Ex.No:01

CONSTRUCTION OF NFA FROM REGULAR EXPRESSSION

Program:

```
#include<stdio.h>
#include<string.h>
int main(){
        char reg[20]; int q[20][3],i=0,j=1,len,a,b;
        printf("Enter the regular expression\n");
        for(a=0;a<20;a++) for(b=0;b<3;b++) q[a][b]=0;
        scanf("%s",reg);
        printf("Given regular expression: %s\n",reg);
        len=strlen(reg);
        while(i<len){
                if(reg[i]=='a'&&reg[i+1]!='|'&&reg[i+1]!='*') { q[j][0]=j+1; j++; }
                if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='*') {     q[j][1]=j+1; j++; }
                if(reg[i]=='e'&&reg[i+1]!='|'&&reg[i+1]!='*') {     q[j][2]=j+1; j++; }
                if(reg[i]=='a'&&reg[i+1]=='|'&&reg[i+2]=='b') {
                  q[j][2]=((j+1)*10)+(j+3); j++;
                  q[j][0]=j+1; j++;
                        q[j][2]=j+3; j++;
                        q[j][1]=j+1; j++;
                        q[j][2]=j+1; j++;
                        i=i+2;
                }
                if(reg[i]=='b'&&reg[i+1]=='|'&&reg[i+2]=='a'){
                        q[j][2]=((j+1)*10)+(j+3); j++;
                        q[j][1]=j+1; j++;
                        q[j][2]=j+3; j++;
                        q[j][0]=j+1; j++;
                        q[j][2]=j+1; j++;
                        i=i+2;}
```

```
            if(reg[i]=='a'&&reg[i+1]=='*'){
                q[j][2]=((j+1)*10)+(j+3); j++;
                q[j][0]=j+1; j++;
                q[j][2]=((j+1)*10)+(j-1); j++;
            }
            if(reg[i]=='b'&&reg[i+1]=='*'){
                q[j][2]=((j+1)*10)+(j+3); j++;
                q[j][1]=j+1; j++;
                q[j][2]=((j+1)*10)+(j-1); j++;
            }
            if(reg[i]==')'&&reg[i+1]=='*'){
                q[0][2]=((j+1)*10)+1;
                q[j][2]=((j+1)*10)+1;
                j++;
            } i++;
        }
        printf("\n\tTransition Table \n");
        printf("_____________________________________________\n");
        printf("Current State |\tInput |\tNext State");
        printf("\n_____________________________________________\n");
        for(i=0;i<=j;i++){
            if(q[i][0]!=0) printf("\n  q[%d]\t     |   a   |  q[%d]",i,q[i][0]);
            if(q[i][1]!=0) printf("\n  q[%d]\t     |   b   |  q[%d]",i,q[i][1]);
            if(q[i][2]!=0) {
                if(q[i][2]<10) printf("\n  q[%d]\t     |   e   |  q[%d]",i,q[i][2]);
                else printf("\n  q[%d]\t     |   e   |  q[%d] , q[%d]",i,q[i][2]/10,q[i][2]%10);
            }}
        printf("\n_____________________________________________\n");
        return 0;
 getch();
}
```

OUTPUT:

Enter the regular expression
(a|b)*

Given regular expression: (a|b)*

Transition Table

Current State	Input	Next State
q[0]	e	q[7] , q[1]
q[1]	e	q[2] , q[4]
q[2]	a	q[3]
q[3]	e	q[6]
q[4]	b	q[5]
q[5]	e	q[6]
q[6]	e	q[7] , q[1]

