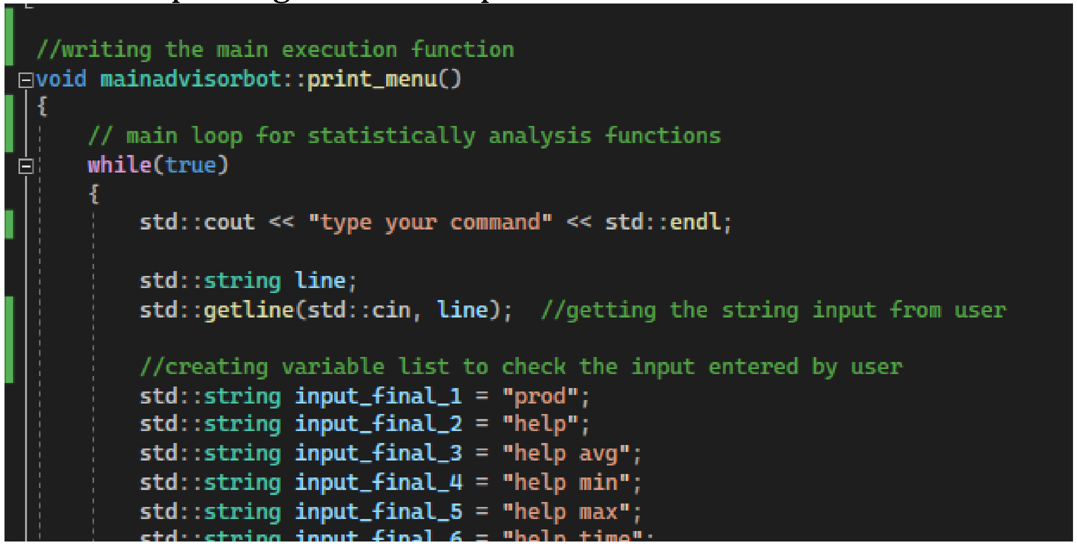
Report of my AdvisorBot

Introduction:
This AdvisorBot is programmed in C++ and it is build on top of the merklerex platform, I have added two more files in it (mainadvisorbot.cpp & mainadvisorbot.h) and integrated it in the MerkelMain.cpp file. The main purpose of this AdvisorBot is to perform statistically analysis for the user. Initially we can only trade on this platform but now a user can also calculate statistical function like average, variance, minimum value or maximum value at a particular time step. And apart from that AdvisorBot can also perform predictions for the next time step (EX: predict max ETH/BTC ask). Through our platform user can not only buy/sell products but also perform some good analysis in between.

Table reporting how many commands I was successfully able to execute:

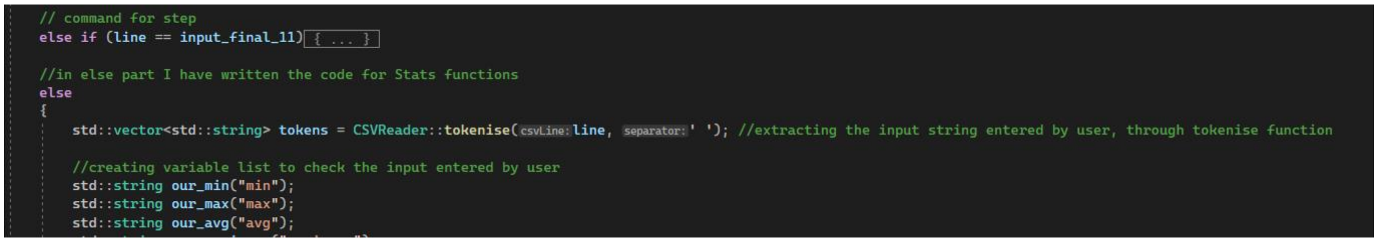
Available Commands	Done	Not Done
help	✓	
help <cmd>	✓	
prod	✓	
time	✓	
step	✓	
min	✓	
max	✓	
avg	✓	
predict	✓	
Task 2 (implement your own code): variance	✓	

Command parsing code description:



