Chapter 1

Daniel Lee

May 7th, 2022

Introduction

- Modern day programs often rely on software to convert low level abstractions to higher level virtual programns
- Compilers are programs that convert a program from one program to another to prepare it for execution
- To do this we need a means of understanding the syntatical meaning of a language and have a scheme to map it from one language to another
- There is a front end to handle the target language and back end to deal with the source language
- Using a blackbox scheme we can potentially view a compiler as this:
  source program $\rightarrow$ **Compiler** $\rightarrow$ **target program**
- Many people believe compilers only produce programs written in assembly, however it is possible for compilers to produce an output that is a high level language
- Interpreters convert an instruction line by line
- Compilers translate the entire program before executing
- Some languages combine both compilers and Interpreters
  - An example of this would be Java
  - Java is compiled into byte code and then the bytecode is ran by an interpreter by the JVM
- A good compiler incorporates many different aspects of computer science to optimize performance

Fundamental Principles:

- Compilers are large, complex programs with many different ways to implement them
- With this in mind, there are certain principles to adhere to for when designing a compiler:
  - Compilers must preserve the original meaning of the program being compiled
  - Compilers must improve input program in a discernible way

Compiler Structures

- A compiler must understand the source program it takes in as an input and maps its functionality to a target machine
- Compiliation thus can be broken down into two subprocesses: frontend and backend
- Frontend focuses on understanding the source language, it maps the source language into the IR or immediate representation to be used by the machine
- Backend focues on mapping the intermediate representation into the instruction set of the program
- A two phase structure allows for simple restructuring of programs:
  - We can develop multiple backends for a single front end to have a source language run on multiple machines
  - Alternatively we can develop multiple front ends for multiple back ends, allowing us to run multiple languages on a single machine using a common immediate representation
  - A front-end produces an IR, which we can further optimized by a mechanism called the optimizer
  - An optimizer can make multiple passes on the IR and rewrite the IR to produce a faster program
  - Source program$\rightarrow$ Front-end $\rightarrow_{IR}$ Optimizer $\rightarrow_{IR}$ Backend $\rightarrow$ Target Program
  -

Translation

- Translation is all about converting abstractions into more concrete processes

The Front End

- Scans program to see if syntax is valid and reports to User Otherwise
- From a mathematical standpoint a source language can be viewed as a set of Strings defined by a rule called **Grammar**
- A scanner and a parser determine if a program follows grammar
- Programming languages often refers to words based on parts of speech, using rules based off of parts of speechs we can have multiple rules define multiple programming language

- Sentence → Subject verb Object endmark
- A scanner takes a stream of characters and converts it to stream of classified words, this can be viewed as a pair (p,s)
  Example: **(noun, Compilers)**
- In practice, the spelling of the words may be stored in a hash table and represented in pairs with an integer index to simplify equality tests
- Once a stream is converted into classified word, we can parse to find derivations for a Sentence
- With every step, we can rewrite a term in the Sentence

Intermediate Representation

- Immediate representations can exist in many forms, some compilers represent program as assembly others represent it as a graph
- The optimizer examines the IR code and rewrites to optimize it

The Back End

- The back end of a compiler determines how resources are gonna be allocated to execute a programn
- From the values that reside in memory to the ones stored in the register, the back-end responsible for optimizing the IR and using the machine instruction set to execute an instruction