EX.No:02

Construction of minimized DFA from a given regular Expression

```
#include<stdio.h>
#include<string.h>
#define STATES 50
struct Dstate
{
char name;
char StateString[STATES+1];
char trans[10];
int is_final;
}Dstates[50];
struct tran
{
char sym;
int tostates[50];
int notran;
};
struct state
{
int no;
struct tran tranlist[50];
};
int stackA[100],stackB[100],c[100],Cptr=-1,Aptr=-1,Bptr=-1;
struct state States[10];
char temp[STATES+1],inp[10];
int nos,noi,nof,j,k,nods=-1;
void pushA(int z)
{
stackA[++Aptr]=z;
}
void pushB(int z)
{
stackB[++Bptr]=z;
```

```
}
int popA()
{
return stackA[Aptr--];
}
void copy(int i)
{
char temp[STATES+1]=" ";
int k=0;
Bptr=-1;
strcpy(temp,Dstates[i].StateString);
while(temp[k]!='\0')
{
pushB(temp[k]-'0');
k++;
}
}
int popB()
{
return stackB[Bptr--];
}
int peekA()
{
return stackA[Aptr];
}
int peekB()
{
return stackA[Bptr];
}
int seek(int arr[],int ptr,int s)
{
int i;
for(i=0;i<=ptr;i++)
```

```
{
if(s==arr[i])
return 1;
}
return 0;
}
void sort()
{
int i,j,temp;
for(i=0;i<Bptr;i++)
{
for(j=0;j<(Bptr-i);j++)
{
if(stackB[j]>stackB[j+1])
{
temp=stackB[j];
stackB[j]=stackB[j+1];
stackB[j+1]=temp;
}
}
  }
    }
void tostring()
{
int i=0;
sort();
for(i=0;i<=Bptr;i++)
{
temp[i]=stackB[i]+'0';
}
temp[i]='\0';
}
void display_DTran()
```

```
{
int i,j;
printf("\n\t\t DFA transition table");
printf("\n\t\t ---------------------------------------------- ");
printf("\n States \tString \tInputs\n");
for(i=0;i<noi;i++)
{
printf("\t %c",inp[i]);
}
printf("\n\t ------------------------------------------------- ");
for(i=0;i<nods;i++)
{
if(Dstates[i].is_final==0)
printf("\n%c",Dstates[i].name);
else
printf("\n*%c",Dstates[i].name);
printf("\t%s",Dstates[i].StateString);
for(j=0;j<noi;j++)
{
printf("\t%c",Dstates[i].trans[j]);
}
    }
     printf("\n");
}
void move(int st,int j)
{
int ctr=0;
while(ctr<States[st].tranlist[j].notran)
{
pushA(States[st].tranlist[j].tostates[ctr++]);
}
}
void lambda_closure(int st)
```

```
{
int ctr=0,in_state=st,curst=st,chk;
while(Aptr!=-1)
{
curst=popA();
ctr=0;
in_state=curst;
while(ctr<=States[curst].tranlist[noi].notran)
{
chk=seek(stackB,Bptr,in_state);
if(chk==0)
pushB(in_state);
in_state=States[curst].tranlist[noi].tostates[ctr++];
chk=seek(stackA,Aptr,in_state);
if(chk==0 && ctr<=States[curst].tranlist[noi].notran)
pushA(in_state);
}
    }
}
void main()
{
int i,final[20],start,fin=0;
char c,ans,st[20];
printf("\n Enter no of states in NFA:");
scanf("%d",&nos);
for(i=0;i<nos;i++)
{
States[i].no=i;
}printf("\n Enter the start states:");
scanf("%d",&start);
printf("Enter the no of final states:");
scanf("%d",&nof);
printf("Enter the final states:\n");
```

```
for(i=0;i<nof;i++)
scanf("%d",&final[i]);
printf("\n Enter the no of input symbols:");
scanf("%d",&noi);
c=getchar();
printf("Enter the input symbols:\n");
for(i=0;i<noi;i++)
{
scanf("%c",&inp[i]);
c=getchar();
}
int[i]=’e’;
printf("\n Enter the transitions:(-1 to stop)\n");
for(i=0;i<nos;i++)
{
for(j=0;j<=noi;j++)
{
States[i].tranlist[j].sym=inp[j];
k=0;
ans='y';
while(ans=='y')
{
printf("move(%d,%c);",i,inp[j]);
scanf("%d",&States[i].tranlist[j].tostates[k++]);
if((States[i].tranlist[j].tostates[k-1]==-1))
{
k--;
ans='n';
break;
}
}
States[i].tranlist[j].notran=k;
}
```

```
}
i=0;nods=0,fin=0;
pushA(start);
lambda_closure(peekA());
tostring();
Dstates[nods].name='A';
nods++;
strcpy(Dstates[0].StateString,temp);
while(i<nods)
{
for(j=0;j<noi;j++)
{
fin=0;
copy(i);
while(Bptr!=-1)
{
move(popB(),j);
}
while(Aptr!=-1)
lambda_closure(peekA());
tostring();
for(k=0;k<nods;k++)
{
if((strcmp(temp,Dstates[k].StateString)==0))
{
Dstates[i].trans[j]=Dstates[k].name;
break;
}
}
if(k==nods)
{
nods++;
for(k=0;k<nof;k++)
```

```
{
fin=seek(stackB,Bptr,final[k]);
if(fin==1)
{
Dstates[nods-1].is_final=1;
break;
}
    }
strcpy(Dstates[nods-1].StateString,temp);
Dstates[nods-1].name='A'+nods-1;
Dstates[i].trans[j]=Dstates[nods-1].name;
}
}
i++;
}
display_DTran();
}
```

OUTPUT:

```
Enter the no of input symbols:2
Enter the input symbols:
a ,b
Enter the transitions:(-1 to stop)
move(0,a);-1
move(0,b);-1
move(0,e);1
move(0,e);7
move(0,e);-1
move(1,a);-1
move(1,b);-1
move(1,e);2
move(1,e);4
move(1,e);-1
```

move(2,a);3

move(2,a);3

move(2,a);-1

move(2,b);-1

move(2,e);-1

move(3,a);-1

move(3,b);-1

move(3,e);6

move(3,e);-1

move(4,a);-1

move(4,b);-1

move(4,e);-1

move(5,a);-1

move(5,b);-1

move(5,e);6

move(5,e);1

move(5,e);-1

move(6,a);-1

move(6,b);-1

move(6,e);-1

move(7,a);-1

move(7,b);-1

move(7,e);-1

Ex.No:03

IMPLEMENTATION OF SYMBOL TABLE

Program:

```
//Implementation of symbol table
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
void main()
{
 int i=0,j=0,x=0,n;
 void *p,*add[5];
 char ch,srch,b[15],d[15],c;
 printf("Expression terminated by $:");
 while((c=getchar())!='$')
 {
  b[i]=c;
  i++;
 }
 n=i-1;
 printf("Given Expression:");
 i=0;
 while(i<=n)
 {
  printf("%c",b[i]);
  i++;
 }
 printf("\n Symbol Table\n");
 printf("Symbol \t addr \t type");
 while(j<=n)
```

```
{
 c=b[j];
 if(isalpha(toascii(c)))
 {
  p=malloc(c);
  add[x]=p;
  d[x]=c;
  printf("\n%c \t %d \t identifier\n",c,p);
  x++;
  j++;
 }
 else
 {
  ch=c;
  if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
  {
   p=malloc(ch);
   add[x]=p;
   d[x]=ch;
   printf("\n %c \t %d \t operator\n",ch,p);
   x++;
   j++;
  }
}
}
}
```

OUTPUT:

```
l2sys29@l2sys29-Veriton-M275: ~/Desktop/syedvirus
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./exp1_symtab
Expression terminated by $:A+B+C=D$
Given Expression:A+B+C=D
 Symbol Table
Symbol    addr     type
A         25731088         identifier

 +        25731168         operator

B         25731232         identifier

 +        25731312         operator

C         25731376         identifier

 =        25731456         operator

D         25731536         identifier
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$
```

Ex.No:04

Develop a lexical analyzer to recognize a few patterns in C.

(Ex. identifiers, constants, comments, operators etc.)

Program:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h> void main()
{
FILE *fi,*fo,*fop,*fk; int flag=0,i=1;
char c,t,a[15],ch[15],file[20]; clrscr();
printf("\n Enter the File Name:"); scanf("%s",&file);
fi=fopen(file,"r"); fo=fopen("inter.c","w");
fop=fopen("Oper.c","r");
fk=fopen("key.c","r"); c=getc(fi); while(!feof(fi))
{
if(isalpha(c)||isdigit(c)||(c=='['||c==']'||c=='.'==1)) fputc(c,fo);
else
{
if(c=='\n') fprintf(fo,"\t$\t");
else fprintf(fo,"\t%c\t",c);
}
c=getc(fi);
}
fclose(fi); fclose(fo);
fi=fopen("inter.c","r"); printf("\n Lexical Analysis"); fscanf(fi,"%s",a);
printf("\n Line: %d\n",i++); while(!feof(fi))
{
if(strcmp(a,"$")==0)
{
printf("\n Line: %d \n",i++); fscanf(fi,"%s",a);
```

```
}
fscanf(fop,"%s",ch);
while(!feof(fop))
{
if(strcmp(ch,a)==0)
{
fscanf(fop,"%s",ch); printf("\t\t%s\t:\t%s\n",a,ch); flag=1;
}
fscanf(fop,"%s",ch);
}
rewind(fop); fscanf(fk,"%s",ch);
while(!feof(fk))
{
if(strcmp(ch,a)==0)
{
fscanf(fk,"%k",ch); printf("\t\t%s\t:\tKeyword\n",a); flag=1;
}
fscanf(fk,"%s",ch);
}
rewind(fk); if(flag==0)
{
if(isdigit(a[0])) printf("\t\t%s\t:\tConstant\n",a);
else
printf("\t\t%s\t:\tIdentifier\n",a);
}
flag=0; fscanf(fi,"%s",a);
}
getch();
}
```

Key.C

int void main char if

for

while else printf scanf FILE

include stdio.h conio.h iostream.h

Oper.C

(open para

) closepara

{ openbrace

} closebrace

< lesser

> greater

" doublequote ' singlequote

: colon

; semicolon

preprocessor

= equal

== asign

% percentage

^ bitwise

& reference

* star

+ add

- sub

\ backslash

/ slash

INPUT.C

