Amarilda Celhaka

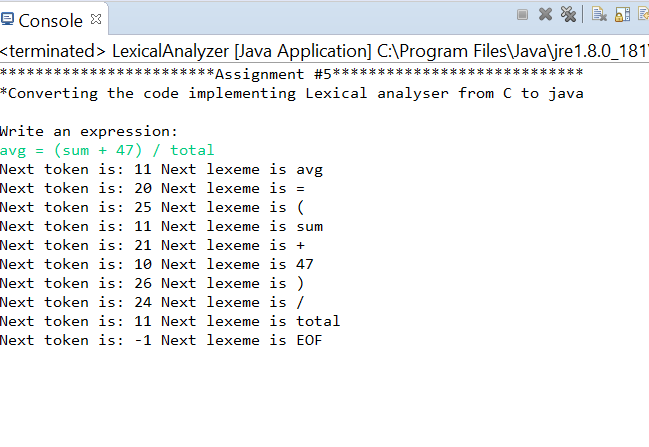
CMP SCI 4250

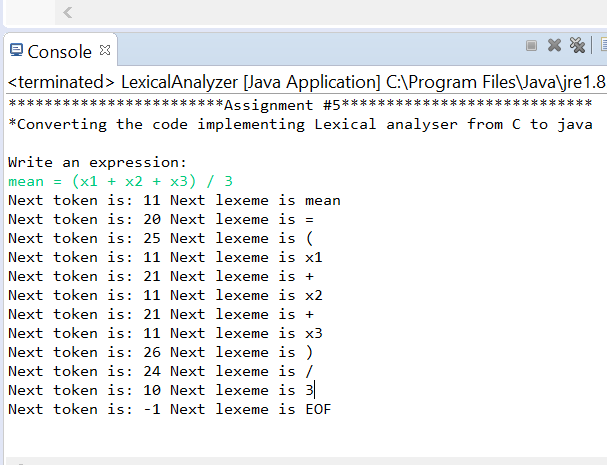
Assignment #5

Dr. P

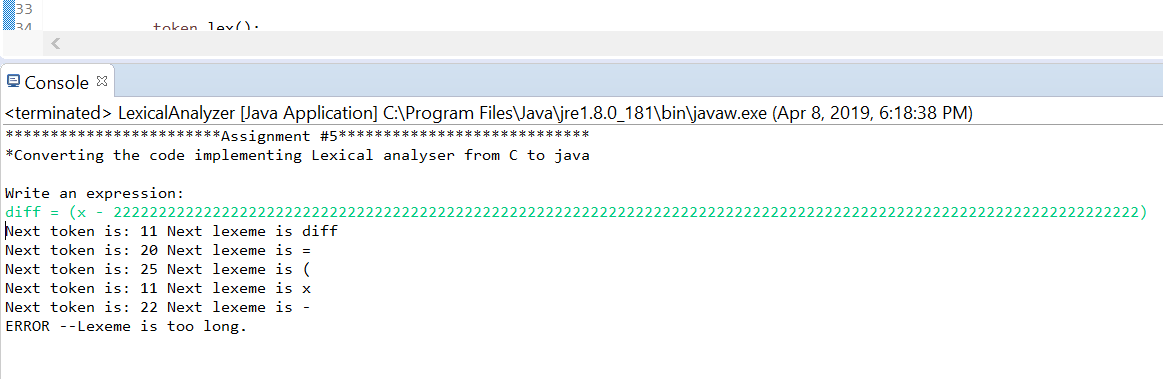
4/8/2019

**Screenshot of testing the code:**





To test the validation of a too long lexeme:



**LexicalAnalyzer.java file**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Amarilda Celhaka

\* CMP SCI 4250

\* Dr. P

\* Assignment #5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**import** java.util.Scanner;

**public** **class** LexicalAnalyzer {

**public** **static** **void** main(String[] args) {

Scanner input = **new** Scanner(System.***in***);

String expression;

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Assignment #5\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("\*Converting the code implementing Lexical analyser from C to java\n");

System.***out***.println("Write an expression: ");

expression = input.nextLine();

Token token = **new** Token(expression);

**while**(Token.*nextToken* != token.EOF) {

token.lex();

/\*if nextToken is set to 0, it indicates that the literal/ int literal was too long\*/

**if**(Token.*lexLen* == -1)

{ System.***out***.println("ERROR --Lexeme is too long.");

