PROGRAM

```
#include<stdio.h>
#include<stdlib.h>

struct node{
   int state;
   char alpha;
   struct node *link;
}*transition[20]; //array to store the transitions

char alpha[26];  // array store the alphabets
int n;  //no. of states
int nalpha; //no. of alp[habets
int nfinal;  //no. of final states
struct node *final = NULL; //set of final states
int ntransitions; //no. of transitions
struct node *merged[20]; //store the states of minimized dfa;
int count; // store the no of states in the minimized dfa

int checkFinal(int s)
{
   struct node *temp = final;
   while(temp != NULL)
   {
      if(temp->state == s)
         return 1;
      temp = temp->link;
   }
   return -1;
}

int mergeGroup(int s)
{
   int found = 0;
   for(int i=0;i<n;i++)
   {
      struct node *temp = merged[i];
      while(temp != NULL)
      {
         if(temp->state == s)
         {
            return i;
            found = 1;
         }
```

```
            temp = temp->link;
        }
    }
    if(found==0)
        return s;
}

int retState(int q,char c)
{
    struct node * temp = transition[q];
    while(temp!=NULL)
    {
        if(temp->alpha == c)
            return mergeGroup(temp->state);
        temp = temp->link;
    }
    return -1;
}

void minimizeDFA()  //minimize the dfa
{

    for(int i=0;i<20;i++)
    {
        merged[i] = NULL;
    }

    for(int i =0;i<n;i++)
    {
        struct node *newnode = (struct node* ) malloc(sizeof(struct node));
        newnode->state = i;
        newnode->link = NULL;
        merged[i] = newnode;
    }

    int merge = 0;

    do
    {
        merge = 0;
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                if(mergeGroup(i) != mergeGroup(j))
```

```
            {
                int check = 1;
                for(int k = 0;k<nalpha;k++)
                {
                    if(retState(i,alpha[k]) != retState(j,alpha[k]))
                    {
                        if(retState(i,alpha[k])==mergeGroup(j) && retState(j,alpha[k])==mergeGroup(i))
                            continue;
                        else
                        {
                            check = 0;
                            break;
                        }
                    }
                }

                if(check == 1)
                {
                    int s = mergeGroup(j);
                    struct node *temp = merged[mergeGroup(i)];
                    if(temp==NULL)
                    {
                        temp = merged[s];
                        merged[s] = NULL;
                    }
                    else
                    {
                        while(temp->link != NULL)
                        {
                            temp = temp->link;
                        }
                        temp->link = merged[s];
                        merged[s] = NULL;
                    }
                    merge = 1;
                }
            }
        }
    }while(merge == 1);
}

void printState(int s)  //print minimized state
{
```

```
    struct node  *temp = merged[s];
    if(temp != NULL)
    {
        printf("{ ");
        while(temp != NULL )
        {
            printf("q%d ",temp->state);
            temp = temp->link;
        }
        printf("}");
    }
}
void main()
{
    int q1,q2;
    char c;
    printf("\nEnter the no. of states: ");
    scanf("%d",&n);

    printf("\nEnter the no. of alphabets: ");
    scanf("%d",&nalpha);

    printf("\nEnter the alphabets:\n");
    for(int i =0;i<nalpha;i++)
        scanf(" %c",&alpha[i]);

    printf("\nEnter the no. of final states: ");
    scanf("%d",&nfinal);

    struct node *temp = final ;
    printf("\nEnter the final states:\n");
    for (int i =0;i<nfinal;i++)
    {
        struct node *newnode =(struct node* ) malloc(sizeof(struct node));
        scanf("%d",&(newnode->state));
        newnode->link = NULL;
        if(final == NULL)
            final = newnode;
        else
        {
            temp->link = newnode;
        }
        temp = newnode;

    }
```

```
printf("\nEnter the no. of transitions: ");
scanf("%d",&ntransitions);
printf("\nEnter the transitions:\n");
for(int i=0;i<ntransitions;i++)
{
   scanf("%d %c%d",&q1,&c,&q2);
   struct node *newnode = (struct node *) malloc(sizeof(struct node));
   newnode->alpha = c;
   newnode->state = q2;
   newnode->link = NULL;

   struct node* temp = transition[q1];
   if(transition[q1] == NULL)
      transition[q1] = newnode;
   else
   {
      temp->link = newnode;
   }
   temp = newnode;
}

minimizeDFA();

printf("The states of the minimized DFA are:\n\n");
for(int i =0;i<n;i++)
{
   printState(i);
   if(merged[i] != NULL)
      printf("\n");

}

printf("\nThe transitions are:\n\n");

for(int i=0;i<n;i++)
{
   if(merged[i] != NULL)
   {
      int s;
      for(int j =0;j<nalpha;j++)
      {
         struct node *temp = transition[merged[i]->state];
         while(temp!=NULL)
         {
            if(temp->alpha == alpha[j])
```

```
                {
                    printState(i);
                    printf(" %c ",alpha[j]);
                    s = mergeGroup(temp->state);
                    printState(s);
                    printf("\n\n");
                }
                temp = temp->link;
            }
        }


    }
}

printf("\nThe Final States are:\n\n");
int finalGroup[n];
for(int i=0;i<n;i++)
{
    finalGroup[i] = 0;
}
for(int i=0;i<nfinal;i++)
{
    int s;
    struct node *temp = final;
    while(temp != NULL)
    {
        s = mergeGroup(temp->state);
        if(finalGroup[s] == 0)
        {
            printState(s);
            printf("\n");
            finalGroup[s] = 1;
        }

        temp = temp->link;
    }
}
}
```

OUTPUT

```
Enter the no. of states: 6

Enter the no. of alphabets: 2

Enter the alphabets:
a b

Enter the no. of final states: 3

Enter the final states:
1 2 4

Enter the no. of transitions: 12

Enter the transitions:
0 a 3
0 b 1
1 a 2
1 b 5
2 a 2
2 b 5
3 a 0
3 b 4
4 a 2
4 b 5
5 a 5
5 b 5
The states of the minimized DFA are:

{ q1 q2 q4 }
{ q3 q0 }
{ q5 }
```

```
The transitions are:

{ q1 q2 q4 } a { q1 q2 q4 }

{ q1 q2 q4 } b { q5 }

{ q3 q0 } a { q3 q0 }

{ q3 q0 } b { q1 q2 q4 }

{ q5 } a { q5 }

{ q5 } b { q5 }


The Final States are:

{ q1 q2 q4 }

E:\Semester 7\Compiler Design Lab\Programs>
```