created is called with the product selected by the user as an argument. It is worth mentioning that its functionality is very similar to that of its plot_data().

4.2 plot_volume() function

plot_volume() accepts as argument a string containing the name of the product and returns nothing. Correspondingly as in task 2, an array of characters is used which hosts the final diagram and is printed in the of end of the function. First, in an iterative process and using the getOrders() function of the OrderBook class, we calculate the trading volume of the product in each time frame from the beginning to the present. Each value of the trading volume and timeframe is stored in two vectors, to be used in the ascii representation later. Next, we follow the same procedure explained in section 3.2 to calculate the values of the y-axis. Next, we initialize the character array with the character ' '.

As a next step, we are led to a similar iterative process to that of the plot_data(), however the plot size differs by 15 positions instead of 5. Here, up to 15 bars of trading volume can be represented. If the bars to be printed are more, more diagrams are printed. At the beginning of the iteration, the y-axis interval where each height of the trading volumes bar is calculated and each bar is placed in the character array as a column of characters '|' , while the time interval to which it belongs is also placed at its base. Finally, before the final chart is printed, some values of the y axis are added to the left of the chart, while at the top are added the name of the product and the total time interval where the bars are.

4.3 Disclaimer

In summary, I would like to admit that although my work took quite some time, the functionality and the code are my own and I was not helped by third parties.