Modulo Three Exercise

You may choose to attempt either the standard or the advanced option depending on your level of comfort with the exercise. Please complete this project in the language of your choice, and include unit tests to verify the accuracy. Here is the rubric we use to evaluate responses to the exercise.

Introduction

Consider a string of ones and zeros representing an unsigned binary integer. Let us implement a modulo-three function that will compute the remainder when the represented value is divided by three.

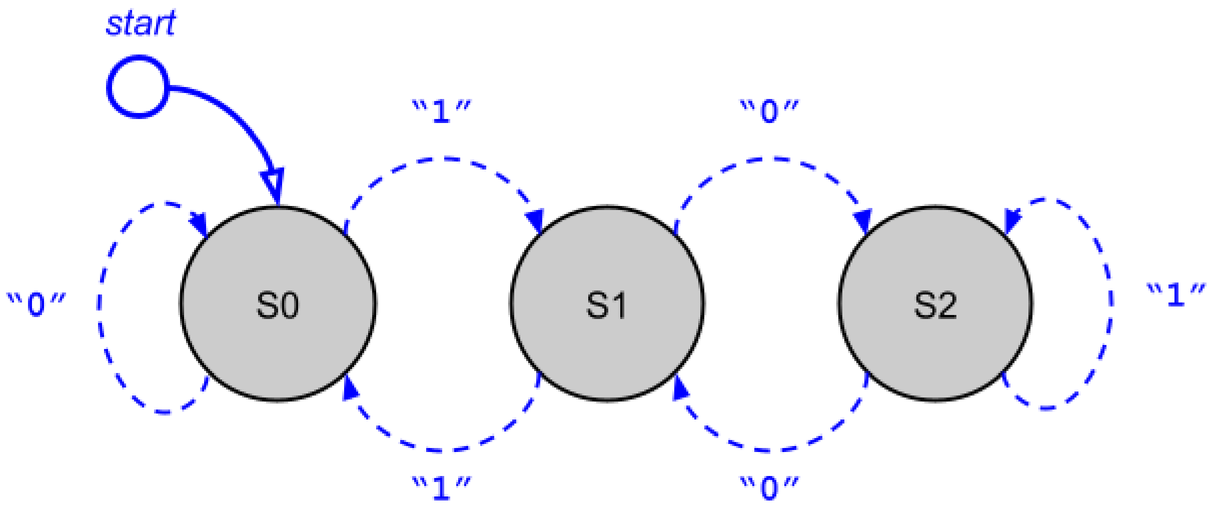
```
function modThree(input: string) : number
{
    …
}

// Applied to binary string representing thirteen
modThree("1101")
=> 1
// fourteen
modThree("1110")
=> 2
// fifteen
modThree("1111")
=> 0
```

Example setup using TypeScript syntax

One way to implement this would be to convert the input string to a number type and use the modulus operator (%). While that approach will produce the correct answer, for this exercise we wish you to use a more efficient and far more interesting method derived from the world of computer hardware:

Finite State Machine (FSM)

Let us build an FSM that takes the input characters, one at a time, MOST significant bit first and transitions between a set of states as specified by the following state transition diagram:



The value that is returned from our function will depend on the state which is selected after the character sequence has been exhausted. The final state will be converted to a remainder value as specified in the following table:

Final State	Output
S0	0
S1	1
S2	2

Please choose **one** exercise option below to complete, either "Standard" **or** "Advanced".

For senior technical roles like Developer III or Senior/Staff/Principal Developer, the Advanced exercise is recommended.

Standard Exercise

With the help of the examples below, implement a state transition algorithm to implement the 'mod-three' procedure. Your solution should be well tested and demonstrate good programming practices.

Example 1

For input string `"110"` the machine will operate as follows:

1. Initial state = S0, Input = 1, result state = S1
2. Current state = S1, Input = 1, result state = S0
3. Current state = S0, Input = 0, result state = S0
4. No more input

5. Print output value (output for state S0 = 0) <---- This is the answer

Example 2

For input string `"1010"` the machine will operate as follows:

1. Initial state = S0, Input = 1, result state = S1
2. Current state = S1, Input = 0, result state = S2
3. Current state = S2, Input = 1, result state = S2
4. Current state = S2, Input = 0, result state = S1
5. No more input
6. Print output value (output for state S1 = 1) <---- This is the answer

Advanced Exercise

The FSM described above is an example of finite state automata. With the help of the abstract definition below, create a software module for generating an FSM. The API of your library should be designed for use by other developers. Implement the 'mod-three' procedure as an example.

Finite Automation

A finite automaton (FA) is a 5-tuple $(Q,\Sigma,q0,F,\delta)$ where,

Q is a finite set of states;
Σ is a finite input alphabet;
$q0 \in Q$ is the initial state;
$F \subseteq Q$ is the set of accepting/final states; and
$\delta:Q\times\Sigma\rightarrow Q$ is the transition function.

For any element q of Q and any symbol $\sigma\in\Sigma$, we interpret $\delta(q,\sigma)$ as the state to which the FA moves, if it is in state q and receives the input σ.

Mod-Three FA

Based on the notation from the definition, our modulo three FSM would be configured as follows:

$Q = (S0, S1, S2)$
$\Sigma = (0, 1)$
$q0 = S0$
$F = (S0, S1, S2)$
$\delta(S0,0) = S0$; $\delta(S0,1) = S1$; $\delta(S1,0) = S2$; $\delta(S1,1) = S0$; $\delta(S2,0) = S1$; $\delta(S2,1) = S2$