```
/* 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\1\12\2\1\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];
```

```
    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"+
```

```
    "\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;

/**
```

```
337     * the number of characters from the last newline up to the start of the
338     * matched text
339     */
340    private int yycolumn;
341
342    /**
343     * zzAtBOL == true <=> the scanner is currently at the beginning of a line
344     */
345    private boolean zzAtBOL = true;
346
347    /** zzAtEOF == true <=> the scanner is at the EOF */
348    private boolean zzAtEOF;
349
350    /** denotes if the user-EOF-code has already been executed */
351    private boolean zzEOFDone;
352
353    /* user code: */
354
355  /**
356   * Return a new Symbol with the given token id, and with the current line and
357   * column numbers.
358   */
359  Symbol newSym(int tokenId) {
360      return new Symbol(tokenId, yyline, yycolumn);
361  }
362
363  /**
364   * Return a new Symbol with the given token id, the current line and column
365   * numbers, and the given token value.  The value is used for tokens such as
366   * identifiers and numbers.
367   */
368  Symbol newSym(int tokenId, Object value) {
369      return new Symbol(tokenId, yyline, yycolumn, value);
370  }
371
372  /**
373   * Overload that takes line and column as args; used for strings.
374   */
375  Symbol newSym(int tokenId, Object value, int line, int column) {
376      return new Symbol(tokenId, line, column, value);
377  }
378
379  /**
380   * Groom a decimal or octal literal sufficiently for consumption by the Long
381   * constructor.  This entails removing a trailing "l" or "L" char, if present.
382   */
383  String groomDecimalOrOctal(String decimal) {
384      return
385          ((decimal.charAt(decimal.length()-1) == 'l') ||
386           (decimal.charAt(decimal.length()-1) == 'L'))
387          ? decimal.substring(0, decimal.length()-1)
388          : decimal;
389  }
390
391  /*
392   * Groom a hex literal sufficiently for consumption by parseLong.  This entails
393   * removing the leading "0x" prefix, converting all letters to upper case, then
394   * grooming as a decimal.
395   */
396  String groomHex(String hex) {
397      return
398          groomDecimalOrOctal((hex.substring(2, hex.length())).toUpperCase());
399  }
400
401  void p(String msg) {System.out.println(msg);}
402  void pt(String msg) {System.out.println(msg + ";" + yytext());}
403
404  int comment_depth = 0;
405  int string_start_line;
406  int string_start_column;
407  String string_body = "";
408  String comment_text = "";
409
410
411
412    /**
413     * Creates a new scanner
414     * There is also a java.io.InputStream version of this constructor.
415     *
416     * @param   in  the java.io.Reader to read input from.
417     */
418    EJayLexer(java.io.Reader in) {
419      this.zzReader = in;
420    }
421
422    /**
423     * Creates a new scanner.
424     * There is also java.io.Reader version of this constructor.
425     *
426     * @param   in  the java.io.Inputstream to read input from.
427     */
428    EJayLexer(java.io.InputStream in) {
429      this(new java.io.InputStreamReader(in));
430    }
431
432
433    /**
434     * Refills the input buffer.
435     *
436     * @return      <code>false</code>, iff there was new input.
437     *
438     * @exception   java.io.IOException  if any I/O-Error occurs
439     */
440    private boolean zzRefill() throws java.io.IOException {
441
442      /* first: make room (if you can) */
443      if (zzStartRead > 0) {
444        System.arraycopy(zzBuffer, zzStartRead,
445                         zzBuffer, 0,
446                         zzEndRead-zzStartRead);
447
448        /* translate stored positions */
```

```
449         zzEndRead-= zzStartRead;
450         zzCurrentPos-= zzStartRead;
451         zzMarkedPos-= zzStartRead;
452         zzPushbackPos-= zzStartRead;
453         zzStartRead = 0;
454       }
455
456       /* is the buffer big enough? */
457       if (zzCurrentPos >= zzBuffer.length) {
458         /* if not: blow it up */
459         char newBuffer[] = new char[zzCurrentPos*2];
460         System.arraycopy(zzBuffer, 0, newBuffer, 0, zzBuffer.length);
461         zzBuffer = newBuffer;
462       }
463
464       /* finally: fill the buffer with new input */
465       int numRead = zzReader.read(zzBuffer, zzEndRead,
466                                               zzBuffer.length-zzEndRead);
467
468       if (numRead < 0) {
469         return true;
470       }
471       else {
472         zzEndRead+= numRead;
473         return false;
474       }
475     }
476
477
478     /**
479      * Closes the input stream.
