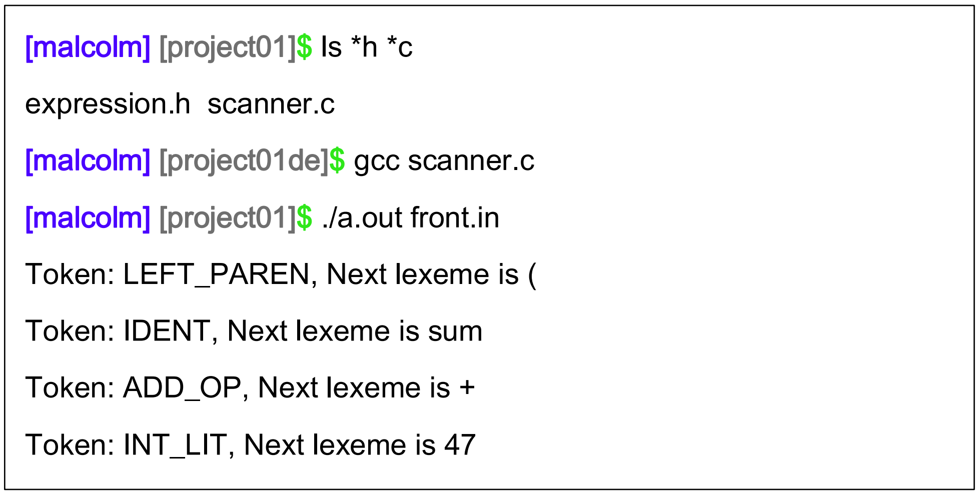
```
csc 321     project01
```

Convert the modified version of lexical analyzer given in Section 4.2 of our textbook, from Imperative design, written in C, to an Object-oriented design, written in Java.

I have provided you with the slightly modified version of the **scanner.c** (and expression.h) from the textbook, and the beginnings of a Scanner class (Scanner.java) and a Token class (Token.java), as a starting point. You are to finish the conversion. i.e. Use the given Token objects and, finish the Scanner class.   The Scanner class requires some major modification to product the correct output. You are not to change Token without talking to the instructor first.

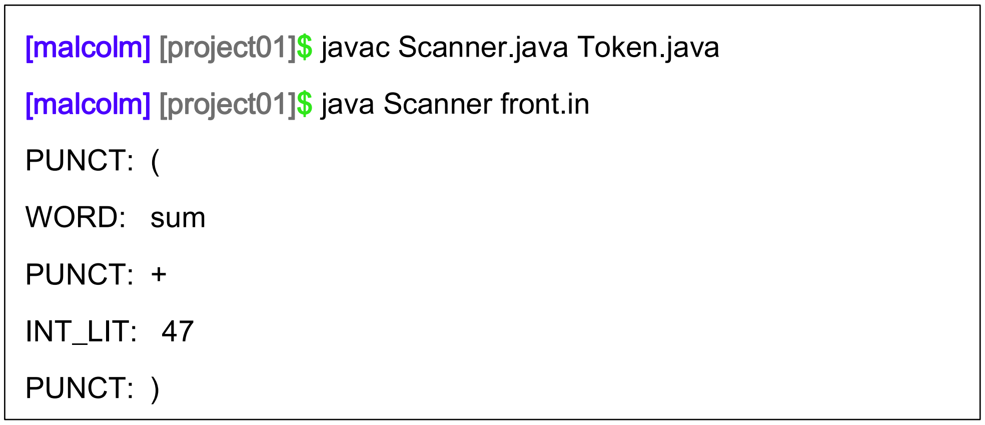
For this project the output should be similar to that of the C version of lexical analyzer (scanner.c) and exactly match the sample out below.

To compile and run the C version you will need a C compiler. I recommend gcc (see syllabus)



```
[malcolm] [project01]$ ls *h *c
expression.h  scanner.c
[malcolm] [project01de]$ gcc scanner.c
[malcolm] [project01]$ ./a.out front.in
Token: LEFT_PAREN, Next lexeme is (
Token: IDENT, Next lexeme is sum
Token: ADD_OP, Next lexeme is +
Token: INT_LIT, Next lexeme is 47
```

As given the java code (Scanner.java) does not produce the correct output. Its output will look like the following:



```
[malcolm] [project01]$ javac Scanner.java Token.java
[malcolm] [project01]$ java Scanner front.in
PUNCT:  (
WORD:   sum
PUNCT:  +
INT_LIT:   47
PUNCT:  )
```

