Finite State Machine Simulation (04/23/2017)
Alberto Li and Joshua Henson

Part 1: Implementation

For this project, we chose to implement our finite state machine using an adjacency list, which was done using vectors in C++. For each state, a `Node_Entry` is created. The entry contains a `name` field, a map of input values to Boolean values to see which inputs have been defined, a vector of `Arc_Entry` elements that originate from the original node, and an optional `output` field (to be used for Moore state machines). Each `Arc_Entry` has a field for the `start_node`, `end_node`, `input`, and `output` (for Mealy machines). The inclusion of the vector of `Arc_Entry` elements is what makes our implementation an adjacency list. The fields included in each entry of this vector make it easier to keep track of the relationship between nodes and make it easier to print out the graph and the table. The entire source code, which is in C++, can be seen in appendix A.

Part 2: Test Plan

In order to test our finite state machine simulation, we have decided to input arbitrary sequences of 1-bit-input, 2-bit-input, 3-bit-input sequences. This will allow us to determine whether or not our program is comparable to the expected output. This, however, only gives the output for a 2-bit input Mealy machine. Since this does not demonstrate the necessary capability, we will also show the output when testing our simulation with a very basic 1-bit Moore machine, and with more complex 3-bit Moore and Mealy machines. The original test case and the 3-bit inputs will have some errors when defining states or arcs in the simulation, since this will allow us to check the expected error-checking capabilities of our simulation. The string of commands used for each test can be seen below. The anticipated errors have been highlighted in yellow.

1-bit input Moore Machine

```
Node Black Dark
Node White Light
Arc Black White x
Arc White Black 0
Arc White White
```

2-bit input Mealy Machine

```
NODE Blue
NODE Red
ARC Red Red 00 / hot
ARC Blue Blue 0x / hold
ARC Red Blue 01 / cold
ARC Orange Blue 10 / cold
ARC Blue Red 11 / on
ARC Red Blue 1x / off
```

3-bit input Moore Machine

Node Red HOT
Node Green CALM
Node Blue COLD
Arc Green Green 01x
Arc Blue Red 000
Arc Red Red xx0
Arc Green Blue 00x
Arc Orange Blue 00x
Arc Red Red 110
Arc Blue Green 11x
Arc Blue Red 001
Arc Green Red 11x
Arc Red Green 001
Arc Blue Green 01x
Arc Red Green 011
Arc Red Blue 101
Arc Green Red 101
Red Blue 111
Arc Red Blue 111
Arc Green Blue 100

3-bit input Mealy Machine

Node Red
Node Green
Node Blue
Arc Red Red xx0 / HOT
Arc Red Blue 001 / COLD
Arc Green Green 01x / CALM
Arc Purple Blue 000 / COLD
Arc Blue Yellow 000 / COLD
Arc Blue Green 110 / CALM
Arc Blue Green x11 / CALM
Arc Green Red 00x / HOT
Arc Red Green 010 / CALM
Arc Red Green 011 / CALM
Arc Red Green 1x1 / CALM
Blue Green 00x / CALM
Arc Green Blue 10x / COLD

These test cases show the expected functionality of the finite state machine simulation. The results of the simulations (with corresponding error messages for the highlighted lines and graph outputs) can be seen in Appendix A.

Appendix A: Terminal Outputs from Test Cases

1-bit Input Moore Machine

FSM Simulator

```
---------------------------------------------- FSM Help -----------------------------------------------
Please enter the machine type and number of input bits as prompted upon starting simulation.
The following commands can be used to define different states and the transitions between them.
---------------------------------------------------------------------------------------------------------
NODE [name] {Mealy} || NODE [name] [output] {Moore}                  -  add a node to the graph
ARC [start] [end] [in / out] {Mealy} || ARC [start] [end] [in] {Moore} -  add arc to a node in the graph
output                                                  -  shows output graph
?                                                       -  display this help menu
quit                                                    -  exit the program

FSM-SIM> Please specify simulation type. Enter MEALY or MOORE: Moore
FSM-SIM> Please specify the number of input bits (1-4): 1
FSM-SIM> Node Black 0
FSM-SIM> Node White 1
FSM-SIM> Arc Black White x
FSM-SIM> Arc White Black 0
FSM-SIM> Arc White White 1
FSM-SIM> output
Output GRAPH:
Black / 0
            White x
White / 1
            Black 0
            White 1

Current | Next State / Output
State       | X = 0      X = 1
--------------------------------
Black       | White/0    White/0
White       | Black/1    White/1
FSM-SIM> quit
Exiting FSM Simulator...
```