480      */
481     public final void yyclose() throws java.io.IOException {
482       zzAtEOF = true;            /* indicate end of file */
483       zzEndRead = zzStartRead;  /* invalidate buffer    */
484
485       if (zzReader != null)
486         zzReader.close();
487     }
488
489
490     /**
491      * Resets the scanner to read from a new input stream.
492      * Does not close the old reader.
493      *
494      * All internal variables are reset, the old input stream
495      * <b>cannot</b> be reused (internal buffer is discarded and lost).
496      * Lexical state is set to <tt>ZZ_INITIAL</tt>.
497      *
498      * @param reader   the new input stream
499      */
500     public final void yyreset(java.io.Reader reader) {
501       zzReader = reader;
502       zzAtBOL  = true;
503       zzAtEOF  = false;
504       zzEndRead = zzStartRead = 0;
505       zzCurrentPos = zzMarkedPos = zzPushbackPos = 0;
506       yyline = yychar = yycolumn = 0;
507       zzLexicalState = YYINITIAL;
508     }
509
510
511     /**
512      * Returns the current lexical state.
513      */
514     public final int yystate() {
515       return zzLexicalState;
516     }
517
518
519     /**
520      * Enters a new lexical state
521      *
522      * @param newState the new lexical state
523      */
524     public final void yybegin(int newState) {
525       zzLexicalState = newState;
526     }
527
528
529     /**
530      * Returns the text matched by the current regular expression.
531      */
532     public final String yytext() {
533       return new String( zzBuffer, zzStartRead, zzMarkedPos-zzStartRead );
534     }
535
536
537     /**
538      * Returns the character at position <tt>pos</tt> from the
539      * matched text.
540      *
541      * It is equivalent to yytext().charAt(pos), but faster
542      *
543      * @param pos the position of the character to fetch.
544      *            A value from 0 to yylength()-1.
545      *
546      * @return the character at position pos
547      */
548     public final char yycharat(int pos) {
549       return zzBuffer[zzStartRead+pos];
550     }
551
552
553     /**
554      * Returns the length of the matched text region.
555      */
556     public final int yylength() {
557       return zzMarkedPos-zzStartRead;
558     }
559
560
```

```
561   /**
562    * Reports an error that occured while scanning.
563    *
564    * In a wellformed scanner (no or only correct usage of
565    * yypushback(int) and a match-all fallback rule) this method
566    * will only be called with things that "Can't Possibly Happen".
567    * If this method is called, something is seriously wrong
568    * (e.g. a JFlex bug producing a faulty scanner etc.).
569    *
570    * Usual syntax/scanner level error handling should be done
571    * in error fallback rules.
572    *
573    * @param   errorCode  the code of the errormessage to display
574    */
575   private void zzScanError(int errorCode) {
576     String message;
577     try {
578       message = ZZ_ERROR_MSG[errorCode];
579     }
580     catch (ArrayIndexOutOfBoundsException e) {
581       message = ZZ_ERROR_MSG[ZZ_UNKNOWN_ERROR];
582     }
583
584     throw new Error(message);
585   }
586
587
588   /**
589    * Pushes the specified amount of characters back into the input stream.
590    *
591    * They will be read again by then next call of the scanning method
592    *
593    * @param number  the number of characters to be read again.
594    *                This number must not be greater than yylength()!
595    */
596   public void yypushback(int number)  {
597     if ( number > yylength() )
598       zzScanError(ZZ_PUSHBACK_2BIG);
599
600     zzMarkedPos -= number;
601   }
602
603
604   /**
605    * Contains user EOF-code, which will be executed exactly once,
606    * when the end of file is reached
607    */
608   private void zzDoEOF() throws java.io.IOException {
609     if (!zzEOFDone) {
610       zzEOFDone = true;
611       yyclose();
612     }
613   }
614
615
616   /**
617    * Resumes scanning until the next regular expression is matched,
618    * the end of input is encountered or an I/O-Error occurs.
