```
/*
 * Paul Heidelman
 * Case Western Reserve University
 * EECS 337 Fall 2009
 * Homework 2
 */


%{
 #include "linked_list_hw2.h"
 /* definitions of manifest constants
 LT, LE, EQ, NE, GT, GE,
 IF, THEN, ELSE, ID, NUMBER, RELOP */
 #define LT 1
 #define LE 2
 #define EQ 3
 #define NE 4
 #define GT 5
 #define GE 6
 #define IF 7
 #define THEN 8
 #define ELSE 9
 #define ID 10
 #define NUMBER 11
 #define RELOP 12
 #define CO 13
 #define BO 14
 #define BC 15
 #define WHILE 16
 #define DO 17
 #define BREAK 18
 #define BASIC 19
 #define TRUE 20
 #define FALSE 21

 void* yylval;
 int int_var;
 int float_var;
 int true_var;
 int false_var;
 int line_commented;
 int block_commented;
 list_head_t *id_list;

%}

/* regular definitions */
delim [\t\n]
ws {delim}+
letter [A-Za-z]
digit [0-9]
id {letter}({letter}|{digit})*
number [+-]?{digit}+
real [+-]?{digit}+(\.{digit}+)?(E[+-]?{digit}+)?
comment (("//".*\n)|("/*".*"*/"))
modifier const|extern|signed|static|unsigned|volatile
custom struct|enum|union|typedef
basic   int|float
type double|long|short|void|char|bool
tf (TRUE|true)|(FALSE|false)

%%

{ws} {/* no action and no return */printf("%s", yytext);}
if {printf("if"); return(RELOP);}
then {printf("then"); return(RELOP);}
else {printf("else"); return(RELOP);}
```

```
while {printf("while"); return(RELOP);}
do {printf("do"); return(RELOP);}
for {printf("for"); return(RELOP);}
break {printf("break"); return(RELOP);}
continue {printf("continue"); return(RELOP);}
switch {printf("switch"); return(RELOP);}
case {printf("case"); return(RELOP);}
default {printf("default"); return(RELOP);}
goto {printf("goto"); return(RELOP);}
return {printf("return"); return(RELOP);}
{modifier} {printf("modifier");}
{custom} {printf("custom");}
{type} {printf("type");}
{comment} {printf("comment");}
{basic} {printf("basic"); if(strcmp(yytext, "int") == 0) yylval = (void*) &int_var; else yylval = (void*)
&float_var; return(RELOP);}
{tf} {if(strcmp(yytext, "true") == 0){printf("true"); yylval = (void*) &true_var;} else {printf("false");
yylval = (void*) &false_var;} return(RELOP);}
{id} {yylval = (int) installID(); printf("<id, %s>", yytext); return(RELOP);}
{number} {printf("<num, %s>", yytext); int* n = malloc(sizeof(atoi(yytext))); *n = atoi(yytext); yylval =
(void*)n; return(RELOP);}
{real} {printf("<real, %s>", yytext); int* n = malloc(sizeof(atoi(yytext))); *n = atoi(yytext); yylval =
(void*)n; return(RELOP);}
"<" {printf("<"); return(RELOP) ;}
"+" {printf("+"); return(RELOP) ;}
"-" {printf("-"); return(RELOP) ;}
"<=" {printf("<="); return(RELOP) ;}
"=" {printf("="); return(RELOP) ;}
"<>" {printf("<>"); return(RELOP) ;}
">" {printf(">"); return(RELOP) ;}
">=" {printf(">="); return(RELOP) ;}
"!=" {printf("!="); return(RELOP) ;}
"==" {printf("=="); return(RELOP) ;}
"&" {printf("&"); return(RELOP) ;}
"&&" {printf("&&"); return(RELOP) ;}
"||" {printf("||"); return(RELOP) ;}
"|" {printf("|"); return(RELOP) ;}
"*" {printf("*"); return(RELOP) ;}
"/" {printf("/"); return(RELOP) ;}
"^" {printf("^"); return(RELOP) ;}
"!" {printf("!"); return(RELOP) ;}
"+=" {printf("+="); return(RELOP) ;}
"-=" {printf("-="); return(RELOP) ;}
"*=" {printf("*="); return(RELOP) ;}
"/=" {printf("/="); return(RELOP) ;}
"++" {printf("++"); return(RELOP) ;}
"--" {printf("--"); return(RELOP) ;}

%%

int installID() {
 /* function to install the lexeme, whose
 first character is pointed to by yytext,
 and whose length is yyleng, into the
 symbol table and return a pointer
 thereto */
 char *str1 = malloc(sizeof(yytext));
 strcpy(str1, yytext);
 //printf("|%s|", str1);
 //printf("%s", list_search(id_list, yytext));
 if(list_search(id_list, yytext) == NULL)
 {
  list_insert(id_list, (void*)str1, NULL);
 }
 return &str1;
}
```

```
int yywrap(){
 return 1;
}

main( argc, argv )
int argc;
char **argv;
{

 id_list = list_init(cmp_string);

 ++argv, --argc; /* skip over program name */
 if ( argc > 0 )
 {
  yyin = fopen( argv[0], "r" );
  printf("\nUsing file %s\n", argv[0]);
 }
 else
  yyin = stdin;

 while(yylex() != 0)
  yylex();


 printf("id's:\n");
 list_entry_t *cur;
 cur = id_list->list;
 while (cur != NULL)
 {
  printf("%s\n",(char*)cur->key);
  cur = cur->next;
 }

 list_delete(id_list);

 return 0;
}
```