CPSC 323 - PROJECT 1

Programming Assignment 1

Course Number	CPSC 323 - 08
Deadline	11th November 2022

This project assignment you must Build a **"Lexer/Scanner".**
Lexical analysis is the process of reading in the stream of characters making up the source code of a program and dividing the input into tokens.
There are TWO options: group work (1-3) or individual work, to build your lexer/scanner. Please choose either way on your preference.

The Lexer

A major component of your assignment will be to write a procedure **(Function) – lexer (),** that returns a token when it is needed. Your **lexer()** should return a record, one field for the token and another field the actual "value" of the token (lexeme), i.e., the instance of a token.

Write a lexer from scratch by designing and implementing your FSA that returns the tokens from the simple source code in C++ in the given file **"input_scode.txt".**

1. In your build, at least you need design and implement FSA for tokens identifier and integer, the rest can be written ad-hoc. Otherwise, there will be a deduction of 3 points!
2. You need to write your REs and draw the FSA for the above required tokens: identifier and integer and put them in a document named as **"FSA _mydesign.doc".**
3. The implementation part of your design should be your executable program and it is expected to write in C/C++/Java. Use the function named as **lexer()** to read and return the next token.
4. Print out the result into two columns, one for tokens and another for lexemes, and save it into a document named as **"output"** (see an example I/O as below).
5. You must write a "readme" file to briefly specify how to setup/run your program if needed.
6. Your submission must have Five (5) files: the given "input_scode.txt", your design file, your program, output file, and a readme file.

Your main program should test the lexer i.e., your program should read a file containing the source code of input_scode.txt to generate tokens and write out the results to an output file. Make sure that you print both, the tokens, and lexemes.

Basically, your main program works as follows,

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

Example: Lexer splits the source text into a sequence of tokens, skipping blanks, newlines, and comments. See the I/O as below.
Note: you can customize your token classes and names, and then attach with the corresponding lexemes.

Input Source code (input_scode)

Source code:

```
while  (s < upper)   t = 33.00;
```

Output:

token	lexeme
keyword	while
separator	(
identifier	s
operator	<
identifier	upper
separator	)
identifier	t
operator	=
real	33.00
separator	;