

**(Regulation 2021)**

# COMPUTER SCIENCE AND ENGINEERING PROGRAMME

<u>LIST OF EXPERIMENTS</u>	Marks
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	
2. 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 l	
3. Write a C program to find $\epsilon$ -closure for all the states in a Non-Deterministic Finite Automata (NFA) with $\epsilon$ -moves	
4. To write a C program to check whether a string belongs to the grammar $S \rightarrow 0 A 1$ $A \rightarrow 0 A \mid 1 A \mid \epsilon$	
5. To write a C program to check whether a string belongs to the grammar $S \rightarrow 0 S 0 \mid 1 S 1 \mid 0 \mid 1 \mid \epsilon$	
6. To write a C program to check whether a string belongs to the grammar $S \rightarrow 0 S 0 \mid A A \rightarrow 1 A \mid \epsilon$	
7. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG) $S \rightarrow 0 S 1 \mid \epsilon$	
8. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG) $S \rightarrow A 1 0 1 A, A \rightarrow 0 A \mid 1 A \mid \epsilon$	
9. Design DFA using simulator to accept the input string "a", "ac", and "bac"	
10. Design a Push Down Automata that accepts the language $L = \{w \mid w \in (a + b)^* \text{ and } n_a(w) = n_b(w)\}$ $n_a(w)$ is the number of a's in w $n_b(w)$ is the number of b's in w	

11. Design PDA using simulator to accept the input string $a^n b^{2n}$ i. $L = \{ a^n b^{2n} \mid w \in (a + b) \}$	
12. Design TM using simulator to accept the input string $a^n b^n$ $L = \{ a^n b^n \mid w \in (a + b) \}$	
13. Design TM using simulator to accept the input string $a^n b^{2n}$	
14. Design TM using simulator to accept the input string Palindrome ababa	
15. Design TM using simulator to accept the input string ww	
16. Design TM using simulator to perform addition of 'aa' and 'aaa'	
17. Design TM using simulator to perform subtraction of aaa-aa	
18. Design DFA using simulator to accept even number of a's	
19. Design DFA using simulator to accept odd number of a's	
20. Design DFA using simulator to accept the string the end with ab over set {a,b} W= aaabab	
21. Design DFA using simulator to accept the string having 'ab' as substring over the set {a,b}	
22. Design DFA using simulator to accept the string start with a or b over the set {a,b}	
23. Design TM using simulator to accept the input string Palindrome bbabb	
24. Design TM using simulator to accept the input string wcw	
25. Design DFA using simulator to accept the string even number of a's and odd number of b's	
26. Design DFA using simulator to accept the input string "bc", "c", and "bcaaa"	
27. Design NFA to accept any number of a's where input={a,b}	
28. Design PDA using simulator to accept the input string $a^n b^n$	
29. Design TM using simulator to perform string comparison where $w = \{aba\}$	
30. Design DFA using simulator to accept the string having 'abc' as substring over the set {a,b,c}	
31. Design DFA using simulator to accept even number of c's over the set {a,b,c}	

32. Design DFA using simulator to accept strings in which a's always appear tripled over input {a,b}	
33. Design NFA using simulator to accept the string the start with a and end with b over set {a,b} and check W= abaab is accepted or not	
34. Design NFA using simulator to accept the string that start and end with different symbols over the input {a,b}	
35. Let L be regular language, L consist set of string over { a,b) number a's minus number b's less than or equal to 2. Design DFA to accept the the language L.	
36. Design DFA using simulator to accept the string the end with abc over set {a,b,c) W= abbaababc	
37. Design NFA to accept any number of b's where input={a,b}	
38. The Automatic Teller Machine (Atm)	
39. Pattern Searching	
40. Vending Machine	
41. Natural Language Processing	
42.construct a Turing Machine to perform the function Multiplication, using Subroutines.	

**EXP NO : 1**

## **DETERMINISTIC FINITE AUTOMATA (DFA)**

**AIM :**

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

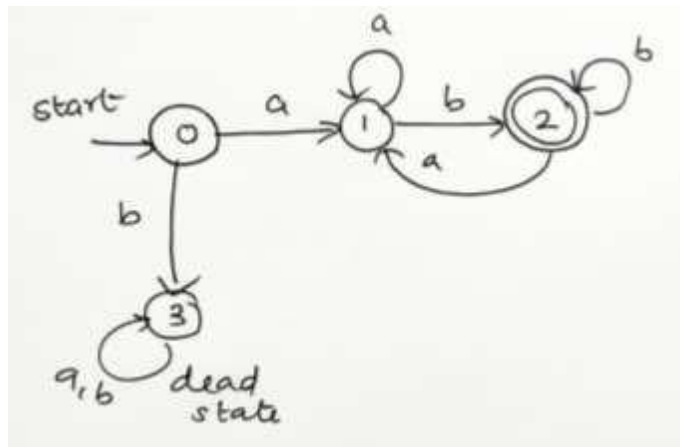
**ALGORITHM :**

1. Draw 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.


### **Example:**

Simulate a DFA for the language representing strings over  $\Sigma=\{a,b\}$  that start with a and end with b

### ***Design of the DFA***



### ***Transition Table:***

State / Input	a	b
→ 0	1	3
1	1	2
 2	1	2
3	3	3

### **PROGRAM :**

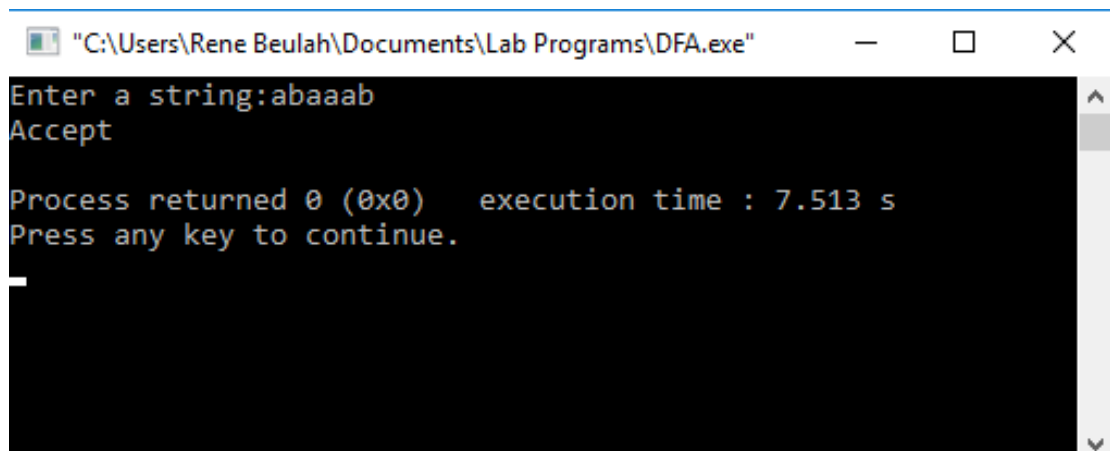
```
#include<stdio.h>
#include<string.h>
#define max 20
int main()
{
    int trans_table[4][2]={{1,3},{1,2},{1,2},{3,3}};
    int final_state=2,i; int
    present_state=0; int
    next_state=0;
    int invalid=0;
    char input_string[max];
```

```

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=1; present_state=next_state;
}
if(invalid==1)
{
    printf("Invalid input");
}
else if(present_state==final_state)
    printf("Accept\n");
else
    printf("Don't Accept\n");
}

```

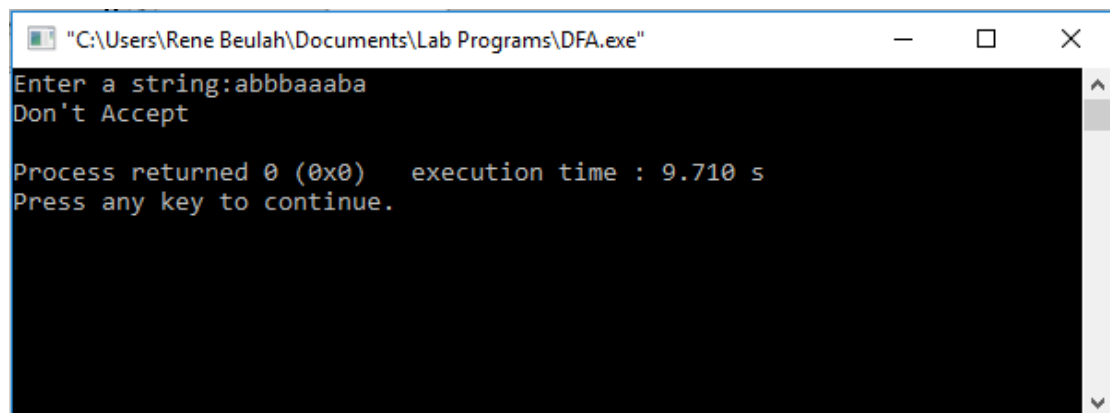
## OUTPUT



```

"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:abbbbaaaba
Don't Accept
Process returned 0 (0x0)   execution time : 9.710 s
Press any key to continue.

```

## EXP NO : 2

### NON-DETERMINISTIC FINITE AUTOMATA (NFA)

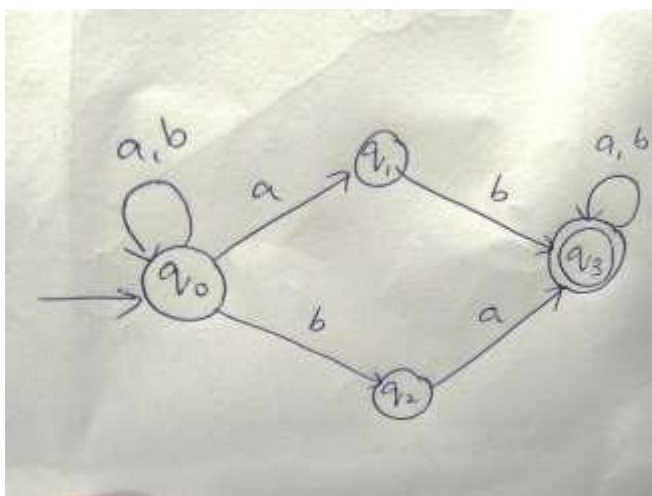
#### AIM :

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

#### 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.

For example, consider the NFA shown below.



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 0<sup>th</sup> symbol is a and 1<sup>st</sup> 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 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]);
            sca
            nf("
            %d
            ",&
            n);
            for(
            k=0
            ;k<
```

n;k  
++)  
{

```

printf("Enter the transition %d from state %d
%d : ",k+1,i,symbol[j]);
                                for the input 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];

```

```

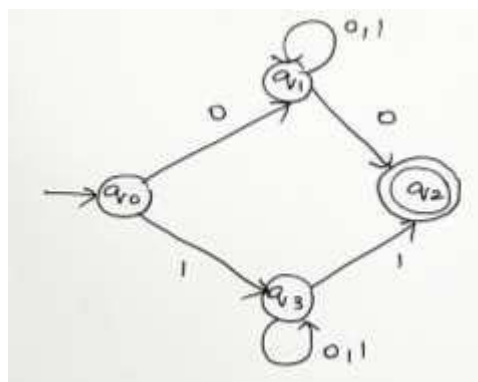
        while(mat[p][inp][k]!=-1)
        {
            next_state[new_trans++]=mat[p][inp][k]; k++;
        }
    }
    for(j=0;j<new_trans;j++)
    {
        present_state[j]=next_state[j];
    }
    prev_trans=new_trans;
}
flag=0;
for(i=0;i<prev_trans;i++)
{
    for(j=0;j<num_final;j++)
    {
        if(present_state[i]==final_state[j])
        {
            flag=1;
            break;
        }
    }
}
if(flag==1)
    printf("Accepted\n");
else
    printf("Not accepted\n");
printf("Try with another input\n");
}
}

```

### **Example:**

Simulate a NFA for the language representing strings over  $\Sigma=\{a,b\}$  that start and end with the same symbol

### ***Design of the NFA***



## Transition Table

State / Input	0	1
→ 0	1	3
1	{1,2}	1
2	-	-
3	3	{2,3}

## OUTPUT:

```
"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
Accepted
Try with another input
Enter the input string : 10010101
Accepted
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

### FINDING $\epsilon$ -CLOSURE FOR NFA WITH $\epsilon$ -MOVES

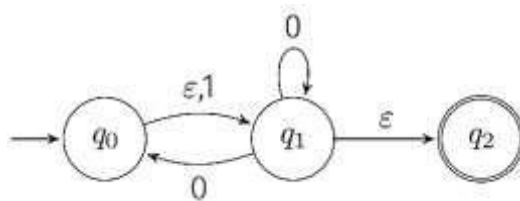
#### AIM :

To write a C program to find  $\epsilon$ -closure of a Non-Deterministic Finite Automata with  $\epsilon$ -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  $\epsilon$
  - 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.

For example, consider the NFA shown below.



There are 3 states 0, 1, and 2

There are three input symbols  $\epsilon$ , 0 and 1. As the array index always starts with 0, we assume 0<sup>th</sup> symbol is  $\epsilon$ , 1<sup>st</sup> symbol is 0 and 2<sup>nd</sup> symbol is 1.

The transitions will be stored in the matrix as follows:

From state 0, for input  $\epsilon$ , 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:  $m[1][0][0]=2$

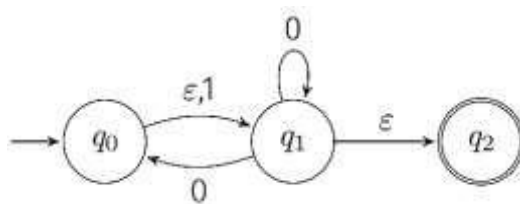
(From state 1, for input  $\epsilon$ , the transition is to state 2)  $m[1][1][0]=1$

(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.  $\epsilon$ -closure of a state  $q$  is defined as the set of all states that can be reached from state  $q$  using only  $\epsilon$ -transitions.

Example:

Consider the NFA with  $\epsilon$ -transitions given below:



$$\epsilon\text{-closure}(0)=\{0,1,2\}$$

$$\epsilon\text{-closure}(1)=\{1,2\}$$

$$\epsilon\text{-closure}(2)=\{2\}$$

Here, we see that  $\epsilon$ -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  $\epsilon$ -closure as follows:

If there is an  $\epsilon$ -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  $\epsilon$

7. For every state, print the  $\epsilon$ -closure values

### **The function *find\_e\_closure(i)***

This function finds  $\epsilon$ -closure of a state recursively by tracing all the  $\epsilon$ - 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 scanf("%d",&n);
for(k=0;k<n;k++)
{
            printf("Enter the transitions %d from state %d
for the input 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)
{

```

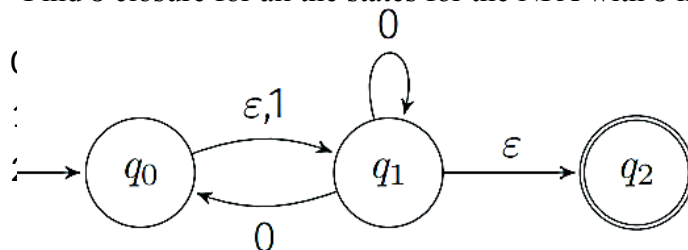
```

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]);
}
}

```

### Example:

Find  $\epsilon$ -closure for all the states for the NFA with  $\epsilon$ -moves given below:



### TRANSITION TABLE :

State / Input	$\epsilon$	0	1
$\rightarrow$ 0	1	-	1
1	2	{0,1}	-
2	-	-	-

## 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**

### **CHECKING WHETHER A STRING BELONGS TO A GRAMMAR**

#### **AIM :**

To write a C program to check whether a string belongs to the grammar

$$S \rightarrow 0 A 1$$

$$A \rightarrow 0 A \mid 1 A \mid \varepsilon$$

#### **Language defined by the Grammar:**

Set of all strings over  $\Sigma=\{0,1\}$  that start with 0 and end with 1

#### **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 :

```
"C:\Users\Rene Beulah\Documents\Lab Programs\ex4a.exe"
enter a string to check:01010111101
string is accepted

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

```
"C:\Users\Rene Beulah\Documents\Lab Programs\ex4a.exe"
enter a string to check:011101010110
string is Not accepted

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

```
"C:\Users\Rene Beulah\Documents\Lab Programs\ex4a.exe"
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 5**

### **CHECKING WHETHER A STRING BELONGS TO A GRAMMAR**

#### **AIM :**

To write a C program to check whether a string belongs to the grammar  $S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \epsilon$

#### **Language defined by the Grammar**

Set of all strings over  $\Sigma = \{0,1\}$  that are palindrome

#### **ALGORITHM :**

1. Get the input string from the user.
2. Find the length of the string. Let it be  $n$ .
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 1<sup>st</sup> symbol and  $n^{\text{th}}$  symbol are the same, 2<sup>nd</sup> symbol and  $(n-1)^{\text{th}}$  symbol are the same and so on, then the given string is palindrome. So, print "String accepted". Otherwise, print "String not accepted"



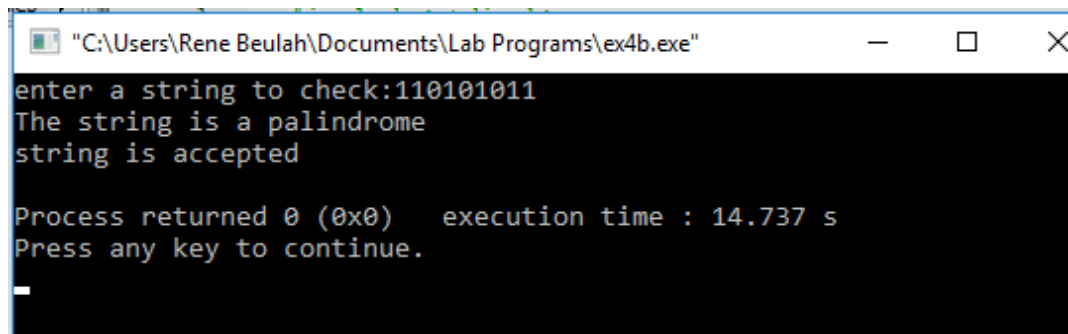
## PROGRAM :

```
#include<stdio.h>
#include<string.h>
void main()
{
    char s[100];
    int i,flag,flag1,a,b;
    int l;
    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)
    {
        flag1=1;
        a=0;b=l-1;
        while(a!=(l/2))
        {
            if(s[a]!=s[b])
            {
                flag1=0;
            }
            a=a+1;
            b=b-1;
        }
        if (flag1==1)
        {
            printf("The string is a palindrome\n");
        }
        else
        {
            printf("string is not accepted\n");
        }
    }
}

printf("The string is not a palindrome\n");
printf("string is Not
```

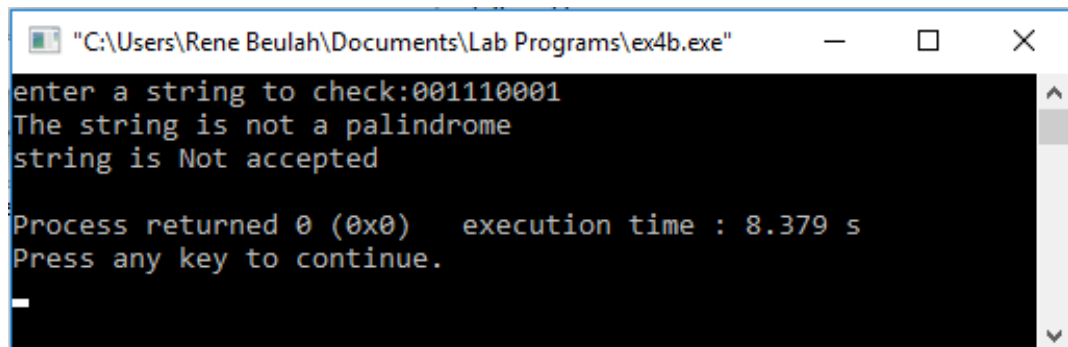
accepted\n");

## OUTPUT :



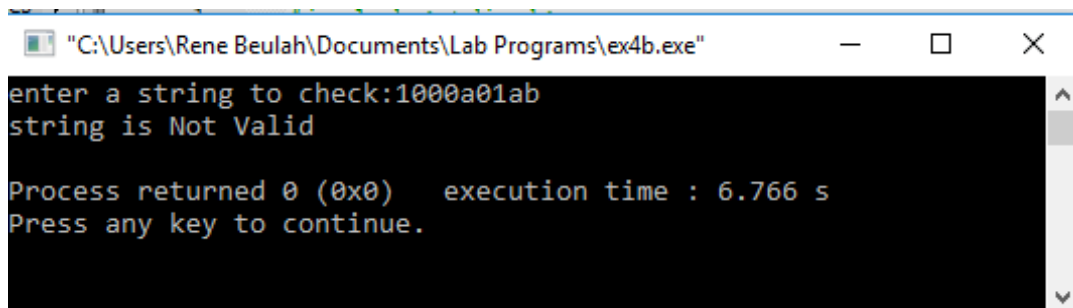
```
"C:\Users\Rene Beulah\Documents\Lab Programs\ex4b.exe"
enter a string to check:110101011
The string is a palindrome
string is accepted

Process returned 0 (0x0)   execution time : 14.737 s
Press any key to continue.
_
```



```
"C:\Users\Rene Beulah\Documents\Lab Programs\ex4b.exe"
enter a string to check:001110001
The string is not a palindrome
string is Not accepted

Process returned 0 (0x0)   execution time : 8.379 s
Press any key to continue.
_
```



```
"C:\Users\Rene Beulah\Documents\Lab Programs\ex4b.exe"
enter a string to check:1000a01ab
string is Not Valid

Process returned 0 (0x0)   execution time : 6.766 s
Press any key to continue.
_
```

## EXP 6

### CHECKING WHETHER A STRING BELONGS TO A GRAMMAR

#### AIM :

To write a C program to check whether a string belongs to the grammar

$$\begin{aligned} S &\rightarrow 0 S 0 \mid \\ A A &\rightarrow 1 A \mid \\ &\epsilon \end{aligned}$$

#### Language defined by the Grammar

Set of all strings over  $\Sigma=\{0,1\}$  satisfying  $0^n 1^m 0^n$

#### 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 :

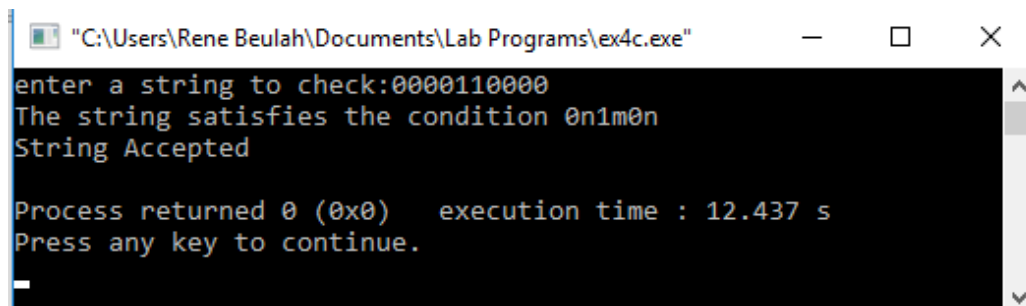
```
#include<stdio.h>
#include<string.h>
void main()
{
    char s[100];
    int i,flag,flag1,a,b;
    int l,count1,count2;
    printf("enter a string to check:");
    scanf("%s",s);
    l=strlen(s);
    flag=1;
    for(i=0;i<l;i++)
    {
        if(s[i]!='0' && s[i]!='1')
        {
            flag=0;
        }
    }
    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");
        }
    }
}

```

## OUTPUT :

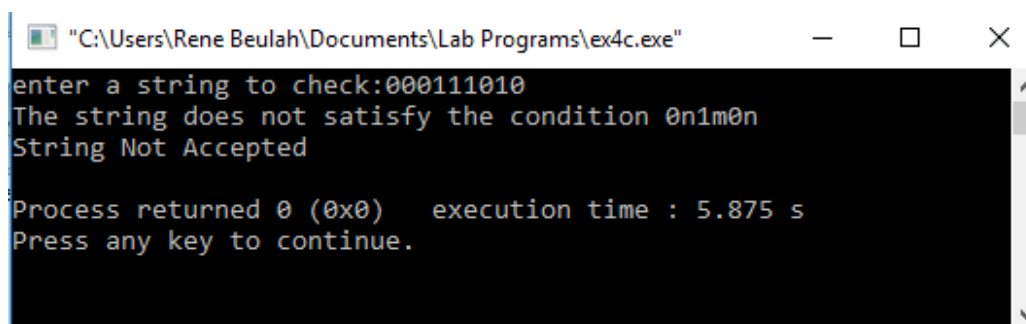


```

"C:\Users\Rene Beulah\Documents\Lab Programs\ex4c.exe"
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.

```

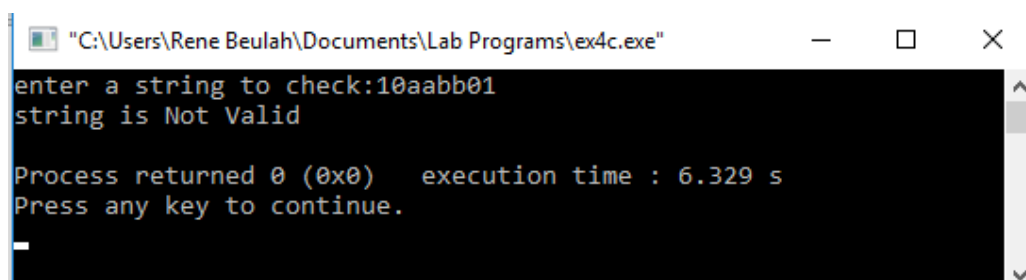


```

"C:\Users\Rene Beulah\Documents\Lab Programs\ex4c.exe"
enter a string to check:0001111010
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.

```



```

"C:\Users\Rene Beulah\Documents\Lab Programs\ex4c.exe"
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 7**

### **CHECKING WHETHER A STRING BELONGS TO A GRAMMAR**

#### **AIM :**

To write a C program to check whether a string belongs to the grammar

$$S \rightarrow 0 S 1 \mid \varepsilon$$

#### **Language defined by the Grammar**

Set of all strings over  $\Sigma = \{0,1\}$  satisfying  $0^n 1^n$

#### **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
            chec
            k
            first
            half
            conta
            ins
            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
01n\n"); printf("String Accepted\n");
        }
        else
        {
            printf("The string does not satisfy the
condition 01n\n"); printf("String Not
Accepted\n");
        }
    }
}

```

## OUTPUT :

```

"C:\Users\Rene Beulah\Documents\Lab Programs\ex4d.exe"
enter a string to check:0000011111
The string satisfies the condition 01n\n
String Accepted

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

```

```

"C:\Users\Rene Beulah\Documents\Lab Programs\ex4d.exe"
enter a string to check:000111010
The string does not satisfy the condition 01n\n
String Not Accepted

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

```

```

"C:\Users\Rene Beulah\Documents\Lab Programs\ex4d.exe"
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 8**

### **CHECKING WHETHER A STRING BELONGS TO A GRAMMAR**

#### **AIM :**

To write a C program to check whether a string belongs to the grammar  $S \rightarrow$

$A 1 0 1 A$

$A \rightarrow 0 A \mid 1 A \mid \epsilon$

#### **Language defined by the Grammar**

Set of all strings over  $\Sigma=\{0,1\}$  having 101 as a substring

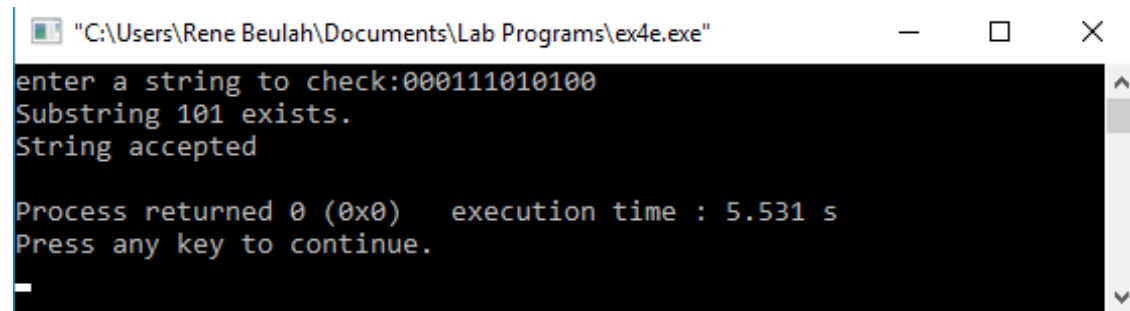
#### **ALGORITHM :**

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  $i^{\text{th}}$  input symbol is 1, check whether  $(i+1)^{\text{th}}$  symbol is 0 and  $(i+2)^{\text{th}}$  symbol is 1. If so, the string has the substring 101. So print “String Accepted”. Otherwise, print “String Not Accepted”

## PROGRAM :

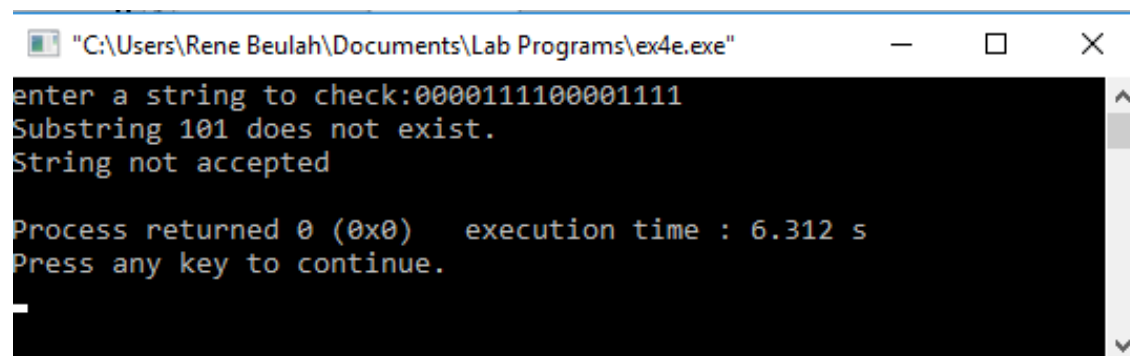
```
#include<stdio.h>
#include<string.h>
int main()
{
    char s[100];
    int i,flag,flag1;
    int l;
    printf("enter a string to check:");
    scanf("%s",s);
    l=strlen(s);
    flag=1;
    for(i=0;i<l;i++)
    {
        if(s[i]!='0' && s[i]!='1')
        {
            flag=0;
        }
    }
    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 :



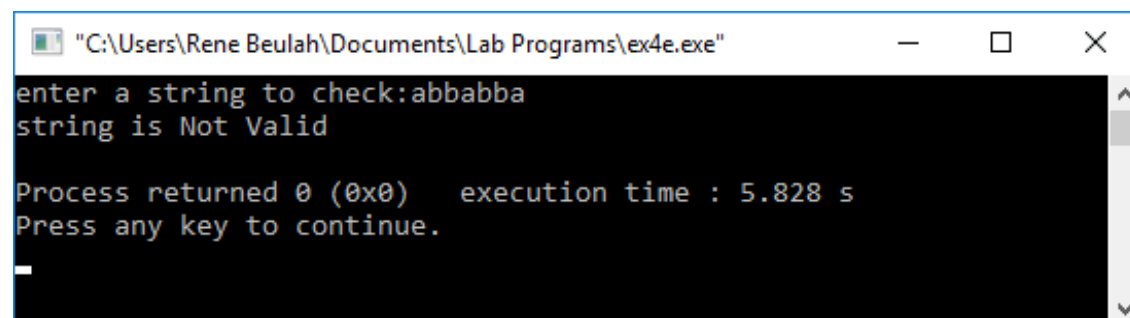
```
"C:\Users\Rene Beulah\Documents\Lab Programs\ex4e.exe"
enter a string to check:000111010100
Substring 101 exists.
String accepted

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



```
"C:\Users\Rene Beulah\Documents\Lab Programs\ex4e.exe"
enter a string to check:0000111100001111
Substring 101 does not exist.
String not accepted

Process returned 0 (0x0)   execution time : 6.312 s
Press any key to continue.
_
```



```
"C:\Users\Rene Beulah\Documents\Lab Programs\ex4e.exe"
enter a string to check:abbabba
string is Not Valid

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