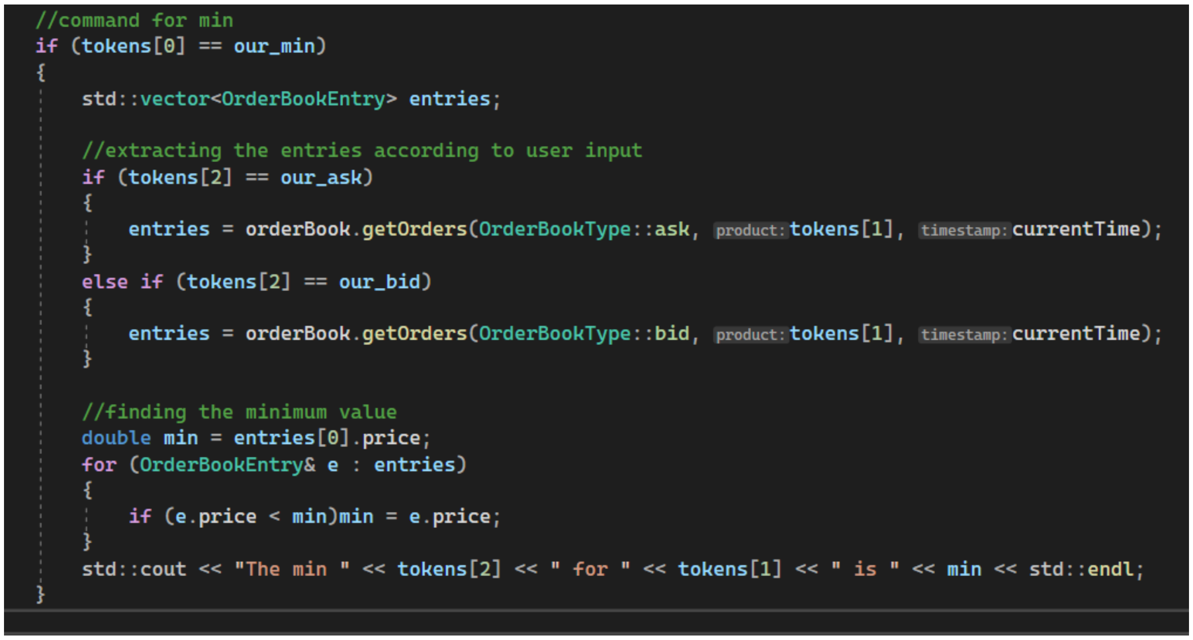
```
//writing the main execution function
void mainadvisorbot::print_menu()
{
    // main loop for statistically analysis functions
    while(true)
    {
        std::cout << "type your command" << std::endl;

        std::string line;
        std::getline(std::cin, line);  //getting the string input from user

        //creating variable list to check the input entered by user
        std::string input_final_1 = "prod";
        std::string input_final_2 = "help";
        std::string input_final_3 = "help avg";
        std::string input_final_4 = "help min";
        std::string input_final_5 = "help max";
```

As we can see in the above image the very first thing is, I'm requesting a string of words from user through getline function in print_menu function.



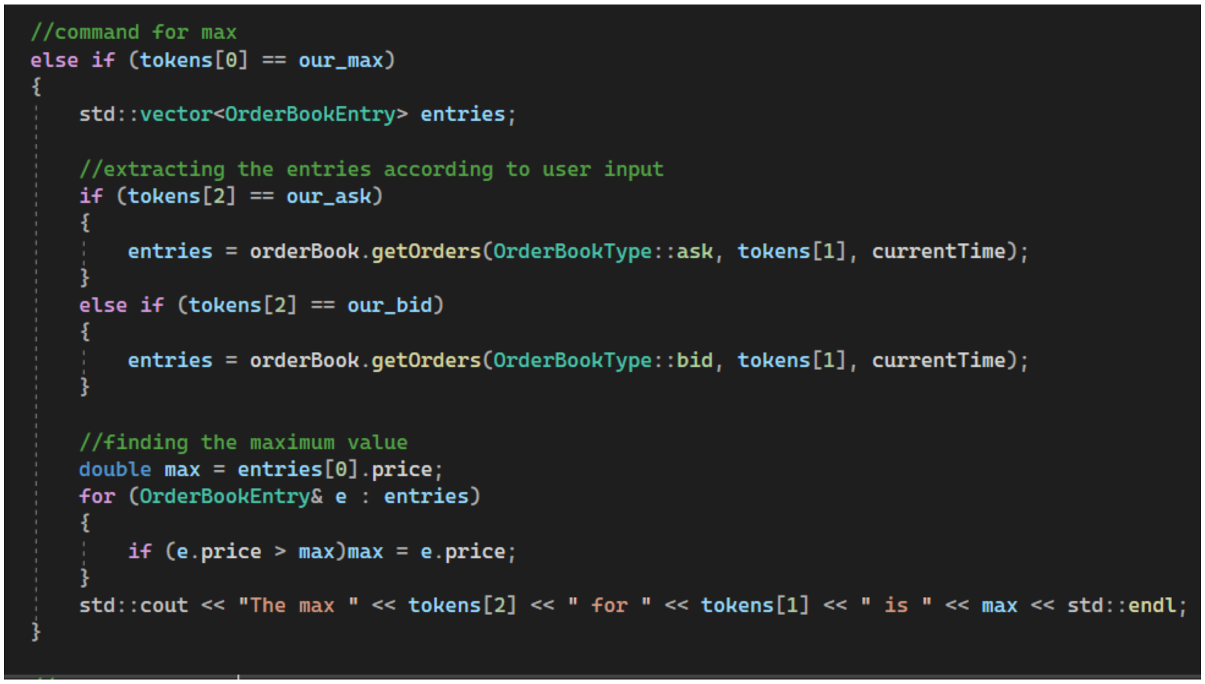
```
// command for step
else if (line == input_final_11){ ... }

//in else part I have written the code for Stats functions
else
{
    std::vector<std::string> tokens = CSVReader::tokenise(csvLine: line, separator: ' '); //extracting the input string entered by user, through tokenise function

    //creating variable list to check the input entered by user
    std::string our_min("min");
    std::string our_max("max");
    std::string our_avg("avg");
```

