DFA Machine Design Documentation

Introduction:

- Automata theory:

Automata theory deals with the definitions and properties of mathematical models of computation. These models play a role in several applied areas of computer science.

One model, called the finite automaton, is used in text processing, compilers, and hardware design.

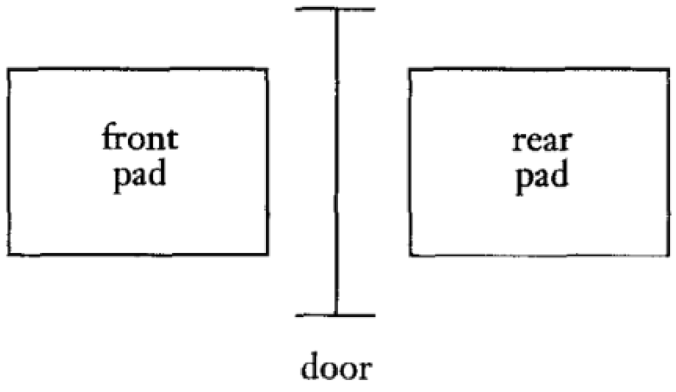
Another model, called context-free grammar, is used in programming languages and artificial intelligence.

Automata theory is an excellent place to begin the study of the theory of computation. Automata theory allows practice with formal definitions of computation as it introduces concepts relevant to other non-theoretical areas of computer science.

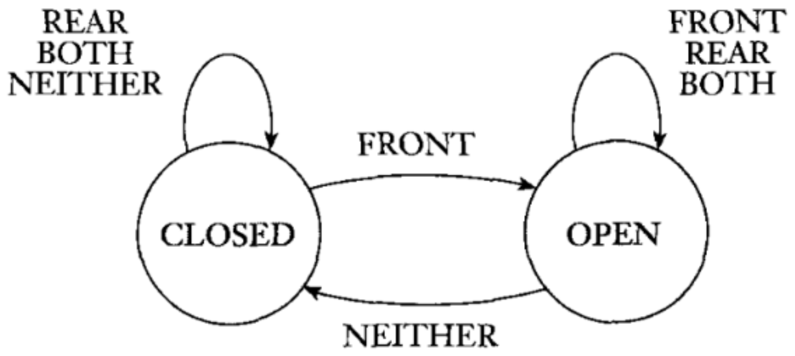
- DFA machines:

Finite automata are good models for computers with an extremely limited amount of memory. What can a computer do with such a small memory? Many useful things! In fact, we interact with such computers all the time, as they lie at the heart of various electromechanical devices.

The controller for an automatic door is one example of such a device. Often found at supermarket entrances and exits, automatic doors swing open when sensing that a person is approaching. An automatic door has a pad in front to detect the presence of a person about to walk through the doorway. Another pad is located to the rear of the doorway so that the controller can hold the door open long enough for the person to pass all the way through and also so that the door does not strike someone standing behind it as it opens. This configuration is shown in the following figure.



The controller is in either of two states: "OPEN" or "CLOSED," representing the corresponding condition of the door. As shown in the following figures, there are four possible input conditions: "FRONT" (meaning that a person is standing on the pad in front of the doorway), "REAR" (meaning that a person is standing on the pad to the rear of the doorway), "BOTH" (meaning that people are standing on both pads), and "NEITHER" (meaning that no one is standing on either pad).



MachineDesign:

- DFA formal definition:

A finite automaton has several parts. It has a set of states and rules for going

from one state to another, depending on the input symbol. It has an input alphabet that indicates the allowed input symbols. It has a start state and a set of

accept states. The formal definition says that a finite automaton is a list of those

five objects: set of states, input alphabet, rules for moving, start state, and accept

states. In mathematical language a list of five elements is often called a 5-tuple.

Hence we define a finite automaton to be a 5-tuple consisting of these five parts.

We use something called a transition function, frequently denoted 3, to define the rules for moving. If the finite automaton has an arrow from a state x to

a state y labeled with the input symbol 1, that means that, if the automaton is

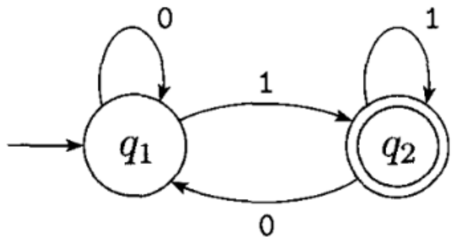
in state x when it reads a 1, it then moves to state y. We can indicate the same

thing with the transition function by saying that $\delta(x, 1) = y$. This notation is a

kind of mathematical shorthand. Putting it all together we arrive at the formal definition of finite automata.

- **Examples of DFA:**

Let's consider a finite automaton M_1. the following figure shows the state diagram of DFA M_1.



In the formal description $M_1 = (\{ q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$.

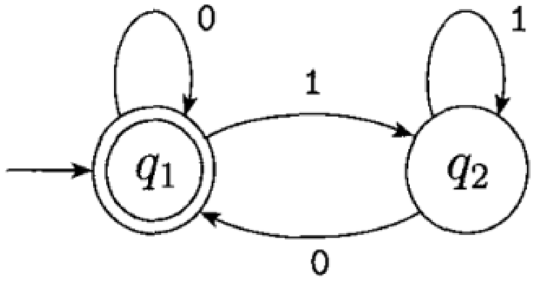
The transition function δ is:

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

On the sample string 1101 the machine M_1 starts in its start state q_1 and proceeds first to state q_2 after reading the first 1, and then to states q_2, q_1, and q_2 after reading 1, 0, and 1. The string is accepted because q_2 is an accept state. But string 110 leaves M_1 in state q_1, so it is rejected. After trying a few more examples, you would see that M_1 accepts all strings that end in a 1. Thus

$L(M_1) = \{ \omega \mid \omega \text{ ends in a } 1 \}$.

Consider the finite automaton M_2:


$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[loop above, "0"] \arrow[r, bend left, "1"] & q_2 \arrow[loop above, "1"] \arrow[l, bend left, "0"]
\end{tikzcd}$$

Machine M_2 is similar to M_1 except for the location of the accept state. As usual, the machine accepts all strings that leave it in an accept state when it has finished reading. Note that, because the start state is also an accept state, M_2 accepts the empty string ε. As soon as a machine begins reading the empty string it is at the end, so if the start state is an accept state, ε is accepted. In addition to the empty string, this machine accepts any string ending with a 0.
Here,
$L(M_2) = \{ \omega \mid \omega$ is the empty string ε or ends in a 0}.

Machine Implementation:

Explanation:

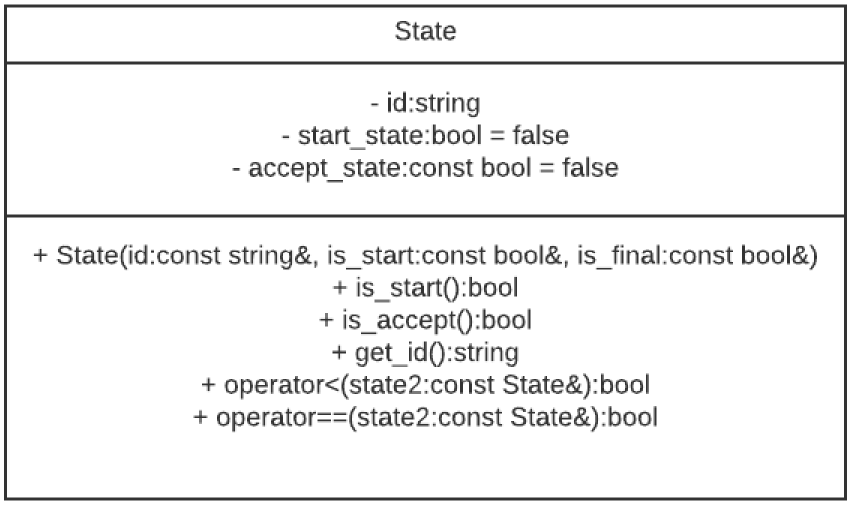
To understand how the application is actually implemented let's start with the object oriented design (OOD) and the idea behind it. From the formal definition we knew that the DFA machine should be described with the following:

