**SAVEETHA SCHOOL OF ENGINEERING**

## SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

**COMPUTER SCIENCE AND ENGINEERING PROGRAMME**

LAB MANUAL CSA13

THEORY OF COMPUTATION

**(Regulation 2021)**

**COMPUTER SCIENCE AND ENGINEERING PROGRAMME**

|  |  |
| --- | --- |
| **LIST OF EXPERIMENTS** | 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 |  |
| 1. 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 |  |
| 1. Write a C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves |  |
| 1. To write a C program to check whether a string belongs to the grammar   S → 0 A 1  A → 0 A | 1 A | ε |  |
| 1. To write a C program to check whether a string belongs to the grammar S -> 0 S 0 | 1 S 1 | 0 | 1 | ε |  |
| 1. To write a C program to check whether a string belongs to the grammar   S -> 0 S 0 | A A -> 1 A | ε |  |
| 1. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG) S → 0S1 | ε |  |
| 1. 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 |  |
| 1. Design DFA using simulator to accept the input string “a” ,”ac”,and ”bac” |  |
| 1. Design a Push Down Automata that accepts the language   is the number of a’s in w  is the number of b’s in w |  |
| 1. Design PDA using simulator to accept the input string anb2n |  |
| 1. Design TM using simulator to accept the input string anbn |  |
| 1. Design TM using simulator to accept the input string anb2n |  |
| 1. Design TM using simulator to accept the input string Palindrome ababa |  |
| 1. .Design TM using simulator to accept the input string ww |  |
| 1. Design TM using simulator to perform addition of ‘aa’ and ‘aaa’ |  |
| 1. Design TM using simulator to perform subtraction of aaa-aa |  |
| 1. .Design DFA using simulator to accept even number of a’s |  |
| 1. Design DFA using simulator to accept odd number of a’s |  |
| 1. Design DFA using simulator to accept the string the end with ab over set {a,b) W= aaabab |  |
| 1. Design DFA using simulator to accept the string having ‘ab’ as substring over the set {a,b} |  |
| 1. Design DFA using simulator to accept the string start with a or b over the set {a,b} |  |
| 1. Design TM using simulator to accept the input string Palindrome bbabb |  |
| 1. Design TM using simulator to accept the input string wcw |  |
| 1. Design DFA using simulator to accept the string even number of a’s and odd number of b’s |  |
| 1. Design DFA using simulator to accept the input string “bc” ,”c”,and ”bcaaa” |  |
| 1. Design NFA to accept any number of a’s where input={a,b} |  |
| 1. Design PDA using simulator to accept the input string anbn |  |
| 1. Design TM using simulator to perform string comparison where w={aba aba} |  |
| 1. Design DFA using simulator to accept the string having ‘abc’ as substring over the set {a,b,c} |  |
| 1. Design DFA using simulator to accept even number of c’s over the set {a,b,c} |  |
| 1. Design DFA using simulator to accept strings in which a’s always appear tripled over input {a,b} |  |
| 1. 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 |  |
| 1. Design NFA using simulator to accept the string that start and end with different symbols over the input {a,b} |  |
| 1. 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. |  |
| 1. Design DFA using simulator to accept the string the end with abc over set {a,b,c) W= abbaababc |  |
| 1. Design NFA to accept any number of b’s where input={a,b} |  |
| 1. The Automatic Teller Machine (Atm) |  |
| 1. Pattern Searching |  |
| 1. Vending Machine |  |
| 1. 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.

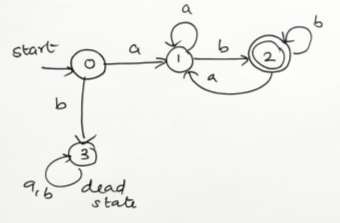
**ALGORTIHM :**

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 𝚺={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=l; present\_state=next\_state;

}

if(invalid==l)

{

printf("Invalid input");

}

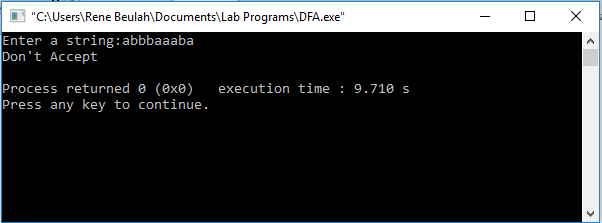
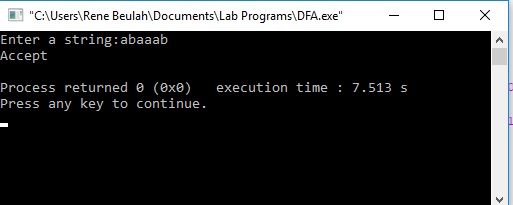
else if(present\_state==final\_state) printf("Accept\n");

else

printf("Don't Accept\n");

}

OUTPUT



# EXP NO : 2

**NON-DETERMINISTIC FINITE AUTOMATA (NFA)**

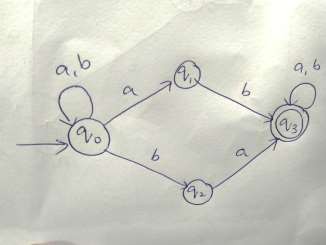
**AIM :**

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

**ALGORTIHM :**

1. Get the following as input from the user.
   1. Number of states in the NFA
   2. Number of symbols in the input alphabet and the symbols
   3. 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 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

1. Get the input string from the user.
2. Find the length of the input string.
3. Read the input string character by character.
4. Repeat step 8 for every character
5. 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.
6. From every state in the next state array, find the list of new transitions and update the next state array.
7. 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++)

{

",i,symbol[j]);

printf("How many transitions from state %d for the input %d :

scanf("%d",&n); for(k=0;k<n;k++)

{

%d : ",k+1,i,symbol[j]);

}

}

}

printf("Enter the transition %d from state %d 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("Acepted\n");

else

printf("Not accepted\n");

printf("Try with another input\n");

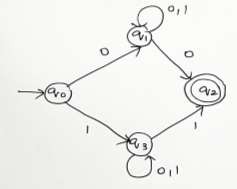
}

}

**Example:**

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

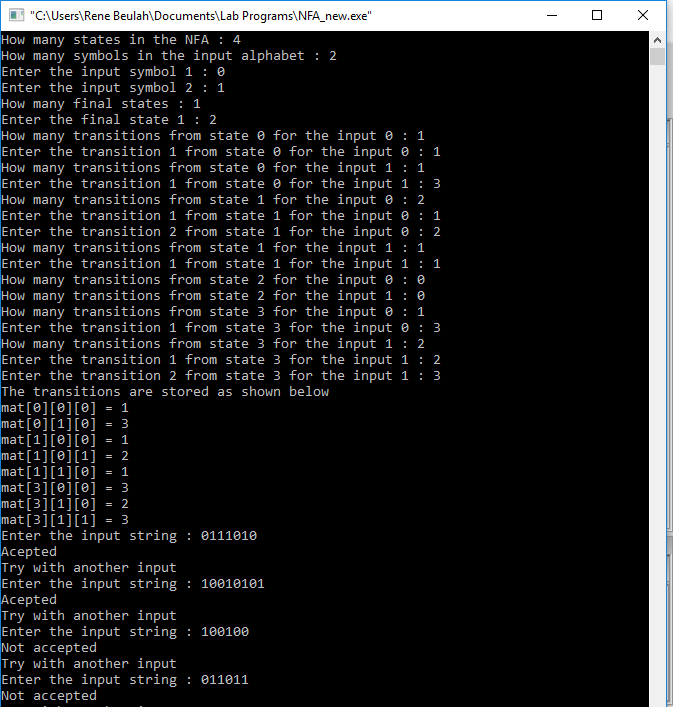
***Design of the NFA***



***Transition Table***

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

**OUTPUT:**



**EXP NO : 3**

**FINDING ε-CLOSURE FOR NFA WITH ε-MOVES**

**AIM :**

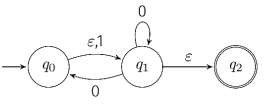
To write a C program to find ε-closure of a Non-Deterministic Finite

Automata with ε-moves

**ALGORTIHM :**

1. Get the following as input from the user.
   1. Number of states in the NFA
   2. Number of symbols in the input alphabet including ε
   3. Input symbols
   4. 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 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, whichcan be stored in the matrix as

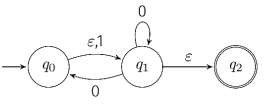
m[0][2][0]=1

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

1. Initialize a two-dimensional matrix e\_closure with -1 in all the entries.
2. 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;

1. 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

1. 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 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++)

{

%c:",i,symbol[j]);

printf("How many transitions from state %d for the input scanf("%d",&n);

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

{

%c :", k+1,i,symbol[j]);

}

}

}

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

{

printf("Enter the transitions %d from state %d for the input scanf("%d",&trans\_table[i][j][k]);

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++)

{

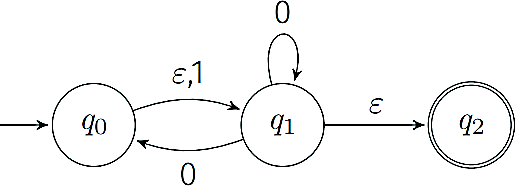
e\_closure[state][ptr]=y[j]; ptr++; find\_e\_closure(y[j]);

}

}

# Example:

Find ε-closure for all the states for the NFA with ε-moves given below:



0

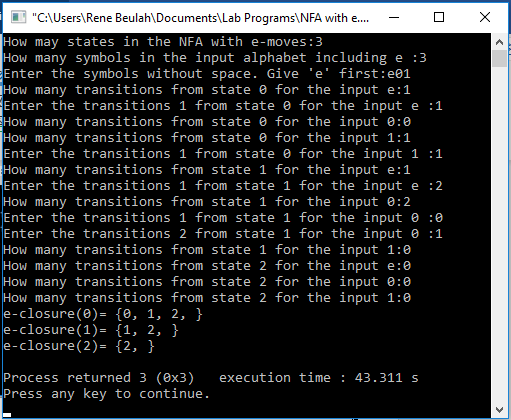
1

2

**TRANSITION TABLE :**

|  |  |  |  |
| --- | --- | --- | --- |
| **State / Input** | **ε** | **0** | **1** |
| → 0 | 1 | - | 1 |
| 1 | 2 | {0,1} | - |
| 2 | - | - | - |

**OUTPUT :**



**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 → 0 A 1

A → 0 A | 1 A | ε

**Language defined by the Grammar:**

Set of all strings over 𝚺={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");

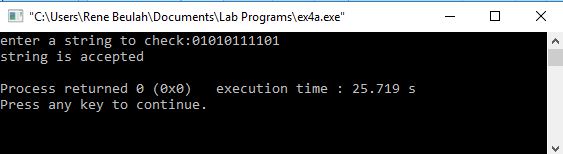
else

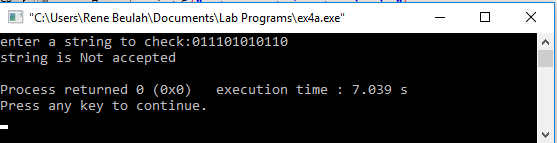
}

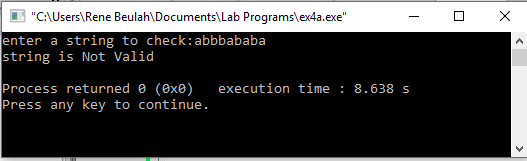
}

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

**OUTPUT :**







**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 -> 0 S 0 | 1 S 1 | 0 | 1 | ε

**Language defined by the Grammar**

Set of all strings over 𝚺={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 1st symbol and nth symbol are the same, 2nd symbol and (n-1)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)

{

}

else

{

}

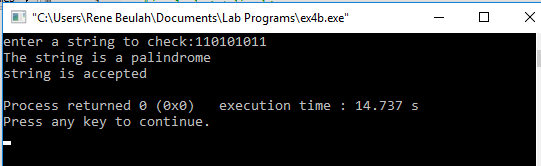
}

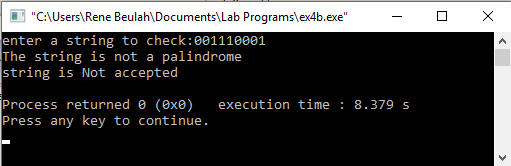
}

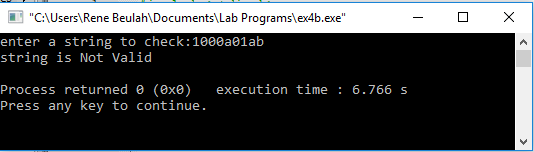
printf("The string is a palindrome\n"); printf("string is accepted\n");

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

**OUTPUT :**







**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

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*

1. Skip all 1’s
2. Count the number of 0’s in the end and store it in the variable *count2*
3. 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

{

}

else

{

} i++;

}

count2++;

flag1=0;

if(flag1==1)

{

if(count1==count2)

{

}

else

{

}

else

{

}

printf("The string satisfies the condition 0n1m0n\n"); printf("String Accepted\n");

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

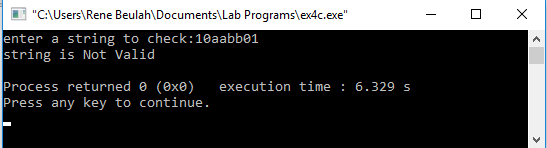
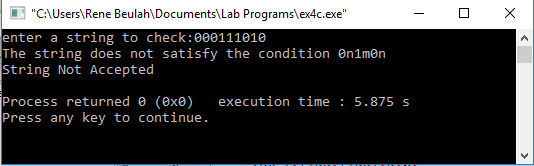
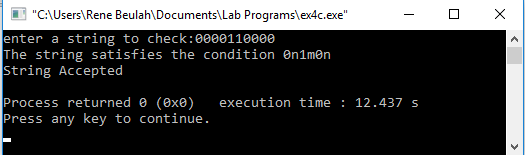
printf("The string does not satisfy the condition 0n1m0n\n"); printf("String Not Accepted\n");

}

}

}

**OUTPUT :**



**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 -> 0 S 1 | ε

**Language defined by the Grammar**

Set of all strings over 𝚺={0,1} satisfying 0n1n

**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

{

}

else

{

printf("The string does not satisfy the condition 0n1n\n"); printf("String Not Accepted\n");

// 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)

{

}

else

{

}

}

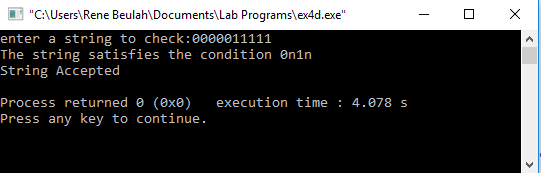
}

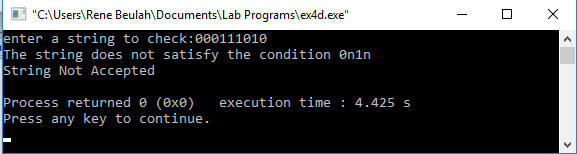
}

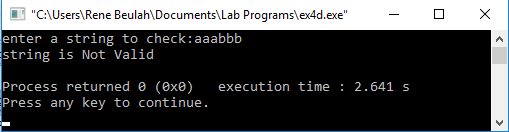
printf("The string satisfies the condition 0n1n\n"); printf("String Accepted\n");

printf("The string does not satisfy the condition 0n1n\n"); printf("String Not Accepted\n");

**OUTPUT :**







**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 -> A 1 0 1 A

A -> 0 A | 1 A | ε

**Language defined by the Grammar**

Set of all strings over 𝚺={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 ith 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;

}

}

}

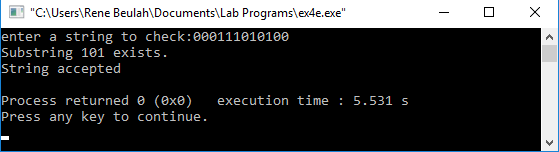
if(flag1==0)

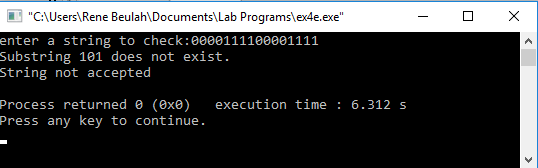
printf("Substring 101 does not exist. String not accepted\n");

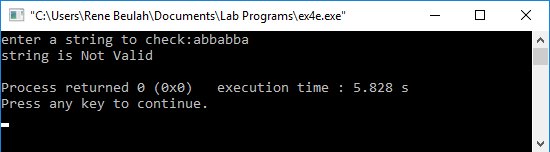
}

}

**OUTPUT :**



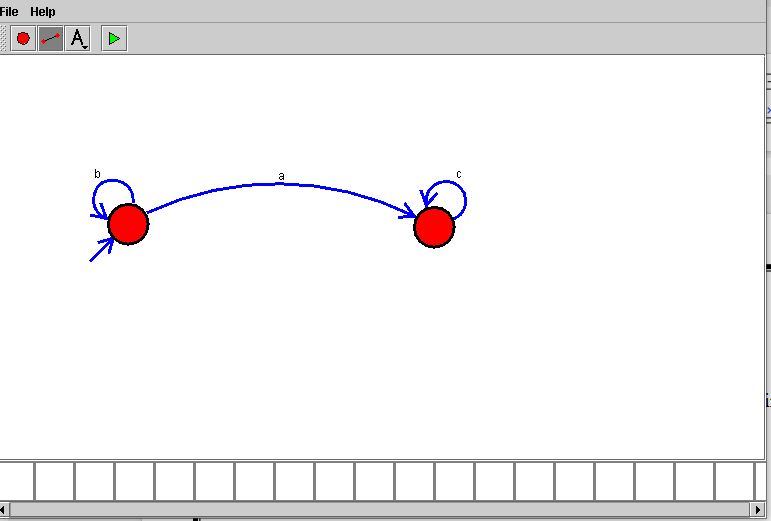




**EXP :9**

**AIM:**Design DFA using simulator to accept the input string “a” ,”ac”,and ”bac”

Simulaton



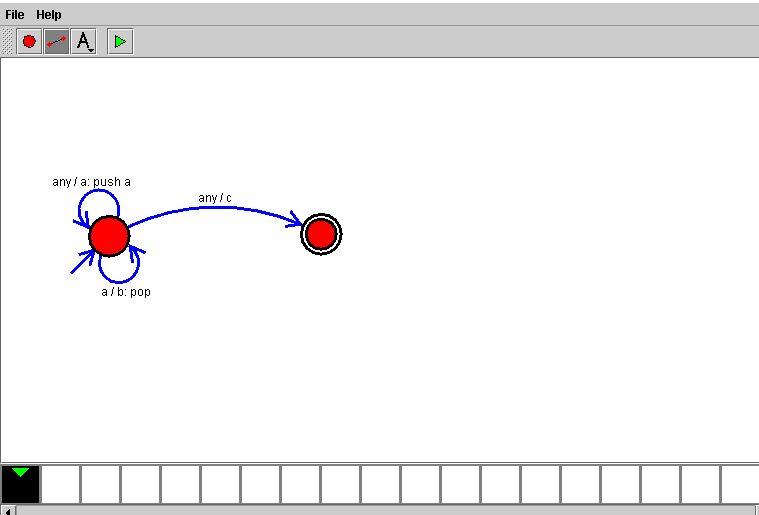
**EXP :10**

**AIM:**Design a Push Down Automata that accepts the language

is the number of a’s in w

is the number of b’s in w

Simulation



**EXP :11**

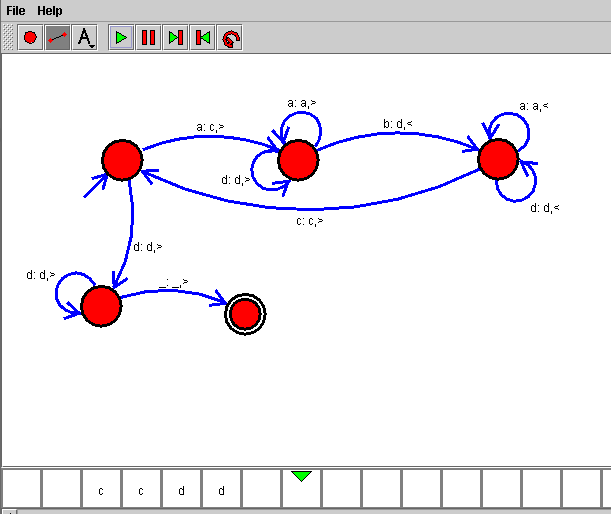
**AIM**:Design PDA using simulator to accept the input string anb2n

Simulation:

**EXP :12**

**AIM**:Design TM using simulator to accept the input string anbn

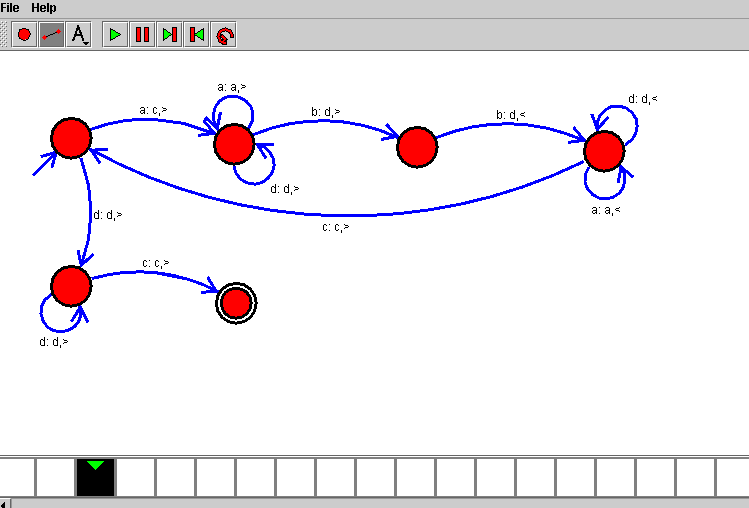
Simulation



**EXP :13**

**AIM**:Design TM using simulator to accept the input string anb2n

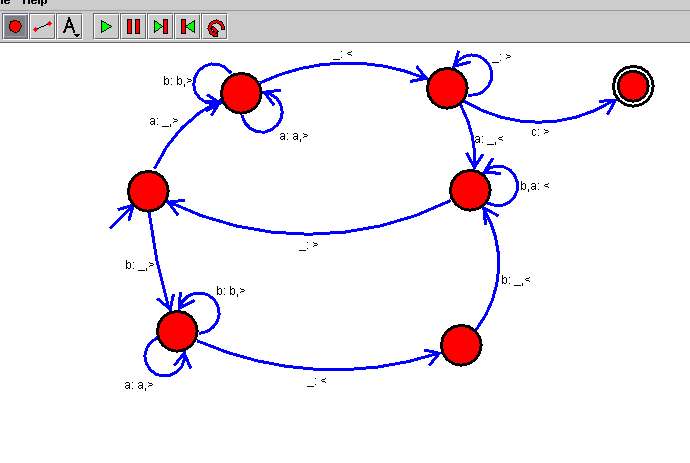
Simulation:



**EXP :14**

**AIM**:Design TM using simulator to accept the input string Palindrome ababa

Simulation:



**EXP :15**

**AIM:**Design TM using simulator to accept the input string ww



**EXP :16**

**AIM:**Design TM using simulator to perform addition of ‘aa’ and ‘aaa’

W= aa+ aaaa

After Addition of a’s = aaaaaa

Simulation:



**EXP :17**

**AIM:**Design TM using simulator to perform subtraction of aaa-aa

**Logic**

W= aaa-aa

The Result of Subtraction is = a

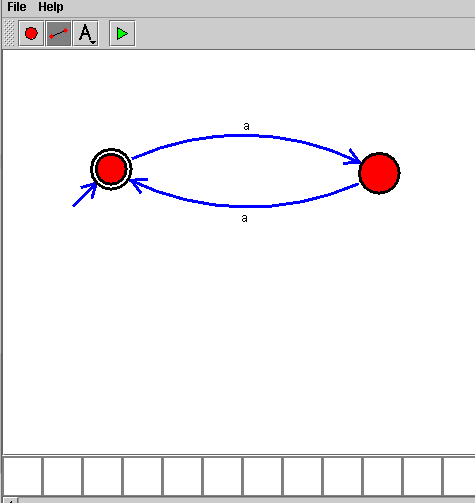


**EXP :18**

**AIM:**Design DFA using simulator to accept even number of a’s

W{ aa, aaaa, aaaaaa}

Simulation

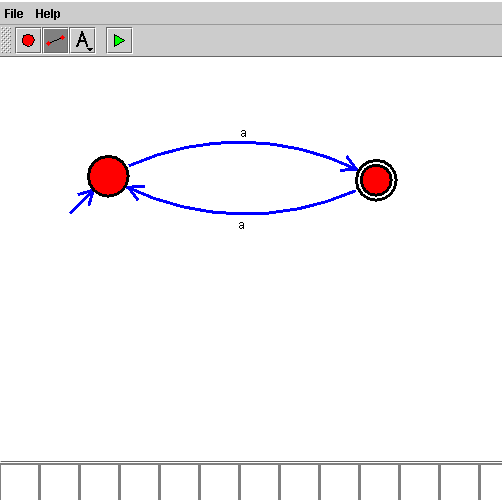


**EXP :19**

**AIM:**Design DFA using simulator to accept odd number of a’s

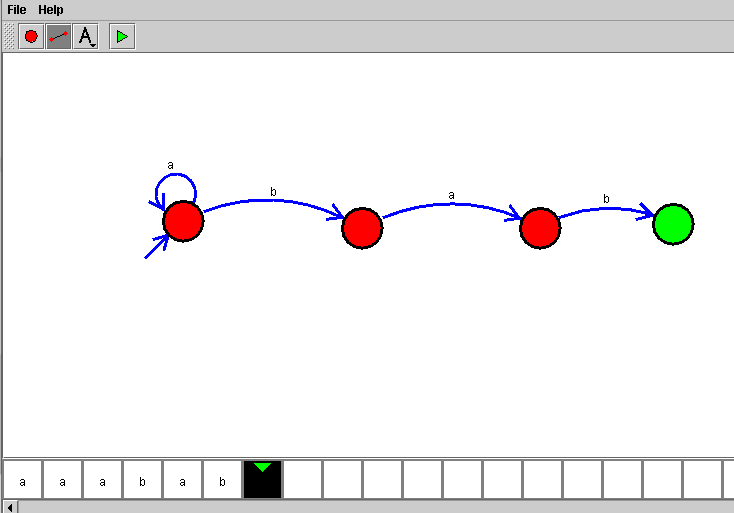
W{a, aaa, aaaaa}

Simulation



**EXP :20**

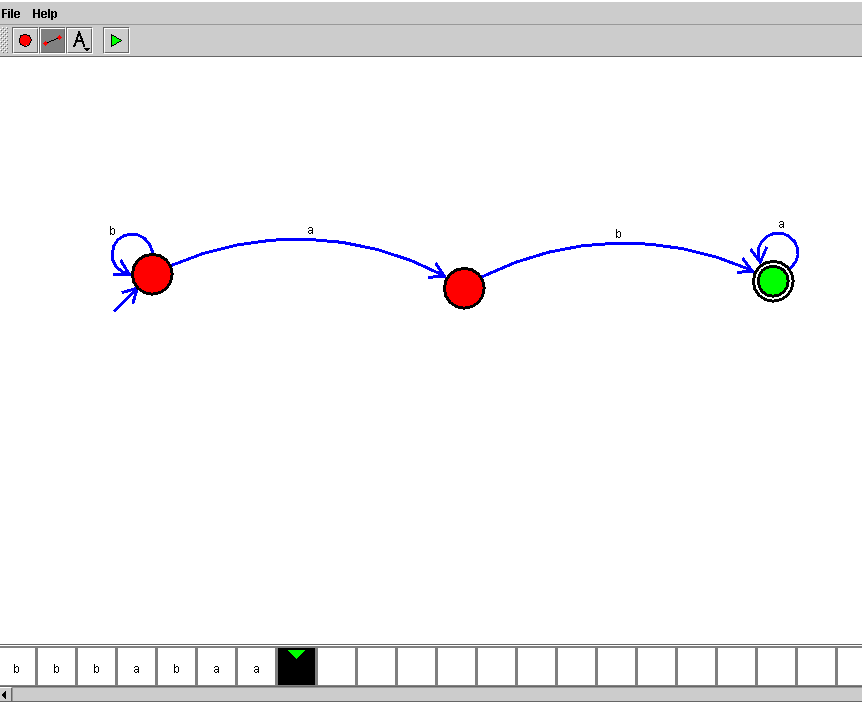
**AIM:**Design DFA using simulator to accept the string the end with ab over set {a,b) W= aaabab



**EXP :21**

**AIM:**Design DFA using simulator to accept the string having ‘ab’ as substring over the set {a,b}

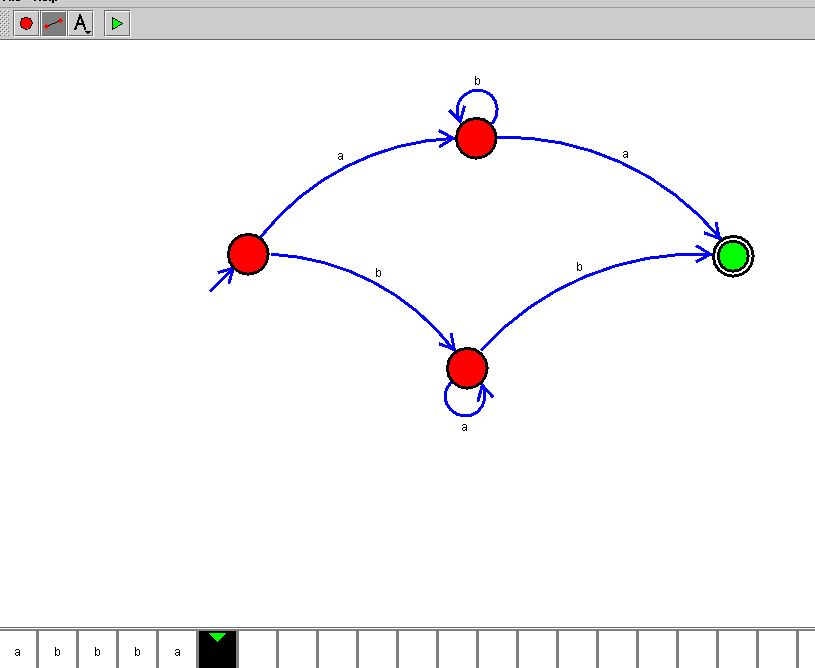
W= babaaaaa



**EXP :22**

**AIM:**Design DFA using simulator to accept the string start with a or b over the set {a,b}

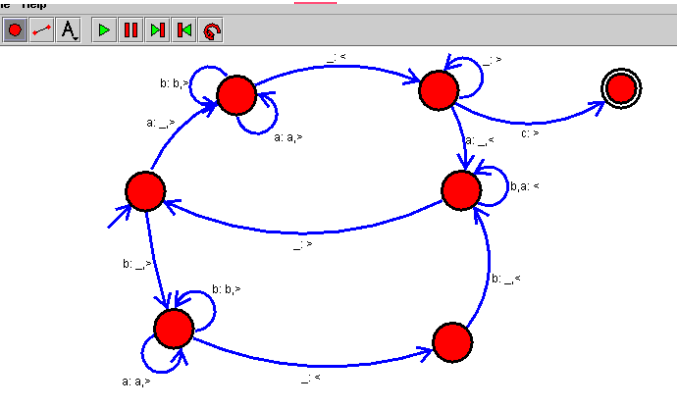
W= { abbbba, baaaaab}



**EXP :23**

**AIM:**Design TM using simulator to accept the input string Palindrome bbabb

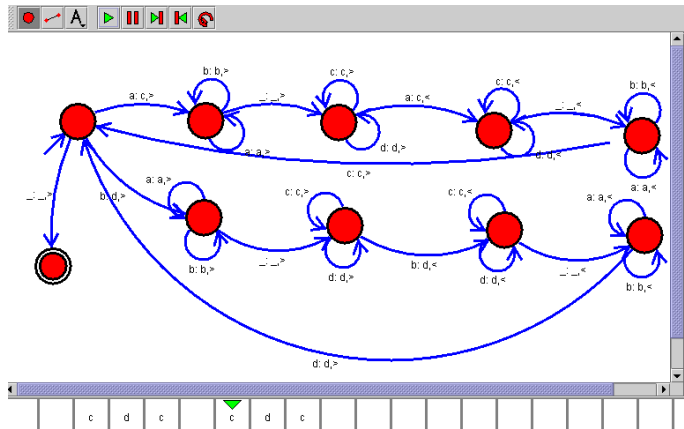
W={bbabb}



**EXP :24**

**AIM:**Design TM using simulator to accept the input string wcw

W={ aa aa,bb bb , ab ab}

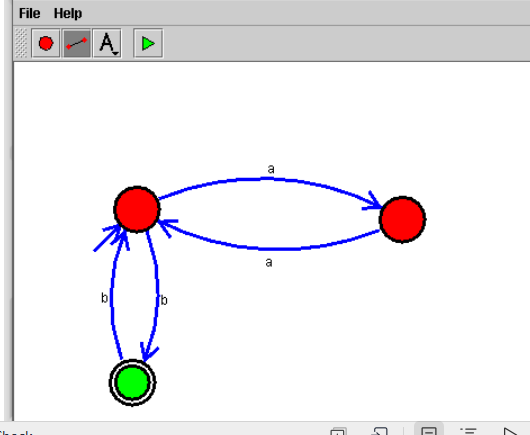


**EXP :25**

**AIM:**Design DFA using simulator to accept the string even number of a’s and odd number of b’s

**W={aab, bbaab}**

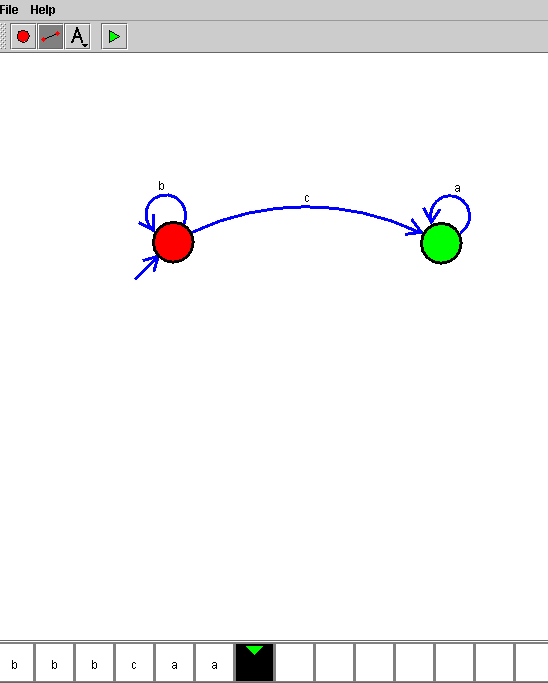
**Simulation**



**EXP :26**

**AIM:**Design DFA using simulator to accept the input string “bc” ,”c”,and ”bcaaa”

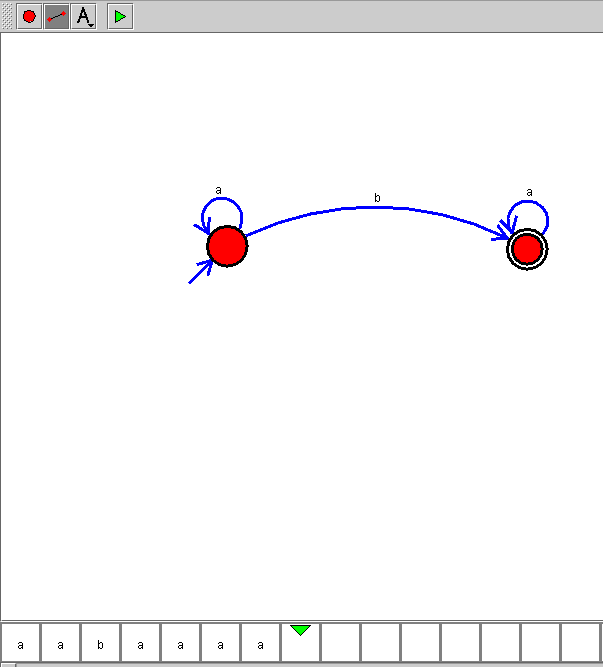
Simulation



**EXP :27**

**AIM:**Design NFA to accept any number of a’s where input={a,b}

W={ aaaab, baaaaaa}



**EXP :28**

**AIM:**Design PDA using simulator to accept the input string anbn

W={ aabb , aaabbb}



**EXP :29**

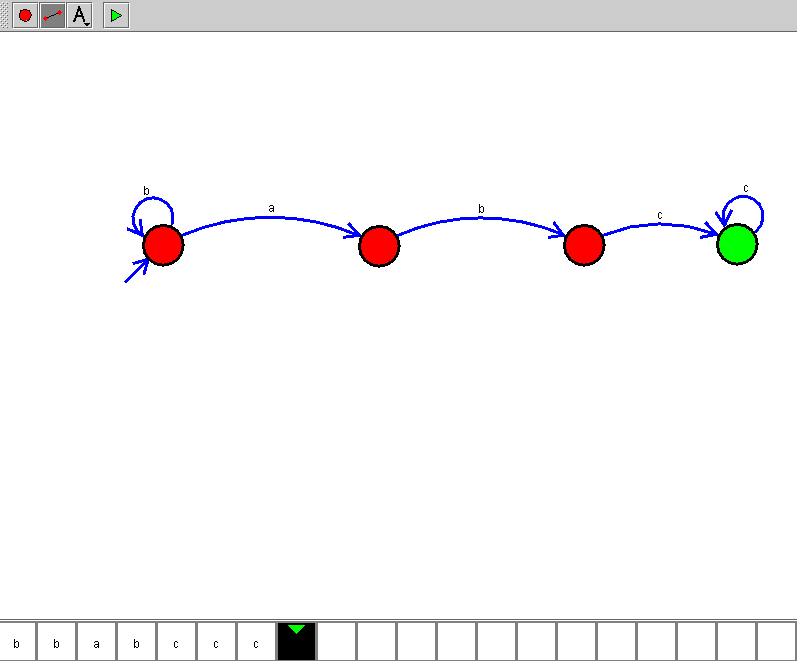
**AIM:**Design TM using simulator to perform string comparison where w={aba aba}



**EXP :30**

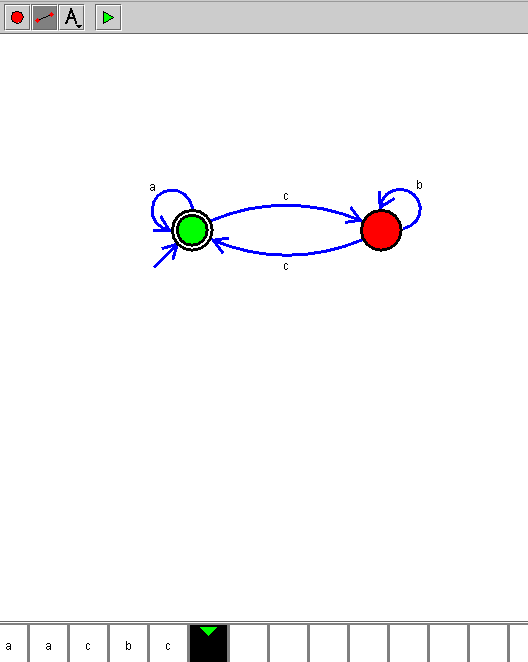
**AIM:**Design DFA using simulator to accept the string having ‘abc’ as substring over the set {a,b,c}

W= { aaaabcccc, abccccc}



**EXP :31**

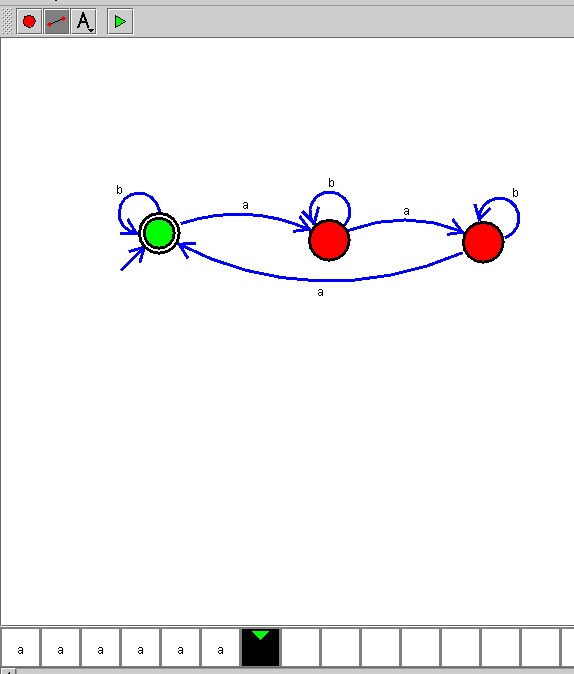
**AIM:**Design DFA using simulator to accept even number of c’s over the set {a,b,c}



**EXP :32**

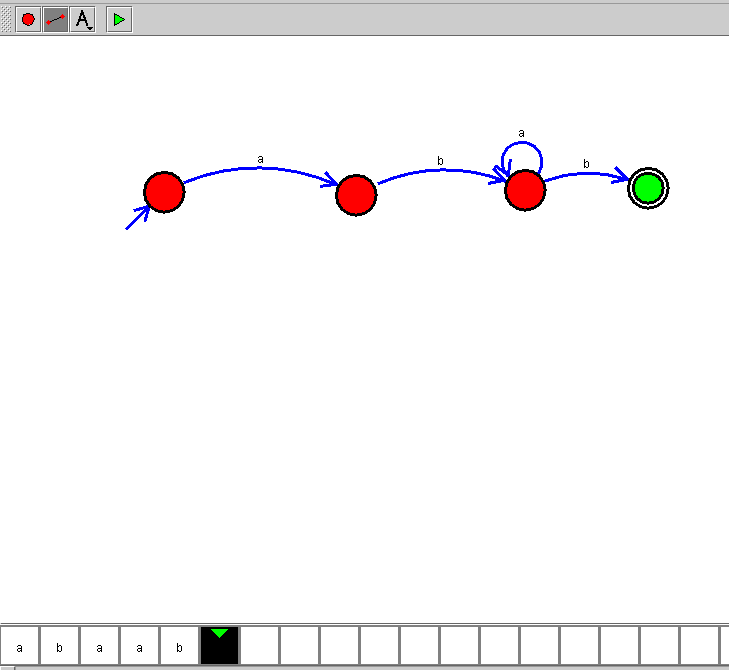
**AIM:**Design DFA using simulator to accept strings in which a’s always appear tripled over input {a,b}

W={ aaa, ababa}



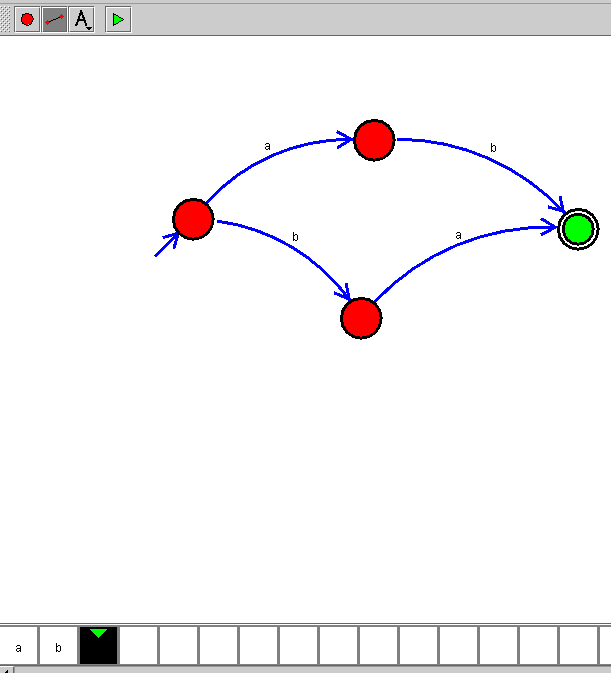
**EXP :33**

**AIM:**Design NFA using simulator to accept the string the start with a and end with b over set {a,b} and check W= abaab



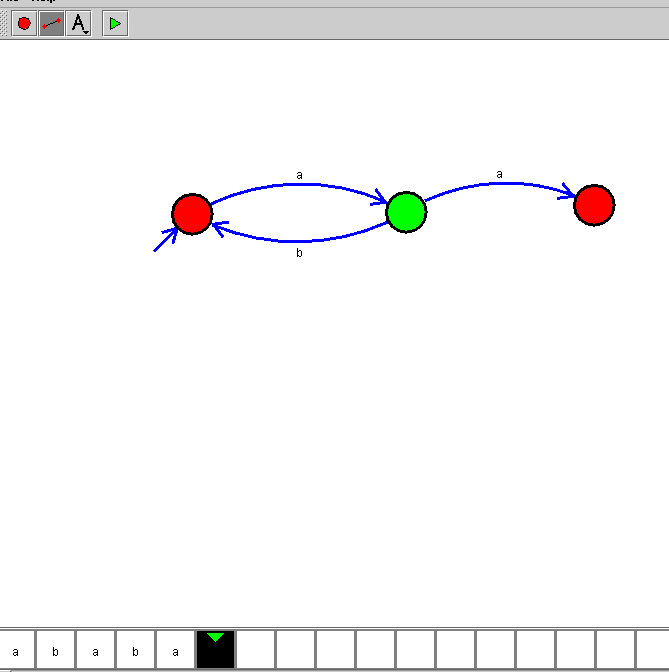
**EXP :34**

**AIM:**Design NFA using simulator to accept the string that start and end with different symbols over the input {a,b}



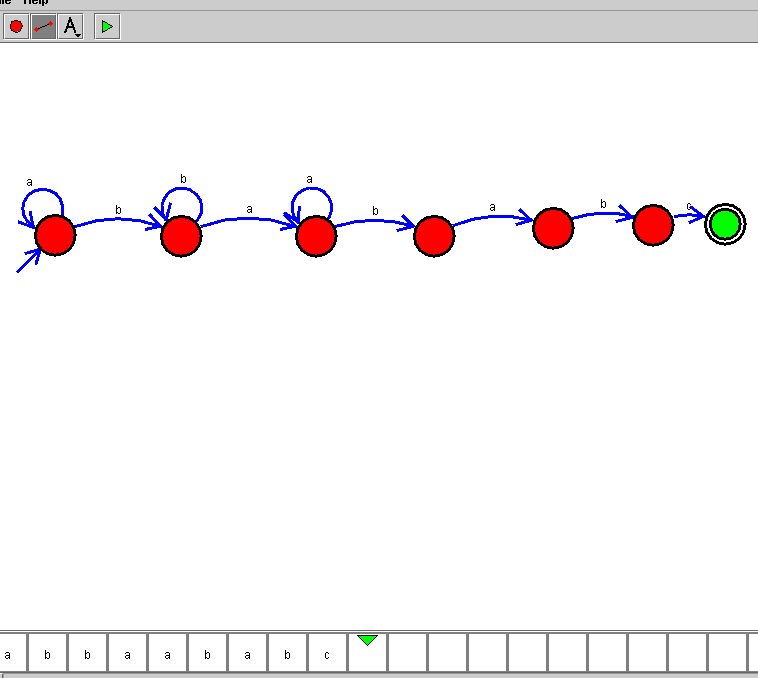
**EXP :35**

**AIM:**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.



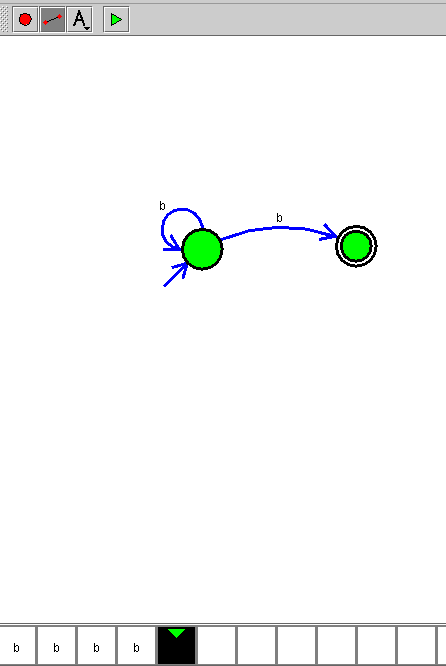
**EXP :36**

**AIM:**Design DFA using simulator to accept the string the end with abc over set {a,b,c) W= abbaababc



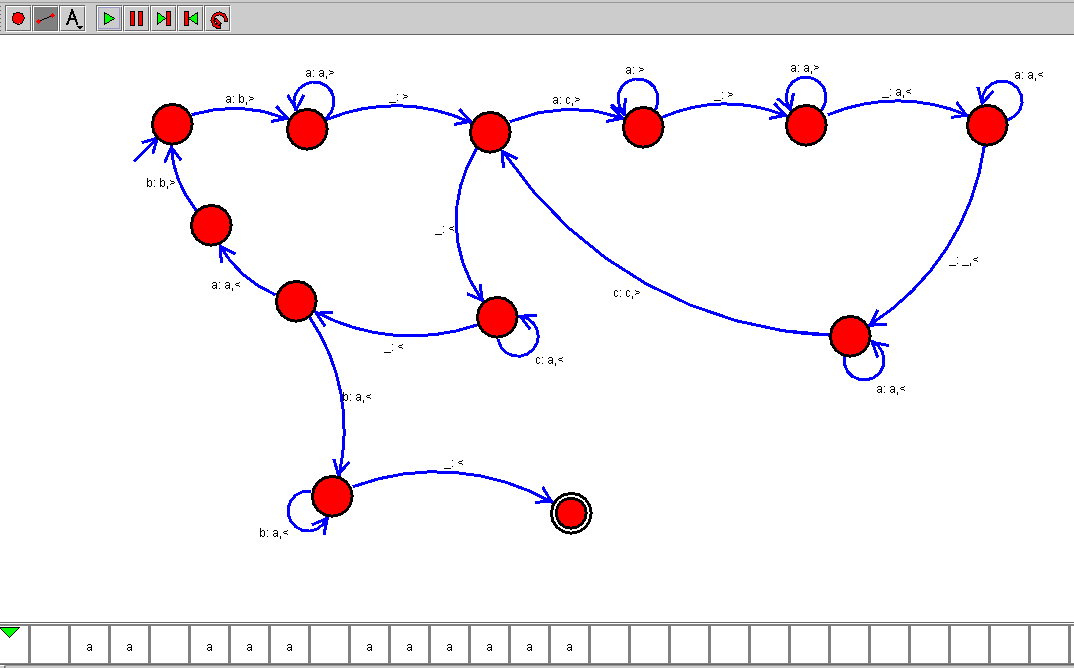
**EXP :37**

**AIM:**Design NFA to accept any number of b’s where input={a,b}



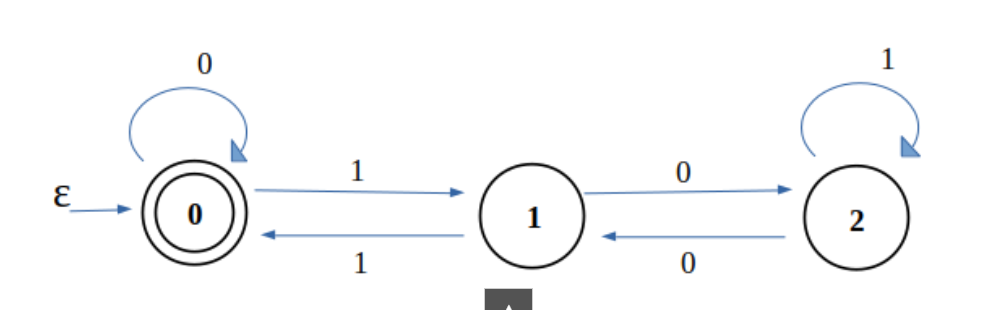
**EXP NO : 38**

AIM: construct a Turing Machine to perform the function Multiplication, using Subroutines.



**EXP NO : 39**

AIM: construct a FA Machine to check string of binary characters, check if it is multiple of 3

.