After that I'm using tokenise function from CSV reader file to add the individual word into our vector. Then we are comparing our vector elements to the appropriate keyword to execute a particular statistical function.



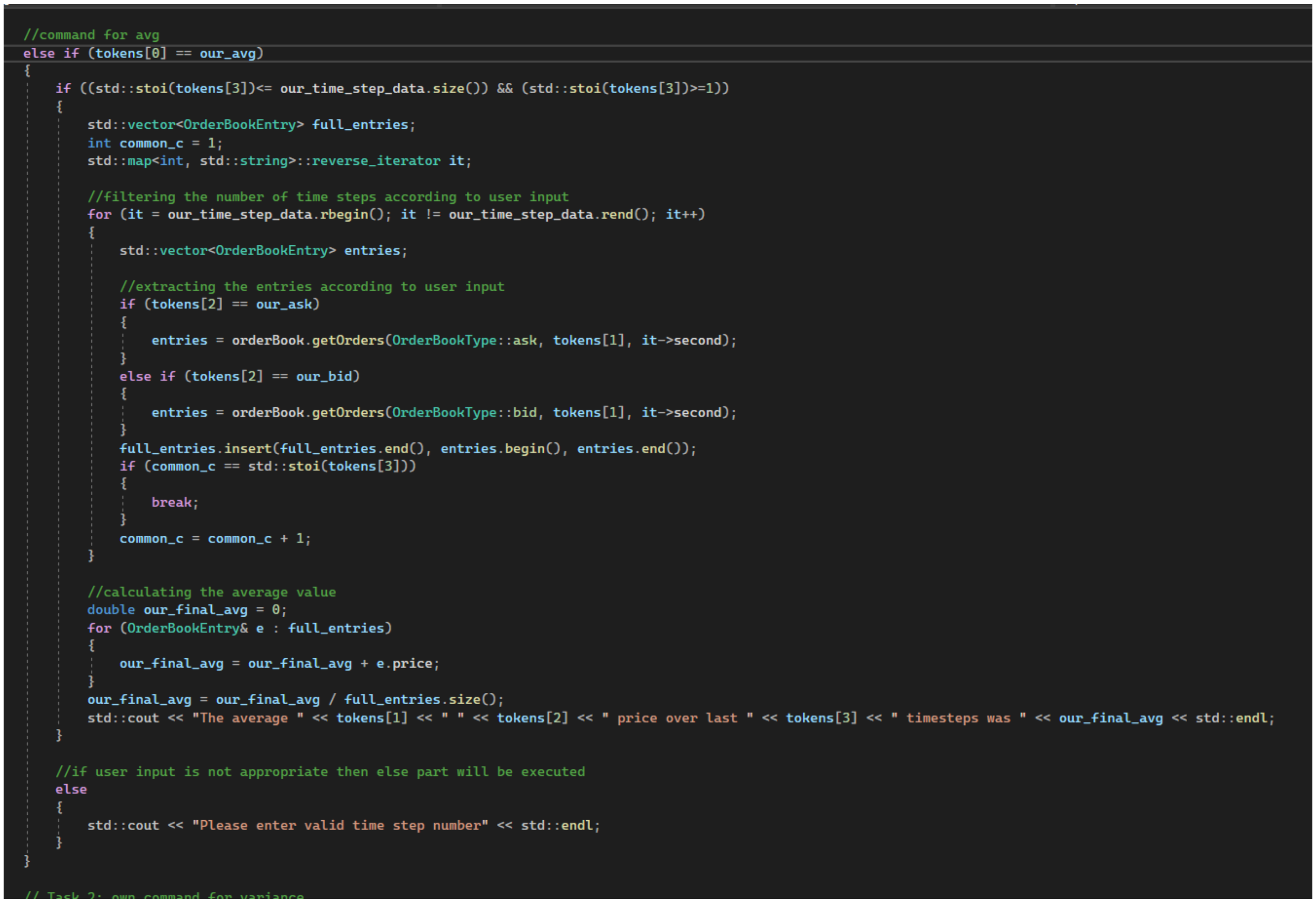
```
//command for min
if (tokens[0] == our_min)
{
    std::vector<OrderBookEntry> entries;

    //extracting the entries according to user input
    if (tokens[2] == our_ask)
    {
        entries = orderBook.getOrders(OrderBookType::ask, product: tokens[1], timestamp: currentTime);
    }
    else if (tokens[2] == our_bid)
    {
        entries = orderBook.getOrders(OrderBookType::bid, product: tokens[1], timestamp: currentTime);
    }

    //finding the minimum value
    double min = entries[0].price;
    for (OrderBookEntry& e : entries)
    {
        if (e.price < min)min = e.price;
    }
    std::cout << "The min " << tokens[2] << " for " << tokens[1] << " is " << min << std::endl;
}
```

