C++ fundamentals with use cases from finance

Lecture 1: Introduction to C++

Ivan Zhdankin

Course Structure

- 2 weeks: 2 lectures and 2 tutorial each week
- In this classes the approach to learn C++ will be to code as we go - please make sure you have Visual Studio Installed and C++ ready for the tutorials
- There will be a multiple choice test after the class
- The questions are based on the lecture notes and the practical examples we cover
- To prepare for the test during every tutorial we will have a quiz that consists of sample questions similar to the test ones
- **Bonus**: During each of the lectures there will be questions from real interviews on C++ and other topics

Final test

- Will take place in QMPlus
- Exam Date, Time: Saturday 5th November
- Duration: 1h, any time during the day
- Set of 50 multiple choice questions
- 60% pass threshold

Course Content

- Lecture 1: C++ Introduction
  - Getting Started with C++
  - Tools
  - Variables and Fundamental Types
  - Functions
  - Flow of Control
- Tutorial 1: interest rate curve interpolation
- Lecture 2: Introduction to OOP in C++
  - Writing Classes
  - User Defined Types
  - More on Functions
- Tutorial 2: Constructing a curve using functions
- Lecture 3: Defining your own structure in C++
  - Operators
  - Templates
  - Indirections
  - Memory
- Tutorial 3: operating with a curve using operators and templates
- Lecture 4: Standard Library and improving your C++
  - Indirections and Inheritance
  - The Standard Library
  - Lambda
  - Exception handling
- Tutorial 4: creating saving and business account bank

Demo: Visual Studio Community Edition installation

- **Mac:**
  - To install and get started please follow: `https://youtu.be/zIIYN_PyUAM`
- **Windows:**
  - To install and get started please follow: `https://docs.microsoft.com/en-us/cpp/build/vscpp-step-0-installation?view=vs-2019`
- To run the code provided in the *.txt* files copy the text from the files and paste it into your *.cpp* file
- **Note:** there can be errors in the preprocessor steps (# *include* commands). Correct them and make sure you are able to run the code.

Why C++

- C++ is a general purpose language
  - All kind of applications around us are/can be developed in C++
  - C++ is used for scripting (consecutive list of commands)
  - C++ is used for Object Oriented Programming (class and objects)
- C++ is about power, performances and memory, their management and control
- C++ is a very popular language
  - Among top languages for the open source projects
  - Among top languages for college students
  - $> 10\textit{mio}$ developers use C++

Is it that hard

- The answer is that Modern C++ is not that hard for beginners
- We do not need to learn *C* first
- Some features are done in C++ in much simpler way
- If you know $C\#$ or Java, learning C++ becomes very easy
- If you already have some experience with any programming language learning C++ is mostly about learning the syntax of the language
- If you do not have any experience in programming language, C++ can be used as a starting point
- C++ is used for scripting (consecutive list of commands)

The Standard Library

- Pretty much all language comes with the Libraries
- In mathematics we have the set of assumptions and the theorems and lemmas are derived from the assumptions. We can use the theorems and lemmas for our further derivation and applications without havering to proves them over and over again.
- In programming we have the syntax and some basic rules of the languages and the Libraries are implementations and derivations of the useful features that we are going to use over and over again.
- Build in capabilities in C++ are implemented in the Standard Library and we will find a lot of useful implementations in the library:
  - String Class
  - Collections (stack, queue, vector, map) - structures used for storing the data
  - Smart pointers - pointers helps with memory management
  - File and Screen IO

Standardization

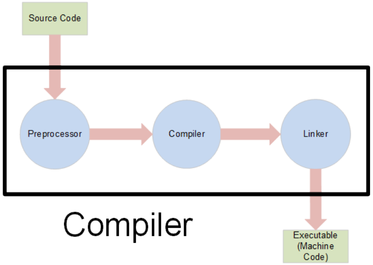
- There is no person or a company that C++ belongs to
- There is a standard that defines the langues and the standard committee that follows the standard
- There are many vendors that implement the compilers, the libraries and other related tools
- Most of those vendors have the representative on the committee
- Some vendors may implements the feature behind or in advance of the standard depending on their own history
- In this course we will be using Visual Studio
- You should be aware of the fact that there are multiple vendors and compilers available each of them meet the standard at their own pace and may be different
- Some useful pages to look up:
  - isocpp.org - webpage for standard C++
  - en.cppreference.com - webpage for looking up keywords, libraries, capabilities of compilers

