Andrew Miller

EECE 5183

Compiler Theory Report

1. **Software Structure**
   My recursive descent compiler is written in C++ with the help of a few standard libraries. The source code for the compiler is all located in the "src/" directory in the repository above. It consists of several files representing different sections of the compiler. The main functionality for the scanner and parser is located in scanner.cpp and parser.cpp respectively. Additionally, the Symbol Table structure is defined by SymbolTable.cpp and its header. Word.cpp and its header contains code describing the structure for tokens that are produced by the scanner.

2. **Build Process**
   The "src/" directory contains a makefile which facilitates building of the project and includes a clean command to destroy the build directory. Calling make invokes the GNU C++ compiler, so it should play nice with a standard Linux box that has an up-to-date gcc version. It was written on Ubuntu Server 20.04, so for compatibility: *it works on my machine*.

3. **Features**
   - *Scanner:*
     The scanner must be initialized with the source file. The scanner is used by calling getNextToken() repeatedly until EOF. While scanning, it first skips whitespace and skips most characters when the comment flags are active. Then, a switch statement sorts the current character in the stream to find what sort of token it makes. Multi-character tokens are all taken from the stream in one pass of getNextToken(). Invalid characters in variable names and literals produces a show-stopping error.

   - *Parser:*
     The parser recursively calls a different function for each part of the grammar to form a parse tree. There are a host of helper functions like match() to catch grammatical errors and properly manage the token stream. There are also many functions that serve to produce specialized error messages. Left-Recursion-Elimination is handled by several helper functions with "Prime" in their name. There is also a stack of Words that the Parser uses to keep track of the current scope, so when new identifiers are declared they will be sent to the appropriate hash table in the Symbol Table.

   - *Symbol Table:*
     The Symbol Table object is comprised of a hash table (custom type symbol_book) of additional hash tables. Each nested hash table represents a scope, which the global scope is inserted during scanner initialization. The symbol_book uses Word objects (tokens) as the keys, and each scope's table then uses token strings as keys to lookup() Record objects.

4. **More Info**
   Explanations of the output text files and precise usage descriptions can be found in the README of the repository: https://github.com/mille5a9/compiler