LIST OF EXPERIMENTS

SL NO	EXPERIMENTS	PAGE NO
1	**DESIGN AND IMPLEMENT A LEXICAL ANALYSER USING C LANGUAGE**	4
2	**IMPLEMENTATION OF LEXICAL ANALYSER USING LEX TOOL**	7
3	**PROGRAM TO DISPLAY NUMBER OF LINES,WORDS AND CHARACTERS**	10
4	**LEX PROGRAM TO CONVERT SUBSTRING abc TO ABC**	13
5	**LEX PROGRAM TO FIND NUMBER OF VOWELS AND CONSONENTS**	15
6	**GENERATE A YACC SPECIFICATION TO RECOGNISE VALID ARITHMETIC EXPRESSION**	17
7	**GENERATE A YACC SPECIFICATION TO RECOGNISE A VALID IDENTIFIER**	19
8	**IMPLEMENTATION OF CALCULATOR**	23
9	**CONVERT THE BNF RULES INTO YACC FORM**	27
10	**ε- CLOSSURE OF ALL STATES OF ANY GIVEN NFA WITH ε TRANSITION**	30
11	**CONVERT ε(epsilon) NFA to NFA**	32
12	**CONVERT NFA TO DFA**	34

EXP 1

DESIGN AND IMPLEMENT A LEXICAL ANALYSER USING C LANGUAGE

AIM

To implement lexical analyzer using C language

PROGRAMS

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
int isKeyword(char buffer[]){
char keywords[32][10] =
{"auto","break","case","char","const","continue","default",
"do","double","else","enum","extern","float","for","goto",
"if","int","long","register","return","short","signed",
"sizeof","static","struct","switch","typedef","union",
"unsigned","void","volatile","while"};
int i, flag = 0;
for(i = 0; i < 32; ++i){
if(strcmp(keywords[i], buffer) == 0){
flag = 1;
break;
}
}
return flag;
}
int main(){
char ch, buffer[15], operators[] = "+-
*/%=",specialch[]=",;[]{}",num[]="1234567890",buf[10];
FILE *fp;
int i,j=0,k=0;
fp = fopen("program.txt","r");
if(fp == NULL){
printf("error while opening the file\n");
exit(0);
}
while((ch = fgetc(fp)) != EOF)
{
for(i = 0; i < 6; ++i)
{
if(ch == operators[i])
{
printf("%c is operator\n", ch);
}
if(ch == specialch[i])
{
printf("%c is special character\n", ch);
}
}
if(isalpha(ch)){
buffer[j++] = ch;}
if(isdigit(ch)){
buf[k++]=ch;
}
else if((ch == ' ' || ch == '\n') && (j != 0)){

