EXPERIMENT 2

```
#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","stati
c","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[]="+-*/%=",symbols[]=".,#$?{}%()&;";
 FILE *fp;
 int i,j=0;
 fp=fopen("program.txt","r");
 if(fp==NULL)
 {
  printf("ERROR WHILE OPENING 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);
  }
  for(i=0;i<12;++i)
  {
   if(ch==symbols[i])
```

```
  printf("%c is symbol\n",ch);
 }
 if(isalnum(ch))
 {
  buffer[j++]=ch;
 }
 else if((ch==' '||ch=='\n')&&(j!=0))
 {
  buffer[j]='\0';
  j=0;
  if(isKeyword(buffer)==1)
  printf("%s is keyword\n",buffer);
  else
  printf("%s is identifer\n",buffer);
 }
}
fclose(fp);
return 0;
}
```

Program.txt

```
void main
{
int a, e, j;
j= a * e;
}
```

Output

```
student@CSNPLPC51:~/nicy$ ./a.out
void is keyword
main is identifier
{ is a symbol
int is keyword
a is identifier
, is a symbol
e is identifier
, is a symbol
; is a symbol
j is identifier
= is operator
j is identifier
a is identifier
* is operator
; is a symbol
e is identifier
} is a symbol
```

EXPERIMENT 4:LEX TOOL

```
  %{
int nlines=0;
%}
DIGIT[0-9]
LETTER[a-z|A-Z]
ID {LETTER}({LETTER}|{DIGIT})*
%%
\n ++nlines;
{DIGIT}+ {printf("An Integer %s\n",yytext);}
{DIGIT}+"."{DIGIT}+ {printf("A floating point number%s\n",yytext);}
IF|THEN|ELSE|BEGIN|END|FOR|WHILE|INT {printf("keyword%s\n",yytext);}
{ID} {printf("An Identifier is %s\n",yytext);}
"+"|"-"|"*"|"/"|"="|"<"|">"|"<="|">=" {printf("an operator%s\n",yytext);}
%%
main()
{
printf("Enter expressions\n");
yylex();
printf("Number of lines=%d",nlines);
}
```

```
Enter expressions
IF a=b
keywordIF
 An Identifier is a
an operator=
An Identifier is b
Number of lines=1student@CSNPLPC51
```

Compiling steps:

flex expname.l

gcc lex.yy.c -lfl

./a.out

CASE WHERE LINE NO IS 2.(just for reference)

```
Number of lines=1student@CSNPLPC51:~/nicy$ ./a.out
Enter expressions
IF a=b
keywordIF
 An Identifier is a
an operator=
An Identifier is b
hell world
An Identifier is hell
 An Identifier is world
Number of lines=2student@CSNPLPC51:~/nicy$
```

EXPERIMENT 5 ***Lex pgm to count the no. of lines,words,characters in a given statement.***

```
%{
  #include<stdio.h>
  int sc=0,wc=0,lc=0,cc=0;
%}
%%
[\n] {lc++;cc+=yyleng;}
[" "] {sc++;cc+=yyleng;}
[^" "\n]+ {wc++;cc+=yyleng;}
%%
int main(int argc, char* argv[])
{
 printf("Enter input\n");
 yylex();
 printf("Number of lines=%d\n",lc);
 printf("Number of spaces=%d\n",sc);
 printf("Number of words=%d\n",wc);
 printf("Number of characters=%d\n",(cc-2));
}
int yywrap()
{
  return 1;
}
```

```
student@CSLAB4PC08:~/nicy$ flex file.l
student@CSLAB4PC08:~/nicy$ gcc lex.yy.c -lfl
student@CSLAB4PC08:~/nicy$ ./a.out
Enter input
hello world
hi superman
Number of lines=2
Number of spaces=2
Number of words=4
Number of characters=22
```

EXPERIMENT 6 LOWER CASE TO UPPERCASE

```
%{
#include <stdio.h>
%}

lower   [a-z]

%%
{lower} { printf("%c", yytext[0] - 32); }
[\t\n]+  { echo(); }
.       { echo(); }

%%
int yywrap() {
    return 1;
}

int main() {
    yylex();
    return 0;
}

void echo() {
    printf("%s", yytext);
}
```

```
student@CSNPLPC51:~/nicy$ ./a.out
hello world
HELLO WORLD
```

EXPERIMENT 7 NO OF VOWELS,CONSONANTS

```
%{
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 consonants:");
yylex();
printf("Number of vowels are: %d\n", vow_count);
printf("Number of consonants are: %d\n", const_count);
return 0;
}
```

```
student@CSNPLPC51:~/nicy$ flex vc.l
student@CSNPLPC51:~/nicy$ gcc lex.yy.c -lfl
student@CSNPLPC51:~/nicy$ ./a.out
Enter the string of vowels and consonants:computer science

Number of vowels are: 6
Number of consonants are: 9
```

Experiment 8: XYZ WHERE Y NOT EQUAL TO Z

```
%{
int a;
int b;
int c;
%}
a [012][0][12]
b [012][1][02]
c [012][2][01]
%%
{a}|{b}|{c} {printf("valid %s\n",yytext);}
.* {printf("invalid %s\n",yytext);}
%%
void main()
{
printf("Enter the string:");
yylex();
}
```

```
student@CSNPLPC51:~/nicy$ ./a.out
Enter the string:200
invalid 200
```

```
student@CSNPLPC51:~/nicy$ ./a.out
Enter the string:012
valid 012
```

ADDITIONAL EXP :

STRINGS OF 0’S AND 1’S WHICH STARTS WITH 0 FOLLOWED BY ZERO OR MORE COMBINATIONS OF 10 OR 01

```
%{
int a;
%}
a [0](01|10)*
%%
{a} {printf("valid %s\n",yytext);}
.* {printf("invalid %s\n",yytext);}
%%
void main()
{
 printf("ENTER STRING:\n");
```

```
 yylex();
}
```

```
student@CSNPLPC52:~/neha$ ./a.out
ENTER STRING:
000
invalid 000

010
valid 010
```

```
student@CSNPLPC51:~/nicy$ flex a2.l
student@CSNPLPC51:~/nicy$ gcc lex.yy.c -lfl
student@CSNPLPC51:~/nicy$ ./a.out
Enter the string:0101
invalid 0101

010
valid 010
```

STRINGS OF WITH EXACTLY 3 0's

```
%{
int a;
%}
a 1*[0]1*[0]1*[0]1*
%%
{a} {printf("valid %s\n",yytext);}
.* {printf("invalid %s\n",yytext);}
%%
void main()
{
 printf("ENTER STRING:\n");
 yylex();
}
```

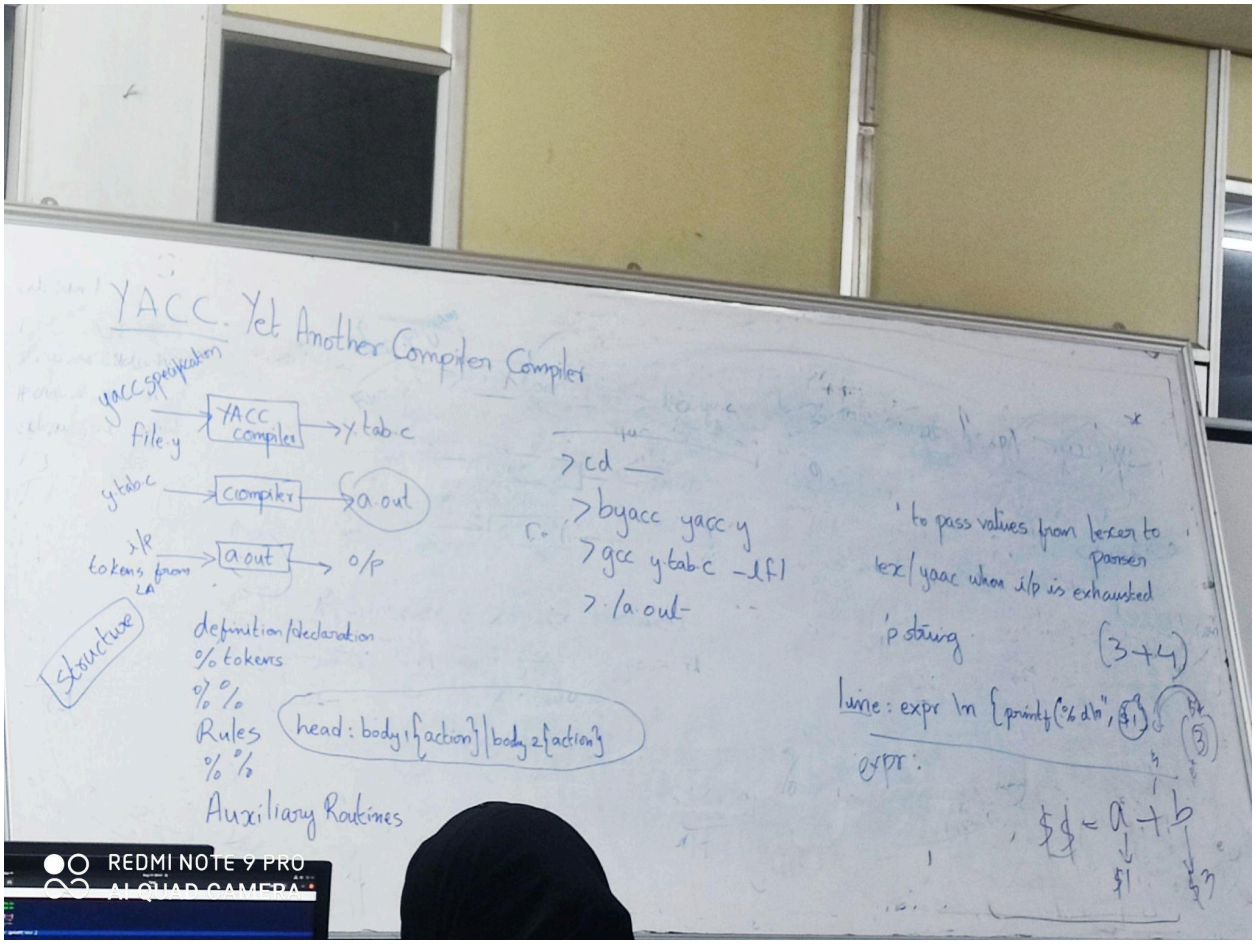
```
student@CSNPLPC51:~/nicy$ ./a.out
Enter the string:101010
valid 101010

00011
valid 00011

010100
invalid 010100
```

