```
Joshuas-MBP:proj1 JoshMChoi$ pwd
/Users/JoshMChoi/desktop/theory-comp/proj1
Joshuas-MBP:proj1 JoshMChoi$ cat FSA.cc
//
//  FSA.cc
//  Project1
//
//  Created by Joshua Choi on 02/10/2020.
//  Copyright 2020 Nanogram, Inc. All rights reserved.
//

#include "FSA.h"
#include <algorithm>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>



using namespace std;


/**
 MARK: - Init
 @param ifs An ifstream reference
 */
FSA::FSA(ifstream &ifs) {
    cout << "called stub version of FSA constructor\n";

    /// Initialized Int value representing the n-number of accept
states
    int numberOfAccepts = 0;

    // Read
    ifs >> sigma >> num_states >> start_state >> numberOfAccepts;

    // Iterate to determine the number of states, starting and
accepting states
    for (int i = 0; i < numberOfAccepts; i++) {
        int acceptingState;
        ifs >> acceptingState;
        accept_states.push_back(acceptingState);
    }

    // Get the state table
    get_state_table(ifs);
}
```

```
/**
 Update the table with data-entries from the input file. This method
returns nothing and only iterates through each row/column in the input
file (delimiter = whitespaces). Preferrable name suggestion:
"set_state_table"
 @param ifs An ifstream input
 */
void FSA::get_state_table(ifstream &ifs) {
    cout << "called stub version of get_state_table()\n";

    /// Initialized Int vector representing the row of the table
initialized w/0s
    vector<int> base_row(sigma.size());

    // Append the row to the table
    state_table.push_back(base_row);

    // Read the contents of the state table from the file
    for (int row = 0; row < num_states; row++) {
        for (unsigned int column = 0; column < sigma.size(); column++)
{
            ifs >> base_row[column];
        }
        // Append to the table
        state_table.push_back(base_row);
    }
}



/**
 Describes and logs the number of states, start state, accept states,
etc. of the FSA.
 */
void FSA::describe() {
    cout <<"called stub version of describe()\n";

    cout << "\n***Describing the FSA...***\n";
    cout << "• Alphabet: " << sigma << "\n";
    cout << "• N-Number of States: " << num_states << "\n";
    cout << "• Start State: " << start_state << "\n";
    // Accept States
    cout << "• N-Number of Accept States: " << accept_states.size() <<
"\n";
    for (unsigned long int i = 0; i <accept_states.size(); i++) {
        cout << accept_states[i] << " " << "\n";
    }
    // Table
```

```
    cout << "• State Table:\n" << "    a    b    \n";
    for (int i = 0; i < sigma.length(); i++) {
        for (unsigned long int j = 0; j < sigma.length(); j++) {
            // Log the value at the specified row/column
            cout << "    " << state_table[i][j] << "\n";
        }
    }
}



/**
 Traces the operation of the FSA object indicating whether or not the
object accepts the input string.
 @param in_string A String reference-value representing the input
 */
void FSA::trace(const string& in_string) {
    cout << "called stub version of trace()\n";

    /// Initialized Int vector representing the state of the FSA's
input
    vector<int> state_trace;
    /// Initialized Int value representing the initial start state
    int current_state = start_state;
    // Append the current state to the vector (to initialize the op of
the FSA)
    state_trace.push_back(current_state);

    // Cast sigma as a vector of characters to prepare to find the
character's index from the input
    const std::vector<char> characterVector(sigma.begin(),
sigma.end());

    // Iterate through the input string
    for (char ch: in_string) {
        // Grab the index of this character
        int index = indexOf(characterVector, ch);

        // If it's valid, append it to the state_trace vecto
        if (index >= 0) {
            current_state = state_table[current_state][index];
            state_trace.push_back(current_state);
        } else {
            cout << "Invalid Input: " << ch << endl;
            exit(1);
        }
    }

    // Iterate through the states and log it
    for (int state: state_trace) {
```

```
        cout << "State: " << state << " ";
    }

    // Next, log whether the input string is valid or invalid
    cout << "\n########\n" << "\"" << in_string << "\"" << " Is ";
    cout << ((indexOf(accept_states, current_state) >= 0) ? "Valid!" :
"Invalid.\nPlease Try Again.");
    cout << "\n########\n";
}



/**
 Method used to get the index, i, of any element, e in a vector, v.
Why does this exist? Because we're too lazy to iterate through
multiple loops to find the correct index of (1) the FSA's states (int)
and (2) the FSA's input (string)
 @param list Dynamic type vector (make sure to cast it as a const for
runtime)
 @param element Dynamic element value
 */
template <typename T> int FSA::indexOf(const vector<T> list, const T
element) {
    // MARK: - Algorithm
    auto i = std::find(list.begin(), list.end(), element);
    return i == list.end() ? -1 : int(i - list.begin());
}
Joshuas-MBP:proj1 JoshMChoi$ make clean
Joshuas-MBP:proj1 JoshMChoi$ make
c++ -Wall -std=c++17    -c -o proj1.o proj1.cc
c++ -Wall -std=c++17    -c -o FSA.o FSA.cc
g++ -o proj1 -Wall -std=c++17  proj1.o FSA.o
Joshuas-MBP:proj1 JoshMChoi$ ./proj1 data1
called stub version of FSA constructor
called stub version of get_state_table()
FSA description:
called stub version of describe()

***Describing the FSA...***
• Alphabet: ab
• N-Number of States: 3
• Start State: 1
• N-Number of Accept States: 1
2
• State Table:
    a    b
    0
    0
    2
    1
```

```
Enter input strings, one line at a time:
? ab
called stub version of trace()
State: 1 State: 2 State: 3
########
"ab" Is Invalid.
Please Try Again.
########
? aabb
called stub version of trace()
State: 1 State: 2 State: 3 State: 1 State: 1
########
"aabb" Is Invalid.
Please Try Again.
########
? aabba
called stub version of trace()
State: 1 State: 2 State: 3 State: 1 State: 1 State: 2
########
"aabba" Is Valid!
########
? aabbaa
called stub version of trace()
State: 1 State: 2 State: 3 State: 1 State: 1 State: 2 State: 3
########
"aabbaa" Is Invalid.
Please Try Again.
########
? ^C
Joshuas-MBP:proj1 JoshMChoi$
```