pascal.lex [plain text]

```
/*
 * pascal.lex: An example PASCAL scanner
 *
 */

%{
#include <stdio.h>
#include "y.tab.h"

int line_number = 0;

void yyerror(char *message);

%}

%x COMMENT1 COMMENT2

white_space       [ \t]*
digit             [0-9]
alpha             [A-Za-z_]
alpha_num         ({alpha}|{digit})
hex_digit         [0-9A-F]
identifier        {alpha}{alpha_num}*
unsigned_integer  {digit}+
hex_integer       ${hex_digit}{hex_digit}*
exponent          e[+-]?{digit}+
i                 {unsigned_integer}
real              ({i}\.{i}?|{i}?\.{i}){exponent}?
string            \'([^'\n]|\'\')+\'
bad_string        \'([^'\n]|\'\')+

%%

"{"                  BEGIN(COMMENT1);
<COMMENT1>[^}\n]+
<COMMENT1>\n            ++line_number;
<COMMENT1><<EOF>>    yyerror("EOF in comment");
<COMMENT1>"}"        BEGIN(INITIAL);

"(*"                 BEGIN(COMMENT2);
<COMMENT2>[^)*\n]+
<COMMENT2>\n            ++line_number;
<COMMENT2><<EOF>>    yyerror("EOF in comment");
<COMMENT2>"*)"       BEGIN(INITIAL);
<COMMENT2>[*)]

 /* note that FILE and BEGIN are already
  * defined in FLEX or C so they can't
  * be used. This can be overcome in
  * a cleaner way by defining all the
  * tokens to start with TOK_ or some
  * other prefix.
  */

and                  return(AND);
array                return(ARRAY);
begin                return(_BEGIN);
case                 return(CASE);
const                return(CONST);
div                  return(DIV);
do                   return(DO);
downto               return(DOWNTO);
else                 return(ELSE);
end                  return(END);
file                 return(_FILE);
```

```
for                     return(FOR);
function                return(FUNCTION);
goto                    return(GOTO);
if                      return(IF);
in                      return(IN);
label                   return(LABEL);
mod                     return(MOD);
nil                     return(NIL);
not                     return(NOT);
of                      return(OF);
packed                  return(PACKED);
procedure               return(PROCEDURE);
program                 return(PROGRAM);
record                  return(RECORD);
repeat                  return(REPEAT);
set                     return(SET);
then                    return(THEN);
to                      return(TO);
type                    return(TYPE);
until                   return(UNTIL);
var                     return(VAR);
while                   return(WHILE);
with                    return(WITH);

"<="|"=<"               return(LEQ);
"=>"|">="               return(GEQ);
"<>"                    return(NEQ);
"="                     return(EQ);

".."                    return(DOUBLEDOT);

{unsigned_integer}      return(UNSIGNED_INTEGER);
{real}                  return(REAL);
{hex_integer}           return(HEX_INTEGER);
{string}                return{STRING};
{bad_string}            yyerror("Unterminated string");

{identifier}            return(IDENTIFIER);

[*/+\-,^.;:()\[\]]      return(yytext[0]);

{white_space}           /* do nothing */
\n                      line_number += 1;
.                       yyerror("Illegal input");

%%

void yyerror(char *message)
{
   fprintf(stderr,"Error: \"%s\" in line %d. Token = %s\n",
           message,line_number,yytext);
   exit(1);
}
```