Coding workshop: Converting the menu to use functions

In this worksheet, we are going to refactor the menu system so it uses functions.

The printMenu function

We'll start with the printMenu function. Add the following code to the top of your cpp file, below the include directives:

```
void printMenu()
{

}
```

Make some observations - does this function take any parameters? Does it return anything?

Take all the code that prints out the menu options and place it here. Just the print outs, *not* the while loop and the user input processing, something like this:

```
void printMenu()
{
    // 1 print help
    std::cout << "1: Print help " << std::endl;
    // more options here...
}
```

Now call printMenu in your while loop:

```
int main()
{
    while (true)
    {
        printMenu();
        // user input capture and processing code here...
    }
    return 0;
}
```

Build and test!

The getUserOption function

Now we are going to create the getUserOption function. Something like this:

```
int getUserOption()
{

}
```

Does this function return anything?

The purpose of this function is to prompt the user to enter an option, read a number from the user, then return that number. You should have the code in your main function to do this already. Move it into the getUserOption function and add a return statement to return the user's choice.

Update the main function so it calls this function:

```
int main()
{
    while (true)
    {
        printMenu();
        int userOption = getUserOption();
        // option processing logic here
    }
    return 0;
}
```

Build and test!

The processUserOption function

Now to make a function that processes the user input we captured and returned from the getUserOption function.

```
void processUserOption(int userOption)
{
    if (userOption == 0) // bad input
    {
        std::cout << "Invalid choice. Choose 1-6" << std::endl;
    }
    // more if statements here to process other menu options.
}
```

Does this function return anything? Does it receive anything?

Add the logic for the other menu options to the processUserOption function. Then call it from the main function:

```
int main()
{
    while (true)
    {
        printMenu();
        int userOption = getUserOption();
        processUserOption(userOption);
    }
    return 0;
}
```

Build and test.

Create functions for each menu option

Now let's function-ise each menu option so it is dealt with by a separate function, e.g.

```
void printHelp()
{
    std::cout << "Help - your aim is to make money." << std::endl;
    std::cout << "Analyse the market and make bids" << std::endl;
    std::cout << "and offers. " << std::endl;
}
```

Go ahead and create a function for each menu item. Then in processUserOption, call these functions, e.g.:

```
void processUserOption(int userOption)
{
    if (userOption == 1) // bad input
    {
        printHelp();
    }
    // more if statements here to process other menu options.
}
```

You can call the functions whatever you like. For example, I gave them the following names:

- printMarketStats
- enterAsk
- enterBid
- printWallet
- gotoNextTimeframe

Build and test.

You can find a detailed breakdown of the syntax for functions in the textbook in Chapter 8 p259.

Challenge

Consider the following program, which uses a fancy C++ technique called function pointers to store the menu in a data structure.

```
#include <iostream>
#include <map>

void printHelp()
{
    std::cout << "Help - your aim is to make money. Analyse the market and make bids and of
}

int main ()
{
    // map from ints to function pointers
    std::map<int,void(*)()> menu;
    // connect 1 to the printHelp function
    menu[1] = printHelp;
    // call option 1
    menu[1]();
}
```

Can you use this idea to make a much leaner version of ‘user inputs int, program calls function’ idea?

Conclusion

We have refactored the program into a set of functions, each with a clearly defined purpose.