The hard way

Learning programming via compiler developement

Lilly H. St Claire

July 8, 2024

Contents

1 Preface

To the uninitiated programming can be quite daunting. From the obscure syntactical choices prevelant in programming languages, to the high level problem solving abundant in every programs goal, programming can pose a unique level, and number of challenges that wouldn't otherwise experience. In writing this book I wish to teach my audience the skills required to make long term projects.

I aim to achieve this by explaining in detail the process required to make compilers of various levels. From a project taking an afternoon, to week even month long processes.

This book is aimed to be understood by *both* true beginners who have never touched code, as well as programmers who have quite a portfolio. Anyone who is interested in creating compilers should, hopefully, be interested in this book.

2 Environment

Every beginner must first create an environment in which you can edit, and run their programs. Due to this tutorial being in Ocaml we will need to install its compiler by following the instructions at https://ocaml.org/docs/installing-ocaml.

Once that has been completed we will be using bash in this tutorial, many of the commands will be similar (if not identical in windows batch, rescourses can be found online).

3 A bit of a mindfuck

To begin with we will create a compiler for Brainfuck an esoteric programming language (meaning a programming language not designed for serious use) created by Urbun Müller in 1993. The stated goal was to make the programming language with the smallest possible compiler.

Brainfuck consists of an infinite number of containers (known as "cells"), each cell is ordered by assigning it a number. We store a number, known as the "pointer" the user can only act on the cell corresponding to the number held by the pointer. We can do the following actions on the pointed cell.

- + : add one to the cells stored number
- - : subtract one from the cells stored number
- , : read a letter from the user and store it in the cell
- . : print the letter stored by the current cell to the screen

You should note that letters inputted by the user (and outputted to the screen) are stored using an encoding system known as ASCII. With ASCII each character is assigned a number from 0 to 127, along with a few control numbers (such as end of transmission, carriage return, new line, etc.).

Additionally we can use `>` to add one to the pointers value, and `<` to subtract one from it.

And finally we can repeat a series of actions by starting a "`loop` " These loops begin with `[` and end with `]` . The program when running into the end of a loop and if the pointed to cell is zero, then it will go back to the start of the loop and do the actions from it to the end statement again.

Any character that does not control the current cell, pointer, or program is ignored by the compiler.

We will test our compiler (by the end of this section) with the following brainfuck program.

```
[
    ,.          read a character into cell 0 and print it
    ----------  subtract 10 from that character
] if that character was 10, then 10-10 = 0 and the loop will end
note: 10 is the ASCII value for a new line (or enter key).
```