CS19001/CS19002 PROGRAMMING AND DATA STRUCTURES LABORATORY
Assignment No: 5
Last Date of Submission: 09–March–2015

A *finite automaton* (plural *automata*) is a simple machine which consists of:

(1) A *finite control* consisting of finitely many states (denoted as 0, 1, …, $n - 1$).
(2) An *input tape* consisting of a sequence of 0's and 1's, delimited by an end-of-input marker.
(3) A *transition function* that, given an input state and an input symbol, evaluates to an output state.
(4) The state 0 is marked as the *start state*.
(5) Some of the states is/are designated as *final states*.

The machine starts in the start state, and keeps on reading symbols from the input tape. Its state changes according to its transition function. When the input is entirely read, the machine *accepts* if it is in a final state, otherwise it *rejects*.

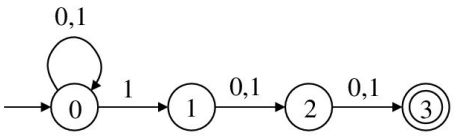
Depending upon the transition function, a finite automaton can be implemented. In a *deterministic finite automaton* (*DFA*), there is a unique output state for every input state and input symbol. This is covered in the tutorial.

In this programming exercise, your task is to extend the code for DFA to work for *NFA* (*non-deterministic finite automata*). In an NFA, the value of the transition function, for a given input state and input symbol, need not be unique. There may be multiple—even zero—possibilities. You need to determine whether some choices of the transfer-function possibilities leads from the start state (before any input is read) to a final state (after the entire input is read).

Example 1

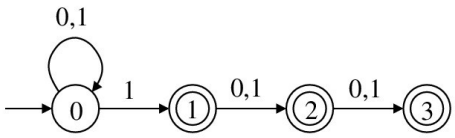
Consider the machine with four states 0, 1, 2, 3 with the only final state 3. The machine has the following transfer function. The machine accepts all patterns with the third last symbol is one.

p	a	$\delta(p,a)$
0	0	0
0	1	0
0	1	1
1	0	2
1	1	2
2	0	3
2	1	3



Example 2

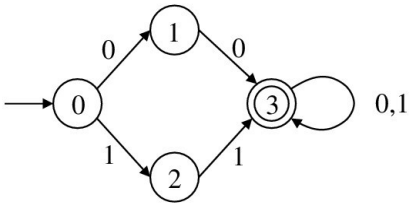
The NFA with four states 0, 1, 2, 3, the final states 1, 2, 3, and the same transition table as in Example 1, accepts all patterns in which at least one 1 is read among the last three symbols.



Example 3

Now consider a machine with four states 0, 1, 2, 3, the only final state 3, and the following transition function:

p	a	$\delta(p,a)$
0	0	1
0	1	2
1	0	3
2	1	3
3	0	3
3	1	3



The machine accepts all patterns whose first two symbols are the same (that is, those that start with 00 or 11).

Part 1

As in the tutorial program, define a structure *rule* to store the triple (p, a, q), where q is one possibility for $\delta(p,a)$. Also define an NFA as a structure of the following items: the number n of states, an array f [ ] to store the information which states are final, the number m of transfer-function rules, and an array T [ ] of m structures of type *rule*. Write a function to populate and return a machine data type M as follows:

(1) First read the number of states
(2) The user then enters the final states one by one (–1 to terminate)
(3) The user finally enters the transition rules (p, a, q) one the by one (p = –1 terminates the reading loop)

The user then runs the NFA on one or more inputs. Each input is read as a sequence of 0's and 1's. If –1 is entered as the next symbol, the input ends. All the symbols read (including the trailing –1) are stored in an array. For each input, run the functions of Parts 2 and 3.

Part 2

Write an iterative function to determine all the possibilities that the machine can reach starting from the start state 0 and after consuming the entire input. If any one of the final possibilities is a final state, output *accept*. If none of the final possibilities is a final state, output *reject*. Maintain an array A of size n (the number of states). $A[i]$ is meant to store whether it is possible to reach state i. Initially, only state 0 is reachable. In each iteration, modify the array A[ ] (in fact, modify a copy, and write back to A). After the entire input is read, check whether any of the final states is reachable.

Part 3

Write a recursive function to determine whether an NFA accepts an input string. Write a recursive function that takes four arguments: the description of the machine M as specified in Part 1, the input array A[ ], and the index i in A[ ] from which to read next, and the current state p of the machine. If $A[i]$ stores –1, then we have reached the end of input. Return *accept* or *reject* depending upon whether p is a final state or not. Otherwise, consult the transfer-function table of M. For *each* possibility of $\delta(p, A[i])$, make a recursive call. (There may be no possibilities.) If any of the recursive calls accepts, accept. If all the recursive calls reject, reject.

Submit only one C source file solving all the three parts.

Sample Output

```
Enter the number of states: 4
The states will be numbered 0 to 3
The start state is 0
Enter the final states one by one (-1 to terminate)
3 -1
Enter the rules one by one
Enter -1 as input state to terminate
0 0 0
0 1 0
0 1 1
1 0 2
1 1 2
2 0 3
2 1 3
-1
Do you want to continue (0/1)? 1
Enter the input symbols one by one (-1 to terminate)
0 1 0 0 0 0 1 -1
+++ Iterative function outputs REJECT
+++ Recursive function outputs REJECT
Do you want to continue (0/1)? 1
Enter the input symbols one by one (-1 to terminate)
0 1 0 0 1 0 1 -1
+++ Iterative function outputs ACCEPT
+++ Recursive function outputs ACCEPT
Do you want to continue (0/1)? 1
Enter the input symbols one by one (-1 to terminate)
-1
+++ Iterative function outputs REJECT
+++ Recursive function outputs REJECT
Do you want to continue (0/1)? 0
```