CPSC 323-03
Project 1

Eric Tran
Victoria Tran
Justin Castillo

I. Problem

This assignment requires our team to create a lexical analyzer. A lexical analyzer essentially breaks down code into meaningful units. These meaningful units include keywords, identifiers, operators, separators and real or integer values.

The lexical analyzer will read the source code from a sample input file, SampleInputFile.txt and identify the various tokens and lexemes. Consequently, the program will create an output file that lists all of the identified tokens (i.e operators, separators, identifiers, and keywords).

II. How to Use our Program

As we built our program, we used two different operating systems. A majority of us used Windows, however, we occasionally switched to the Mac OS for some implementation. Overall, we have concluded that using the Windows operating system is best to execute the code.

Thus, to use our program, please use a Windows environment and complete the following steps:

1. Download all files necessary
2. Open the "CPSC 323 Project 1" folder.
3. Open the "Debug" folder.
4. Double-click the "CPSC 323 Project 1" Application.
5. Open the newly created "Lexical Analysis" text file to see the output of our code.

III. Program Design

We essentially chose a brute force method in which the program reads the input file, character by character. We created separate char arrays for keywords, separators, and operators to identify the specific lexemes in the input file. For every char read by the program, it was compared to the arrays until the lexeme was found. This specifically helped with single character lexemes such as separators and operators.

When it came to identifiers that contained multiple characters, such as words, we created a char[] to contain all characters until a stopping point was reached (such as a space, separator, or end line). For example, if the bracket was found, a flag would be set to notify the program to label the word before the bracket as an identifier and the bracket itself as a separator. The program would then continue with the next element being a completely new lexeme.
Furthermore, we had to check identifiers that had the character '$' attached to it. To account for the '$', we simply checked if the test character was an alphanumeric character or a '$'.

To check if the input was a digit or a number, we called specific functions such as isdigit() and isalnum(). We also had to account for comments, thus we checked if the input had the character '!' and if it did have the character, it would continue reading the line until it reached another "!".

IV. Limitations

Pertaining to the creation of the project, we were not faced with too many limitations. We had the necessary resources and an ample amount of time to work. However, we did experience a few difficulties that are not related to the program's functionalities.

One non-project related limitation we faced was a lack of lab time because we were unable to find time together as a team to go to a lab room to test it together. We fixed this limitation by going to a lab to test our code at separate times.

Another problem that is not related to the program, we faced during this assignment was trying to create a working product with different codes/ styles. We each had a different idea on how to approach the problem, making it difficult to

V. Shortcomings

We were unable to implement the FSM function to our code, although it was required by the assignment. Consequently, we could not implement a numeric table that represents the logic of the source code in the input file.