**EXP NO : 1**

Write a C program to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with a and end with a

**AIM :**

To write a C program to simulate a Deterministic Finite Automata.

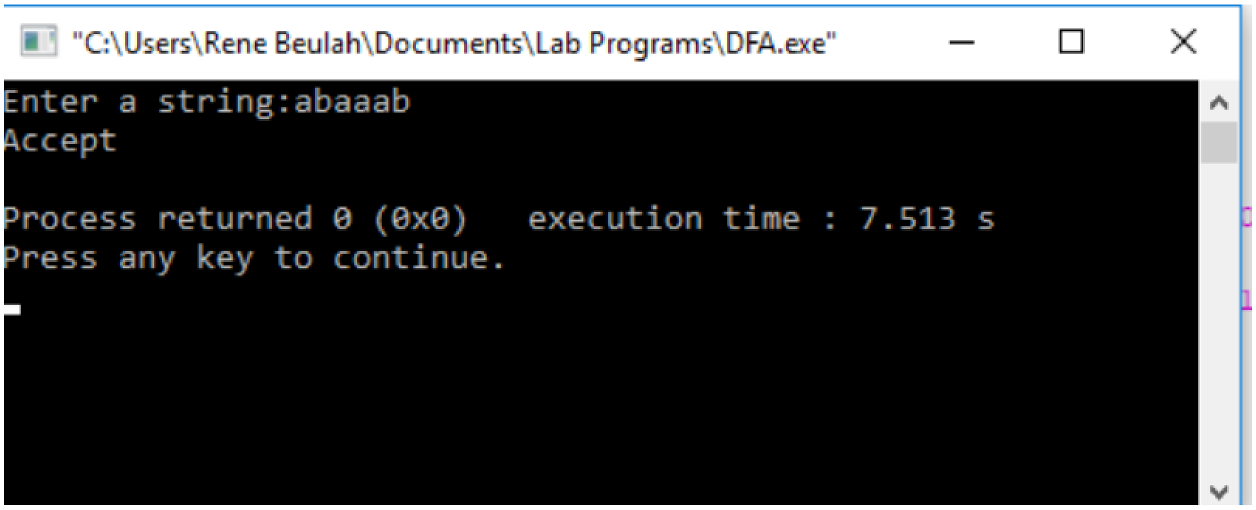
## ALGORTIHM :

aw a DFA for the given language and construct the transition table.
2. Store the transition table in a two-dimensional array.
3. Initialize present_state, next_state and final_state
4. Get the input string from the user.
5. Find the length of the input string.
6. Read the input string character by character.
7. Repeat step 8 for every character
8. Refer the transition table for the entry corresponding to the present state and the current input symbol and update the next state.
9. When we reach the end of the input, if the final state is reached, the input is accepted. Otherwise the input is not accepted

## Program:

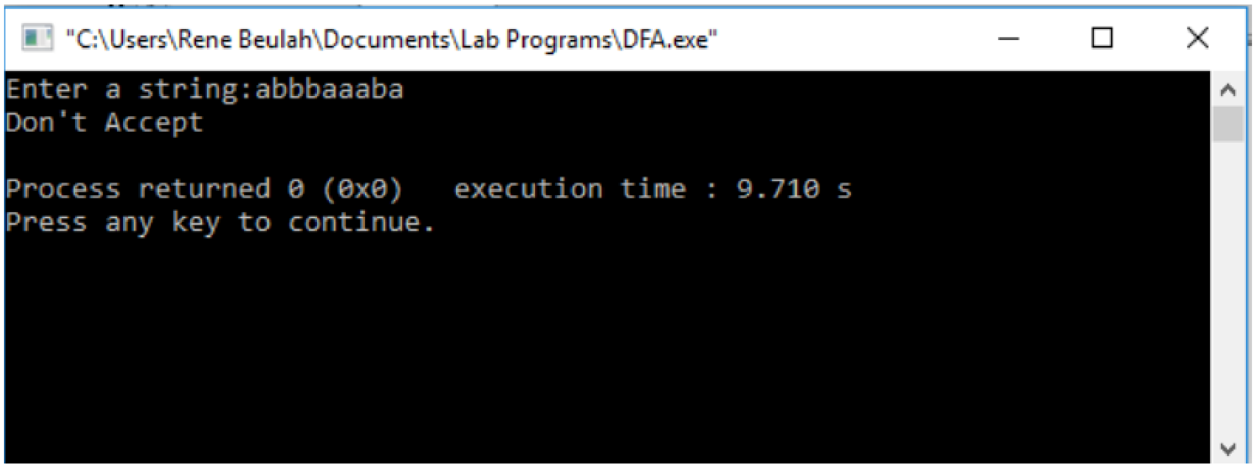
```
#include<stdio.h>
#include<string.h>
#define max 20
int main()
{
int trans_table[4][2]={{1,3},{1,2},{1,2},{3,3}};
int final_state=2,i;
int present_state=0;
int next_state=0;
int invalid=0;
char input_string[max];
printf("Enter a string:");
scanf("%s",input_string);
int l=strlen(input_string);
for(i=0;i<l;i++)
{
if(input_string[i]=='a')
next_state=trans_table[present_state][0];
else if(input_string[i]=='b')
next_state=trans_table[present_state][1];
else
invalid=l;
present_state=next_state;
}
if(invalid==l)
{
printf("Invalid input");
}
else if(present_state==final_state)
printf("Accept\n");
else
printf("Don't Accept\n");
}
```

**output:**



"C:\Users\Rene Beulah\Documents\Lab Programs\DFA.exe"

```
Enter a string:abaaab
Accept

Process returned 0 (0x0)   execution time : 7.513 s
Press any key to continue.
```



"C:\Users\Rene Beulah\Documents\Lab Programs\DFA.exe"

```
Enter a string:abbbaaaba
Don't Accept

Process returned 0 (0x0)   execution time : 9.710 s
Press any key to continue.
```

**Result:**

Hence c program is Executed Successfully.

**EXP NO : 2**

**Write a C program to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with 0 and end with 1**

**AIM :**

To write a C program to simulate a Non-Deterministic Finite Automata.

**ALGORTIHM** :

1. Get the following as input from the user.

i. Number of states in the NFA

ii. Number of symbols in the input alphabet and the symbols

iii. Number of final states and their names 2. Declare a 3-dimensional matrix to store the transitions and initialize all the entries with -1

3. Get the transitions from every state for every input symbol from the user and store it in the matrix. .

There are 4 states 0, 1, 2 and 3

There are two input symbols a and b. As the array index always

starts with 0, we assume 0th symbol is a and 1st symbol is b.

The transitions will be stored in the matrix as follows:

From state 0, for input a, there are two transitions to state 0 and 1,

which can be stored in the matrix as

m[0][0][0]=0

m[0][0][1]=1

Similarly, the other transitions can be stored as follows:

m[0][1][0]=0 (From state 0, for input b, one transition is to state 0)

m[0][1][1]=2 (From state 0, for input b, next transition is to state 2) m[1][1][0]=3 (From state 1, for input b, move to state 3)

m[2][0][0]=3 (From state 2, for input a, move to state 3)

m[3][0][0]=3 (From state 3, for input a, move to state 3)

m[3][1][0]=3 (From state 3, for input b, move to state 3)

All the other entries in the matrix will be -1 indicating no moves

4. Get the input string from the user.

5. Find the length of the input string.

6. Read the input string character by character.

7. Repeat step 8 for every character

8. Refer the transition table for the entry corresponding to the present state and the current input symbol and update the next state. As there can be more than one transition, the next state will be an array.

9. From every state in the next state array, find the list of new transitions and update the next state array.

10. When we reach the end of the input, if at least one of the final states is present in the next state array, it means there is a path to a final state. So the input is accepted. Otherwise the input is not accept

