Comp 333 Project #1 (Lexical Analyzer Project)

Due: September 22
Points: 30

This is an individual project.

1. In this project you will create a lexical analyzer in Java that identifies the following tokens in any string of characters.
   PLUS: +
   MINUS: -
   TIMES: *
   DIVIDE: /
   ASSIGNMENT: =
   LEFTPAREN: (
   RIGHTPAREN:)
   SEMICOLON: ;
   INTEGER: sequence of 1 or more digits
   FLOAT: sequence of digits with one decimal point. There must be at least one digit before and after the decimal point
   KEYWORD: read, write (lower-case letters)
   IDENTIFIER: lower-case letter followed by zero or more lower-case letters or digits

   Your lexical analyzer should find the tokens in an input string and print a token list that labels the tokens as to their type and value in the order found. The exact format of the input and output is listed below in item 3. If your program finds an unidentified character or any other error it should print a useful message (character and position number of the char) and **exit** the program.

   Notes:
   a. You **must** use the algorithm on the attached pseudocode.
   b. Do not assume that there are spaces separating any of the tokens
   c. To find an identifier read as many characters as possible.
      For example readme is not "read" + "me" it is "readme"
      For example x1=6 "x1" not "x" followed by an integer. It is "x1"
   d. To find a number read as many digits as possible.
      For example "2345" is the number 2345.
   e. The input string does not have to be a syntactically correct program. Your syntax analyzer should work on any string. It should never crash. If an error is detected it should gracefully exit with your won error message, printing only those tokens it has recognized up to the detected error.

2. Use object oriented programming (in Java) to develop your program. Create a class called LexAnalyzer. Create a main program that inputs the string to be analyzed, creates LexAnalyzer object and then calls the method prints the tokens as follows:

```
String s = ″ x =    12.78  ; y = apple   +  5 * orange;z=3*(4 * y)  ″ ;
System.out.println("Input String: " + s);
LexAnalyzer lex = new LexAnalyzer(s);
lex.printTokenList();
```

3. *A typical test case should look like this:*

```
public static void main (String[] args)
{
        String s=″  first  =  5.62 ; z1=8*( 34 + art2b)  read  ″;
        System.out.println(“Input String:  “ + s);
        LexAnalyzer lex = new LexAnalyzer(s);
        System.out.println( “Token List”);
        lex.printTokenList() ;
}
```

The output of the test case should look like this:

```
Input String:     first  =  5.62 ; z1=8*( 34 + art2b)  read

Token List
IDENTIFIER  first
ASSIGNMENT
FLOAT  5.62
SEMICOLON
IDENTIFIER  z1
ASSIGNMENT
INTEGER  8
TIMES
LEFTPAREN
INTEGER 34
PLUS
IDENTIFIER  art2b
RIGHTPAREN
KEYWORD  read
```

4. Test your program thoroughly. Test good and bad strings, not just the examples below. Your instructor will run your program on a set of test cases which will not be known to you. At least 50% of your grade will be based on how well your program runs on the instructor test cases.

5. **Submit your LexAnalyzer.java program to Moodle on Sept 22 by 7:30 am**. Your program must be named LexAnalyzer.java and meet the conditions of the project. See attached for pseudocode needed and an outline of the program.

6. **Hand in in class on Sept 22: (stapled together in this order)**
    a. Title Page with course , project number, date and your name
    b. LexAnalyzer.java source code
    c. Results for your program run on these test cases. Label test cases. Hand in the results of these test cases.

```
//Test Case 1    ″x =     12.78   ;  y = apple    +   5 * orange;z=3*(4 * y)   ‘‘
//Test Case 2 –  ″read write 34  +5.678  -789.001  goodread+-*  ″
//Test Case 3 – ″ int y = 7 ; double z = 78.01  ″
//Test Case 4 - ″ qwerty :=  1234″
//Test Case 5 -  ″ total is 23. dollars ″
//Test Case 6 -    ″x = art2.1b″
//Test Case 7:    ″       ″     //This is not an error. You should print EMPTY TOKEN LIST
```

Pseudocode for printing the next token in a string. Note that you need to manipulate the curr_char variable with care.

```
print_next_token()
{
        skip over white space
        if curr_char is  in  the set  {  ( ,  ) ,  +,  -,  * ,  /,    ;   , = }
                print appropriate token type
                return true
        else if curr_char is a digit
                read additional digits and at most one decimal point
                 print  number token type with found number
        else if curr_char is a letter
                read any additional letters and digits
                check to see if resulting string is read or write
                if so, print keyword token type with the keyword
                else print IDENTIFIER with the name of identifier
        else
                print error message
}
```

Pseudocode to print all tokens in string.

```
Set curr_char to first character in string
while  (not at end of string )
{
        print_next_token();

}
```

Outline of the public methods of LexAnalyzer.

Your program should be modular and easy to read. Add private fields and methods as needed.

```
public class LexAnalyzer
{

      // define your private fields as needed

     public LexAnalyzer ( String s)
     {
            //fill in code

      }

     public void printTokenList()
    {

         //fill in code
     }

    //define other private methods as needed
}
```