```
#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[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);
```

```
 66     insert(r,c,s);
 67
 68    }
 69    for(i=0;i<20;i++)
 70    {
 71     for(j=0;j<20;j++)
 72     hash[i].nst[j]=0;
 73    }
 74    complete=-1;
 75    i=-1;
 76    printf("\nEquivalent DFA.....\n");
 77    printf(".....\n");
 78
 79    printf("Trnsitions of DFA\n");
 80
 81    newstate.nst[start]=start;
 82    insertdfastate(newstate);
 83    while(i!=complete)
 84    {
 85     i++;
 86     newstate=hash[i];
 87     for(k=0;k<noalpha;k++)
 88     {
 89      c=0;
 90      for(j=1;j<=nostate;j++)
 91      set[j]=0;
 92      for(j=1;j<=nostate;j++)
 93      {
 94       l=newstate.nst[j];
 95       if(l!=0)
 96       {
 97        temp=transition[l][k];
 98        while(temp!=NULL)
 99        {
100         if(set[temp->st]==0)
101         {
102          c++;
103          set[temp->st]=temp->st;
104         }
105         temp=temp->link;
106
107
108        }
109       }
110      }
111      printf("\n");
112      if(c!=0)
113      {
114       for(m=1;m<=nostate;m++)
115        tmpstate.nst[m]=set[m];
116
117       insertdfastate(tmpstate);
118
119       printnewstate(newstate);
120       printf("%c\t",alphabet[k]);
121       printnewstate(tmpstate);
122       printf("\n");
123       }
124       else
125       {
126       printnewstate(newstate);
127       printf("%c\t", alphabet[k]);
128       printf("NULL\n");
129       }
130
131     }
```

```
132       }
133      printf("\nStates of DFA:\n");
134      for(i=0;i<=complete;i++)
135      printnewstate(hash[i]);
136      printf("\n Alphabets:\n");
137      for(i=0;i<noalpha;i++)
138      printf("%c\t",alphabet[i]);
139      printf("\n Start State:\n");
140       printf("q%d",start);
141      printf("\nFinal states:\n");
142      findfinalstate();
143
144     }
145     int insertdfastate(struct node1 newstate)
146     {
147      int i;
148      for(i=0;i<=complete;i++)
149      {
150       if(compare(hash[i],newstate))
151        return 0;
152      }
153      complete++;
154      hash[complete]=newstate;
155      return 1;
156     }
157     int compare(struct node1 a,struct node1 b)
158     {
159      int i;
160
161
162       for(i=1;i<=nostate;i++)
163       {
164        if(a.nst[i]!=b.nst[i])
165          return 0;
166       }
167       return 1;
168
169
170     }
171
172     void insert(int r,char c,int s)
173     {
174            int j;
175            struct node *temp;
176            j=findalpha(c);
177            if(j==999)
178            {
179       printf("error\n");
180       exit(0);
181            }
182            temp=(struct node *) malloc(sizeof(struct node));
183            temp->st=s;
184            temp->link=transition[r][j];
185            transition[r][j]=temp;
186     }
187
188     int findalpha(char c)
189     {
190      int i;
191      for(i=0;i<noalpha;i++)
192      if(alphabet[i]==c)
193       return i;
194
195       return(999);
196
197
```

```
198  }
199
200
201  void findfinalstate()
202  {
203   int i,j,k,t;
204
205    for(i=0;i<=complete;i++)
206    {
207    for(j=1;j<=nostate;j++)
208    {
209     for(k=0;k<nofinal;k++)
210     {
211      if(hash[i].nst[j]==finalstate[k])
212      {
213       printnewstate(hash[i]);
214       printf("\t");
215       j=nostate;
216       break;
217      }
218     }
219    }
220    }
221  }
222
223
224  void printnewstate(struct node1 state)
225  {
226   int j;
227   printf("{");
228    for(j=1;j<=nostate;j++)
229    {
230     if(state.nst[j]!=0)
231      printf("q%d,",state.nst[j]);
232    }
233    printf("}\t");
234
235  }
236  /*OUTPUT
237  Enter the number of alphabets?
238  NOTE:- [ use letter e as epsilon]
239  NOTE:- [e must be last character ,if it is present]
240
241  Enter No of alphabets and alphabets?
242  2
243  a
244  b
245  Enter the number of states?
246  4
247  Enter the start state?
248  1
249  Enter the number of final states?
250  1
251  Enter the final states?
252  4
253  Enter no of transition?
254  8
255  NOTE:- [Transition is in the form-> qno alphabet qno]
256  NOTE:- [States number must be greater than zero]
257
258  Enter transition?
259  1 a 1
260  1 b 1
261  1 a 2
262  2 b 2
263  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,q4,}          {q1,q2,q4,}
PS D:\COMPILER LAB>*/
```