CPSC323 Documentation

1. **Problem Statement**

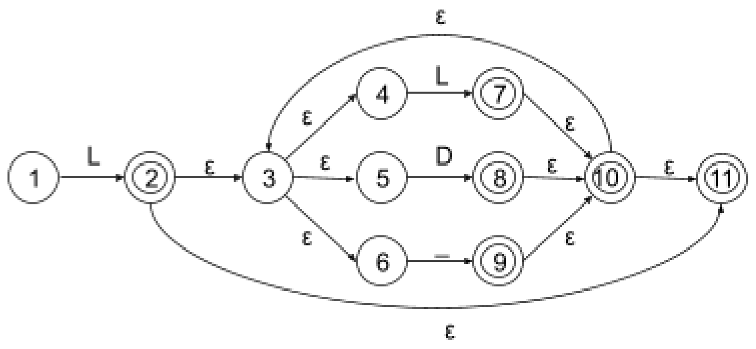
   The first assignment is to write a lexical analyzer (lexer) that parses and identifies tokens from a text file, and at least uses an FSM for the identifier, integer and real tokens. A major component of this assignment is to write a function – lexer (), that returns a record, one field for the token and another field the actual "value" of the token (lexeme), i.e. the instance of a token. The main program tests the lexer, or in other words, it reads a file containing the source code of Rat24S to generate tokens and write out both the tokens and lexemes to an output file.

2. **How to use your program**
   1. Open Windows Command Prompt.
   2. Navigate to the project directory.
   3. Type: code .
      a. This opens the project in VSCode
   4. Open a terminal window (powershell is fine)
   5. Type: ./main.exe
   6. You will be prompted to enter a source code text file.
      a. Please type the name of the text file in the project directory that holds the RAT24S code (or junk code) that you wish to use.
      b. Example: Type: test1.txt
   7. You will be prompted to enter another file name for the output of your LA.
      a. Please either *add an empty text file yourself* to the project directory for your own test files, *or use the empty file (jout.txt) provided for you*.
      b. Example: Type: output1.txt

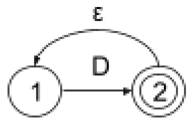
3. **Design of your program**

   The ***main function*** opens a user provided source code text file, and begins a while loop that skips comments, skips whitespace, or calls the lexer and adds the return value to a vector of pairs. I used a ***vector of pairs*** because we needed a data structure that could store a pair of values (lexeme, token) in the order added and resize itself if needed. Then a range loop is used to print out the vector to a user provided text file. The ***lexer function*** takes a character and a file pointer, and returns a pair to the main function to add to the vector. The lexer uses the ***string's find_first_of(char) function*** to check if the current character is in the ***string*** of letters, digits, separators, or operators defined above because it is quick. If it is a letter, then the ***ID_FSM function*** is called, and it takes the current character and the file pointer, and returns the lexeme string to the lexer. Then, that string is compared to an ***unordered set*** of keywords, since the hashed keywords can be compared on O(1) time with the ***unordered set's find function***, and if found, it is returned to main as lexeme, "keyword", else as lexeme, "identifier". If the character was a digit however, the ***Int_Real_DFSM function*** would be called, which takes the current character and the file pointer and returns a lexeme and either "integer" or "real" depending on the final state, as a pair that is returned to main and added to the vector. Both FSM functions use ***2D arrays***, ***idDFSM*** and ***realDFSM***, as their transition tables to determine state because values can be accessed on O(1) time.

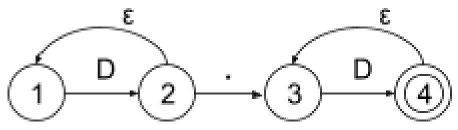
If L = {a|b|c|...|z}
D = {0|1|2|...|9}
Identifier: L(L|D|_)*



Integer: D+



Real number: D+.D+



4. **Any Limitation**
   None
5. **Any shortcomings**
   None