Experiment 9. Yacc

```
%{
#include<stdio.h>
#include<ctype.h>
%}
%token DIGIT
%left '-'"+'  //left associative
%left '/'"*'
%%
line:expr '\n' {printf("%d\n",$1);};
expr:expr'+'expr {$$=$1+$3;} //stores in $$ .
|expr'-'expr {$$=$1-$3;}
|expr'*'expr {$$=$1*$3;}
|expr'/'expr {$$=$1/$3;}
|'('expr')' {$$=($2);}
|'-'expr {$$=-$2;}
|DIGIT {$$=$1;};
%%

main()
{
```

```
 printf("Enter an expresion\n");
 yyparse();
}

yyerror(char *s)
{
 printf("%s",s);
}

int yylex(void)  //converts code to yac
{
 int c;
 c=getchar();
 if(isdigit(c))
 {
  yylval=c-'0';  //character to integer conversion.
  return DIGIT;
 }
 return c;
}
```

```
student@CSNPLPC51:~/nicy$ ./a.out
Enter an expresion
3+4
7
```



EXPERIMENT 10 (YACC AND LEX TOGETHER)

ly.l(lex pgm)

```
%{
#include"y.tab.h"
%}
%%
[a-zA-Z_] {return ALPHA;}
[0-9]+ {return DIGIT;}
"\n" {return ENTER;}
. {return ER;}
%%
yywrap()
{}
```

yl.y(yacc pgm)

```
%{
```

```
#include<stdio.h>
#include<stdlib.h>
%}
%token ALPHA DIGIT ENTER ER
%%
var:v ENTER {printf("valid Variable \n"); exit(0);}
v:ALPHA exp1
exp1:ALPHA exp1|DIGIT exp1|;
%%
yyerror()
{
printf("invalid variable\n");
}
main()
{
printf("enter the expression\n");
yyparse();
}
```

```
student@CSNPLPC51:~/nicy$ ./a.out
enter the expression
a3
valid Variable
student@CSNPLPC51:~/nicy$ ./a.out
enter the expression
23a
invalid variable
```

<u>EXPERIMENT 11 CALCULATOR PGM</u>

```
student@CSNPLPC49:~/nicy$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Divison, Modulus and Round brackets:
1+2

Result=3

Entered arithmetic expression is Valid
```

Lex.l

```
%{
#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;
}
```

Yacprg.y

```
%{

   #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;

}
```

