**CPSC 323 Documentation**

1. **Problem Statement**

Create a lexical analyzer that reads the source code of a program to break it down into a sequence of tokens or lexemes: identifiers, keywords, operators, literals, etc. The lexical analyzer should identify the patterns of characters that represent specific language constructs.

1. **How to Use the Program**

**Dependencies**

* OS: Mac OS || Linux

**Installing**

* Download repository [here](https://drive.google.com/drive/folders/1-hHyjDu8-U9jEtvkZXcql5gZyLFBQCNO?usp=share_link)
* Change main executable file permission
  + chmod +x main

**How to run the program**

* Open the command line of your OS
* Change into downloaded repository directory
* Execute the folliwng command:
  + ./main <input\_file>
* NOTE: the <input\_file> can ONLY be one of the following:
  + input\_01
  + input\_02
  + input\_03

1. **Design of your program**

To identify language construct patterns, the lexical analyzer uses a fininte state machine (FSM). The FSM's state transition function is represented as a two-dimensional array, where each column represents an input symbol, and each row as different states. It lists the current state of the machine, the input symbol being read, and the next state to which the machine transitions, given that input symbol. The column number of the state transition table is tracked by a hashmap, where the key is the input symbol and the value is the corresponding column number of the state transition table.  
  
The high level workflow is as follows: The FSM starts off at the starting state — state 0 and reads the input, character by character. Then it retrieves the input symbol’s column number on the state transition table using the hashmap. To get the next state, the FSM uses the state transition table — it goes to the row that corresponds to the current state, and column that corresponds to the current input symbol. If the state is an accepting state, it stores the current token and lexeme.

1. **Any Limitation**

The FSM currently identifies the following Tokens

* KEYWORDS
  + for, while, if, else, int, return
* IDENTIFIERS
* INTEGERS
* REAL
* OPERATORS
  + +,-,/,%
* SEPARATORS
  + (, ), {, }

1. **Any shortcomings**

We would like to expand the available keywords