INDEX

1. Program to convert infix to postfix using Lex and Yacc

2. Program to implement symbol table

3. Program to implement Calculator using Lex and Yacc

4. Program to implement lexical analyser for C program

5. Program to implement Parser for C statements

6. Program to generate 3 address code for C statements

**1.Program to convert infix to postfix using Lex and Yacc**

**(Lex Program: inpost.l)**

%{

#include<stdio.h>

#include "y.tab.h"

%}

%%

[a-zA-Z] { strcpy(yylval.s,yytext);

return ID;

}

[+\*-/] { return yytext[0];}

[0-9] { return yytext[0];}

[\n] { return 0;}

"("|")" { return yytext[0];}

. {;}

%%

**(Yacc Program: ift.y)**

%{

#include <stdio.h>

#include <string.h>

void yyerror(const char \*str)

{

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

}

int yywrap()

{

return 1;

}

main()

{

yyparse();

}

%}

%left '+' '-'

%left '\*' '/'

%nonassoc UMINUS

%union {

char s[20];

}

%token <s> ID NUM

%type <s> E

%%

S:E {printf("%s\n",$1);}

;

E:E'+'E {

strcat($1,$3);

strcat($1,"+");

strcpy($$,$1);

}

|E'\*'E {

strcat($1,$3);

strcat($1,"\*");

strcpy($$,$1);

}

|E'/'E { strcat($1,$3);

strcat($1,"/");

strcpy($$,$1);

}

|E'-'E {

strcat($1,$3);

strcat($1,"-");

strcpy($$,$1);

}

|'-'E %prec UMINUS {

strcat($2,"UMINUS");

strcpy($$,$2);

}

|ID {strcpy($$,$1);}

|NUM {strcpy($$,$1);}

|'('E')' {strcpy($$,$2);}

;

%%

/\*void main()

{

yyparse();

}\*/

/\*yyerror() {

printf("Error.");

}\*/

**OUTPUT**

((a+b)\*c)/d-e

ab+c\*d/e-

**2.Program to implement symbol table**

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

#include<string.h>

#include<stdlib.h>#define NULL 0

int size=0;

void Insert();

void Display();

void Delete();

int Search(char lab[]);

void Modify();

struct SymbTab

{

char label[10],symbol[10];

int addr;

struct SymbTab \*next;};

struct SymbTab \*first,\*last;

void main()

{

int op,y;

char la[10];

clrscr();

do

{

printf("\n\tSYMBOL TABLE IMPLEMENTATION\n");

printf("\n\t1.INSERT\n\t2.DISPLAY\n\t3.DELETE\n\t4.SEARCH\n\t5.MODIFY\n\t6.END\n");

printf("\n\tEnter your option : ");

scanf("%d",&op);

switch(op)

{

case 1:

Insert();

break;

case 2:

Display();

break;

case 3:

Delete();

break;

case 4:

printf("\n\tEnter the label to be searched : ");

scanf("%s",la);

y=Search(la);

printf("\n\tSearch Result:");

if(y==1)

printf("\n\tThe label is present in the symbol table\n");

else

printf("\n\tThe label is not present in the symbol table\n");

break;

case 5:

Modify();

break;

case 6:

exit(0);

}

}while(op<6);

getch();

}

void Insert()

{

int n;

char l[10];

printf("\n\tEnter the label : ");

scanf("%s",l);

n=Search(l);

if(n==1)

printf("\n\tThe label exists already in the symbol table\n\tDuplicate can't be inserted");

else

{

struct SymbTab \*p;

p=malloc(sizeof(struct SymbTab));

strcpy(p->label,l);

printf("\n\tEnter the symbol : ");

scanf("%s",p->symbol);

printf("\n\tEnter the address : ");

scanf("%d",&p->addr);

p->next=NULL;

if(size==0)

{

first=p;

last=p;

}

else

{

last->next=p;

last=p;

}

size++;

}

printf("\n\tLabel inserted\n");

}

void Display()

{

int i;

struct SymbTab \*p;

p=first;

printf("\n\tLABEL\t\tSYMBOL\t\tADDRESS\n");

for(i=0;i<size;i++)

{

printf("\t%s\t\t%s\t\t%d\n",p->label,p->symbol,p->addr);

p=p->next;

