CSC 450

Sam Giese

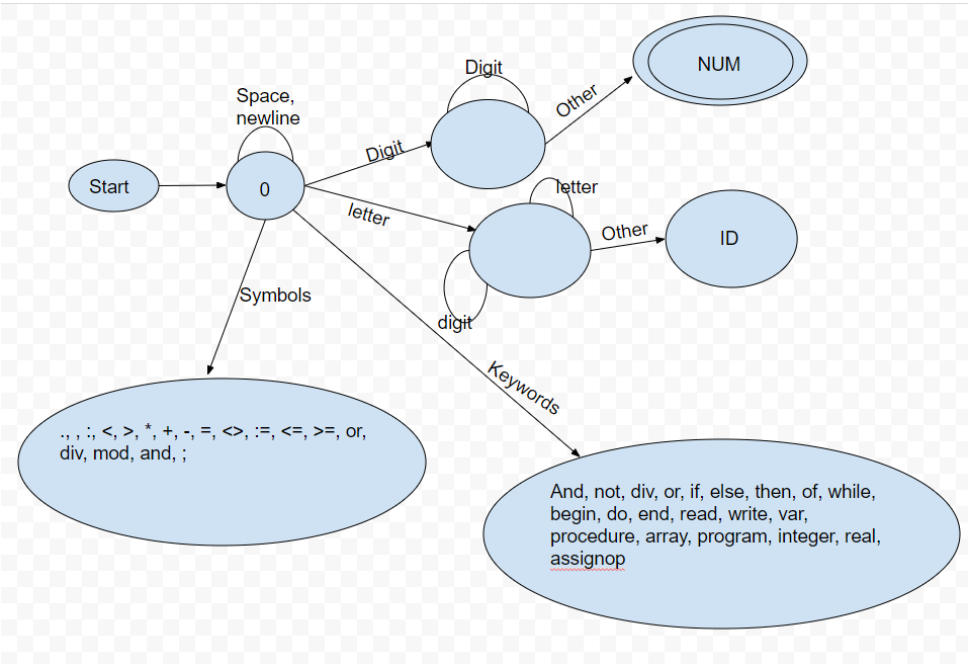
Micro Pascal Compiler

Overview:

The finished product of this project will be a completed set of programs that when put together operate as a compiler for the mini-pascal language. The parts consist of a Scanner, Parser, Symbol Table, Syntax Tree, Semantic Analysis, and Code Generation. All of these sections together will take an input of micro pascal code, and convert it into MIPS assembly language.

Scanner:

The MyScanner class will recognize certain keywords and symbols as tokens, and it will determine them as such depending on their content. The following FSM is a representation of how the tokens will be recognized and assigned:

The 

Jflex file MyScanner will generate a java class scanner that will incorporate the Token java class.

The Token java class will be used to return each token back to the scanner. This scanner will be tested with the help of the TestMyScanner java class, which will import the text file, and run through it and break it into tokens.

The two input files that will be tested by the scanner are called input.txt, and failtest.txt. The input.txt should theoretically pass in all of the mini-pascal's keywords and symbols and not return any illegal characters. The failtest.txt should then theoretically take in some legal characters, and then some purposefully placed illegal characters to try and catch them.

Parser:

The Parser part of the compiler will take input text and compare it to the grammar rules of the mini-pascal language and see if it is following its rules. It is created with a Java class under the Recognizer package and will have around 40 different methods mostly all linked to each other. These methods will check the structure of the grammar and will tell the user if there was an error in their pascal code. If an error was detected in the input code, it will return an error with a RuntimeException message, and what type of error it is. It also incorporates methods from the SyntaxTree to create the syntax tree.

Symbol Table:

The SymbolTable is a class that can add identifiers, and their types into it and store them. It can store their name, datatype, and kind (program, procedure, variable, function). The Recognizer class will utilize this class by creating a symbol table and adding the symbols into it when ID's are matched inside the Recognizer's functions. The symbol table class contains a toString that can be called to print out each symbol, and its kind.

Syntax Tree:

The Syntax Tree is a package containing classes that represent nodes in the syntax tree. These classes have methods that connect them to the other nodes. The parser will utilize these classes and methods and create the syntax tree. This tree will then be given to the code generation to convert it into MIPS assembly code.

Semantic Analysis:

A package that takes in a program node and goes through its syntax tree. While doing this is checks to see if the given variables have been declared. The Semantic Analyzer also labels expressions as real or integer data types. Uses these main functions to operate:

analysis – Takes the variables and their types and puts them into a hashmap. Also calls the verifyVariable function to check that they are declared.

assignTypes – Assigns either type real or integer to the expressions.

verifyVariables – Checks to see if the variables given are declared.

Code Generation:

This package contains one class with the required code, and one class to test that code. The CodeGeneration class is the one with the code, and its purpose is to create the MIPS assembly language. It does this by taking in the given syntax tree, and generating the MIPS based on that.

Change Log:

2/15/19 – Added the recognizer segment. Also adjusted the SDD structure to align with the entire compiler instead of just the scanner.

2/24/19 – Added the segment for the isolated Symbol Table class.

3/3/19 – Added onto the Recognizer section, and updated the Symbol Table section to fit with its integration with the Recognizer. Also changed Channel Log to Change Log.

3/10/19 – Added the Syntax Tree section.

4/7/2019 – added the Semantic Analysis section.

4/30/2019 – Updated overview, renamed Recognizer section to Parser, created Code Generation section.