Object Oriented Programming Course Notes

Felipe Balbi

April 23, 2020

Contents

Week 1

Key Concepts

- Write, compile and run a C++ program that prints messages to the console
- Use the standard library to do text I/O in the console
- Write and call simple functions

1.004 Demonstrating the first example application: merkelsim currency exchange simulation

Merklerex is a currency trading simulator. It allows us to work on a snapshot of real trading data downloaded from an exchange service. We can keep track of our wallet, trade currencies by buying and selling them and try to make "money".

1.005 Demonstrating the second example application: otodecks DJ application

OtoDecks is a DJ application. The second half of the course will be building this application. It allows us to load different tracks and mix them together on the fly.

1.006 Reading for Topic 1

The textbook for this course is:

Horton, I. and P. Van Weert Beginning C++17: From novice to professional. (Berkeley, CA: Apress, 2018) 5th edition [9781484233665].

We will point you to the specific sections of the textbook you should read in the lesson worksheets. In this week's worksheets, you will see the following sections:

- Chapter 2, p.38: std::cin
- Chapter 5, p.138: while loops
- Chapter 8, p.259: functions

Accessible from here.

1.101 Introduction to Topic 1

During topic 1 we learn to write, compile and run C++ programs.

1.102 C++ is a low-level language

But high-level and low-level we refer to the level of abstraction between the program we write and the HW it runs on.

When we write SW in a high-level language, such as JavaScript or Python, we don't need to worry much about how the underlying machine is carrying out its work. Details such as memory management is taken care of by the language itself.

Conversely, a low-level language, such as C/C++ or Assembly, give us access to very small details of the machine. We must handle memory manually, sometimes control specific machine registers and so on.

1.105 Writing our first program

In order to convert a C++ source code into an executable, a compiler will parse the C++ source and generate a target machine assembly, from there we generate object code which is later linked to produce the final executable.

```
g++ -S main.cpp  # produces assembly
g++ -c main.cpp  # produces object code
g++ main.cpp     # compile, assemble, and link
```

1.107 Development environments

During this course we will use an IDE, or an Integrated Development Environment, which is basically a text editor combined with all other tools needed during the development phase: a debugger, compiler, linker, file manager, console, and so on.

The course provides a web-based version of Visual Studio Code for our consumption. It's accessible from here.

Week 2

Key Concepts

- Write, compile and run a C++ program that prints messages to the console
- Use the standard library to do text I/O in the console
- Write and call simple functions

1.601 What is refactoring?

Refactoring is modifying code without modifying behavior. In other words, it's the practice of extracting "ideas" from code and splitting it into a separate function, either for aesthetics or so the code can be reused elsewhere.

1.603 Write a print menu function and a processOption function

With the knowledge of refactoring, we can refactor our big `main()` function by splitting it into smaller functions.

We introduce `printMenu()`, `getUserOption()`, and `processUserOption()`. Each of these functions is concerned with a single thing. They know the minimum amount of details they need to know.

For example, all the `std::cout` statements that make up our menu, can be moved to `printMenu()`.

```cpp
void printMenu(void)
{
  std::cout << "1. Show Help" << std::endl;
  std::cout << "2. Stats" << std::endl;
  std::cout << "3. Offer" << std::endl;
  std::cout << "4. Bid" << std::endl;
  std::cout << "5. Wallet" << std::endl;
  std::cout << "6. Continue" << std::endl;
}
```

All that code can be removed from `main()` and replaced with a call to `printMenu()`. The same idea is applied to `getUserOption()` and `processUserOption()`.

1.605 Write menu functions

Following the same idea as the previous section, let's augment `processUserOption()` with smaller helper of its own. At the moment, these helper functions will only print out the relevant message but eventually more functionality will be added.

We have a total of 6 menu options and, therefore, will add 6 functions. They are:

1. `printHelp()`
2. `printMarketStats()`
3. `enterOffer()`
4. `enterBid()`
5. `printWallet()`
6. `gotoNextTimeFrame()`