}

}

int Search(char lab[])

{

int i,flag=0;

struct SymbTab \*p;

p=first;

for(i=0;i<size;i++)

{

if(strcmp(p->label,lab)==0)

flag=1;

p=p->next;

}

return flag;

}

void Modify()

{

char l[10],nl[10];

int add,choice,i,s;

struct SymbTab \*p;

p=first;

printf("\n\tWhat do you want to modify?\n");

printf("\n\t1.Only the label\n\t2.Only the address\n\t3.Both the label and address\n");

printf("\tEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\n\tEnter the old label : ");

scanf("%s",l);

s=Search(l);

if(s==0)

printf("\n\tLabel not found\n");

else

{

printf("\n\tEnter the new label : ");

scanf("%s",nl);

for(i=0;i<size;i++)

{

if(strcmp(p->label,l)==0)

strcpy(p->label,nl);

p=p->next;

}

printf("\n\tAfter Modification:\n");

Display();

}

break;

case 2:

printf("\n\tEnter the label where the address is to be modified : ");

scanf("%s",l);

s=Search(l);

if(s==0)

printf("\n\tLabel not found\n");

else

{

printf("\n\tEnter the new address : ");

scanf("%d",&add);

for(i=0;i<size;i++)

{

if(strcmp(p->label,l)==0)

p->addr=add;

p=p->next;

}

printf("\n\tAfter Modification:\n");

Display();

}

break;

case 3:

printf("\n\tEnter the old label : ");

scanf("%s",l);

s=Search(l);

if(s==0)

printf("\n\tLabel not found\n");

else

{

printf("\n\tEnter the new label : ");

scanf("%s",nl);

printf("\n\tEnter the new address : ");

scanf("%d",&add);

for(i=0;i<size;i++)

{

if(strcmp(p->label,l)==0)

{

strcpy(p->label,nl);

p->addr=add;

}

p=p->next;

}

printf("\n\tAfter Modification:\n");

Display();

}

break;

}

}

void Delete()

{

int a;

char l[10];

struct SymbTab \*p,\*q;

p=first;

printf("\n\tEnter the label to be deleted : ");

scanf("%s",l);

a=Search(l);

if(a==0)

printf("\n\tLabel not found\n");

else

{

if(strcmp(first->label,l)==0)

first=first->next;

else if(strcmp(last->label,l)==0)

{

q=p->next;

while(strcmp(q->label,l)!=0)

{

p=p->next;

q=q->next;

}

p->next=NULL;

last=p;

}

else

{

q=p->next;

while(strcmp(q->label,l)!=0)

{

p=p->next;

q=q->next;

}

p->next=q->next;

}

size--;

printf("\n\tAfter Deletion:\n");

Display();

}

**OUTPUT**

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option : 1

Enter the label : if

Enter the symbol : IF

Enter the address : 102

Label inserted

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option : 2

LABEL SYMBOL ADDRESS

if IF 102

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option : 1

Enter the label : FOR

Enter the symbol : for

Enter the address : 103

Label inserted

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option : 2

LABEL SYMBOL ADDRESS

if IF 102

FOR for 103

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option : 4

Enter the label to be searched : if

Search Result:

The label is present in the symbol table

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option : 5

What do you want to modify?

1.Only the label

2.Only the address

3.Both the label and address

Enter your choice : 1

Enter the old label : if

Enter the new label : If

After Modification:

LABEL SYMBOL ADDRESS

If IF 102

FOR for 103

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option : 3

Enter the label to be deleted : If

After Deletion:

LABEL SYMBOL ADDRESS

FOR for 103

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

3.DELETE

4.SEARCH

5.MODIFY

6.END

Enter your option : 6

**3. Program to implement Calculator using Lex and Yacc**

**(calc.l)**

%{

#include<math.h>

#include"y.tab.h" //for left,right,up & down

%}

%%

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

yylval.p = atof(yytext);

return num; //return nonterminal

}

sin {return SIN;} //return token SIN to YACC

cos {return COS;} //return token COS to YACC

tan return TAN; //return token TAN to YACC

log return LOG; //return token LOG to YACC

sqrt return SQRT; //return token SQRT to YACC

[\t];

\n return 0;

. return yytext[0];

