April 22, 2017

Derin Ozturk and Victor Barr

SPRING 2017

Dr. Joseph Hughes

IMplementing Mealy and Moore StatE Machines

ECE 3020 - Project 3

# Description of Code:

We created structs to define the nodes and edges/arcs of both the mealy and moore versions of this state machine. These structs contained strings for storing the name, input, output and other related information of the node and arc. For storing the structs of nodes and arcs for a given graph, we used std::lists, one list for the arcs and one list for the nodes. The std::lists library generally uses a doubly linked list for its storage process.



# Test 1: Mealy | 4 states | 2 bit input

Initialization of:

1. Number of states
2. Number of bits
3. Mealy or Moore

Welcome to Derin and Victor's State Generator

Press enter after each of the following input specifications

How many states will you have? (INT 1 - 25)

4

How many input bits? (INT 1-4)

2

Mealy (Press 1) or Moore Machine (Press 0)?

1

Begin adding NODE and ARC for MEALY State Machine

NODE name or ARC fromNode toNode inputs / outputs

Type GRAPH to print graph and TABLE to print table

Type DONE to finish

Error Checking if a node has already been declared with the same name

NODE apple

1

apple

NODE apple

ERROR: This input has already been declared apple

Creating a NODE

“NODE name”

NODE orange

2

orange

NODE plantain

3

plantain

NODE water

4

Creating an ARC

“ARC to from input / out”

water

ARC orange plantain xx / yum

from node: orange to node: plantain inputs: xx outputs: yum

ARC plantain water xx / tasty

from node: plantain to node: water inputs: xx outputs: tasty

ARC water orange xx / rawr

from node: water to node: orange inputs: xx outputs: rawr

Displaying a table

TABLE

Curr State| Next State/Output

|0000 0001 0010 0011

------------------------------------------------------------------------------------------

apple | X / X X / X X / X X / X

orange |plantain / yum plantain / yum plantain / yum plantain / yum

plantain |water / tasty water / tasty water / tasty water / tasty

water |orange / rawr orange / rawr orange / rawr orange / rawr

GRAPH

Displaying a graph (note the warnings for missing inputs for states)

NODE: apple

%error: missing inputs for state changes for NODE apple%

NODE: orange

orange plantain 0000 / yum

orange plantain 0001 / yum

orange plantain 0011 / yum

orange plantain 0010 / yum

NODE: plantain

plantain water 0000 / tasty

plantain water 0001 / tasty

plantain water 0011 / tasty

plantain water 0010 / tasty

NODE: water

Error Checking for adding an ARC with undefined NODES

water orange 0000 / rawr

water orange 0001 / rawr

water orange 0011 / rawr

water orange 0010 / rawr

ARC water wine xx / nope

from node: water to node: wine inputs: xx outputs: nope

ERROR!! Either wine or water are not in the array :

# Test 2: Moore | 5 states | 3 bit input

Welcome to Derin and Victor's State Generator

Press enter after each of the following input specifications

How many states will you have? (INT 1 - 25)

5

Initialization of:

1. Number of states
2. Number of bits
3. Mealy or Moore

How many input bits? (INT 1-4)

3

Mealy (Press 1) or Moore Machine (Press 0)?

0

Begin adding NODE and ARC for MOORE State Machine

NODE name / output or ARC fromNode toNode inputs

Create a NODE:

“NODE name / output”

Type GRAPH to print graph and TABLE to print table

Type DONE to finish

NODE wine / class

name: wine output: class

1

wine

NODE vodka / trash  
name: vodka output: trash

2

vodka

NODE coors / yuck  
name: coors output: yuck

Create a NODE:

“ARC to from input”

3

coors

NODE ipa / yum

name: ipa output: yum

4

ipa

NODE water / drive

name: water output: drive

5

Create a TABLE

water

ARC water ipa xx1

from: water to: ipa inputs: xx1

ARC water vodka 000

from: water to: vodka inputs: 000

ARC coors vodka xxx  
from: coors to: vodka inputs: xxx

ARC wine coors x0x

from: wine to: coors inputs: x0x

ARC wine ipa 010

from: wine to: ipa inputs: 010

TABLE

Curr State / | Next State

Output |0000 0001 0010 0011 0100 0101 0110 0111

----------------------------------------------------------------------------------------------------------------