buffer[j] = '\0';
j = 0;
if(isKeyword(buffer) == 1)
printf("%sis keyword\n", buffer);
else{
printf("%s is identifier\n", buffer);
printf("%s is constant\n", buf);
}}
}
fclose(fp);
return 0;
```

RESULT

The programs executed successfully and output obtained

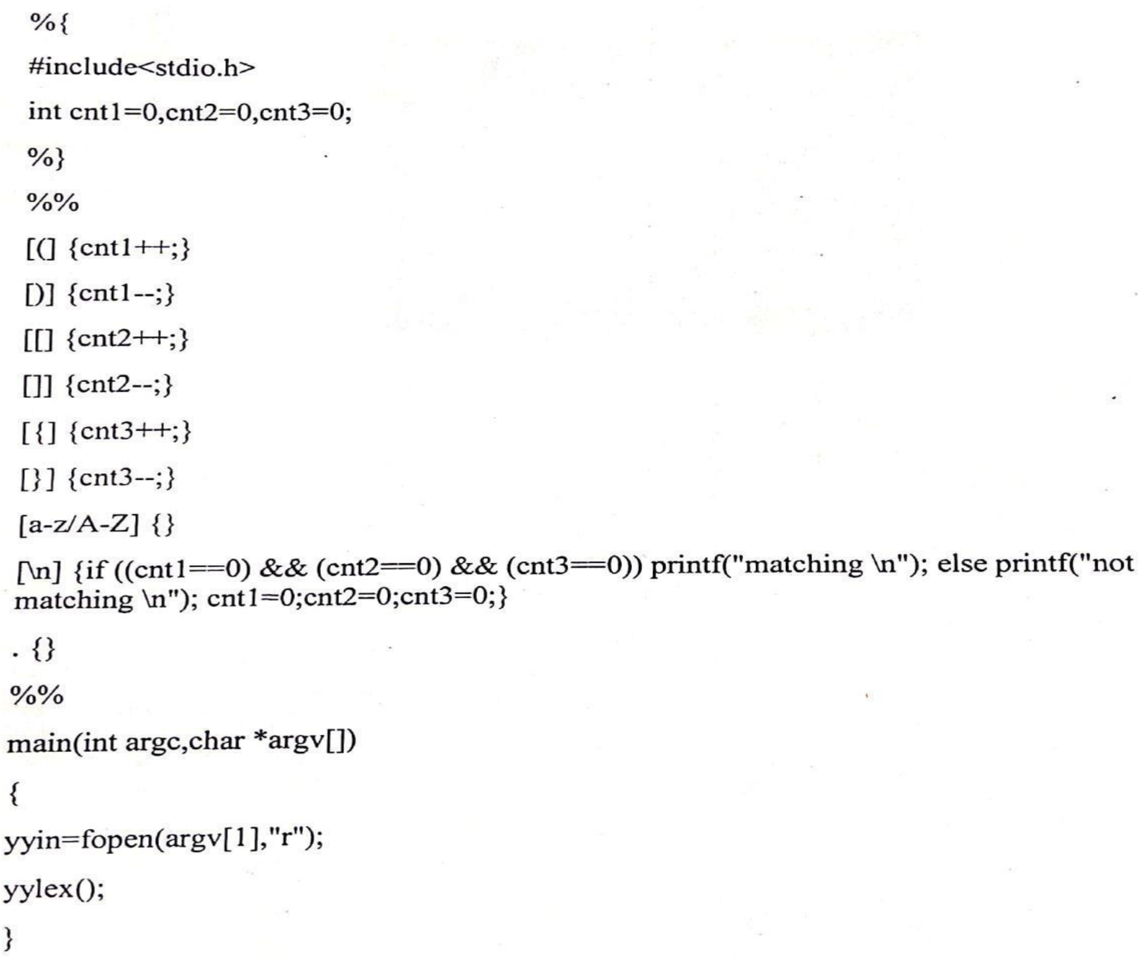
EXP 2

IMPLEMENTATION OF LEXICAL ANALYSER USING LEX TOOL

AIM

To implement lexical analyzer using lex tool

PROGRAM



```
%{
#include<stdio.h>
int cnt1=0,cnt2=0,cnt3=0;
%}
%%
[(] {cnt1++;}
[)] {cnt1--;}
[[] {cnt2++;}
[]] {cnt2--;}
[{] {cnt3++;}
[}] {cnt3--;}
[a-z/A-Z] {}
[\n] {if ((cnt1==0) && (cnt2==0) && (cnt3==0)) printf("matching \n"); else printf("not
matching \n"); cnt1=0;cnt2=0;cnt3=0;}
. {}
%%
main(int argc,char *argv[])
{
yyin=fopen(argv[1],"r");
yylex();
}
```

RESULT

The programs executed successfully and output obtained

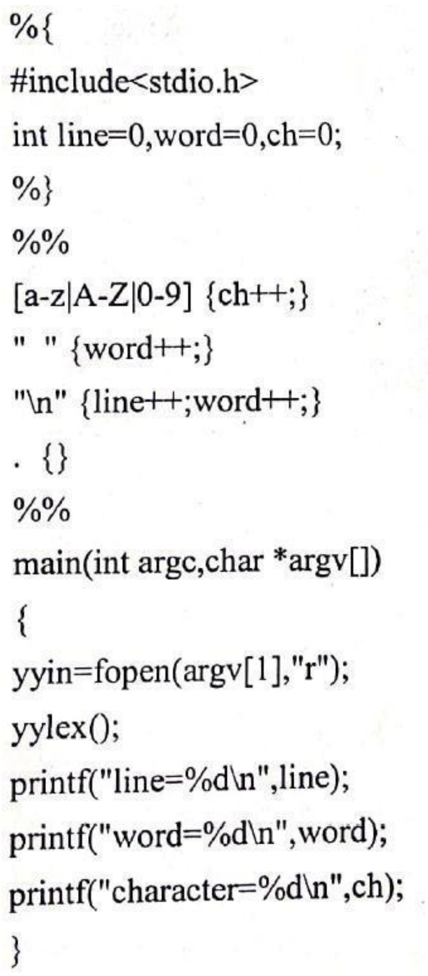
EXP 3

PROGRAM TO DISPLAY NUMBER OF LINES,WORDS AND CHARACTERS

AIM

To implement program to display number of lines,words and characters

PROGRAM



```
%{
#include<stdio.h>
int line=0,word=0,ch=0;
%}
%%
[a-z|A-Z|0-9] {ch++;}
" " {word++;}
"\n" {line++;word++;}
. {}
%%
main(int argc,char *argv[])
{
yyin=fopen(argv[1],"r");
yylex();
printf("line=%d\n",line);
printf("word=%d\n",word);
printf("character=%d\n",ch);
}
```

RESULT

The programs executed successfully and output obtained

EXP 4

LEX PROGRAM TO CONVERT THE SUBSTRING abc TO ABC

AIM

To implement lex program to convert the substring abc to ABC

PROGRAMS

```
%{
#include
#include
int i;
%}
%%
[a-z A-Z]* {
for(i=0;i<=yyleng;i++)
{
if((yytext[i]=='a')&&(yytext[i+1]=='b')&&(yytext[i+2]=='c'))
{
yytext[i]='A';
yytext[i+1]='B';
yytext[i+2]='C';
}
}
printf("%s",yytext);
}
[\t]* return;
```

```
.* {ECHO;}
\n {printf("%s",yytext);}
%%
main()
{
yylex();
}
int yywrap()
{
return 1;
}
```

RESULT

The programs executed successfully and output obtained

EXP 5

FIND NUMBER OF VOWELS AND CONSONANTS FROM THE GIVEN INPUT STRING

Aim

LEX Progaram to find out total number of vowels and consonants from the given input string.

PROGRAM

```
%{
int vow_count=0;
int const_count =0;
%}

%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap(){}
int main()
{
printf("Enter the string of vowels and consonents:");
yylex();
printf("Number of vowels are: %d\n", vow_count);
printf("Number of consonants are: %d\n", const_count);
return 0;
}
```

RESULT

The program executed successfully and output obtained

EXP 6

YACC SPECIFICATION TO RECOGNIZE A VALID ARITHMETIC EXPRESSION

AIM

To Generate a YACC specification to recognize a valid arithmetic expression that uses operators +,-,*,/ and parenthesis.

ALGORITHM:

Step1: Start the program.

Step2: Reading an expression .

Step3: Checking the validating of the given expression according to the rule using yacc.

Step4: Using expression rule print the result of the given values
Step5: Stop the program.

PROGRAMS

```
%{
/* Definition section */
#include <stdio.h>
%}

%token NUMBER ID
// setting the precedence
// and associativity of operators
%left '+' '-'
%left '*' '/'

/* Rule Section */
%%
E : T {
printf("Result = %d\n", $$);
return 0;
}
```

```
T :
T '+' T { $$ = $1 + $3; }
| T '-' T { $$ = $1 - $3; }
| T '*' T { $$ = $1 * $3; }
| T '/' T { $$ = $1 / $3; }
| '-' NUMBER { $$ = -$2; }
| '-' ID { $$ = -$2; }
| '(' T ')' { $$ = $2; }
| NUMBER { $$ = $1; }
| ID { $$ = $1; };
% %

int main() {
printf("Enter the expression\n");
yyparse();
}

/* For printing error messages */
int yyerror(char* s) {
printf("\nExpression is invalid\n");
}
```

RESULT

The program executed successfully and output obtained

EXP 7

YACC SPECIFICATION TO RECOGNIZE A VALID IDENTIFIER WHICH STARTS WITH A LETTER

AIM

To generate a YACC specification to recognize a valid identifier which starts with a letter followed by any number of letters or digits.

ALGORITHM

Step1: Start the program

Step2: Reading an expression

Step3: Checking the validating of the given expression according to the rule using yacc.

Step4: Using expression rule print the result of the given values

Step5: Stop the program

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>
nt 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");
 }
}
```

RESULT

The program executed successfully and output obtained

EXP 8

IMPLEMENTATION OF CALCULATOR

AIM

To implementation of calculator using LEX and YACC.

ALGORITHM:

Step1: A Yacc source program has three parts as follows:

Declarations %% translation rules %% supporting C routines

Step2: Declarations Section: This section contains entries that:

i. Include standard I/O header file.

ii. Define global variables.

iii. Define the list rule as the place to start processing.

iv. Define the tokens used by the parser. v. Define the operators and their precedence.

Step3: Rules Section: The rules section defines the rules that parse the input stream. Each rule of a grammar production and the associated semantic action.

Step4: Programs Section: The programs section contains the following subroutines. Because these subroutines are included in this file, it is not necessary to use the yacc library when processing this file.

Step5: Main- The required main program that calls the yyparse subroutine to start the program.

Step6: yyerror(s) -This error-handling subroutine only prints a syntax error message.

Step7: yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file contains include statements for standard input and output, as programmar file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.

Step8: calc.lex contains the rules to generate these tokens from the input stream.

PROGRAMS

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;
}
```

RESULT

The program executed successfully and output obtained

EXP .9

COVERT BNF RULES INTO YACC FORM

AIM

Convert the BNF rules into YACC form and write code to generate abstract syntax tree.

ALGORITHM

1. Start

2. Include the header file.

3. In int code.l,declare the variable lie no as integer and assign it to be equal to '1'.

4. Start the int code.l with declarative section.

5. In translation rules section define keywords ,data types and integer along with their actions .

6. Start the main block. In main block check the statement

7.

1.declarative

2.assignment

3.conditional

4.if and else

5.While assignment.

8. Perform the actions of that particular block.

9. In main program declare the parameters arg c as int end *argv[] as char.

10. In main program open file in read mode.

11. Print the output in a file.

12.End

PROGRAM

Lex

```
%{
#include"y.tab.h"
#include
#include
int LineNo=1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
return VAR;}
{number} {strcpy(yylval.var,yytext);
return NUM;}
\< |
\> |
\>= |
\<= |
== {strcpy(yylval.var,yytext);
return RELOP;}
[ \t] ;
\n LineNo++;
. return yytext[0];
%%
```

Yacc

```
%{
#include
#include
struct quad
{
char op[5];
char arg1[10];
char arg2[10];
char result[10];
}QUAD[30];
struct stack
{
int items[100];
int top;
}stk;
int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
%}
%union
{
char var[10];
}
%token  NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE
```

```
%type  EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-' '+'
%left '*' '/'
%%
PROGRAM : MAIN BLOCK
;
BLOCK: '{' CODE '}'
;
CODE: BLOCK
| STATEMENT CODE
| STATEMENT
;
STATEMENT: DESCT ';'
| ASSIGNMENT ';'
| CONDST
| WHILEST

;
DESCT: TYPE VARLIST
;
VARLIST: VAR ',' VARLIST
| VAR
;
ASSIGNMENT: VAR '=' EXPR{
strcpy(QUAD[Index].op,"=");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,$1);
strcpy($$,QUAD[Index++].result);
};
```

```
EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}
| EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);}
| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}
| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}
| '(' EXPR ')' {strcpy($$,$2);}
| VAR
| NUM
;
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
| IFST ELSEST
;
IFST: IF '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
```

```
push(Index);

Index++;

};

ELSEST: ELSE{

tInd=pop();

Ind=pop();

push(tInd);

sprintf(QUAD[Ind].result,"%d",Index);

}

BLOCK{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

};

CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);

StNo=Index-1;

}

| VAR

| NUM

;

WHILEST: WHILELOOP{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",StNo);

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

};