%%

**(calc.y)**

%{

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

%}

%union //to define possible symbol types

{ double p;}

%token<p>num

%token SIN COS TAN LOG SQRT

/\*Defining the Precedence and Associativity\*/

%left '+' '-' //lowest precedence

%left '\*' '/' //highest precedenc

%nonassoc uminu //no associativity

%type<p>exp //Sets the type for non - terminal

%%

/\* for storing the answer \*/

ss: exp {printf("=%g\n",$1);}

/\* for binary arithmatic operators \*/

exp : exp'+'exp { $$=$1+$3; }

|exp'-'exp { $$=$1-$3; }

|exp'\*'exp { $$=$1\*$3; }

|exp'/'exp {

if($3==0)

{

printf("Divide By Zero");

exit(0);

}

else $$=$1/$3;

}

|'-'exp {$$=-$2;}

|'('exp')' {$$=$2;}

|SIN'('exp')' {$$=sin($3);}

|COS'('exp')' {$$=cos($3);}

|TAN'('exp')' {$$=tan($3);}

|LOG'('exp')' {$$=log($3);}

|SQRT'('exp')' {$$=sqrt($3);}

|num;

%%

/\* extern FILE \*yyin; \*/

main()

{

do

{

yyparse(); /\* repeatedly tries to parse the sentence until the i/p runs out \*/

}while(1);

}

yyerror(s) /\* used to print the error message when an error is parsing of i/p \*/

char \*s;

{

printf("ERROR");

}

**OUTPUT**

3+2

=5

sin(3.14)

=0.00159265

tan(3.14)

=-0.00159265

log(10)

=2.30259

4\*8/5

=6.4

**4.Program to implement lexical analyser for C program**

letter [a-zA-Z]

digit[0-9]

%%

{digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\tis real number",yytext);

{digit}+"."{digit}+("E"("+"|"-")?{digit}+)? printf("\n%s\t is floating pt no ",yytext);

"if"|"else"|"int"|"char"|"scanf"|"printf"|"switch"|"return"|"struct"|"do"|"while"|"void"|"for"|"float" printf("\n%s\t is keywords",yytext);

"\a"|"\\n"|"\\b"|"\t"|"\\t"|"\b"|"\\a" printf("\n%s\tis Escape sequences",yytext);

{letter}({letter}|{digit})\* printf("\n%s\tis identifier",yytext);

"&&"|"<"|">"|"<="|">="|"="|"+"|"-"|"?"|"\*"|"/"|"%"|"&"|"||" printf("\n%s\toperator ",yytext);

"{"|"}"|"["|"]"|"("|")"|"#"|"'"|"."|"\""|"\\"|";"|"," printf("\n%s\t is a special character",yytext);

"%d"|"%s"|"%c"|"%f"|"%e" printf("\n%s\tis a format specifier",yytext);

%%

int yywrap()

{

return 1;

}

int main(int argc,char \*argv[])

{

yyin=fopen(argv[1],"r");

yylex();

fclose(yyin);

return 0;

}

**OUTPUT**

/ operator

/ operator

INPUT is identifier

PROGRAM is identifier

# is a special character

include is identifier

< operator

stdio is identifier

. is a special character

h is identifier

> operator

void is keywords

main is identifier

( is a special character

) is a special character

{ is a special character

pringtf is identifier

( is a special character

" is a special character

\n is Escape sequences

hai is identifier

\n is Escape sequences

" is a special character

) is a special character

; is a special character

} is a special character

**5.Program to implement Parser for C statements**

**---->Parser for IF-THEN-ELSE Statement   
  
(Lex Program: ift.l)**  
alpha [A-Za-z]  
digit [0-9]  
%%  
[ \t\n]      
if    return IF;  
then    return THEN;  
else    return ELSE;  
{digit}+    return NUM;  
{alpha}({alpha}|{digit})\*    return ID;  
"<="    return LE;  
">="    return GE;  
"=="    return EQ;  
"!="    return NE;  
"||"    return OR;  
"&&"    return AND;  
.    return yytext[0];  
%%  
  
**(Yacc Program: ift.y)**  
%{  
#include <stdio.h>  
#include <stdlib.h>  
%}  
%token ID NUM IF THEN LE GE EQ NE OR AND ELSE  
%right '='  
%left AND OR  
%left '<' '>' LE GE EQ NE  
%left '+''-'  
%left '\*''/'  
%right UMINUS  
%left '!'  
%%  
  