wine / class|coors coors ipa X coors coors X X

vodka / trash|X X X X X X X X

coors / yuck |vodka vodka vodka vodka vodka vodka vodka vodka

ipa / yum |X X X X X X X X

water / drive|vodka ipa X ipa X ipa X ipa

GRAPH

NODE: coors / yuck

Create a Graph, Note the errors for sections that have missing inputs for state changes

coors vodka 0000

coors vodka 0001

coors vodka 0011

coors vodka 0111

coors vodka 0110

coors vodka 0100

coors vodka 0010

coors vodka 0101

%error: missing inputs for state changes for NODE coors%

NODE: ipa / yum

%error: missing inputs for state changes for NODE ipa%

NODE: vodka / trash

%error: missing inputs for state changes for NODE vodka%

NODE: water / drive

water ipa 0001

water ipa 0011

water ipa 0111

water ipa 0101

water vodka 0000

%error: missing inputs for state changes for NODE water%

NODE: wine / class

wine coors 0000

Error: creating a NODE with a duplicate name

wine coors 0001

wine coors 0101

wine coors 0100

wine ipa 0010

%error: missing inputs for state changes for NODE wine%

NODE water / herp

name: water output: herp

ERROR: This input has already been declared water

ARC yogurt chili xxx

from: yogurt to: chili inputs: xxx

Error: creating ARC with undefined NODES

ERROR!! Either chili or yogurt are not in the array :

# Program Source Code

*/\*\**

*\* Derin Ozturk and Victor Barr*

*\* ECE3020 Project 3*

*\* Spring 2017*

*\* Derin: Write the code for generating the two output formats*

*\* Victor: Wrote the code for processing of the input statements*

*\*/*

*//----------INCLUDE STATEMENTS---------//*

#include <stdio.h>

#include <string.h>

#include <list>

#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <stdlib.h>

#include <cmath>

#include <bitset>

#include <string>

using namespace std;

*//----------STRUCT DECLARATIONS---------//*

struct nodeMealy {

nodeMealy(string name\_) : name(name\_) {}

nodeMealy(): name("NULL") {}

string name;

};

struct nodeMoore {

nodeMoore(string name\_, string outputs\_) : name(name\_), outputs(outputs\_) {}

nodeMoore(): name("NULL"), outputs("NULL") {}

string name;

string outputs;

};

struct edgeMealy {

edgeMealy(nodeMealy\* fromNode\_, nodeMealy\* toNode\_, long inputs\_, string outputs\_) :

fromNode(fromNode\_), toNode(toNode\_), inputs(inputs\_), outputs(outputs\_) {}

edgeMealy() : fromNode(NULL), toNode(NULL), inputs(0), outputs("NULL") {}

nodeMealy\* fromNode;

nodeMealy\* toNode;

long inputs;

string outputs;

};

struct edgeMoore {

edgeMoore(nodeMoore\* fromNode\_, nodeMoore\* toNode\_, long inputs\_) :

fromNode(fromNode\_), toNode(toNode\_), inputs(inputs\_) {}

edgeMoore() : fromNode(NULL), toNode(NULL), inputs(0) {}

nodeMoore\* fromNode;

nodeMoore\* toNode;

long inputs;

};

*//----------GLOBAL VARIABLES---------//*

int isMealy = 0;

int numStates = 25;

int numInputBits = 4;

char separator = ' ';

std::list<nodeMealy> mealyNodes;

std::list<nodeMoore> mooreNodes;

std::list<edgeMealy> mealyEdges;

std::list<edgeMoore> mooreEdges;

std::list<nodeMealy>::iterator mealyNodeIter;

std::list<nodeMoore>::iterator mooreNodeIter;

std::list<edgeMealy>::iterator mealyEdgeIter;

std::list<edgeMoore>::iterator mooreEdgeIter;

bool compareMealyName(const nodeMealy &lhs, const nodeMealy &rhs) {

int comp = (lhs.name).compare(rhs.name);

*//int comp = strcmp(lhs.name, rhs.name);*

if (comp < 0) {

return 1;

} else if (comp > 0) {

return 0;

}

}

bool compareMooreName(const nodeMoore &lhs, const nodeMoore &rhs) {

int comp = (lhs.name).compare(rhs.name);

*//int comp = strcmp(lhs.name, rhs.name);*

if (comp < 0) {

return 1;

} else if (comp > 0) {

return 0;