2-bit Input Mealy Machine

FSM Simulator

```
----------------------------------------------- FSM Help -----------------------------------------------
Please enter the machine type and number of input bits as prompted upon starting simulation.
The following commands can be used to define different states and the transitions between them.
-----------------------------------------------------------------------------------------------------------
NODE [name] {Mealy} || NODE [name] [output] {Moore}          -  add a node to the graph
ARC [start] [end] [in / out] {Mealy} || ARC [start] [end] [in] {Moore} -  add arc to a node in the graph
output                                         -  shows output graph
?                                            -  display this help menu
quit                                          -  exit the program

FSM-SIM> Please specify simulation type. Enter MEALY or MOORE: Mealy
FSM-SIM> Please specify the number of input bits (1-4): 2
FSM-SIM> Node Blue
FSM-SIM> Node Red
FSM-SIM> Arc Red Red 00 / hot
FSM-SIM> Arc Blue Blue 0x / hold
FSM-SIM> Arc Red Blue 01 / cold
FSM-SIM> Arc Orange Blue 10 / cold
%% error: state "Orange" not defined %%
FSM-SIM> Arc Blue Red 11 / on
FSM-SIM> Arc Red Blue 1x off
off
FSM-SIM> output
Output GRAPH:
Blue
        Blue 0x / hold
        Red 11 / on
        %% warning: input 10 not specified %%
Red
        Red 00 / hot
        Blue 01 / cold
        Blue 1x / off

Current   |          Next State / Output
State               | X = 00     X = 01     X = 10     X = 11
-------------------------------------------------------
Blue                | Blue/hold   Blue/hold   x/x        Red/on
Red                 | Red/hot     Blue/cold   Blue/off    Blue/off
```

3-bit Input Moore Machine

```
FSM Simulator

------------------------------------------------ FSM Help ------------------------------------------------
Please enter the machine type and number of input bits as prompted upon starting simulation.
The following commands can be used to define different states and the transitions between them.
--------------------------------------------------------------------------------------------------------------
NODE [name] {Mealy} || NODE [name] [output] {Moore}             - add a node to the graph
ARC [start] [end] [in / out] {Mealy} || ARC [start] [end] [in] {Moore} - add arc to a node in the graph
output                                                  - shows output graph
?                                                       - display this help menu
quit                                                    - exit the program

FSM-SIM> Please specify simulation type. Enter MEALY or MOORE: MOORE
FSM-SIM> Please specify the number of input bits (1-4): 3
FSM-SIM> Node Red HOT
FSM-SIM> Node Green CALM
FSM-SIM> Node Blue COLD
FSM-SIM> Arc Green Green 01x
FSM-SIM> Arc Blue Red 000
FSM-SIM> Arc Red Red xx0
FSM-SIM> Arc Green Blue 00x
FSM-SIM> Arc Orange Blue 00x
%% error: state "Orange" not defined %%
FSM-SIM> Arc Red Red 110
Invalid action, this Arc has already been added!
FSM-SIM> Arc Blue Green 11x
FSM-SIM> Arc Blue Red 001
FSM-SIM> Arc Green Red 11x
FSM-SIM> Arc Red Green 001
FSM-SIM> Arc Blue Green 01x
FSM-SIM> Arc Red Green 011
FSM-SIM> Arc Red Blue 101
FSM-SIM> Arc Green Red 101
FSM-SIM> Red Blue 111
Invalid Command
FSM-SIM> Arc Red Blue 111
FSM-SIM> Arc Green Blue 100
FSM-SIM> output
Output GRAPH:
Blue / COLD
        Red 000
        Green 11x
        Red 001
        Green 01x
        %% warning: input 100 not specified %%
        %% warning: input 101 not specified %%
Green / CALM
        Green 01x
        Blue 00x
        Red 11x
        Red 101
        Blue 100
Red / HOT
        Red xx0
        Green 001
        Green 011
        Blue 101
        Blue 111

Current   |         Next State / Output
State               | X = 000   X = 001   X = 010   X = 011   X = 100   X = 101   X = 110   X = 111
------------------------------------------------------------------------------------------------------
Blue                | Red/COLD   Red/COLD   Green/COLD  Green/COLD  x/COLD     x/COLD     Green/COLD  Green/COLD
Green               | Blue/CALM  Blue/CALM  Green/CALM  Green/CALM  Blue/CALM  Red/CALM   Red/CALM    Red/CALM
Red                 | Red/HOT    Green/HOT  Red/HOT     Green/HOT   Red/HOT    Blue/HOT   Red/HOT     Blue/HOT
FSM-SIM> quit
Exiting FSM Simulator...
```

3-bit Input Mealy Machine

```
FSM Simulator

------------------------------------------------ FSM Help ------------------------------------------------
Please enter the machine type and number of input bits as prompted upon starting simulation.
The following commands can be used to define different states and the transitions between them.
```

```
-----------------------------------------------------------------------------------------------------------
NODE [name] {Mealy} || NODE [name] [output] {Moore}              -  add a node to the graph
ARC [start] [end] [in / out] {Mealy} || ARC [start] [end] [in] {Moore} -  add arc to a node in the graph
output                                              -  shows output graph
?                                                 -  display this help menu
quit                                                -  exit the program

FSM-SIM> Please specify simulation type. Enter MEALY or MOORE: MEALY
FSM-SIM> Please specify the number of input bits (1-4): 3
FSM-SIM> Node Red
FSM-SIM> Node Green
FSM-SIM> Node Blue
FSM-SIM> Arc Red Red xx0 / HOT
FSM-SIM> Arc Red Blue 001 / COLD
FSM-SIM> Arc Green Green 01x / CALM
FSM-SIM> Arc Purple Blue 000 / COLD
%% error: state "Purple" not defined %%
FSM-SIM> Arc Blue Yellow 000 / COLD
%% error: state "Yellow" not defined %%
FSM-SIM> Arc Blue Green 110 / CALM
FSM-SIM> Arc Blue Green x11 / CALM
FSM-SIM> Arc Green Red 00x / HOT
FSM-SIM> Arc Red Green 010 / CALM
Invalid action, this Arc has already been added!
FSM-SIM> Arc Red Green 011 / CALM
FSM-SIM> Arc Red Green 1x1 / CALM
FSM-SIM> Arc Blue Green 00x / CALM
FSM-SIM> Arc Green Blue 10x / COLD
FSM-SIM> output
Output GRAPH:
Blue
            Green 110 / CALM
            Green x11 / CALM
            Green 00x / CALM
            %% warning: input 010 not specified %%
            %% warning: input 100 not specified %%
            %% warning: input 101 not specified %%
Green
            Green 01x / CALM
            Red 00x / HOT
            Blue 10x / COLD
            %% warning: input 110 not specified %%
            %% warning: input 111 not specified %%
Red
            Red xx0 / HOT
            Blue 001 / COLD
            Green 011 / CALM
            Green 1x1 / CALM

Current    |            Next State / Output
State                   | X = 000    X = 001    X = 010    X = 011    X = 100    X = 101    X = 110    X = 111
----------------------------------------------------------------------------------------------------------
Blue                    | Green/CALM  Green/CALM  x/x        Green/CALM  x/x        x/x        Green/CALM  Green/CALM
Green                   | Red/HOT    Red/HOT    Green/CALM  Green/CALM  Blue/COLD   Blue/COLD   x/x        x/x
Red                     | Red/HOT    Blue/COLD   Red/HOT    Green/CALM  Red/HOT    Green/CALM  Red/HOT    Green/CALM
FSM-SIM> quit
Exiting FSM Simulator...
```

Appendix B: Finite State Machine Source Code

```
/**
 * Alberto Li
 * Joshua Henson
 * ECE 3020, Dr. Hughes
 * 04/23/17
 * Programming Assignment #3 -- Finite State Machine Simulation
 * This Simulation was inspired by the ECE 3056 LC-3b Simulator
 * Input Functions (Albert) - addArc(), addNode()
 * Output Functions (Joshua) - printOutput(), printTable()
 */

#include <algorithm>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <list>
#include <vector>
#include <map>
#include <cmath>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloca.h>

using namespace std;

/***************************************************************/
/* DEFINITION OF CONSTANTS PROVIDED IN PROBLEM STATEMENT       */
/***************************************************************/

#define TRUE  1
#define FALSE 0
#define MAX_NODES 25
#define MAX_INPUT_BITS 4
#define MAX_STATE_NAME 8
#define MAX_OUTPUT_NAME 5

/***************************************************************/
/* DEFINITION OF STRUCTURES USED TO REPRESENT STATES IN FSM    */
/***************************************************************/

typedef struct Arc_Struct {
    string start_node; //node that arc starts from
    string end_node; //node that arc ends at
    string input; //input that results in state change
    string output; //only used for MEALY state machine
} Arc_Entry;

typedef struct Node_Struct {
    string name; //name of node
    string output; //ouput displayed, used only for MOORE state machine
    map<string,bool> io; //input / output string
    vector<Arc_Entry> arcs; //vector of arcs from this state
} Node_Entry;
```

```
/****************************************************************/
/* GLOBAL VARIABLES                                             */
/****************************************************************/

string MACHINE; //type of machine selected by user
int INPUT_BITS; //number of input bits selected by user
Node_Entry NODE; //data structure for node
vector<Node_Entry> NV; //vector that holds all nodes
Arc_Entry ARC; //data structure for arc

/****************************************************************/
/* GLOBAL FUNCTIONS                                             */
/****************************************************************/

/**
 * Prints help menu at beginning of simuation and when requested by user
 */
void help() {
    printf("------------------------------------------------ FSM Help -----------------
------------------------------- \n");
    printf("Please enter the machine type and number of input bits as prompted upon
starting simulation.              \n");
    printf("The following commands can be used to define different states and the
transitions between them.           \n");
    printf("----------------------------------------------------------------------------
------------------------------- \n");
    printf("NODE [name] {Mealy} || NODE [name] [output] {Moore}                    -
add a node to the graph         \n");
    printf("ARC [start] [end] [in / out] {Mealy} || ARC [start] [end] [in] {Moore} -
add arc to a node in the graph  \n");
    printf("output                                                                 -
shows output graph              \n");
    printf("?                                                                      -
display this help menu          \n");
    printf("quit                                                                   -
exit the program                \n\n");
}

/*
* Wildcard Matching, used for 'x' in input conditions
* @param first
* @param second
* @return bool whether or not first and second are the same with wildcard
*       checking
* Source(Modified): geeksforgeeks.org/wildcard-character-matching/
*/
bool match(char *first, char *second) {

    // If we reach at the end of both strings, we are done
    if (*first == '\0' && *second == '\0') {
        return true;
    }

    // Make sure that the characters after 'x' are present
    // in second string. This function assumes that the first
    // string will not contain two consecutive 'x'
    if (*first == 'x' && *(first + 1) != '\0' && *second == '\0') {
        return false;
    }

    // If the first string contains '?', or current characters
    // of both strings match
    if (*first == '?' || *first == *second) {
        return match(first + 1, second + 1);
    }

    // If there is 'x', then there are two possibilities
    // a) We consider current character of second string
    // b) We ignore current character of second string.
    if (*first == 'x') {
        return match(first + 1, second) || match(first, second + 1);
    }

    return false;
}

/*
* Compares the name fields of two nodes, used for sorting alpahbetically
* @param a
* @param b
* @returns true if a comes before b alphabetically
*/
bool alphabetic(const Node_Entry &a, const Node_Entry &b) {
    return a.name < b.name;
}

/**
 * Changes the case of a given string to upper case
 * @param  str
 * @return transformed string
 */
string toUpper(const string& str) {
    string result;
    locale loc;
    for (unsigned int i = 0; i < str.length(); ++i) {
        result += toupper(str.at(i), loc);
    }
    return result;
}

/**
 * Modifies iterator to point to the NODE specifide by @name
 * @param  it
 * @param  name
 * @return true if the NODE is inside the graph
 */
bool locate(vector<Node_Entry>::iterator& it, string name) {
    for (it = NV.begin(); it < NV.end(); ++it) {
        if ((*it).name == name) {
            return true;
        }
    }
    return false;
}

/*
* Adds arc to an existing node in a Mealy Machine
* @param startNode
* @param endNode
* @param input
* @param output
*/
void addArc(string startNode, string endNode, string input, string output) {
    vector<Node_Entry>::iterator vit;
    locate(vit,startNode); // moves vector_iterator to startNode
    vector<Arc_Entry>::iterator ait = (*vit).arcs.begin();
    string temp1;

    // Update Arc States
    ARC.start_node = startNode;
    ARC.end_node = endNode;
    ARC.input = input;
    ARC.output = output;

    // Error Check input
    if ((input.size() != INPUT_BITS)) {
        cout << input.size() << " " << INPUT_BITS;
        cout << "invalid input, please specify an input using 0,1,x of " << INPUT_BITS
<< " bits!" << endl;
        return;
    }
    if (output.size() > MAX_OUTPUT_NAME) {
        cout << "invalid output, please specify an output of at most " <<
MAX_OUTPUT_NAME << " characters!" << endl;
        return;
    }

    // Update Possible Arcs
    map<string, bool>::iterator it;
    for (it = (*vit).io.begin(); it != (*vit).io.end(); ++it) {
        if (match(strdup(input.c_str()),
            strdup((it->first).c_str())) && it->second) { // Check for Adding same
node
            cout << "Invalid action, this Arc has already been added!" << endl;
            return;
        }

        if (match(strdup(input.c_str()), strdup((it->first).c_str()))) {
            it->second = true;
        }
    }

    // Add Arc to designated node
    (*vit).arcs.push_back(ARC);
}
```

```
/*
* Adds arc to an existing node in a Moore Machine
* @param startNode
* @param endNode
* @param input
*/
void addArc(string startNode, string endNode, string input) {
    vector<Node_Entry>::iterator vit;
    locate(vit,startNode); // moves vector_iterator to startNode
    vector<Arc_Entry>::iterator ait = (*vit).arcs.begin();
    string temp1;

    // Update Arc States
    ARC.start_node = startNode;
    ARC.end_node = endNode;
    ARC.input = input;

    // Error Check input
    if ((input.size() != INPUT_BITS)) {
        cout << input.size() << " " << INPUT_BITS;
        cout << "invalid input, please specify an input using 0,1,x of " << INPUT_BITS
<< " bits!" << endl;
        return;
    }

    // Update Possible Arcs
    map<string, bool>::iterator it;
    for (it = (*vit).io.begin(); it != (*vit).io.end(); ++it) { //arcs
        if (match(strdup(input.c_str()),
            strdup((it->first).c_str())) && it->second) { // Check for Adding same
node
            cout << "Invalid action, this Arc has already been added!" << endl;
            return;
        }

        if (match(strdup(input.c_str()), strdup((it->first).c_str()))) {
            it->second = true;
        }
    }

    // Add Arc to designated node
    (*vit).arcs.push_back(ARC);
}


/*
* Adds node graph in a Mealy Machine
* @param name
* @return true if node can be added to graph
*/
bool addNode(const string name) {

    // Make sure Nodes in graph does not exceed MAX_NODES def
    if (NV.size() >= 25) {
        cout << "Cannot add to graph. Your graph is already at a max capacity of "
        << MAX_NODES << " nodes!" << endl;
        return false;
    }

    // Iterate through vector to check if is node is already in graph
    bool inside = FALSE;
    for (int i = 0; i < NV.size(); ++i) {
        if(NV[i].name == name) {
            inside = TRUE;
        }
    }

    // Add node to graph
    if (inside) { // Already in graph, do not add
        cout << name << " is already in the graph! It cannot be added again"
        << endl;
        return false;
    } else { // Not in graph, add the node!
        NODE.name = name;

        switch (INPUT_BITS) { // Initialize possible input combos to false
