```
class FiniteAutomaton:
    def __init__(self, states, alphabet, transition_rules, initial_state,
accepting_states):
        self.states = states
        self.alphabet = alphabet
        self.transition_rules = transition_rules
        self.current_state = initial_state
        self.accepting_states = accepting_states
        self.initial_state = initial_state

    def reset(self):
        self.current_state = self.initial_state

    def process_input(self, input_symbol):
        if input_symbol not in self.alphabet:
            raise ValueError(f"Input symbol '{input_symbol}' not in the
alphabet.")

        if self.current_state not in self.states:
            raise ValueError(f"Invalid current state: {self.current_state}")

        if (self.current_state, input_symbol) in self.transition_rules:
            self.current_state = self.transition_rules[(self.current_state,
input_symbol)]
        else:
            raise ValueError(f"No transition rule defined for current state
{self.current_state} and input symbol {input_symbol}")

    def is_accepting(self):
        return self.current_state in self.accepting_states

from Utils.FiniteAutomation import FiniteAutomaton




def FiniteAutomataTests():
    try:
        file_path = "FA.txt"
        with open(file_path, 'r') as file:
            lines = file.readlines()

            # Initialize variables
            states, alphabet, transition_rules, initial_state,
accepting_states = set(), set(), {}, '', set()
            current_section = None

            # Process each line in the file
            for line in lines:
                line = line.strip()

                # Check for section markers
                if line.startswith('States:'):
                    current_section = 'States'
                    continue
                elif line.startswith('Alphabet:'):
                    current_section = 'Alphabet'
                    continue
                elif line.startswith('Transitions:'):
                    current_section = 'Transitions'
                    continue
                elif line.startswith('Initial:'):
                    current_section = 'Initial'
                    continue
                elif line.startswith('Accepting:'):
                    current_section = 'Accepting'
                    continue

                # Process lines based on the current section
                if current_section == 'States':
                    states.update(line.split(','))
                elif current_section == 'Alphabet':
                    alphabet.update(line.split(','))
                elif current_section == 'Transitions':
                    source, symbol, target = map(str.strip, line.split(','))
                    transition_rules[(source, symbol)] = target
                elif current_section == 'Initial':
                    initial_state = line
                elif current_section == 'Accepting':
                    accepting_states.update(line.split(','))

    except FileNotFoundError:
        print(f"Error: File not found at {file_path}")
        return

    dfa = FiniteAutomaton(states, alphabet, transition_rules, initial_state,
accepting_states)

    input_string_list = ['111', '101', '1001']

    for input_string in input_string_list:
        for symbol in input_string:
            try:
                dfa.process_input(symbol)
            except ValueError as e:
                print(f"Error: {e}")
                break

        if dfa.is_accepting():
            print("Input accepted!")
        else:
            print("Input rejected.")

        dfa.reset()
```

```
from Tests.ScannerTests import ScannerTests
from Tests.SymTableTests import SymTableTests
from Tests.FiniteAutomataTests import FiniteAutomataTests

if __name__ == "__main__":
    #SymTableTests()
    #ScannerTests()
    FiniteAutomataTests()
```