```java
/* The following code was generated by JFlex 1.4.1 on 4/30/05 10:55 AM */

/*-***
 *
 * This file defines a lexical analyzer for the EJay programming language, as
 * specified in CSC 330 Assignments 2 and 3.  This lexer is intended to be
 * integrated with the will later be integrated with a CUP-based parser
 * solution to Assignment 3.
 *
 */


import java_cup.runtime.*;



/**
 * This class is a scanner generated by
 * <a href="http://www.jflex.de/">JFlex</a> 1.4.1
 * on 4/30/05 10:55 AM from the specification file
 * <tt>ejay.jflex</tt>
 */
class EJayLexer implements java_cup.runtime.Scanner {

  /** This character denotes the end of file */
  public static final int YYEOF = -1;

  /** initial size of the lookahead buffer */
  private static final int ZZ_BUFFERSIZE = 16384;

  /** lexical states */
  public static final int STRING = 2;
  public static final int YYINITIAL = 0;
  public static final int COMMENT = 1;

  /**
   * Translates characters to character classes
   */
  private static final char [] ZZ_CMAP = {
     0,  0,  0,  0,  0,  0,  0,  0,  0, 19, 17,  0,  0, 14,  0,  0,
     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    19, 51, 13,  0,  0,  0, 52, 54, 39, 40, 18, 48, 44, 11, 12, 16,
     2, 55, 55, 55,  4,  4,  4,  4,  3,  3,  0, 43, 49, 47, 50,  0,
     0,  5,  5,  5,  9, 10,  9,  1,  1,  1,  1,  1,  7,  1,  1,  1,
     1,  1,  1,  1,  1,  1,  1,  1,  8,  1,  1, 45, 53, 46,  0, 15,
     0, 24, 20, 37, 31, 23, 28, 36, 33, 27,  1,  1, 22,  1, 25, 21,
    38,  1, 34, 26, 29, 35, 30, 32,  8,  1,  1, 41,  6, 42,  0,  0
  };

  /**
   * Translates DFA states to action switch labels.
   */
  private static final int [] ZZ_ACTION = zzUnpackAction();

  private static final String ZZ_ACTION_PACKED_0 =
    "\3\0\1\1\1\2\2\3\1\1\1\4\1\5\1\6"+
    "\1\7\1\10\1\11\12\2\1\12\1\13\1\14\1\15"+
    "\1\16\1\17\1\20\1\21\1\22\1\23\1\24\1\25"+
    "\1\26\1\1\3\27\1\30\1\31\1\0\1\32\1\0"+
    "\2\3\1\0\1\33\1\0\1\33\1\34\1\35\1\36"+
    "\4\2\1\37\7\2\1\40\1\41\1\42\1\43\1\44"+
    "\1\45\1\46\2\47\1\50\1\51\1\52\1\53\1\54"+
    "\1\55\1\56\1\57\1\32\1\60\1\33\1\0\3\2"+
    "\1\61\5\2\1\62\2\2\1\47\1\60\1\2\1\63"+
    "\4\2\1\64\1\65\6\2\1\66\1\67\1\70\1\2"+
    "\1\71\1\2\1\72\1\73\1\74\1\75";

  private static int [] zzUnpackAction() {
    int [] result = new int[126];
    int offset = 0;
    offset = zzUnpackAction(ZZ_ACTION_PACKED_0, offset, result);
    return result;
  }

  private static int zzUnpackAction(String packed, int offset, int [] result) {
    int i = 0;       /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int count = packed.charAt(i++);
      int value = packed.charAt(i++);
      do result[j++] = value; while (--count > 0);
    }
    return j;
  }


  /**
   * Translates a state to a row index in the transition table
   */
  private static final int [] ZZ_ROWMAP = zzUnpackRowMap();

  private static final String ZZ_ROWMAP_PACKED_0 =
    "\0\0\0\70\0\160\0\250\0\340\0\u0118\0\u0150\0\u0188"+
    "\0\250\0\u01c0\0\250\0\u01f8\0\250\0\250\0\u0230\0\u0268"+
    "\0\u02a0\0\u02d8\0\u0310\0\u0348\0\u0380\0\u03b8\0\u03f0\0\u0428"+
    "\0\250\0\250\0\250\0\250\0\250\0\250\0\250\0\250"+
