Lex program:

%{

#include<stdio.h>

#include<stdlib.h>

struct data{

int val;

char \*code;

char \*var;

};

#include"y.tab.h"

extern YYSTYPE yylval;

void yyerror(char\*);

%}

%%

[-+]?[0-9]+ {

yylval.val=atoi(yytext);

return NUMBER;

}

[-+]?[0-9]\*[.][0-9]+ {

yylval.val=atoi(yytext);

return NUMBER;

}

for {return FOR;}

while {return WHILE;}

if {return IF;}

then {return THEN;}

else {return ELSE;}

end\_if {return ENDIF;}

integer {return INT;}

real {return REAL;}

char {return CHAR;}

begin {return BEG;}

end {return END;}

var {return VAR;}

[a-zA-Z\_]([a-zA-z\_]|[0-9])\* {yylval.info.var=(char\*)malloc(10);strcpy(yylval.info.var,yytext);return ID;}

[{};()] {return \*yytext;}

[-+\*/^()=&|%:] {return \*yytext;}

"<"|">" {return \*yytext;}

">=" {return GTE;}

"<=" {return LTE;}

"!=" {return NE;}

"==" {return EQ;}

"&&" {return AND;}

"||" {return OR;}

"!" {return NOT;}

"<<" {return LS;}

">>" {return RS;}

[\t] ;

[\n] ;

[ ] ;

. {yyerror("invalid case");}

%%

int yywrap(void)

{

return 1;

}

Yacc Program:

%{

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

struct SymTable

{

char \*var;

int val;

char \*type;

}s\_table[20];

int yylex(void);

void yyerror(char \*str);

void find(char \*var);

int label=1,t=1,count=0;

void disp();

void find(char \*var)

{

int j=0;

for(j=0;j<count;j++)

{

if(!strcmp(var,s\_table[j].var))

return;

}

char err[100];

sprintf(err,"%s not found\n",var);

yyerror(err);

exit(0);

return;

}

void disp()

{

int j=0;

printf("\tSYMBOL TABLE\n");

printf("Name Type Value\n");

for(j=0;j<count;j++)

{

printf("%-10s %-10s %-10d\n",s\_table[j].var,s\_table[j].type,s\_table[j].val);

}

}

struct data{

int val;

char \*code;

char \*var;

};

%}

%token NUMBER OR AND NOT GTE LTE NE EQ LS RS FOR IF ELSE ID WHILE INT REAL CHAR BEG END THEN ENDIF VAR

%union{

struct data info;

int val;

char \*code;

}

%type<code> S BLOCK BODY SA;

%type<info> E C ID

%type<val> NUMBER

%right '='

%left OR

%left AND

%left NOT

%left '|'

%left '&'

%left EQ NE

%left '<' '>' GTE LTE

%left LS RS

%left '\*' '/'

%right '+' '-'

%left '%'

%left '^'

%left '(' ')'

%%

S:DECL BEG BLOCK END {printf("BEGIN\n%sEND\n",$3);disp();return 0;};

DECL:DECL DECL

|VAR ID':' INT'='NUMBER';' {s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"INT");s\_table[count++].val=$6;}

|VAR ID':' REAL'='NUMBER';'{s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"REAL");s\_table[count++].val=$6;}

|VAR ID':' REAL';' {s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"REAL");s\_table[count++].val=0;}

|VAR ID':' INT';' {s\_table[count].var=(char\*)malloc(10);s\_table[count].type=(char\*)malloc(10);strcpy(s\_table[count].var,$2.var);strcpy(s\_table[count].type,"INT");s\_table[count++].val=0;}

|

;

BLOCK:'{'BODY'}' {$$=(char\*)malloc(2000);sprintf($$,"%s",$2);}

|BODY {$$=(char\*)malloc(2000);sprintf($$,"%s",$1);}

;

BODY:BODY BODY {$$=(char\*)malloc(2000);sprintf($$,"%s%s",$1,$2);}

|SA';' {$$=(char\*)malloc(2000);sprintf($$,"%s\n",$1);}

|IF'('C')' THEN BLOCK ELSE BLOCK ENDIF {$$=(char\*)malloc(2000);sprintf($$," if %s goto L%d\n goto L%d\nL%d:\n%s goto L%d\nL%d:\n%sL%d:\n",$3.code,label,label+1,label,$6,label+2,label+1,$8,label+2);label+=3;}

|IF'('C')' THEN BLOCK ENDIF {$$=(char\*)malloc(2000);sprintf($$," if %s goto L%d\n goto L%d\nL%d:\n%sL%d:\n",$3.code,label,label+1,label,$6,label+1);label+=2;}

|BLOCK {$$=(char\*)malloc(2000);sprintf($$,"%s",$1);}

| {$$=(char\*)malloc(2000);strcpy($$,"");}

;

SA:ID'='E {find($1.var);$$=(char\*)malloc(2000); sprintf($$,"%s %s=%s",$3.code,$1.var,$3.var);}

|ID'+''+' {find($1.var);$$=(char\*)malloc(2000);sprintf($$," %s++",$1.var);}

|ID'-''-' {find($1.var);$$=(char\*)malloc(2000);sprintf($$," %s--",$1.var);}

;

E:E'+'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s + %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'-'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s - %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'\*'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s \* %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'/'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s / %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'%'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s mod %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'^'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s ^ %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|'('E')' {$$.var=(char\*)malloc(3);sprintf($$.var,"%s",$2.var);$$.code=(char\*)malloc(300);sprintf($$.code,"%s\n",$2.code);}

|E'&'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s & %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E'|'E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s | %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E LS E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s << %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

|E RS E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s%s %s = %s >> %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}

| '-' E {$$.var=(char\*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s %s=-%s\n",$2.code,$$.var,$2.var);}

|E'+''+'{$$.var=(char\*)malloc(3);sprintf($$.var,"%s",$1.var);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s %s++\n",$1.code,$1.var);}

|E'-''-'{$$.var=(char\*)malloc(3);sprintf($$.var,"%s",$1.var);t+=1;$$.code=(char\*)malloc(300);sprintf($$.code,"%s %s--\n",$1.code,$1.var);}

|NUMBER {$$.var=(char\*)malloc(10);sprintf($$.var,"%d",$1);$$.code=(char\*)malloc(1);strcpy($$.code,"");}

|ID {find($1.var);$$.var=(char\*)malloc(10);sprintf($$.var,"%s",$1.var);$$.code=(char\*)malloc(1);strcpy($$.code,"");}

;

C:NOT E {$$.code=(char\*)malloc(300);sprintf($$.code,"!%s",$2.var);}

|E'<'E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s < %s",$1.var,$3.var);}

|E'>'E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s > %s",$1.var,$3.var);}

|E GTE E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s >= %s",$1.var,$3.var);}

|E LTE E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s <= %s",$1.var,$3.var);}

|E EQ E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s == %s",$1.var,$3.var);}

|E NE E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s != %s",$1.var,$3.var);}

|E AND E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s && %s",$1.var,$3.var);}

|E OR E {$$.code=(char\*)malloc(300);sprintf($$.code,"%s || %s",$1.var,$3.var);}

%%

void yyerror(char \*str)

{

fprintf(stderr,"%s\n",str);

}

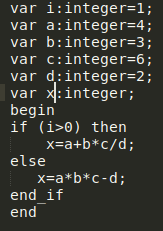
void main()

{

yyparse();

}

INPUT SCREENSHOT:



OUTPUT SCREENSHOT:

