Samprayash Dahal

Gonzalo Plascencia

Andrew Ta

CPSC 323

**PROJECT DOCUMENTATION**

**1. Problem Statement**

The third assignment is focused on expanding the syntax analyzer from Project 2 . The main goal is to create a program that should read a file containing the source code given from class in order to rewrite all the grammar into their respective production rules into an output file while using a Top-Down Approach for the Syntax Analyzer.

**2. How to use the program**

1. The program has only been tested to run on windows

2. We used the IDE Visual Studios (not xcode or others)

3. You will need to place a text file in the same directory as the executable file and enter the file name when you run the program (ex: test1.txt)

4. To run the program, just simply double click on the Project2.exe

5. The result will be displayed on the screen, and also be saved in the same directory as the input and executable with name “output.txt”

**3. Design of the program**

**Lexer Header File**

This header file starts with the lexer class which contains various functions which allow the lexemes to be separated into its respective tokens such as separators, operators, and keywords.

After categorizing one of the lexemes into its respective tokens, the state table will indicate that it’s time to move onto the next lexeme, thus continuing a loop until we finally reach the end.

The header file also contains a function called lexrecord, which allows a file name to be entered and to be translated it into a symbol table.

The variables are initialized in the private section of the header file.

**SyntaxAnalyzer.h**

This header file creates a symbol for all the tokens and lexemes.

The Statement function reads the information from the text file and generate a symbol table for each respective character. It calls several other functions to help with adding identifiers into the table. If there is any error within each construct of the functions, there will be an error message, stating that the identifier is expected, but was not found.

There are also functions that assigns the tokens and lexers to each memory address location, so the program can know where everything is on the symbol table.

The header file lastly uses the Print() function to a segment of assembly code as well as the memory locations.

**InstructionTable.h**

This header file just initializes variables for assembly code.

**main.cpp**

The program will start by asking the user for a filename to check if the file exists in the computer’s hardware. If it does, the file will open and the program will start its functionality and display the respective tokens and lexemes within the desired file; however, if it fails and the text file does not exist, an error message will be displayed, ending the program.

Then, the program will call SyntaxAnalyzer in order to print the memory address locations for the respective lexemes and tokens as well as an instruction table for assembly code.

DFSM for Identifiers:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | “ “ | D | “.” | $ | L | ! | O |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| 2 | 3 | 2 | 0 | 2 | 2 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

DFSM for integers(D):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | “ “ | D | “.” | $ | L | ! | O |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |

DFSM for Real:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | “ “(space) | D(igit) | “.” | $ | L(etter) | ! | O(perator) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 2 | 3 | 0 | 0 | 0 | 0 |
| 2 | 0 | 2 | 3 | 0 | 0 | 0 | 0 |
| 3 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |

**REs:**

Identifier: (L(L|D)\*) | (L(L|D)\*$)

Integer: D+

Real: D\* . D+

**4. Any Limitations**

There is a limitation to the program. The program will start by asking the user for the name of the text file. The text file must have correct documentation and format for the program to recognize it. Also, since we have done the program solely on WINDOWS 10, we are not sure if the program will run on different devices such as the Apple MAC or a LINUX operating system.

**5. Shortcomings**

NONE