Coding workshop: using header files and making a class with functions

In this worksheet, we will organise our program into header files and CPP files, and we will create a class (with functions) that represents the main application.

Re-organise the OrderBookEntry class into a header and CPP file

You are going to add some new files to your project. The way you do this depends on the IDE you are using.

Browser-based Visual Studio Code

It is best to create a folder for your app at this point.

- Create a folder by right clicking on the file list area and choosing the appropriate option to make a folder or directory.
- Call the folder src.

- Right-click on src and select new file. Do this twice - once for OrderBookEntry.h, once for OrderBookEntry.cpp.

Visual Studio on Windows

Since Visual Studio is managing the build for your project, you need to add files through the Visual Studio user interface. Right-click on the Source Files item in your file browser inside Visual Studio. Select new file then class. You will be asked to name the class. Call it OrderBookEntry. Click the button to confirm the action. Now it should create a new OrderBookEntry.cpp file in your Source Files section in the VS file browser and an OrderBookEntry.h file in your Header Files section.

XCode on macOS

Similarly to Visual Studio on Windows, XCode will want to manage the build process of your app, so you need to add your new class via XCode. Right-click on your Source folder in the C++ command-line project you created earlier and work through the dialogues to create a new class with separate header and CPP file. You will find that XCode creates a .hpp file for the header instead of a .h file.

What to put in the OrderBookEntry.h file

You should put the class definition into the OrderBookEntry.h file, for example:

```
#pragma once
```

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

        double price;
};
```

Notice my use of the pragma once pre-processor directive. This means that the file will only be included once to prevent duplications of its content.

Your class should have more data members than the one above, enough to cover the five fields in the CSV data file.

What to put in the OrderBookEntry.cpp file

The CPP file should contain the implementation of the class. In my case, this is just the constructor:

```
#include "OrderBookEntry.h"

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

}
```

Note the use of the include pre-processor directive. This means the OrderBookEntry.cpp will have that class definition tacked on the top, which allows you to then enter the namespace and define the constructor.

Compile and run

Now you can compile your multi-file program with the following command in the Visual Studio Code terminal:

```
g++ OrderBook.cpp main.cpp -o merklerex
```

In the other IDEs, they should automatically include the new CPP files in the build.

Now for the MerkelMain class

Now we are going to define the MerkelMain class. I would first like to note that the 'Merkle' in the Merkle tree data structures that underly cryptocurrencies is

spelt differently from the Merkel in my app. Sorry about the confusion!

Add another class to your project and call it MerkelMain. In the header file, MerkelMain.h, put the following:

```
#pragma once

#include <vector>
#include "OrderBookEntry.h"

class MerkelMain
{
    public:
        MerkelMain() = default;
        /** Call this to start the sim */
        void init();
    private:
        void loadOrderBook();
        void printMenu();
        void printHelp();
        void printMarketStats();
        void enterOffer();
        void enterBid();
        void printWallet();
        void gotoNextTimeframe();
        int getUserOption();
        void processUserOption(int userOption);


        std::vector<OrderBookEntry> orders;
};
```

Things to note:

- #include “OrderBookEntry.h”.

This gives us access to the OrderBookEntry class that we defined in OrderBook.h.

- MerkelMain() = default; This is good standard practice as it explicitly instructs the compiler to create a default constructor. If we later add a non-default constructor (i.e. one with our own code in), the line above will make sure there is also a default constructor. See this link https://www.cplusplus.com/doc/tutorial/classes/ or (for degree students) textbook Chapter 11 p388 for more reading about default constructors.
- private. We have defined a load of functions as being private. The idea is that we only expose a single function to the outside world: init. Everything else is handled internally to the class.

Implement all the functions

Now it is your job to implement all those functions. You should already have the code in the previous version, in the same file as the main function. Here is an example of what you are aiming to create in the merkelmain.cpp file to get you started:

```
#include "MerkelMain.h"
#include <iostream>
#include <vector>

/** your main function should call this */
void MerkelMain::init()
{
    loadOrderBook();
    int input;
    while(true)
    {
        printMenu();
        input = getUserOption();
        processUserOption(input);
    }
}

/** load up some dummy data for now */
void MerkelMain::loadOrderBook()
{

    orders.push_back(  OrderBookEntry{1000,
                        0.02,
                        "2020/03/17 17:01:24.884492",
                        "BTC/USDT",
                        OrderBookType::bid}    );

    orders.push_back(  OrderBookEntry{2000,
                        0.02,
                        "2020/03/17 17:01:24.884492",
                        "BTC/USDT",
                        OrderBookType::bid}    );

}
```

Go ahead and write the rest of those functions!

Once you have implemented all the functions from the merkelmain.h file, you should create a main function in the main file for the project which looks like this:

```
#include "MerkelMain.h"

int main()
{
    MerkelMain app{};
    app.init();

}
```

To compile the program in the web-based Visual Studio Code editor, run the following command:

```
g++ MerkelMain.cpp OrderBookEntry.cpp main.cpp -o merklerex
```

To run it:

```
./merklerex
```

Now try out the menu. It should be working, assuming you started with a working menu before your refactored the code. Remember you can always refer to the weekly code zip file if you get stuck.

Conclusion

In this worksheet, we have refactored the OrderBookEntry class into two separate files, a header and a CPP file. We have written a new class to represent the whole application, which includes a single public entry point, the init function.