S      : ST {printf("Input accepted.\n");exit(0);};  
ST    : IF '(' E2 ')' THEN ST1';' ELSE ST1';'  
        | IF '(' E2 ')' THEN ST1';'  
        ;  
ST1  : ST  
        | E  
        ;  
E    : ID'='E  
      | E'+'E  
      | E'-'E  
      | E'\*'E  
      | E'/'E  
      | E'<'E  
      | E'>'E  
      | E LE E  
      | E GE E  
      | E EQ E  
      | E NE E  
      | E OR E  
      | E AND E  
      | ID  
      | NUM  
      ;  
E2  : E'<'E  
      | E'>'E  
      | E LE E  
      | E GE E  
      | E EQ E  
      | E NE E  
      | E OR E  
      | E AND E  
      | ID  
      | NUM  
      ;  
  
%%  
  
#include "lex.yy.c"  
  
main()  
{  
  printf("Enter the exp: ");  
  yyparse();  
}  
          
**Output**:  
  
Enter the exp: if(a==1)  then b=1; else b=2;  
Input accepted.

### --->Parser for FOR Loop Statements (Lex file: for.l) alpha [A-Za-z] digit [0-9] %% [\t \n] for             return FOR; {digit}+    return NUM; {alpha}({alpha}|{digit})\* return ID; "<="         return LE; ">="         return GE; "=="         return EQ; "!="          return NE; "||"            return OR; "&&"         return AND; .                return yytext[0]; %% ( Yacc file: for.y) %{ #include <stdio.h> #include <stdlib.h> %} %token ID NUM FOR LE GE EQ NE OR AND %right "=" %left OR AND %left '>' '<' LE GE EQ NE %left '+' '-' %left '\*' '/' %right UMINUS %left '!' %%     S         : ST {printf("Input accepted\n"); exit(0);} ST       : FOR '(' E ';' E2 ';' E ')' DEF            ; DEF    : '{' BODY '}'            | E';'            | ST            |            ; BODY  : BODY BODY            | E ';'                   | ST            |                        ;         E        : ID '=' E           | E '+' E           | E '-' E           | E '\*' E           | E '/' E           | E '<' E           | E '>' E           | E LE E           | E GE E           | E EQ E           | E NE E           | E OR E           | E AND E           | E '+' '+'           | E '-' '-'           | ID            | NUM           ;     E2     : E'<'E          | E'>'E          | E LE E          | E GE E          | E EQ E          | E NE E          | E OR E          | E AND E          ;    %% #include "lex.yy.c" main() {     printf("Enter the expression:\n");     yyparse(); }            Output: Enter the expression: for(i=0;i<n;i++) i=i+1; Input accepted

### --->Parser for WHILE Statements

### (wh.l) alpha [A-Za-z] digit [0-9] %% [ \t\n]   while    return WHILE; {digit}+    return NUM; {alpha}({alpha}|{digit})\*    return ID; "<="    return LE; ">="    return GE; "=="    return EQ; "!="    return NE; "||"    return OR; "&&"    return AND; .    return yytext[0]; %% (wh.y) %{ #include <stdio.h> #include <stdlib.h> %} %token ID NUM WHILE LE GE EQ NE OR AND %right '=' %left AND OR %left '<' '>' LE GE EQ NE %left '+''-' %left '\*''/' %right UMINUS %left '!' %% S        : ST1 {printf("Input accepted.\n");exit(0);}; ST1    :    WHILE'(' E2 ')' '{' ST '}' ST      :     ST ST           | E';'           ; E       : ID'='E           | E'+'E           | E'-'E           | E'\*'E           | E'/'E           | E'<'E           | E'>'E           | E LE E           | E GE E           | E EQ E           | E NE E           | E OR E           | E AND E           | ID           | NUM           ; E2     : E'<'E           | E'>'E           | E LE E           | E GE E           | E EQ E           | E NE E           | E OR E           | E AND E           | ID           | NUM           ; %% #include "lex.yy.c" main() {    printf("Enter the exp: ");    yyparse(); } Output: Enter the exp: while(a>1){ b=1;} Input accepted.