}

}

void printGraph() {

bool nonNull = 0;

if (isMealy) {

mealyNodes.sort(&compareMealyName);

for (mealyNodeIter = mealyNodes.begin(); mealyNodeIter != mealyNodes.end(); ++mealyNodeIter) {

cout << "NODE: " << mealyNodeIter->name << endl;

*//printf("NODE: %s\n", mealyNodeIter->name);*

int count = 0;

for (mealyEdgeIter = mealyEdges.begin(); mealyEdgeIter != mealyEdges.end(); ++mealyEdgeIter) {

if (mealyNodeIter->name == mealyEdgeIter->fromNode->name) {

cout << "\t" << mealyEdgeIter->fromNode->name << " " << mealyEdgeIter->toNode->name << " ";

*//NOTE I think this \n isnt suppose to be here*

*//printf("\t%s %s \n", mealyEdgeIter->fromNode->name, mealyEdgeIter->toNode->name);*

cout << std::bitset<4>(mealyEdgeIter->inputs).to\_string() << " / " << mealyEdgeIter->outputs << endl;

count++;

nonNull = 1;

}

}

if ((count != pow(numInputBits,2)) || !nonNull) {

cout << "%error: missing inputs for state changes for NODE " << mealyNodeIter->name << "%" << endl;

}

}

} else {

mooreNodes.sort(&compareMooreName);

for (mooreNodeIter = mooreNodes.begin(); mooreNodeIter != mooreNodes.end(); ++mooreNodeIter) {

cout << "NODE: " << mooreNodeIter->name << " / " << mooreNodeIter->outputs << endl;

*//printf("NODE: %s / %s\n", mooreNodeIter->name, mooreNodeIter->outputs);*

int count = 0;

for (mooreEdgeIter = mooreEdges.begin(); mooreEdgeIter != mooreEdges.end(); ++mooreEdgeIter) {

if (mooreNodeIter->name == mooreEdgeIter->fromNode->name) {

cout << "\t" << mooreEdgeIter->fromNode->name << " " << mooreEdgeIter->toNode->name << " ";

*//printf("\t%s %s ", mooreEdgeIter->fromNode->name, mooreEdgeIter->toNode->name);*

cout << std::bitset<4>(mooreEdgeIter->inputs).to\_string() << endl;

count++;

nonNull = 1;

}

}

if ((count != pow(numInputBits,2)) || !nonNull) {

cout << "%error: missing inputs for state changes for NODE " << mooreNodeIter->name << "%" << endl;

}

}

}

}

template<typename T> void printElement(T t, const int& width)

{

cout << left << setw(width) << setfill(separator) << t;

}

void printStateMachine() {

bool foundInput = 0;

if (isMealy) {

printElement("Curr State", 10);

printElement("|", 1);

printElement(" ", (pow(2,numInputBits) \* 10) - 10);

printElement("Next State/Output", 17);

cout << endl;

printElement("", 10);

printElement("|",1);

for (int i = 0; i < pow(2,numInputBits); i++) {

printElement(std::bitset<4>(i), 20);

}

cout << endl;

separator = '-';

printElement('-', pow(2,numInputBits) \* 20 + 10);

separator = ' ';

cout << endl;

mealyNodes.sort(&compareMealyName);

for (mealyNodeIter = mealyNodes.begin(); mealyNodeIter != mealyNodes.end(); ++mealyNodeIter) {

printElement(mealyNodeIter->name, 10);

printElement("|", 1);

for (int i = 0; i < pow(2,numInputBits); i++) {

for (mealyEdgeIter = mealyEdges.begin(); mealyEdgeIter!= mealyEdges.end(); ++mealyEdgeIter) {

if (i == mealyEdgeIter->inputs && mealyNodeIter->name.compare(mealyEdgeIter->fromNode->name) == 0) {

printElement(mealyEdgeIter->toNode->name, 8);

cout << " / ";

printElement(mealyEdgeIter->outputs, 9);

foundInput = 1;

break;

}

}

if (!foundInput) {

printElement(" ", 7);

printElement("X",1);

cout << " / ";

printElement("X",9);

}

foundInput = 0;

}

cout<<endl;

}

} else {

printElement("Curr State /", 16);

printElement("|", pow(2, numInputBits) \* 6 - 5);

printElement("Next State", 10);

cout << endl;

printElement("Output", 16);

printElement("|",1);

for (int i = 0; i < pow(2,numInputBits); i++) {

printElement(std::bitset<4>(i), 12);