            case 1:
                NODE.io.insert(make_pair("0",false));
                NODE.io.insert(make_pair("1",false));
                break;
            case 2:
                NODE.io.insert(make_pair("00",false));
                NODE.io.insert(make_pair("01",false));
                NODE.io.insert(make_pair("10",false));
                NODE.io.insert(make_pair("11",false));
                break;
            case 3:
                NODE.io.insert(make_pair("000",false));
                NODE.io.insert(make_pair("001",false));
                NODE.io.insert(make_pair("010",false));
                NODE.io.insert(make_pair("011",false));
                NODE.io.insert(make_pair("100",false));
                NODE.io.insert(make_pair("101",false));
                NODE.io.insert(make_pair("110",false));
                NODE.io.insert(make_pair("111",false));
                break;
            case 4:
                NODE.io.insert(make_pair("0000",false));
                NODE.io.insert(make_pair("0001",false));
                NODE.io.insert(make_pair("0010",false));
                NODE.io.insert(make_pair("0011",false));
                NODE.io.insert(make_pair("0100",false));
                NODE.io.insert(make_pair("0101",false));
                NODE.io.insert(make_pair("0110",false));
                NODE.io.insert(make_pair("0111",false));
                NODE.io.insert(make_pair("1000",false));
                NODE.io.insert(make_pair("1001",false));
                NODE.io.insert(make_pair("1010",false));
                NODE.io.insert(make_pair("1011",false));
                NODE.io.insert(make_pair("1100",false));
                NODE.io.insert(make_pair("1101",false));
                NODE.io.insert(make_pair("1110",false));
                NODE.io.insert(make_pair("1111",false));
                break;
        }
        NV.push_back(NODE);
        sort(NV.begin(), NV.end(), alphabetic);
        return true;
    }
}

/*
* Adds node to graph in a Moore Machine
* @param name
* @param output
* @return true if node can be added to graph
*/
bool addNode(const string name, const string output) {
    // Make sure Nodes in graph does not exceed MAX_NODES def
    if (NV.size() >= 25) {
        cout << "Cannot add to graph. Your graph is already at a max capacity of "
                << MAX_NODES << " nodes!" << endl;
        return false;
    }

    // Iterate through vector to check if is node is already in graph
    bool inside = FALSE;
    for (int i = 0; i < NV.size(); ++i) {
        if(NV[i].name == name) {
            inside = TRUE;
        }
    }

    // Add node to graph
    if (inside) { // Already in graph, do not add
        cout << name << " is already in the graph! It cannot be added again." << endl;
        return false;
    } else { // Not in graph, add the node!
        NODE.name = name;
        NODE.output = output;
        switch (INPUT_BITS) { // Initialize possible input combos to false
            case 1:
                NODE.io.insert(make_pair("0",false));
                NODE.io.insert(make_pair("1",false));
                break;
            case 2:
                NODE.io.insert(make_pair("00",false));
                NODE.io.insert(make_pair("01",false));
                NODE.io.insert(make_pair("10",false));
                NODE.io.insert(make_pair("11",false));
                break;
            case 3:
                NODE.io.insert(make_pair("000",false));
                NODE.io.insert(make_pair("001",false));
                NODE.io.insert(make_pair("010",false));
                NODE.io.insert(make_pair("011",false));
                NODE.io.insert(make_pair("100",false));
                NODE.io.insert(make_pair("101",false));
                NODE.io.insert(make_pair("110",false));
                NODE.io.insert(make_pair("111",false));
                break;
            case 4:
                NODE.io.insert(make_pair("0000",false));
                NODE.io.insert(make_pair("0001",false));
                NODE.io.insert(make_pair("0010",false));
                NODE.io.insert(make_pair("0011",false));
                NODE.io.insert(make_pair("0100",false));
                NODE.io.insert(make_pair("0101",false));
                NODE.io.insert(make_pair("0110",false));
                NODE.io.insert(make_pair("0111",false));
                NODE.io.insert(make_pair("1000",false));
                NODE.io.insert(make_pair("1001",false));
                NODE.io.insert(make_pair("1010",false));
                NODE.io.insert(make_pair("1011",false));
                NODE.io.insert(make_pair("1100",false));
                NODE.io.insert(make_pair("1101",false));
                NODE.io.insert(make_pair("1110",false));
                NODE.io.insert(make_pair("1111",false));
                break;
        }
        NV.push_back(NODE);
        sort(NV.begin(), NV.end(), alphabetic);
        return true;
    }
}

/*
* Prints all nodes with their associated arcs
*/
void printOutput() {
    map<string, bool>::iterator it;

    if (MACHINE == "MEALY") {
        cout << "Output GRAPH:" << endl;
        for (int i = 0; i < NV.size(); ++i) { //iterate through nodes
            cout << NV[i].name << endl;
            for (int j = 0; j < NV[i].arcs.size(); ++j) { //iterate through arcs
                cout << "\t";
                cout << NV[i].arcs[j].end_node << " " <<NV[i].arcs[j].input << " / "
                << NV[i].arcs[j].output << endl;
            }

            // Check unspecified inputs
            for (it = NV[i].io.begin(); it != NV[i].io.end(); ++it) {
                if (!(it->second)) {
                    cout << "\t%% warning: input " << it->first <<
                    " not specified %%" << endl;
                }
            }
        }

        cout << endl;
    } else if (MACHINE == "MOORE") {
        cout << "Output GRAPH:" << endl;
        for (int i = 0; i < NV.size(); ++i) { //iterate through nodes
            cout << NV[i].name << " / " << NV[i].output << endl;
            for (int j = 0; j < NV[i].arcs.size(); ++j) { //iterate through arcs
                cout << "\t";
                cout << NV[i].arcs[j].end_node << " " <<NV[i].arcs[j].input
                << endl;
            }

            // Check unspecified inputs
            for (it = NV[i].io.begin(); it != NV[i].io.end(); ++it) {
                if (!(it->second)) {
                    cout << "\t%% warning: input " << it->first <<
                    " not specified %%" << endl;
                }
            }
        }
        cout << endl;
    }
}

/*
* Prints all nodes and their associated arcs in a state transition table
*/
void printTable() {
    if (MACHINE == "MEALY") { // for mealy machines
        if (NV.empty()) {
            cout << "Please add nodes to the graph before using (o)utput" << endl;
            return;
        }
        vector<Node_Entry>::iterator vit;
        vector<Arc_Entry>::iterator ait;

        // Inputs Line
        map<string, bool>::iterator it;
        cout << left;
        cout << "Current\t|\tNext State / Output" << endl;
        cout << "State\t| ";
        for (it = NV[0].io.begin(); it != NV[0].io.end(); ++it) {
            cout << setw(12) << ("X = " + it->first);
        }

        cout << "\n--------";
        for (int i = 0; i < pow(2, INPUT_BITS); i++) {
            cout << "------------";
        }
        cout << endl;

        bool amatch = false;

        // EndNode / Output
        for (vit = NV.begin(); vit < NV.end(); ++vit) { //nodes
                cout << (*vit).name << "\t| ";
            for (it = NV[0].io.begin(); it != NV[0].io.end(); ++it) { // states
                for (ait = (*vit).arcs.begin(); ait != (*vit).arcs.end(); ++ait) {
//arcs
                    if (match(strdup(((*ait).input).c_str()), strdup((it-
>first).c_str()))) { //input vs. arc
                        cout << setw(12) << ((*ait).end_node + "/" + (*ait).output);
                        amatch = true;
                        break;
                    } else {
                        amatch = false;
                    }
                }
                if (!amatch) { //input vs. arc
                    cout << setw(12) << "x/x";
                }
                amatch = false;
            }
                cout << endl;
        }
    } else if (MACHINE == "MOORE") { // for moore machines
        if (NV.empty()) {
            cout << "Please add nodes to the graph before using (o)utput" << endl;
            return;
        }
        vector<Node_Entry>::iterator vit;
        vector<Arc_Entry>::iterator ait;

        // Inputs Line
        map<string, bool>::iterator it;
        cout << left;
        cout << "Current\t|\tNext State / Output" << endl;
        cout << "State\t| ";
        for (it = NV[0].io.begin(); it != NV[0].io.end(); ++it) {
            cout << setw(12) << ("X = " + it->first);
        }

        cout << "\n--------";
        for (int i = 0; i < pow(2, INPUT_BITS); i++) {
            cout << "------------";
        }
        cout << endl;

        bool amatch = false;

        // EndNode / Output
        for (vit = NV.begin(); vit < NV.end(); ++vit) { //nodes
                cout << (*vit).name << "\t| ";
            for (it = NV[0].io.begin(); it != NV[0].io.end(); ++it) { // states
                for (ait = (*vit).arcs.begin(); ait != (*vit).arcs.end(); ++ait) {
//arcs
                    if (match(strdup(((*ait).input).c_str()), strdup((it-
>first).c_str()))) { //input vs. arc
                        cout << setw(12) << ((*ait).end_node + "/" + (*vit).output);
                        amatch = true;
                        break;
                    } else {
                        amatch = false;
                    }
                }
                if (!amatch) { //input vs. arc
                    cout << setw(12) << ("x/" + (*vit).output);
                }
                amatch = false;
            }
            cout << endl;
        }
    }
}

/*
* Prompts the user for the type of machine and the number of input bits they
*       would like to use
*/
void initialize() {

    do {
        cout << "FSM-SIM> Please specify simulation type. Enter MEALY or MOORE: ";
        cin >> MACHINE;
        MACHINE = toUpper(MACHINE);
    } while(!(toUpper(MACHINE) == "MEALY" || toUpper(MACHINE) == "MOORE"));

    char buffer[10];

    do {
        cout << "FSM-SIM> Please specify the number of input bits (1-4): ";
        cin >> buffer;
        INPUT_BITS = (int) buffer[0] - '0';
    } while(!(INPUT_BITS >= 1 && INPUT_BITS <= 4));
}

/*
* Read command from user input
*/
void get_command() {
    // User Inputs
    char buffer[20];
    string name;
    string startNode;
    string endNode;
    string input;
    string forwardslash;
    string output;

    vector<Node_Entry>::iterator it;
    printf("FSM-SIM> ");
    cin >> buffer;

    switch(buffer[0]) {
        // Add NODE
        case 'N':
        case 'n':
        if (MACHINE == "MEALY") {
            cin >> name;
            if (name.size() > MAX_STATE_NAME) {
                cout << "Please specify a state name of at most " << MAX_STATE_NAME
                << " alphanumeric characters." << endl;
            } else {
                addNode(name);
            }
        } else if (MACHINE == "MOORE") {
            cin >> name;
            cin >> output;
            if (name.size() > MAX_STATE_NAME) {
                cout << "Please specify a state name of at most " << MAX_STATE_NAME
                << " alphanumeric characters." << endl;
            } else {
                addNode(name,output);
            }
        }
        break;

        // Add ARC
        case 'A':
        case 'a':
            if (MACHINE == "MEALY") {
                cin >> startNode >> endNode >> input >> forwardslash >> output;
                if (!locate(it,startNode)) { //Invalid StartNode
                    cout << "%% error: state \"" << startNode << "\" not defined %%"
<< endl;
                } else if (!locate(it,endNode)) { //Invalid EndNode
                    cout << "%% error: state \"" << endNode << "\" not defined %%" <<
endl;
                } else { //All Good! Add ARC!
                    replace( input.begin(), input.end(), 'X', 'x');
                    addArc(startNode,endNode,input,output);
                }
            } else if (MACHINE == "MOORE") {
                cin >> startNode >> endNode >> input;
                if (!locate(it,startNode)) { //Invalid StartNode
                    cout << "%% error: state \"" << startNode << "\" not defined %%"
<< endl;
                } else if (!locate(it,endNode)) { //Invalid EndNode
                    cout << "%% error: state \"" << endNode << "\" not defined %%" <<
endl;
                } else { //All Good! Add ARC!
                    replace( input.begin(), input.end(), 'X', 'x');
                    addArc(startNode,endNode,input,output);
                }
            }

            break;
        // Help
        case '?':
        case 'H':
        case 'h':
            help();
            break;

        // Quit
        case 'Q':
        case 'q':
            printf("Exiting FSM Simulator...\n");
            exit(0);

        // Output
        case 'O':
        case 'o':
            printOutput();
            printTable();
            break;

        // Invalid Command
        default:
            printf("Invalid Command\n");
            break;
    }
}

/***************************************************************/
/* MAIN FUNCTION                                               */
/***************************************************************/

int main(int argc, char *argv[]) {

    printf("FSM Simulator\n\n");
    help();

    initialize();

    while (1) {
      get_command();
    }
}
```