Coding workshop: Modelling an order book entry as a class.

In this worksheet, we will define a class that can store the data for an entry in the order book, then we will create instances of the class (objects) and store them into a std::vector object.

Write out the class definition

Put this at the top of your CPP file (the one with all the menu code in it)

```
class OrderBookEntry // this is the name of the class
{
    public: // parts of the class that can be seen from outside
        double price; // this is a data member (AKA a variable inside a class)

}; // note the semi-colon
```

Now in the main function, try creating some OrderBookEntry objects:

```
OrderBookEntry obe1;
obe1.price = 0.125;
OrderBookEntry obe2;
obe2.price = 0.5;
```

Now edit the class definition, so it has all the data members you need to represent one row in the dataset. For reference, here is a row from the dataset:

```
2020/03/17 17:01:24.884492,ETH/BTC,bid,0.02186299,0.1
```

Use the OrderBookType enum class in your OrderBookEntry class. Update the code where we created an OrderBookEntry object, so it puts example data into the data members.

The textbook discusses the syntax for a class on Chapter 11 p386. It includes information about how to add function members to classes. Do not worry too much about that for now, we will get into that later!

Add a constructor to the OrderBookEntry class

Class constructors contain code that runs when we create instances of the class (objects). Here is an example of a class definition for OrderBookEntry with a constructor:

```
class OrderBookEntry
{
    public:
```

```
        /** Create a new OrderBookEntry with the price
        * set to the sent value
        */
        OrderBookEntry( double _price);

        double price;
};
```

However, we also need to provide the implementation of the constructor. Consider this line:

```
OrderBookEntry( double _price);
```

This line does not contain the implementation of the constructor - it does not say what the constructor should actually do. Here is an example of a constructor implementation:

```
OrderBookEntry::OrderBookEntry( double _price)
{
    price = _price;
}
```

Note that you need to put this outside of the class wrapper:

```
class OrderBook
{

};

// put the implementation here
```

This implementation takes the incoming argument _price and assigns it to the OrderBookEntry data member price. Notice that this constructor is namespaced into OrderBookEntry. It is OrderBookEntry::OrderBookEntry not just OrderBookEntry. Since it is namespaced inside the OrderBookEntry class, it can directly access the price data member of OrderBookEntry. It could access price even if price were placed in a private section of the class.

The assignment style syntax is not the preferred one in modern C++. Member initialiser lists are the preferred syntax:

```
OrderBookEntry::OrderBookEntry( double _price)
: price(_price)
{

}
```

the ': price(_price)' assigns the incoming _price parameter to price.

See Chapter 11 page 388 in the textbook for more information about constructors, including an explanation of why we were able to use the class even before we had

written a constructor (i.e. because the compiler provides a default constructor). Member initialiser lists are covered in Chapter 11 page 394.

Implement a constructor for your OrderBookEntry class

Now go ahead and implement a constructor for your OrderBookEntry class which takes all five columns of the dataset as parameters, and assigns them appropriately to the class data members.

Create a vector of objects

Now we can create a vector of objects:

```
std::vector<OrderBookEntry> entries;
entries.push_back(obe1);
entries.push_back(obe2);
```

Then we can acess the data for printing:

```
std::cout << entries[0].price << std::endl;
```

We can iterate over the vector using a range-based loop, like this:

```
for (OrderBookEntry& e : entries)
{
  std::cout << e.price << std::endl;
}
```

Remind yourself why we put the ampersand after the type in this loop. Check out Chapter 6, p 216 in the textbook for examples and further explanation about using references and range loops. Do you think we should be using a const reference if we just want to print out data from the objects in the vector? Why?

Challenge

Write some test code that creates multiple OrderBookEntry objects with different values and stores them into a vector. Now write some useful functions. The names should tell you what you need to calculate:

```
double computeAveragePrice(std::vector<OrderBookEntry>& entries)

double computeLowPrice(std::vector<OrderBookEntry>& entries)

double computeHighPrice(std::vector<OrderBookEntry>& entries)

double computePriceSpread(std::vector<OrderBookEntry>& entries)
```

Conclusion

In this worksheet, we have written a simple class that can represent a row in our dataset. We have then created instances of that class (objects) and stored them into a std::vector.