Program :

```
// Program to find Epsilon-closure of all states of any given NFA with Epsilon transitions.
#include<stdio.h>
#include<stdlib.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("\n Epsilon closure of %s = { ",copy);
	while(k < n)
	{
		printf("%s",result[k]);
		k++;
	}
	printf(" } \n");
}

int main()
{
	FILE *INPUT;
	char state[3];
	int end,i=0,n,k=0,trans;
	char state1[3],input[3],state2[3];
	printf("\nEnter the number of states : ");
	scanf("%d",&n);
	printf("Enter the states : ");
	for(k=0;k<n;k++)
		scanf("%s",states[k]);
	INPUT = fopen("trans.table","w");
	if(INPUT == NULL)
	{
		printf("Error!");
		exit(1);
	}
	printf("Enter the number of transations : ");
	scanf("%d",&trans);
	printf("Enter transitions {state1 || input || state2 }\n");
	for(i=0;i<trans;i++)
	{
		scanf("%s%s%s",state1,input,state2);
		fprintf(INPUT,"%s\t%s\t%s\n",state1,input,state2);
	}
	fclose(INPUT);
```

```
        INPUT=fopen("trans.table","r");
        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;
}
//End of the program
```

Output :

```
:~/Desktop/Lab$ gcc 20.c
:~/Desktop/Lab$ ./a.out

Enter the number of states : 3
Enter the states : 1 2 3
Enter the number of transations : 4
Enter transitions {state1 || input || state2 }
1 e 2
2 0 2
2 e 3
3 1 3

 Epsilon closure of 1 = { 123 }

 Epsilon closure of 2 = { 23 }

 Epsilon closure of 3 = { 3 }
:~/Desktop/Lab$
```

Program :

```
//Program to convert NFA with epsilon transitions to NFA without epsilon transitions.
#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("\nEnter the number of alphabets : ");
      scanf("%d",&noalpha);
      getchar();
      printf("\nNOTE:- [use letter e as epsilon]\n");
      printf("NOTE:- [e must be last character ,if it is present]\n");
        printf("\nEnter alphabets : ");
      for(i=0;i<noalpha;i++)
      {
              alphabet[i]=getchar();
          getchar();
      }
      printf("Enter the number of states : ");
      scanf("%d",&nostate);
      printf("Enter the start state : ");
      scanf("%d",&start);
      printf("Enter the number of final states : ");
      scanf("%d",&nofinal);
      printf("Enter the final states : ");
      for(i=0;i<nofinal;i++)
          scanf("%d",&finalstate[i]);
      printf("Enter number of transition : ");
      scanf("%d",&notransition);
      printf("\nNOTE:- [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("\nStates : " );
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("\nFinal states:");
    findfinalstate();
    printf("\n");
}

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");
}
//End of the program
```

Output :

```
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 : 3
Enter the start state : 1
Enter the number of final states : 1
Enter the final states : 3
Enter number of transition : 5

NOTE:- [Transition is in the form--> qno   alphabet   qno]
NOTE:- [States number must be greater than zero]

Enter transition :
1 0 1
1 e 2
2 1 2
2 e 3
3 0 3

Equivalent NFA without epsilon
-----------------------------------
Start state : {q1,q2,q3,}
Alphabets: 0 1 e
States : {q1,q2,q3,}    {q2,q3,}        {q3,}
Tnransitions are :

{q1,q2,q3,}     0       {q1,q2,q3,}
{q1,q2,q3,}     1       {q2,q3,}
{q2,q3,}        0       {q3,}
{q2,q3,}        1       {q2,q3,}
{q3,}   0       {q3,}
{q3,}   1       {}
Final states:{q1,q2,q3,}        {q2,q3,}        {q3,}
:~/Desktop/Lab$
```

Program :

```
//Program to convert e-nfa to DFA
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LEN 100

char NFA_FILE[MAX_LEN];
char buffer[MAX_LEN];
int zz = 0;

struct DFA
{
	char *states;
	int count;
} dfa;

int last_index = 0;
FILE *fp;
int symbols;

void reset(int ar[], int size)
{
	int i;
	for (i = 0; i < size; i++)
	{
		ar[i] = 0;
	}
}

void check(int ar[], char S[])
{
	int i, j;
	int len = strlen(S);
	for (i = 0; i < len; i++)
	{
		j = ((int)(S[i]) - 65);
		ar[j]++;
	}
}

void state(int ar[], int size, char S[])
{
	int j, k = 0;
	for (j = 0; j < size; j++)
	{
		if (ar[j] != 0)
			S[k++] = (char)(65 + j);
	}
	S[k] = '\0';
}
```

```
int closure(int ar[], int size)
{
	int i;
	for (i = 0; i < size; i++)
	{
		if (ar[i] == 1)
			return i;
	}
	return (100);
}

int indexing(struct DFA *dfa)
{
	int i;
	for (i = 0; i < last_index; i++)
	{
		if (dfa[i].count == 0)
			return 1;
	}
	return -1;
}

void Display_closure(int states, int closure_ar[],char *closure_table[],char *NFA_TABLE[]
[symbols + 1],char *DFA_TABLE[][symbols])
{
	int i;
	for (i = 0; i < states; i++)
	{
		reset(closure_ar, states);
		closure_ar[i] = 2;
		if (strcmp(&NFA_TABLE[i][symbols], "-") != 0)
		{
			strcpy(buffer, &NFA_TABLE[i][symbols]);
			check(closure_ar, buffer);
			int z = closure(closure_ar, states);
			while (z != 100)
			{
				if (strcmp(&NFA_TABLE[z][symbols], "-") != 0)
				{
					strcpy(buffer, &NFA_TABLE[z][symbols]);
					check(closure_ar, buffer);
				}
				closure_ar[z]++;
				z = closure(closure_ar, states);
			}
		}
		printf("\n e-Closure (%c) :\t", (char)(65 + i));
		bzero((void *)buffer, MAX_LEN);
		state(closure_ar, states, buffer);
		strcpy(&closure_table[i], buffer);
		printf("%s\n", &closure_table[i]);
```

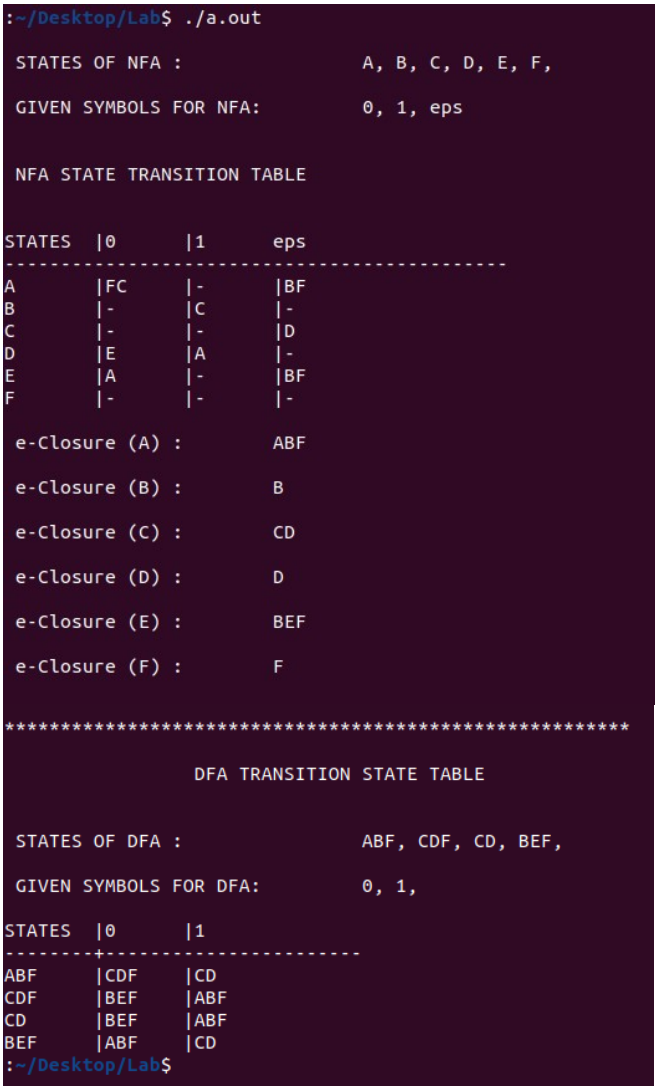
```
	}
}

int new_states(struct DFA *dfa, char S[])
{
	int i;
	for (i = 0; i < last_index; i++)
	{
		if (strcmp(&dfa[i].states, S) == 0)
		return 0;
	}
	strcpy(&dfa[last_index++].states, S);
	dfa[last_index - 1].count = 0;
	return 1;
}

void trans(char S[], int M, char *clsr_t[], int st,char *NFT[][symbols + 1], char TB[])
{
	int len = strlen(S);
	int i, j, k, g;
	int arr[st];
	int sz;
	reset(arr, st);
	char temp[MAX_LEN], temp2[MAX_LEN];
	char *buff;
	for (i = 0; i < len; i++)
	{
		j = ((int)(S[i] - 65));
		strcpy(temp, &NFT[j][M]);
		if (strcmp(temp, "-") != 0)
		{
			sz = strlen(temp);
			g = 0;
			while (g < sz)
			{
				k = ((int)(temp[g] - 65));
				strcpy(temp2, &clsr_t[k]);
				check(arr, temp2);
				g++;
			}
		}
	}
	bzero((void *)temp, MAX_LEN);
	state(arr, st, temp);
	if (temp[0] != '\0')
	{
		strcpy(TB, temp);
	}
	else
		strcpy(TB, "-");
}
```

```
void Display_DFA(int last_index, struct DFA *dfa_states,char *DFA_TABLE[][symbols])
{
	int i, j;
	printf("\n\n**********************************************************\n\n");
	printf("\t\t DFA TRANSITION STATE TABLE \t\t \n\n");
	printf("\n STATES OF DFA :\t\t");
	for (i = 1; i < last_index; i++)
		printf("%s, ", &dfa_states[i].states);
	printf("\n");
	printf("\n GIVEN SYMBOLS FOR DFA: \t");
	for (i = 0; i < symbols; i++)
		printf("%d, ", i);
	printf("\n\n");
	printf("STATES\t");
	for (i = 0; i < symbols; i++)
		printf("|%d\t", i);
	printf("\n");
	printf("--------+----------------------\n");
	for (i = 0; i < zz; i++)
	{
		printf("%s\t", &dfa_states[i + 1].states);
		for (j = 0; j < symbols; j++)
		{
			printf("|%s \t", &DFA_TABLE[i][j]);
		}
		printf("\n");
	}
}
int main()
{
	int i, j, states;
	char T_buf[MAX_LEN];
	struct DFA *dfa_states = malloc(MAX_LEN * (sizeof(dfa)));
	states = 6, symbols = 2;
	printf("\n STATES OF NFA :\t\t");
	for (i = 0; i < states; i++)
		printf("%c, ", (char)(65 + i));
	printf("\n");
	printf("\n GIVEN SYMBOLS FOR NFA: \t");
	for (i = 0; i < symbols; i++)
		printf("%d, ", i);
	printf("eps");
	printf("\n\n");
	char *NFA_TABLE[states][symbols + 1];
	char *DFA_TABLE[MAX_LEN][symbols];
	strcpy(&NFA_TABLE[0][0], "FC");
	strcpy(&NFA_TABLE[0][1], "-");
	strcpy(&NFA_TABLE[0][2], "BF");
	strcpy(&NFA_TABLE[1][0], "-");
	strcpy(&NFA_TABLE[1][1], "C");
	strcpy(&NFA_TABLE[1][2], "-");
	strcpy(&NFA_TABLE[2][0], "-");
```

```
strcpy(&NFA_TABLE[2][1], "-");
strcpy(&NFA_TABLE[2][2], "D");
strcpy(&NFA_TABLE[3][0], "E");
strcpy(&NFA_TABLE[3][1], "A");
strcpy(&NFA_TABLE[3][2], "-");
strcpy(&NFA_TABLE[4][0], "A");
strcpy(&NFA_TABLE[4][1], "-");
strcpy(&NFA_TABLE[4][2], "BF");
strcpy(&NFA_TABLE[5][0], "-");
strcpy(&NFA_TABLE[5][1], "-");
strcpy(&NFA_TABLE[5][2], "-");
printf("\n NFA STATE TRANSITION TABLE \n\n\n");
printf("STATES\t");
for (i = 0; i < symbols; i++)
        printf("|%d\t", i);
printf("eps\n");
printf("-----------------------------------------\n");
for (i = 0; i < states; i++)
{
        printf("%c\t", (char)(65 + i));
        for (j = 0; j <= symbols; j++)
        {
                printf("|%s \t", &NFA_TABLE[i][j]);
        }
        printf("\n");
}
int closure_ar[states];
char *closure_table[states];
Display_closure(states, closure_ar, closure_table, NFA_TABLE, DFA_TABLE);
strcpy(&dfa_states[last_index++].states, "-");
dfa_states[last_index - 1].count = 1;
bzero((void *)buffer, MAX_LEN);
strcpy(buffer, &closure_table[0]);
strcpy(&dfa_states[last_index++].states, buffer);
int Sm = 1, ind = 1;
int start_index = 1;
while (ind != -1)
{
        dfa_states[start_index].count = 1;
        Sm = 0;
        for (i = 0; i < symbols; i++)
        {
                trans(buffer, i, closure_table, states, NFA_TABLE, T_buf);
                strcpy(&DFA_TABLE[zz][i], T_buf);
                Sm = Sm + new_states(dfa_states, T_buf);
        }
        ind = indexing(dfa_states);
        if (ind != -1)
                strcpy(buffer, &dfa_states[++start_index].states);
        zz++;
}
Display_DFA(last_index, dfa_states, DFA_TABLE);
```

```
    return 0;
}
// End of the program
```

Output :



```
:~/Desktop/Lab$ ./a.out

 STATES OF NFA :                A, B, C, D, E, F,

 GIVEN SYMBOLS FOR NFA:         0, 1, eps


 NFA STATE TRANSITION TABLE


STATES  |0      |1      eps
-----------------------------------------
A       |FC     |-      |BF
B       |-      |C      |-
C       |-      |-      |D
D       |E      |A      |-
E       |A      |-      |BF
F       |-      |-      |-

 e-Closure (A) :        ABF

 e-Closure (B) :        B

 e-Closure (C) :        CD

 e-Closure (D) :        D

 e-Closure (E) :        BEF

 e-Closure (F) :        F

*****************************************************

                DFA TRANSITION STATE TABLE


 STATES OF DFA :                ABF, CDF, CD, BEF,

 GIVEN SYMBOLS FOR DFA:         0, 1,

STATES  |0      |1
--------+----------------------
ABF     |CDF    |CD
CDF     |BEF    |ABF
CD      |BEF    |ABF
BEF     |ABF    |CD
:~/Desktop/Lab$
```

Program :

```
//Program to minimize any given DFA
#include <stdio.h>
#include <string.h>

#define STATES  99
#define SYMBOLS 20

int N_symbols;
int N_DFA_states;
char *DFA_finals;
int DFAtab[STATES][SYMBOLS];

char StateName[STATES][STATES+1];

int N_optDFA_states;
int OptDFA[STATES][SYMBOLS];
char NEW_finals[STATES+1];

void print_dfa_table(int tab[][SYMBOLS],int nstates,int nsymbols,char *finals)
{
        int i, j;
        puts("\nDFA: STATE TRANSITION TABLE");
        printf("     | ");
        for (i = 0; i < nsymbols; i++)
                printf("  %c  ", '0'+i);
        printf("\n--------");
        for (i = 0; i < nsymbols; i++)
                printf("-----");
        printf("\n");
        for (i = 0; i < nstates; i++)
        {
        printf("  %c  | ", 'A'+i);
        for (j = 0; j < nsymbols; j++)
                        printf("  %c  ", tab[i][j]);
        printf("\n");
        }
        printf("Final states = %s\n", finals);
}

void load_DFA_table()
{

        DFAtab[0][0] = 'B'; DFAtab[0][1] = 'C';
        DFAtab[1][0] = 'E'; DFAtab[1][1] = 'F';
        DFAtab[2][0] = 'A'; DFAtab[2][1] = 'A';
        DFAtab[3][0] = 'F'; DFAtab[3][1] = 'E';
        DFAtab[4][0] = 'D'; DFAtab[4][1] = 'F';
```

```
        DFAtab[5][0] = 'D'; DFAtab[5][1] = 'E';
        DFA_finals = "EF";
        N_DFA_states = 6;
        N_symbols = 2;
}

void get_next_state(char *nextstates, char *cur_states,int dfa[STATES][SYMBOLS], int symbol)
{