    "\0\u0460\0\250\0\u0498\0\u04d0\0\u0508\0\u0540\0\250\0\u0578"+
    "\0\u05b0\0\250\0\250\0\u05e8\0\u0620\0\u0658\0\u0690\0\250"+
    "\0\u0690\0\250\0\u06c8\0\u0700\0\250\0\u0738\0\250\0\u0770"+
    "\0\u07a8\0\u07e0\0\u0818\0\340\0\u0850\0\u0888\0\u08c0\0\u08f8"+
    "\0\u0930\0\u0968\0\u09a0\0\250\0\250\0\250\0\250\0\250"+
    "\0\250\0\250\0\u09d8\0\u0a10\0\250\0\250\0\250\0\250"+
    "\0\250\0\250\0\250\0\250\0\250\0\u0a48\0\u0a80\0\u0ab8"+
    "\0\u0af0\0\u0b28\0\u0b60\0\340\0\u0b98\0\u0bd0\0\u0c08\0\u0c40"+
    "\0\u0c78\0\340\0\u0cb0\0\u0ce8\0\250\0\250\0\u0d20\0\340"+
    "\0\u0d58\0\u0d90\0\u0dc8\0\u0e00\0\340\0\340\0\u0e38\0\u0e70"+
    "\0\u0ea8\0\u0ee0\0\u0f18\0\u0f50\0\340\0\340\0\340\0\u0f88"+
    "\0\340\0\u0fc0\0\340\0\340\0\340\0\340";

  private static int [] zzUnpackRowMap() {
    int [] result = new int[126];
```

```java
    int offset = 0;
    offset = zzUnpackRowMap(ZZ_ROWMAP_PACKED_0, offset, result);
    return result;
  }

  private static int zzUnpackRowMap(String packed, int offset, int [] result) {
    int i = 0;  /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int high = packed.charAt(i++) << 16;
      result[j++] = high | packed.charAt(i++);
    }
    return j;
  }

  /**
   * The transition table of the DFA
   */
  private static final int [] ZZ_TRANS = zzUnpackTrans();

  private static final String ZZ_TRANS_PACKED_0 =
    "\1\4\1\5\1\6\2\7\1\5\1\10\4\5\1\11"+
    "\1\12\1\13\2\4\1\14\1\15\1\16\1\15\1\17"+
    "\2\5\1\20\2\5\1\21\1\22\1\23\1\24\1\25"+
    "\1\5\1\26\1\5\1\27\3\5\1\30\1\31\1\32"+
    "\1\33\1\34\1\35\1\36\1\37\1\40\1\41\1\42"+
    "\1\43\1\44\1\45\1\46\2\4\1\7\20\47\1\50"+
    "\1\47\1\51\45\47\15\52\1\53\1\0\2\52\1\0"+
    "\43\52\1\54\2\52\71\0\5\5\1\0\4\5\4\0"+
    "\1\5\4\0\23\5\20\0\1\5\2\0\1\55\1\56"+
    "\1\55\1\0\1\57\1\60\1\61\1\62\1\63\1\0"+
    "\1\64\11\0\1\60\1\63\4\0\1\62\2\0\1\62"+
    "\27\0\1\55\2\0\3\7\1\0\2\60\1\0\1\62"+
    "\1\63\1\0\1\64\11\0\1\60\1\63\4\0\1\62"+
    "\2\0\1\62\27\0\1\7\6\0\1\65\63\0\3\64"+
    "\62\0\1\64\20\0\1\66\1\0\1\67\46\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\1\5\1\70\21\5"+
    "\20\0\1\5\1\0\5\5\1\0\4\5\4\0\1\5"+
    "\4\0\2\5\1\71\20\5\20\0\1\5\1\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\11\5\1\72\11\5"+
    "\20\0\1\5\1\0\5\5\1\0\4\5\4\0\1\5"+
    "\4\0\5\5\1\73\2\5\1\74\12\5\20\0\1\5"+
    "\1\0\5\5\1\0\4\5\4\0\1\5\4\0\2\5"+
    "\1\75\1\5\1\76\16\5\20\0\1\5\1\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\16\5\1\77\4\5"+
    "\20\0\1\5\1\0\5\5\1\0\4\5\4\0\1\5"+
    "\4\0\1\5\1\100\21\5\20\0\1\5\1\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\15\5\1\101\5\5"+
    "\20\0\1\5\1\0\5\5\1\0\4\5\4\0\1\5"+
    "\4\0\3\5\1\102\17\5\20\0\1\5\1\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\16\5\1\103\4\5"+
    "\20\0\1\5\57\0\1\104\67\0\1\105\67\0\1\106"+
    "\67\0\1\107\74\0\1\110\25\0\1\111\65\0\1\112"+
    "\51\0\1\113\1\0\1\114\10\0\1\115\6\0\1\116"+
    "\4\0\1\117\2\0\1\120\1\121\4\0\1\122\22\0"+
    "\1\123\1\124\1\113\2\0\1\55\1\56\1\55\1\0"+
    "\2\125\1\0\1\62\1\63\1\0\1\64\11\0\1\125"+
    "\1\63\4\0\1\62\2\0\1\62\27\0\1\55\2\0"+
    "\3\56\4\0\1\62\1\63\1\0\1\64\12\0\1\63"+
