EXP NO : 1

DETERMINISTIC FINITE AUTOMATA (DFA)

AIM :-

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

ALGORITHM :-

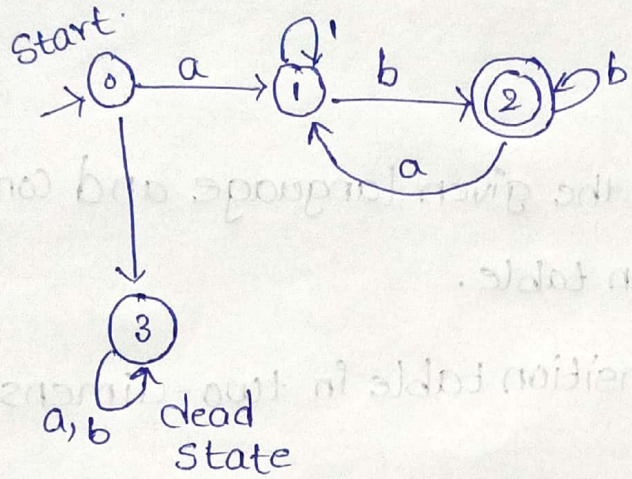
1. Draw DFA for the given language and construct the transition table.
2. Store the transition table in two-dimension array.
3. Initialize present_state, next_state and final state.
4. Get the input string from user.
5. Find the length of input string
6. Read the input string character by character.
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, if final state is
9. when we reach the end of the input, if final state reached the input accepted.

Scanned with CamScanner

EXAMPLE:-

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

Design of the DFA:-



Transition Table :-

Input	a	b
0	1	3
1	1	2
(2)	1	2
3	1	3

Scanned with CamScanner

PROGRAM :-

```
#include <stdio.h>
#include <string.h>
#define Max 20
  int main()
{
    int tran_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 string");
     scanf("%s", input_string);
     int l = strlen[input_string];
     for(i=0; i<l; i++)
     {
        if(input_state[i] == 'a')
        next_state = tran_table[present_state[0];
         else if(input_string[i] == 'b')
        next_state = tran_table[present_state][1];
        else
          invalid = 1;
          present_state = next_state;
     }
```

Scanned with CamScanner

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

Output:-

Enter string : abaaab

Don't Accept

Enter String : abbaba

Accept.

Scanned with CamScanner

Ex No :- 2 CHECKING WHETHER A STRING BELONGS TO A GRAMMAR

Aim :-

To write C program to check whether a string belongs to grammar $S \rightarrow 0A1$

$A \rightarrow 0A1 | 1A | \epsilon$

Language defined by the Grammer :-

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

Algorithm :-

1. Get the Input String from the user
2. Find the length of the string.
3. check wether all the symbols in the input are either 0 or 1. if so, print "string is vaild" and go to step 4. otherwise print "string not valid" and quit the program.
4. If the symbol 0 and the last symbol is 1, print "String accepted". Otherwise print "string not accepted".

Scanned with CamScanner

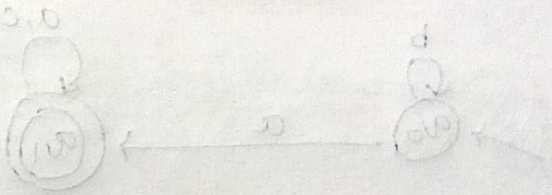
program :-

```
#include <stdio.h>
#include <string.h>
 int main()
{
  char s[100];
  int i, flag;
   int I;
  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");
   if (flag == 1)
   {
        if (s[0] == '0' && s[-1] == '1')
```

Scanned with CamScanner

```
        printf("string accepted");
    else
        printf("string is not accepted");
    }
}
```