EXPERIMENT 12

program to find e-closure of all states of given NFA with e-transition.

```
#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 print_e_closure(int);
static int set[20],nostate,noalpha,s,notransition,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("Enter 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 number of transitions\n");
    scanf("%d",&notransition);
    printf("NOTE:- [Transition is in the form -> qno alphabet qno]\n");
    printf("NOTE:- [States number be greater than zero\n");
    printf("Enter transitions\n");
    for(i=0;i<notransition;i++)
    {
        scanf("%d %c%d", &r,&c,&s);
        insert_trantbl(r,c,s);
    }
    printf("\n");
    printf("e-closure of states....\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("\ne-closure(q%d):",i);
        print_e_closure(i);
    }
```

```
}
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 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("}");}
```

```
ubuntu@ubuntu: ~/nicy
ubuntu@ubuntu:~/nicy$ ./a.out
Enter the no. of alphabets:
3
NOTE:-[use letter e as epsilon]
NOTE:-[e must be last character,if it's present]

Enter alphabets?
a
b
e
Enter the no. of states:
7

Enter the no. of transitions:
7
NOTE:-[Transition is in the form->qno alphabet qno]
NOTE:-[States number must be greater than zero]

Enter transitions:
1 e 2
2 e 3
3 e 6
1 e 4
4 a 5
```

```
ubuntu@ubuntu: ~/nicy
NOTE:-[Transition is in the form->qno alphabet qno]
NOTE:-[States number must be greater than zero]

Enter transitions:
1 e 2
2 e 3
3 e 6
1 e 4
4 a 5
5 b 6
5 e 7

e-closure of states...
____________

e-closure(q1):{q1, q4, q2, q3, q6}
e-closure(q2):{q2, q3, q6}
e-closure(q3):{q3, q6}
e-closure(q4):{q4}
e-closure(q5):{q5, q7}
e-closure(q6):{q6}
e-closure(q7):{q7}ubuntu@ubuntu:~/nicy$ gcc eclosure.c
ubuntu@ubuntu:~/nicy$ ./a.out
Enter the no. of alphabets:
```

```
Enter the number of alphabets
3
NOTE:- [use letter e as epsilon]
NOTE:- [e must be last character, if it is present]
Enter alphabets
0 1 e
Enter the number of states
4
Enter the number of transitions
6
NOTE:- [Transition is in the form -> qno alphabet qno]
NOTE:- [States number be greater than zero
Enter transitions
1 0 1
1 e 2
2 0 4
2 e 3
3 1 3
4 1 2

e-closure of states....

e-closure(q1):{q1,q2,q3,}
e-closure(q2):{q2,q3,}
e-closure(q3):{q3,}
e-closure(q4):{q4,}
```

EXPERIMENT -13 PROGRAM TO CONVERT NFA WITH EPSILON TRANSITION TO NFA WITHOUT EPSILON TRANSITIONS

```
student@CSNPLPC49:~/nicy$ ./a.out
enter the number of alphabets?
4
NOTE:- [ use letter e as epsilon]
NOTE:- [e must be last character ,if it is present]

Enter alphabets?
a b c e
Enter the number of states?
3
Enter the start state?
1
Enter the number of final states?
1
Enter the final states?
3
Enter no of transition?
5
NOTE:- [Transition is in the form--> qno alphabet qno]
NOTE:- [States number must be greater than zero]

Enter transition?
1 a 1
1 e 2
2 b 2
2 e 3
3 c 3

Equivalent NFA without epsilon
-----------------------------------
start state:{q1,q2,q3,}
Alphabets:a b c e
 States :{q1,q2,q3,}    {q2,q3,}        {q3,}
Tnransitions are...:

{q1,q2,q3,}     a       {q1,q2,q3,}
{q1,q2,q3,}     b       {q2,q3,}
{q1,q2,q3,}     c       {q3,}
{q2,q3,}        a       {}
{q2,q3,}        b       {q2,q3,}
{q2,q3,}        c       {q3,}
{q3,}   a       {}
{q3,}   b       {}
{q3,}   c       {q3,}
student@CSNPLPC49:~/nicy$
```

```
#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[2
0];
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 unionclosure(int i)
{
         int j=0,k;
        while(e_closure[i][j]!=0)
        {
                k=e_closure[i][j];
                set[k]=1;
                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");
}




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 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);
                                }
                        }
                }
        }
 }
```

EXPERIMENT 14 PRG TO CONVERT NFA TO DFA

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int st;
  struct node *link;
};
struct node1
{
 int nst[20];
};
void insert(int ,char, int);
```

```
int findalpha(char);
void findfinalstate(void);
int insertdfastate(struct node1);
int compare(struct node1,struct node1);
void printnewstate(struct node1);
static int
set[20],nostate,noalpha,s,notransition,nofinal,start,finalstate[20],c,r,buffer[2
0];
int complete=-1;
char alphabet[20];
static int eclosure[20][20]={0};
struct node1 hash[20];
struct node * transition[20][20]={NULL};
void main()
{
  int i,j,k,m,t,n,l;
  struct node *temp;
  struct node1 newstate={0},tmpstate={0};
  printf("Enter the number of alphabets?\n");
  printf("NOTE:- [ use letter e as epsilon]\n");
  printf("NOTE:- [e must be last character ,if it is present]\n");
  printf("\nEnter No of alphabets and alphabets?\n");
  scanf("%d",&noalpha);
  getchar();
  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(r,c,s);
  }
  for(i=0;i<20;i++)
  {
    for(j=0;j<20;j++)
     hash[i].nst[j]=0;
  }
  complete=-1;
  i=-1;
  printf("\nEquivalent DFA.....\n");
  printf(".......................\n");
  printf("Trnsitions of DFA\n");
  newstate.nst[start]=start;
  insertdfastate(newstate);
  while(i!=complete)
  {
    i++;
    newstate=hash[i];
    for(k=0;k<noalpha;k++)
    {
      c=0;
      for(j=1;j<=nostate;j++)
       set[j]=0;
```

```
for(j=1;j<=nostate;j++)
{
  l=newstate.nst[j];
  if(l!=0)
  {
    temp=transition[l][k];
    while(temp!=NULL)
    {
      if(set[temp->st]==0)
      {
        c++;
        set[temp->st]=temp->st;
      }
      temp=temp->link;
    }
  }
}
printf("\n");
if(c!=0)
{
  for(m=1;m<=nostate;m++)
    tmpstate.nst[m]=set[m];
  insertdfastate(tmpstate);
  printnewstate(newstate);
  printf("%c\t",alphabet[k]);
  printnewstate(tmpstate);
  printf("\n");
}
else
{
  printnewstate(newstate);
  printf("%c\t", alphabet[k]);
  printf("NULL\n");
}
```

```
    }
    }
  printf("\nStates of DFA:\n");
  for(i=0;i<=complete;i++)
  printnewstate(hash[i]);
  printf("\n Alphabets:\n");
  for(i=0;i<noalpha;i++)
   printf("%c\t",alphabet[i]);
  printf("\n Start State:\n");
   printf("q%d",start);
  printf("\nFinal states:\n");
  findfinalstate();
}

int insertdfastate(struct node1 newstate)
{
  int i;
  for(i=0;i<=complete;i++)
  {
    if(compare(hash[i],newstate))
     return 0;
  }
  complete++;
  hash[complete]=newstate;
  return 1;
}

int compare(struct node1 a,struct node1 b)
{
  int i;
   for(i=1;i<=nostate;i++)
   {
    if(a.nst[i]!=b.nst[i])
      return 0;
   }
```

```
   return 1;
}

void insert(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 findfinalstate()
{
  int i,j,k,t;
  for(i=0;i<=complete;i++)
  {
    for(j=1;j<=nostate;j++)
    {
```

```
      for(k=0;k<nofinal;k++)
      {
        if(hash[i].nst[j]==finalstate[k])
        {
          printnewstate(hash[i]);
          printf("\t");
          j=nostate;
          break;
        }
      }
    }
  }
}

void printnewstate(struct node1 state)
{
  int j;
  printf("{");
   for(j=1;j<=nostate;j++)
   {
     if(state.nst[j]!=0)
      printf("q%d,",state.nst[j]);
   }
   printf("}\t");
}
```

```
student@CSNPLPC51:~/nicy$ ./a.out
Enter the number of alphabets?
NOTE:- [ use letter e as epsilon]
NOTE:- [e must be last character ,if it is present]

Enter No of alphabets and alphabets?
2 a b
Enter the number of states?
4
Enter the start state?
1
Enter the number of final states?
2
Enter the final states?
3 4
Enter no of transition?
8
NOTE:- [Transition is in the form-> qno alphabet qno]
NOTE:- [States number must be greater than zero]