    "\4\0\1\62\2\0\1\62\27\0\1\56\2\0\4\126"+
    "\3\0\2\126\11\0\1\126\2\0\2\126\3\0\1\126"+
    "\2\0\1\126\5\0\1\126\21\0\1\126\2\0\3\127"+
    "\6\0\1\130\44\0\1\130\6\0\1\127\2\0\3\64"+
    "\4\0\1\62\1\63\14\0\1\63\4\0\1\62\2\0"+
    "\1\62\27\0\1\64\21\66\1\0\46\66\1\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\1\5\1\131\21\5"+
    "\20\0\1\5\1\0\5\5\1\0\4\5\4\0\1\5"+
    "\4\0\6\5\1\132\14\5\20\0\1\5\1\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\16\5\1\133\4\5"+
    "\20\0\1\5\1\0\5\5\1\0\4\5\4\0\1\5"+
    "\4\0\11\5\1\134\11\5\20\0\1\5\1\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\1\5\1\135\21\5"+
    "\20\0\1\5\1\0\5\5\1\0\4\5\4\0\1\5"+
    "\4\0\2\5\1\136\20\5\20\0\1\5\1\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\17\5\1\137\3\5"+
    "\20\0\1\5\1\0\5\5\1\0\4\5\4\0\1\5"+
    "\4\0\7\5\1\140\13\5\20\0\1\5\1\0\5\5"+
    "\1\0\4\5\4\0\1\5\4\0\7\5\1\141\13\5"+
    "\20\0\1\5\1\0\5\5\1\0\4\5\4\0\1\5"+
    "\4\0\10\5\1\142\1\143\11\5\20\0\1\5\1\0"+
    "\5\5\1\0\4\5\4\0\1\5\4\0\7\5\1\144"+
    "\13\5\20\0\1\5\2\0\1\114\1\0\1\114\62\0"+
    "\1\114\2\0\1\145\1\0\1\145\62\0\1\145\2\0"+
    "\4\126\2\146\1\0\2\126\11\0\1\126\1\0\1\146"+
    "\2\126\3\0\1\126\2\0\1\126\5\0\1\126\21\0"+
    "\1\126\2\0\3\127\4\0\1\62\22\0\1\62\2\0"+
    "\1\62\27\0\1\127\2\0\3\127\62\0\1\127\1\0"+
    "\5\5\1\0\4\5\4\0\1\5\4\0\2\5\1\147"+
    "\20\5\20\0\1\5\1\0\5\5\1\0\4\5\4\0"+
    "\1\5\4\0\3\5\1\150\17\5\20\0\1\5\1\0"+
    "\5\5\1\0\4\5\4\0\1\5\4\0\7\5\1\151"+
    "\7\5\1\152\3\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\4\5\1\153\16\5\20\0"+
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\6\5\1\154\14\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\3\5\1\155\17\5\20\0"+
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\13\5\1\156\7\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\2\5\1\157\20\5\20\0"+
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\17\5\1\160\3\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\5\5\1\161\15\5\20\0"+
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\3\5\1\162\17\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\5\5\1\163\15\5\20\0"+
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\21\5\1\164\1\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\11\5\1\165\11\5\20\0"+
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\3\5\1\166\17\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\3\5\1\167\17\5\20\0"+
```

```java
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\16\5\1\170\4\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\11\5\1\171\11\5\20\0"+
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\4\5\1\172\16\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\20\5\1\173\2\5\20\0"+
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\11\5\1\174\11\5\20\0\1\5\1\0\5\5\1\0"+
    "\4\5\4\0\1\5\4\0\5\5\1\175\15\5\20\0"+
    "\1\5\1\0\5\5\1\0\4\5\4\0\1\5\4\0"+
    "\5\5\1\176\15\5\20\0\1\5";

  private static int [] zzUnpackTrans() {
    int [] result = new int[4088];
    int offset = 0;
    offset = zzUnpackTrans(ZZ_TRANS_PACKED_0, offset, result);
    return result;
  }

