Object-oriented programming report

Introduction

The advisorbot is a program made for investors to analyse various cryptocurrency products on the market. With the commands implemented, users can get various information on a specific product of their choice. This includes finding out the minimum, maximum and average price for each time frame and even a prediction function to forecast whether the price is likely to increase or decrease.

Table of commands

Commands	**Implementation**
help	Done
help cmd	Done
prod	Done
min	Done
max	Done
avg	Done
predict	Done
time	Done
step	Done
custom command	Done

Parsing code

In my procecessUserInput() function, the user's input is taken in using the getUserInput() function and then tokenised using the CSVReader::tokenise() function. The tokenise function remains largely the same as the one done in the merklerex simulation. Based on the first word, which would be the first token, token[0], it then passes through various conditionals to match it to the correct function. The length of the user input is checked as well to determine the number of tokens and the order of the tokens is checked as well. This process is done in the processUserInput() function. For instance, the prod command would only have one token while others like the help cmd would have two. Should the token not match with any functions, an error message is displayed.

Custom command

For my custom commands I have added a getPriceDiff() function. The function returns the difference between the highest and lowest price product for either a bid or ask. This lets users see the range difference of bid and ask prices. It makes use of the getHighPrice() and getLowPrice() functions defined in orderBook.cpp and uses the values from those to return the price difference. I have also implemented a skip command which works the same as the step command but it lets users specify how far they want to skip. Since my predict command makes use of a moving average, it requires a minimum amount of time steps to pass before being able to give a prediction. This skip command would help users pass through time frames quicker without having to use the step command multiple times. The skip command is implemented the same way as the step command except it now takes in a number. The logic for step is then looped up to the same number of times as the number inputted. I have also implemented an exit command that lets users safely exit the program. This is done by returning a 0 value when the first token is equal to ‘exit’ and is then used to break out of the while loop that shows the main menu.

Optimisations

When I took a glance at the CSV file provided, I noticed that there were more than a million entries in the file. Tokenising it with the usual method used in the merklerex simulation might not be the most efficient as the data we are dealing with now is multiple times the size of that. To improve processing time, I limited the lines tokenised to 1000. When the next time frame is stepped into, another 1000 lines are tokenised and so on. This is done using the readNewEntry function which compares read and unread CSV entries before tokenising the entries and inserting them into the vector.