WHILELOOP: WHILE '(' CONDITION ')' {

strcpy(QUAD[Index].op,"==");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"FALSE");

strcpy(QUAD[Index].result,"-1");
```

```
push(Index);

Index++;

}

BLOCK {

strcpy(QUAD[Index].op,"GOTO");

strcpy(QUAD[Index].arg1,"");

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

};

%%

extern FILE *yyin;

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

{

FILE *fp;

int i;

if(argc>1)

{

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

if(!fp)

{

printf("\n File not found");

exit(0);

}

yyin=fp;


}

yyparse();

printf("\n\n\t\t ----------------------------\n\t\t Pos Operator Arg1 Arg2 Result\n\t\t-------------------");
```

```
for(i=0;i

{

printf("\n\t\t %d\t %s\t %s\t %s\t %s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);

}

printf("\n\t\t -----------------------");

printf("\n\n");

return 0;

}

void push(int data)

{s

tk.top++;

if(stk.top==100)

{

printf("\n Stack overflow\n");

exit(0);

}s

tk.items[stk.top]=data;

} int pop()

{ int data;

if(stk.top==-1)

{

printf("\n Stack underflow\n");

exit(0);

}

data=stk.items[stk.top--];

return data;

}

void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])

{s

trcpy(QUAD[Index].op,op);
strcpy(QUAD[Index].arg1,arg1);

strcpy(QUAD[Index].arg2,arg2);

sprintf(QUAD[Index].result,"t%d",tIndex++);

strcpy(result,QUAD[Index++].result);

}

yyerror()

{

printf("\n Error on line no:%d",LineNo);

}
```

Input

```
main()

{ int a,b,c;

if(a

{a=a+b;}

while(a

{a=a+b;}

if(a<=b)

{c=a-b;}

else

{c=a+b;}

}
```

RESULT

The program executed successfully and output obtained

EXP.10

ε- CLOSSURE OF ALL STATES OF ANY GIVEN NFA WITH ε TRANSITION

AIM

Write program to find ε – closure of all states of any given NFA with ε transition.

PROGRAM

```
#include<stdio.h>

#include<string.h>

char result[20][20], copy[3], states[20][20];

void add_state(char a[3], int i) {
  strcpy(result[i], a);
}

void display(int n) {
  int k = 0;
  printf("nnn Epsilon closure of %s = { ", copy);
  while (k < n) {
    printf(" %s", result[k]);
    k++;
  }
  printf(" } nnn");
}

int main() {
  FILE * INPUT;
  INPUT = fopen("input.dat", "r");
  char state[3];
  int end, i = 0, n, k = 0;
  char state1[3], input[3], state2[3];
```

```
printf("n Enter the no of states: ");
scanf("%d", & n);
printf("n Enter the states n");
for (k = 0; k < 3; k++) {
  scanf("%s", states[k]);

}

for (k = 0; k < n; k++) {
  i = 0;
  strcpy(state, states[k]);
  strcpy(copy, state);
  add_state(state, i++);
  while (1) {
    end = fscanf(INPUT, "%s%s%s", state1, input, state2);
    if (end == EOF) {
      break;
    }

    if (strcmp(state, state1) == 0) {
      if (strcmp(input, "e") == 0) {
        add_state(state2, i++);
        strcpy(state, state2);
      }
    }

  }
  display(i);
  rewind(INPUT);
}

return 0;
```

```
}
```

Input-

```
q0 0 q0
q0 1 q1
q0 e q1
q1 1 q2
q1 e q2
```

RESULT

The program executed successfully and output obtained

EXP 11.

CONVERT ε(epsilon) NFA to NFA

AIM

C Program for Converting ε(epsilon) NFA to NFA

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int st;
    struct node *link;
};

void findclosure(int,int);
void insert_trantbl(int ,char, int);
int findalpha(char);
void findfinalstate(void);
void unionclosure(int);
void print_e_closure(int);
static int set[20],nostate,noalpha,s,notransition,nofinal,start,finalstate[20],c,r,buffer[20];
char alphabet[20];
static int e_closure[20][20]={0};
struct node * transition[20][20]={NULL};
void main()
{
     int i,j,k,m,t,n;

     struct node *temp;
     printf("enter the number of alphabets?\n");
     scanf("%d",&noalpha);
```

```
  getchar();
  printf("NOTE:- [ use letter e as epsilon]\n");

 printf("NOTE:- [e must be last character ,if it is present]\n");

 printf("\nEnter alphabets?\n");
 for(i=0;i<noalpha;i++)
 {

        alphabet[i]=getchar();
        getchar();
 }
printf("Enter the number of states?\n");
scanf("%d",&nostate);
printf("Enter the start state?\n");
scanf("%d",&start);
printf("Enter the number of final states?\n");
scanf("%d",&nofinal);
printf("Enter the final states?\n");
for(i=0;i<nofinal;i++)
        scanf("%d",&finalstate[i]);
 printf("Enter no of transition?\n");
scanf("%d",&notransition);
printf("NOTE:- [Transition is in the form--> qno   alphabet   qno]\n",notransition);
printf("NOTE:- [States number must be greater than zero]\n");
printf("\nEnter transition?\n");
for(i=0;i<notransition;i++)
{


        scanf("%d %c%d",&r,&c,&s);
        insert_trantbl(r,c,s);
```

```
}

printf("\n");

for(i=1;i<=nostate;i++)
{
        c=0;
        for(j=0;j<20;j++)

        {
                      buffer[j]=0;
                       e_closure[i][j]=0;
        }
        findclosure(i,i);
}
printf("Equivalent NFA without epsilon\n");
printf("-----------------------------------\n");
printf("start state:");
print_e_closure(start);
printf("\nAlphabets:");
for(i=0;i<noalpha;i++)
          printf("%c ",alphabet[i]);
printf("\n States :" );
for(i=1;i<=nostate;i++)
          print_e_closure(i);

printf("\nTnransitions are...:\n");

for(i=1;i<=nostate;i++)
{

          for(j=0;j<noalpha-1;j++)
        {
                for(m=1;m<=nostate;m++)
                        set[m]=0;
                for(k=0;e_closure[i][k]!=0;k++)
                {

                    t=e_closure[i][k];
                    temp=transition[t][j];
                    while(temp!=NULL)
                    {

                        unionclosure(temp->st);
                        temp=temp->link;
                    }
                }
               printf("\n");
               print_e_closure(i);
               printf("%c\t",alphabet[j]   );
               printf("{");
               for(n=1;n<=nostate;n++)
               {
                    if(set[n]!=0)
                        printf("q%d,",n);
               }
               printf("}");
        }
  }
  printf("\n Final states:");
  findfinalstate();
```

```
}

void findclosure(int x,int sta)

{

            struct node *temp;

            int i;

           if(buffer[x])

                    return;

             e_closure[sta][c++]=x;

            buffer[x]=1;

             if(alphabet[noalpha-1]=='e' && transition[x][noalpha-1]!=NULL)

                {

                           temp=transition[x][noalpha-1];

                           while(temp!=NULL)

                          {

                                       findclosure(temp->st,sta);

                                       temp=temp->link;

                           }

                }

  }

void insert_trantbl(int r,char c,int s)

{

           int j;

           struct node *temp;

            j=findalpha(c);

          if(j==999)

          {

                    printf("error\n");

                   exit(0);

          }
```

```
    temp=(struct node *) malloc(sizeof(struct node));
    temp->st=s;
    temp->link=transition[r][j];
    transition[r][j]=temp;
}

int findalpha(char c)
{
        int i;
        for(i=0;i<noalpha;i++)
            if(alphabet[i]==c)
                return i;

          return(999);


}

void unionclosure(int i)
{
         int j=0,k;
        while(e_closure[i][j]!=0)
        {
             k=e_closure[i][j];
             set[k]=1;
             j++;
        }
}
void findfinalstate()
{
        int i,j,k,t;
        for(i=0;i<nofinal;i++)
        {
                for(j=1;j<=nostate;j++)
                {
                        for(k=0;e_closure[j][k]!=0;k++)
                          {
                                if(e_closure[j][k]==finalstate[i])
                                {

                                        print_e_closure(j);
                                }
                         }
                }
         }


  }

void print_e_closure(int i)
{
        int j;
        printf("{");
        for(j=0;e_closure[i][j]!=0;j++)
                        printf("q%d,",e_closure[i][j]);
         printf("}\t");
}
```

RESULT

The program executed successfully and output obtained

EXP 12.

CONVERT NFA TO DFA

AIM

program to convert NFA to DFA

PROGRAM

```
#include<stdio.h>

#include<string.h>

#include<math.h>


int ninputs;

int dfa[100][2][100] = {0};

int state[10000] = {0};

char ch[10], str[1000];

int go[10000][2] = {0};

int arr[10000] = {0};


int main()

{

    int st, fin, in;

    int f[10];

    int i,j=3,s=0,final=0,flag=0,curr1,curr2,k,l;

    int c;


    printf("\nFollow the one based indexing\n");


    printf("\nEnter the number of states::");

    scanf("%d",&st);


    printf("\nGive state numbers from 0 to %d",st-1);
```

```
for(i=0;i<st;i++)
       state[(int)(pow(2,i))] = 1;

printf("\nEnter number of final states\t");
scanf("%d",&fin);

printf("\nEnter final states::");
for(i=0;i<fin;i++)
{
     scanf("%d",&f[i]);
}

int p,q,r,rel;


printf("\nEnter the number of rules according to NFA::");
scanf("%d",&rel);


printf("\n\nDefine transition rule as \"initial state input symbol final state\"\n");




for(i=0; i<rel; i++)
{
     scanf("%d%d%d",&p,&q,&r);
     if (q==0)
       dfa[p][0][r] = 1;
     else
       dfa[p][1][r] = 1;
}

printf("\nEnter initial state::");
scanf("%d",&in);
```

```
in = pow(2,in);

i=0;


printf("\nSolving according to DFA");


int x=0;
for(i=0;i<st;i++)
{
    for(j=0;j<2;j++)
    {
        int stf=0;
        for(k=0;k<st;k++)
        {
            if(dfa[i][j][k]==1)
              stf = stf + pow(2,k);
        }



        go[(int)(pow(2,i))][j] = stf;
        printf("%d-%d-->%d\n",(int)(pow(2,i)),j,stf);
        if(state[stf]==0)
          arr[x++] = stf;
        state[stf] = 1;
    }

}


//for new states
```

```
for(i=0;i<x;i++)
{
    printf("for %d ---- ",arr[x]);
    for(j=0;j<2;j++)
    {
        int new=0;
        for(k=0;k<st;k++)
        {
            if(arr[i] & (1<<k))
            {
                int h = pow(2,k);

                if(new==0)
                   new = go[h][j];
                new = new | (go[h][j]);



            }
        }

        if(state[new]==0)
        {
           arr[x++] = new;
           state[new] = 1;
        }
    }
}

printf("\nThe total number of distinct states are::\n");
printf("STATE     0   1\n");


for(i=0;i<10000;i++)
{
       if(state[i]==1)
       {
               //printf("%d**",i);
               int y=0;
               if(i==0)
                   printf("q0 ");


               else
               for(j=0;j<st;j++)
               {
                       int x = 1<<j;
                       if(x&i)
                       {
                           printf("q%d ",j);
                           y = y+pow(2,j);
                           //printf("y=%d  ",y);
                       }
               }
               //printf("%d",y);
               printf("       %d   %d",go[y][0],go[y][1]);
               printf("\n");
       }
}
```

```
j=3;
while(j--)
{
    printf("\nEnter string");
    scanf("%s",str);
    l = strlen(str);
    curr1 = in;
    flag = 0;
    printf("\nString takes the following path-->\n");
    printf("%d-",curr1);

    for(i=0;i<l;i++)
    {
      curr1 = go[curr1][str[i]-'0'];
      printf("%d-",curr1);
    }


    printf("\nFinal state - %d\n",curr1);


    for(i=0;i<fin;i++)
    {
        if(curr1 & (1<<f[i]))
        {
            flag = 1;
            break;
        }
    }


    if(flag)
      printf("\nString Accepted");
```

```
        else

            printf("\nString Rejected");


    }



    return 0;

}
```

RESULT

The program executed successfully and output obtained