619    *
620    * @return      the next token
621    * @exception   java.io.IOException  if any I/O-Error occurs
622    */
623   public java_cup.runtime.Symbol next_token() throws java.io.IOException {
624     int zzInput;
625     int zzAction;
626
627     // cached fields:
628     int zzCurrentPosL;
629     int zzMarkedPosL;
630     int zzEndReadL = zzEndRead;
631     char [] zzBufferL = zzBuffer;
632     char [] zzCMapL = ZZ_CMAP;
633
634     int [] zzTransL = ZZ_TRANS;
635     int [] zzRowMapL = ZZ_ROWMAP;
636     int [] zzAttrL = ZZ_ATTRIBUTE;
637
638     while (true) {
639       zzMarkedPosL = zzMarkedPos;
640
641       boolean zzR = false;
642       for (zzCurrentPosL = zzStartRead; zzCurrentPosL < zzMarkedPosL;
643                                                              zzCurrentPosL++) {
644         switch (zzBufferL[zzCurrentPosL]) {
645         case '\u000B':
646         case '\u000C':
647         case '\u0085':
648         case '\u2028':
649         case '\u2029':
650           yyline++;
651           yycolumn = 0;
652           zzR = false;
653           break;
654         case '\r':
655           yyline++;
656           yycolumn = 0;
657           zzR = true;
658           break;
659         case '\n':
660           if (zzR)
661             zzR = false;
662           else {
663             yyline++;
664             yycolumn = 0;
665           }
666           break;
667         default:
668           zzR = false;
669           yycolumn++;
670         }
671       }
672
```

```
673       if (zzR) {
674         // peek one character ahead if it is \n (if we have counted one line too much)
675         boolean zzPeek;
676         if (zzMarkedPosL < zzEndReadL)
677           zzPeek = zzBufferL[zzMarkedPosL] == '\n';
678         else if (zzAtEOF)
679           zzPeek = false;
680         else {
681           boolean eof = zzRefill();
682           zzEndReadL = zzEndRead;
683           zzMarkedPosL = zzMarkedPos;
684           zzBufferL = zzBuffer;
685           if (eof)
686             zzPeek = false;
687           else
688             zzPeek = zzBufferL[zzMarkedPosL] == '\n';
689         }
690         if (zzPeek) yyline--;
691       }
692       zzAction = -1;
693
694       zzCurrentPosL = zzCurrentPos = zzStartRead = zzMarkedPosL;
695
696       zzState = zzLexicalState;
697
698
699       zzForAction: {
700         while (true) {
701
702           if (zzCurrentPosL < zzEndReadL)
703             zzInput = zzBufferL[zzCurrentPosL++];
704           else if (zzAtEOF) {
705             zzInput = YYEOF;
706             break zzForAction;
707           }
708           else {
709             // store back cached positions
710             zzCurrentPos  = zzCurrentPosL;
711             zzMarkedPos   = zzMarkedPosL;
712             boolean eof = zzRefill();
713             // get translated positions and possibly new buffer
714             zzCurrentPosL  = zzCurrentPos;
715             zzMarkedPosL   = zzMarkedPos;
716             zzBufferL      = zzBuffer;
717             zzEndReadL     = zzEndRead;
718             if (eof) {
719               zzInput = YYEOF;
720               break zzForAction;
721             }
722             else {
723               zzInput = zzBufferL[zzCurrentPosL++];
724             }
725           }
726           int zzNext = zzTransL[ zzRowMapL[zzState] + zzCMapL[zzInput] ];
727           if (zzNext == -1) break zzForAction;
728           zzState = zzNext;
729
730           int zzAttributes = zzAttrL[zzState];
731           if ( (zzAttributes & 1) == 1 ) {
732             zzAction = zzState;
733             zzMarkedPosL = zzCurrentPosL;
734             if ( (zzAttributes & 8) == 8 ) break zzForAction;
735           }
736
737         }
738       }
739
740       // store back cached position
741       zzMarkedPos = zzMarkedPosL;
742
743       switch (zzAction < 0 ? zzAction : ZZ_ACTION[zzAction]) {
744         case 60:
745           { return newSym(sym.RETURN);
746           }
747         case 62: break;
748         case 47:
749           { string_body += "\'";
750           }
751         case 63: break;
752         case 53:
753           { return newSym(sym.VOID);
754           }
755         case 64: break;
756         case 21:
757           { return newSym(sym.GTR);
758           }
759         case 65: break;
760         case 56:
761           { return newSym(sym.WHILE);
762           }
763         case 66: break;
764         case 51:
765           { return newSym(sym.ELSE);
766           }
767         case 67: break;
768         case 13:
769           { return newSym(sym.RT_BRACE);
770           }
771         case 68: break;
772         case 44:
773           { string_body += "\t";
774           }
775         case 69: break;
776         case 27:
777           { return newSym(sym.FLOATING_PT, new Double(yytext()));
778           }
779         case 70: break;
780         case 30:
781           { comment_depth++; comment_text = "/*"; yybegin(COMMENT);
782           }
783         case 71: break;
784         case 54:
```

```
785           { return newSym(sym.FLOAT);
786           }
787         case 72: break;
788         case 41:
789           { string_body += "\b";
790           }
791         case 73: break;
792         case 2:
793           { return newSym(sym.IDENT, yytext());
794           }
795         case 74: break;
796         case 59:
797           { return newSym(sym.STRUCT);
798           }
799         case 75: break;
800         case 50:
801           { return newSym(sym.REF);
802           }
803         case 76: break;
804         case 4:
805           { return newSym(sym.MINUS);
806           }
807         case 77: break;
808         case 57:
809           { return newSym(sym.PRINT);
810           }
811         case 78: break;
812         case 6:
813           { string_body = ""; string_start_line = yyline;
814                     string_start_column = yycolumn;  yybegin(STRING);
815           }
816         case 79: break;
817         case 14:
818           { return newSym(sym.SEMI);
819           }
820         case 80: break;
821         case 12:
822           { return newSym(sym.LEFT_BRACE);
823           }
824         case 81: break;
825         case 35:
826           { return newSym(sym.NOT_EQ);
827           }
828         case 82: break;
829         case 45:
830           { string_body += "\r";
831           }
832         case 83: break;
833         case 3:
834           { return newSym(sym.INTEGER, new Long(
835                       groomDecimalOrOctal(yytext())));
836           }
837         case 84: break;
838         case 32:
839           { return newSym(sym.EQ_EQ);
840           }
841         case 85: break;
842         case 39:
843           { string_body +=
844                       (char) (Integer.parseInt(yytext().substring(1), 8));
845           }
846         case 86: break;
847         case 8:
848           { /* Ignore whitespace. */
849           }
850         case 87: break;
851         case 38:
852           { comment_text += yytext();
853                     if (--comment_depth == 0) {
854                         yybegin(YYINITIAL);
855                     }
856           }
857         case 88: break;
858         case 15:
859           { return newSym(sym.COMMA);
860           }
861         case 89: break;
862         case 31:
863           { return newSym(sym.IF);
864           }
865         case 90: break;
866         case 24:
867           { string_body += yytext();
868           }
869         case 91: break;
870         case 26:
871           { return newSym(sym.INTEGER, new Long(
872                       Long.parseLong(groomDecimalOrOctal(yytext()), 8)));
873           }
874         case 92: break;
875         case 55:
876           { return newSym(sym.FALSE);
877           }
878         case 93: break;
879         case 11:
880           { return newSym(sym.RT_PAREN);
881           }
882         case 94: break;
883         case 28:
884           { return newSym(sym.OR);
885           }
886         case 95: break;
887         case 18:
888           { return newSym(sym.EQ);
889           }
890         case 96: break;
891         case 23:
892           { comment_text += yytext();
893           }
894         case 97: break;
895         case 17:
896           { return newSym(sym.RT_BRKT);
```

```
          }
        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);
          }
```

```
1009          }
1010        }
1011      }
1012
1013
1014    }
```