Código LEX en C

D [0-9]

L [a-zA-Z\_]

H [a-fA-F0-9]

E [Ee][+-]?{D}+

FS (f|F|l|L)

IS (u|U|l|L)\*

%{

#include <stdio.h>

#include "y.tab.h"

void count();

%}

%%

"/\*" { comment(); }

"auto" { count(); return(AUTO); }

"break" { count(); return(BREAK); }

"case" { count(); return(CASE); }

"char" { count(); return(CHAR); }

"const" { count(); return(CONST); }

"continue" { count(); return(CONTINUE); }

"default" { count(); return(DEFAULT); }

"do" { count(); return(DO); }

"double" { count(); return(DOUBLE); }

"else" { count(); return(ELSE); }

"enum" { count(); return(ENUM); }

"extern" { count(); return(EXTERN); }

"float" { count(); return(FLOAT); }

"for" { count(); return(FOR); }

"goto" { count(); return(GOTO); }

"if" { count(); return(IF); }

"int" { count(); return(INT); }

"long" { count(); return(LONG); }

"register" { count(); return(REGISTER); }

"return" { count(); return(RETURN); }

"short" { count(); return(SHORT); }

"signed" { count(); return(SIGNED); }

"sizeof" { count(); return(SIZEOF); }

"static" { count(); return(STATIC); }

"struct" { count(); return(STRUCT); }

"switch" { count(); return(SWITCH); }

"typedef" { count(); return(TYPEDEF); }

"union" { count(); return(UNION); }

"unsigned" { count(); return(UNSIGNED); }

"void" { count(); return(VOID); }

"volatile" { count(); return(VOLATILE); }

"while" { count(); return(WHILE); }

{L}({L}|{D})\* { count(); return(check\_type()); }

0[xX]{H}+{IS}? { count(); return(CONSTANT); }

0{D}+{IS}? { count(); return(CONSTANT); }

{D}+{IS}? { count(); return(CONSTANT); }

L?'(\\.|[^\\'])+' { count(); return(CONSTANT); }

{D}+{E}{FS}? { count(); return(CONSTANT); }

{D}\*"."{D}+({E})?{FS}? { count(); return(CONSTANT); }

{D}+"."{D}\*({E})?{FS}? { count(); return(CONSTANT); }

L?\"(\\.|[^\\"])\*\" { count(); return(STRING\_LITERAL); }

"..." { count(); return(ELLIPSIS); }

">>=" { count(); return(RIGHT\_ASSIGN); }

"<<=" { count(); return(LEFT\_ASSIGN); }

"+=" { count(); return(ADD\_ASSIGN); }

"-=" { count(); return(SUB\_ASSIGN); }

"\*=" { count(); return(MUL\_ASSIGN); }

"/=" { count(); return(DIV\_ASSIGN); }

"%=" { count(); return(MOD\_ASSIGN); }

"&=" { count(); return(AND\_ASSIGN); }

"^=" { count(); return(XOR\_ASSIGN); }

"|=" { count(); return(OR\_ASSIGN); }

">>" { count(); return(RIGHT\_OP); }

"<<" { count(); return(LEFT\_OP); }

"++" { count(); return(INC\_OP); }

"--" { count(); return(DEC\_OP); }

"->" { count(); return(PTR\_OP); }

"&&" { count(); return(AND\_OP); }

"||" { count(); return(OR\_OP); }

"<=" { count(); return(LE\_OP); }

">=" { count(); return(GE\_OP); }

"==" { count(); return(EQ\_OP); }

"!=" { count(); return(NE\_OP); }

";" { count(); return(';'); }

("{"|"<%") { count(); return('{'); }

("}"|"%>") { count(); return('}'); }

"," { count(); return(','); }

":" { count(); return(':'); }

"=" { count(); return('='); }

"(" { count(); return('('); }

")" { count(); return(')'); }

("["|"<:") { count(); return('['); }

("]"|":>") { count(); return(']'); }

"." { count(); return('.'); }

"&" { count(); return('&'); }

"!" { count(); return('!'); }

"~" { count(); return('~'); }

"-" { count(); return('-'); }

"+" { count(); return('+'); }

"\*" { count(); return('\*'); }

"/" { count(); return('/'); }

"%" { count(); return('%'); }

"<" { count(); return('<'); }

">" { count(); return('>'); }

"^" { count(); return('^'); }

"|" { count(); return('|'); }

"?" { count(); return('?'); }

[ \t\v\n\f] { count(); }

. { /\* ignorar los caracteres incorrectos \*/ }

%%

yywrap()

{

return(1);

}

comment()

{

char c, c1;

loop:

while ((c = input()) != '\*' && c != 0)

putchar(c);

if ((c1 = input()) != '/' && c != 0)

{

unput(c1);

goto loop;

}

if (c != 0)

putchar(c1);

}

int column = 0;

void count()

{

int i;

for (i = 0; yytext[i] != '\0'; i++)

if (yytext[i] == '\n')

column = 0;

else if (yytext[i] == '\t')

column += 8 - (column % 8);

else

column++;

ECHO;

}

int check\_type()

{

/\*

\* pseudo código --- esto es lo que debería verificar

\*

\* if (yytext == type\_name)

\* return(TYPE\_NAME);

\*

\* return(IDENTIFIER);

\*/

/\*

\* En realidad solo devolverá el IDENTIFICADOR

\*/

return(IDENTIFIER);

}