}

cout << endl;

separator = '-';

printElement('-', pow(2,numInputBits) \* 12 + 16);

separator = ' ';

cout << endl;

mealyNodes.sort(&compareMealyName);

for (mooreNodeIter = mooreNodes.begin(); mooreNodeIter != mooreNodes.end(); ++mooreNodeIter) {

printElement(mooreNodeIter->name, 8);

cout << " / ";

printElement(mooreNodeIter->outputs, 5);

cout << "|";

for (int i = 0; i < pow(2,numInputBits); i++) {

for (mooreEdgeIter = mooreEdges.begin(); mooreEdgeIter!= mooreEdges.end(); ++mooreEdgeIter) {

if (i == mooreEdgeIter->inputs && mooreNodeIter->name.compare(mooreEdgeIter->fromNode->name) == 0) {

printElement(mooreEdgeIter->toNode->name, 12);

foundInput = 1;

break;

}

}

if (!foundInput) {

printElement("X",12);

}

foundInput = 0;

}

cout<<endl;

}

}

}

void setUp() {

std::cout << "Welcome to Derin and Victor's State Generator" << std::endl;

std::cout << "Press enter after each of the following input specifications" << std::endl;

std::cout << "How many states will you have? (INT 1 - 25)" << std::endl;

int numStates\_temp = 0;

std::cin >> numStates\_temp;

if (numStates\_temp <= 25 && numStates\_temp > 0) {

numStates = numStates\_temp; }

else {

std::cout << "Incorrect number of states" << std::endl;

std::exit(0);

}

std::cout << "How many input bits? (INT 1-4)" << std::endl;

int numInputBits\_temp = 0;

std::cin >> numInputBits\_temp;

if (numInputBits\_temp <= 4 && numInputBits\_temp > 0)

numInputBits = numInputBits\_temp;

else {

std::cout << "Incorrect number of input bits" << std::endl;

std::exit(0);

}

std::cout << "Mealy (Press 1) or Moore Machine (Press 0)?" << std::endl;

int isMealy\_temp = 0;

std::cin >> isMealy\_temp;

if (isMealy\_temp != 0 && isMealy\_temp != 1) {

std::cout << "Not a 0 or 1 input, exit program";

std::exit(0);

} else isMealy = isMealy\_temp;

}

void createGraph() {

string input;

if (isMealy) {

*//INTRO PRINT OUT STATEMENTS*

std::cout << "Begin adding NODE and ARC for MEALY State Machine" << endl;

std::cout << "NODE name" << " or ARC fromNode toNode inputs / outputs" << endl;

std::cout << "Type GRAPH to print graph and TABLE to print table" << endl;

std::cout << "Type DONE to finish" << endl;

*//DEFINE VARIABLES*

string option;

string name;

string outputs;

string from;

string to;

string inputs;

nodeMealy\* before = NULL;

nodeMealy\* after = NULL;

*//Begin parsing*

getline(cin, input);

while(input.compare("DONE")) {

int pos = 0;

pos = input.find(" ", pos); *//finds the location of the first space*

option = input.substr(0,pos); *//option is now the start of the string to the space*

if (option.compare("NODE") == 0) {

name = input.substr(pos + 1); *//name is now the rest of the string (after space 1)*

*//cout << "name is " << name << endl;*

nodeMealy nodeMealy\_ = nodeMealy(name);

*//cout << "goes through const" << endl;*

*//cout << nodeMealy\_.name << endl;*

int isValid = 1;

for (mealyNodeIter = mealyNodes.begin(); mealyNodeIter != mealyNodes.end(); ++mealyNodeIter) {

if ((mealyNodeIter->name).compare(name) == 0) {

isValid = 0;

}

}

if (isValid) {

mealyNodes.push\_back(nodeMealy\_);

*//cout << "goes through pushback" << endl;*

cout << mealyNodes.size() << endl;

*//cout << mealyNodes.back().name << endl;*

cout << nodeMealy\_.name << endl;

} else {

cout << "ERROR: This input has already been declared " << name << endl;

}

} else if (option.compare("ARC") == 0) {

input = input.substr(pos + 1); *//Update input from pos loc; "from to in / out"*

pos = input.find(" ", 0); *//finds location of space; index of space*

from = input.substr(0,pos); *//places from; "from"*

input = input.substr(pos + 1); *//Update input from pos loc*

pos = input.find(" ", 0); *//finds the next space*

to = input.substr(0,pos); *//sets this part to to*

input = input.substr(pos + 1); *//Update input from pos loc*

pos = input.find(" ", 0); *// ----*

inputs = input.substr(0,pos); *// ----*

input = input.substr(pos+3); *//Updates input to the end of the string (add 2) for '/ '*

outputs = input; *//Set this to outputs*

int count = 0;

char \* ptr;

cout << "from node: " << from << " to node: " << to << " inputs: " << inputs << " outputs: " << outputs << endl;

*/\*\**

*\* Check that the name of the arcs (to and from) exist in the std::list*

*\*/*

for (mealyNodeIter = mealyNodes.begin(); mealyNodeIter != mealyNodes.end(); ++mealyNodeIter) {

*//cout << mealyNodeIter->name << endl;*

if ((mealyNodeIter->name).compare(from) == 0) {

*//cout << "from is in arc" << endl;*

count++;

*//cout << &(\*mealyNodeIter) <<endl;*

before = &(\*mealyNodeIter);

*//cout << &(\*mealyNodeIter) <<endl;*

}

if ((mealyNodeIter->name).compare(to) == 0) {

*//cout << "to is in arc " << endl;*

count++;

after = &(\*mealyNodeIter);

}

}

if (count!=2) {

cout << "ERROR!! Either " << to << " or " << from <<" are not in the array :" << endl;

getline(cin, input);

continue; *//asks for next input*

}

*//cout << "now looking at the x possible values" << endl;*

*/\*\**

*\* now lets figure how many x's there are in the string*

*\*/*

const char\* inputsCONST = inputs.c\_str();

char inputsCopy[numInputBits];

strcpy(inputsCopy, inputsCONST);

int numX = 0;

int searchLoc = 0;

searchLoc = inputs.find('x',searchLoc);

while(searchLoc!=-1) {

numX++; *//tells me the size*

searchLoc = inputs.find('x',searchLoc + 1);

}

int size = pow(2,numX);

long numInput[size];

*//cout << "size of numInputs " << size << endl;*

edgeMealy edgeMealy\_ = edgeMealy();

if (numX == 1) {

char\* loc = strchr(inputsCopy,'x');

\*loc = '0';

numInput[0] = strtol(inputsCopy, &ptr, 2);

\*loc = '1';

numInput[1] = strtol(inputsCopy, &ptr, 2);

for (int i = 0; i < size; i++) {

edgeMealy\_ = edgeMealy(before, after, numInput[i], outputs);

mealyEdges.push\_back(edgeMealy\_);

}

} else if (numX == 2) {

char\* loc = strchr(inputsCopy,'x');

\*loc = '0';

char\* loc2 = strchr(inputsCopy,'x');

\*loc2 = '0';

numInput[0] = strtol(inputsCopy, &ptr, 2); *//00*

\*loc2 = '1';

numInput[1] = strtol(inputsCopy, &ptr, 2); *//01*

\*loc = '1';

numInput[2] = strtol(inputsCopy, &ptr, 2); *//11*

\*loc2 = '0';

numInput[3] = strtol(inputsCopy, &ptr, 2); *//10*

for (int i = 0; i < size; i++) {

edgeMealy\_ = edgeMealy(before, after, numInput[i], outputs);

mealyEdges.push\_back(edgeMealy\_);

}

} else if (numX == 3) {

char\* loc = strchr(inputsCopy,'x');

\*loc = '0';

char\* loc2 = strchr(inputsCopy,'x');

\*loc2 = '0';

char\* loc3 = strchr(inputsCopy,'x');

\*loc3 = '0';

numInput[0] = strtol(inputsCopy, &ptr, 2); *//000*

\*loc3 = '1';

numInput[1] = strtol(inputsCopy, &ptr, 2); *//001*

\*loc2 = '1';

numInput[2] = strtol(inputsCopy, &ptr, 2); *//011*

\*loc = '1';

numInput[3] = strtol(inputsCopy, &ptr, 2); *//111*

\*loc3 = '0';

numInput[4] = strtol(inputsCopy, &ptr, 2); *//110*

\*loc2 = '0';

numInput[5] = strtol(inputsCopy, &ptr, 2); *//100*

\*loc = '0';

\*loc2 = '1';

numInput[6] = strtol(inputsCopy, &ptr, 2); *//010*

\*loc = '1';