Enter transition?
1 a 1
1 b 1
1 a 2
2 b 2
2 a 3
3 a 4
3 b 4
4 b 3

Equivalent DFA.....
.......................
Trnsitions of DFA

{q1,}   a       {q1,q2,}

{q1,}   b       {q1,}

{q1,q2,}        a       {q1,q2,q3,}

{q1,q2,}        b       {q1,q2,}

{q1,q2,q3,}     a       {q1,q2,q3,q4,}

{q1,q2,q3,}     b       {q1,q2,q4,}
```

```
{q1,q2,q3,q4,}  a       {q1,q2,q3,q4,}

{q1,q2,q3,q4,}  b       {q1,q2,q3,q4,}

{q1,q2,q4,}     a       {q1,q2,q3,}

{q1,q2,q4,}     b       {q1,q2,q3,}

States of DFA:
{q1,}   {q1,q2,}        {q1,q2,q3,}     {q1,q2,q3,q4,}  {q1,q2,q4,}
 Alphabets:
a       b
 Start State:
q1
Final states:
{q1,q2,q3,}             {q1,q2,q3,q4,}          {q1,q2,q4,}
```

EXPERIMENT 15

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
int n,m,p,i=0,j=0;
char a[10][10],f[10];
void follow(char c);
void first(char c);
int main(){
 int i,z;
 char c,ch;
 printf("Enter the no of productions:\n");
 scanf("%d",&n);
 printf("Enter the productions:\n");
 for(i=0;i<n;i++)
  scanf("%s%c",a[i],&ch);
  do{
   m=0;
   printf("Enter the elements whose first & follow is to be found:");
```

```
    scanf("%c",&c);
    first(c);
    printf("First(%c)={",c);
    for(i=0;i<m;i++)
     printf("%c,",f[i]);
    printf("}\n");
    strcpy(f," ");
    m=0;
    follow(c);
    printf("Follow(%c)={",c);
    for(i=0;i<m;i++)
     printf("%c,",f[i]);
    printf("}\n");
    printf("Continue(0/1)?");
    scanf("%d%c",&z,&ch);
   }while(z==1);
  return(0);
}
void first(char c)
{
 int k;
 if(!isupper(c))
  f[m++]=c;
 for(k=0;k<n;k++)
 {
  if(a[k][0]==c)
  {
   if(a[k][2]=='$')
    follow(a[k][0]);
   else if(islower(a[k][2]))
    f[m++]=a[k][2];
   else first(a[k][2]);
  }
 }
}
```

```
void follow(char c)
{
 if(a[0][0]==c)
  f[m++]='$';
 for(i=0;i<n;i++)
 {
  for(j=2;j<strlen(a[i]);j++)
  {
   if(a[i][j]==c)
   {
    if(a[i][j+1]!='\0')
    first(a[i][j+1]);
    if(a[i][j+1]=='\0' && c!=a[i][0])
     follow(a[i][0]);
   }
  }
 }
}
```

```
student@CSNPLPC52:~/neha$ ./a.out
Enter the no of prooductions:
8
Enter the productions:
S=ABc
S=ghi
S=jkl
A=a
A=b
A=c
B=b
D=d
Enter the elements whose first & follow is to be found:S
First(S)={a,b,c,g,j,}
Follow(S)={$,}
Continue(0/1)?1
Enter the elements whose first & follow is to be found:A
First(A)={a,b,c,}
Follow(A)={b,}
Continue(0/1)?1
Enter the elements whose first & follow is to be found:B
First(B)={b,}
Follow(B)={c,}
Continue(0/1)?1
Enter the elements whose first & follow is to be found:D
First(D)={d,}
Follow(D)={}
Continue(0/1)?
```

EXPERIMENT 16 RECURSIVE DESCENT PARSER

```
#include<stdio.h>
#include<string.h>
int i=0,flag=1,len=0;
char buffer[100],ch;
void E();
void E_dash();
void T();
void T_dash();
void F();
void advance();
void advance()
{
 i++;
 if(i==len)
  flag=1;
  else
  return;
}
void error()
{
 flag=0;
 printf("Invalid\n");
 return;
}
void E()
{
 T();
 E_dash();
}
void E_dash()
{
 if(buffer[i]=='+')
 {
  advance();
```

```
  T();
  E_dash();
 }
 else
  return;
}
void T()
{
 F();
 T_dash();
}
void T_dash()
{
 if(buffer[i]=='*')
 {
  advance();
  F();
  T_dash();
 }
 else
  return;
}
void F()
{
 if(buffer[i]=='(')
 {
  advance();
  E();
  if(buffer[i]==')')
   advance();
 }
 else if(buffer[i]=='i')
  advance();
 else
  error();
```

```
}

void main()
{
 printf("The grammar
is:\nE->TE'\nE'->+TE'|e\nT->FT'\nT'->*FT'|e\nF->(E)|i\n");
 printf("Enter input string: ");
 gets(buffer);
 flag=0;
 len=strlen(buffer);
 E();
 if(i==len)
  flag=1;
 if(flag==1)
  printf("\nString is valid\n");
 else
  printf("\nString is invalid\n");
}
```

```
student@CSPL51:~/nicy$ ./a.out
The grammar is:
E->TE'
E'->+TE'|e
T->FT'
T'->*FT'|e
F->(E)|i
Enter input string: i+1
Invalid

String is invalid
student@CSPL51:~/nicy$ ./a.out
The grammar is:
E->TE'
E'->+TE'|e
T->FT'
T'->*FT'|e
F->(E)|i
Enter input string: i-1

String is invalid
student@CSPL51:~/nicy$ ./a.out
The grammar is:
E->TE'
E'->+TE'|e
T->FT'
T'->*FT'|e
F->(E)|i
Enter input string: i+i

String is valid
student@CSPL51:~/nicy$ bg
```