        int i, ch;
        for (i = 0; i < strlen(cur_states); i++)
        *nextstates++ = dfa[cur_states[i]-'A'][symbol];
        *nextstates = '\0';
}

char equiv_class_ndx(char ch, char stnt[][STATES+1], int n)
{
        int i;
        for (i = 0; i < n; i++)
        if (strchr(stnt[i], ch))
                return i+'0';
        return -1;
}

char is_one_nextstate(char *s)
{
        char equiv_class;
        while (*s == '@') s++;
        equiv_class = *s++;
        while (*s)
        {
        if (*s != '@' && *s != equiv_class)
                return 0;
        s++;
        }
        return equiv_class;
}

int state_index(char *state, char stnt[][STATES+1], int n, int *pn,int cur)
{
        int i;
        char state_flags[STATES+1];
        if (!*state)
                return -1;
        for (i = 0; i < strlen(state); i++)
        state_flags[i] = equiv_class_ndx(state[i], stnt, n);
        state_flags[i] = '\0';
        printf("   %d:[%s]\t--> [%s] (%s)\n",
      cur, stnt[cur], state, state_flags);
        if (i=is_one_nextstate(state_flags))
        return i-'0';
        else
        {
```

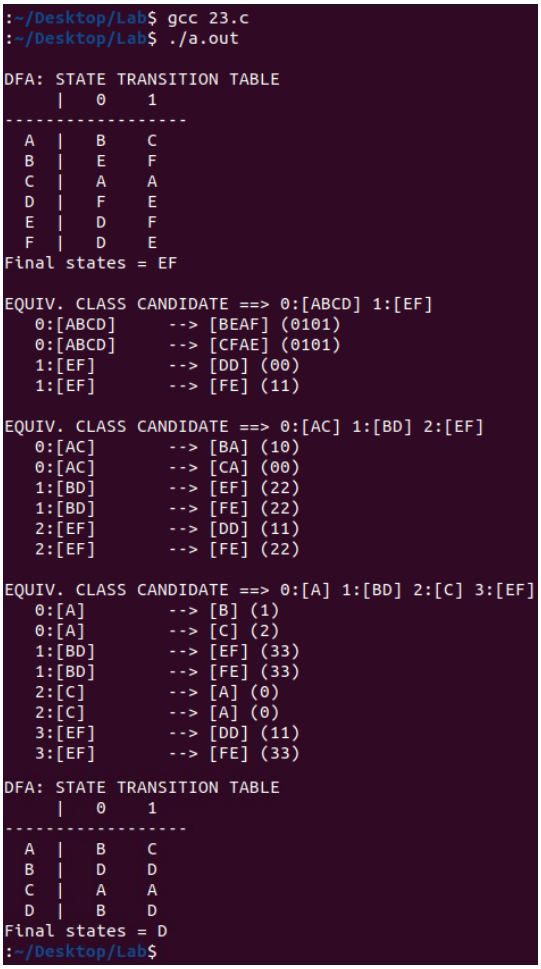
```
        strcpy(stnt[*pn], state_flags);
        return (*pn)++;
        }
}

int init_equiv_class(char statename[][STATES+1], int n, char *finals)
{
        int i, j;
        if (strlen(finals) == n)
        {
        strcpy(statename[0], finals);
        return 1;
        }
        strcpy(statename[1], finals);
        for (i=j=0; i < n; i++)
        {
        if (i == *finals-'A')
        {
                        finals++;
        }
        else
                statename[0][j++] = i+'A';
        }
        statename[0][j] = '\0';
        return 2;
}

int get_optimized_DFA(char stnt[][STATES+1], int n,int dfa[][SYMBOLS], int n_sym, int
newdfa[][SYMBOLS])
{
        int n2=n;
        int i, j;
        char nextstate[STATES+1];
        for (i = 0; i < n; i++)
        {
                for (j = 0; j < n_sym; j++)
                {
                        get_next_state(nextstate, stnt[i], dfa, j);
                        newdfa[i][j] = state_index(nextstate, stnt, n, &n2, i)+'A';
        }
        }
        return n2;
}

void chr_append(char *s, char ch)
{
            int n=strlen(s);
            *(s+n) = ch;
            *(s+n+1) = '\0';
}

void sort(char stnt[][STATES+1], int n)
```

```
{
	int i, j;
	char temp[STATES+1];
	for (i = 0; i < n-1; i++)
	for (j = i+1; j < n; j++)
			if (stnt[i][0] > stnt[j][0])
			{
				strcpy(temp, stnt[i]);
				strcpy(stnt[i], stnt[j]);
				strcpy(stnt[j], temp);
		}
}

int split_equiv_class(char stnt[][STATES+1],int i1,int i2,int n,int n_dfa)
{
	char *old=stnt[i1], *vec=stnt[i2];
	int i, n2, flag=0;
	char newstates[STATES][STATES+1];
	for (i=0; i < STATES; i++)
		newstates[i][0] = '\0';
	for (i=0; vec[i]; i++)
	chr_append(newstates[vec[i]-'0'], old[i]);
	for (i=0, n2=n; i < n_dfa; i++)
	{
	if (newstates[i][0])
	{
			if (!flag)
			{
			strcpy(stnt[i1], newstates[i]);
			flag = 1;
			}
			else
			strcpy(stnt[n2++], newstates[i]);
	}
	}
	sort(stnt, n2);
	return n2;
}

int set_new_equiv_class(char stnt[][STATES+1], int n,int newdfa[][SYMBOLS], int n_sym, int
n_dfa)
{
	int i, j, k;
	for (i = 0; i < n; i++)
	{
		for (j = 0; j < n_sym; j++)
		 {
			k = newdfa[i][j]-'A';
			if (k >= n)
			return split_equiv_class(stnt, i, k, n, n_dfa);
	}
	}
```

```
	return n;
}

void print_equiv_classes(char stnt[][STATES+1], int n)
{
	int i;
	printf("\nEQUIV. CLASS CANDIDATE ==>");
	for (i = 0; i < n; i++)
	printf(" %d:[%s]", i, stnt[i]);
	printf("\n");
}

int optimize_DFA(int dfa[][SYMBOLS],int n_dfa,int n_sym,char *finals,char stnt[]
[STATES+1],int newdfa[][SYMBOLS])
{
	char nextstate[STATES+1];
	int n;
	int n2;
	n = init_equiv_class(stnt, n_dfa, finals);
	while (1)
	{
	print_equiv_classes(stnt, n);
	n2 = get_optimized_DFA(stnt, n, dfa, n_sym, newdfa);
	if (n != n2)
			n = set_new_equiv_class(stnt, n, newdfa, n_sym, n_dfa);
		else break;
	}
	return n;
}

int is_subset(char *s, char *t)
{
	int i;
	for (i = 0; *t; i++)
		if (!strchr(s, *t++))
			return 0;
	return 1;
}

void get_NEW_finals(char *newfinals,char *oldfinals,char stnt[][STATES+1],int n)
{
	int i;
	for (i = 0; i < n; i++)
		if (is_subset(oldfinals, stnt[i])) *newfinals++ = i+'A';
			*newfinals++ = '\0';
}

void main()
{
	load_DFA_table();
	print_dfa_table(DFAtab, N_DFA_states, N_symbols, DFA_finals);
	N_optDFA_states = optimize_DFA(DFAtab, N_DFA_states,
```

```
        N_symbols, DFA_finals, StateName, OptDFA);
        get_NEW_finals(NEW_finals, DFA_finals, StateName, N_optDFA_states);
        print_dfa_table(OptDFA, N_optDFA_states, N_symbols, NEW_finals);
}
//End of the program
```

Output :



```
:~/Desktop/Lab$ gcc 23.c
:~/Desktop/Lab$ ./a.out

DFA: STATE TRANSITION TABLE
     |   0    1
-----------------
  A  |   B    C
  B  |   E    F
  C  |   A    A
  D  |   F    E
  E  |   D    F
  F  |   D    E
Final states = EF

EQUIV. CLASS CANDIDATE ==> 0:[ABCD] 1:[EF]
   0:[ABCD]     --> [BEAF] (0101)
   0:[ABCD]     --> [CFAE] (0101)
   1:[EF]       --> [DD] (00)
   1:[EF]       --> [FE] (11)

EQUIV. CLASS CANDIDATE ==> 0:[AC] 1:[BD] 2:[EF]
   0:[AC]       --> [BA] (10)
   0:[AC]       --> [CA] (00)
   1:[BD]       --> [EF] (22)
   1:[BD]       --> [FE] (22)
   2:[EF]       --> [DD] (11)
   2:[EF]       --> [FE] (22)

EQUIV. CLASS CANDIDATE ==> 0:[A] 1:[BD] 2:[C] 3:[EF]
   0:[A]        --> [B] (1)
   0:[A]        --> [C] (2)
   1:[BD]       --> [EF] (33)
   1:[BD]       --> [FE] (33)
   2:[C]        --> [A] (0)
   2:[C]        --> [A] (0)
   3:[EF]       --> [DD] (11)
   3:[EF]       --> [FE] (33)

DFA: STATE TRANSITION TABLE
     |   0    1
-----------------
  A  |   B    C
  B  |   D    D
  C  |   A    A
  D  |   B    D
Final states = D
:~/Desktop/Lab$
```