Constructing Compilers

The foundation of modern computation

For my grandpa, who would remind me that I could always be a carpenter when that computer-fad wore out.

Preface

This book is to teach you how to understand the inner workings of a compiler by constructing a byte-code compiler and virtual machine. The compiler will compile the language f' (f-prime) described in this book. This approach, as opposed to outputting instructions for a specific architecture, allows us to write f' code and execute it on any machine that has the f-prime compiler installed.

The main intent to writing this book is to explain the theory of compilers and apply those theories to a real-life project. Though some topics will emphasize the mathematics behind the application, a majority of this book will focus around the implementation and tools.

A skills approach

Writing a compiler and explaining how it functions is not an easy task. In my experience, the best approach is to learn by doing. There are a variety of skills required in order to build a working compiler and, in order to effectively understand how these components work, we will be constructing applications that revolve around specific skills.

Why learn how to write a compiler?

In order to construct "user-friendly" software, the way we interface with the computer must also be "user-friendly." We interface with the computer with a large array of technologies, from assembler to ruby -- the language we utilize plays a major role into determining how friendly the software is.

As we become more dependent on software for automating tasks, we also demand more flexibility with our applications. In order to keep up with the demand, advances in compiler technology will continue to arise and, eventually, be implemented in our language of choice. Whether this is by the team that maintains the compiler or by us contributing to the language, if we want to keep up with the technological advances; we have to understand the inner workings of a compiler.

Prerequisites

There are a few assumptions made in this book. I assume you are already familiar with the C language. If not, learning C through constructing a compiler is not the best approach. Other concepts and including recursion, data structures, stacks, pointers, linked lists, and binary trees.

System Requirements

We will be utilizing GNU Make for compiling our compiler. This can be modified to support your operating system. All code examples have been tested on Mac OSX (64bit) and CentOS (64bit). I cannot guarantee the integrity of the compiler on the Windows operating system. Maybe in a future revision.

Bugs

I personally don't like having code samples that do not work out of the box. Due to there being several C compilers and operating systems: I can only make a guess to how your C environment is set up. As a helpful assurance, throughout this book I won't rely on any third-party libraries aside from explaining how to extend our language. If your code isn't compiling, cross-examine with the github repository and check your Makefile settings.

If you run across any bugs in the code or find issues in the text; send a pull request on github.

CHAPTER I

Introduction