EXPERIMENT 17 SHIFT REDUCE PARSER

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int z = 0, i = 0, j = 0, c = 0;
char a[16], ac[20], stk[15], act[10];
void check()
{
strcpy(ac,"REDUCE TO E -> ");
for(z = 0; z < c; z++)
{
if(stk[z] == '4')
{
printf("%s4", ac);
stk[z] = 'E';
stk[z + 1] = '\0';
printf("\n$%s\t%s$\t", stk, a);
```

```
}
}

for(z = 0; z < c - 2; z++)
{
if(stk[z] == '2' && stk[z + 1] == 'E' && stk[z + 2] == '2')
{
printf("%s2E2", ac);
stk[z] = 'E';
stk[z + 1] = '\0';
stk[z + 2] = '\0';
printf("\n$%s\t%s$\t", stk, a);
i = i - 2;
}

}

for(z=0; z<c-2; z++)
{
if(stk[z] == '3' && stk[z + 1] == 'E' && stk[z + 2] == '3')
{
printf("%s3E3", ac);
stk[z]='E';
stk[z + 1]='\0';
stk[z + 1]='\0';
printf("\n$%s\t%s$\t", stk, a);
i = i - 2;
}
}
return ;
}

int main()
{
printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");
```

```
printf("Enter input string: ");
gets(a);
c=strlen(a);
strcpy(act,"SHIFT");
printf("\nstack \t input \t action");
printf("\n$\t%s$\t", a);
for(i = 0; j < c; i++, j++)
{
printf("%s", act);
stk[i] = a[j];
stk[i + 1] = '\0';
a[j]=' ';
printf("\n$%s\t%s$\t", stk, a);
check();
}
check();
if(stk[0] == 'E' && stk[1] == '\0')
printf("Accept\n");
else
printf("Reject\n");
}
```

```
student@CSPL51:~/nicy$ ./a.out
GRAMMAR is -
E->2E2
E->3E3
E->4
Enter input string: 242

stack    input   action
$       242$    SHIFT
$2       42$    SHIFT
$24       2$    REDUCE TO E -> 4
$2E       2$    SHIFT
$2E2       $    REDUCE TO E -> 2E2
$E         $    Accept
student@CSPL51:~/nicy$ ./a.out
GRAMMAR is -
E->2E2
E->3E3
E->4
Enter input string: 243

stack    input   action
$       243$    SHIFT
$2       43$    SHIFT
$24       3$    REDUCE TO E -> 4
$2E       3$    SHIFT
$2E3       $    Reject
student@CSPL51:~/nicy$ z
```

EXPERIMENT 18: CONSTANT PROPAGATION

```
#include <stdio.h>
#include<string.h>
#include<ctype.h>
void input();
void output();
void change(int p,char *res);
void constant();
struct expr{
   char op[2],op1[5],op2[5],res[5];
}arr[10];
int n;
void main()
```

```
{
    input();
    constant();
    output();
}
void input(){
    int i;
    printf("\n\nEnter the maximum number of expressions: ");
    scanf("%d",&n);
    printf("\nEnter the input:\n");
    for(int i=0;i<n;i++){
        scanf("%s",arr[i].op);
        scanf("%s",arr[i].op1);
        scanf("%s",arr[i].op2);
        scanf("%s",arr[i].res);
    }
}
void output(){
    int i=0;
    printf("\nOptimised code is: ");
    for(i=0;i<n;i++){
            printf("\n%s %s %s %s", arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
    }
}
void change(int p,char *res){
    int i;
    for(i=p+1;i<n;i++){
        if(strcmp(arr[p].res,arr[i].op1)==0)
            strcpy(arr[i].op1,res);
        else if(strcmp(arr[p].res,arr[i].op2)==0)
            strcpy(arr[i].op2,res);
    }
}
void constant(){
    int i;
```

```
    int op1,op2,res;
    char op,res1[5];
    for(i=0;i<n;i++){
        if(isdigit(arr[i].op1[0])&&isdigit(arr[i].op2[0])||strcmp(arr[i].op,"=")==0){
            op1=atoi(arr[i].op1);
            op2=atoi(arr[i].op2);
            op=arr[i].op[0];
            switch(op){
                case '+':
                res=op1+op2;
                break;
                case '-':
                res=op1-op2;
                break;
                case '*':
                res=op1*op2;
                break;
                case '/':
                res=op1/op2;
                break;
                case '=':
                res=op1;
                break;
            }
            sprintf(res1,"%d",res);
            change(i,res1);
        }
    }
}
```

