|  |  |
| --- | --- |
| **CM2010 Software design and development** | |
|  | |
| **Student Name:** | Owen Lee Wei Hern |
| **Date Submitted:** | 16th December 2022 |
| **Degree Title:** | Computer Science |
| **Local Institution:** | Singapore Institute of Management |
| **Student ID:** | 220218799 |

**Part 1**

For this task, I will be taking a look at the merkelsim program I developed during my Object-Oriented Programming module.

**Description of the purpose of the program and the technology it uses**

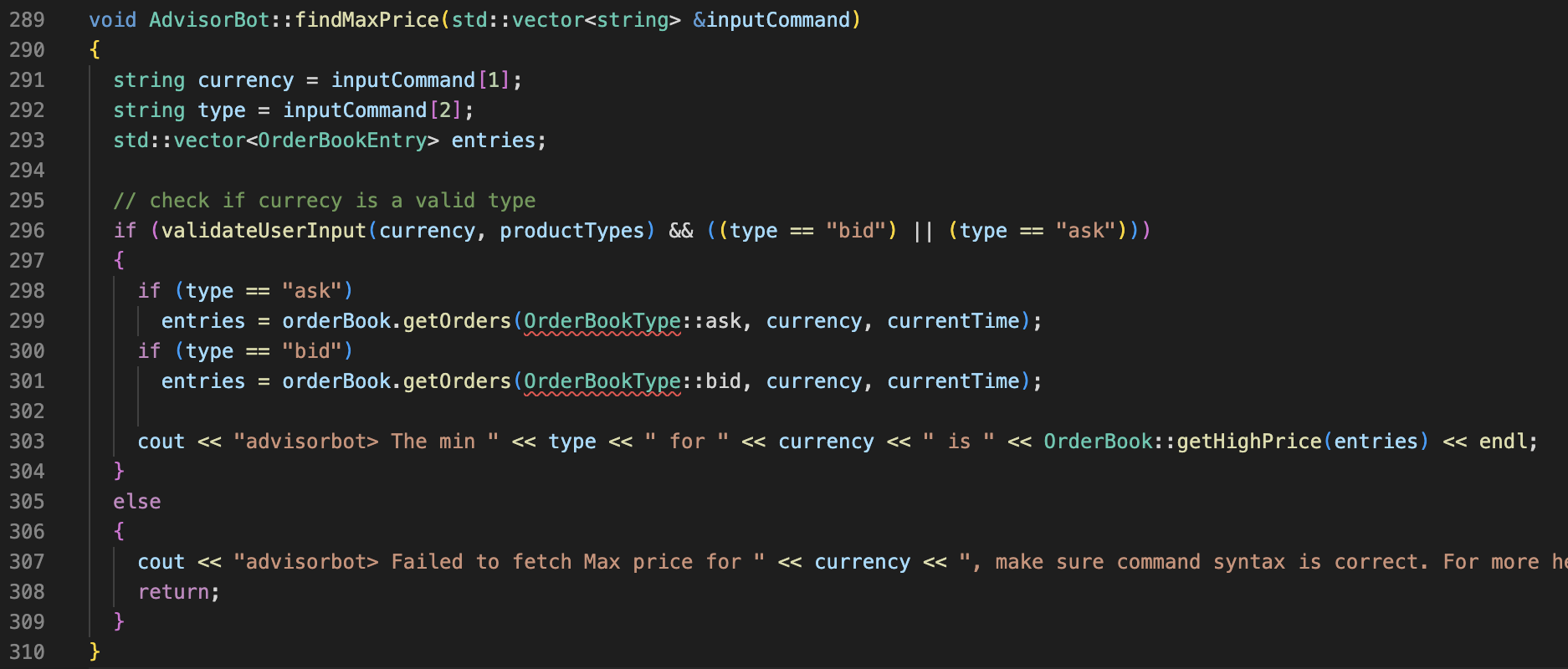
The purpose of this program was to teach us how to work with classes and code using an Object-oriented programming paradigm using a real-world example, in this case, the stock/crypto market. We create a bot that can perform functions on the cryptocurrency market for us, functions like fetching the time, the high and low prices and asking for predictions. This program was coded using C++ as the language and the technologies that helped me achieve this were Visual Studio Code, GitHub and the GNU Compiler.