```
#include "stdio.h"

#include "conio.h" void main()

{

int a=10,b,c; a=b*c; getch();

}
```

OUTPUT:

Line:1

: preprocessor include : Identifier " : doublequote stdio.h : Keyword " :

doublequote

Line: 2

: preprocessor include : Identifier " : doublequote conio.h : Keyword " :

doublequote

Line: 3

void : Keyword main : Keyword (: open

) : closepara

Line: 4

{ : openbrace

Line: 5

int : Keyword a : Identifier

= : equal

10 : Constant

, : Identifier b : Identifier

, : Identifier c : Identifier

; : semicolon

Line: 6

a : Identifier

= : equal

b : Identifier

* : star

c : Identifier

; : semicolon

Line: 7

getch : Identifier (: open

) : closepara

; : semicolon

Line: 8

} : clos

Ex.No:05

Program Which Prints Number Of Characters, Spaces, Tabs And Lines In A Text File

Program:

```
#include <stdio.h>

int main()

{

   char in_name[80];

   FILE *in_file;

   int ch, character = 0, line = 0, space = 0, tab = 0;

   printf("Enter file name:\n");

   scanf("%s", in_name);

    in_file = fopen(in_name, "r");

     if (in_file == NULL)

       printf("Can't open %s for reading.\n", in_name);

    else

   {

       while ((ch = fgetc(in_file)) != EOF)

       {

      character++;

           if (ch == ' ')

               space++;

           if (ch == '\n')

               line++;

           if (ch == '\t')

               tab++;

       }

       fclose(in_file);

       printf("\nNumber of characters = %d", character);

       printf("\nNumber of spaces = %d", space);

       printf("\nNumber of tabs = %d", tab);
```

```
printf("\nNumber of lines = %d", line);

    }

    return 0;

}
```

Count.txt

```
Hello,

This is line 1.
This is line 2.
This is line 3.
This is line 4.

                Thanks.
```

OUTPUT:

```
Enter file name:
count.txt

Number of characters = 82
Number of spaces = 12
Number of tabs = 1
Number of lines = 8
```

Ex.No:06

IMPLEMENTATION OF SYMBOL TABLE

Program:

LEX PART:

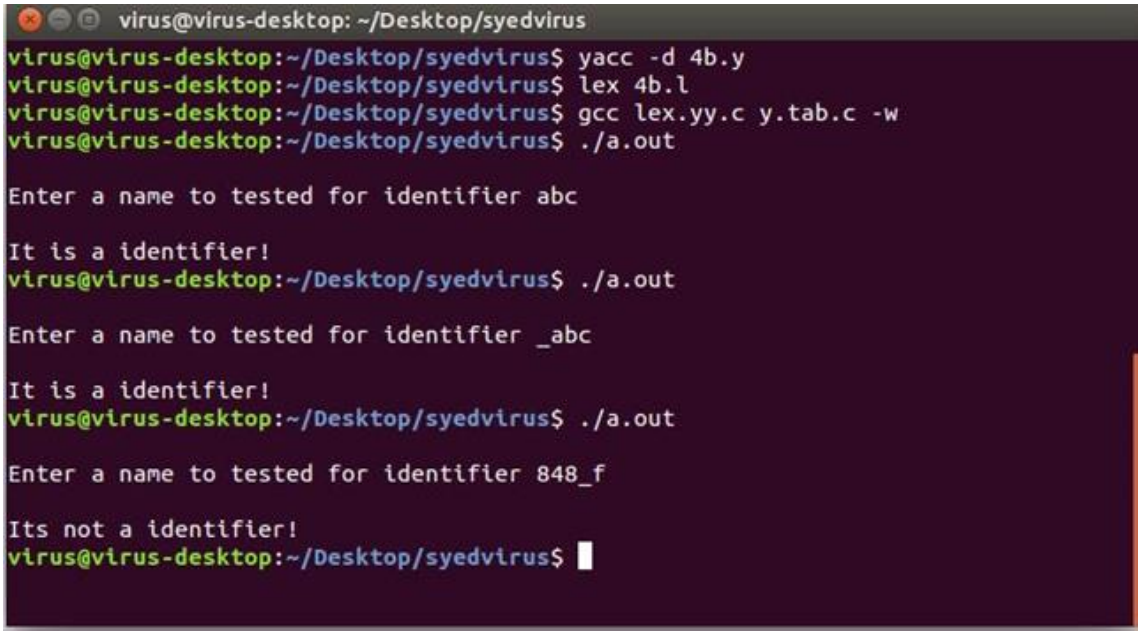
```
%{
  #include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return letter;
[0-9]                  return digit;
.                  return yytext[0];
\n                  return 0;
%%
int yywrap()
{
return 1;
}
```

YACC PART:

```
%{
 #include<stdio.h>
int valid=1;
%}
%token digit letter
%%
start : letter s
s :    letter s
     | digit s
     |
     ;
%%
```

```
int yyerror()
{
   printf("\nIts not a identifier!\n");
    valid=0;
   return 0;
}
int main()
{
   printf("\nEnter a name to tested for identifier ");
   yyparse();
   if(valid)
   {
      printf("\nIt is a identifier!\n");
   }
}
```

OUTPUT:



Ex.No:07

TO Implement Shift Reduce Parse

Program:

```
#include"stdio.h"

#include"stdlib.h"

#include"conio.h"

#include"string.h"

char ip_sym[15],stack[15];

int ip_ptr=0,st_ptr=0,len,i;

char temp[2],temp2[2];

char act[15];

void check();

void main(){

clrscr();

printf("\n\t\t SHIFT REDUCE PARSER\n");

printf("\n GRAMMER\n");

printf("\n E->E+E\n E->E/E");

printf("\n E->E*E\n E->a/b");

printf("\n enter the input symbol:\t");

gets(ip_sym);

printf("\n\t stack implementation table");

printf("\n stack\t\t input symbol\t\t action");

printf("\n______\t\t _____\t\t ______\n");

printf("\n $\t\t%s$\t\t\t--",ip_sym);

strcpy(act,"shift ");

temp[0]=ip_sym[ip_ptr];
```

```
temp[1]='\0';

strcat(act,temp);

len=strlen(ip_sym);

for(i=0;i<=len-1;i++){

stack[st_ptr]=ip_sym[ip_ptr];

stack[st_ptr+1]='\0';

ip_sym[ip_ptr]=' ';

ip_ptr++;

printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act);

strcpy(act,"shift ");

temp[0]=ip_sym[ip_ptr];

temp[1]='\0';

strcat(act,temp);

check();

st_ptr++;

}

st_ptr++;

check();

}

void check()

{

int flag=0;

temp2[0]=stack[st_ptr];

temp2[1]='\0';

if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))
```

```
{
 stack[st_ptr]='E';
 if(!strcmpi(temp2,"a"))
  printf("\n $%s\t\t%s$\t\t\tE->a",stack, ip_sym);
 else
  printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip_sym);
 flag=1;
}
if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"*"))||(!strcmpi(temp2,"/")))
{
 flag=1;
}
if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E*E")))
{
strcpy(stack,"E");
st_ptr=0;
if(!strcmpi(stack,"E+E"))
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
else
if(!strcmpi(stack,"E\E"))
printf("\n $%s\t\t %s$\t\t\tE->E\E",stack,ip_sym);
else
printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
flag=1;
}
```

```
if(!strcmpi(stack,"E")&&ip_ptr==len)

{

printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip_sym);

getch();

exit(0);

}

if(flag==0)

{

printf("\n%s\t\t\t%s\t\t reject",stack,ip_sym);

exit(0);

}

return;

}
```

OUTPUT:

```
                SHIFT REDUCE PARSER

GRAMMER

E->E+E
E->E/E
E->E*E
E->E/e
E->a/b
enter the input symbol:        a+b

        stack implementation table
stack           input symbol            action
------          ------------            ------