```
Enter the maximum number of expressions: 4

Enter the input:
= 4 - a
= 3 - b
+ a b c
- b a d

Optimised code is:
= 4 - a
= 3 - b
+ 4 3 c
- 3 4 d
```

EXPERIMENT 19: INTERMEDIATE CODE GENERATION

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp{
   int pos;
   char op;
}k[15];
void main()
{
   printf("\t\tINTERMEDIATE CODE GENERATION\n\n");
   printf("Enter the expression: ");
   scanf("%s",str);
```

```
    printf("The intermediate code:\n");
    findopr();
    explore();
}
void findopr(){
    for(i=0;str[i]!='\0';i++)
    {
        if(str[i]==':'){
            k[j].pos=i;
            k[j++].op=':';
        }
    }
    for(i=0;str[i]!='\0';i++){
        if(str[i]=='/'){
            k[j].pos=i;
            k[j++].op='/';
        }
    }
    for(i=0;str[i]!='\0';i++){
        if(str[i]=='*'){
            k[j].pos=i;
            k[j++].op='*';
        }
    }
    for(i=0;str[i]!='\0';i++){
        if(str[i]=='+'){
            k[j].pos=i;
            k[j++].op='+';
        }
    }
    for(i=0;str[i]!='\0';i++){
        if(str[i]=='-'){
            k[j].pos=i;
            k[j++].op='-';
        }
```

```
    }
}
void explore(){
    i=1;
    while(k[i].op!='\0'){
        fleft(k[i].pos);
        fright(k[i].pos);
        str[k[i].pos]=tmpch--;
        printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
        printf("\n");
        i++;
    }
    fright(-1);
    if(no==0){
        fleft(strlen(str));
        printf("\t%s := %s",right,left);
        exit(0);
    }
    printf("\t%s := %c",right,str[k[--i].pos]);
}
void fleft(int x){
    int w=0,flag=0;
    x--;

while(x!=-1&&str[x]!='+'&&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&st
r[x]!='/'&&str[x]!=':'){
        if(str[x]!='$'&&flag==0){
            left[w++]=str[x];
            left[w]='\0';
            str[x]='$';
            flag=1;
        }
        x--;
    }
}
```

```
void fright(int x){
   int w=0,flag=0;
   x++;

while(x!=-1&&str[x]!='+'&&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&st
r[x]!='/'&&str[x]!=':'){
      if(str[x]!='$'&&flag==0){
         right[w++]=str[x];
         right[w]='\0';
         str[x]='$';
         flag=1;
      }
      x++;
   }
}
```

```
                    INTERMEDIATE CODE GENERATION

Enter the expression: w:=a*b+c*d-f
The intermediate code:
        Z := a*b
        Y := c*d
        X := Z+Y
        W := X-f
        w := W
```

EXPERIMENT 20 IMPLEMENT BACKEND OF COMPILER USING 8086 ASSEMBLER

```
#include <stdio.h>
#include<string.h>
void main()
{
   char icode[10][30],str[20],opr[10];
   int i=0;
   printf("\nEnter the set of intermediate code(terminated by exit):\n");
```

```
    do{
        scanf("%s",icode[i]);
    }while(strcmp(icode[i++],"exit")!=0);
    printf("\ntarget code generation");
    printf("\n*********************");
    i=0;
    do{
        strcpy(str,icode[i]);
        switch(str[3]){
            case '+':
            strcpy(opr,"ADD");
            break;
            case '-':
            strcpy(opr,"SUB");
            break;
            case '*':
            strcpy(opr,"MUL");
            break;
            case '/':
            strcpy(opr,"DIV");
            break;
        }
        printf("\n\tMov %c,R%d",str[2],i);
        printf("\n\t%s%c,R%d",opr,str[4],i);
        printf("\n\tMov R%d,%c",i,str[0]);
    }while(strcmp(icode[++i],"exit")!=0);
}
```

```
Enter the set of intermediate code(terminated by exit):
d=z/3
c=4/5
a=2*e
exit

target code generation
**********************
        Mov z,R0
        DIV3,R0
        Mov R0,d
        Mov 4,R1
        DIV5,R1
        Mov R1,c
        Mov 2,R2
        MULe,R2
        Mov R2,a
```