Homework 1 Solution
CSc 4330/6330 Spring 2018

This solution evaluates the expressions as they are parsed. Much more compact code because it does not construct the expression tree.

LISP.g4

```
grammar LISP;

// Grammar rules
lispExpr returns [double value]:
  v=numExpr SEMI {$value = $v.value;};

numExpr returns [double value]:
    num=NUMBER {$value = Double.parseDouble($num.text);}
  | LPAREN p=OP v1=numExpr v2=numExpr RPAREN
  {
    double answer = 0.0;
    if ($p.text.equals("+"))
      answer = $v1.value+$v2.value;
    else if ($p.text.equals("-"))
      answer = $v1.value-$v2.value;
    else if ($p.text.equals("*"))
      answer = $v1.value*$v2.value;
    else
      answer = $v1.value/$v2.value;
    $value = answer;
  }
  | LPAREN CAR v=numListExpr RPAREN {$value = $v.value.getValue();} ;

numListExpr returns [LISPNode value]:
    LPAREN s=numSequence RPAREN {$value = $s.value;}
  | LPAREN CDR v=numListExpr RPAREN {$value = $v.value.getNext();};
```

```
numSequence returns [LISPNode value]:
   v=numExpr
  {
   LISPNode n = new LISPNode();
   n.setNext(null);
   n.setValue($v.value);
   $value = n;
  }
  | v=numExpr s=numSequence
  {
   LISPNode n = new LISPNode();
   n.setNext($s.value);
   n.setValue($v.value);
   $value = n;
  }
;

// Lexical Rules
fragment C : ('C'|'c') ;
fragment A : ('A'|'a') ;
fragment D : ('D'|'d') ;
fragment R : ('R'|'r') ;
OP : '+' | '-' | '*' | '/';
NUMBER : ('0'..'9')+ | ('0'..'9')* '.' ('0'..'9')+;
LPAREN : '(';
RPAREN : ')';
SEMI : ';';
CAR : C A R;
CDR : C D R;
WS : [ \r\n\t]+ -> skip;
```

LISPNode.java

```
public class LISPNode {
  double value;
  LISPNode next;

  void setValue(double val) {
    value = val;
  }

  void setNext(LISPNode n) {
    next = n;
  }

  double getValue() {
    return value;
  }

  LISPNode getNext() {
    return next;
  }

}
```

LISP.java

```
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.BailErrorStrategy;
import org.antlr.v4.runtime.misc.ParseCancellationException;
import java.io.*;

public class LISP {

 static public void main(String argv[]) {
  System.out.print("LISP> ");
  do {
   String input = readInput().trim();
   if (input.equals("exit"))
    break;
   else input += ";";
   try {
    CharStream in = CharStreams.fromString(input);
    LISPLexer lexer = new LISPLexer(in);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    LISPParser parser = new LISPParser(tokens);
    parser.setErrorHandler(new BailErrorStrategy());
    double answer = parser.lispExpr().value;
    System.out.println("\nThe value is "+answer+"\n");
   } catch (ParseCancellationException e) {
     System.out.println("\nSYNTAX ERROR\n");
     //e.printStackTrace();
    }
    catch (Exception e) {
     System.out.println("\nEVALUATION ERROR: CDR produces an empty list!\n");
     //e.printStackTrace();
    }
  } while (true);
 }
```

```
  static String readInput() {
    try {
      StringBuffer buffer = new StringBuffer();
      System.out.flush();
      int c = System.in.read();
      while(c != ';' && c != -1) {
        if (c != '\n')
          buffer.append((char)c);
        else {
          buffer.append(" ");
          System.out.print("LISP> ");
          System.out.flush();
        }
        c = System.in.read();
      }
      return buffer.toString().trim();
    } catch (IOException e) {
        return "";
      }
  }
}
```