  private static int zzUnpackTrans(String packed, int offset, int [] result) {
    int i = 0;       /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int count = packed.charAt(i++);
      int value = packed.charAt(i++);
      value--;
      do result[j++] = value; while (--count > 0);
    }
    return j;
  }


  /* error codes */
  private static final int ZZ_UNKNOWN_ERROR = 0;
  private static final int ZZ_NO_MATCH = 1;
  private static final int ZZ_PUSHBACK_2BIG = 2;

  /* error messages for the codes above */
  private static final String ZZ_ERROR_MSG[] = {
    "Unkown internal scanner error",
    "Error: could not match input",
    "Error: pushback value was too large"
  };

  /**
   * ZZ_ATTRIBUTE[aState] contains the attributes of state <code>aState</code>
   */
  private static final int [] ZZ_ATTRIBUTE = zzUnpackAttribute();

  private static final String ZZ_ATTRIBUTE_PACKED_0 =
    "\3\0\1\11\4\1\1\11\1\1\1\11\1\1\2\11"+
    "\12\1\10\11\1\1\1\11\4\1\1\11\2\1\2\11"+
    "\1\0\1\1\1\0\1\1\1\11\1\0\1\11\1\0"+
    "\1\1\1\11\1\1\1\11\14\1\7\11\2\1\11\11"+
    "\2\1\1\0\14\1\2\11\30\1";

  private static int [] zzUnpackAttribute() {
    int [] result = new int[126];
    int offset = 0;
    offset = zzUnpackAttribute(ZZ_ATTRIBUTE_PACKED_0, offset, result);
    return result;
  }

  private static int zzUnpackAttribute(String packed, int offset, int [] result) {
    int i = 0;       /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int count = packed.charAt(i++);
      int value = packed.charAt(i++);
      do result[j++] = value; while (--count > 0);
    }
    return j;
  }

  /** the input device */
  private java.io.Reader zzReader;

  /** the current state of the DFA */
  private int zzState;

  /** the current lexical state */
  private int zzLexicalState = YYINITIAL;

  /** this buffer contains the current text to be matched and is
      the source of the yytext() string */
  private char zzBuffer[] = new char[ZZ_BUFFERSIZE];

  /** the textposition at the last accepting state */
  private int zzMarkedPos;

  /** the textposition at the last state to be included in yytext */
  private int zzPushbackPos;

  /** the current text position in the buffer */
  private int zzCurrentPos;

  /** startRead marks the beginning of the yytext() string in the buffer */
  private int zzStartRead;

  /** endRead marks the last character in the buffer, that has been read
      from input */
  private int zzEndRead;

  /** number of newlines encountered up to the start of the matched text */
  private int yyline;

  /** the number of characters up to the start of the matched text */
  private int yychar;

  /**
```

```
   * the number of characters from the last newline up to the start of the
   * matched text
   */
  private int yycolumn;

  /**
   * zzAtBOL == true <=> the scanner is currently at the beginning of a line
   */
  private boolean zzAtBOL = true;

  /** zzAtEOF == true <=> the scanner is at the EOF */
  private boolean zzAtEOF;

  /** denotes if the user-EOF-code has already been executed */
  private boolean zzEOFDone;

  /* user code: */

/**
 * Return a new Symbol with the given token id, and with the current line and
 * column numbers.