\*loc2 = '0';

\*loc3 = '1';

numInput[7] = strtol(inputsCopy, &ptr, 2); *//101*

for (int i = 0; i < size; i++) {

edgeMealy\_ = edgeMealy(before, after, numInput[i], outputs);

mealyEdges.push\_back(edgeMealy\_);

}

} else if (numX == 4) {

char\* loc = strchr(inputsCopy,'x');

\*loc = '0';

char\* loc2 = strchr(inputsCopy,'x');

\*loc2 = '0';

char\* loc3 = strchr(inputsCopy,'x');

\*loc3 = '0';

char\* loc4 = strchr(inputsCopy,'x');

\*loc4 = '0';

numInput[0] = strtol(inputsCopy, &ptr, 2); *//0000*

\*loc3 = '1';

numInput[1] = strtol(inputsCopy, &ptr, 2); *//0001*

\*loc2 = '1';

numInput[2] = strtol(inputsCopy, &ptr, 2); *//0011*

\*loc = '1';

numInput[3] = strtol(inputsCopy, &ptr, 2); *//0111*

\*loc3 = '0';

numInput[4] = strtol(inputsCopy, &ptr, 2); *//0110*

\*loc2 = '0';

numInput[5] = strtol(inputsCopy, &ptr, 2); *//0100*

\*loc = '0';

\*loc2 = '1';

numInput[6] = strtol(inputsCopy, &ptr, 2); *//0010*

\*loc = '1';

\*loc2 = '0';

\*loc3 = '1';

numInput[7] = strtol(inputsCopy, &ptr, 2); *//1101*

\*loc4 = '1';

numInput[8] = strtol(inputsCopy, &ptr, 2); *//1000*

\*loc3 = '1';

numInput[9] = strtol(inputsCopy, &ptr, 2); *//1001*

\*loc2 = '1';

numInput[10] = strtol(inputsCopy, &ptr, 2); *//1011*

\*loc = '1';

numInput[11] = strtol(inputsCopy, &ptr, 2); *//1111*

\*loc3 = '0';

numInput[12] = strtol(inputsCopy, &ptr, 2); *//1110*

\*loc2 = '0';

numInput[13] = strtol(inputsCopy, &ptr, 2); *//1100*

\*loc = '0';

\*loc2 = '1';

numInput[14] = strtol(inputsCopy, &ptr, 2); *//1010*

\*loc = '1';

\*loc2 = '0';

\*loc3 = '1';

numInput[15] = strtol(inputsCopy, &ptr, 2); *//1101*

for (int i = 0; i < size; i++) {

edgeMealy\_ = edgeMealy(before, after, numInput[i], outputs);

mealyEdges.push\_back(edgeMealy\_);

}

}

else {

numInput[0] = strtol(inputsCopy, &ptr, 2);

edgeMealy edgeMealy\_ = edgeMealy(before, after, numInput[0], outputs);

mealyEdges.push\_back(edgeMealy\_);

*//cout << mealyEdges.size() << endl;*

*//cout << mealyEdges.back().outputs << endl;*

*//cout << edgeMealy\_.outputs << endl;*

}

} else if (option.compare("GRAPH") == 0) {

cout << endl;

printGraph();

} else if (option.compare("TABLE") == 0) {

cout << endl;

printStateMachine();

cout << endl;

}

getline(cin, input);

}

}

else {

*//INTRO PRINT OUT STATEMENTS*

std::cout << "Begin adding NODE and ARC for MOORE State Machine" << endl;

std::cout << "NODE name / output" << " or ARC fromNode toNode inputs" << endl;

std::cout << "Type GRAPH to print graph and TABLE to print table" << endl;

std::cout << "Type DONE to finish" << endl;

*//DEFINE VARIABLES*

string option;

string name;

string outputs;

string from;

string to;

string inputs;

nodeMoore\* before = NULL;

nodeMoore\* after = NULL;