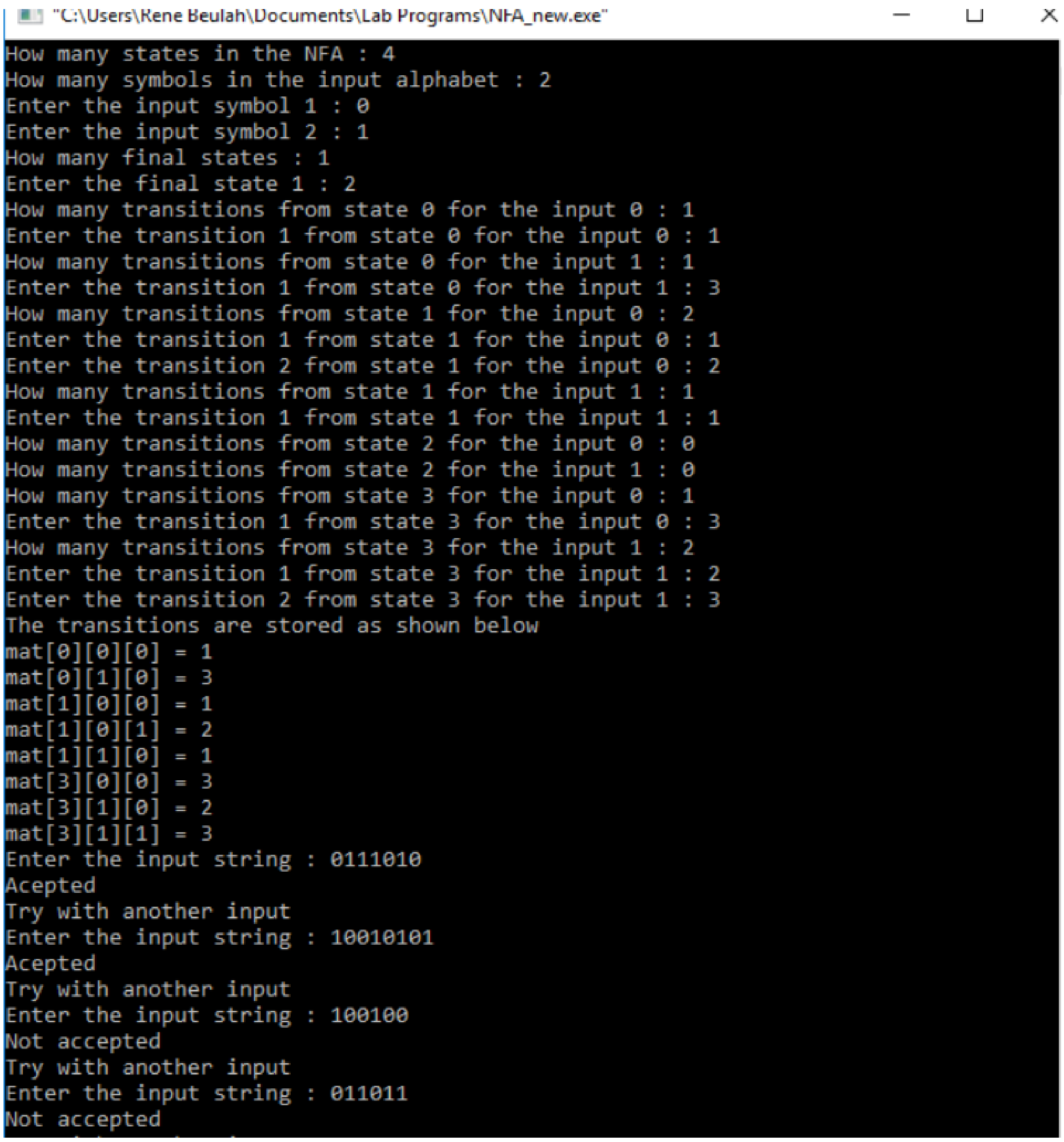
**Program:**

```
#include<stdio.h>

#include<string.h>

int main()

{

int i,j,k,l,m,next_state[20],n,mat[10][10][10],flag,p;

int num_states,final_state[5],num_symbols,num_final;

int present_state[20],prev_trans,new_trans;
```

```
char ch,input[20];

int symbol[5],inp,inp1;

printf("How many states in the NFA : ");

scanf("%d",&num_states);

printf("How many symbols in the input alphabet : ");

scanf("%d",&num_symbols);

for(i=0;i<num_symbols;i++)

{

printf("Enter the input symbol %d : ",i+1);

scanf("%d",&symbol[i]);

}

printf("How many final states : ");

scanf("%d",&num_final);

for(i=0;i<num_final;i++)

{

printf("Enter the final state %d : ",i+1);

scanf("%d",&final_state[i]);

}

//Initialize all entries with -1 in Transition table

for(i=0;i<10;i++)

{

for(j=0;j<10;j++)

{
```

```
for(k=0;k<10;k++)
{
mat[i][j][k]=-1;
}
}
}
//Get input from the user and fill the 3D transition table
for(i=0;i<num_states;i++)
{
for(j=0;j<num_symbols;j++)
{
printf("How many transitions from state %d for the input %d :
",i,symbol[j]);
scanf("%d",&n);
for(k=0;k<n;k++)
{
printf("Enter the transition %d from state %d for the input
%d : ",k+1,i,symbol[j]);
scanf("%d",&mat[i][j][k]);
}
}
}
printf("The transitions are stored as shown below\n");
```

```
 for(i=0;i<10;i++)
{
for(j=0;j<10;j++)
{
for(k=0;k<10;k++)
{
 if(mat[i][j][k]!=-1)
 printf("mat[%d][%d][%d] = %d\n",i,j,k,mat[i][j][k]);
}
}
}
while(1)
{
printf("Enter the input string : ");
scanf("%s",input);
present_state[0]=0;
prev_trans=1;
l=strlen(input);
for(i=0;i<l;i++)
{
 if(input[i]=='0')
 inp1=0;
 else if(input[i]=='1')
 inp1=1;
 else
 {
 printf("Invalid input\n");
 exit(0);
 }
 for(m=0;m<num_symbols;m++)
{
if(inp1==symbol[m])
{
inp=m;
break;
}
}
new_trans=0;
for(j=0;j<prev_trans;j++)
{
k=0;
p=present_state[j];
while(mat[p][inp][k]!=-1)
{
next_state[new_trans++]=mat[p][inp][k];
k++;
}
}
for(j=0;j<new_trans;j++)
{
present_state[j]=next_state[j];
}
prev_trans=new_trans;
}
flag=0;
for(i=0;i<prev_trans;i++)
{
for(j=0;j<num_final;j++)
{
if(present_state[i]==final_state[j])
{
flag=1;
break;
}
}
}
if(flag==1)
printf("Acepted\n");else
printf("Not accepted\n");
printf("Try with another input\n");

}

}
```



"C:\Users\Rene Beulah\Documents\Lab Programs\NFA_new.exe"

```
How many states in the NFA : 4
How many symbols in the input alphabet : 2
Enter the input symbol 1 : 0
Enter the input symbol 2 : 1
How many final states : 1
Enter the final state 1 : 2
How many transitions from state 0 for the input 0 : 1
Enter the transition 1 from state 0 for the input 0 : 1
How many transitions from state 0 for the input 1 : 1
Enter the transition 1 from state 0 for the input 1 : 3
How many transitions from state 1 for the input 0 : 2
Enter the transition 1 from state 1 for the input 0 : 1
Enter the transition 2 from state 1 for the input 0 : 2
How many transitions from state 1 for the input 1 : 1
Enter the transition 1 from state 1 for the input 1 : 1
How many transitions from state 2 for the input 0 : 0
How many transitions from state 2 for the input 1 : 0
How many transitions from state 3 for the input 0 : 1
Enter the transition 1 from state 3 for the input 0 : 3
How many transitions from state 3 for the input 1 : 2
Enter the transition 1 from state 3 for the input 1 : 2
Enter the transition 2 from state 3 for the input 1 : 3
The transitions are stored as shown below
mat[0][0][0] = 1
mat[0][1][0] = 3
mat[1][0][0] = 1
mat[1][0][1] = 2
mat[1][1][0] = 1
mat[3][0][0] = 3
mat[3][1][0] = 2
mat[3][1][1] = 3
Enter the input string : 0111010
Acepted
Try with another input
Enter the input string : 10010101
Acepted
Try with another input
Enter the input string : 100100
Not accepted
Try with another input
Enter the input string : 011011
Not accepted
```

**EXP NO:3**

**Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)**

**S → 0A1** **A → 0A | 1A | ε**

**AIM :**

To write a C program to find ε-closure of a Non-Deterministic Finite Automata with ε-moves

**ALGORTIHM** :

1. Get the following as input from the user.

i. Number of states in the NFA

ii. Number of symbols in the input alphabet including ε

iii. Input symbols iv. Number of final states and their names

2. Declare a 3-dimensional matrix to store the transitions and initialize all the entries with -1

3. Get the transitions from every state for every input symbol from the user and store it in the matrix.

There are 3 states 0, 1, and 2

There are three input symbols ε, 0 and 1. As the array index always starts with 0, we assume 0th symbol is ε, 1st symbol is 0 and 2nd symbol is 1.

The transitions will be stored in the matrix as follows:

From state 0, for input ε, there is one transition to state 1, which can be stored in the matrix as

m[0][0][0]=1

From state 0, for input 0, there is no transition.

From state 0, for input 1, there is one transition to state 1, which can be stored in the matrix as

m[0][2][0]=1

Similarly, the other transitions can be stored as follows:

m[1][0][0]=2 (From state 1, for input ε, the transition is to state 2)

m[1][1][0]=1 (From state 1, for input 0, the transition is to state 1)

All the other entries in the matrix will be -1 indicating no moves

4. Initialize a two-dimensional matrix e_closure with -1 in all the entries.

5. ε-closure of a state q is defined as the set of all states that can be reached from state q using only ε-transitions.

Example: Consider the NFA with ε-transitions given below:

ε-closure(0)={0,1,2)

ε-closure(1)={1,2}

ε-closure(2)={2}

Here, we see that ε-closure of every state contains that state first. So initialize the first entry of the array e_closure with the same state.

e_closure(0,0)=0;

e_closure(1,0)=1;

e_closure(2,0)=2;

6. For every state i, find ε-closure as follows: If there is an ε-transition from state i to state j, add j to the matrix e_closure[i]. Call the recursive function find_e_closure(j) and add the other states that are reachable from i using ε

7. For every state, print the ε-closure values The function find_e_closure(i) This function finds ε-closure of a state recursively by tracing all the εtransitions

**Program:**

```
#include<stdio.h>

#include<string.h>

int trans_table[10][5][3];
```

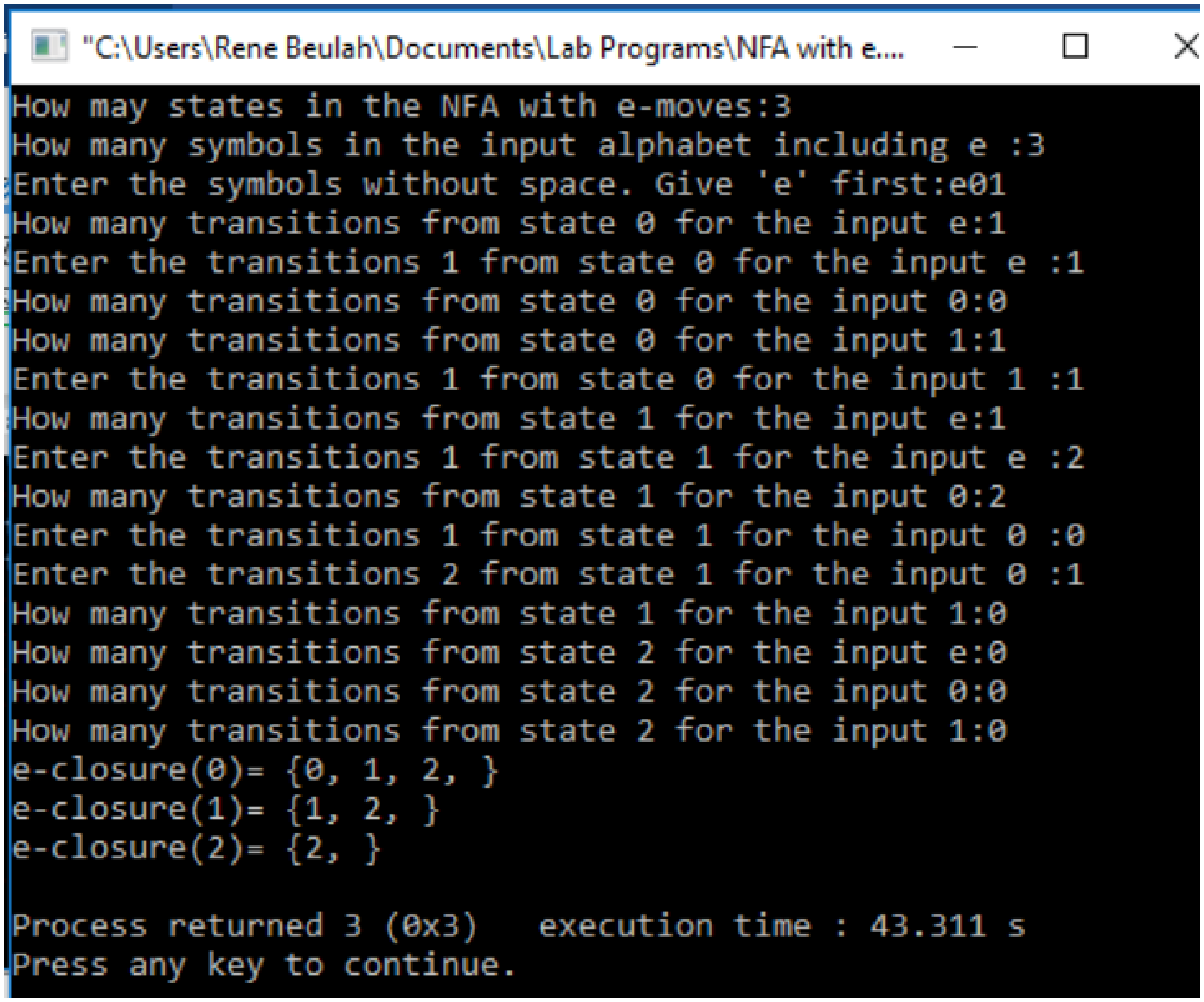
```
char symbol[5],a;

int e_closure[10][10],ptr,state;

void find_e_closure(int x);

int main()

{

int i,j,k,n,num_states,num_symbols;

for(i=0;i<10;i++)

{

for(j=0;j<5;j++)

{

for(k=0;k<3;k++)

{

trans_table[i][j][k]=-1;

}

}

}

printf("How may states in the NFA with e-moves:");

scanf("%d",&num_states);

printf("How many symbols in the input alphabet including e :");

scanf("%d",&num_symbols);

printf("Enter the symbols without space. Give 'e' first:");

scanf("%s",symbol);

for(i=0;i<num_states;i++)
```

```
{
for(j=0;j<num_symbols;j++)
{
printf("How many transitions from state %d for the input
%c:",i,symbol[j]);
scanf("%d",&n);
for(k=0;k<n;k++)
{
printf("Enter the transitions %d from state %d for the input
%c :", k+1,i,symbol[j]);
scanf("%d",&trans_table[i][j][k]);
}
}
}
for(i=0;i<10;i++)
{
for(j=0;j<10;j++)
{
e_closure[i][j]=-1;
}
}
for(i=0;i<num_states;i++)
e_closure[i][0]=i;
```

```
for(i=0;i<num_states;i++)
{
if(trans_table[i][0][0]==-1)
continue;
else
{
state=i;
ptr=1;
find_e_closure(i);
}
}
for(i=0;i<num_states;i++)
{
printf("e-closure(%d)= {",i);
for(j=0;j<num_states;j++)
{
if(e_closure[i][j]!=-1)
{
printf("%d, ",e_closure[i][j]);
}
}
printf("}\n");
}
```

```
}
void find_e_closure(int x)
{
int i,j,y[10],num_trans;
i=0;
while(trans_table[x][0][i]!=-1)
{
y[i]=trans_table[x][0][i];
i=i+1;
}
num_trans=i;
for(j=0;j<num_trans;j++)
{
e_closure[state][ptr]=y[j];
ptr++;
find_e_closure(y[j]);
}
}
```

**output:**



```
"C:\Users\Rene Beulah\Documents\Lab Programs\NFA with e....
How may states in the NFA with e-moves:3
How many symbols in the input alphabet including e :3
Enter the symbols without space. Give 'e' first:e01
How many transitions from state 0 for the input e:1
Enter the transitions 1 from state 0 for the input e :1
How many transitions from state 0 for the input 0:0
How many transitions from state 0 for the input 1:1
Enter the transitions 1 from state 0 for the input 1 :1
How many transitions from state 1 for the input e:1
Enter the transitions 1 from state 1 for the input e :2
How many transitions from state 1 for the input 0:2
Enter the transitions 1 from state 1 for the input 0 :0
Enter the transitions 2 from state 1 for the input 0 :1
How many transitions from state 1 for the input 1:0
How many transitions from state 2 for the input e:0
How many transitions from state 2 for the input 0:0
How many transitions from state 2 for the input 1:0
e-closure(0)= {0, 1, 2, }
e-closure(1)= {1, 2, }
e-closure(2)= {2, }

Process returned 3 (0x3)   execution time : 43.311 s
Press any key to continue.
```

**EXP NO:4**

1. **Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)**

**S → 0S0 | 1S1 | 0 | 1 | ε**

**AIM :**

To write a C program to check whether a string belongs to the grammar S → 0 A 1 A → 0 A | 1 A | ε

**ALGORTIHM :**

1. Get the input string from the user.

2. Find the length of the string.

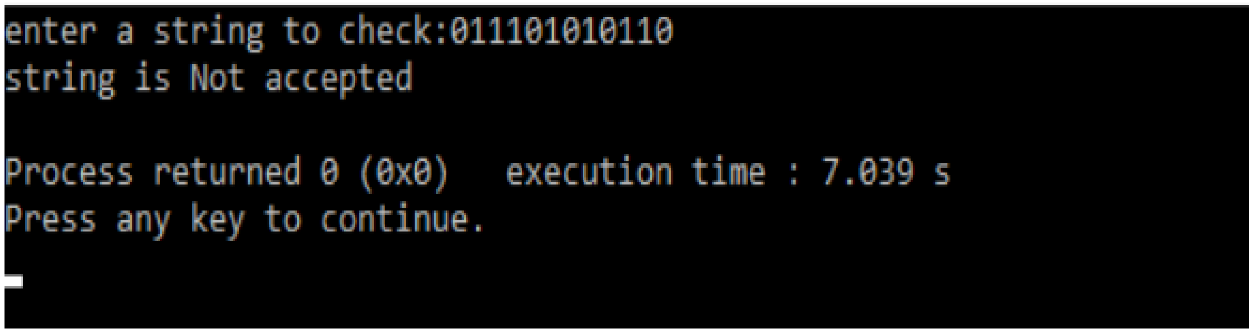
3. Check whether all the symbols in the input are either 0 or 1. If so, print “String is valid” and go to step

4. Otherwise print “String not valid” and quit the program. 4. If the first symbol is 0 and the last symbol is 1, print “String accepted”. Otherwise, print “String not accepted”

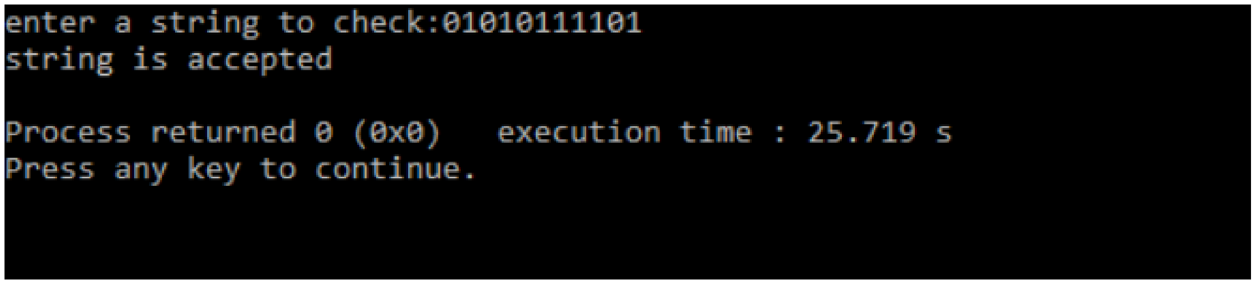
**PROGRAM:**

```
#include<stdio.h>

#include<string.h>

int main(){

char s[100];

int i,flag;

int l;

printf("enter a string to check:");

scanf("%s",s);

l=strlen(s);

flag=1;

for(i=0;i<l;i++)

{

if(s[i]!='0' && s[i]!='1')

{

 flag=0;

}

}
if(flag!=1)

printf("string is Not Valid\n");

if(flag==1)

{

if (s[0]=='0'&&s[l-1]=='1')

printf("string is accepted\n");

else

printf("string is Not accepted\n");

}

}
```

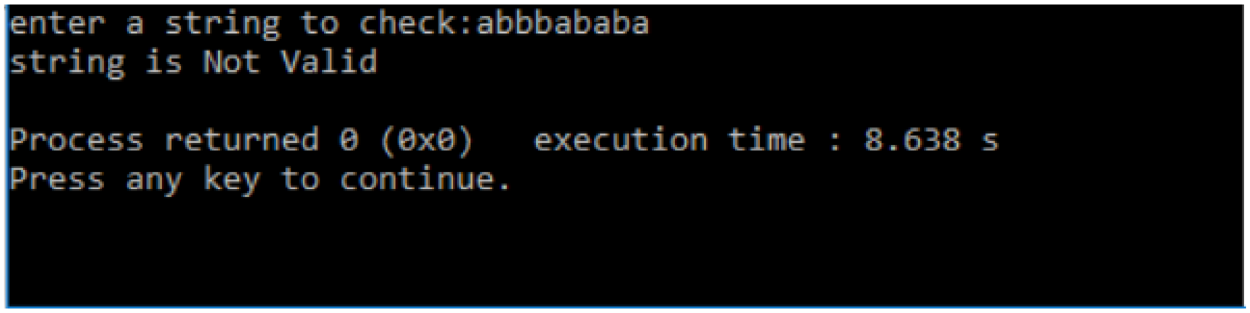
**OUTPUT:**



```
enter a string to check:011101010110
string is Not accepted

Process returned 0 (0x0)   execution time : 7.039 s
Press any key to continue.
```



```
enter a string to check:01010111101
string is accepted

Process returned 0 (0x0)   execution time : 25.719 s
Press any key to continue.
```



```
enter a string to check:abbbababa
string is Not Valid

Process returned 0 (0x0)   execution time : 8.638 s
Press any key to continue.
```

**EXP NO:5**

Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S0 | A    A → 1A | ε

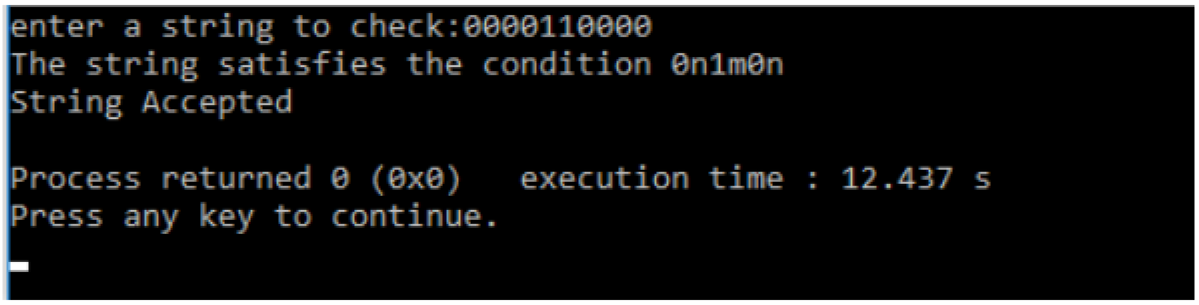
**AIM :**
To write a C program to check whether a string belongs to the grammar S -> 0 S 0 | A A -> 1 A | ε Language defined by the Grammar Set of all strings over **Σ**={0,1} satisfying 0n1m0n

**ALGORITHM :**
1. Get the input string from the user.
2. Find the length of the string.
3. Check whether all the symbols in the input are either 0 or 1. If so, print "String is valid" and go to step 4 Otherwise print "String not valid" and quit the program.
4. Read the input string character by character
5. Count the number of 0's in the front and store it in the variable count1 6. Skip all 1's 7. Count the number of 0's in the end and store it in the variable count2 8. If count1==count2, print "String Accepted". Otherwise print "String Not Accepted"

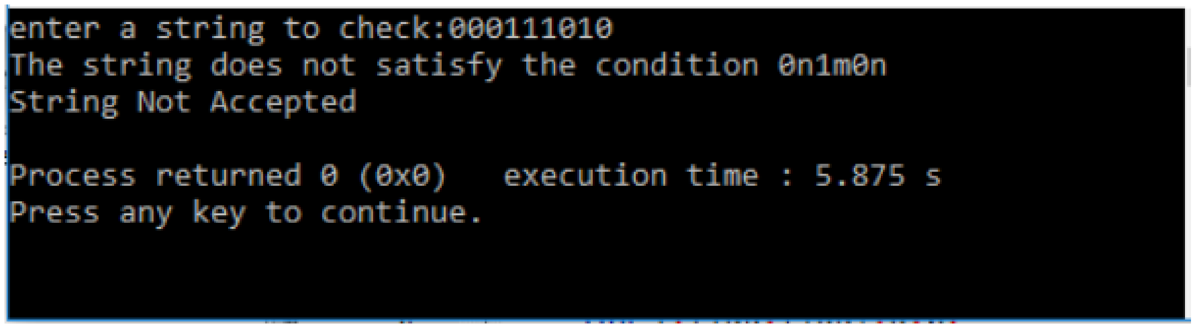
**PROGRAM:**
```
#include<stdio.h>
#include<string.h>
void main()
{
char s[100];
int i,flag,flag1,a,b;
int l,count1,count2;
printf("enter a string to check:");
scanf("%s",s);
l=strlen(s);
flag=1;
for(i=0;i<l;i++)
{
if(s[i]!='0' && s[i]!='1')
{
 flag=0;
}
}
if(flag!=1)
printf("string is Not Valid\n");
if(flag==1)
{
i=0;count1=0;
while(s[i]=='0') // Count the no of 0s in the front
{
count1++;
i++;
}
while(s[i]=='1')
{
i++; // Skip all 1s
}
flag1=1;
count2=0;
while(i<l)
{
if(s[i]=='0')// Count the no of 0s at the end
{
count2++;
}
else
{
flag1=0;
}
i++;
}
if(flag1==1)
{
if(count1==count2)
 {
 printf("The string satisfies the condition 0n1m0n\n");
 printf("String Accepted\n");
 }
 else
 {
 printf("The string does not satisfy the condition 0n1m0n\n");
 printf("String Not Accepted\n");
 }
}
```

```
else
 {
 printf("The string does not satisfy the condition 0n1m0n\n");
 printf("String Not Accepted\n");
}
}
}
```

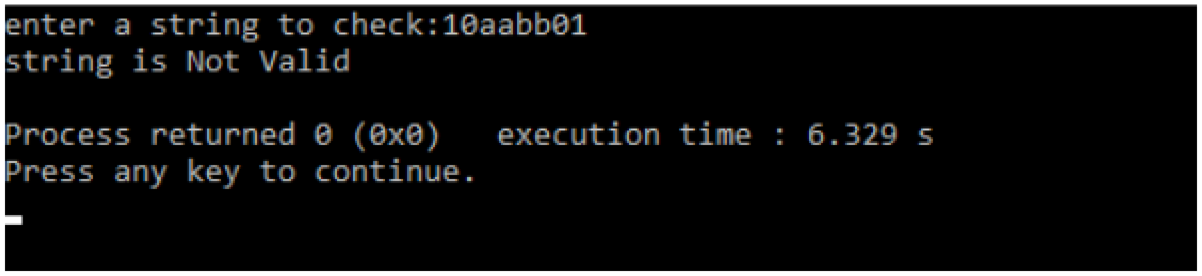
**OUTPUT :**



```
enter a string to check:0000110000
The string satisfies the condition 0n1m0n
String Accepted

Process returned 0 (0x0)   execution time : 12.437 s
Press any key to continue.
```



```
enter a string to check:000111010
The string does not satisfy the condition 0n1m0n
String Not Accepted

Process returned 0 (0x0)   execution time : 5.875 s
Press any key to continue.
```



```
enter a string to check:10aabb01
string is Not Valid

Process returned 0 (0x0)   execution time : 6.329 s
Press any key to continue.
```

**EXP NO:6**

Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0S1 | ε

**AIM :**

To write a C program to check whether a string belongs to the grammar S -> 0 S 1 | ε

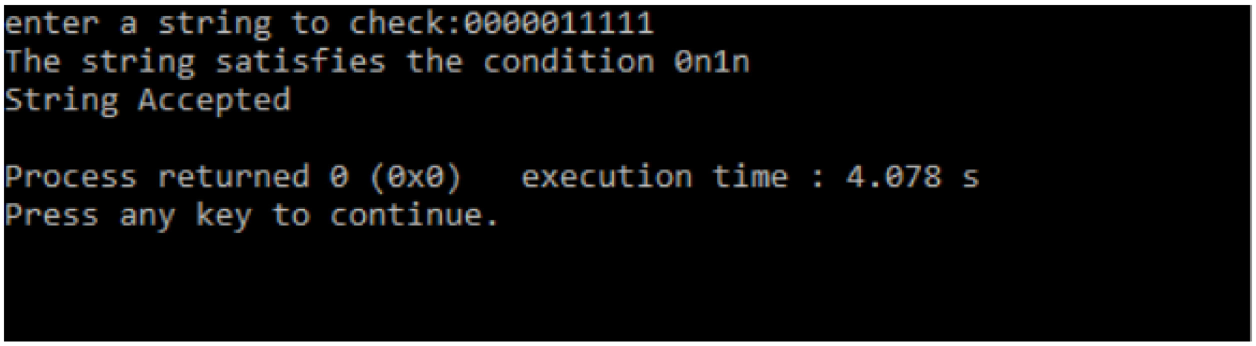
**ALGORITHM:**

```
1. Get the input string from the user.
 2. Find the length of the string.
 3. Check whether all the symbols in the input are either 0 or 1. If so, print "String is valid" and go to step
4. Otherwise print "String not valid" and quit the program.
 4. Find the length of the string. If the length is odd, then print "String not accepted" and quit the program.
If the length is even, then go to step 5.
 5. Divide the string into two halves.
 6. If the first half contains only 0s and the second half contains only 1s then print "String Accepted".
Otherwise print "String Not Accepted"
```

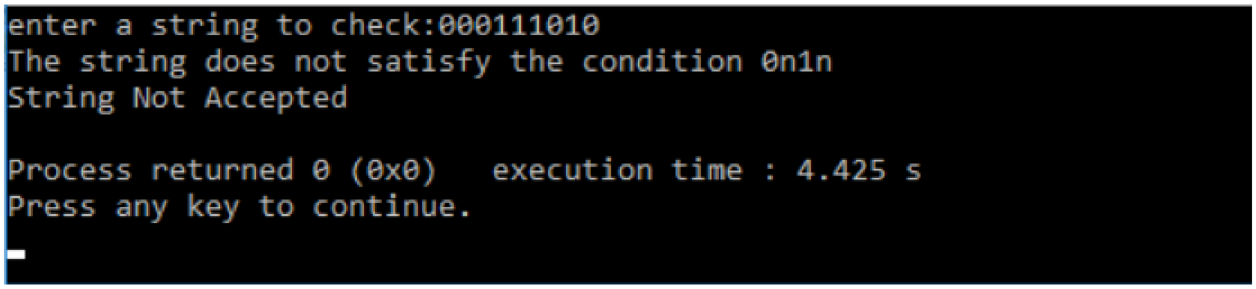
**PROGRAM:**

```
#include<stdio.h>
#include<string.h>
void main()
{
char s[100];
int i,flag,flag1,flag2;
int l;
printf("enter a string to check:");
scanf("%s",s);
l=strlen(s);
flag=1;
for(i=0;i<l;i++)
{
if(s[i]!='0' && s[i]!='1')
{
 flag=0;
}
}
if(flag!=1)
printf("string is Not Valid\n");
if(flag==1)
{
if(l%2!=0) // If string length is odd
 {
 printf("The string does not satisfy the condition 0n1n\n");
 printf("String Not Accepted\n");
 }
else
{
// To check first half contains 0s
flag1=1;
for(i=0;i<(l/2);i++)
{
if(s[i]!='0')
{
flag1=0;
}
}
// To check second half contains 1s
flag2=1;
for(i=l/2;i<l;i++)
{
if(s[i]!='1')
{
flag2=0;
}
}
if(flag1==1 && flag2==1)
 {
 printf("The string satisfies the condition 0n1n\n");
 printf("String Accepted\n");
 }
else
 {
 printf("The string does not satisfy the condition 0n1n\n");
 printf("String Not Accepted\n");
 }
}
}
}
```

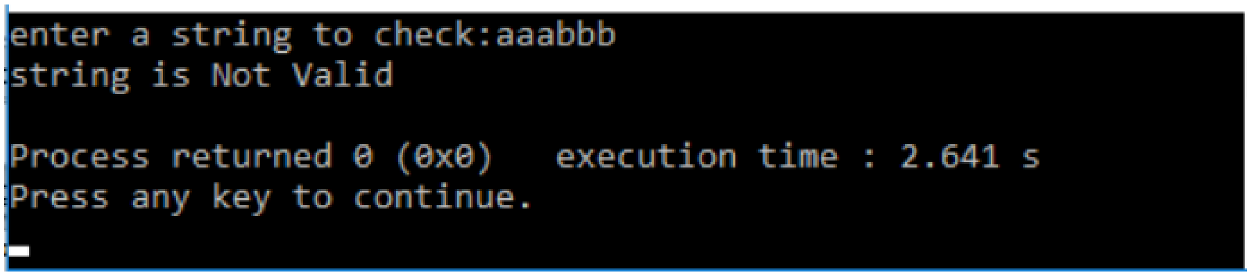
**OUTPUT:**



```
enter a string to check:0000011111
The string satisfies the condition 0n1n
String Accepted

Process returned 0 (0x0)   execution time : 4.078 s
Press any key to continue.
```



```
enter a string to check:000111010
The string does not satisfy the condition 0n1n
String Not Accepted

Process returned 0 (0x0)   execution time : 4.425 s
Press any key to continue.
```



```
enter a string to check:aaabbb
string is Not Valid

Process returned 0 (0x0)   execution time : 2.641 s
Press any key to continue.
```

**EXP NO:7**

Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → A101A,   A → 0A | 1A | ε

**AIM :**

To write a C program to check whether a string belongs to the grammar S -> A 1 0 1 A A -> 0 A | 1 A | ε

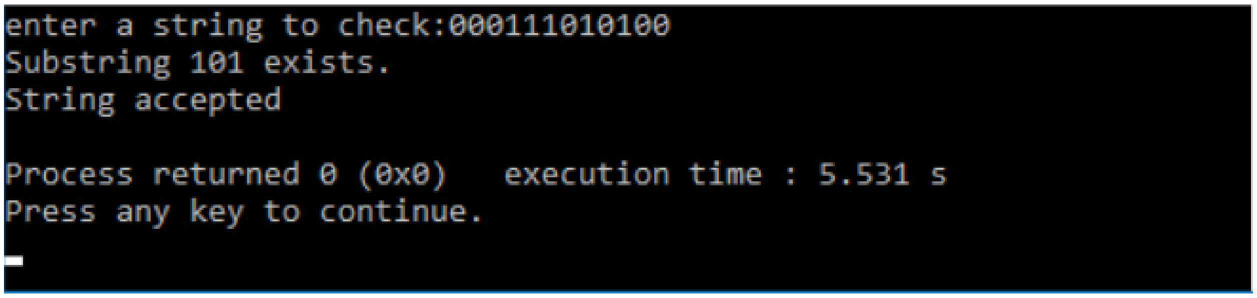
**ALGORITHM** :

1. Get the input string from the user.
2. Find the length of the string.
3. Check whether all the symbols in the input are either 0 or 1. If so, print "String is valid" and go to step
4. Otherwise print "String not valid" and quit the program.
4. Read the input string character by character
5. If the i th input symbol is 1, check whether (i+1) th symbol is 0 and (i+2) th symbol is 1. If so, the string has the substring 101. So print "String Accepted". Otherwise, print "String Not Accepted"

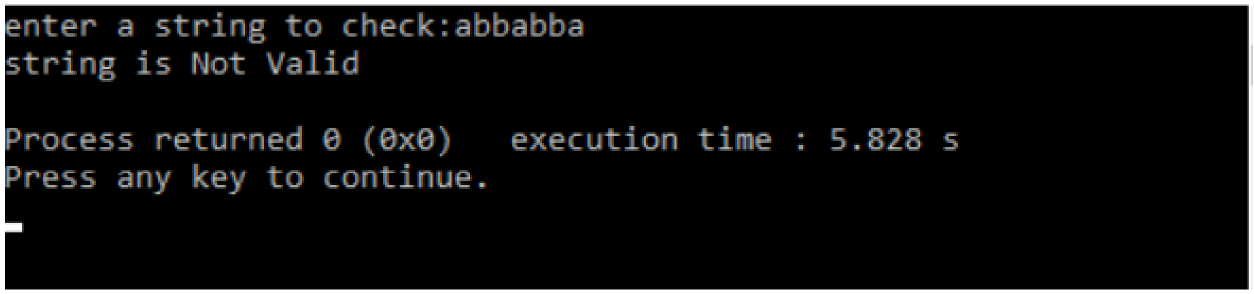
**PROGRAM:**

```
#include<stdio.h>
#include<string.h>
int main()
{
char s[100];
int i,flag,flag1;
int l;
printf("enter a string to check:");
scanf("%s",s);
l=strlen(s);
flag=1;
for(i=0;i<l;i++)
{
if(s[i]!='0' && s[i]!='1')
{
 flag=0;
}
}
if(flag==1)
printf("string is Valid\n");
else
printf("string is Not Valid\n");
if(flag==1)
{
flag1=0;
for(i=0;i<l-2;i++)
{
if(s[i]=='1')
{
if(s[i+1]=='0' && s[i+2]=='1')
{
flag1=1;
printf("Substring 101 exists. String accepted\n");
break;
}
}
}
if(flag1==0)
printf("Substring 101 does not exist. String not accepted\n");
}
}
```

**OUTPUT:**



```
enter a string to check:000111010100
Substring 101 exists.
String accepted

Process returned 0 (0x0)   execution time : 5.531 s
Press any key to continue.
```



```
enter a string to check:abbabba
string is Not Valid

Process returned 0 (0x0)   execution time : 5.828 s
Press any key to continue.
```

**EXP NO:8**

Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given languagerepresenting strings that start with b and end with a

**AIM:**

C program to simulate a Non-Deterministic Finite Automata (NFA) for the given languagerepresenting strings that start with b and end with a

**ALGORITHM:**

1. Get the input from the user.
2. Declares 3d matrix to store transitions and initialise all entries with -1.
3. m[0][0][0]=0
   m[0][0][1]=1
   m[0][1][0]=0
   m[0][1][1]=2
   m[1][1][0]=3
   m[2][0][0]=3

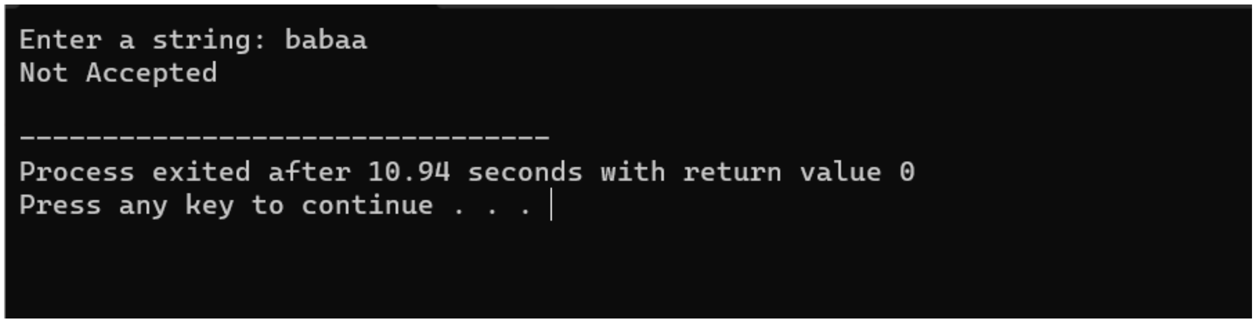
4.repeat steps for every character.

**PROGRAM:**

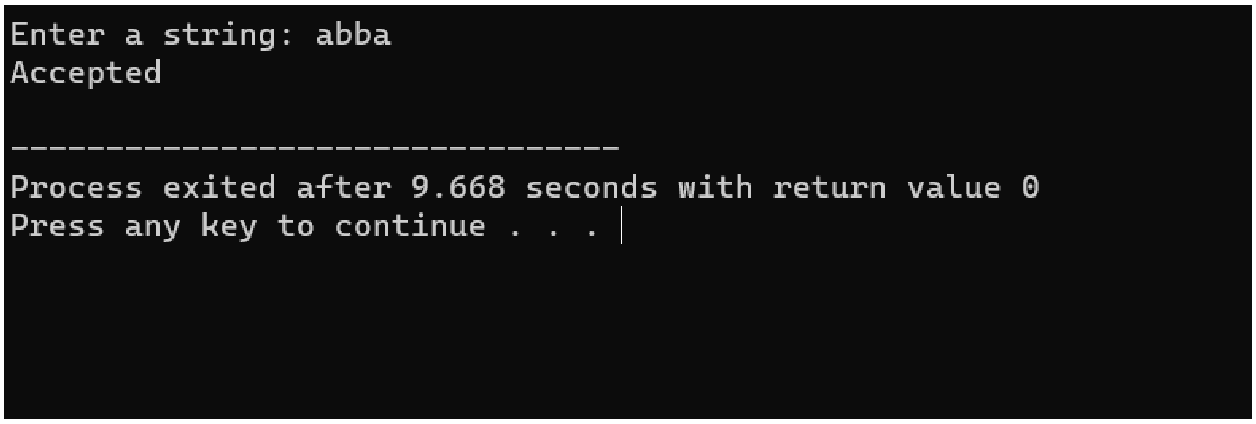
```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

// Function to simulate NFA for strings that start with 'b' and end with 'a'
bool isAccepted(char *str) {
    int state = 0;
    int len = strlen(str);

    for (int i = 0; i < len; i++) {
        switch (state) {
            case 0:
                if (str[i] == 'b') {
```

```
                state = 1;
            }
            break;

        case 1:
            if (str[i] == 'a') {
                state = 2;
            } else if (str[i] != 'b') {
                return false;
            }
            break;

        case 2:
            if (i != len - 1) {
                return false;
            }
            if (str[i] != 'a') {
                return false;
            }
            break;
        }
    }

    return (state == 2);
}

int main() {
    char input[100];
    printf("Enter a string: ");
    scanf("%s", input);

    if (isAccepted(input)) {
        printf("Accepted\n");
    } else {
        printf("Not Accepted\n");
    }

    return 0;
}
```

**OUTPUT:**



```
Enter a string: babaa
Not Accepted

--------------------------------
Process exited after 10.94 seconds with return value 0
Press any key to continue . . .
```



```
Enter a string: abba
Accepted

--------------------------------
Process exited after 9.668 seconds with return value 0
Press any key to continue . . .
```

**EXP NO:9**

Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given language representing strings that start with o and end with 1

**ALGORITHM:**

1. Get the following as input from the user.
 i. Number of states in the NFA
 ii. Number of symbols in the input alphabet and the symbols
 iii. Number of final states and their names 2. Declare a 3-dimensional matrix to store the transitions and initialize all the entries with -1
 3. Get the transitions from every state for every input symbol from the user and store it in the matrix.

m[0][0][0]=0 m[0][0][1]=1
Similarly, the other transitions can be stored as follows:
m[0][1][0]=0 (From state 0, for input b, one transition is to state 0)
m[0][1][1]=2 (From state 0, for input b, next transition is to state 2)
m[1][1][0]=3 (From state 1, for input b, move to state 3)
m[2][0][0]=3 (From state 2, for input a, move to state 3)
m[3][0][0]=3 (From state 3, for input a, move to state 3)
m[3][1][0]=3 (From state 3, for input b, move to state 3)
All the other entries in the matrix will be -1 indicating no moves
4. Get the input string from the user.

5. Find the length of the input string.
6. Read the input string character by character.
7. Repeat step 8 for every character
8. Refer the transition table for the entry corresponding to the present state and the current input symbol and update the next state. As there can be more than one transition, the next state will be an array.
9. From every state in the next state array, find the list of new transitions and update the next state array.
10. When we reach the end of the input, if at least one of the final states is present in the next state array, it means there is a path to a final state. So the input is accepted. Otherwise the input is not accepted.

**PROGRAM:**

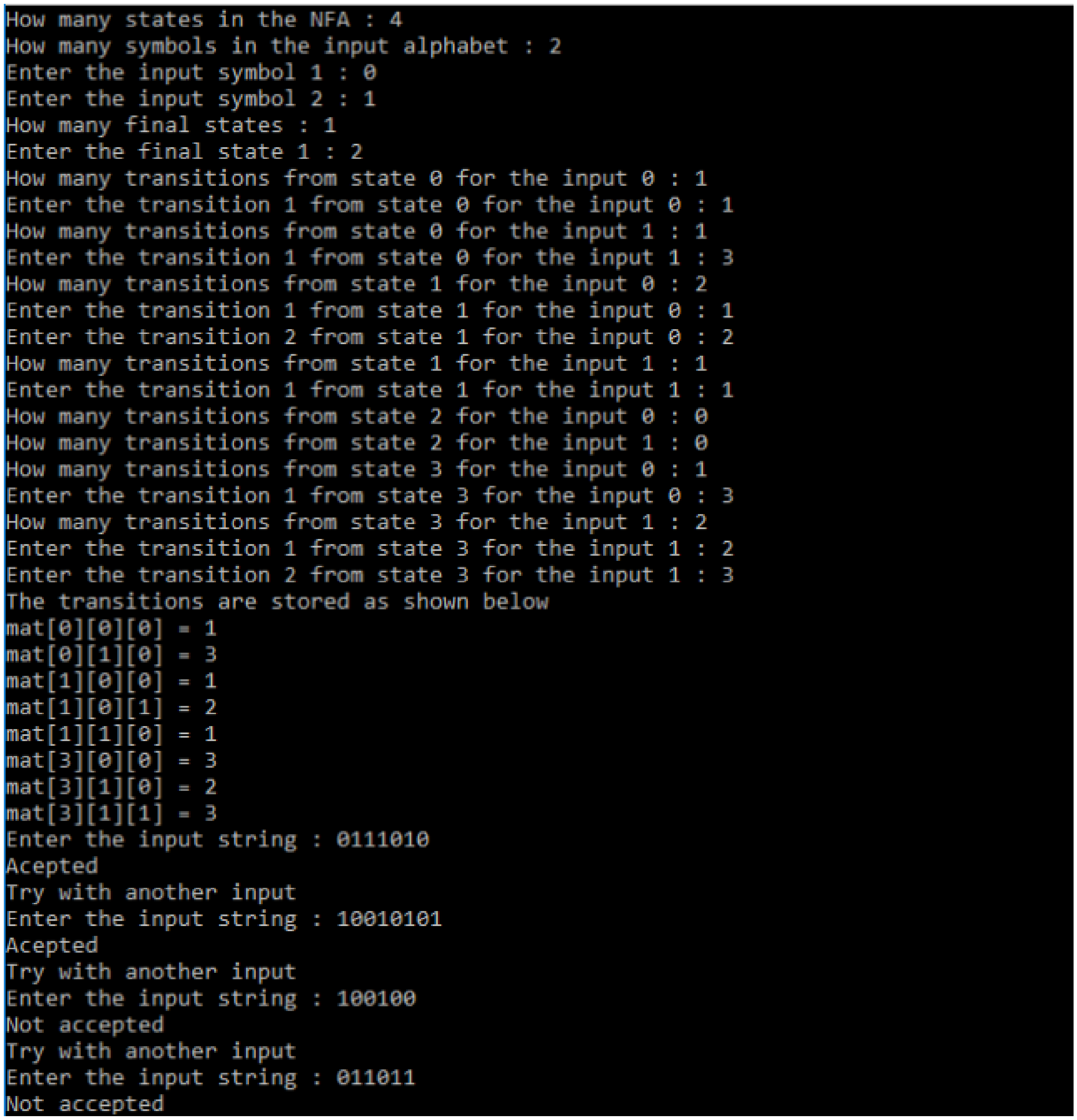
```
#include<stdio.h>
#include<string.h>
int main()
{
int i,j,k,l,m,next_state[20],n,mat[10][10][10],flag,p;
int num_states,final_state[5],num_symbols,num_final;
int present_state[20],prev_trans,new_trans;
char ch,input[20];
int symbol[5],inp,inp1;
printf("How many states in the NFA : ");
scanf("%d",&num_states);
printf("How many symbols in the input alphabet : ");
scanf("%d",&num_symbols);
for(i=0;i<num_symbols;i++)
{
printf("Enter the input symbol %d : ",i+1);
scanf("%d",&symbol[i]);
}
printf("How many final states : ");
scanf("%d",&num_final);
for(i=0;i<num_final;i++)
{
printf("Enter the final state %d : ",i+1);
scanf("%d",&final_state[i]);
}
//Initialize all entries with -1 in Transition table
for(i=0;i<10;i++)
{
```

```
for(j=0;j<10;j++)
{
for(k=0;k<10;k++)
{
mat[i][j][k]=-1;
}
}
}
//Get input from the user and fill the 3D transition table
for(i=0;i<num_states;i++)
{
for(j=0;j<num_symbols;j++)
{
printf("How many transitions from state %d for the input %d :
",i,symbol[j]);
scanf("%d",&n);
for(k=0;k<n;k++)
{
printf("Enter the transition %d from state %d for the input
%d : ",k+1,i,symbol[j]);
scanf("%d",&mat[i][j][k]);
}
}
}
printf("The transitions are stored as shown below\n");
 for(i=0;i<10;i++)
{
for(j=0;j<10;j++)
{
for(k=0;k<10;k++)
{
 if(mat[i][j][k]!=-1)
 printf("mat[%d][%d][%d] = %d\n",i,j,k,mat[i][j][k]);
}
}
}
while(1)
{
printf("Enter the input string : ");
scanf("%s",input);
```

```
present_state[0]=0;
prev_trans=1;
l=strlen(input);
for(i=0;i<l;i++)
{
 if(input[i]=='0')
 inp1=0;
 else if(input[i]=='1')
 inp1=1;
 else
 {
 printf("Invalid input\n");
 exit(0);
 }
 for(m=0;m<num_symbols;m++)
{
if(inp1==symbol[m])
{
inp=m;
break;
}
}
new_trans=0;
for(j=0;j<prev_trans;j++)
{
k=0;
p=present_state[j];
while(mat[p][inp][k]!=-1)
{
next_state[new_trans++]=mat[p][inp][k];
k++;
}
}
for(j=0;j<new_trans;j++)
{
present_state[j]=next_state[j];
}
prev_trans=new_trans;
}
flag=0;
```

```
for(i=0;i<prev_trans;i++)
{
for(j=0;j<num_final;j++)
{
if(present_state[i]==final_state[j])
{
flag=1;
break;
}
}
}
if(flag==1)
printf("Acepted\n");
else
printf("Not accepted\n");
printf("Try with another input\n");
}
}
```

**OUTPUT:**



```
How many states in the NFA : 4
How many symbols in the input alphabet : 2
Enter the input symbol 1 : 0
Enter the input symbol 2 : 1
How many final states : 1
Enter the final state 1 : 2
How many transitions from state 0 for the input 0 : 1
Enter the transition 1 from state 0 for the input 0 : 1
How many transitions from state 0 for the input 1 : 1
Enter the transition 1 from state 0 for the input 1 : 3
How many transitions from state 1 for the input 0 : 2
Enter the transition 1 from state 1 for the input 0 : 1
Enter the transition 2 from state 1 for the input 0 : 2
How many transitions from state 1 for the input 1 : 1
Enter the transition 1 from state 1 for the input 1 : 1
How many transitions from state 2 for the input 0 : 0
How many transitions from state 2 for the input 1 : 0
How many transitions from state 3 for the input 0 : 1
Enter the transition 1 from state 3 for the input 0 : 3
How many transitions from state 3 for the input 1 : 2
Enter the transition 1 from state 3 for the input 1 : 2
Enter the transition 2 from state 3 for the input 1 : 3
The transitions are stored as shown below
mat[0][0][0] = 1
mat[0][1][0] = 3
mat[1][0][0] = 1
mat[1][0][1] = 2
mat[1][1][0] = 1
mat[3][0][0] = 3
mat[3][1][0] = 2
mat[3][1][1] = 3
Enter the input string : 0111010
Acepted
Try with another input
Enter the input string : 10010101
Acepted
Try with another input
Enter the input string : 100100
Not accepted
Try with another input
Enter the input string : 011011
Not accepted
```

## EXP NO:10

Write a C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

**AIM**:C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

**ALGORITHM:**

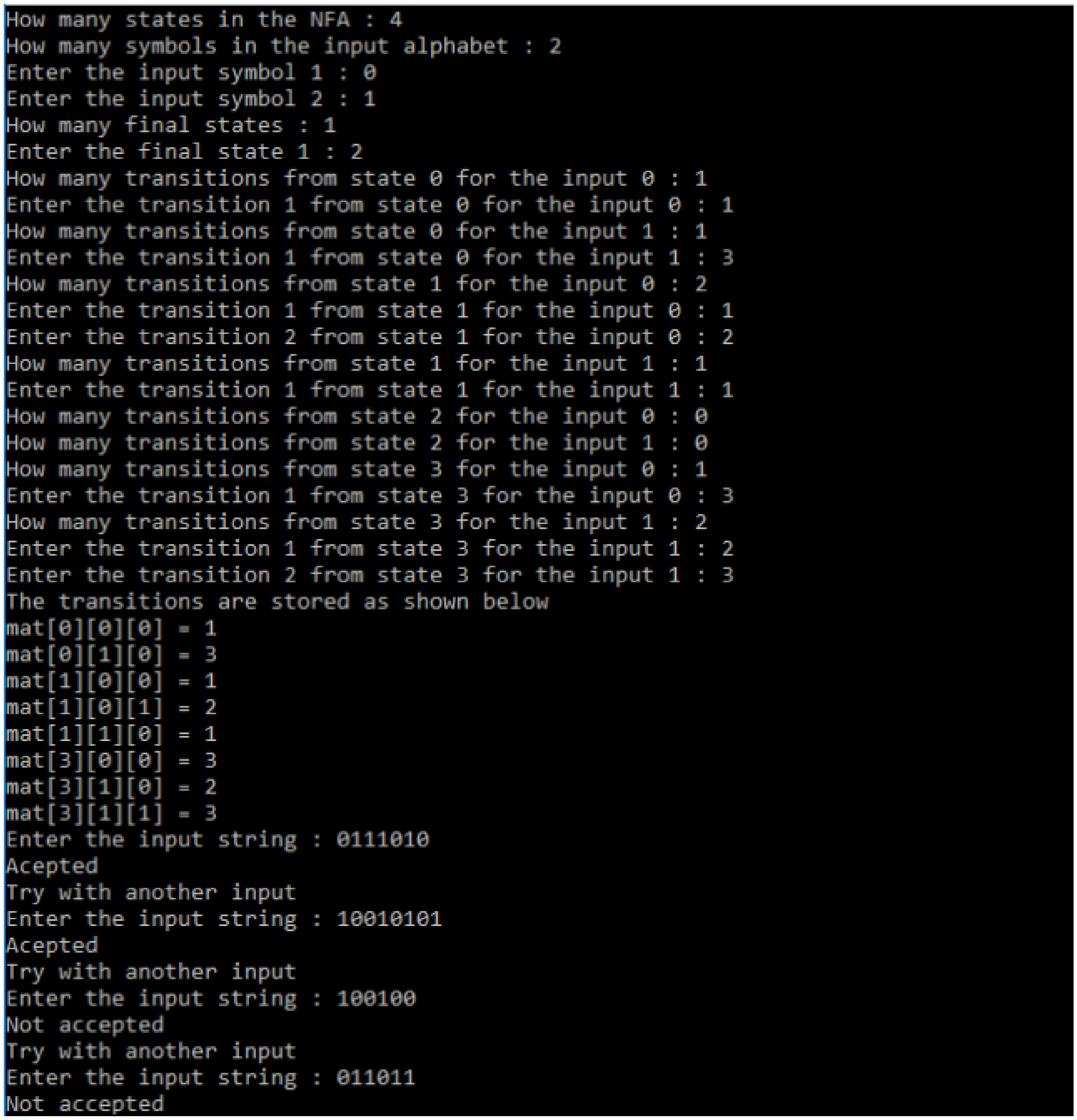
1. Get the following as input from the user.
i. Number of states in the NFA
ii. Number of symbols in the input alphabet including ε
iii. Input symbols iv. Number of final states and their names
2. Declare a 3-dimensional matrix to store the transitions and initialize all the entries with -1 3. Get the transitions from every state for every input symbol from the user and store it in the matrix
m[0][0][0]=1
From state 0, for input 0, there is no transition.
From state 0, for input 1, there is one transition to state 1,
m[0][2][0]=1 Similarly, the other transitions can be stored as follows:
m[1][0][0]=2 (From state 1, for input ε, the transition is to state 2)
m[1][1][0]=1 (From state 1, for input 0, the transition is to state 1)
All the other entries in the matrix will be -1 indicating no moves
4. Initialize a two-dimensional matrix e_closure with -1 in all the entries.
5. ε-closure of a state q is defined as the set of all states that can be reached from state q using only ε-transitions.