 */
Symbol newSym(int tokenId) {
    return new Symbol(tokenId, yyline, yycolumn);
}

/**
 * Return a new Symbol with the given token id, the current line and column
 * numbers, and the given token value.  The value is used for tokens such as
 * identifiers and numbers.
 */
Symbol newSym(int tokenId, Object value) {
    return new Symbol(tokenId, yyline, yycolumn, value);
}

/**
 * Overload that takes line and column as args; used for strings.
 */
Symbol newSym(int tokenId, Object value, int line, int column) {
    return new Symbol(tokenId, line, column, value);
}

/**
 * Groom a decimal or octal literal sufficiently for consumption by the Long
 * constructor.  This entails removing a trailing "l" or "L" char, if present.
 */
String groomDecimalOrOctal(String decimal) {
    return
        ((decimal.charAt(decimal.length()-1) == 'l') ||
         (decimal.charAt(decimal.length()-1) == 'L'))
        ? decimal.substring(0, decimal.length()-1)
        : decimal;
}

/*
 * Groom a hex literal sufficiently for consumption by parseLong.  This entails
 * removing the leading "0x" prefix, converting all letters to upper case, then
 * grooming as a decimal.
 */
String groomHex(String hex) {
    return
        groomDecimalOrOctal((hex.substring(2, hex.length())).toUpperCase());
}

void p(String msg) {System.out.println(msg);}
void pt(String msg) {System.out.println(msg + ";" + yytext());}

int comment_depth = 0;
int string_start_line;
int string_start_column;
String string_body = "";
String comment_text = "";



  /**
   * Creates a new scanner
   * There is also a java.io.InputStream version of this constructor.
   *
   * @param   in  the java.io.Reader to read input from.
   */
  EJayLexer(java.io.Reader in) {
    this.zzReader = in;
  }

  /**
   * Creates a new scanner.
   * There is also java.io.Reader version of this constructor.
   *
   * @param   in  the java.io.Inputstream to read input from.
   */
  EJayLexer(java.io.InputStream in) {
    this(new java.io.InputStreamReader(in));
  }


  /**
   * Refills the input buffer.
   *
   * @return      <code>false</code>, iff there was new input.
   *
   * @exception   java.io.IOException  if any I/O-Error occurs
   */
  private boolean zzRefill() throws java.io.IOException {

    /* first: make room (if you can) */
    if (zzStartRead > 0) {
      System.arraycopy(zzBuffer, zzStartRead,
                       zzBuffer, 0,
                       zzEndRead-zzStartRead);

      /* translate stored positions */
```

```
      zzEndRead-= zzStartRead;
      zzCurrentPos-= zzStartRead;
      zzMarkedPos-= zzStartRead;
      zzPushbackPos-= zzStartRead;
      zzStartRead = 0;
    }

    /* is the buffer big enough? */
    if (zzCurrentPos >= zzBuffer.length) {
      /* if not: blow it up */
      char newBuffer[] = new char[zzCurrentPos*2];
      System.arraycopy(zzBuffer, 0, newBuffer, 0, zzBuffer.length);
      zzBuffer = newBuffer;
    }

    /* finally: fill the buffer with new input */
    int numRead = zzReader.read(zzBuffer, zzEndRead,
                                            zzBuffer.length-zzEndRead);

    if (numRead < 0) {
      return true;
    }
    else {
      zzEndRead+= numRead;
      return false;
    }
  }


  /**
   * Closes the input stream.
   */
  public final void yyclose() throws java.io.IOException {
    zzAtEOF = true;            /* indicate end of file */
    zzEndRead = zzStartRead;  /* invalidate buffer    */

    if (zzReader != null)
      zzReader.close();
  }


  /**
   * Resets the scanner to read from a new input stream.
   * Does not close the old reader.
   *
   * All internal variables are reset, the old input stream
   * <b>cannot</b> be reused (internal buffer is discarded and lost).
   * Lexical state is set to <tt>ZZ_INITIAL</tt>.