**Module coupling example 1: Common Environment Coupling**

Common environment coupling can be defined as coupling where different components in a software share and operate on the same global data. Text

Description automatically generated

Text

Description automatically generated****

In the code extracts above, I have several functions within the AdvisorBot class where there is common environment coupling between the OrderBook, OrderBookEntry and AdvisorBot classes. The OrderBookEntry and OrderBook classes are contained within the AdvisorBot class. This makes the AdvisorBot class the global environment where functions such as findMinPrice(), findMaxPrice() and getPopularTrends() operate within making use of the OrderBookEntry type vector saved in AdvisorBot as a global variable.

**Module coupling example 2: Control Coupling**

Control coupling can be defined as the manner of how much a component influences the execution of another component in the software.

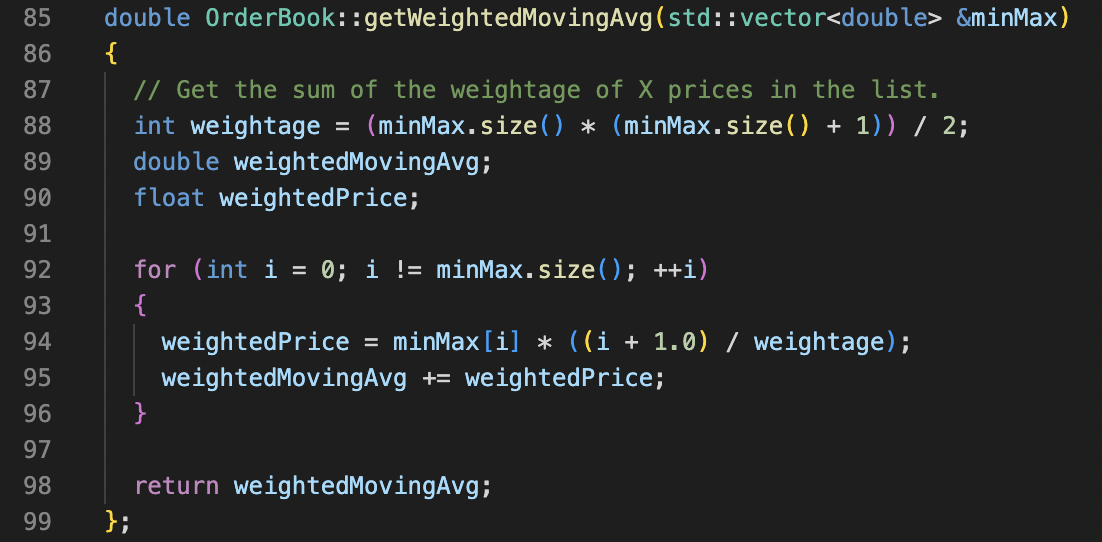




This is an extract of the processUserInput function in AdvisorBot.cpp, what it does is it takes in an inputCommand, which is a string returned from the promptUserInput function that parses the key typed from the command line. Depending on the cmd keyed in, AdvisorBot will perform one of its many functions like listing the commands it knows, listing the available currency on the market, fetching the minimum price of a currency etc. This can be considered control coupling as the user keyed in command is the flag that will dictate which pathway the processUserInput function will take.

