**Practical No-1**

**Aim :** Write and Implement a program to simulate Deterministic Finite Automata

**Theory :**

In the [theory of computation](http://en.wikipedia.org/wiki/Theory_of_computation) and [automata theory](http://en.wikipedia.org/wiki/Automata_theory), a deterministic finite state machine—also known as deterministic finite automaton (DFA)—is a [finite state machine](http://en.wikipedia.org/wiki/Finite_state_machine) accepting finite strings of symbols. For each state, there is a transition arrow leading out to a next state for each symbol. Upon reading a symbol, a DFA jumps *deterministically* from a state to another by following the transition arrow. Deterministic means that there is only one outcome (i.e. move to next state when the symbol matches (S0 -> S1) or move back to the same state (S0 -> S0)). A DFA has a [start state](http://en.wikipedia.org/wiki/Finite_state_machine#Start_state) (denoted graphically by an arrow coming in from nowhere) where computations begin, and a [set](http://en.wikipedia.org/wiki/Set_(mathematics)) of [accept states](http://en.wikipedia.org/wiki/Finite_state_machine#Accept_state) (denoted graphically by a double circle) which help define when a computation is successful.

**Strings**

* A string over some alphabet is a sequence of symbols drawn from that alphabet.

For example, *banana* is a sequence of six symbols drawn from the ASCII alphabet.

* The empty string, denoted by epsilon, e , is a special string with zero symbols.
* If *x* and *y* are two strings then the concatenation of *x* and *y*, written *xy*, is the string formed by appending *y* after *x* .

For example if **x = dog** and **y = house** then **xy = doghouse**.

By definition x0 is the empty string, e , and x For ex., if x = ba and y = na then xy2= banana.

**Languages**

A language is a set of strings over some fixed alphabet: The set may contain a finite number or an infinite number of strings. If L and M are two languages then their union, denoted by L union M, is the language containing every string in L and every string in M (any string thatis in both L and M appears only once in L union M )

If L and M are two languages then their concatenation, denoted by LM, is the language containing every string in L concatenated with every string in M .

For example if L = {dog, ba, na} and M = {house, ba} then

L union M = {dog, ba, na,house} and LM = {doghouse, dogba, bahouse, baba, nahouse, naba}

Language exponentiation concatenates a language with itself a given number of times: L2 = LL,

By definition L0 = { } and L1 = L .

The Kleen closure of a language L, denoted by L\*, equals



For example, if L = {a, b} then L\* = { e, a, b, aa, ab, ba, bb,aaa, aba, baa, bba, aab, bab, bbb, .. }.

The positive closure of a language L, denoted by L+

For example, if L = {a, b} then L+ = {a, b, aa, ab, ba, bb, aaa,aba, baa, bba, aab, abb, bab, ... }.

Let L = {A, B, ... , Z, a, b, ..., z} be the set of all capital and lower-case letters and let D = {0, 1, ... , 9} be the set of all digits. Then:

L union D is the set of all letters and digits.

LD is the set of strings consisting of a letter followed by a digit.

L4 is the set of all four-letter strings.

L\* is the set of all strings of letters including the empty string, e.

L (L union D )\* is the set of all strings of letters and digits that begin with a letter.

D+ is the set of all strings of one or more digits.

**Regular Expressions**

* Specify simple (possibly infinite) strings
* The set of strings defined by a RE are called *regular sets*
* Start with a finite character set called a *vocabulary,* or V
* Empty strings are allowed
* Strings are created by concatenating characters from the vocabulary
* just like used for lex
* REs are great for defining tokens
* Every regular expression specifies a language according to the following rules:

e is a regular expression that denotes { }, the set containing just the empty string.

If *a* is a symbol in the alphabet then the regular expression *a* denotes {a}, the set containing just that symbol.

Suppose *s* and *t* are regular expressions denoting languages L(s ) and L(t ),respectively. Then:

* (s ) | (t ) is a regular expression denoting L(s ) union L(t ).
* (s )(t ) is a regular expression denoting L(s )L(t ).
* (s )\* is a regular expression denoting L(s )\*.
* (s ) is another regular expression denoting L(s ) so we can put extra pairs of parentheses around regular expressions if we desire.

