```
%{
/*
Vahe Karamian - CS 440 - Project 1
Filename: project1.l, project1.h, project1.c
*/

#include <stdio.h>
#include <math.h>
#include "proj2.h"

/* this function I use for my symbol table */
void SymbolTable( char* id );

/* global index variable used by the symbol table */
int  vk_index = 0;

/* symbol table data structure */
char **symbolTable[100];

/* declare enum for the keywords and etc... */
/*
enum{IF=10001,ELSE,WHILE,BREAK,RETURN,VOID,INT};
enum{ADDOP=20001,SUBOP,MULOP,DIVOP};
enum{LTOP=30001,GTOP,EQOP,LEOP,GEOP,NEOP,ASOP};
enum{SEMI=40001,COMA};
enum{LPAREN=50001,RPAREN,LBRACE,RBRACE,LBRAKT,RBRAKT};
*/
enum{ BREAK = 9504 };
%}

/* digit, letter, comment */
DIGIT [0-9]
LETTER   [A-Za-z]
%x     COMMENT

%%

/* begining of comment section detection */
"/*" { BEGIN COMMENT; }    /* enter comment eating state */
"/*".*"*/"[ \t]*\n{ ; } /* self contained comment */

<COMMENT>"*/"[ \t]*\n  { BEGIN 0; }
<COMMENT>"*/"        { BEGIN 0; }
<COMMENT>\n          { ; }
<COMMENT>.\n         { ; }
/* end of comment section detection */

/* begining of keyword detection */
if     { printf("(t_if [%d])", IF ); return(IF); }
else   { printf("(t_else [%d])", ELSE ); return(ELSE); }
while  { printf("(t_while [%d])", WHILE ); return(WHILE); }
break { printf("(t_break [%d])", BREAK ); }
return { printf("(t_return [%d])", RETURN ); return(RETURN); }
void{ printf("(t_void [%d])", VOID ); return(VOID); }
int      { printf("(t_int [%d])", INT ); return(INT); }
/* end of keyword detection */

/* begining of operator detection */
"+"  { printf("(t_ADD %d)", ADDOP ); return(ADDOP); }
```

```
"-"   { printf("(t_SUB %d)", ADDOP ); return(ADDOP); }
"/"   { printf("(t_DIV %d)", MULOP ); return(MULOP); }
"*"   { printf("(t_MUL %d)", MULOP ); return(MULOP); }
/* end of operator detection */

/* begining of relational operator detection */
"<"  { printf("(t_LT %d)", RELOP ); return(RELOP); }
">"  { printf("(t_GT %d)", RELOP ); return(RELOP); }

"=="   { printf("(t_EQ %d)", RELOP ); return(RELOP); }
"<="   { printf("(t_LE %d)", RELOP ); return(RELOP); }
">="   { printf("(t_GE %d)", RELOP ); return(RELOP); }
"!=" { printf("(t_NE %d)", RELOP ); return(RELOP); }

"="  { printf("(t_AS %d)", RELOP ); return(RELOP); }
/* enf od relational operation detection */

";"   { printf("(t_semi %d)\n", SEMICOLON ); return(SEMICOLON); }
","   { printf("(t_coma %d)", COMMA ); return(COMMA); }

"("   { printf("(t_lparen %d)", OPAR ); return(OPAR); }
")"   { printf("(t_rparen %d)", CPAR ); return(CPAR); }
"{"   { printf("(t_lbrace %d)\n", OBRAC ); return(OBRAC); }
"}"   { printf("(t_rbrace %d)\n", CBRAC ); return(CBRAC); }
"["   { printf("(t_lbrakt %d)", OBRAK ); return(OBRAK); }
"]"   { printf("(t_rbrakt %d)", CBRAK ); return(CBRAK); }

/* detecting digits */
{DIGIT}+              { printf("(t_num %s)", yytext ); return(NUM); }

/* detecting floats */
{DIGIT}+"."{DIGIT}* { printf("(t_float %s)", yytext ); return(NUM); }

/* detecting the identifiers */
{LETTER}({LETTER}|{DIGIT})* { printf("(t_id %s)", yytext ); return(ID); }

[ \t\n]   ; /* eat up white spaces */

/* unrecognized character */
.                   printf("(\nt_error: %s\n)", yytext );

%%

/* symbol table function */

void SymbolTable( char* id )
{
  symbolTable[vk_index] = id;
  printf("(t_id {%s} L[%i])", symbolTable[vk_index], vk_index );
  vk_index++;
}
```