CS 130 Theory of Computation   **2nd Semester, SY 2016-17**
Project 1

Lexical Analysis for Arithmetic Calculations

This is Part I of a two-part project. Write a lexical analyzer for a simple calculator language. The input will be a source program and the output will be tokens and their corresponding lexemes, and possibly some error messages (there aren't too many of these). Eventually, these tokens will be input to a parser (Part II of the project) which completes the translation process.

The goal of lexical analysis is to scan source code, filter out white spaces and comments, identify lexical errors, and most importantly, break up the code (which is a stream of characters) into *lexical tokens*, the most basic elements of a program. Given the following sample code, for instance:

```
// this program calculates the roots of a quadratic equation
10**2 + (4*5.5*(-3)) =
SAVE 1. # save the discriminant
(-10 + SQRT(RECALL(1)))/(2*5.5) =
PRINT.
(-10 - SQRT(RECALL(1)))/(2*5.5) =
PRINT.
# end of the program
```

The lexical analyzer should produce the following tokens and lexemes (note: the output is incomplete; more should follow—refer to moodle for complete output)

```
TOKEN LEXEME
-------------------------------------------------
NUMBER     10
EXP        **
NUMBER     2
PLUS       +
LPAREN     (
NUMBER     4
MULT       *
NUMBER     5.5
MULT       *
LPAREN     (
MINUS      -
NUMBER     3
RPAREN     )
RPAREN     )
EQUALS     =
IDENT      SAVE
NUMBER     1
PERIOD     .
LPAREN     (
MINUS      -
NUMBER     10
PLUS       +
```

The program should recognize the following tokens:

```
PLUS          +
MINUS         -
MULT          *
DIVIDE        /
MODULO        %
EXP           **
LPAREN        (
RPAREN        )
COMMA         ,
PERIOD        .
EQUALS        =
```

```
NUMBER        examples:  0 123 5.5  2.35 0.88888 1e20 2.2E-5
IDENT         <any sequence of letters>  examples:  SQRT  SAVE print
STRING        examples: 'hello'  "this is the answer"
EOF           end-of-file token
```

Your program should have a getToken() function or method which returns the next token from the file. Your driver code will be similar to the following segment:

```
Token t;
t = program.getToken();
while ( t.getId() != EOF )
{
      System.out.println( t.getId() + "\t" + t.getLexeme() );
      t = program.getToken();
}
```

There are only three lexical errors possible:

- badly formed number—occurs when text like this exist 5.=
- illegal character
- un-terminated string

You may write this program in any language you wish. This is an exercise in finite automata so the getToken method should simulate a finite automaton. You are not allowed to use lexical analysis tools or libraries (such as lex or flex). Visit the course website for any updates and clarifications. This program is due on March 3 (midnight); submit your source code via moodle, or as specified by your instructor.

Note: Do not worry about language syntax, for now. I may, in fact, change the syntax of the language when we specify project 2, so that it is more complex than the sample program provided.