$               a+b$                    --
$a               +b$                    shift  a
$E               +b$                    E->a
$E+               b$                    shift +
$E+b               $                    shift b
$E+E               $                    E->b
$E                 $                    E->E*E
$E                 $                    ACCEPT_
```

Ex.No:08

Construction of LR Parsing table

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
char stack[30];
int top=-1;
void push(char c)
{
top++;
stack[top]=c;
}
char pop()
{
char c;
if(top!=-1)
{
c=stack[top];
top--;
return c;
}
return'x';
}
void printstat()
{
int i;
printf("\n\t\t\t $");
for(i=0;i<=top;i++)
printf("%c",stack[i]);
```

```
}
void main()
{
int i,j,k,l;
char s1[20],s2[20],ch1,ch2,ch3;
clrscr();
printf("\n\n\t\t LR PARSING");
printf("\n\t\t ENTER THE EXPRESSION");
scanf("%s",s1);
l=strlen(s1);
j=0;
printf("\n\t\t $");
for(i=0;i
{
if(s1[i]=='i' && s1[i+1]=='d')
{
s1[i]=' ';
s1[i+1]='E';
printstat(); printf("id");
push('E');
printstat();
}
else if(s1[i]=='+'||s1[i]=='-'||s1[i]=='*' ||s1[i]=='/' ||s1[i]=='d')
{
push(s1[i]);
printstat();
}
}
printstat();
l=strlen(s2);
while(l)
```

```
{
ch1=pop();
if(ch1=='x')
{
printf("\n\t\t\t $");
break;
}
if(ch1=='+'||ch1=='/'||ch1=='*'||ch1=='-')
{
ch3=pop();
if(ch3!='E')
{
printf("errror");
exit();
}
else
{
push('E');
printstat();
}
}
ch2=ch1;
}
getch();
}
```

OUTPUT:

LR PARSING

ENTER THE EXPRESSION

id+id*id-id

$

$id

$E

$E+

$E+id

$E+E

$E+E*

$E+E*id

$E+E*E

$E+E*E-

$E+E*E-id

$E+E*E-E

$E+E*E-E

$E+E*E

$E

$

Ex.No:09

IMPLEMENTATION OF CALCULATOR USING Lex & YACC

Program:

LEX PART:

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {
        yylval=atoi(yytext);
        return NUMBER;
      }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

YACC PART:

```
%{
   #include<stdio.h>
   int flag=0;
%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
```

```
%left '(' ')'
%%
ArithmeticExpression: E{
        printf("\nResult=%d\n",$$);
        return 0;
       };
E:E'+'E {$$=$1+$3;}
 |E'-'E {$$=$1-$3;}
 |E'*'E {$$=$1*$3;}
 |E'/'E {$$=$1/$3;}
 |E'%'E {$$=$1%$3;}
 |'('E')' {$$=$2;}
 | NUMBER {$$=$1;}
;
%%
void main()
{
   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction,
Multiplication, Divison, Modulus and Round brackets:\n");
   yyparse();
  if(flag==0)
   printf("\nEntered arithmetic expression is Valid\n\n");
}
void yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");
   flag=1;
}
```

OUTPUT:

```
virus@virus-desktop: ~/Desktop/syedvirus
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 4c.y
virus@virus-desktop:~/Desktop/syedvirus$ lex 4c.l
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Divison, Modulus and Round brackets:
((5+6+10+4+5)/5)%2

Result=0

Entered arithmetic expression is Valid

virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Divison, Modulus and Round brackets:
(9=0)

Entered arithmetic expression is Invalid

virus@virus-desktop:~/Desktop/syedvirus$
```

Ex.No:10

To Recognize A Valid Arithmetic Expression

Program:

LEX PART:

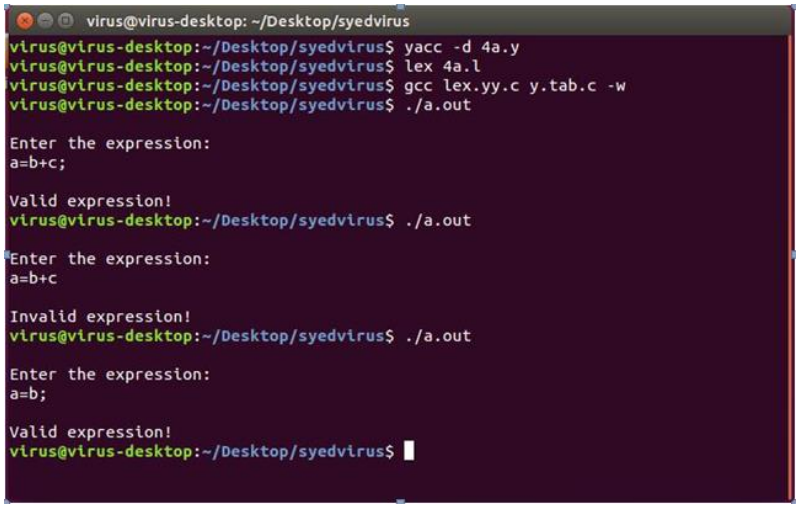
```
%{
   #include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return id;
[0-9]+(\.[0-9]*)?      return num;
[+/*]                  return op;
.                      return yytext[0];
\n                     return 0;
%%
int yywrap()
{
return 1;
}
```