For example if the first element of our vector is min, then I have written the program to calculate the min, which starts by extracting entries through get order function from order book file (as we can notice in the above image I have filtered our get order query through vector elements). Once we have extracted the entries, we can iterate over it to get the minimum price, and finally writing the console output statement.



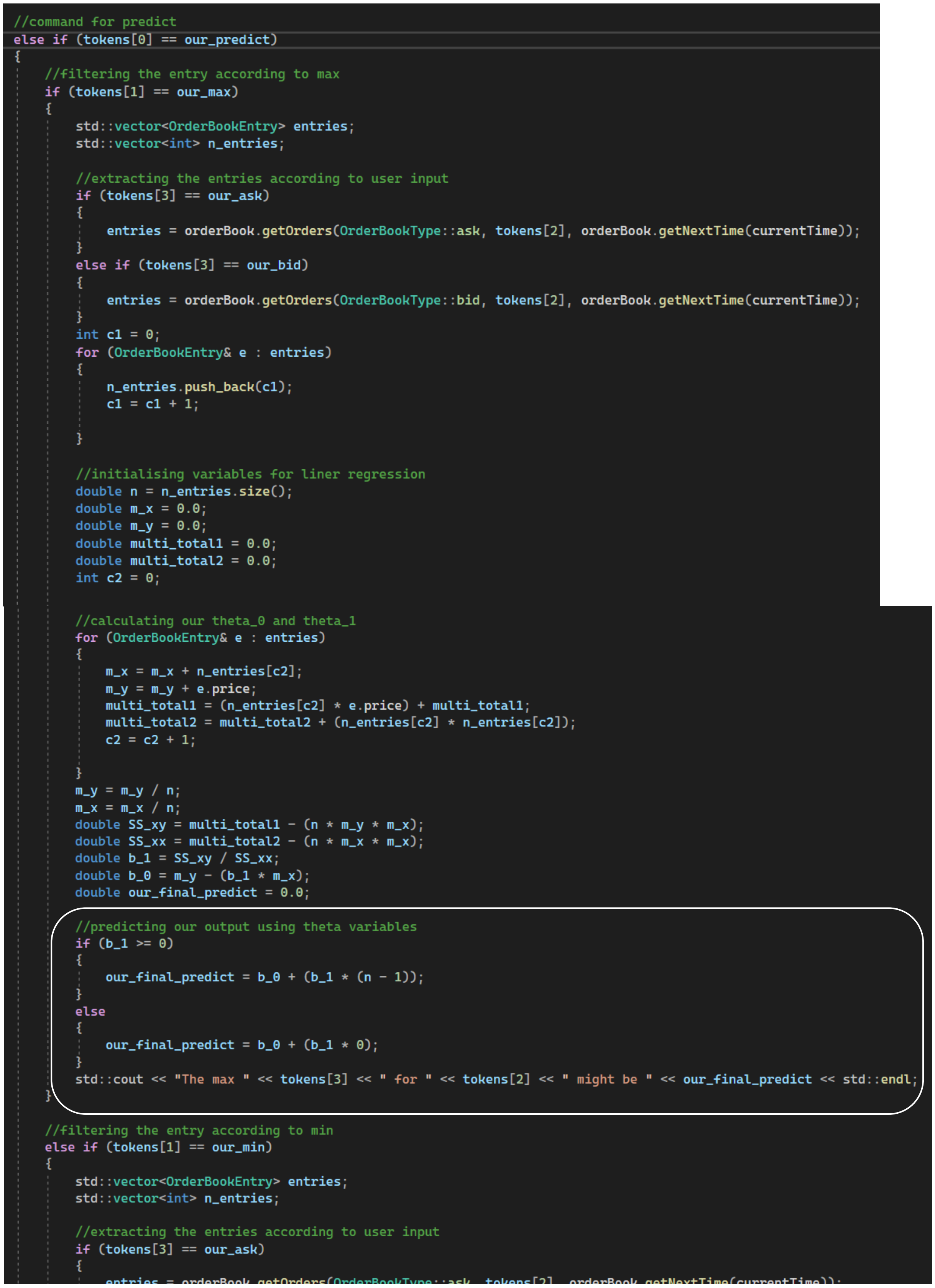
```cpp
//command for max
else if (tokens[0] == our_max)
{
    std::vector<OrderBookEntry> entries;

    //extracting the entries according to user input
    if (tokens[2] == our_ask)
    {
        entries = orderBook.getOrders(OrderBookType::ask, tokens[1], currentTime);
    }
    else if (tokens[2] == our_bid)
    {
        entries = orderBook.getOrders(OrderBookType::bid, tokens[1], currentTime);
    }

    //finding the maximum value
    double max = entries[0].price;
    for (OrderBookEntry& e : entries)
    {
        if (e.price > max)max = e.price;
    }
    std::cout << "The max " << tokens[2] << " for " << tokens[1] << " is " << max << std::endl;
}
```

Just below the min function I have written max function, which is more or less similar to min function logic. However, in the end our for loop logic is just inverse, to find the max price.



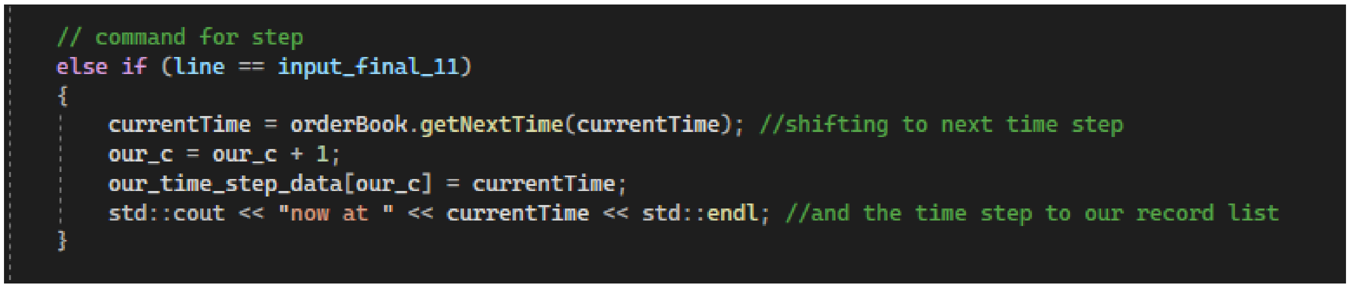
```cpp
//command for avg
else if (tokens[0] == our_avg)
{
    if ((std::stoi(tokens[3])<= our_time_step_data.size()) && (std::stoi(tokens[3])>=1))
    {
        std::vector<OrderBookEntry> full_entries;
        int common_c = 1;
        std::map<int, std::string>::reverse_iterator it;

        //filtering the number of time steps according to user input
        for (it = our_time_step_data.rbegin(); it != our_time_step_data.rend(); it++)
        {
            std::vector<OrderBookEntry> entries;

            //extracting the entries according to user input
            if (tokens[2] == our_ask)
            {
                entries = orderBook.getOrders(OrderBookType::ask, tokens[1], it->second);
            }
            else if (tokens[2] == our_bid)
            {
                entries = orderBook.getOrders(OrderBookType::bid, tokens[1], it->second);
            }
            full_entries.insert(full_entries.end(), entries.begin(), entries.end());
            if (common_c == std::stoi(tokens[3]))
            {
                break;
            }
            common_c = common_c + 1;
        }

        //calculating the average value
        double our_final_avg = 0;
        for (OrderBookEntry& e : full_entries)
        {
            our_final_avg = our_final_avg + e.price;
        }
        our_final_avg = our_final_avg / full_entries.size();
        std::cout << "The average " << tokens[1] << " " << tokens[2] << " price over last " << tokens[3] << " timesteps was " << our_final_avg << std::endl;
    }

    //if user input is not appropriate then else part will be executed
    else
    {
        std::cout << "Please enter valid time step number" << std::endl;
    }
}

// Task 2: own command for variance
```

For the average function I have first converted the third vector element to integer (which is the number of previous time step given by the user) and then extracting the entries the help our time step record list (i.e the private variable of our mainadvisorbot class), and to implement that I have used C++ map functions to iterate for the record. And then finally performing the summation to calculate the average price.



