Preston Phan

Luan Nguyen

Calvin Nguyen

CPSC 323 – Section 02

CS323 Project 1 Documentation

1. Problem Statement

Assignment 1 of CPSC 323 is identified as a lexical analyzer. The lexical analyzer is a finite state machine which takes in input through a text file and outputs as a text file that is split into two categories, tokens and lexemes. The goal of the assignment is to read in the text file and identify the different tokens and output the correct lexeme. The lexer takes in a token and returns a lexeme for the specified token.

2. How to use your program

The program can be run by simply running the python command followed by the filename. In the command prompt window for Windows 10, find the directory for the project folder. Then type in "**py wholething.py**" to compile the code and the output will be generated into the **output.txt** file. Other codes can be tested by replacing the texts in the **input.txt** file.

3. Design of your program

The design of the program started off by implementing the sample token list provided on Titanium. Once the tokens and lexemes are configured, we created helper functions in order to identify the tokens and assign if it is the correct lexeme or not. This can be represented as a Boolean state of true or false. Once the helper functions had been established, we used python lists and the split function to analyze each individual entry inside the text file. In order to do this we stored the length of the text file split to an assigned variable called fileinput. We created an index of 0 and used a while loop until the length of the text file has been reached through the use of incrementing an iterator, indicating there are no more entries inside the text file. We implemented the same logic to the writing function to an output file. We used “with open output.txt " to append to an output.txt file every time the program is called. Every time the program is run, old information is overwritten with any changes to the input.txt.

Then we defined the rules to our token machine. It takes in the input text and reads through each lexeme. We stored each lexeme’s list index and at the same time we used functions that would check the type of token. The functions call the helper functions which have defined the different tokens and categorizes the lexeme correctly. Each time this is done, we append the token to a tokenList using the append function of the list in the Python Standard Library. We return a list and put the list into the output function. The result is an output that categorizes each lexeme into a token.

4. Any Limitation

The input file needs to have proper white space for the program to identify the lexemes. For example, ‘**test!**’ will be identified as a single token instead of ‘**test**’ and ‘**!**’.

5. Any Shortcomings

Identifying comments and removing them is currently not working.

Table:

	Operators **O**	Separators **S**	Other Tokens **T**
1	2	2	1
2	1	1	

State Description:

Initial State 1: Read next entry

State 2: Analyze Token

Diagram: