**Group Members:**  
Audrey Hsu

Eduardo Nunez

Alex Bui

**CS323 Documentation**

**1. Problem Statement:**

The assignment is to create a lexical analyzer using a FSM for identifiers, integers, and real tokens. The main components of the program will consist of the function called lexer(), which will return a token when necessary. The lexer will read a file containing ‘Rat23F source code’ and write the results to an output file.

**2. How to use your program:**

Download python file named ‘lexer’

Type ‘python lexer’ in the the terminal window

Input Files are already pre-designed.

Output files will be displayed, separating tokens and lexers.

**3. Design of your program:**

We initialize two classes. A ‘**Token class**’ and a ‘**Lexer class**’. The program defines a **Token class**, which represents individual tokens in the source code. Each token has two attributes: token type, and lexeme.

The **Lexer class** is responsible for tokenizing the source code. It contains methods for identifying and categorizing tokens. It also has lists of keywords, operators, and separators.

We utilize different methods, such as **is\_alpha()**, **is\_digit(**), and **is\_alphanumeric()**: These methods are used to check if a character is an alphabetic character, a digit, or a combination of both.  
 The **lexer() function:** This is the main lexer function. It processes the input source code character by character, identifying and yielding tokens as it encounters them. It handles keywords, identifiers, integers, real numbers, operators, and separators. It also recognizes comments enclosed in [\* ... \*].  
 **The** **Main() function** reads source code from input files (e.g., 'input1.txt', 'input2.txt') and writes the resulting tokens and lexemes to output files (e.g., 'output1.txt', 'output2.txt').  
  
**Construct:**

FSM id (ℓ (ℓ | d)\* ℓ | ℓ)

FSM real d+, d+

FSM int d+

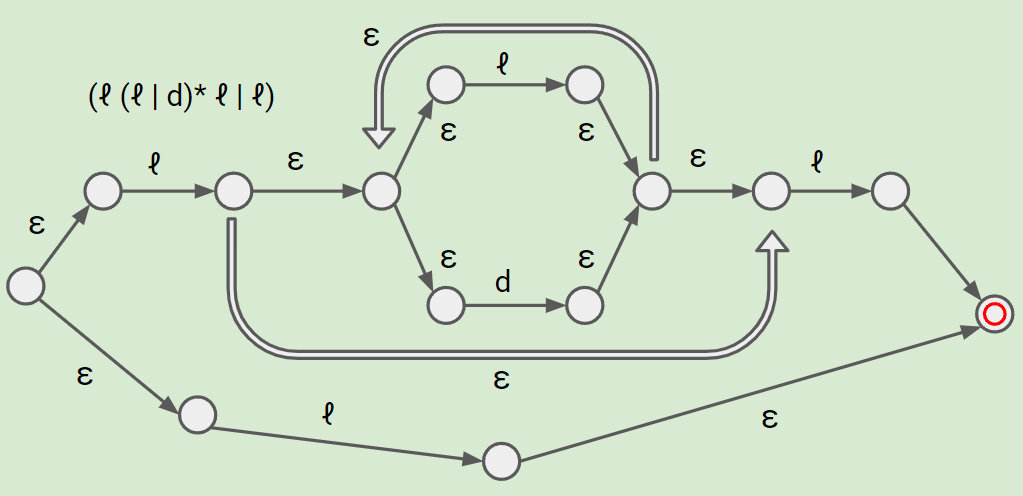
FSM keywords while (check identifiers against list of keywords)

FSM separators , : (check whether that character is part of separators list)

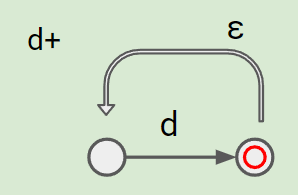
FSM operators + - \* divide (check character operators, < >)

**Machines:**

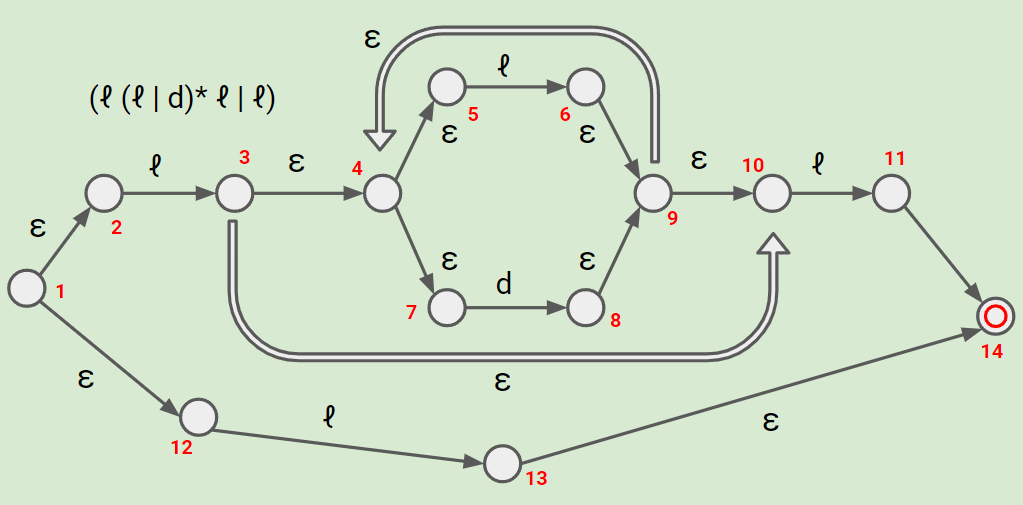
(ℓ (ℓ | d)\* ℓ | ℓ)

****

d+



**NFSM to DFSM Conversion for Identifier**

****

**Epsilon-Closures:**

(1) = {1, 2, 12}

(2) = {2}

(3) = {3, 4, 5, 7, 10}

(4) = {4, 5, 7}

(5) = {5}

(6) = {6, 9, 4, 5, 7, 10}

(7) = {7}

(8) = {8, 9, 4, 5, 7, 10}

(9) = {9, 4, 5, 7, 10}

(10) = {10, 4, 5, 7}

(11) = {11, 14}

(12) = {12}

(13) = {13, 14}

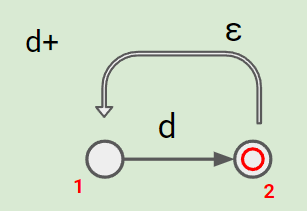
(14) = {14}

**DFSM Conversion**

(Bolded Red: Accepting States)

| q0 = [1] | letter | digit |
| --- | --- | --- |
| ~~[1]~~ **[1, 2, 12]** | ~~[3, 13]~~ [3, 4, 5, 7, 10, 13, 14] | [ ] |
| [3, 4, 5, 7, 10, 13, 14] | ~~[6, 11]~~ [6, 9, 4, 5, 7, 10, 11, 14] | ~~[8]~~ [8, 9, 4, 5, 7, 10] |
| [6, 9, 4, 5, 7, 10, 11, 14] | ~~[6, 11]~~ [6, 9, 4, 5, 7, 10, 11, 14] | ~~[8]~~ [8, 9, 4, 5, 7, 10] |
| **[8, 9, 4, 5, 7, 10]** | ~~[11]~~ [11, 14] | ~~[8]~~ [8, 9, 4, 5, 7, 10] |
| **[11, 14]** | [ ] | [ ] |

**NFSM to DFSM Conversion for int (same machine)**

****

**Epsilon-Closures:**

(1) = {1}

(2) = {2, 1}

**DFSM Conversion**

(Bolded Red: Accepting States)

| q0 = [1] | digit |
| --- | --- |
| ~~[1]~~ [1] | ~~[2]~~ [2, 1] |
| **[2, 1]** | [ ] |

**4. Any Limitation:**

The list of operators and separators is limited. Depending on the programming language you intend to support, you may need to expand these lists. Additionally, The code uses a simple list of keywords for detection. In a real-world lexer, keyword detection might be more complex. The code assumes that the input source code is read from files with specific filenames, which is not really flexible.

**5. Any shortcomings:**

None