- States
    - including the start state and the set of accept states.
- Alphabet
- Transitions

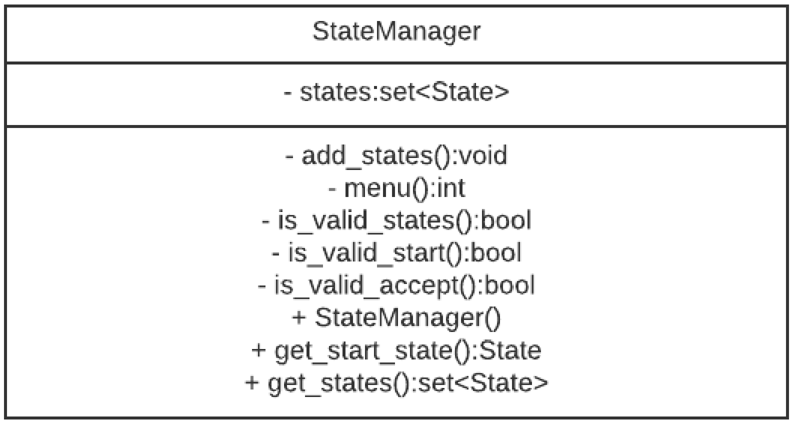
Therefore we can simply construct the DFA machine by getting those as inputs, then read the strings to be processed based on this machine. Moreover, to apply the OOP concepts and principles we shouldn’t only create a class for each of them with the correct attributes and methods but also create additional classes to manage them and manage the overall system in a clean way.

UML Class Diagram:

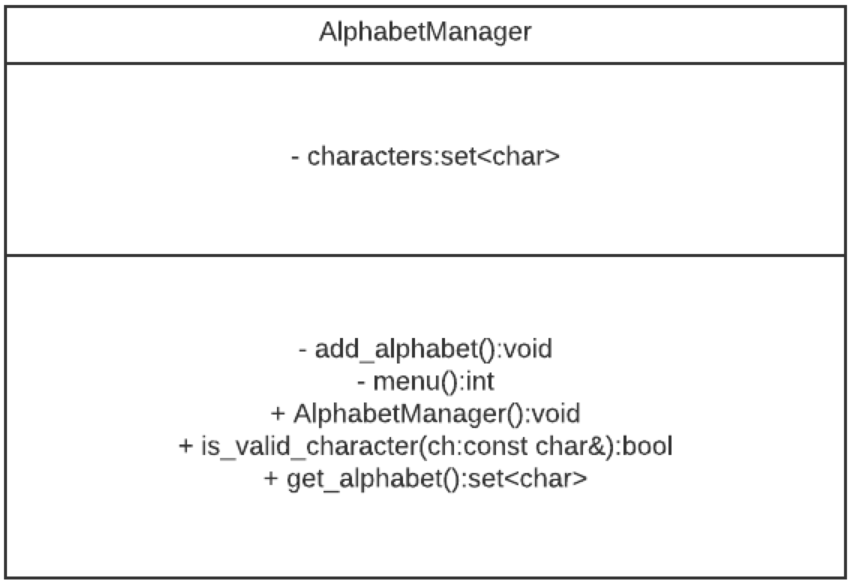
State:


State
- id:string
- start_state:bool = false
- accept_state:const bool = false
+ State(id:const string&, is_start:const bool&, is_final:const bool&)
+ is_start():bool
+ is_accept():bool
+ get_id():string
+ operator<(state2:const State&):bool
+ operator==(state2:const State&):bool

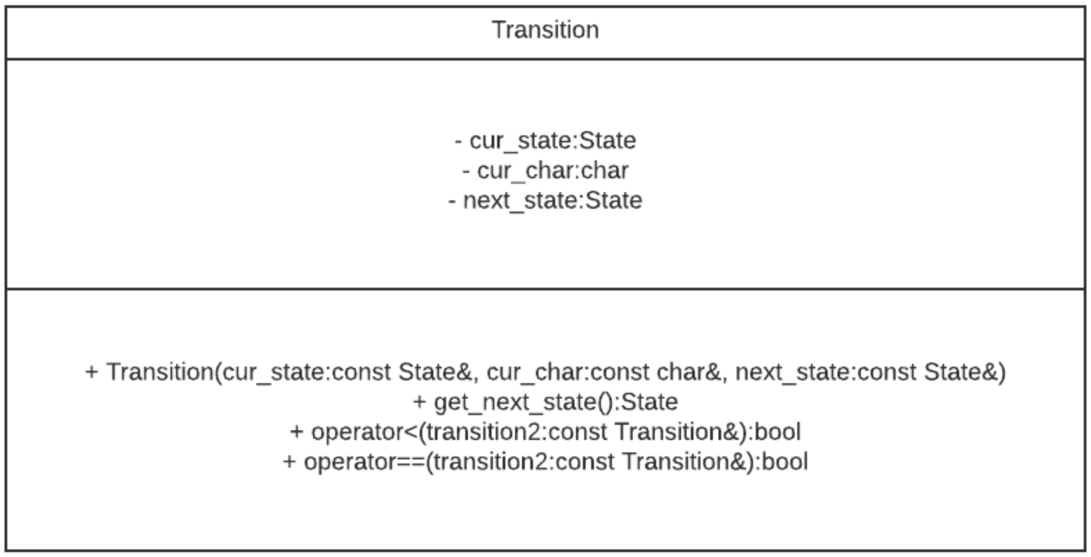
StateManager:



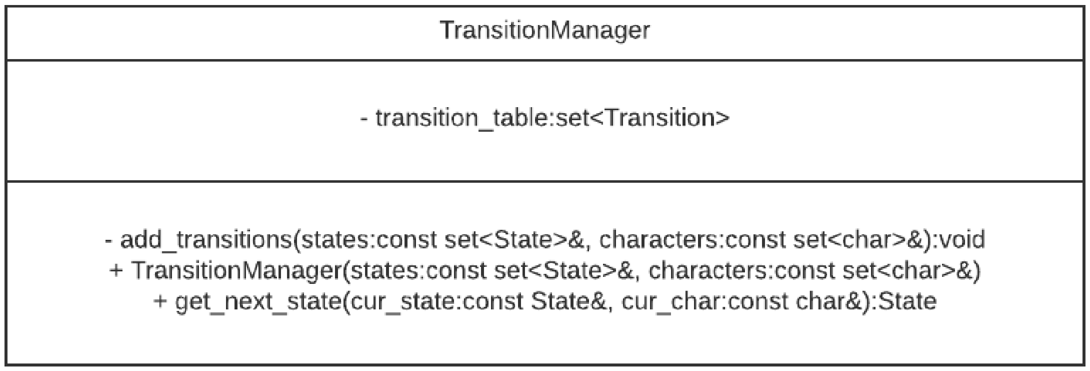
AlphabetManager:



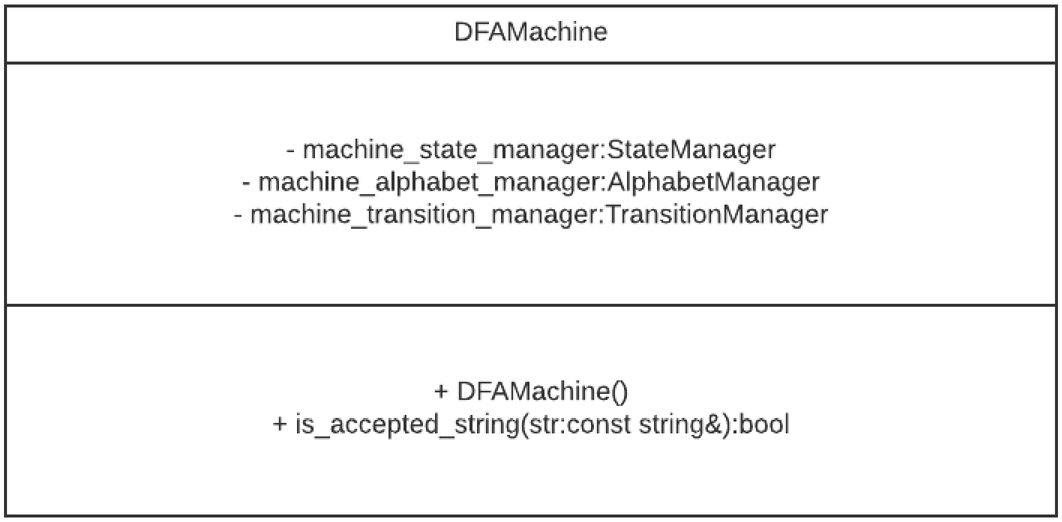
Transition:


Transition
- cur_state:State
- cur_char:char
- next_state:State
+ Transition(cur_state:const State&, cur_char:const char&, next_state:const State&)
+ get_next_state():State
+ operator<(transition2:const Transition&):bool
+ operator==(transition2:const Transition&):bool

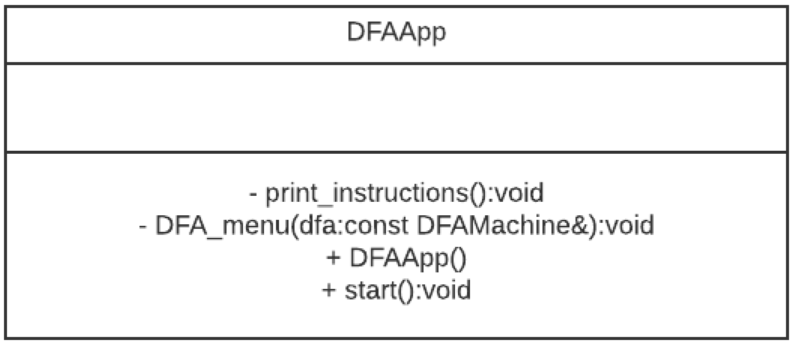
TransitionManager:


TransitionManager
- transition_table:set<Transition>
- add_transitions(states:const set<State>&, characters:const set<char>&):void
+ TransitionManager(states:const set<State>&, characters:const set<char>&)
+ get_next_state(cur_state:const State&, cur_char:const char&):State

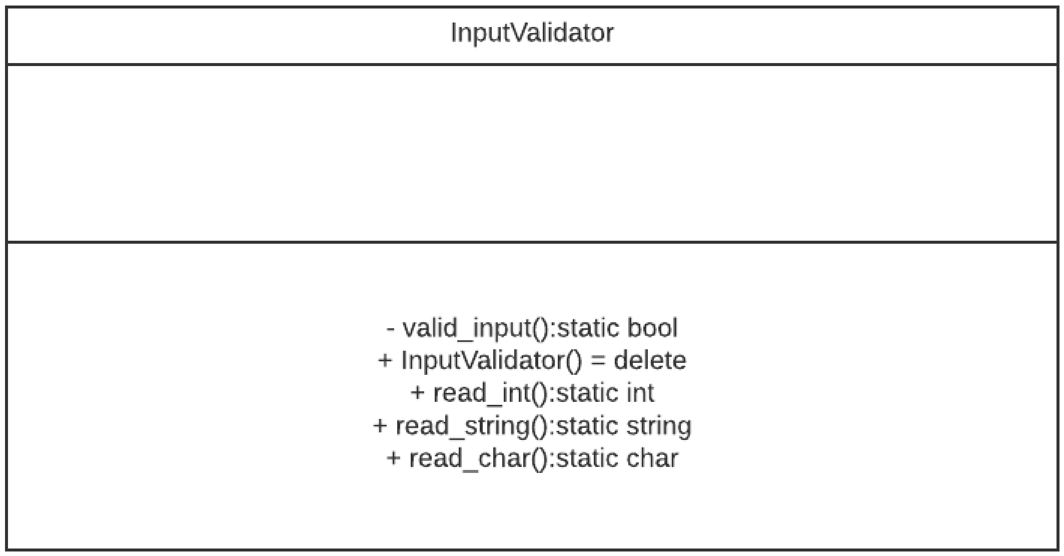
DFAMachine:


DFAMachine
- machine_state_manager:StateManager
- machine_alphabet_manager:AlphabetManager
- machine_transition_manager:TransitionManager
+ DFAMachine()
+ is_accepted_string(str:const string&):bool

DFAApp:

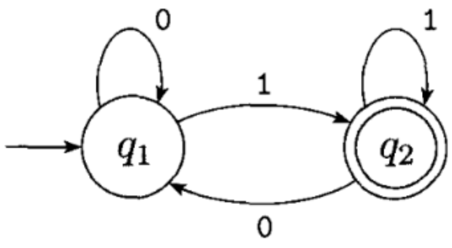

DFAApp
- print_instructions():void
- DFA_menu(dfa:const DFAMachine&):void
+ DFAApp()
+ start():void

InputValidator:

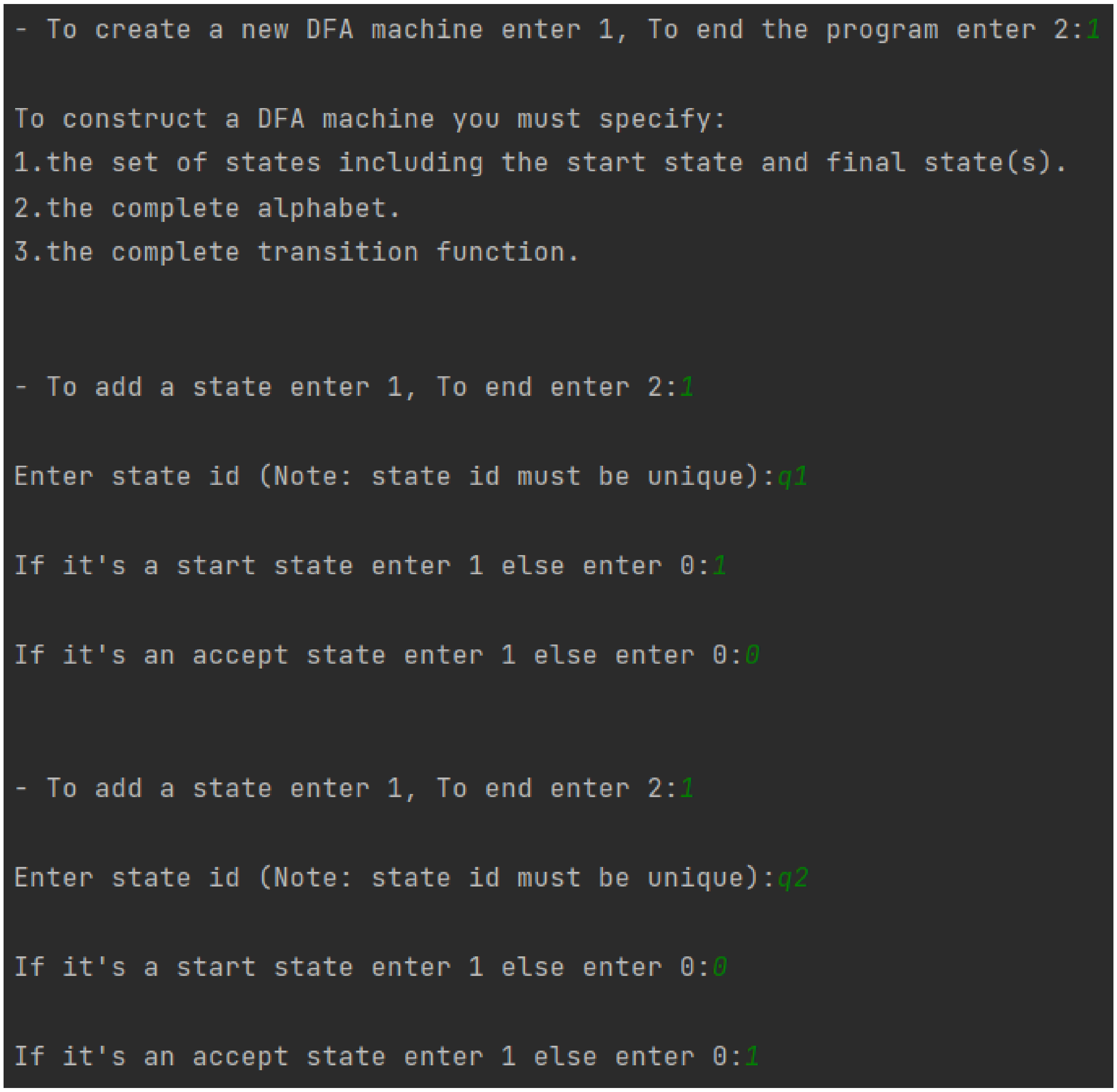

InputValidator
- valid_input():static bool
+ InputValidator() = delete
+ read_int():static int
+ read_string():static string
+ read_char():static char

Samples:

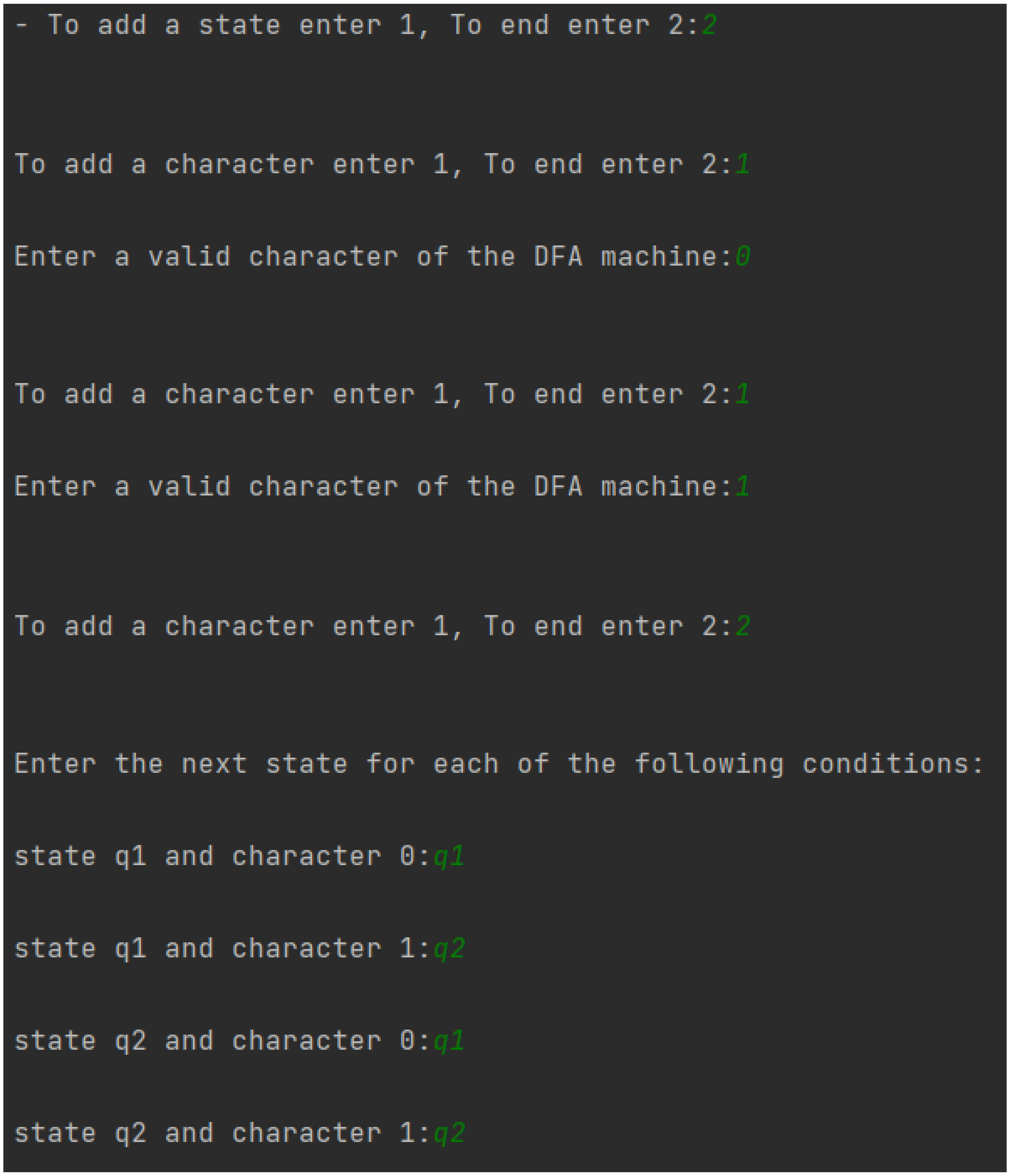
- **Machine 1:**



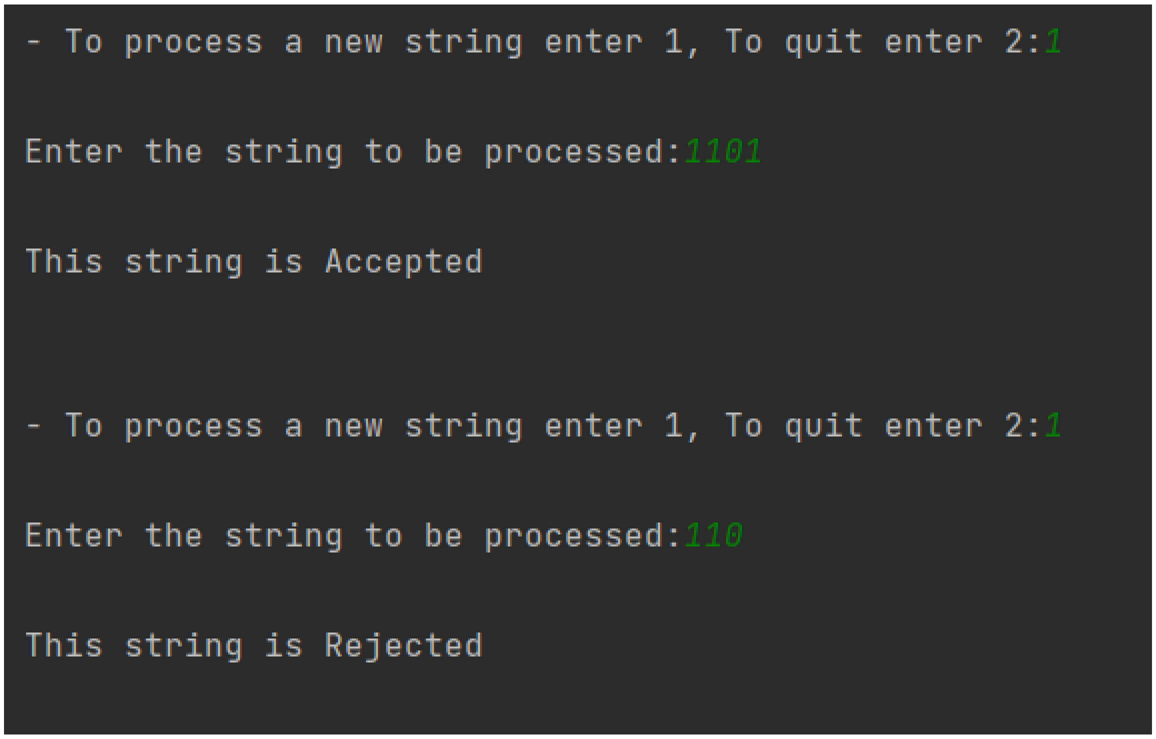
$$\begin{tikzcd} \arrow[r] & q_1 \arrow[loop above, "0"] \arrow[r, bend left, "1"] & q_2 \arrow[loop above, "1"] \arrow[l, bend left, "0"] \end{tikzcd}$$



```
- To create a new DFA machine enter 1, To end the program enter 2:1

To construct a DFA machine you must specify:
1.the set of states including the start state and final state(s).
2.the complete alphabet.
3.the complete transition function.


- To add a state enter 1, To end enter 2:1

Enter state id (Note: state id must be unique):q1

If it's a start state enter 1 else enter 0:1

If it's an accept state enter 1 else enter 0:0


- To add a state enter 1, To end enter 2:1

Enter state id (Note: state id must be unique):q2

If it's a start state enter 1 else enter 0:0

If it's an accept state enter 1 else enter 0:1
```

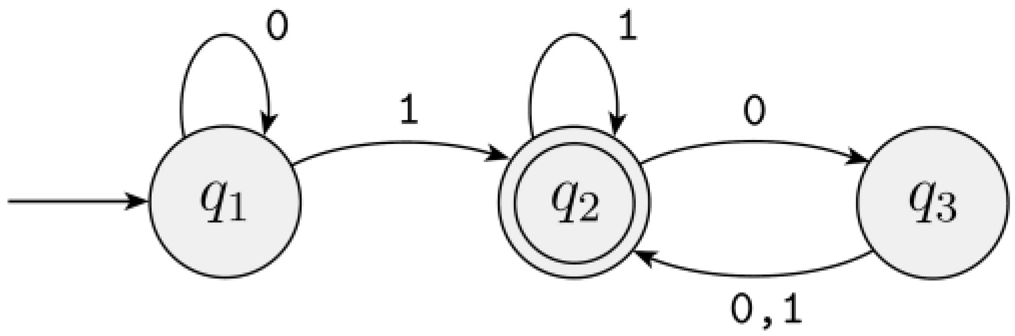


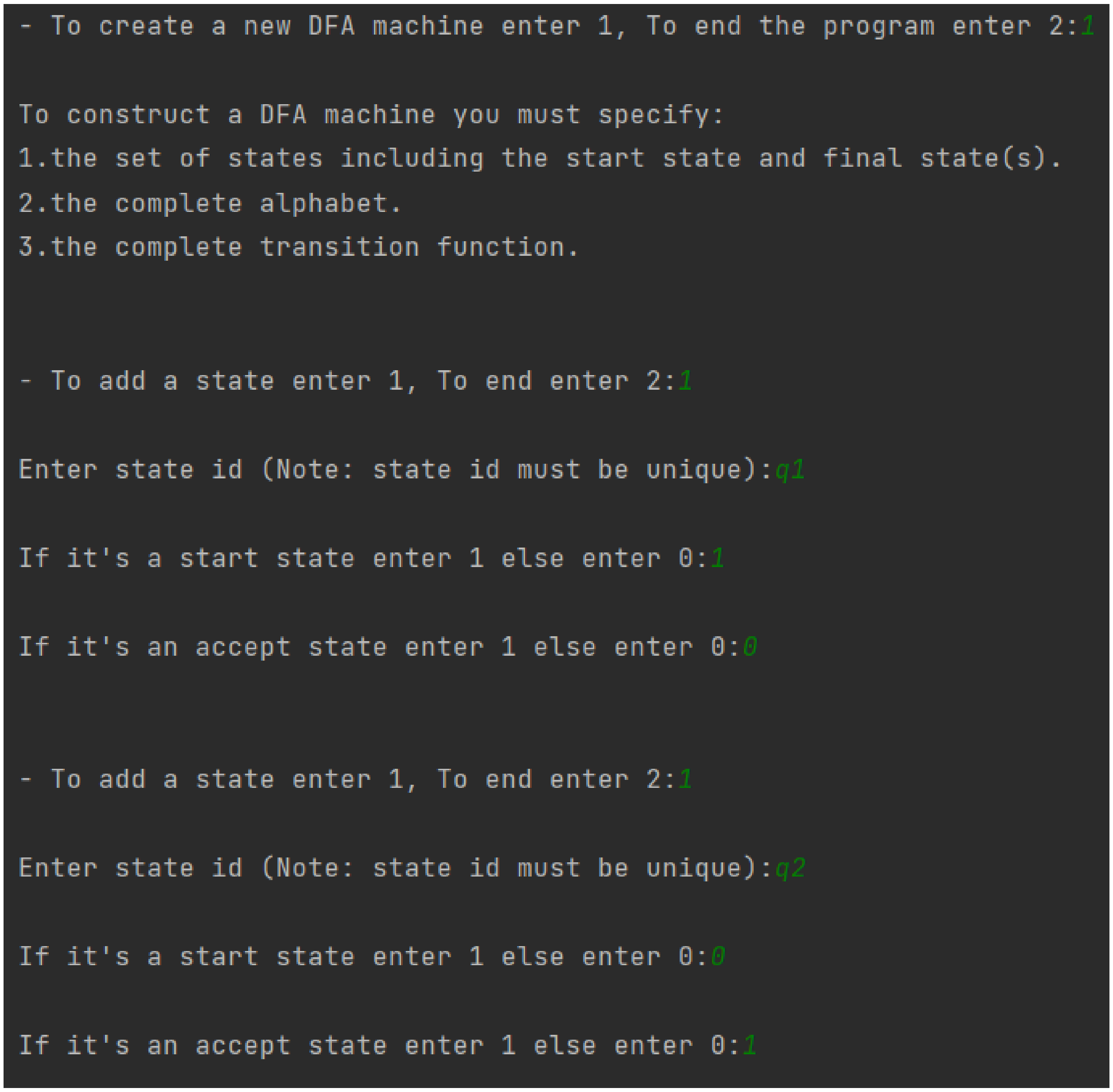
```
- To add a state enter 1, To end enter 2:2


To add a character enter 1, To end enter 2:1

Enter a valid character of the DFA machine:0


To add a character enter 1, To end enter 2:1

Enter a valid character of the DFA machine:1


To add a character enter 1, To end enter 2:2


Enter the next state for each of the following conditions:

state q1 and character 0:q1

state q1 and character 1:q2

state q2 and character 0:q1

state q2 and character 1:q2
```

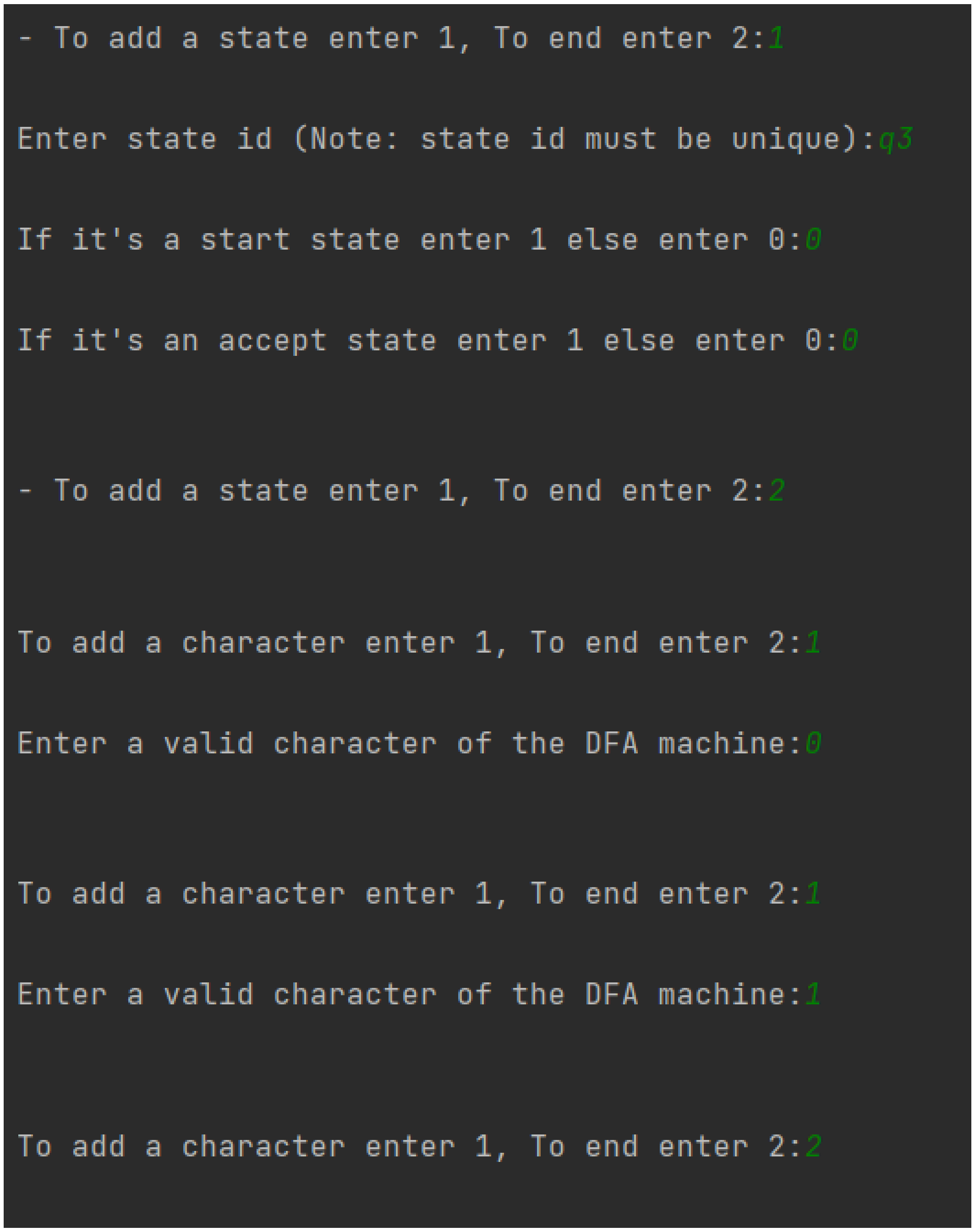


```
- To process a new string enter 1, To quit enter 2:1

Enter the string to be processed:1101

This string is Accepted


- To process a new string enter 1, To quit enter 2:1

Enter the string to be processed:110

This string is Rejected
```

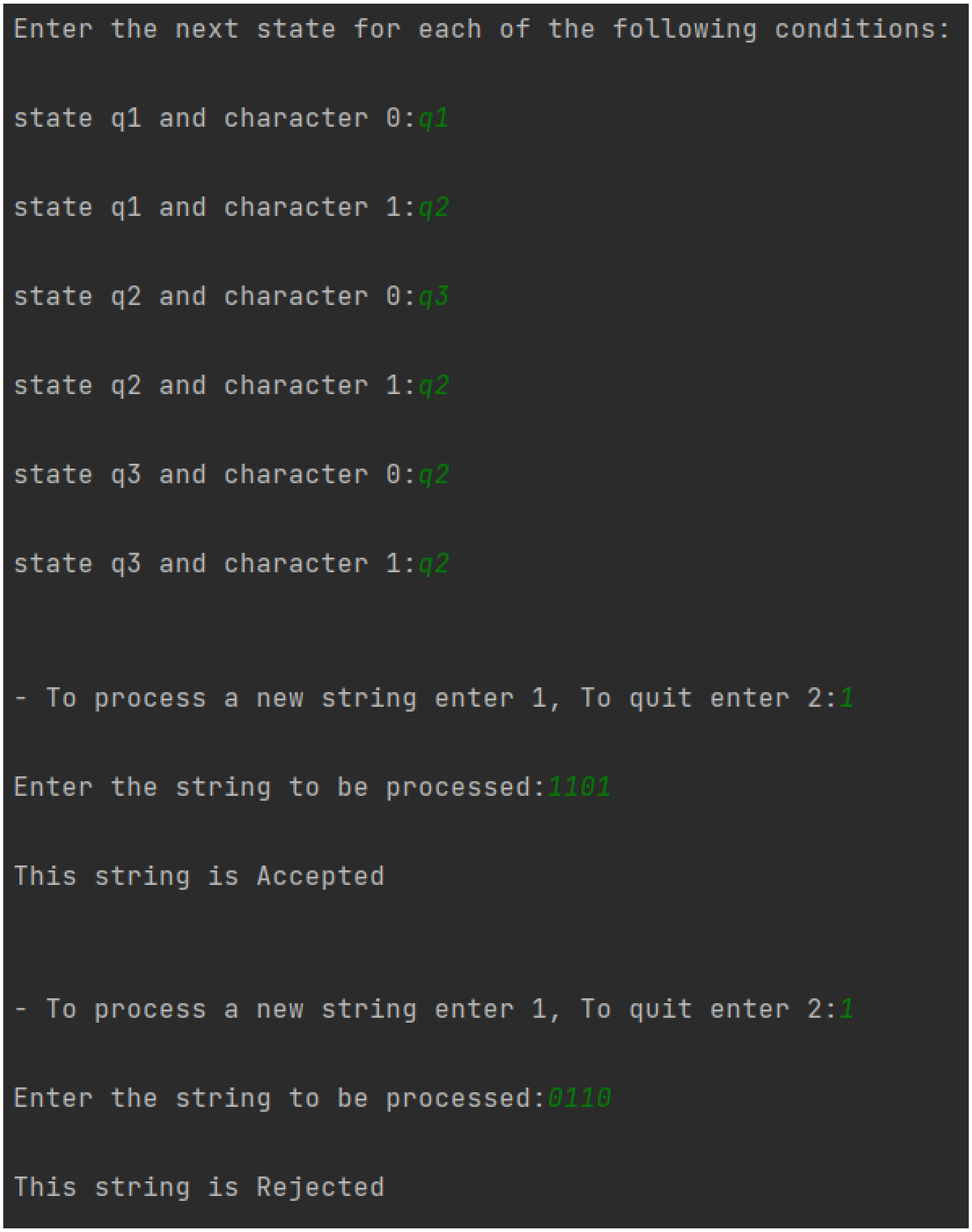
- **Machine 2:**


$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[loop above, "0"] \arrow[r, bend left, "1"] & q_2 \arrow[loop above, "1"] \arrow[r, bend left, "0"] & q_3 \arrow[l, bend left, "{0,1}"]
\end{tikzcd}$$


```
- To create a new DFA machine enter 1, To end the program enter 2:1

To construct a DFA machine you must specify:
1.the set of states including the start state and final state(s).
2.the complete alphabet.
3.the complete transition function.


- To add a state enter 1, To end enter 2:1

Enter state id (Note: state id must be unique):q1

If it's a start state enter 1 else enter 0:1

If it's an accept state enter 1 else enter 0:0


- To add a state enter 1, To end enter 2:1

Enter state id (Note: state id must be unique):q2

If it's a start state enter 1 else enter 0:0

If it's an accept state enter 1 else enter 0:1
```

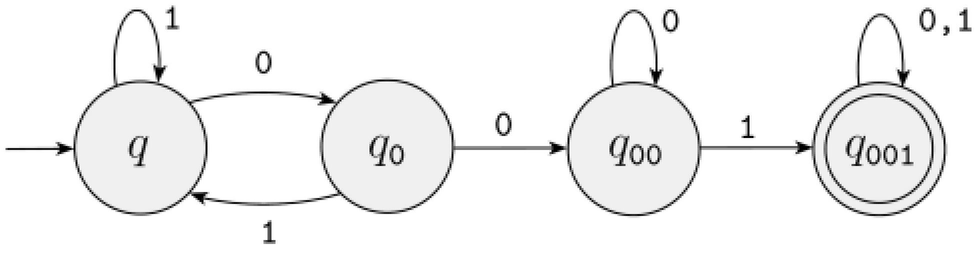


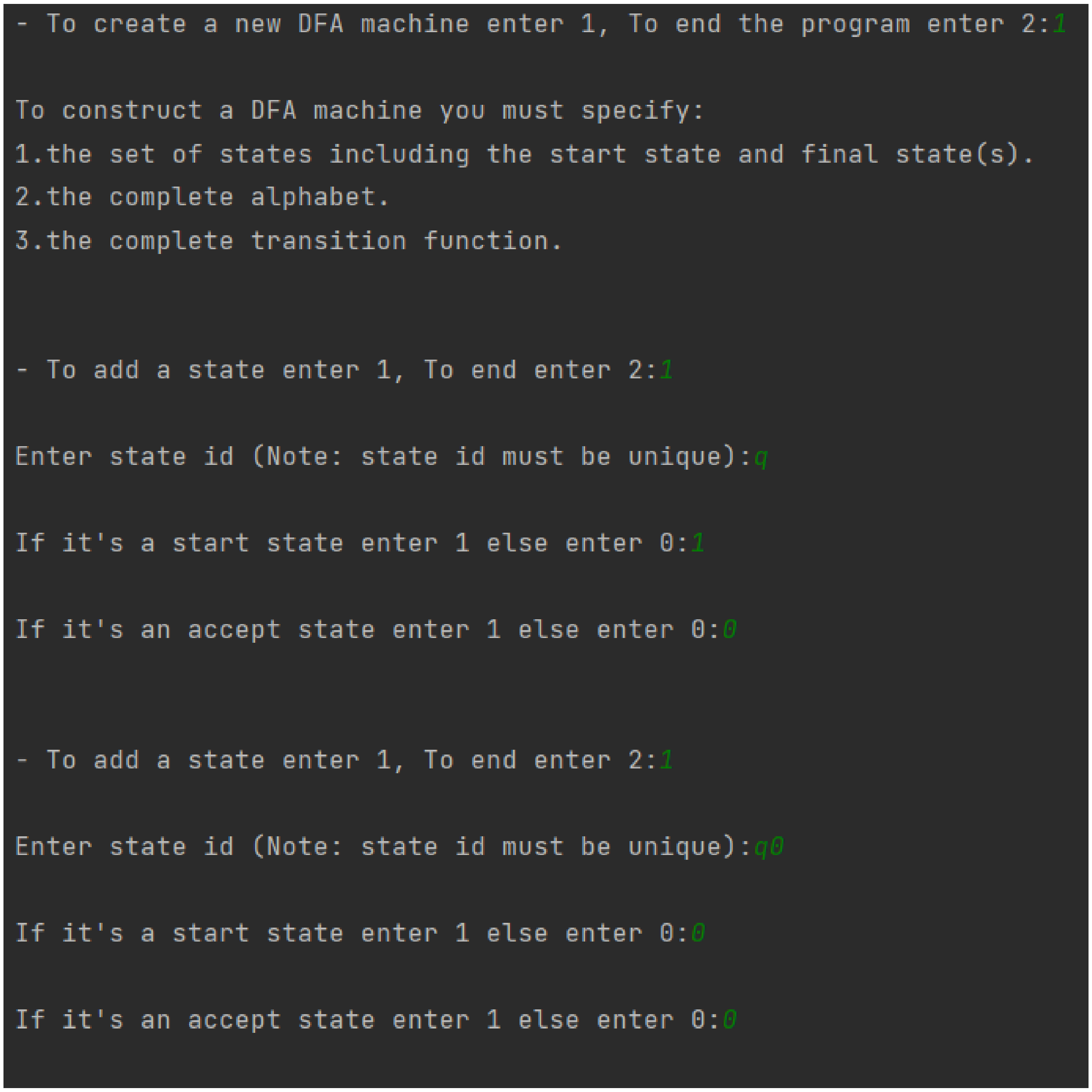
```
- To add a state enter 1, To end enter 2:1

Enter state id (Note: state id must be unique):q3

If it's a start state enter 1 else enter 0:0

If it's an accept state enter 1 else enter 0:0


- To add a state enter 1, To end enter 2:2


To add a character enter 1, To end enter 2:1

Enter a valid character of the DFA machine:0


To add a character enter 1, To end enter 2:1

Enter a valid character of the DFA machine:1


To add a character enter 1, To end enter 2:2
```



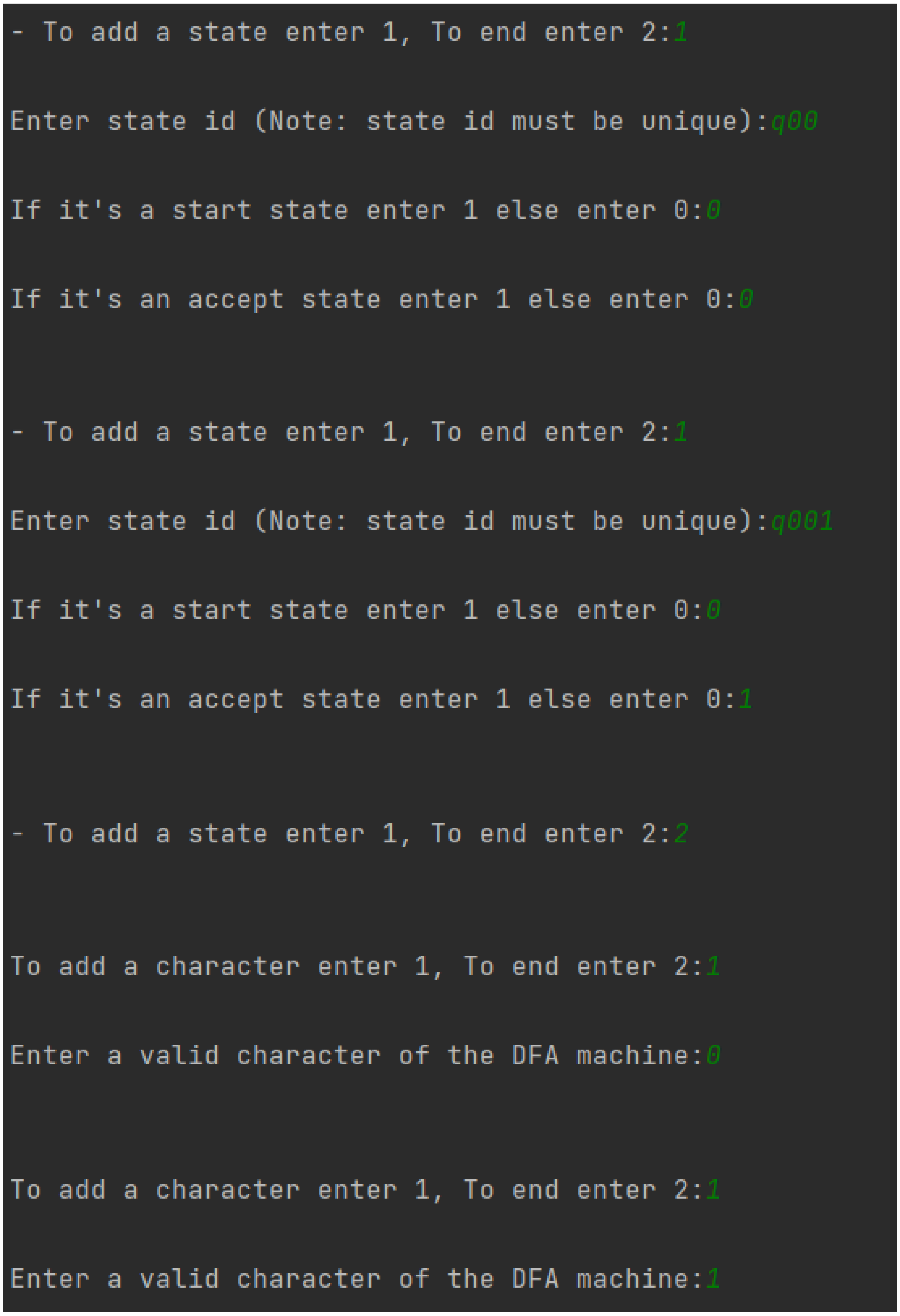
```
Enter the next state for each of the following conditions:

state q1 and character 0:q1

state q1 and character 1:q2

state q2 and character 0:q3

state q2 and character 1:q2

state q3 and character 0:q2

state q3 and character 1:q2


- To process a new string enter 1, To quit enter 2:1

Enter the string to be processed:1101

This string is Accepted


- To process a new string enter 1, To quit enter 2:1

Enter the string to be processed:0110

This string is Rejected
```

- **Machine 3:**

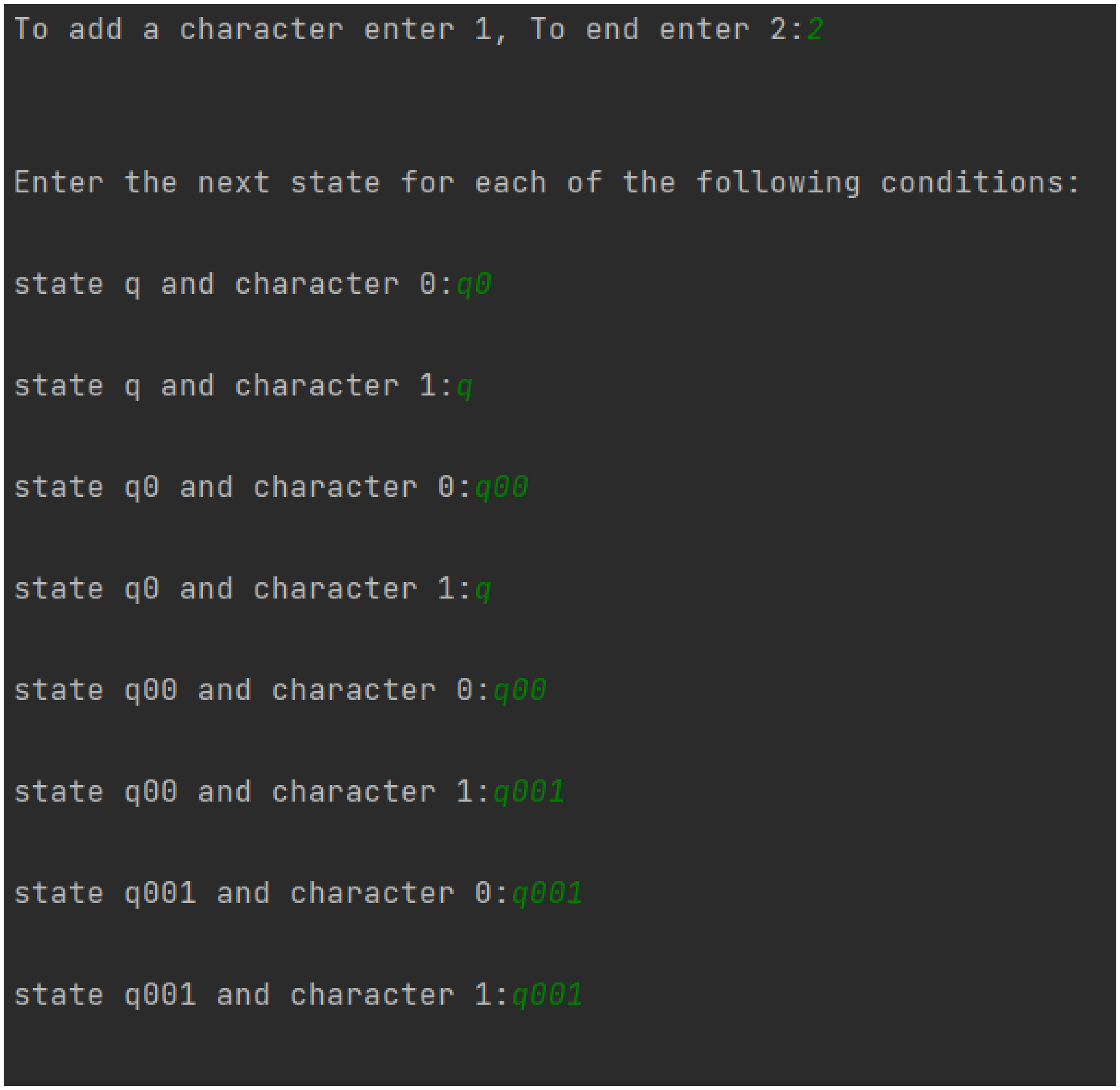




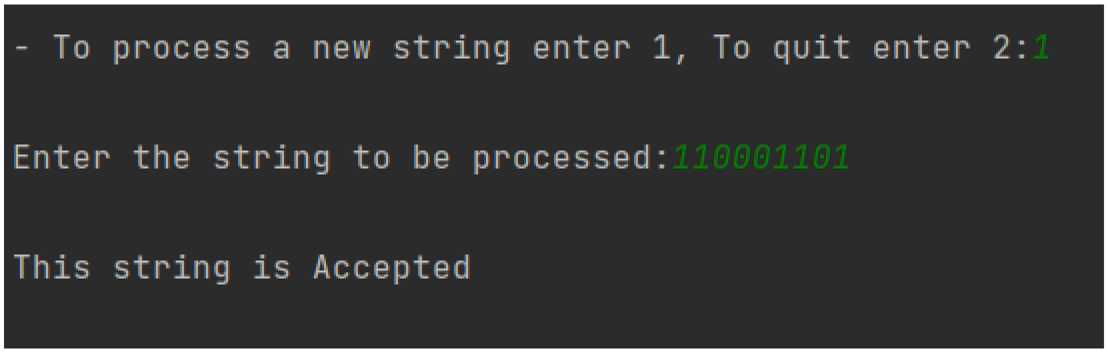
```
- To create a new DFA machine enter 1, To end the program enter 2:1

To construct a DFA machine you must specify:
1.the set of states including the start state and final state(s).
2.the complete alphabet.
3.the complete transition function.



- To add a state enter 1, To end enter 2:1

Enter state id (Note: state id must be unique):q

If it's a start state enter 1 else enter 0:1

If it's an accept state enter 1 else enter 0:0


- To add a state enter 1, To end enter 2:1

Enter state id (Note: state id must be unique):q0

If it's a start state enter 1 else enter 0:0

If it's an accept state enter 1 else enter 0:0
```



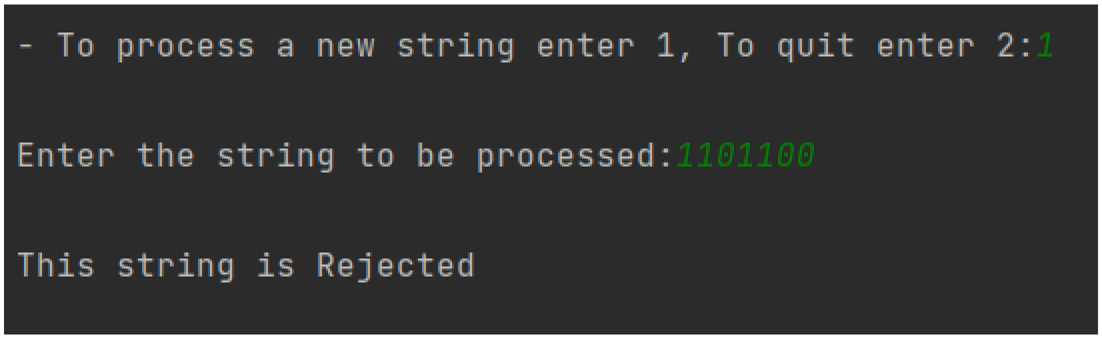
```
- To add a state enter 1, To end enter 2:1

Enter state id (Note: state id must be unique):q00

If it's a start state enter 1 else enter 0:0

If it's an accept state enter 1 else enter 0:0


- To add a state enter 1, To end enter 2:1

Enter state id (Note: state id must be unique):q001

If it's a start state enter 1 else enter 0:0

If it's an accept state enter 1 else enter 0:1


- To add a state enter 1, To end enter 2:2


To add a character enter 1, To end enter 2:1

Enter a valid character of the DFA machine:0


To add a character enter 1, To end enter 2:1

Enter a valid character of the DFA machine:1
```



```
To add a character enter 1, To end enter 2:2


Enter the next state for each of the following conditions:

state q and character 0:q0

state q and character 1:q

state q0 and character 0:q00

state q0 and character 1:q

state q00 and character 0:q00

state q00 and character 1:q001

state q001 and character 0:q001

state q001 and character 1:q001
```



```
- To process a new string enter 1, To quit enter 2:1

Enter the string to be processed:110001101

This string is Accepted
```



```
- To process a new string enter 1, To quit enter 2:1

Enter the string to be processed:1101100

This string is Rejected
```

Appendix:

https://github.com/Mohamed-Sawy/DFA_Machine_App

References:

- Michael Sipser, Introduction to the Theory of Computation, 2nd Edition, Thomson Course Technology, 2006.

- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd Edition, Pearson, 2007.