.Lexer Documentation

Cody Thompson, Tracy Tonnu, Russel Koh

02/25/2020

1. Problem Statement

The purpose of the assignment was to create a program to run a lexical analysis on a sequence of characters inputted from either the console or a text file. The lexer is to determine if the input text is valid by breaking the sequence of characters into tokens, a series of strings with identified meaning, based on given delimiters. The tokenized input is then to be passed through a finite state machine, which determines if the input is valid based on a set state table, then pushes token and lexeme pairs into a queue to be printed to the screen.

2. How to use your Program

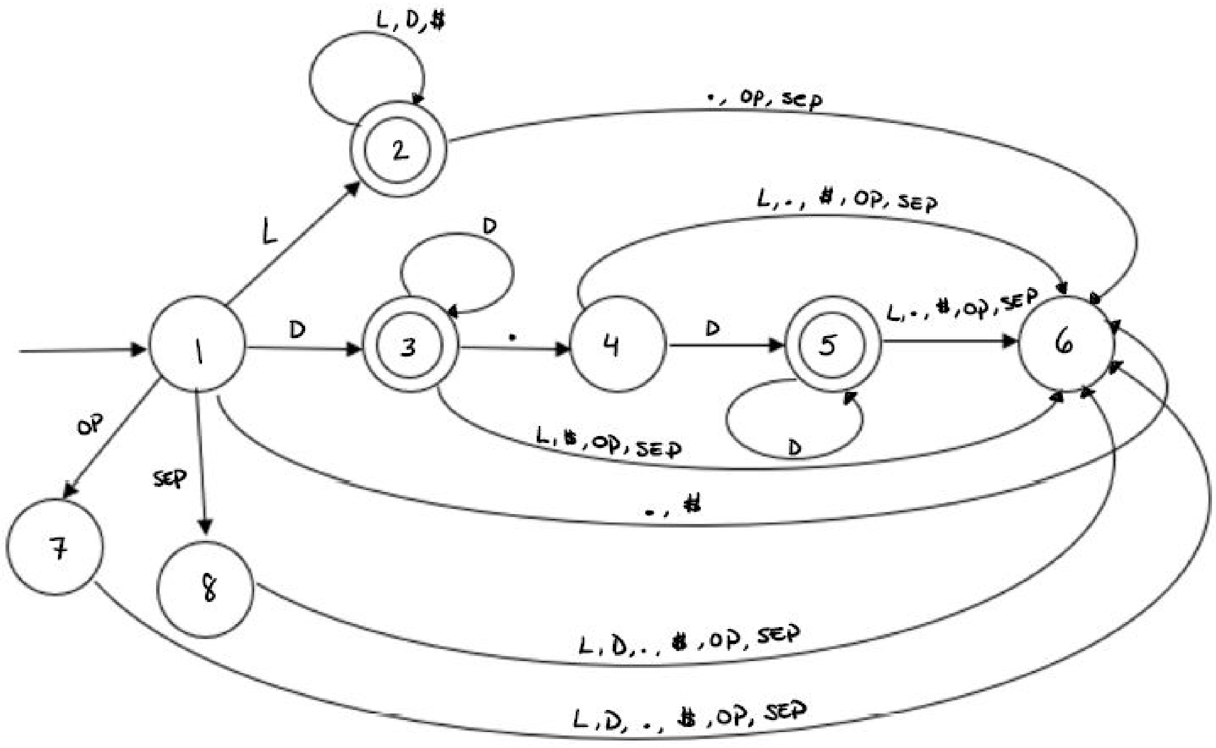
In order to access the lexical analyzer, a Lexer object is instantiated, which accepts a text file as a parameter. Once the program has verified that the text file has been successfully opened, each separate line of the file, indicated by the newline character (\n), is passed through as a parameter to the parseTokens function.

The parseTokens function removes the individual string of characters from the queue and pushes each token, determined by particular delimiters, into a separate tokens queue to be accessed by the FSM function.

