```
/*-***
 *
 * This file defines a stand-alone lexical analyzer for a subset of the Pascal
 * programming language.  This is the same lexer that will later be integrated
 * with a CUP-based parser.  Here the lexer is driven by the simple Java test
 * program in ./PascalLexerTest.java, q.v.  See 330 Lecture Notes 2 and the
 * Assignment 2 writeup for further discussion.
 *
 */


import java_cup.runtime.*;


%%
/*-*
 * LEXICAL FUNCTIONS:
 */

%cup
%line
%column
%class PascalLexer

%{

/**
 * 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);
}

%}


/*-*
 * PATTERN DEFINITIONS:
 */
letter          = [A-Za-z]
digit           = [0-9]
alphanumeric    = {letter}|{digit}
other_id_char   = [_]
identifier      = {letter}({alphanumeric}|{other_id_char})*
integer         = {digit}*
real            = {integer}\.{integer}
char            = '.'
leftbrace       = \{
rightbrace      = \}
nonrightbrace   = [^}]
comment_body    = {nonrightbrace}*
comment         = {leftbrace}{comment_body}{rightbrace}
whitespace      = [ \n\t]


%%
/**
 * LEXICAL RULES:
 */
and             { return newSym(sym.AND); }
array           { return newSym(sym.ARRAY); }
begin           { return newSym(sym.BEGIN); }
else            { return newSym(sym.ELSE); }
end             { return newSym(sym.END); }
if              { return newSym(sym.IF); }
of              { return newSym(sym.OF); }
not             { return newSym(sym.NOT); }
or              { return newSym(sym.OR); }
program         { return newSym(sym.PROGRAM); }
procedure       { return newSym(sym.PROCEDURE); }
then            { return newSym(sym.THEN); }
type            { return newSym(sym.TYPE); }
var             { return newSym(sym.VAR); }
"*"             { return newSym(sym.TIMES); }
"+"             { return newSym(sym.PLUS); }
"-"             { return newSym(sym.MINUS); }
"/"             { return newSym(sym.DIVIDE); }
";"             { return newSym(sym.SEMI); }
","             { return newSym(sym.COMMA); }
"("             { return newSym(sym.LEFT_PAREN); }
")"             { return newSym(sym.RT_PAREN); }
"["             { return newSym(sym.LEFT_BRKT); }
"]"             { return newSym(sym.RT_BRKT); }
"="             { return newSym(sym.EQ); }
"<"             { return newSym(sym.GTR); }
">"             { return newSym(sym.LESS); }
"<="            { return newSym(sym.LESS_EQ); }
">="            { return newSym(sym.GTR_EQ); }
"!="            { return newSym(sym.NOT_EQ); }
":"             { return newSym(sym.COLON); }
":="            { return newSym(sym.ASSMNT); }
"."             { return newSym(sym.DOT); }
{identifier}    { return newSym(sym.IDENT, yytext()); }
{integer}       { return newSym(sym.INT, new Integer(yytext())); }
{real}          { return newSym(sym.REAL, new Double(yytext())); }
{char}          { return newSym(sym.CHAR, new Character(yytext().charAt(1))); }
{comment}       { }
{whitespace}    { /* Ignore whitespace. */ }
.               { System.out.println("Illegal char, line: " +
                    yyline + ", column: " + yychar); }
```