```java
/* Generated By:JavaCC: Do not edit this line. Parser.java */
import AST.*;

public class Parser implements ParserConstants {
  /** Main entry point. */
  public static void main(String args [])
  {
    Parser parser = new Parser(System.in);
    ASTNode exp;
    while (true)
    {
      try
      {
        exp = parser.Start();
        System.out.println(exp.eval(new Environment()));
      }
      catch (Exception e)
      {
        System.out.println("Syntax Error!");
        parser.ReInit(System.in);
      }
    }
  }

  static final public ASTNode Start() throws ParseException {
  ASTNode t;
    t = Seq();
    jj_consume_token(EL);
    {if (true) return t;}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode Seq() throws ParseException {
  ASTNode e1, e2;
    e1 = Comp();
    label_1:
    while (true) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case SEMICOL:
        ;
        break;
      default:
        jj_la1[0] = jj_gen;
        break label_1;
      }
      jj_consume_token(SEMICOL);
      e2 = Seq();
      e1 = new ASTSeq(e1, e2);
    }
    {if (true) return e1;}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode Comp() throws ParseException {
  Token op;
  ASTNode e1, e2;
    e1 = Exp();
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case LOET:
    case GOET:
    case LT:
    case GT:
```

```
    case COMP_EQUALS:
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LOET:
        op = jj_consume_token(LOET);
        break;
      case GOET:
        op = jj_consume_token(GOET);
        break;
      case LT:
        op = jj_consume_token(LT);
        break;
      case GT:
        op = jj_consume_token(GT);
        break;
      case COMP_EQUALS:
        op = jj_consume_token(COMP_EQUALS);
        break;
      default:
        jj_la1[1] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
      e2 = Exp();
      switch (op.kind)
      {
        case LOET :
        e1 = new ASTLOET(e1, e2);
        break;
        case GOET :
        e1 = new ASTGOET(e1, e2);
        break;
        case LT :
        e1 = new ASTLT(e1, e2);
        break;
        case GT :
        e1 = new ASTGT(e1, e2);
        break;
        case COMP_EQUALS :
        e1 = new ASTEq(e1, e2);
        break;
      }
      break;
    default:
      jj_la1[2] = jj_gen;
      ;
    }
    {if (true) return e1;}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode Exp() throws ParseException {
  Token op;
  ASTNode t1, t2;
    t1 = Term();
    label_2:
    while (true) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case PLUS:
      case MINUS:
        ;
        break;
      default:
```

```java
        jj_la1[3] = jj_gen;
        break label_2;
      }
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case PLUS:
        op = jj_consume_token(PLUS);
        break;
      case MINUS:
        op = jj_consume_token(MINUS);
        break;
      default:
        jj_la1[4] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
      t2 = Exp();
      if (op.kind == PLUS)
      t1 = new ASTPlus(t1, t2);
      else t1 = new ASTSub(t1, t2);
    }
    {if (true) return t1;}
    throw new Error("Missing return statement in function");
  }

  static final public List < String > ParamList() throws ParseException {
  List < String > params = new LinkedList < String > ();
  Token onePar, multiplePar;
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case Id:
      onePar = jj_consume_token(Id);
      params.add(onePar.image);
      label_3:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case COMMA:
          ;
          break;
        default:
          jj_la1[5] = jj_gen;
          break label_3;
        }
        jj_consume_token(COMMA);
        multiplePar = jj_consume_token(Id);
        params.add(multiplePar.image);
      }
      break;
    default:
      jj_la1[6] = jj_gen;
      ;
    }
    {if (true) return params;}
    throw new Error("Missing return statement in function");
  }

  static final public List < ASTNode > ArgsList() throws ParseException {
  List < ASTNode > args = new LinkedList < ASTNode > ();
  ASTNode oneArg, multipleArgs;
    oneArg = Seq();
      args.add(oneArg);
    label_4:
    while (true) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
```

```
    case COMMA:
      ;
      break;
    default:
      jj_la1[7] = jj_gen;
      break label_4;
    }
    jj_consume_token(COMMA);
    multipleArgs = Seq();
      args.add(multipleArgs);
  }
  {if (true) return args;}
  throw new Error("Missing return statement in function");
}

static final public ASTNode Term() throws ParseException {
Token op;
ASTNode f, t;
List < ASTNode > args;
  f = Fact();
  switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
  case LPAR:
    jj_consume_token(LPAR);
    args = ArgsList();
    jj_consume_token(RPAR);
    f = new ASTApply(f, args);
    break;
  default:
    jj_la1[8] = jj_gen;
    ;
  }
  switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
  case ASSIGN:
    jj_consume_token(ASSIGN);
    t = Comp();
      f = new ASTAssign(f, t);
    break;
  default:
    jj_la1[11] = jj_gen;
    label_5:
    while (true) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case TIMES:
      case DIV:
        ;
        break;
      default:
        jj_la1[9] = jj_gen;
        break label_5;
      }
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case TIMES:
        op = jj_consume_token(TIMES);
        break;
      case DIV:
        op = jj_consume_token(DIV);
        break;
      default:
        jj_la1[10] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
```

```
        t = Term();
        if (op.kind == TIMES)
        f = new ASTMul(f, t);
        else f = new ASTDiv(f, t);
      }
    }
    {if (true) return f;}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode Fact() throws ParseException {
  Token n;
  ASTNode t;
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case Id:
      n = jj_consume_token(Id);
      t = new ASTId(n.image);
      break;
    case Num:
      n = jj_consume_token(Num);
      t = new ASTNum(Integer.parseInt(n.image));
      break;
    case TRUE:
    case FALSE:
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case TRUE:
        n = jj_consume_token(TRUE);
        break;
      case FALSE:
        n = jj_consume_token(FALSE);
        break;
      default:
        jj_la1[12] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
      t = new ASTBool(Boolean.parseBoolean(n.image));
      break;
    case LET:
      t = LetBuild();
      break;
    case FUN:
      t = FunBuild();
      break;
    case LPAR:
      jj_consume_token(LPAR);
      t = Seq();
      jj_consume_token(RPAR);
      break;
    case NEW:
      t = NewBuild();
      break;
    case DESREF:
      t = DesrefBuild();
      break;
    case WHILE:
      t = WhileBuild();
      break;
    case IF:
      t = IfBuild();
      break;
    default:
      jj_la1[13] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
    }
    {if (true) return t;}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode IfBuild() throws ParseException {
  ASTNode cond, e1, e2;
    jj_consume_token(IF);
    cond = Seq();
    jj_consume_token(THEN);
    e1 = Seq();
    jj_consume_token(ELSE);
    e2 = Seq();
    jj_consume_token(END);
    {if (true) return new ASTIf(cond, e1, e2);}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode NewBuild() throws ParseException {
  ASTNode f;
    jj_consume_token(NEW);
    f = Fact();
    {if (true) return new ASTNew(f);}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode DesrefBuild() throws ParseException {
  ASTNode f;
    jj_consume_token(DESREF);
    f = Fact();
    {if (true) return new ASTDesref(f);}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode LetBuild() throws ParseException {
  List < String > ids = new LinkedList < String > ();
  List < ASTNode > exps = new LinkedList < ASTNode > ();
  Token id;
  ASTNode exp_init, exp_body;
    jj_consume_token(LET);
    label_6:
    while (true) {
      id = jj_consume_token(Id);
      jj_consume_token(EQUALS);
      exp_init = Seq();
        ids.add(id.image);
        exps.add(exp_init);
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case Id:
        ;
        break;
      default:
        jj_la1[14] = jj_gen;
        break label_6;
      }
    }
    jj_consume_token(IN);
    exp_body = Seq();
    jj_consume_token(END);
```

```java
    {if (true) return new ASTLet(ids, exps, exp_body);}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode WhileBuild() throws ParseException {
  ASTNode cond, e;
    jj_consume_token(WHILE);
    cond = Seq();
    jj_consume_token(DO);
    e = Seq();
    jj_consume_token(END);
    {if (true) return new ASTWhile(cond, e);}
    throw new Error("Missing return statement in function");
  }

  static final public ASTNode FunBuild() throws ParseException {
  List < String > param;
  ASTNode exp;
    jj_consume_token(FUN);
    param = ParamList();
    jj_consume_token(ARROW);
    exp = Seq();
    jj_consume_token(END);
    {if (true) return new ASTFun(param, exp);}
    throw new Error("Missing return statement in function");
  }

  static private boolean jj_initialized_once = false;
  /** Generated Token Manager. */
  static public ParserTokenManager token_source;
  static SimpleCharStream jj_input_stream;
  /** Current token. */
  static public Token token;
  /** Next token. */
  static public Token jj_nt;
  static private int jj_ntk;
  static private int jj_gen;
  static final private int[] jj_la1 = new int[15];
  static private int[] jj_la1_0;
  static private int[] jj_la1_1;
  static {
      jj_la1_init_0();
      jj_la1_init_1();
   }
   private static void jj_la1_init_0() {
      jj_la1_0 = new int[] {0x200000,0x1f000,0x1f000,0xc0000000,0xc0000000,0x100000,0x10000000,0x100000,0x0,0x0,0x0,0x20000,0x60,0x314c0a60,0x10000000,};
   }
   private static void jj_la1_init_1() {
      jj_la1_1 = new int[] {0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x4,0x3,0x3,0x0,0x0,0x4,0x0,};
   }

  /** Constructor with InputStream. */
  public Parser(java.io.InputStream stream) {
     this(stream, null);
  }
  /** Constructor with InputStream and supplied encoding */
  public Parser(java.io.InputStream stream, String encoding) {
    if (jj_initialized_once) {