**6. Program to generate 3 address code for C statements**

### --->Intermediate Code Generator for If then else (lex Program : intif.l) ALPHA [A-Za-z] DIGIT [0-9] %% if                 return IF; then                 return THEN; else                 return ELSE; {ALPHA}({ALPHA}|{DIGIT})\*    return ID; {DIGIT}+             {yylval=atoi(yytext); return NUM;} [ \t]                 ; \n                yyterminate(); .                 return yytext[0]; %% (Yacc Program : intif.y) %token ID NUM IF THEN ELSE %right '=' %left '+' '-' %left '\*' '/' %left UMINUS %% S : IF '(' E ')'{lab1();} THEN E ';'{lab2();} ELSE E ';'{lab3();}   ; E :V '='{push();} E{codegen\_assign();}   | E '+'{push();} E{codegen();}   | E '-'{push();} E{codegen();}   | E '\*'{push();} E{codegen();}   | E '/'{push();} E{codegen();}   | '(' E ')'   | '-'{push();} E{codegen\_umin();} %prec UMINUS   | V   | NUM{push();}   ; V : ID {push();}   ; %% #include "lex.yy.c" #include<ctype.h> char st[100][10]; int top=0; char i\_[2]="0"; char temp[2]="t"; int label[20]; int lnum=0; int ltop=0; main()  {  printf("Enter the expression : ");  yyparse();  } push()  {   strcpy(st[++top],yytext);  } codegen()  {  strcpy(temp,"t");  strcat(temp,i\_);   printf("%s = %s %s %s\n",temp,st[top-2],st[top-1],st[top]);   top-=2;  strcpy(st[top],temp);  i\_[0]++;  } codegen\_umin()  {  strcpy(temp,"t");  strcat(temp,i\_);  printf("%s = -%s\n",temp,st[top]);  top--;  strcpy(st[top],temp);  i\_[0]++;  } codegen\_assign()  {  printf("%s = %s\n",st[top-2],st[top]);  top-=2;  } lab1() {  lnum++;  strcpy(temp,"t");  strcat(temp,i\_);  printf("%s = not %s\n",temp,st[top]);  printf("if %s goto L%d\n",temp,lnum);  i\_[0]++;  label[++ltop]=lnum; } lab2() { int x; lnum++; x=label[ltop--]; printf("goto L%d\n",lnum); printf("L%d: \n",x); label[++ltop]=lnum; } lab3() { int y; y=label[ltop--]; printf("L%d: \n",y); } Output: Enter the expression : if(k+8) then k=18;else c=s; t0 = k + 8 t1 = not t0 if t1 goto L1 k = 18 goto L2 L1: c = s L2:

### --->Intermediate Code Generator for For Loop // Lex file: im4.l alpha [A-Za-z] digit [0-9] %% [\t \n] for             return FOR; {digit}+    return NUM; {alpha}({alpha}|{digit})\* return ID; "<="         return LE; ">="         return GE; "=="         return EQ; "!="          return NE; "||"            return OR; "&&"         return AND; .                return yytext[0]; %% // Yacc file: im4.y %{ #include <stdio.h> #include <stdlib.h> %} %token ID NUM FOR LE GE EQ NE OR AND %right "=" %left OR AND %left '>' '<' LE GE EQ NE %left '+' '-' %left '\*' '/' %right UMINUS %left '!' %% S       : FOR '(' E ';'{lab1();} E {lab2();}';' E {lab3();}')' E';'{lab4(); exit(0);}          ; E       : V '='{push();} E{codegen\_assign();}          | E '+'{push();} E{codegen();}          | E '-'{push();} E{codegen();}          | E '\*'{push();} E{codegen();}          | E '/'{push();} E{codegen();}          | '(' E ')'          | '-'{push();} E{codegen\_umin();} %prec UMINUS          | V          | NUM{push();}          ; V       : ID {push();}          ; %% #include "lex.yy.c" #include<ctype.h> char st[100][10]; int label[20]; int top=0; char i\_[2]="0"; char temp[2]="t"; int lno=0,ltop=0; int start=1; main() {     printf("Enter the expression:\n");     yyparse(); } push() {    strcpy(st[++top],yytext); } codegen() {     strcpy(temp,"t");     strcat(temp,i\_);     printf("%s = %s %s %s\n",temp,st[top-2],st[top-1],st[top]);     top-=2;     strcpy(st[top],temp);     i\_[0]++; } codegen\_umin() {     strcpy(temp,"t");     strcat(temp,i\_);     printf("%s = -%s\n",temp,st[top]);     top--;     strcpy(st[top],temp);     i\_[0]++; } codegen\_assign() {     printf("%s = %s\n",st[top-2],st[top]);     top-=2; } lab1() {     printf("L%d: \n",lno++); } lab2() {     strcpy(temp,"t");     strcat(temp,i\_);     printf("%s = not %s\n",temp,st[top]);     printf("if %s goto L%d\n",temp,lno);     i\_[0]++;     label[++ltop]=lno;     lno++;     printf("goto L%d\n",lno);     label[++ltop]=lno;     printf("L%d: \n",++lno);  } lab3() {     int x;     x=label[ltop--];     printf("goto L%d \n",start);     printf("L%d: \n",x); } lab4() {     int x;     x=label[ltop--];     printf("goto L%d \n",lno);        printf("L%d: \n",x); } Output: Enter the expression: for(i=0;i=b;i=i+1) a=a+b; i = 0 L0: i = b t0 = not i if t0 goto L1 goto L2 L3: t1 = i + 1 i = t1 goto L0 L2: t2 = a + b a = t2 goto L3 L1:

### --->Intermediate Code Generator for While

(Lex Program: intwh.l)  
ALPHA [A-Za-z]  
DIGIT [0-9]  
%%  
while                return WHILE;  
{ALPHA}({ALPHA}|{DIGIT})\*    return ID;  
{DIGIT}+             {yylval=atoi(yytext); return NUM;}  
[ \t]                 ;  
\n                yyterminate();  
.                 return yytext[0];  
%%  
 **(Yacc Program: intwh.y)**  
%token ID NUM WHILE  
%right '='  
%left '+' '-'  
%left '\*' '/'  
%left UMINUS  
%%  
  
S : WHILE{lab1();} '(' E ')'{lab2();} E ';'{lab3();}  
  ;  
E :V '='{push();} E{codegen\_assign();}  
  | E '+'{push();} E{codegen();}  
  | E '-'{push();} E{codegen();}  
  | E '\*'{push();} E{codegen();}  
  | E '/'{push();} E{codegen();}  
  | '(' E ')'  
  | '-'{push();} E{codegen\_umin();} %prec UMINUS  
  | V  
  | NUM{push();}  
  ;  
V : ID {push();}  
  ;  
%%  
  
#include "lex.yy.c"  
#include<ctype.h>  
char st[100][10];  
int top=0;  
char i\_[2]="0";  
char temp[2]="t";  
  
int lnum=1;  
int start=1;  
main()  
 {  
 printf("Enter the expression : ");  
 yyparse();  
 }  
  
  
  
push()  
 {  
  strcpy(st[++top],yytext);  
 }  
  
codegen()  
 {  
 strcpy(temp,"t");  
 strcat(temp,i\_);  
  printf("%s = %s %s %s\n",temp,st[top-2],st[top-1],st[top]);  
  top-=2;  
 strcpy(st[top],temp);  
 i\_[0]++;  
 }  
  
codegen\_umin()  
 {  
 strcpy(temp,"t");  
 strcat(temp,i\_);  
 printf("%s = -%s\n",temp,st[top]);  
 top--;  
 strcpy(st[top],temp);  
 i\_[0]++;  
 }  
  
codegen\_assign()  
 {  
 printf("%s = %s\n",st[top-2],st[top]);  
 top-=2;  
 }  
lab1()  
{  
printf("L%d: \n",lnum++);  
}  
lab2()  
{  
 strcpy(temp,"t");  
 strcat(temp,i\_);  
 printf("%s = not %s\n",temp,st[top]);  
 printf("if %s goto L%d\n",temp,lnum);  
 i\_[0]++;  
 }  
lab3()  
{  
printf("goto L%d \n",start);  
printf("L%d: \n",lnum);  
}

**Output :**  
  
Enter the expression : while(k=c/s)k=k\*c+d;  
L1:   
t0 = c / s  
k = t0  
t1 = not k  
if t1 goto L0  
t2 = k \* c  
t3 = t2 + d  
k = t3  
goto L1   
L0: