**COSC 385.001 – Automata**

**Spring 2018**

**Project #1**

**Deadline: Wednesday, February 14, 2018, 3:00pm**

**Problem-**

**Jose Dixon**

**Vojislav Stojkovic**

**2/15/2018**

**Points: 15**

**Part 2. Description of Text/Problem**

Names of mathematical functions of the C programming language.

// acos, asin, atan, atan2, ceil, cos, cosh, exp, fabs, floor, fmod, frexp, ldexp, log, log10, modf, pow, sin, sinh, sqrt, etc.

// For more information see <math.h>

**Part 3. Finite Automaton**

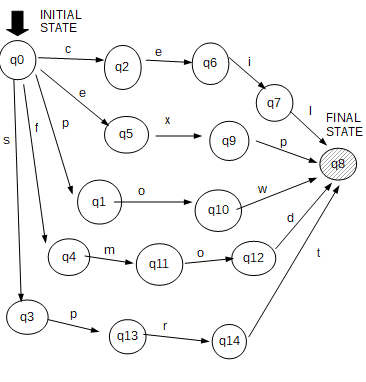
M= {Q, ∑, ∆, q0, F}

Q = {q0, q1, q2, q3, q4, q5, q6,q7, q8, q9, q10, q11, q12, q13, q14}

∑ = {c, e, i, l, p, o, w, s, q, r, t, f, m, d, x}

q0 = q0

F = q8

****

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ∆/∑ | c | e | i | l | p | o | w | s | q | r | t | f | m | d | x |
| Q0 | Q2 | Q5 |  |  | Q1 |  |  | Q3 |  |  |  | Q4 |  |  |  |
| Q1 |  |  |  |  |  | Q10 |  |  |  |  |  |  |  |  |  |
| Q2 |  | Q6 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Q3 |  |  |  |  |  |  |  |  | Q13 |  |  |  |  |  |  |
| Q4 |  |  |  |  |  |  |  |  |  |  |  |  | Q11 |  |  |
| Q5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Q9 |
| Q6 |  |  | Q7 |  |  |  |  |  |  |  |  |  |  |  |  |
| Q7 |  |  |  | Q8 |  |  |  |  |  |  |  |  |  |  |  |
| Q8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Q9 |  |  |  |  | Q8 |  |  |  |  |  |  |  |  |  |  |
| Q10 |  |  |  |  |  |  | Q8 |  |  |  |  |  |  |  |  |
| Q11 |  |  |  |  |  | Q12 |  |  |  |  |  |  |  |  |  |
| Q12 |  |  |  |  |  |  |  |  |  |  |  |  |  | Q8 |  |
| Q13 |  |  |  |  |  |  |  |  |  | Q14 |  |  |  |  |  |
| Q14 |  |  |  |  |  |  |  |  |  |  | Q8 |  |  |  |  |

**Part 4. Explanations and Algorithms**

The term "Automata" is derived from the Greek word "αὐτόματα" which means "self-acting". An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

An automaton with a finite number of states is called a **Finite Automaton**(FA) or **Finite State Machine** (FSM).

### **Formal definition of a Finite Automaton**

An automaton can be represented by a 5-tuple (Q, ∑, δ, q0, F), where −

* **Q** is a finite set of states.
* **∑** is a finite set of symbols, called the **alphabet** of the automaton.
* **δ** is the transition function.
* **q0** is the initial state from where any input is processed (q0 ∈ Q).
* **F** is a set of final state/states of Q (F ⊆ Q).

## Related Terminologies

### **Alphabet**

* **Definition** − An **alphabet** is any finite set of symbols.
* **Example** − ∑ = {a, b, c, d} is an **alphabet set** where ‘a’, ‘b’, ‘c’, and ‘d’ are **symbols**.

### **String**

* **Definition** − A **string** is a finite sequence of symbols taken from ∑.
* **Example** − ‘cabcad’ is a valid string on the alphabet set ∑ = {a, b, c, d}

### **Length of a String**

* **Definition** − It is the number of symbols present in a string. (Denoted by **|S|**).
* **Examples** −
  + If S = ‘cabcad’, |S|= 6
  + If |S|= 0, it is called an **empty string** (Denoted by **λ** or **ε**)

### **Kleene Star**

* **Definition** − The Kleene star, **∑\***, is a unary operator on a set of symbols or strings, **∑**, that gives the infinite set of all possible strings of all possible lengths over **∑** including **λ**.
* **Representation** − ∑\* = ∑0 ∪ ∑1 ∪ ∑2 ∪……. where ∑p is the set of all possible strings of length p.
* **Example** − If ∑ = {a, b}, ∑\* = {λ, a, b, aa, ab, ba, bb,………..}

### **Kleene Closure / Plus**

* **Definition** − The set **∑+** is the infinite set of all possible strings of all possible lengths over ∑ excluding λ.
* **Representation** − ∑+ = ∑1 ∪ ∑2 ∪ ∑3 ∪…….

∑+ = ∑\* − { λ }

* **Example** − If ∑ = { a, b } , ∑+ = { a, b, aa, ab, ba, bb,………..}

### **Language**

* **Definition** − A language is a subset of ∑\* for some alphabet ∑. It can be finite or infinite.
* **Example** − If the language takes all possible strings of length 2 over ∑ = {a, b}, then L = { ab, bb, ba, bb}

**Part 5. Program Code**

class DFA:

current\_state = None;

def \_\_init\_\_(self, states, alphabet, transition\_function, start\_state, accept\_states):

self.states = states;

self.alphabet = alphabet;

self.transition\_function = transition\_function;

self.start\_state = start\_state;

self.accept\_states = accept\_states;

self.current\_state = start\_state;

return;

def transition\_to\_state\_with\_input(self, input\_value):

if ((self.current\_state, input\_value) not in self.transition\_function.keys()):

self.current\_state = None;

return;

self.current\_state = self.transition\_function[(self.current\_state, input\_value)];

return;

def in\_accept\_state(self):

return self.current\_state in accept\_states;

def go\_to\_initial\_state(self):

self.current\_state = self.start\_state;

return;

def run\_with\_input\_list(self, input\_list):

self.go\_to\_initial\_state();

for inp in input\_list:

self.transition\_to\_state\_with\_input(inp);

continue;

return self.in\_accept\_state();

pass;

states = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14};

alphabet = {'c', 'e', 'i', 'l', 'p', 'o', 'w', 's', 'q', 'r', 't', 'f', 'm', 'd', 'x'};

tf = dict();

#0

tf[(0, 'c')] = 2;

tf[(0, 'e')] = 5;

tf[(0, 'i')] = 0;

tf[(0, 'l')] = 0;

tf[(0, 'p')] = 1;

tf[(0, 'o')] = 0;

tf[(0, 'w')] = 0;

tf[(0, 's')] = 3;

tf[(0, 'q')] = 0;

tf[(0, 'r')] = 0;

tf[(0, 't')] = 0;

tf[(0, 'f')] = 4;

tf[(0, 'm')] = 0;

tf[(0, 'd')] = 0;

tf[(0, 'x')] = 0;

#1

tf[(1, 'c')] = 0;

tf[(1, 'e')] = 0;

tf[(1, 'i')] = 0;

tf[(1, 'l')] = 0;

tf[(1, 'p')] = 0;

tf[(1, 'o')] = 10;

tf[(1, 'w')] = 0;

tf[(1, 's')] = 0;

tf[(1, 'q')] = 0;

tf[(1, 'r')] = 0;

tf[(1, 't')] = 2;

tf[(1, 'f')] = 0;

tf[(1, 'm')] = 0;

tf[(1, 'd')] = 0;

tf[(1, 'x')] = 0;

#2

tf[(2, 'c')] = 0;

tf[(2, 'e')] = 6;

tf[(2, 'i')] = 0;

tf[(2, 'l')] = 8;

tf[(2, 'p')] = 0;

tf[(2, 'o')] = 3;

tf[(2, 'w')] = 0;

tf[(2, 's')] = 0;

tf[(2, 'q')] = 0;

tf[(2, 'r')] = 0;

tf[(2, 't')] = 2;

tf[(2, 'f')] = 0;

tf[(2, 'm')] = 0;

tf[(2, 'd')] = 0;

tf[(2, 'x')] = 0;

#3

tf[(3, 'c')] = 0;

tf[(3, 'e')] = 0;

tf[(3, 'i')] = 0;

tf[(3, 'l')] = 0;

tf[(3, 'p')] = 0;

tf[(3, 'o')] = 0;

tf[(3, 'w')] = 0;

tf[(3, 's')] = 0;

tf[(3, 'q')] = 13;

tf[(3, 'r')] = 0;

tf[(3, 't')] = 0;

tf[(3, 'f')] = 0;

tf[(3, 'm')] = 0;

tf[(3, 'd')] = 0;

tf[(3, 'x')] = 0;

#4

tf[(4, 'c')] = 0;

tf[(4, 'e')] = 0;

tf[(4, 'i')] = 0;

tf[(4, 'l')] = 0;

tf[(4, 'p')] = 0;

tf[(4, 'o')] = 0;

tf[(4, 'w')] = 0;

tf[(4, 's')] = 0;

tf[(4, 'q')] = 0;

tf[(4, 'r')] = 0;

tf[(4, 't')] = 0;

tf[(4, 'f')] = 0;

tf[(4, 'm')] = 11;

tf[(4, 'd')] = 0;

tf[(4, 'x')] = 0;

#5

tf[(5, 'c')] = 0;

tf[(5, 'e')] = 0;

tf[(5, 'i')] = 0;

tf[(5, 'l')] = 0;

tf[(5, 'p')] = 0;

tf[(5, 'o')] = 0;

tf[(5, 'w')] = 0;

tf[(5, 's')] = 0;

tf[(5, 'q')] = 0;

tf[(5, 'r')] = 0;

tf[(5, 't')] = 0;

tf[(5, 'f')] = 0;

tf[(5, 'm')] = 0;

tf[(5, 'd')] = 0;

tf[(5, 'x')] = 9;

#6

tf[(6, 'c')] = 0;

tf[(6, 'e')] = 0;

tf[(6, 'i')] = 7;

tf[(6, 'l')] = 0;

tf[(6, 'p')] = 0;

tf[(6, 'o')] = 0;

tf[(6, 'w')] = 0;

tf[(6, 's')] = 0;

tf[(6, 'q')] = 0;

tf[(6, 'r')] = 0;

tf[(6, 't')] = 0;

tf[(6, 'f')] = 0;

tf[(6, 'm')] = 0;

tf[(6, 'd')] = 0;

tf[(6, 'x')] = 0;

#7

tf[(7, 'c')] = 0;

tf[(7, 'e')] = 0;

tf[(7, 'i')] = 0;

tf[(7, 'l')] = 8;

tf[(7, 'p')] = 0;

tf[(7, 'o')] = 0;

tf[(7, 'w')] = 0;

tf[(7, 's')] = 0;

tf[(7, 'q')] = 0;

tf[(7, 'r')] = 0;

tf[(7, 't')] = 0;

tf[(7, 'f')] = 0;

tf[(7, 'm')] = 0;

tf[(7, 'd')] = 0;

tf[(7, 'x')] = 0;

#8

tf[(8, 'c')] = 0;

tf[(8, 'e')] = 0;

tf[(8, 'i')] = 0;

tf[(8, 'l')] = 0;

tf[(8, 'p')] = 0;

tf[(8, 'o')] = 0;

tf[(8, 'w')] = 0;

tf[(8, 's')] = 0;

tf[(8, 'q')] = 0;

tf[(8, 'r')] = 0;

tf[(8, 't')] = 0;

tf[(8, 'f')] = 0;

tf[(8, 'm')] = 0;

tf[(8, 'd')] = 0;

tf[(8, 'x')] = 0;

#9

tf[(9, 'c')] = 0;

tf[(9, 'e')] = 0;

tf[(9, 'i')] = 0;

tf[(9, 'l')] = 0;

tf[(9, 'p')] = 8;

tf[(9, 'o')] = 0;

tf[(9, 'w')] = 0;

tf[(9, 's')] = 0;

tf[(9, 'q')] = 0;

tf[(9, 'r')] = 0;

tf[(9, 't')] = 0;

tf[(9, 'f')] = 0;

tf[(9, 'm')] = 0;

tf[(9, 'd')] = 0;

tf[(9, 'x')] = 0;

#10

tf[(10, 'c')] = 0;

tf[(10, 'e')] = 0;

tf[(10, 'i')] = 0;

tf[(10, 'l')] = 0;

tf[(10, 'p')] = 0;

tf[(10, 'o')] = 0;

tf[(10, 'w')] = 8;

tf[(10, 's')] = 0;

tf[(10, 'q')] = 0;

tf[(10, 'r')] = 0;

tf[(10, 't')] = 0;

tf[(10, 'f')] = 0;

tf[(10, 'm')] = 0;

tf[(10, 'd')] = 0;

tf[(10, 'x')] = 0;

#11

tf[(11, 'c')] = 0;

tf[(11, 'e')] = 0;

tf[(11, 'i')] = 0;

tf[(11, 'l')] = 0;

tf[(11, 'p')] = 0;

tf[(11, 'o')] = 12;

tf[(11, 'w')] = 0;

tf[(11, 's')] = 0;

tf[(11, 'q')] = 0;

tf[(11, 'r')] = 0;

tf[(11, 't')] = 0;

tf[(11, 'f')] = 0;

tf[(11, 'm')] = 0;

tf[(11, 'd')] = 0;

tf[(11, 'x')] = 0;

#12

tf[(12, 'c')] = 0;

tf[(12, 'e')] = 0;

tf[(12, 'i')] = 0;

tf[(12, 'l')] = 0;

tf[(12, 'p')] = 0;

tf[(12, 'o')] = 0;

tf[(12, 'w')] = 0;

tf[(12, 's')] = 0;

tf[(12, 'q')] = 0;

tf[(12, 'r')] = 0;

tf[(12, 't')] = 0;

tf[(12, 'f')] = 0;

tf[(12, 'm')] = 0;

tf[(12, 'd')] = 8;

tf[(12, 'x')] = 0;

#13

tf[(13, 'c')] = 1;

tf[(13, 'e')] = 0;

tf[(13, 'i')] = 0;

tf[(13, 'l')] = 0;

tf[(13, 'p')] = 0;

tf[(13, 'o')] = 0;

tf[(13, 'w')] = 0;

tf[(13, 's')] = 0;

tf[(13, 'q')] = 0;

tf[(13, 'r')] = 14;

tf[(13, 't')] = 0;

tf[(13, 'f')] = 0;

tf[(13, 'm')] = 0;

tf[(13, 'd')] = 0;

tf[(13, 'x')] = 0;

#14

tf[(14, 'c')] = 0;

tf[(14, 'e')] = 0;

tf[(14, 'i')] = 0;

tf[(14, 'l')] = 0;

tf[(14, 'p')] = 0;

tf[(14, 'o')] = 0;

tf[(14, 'w')] = 0;

tf[(14, 's')] = 0;

tf[(14, 'q')] = 0;

tf[(14, 'r')] = 0;

tf[(14, 't')] = 8;

tf[(14, 'f')] = 0;

tf[(14, 'm')] = 0;

tf[(14, 'd')] = 0;

tf[(14, 'x')] = 0;

start\_state = 0;

accept\_states = {8};

d = DFA(states, alphabet, tf, start\_state, accept\_states);

test = raw\_input("Input a key word ")

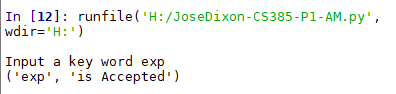
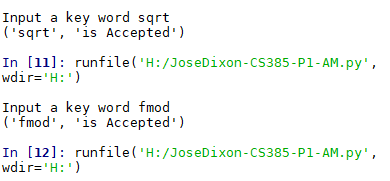
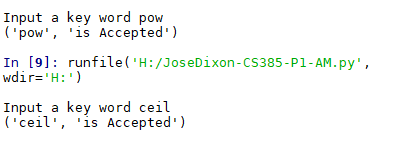
if d.run\_with\_input\_list(test) is True:

print(test, 'is Accepted')

else:

print(test, "is Rejected")

**Part 6. Test Examples/Output**

****