**Regular Definitions**

A regular definition gives names to certain regular expressions and uses those names in other regular expressions. As an example:

letter --> A | B | ... | Z | a | b | ... | z

digit --> 0 | 1 | ... | 9

id --> letter ( letter | digit )\*

This defines letter to be the regular expression for the set of all upper-case and lower case letters in the English alphabet, digit to be the regular expression for the set of all decimal digits, and id to be the regular expression for all strings of letters and digits that begin with a letter.

Note that id is the pattern for the Pascal identifier token.

The pattern for the Pascal number token can be specified as follows:

**digit --> 0 | 1 | ... | 9**

**digits --> digit digit\***

**optional\_fraction --> . digits | e**

**optional\_exponent --> ( E ( + | - | e ) digits ) | e**

This definition says that an optional\_fraction is either a decimal point followed by one or more digits or it is missing (the empty string).

* An optional\_exponent is either the empty string or it is the letter E followed by an optional + or - sign, followed by one or more digits.
* num --> digits optional\_fraction optional\_exponent
* If r is a regular expression then:

**(r )+ = r r \* and r ? = (r | e)**

With these shorthand notations one can write:

**num = digits (. digits)? (E ( + | - )? digits)?| digit+ (.digit+)? (E (+ | - )? digit+)?**

**Finite Automaton**

A deterministic finite automaton *M* is a 5-tuple, (*Q*, Σ, δ, *q0*, *F*), consisting of

* a finite set of states (*Q*)
* a finite set of input symbols called the alphabet (Σ)
* a transition function (δ : *Q* × Σ → *Q*)
* a start state (*q0* ∈ *Q*)
* a set of accept states (*F* ⊆ *Q*)

Let *w = a1a2 ... an* be a string over the alphabet Σ. The automaton *M* accepts the string *w* if a sequence of states, *r0,r1, ..., rn*, exists in *Q* with the following conditions:

1. *r0* = *q0*
2. *ri+1* = δ(*ri*, *ai+1*), for *i* = *0, ..., n−1*
3. *rn* ∈ *F*.

In words, the first condition says that the machine starts in the start state *q*0. The second condition says that given each character of string *w*, the machine will transition from state to state according to the transition function δ. The last condition says that the machine accepts *w* if the last input of *w* causes the machine to halt in one of the accepting states. Otherwise, it is said that the automaton *rejects* the string. The set of strings *M* accepts is the language *recognized* by *M* and this language is denoted by *L (M)*.

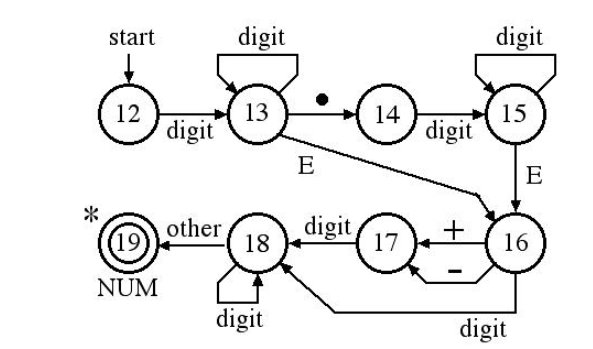
A deterministic finite automaton without accept states and without a starting state is known as a transition system or semi automaton.

**DFA Algorithm**

1. S:= S0 ;
2. C : =first character of input;
3. move (S,C) gives state to which transition occurs when current state is S and input is C;.
4. while C != eof do

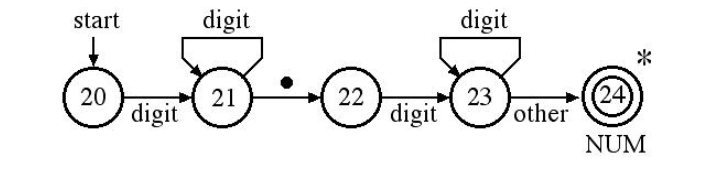
                        S : = move ( S,C);  
                       C : = next\_char of input.  
           end;  
     5. If S is in F (set of accepting states)  
         then return "YES";  
         else return "NO";  
**Recognition of Tokens**

One way to recognize a token is with a finite state automaton following a particular transition diagram. The recognition of the NUM token of Pascal can be done with three different transition diagrams. The first diagram accepts real numbers with exponents:

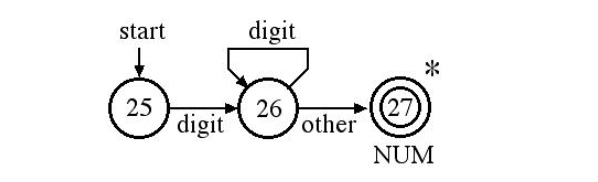
****

If the first diagram fails to reach its accept state then we should try the second diagram which

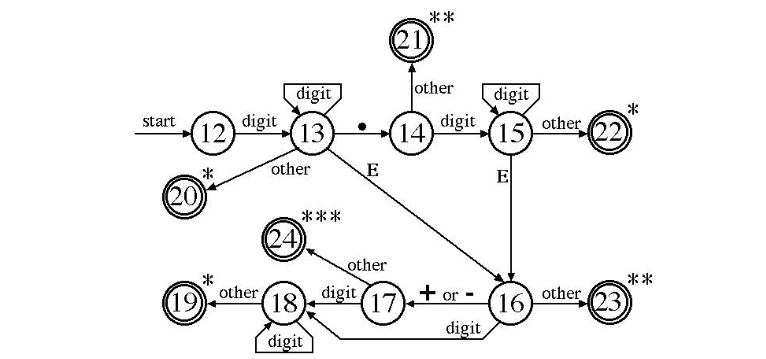
accepts real numbers with fractions but no exponents:

****

If the second diagram also fails then we should try the third diagram which accepts integers:



One can combine the three diagrams for unsigned numbers into one diagram but some of the accept states in this diagram have multiple stars to indicate that two or more characters must be pushed back onto the input:

****

The number is real if state 15 or 18 (or both) is encountered by the transition - if neither state 15 nor state 18 is encountered then the number is an integer.

**Program :**

#include<ctype.h>

#include<stdio.h>

#define MAX 100

int transit(int,char);

int main()

{

int cstate=0,n,i;

static char \*inp,input[MAX];

printf("enter the string and its length\n");

scanf("%s %d",&input,&n);

//n=sizeof(input);

inp=input;

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

{

cstate=transit(cstate,\*inp);

printf("%d\n",cstate);

inp++;

}

if(cstate==1||cstate==4||cstate==6)

printf("string is accepted\n");

else

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

return 0;

}

int transit(int r,char ch)

{

int transtab[8][5]={{1,7,7,7,7},

{1,7,2,3,7},

{4,7,7,7,7},

{7,5,7,7,7},

{4,7,7,3,7},

{6,7,7,7,7},

{6,7,7,7,7},

{7,7,7,7,7}},col;

if(isdigit(ch))

{

col=0;

}

else

{ if(ch=='.')

col=2;

else

{

if(ch=='E')

col=3;

else

{

if(ch=='+' || ch=='-')

col=1;

else

{

col=4;