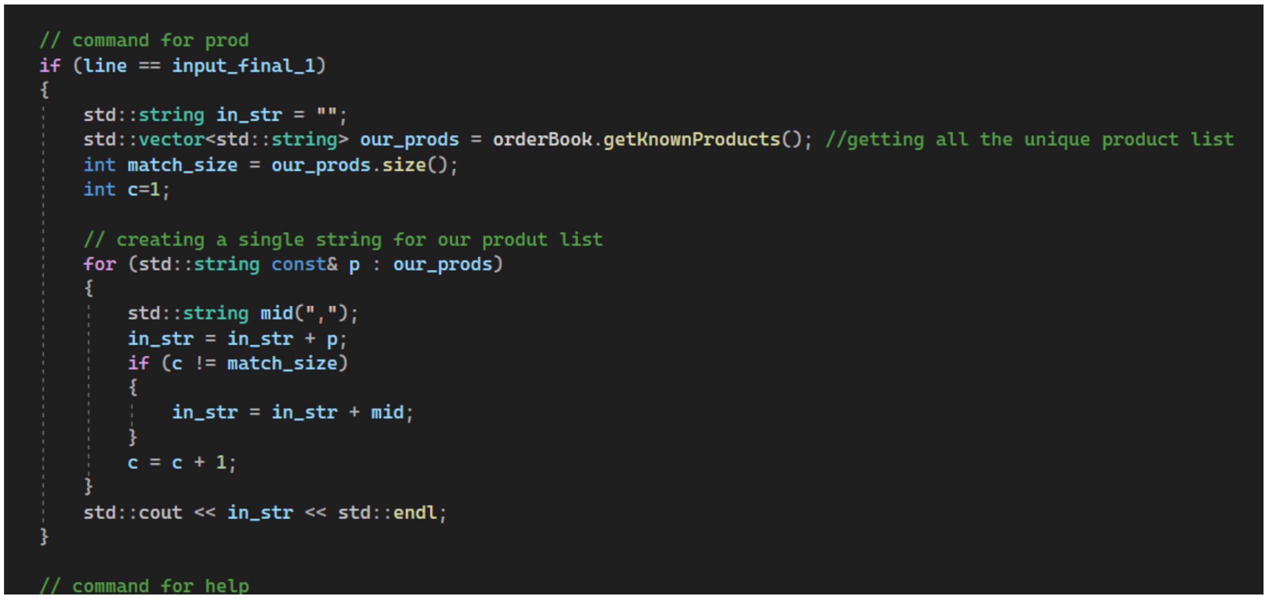
```
//command for predict
else if (tokens[0] == our_predict)
{
    //filtering the entry according to max
    if (tokens[1] == our_max)
    {
        std::vector<OrderBookEntry> entries;
        std::vector<int> n_entries;

        //extracting the entries according to user input
        if (tokens[3] == our_ask)
        {
            entries = orderBook.getOrders(OrderBookType::ask, tokens[2], orderBook.getNextTime(currentTime));
        }
        else if (tokens[3] == our_bid)
        {
            entries = orderBook.getOrders(OrderBookType::bid, tokens[2], orderBook.getNextTime(currentTime));
        }
        int c1 = 0;
        for (OrderBookEntry& e : entries)
        {
            n_entries.push_back(c1);
            c1 = c1 + 1;

        }

        //initialising variables for liner regression
        double n = n_entries.size();
        double m_x = 0.0;
        double m_y = 0.0;
        double multi_total1 = 0.0;
        double multi_total2 = 0.0;
        int c2 = 0;

        //calculating our theta_0 and theta_1
        for (OrderBookEntry& e : entries)
        {
            m_x = m_x + n_entries[c2];
            m_y = m_y + e.price;
            multi_total1 = (n_entries[c2] * e.price) + multi_total1;
            multi_total2 = multi_total2 + (n_entries[c2] * n_entries[c2]);
            c2 = c2 + 1;

        }
        m_y = m_y / n;
        m_x = m_x / n;
        double SS_xy = multi_total1 - (n * m_y * m_x);
        double SS_xx = multi_total2 - (n * m_x * m_x);
        double b_1 = SS_xy / SS_xx;
        double b_0 = m_y - (b_1 * m_x);
        double our_final_predict = 0.0;

        //predicting our output using theta variables
        if (b_1 >= 0)
        {
            our_final_predict = b_0 + (b_1 * (n - 1));
        }
        else
        {
            our_final_predict = b_0 + (b_1 * 0);
        }
        std::cout << "The max " << tokens[3] << " for " << tokens[2] << " might be " << our_final_predict << std::endl;
    }

    //filtering the entry according to min
    else if (tokens[1] == our_min)
    {
        std::vector<OrderBookEntry> entries;
        std::vector<int> n_entries;

        //extracting the entries according to user input
        if (tokens[3] == our_ask)
        {
            entries = orderBook.getOrders(OrderBookType::ask, tokens[2], orderBook.getNextTime(currentTime));
```

To perform prediction functionality I have written code for liner regression algorithm (i.e y=a+bx), where first we divide the code according to min or max price prediction, as per the user input. Once we have fetched the entries of next time step (which will be either ask or bid entries), we can calculate our theta_0 and theta_1 for the prediction-model. And then as we can see in the above circled image I have used theta_0 and theta_1 to predict the maximum price according to graph slope. For predicting minimum price the logic to calculate theta_0 and theta_1 is same, but output is inverse as per the graph slope.