*//Begin parsing*

getline(cin, input);

while(input.compare("DONE")) {

int pos = 0;

pos = input.find(" ", pos); *//finds the location of the first space*

option = input.substr(0,pos); *//option is now the start of the string to the space*

if (option.compare("NODE") == 0) {

input = input.substr(pos + 1); *//Update input from pos loc*

pos = input.find(" ", 0); *//finds location of space*

name = input.substr(0,pos); *//places from*

input = input.substr(pos+3); *//Updates input to the end of the string (add 2) for '/ '*

outputs = input; *//Set this to outputs*

cout << "name: " << name << " output: " << outputs << endl;

nodeMoore nodeMoore\_ = nodeMoore(name, outputs);*//creates nodeMoore*

int isValid = 1;

for (mooreNodeIter = mooreNodes.begin(); mooreNodeIter != mooreNodes.end(); mooreNodeIter++) {

if ((mooreNodeIter->name).compare(name) == 0) {

isValid = 0;

}

}

if (isValid) {

mooreNodes.push\_back(nodeMoore\_);

cout << mooreNodes.size() << endl;

cout << mooreNodes.back().name << endl;

} else {

cout << "ERROR: This input has already been declared " << name << endl;

}

} else if (option.compare("ARC") == 0) {

input = input.substr(pos+1); *//Update input from pos loc*

pos = input.find(" ", 0); *//finds location of space*

from = input.substr(0,pos); *//places from*

input = input.substr(pos+1); *//Update input from pos loc*

pos = input.find(" ", 0); *//finds the next space*

to = input.substr(0,pos); *//sets this part to to*

input = input.substr(pos+1); *//Update input from pos loc*

inputs = input;

cout << "from: " << from << " to: " << to << " inputs: " << inputs << endl;

int count = 0;

char \* ptr;

*/\*\**

*\* Check that the name of the arcs (to and from) exist in the std::list*

*\*/*

for (mooreNodeIter = mooreNodes.begin(); mooreNodeIter != mooreNodes.end(); mooreNodeIter++) {

if ((mooreNodeIter->name).compare(from) == 0) {

count++;

before = &(\*mooreNodeIter);

}

if ((mooreNodeIter->name).compare(to) == 0) {

count++;

after = &(\*mooreNodeIter);

}

}

if (count!=2) {

cout << "ERROR!! Either " << to << " or " << from <<" are not in the array :" << endl;

getline(cin, input);

continue; *//asks for next input*

}

*/\*\**

*\* now lets figure how many x's there are in the string*

*\*/*

const char\* inputsCONST = inputs.c\_str();

char inputsCopy[numInputBits];

strcpy(inputsCopy, inputsCONST);

int numX = 0;

int searchLoc = 0;

searchLoc = inputs.find('x',searchLoc);

while(searchLoc!=-1) {

numX++; *//tells me the size*

searchLoc = inputs.find('x',searchLoc + 1);

}

int size = pow(2,numX);

long numInput[size];

edgeMoore edgeMoore\_ = edgeMoore();

if (numX == 1) {

char\* loc = strchr(inputsCopy,'x');

\*loc = '0';

numInput[0] = strtol(inputsCopy, &ptr, 2);

\*loc = '1';

numInput[1] = strtol(inputsCopy, &ptr, 2);

for (int i = 0; i < size; i++) {

edgeMoore\_ = edgeMoore(before, after, numInput[i]);

mooreEdges.push\_back(edgeMoore\_);

*//cout << mooreEdges.size() << endl;*

*//cout << mooreEdges.back().inputs << endl;*

}

} else if (numX == 2) {

char\* loc = strchr(inputsCopy,'x');

\*loc = '0';

char\* loc2 = strchr(inputsCopy,'x');

\*loc2 = '0';

numInput[0] = strtol(inputsCopy, &ptr, 2); *//00*

\*loc2 = '1';

numInput[1] = strtol(inputsCopy, &ptr, 2); *//01*

\*loc = '1';

numInput[2] = strtol(inputsCopy, &ptr, 2); *//11*

\*loc2 = '0';

numInput[3] = strtol(inputsCopy, &ptr, 2); *//10*

for (int i = 0; i < size; i++) {

edgeMoore\_ = edgeMoore(before, after, numInput[i]);

mooreEdges.push\_back(edgeMoore\_);

*//cout << mooreEdges.size() << endl;*

*//cout << mooreEdges.back().inputs << endl;*

}

} else if (numX == 3) {

char\* loc = strchr(inputsCopy,'x');

\*loc = '0';

char\* loc2 = strchr(inputsCopy,'x');

\*loc2 = '0';

char\* loc3 = strchr(inputsCopy,'x');

