Coding workshop: Integrating the file reading code to our main program

In this workshop, we will wrap up the tokenise and file reading code into a CSVReader class, then integrate it into the MerkelMain class.

Completing the code that converts from tokens to OrderBookEntry objects

First, we need to complete the code that converts the data from the file into OrderBookEntry objects.

From the previous worksheet, you should have something like this main function, along with the tokenise function:

```
int main()
{
    std::string csvFilename{"20200317.csv"};
    std::ifstream csvFile{csvFilename};
    std::string line;
    if (csvFile.is_open())
    {
        std::cout << "Opened file " << csvFilename << std::endl;
        std::string line;
        while(getline(csvFile, line))
        {
            std::vector<std::string> tokens = tokenise(line, ',');
            std::cout << "Read " << tokens.size() << " tokens " << std::endl;
            if (tokens.size() != 5) continue;

            try
            {
                std::string timestamp = tokens[0];
                std::string product = tokens[1];
                // what about tokens[2]?
                double price = std::stod(tokens[3]);
                double amount = std::stod(tokens[4]);
            }
            catch (const std::exception& e)
            {
                continue;
            }
        }
    }
    else
    {
```

```
        std::cout << "Problem opening file " << csvFilename << std::endl;
    }

    csvFile.close();
    return 0;
}
```

The missing token processing code is the conversion from "bid" or "ask" to an OrderBookType.

The OrderBookType

We need to convert from a std::string, e.g. "bid" into an OrderBookType, e.g. OrderBookType::bid. To do that, we need a function. We are going to add to the OrderBookEntry class since it knows about OrderBookTypes. In OrderBookEntry.h, in the public section:

```
static OrderBookType stringToOrderBookType(const std::string& s);
```

Note that it has a few features:

- It is declared as static because it does not need to access the internal data members of the OrderBookEntry class to do its work
- The incoming std::string is declared as const as we only need to read it, we do not need to write to it.
- The incoming std::string is a reference as there is no need for the system to make a copy of the string we are going to pass in.

Now the implementation of the function, in OrderBookEntry.cpp:

```
OrderBookType OrderBookEntry::stringToOrderBookType(const std::string& s)
{
    if (s == "ask")
    {
        return OrderBookType::ask;
    }
    if (s == "bid")
    {
        return OrderBookType::bid;
    }
    return OrderBookType::unknown;
}
```

We have added a new category of OrderBookType here - OrderBookType::unknown. Can you work out how to add a new category of OrderBookType?

Add the final token processing line

Now, inside the while loop, add the line that calls our new stringToOrderBookType function:

```
OrderBookType orderType = OrderBookEntry::stringToOrderBookType(tokens[2]);
```

Put it all together and create an OrderBookEntry object

Now you are ready to create OrderBookEntry objects using the two doubles, the two strings and the OrderBookType:

```
OrderBookEntry obe{price,
                   amount,
                   timestamp,
                   product,
                   orderType};
```

Refactoring into a CSVReader class

Now we are going to refactor the code into a CSVReader class. Add two new files to your project: CSVReader.h and CSVReader.cpp.

Put this into the CSVReader.h file:

```
#pragma once

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

class CSVReader
{
    public:

    CSVReader();
    /** parse the sent CSV file into multiple
    * OrderBookEntry objects which should contain
    * lines like:
    * 2020/03/17 17:01:24.884492,ETH/BTC,bid,0.02187305,6.85567013
    */
    static std::vector<OrderBookEntry> readCSV(std::string csvFile);

    private:

    static std::vector<std::string> tokenise(std::string csvLine, char separator);
    static OrderBookEntry stringsToOBE(std::vector<std::string> strings);

};
```

Note there is only one public function and that all the functions are static since they do not need to access any data member variables on the CSVReader class. We are really using this class as a wrapper or namespace for these functions.

Now put a load of empty functions into CSVReader.cpp so that it compiles:

```
#include "CSVReader.h"
#include <iostream>
#include <fstream>

CSVReader::CSVReader()
{

}

std::vector<OrderBookEntry> CSVReader::readCSV(std::string csvFilename)
{
    std::vector<OrderBookEntry> entries;
    return entries;
}

std::vector<std::string> CSVReader::tokenise(std::string csvLine, char separator
)
{
std::vector<std::string> tokens;
return tokens;
}

OrderBookEntry CSVReader::stringsToOBE(std::vector<std::string> tokens)
{
    OrderBookEntry obe{0, 0, "", "", OrderBookType::bid};
    return obe;
}
```

Note that this is really the minimal code to make this compile. The functions do not do anything yet.

