

Introduction

The purpose of this project is to demonstrate practical knowledge of the java programming language, regular expressions, finite automata, software engineering and design, as well as general problem solving skills in a group dynamic. Throughout this project we have applied knowledge acquired during our undergraduate studies to design, build, test and deploy a working compiler for a LITTLE language.

A working compiler, for even a LITTLE language, is a positive for future endeavors and job searches. It serves to show that we not only understand how to program but that we also understand the foundations of the computer science field. By completing this project, we have gained insight into each of the elements that make up computer science and how they fit together to create a finished product.

In the course of this report we will touch on the background and purpose of a compiler, each of the components that make up a compiler and a summary of what we learned, what we could have done better, where we will go from here.

Background

In general, a compiler takes instructions, written in a specific programming language, and turns them into a machine assembly language. This is necessary because a computer processor cannot 'read' written instructions. A compiler is made up of several different sub-routines or steps.

The first step is scanning. A scanner reads character input, in this case a computer program, and converts the incoming characters into tokens using a grammar made up of regular expressions. This process is known as a lexical analysis.

The next step is parsing the tokens using the defined grammar for the language. During this step a data structure, usually a parse tree, is created using the input. The parser looks at the token and compares it to the language grammar. If the token is a part of the grammar it is accepted. If it is not a part of the grammar it is rejected.

Methods and Discussion

Scanner

To begin, we looked at many options for implementing the given grammar and reading, we decided on a strategy that would simplify development. We determined that Java was the programming language we are the most comfortable with, so we decided to use it. While researching Antlr, we saw that setting it up in an IDE, Integrated Development Environment, was complex and IDE specific. So, given our familiarity with Linux and OSX we decided to do all of our development in the terminal using a text editor. This made running the code simple with no extra IDE configuration. Given the differences between Windows and Linux shell commands, we decided to use Linux as our default environment for creating, running and testing our bash scripts. So far, no issues have been found when using OSX to run and test our bash scripts. However, if issues arise later in development Linux will become the only operating system used.

After researching how to use Antlr and run the scanner in general, we ran into few difficulties. Some minor grammatical errors and difficulties with regular expressions were the worst issues we ran into getting our grammar set up. At first, getting everything to communicate with Antlr properly caused some minor trouble. Running demonstration files, when they were available, helped us work through those difficulties. It was the demonstration files and what we learned from them that led us to use Linux and OSX rather than Windows for our development.

Parser

Symbol Table

Semantic Routines

Full-fledged Compiler