YACC PART:

```
%{
   #include<stdio.h>
   int valid=1;
%}
%token num id op
%%
start : id '=' s ';'
s :     id x
      | num x
      | '-' num x
      | '(' s ')' x
```

```
    ;
x :    op s
    | '-' s
    |
    ;
%%
int yyerror()
{
   valid=0;
   printf("\nInvalid expression!\n");
   return 0;
}
int main()
{
   printf("\nEnter the expression:\n");
   yyparse();
   if(valid)
   {
      printf("\nValid expression!\n");
   }
}
```

OUTPUT:



Ex.No:11

To Implement Syntax Tree

Program:

```
#include<conio.h>
#include<stdio.h>
void main()
{
FILE *fp;
int i=0,j=0,k,l,row,col,s,x;
char a[10][10],ch,main[50],search;
clrscr();
fp=fopen("syntax.txt","r+");
while((ch=fgetc(fp))!=EOF)
{
if(ch=='\n')
   {
   row=i;
   col=j;
   j=0;
   i++;
   }
else
   {
   a[i][j]=ch;
   j++;
   }
}
printf("\n");
for(k=0;k<row+1;k++)
{
for(l=0;l<col;l++)
```

```
{
printf("%c",a[k][l]);
}
printf("\n");
}
i=0;
s=0;
for(k=0;k<row+1;k++)
{
        main[i]=a[k][1];
        i++;
        if(a[k][3]=='t')
        {
            search=a[k][4];
            for(l=0;l<i;l++)
            {
            if(main[l]==search)
                {
                main[i]=main[l];
                i++;
                break;
                }
            }
        main[i]=a[k][5];
        s=5;
        i++;
        }
        else
        {
            main[i]=a[k][3];
                //    printf("\n%c",main[i]);
```

```
        i++;
        main[i]=a[k][4];
          //    printf(",%c\n",main[i]);
        s=4;
        i++;
      }
      s++;
      if(a[k][s]=='t')
      {
      s++;
      search=a[k][s];
      for(l=0;l<i;l++)
        {
        if(main[l]==search)
          {
          main[i]=main[l];
          i++;
          break;
          }
        }
      }
      else
      {
      main[i]=a[k][s];
      i++;
      }
}
      for(x=i-1;x>=0;x=x-4)
      {
      printf("\ntt%c: root->%c ",main[x-3],main[x-1]);
      if(main[x-2]>48 &&main[x-2]<59)
```

```
        printf("lc->t%c ",main[x-2]);
    else
        printf("lc->%c ",main[x-2]);
    if(main[x]>48 &&main[x]<59)
        printf("rc->t%c ",main[x]);
    else
        printf("rc->%c ",main[x]);
    }
getch();
}
```

Syntax.txt

```
t1=a+b
```

OUTPUT:

```
t1=a+b
tt1:root->+     lc->a     rc->b
```

Ex.No:12

Three address code generation for assignment statement

Program:

```
#include<stdio.h>
char s[20],t[20];
void main()
{
printf("\nEnter expression:");
scanf("%s",&t);
printf("\nIntermediate code is:");
if(isalpha(t[2])&&isalpha(t[0])&&isalpha(t[4]))
{
printf("\n mov%c.r",t[2]);
else
printf("\nEnter correct expression!");switch(t[3])
{
case '*':
printf("\n mul %c.r".t[4]);
printf("\n mov r.%c",t[0]); break;
case '+':
printf("\n add %c.r",t[4]);
printf("\n mov r.%c",t[0]); break;
case '-':
printf("\n sub %c.r",t[4]);
printf("\n mov r.%c",t[0]); break;
case '/':
printf("\n div %c.r",t[4]);
printf("\n mov r.%c",t[0]); break;
default:
printf("\nInvalid expression!"); break;
} }
```

Output:

```
./a.out
Enter expression:a=a+b
Intermediate code is:
mova.r
add b.r
```

Ex.No:13

Three address code generation for Conditional Expression.

