1.Write a c program to simulate a deterministic finite automata (DFA) for the given language representing strings that starts with a and end with a.

Aim:

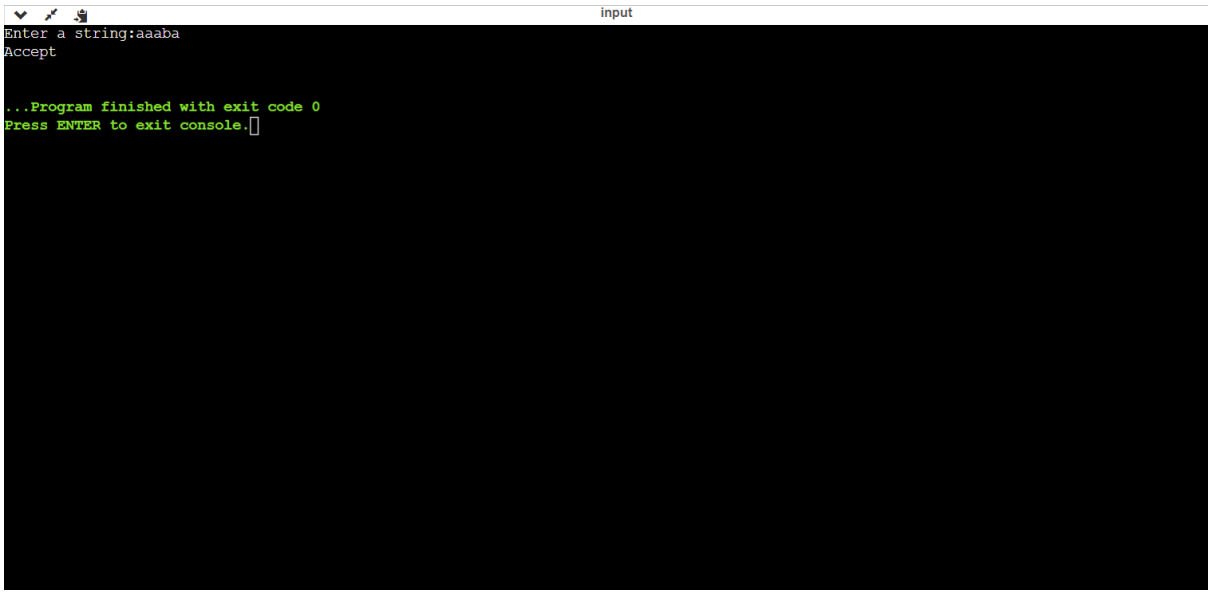
To write a c program to simulate a deterministic finite automata (DFA) for the given language representing strings that starts with a and end with a.

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("dontAccept\n");
else
printf("Accept\n");
}
```

Output:



2.Write a c program to simulate a deterministic finite automata (DFA) for the given language representing strings that starts with 0 and end with 1.

Aim:

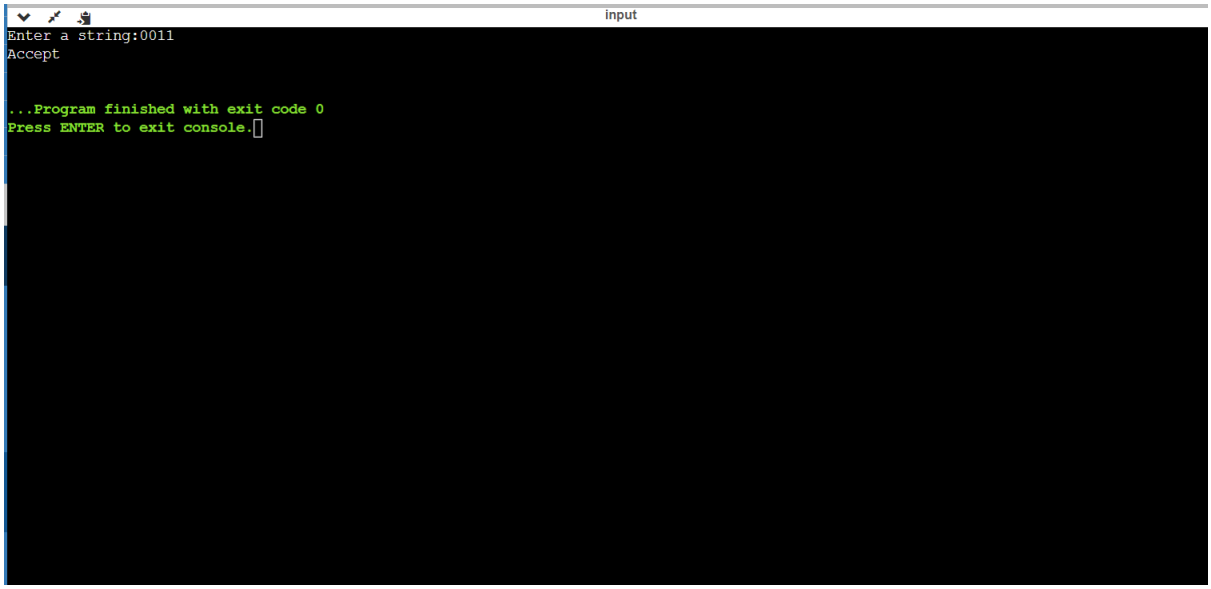
To write a c program to simulate a deterministic finite automata (DFA) for the given language representing strings that starts with 0 and end with 1.

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]=='0')
next_state=trans_table[present_state][0];
else if(input_string[i]=='1')
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("dont Accept\n");
}
```

Output:



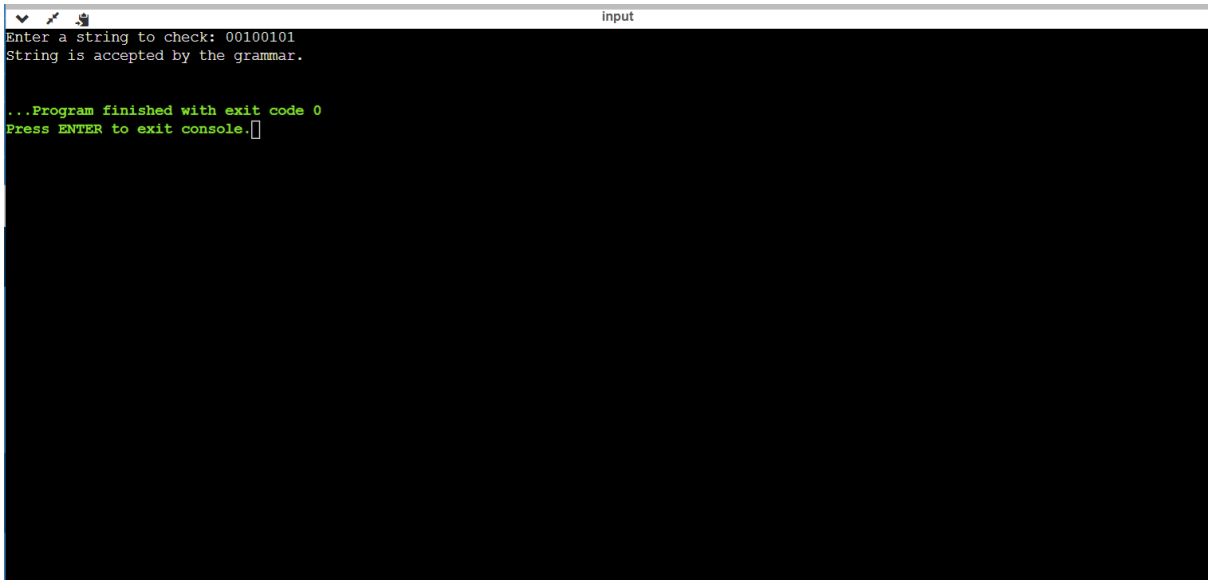
3.Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG)

S → 0A1 A → 0A | 1A | ε

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

int S(char []);
int A(char []);

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

if(S(str)) {
printf("String is accepted by the grammar.\n");
} else {
printf("String is not accepted by the grammar.\n");
}

return 0;
}

int S(char str[]) {
if(str[0] == '0' && str[strlen(str)-1] == '1') {
return A(&str[1]);
} else {
return 0;
}
}

int A(char str[]) {
if(strlen(str) == 0) {
return 1;
} else if(str[0] == '0') {
return A(&str[1]);
} else if(str[0] == '1') {
return A(&str[1]);
} else {
return 0;
}
```

```
}
```

OUTPUT:


input
Enter a string to check: 00100101
String is accepted by the grammar.
...Program finished with exit code 0
Press ENTER to exit console.

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

$S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

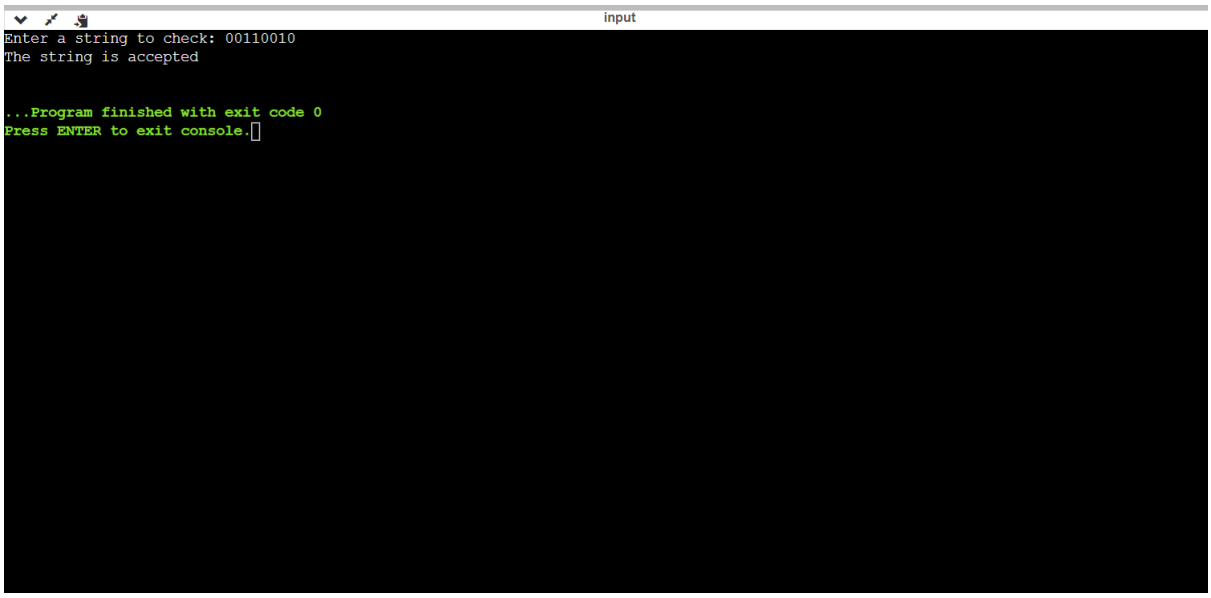
PROGRAM:

```
#include <stdio.h>

#include <string.h>

int main() {

   char input[100];

   int length, i;

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

   scanf("%s", input);


   length = strlen(input);


   // Check if the input string is valid

   for (i = 0; i < length; i++) {

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

         printf("Invalid string: contains characters other than 0 and 1\n");

         return 0;

      }

   }
```

```
    if (length == 0 || input[0] == '0' || input[0] == '1') {
        for (i = 1; i < length; i++) {
            if (input[i] != '0' && input[i] != '1') {
                printf("Invalid string: contains characters other than 0 and 1\n");
                return 0;
            }
        }
        printf("The string is accepted\n");
    } else if (input[0] == '0' && input[length - 1] == '0') {
        char substring[100];
        strncpy(substring, input + 1, length - 2);
        substring[length - 2] = '\0';
        if (strlen(substring) == 0 || (strlen(substring) == 1 && (substring[0] == '0' || substring[0] == '1'))) {
            printf("The string is accepted\n");
        } else {
            printf("The string is not accepted\n");
        }
    } else if (input[0] == '1' && input[length - 1] == '1') {
        char substring[100];
        strncpy(substring, input + 1, length - 2);
        substring[length - 2] = '\0';
        if (strlen(substring) == 0 || (strlen(substring) == 1 && (substring[0] == '0' || substring[0] == '1'))) {
            printf("The string is accepted\n");
        } else {
            printf("The string is not accepted\n");
        }
    } else {
        printf("The string is not accepted\n");
```

```
    }

    return 0;
}
```

OUTPUT:



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

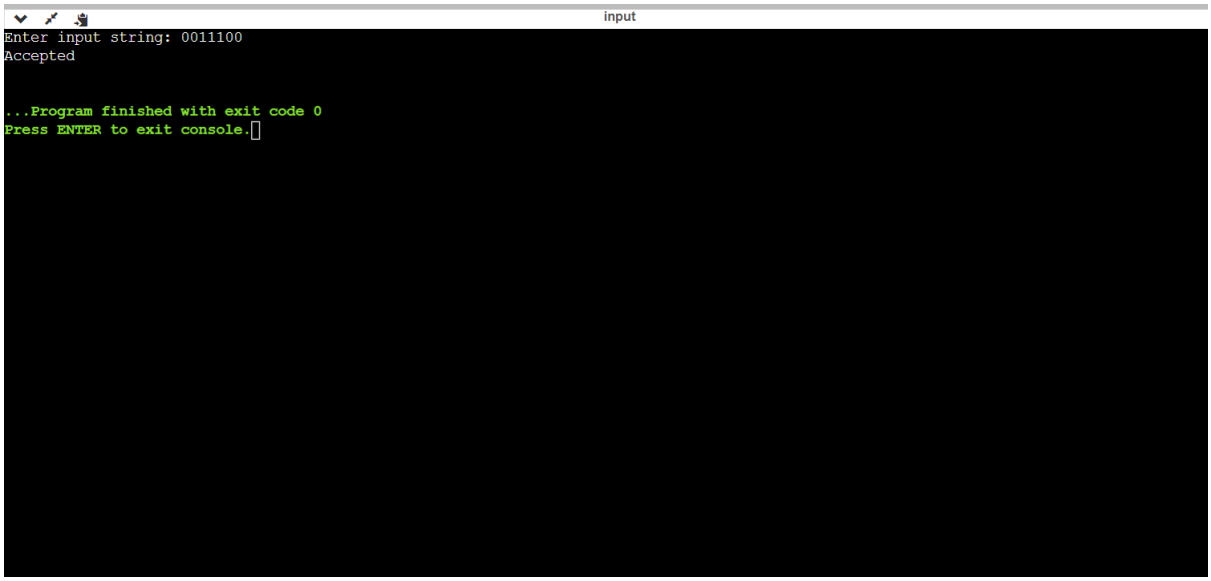
S → 0S0 | A A → 1A | ε

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#define MAX_LENGTH 100
int is_valid(char string[], int start, int end);
int S(char string[], int start, int end) {
   if (string[start] == '0' && string[end] == '0' && is_valid(string, start+1, end-1)) {
      return 1;
   } else {
      return 0;
   }
}
int A(char string[], int start, int end) {
   if (start > end) {
      return 1;
   } else if (string[start] == '1' && A(string, start+1, end)) {
      return 1;
   } else {
      return 0;
   }
}
int is_valid(char string[], int start, int end) {
   for (int i = start; i <= end; i++) {
     if (string[i] != '0' && string[i] != '1') {
         return 0;
      }
   }
   return 1;
```

```
}
int main() {
   char input_string[MAX_LENGTH];
   printf("Enter input string: ");
   scanf("%s", input_string);
   int string_length = strlen(input_string);
   if (S(input_string, 0, string_length - 1)) {
      printf("Accepted\n");
   } else {
      printf("Rejected\n");
   }
 return 0;
}
```

OUTPUT:



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

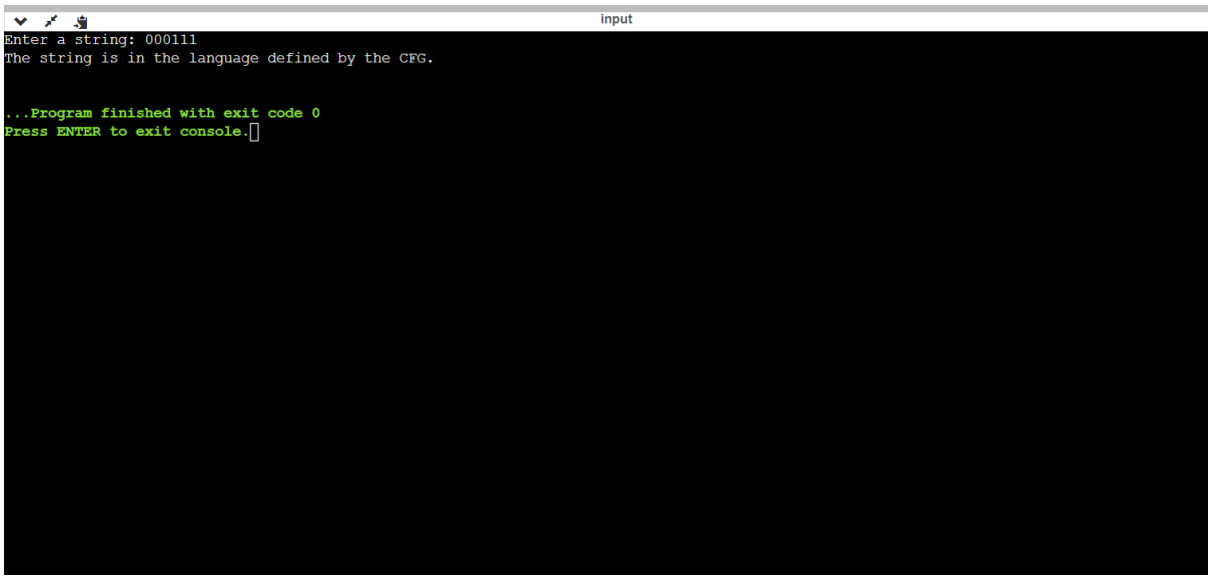
$S \rightarrow 0S1 \mid \varepsilon$

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define MAX_LENGTH 1000


int is_valid(char* string);


int main() {
   char string[MAX_LENGTH];
   printf("Enter a string: ");
   scanf("%s", string);
   if (is_valid(string)) {
      printf("The string is in the language defined by the CFG.\n");
   } else {
      printf("The string is not in the language defined by the CFG.\n");
   }
   return 0;
}


int is_valid(char* string) {
   if (strlen(string) == 0) {
      return 1; // the empty string is in the language
   }
   if (string[0] == '0' && string[strlen(string)-1] == '1') {
      char* new_string = (char*) malloc((strlen(string)-2) * sizeof(char));
      strncpy(new_string, string+1, strlen(string)-2);
```

```
        new_string[strlen(string)-2] = '\0';
        if (is_valid(new_string)) {
            return 1;
        }
    }
    return 0;
}
```

OUTPUT:



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

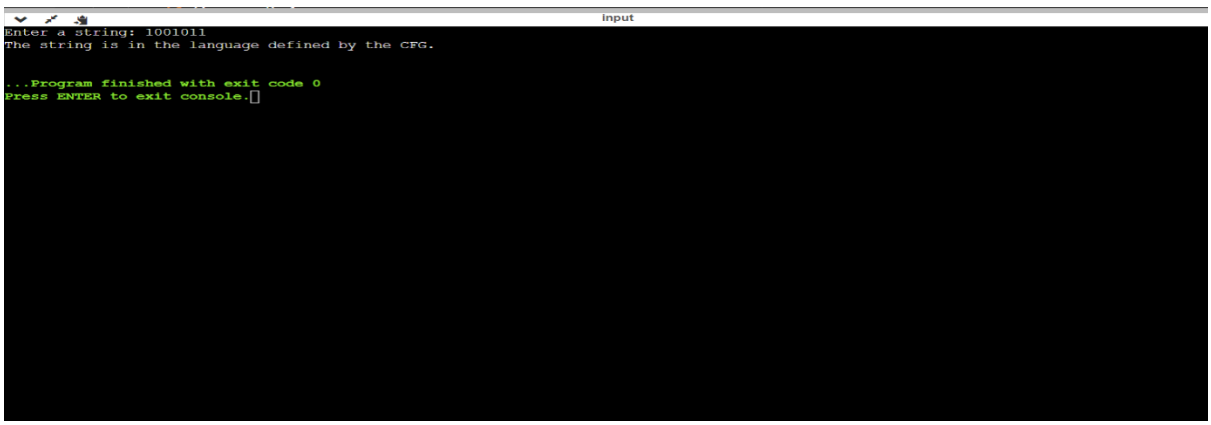
S → A101A, A → 0A | 1A | ε

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LENGTH 1000
int is_valid(char* string, int start, int end);
int main() {
   char string[MAX_LENGTH];
   printf("Enter a string: ");
   scanf("%s", string);
   if (is_valid(string, 0, strlen(string)-1)) {
      printf("The string is in the language defined by the CFG.\n");
   } else {
      printf("The string is not in the language defined by the CFG.\n");
   }
   return 0;
}
int is_valid(char* string, int start, int end) {
   if (start > end) {
      return 1; // the empty string is in the language
   }
   if (string[start] == '1' && string[end] == '1' && end == start + 2) {
      return 1; // S -> A101A
   }
   if (string[start] == '0' || string[start] == '1') {
      for (int i = start+1; i <= end; i++) {
         if (string[i] != '0' && string[i] != '1') {
            break;
```

```
        }
        if (i == end) {
            return 1; // A -> 0A | 1A | e
        }
    }
    for (int i = end-1; i >= start; i--) {
        if (string[i] != '0' && string[i] != '1') {
            break;
        }
        if (i == start) {
            return 1; // A -> 0A | 1A | e
        }
    }
    for (int i = start+1; i <= end-1; i++) {
        if (string[i] == '0' || string[i] == '1') {
            if (is_valid(string, start+1, i-1) && is_valid(string, i+1, end)) {
                return 1; // A -> 0A | 1A | e
            }
        }
    }
  }
  return 0;}
```

OUTPUT:



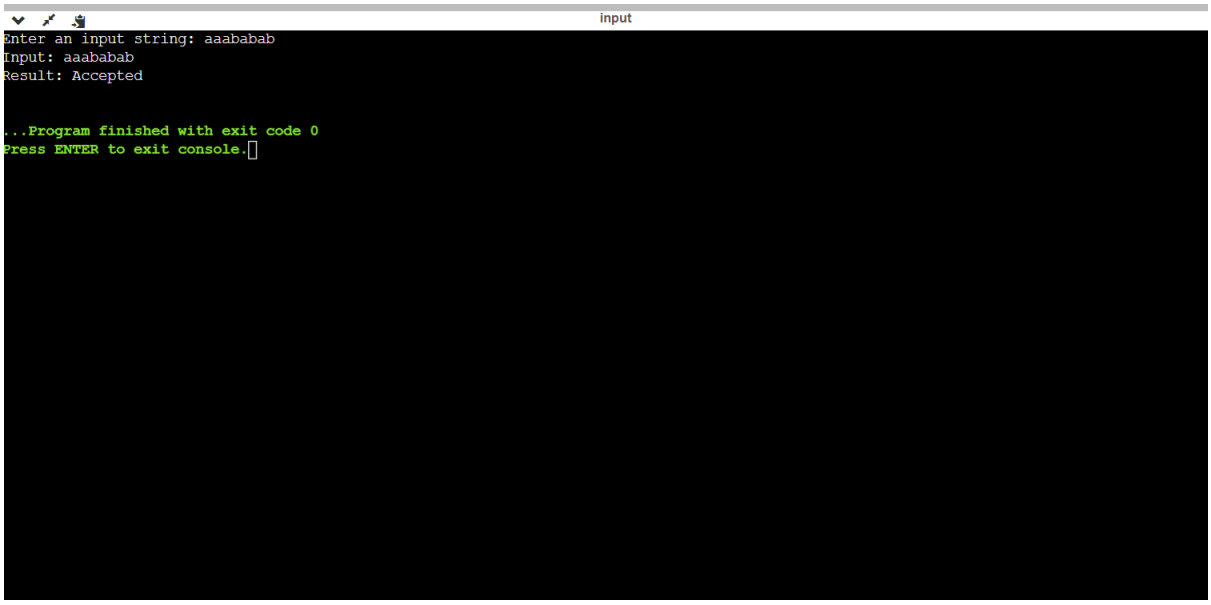
8.Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given language representing strings that start with b and end with a

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#define MAX_LENGTH 100

int nfa[MAX_LENGTH][2];

int start_state = 0;
int accept_state = 1;

int simulate_nfa(char* input_string) {
   int input_length = strlen(input_string);
   int current_state = start_state;
   for (int i = 0; i < input_length; i++) {
      int symbol = (input_string[i] == 'a') ? 1 : 0;
      int next_state = nfa[current_state][symbol];
      if (next_state == -1) {
         return 0;
      }
      current_state = next_state;
   }
   return current_state == accept_state;
}

int main() {
   nfa[0][0] = -1; // Reject b as the first symbol
   nfa[0][1] = 1;
   nfa[1][0] = 1;
```

```
    nfa[1][1] = 1;

    char input[MAX_LENGTH];
    printf("Enter an input string: ");
    scanf("%s", input);
    printf("Input: %s\nResult: %s\n", input, simulate_nfa(input) ? "Accepted" : "Rejected");

    return 0;
}
```

OUTPUT:

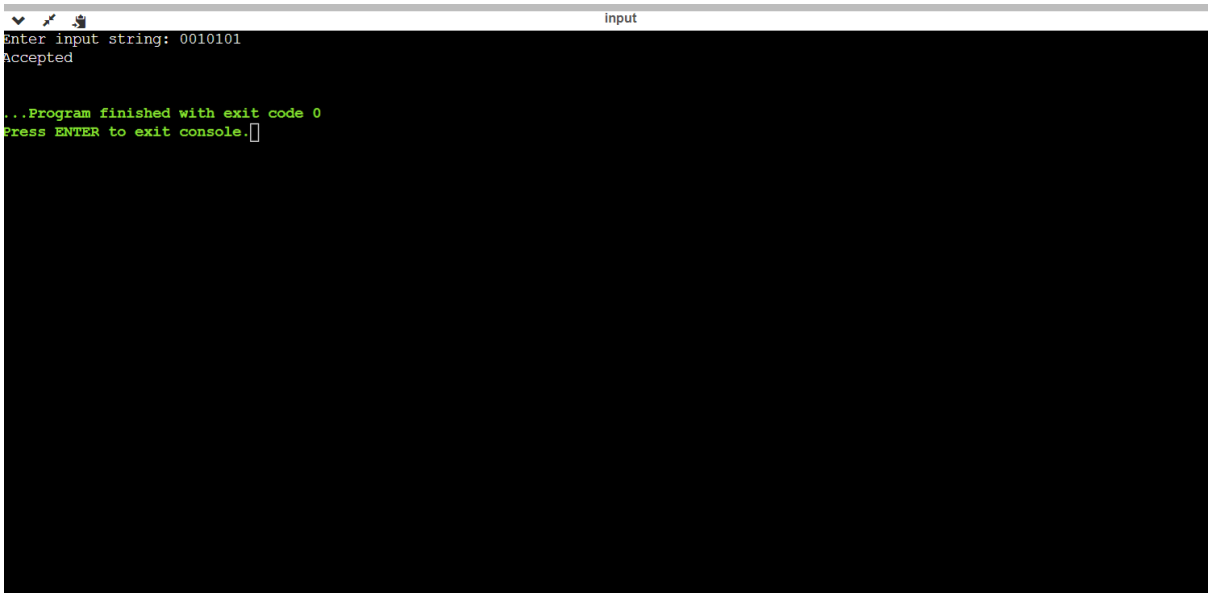


9.Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given languagerepresenting strings that start with o and end with 1

PROGRAM:

```
#include <stdio.h>

#include <string.h>

#define MAX_LENGTH 100int main() {

   char input_string[MAX_LENGTH];

   printf("Enter input string: ");

   scanf("%s", input_string);

   int string_length = strlen(input_string);

   if (string_length > 0 && input_string[0] == '0' && input_string[string_length - 1] == '1') {

      printf("Accepted\n");

   } else {

      printf("Rejected\n");

   }

   return 0;

}
```

OUTPUT:



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

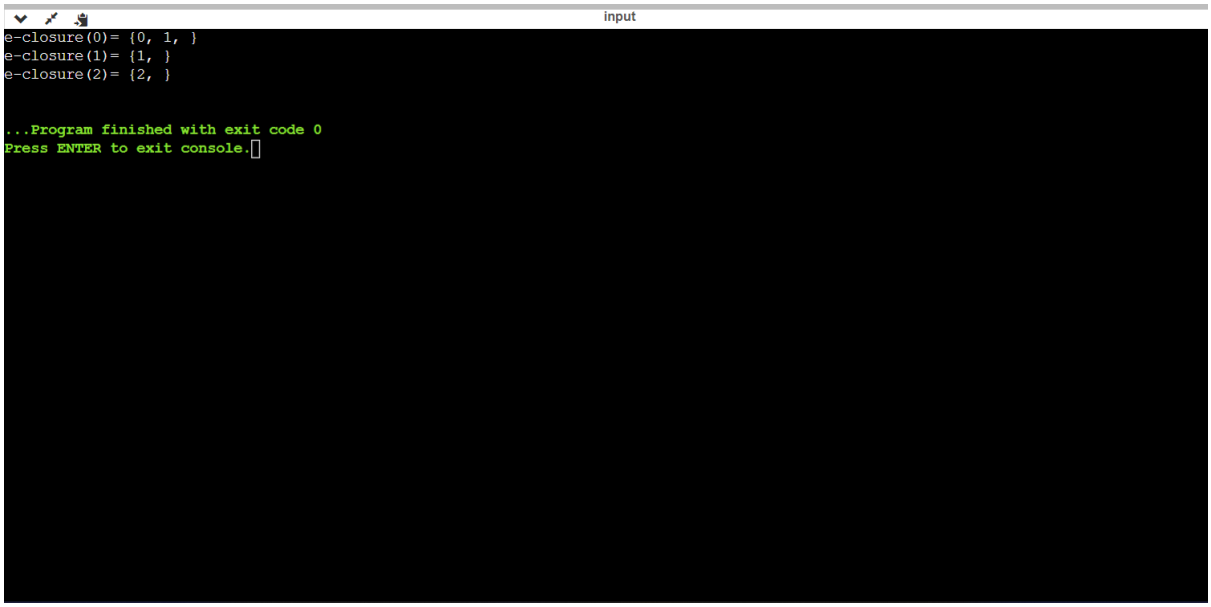
PROGRAM:

```
#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;
} } }
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;
} }
```

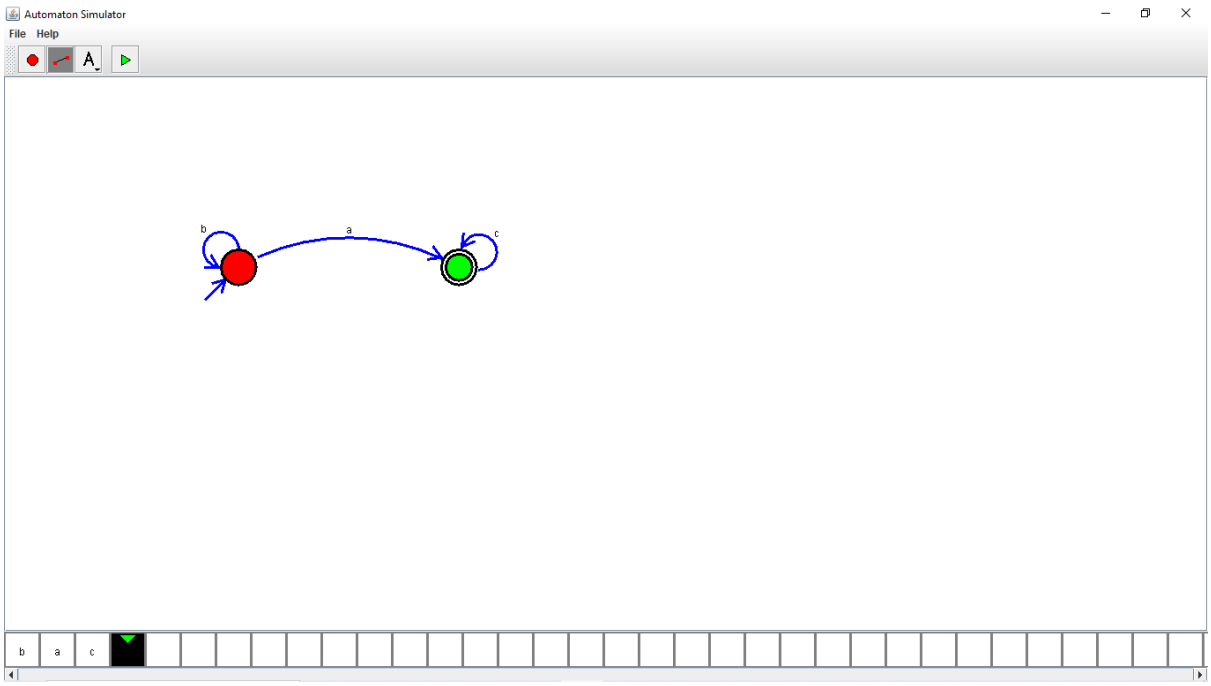
```
for(i=0;i<num_states;i++)
e_closure[i][0]=i;
for(i=0;i<num_states;i++)
{
if(trans_table[i][0][0]==-1)
continue;
else
{
state=i;
ptr=1;
find_e_closure(i);
} }
for(i=0;i<num_states;i++)
{
printf("e-closure(%d)= {",i);
for(j=0;j<num_states;j++)
{
if(e_closure[i][j]!=-1)
{
printf("%d, ",e_closure[i][j]);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

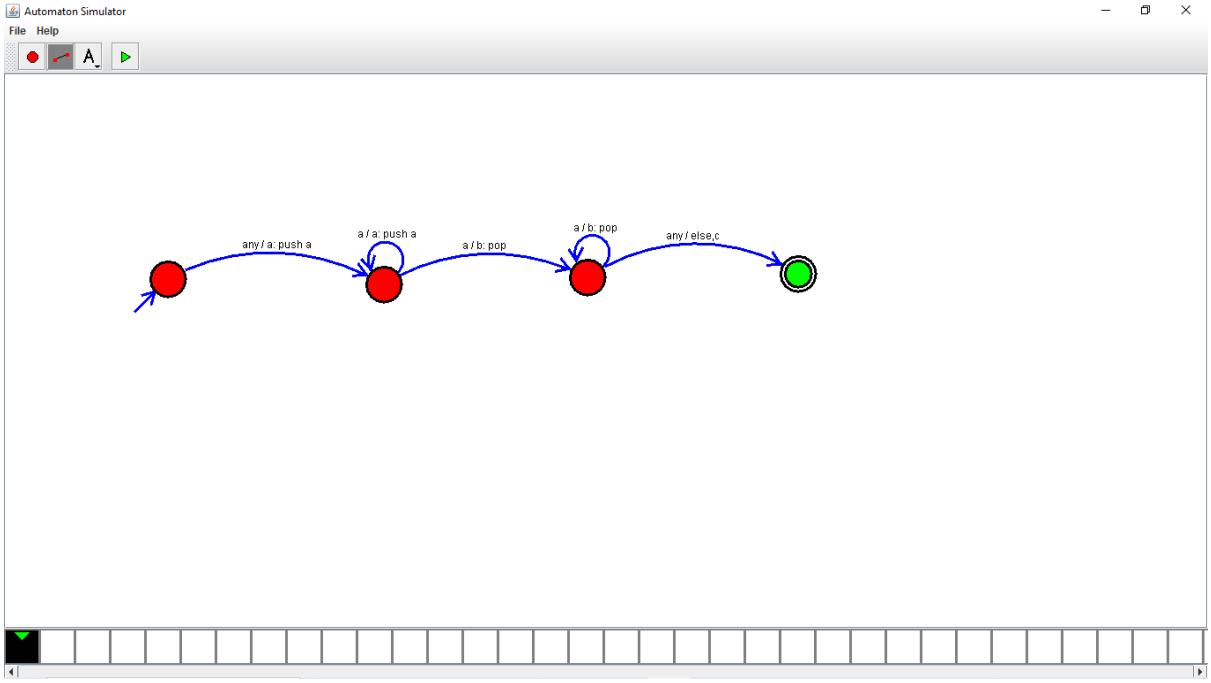
OUTPUT:



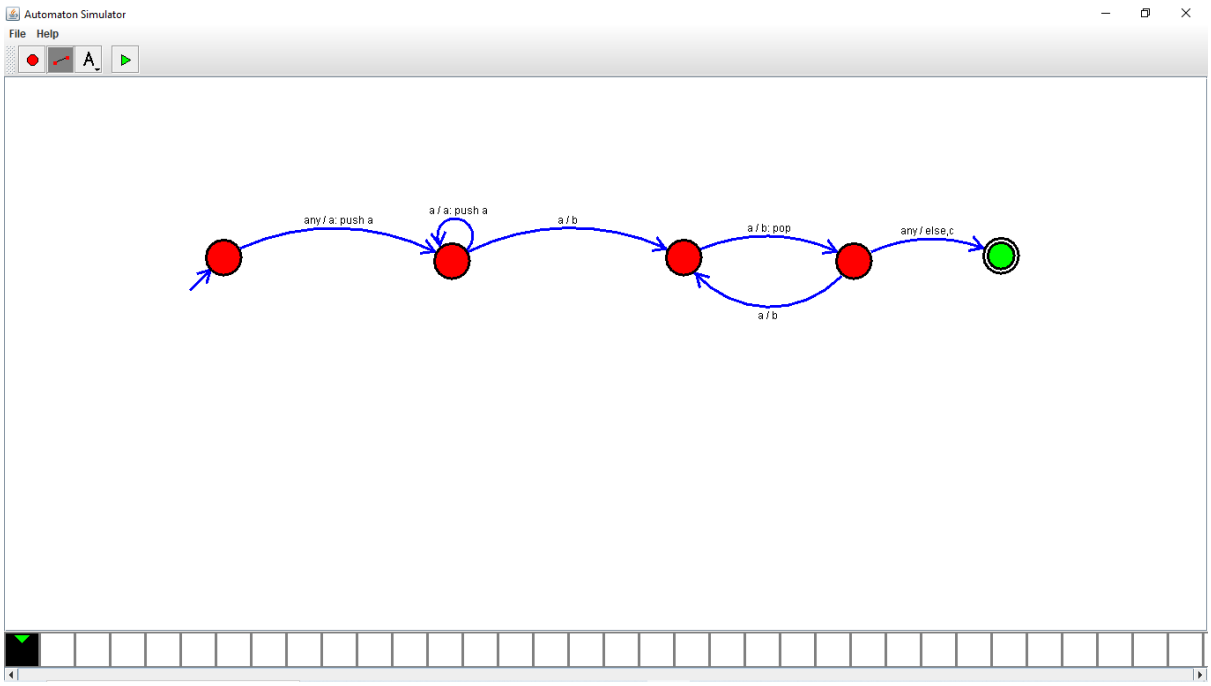
12. Design DFA using simulator to accept the input string “a” ,”ac”,and ”bac”.



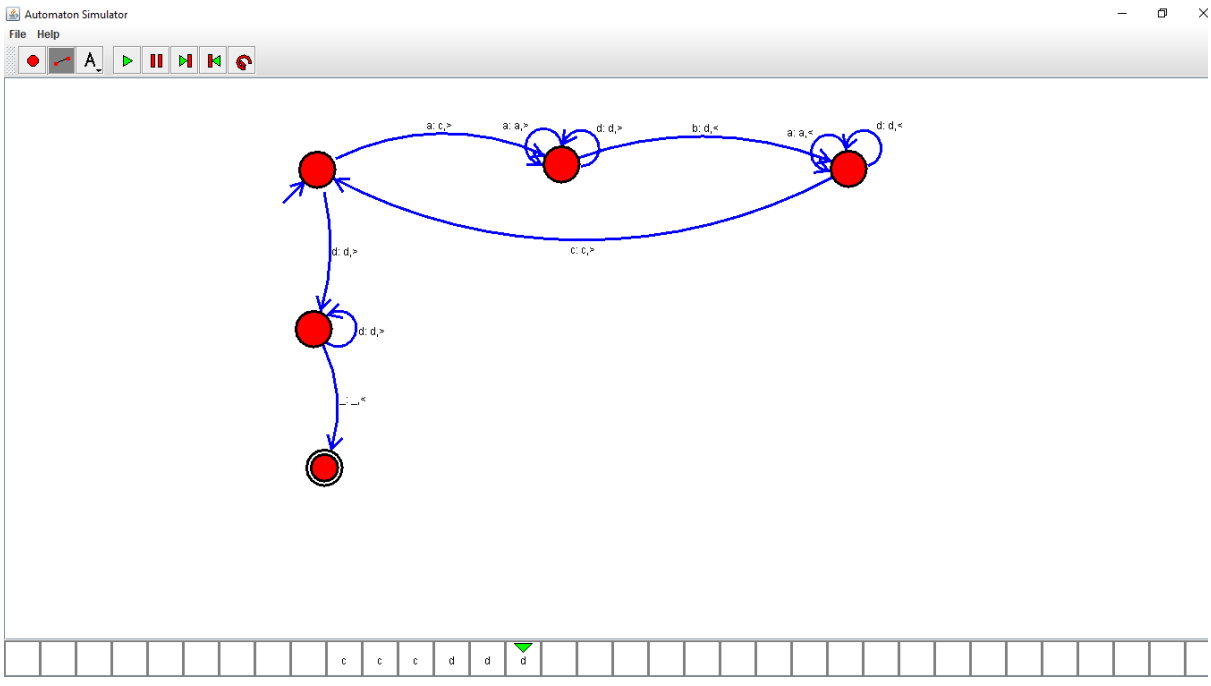
13. Design PDA using simulator to accept the input string aabb



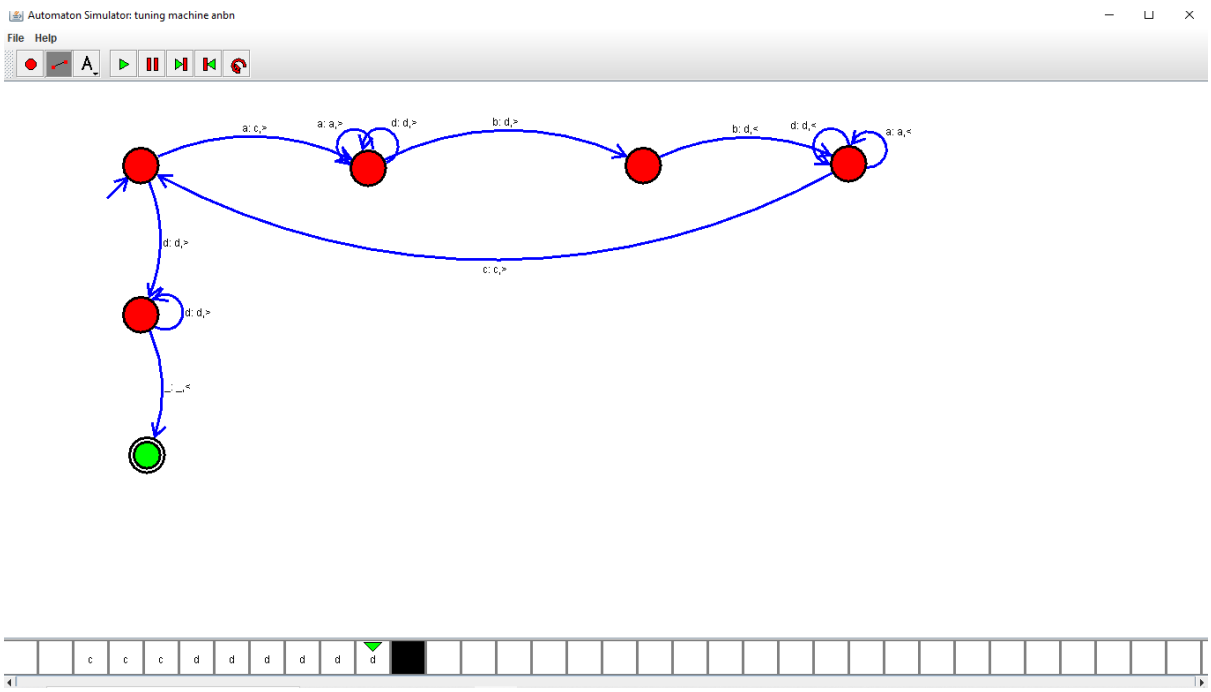
14.Design PDA using simulator to accept the input string $a^n b^{2n}$



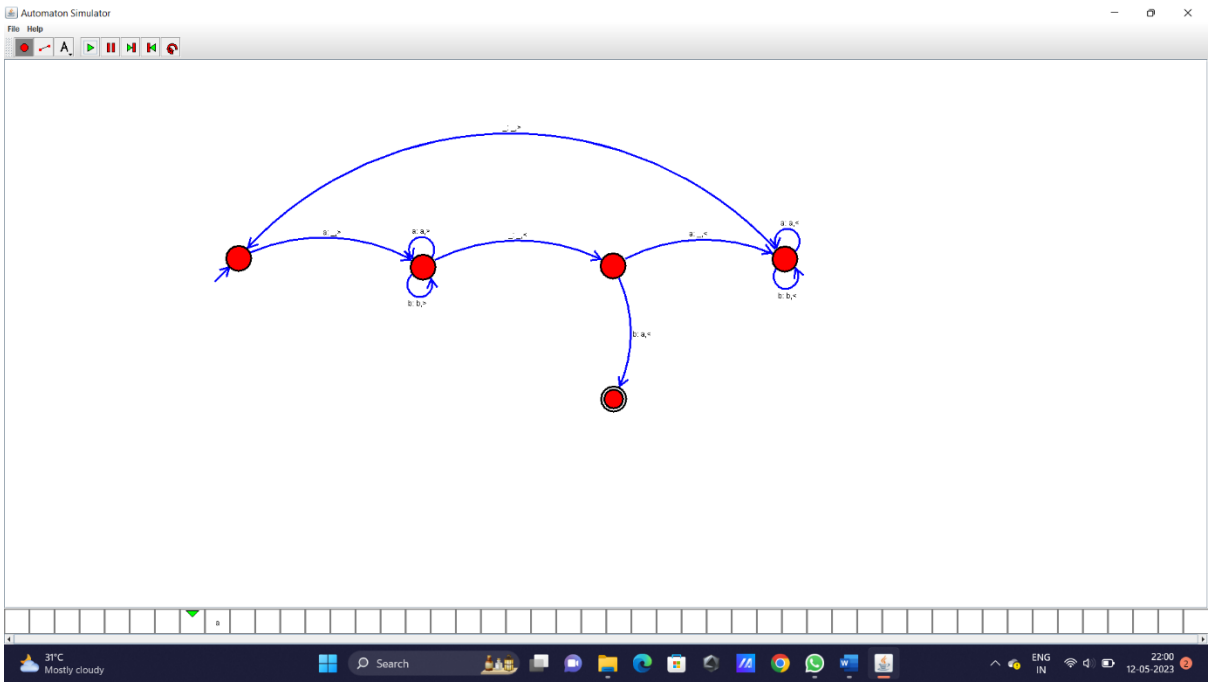
15.Design TM using simulator to accept the input string $a^n b^n$



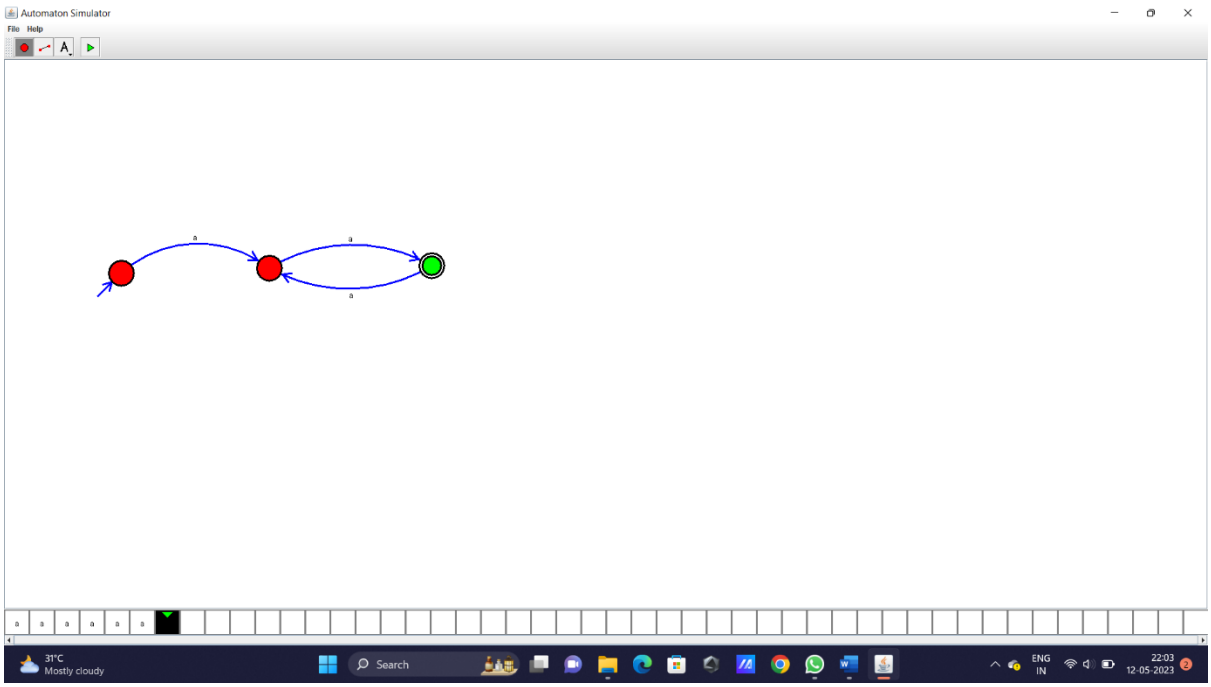
16.Design TM using simulator to accept the input string $a^n b^{2n}$



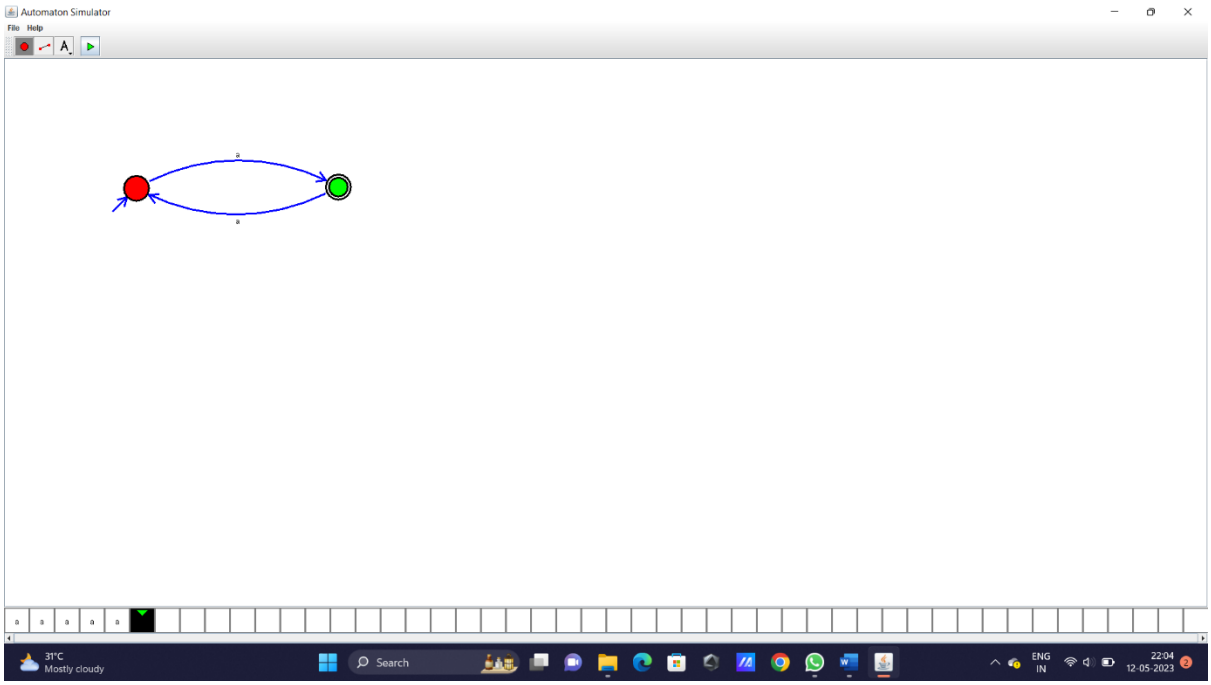
20.Design TM using simulator to perform subtraction of aaa-aa



21.Design DFA using simulator to accept even number of a's

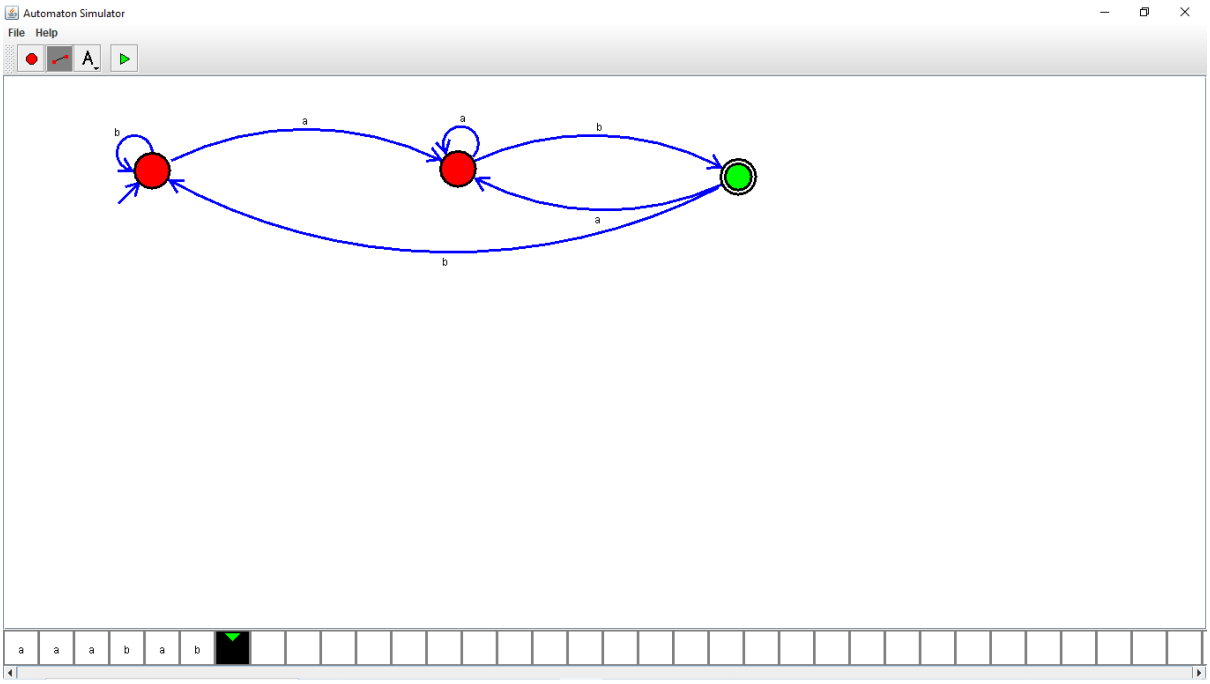


22.Design DFA using simulator to accept odd number of a's

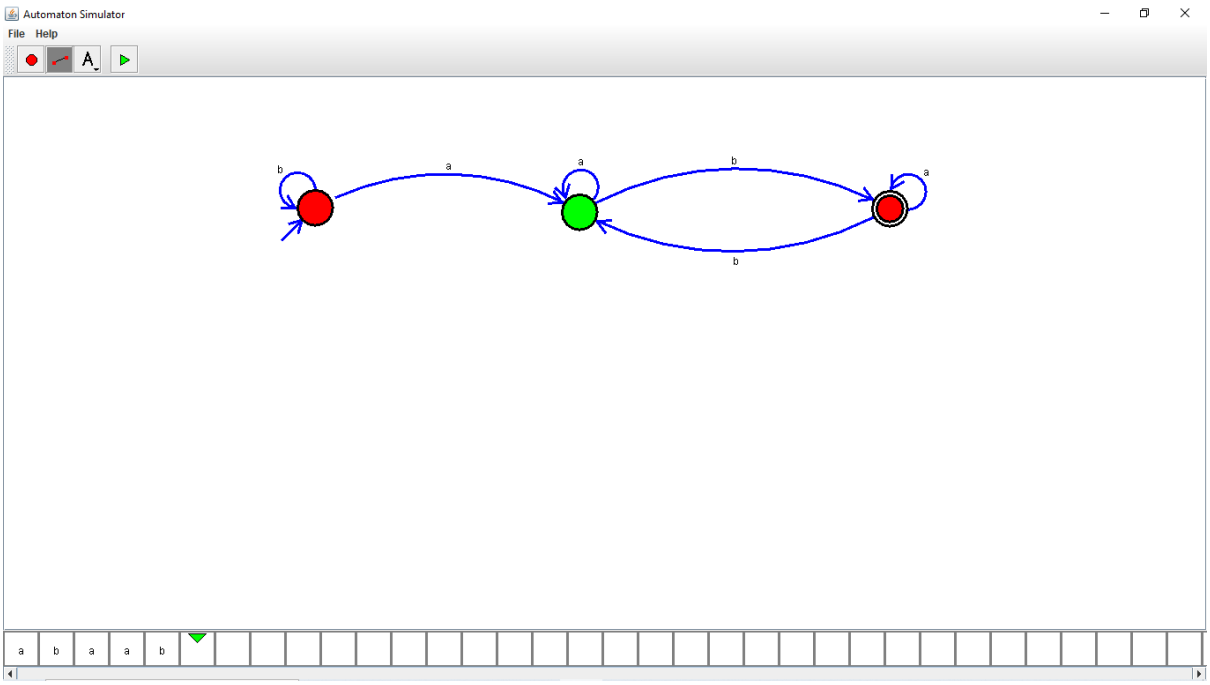


23.Design DFA using simulator to accept the string the end with ab over set {a,b)

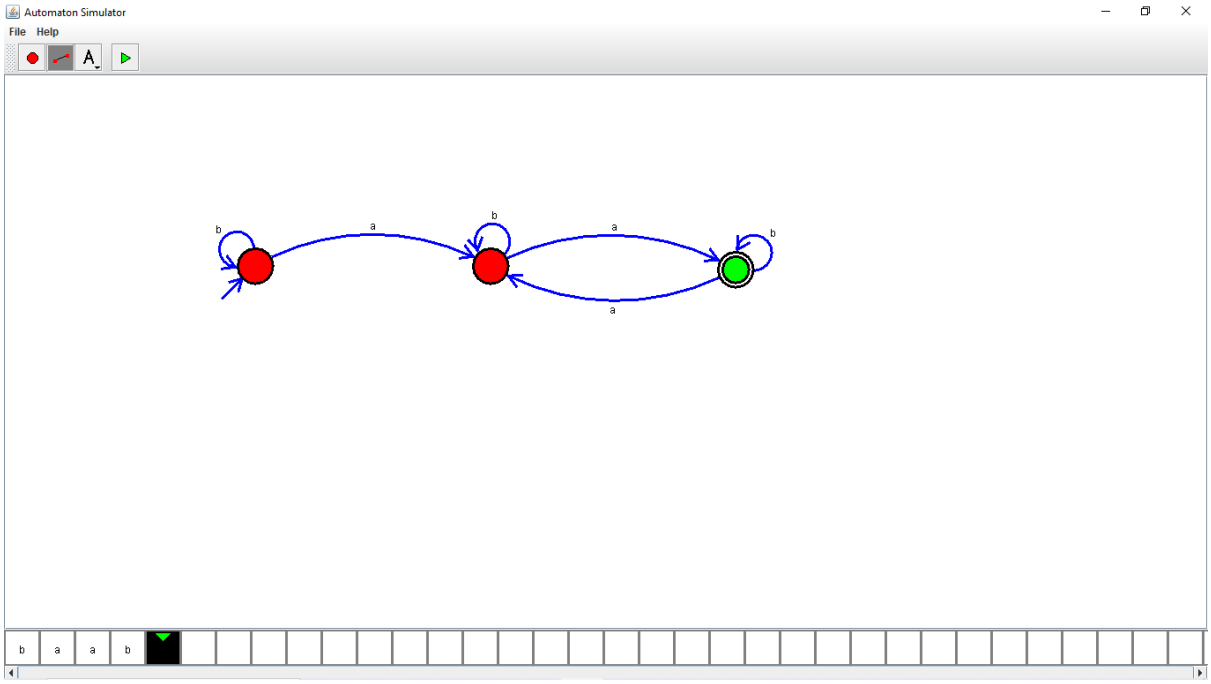
W= aaabab



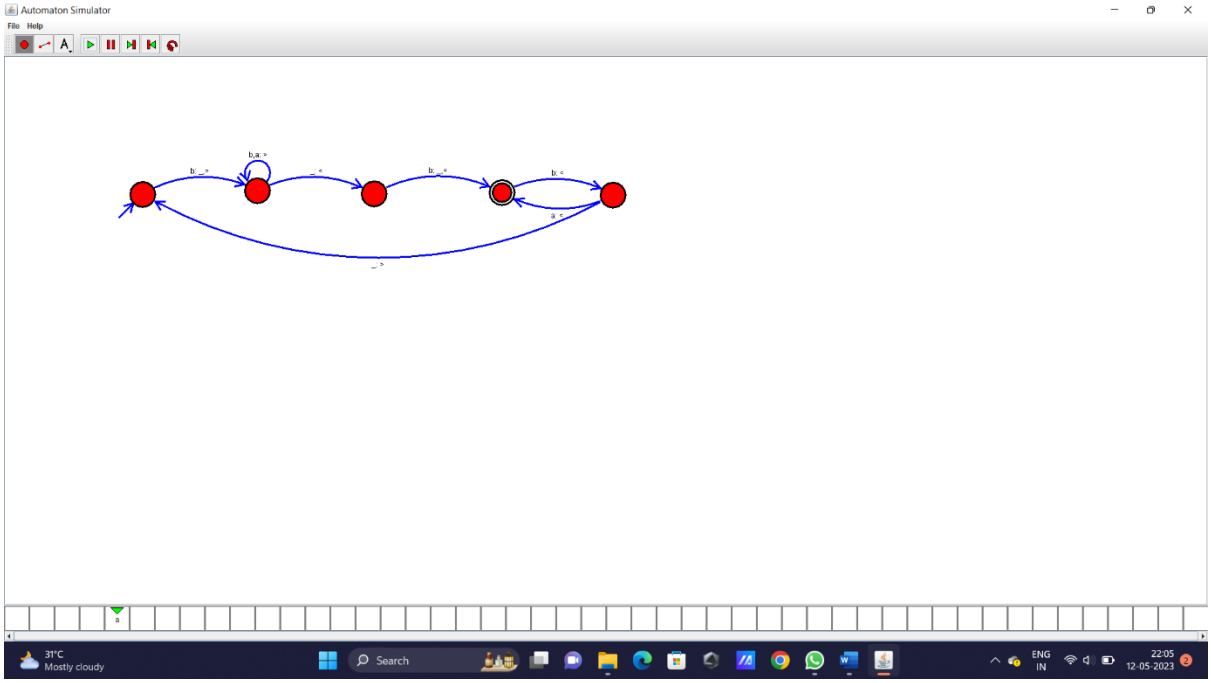
24.Design DFA using simulator to accept the string having ‘ab’ as substring over the set {a,b}



25.Design DFA using simulator to accept the string start with a or b over the set {a,b}

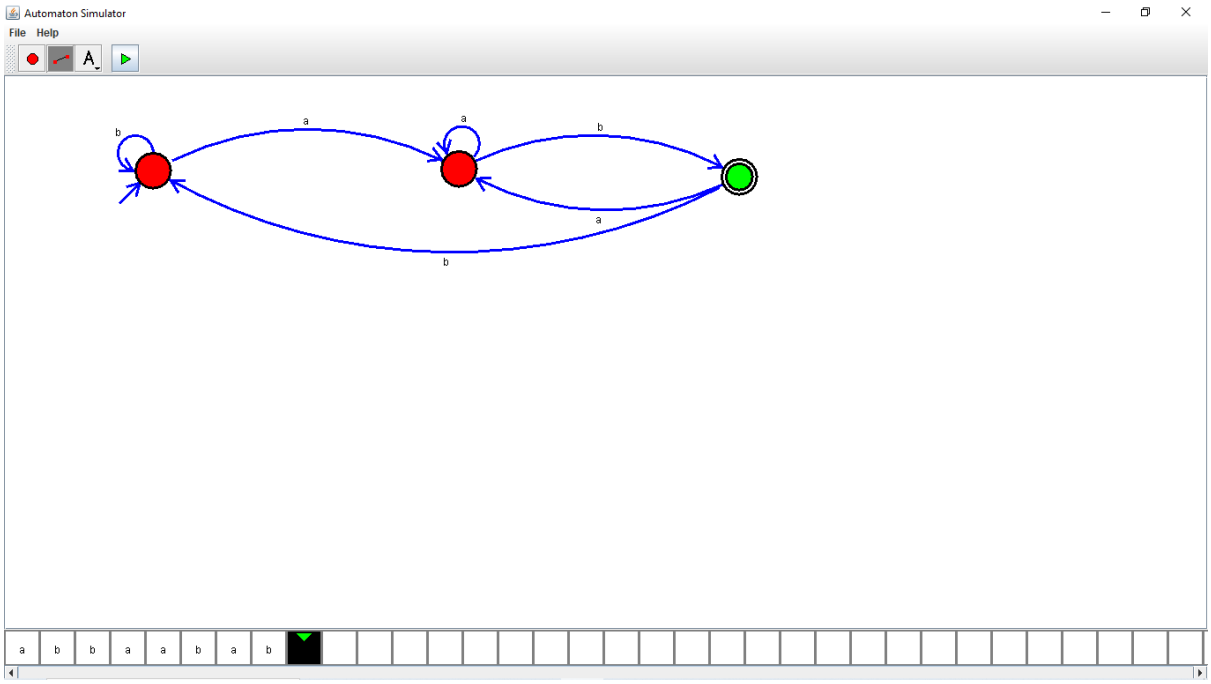


26.Design TM using simulator to accept the input string Palindrome bbabb

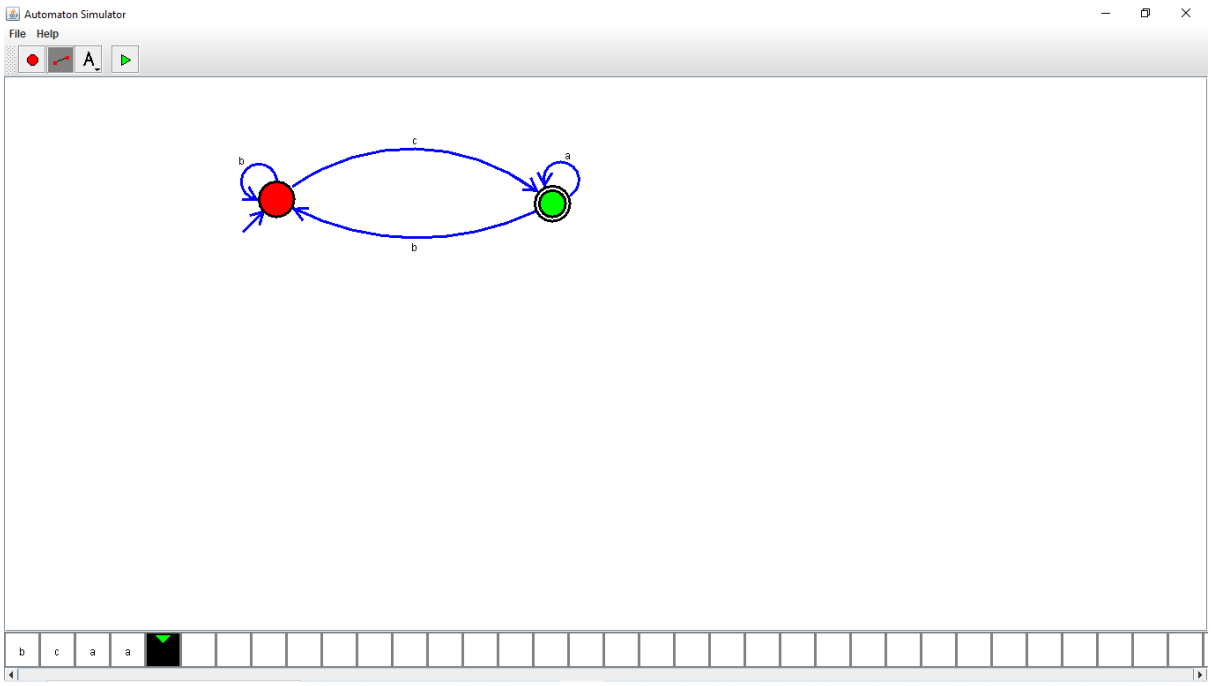


28.Design DFA using simulator to accept the string the end with ab over set {a,b)

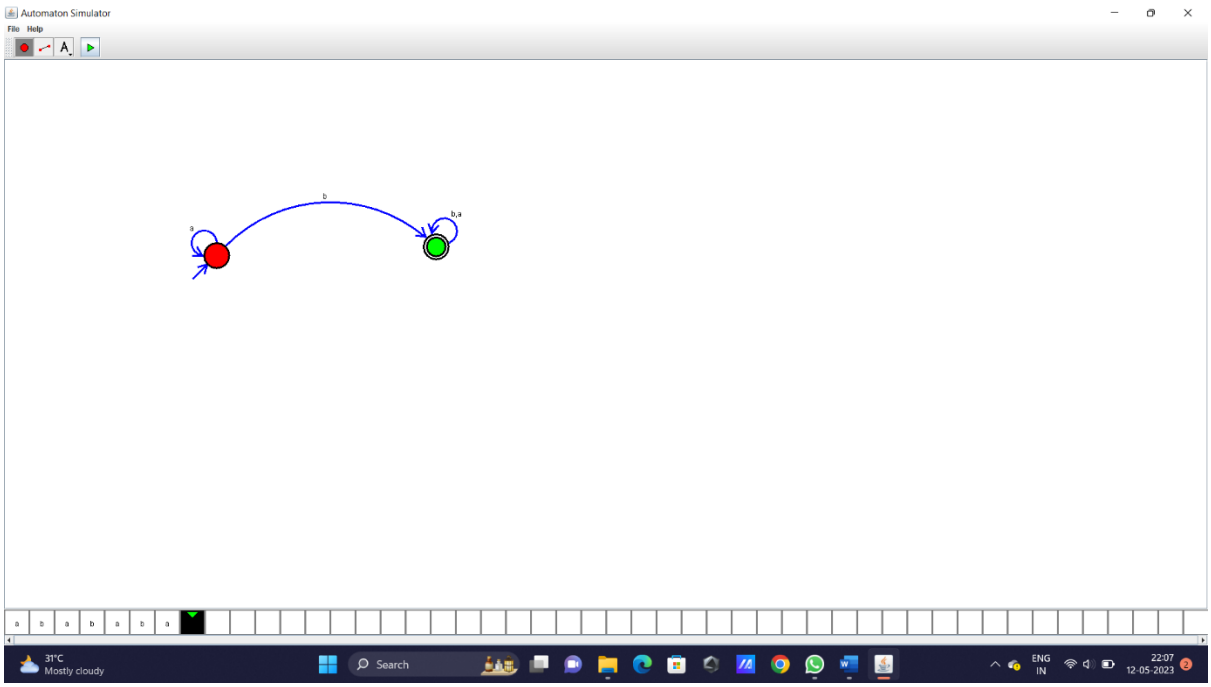
W= abbaabab



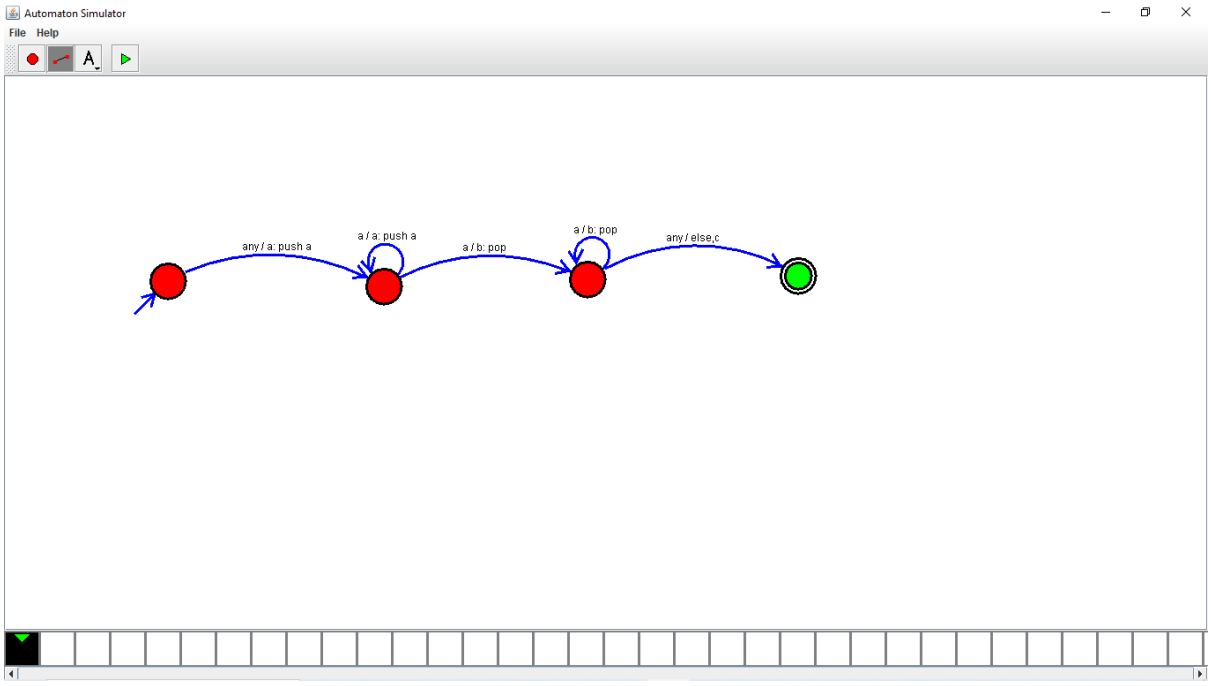
29.Design DFA using simulator to accept the input string “bc” ,”c”,and ”bcaaa”.



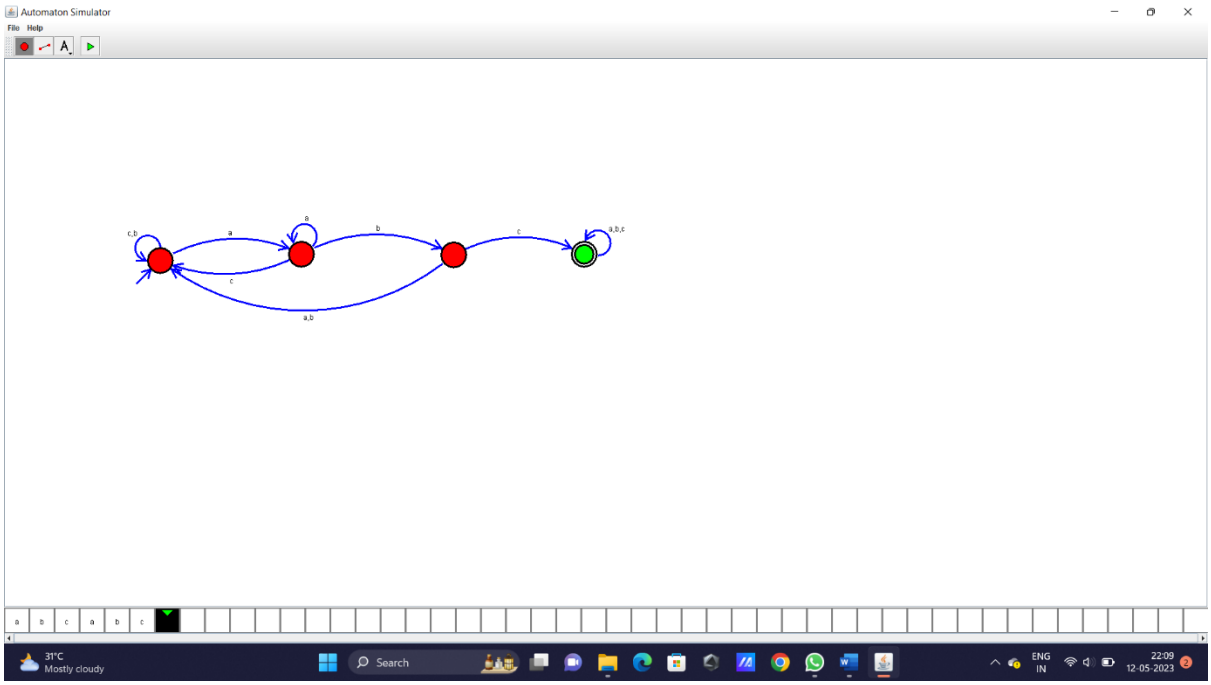
30.Design NFA to accept any number of a's where input={a,b}.



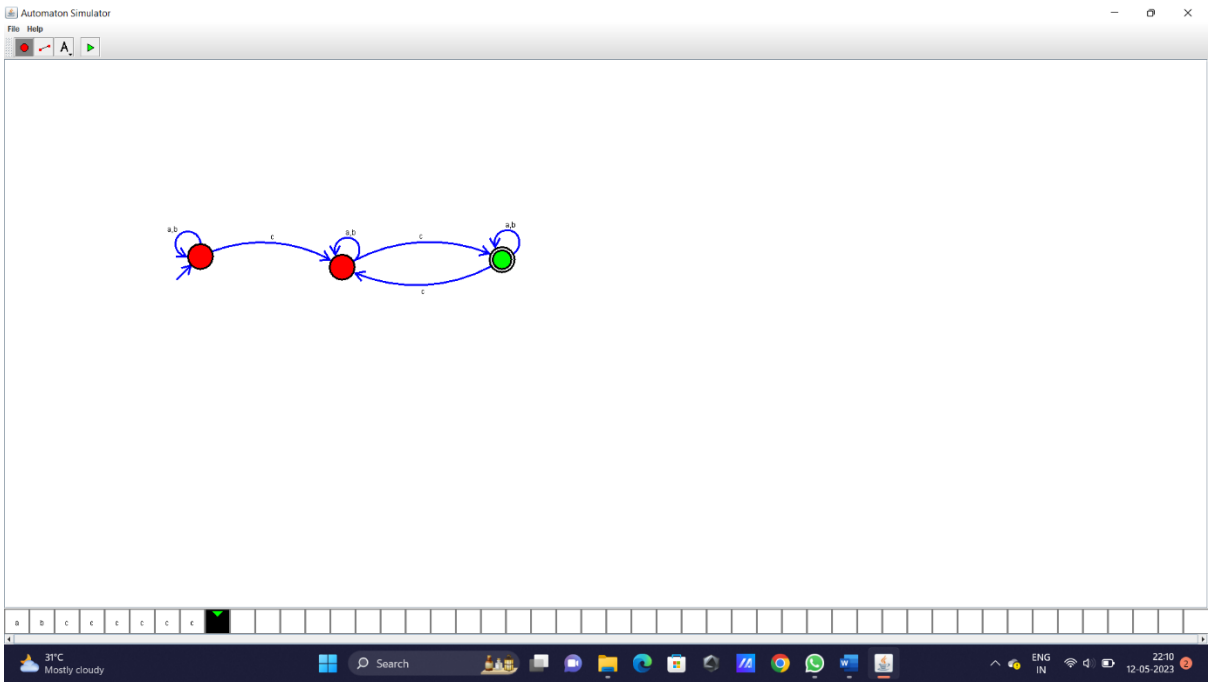
31.Design PDA using simulator to accept the input string $a^n b^n$



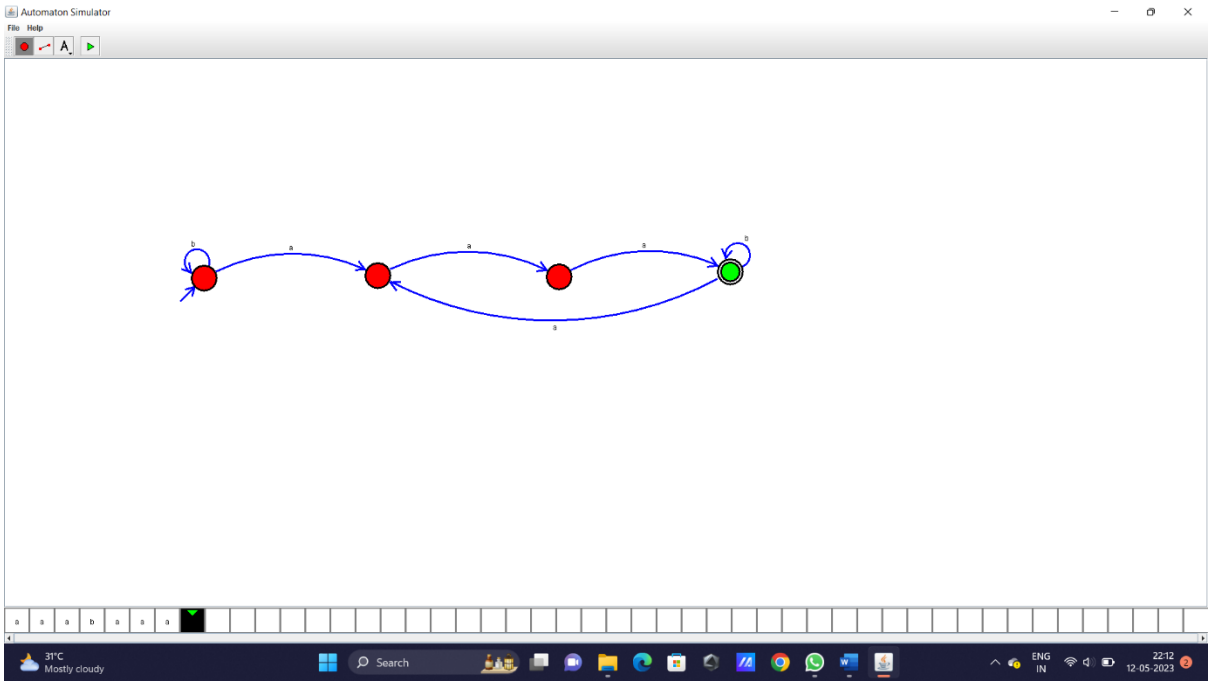
33.Design DFA using simulator to accept the string having ‘abc’ as substring over the set {a,b,c}



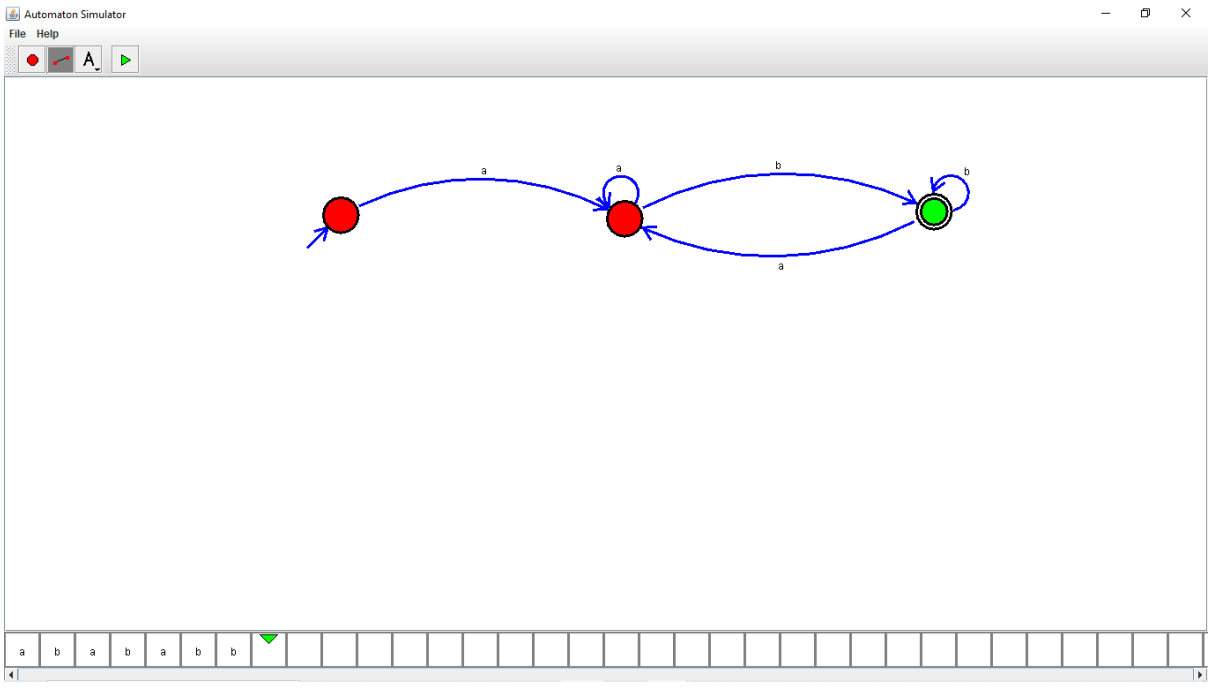
34.Design DFA using simulator to accept even number of c’s over the set {a,b,c}



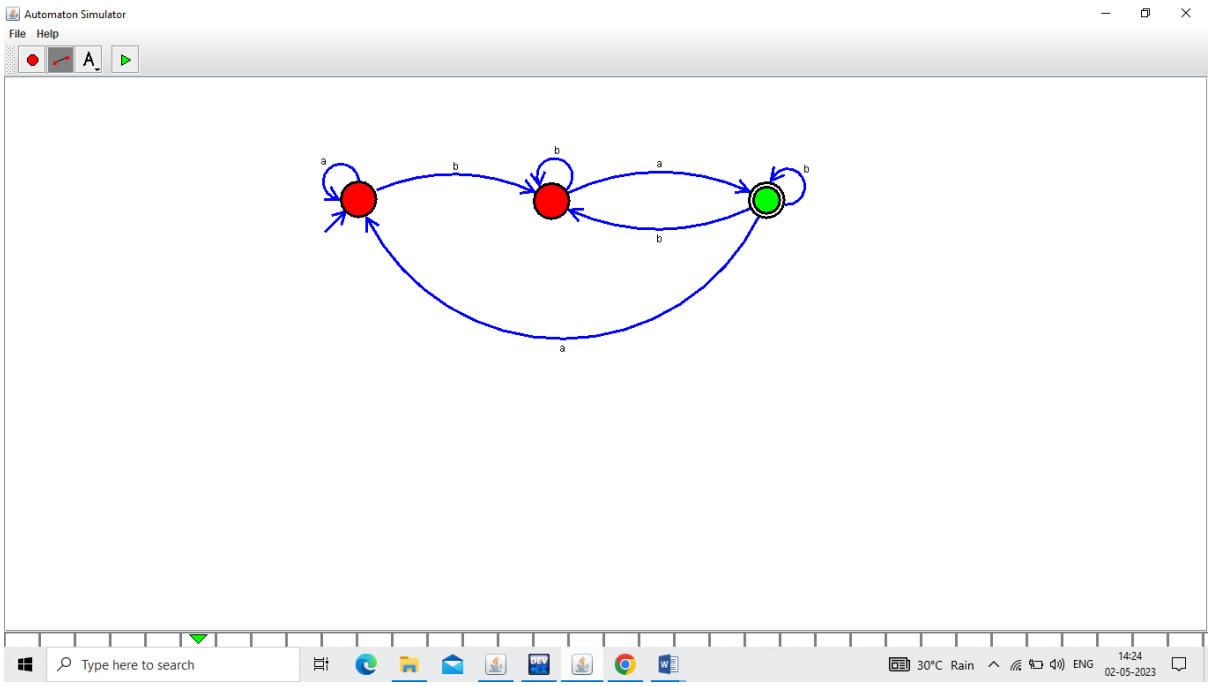
35.Design DFA using simulator to accept strings in which a's always appear tripled over input {a,b}



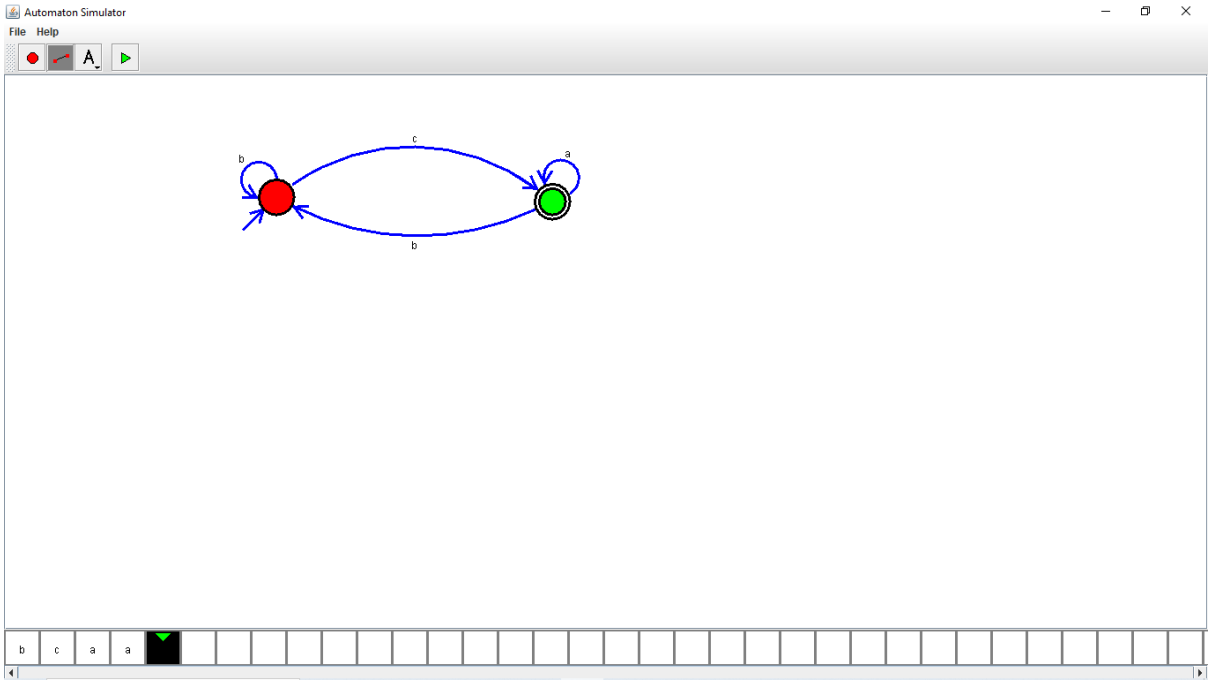
36.Design NFA using simulator to accept the string the start with a and end with b over set {a,b} and check W= abaab is accepted or not.



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

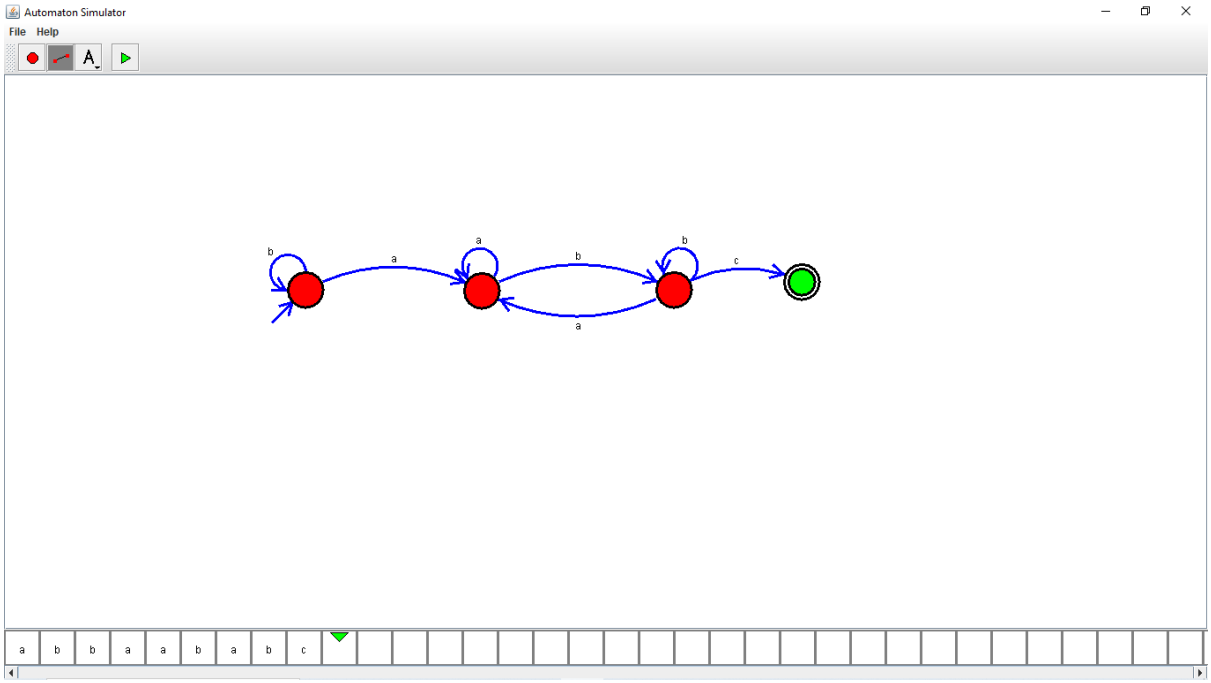


38.Design NFA using simulator to accept the input string “bbc” ,”c”,and ”bcaaa”.



39.Design DFA using simulator to accept the string the end with abc over set {a,b,c)

W= abbaababc



40.Design NFA to accept any number of b's where input={a,b}.