THEORY OF COMPUTATION LAB EXPERIMENTS:

EXPERIMENT:1

#include<stdio.h>

int main(){

int i;

char a[100];

printf("enter the string");

scanf("%s",&a);

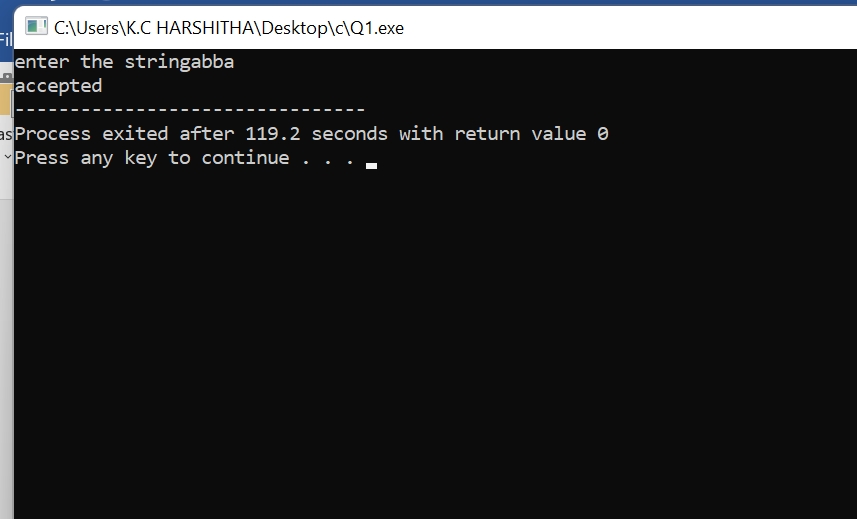
if (a[0]=='a'&&a[strlen(a)-1]=='a')

printf("accepted");

else printf("not accepted");

return 0;

}



EXPERIMENT:2

#include<stdio.h>

int main(){

int i,l;

char st[100];

printf("enter the string:");

scanf("%[^\n]c",&st);

l=strlen(st);

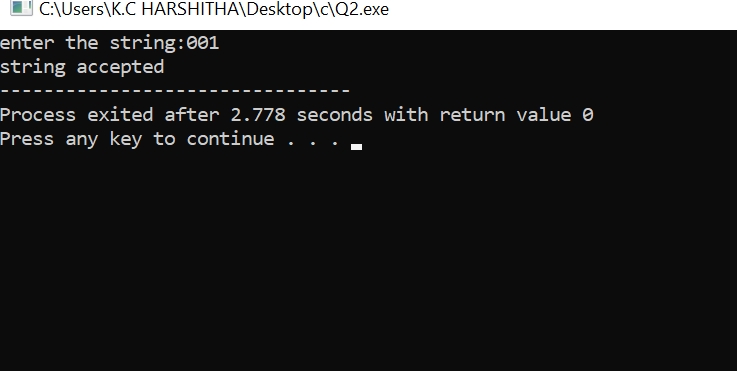
if(st[0]=='0'&&st[l-1]=='1')

printf("string accepted");

else

printf("string not accepted");

}



EXPERIMENT:3

#include<stdio.h>

int main(){

char s[100];

int i,flag;

int l;

printf("enter the 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");

if(flag==1)

{

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

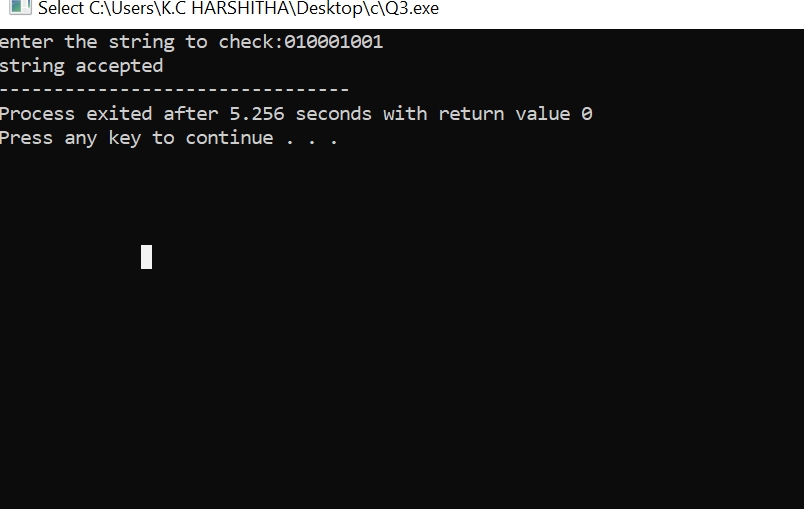
printf("string accepted");

else

printf ("string not accepted");

}

}



EXPERIMENT:4

#include<stdio.h>

#include<string.h>

int 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");

printf("string is accepted\n");

}

else

{

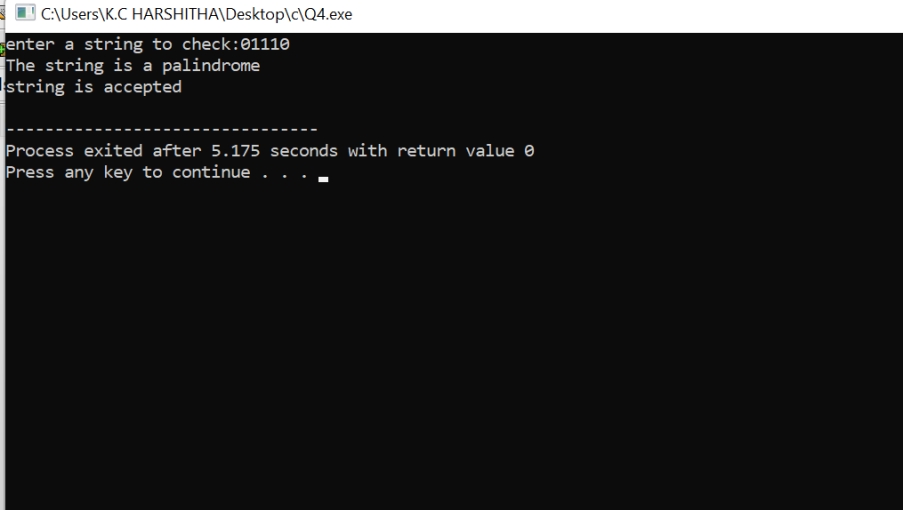
printf("The string is not a palindrome\n");