Modify the Scanner.java so you get the following output:

```
[malcolm] [solution]$ javac Scanner.java  & java  Scanner front.in


 Token                    Lexeme      Value

----------------------------------------

 LEFT_PAREN           (          (0)

 IDENT               sum       (0)

 ADD_OP                +        (0)

 INT_LIT                     (47)

 RIGHT_PAREN          )        (0)
```

Note, you are NOT to rewrite the Token class or build this project from scratch. You are to use the code given (no exceptions). If you turn in a project that is not based on the given code it will not be accepted. The one exception to this is for bug fixes and added functionality. You may NOT make any changes the existing interface (break other existing code) but you may add improvements or fix existing issues in the implementation. If you wish to improve the code (extra credit) email the instructor to discus and make arrangements. Naming, Do not use the course file naming convention (as specified in the Syllabus) for these two files/classes. With Java it is easier to keep class names (filenames) simple and unified

Note, there is an updated Token.java class you might want to user (minor changes – updated 28/9/2020)

And here is what my main method from the Scanner class looks like:

```
public class Scanner {

  Token token;              // current token

  int next;                 // keep current ascii value of lexeeme 1st char in variable "next":

  boolean verbose = false;     //true;

  StreamTokenizer tokenstream;



 public static void main(String argv[]) throws IOException {
        Scanner scanner = new Scanner(argv);
        Token tok;

        System.out.println(" Token\t\t\tLexeme\t   Value");
        System.out.println("-----------------------------------------");
        while( (tok = scanner.lex()).type  != Token.EOF )
           System.out.println(" " + tok);

    }
```

To submit you programming project assignment (we will be trying something new – Gradescope)
0) You will be submitting JUST your Scanner.java   Do NOT submit anything else. It must be named Sacanner.java, that is "Scanner" with the ".java" fine ending. It must be plain ascii text (jpg, pdf, doc, rtf, ... are unacceptable). If you wish to submit a bug fix or major improvement to Token contact the instructor via email.

1) LOG IN: to gradescope.com on a computer (or device) where you can access your Scanner.java. Log in with your campus email address.
2) On your Gradescope Dashboard, select the correct course and the assignment (321 project 01 - Scanner).
3) After you select this programming assignment, a dialog box will appear. Then select your file or drag and drop the code file to the dialog box
4) Once you've chosen the correct file, select the **Upload** button. When your upload is successful, you'll see a confirmation message on your screen and you'll receive an email.

Future...
You may notice the new version of Token recognize a more substantial list of tokens. E.g. the following list of operators and reserved words:

```
     , . ( ) { } + - */ % = == != < <= > >= main procedure if  while
read  write funcall int
```

These are some of the operator, keyboards, and punctuation of our Toy languages that we may use we are going to use in Project 02

If needed some help

  ...

```
public class Scanner {
   Token token;                  // current token
   int next;                     // keep current ascii value of lexeeme 1st char in variable "next":
   boolean verbose = false;      //true;
   StreamTokenizer tokenstream;

 . . .

  public Token lex() throws IOException  {
     token = new Token();
     if ((next = tokenstream.nextToken()) == tokenstream.TT_EOF) {
         token.setType(Token.EOF);
         return token;
     }
     else if (next == tokenstream.TT_NUMBER){
         token.setValue( (int) tokenstream.nval);   // nval is a double
         token.setType(Token.INT_LIT);
         print(" "+token);
         return token;
     }
     else if (next == tokenstream.TT_WORD) {
        token.setLexeme(tokenstream.sval);
        switch(tokenstream.sval.toUpperCase()) {
           case "INT":
              token.setType(Token.INT);
              break;

               . . .

           case "IF":
              token.setType(Token.IF);
              break;

              . . .

           case "PROCEDURE":
              token.setType(Token.PROCEDURE);
              break;
           default:
              token.setType(Token.IDENT);
        }
```

```
            print(" "+token);
            return token;
         } // switch
         else {
            token.setLexeme(""+(char)next);
            switch (next) {
               case '(':
                  token.setType(Token.LEFT_PAREN);
                  break;

                 . . .

               case '}':
                  token.setType(Token.RIGHT_BRACKET);
                  break;
               case '=':
                   // okay so we have '=' it could be ASSIGN or if next token is '=' it would be EQUAL
                  // so lets look at next token
                  if(((next = tokenstream.nextToken()) == tokenstream.TT_EOF))
                    break;                           // (if next is EOF - we done here)
                 if(next == '=') {                     // if we  have a =  then toen is <=
                    token.setType(Token.EQUAL);
                    break;
                 }
                 tokenstream.pushBack();               // put token back into stream
                 token.setType(Token.ASSIGN_OP);
                 break;

                . . .

              default:
                token.setType(Token.UNKNOWN);
                break;
            }//switch
            return token;
         } // else
      } // lex
   } // Scanner
```