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

Nate Bachmeier

March 31st, 2019

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 by orchestrating very low-level operations.

At the lowest layer is the assembly language which represents individual load and store operations between memory and hardware registers (Scott, 2015). These assembler operations align 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 by 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 any RISC or CISC processor. How the application differs should be a matter of performance not correctness. This helped steer developers toward higher-level languages, such as C and Fortran. Software written in a portable language can then be transformed through a compiler into the desired target assembly architecture at build time.

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 (Liberty, 1997).

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

There are higher level languages that have been built on-top of C++ such as C# and Java. However, C++ is still the language of choice for many scenarios as it is the “highest low-level language.” The combination of abstract language constructs combined with the ability to compiled into raw assembly language-- allows for very complex solutions to be executed on virtually any compute environment.

# 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 that next target is an interop layer for the Java Native Interface (JNI) or x86-64 assembly. The cycle of consume, parse, emit continues until the abstractions have been properly generated or reduced. Given the relative flexibility to this model, selecting the final targets can be a business decision instead of a technical limitation.

# Diagram of Compilation Process

1. The application source code is fed into the front-end compiler
2. The compiler will generate the Abstract Syntax Tree (AST)
3. The optimizer and other mutation plugins alter the AST
4. The AST is translated and linked into the target language (platform)
5. The compiler chain will optionally feed the output into another iteration
6. The final solution output will be staged and is ready for deployment

# References

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

Liberty, J. (1997). *Teach Yourself C++ in 21 Days.* Sams Publishing.

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