Tools

- We need a **compiler** and **linker**: these tools transform the source code to the something which can be executed on the platform you are working
- We need an **editor** for the source code writing, reading and editing the source code. The editor should also help us in writing the programm by highlighting the potential source of errors and helping with documentation
- A good thing to remember about the programming is that the programm does what we tell it to do
- We need a **debugger** as we are going to write a bugs. Debugger can hep to find the bugs by pruning the programm step by step, pausing the programm and looking at what happens at each incremental step.

Building

- We have mentioned compiler and linker - the tools is under the hood of the process called building
- There is also *preprocessor* step - modify the code to preprocess command that starts with #



- Building consists of the following main steps:
  - We start with the Source file test.cpp
  - The compiler compiles the files with the Source code into Object file test.obj
  - If we have multiple Source files, each of them will be compiled into corresponding Object file
  - Then all of the Object files will be linked together by the linker to produce a single Executable file test.exe
- In large applications you have many object files
- On top of the files we normally link the standard libraries or some other code which was coded up before, that happens at **preprocessor** step
- Different stages of building can lead to different types of errs: Compiler and Linker errors.

Some popular tools

- Compiler and Linker:
  - GCC
  - Clang
- IDE (Integrated Development Environment):
  - Visual Studio (for Windows)
  - Visual Studio Code (for Mac and Linux)
  - CLion
- Different Online Compilers
- To start go to isocpp.org/get-started

Visual Studio

- Visual Studio os IDE, meaning that it has **editor**, **compiler**, **linker**, **debugger** and some other related tools
- There several editions of the Visual Studio:
  - Communities - completely free, it is not restricted to what you are going to use your code for
  - Professional - not free, for individuals and small team, who work in large companies
  - Enterprise - not free and has some additional features
- For details go to visualstudio.com/vs/compare
- In Visual Studio we can create different kind of applications:
  - Phone and other client apps including games
  - Console Applications - can only use input from the keyboard
  - Services - background, infrastructure projects
  - Server - handles the requests from the clients and react to them
  - Libraries can be used by all of the above
- We are going to create only *console application* in this course as they are the simplest and let us learn the syntax of C++

Demo: Hello, world

- Console application structure:
  - There are different ways to comment out something in C++
  - We have the following preprocessor commands that starts with a $\#$ sign
  - The preprocessor is actually the step before compiling and it brings other files into the file that we compile. In this case we bring the new libraries.
  - **std** identifies the standard namespace
  - **::** scope resolution operator
  - **cout** console output
  - $<<$ indicates whatever comes next is going to be sent to the console output
  - **endl** end of the line
  - **cin** used to get the console input
  - $>>$ extraction operator

Compiler Errors

- In real life we make errors and compiler helps us to identify and correct them
- All errors that you see during the building process are **compiler** errors
- Compiler can also give you warnings which new developers should not ignore

Variables and Fundamental Types

- C++ is a strongly typed language meaning that variables can be only of a particular type
- When we declare the variable we also declare its type
- There are some fundamentals types build in C++:
    - Numbers, single characters, boolean
- There are user defined types - the types that we define ourself or defined in Libraries for us:
    - Strings, Dates
    - Structs, Classes
- The user defined types has the same functionality as fundamental types

Fundamental Types

- When we define types we need to include short name for it before the variable: *int*, *char*
- Example: *stringFirstName*
- For integers: *int*, *long*, *short*
- For real numbers: *float*, *double*
- For characters: *char*
- For boolean: *bool*
- For details look up: ”https://docs.microsoft.com/en-us/cpp/cpp/cpp-type-system-modern-cpp?view=vs-2019”

Auto

- Sometimes to define the variable type can be difficult of impossible as we do not know it in advance
- For such cases the keyword *Auto* exists:
  - *Auto* tells the compiler to deduce the type. For example, if variable is assigned from the function return and return is **int** the compiler automatically assign integer type to the variable
  - The variable is still strongly typed - we can not change the type later on
- Let us take a look at the demo: "*Fundamental_types.cpp*"

