Program Code

```
#include<stdio.h>

#include<stdlib.h>

struct node {
  int st;
  struct node * link;
};
struct node1 {
  int nst[20];
};

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};

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;
}

int insert_dfa_state(struct node1 newstate)
{
  int i;
  for (i = 0; i <= complete; i++)
  {
    //checking if the state was already added
    if (compare(hash[i], newstate))
      return 0;
  }
  complete++;

  //marking the new state as completed
  hash[complete] = newstate;
  return 1;
}

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

  return (999);
}
```

```
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;
}

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

}

void main()
{
  int i, j, k, m, t, n, l;
  struct node * temp;
  struct node1 newstate = {0},tmpstate = {0};

  printf("\nEnter No of alphabets : ");
  scanf("%d",&noalpha);
  printf("\nEnter the alphabet: ");
  for (i = 0; i < noalpha; i++)
  {
    scanf(" %c",&alphabet[i]);
  }
  printf("\nEnter the number of states :");
  scanf("%d", & nostate);
  printf("\nEnter the start state :");
  scanf("%d", & start);
  printf("\nEnter the number of final states :");
  scanf("%d", & nofinal);
  printf("\nEnter the final states :");
  for (i = 0; i < nofinal; i++)
    scanf("%d",&finalstate[i]);
  printf("\nEnter no of transition :");

  scanf("%d", & notransition);

  printf("\nEnter transition :");


  //Create the transition table

  char y;
  for (i = 0; i < notransition; i++)
```

```
{
  scanf("%d %c%d", & r, & y, & s);
  insert(r, y, 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");

//adding initial state
newstate.nst[start] = start;
insert_dfa_state(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];

      insert_dfa_state(tmpstate);
      printf("(");
      printnewstate(newstate);
      printf("|%c", alphabet[k]);
      printf(")=>");
      printnewstate(tmpstate);
      printf("\n");
    }
    else
```

```
        {
          printf("(");
          printnewstate(newstate);
          printf("|%c", alphabet[k]);
          printf(")=>");
          printf("NULL\n");
        }

      }
    }
}
```

Output