   *
   * @param reader   the new input stream
   */
  public final void yyreset(java.io.Reader reader) {
    zzReader = reader;
    zzAtBOL  = true;
    zzAtEOF  = false;
    zzEndRead = zzStartRead = 0;
    zzCurrentPos = zzMarkedPos = zzPushbackPos = 0;
    yyline = yychar = yycolumn = 0;
    zzLexicalState = YYINITIAL;
  }


  /**
   * Returns the current lexical state.
   */
  public final int yystate() {
    return zzLexicalState;
  }


  /**
   * Enters a new lexical state
   *
   * @param newState the new lexical state
   */
  public final void yybegin(int newState) {
    zzLexicalState = newState;
  }


  /**
   * Returns the text matched by the current regular expression.
   */
  public final String yytext() {
    return new String( zzBuffer, zzStartRead, zzMarkedPos-zzStartRead );
  }


  /**
   * Returns the character at position <tt>pos</tt> from the
   * matched text.
   *
   * It is equivalent to yytext().charAt(pos), but faster
   *
   * @param pos the position of the character to fetch.
   *            A value from 0 to yylength()-1.
   *
   * @return the character at position pos
   */
  public final char yycharat(int pos) {
    return zzBuffer[zzStartRead+pos];
  }


  /**
   * Returns the length of the matched text region.
   */
  public final int yylength() {
    return zzMarkedPos-zzStartRead;
  }

```

```java
  /**
   * Reports an error that occured while scanning.
   *
   * In a wellformed scanner (no or only correct usage of
   * yypushback(int) and a match-all fallback rule) this method
   * will only be called with things that "Can't Possibly Happen".
   * If this method is called, something is seriously wrong
   * (e.g. a JFlex bug producing a faulty scanner etc.).
   *
   * Usual syntax/scanner level error handling should be done
   * in error fallback rules.
   *
   * @param   errorCode  the code of the errormessage to display
   */
  private void zzScanError(int errorCode) {
    String message;
    try {
      message = ZZ_ERROR_MSG[errorCode];
    }
    catch (ArrayIndexOutOfBoundsException e) {
      message = ZZ_ERROR_MSG[ZZ_UNKNOWN_ERROR];
    }

    throw new Error(message);
  }


  /**
   * Pushes the specified amount of characters back into the input stream.
   *
   * They will be read again by then next call of the scanning method
   *
   * @param number  the number of characters to be read again.
   *                This number must not be greater than yylength()!
   */
  public void yypushback(int number)  {
    if ( number > yylength() )
      zzScanError(ZZ_PUSHBACK_2BIG);

    zzMarkedPos -= number;
  }


  /**
   * Contains user EOF-code, which will be executed exactly once,
   * when the end of file is reached
   */
  private void zzDoEOF() throws java.io.IOException {
    if (!zzEOFDone) {
      zzEOFDone = true;
      yyclose();
    }
  }


  /**
   * Resumes scanning until the next regular expression is matched,
   * the end of input is encountered or an I/O-Error occurs.
   *
   * @return      the next token
   * @exception   java.io.IOException  if any I/O-Error occurs
   */
  public java_cup.runtime.Symbol next_token() throws java.io.IOException {
    int zzInput;
    int zzAction;

    // cached fields:
    int zzCurrentPosL;
    int zzMarkedPosL;
    int zzEndReadL = zzEndRead;
    char [] zzBufferL = zzBuffer;
    char [] zzCMapL = ZZ_CMAP;

    int [] zzTransL = ZZ_TRANS;
    int [] zzRowMapL = ZZ_ROWMAP;
    int [] zzAttrL = ZZ_ATTRIBUTE;

    while (true) {
      zzMarkedPosL = zzMarkedPos;

      boolean zzR = false;
      for (zzCurrentPosL = zzStartRead; zzCurrentPosL < zzMarkedPosL;
                                                             zzCurrentPosL++) {
        switch (zzBufferL[zzCurrentPosL]) {
        case '\u000B':
        case '\u000C':
        case '\u0085':
        case '\u2028':
        case '\u2029':
          yyline++;
          yycolumn = 0;
          zzR = false;
          break;
        case '\r':
          yyline++;
          yycolumn = 0;
          zzR = true;
          break;
        case '\n':
          if (zzR)
            zzR = false;
          else {
            yyline++;
            yycolumn = 0;
          }
          break;
        default:
          zzR = false;
          yycolumn++;
        }
      }

```

```
    if (zzR) {
      // peek one character ahead if it is \n (if we have counted one line too much)
