**Week 4: Understanding C++ (Part A)**

Nate Bachmeier

Some Date

TIM-8101: Principals of Computer Science

Northcentral University

# What is machine and assembly languages?

While computers can do amazing things, they are no smarter than elementary school children (Kurzgesagt, 2015). They can perform basic arithmetic and tell if two short series of bits match one another. To make software as complex as say a modern web browser, it takes many layers of abstraction to build up primitives so that these very low-level operations can be appropriately orchestrated.

At the lowest layer is the assembly language which represents the literal load and store binary words between memory and register (Scott, 2015). These operations tend to align 1:1 with the processor’s specific implementation. Different processors have different instruction sets as they had different design goals. Consider the difference between RISC and CISC systems such as ARM and x86. ARM is used by mobile devices and has a finite power supply versus x86 is used for general purpose desktop machines.

# Where is C++ used in the real world

In the utopian model software is written once and able to run correctly on either the RISC or CISC processor. This led computer scientists to create higher-level languages, such as C and Fortran. Software written in these languages could then be transformed through a compiler into the target architecture.

Over time the need to write more complex abstractions gave rise to C++, which provided extensions to base C language. Examples include oriented design patterns, templating, and structured exception handling.

Initially these extensions were provided by compiling first from C++ to C and then C to assembly (Scott, 2015). Scott also describes that many prototype languages and even mainstream languages, such as Python, use this recursive compilation approach

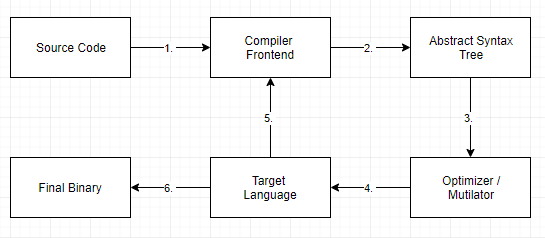
There are higher level languages that have been built on-top of C++ such as C# or Java. However, C++ is still the language of choice for many scenarios as it is the “highest low-level.” Since it can be translated into raw assembly language it can run virtually anywhere. Contrast that with Java which has a hard dependency on the Java Virtual Machine (JVM) runtime.

# What is the purpose of a compiler?

A compiler is responsible for translating source code from one format to another. For example, the LLVM compiler collection can translate C++ code into byte code called Intermediate Representation (Wilde, 2016).

Tool chains can then be built to analyze or mutate the IR representation to other forms. For example, an optimizer might simplify mathematical expressions, or a static analyzer look for buffer overflow scenarios.

After the tool chains have executed the build engineer can transform the IR into the next target. Perhaps it is an interop layer for the Java Native Interface (JNI) or into x86-64 assembly to be executed on a modern server. This cycle of consume, parse, emit continues until the abstractions have been properly generated or reduced.



# References

Kurzgesagt. (2015, December 8). *Quantum Computers Explained – Limits of Human Technology.* Retrieved from YouTube: https://www.youtube.com/watch?v=JhHMJCUmq28

Scott, M. (2015). *Programming language pragmatics.* San Francisco, California: Elsevier Science .

Wilde, M. (2016, January 26). *A Brief Introduction to LLVM.* Retrieved from YouTube: https://www.youtube.com/watch?v=a5-WaD8VV38&feature=youtu.be