**return**;

}

input.close();

}

}

**Token.java file**

**public** **class** Token {

/\* Character classes, defined as constants \*/

**public** **final** **int** LETTER = 0;

**final** **int** DIGIT = 1;

**final** **int** UNKNOWN = 99;

**final** **int** SPACE = 3;

**public** **final** **int** EOF = -1;

/\* Token codes \*\*/

**final** **int** INT\_LIT = 10;

**final** **int** IDENT = 11;

**final** **int** ASSIGN\_OP = 20;

**final** **int** ADD\_OP = 21;

**final** **int** SUB\_OP = 22;

**final** **int** MULT\_OP = 23;

**final** **int** DIV\_OP = 24;

**final** **int** LEFT\_PAREN = 25;

**final** **int** RIGHT\_PAREN = 26;

**final** **int** OTHER = -2;

/\* Global declarations \*/

**static** **int** *charClass*;

**static** **char** *lex*[] = **new** **char**[1000];

**static** **char** *nextChar*;

**static** **int** *lexLen* = 0;

**static** **int** *token*;

**static** **int** *nextToken*;

**public** String expression;

String lexeme = "";

**int** j = 0;

**public** Token(String expression) {

**this**.expression = expression;

**for** (**int** i = 0; i < expression.length(); i++) {

*lex*[i] = expression.charAt(i);

}

*lex*[expression.length() + 1] = '\0';

}

**void** getChar() {

*nextChar* = *lex*[j++];

getNonBlank();

//System.out.println("next Char " + nextChar);

**if** (*nextChar* == '\0') {

*charClass* = EOF;

//System.out.println("EOF");

}

**else** **if** (Character.*isDigit*(*nextChar*)) {

*charClass* = DIGIT;

//System.out.println("DIGIT");

}

**else** **if** (Character.*isLetter*(*nextChar*)) {

*charClass* = LETTER;

//System.out.println("LETTER");

}

**else** {

*charClass* = UNKNOWN;

//System.out.println("UNKNOWN");

}

}

/\*function to skip whitespaces\*/

**public** **void** getNonBlank() {

**while** (Character.*isWhitespace*(*nextChar*)) {

*nextChar* = *lex*[j++];

}

}

**void** addChar() {

*lexLen*++;

**if**(*lexLen* > 98) {

*lexLen* = -1;

}

}

**public** **int** lex() {

*lexLen* = 0;

lexeme = "";

getChar();

lexeme += *nextChar*;

**switch** (*charClass*) {

/\* parse identifiers \*/

**case** LETTER:

**while** (*charClass* == LETTER || *charClass* == DIGIT ) {

getChar();

addChar();

**if** ( *charClass* == LETTER || *charClass* == DIGIT ) {

lexeme += *nextChar*;

}

**if**(*lexLen* == -1)

{

**return** 0;

}

}

j--;

*nextToken* = IDENT;

**break**;

/\* parse integer literals \*/

**case** DIGIT:

**while** (*charClass* == DIGIT) {

getChar();

addChar();

**if** (*charClass* == DIGIT) {

lexeme += *nextChar*;

}

**if**(*lexLen* == -1)

{

**return** 0;

}

}

j--;

*nextToken* = INT\_LIT;

**break**;

**case** EOF:

*nextToken* = EOF;

lexeme = "";

lexeme = "EOF";

addChar();

**break**;

**case** UNKNOWN:

lookup(*nextChar*);

addChar();

**break**;

}

System.***out***.println("Next token is: " + *nextToken* + " Next lexeme is " + lexeme);

**return** *nextToken*;

} /\* End of switch case \*/

**int** lookup(**char** ch) {

**switch** (ch) {

**case** '(':

*nextToken* = LEFT\_PAREN;

**break**;

**case** ')':

*nextToken* = RIGHT\_PAREN;

**break**;

**case** '+':

*nextToken* = ADD\_OP;

**break**;

**case** '-':

*nextToken* = SUB\_OP;

**break**;

**case** '\*':

*nextToken* = MULT\_OP;

**break**;

**case** '/':

*nextToken* = DIV\_OP;

**break**;

**case** '=':

*nextToken* = ASSIGN\_OP;

**break**;

**default**:

*nextToken* = OTHER;

**break**;

}

**return** *nextToken*;

}

}