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

}

}

}



EXPERIMENT:5

#include<stdio.h>

#include<string.h>

int 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')

{

count1++;

i++;

}

while(s[i]=='1')

{

i++;

}

flag1=1;

count2=0;

while(i<l)

{

if(s[i]=='0')

{

count2++;

}

else

{

flag1=0;

}

i++;

}

if(flag1==1)

{

if(count1==count2)

{

printf("String Accepted\n");

}

else

{

printf("String Not Accepted\n");

}

}

else

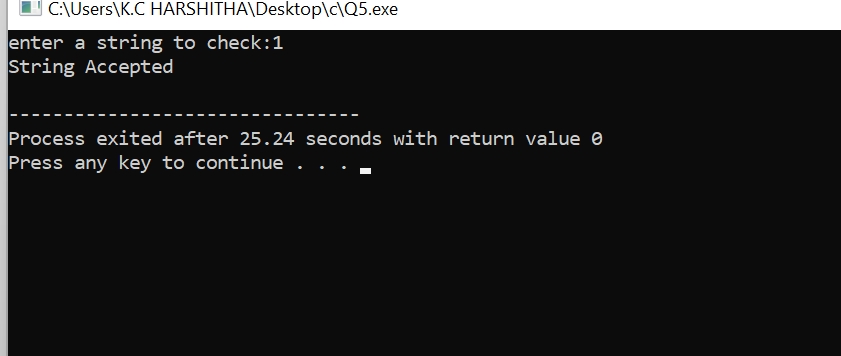
{

printf("String Not Accepted\n");

}

}

}



EXPERIMENT:6

#include<stdio.h>

#include<string.h>

int 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)

{

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

printf("String Not Accepted\n");

}

else

{

flag1=1;

for(i=0;i<(l/2);i++)

{

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

{

flag1=0;

}

}

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");

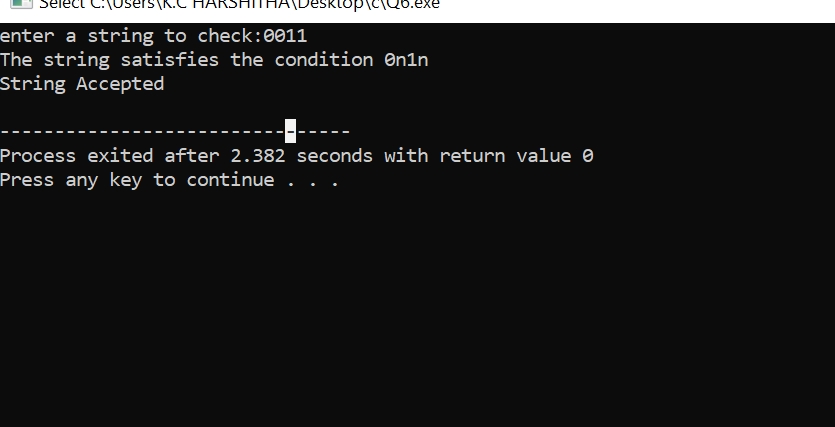
printf("String Not Accepted\n");

}

}

}

}



EXPERIMENT:7

#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;

}

}

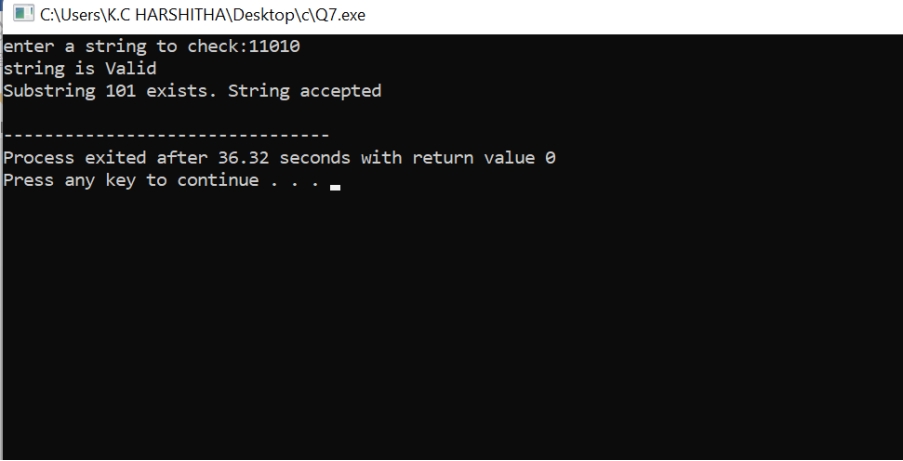
}

if(flag1==0)

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

}

}



EXPERIMENT:8

#include <stdio.h>

#include <stdbool.h>

// NFA transition function

bool transition(int state, char input) {

switch(state) {

case 0:

return input == 'b' ? true : false;

case 1:

return input == 'a' ? true : false;

default:

return false;

}

}

// NFA simulation function

bool simulateNFA(char \*input) {

int currentState = 0; // Start state

int i = 0;

while(input[i] != '\0') {

if (transition(currentState, input[i])) {

currentState++;

}

i++;

}

return currentState == 2; // Accept if reached the end state

}

int main() {

char input[100];

printf("Enter a string: ");

scanf("%s", input);

if (simulateNFA(input)) {

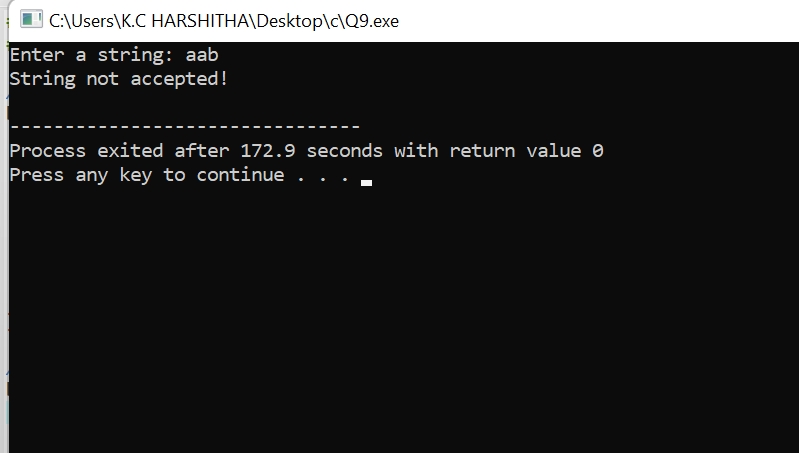
printf("String accepted!\n");

} else {

printf("String not accepted!\n");

}

return 0;}



EXPERIMENT:9

#include<stdio.h>

#include<string.h>

#define max 20

int main()

{

int table[4][2]={{1,3},{1,2},{1,2},{3,3}};

int final=2,i,l;

int present=0;

int next=0;

int invalid=0;

char input[max];

printf("Enter a string:");

scanf("%s",input);

l=strlen(input);

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

{

if(input[i]=='0')

next=table[present][0];

else if(input[i]=='1')

next=table[present][1];

else

invalid=l;

present=next;

}

if(invalid==l)

{

printf("Invalid input!");

}

else if(present==final)

printf("The given string is accepted for given condition.\n");

else

printf("The given is not accepted for given condition.\n");

}



EXPERIMENT:11

#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;

} } }

num\_states=3;

num\_symbols=2;

symbol[10]='e';

n=1;

trans\_table[0][0][0]=1;

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++)

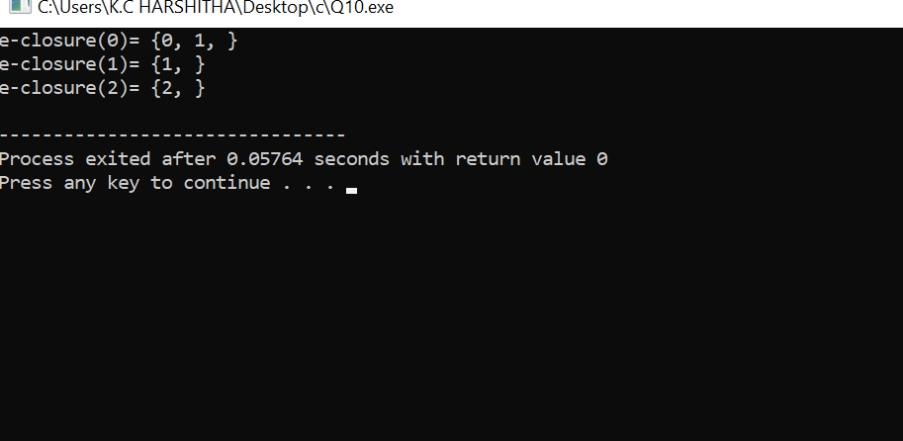
{

e\_closure[state][ptr]=y[j];

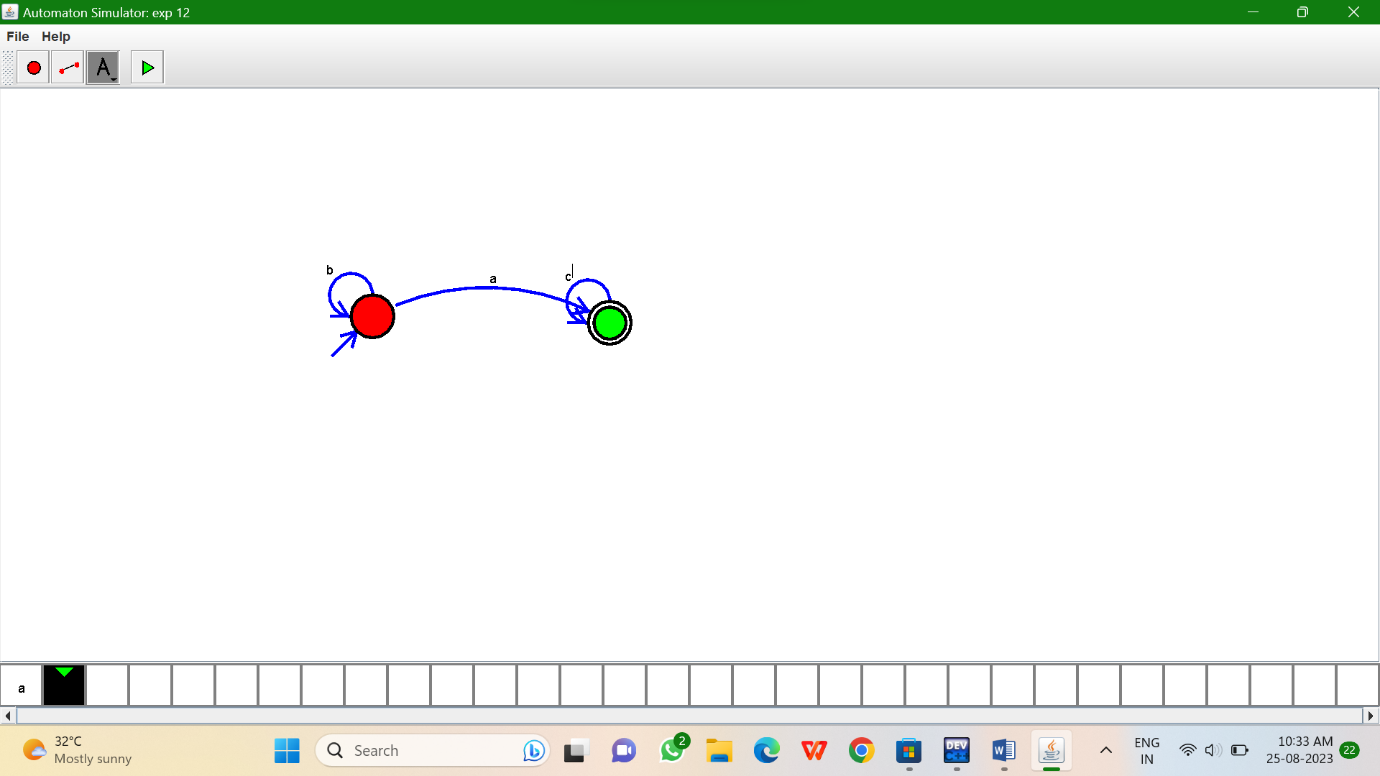
ptr++;

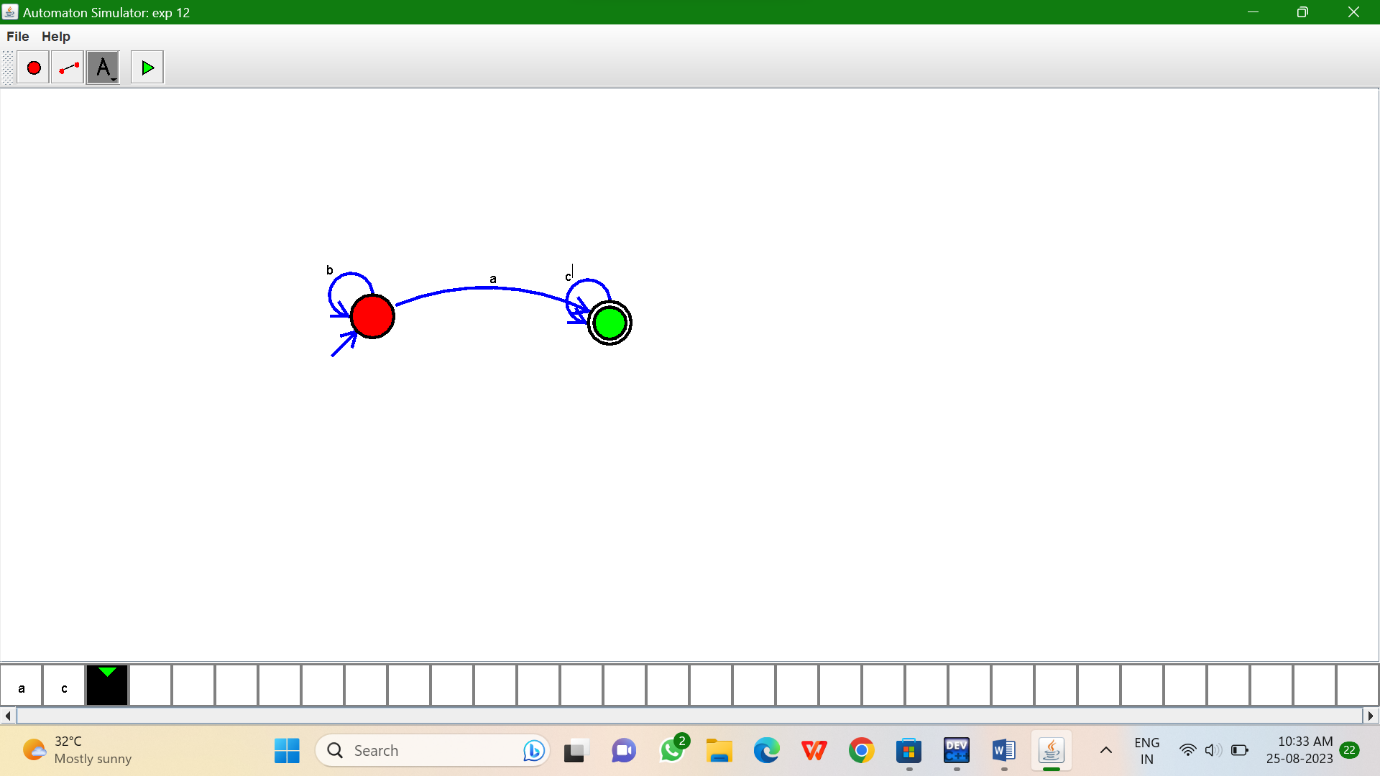
find\_e\_closure(y[j]);

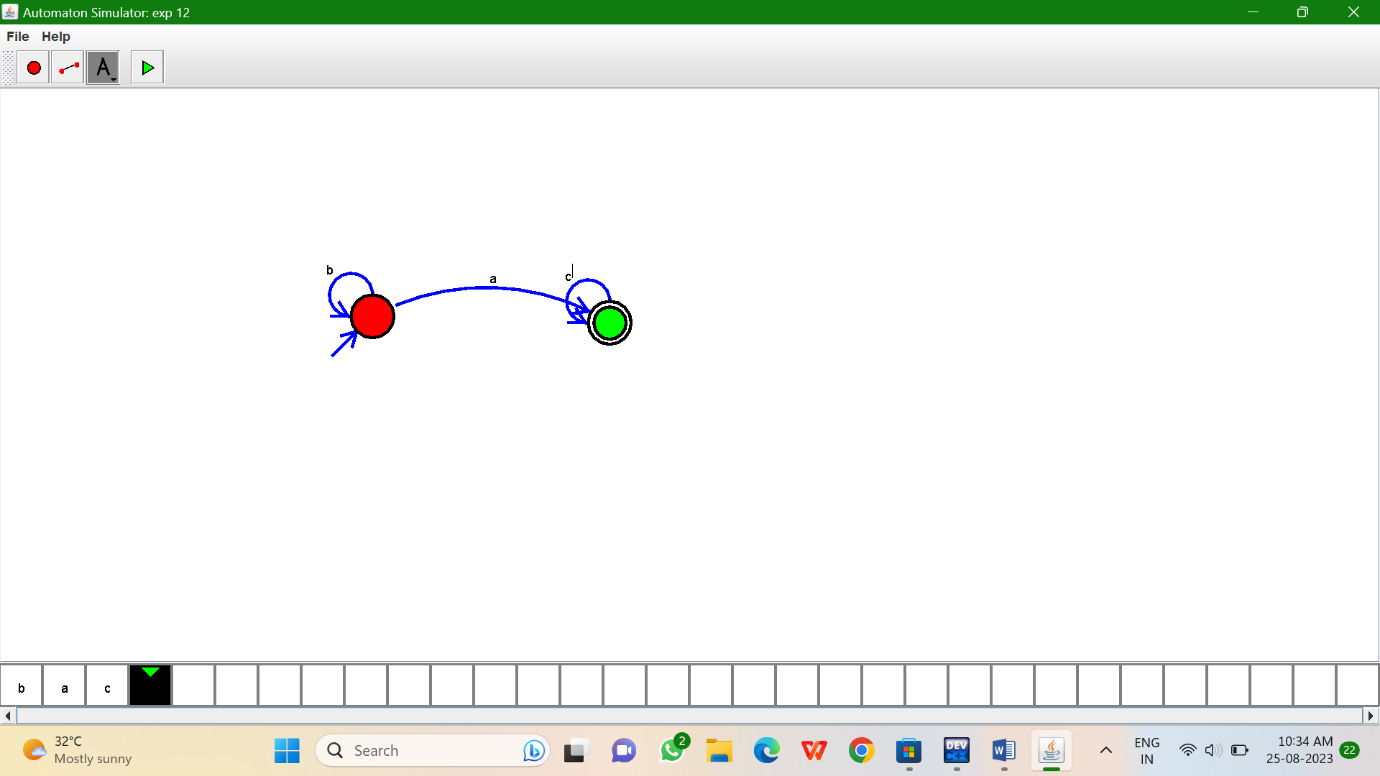
} }



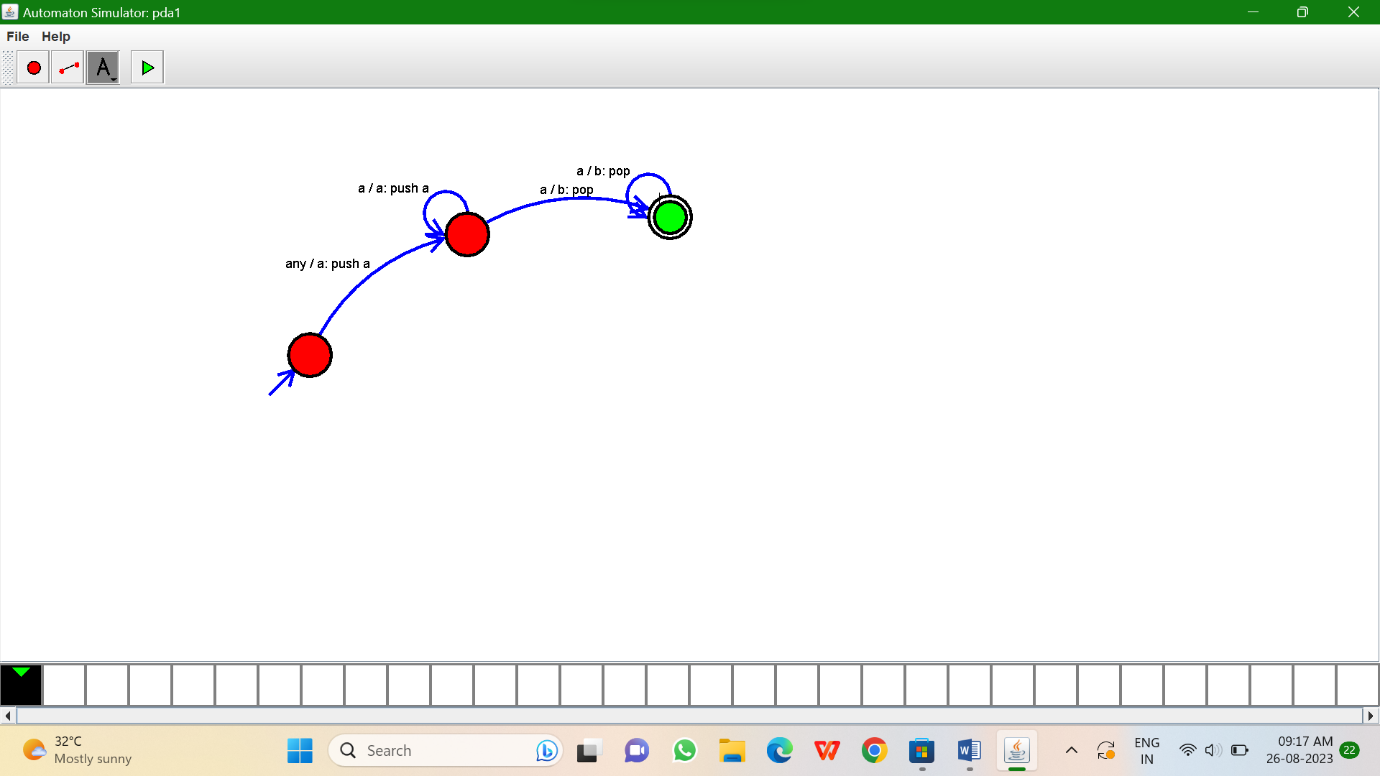
EXP:12



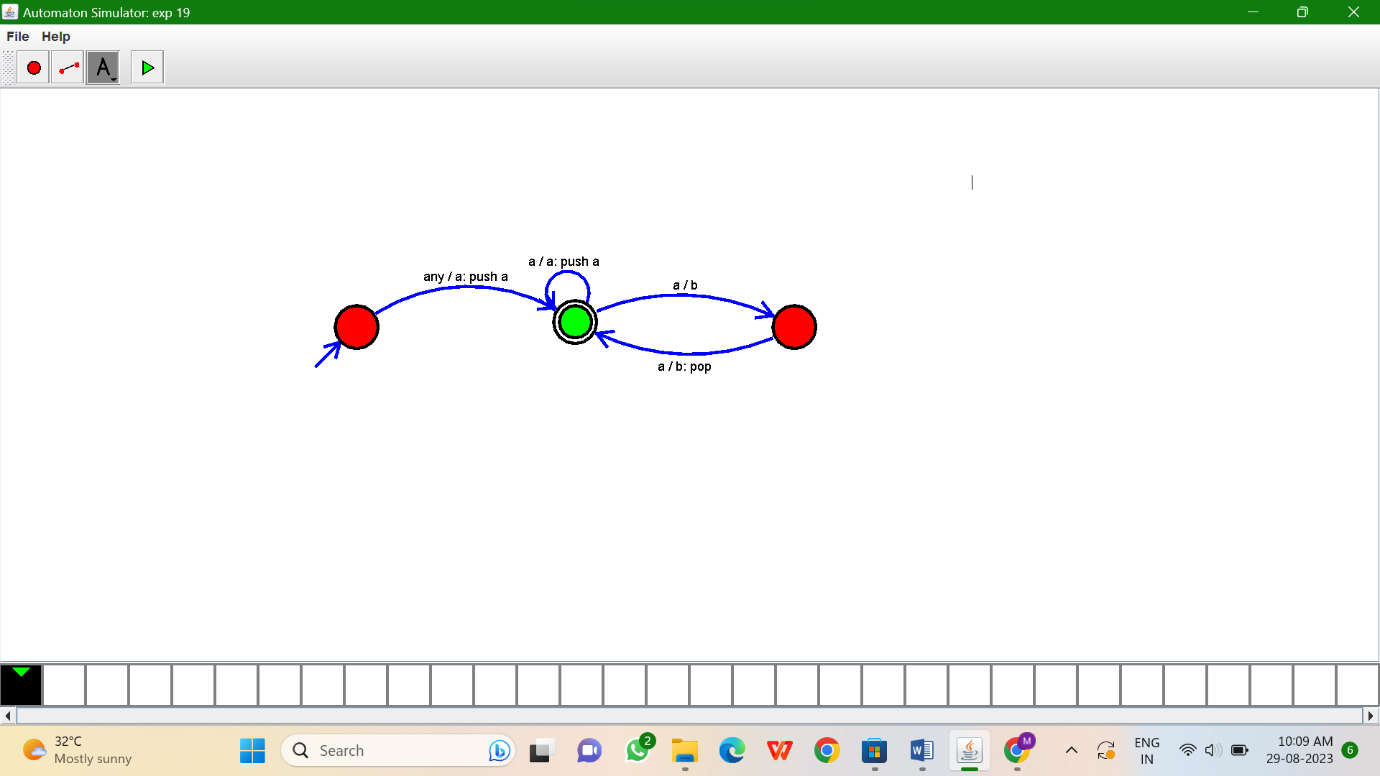




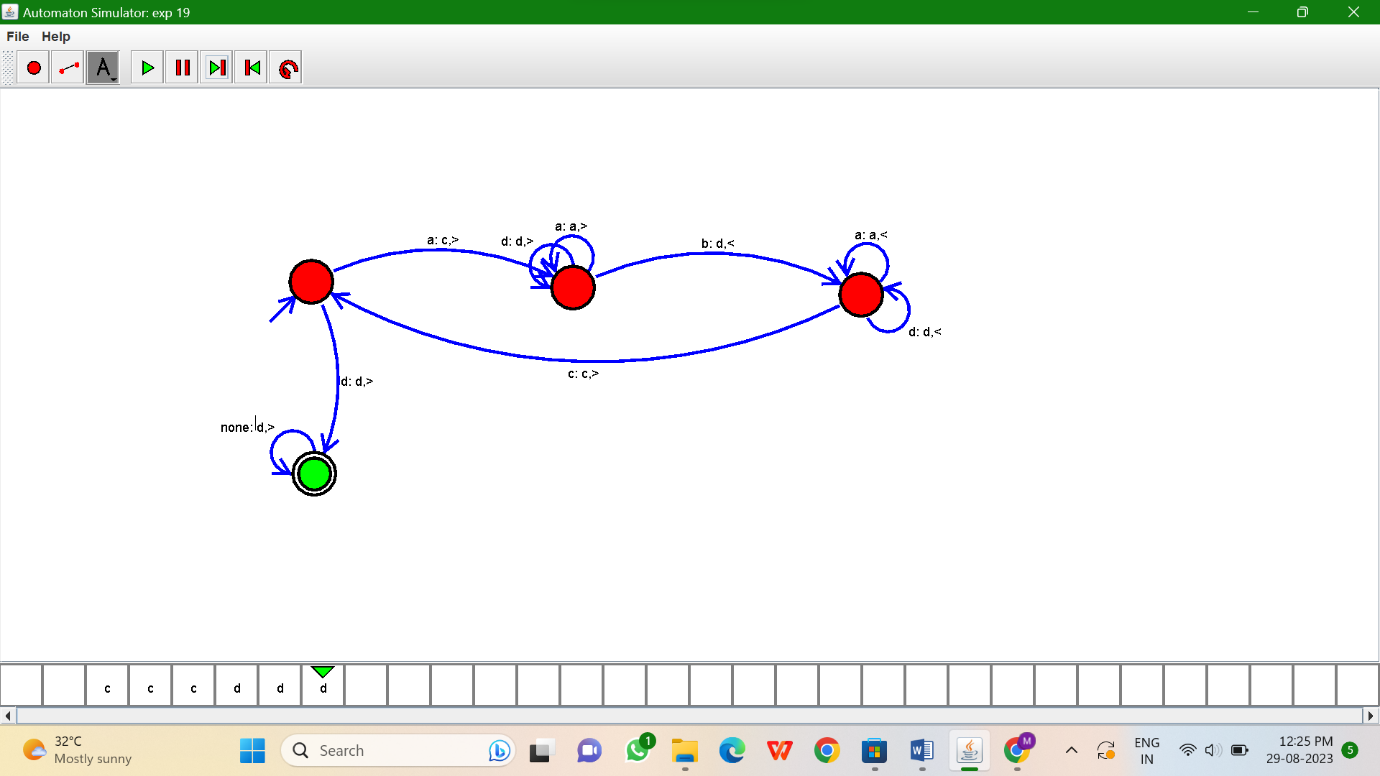
EXP:13



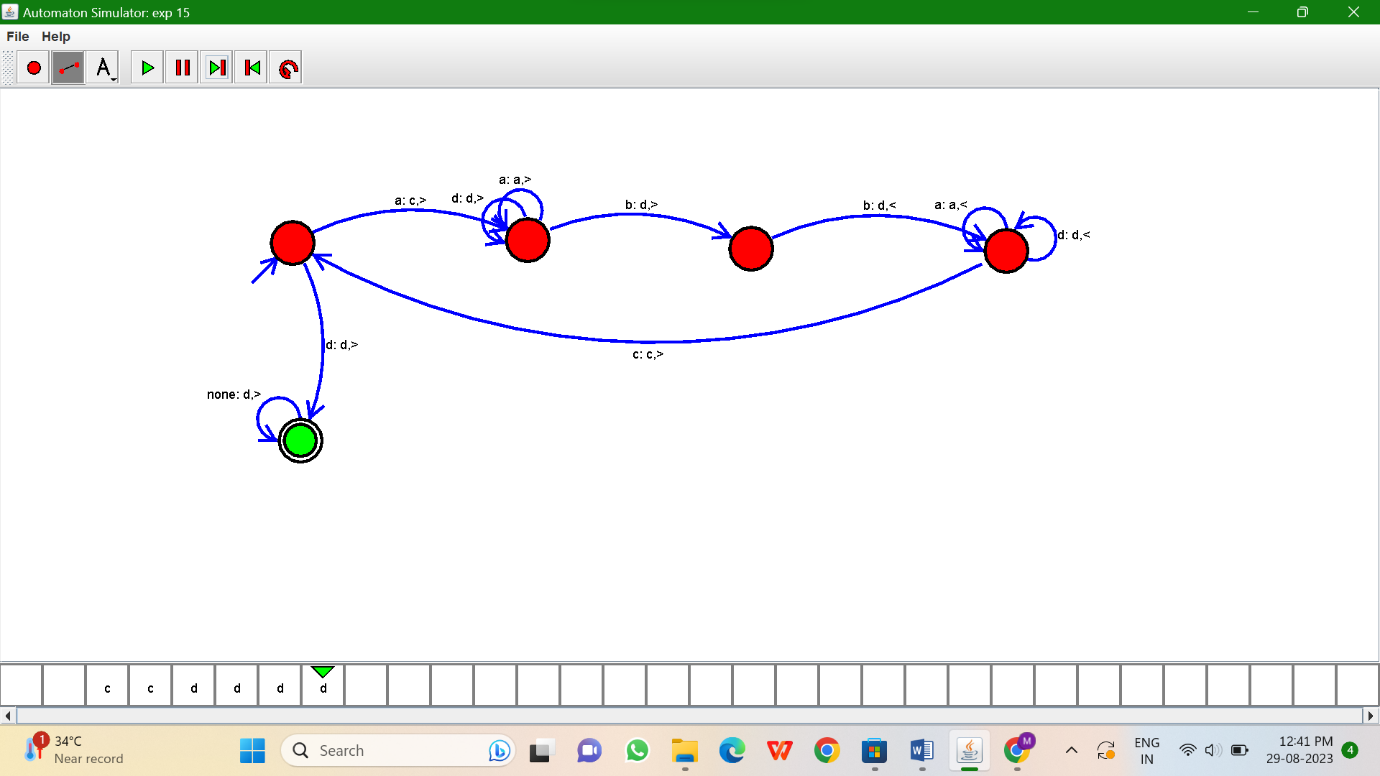
EXP:14



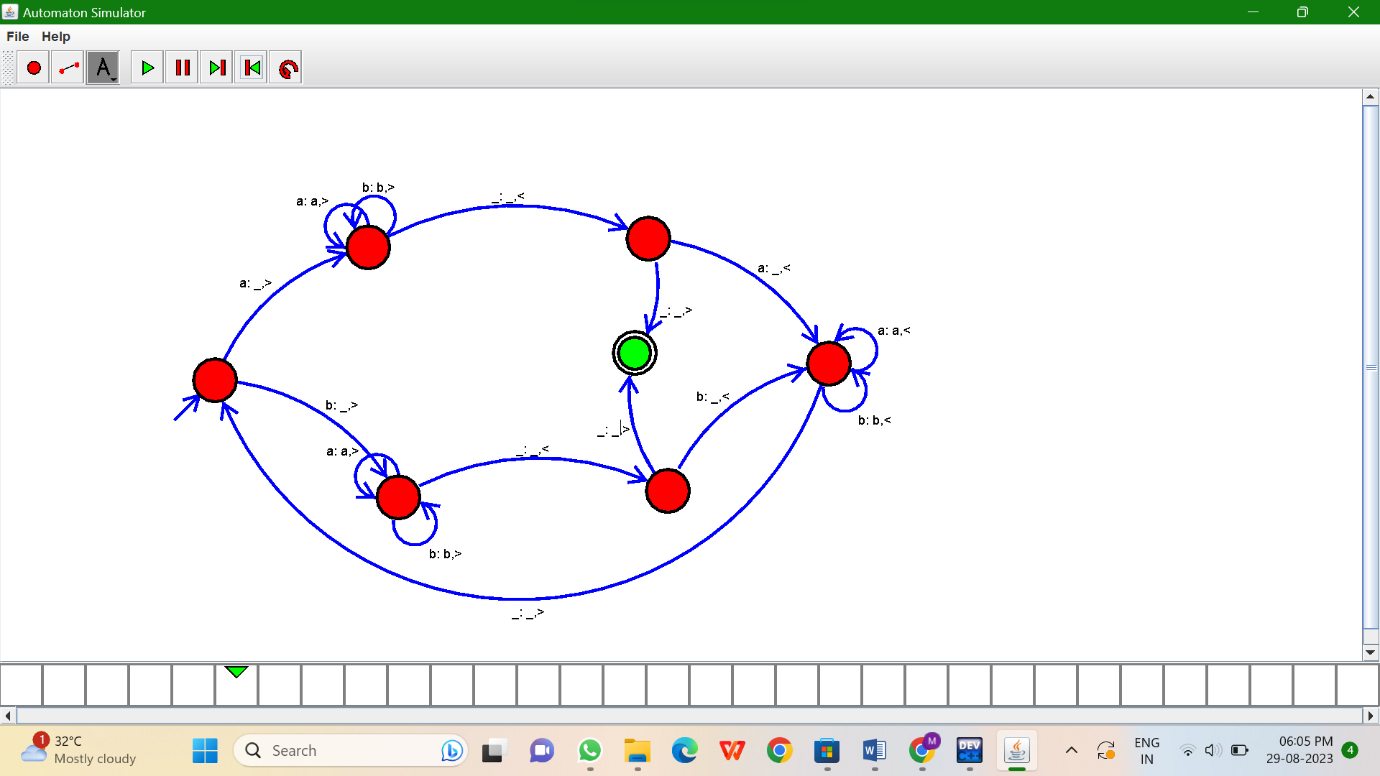
EXP:15



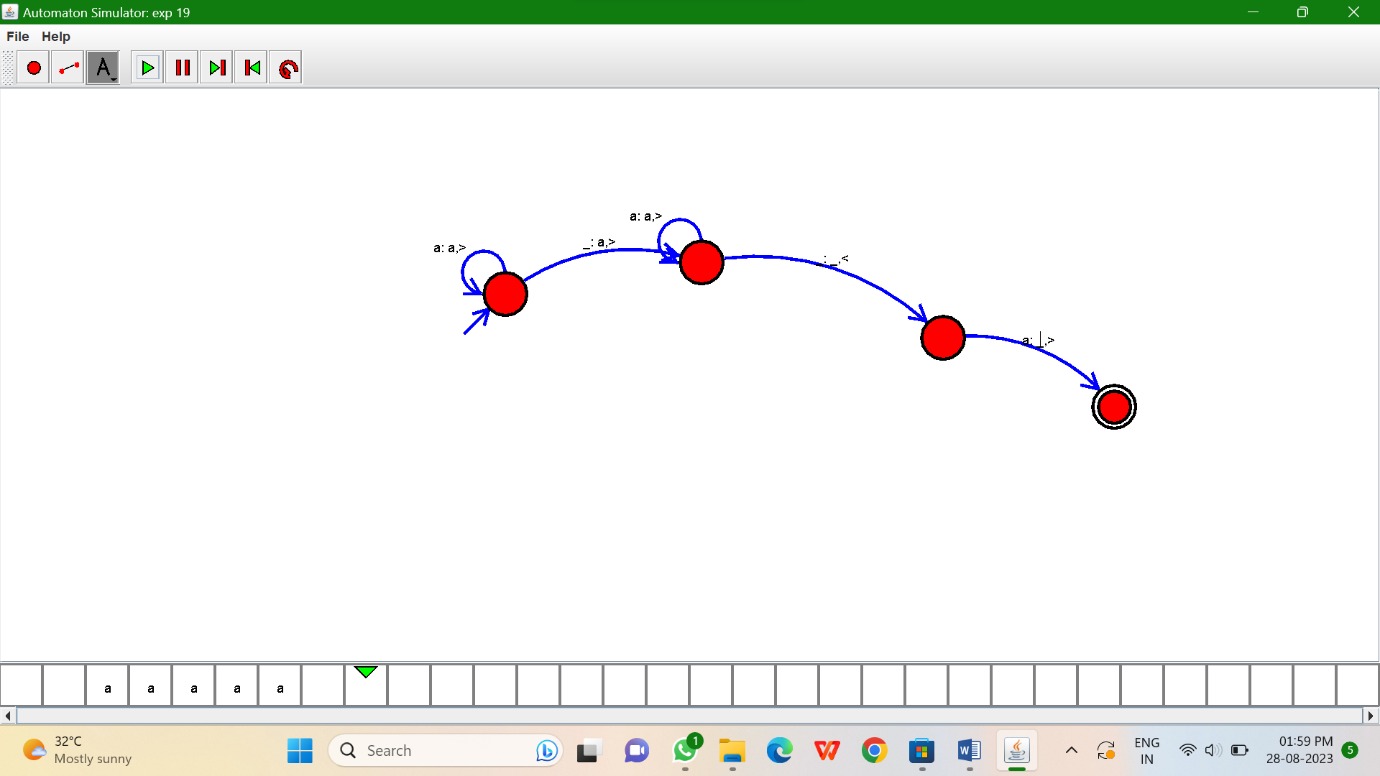
EXP:16



EXP:17



,EXP:19



EXP:20

