**Beginning C++ Programming by Frank J. Mitropoulos**My Course Notes

Troubleshooting

* CMake/NMake errors:
* Ensure that your antivirus isn’t blocking the build process by quarantining files.
* Restart computer.
* “Error: could not load cache”:
* Tools > CMake > Reload CMake Project.

Introduction

* There are multiple versions of C++: C++98, C++03, **C++11**, **C++14**, and **C++17**. The digits represent the year that version was released. The former two versions are referred to as classic C++, while the latter three are referred to as modern C++.
* C++98 was the first official standard. C++11 added many new features to the language. The other versions mainly corrected issues with the language or simplified pre-existing features.
* The process to create run an application is as follows:
* The developer inputs C++ code into header (.h) and source (.cpp) files in an IDE.
* The compiler converts the high-level C++ code into low-level machine/binary/object code (.obj).
* The linker combines our code with other libraries and outputs an executable (.exe).
* The course will use CodeLite since it’s free, but the instructor stated that CLion by JetBrains is his IDE of choice. I have opted to use CLion as I have access to it and think highly of JetBrains.

Installation and Setup

* You can execute C++ code through an IDE, the CLI (command line interface), or via a website such as ‘repl.it’.

Curriculum Overview

* No notes taken.

Getting Started

* Code completion aids the developer by predicting what they will input and suggesting it to save time.
* Pre-processor directives don’t end in a semi-colon.
* *cout* is tied to the console and is used to output data.
* *<<* is the insertion operator that outputs the following data.
* Text between quotation marks represents a string literal: e.g. “Hello world!”.
* Statements end in a semicolon.
* *‘return 0;*’ in *main()* to indicate that there weren’t any problems.
* *cin* is also tied to the console and is used to input data from the user.
* *>>* is the extraction operator that stores the input.
* **Variables** store data.
* To declare a variable state its type and give it a name, e.g. ‘*int favourite\_number;*’.
* *#include <iostream>* includes the input/output library where *cout* and *cin* are defined.
* *endl* prints a new line and flushes the buffer.
* To build means to compile it and link it. This results in object files. The build process saves time by only building the files that it has to.
* The clean process removes the object files, but then you must build your program again.
* **Compiler errors** enforce programming rules. It does this by identifying syntax and semantic errors.
* **Syntax error** refers to when the structure of the code is incorrect, e.g. ‘*cout << “Errors << endl*’ in this case the trailing quotation character is missing.
* **Semantic error** refers to when the structure is correct, but the code is undefined, e.g. ‘*int a = b + c’* if *a* and *b* are *int*s and *c* is a person then it may not make sense to add them.
* Making one error will lead the compiler to detect many errors. So fixing one error will resolve many compiler errors.
* **Compiler warnings** occur when code can be compiled, but has potential issues, e.g. printing an uninitialized variable ‘*int data; cout << data;*’.
* Both warnings and errors should be avoided whenever possible.

Structure of a C++ Program

Variables and Constants

Arrays and Vectors

Statements and Operators

Controlling Program Flow

Characters and Strings

Functions

Pointers and References

OOP - Classes and Objects

Operator Overloading

Inheritance

Polymorphism

Smart Pointers

Exception Handling

I/O and Streams

The Standard Template Library (STL)

Bonus Material and Source Code