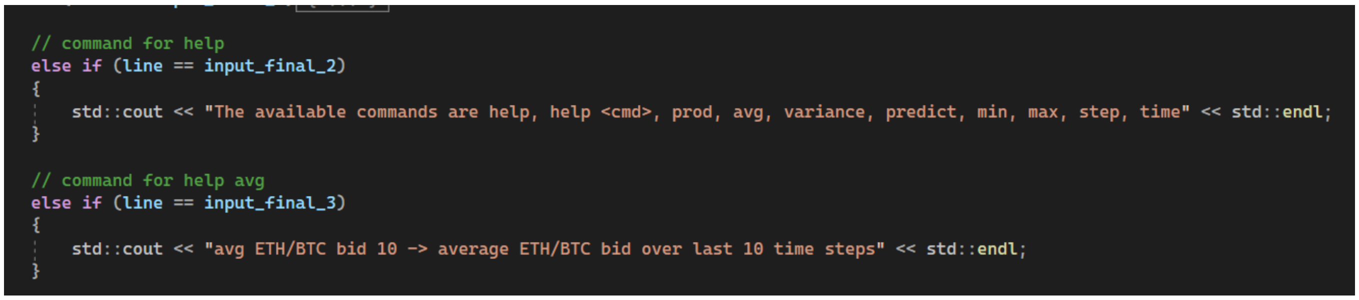
```
// command for step
else if (line == input_final_11)
{
    currentTime = orderBook.getNextTime(currentTime); //shifting to next time step
    our_c = our_c + 1;
    our_time_step_data[our_c] = currentTime;
    std::cout << "now at " << currentTime << std::endl; //and the time step to our record list
}
```

To move to next time step I used the function from orderbook file, and then adding it to our_time_step_data map to maintain the time step record for average and variance function.



```
// command for prod
if (line == input_final_1)
{
    std::string in_str = "";
    std::vector<std::string> our_prods = orderBook.getKnownProducts(); //getting all the unique product list
    int match_size = our_prods.size();
    int c=1;

    // creating a single string for our produt list
    for (std::string const& p : our_prods)
    {
        std::string mid(",");
        in_str = in_str + p;
        if (c != match_size)
        {
            in_str = in_str + mid;
        }
        c = c + 1;
    }
    std::cout << in_str << std::endl;
}

// command for help
```

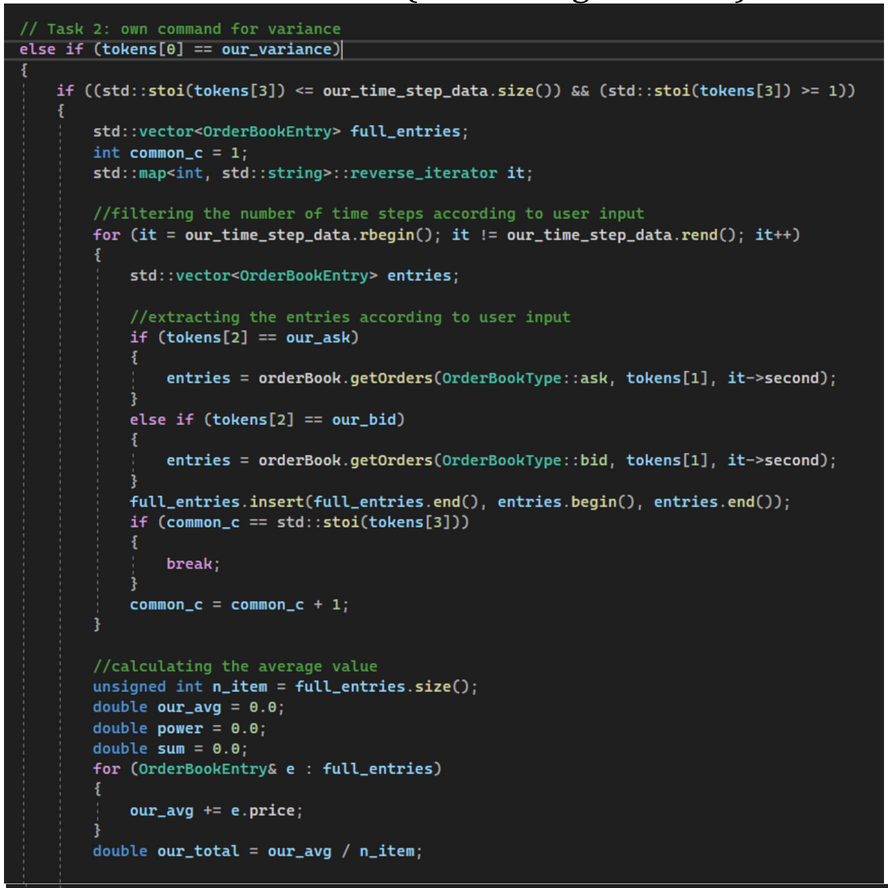
To implement the prod command, I'm first fetching all the unique product values to the our_prods vector, with the help of getnownproduct() function from orderbook file. Then we are iterating over vector elements to make a single string of all the elements.