      System.out.println("ERROR: Second call to constructor of static parser.  ");
      System.out.println("       You must either use ReInit() or set the JavaCC option
STATIC to false");
      System.out.println("       during parser generation.");
      throw new Error();
    }
    jj_initialized_once = true;
    try { jj_input_stream = new SimpleCharStream(stream, encoding, 1, 1); } catch(java.io.UnsupportedEncodingException e) { throw new RuntimeException(e); }
    token_source = new ParserTokenManager(jj_input_stream);
    token = new Token();
    jj_ntk = -1;
    jj_gen = 0;
    for (int i = 0; i < 15; i++) jj_la1[i] = -1;
  }

  /** Reinitialise. */
  static public void ReInit(java.io.InputStream stream) {
     ReInit(stream, null);
  }
  /** Reinitialise. */
  static public void ReInit(java.io.InputStream stream, String encoding) {
    try { jj_input_stream.ReInit(stream, encoding, 1, 1); } catch(java.io.UnsupportedEncodingException e) { throw new RuntimeException(e); }
    token_source.ReInit(jj_input_stream);
    token = new Token();
    jj_ntk = -1;
    jj_gen = 0;
    for (int i = 0; i < 15; i++) jj_la1[i] = -1;
  }

  /** Constructor. */
  public Parser(java.io.Reader stream) {
    if (jj_initialized_once) {
      System.out.println("ERROR: Second call to constructor of static parser. ");
      System.out.println("       You must either use ReInit() or set the JavaCC option STATIC to false");
      System.out.println("       during parser generation.");
      throw new Error();
    }
    jj_initialized_once = true;
    jj_input_stream = new SimpleCharStream(stream, 1, 1);
    token_source = new ParserTokenManager(jj_input_stream);
    token = new Token();
    jj_ntk = -1;
    jj_gen = 0;
    for (int i = 0; i < 15; i++) jj_la1[i] = -1;
  }

  /** Reinitialise. */
  static public void ReInit(java.io.Reader stream) {
    jj_input_stream.ReInit(stream, 1, 1);
    token_source.ReInit(jj_input_stream);
    token = new Token();
    jj_ntk = -1;
    jj_gen = 0;
    for (int i = 0; i < 15; i++) jj_la1[i] = -1;
  }

  /** Constructor with generated Token Manager. */
  public Parser(ParserTokenManager tm) {
    if (jj_initialized_once) {
      System.out.println("ERROR: Second call to constructor of static parser. ");
      System.out.println("       You must either use ReInit() or set the JavaCC option
STATIC to false");
      System.out.println("       during parser generation.");
      throw new Error();
    }
    jj_initialized_once = true;
    token_source = tm;
    token = new Token();
    jj_ntk = -1;
    jj_gen = 0;
    for (int i = 0; i < 15; i++) jj_la1[i] = -1;
  }

  /** Reinitialise. */
  public void ReInit(ParserTokenManager tm) {
    token_source = tm;
    token = new Token();
    jj_ntk = -1;
    jj_gen = 0;
    for (int i = 0; i < 15; i++) jj_la1[i] = -1;
  }

  static private Token jj_consume_token(int kind) throws ParseException {
    Token oldToken;
    if ((oldToken = token).next != null) token = token.next;
    else token = token.next = token_source.getNextToken();
    jj_ntk = -1;
    if (token.kind == kind) {
      jj_gen++;
      return token;
    }
    token = oldToken;
    jj_kind = kind;
    throw generateParseException();
  }


/** Get the next Token. */
  static final public Token getNextToken() {
    if (token.next != null) token = token.next;
    else token = token.next = token_source.getNextToken();
    jj_ntk = -1;
    jj_gen++;
    return token;
  }

/** Get the specific Token. */
  static final public Token getToken(int index) {
    Token t = token;
    for (int i = 0; i < index; i++) {
      if (t.next != null) t = t.next;
      else t = t.next = token_source.getNextToken();
    }
    return t;
  }

  static private int jj_ntk() {
    if ((jj_nt=token.next) == null)
      return (jj_ntk = (token.next=token_source.getNextToken()).kind);
    else
      return (jj_ntk = jj_nt.kind);
  }

```

```java
  static private java.util.List<int[]> jj_expentries = new java.util.ArrayList<int[]>();
  static private int[] jj_expentry;
  static private int jj_kind = -1;

  /** Generate ParseException. */
  static public ParseException generateParseException() {
    jj_expentries.clear();
    boolean[] la1tokens = new boolean[37];
    if (jj_kind >= 0) {
      la1tokens[jj_kind] = true;
      jj_kind = -1;
    }
    for (int i = 0; i < 15; i++) {
      if (jj_la1[i] == jj_gen) {
        for (int j = 0; j < 32; j++) {
          if ((jj_la1_0[i] & (1<<j)) != 0) {
            la1tokens[j] = true;
          }
          if ((jj_la1_1[i] & (1<<j)) != 0) {
            la1tokens[32+j] = true;
          }
        }
      }
    }
    for (int i = 0; i < 37; i++) {
      if (la1tokens[i]) {
        jj_expentry = new int[1];
        jj_expentry[0] = i;
        jj_expentries.add(jj_expentry);
      }
    }
    int[][] exptokseq = new int[jj_expentries.size()][];
    for (int i = 0; i < jj_expentries.size(); i++) {
      exptokseq[i] = jj_expentries.get(i);
    }
    return new ParseException(token, exptokseq, tokenImage);
  }

  /** Enable tracing. */
  static final public void enable_tracing() {
  }

  /** Disable tracing. */
  static final public void disable_tracing() {
  }

}
```