\*loc3 = '0';

numInput[0] = strtol(inputsCopy, &ptr, 2); *//000*

\*loc3 = '1';

numInput[1] = strtol(inputsCopy, &ptr, 2); *//001*

\*loc2 = '1';

numInput[2] = strtol(inputsCopy, &ptr, 2); *//011*

\*loc = '1';

numInput[3] = strtol(inputsCopy, &ptr, 2); *//111*

\*loc3 = '0';

numInput[4] = strtol(inputsCopy, &ptr, 2); *//110*

\*loc2 = '0';

numInput[5] = strtol(inputsCopy, &ptr, 2); *//100*

\*loc = '0';

\*loc2 = '1';

numInput[6] = strtol(inputsCopy, &ptr, 2); *//010*

\*loc = '1';

\*loc2 = '0';

\*loc3 = '1';

numInput[7] = strtol(inputsCopy, &ptr, 2); *//101*

for (int i = 0; i < size; i++) {

edgeMoore\_ = edgeMoore(before, after, numInput[i]);

mooreEdges.push\_back(edgeMoore\_);

*//cout << mooreEdges.size() << endl;*

*//cout << mooreEdges.back().inputs << endl;*

}

} else if (numX == 4) {

char\* loc = strchr(inputsCopy,'x');

\*loc = '0';

char\* loc2 = strchr(inputsCopy,'x');

\*loc2 = '0';

char\* loc3 = strchr(inputsCopy,'x');

\*loc3 = '0';

char\* loc4 = strchr(inputsCopy,'x');

\*loc4 = '0';

numInput[0] = strtol(inputsCopy, &ptr, 2); *//0000*

\*loc3 = '1';

numInput[1] = strtol(inputsCopy, &ptr, 2); *//0001*

\*loc2 = '1';

numInput[2] = strtol(inputsCopy, &ptr, 2); *//0011*

\*loc = '1';

numInput[3] = strtol(inputsCopy, &ptr, 2); *//0111*

\*loc3 = '0';

numInput[4] = strtol(inputsCopy, &ptr, 2); *//0110*

\*loc2 = '0';

numInput[5] = strtol(inputsCopy, &ptr, 2); *//0100*

\*loc = '0';

\*loc2 = '1';

numInput[6] = strtol(inputsCopy, &ptr, 2); *//0010*

\*loc = '1';

\*loc2 = '0';

\*loc3 = '1';

numInput[7] = strtol(inputsCopy, &ptr, 2); *//1101*

\*loc4 = '1';

numInput[8] = strtol(inputsCopy, &ptr, 2); *//1000*

\*loc3 = '1';

numInput[9] = strtol(inputsCopy, &ptr, 2); *//1001*

\*loc2 = '1';

numInput[10] = strtol(inputsCopy, &ptr, 2); *//1011*

\*loc = '1';

numInput[11] = strtol(inputsCopy, &ptr, 2); *//1111*

\*loc3 = '0';

numInput[12] = strtol(inputsCopy, &ptr, 2); *//1110*

\*loc2 = '0';

numInput[13] = strtol(inputsCopy, &ptr, 2); *//1100*

\*loc = '0';

\*loc2 = '1';

numInput[14] = strtol(inputsCopy, &ptr, 2); *//1010*

\*loc = '1';

\*loc2 = '0';

\*loc3 = '1';

numInput[15] = strtol(inputsCopy, &ptr, 2); *//101*

for (int i = 0; i < size; i++) {

edgeMoore\_ = edgeMoore(before, after, numInput[i]);

mooreEdges.push\_back(edgeMoore\_);

*//cout << mooreEdges.size() << endl;*

*//cout << mooreEdges.back().inputs << endl;*

}

}

else {

numInput[0] = strtol(inputsCopy, &ptr, 2);

edgeMoore edgeMoore\_ = edgeMoore(before, after, numInput[0]);

mooreEdges.push\_back(edgeMoore\_);

*//cout << mooreEdges.size() << endl;*

*//cout << mooreEdges.back().inputs << endl;*

}

} else if (option.compare("GRAPH") == 0) {

cout << endl;

printGraph();

} else if (option.compare("TABLE") == 0) {

cout << endl;

printStateMachine();

cout << endl;

}

getline(cin, input);

}

}

}

int main() {

*/\*\**

*\* Setup asks for iniital parameters.*

*\*/*

setUp();

*/\*\**

*\* Creates the graph and runtime environment for designing and manipulation*

*\*/*

createGraph();

return 0;

}