Maxwell Herron
CSC 451

Micro-C Compiler SDD

General Overview

This project is to be used to compile and run programs written in the Micro-C programming language. It is written in Java, and consists of four main packages: scanner, parser, semantic analyzer, and code generator. These packages all lie within the src folder of the root directory, titled Compiler. Each package is described in detail in the following subsections.

Scanner

The scanner used for this compiler is generated by a Jflex file, and all the necessary files for this component are found in the scanner package. The design of my Jflex scanner is as follows: the jflex file Scanner.jflex contains several regex patterns that the generated scanner will parse for. It also contains the instructions for what the scanner will do upon finding each pattern. There is a Token class, which has the attributes lexeme and tokentype. The lexeme is the literal string that is parsed, while the tokentype is another object, created by the TokenType class. TokenType is simply an enum of all necessary types of tokens for this assignment. Finally, there is a LookupTable class. The LookupTable is a hashmap that is used for all the symbol, operator, and keyword tokens. When the scanner picks up any of those tokens, it sends the lexeme as a key to the LookupTable to get back the value associated with it, and assigns that value as its TokenType. There is also a class for a custom exception object, called BadCharacterException. This exception is thrown when the scanner picks up a lexeme that is not associated with any of

the regular expression patterns defined in Scanner.jflex. It holds an error message that lets the user know what the bad character was.

Parser

In its current state, my compiler is actually using a recognizer, and not a parser (the parser will come later). The purpose of the recognizer is to simply read in an input, and determine if that input matches the requirements of a valid Micro-C program. The way this works, is that given a provided list of production rules, it runs through those rules to determine the validity of the input. Each non-terminal symbol is represented as a function in Recognizer.java. It works through the production rules, and makes a recursive descent through the rules, calling the necessary functions that are mapped to each nonterminal, until it finally reaches terminal symbols. Tests for my recognizer can be found in RecognizerTests.java, within the parser package. These tests are run on six different production rules, and are meant to be valid Micro-C expressions.

Symbol Table

In its current state, the symbol table is not integrated with anything useful, and is only able to be tested. The purpose of the symbol table is to encapsulate information about any identifier that the recognizer comes across. Some relevant information to hold would be the kind of identifier (variable, function, array, program), along with the datatype of the identifier. This will allow the compiler to keep track of all the identifiers, and actually have an idea as to what they are, as opposed to simply seeing them as just generic identifiers.

Semantic Analyzer

// Will describe when created

Code Generator

// Will describe when created