Johanna Nguyen

CPSC 323 – Lexical Analyzer

**1. Problem Statement**

The purpose of this assignment is to create a program acting as a lexical analyzer by using deterministic finite state machines.

**2. How to Use the Program**

User must have a text file (to be analyzed) saved on his/her machine.

User must input the full file name and location, using double slashes (\\).

Example: C:\\Desktop\\Users\\Username\\input.txt

The result will be printed to a text file entitled “output.txt”.

**3. Design of the Program**

The program starts with a helper function to convert a given character into a number. The function will return 0, 1, 2, 3, 4 to represent letters, digits, “$”, “.”, and all other symbols, respectively. The number will serve as the column number the input will be converted to once it makes use of the DFSM.

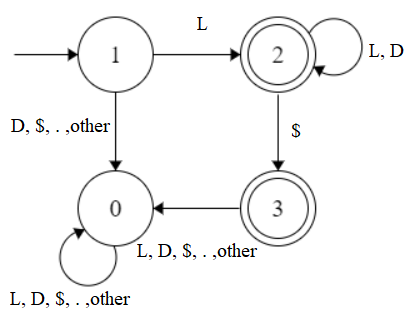
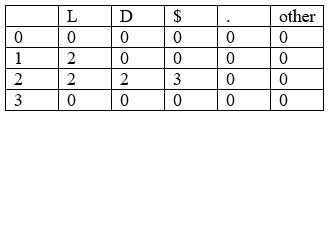
The functions to check for identifiers, integers, and floating-point numbers use a static 2- dimensional array to build a deterministic finite state machine table. It reads in a string and supplies the correct state based on the input. For identifiers, it must start with a letter. Following the letter can be 0 or more numbers and letters. The identifier may or may not be finished with a “$”. This function will return true if all these conditions are met and return false otherwise. For integers, the program will return true if all inputs are digits and will return false once it reads a character that is not a digit. For floating point numbers, the program recognizes that the string may only contain one “.”, the rest of the string must be digits. If these conditions are met, the function will return true and will return false otherwise.

The functions to check for separators, operators, and keywords use a static 1-dimensional array of strings or characters. Since there are a finite amount of these elements, each element of the array is a different separator, operator, or keyword. The functions will take in a string a procedurally compare the string to each element in the arrays. If the functions find a match it will return true, otherwise it will return false.

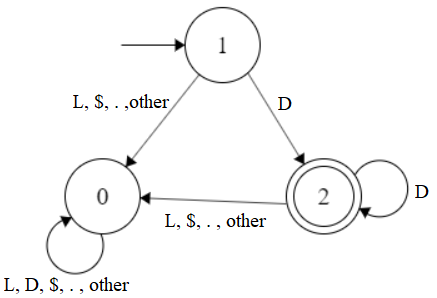
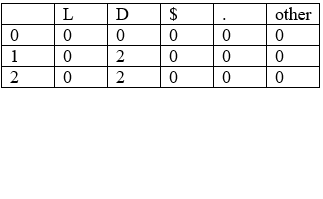
The function to check for comments character by character. It uses a Boolean value endComment as a flag. It starts by recognizing the first “!” it encounters. It then continuously checks the next character that is read until it finds the second “!”. Once this is achieved, endComment is switched to true.

The actual lexical analyzer function combines all the functions. It starts by reading in the first character and all leading whitespaces. It then starts to build a lexeme until a separator or operator is met. When it finds a “!”, it calls the function isComment() to determine if it has found the beginning of a comment. After checking for comments, it then calls the functions isSeparator() and isOperator(). If a separator or operator is not found, it realizes it has come to the end of a lexeme. From this stage, the analyzer then calls isIdentifier(), isInteger(), or isFloat() and determines which category it belongs in. The program returns a tuple of two strings – one being the category of token, the other is the lexeme. If the lexeme cannot be properly categorized, the program will return “UNKNOWN” and print the corresponding lexeme.

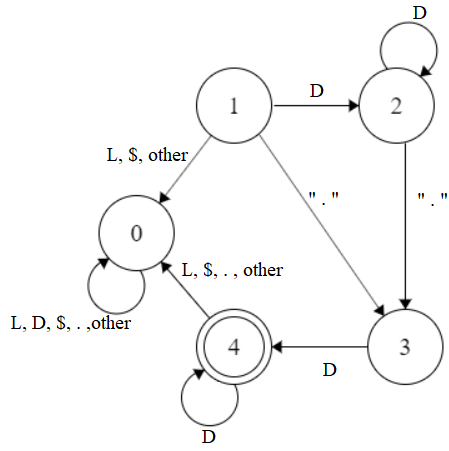
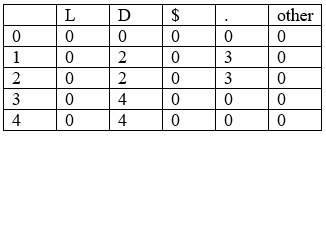
DFSM for Identifiers:

DFSM for Integers:

DFSM for Floating-Point

RE:

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

Integer: D+

Floating-point: D\* . D+

**4. Limitations**

The program has two limitations. Since the user input (file name and location) is saved as a string, the program is limited to the file location and name being less than max\_size();.

The other limitation is that it does not run on Linux-based machines.

**5. Shortcomings**

The program does not have any shortcomings.