}

}

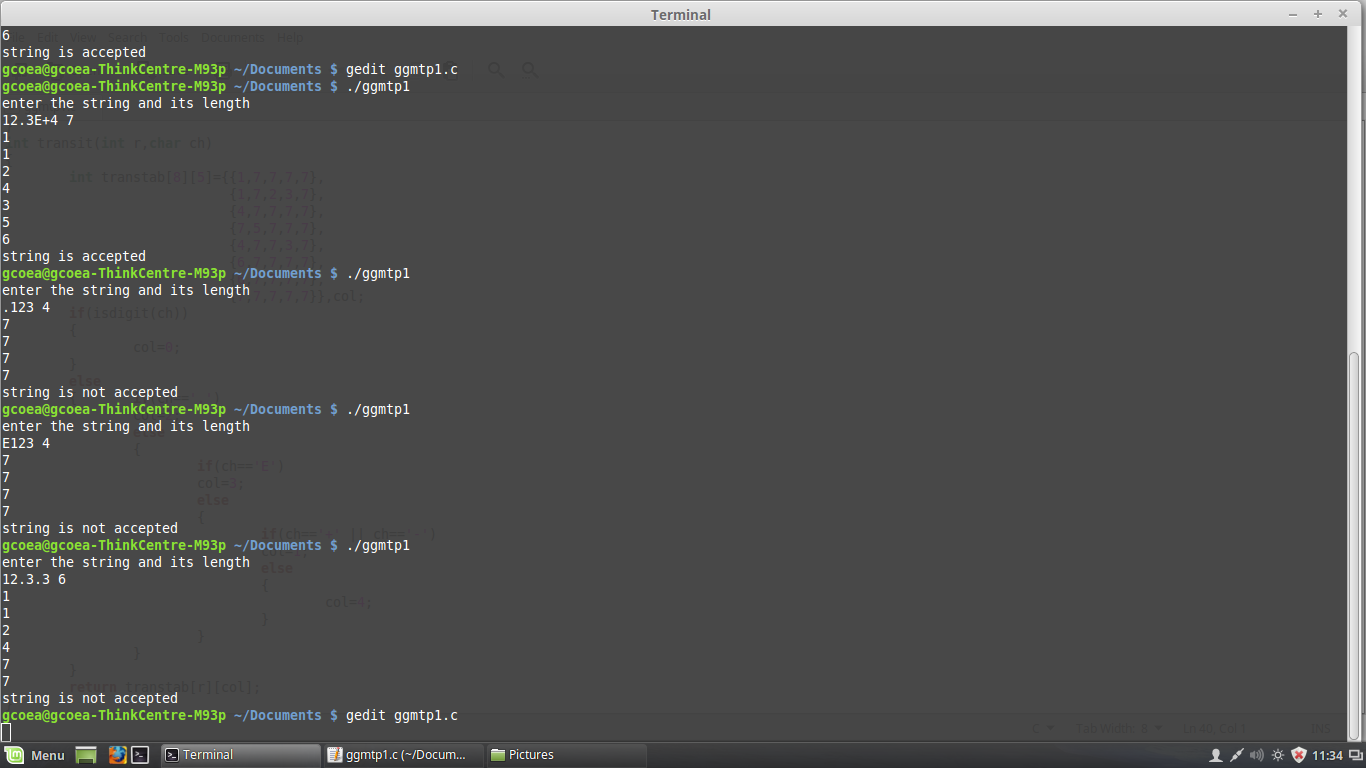
}

}

return transtab[r][col];

}

**Output :**

****

**Conclusion :**  The concept of DFA and regular expressions has been understood and a simulation on the basis of the same, that accepts a string of the given pattern has been made.

**Viva questions:-**

1) Which of the following is same as the given DFA?  
[22-q1](http://www.sanfoundry.com/wp-content/uploads/2016/07/22-q1.png)  
a*) (0+1)\*001(0+1)\** b) 1\*001(0+1)\* c) (01)\*(0+0+1)(01)\* d) None of these

2) The total number of states required to automate the given regular expression (00)\*(11)\*  
a) 3 b) 4 *c) 5* d) 6

3) Which of the given regular expressions correspond to the automata shown?

[22-q4](http://www.sanfoundry.com/wp-content/uploads/2016/07/22-q4.png)

a) (110+1)\*0 b) (11+110)\*1 *c) (110+11)\*0* d)(1+110)\*1

4)The minimum number of transitions to pass to reach the final state as per the following regular expression is:{a,b}\*{baaa}  
*a) 4* b) 5 c) 6 d) 3

5)Generate a regular expression for the following problem statement:P(x): String of length 6 or less for å={0,1}\*  
a) (1+0+e)6 b) (10)6 c) (1+0)(1+0)(1+0)(1+0)(1+0)(1+0) d) More than one of the mentioned is correct 