Program:

```
#include<stdio.h>
#include<string.h>
Void pm();
void plus();
void div();
int i,ch,j,l,addr=100;
char ex[10],exp[10],exp1[10],exp2[10],id1[5],op[5],id2[5];
 void main(){
clrscr();
while(1){
printf("\n1.assignment\n2.arithmetic\n3.relational\n4.Exit\nEnter the choice:”);
scanf("%d",&ch);{
case 1:
 printf("\nEnter the expression with assignment operator:”);
scanf("%s",exp);
l=strlen(exp);
exp2[0]=’\0’;
i=0;
while(exp[i]!=’=’){
l++;
}
strncat(exp2,exp,i);
strrev(exp);
exp1[0]=’\0’;
strncat(exp1,exp,l-(i+1));
strrev(exp1);
print(“Three address code:\ntemp=%s\n%s=temp\n”,exp1,exp2);
break;
case 2:
printf("\nEnter the expression with arithmetic operator:”);
```

```
scanf("%s",ex);
strcpy(exp,ex);
l=strlen(exp);
exp1[0]='\0';

for(i=0;i<1;i++){
if(exp[i]=='+'||exp[i]=='-'){
if(exp[i+2]=='/'||exp[i+2]=='*'){
pm();
break;
}
else{
plus();
break;
}}
else if(exp[i]=='/'||exp[i]=='*'){
div();
break;
}}
break;
case 3:
printf("Enter the expression with relational operator");
scanf("%s%s%s",&id1,&op,&id2);
if(((strcmp(op,"<")==0)||(strcmp(op,">")==0)||(strcmp(op,"<=")==0)||(strcmp(op,
">=")==0)||(strcmp(op,"==")= =0)||(strcmp(op,"!=")==0))==0)
printf("Expression is error");
else{
printf("\n%d\tif%s%s%s goto %d",addr,id1,op,id2,addr+3);
addr++;
printf("\n%d\t T:=0",addr);
addr++;
printf("\n%d\t goto %d",addr,addr+2);
addr++;
```

```
printf("\n%d\t T:=1",addr);
}
break;
case 4:
exit(0);
}}}
void pm(){
strrev(exp);
j=l-i-1;
strncat(exp1,exp,j);
strrev(exp1);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[j+1],exp[j]);
}
void div(){
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[i+2],exp[i+3]);
}
void plus()
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[i+2],exp[i+3]);
}
```

OUTPUT:

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:1

Enter expression the with assignment operator: a=b

Three address code:

temp=b

a=temp

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:2

Enter expression the with assignment operator: a+b-c

Three address code:

Temp = a+b

temp1=temp-c

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:2

Enter expression the with assignment operator: a-b/c

Three address code:

temp=b/c

temp1=a-temp

1.assignment

2.arithmetic

3.relational

4.Exit

```
Enter the choice:2

Enter expression the with assignment operator: a*b-c

Three address code:

temp=a*b

temp1=temp-c

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:2

Enter expression the with assignment operator: a/b*c

Three address code:

temp=a/b

temp1=temp*c

1.assignment

2.arithmetic

3.relational

4.Exit

Enter the choice:3

Enter expression the with assignment operator: a<=b

100 if a<=b goto 103

101 T:=0

102 goto 104

103 T:=1

1.assignment

2.arithmetic

3.relational

4.Exit
```

Ex.No:14

IMPLEMENTATION OF SYMBOL TABLE

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MIN_PER_RANK 1
#define MAX_PER_RANK 5
#define MIN_RANKS 3
#define MAX_RANKS 5
#define PERCENT 30
void main()
{
int i,j,k,nodes=0;
srand(time(NULL));
int ranks=MIN_RANKS+(rand()%(MAX_RANKS-MIN_RANKS+1));
printf("DIRECTED ACYCLIC GRAPH\n");
for(i=1;i<ranks;i++)
{
int new_nodes=MIN_PER_RANK+(rand()%(MAX_PER_RANK-MIN_PER_RANK+1));
for(j=0;j<nodes;j++)
for(k=0;k<new_nodes;k++)
if((rand()%100)<PERCENT)
printf("%d->%d;\n",j,k+nodes);
nodes+=new_nodes;
}
}
```

OUTPUT:

```
l2sys29@l2sys29-Veriton-M275: ~/Desktop/syedvirus
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./a.out
DIRECTED ACYCLIC GRAPH
0->4;
0->6;
0->7;
0->9;
1->6;
1->7;
2->6;
3->7;
3->8;
4->7;
4->9;
5->6;
5->8;
1->10;
1->11;
1->12;
3->11;
4->10;
5->10;
5->11;
7->10;
8->10;
9->12;
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$
```

Ex.No:15

Code Optimization

Program code:(before.c)

```
#include<stdio.h>
void main()
{
int i,n; int fact=1;
printf("\nEnter a number: ");scanf("%d",&n);
for(i=n; i>=1 ; i++) fact = fact * i;
printf("The factorial value is:%d",fact);
}
```

Program code:(after.c)

```
#include<stdio.h>
void main()
{
int i,f; f=1;
printf("Enter a number:\n ");scanf("%d",&n);
do
f = f * n; n--;
}while(n>0)
printf("The factorial value is:%d",f);
}
```

OUTPUT:

```
./a.out
Enter a number: 5
The factorial value is:120
```