      boolean zzPeek;
      if (zzMarkedPosL < zzEndReadL)
        zzPeek = zzBufferL[zzMarkedPosL] == '\n';
      else if (zzAtEOF)
        zzPeek = false;
      else {
        boolean eof = zzRefill();
        zzEndReadL = zzEndRead;
        zzMarkedPosL = zzMarkedPos;
        zzBufferL = zzBuffer;
        if (eof)
          zzPeek = false;
        else
          zzPeek = zzBufferL[zzMarkedPosL] == '\n';
      }
      if (zzPeek) yyline--;
    }
    zzAction = -1;

    zzCurrentPosL = zzCurrentPos = zzStartRead = zzMarkedPosL;

    zzState = zzLexicalState;


    zzForAction: {
      while (true) {

        if (zzCurrentPosL < zzEndReadL)
          zzInput = zzBufferL[zzCurrentPosL++];
        else if (zzAtEOF) {
          zzInput = YYEOF;
          break zzForAction;
        }
        else {
          // store back cached positions
          zzCurrentPos  = zzCurrentPosL;
          zzMarkedPos   = zzMarkedPosL;
          boolean eof = zzRefill();
          // get translated positions and possibly new buffer
          zzCurrentPosL  = zzCurrentPos;
          zzMarkedPosL   = zzMarkedPos;
          zzBufferL      = zzBuffer;
          zzEndReadL     = zzEndRead;
          if (eof) {
            zzInput = YYEOF;
            break zzForAction;
          }
          else {
            zzInput = zzBufferL[zzCurrentPosL++];
          }
        }
        int zzNext = zzTransL[ zzRowMapL[zzState] + zzCMapL[zzInput] ];
        if (zzNext == -1) break zzForAction;
        zzState = zzNext;

        int zzAttributes = zzAttrL[zzState];
        if ( (zzAttributes & 1) == 1 ) {
          zzAction = zzState;
          zzMarkedPosL = zzCurrentPosL;
          if ( (zzAttributes & 8) == 8 ) break zzForAction;
        }

      }
    }

    // store back cached position
    zzMarkedPos = zzMarkedPosL;

    switch (zzAction < 0 ? zzAction : ZZ_ACTION[zzAction]) {
      case 60:
        { return newSym(sym.RETURN);
        }
      case 62: break;
      case 47:
        { string_body += "\'";
        }
      case 63: break;
      case 53:
        { return newSym(sym.VOID);
        }
      case 64: break;
      case 21:
        { return newSym(sym.GTR);
        }
      case 65: break;
      case 56:
        { return newSym(sym.WHILE);
        }
      case 66: break;
      case 51:
        { return newSym(sym.ELSE);
        }
      case 67: break;
      case 13:
        { return newSym(sym.RT_BRACE);
        }
      case 68: break;
      case 44:
        { string_body += "\t";
        }
      case 69: break;
      case 27:
        { return newSym(sym.FLOATING_PT, new Double(yytext()));
        }
      case 70: break;
      case 30:
        { comment_depth++; comment_text = "/*"; yybegin(COMMENT);
        }
      case 71: break;
      case 54:
```

```java
            { return newSym(sym.FLOAT);
            }
          case 72: break;
          case 41:
            { string_body += "\b";
            }
          case 73: break;
          case 2:
            { return newSym(sym.IDENT, yytext());
            }
          case 74: break;
          case 59:
            { return newSym(sym.STRUCT);
            }
          case 75: break;
          case 50:
            { return newSym(sym.REF);
            }
          case 76: break;
          case 4:
            { return newSym(sym.MINUS);
            }
          case 77: break;
          case 57:
            { return newSym(sym.PRINT);
            }
          case 78: break;
          case 6:
            { string_body = ""; string_start_line = yyline;
                      string_start_column = yycolumn;  yybegin(STRING);
            }
          case 79: break;
          case 14:
            { return newSym(sym.SEMI);
            }
          case 80: break;
          case 12:
            { return newSym(sym.LEFT_BRACE);
            }
          case 81: break;
          case 35:
            { return newSym(sym.NOT_EQ);
            }
          case 82: break;
          case 45:
            { string_body += "\r";
            }
          case 83: break;
          case 3:
            { return newSym(sym.INTEGER, new Long(
                          groomDecimalOrOctal(yytext())));
            }
          case 84: break;
          case 32:
            { return newSym(sym.EQ_EQ);
            }
          case 85: break;
          case 39:
            { string_body +=
                            (char) (Integer.parseInt(yytext().substring(1), 8));
            }
          case 86: break;
          case 8:
            { /* Ignore whitespace. */
            }
          case 87: break;
          case 38:
            { comment_text += yytext();
                      if (--comment_depth == 0) {
                          yybegin(YYINITIAL);
                      }
            }
          case 88: break;
          case 15:
            { return newSym(sym.COMMA);
            }
          case 89: break;
          case 31:
            { return newSym(sym.IF);
            }
          case 90: break;
          case 24:
            { string_body += yytext();
            }
          case 91: break;
          case 26:
            { return newSym(sym.INTEGER, new Long(
                          Long.parseLong(groomDecimalOrOctal(yytext()), 8)));
            }
          case 92: break;
          case 55:
            { return newSym(sym.FALSE);
            }
          case 93: break;
          case 11:
            { return newSym(sym.RT_PAREN);
            }
          case 94: break;
          case 28:
            { return newSym(sym.OR);
            }
          case 95: break;
          case 18:
            { return newSym(sym.EQ);
            }
          case 96: break;
          case 23:
            { comment_text += yytext();
            }
          case 97: break;
          case 17:
            { return newSym(sym.RT_BRKT);
```

```java
          }
        case 98: break;
        case 58:
          { return newSym(sym.STRING);
          }
        case 99: break;
        case 7:
          { return newSym(sym.DIVIDE);
          }
        case 100: break;
        case 5:
          { return newSym(sym.DOT);
          }
        case 101: break;
        case 40:
          { string_body += "\"";
          }
        case 102: break;
        case 61:
          { return newSym(sym.BOOLEAN);
          }
        case 103: break;
        case 10:
          { return newSym(sym.LEFT_PAREN);
          }
        case 104: break;
        case 48:
          { return newSym(sym.INTEGER, new Long(
                        Long.parseLong(groomHex(yytext()), 16)));
          }
        case 105: break;
        case 43:
          { string_body += "\f";
          }
        case 106: break;
        case 34:
          { return newSym(sym.GTR_EQ);
          }
        case 107: break;
        case 49:
          { return newSym(sym.INT);
          }
        case 108: break;
        case 42:
          { string_body += "\n";
          }
        case 109: break;
        case 36:
          { return newSym(sym.AND);
          }
        case 110: break;
        case 22:
          { return newSym(sym.NOT);
          }
        case 111: break;
        case 46:
          { string_body += "\\";
          }
        case 112: break;
        case 33:
          { return newSym(sym.LESS_EQ);
          }
        case 113: break;
        case 1:
          { System.out.println("Illegal char, '" + yytext() +
                    "' line: " + yyline + ", column: " + yychar);
          }
        case 114: break;
        case 25:
          { yybegin(YYINITIAL);
                    return(newSym(sym.STRING_LIT, string_body,
                        string_start_line, string_start_column));
          }
        case 115: break;
        case 19:
          { return newSym(sym.PLUS);
          }
        case 116: break;
        case 16:
          { return newSym(sym.LEFT_BRKT);
          }
        case 117: break;
        case 20:
          { return newSym(sym.LESS);
          }
        case 118: break;
        case 52:
          { return newSym(sym.TRUE);
          }
        case 119: break;
        case 37:
          { comment_text += yytext();
                    comment_depth++;
          }
        case 120: break;
        case 9:
          { return newSym(sym.TIMES);
          }
        case 121: break;
        case 29:
          {
          }
        case 122: break;
        default:
          if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {
            zzAtEOF = true;
            zzDoEOF();
              { return new java_cup.runtime.Symbol(sym.EOF); }
          }
          else {
            zzScanError(ZZ_NO_MATCH);
          }
```

```
            }
        }
    }


}
```