Name: Naseem Shah	**Roll No:** 058
Exp. No: 02 07/09/2021	**Date:**

DFA Implementation of 1*2*3*

Aim: Write a C program to implement a DFA for the regular expression 1*2*3* using transition table

```
/*
C program to implement a DFA for the regular expression 1*2*3* using
transition table.
*/
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

const int transition_table[][3] = {
    {1, 2, 3},  // state 0, initial & final state
    {1, 2, 3},  // state 1, final state
    {4, 2, 3},  // state 2, final state
    {4, 4, 3},  // state 3, final state
    {4, 4, 4}   // dead state
};

const int final_state[] = {0, 1, 2, 3};
const int num_final_states =
sizeof(final_state)/sizeof(final_state[0]);

void main() {
    char s[1000];
    bool valid = false;
    int state = 1;
    printf("input string : ");
    scanf("%s", s);

    for(int i=0; s[i] != '\0' ; i++){
        if(s[i] != '1' && s[i] != '2' && s[i] != '3')
            state = 4;

        if(state == 4)
            break;
        state = transition_table[state][s[i] - '1'];
    }

    for(int i=0; i<num_final_states; i++)
        if(state == final_state[i]){
            valid = true;
            break;
        }

    if(valid)
        printf("Valid string!\n");
    else
        printf("Invalid string!\n");

    return;
}
```

Result: Successfully written C program to implement a DFA for the regular expression 1*2*3* using transition table

Remarks:(To be filled by faculty)

Algorithm

```
1. Start
2. Create a NFA and, then a DFA, for the given regular expression, 1*2*3*.
3. Create transition table based on the DFA obtained, transition_table, where each
   transition_table[i][j] denotes the current state i, and the next state when input is j.
   transition_table = { {1,2,3}, {1,2,3}, {4,2,3}, {4,4,3}, {4,4,4}}
4. Set final_states = {0, 1, 2, 3}
5. Read the input string, s
6. Set state = 0, valid = false
7. for each character ch in s, do
      a. if ch != ‘1’ and s[i] != ‘2’ and ch != ‘3’, then set state = 4
      b. if state == 4, then break from the loop
      c. state = transition_table[state][ch - ‘1’]
8. for i in final_states, do
      a. if(state == i), then
            i.  valid = true
            ii. break from the loop
9. if valid == true, then print “Valid string”, else print “Invalid string”
10. Stop
```

Diagrams & Tables

NFA

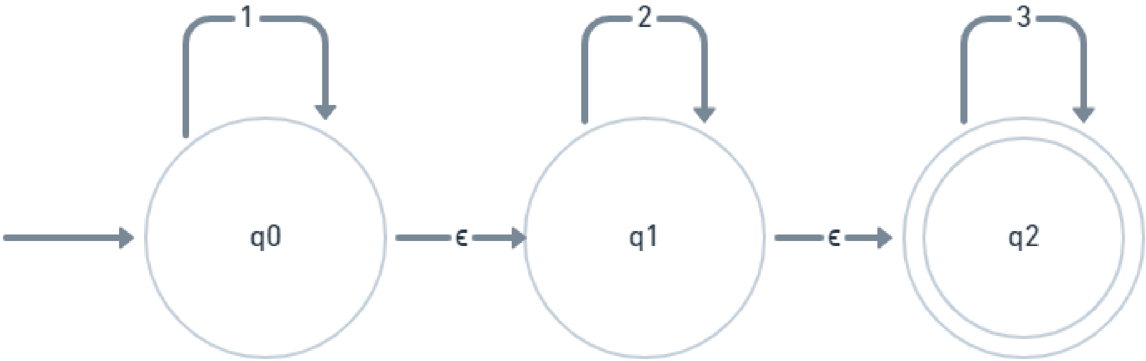


Table to Remove Epsilon transitions

State	1	2	3
q0	{q0, q1, q2}	{q1, q2}	{q2}
q1	ϕ	{q1, q2}	{q2}
q2	ϕ	ϕ	{q2}

NFA after Removing Epsilon Transitions

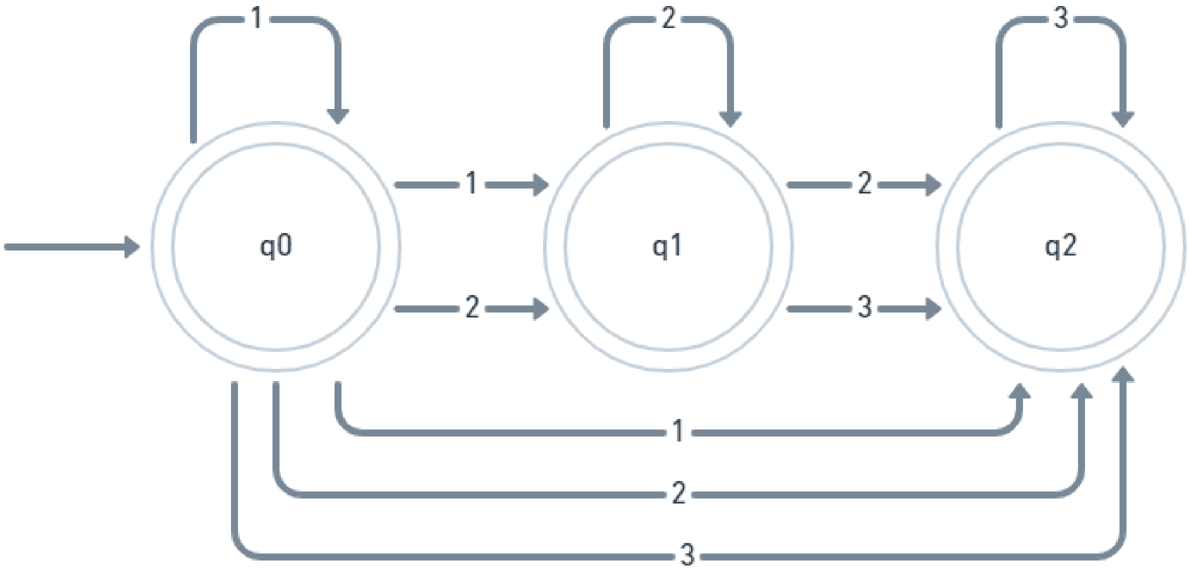
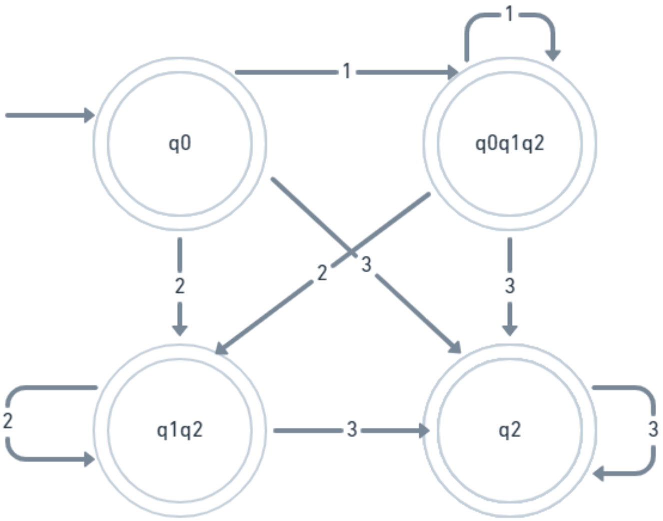


Table for DFA

State	1	2	3
q0	[q0q1q2]	[q1q2]	q2
[q0q1q2]	[q0q1q2]	[q1q2]	q2
[q1q2]	ϕ	[q1q2]	q2
q2	ϕ	ϕ	q2

DFA

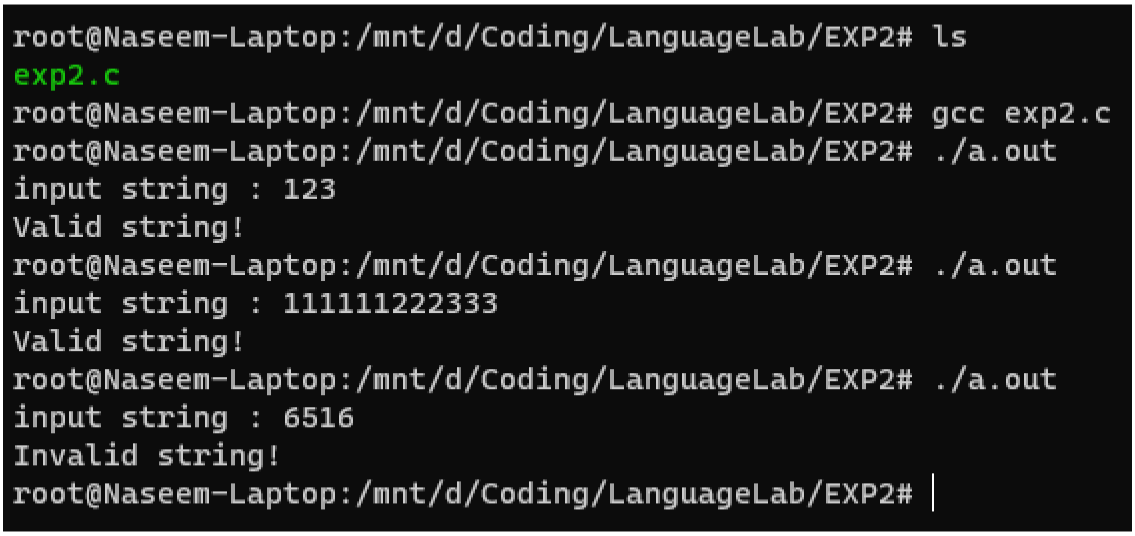


Transition Table

State	1	2	3
*q0	q1	q2	q3
*q1	q1	q2	q3
*q2	q4	q2	q3
*q3	q4	q4	q3
q4	q4	q4	q4

With q0 as q0, q0q1q2 as q1, q1q2 as q2, q2 as q3, and q4 as dead state.

Sample output



```
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/EXP2# ls
exp2.c
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/EXP2# gcc exp2.c
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/EXP2# ./a.out
input string : 123
Valid string!
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/EXP2# ./a.out
input string : 111111222333
Valid string!
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/EXP2# ./a.out
input string : 6516
Invalid string!
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/EXP2#
```