CPSC 323
Project #1:
Group member: Zihao Qiu, Andrew Gomez, Evan Purpura
Oct. 7th

1. **Problem statement**: create a lexer using fsm that can parse program into Tokens by calling lexer()
2. **How to use**: To utilize our program you must:
    a. Download all of the files from the titanium submission
    b. Make sure you have the JDK and JRE downloaded so the Java Virtual Machine can compile java files on your machine
    c. Set up your system path variables to recognize the JDK path: resources
    d. The important files in this program are:
        i. src <
            1. compilers.jar
            2. **run_jar.bat**
            3. Test.txt
            4. Test2.txt
            5. Test3.txt
    e. You will locate the project folder on your computer directory
    f. Place your test file in the same directory as the downloaded compilers.jar file
    g. Go into the folder on the command prompt
    h. Then run the command: "**java -jar compilers.jar**"
    OR
    i. Double click the batch file **"run_jar.bat" if you are running on a windows machine**
    j. Follow prompt and enter in the name of the file to send to the program
    example: **test.txt**
    k. And the output on command line will print the Token and lexeme
    l. Your output will also be written to the file **output.txt**
    m. **Example run of commands shown below:**

```
C:\Users\Andrew's Laptop\IdeaProjects\compilers\out\artifacts\compilers_jar>java -jar compilers.jar
Please input the filename that you want to test in the directory
Example: test.txt
:test.txt
KEYWORD                 function
IDENTIFIER                      convert1x
SEPARATOR                       (
IDENTIFIER                      fahr
SEPARATOR                       :
KEYWORD                 int
SEPARATOR                       )
SEPARATOR                       {
KEYWORD                 return
INTEGER                 5
OPERATOR                        *
SEPARATOR                       (
IDENTIFIER                      fahr
OPERATOR                        -
REAL                    38.8978978909087
SEPARATOR                       )
OPERATOR                        /
INTEGER                 9
SEPARATOR                       ;
SEPARATOR                       }
KEYWORD                 $$
KEYWORD                 int
IDENTIFIER                      low
SEPARATOR                       ,
IDENTIFIER                      high
```

3. **Design:**
    a. Design is class structure
        i. Main.java
            1. Main class where the program is ran
            2. Utilizes Scanner to scan input file for next “word” or each input that has a space in between it.
            3. Removes comments if the comment symbol is seen in the lexeme
                a. Checks for end comment symbol in this lexeme and all following lexemes until one is found and the parsing can continue.
            4. parseSeparatorOperator(): Function as part of main class that utilizes an **ArrayList** to hold the lexemes that can be parsed from removing separators and operators that do not have a space.
        ii. Compiler_LA.java
            1. Main Lexical analyzer class that identifies a Token class of 7 types:
                a. Identifier
                b. Real
                c. Keyword
                d. Integer
                e. Operator
                f. Separator
                g. Invalid
            2. Has the main **lexer()** method:

a. Method first checks if any of the defined separator, operator and keyword options match the lexeme and if so returns Token created from lexeme
b. If not then the functions **isInteger()**, **isReal()** and **isIdentifier()** are called and return the new Token of specified type.

iii. FSMIdentifier.java
1. Has a two dimensional array of Integers that hold the state table for this given FSM
2. Constructor enters in the **ArrayList's** the valid states and acceptance states and first state
3. **isIdentifier():** Function that handles checking if the current state is valid and not null. If the state is valid grabs the next state from the current state and next input.
4. **nextState():** Takes in the current state and the next char input and checks what the table shows the next state to be depending on currentState and input char that is mapped to a column.
5. **char_to_col():** Function that gets the column number of the input char and returns it.

iv. FSMInteger.java
1. Constructor enters in the **ArrayList's** the valid states and acceptance states and first state.
2. **isIdentifier():** Function that handles checking if the current state is valid and not null. If the state is valid grabs the next state from the current state and next input.
3. **nextState():** Since we only have one state besides the initial state in this state table, you can simply represent it as one switch statement in the nextState() that with any other input besides a number goes to an invalid NULL state.

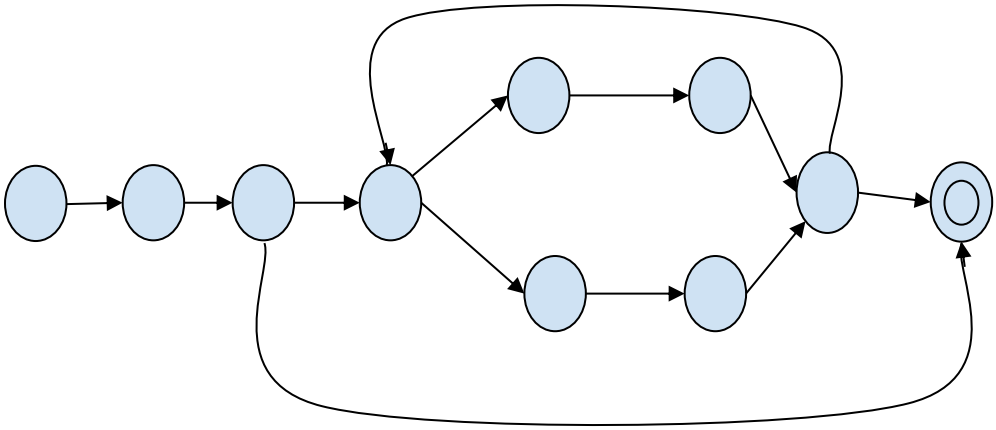
v. FSMReal.java
1. Same structure as FSMIdentifier, but has a different state table for Real numbers.

4. **Any limitations**: **None**
5. **Any shortcomings**: **None**

Design:

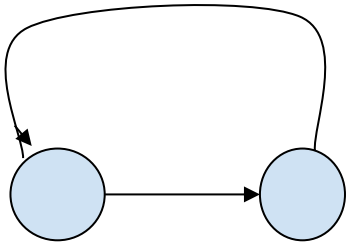
Lexer:
L = [A-Za-z]
D = [0-9]
RE for indentifier: L|L(L|D)*L

NFSM using Thompson for identifier:



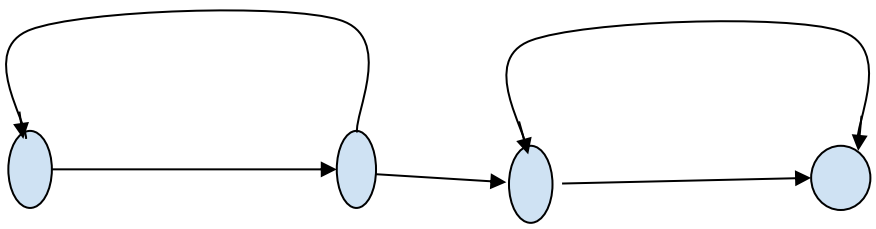
RE for integer: D+

NFSM using Thompson for integer:



RE for real: D+.D+

NFSM using Thompson for real:



Operator: ‘=’,’+’,etc.
Keyword: ‘while’, ‘if’, etc.
Separator: ‘(‘ ‘)’,etc.

Main:

```
while not finished (i.e. not end of the source file) do
        call the lexer for a token
        print the token and lexeme
endwhile
```