Output :-

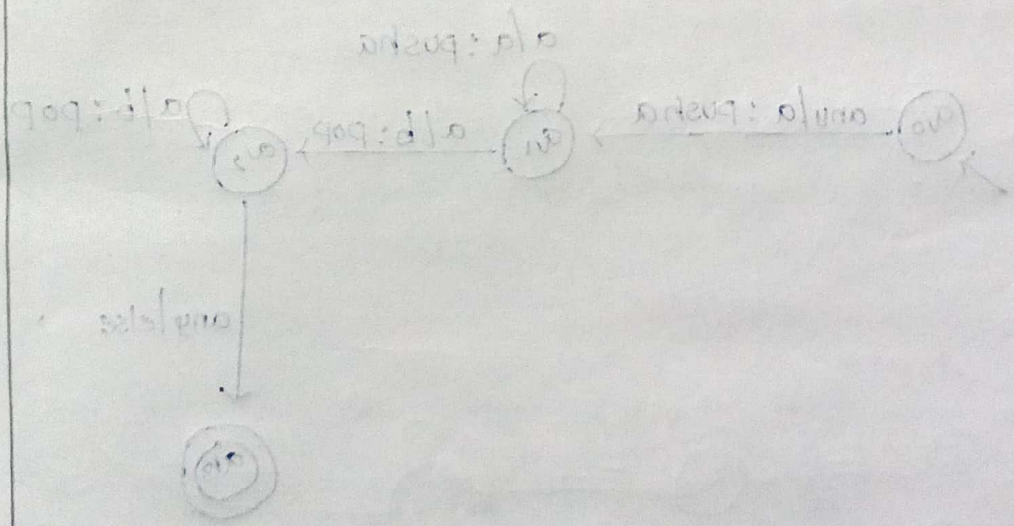


Enter a string to check : 0101011101

String is accepted.

Enter a string to check : 0110110

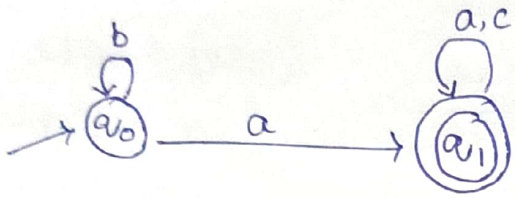
String is Not accepted.



Scanned with CamScanner

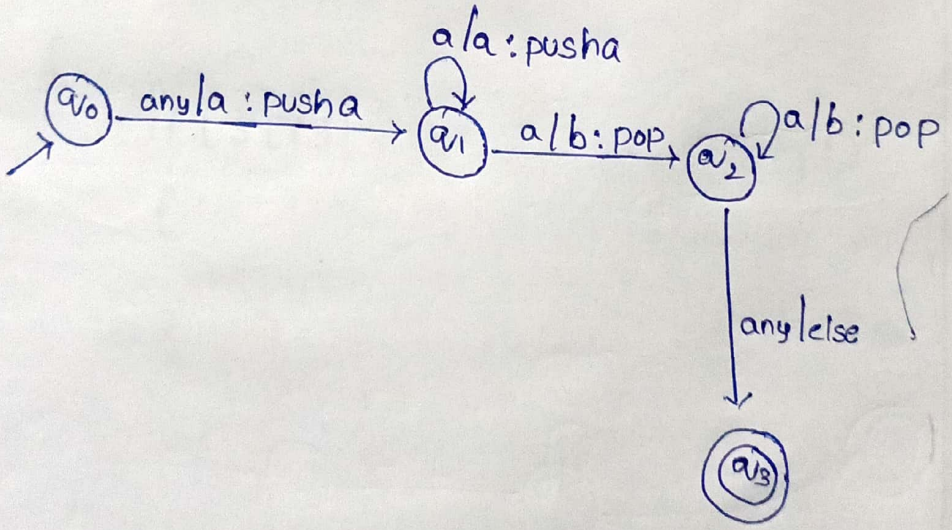
Ex:3 - Design DFA using simulator to accept the input string "a", "ac", and "bac".

Input:-



Ex4:- Design PDA using simulator to accept the input string aabb.

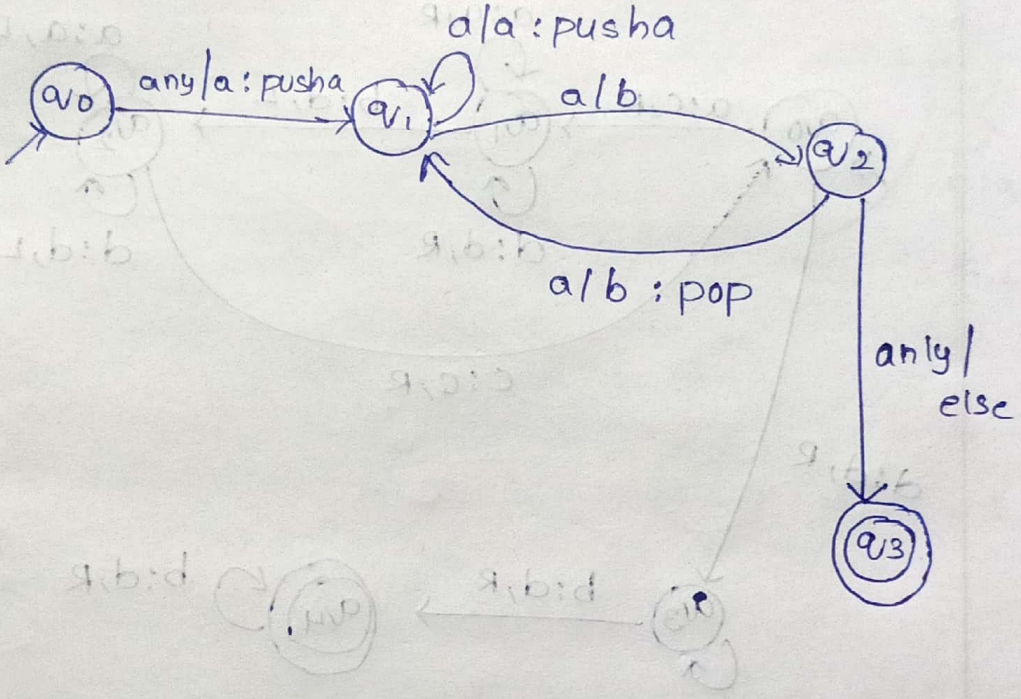
Input :-



Scanned with CamScanner

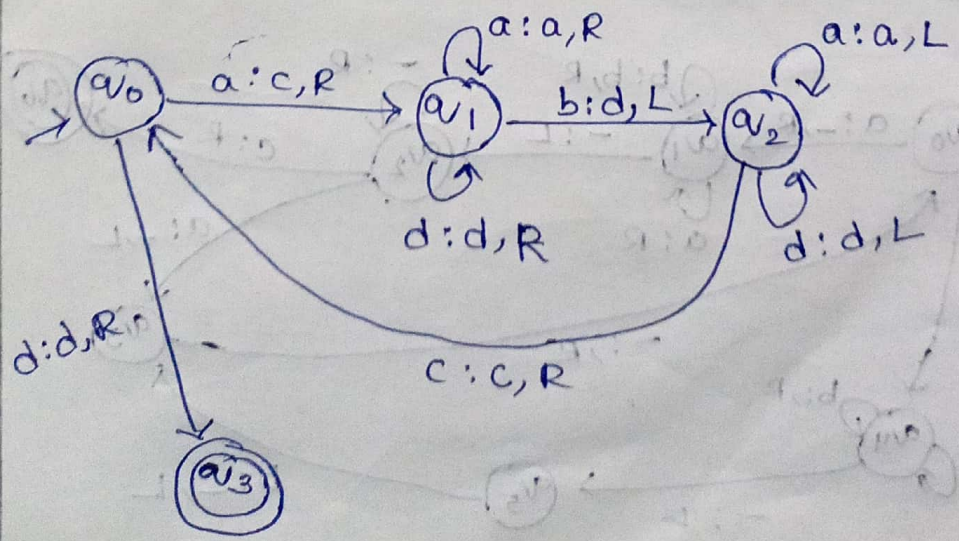
Ex:5:- Design PDA using Simulator to accept the input string $a^n b^{2n}$.

Input :- a a b b b



Ex 6:- Design TM using Simulator to accept the input string $A^n B^n$.

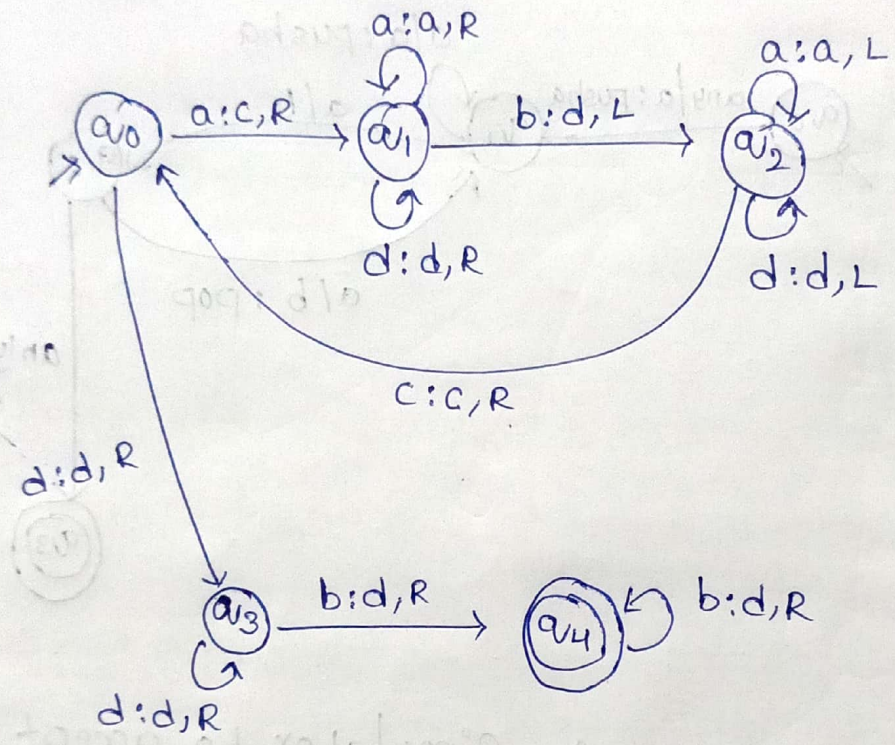
Input :- a a b b



Scanned with CamScanner

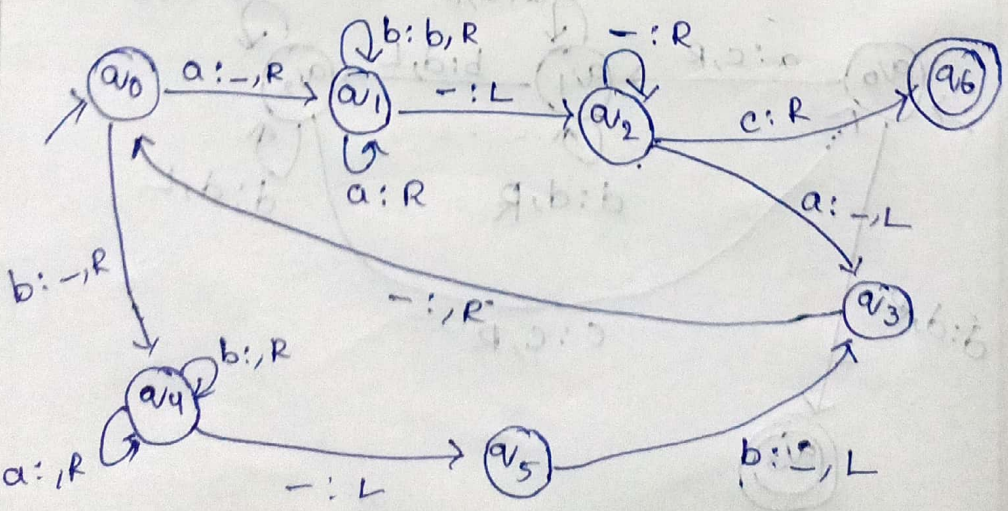
Ex 7:- Design TM using Simulator to accept the input string $A^n B^{2n}$.

Input :- a a b b b b



Ex 8:- Design TM Using Simulator to accept the input string palindrome ababa

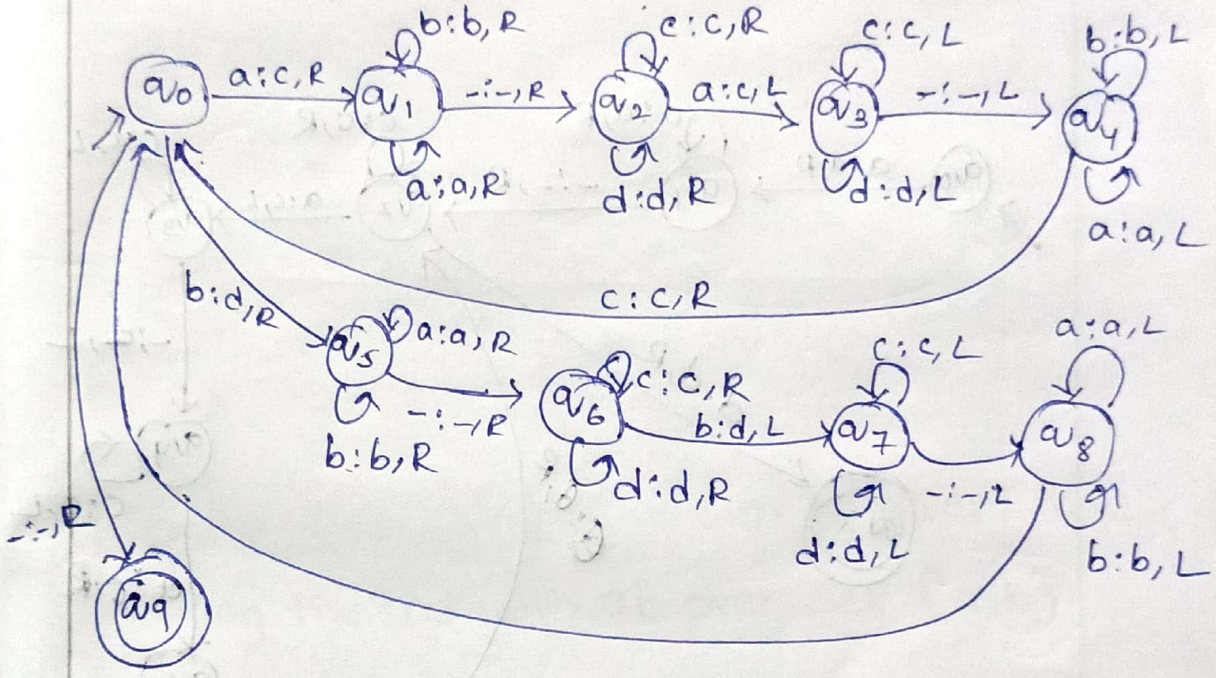
INPUT :- ababa c



Scanned with CamScanner

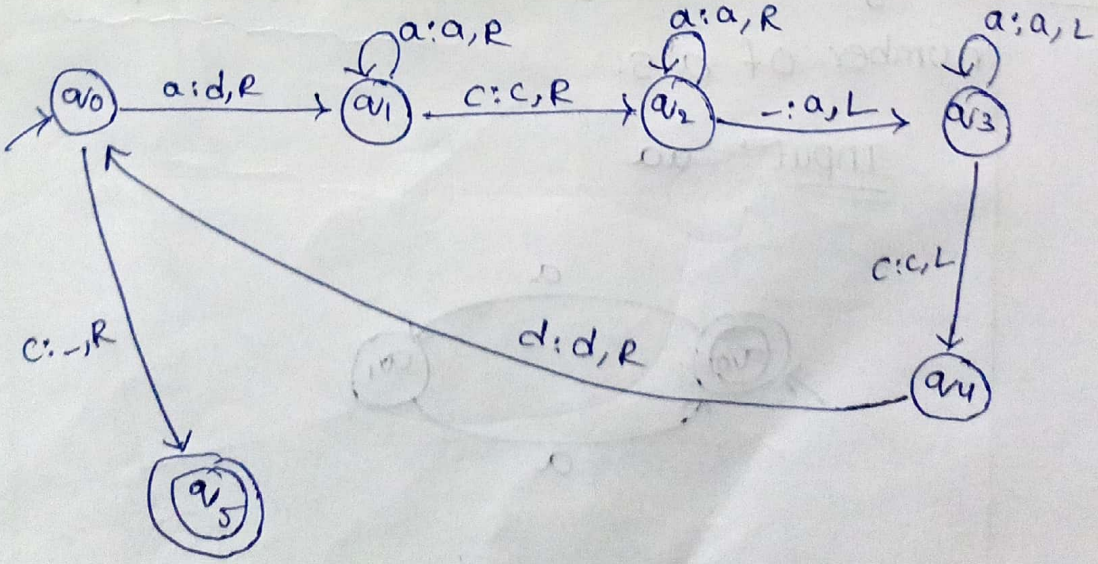
Ex:9:- Design TM using Simulator to accept the Input String ww

Input :- w = aba aba



Ex10:- Design TM using Simulator to perform addition of "aa" and "aaa"

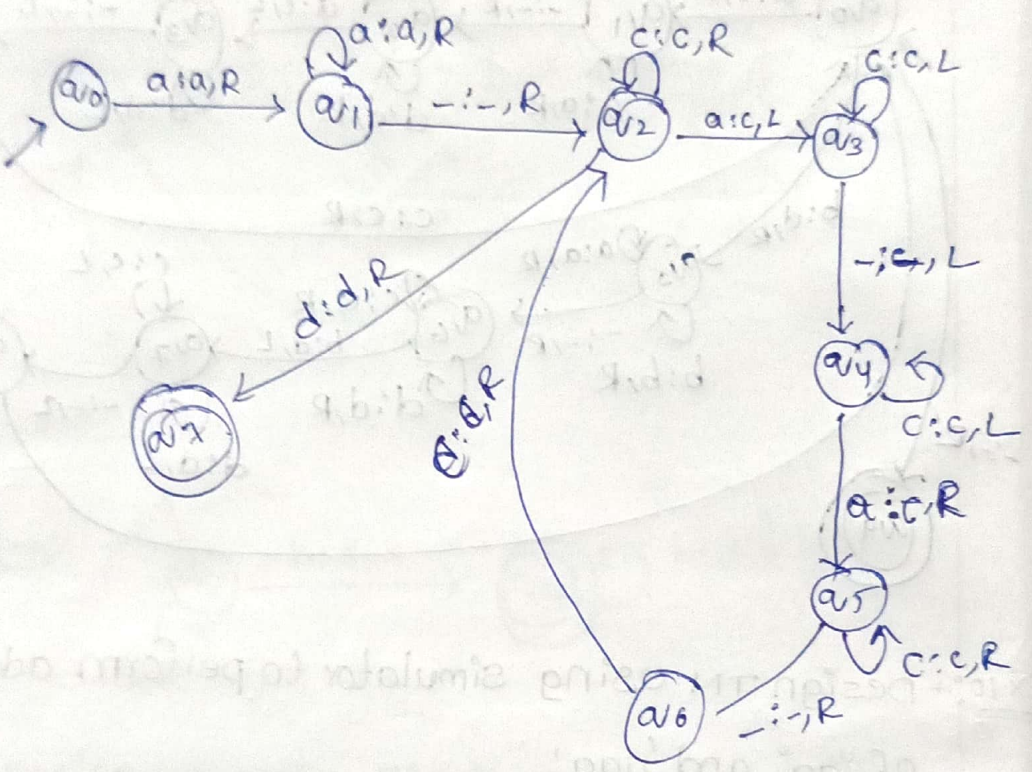
Input:- aa c aaa



Scanned with CamScanner

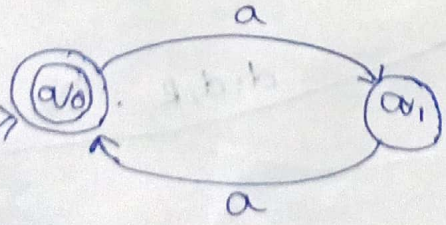
Ex 11:- Design TM using simulator to perform subtraction of aaa - aa

Input :- aaa-aa



Ex 12:- Design DFA using simulator to accept Even number of a's.

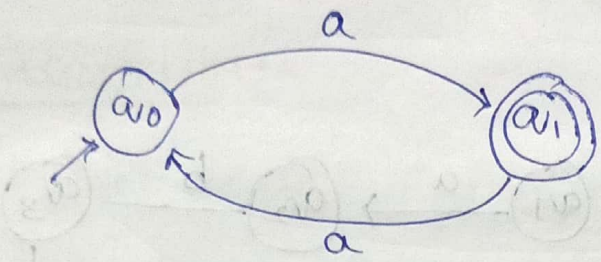
Input :- aa



Scanned with CamScanner

Ex:13:- Design DFA using Simulator to accept odd number of a's.

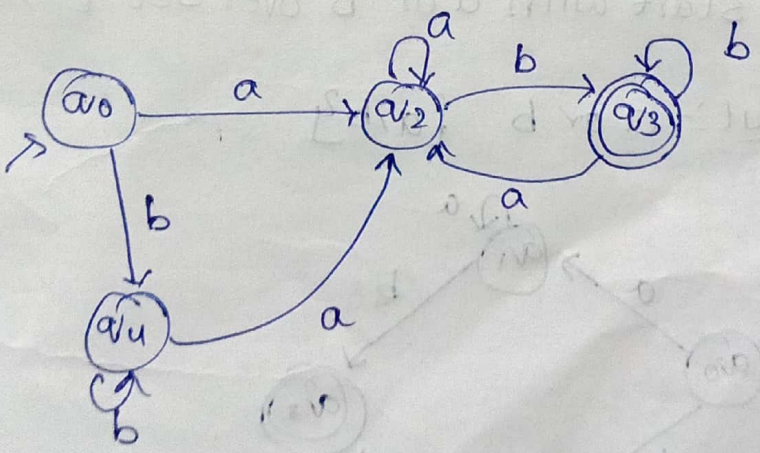
Input:- aaa



Ex 14:- Design DFA using Simulator to accept the string the end with ab over set {a,b}

$w = aaabab$.

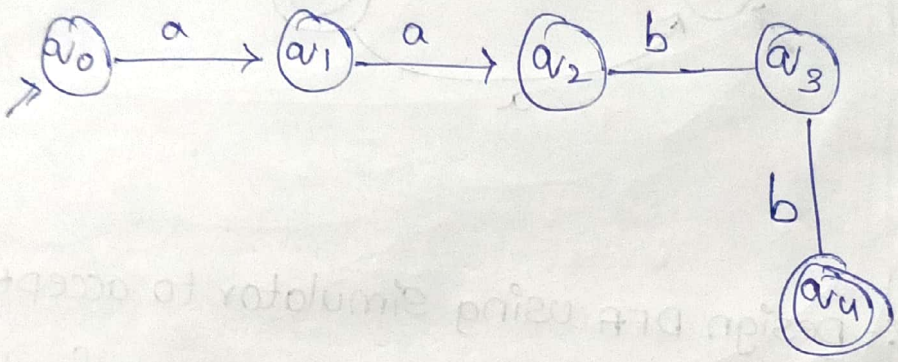
Input:- aaabab



Scanned with CamScanner

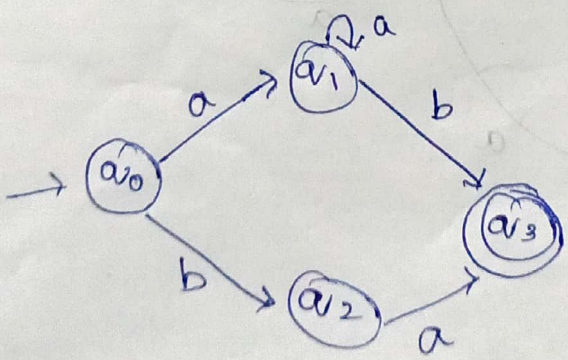
EX:15:- Design DFA using simulator to accept the string having "ab" as substring over the set {a,b}

Input:- aabb



EX:16:- Design DFA using simulator to accept the string start with a or b over set {a,b}

Input:- a or b {a,b}



Scanned with CamScanner

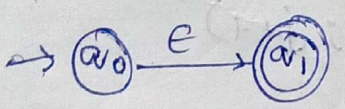
Ex 17:- FINDING ε-CLOSURE FOR NFA WITH ε-MOVES:-

AIM:-

To write a c program to find ε-closure of a Non-Deterministic finite Automata with ε-move.

ALGORTIHM:-

1. Get the following input from the user.
   (i) Number of states in NFA
   (ii) Number of symbol input alphabet includi ε.
   (iii) input symbols.
   (iv) Number of final states and their name.
2. Declear a 3-dimensional matrix to store the transistions and initialize.
3. Get the transitions from every state for input symbol from the user and store.



4. Initialize of two-dimensional matrix ε-closure with -1 in all Entries.
5. ε-closure of state q is defined at set State that can be reached.

Scanned with CamScanner

ε - closure (0) = 0, 1

ε - closure (1) = 1,

ε - closure (2) = 2,

6. For every state - print e- closure values.

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_state, 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;
```

Scanned with CamScanner

```
symbol[10] = 'e';
n=1;
tran_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_state; i++)
  e_closure[i][0]= i;
for(i=0; i<num_state; i++)
{
if(trans_table[i][0][0]== -1)
continue;
else
{
  state=i;
  ptr=1;
  find_e_closure(i);
} }
For(i=0; i<num_state; i++)
{
printf("e-closure(%d)={",i);
 For(j=0; j<num_states; j++)
{
 if(e-closure[i][j]!=-1)
 {
```

Scanned with CamScanner

```
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 :-

e-closure(0) = {0, 1, }

e-closure(1) = {1, }

e-closure(2) = {2, }

Scanned with CamScanner