Pull it into the main file.

In main.cpp (or whatever your main program file with the main function is called), put the following code, commenting out whatever was there before if necessary:

```
#include "CSVReader.h"
#include "OrderBookEntry.h"

int main()
{
```

```
    std::vector<OrderBookEntry> orderBook = CSVReader::readCSV("20200317.csv");
}
```

You will need the OrderBookEntry.h and CPP files in your project.

Check that you can compile and run this program.

implement stringsToOBE

Now we have the CSVReader class compiling, we can proceed with implementing the functions. The general idea is that CSVReader::readCSV will iterate over the lines in the file, call CSVReader::tokenise on each line, and then pass the resulting tokens to stringsToOBE. stringsToOBE will convert the tokens into an OrderBookEntry object, using appropriate exception handling and checking code.

Let's start with the stringsToOBE function.

Open up CSVReader.cpp and put in the code that you wrote before:

```
OrderBookEntry CSVReader::stringsToOBE(std::vector<std::string> tokens)
{
    double price, amount;

    if (tokens.size() != 5) // bad
    {
        std::cout << "Bad line " << std::endl;
        throw std::exception{};
    }
    // we have 5 tokens
    try {
        price = std::stod(tokens[3]);
        amount = std::stod(tokens[4]);
    }catch(const std::exception& e){
        std::cout << "Bad float! " << tokens[3]<< std::endl;
        std::cout << "Bad float! " << tokens[4]<< std::endl;
        throw;
    }

    OrderBookEntry obe{price,
                       amount,
                       tokens[0],
                       tokens[1],
                       OrderBookEntry::stringToOrderBookType(tokens[2])};

    return obe;
}
```

Note that this code throws an exception if it cannot parse the strings you send

it. This means you will need to catch that exception in the code which calls CSVReader::stringsToOBE.

The tokenise function

Now it is your turn - can you get the code from the tokenise function and put it into CSVReader::tokenise?

The readCSV

Now readCSV. Can you take the code we wrote before which opens the file and iterates over the lines and put it into the readCSV function? Things to note:

- The function takes a string argument which you should use for the file name
- When you convert the tokenised strings into an OrderBookEntry object using stringsToOBE, you need to catch the exception and decide what to do (probably just call continue!)

Test it

Go back to your main function and check you are getting data from readCSV:

```
#include "CSVReader.h"
#include "OrderBookEntry.h"
#include <iostream>

int main()
{
    std::vector<OrderBookEntry> orderBook = CSVReader::readCSV("20200317.csv");
    std::cout << "read " << orderBook.size() << " orders" << std::endl;
}
```

You should see something like:

```
read 3540 orders
```

Integrate the new CSVReader class into the MerkMain::init function

The final step is to integrate CSVReader into the MerkelMain class by calling readCSV in the MerkelMain init function and storing the resulting vector of OrderBookEntry objects onto a data member on MerkelMain.

Make sure MerkMain has init and loadOrderBook functions:

In MerkMain.h:

```
class MerkelMain
{
    public:
        ...
        /** call this to fire up the application */
        void init();
        ...
    private:
        void loadOrderBook()
        ...
        std::vector<OrderBookEntry> orders;
```

Make sure you include OrderBookEntry.h. Since MerkelMain.h does not access anything on CSVReader, you don't need to include CSVReader.h.

In MerkelMain.cpp:

```
void MerkelMain::loadOrderBook()
{
    orders = CSVReader::readCSV("20200317.csv");
    std::cout << "MerkelMain::loadOrderBook read " << orders.size() << "orders" << std::endl
}


void MerkelMain::init()
{
    loadOrderBook();
    int input;
    while(true)
    {
        printMenu();
        input = getUserOption();
        processUserOption(input);
    }
}
```

Make sure you include CSVReader.h as the loadOrderBook function needs to access CSVReader.

Update the main function to use MerkelMain

Back to the main function again:

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

int main()
{
```

```
    MerkelMain app{};
    app.init();
}
```

That should be enough to kick off the whole chain. You should see some output like this, as the app reads in the data then displays the menu:

```
MerkelMain::loadOrderBook read 3540orders
1: Print help
2: Print exchange stats
3: Make an ask
4: Make a bid
5: Print wallet
6: Continue
==============
Type in 1-6
```

Conclusion

In this worksheet, we refactored the CSV parsing code into some functions in a CSVReader class. We integrated that class into the MerkelMain class so we can fire up the whole application, including the dataset from a single function call.