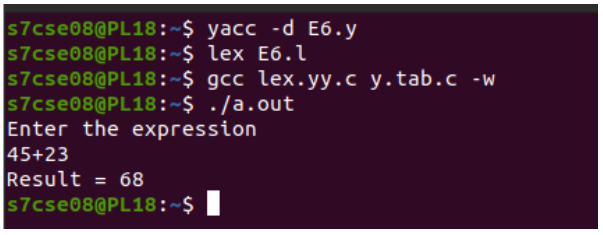
E6.l

```
%{
   /* Definition section*/
   #include "y.tab.h"
   extern yylval;
%}

%%
[0-9]+    {
            yylval = atoi(yytext);
            return NUMBER;
          }


[ \t]+         ;  /*For skipping whitespaces*/

\n            { return 0; }
.           { return yytext[0]; }

%%
yywrap()
{
}
```

E6.Y

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

%token NUMBER
// 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 { $$ = -$2; }
   | '(' T ')' { $$ = $2; }
   | NUMBER { $$ = $1; }
%%

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

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

Output:



E7.L

```
---------------------]
%{
   /* Definition section*/
   #include "y.tab.h"
   extern yylval;
%}

%%
[A-Z a-z]+ {
       yylval = atoi(yytext);
       return LETTER;
}


[0-9]+    {
            yylval = atoi(yytext);
            return NUMBER;
          }


[ \t]+         ;  /*For skipping whitespaces*/

\n            { return 0; }
.           { return yytext[0]; }

%%
yywrap()
{
}
```

E7.Y

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

%token NUMBER, LETTER

%%
E : LETTER T      {
            printf("Valid Identifier\n");
            return 0;
          }

T :
   LETTER NUMBER
   |NUMBER LETTER
   |LETTER
   | NUMBER
   |;

%%

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

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

Output:

```
s7cse08@PL18:~$ yacc -d E7.y
s7cse08@PL18:~$ lex E7.l
s7cse08@PL18:~$ gcc lex.yy.c y.tab.c -w
s7cse08@PL18:~$ ./a.out
Enter the expression
abc1232
Valid Identifier
s7cse08@PL18:~$ ./a.out
Enter the expression
123ad

Expression is invalid
s7cse08@PL18:~$ ./a.out
Enter the expression
a
Valid Identifier
s7cse08@PL18:~$
```

E8.L

```
%{
   /* Definition section*/
   #include "y.tab.h"
   extern yylval;
%}

%%

[0-9]+    {
            yylval = atoi(yytext);
            return NUMBER;
          }


[ \t]+         ;  /*For skipping whitespaces*/

\n            { return 0; }
.           { return yytext[0]; }

%%
yywrap()
{
}
```

E8.Y

```
%{
   /* Definition section */
  #include <stdio.h>
%}
%token NUMBER
// setting the precedence
// and associativity of operators
%left '+' '-'
%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 { $$ = $1 / $3; }
   | T '%' T { $$ = $1 % $3; }
   | '-' NUMBER { $$ = -$2; }
   | '(' T ')' { $$ = $2; }
   | NUMBER { $$ = $1; }
%%
int main() {
   printf("Enter the expression\n");
   yyparse();
}
yyerror()
{
   printf("\nExpression is invalid\n");
}
```

Output:

```
s7cse08@PL18:~$ yacc -d exp8.y
s7cse08@PL18:~$ lex exp8.l
s7cse08@PL18:~$ gcc lex.yy.c y.tab.c -w
s7cse08@PL18:~$ ./a.out
Enter the expression
2*3/2+1
Result = 4
s7cse08@PL18:~$ ./a.out
Enter the expression
4%2
Result = 0
s7cse08@PL18:~$
```

E9

```
#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("Epsilon closure of %s = { ", copy);
  while (k < n) {
    printf(" %s", result[k]);
    k++;
  }
  printf(" } \n");
}


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("Enter the no of states: \n");
  scanf("%d", & n);
  printf("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;
}
```

Output:

```
s7cse08@PL18:~$ gcc exp9.c
s7cse08@PL18:~$ ./a.out
Enter the no of states:
3
Enter the states :
Q0 Q1 Q2
Epsilon closure of Q0 = {  Q0 }
Epsilon closure of Q1 = {  Q1 }
Epsilon closure of Q2 = {  Q2 }
s7cse08@PL18:~$
```

E10

```
#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");
}
```

Output:

```
s7cse08@PL18:~$ ./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,}
```

E11

```
--------------
#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,fi
nalstate[20],c,r,buffer[20];
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("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");
}
```

Output:

```
s7cse08@PL18:~$ ./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?
3
Enter the start state?
1
Enter the number of final states?
1
Enter the final states?
3
Enter no of transition?
4
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 3

Equivalent DFA.....
Trnsitions of DFA

{q1,}   a       {q1,q2,}

{q1,}   b       {q1,}

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

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

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

{q1,q3,}        b       {q1,}

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


LITEON
SAA-1005-E4
SAFETY
19012-11
Rev.03
ADT
KPI1000170032C0E8F8E2