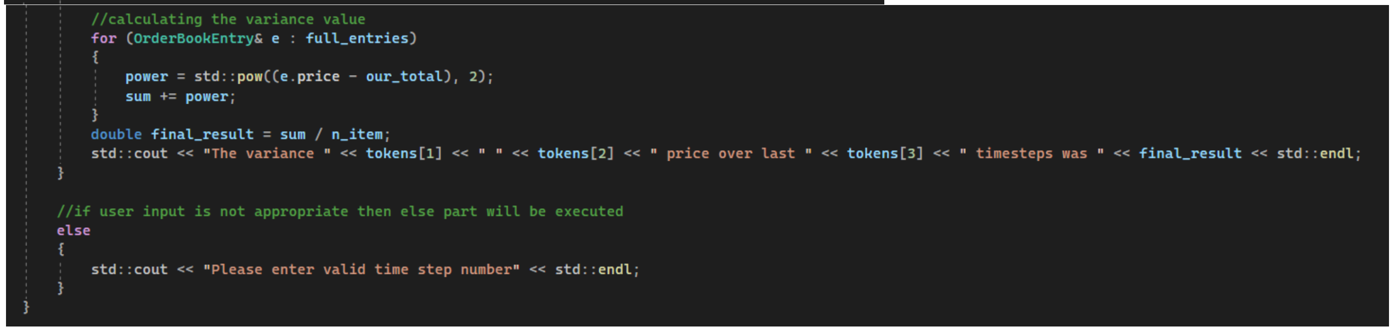
```
// command for help
else if (line == input_final_2)
{
    std::cout << "The available commands are help, help <cmd>, prod, avg, variance, predict, min, max, step, time" << std::endl;
}

// command for help avg
else if (line == input_final_3)
{
    std::cout << "avg ETH/BTC bid 10 -> average ETH/BTC bid over last 10 time steps" << std::endl;
}
```

To implement help or help <cmd> command, we are first checking user input string with the desired value and then printing it accordingly in console.

Task 2 custom command (calculating variance):



```
// Task 2: own command for variance
else if (tokens[0] == our_variance)
{
    if ((std::stoi(tokens[3]) <= our_time_step_data.size()) && (std::stoi(tokens[3]) >= 1))
    {
        std::vector<OrderBookEntry> full_entries;
        int common_c = 1;
        std::map<int, std::string>::reverse_iterator it;

        //filtering the number of time steps according to user input
        for (it = our_time_step_data.rbegin(); it != our_time_step_data.rend(); it++)
        {
            std::vector<OrderBookEntry> entries;

            //extracting the entries according to user input
            if (tokens[2] == our_ask)
            {
                entries = orderBook.getOrders(OrderBookType::ask, tokens[1], it->second);
            }
            else if (tokens[2] == our_bid)
            {
                entries = orderBook.getOrders(OrderBookType::bid, tokens[1], it->second);
            }
            full_entries.insert(full_entries.end(), entries.begin(), entries.end());
            if (common_c == std::stoi(tokens[3]))
            {
                break;
            }
            common_c = common_c + 1;
        }

        //calculating the average value
        unsigned int n_item = full_entries.size();
        double our_avg = 0.0;
        double power = 0.0;
        double sum = 0.0;
        for (OrderBookEntry& e : full_entries)
        {
            our_avg += e.price;
        }
        double our_total = our_avg / n_item;
```



```
        //calculating the variance value
        for (OrderBookEntry& e : full_entries)
        {
            power = std::pow((e.price - our_total), 2);
            sum += power;
        }
        double final_result = sum / n_item;
        std::cout << "The variance " << tokens[1] << " " << tokens[2] << " price over last " << tokens[3] << " timesteps was " << final_result << std::endl;
    }

    //if user input is not appropriate then else part will be executed
    else
    {
        std::cout << "Please enter valid time step number" << std::endl;
    }
}
```

My custom command calculates the variance for a set number of time steps, the way it works is quite similar to average command, example:

```
user> variance ETH/BTC ask 10
advisorbot> The variance ETH/BTC ask price over last 10 timesteps was 1.354
```

In this command we have to specify the product time, ask/bid and number of timesteps. Logic behind the program is quite similar to calculating average till we reach the order-entry extraction stage, which starts with the conversion of third element of our token vector to integer, then iterating over time-step record list (i.e the private variable of our mainadvisorbot class) to add each entry in our full_entries vector, by creating a reverse iterator of map class.

Now once we get all the entries according to user input filter, we can start doing calculation for variance.

Formula for variance $= \frac{\sum_{i=0}^{n}(list[i]-mean)^2}{n}$

Here, I have first calculated the mean after that we iterate over the full_entries vector again to compute $\sum_{\mathrm{i=0}}^{\mathrm{n}}(\mathrm{list[i]} - \mathrm{mean})^2$ with the help of power function of maths library. Then divide the final output with vector size, And in the end we are printing the output through C++ console.