**PROGRAM:**

```
#include<stdio.h>
#include<string.h>
int trans_table[10][5][3];
char symbol[5],a;
int e_closure[10][10],ptr,state;
void find_e_closure(int x);
int main()
{
int i,j,k,n,num_states,num_symbols;
for(i=0;i<10;i++)
{
for(j=0;j<5;j++)
{
for(k=0;k<3;k++)
{
trans_table[i][j][k]=-1;
}
}
}
printf("How may states in the NFA with e-moves:");
scanf("%d",&num_states);
printf("How many symbols in the input alphabet including e :");
```

```
scanf("%d",&num_symbols);
printf("Enter the symbols without space. Give 'e' first:");
scanf("%s",symbol);
for(i=0;i<num_states;i++)
{
for(j=0;j<num_symbols;j++)
{
printf("How many transitions from state %d for the input
%c:",i,symbol[j]);
scanf("%d",&n);
for(k=0;k<n;k++)
{
printf("Enter the transitions %d from state %d for the input
%c :", k+1,i,symbol[j]);
scanf("%d",&trans_table[i][j][k]);
}
}
}
for(i=0;i<10;i++)
{
for(j=0;j<10;j++)
{
e_closure[i][j]=-1;
}
}
for(i=0;i<num_states;i++)
e_closure[i][0]=i;
for(i=0;i<num_states;i++)
{
if(trans_table[i][0][0]==-1)
continue;
else
{
state=i;
ptr=1;
find_e_closure(i);
}
}
for(i=0;i<num_states;i++)
{
printf("e-closure(%d)= {",i);
for(j=0;j<num_states;j++)
{
if(e_closure[i][j]!=-1)
{
printf("%d, ",e_closure[i][j]);
}
}
printf("}\n");
}
}
void find_e_closure(int x)
{
int i,j,y[10],num_trans;
i=0;
while(trans_table[x][0][i]!=-1)
{
y[i]=trans_table[x][0][i];
i=i+1;
}
num_trans=i;
for(j=0;j<num_trans;j++)
{
e_closure[state][ptr]=y[j];
ptr++;
find_e_closure(y[j]);
}
}
```

**OUTPUT:**



```
How many states in the NFA : 4
How many symbols in the input alphabet : 2
Enter the input symbol 1 : 0
Enter the input symbol 2 : 1
How many final states : 1
Enter the final state 1 : 2
How many transitions from state 0 for the input 0 : 1
Enter the transition 1 from state 0 for the input 0 : 1
How many transitions from state 0 for the input 1 : 1
Enter the transition 1 from state 0 for the input 1 : 3
How many transitions from state 1 for the input 0 : 2
Enter the transition 1 from state 1 for the input 0 : 1
Enter the transition 2 from state 1 for the input 0 : 2
How many transitions from state 1 for the input 1 : 1
Enter the transition 1 from state 1 for the input 1 : 1
How many transitions from state 2 for the input 0 : 0
How many transitions from state 2 for the input 1 : 0
How many transitions from state 3 for the input 0 : 1
Enter the transition 1 from state 3 for the input 0 : 3
How many transitions from state 3 for the input 1 : 2
Enter the transition 1 from state 3 for the input 1 : 2
Enter the transition 2 from state 3 for the input 1 : 3
The transitions are stored as shown below
mat[0][0][0] = 1
mat[0][1][0] = 3
mat[1][0][0] = 1
mat[1][0][1] = 2
mat[1][1][0] = 1
mat[3][0][0] = 3
mat[3][1][0] = 2
mat[3][1][1] = 3
Enter the input string : 0111010
Acepted
Try with another input
Enter the input string : 10010101
Acepted
Try with another input
Enter the input string : 100100
Not accepted
Try with another input
Enter the input string : 011011
Not accepted
```

## EXP NO 11

Write a C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

**AIM:**C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

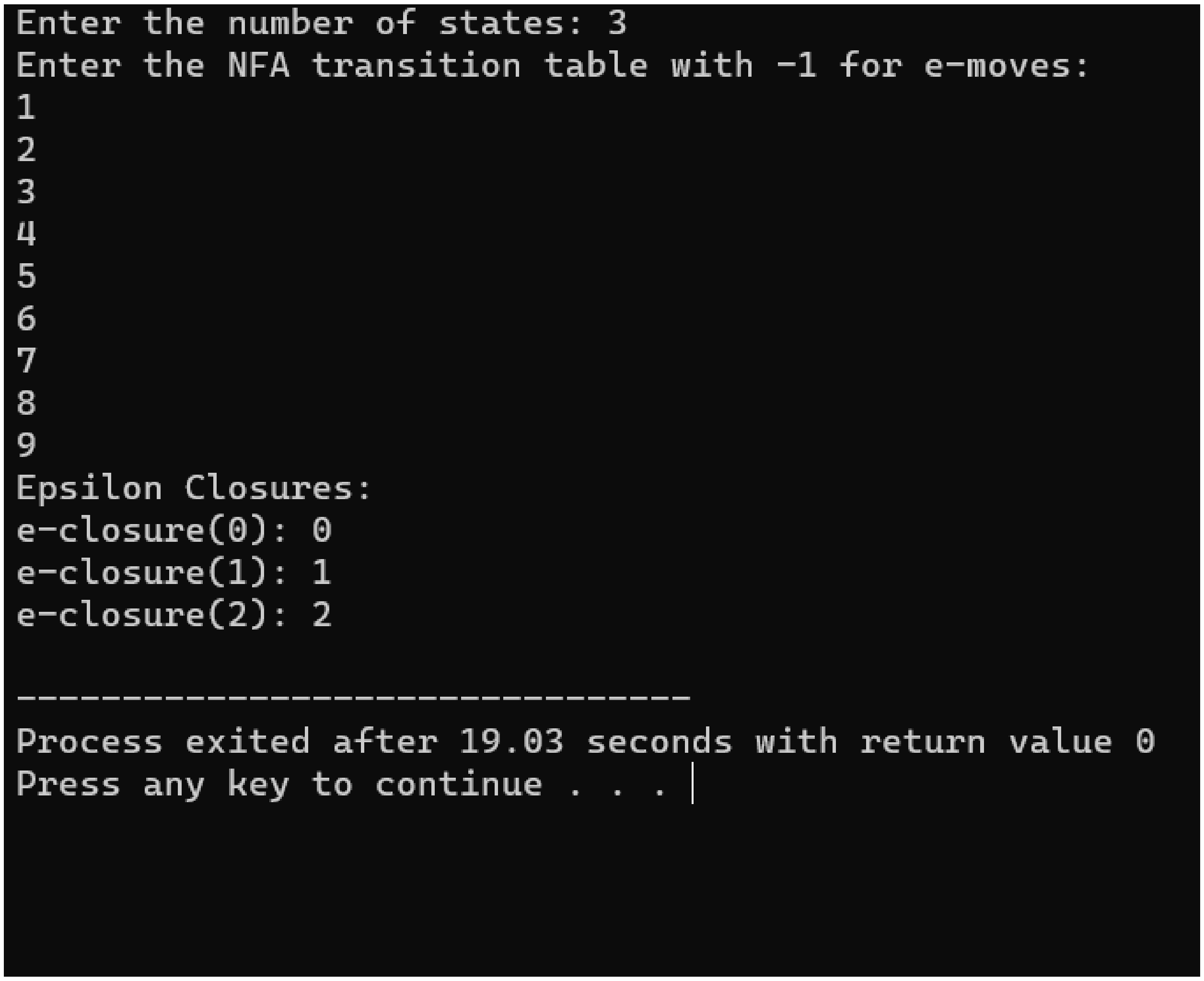
**ALGORITHM:**

1. Get the following as input from the user.
i. Number of states in the NFA
ii. Number of symbols in the input alphabet including ε
iii. Input symbols iv. Number of final states and their names
2. Declare a 3-dimensional matrix to store the transitions and initialize all the entries with -1 3. Get the transitions from every state for every input symbol from the user and store it in the matrix
m[0][0][0]=1
From state 0, for input 0, there is no transition.
From state 0, for input 1, there is one transition to state 1,
m[0][2][0]=1 Similarly, the other transitions can be stored as follows:
m[1][0][0]=2 (From state 1, for input ε, the transition is to state 2)
m[1][1][0]=1 (From state 1, for input 0, the transition is to state 1)
All the other entries in the matrix will be -1 indicating no moves
4. Initialize a two-dimensional matrix e_closure with -1 in all the entries.
5. ε-closure of a state q is defined as the set of all states that can be reached from state q using only ε-transitions.

**PROGRAM:**

```
#include <stdio.h>



#define MAX_STATES 10

#define MAX_ALPHABET 10



// Function to find ε-closure for a given state

void findEpsilonClosure(int nfa[MAX_STATES][MAX_ALPHABET], int n, int state, int visited[]) {

  if (visited[state]) {
      return;
  }

  visited[state] = 1;
  printf("%d ", state);

  // Iterate through transitions with ε-moves
  for (int i = 0; i < n; i++) {
      if (nfa[state][i] == -1) {
          findEpsilonClosure(nfa, n, i, visited);
      }
  }
}

// Function to find ε-closure for all states
void findAllEpsilonClosures(int nfa[MAX_STATES][MAX_ALPHABET], int n) {
  printf("Epsilon Closures:\n");

  for (int i = 0; i < n; i++) {
      int visited[MAX_STATES] = {0};
      printf("ε-closure(%d): ", i);
      findEpsilonClosure(nfa, n, i, visited);
      printf("\n");
  }
}

int main() {
  int n; // Number of states
  printf("Enter the number of states: ");
  scanf("%d", &n);

  int nfa[MAX_STATES][MAX_ALPHABET];

  // Initialize the NFA transition table
  printf("Enter the NFA transition table with -1 for ε-moves:\n");
  for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
          scanf("%d", &nfa[i][j]);
      }
  }

  findAllEpsilonClosures(nfa, n);

  return 0;
}
```

**OUTPUT:**



```
Enter the number of states: 3
Enter the NFA transition table with -1 for e-moves:
1
2
3
4
5
6
7
8
9
Epsilon Closures:
e-closure(0): 0
e-closure(1): 1
e-closure(2): 2

--------------------------------
Process exited after 19.03 seconds with return value 0
Press any key to continue . . .
```