**Module cohesion example 1: Functional Cohesion**

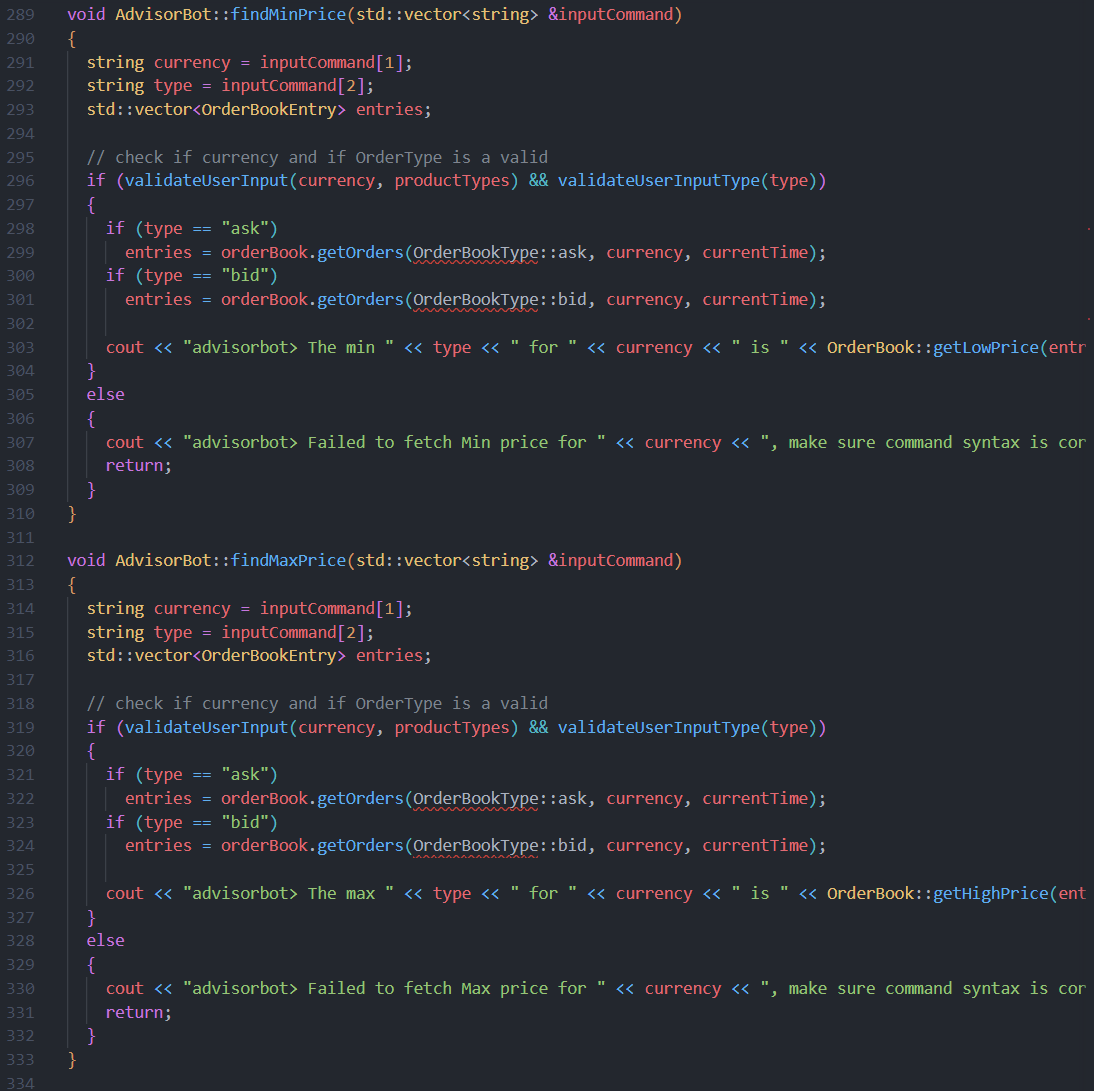
Functional cohesion is defined as when parts of a module are grouped as they all contribute to a single task of the module.



The getWeightedMovingAvg() function in OrderBook class is a good example of functional cohesion. The function finds a weighted moving average of a specified product/stock on the exchange and every line of code in this function works together to achieve that, there is no extra line of code which executes and contribute to a separate result.

**Module cohesion example 2: Communicational Cohesion**

Communicational Cohesion can be defined as different modules that operate on the same input data and/or contribute towards the same output data.



An example of communicational cohesion can be found in the findMinPrice and findMaxPrice functions. Although both functions perform different tasks, min finding the minimum price of a currency and max finding the maximum price of a currency. They are communicationally cohesive as they take in the same parameters, being the user’s input command and output a similar message to the console.