Once the string of characters has been parsed into tokens, the FSM function is called within the constructor, which accepts no parameters. The FSM function has access to the tokens queue and steps through each character stored and loops through a state table. Depending on the current token and current state, the FSM will determine if the input will end in a valid current state or a dead state. If the input of characters is valid, it will push the token and character into a separate queue for output.

To print the tokenized characters, the output function is called and prints the tokens and lexemes to the screen.

3. **Design of your program**



Current State	D	L	.	$	op	sep
1	3	2	6	6	7	8
2	2	2	6	2	6	6
3	3	6	4	6	6	6
4	5	6	6	6	6	6
5	6	6	6	6	6	6
6	6	6	6	6	6	6
7	6	6	6	6	6	6
8	6	6	6	6	6	6

The lexical analyzer consists of a Lexer class with constructors and functions to perform the lexical analysis on an inputted text file. The class contains two constructors, five functions, and four private member attributes.

The regular expressions that are used in the program to indicate digits or letters include [0-9] for digits and [a-zA-Z] for letters. These provided a simple way to compare inputs being passed through the FSM function to determine the validity of the input and when parsing tokens.

CONSTRUCTORS for the Lexer Class:

Lexer()

The Lexer object is instantiated as an empty object. This allows for the creation of a lexer object with no passed parameters. Once called, it creates an empty string of text to be stored in *line*.

Lexer(std::string filename)

When called, the Lexer object is instantiated and a text file is passed through as a parameter. The constructor will attempt to open the text file, and if successful, the constructor will read in the string of characters, pushing individual lines (delimited by \n) as a parameter to the parseTokens function to be separated into tokens according to particular delimiters. Once the parseTokens function has completed executing, the constructor calls on the FSM function to determine if the input is valid or not.

The library facilities used include string, queue, and fstream.

FUNCTIONS for the Lexer class:

parseTokens(std::string input)

The logic of the function starts with a while loop in order to find the first instance of a delimiter. Inside the while loop, the starting statement determines if the position in the line is before the separator denoted by ‘;’. If this is the case, the program will get the lexeme found before the separator and push the lexeme and separator into the *tokens* queue. The following else-if statement determines if a comment symbol has been read. If a comment symbol has been found, the text following it is ignored until the closing comment symbol is found.

If neither conditions are met, the program enters a nested if statement and will look for the delimiter ‘.’ to determine if the input is a float by examining the characters following the delimiter. If the token is a number, then a float has been inputted. If a float is not found, the program will determine if the previous character is alphanumeric. If the previous character is alphanumeric, the program will go back by and push the lexeme into the *tokens* queue. If it is determined that the input is neither a float or alphanumeric, the program will push the single lexeme onto the *tokens* queue.

The library facilities used include string, queue, and regex.

FSM()

The FSM function identifies the token of all given lexemes from the *tokens* queue. While *tokens* is not empty, the function will step through each individual character in the string of characters pulled from *tokens* and traverse through a state table to determine if the text is a valid input (ending in a valid state) or not (ending in a dead state).

Once the tokens and lexemes have been identified, they are placed into a queue of pairs (*tokenLex*) using a switch statement. If the program finds that current state has ended indicating an identifier, the program will call the isKeyword function, passing the string of text through as a parameter. Once completed, the stored values can be accessed for output.

The library facilities used include queue, iostream, regex, and string.

output()

The output function prints the pairs of tokens and lexemes to the screen and formats the output for legibility. It reads in the pairs from the *tokenLex* queue and continues to print each pair until the *tokenLex* queue is empty.

The library facilities used include queue and iostream.

isKeyword(std::string ident)

The main function of isKeyword is to compare the strings being passed through to an array of legal keywords stored within the class. If the passed string matches any strings in the keyword array, the function will return True.

4. **Any Limitations**
   None

5. **Any Shortcomings**
   In order for the last identifier to be read in in a line of text, a delimiter (such as ‘;’) must be present at the end of the string.