Casting

- The compiler can convert one type into another when they are compatible: for example integer to double, showing the warnings
- Using **casting** you can make this intention of converting one type to another clear to the compiler. In this case no warnings will be shown.
- Be careful with casting, as once converted you can easy forget about this.
- Let us take a look at the demo: '' *Fundamental_types.cpp*''

Function

- **Function** is a piece of code that we would like to use repeatedly
- Functions are called by other piece of code
- The functions must be declared to be called
- Functions have a return type
- Functions that do not return anything are proceeded with default type *void*
- Function can have parameters, which have types and names
- The functions are type safe, and the rules that work with type conversions also work in functions
- In case the type of the variables passed into the function are different from the types of the function parameters the compiler will warn you
- The same works with the return
- Functions demo: ”Functions.cpp”

Overloading

- Assume we now have to sum up 3 numbers, or 4 numbers?
- Shall we create new name for the function and code it up?
- In C++ as soon as the compiler can distinguish between the arguments we do not need to change the name of the function
- This is called *function overloading*
- However, return type can not be used to distinguish overloads
- Taking sane number of parameters but of different types can be risky
- Functions demo: ”Functions.cpp”

Files

- So far we were using only one file for the source code with extension .cpp
- Imagine the big organization with code that has several thousands lines
  - Requires compiling it all when making small changes
  - Difficult to coordinate the work of the developers
  - Difficult to find anything in one single file
- In practise the code organised in multiple files to resolve those issues
- In case of multiple files one has to compile each of the files in to object file and then link them all using the linker
- If we use the other files in our code we need to explicitly tell the compiler about it

Header Files

- In previous implementation we had to declare the functions before we can use them
- Consider the project that has tens of functions and classes, this may become frustrating
- We can put all of the declaration in a separate files and then include the file into the code we would like to compile
- The file is called *Header* file and has extension .h
- The directive to include the file would ne: *#include*
- Everything that starts with # will be included into the code by the *preprocessor* and after that the whole file will be compiled
- As a result your code would look nicer, be more maintainable and easier to understand
- Demo of moving function to a separate header file

Errors

- There are two steps when building the code: compiling and linking.
- The errors that happen at different steps have corresponding names
- If we forget to declare functions the resulting errors will be *compiler errors*
- If we include the declarations but forgot to implement the functions the resulted error will be the *linker error* as the linker can not link the files
- Do not fix what is not wrong
- Demo of errors

Flow of Control

- Normally the code is executed from top to bottom
- There are some key words that can change it:
  - if
  - else
  - while
  - for
- The following expressions are logical:
  - $(x < y)$
  - $(x > 3)$
- The following operators are used for comparisons:
  - $<, >, >=, <=, ==, !=$
- The result is *true* or *false*

Switch, break and immediate if

- If there are many *if* statements the code become difficult to understand
- In this case it is better to use *switch* word and list the cases below
- If we want to escape from the look we need to use *break* word
- *Immediate if* has the following form:
- $result > 0?1 : 0;$
- Let us take a look at demo ''*Flow_of_Control.cpp*''

Interview Questions: C++ and development environment

- What is IDE?
- Which IDE have you used in practise?
- What are the advantages of C++ over Python?

Interview Questions: Compilation and Debugging

- Mention main three steps in building process from Source Code to the Machine Code?
- What are the extensions of C++ files?
- What debugger is responsible for?
- What type of errors one can have when building a project?

Interview Questions: Variable types, Header Files, Functions

- What is casting?
- What is an extension for the header file?
- Describe in two words the following:

```
double product(double x1, double x2){
return x1*x2;}

int product(int x1, int x2){
return x1*x2;}
```

Useful references to study C++

- C++ Primer, by Lippman, Lajoie, Moo
  - Good C++ book for programmers of different levels
- The C++ Programming Language, by Bjarne Stroustrup
  - With more focus on specific features of C++
- YouTube tutorial by Derek Banas on C++ starting from beginning and with coding examples
- http://www.allitebooks.org/
  - Free source for many IT and programming books