Assign-Lexical Analyzer-Java-DESCR-Fall23

Version: Refer to the version in the file name

Contents

1 Description

The Java program "**Lex.java**" is equivalent to the front.c program, which is given in your textbook, but with the following differences:

i. Lex.java displays the tokens in a textual form rather than numeric form. For example, instead of displaying the following

```
Next token is: 25      Next lexeme is (
```

it displays

```
Next token is: LEFT_PAREN   Next lexeme is (
```

ii. Additionally, it recognizes the lexemes and respective tokens shown in the next section. The respective grammar, which we will use for the parser is given in the appendix.

On this assignment, you are to modify the program as described below.

Declare lexeme as a String variable rather than an array of characters. In this case, you need not to constrain the length of lexeme to 100 anymore.

Make appropriate changes in the program to build and use the lexemes, appropriately. For example, lexeme need not be built character by character. Rather, use appropriate Java library and user-defined methods and regular expressions to identify individual lexemes. In this case, getChar, for example, may not be needed.

Please spend time on your algorithm rather than on your coding, first. Your algorithm is to address the following Lex objectives:

```
1. Open the program file
2. While there are program lines
    a. Input a line of program
    b. Tokenize the input line
    c. While there are tokens
        i. Display and return a token
    d. End while
3. End while
4. Close the program file
```

Note: Make sure to run and test the original Lex.Java program, before you modify it. The results of the original program and the updated program must be exactly the same.

2 Tokens

Token	Lexeme
ADD_OP	+
ASSIGN_OP	=
DIV_OP	/
END_KEYWORD	END
END_OF_FILE	EOF
IDENT	A valid identifier such as Radius
IF_KEYWORD	If
INT_LIT	A valid integer literal such as 253
LEFT_PAREN	(
MULT_OP	*
PRINT_KEYWORD	print
PROGRAM_KEYWORD	PROGRAM
READ_KEYWORD	Read
RIGHT_PAREN	)
SEMICOLON	;
SUB_OP	-
THEN_KEYWORD	then

3 Structure of Your Lexical Analyzer Program

Your program must be well formatted, commented and readable; and generate the results similar to the one shown in the following example. Include **your name and class information** both at the **top of your program listing, as comments,** and as shown in the example below, **in your output**.

4 Example Input and Output

Given the following program stored in an input file:

```
PROGRAM
     sumTotal = (sum + 47       ) /;
     Area=PI *         Radius * Radius;
END
```

Your lexical analyzer must produce the following output. Please note the exact format of the output.

```
Daniel Barker, CSCI4200, Spring 2023, Lexical Analyzer
********************************************************************
************
PROGRAM
Next token is: PROGRAM_KEYWORD    Next lexeme is PROGRAM

sumTotal = (sum + 47       ) /;
Next token is: IDENT              Next lexeme is sumTotal
Next token is: ASSIGN_OP          Next lexeme is =
Next token is: LEFT_PAREN         Next lexeme is (
Next token is: IDENT              Next lexeme is sum
Next token is: ADD_OP             Next lexeme is +
Next token is: INT_LIT            Next lexeme is 47
Next token is: RIGHT_PAREN        Next lexeme is )
Next token is: DIV_OP             Next lexeme is /
Next token is: SEMICOLON          Next lexeme is ;

Area=PI *        Radius * Radius;
Next token is: IDENT              Next lexeme is Area
Next token is: ASSIGN_OP          Next lexeme is =
Next token is: IDENT              Next lexeme is PI
Next token is: MULT_OP            Next lexeme is *
Next token is: IDENT              Next lexeme is Radius
Next token is: MULT_OP            Next lexeme is *
Next token is: IDENT              Next lexeme is Radius
Next token is: SEMICOLON          Next lexeme is ;

END
Next token is: END_KEYWORD        Next lexeme is END

Next token is: END_OF_FILE        Next lexeme is EOF
Lexical analysis of the program is complete!
```

5 A Test Program to Tokenize

Run your lexical analyzer on the following program, keeping the spacing and format of the input intact. I will test your program using the following and other programs.

```
PROGRAM
     read(count);
     total = 25
     if (count) then average = total/count;
           Area = (length+width) * 2
     C     =          age  - 5 * (D / C);
     print(C+Area);
END
```

6 How to Structure and Organize Your Program

a. In your IDE (e.g., Eclipse) create a Java project called **CSCI4200Fa23**.

b. Store your program, **Lex**, in the **default** package of **src** within **CSCI4200Fa23**.

c. Store all the lines to be tokenized in a single input file called **lexInput.txt**.

d. Store the lexInput.txt file in the same project, **CSCI4200Fa23**. This will allow you to open the input file as “lexInput.txt” without specifying a path.

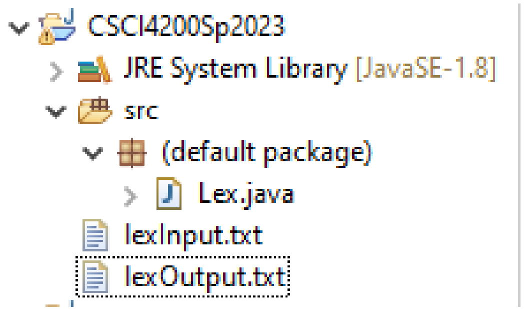
e. Store your output in a text file called **lexOutput.txt**, which must be created in the same project. You must create and open the output file as “lexOutput.txt” without specifying a path.

f. On the output, display your name, class information and a program title, followed by a line of 80 asterisks (*).

g. On the output, display each input line before you list its corresponding tokens and lexemes, as shown in the above example.

h. The END_OF_FILE token and its corresponding lexeme, EOF, must be printed after all the input lines are processed.

i. Following is an example of how your program should be organized:



```
CSCI4200Sp2023
  JRE System Library [JavaSE-1.8]
  src
    (default package)
      Lex.java
  lexInput.txt
  lexOutput.txt
```

7 What to turn in:

i. Lex.java

ii. lexInput.txt

iii. lexOutput.txt

8 Grading Rubrics

Category	Description	Points
Program Readability	Sufficient and readable comments (don't overdo it!) Readable program – descriptive identifier names; use of constant identifiers; appropriate program structures such as loops, decisions, etc.; appropriate indentations and spacing, etc. All the project-, class-, input file- and output file names must be exactly as specified.	15
Program logic and output	Use of correct straightforward (simple) and understandable logic Correct and complete results Correct output format	80
Input file	Correct contents Correct format (as specified by the requirements)	5

9 Appendix – Grammar

1) <program> → PROGRAM <statement>{;<statement>} END
2) <statement> → <output> | <assign> | <input> | <selection>
3) <output> → print (<expr>)
4) <assign> → IDENT = <expr>
5) <expr> → <term> {(+ | -) <term>}
6) <term> → <factor> {(* | /) <factor>}
7) <factor> → IDENT | INT_LIT | (<expr>)
8) <input> → read (IDENT)
9) <selection> → if (<expr>) then <statement>