/\*

- ChhunJ\_prog1.cpp

-Programmed by James D. Chhun

-Spring 2018, 3/22/2018 -> 3/25/2018 (completion date)

-Class CS421 - Theory of Computing by Rocio

- NOTE

- There are two other files that this program depens on, 'Helpers.h' and "Helpers.cpp', both will be included

with online submission as well as hard copy source code

- DESCRIPTION

- This program parses through a file of email messages, these email messages are wrapped and formatted in XML tags.

Within the program, the email messages are parsed for their message ID as well as body message, individual characters of the body message

are then passed into the FiniteAutomaton function, where a Deterministic Finite Automaton is simulated. This DFA has 44 states,

state 44 is the Final state, once this state is reached the email message has been identified as a SPAM message. Because this is a spam message

for every character that follows after reaching the final state, the state transition stays trapped within the final state because it is has been accepted.

- If the final state has been reached, a vector containing the message IDs parsed from the messages earlier will either keep the ID or pop the ID

from it's contents, at the end of the program the contents of the vector are displayed, where each element in the vector

is a ID identifying a SPAM email

- If want to change the file to be parsed, within the main, change filename to "<anyFileName within same directory"

- NOTES, there is a 'Helpers.h' header file, this class is not required however it increases the modularity of the program as

- this main .cpp file is long and requires the enum States and State functions to be within it.

-README.md will be included

\*

/\*

Program charroduction up here

\*/

#include <iostream>

#include <fstream>

#include <string>

#include <stack>

#include <vector>

using namespace std;

enum State {

state\_start,

state\_f,

state\_fr,

state\_fre,

state\_free,

state\_free\_,

state\_free\_a,

state\_free\_ac,

state\_free\_acc,

state\_free\_acce,

state\_free\_access,

state\_free\_s,

state\_free\_so,

state\_free\_sof,

state\_free\_soft,

state\_free\_softw,

state\_free\_softwa,

state\_free\_softwar,

state\_free\_software,

state\_free\_v,

state\_free\_va,

state\_free\_vac,

state\_free\_vaca,

state\_free\_vacat,

state\_free\_vacati,

state\_free\_vacatio,

state\_free\_vacation,

state\_free\_t,

state\_free\_tr,

state\_free\_tri,

state\_free\_tria,

state\_free\_trial,

state\_free\_trials,

state\_w,

state\_wi,

state\_win,

state\_winn,

state\_winni,

state\_winnin,

state\_winning,

state\_winnings,

state\_winne,

state\_winner,

state\_winners,

//FINAL GOAL

state\_final

};

enum State start(char c); //same as state\_start

//Enum state prototypes handling the string 'free'

enum State f(char c);

enum State fr(char c);

enum State fre(char c);

enum State free(char c);

enum State free\_(char c);

//state prototypes handling free\_access

enum State free\_a(char c);

enum State free\_ac(char c);

enum State free\_acc(char c);

enum State free\_acce(char c);

enum State free\_access(char c);

//state prototypes handling ree\_software

enum State free\_s(char c);

enum State free\_so(char c);

enum State free\_sof(char c);

enum State free\_soft(char c);

enum State free\_softw(char c);

enum State free\_softwa(char c);

enum State free\_softwar(char c);

enum State free\_software(char c);

//state prototypes handling free\_vacation

enum State free\_v(char c);

enum State free\_va(char c);

enum State free\_vac(char c);

enum State free\_vaca(char c);

enum State free\_vacat(char c);

enum State free\_vacati(char c);

enum State free\_vacatio(char c);

enum State free\_vacation(char c);

//state prortypes free\_trials

enum State free\_t(char c);

enum State free\_tr(char c);

enum State free\_tri(char c);

enum State free\_tria(char c);

enum State free\_trial(char c);

enum State free\_trials(char c);

//FINAL

enum State final(char c);

/\*

- STATES free-win

\*/

enum State w(char c);

enum State wi(char c);

enum State win(char c);

/\*

- STATES free-winnings

\*/

enum State winn(char c);

enum State winni(char c);

enum State winnin(char c);

enum State winning(char c);

enum State winnings(char c);

/\*

- STATES free-winner(s)

\*/

enum State winne(char c);

enum State winner(char c);

enum State winners(char c);

//Main function prototypes

void printStateTransition(enum State currentState, char c);

//Finite Automata prototype

void FiniteAutomata(char c); //accept a intacter

//PRINT FUNCTIONS

void printState(enum State currentState); //function for testing/debug, will print currentState

void printVectIDs(vector < string > vectSpamIDs); // function will print the IDs of the emails identified as being SPAM

void printFinalStates(); //function to print the Final states as numbers to the user

//Truthy/Falsy functions, to determine what line is being read in the email

bool contains\_MSG\_ID(string lineFromFile); // check if <DOCID> and 'msg' is present in the string,

bool contains\_isStartOfEmail(string lineFromFile); //search for <DOC>

bool contains\_isNotSubject(string lineFromFile); //Check if 'Subject' is present, if so, return false to skip

bool contains\_isEndOfEmail(string lineFromFile); //check if </DOC>, end of the email

bool contains\_quotes(string lineFromFile); //check if string contains quotes, return false if doesnt, return true if does

//get MSGID Functions

string get\_MSG\_ID(string lineFromFile);

string get\_cleanString(string lineFromFile); //returns a string without double quotes

//parse BODY functions

//Global State variable, will represent and hold the current/final state the input reaches

enum State dfaState = state\_start;

//PRINT FUNCTIONS

void printState(enum State currentState) {

cout << "Showing State : " << currentState << endl;

}

void printVectIDs(vector < string > vect\_spamIDs) {

cout << endl << endl << "Printing IDs of Emails identified as SPAM" << endl;

for (int i = 0; i < vect\_spamIDs.size(); i++) {

cout << "spamID = " << vect\_spamIDs[i] << endl;

}

}

void printFinalStates() {

cout << "Called printFinalStates from Helper Class" << endl;

}

//FILE PARSING FUNCTIONS

bool contains\_MSG\_ID(string lineFromFile) {

string srch\_MSGID = "msg", srch\_DOCID = "</DOCID>";

int temp = -1, temp2 = -1;

temp = lineFromFile.find("msg"); //search for starting index of 'msg'

temp2 = lineFromFile.find("</DOCID>"); //search for DOCID string

if ((temp != -1) && (temp2 != -1)) {

return true;

}

return false;

}

bool contains\_isNotSubject(string lineFromFile) {

//Function will check if this line is the Subject:

int temp = -1;

temp = lineFromFile.find("Subject:"); //search for 'Subject'

if (temp != -1) {

return true;

} //false, there is no subject stirng

return false;

}

bool contains\_isEndOfEmail(string lineFromFile) {

//Function will check if this line is the end of the Email

int temp = -1;

temp = lineFromFile.find("</DOC>");

if (temp != -1) {

return true;

}

return false;

}

bool contains\_isStartOfEmail(string lineFromFile) {

//Function checks if starting line

int temp = -1;

temp = lineFromFile.find("<DOC>");

if (temp != -1) {

return true;

}

return false;

}

bool contains\_quotes(string lineFromFile) {

int temp = -1;

temp = lineFromFile.find('"');

if (temp == -1) {

return false;

}

return true;

}

//GET FUNCTIONS

string get\_MSG\_ID(string lineFromFile) {

//return a string containing the msgID

string temp = "";

int messageID = -1, docID = -1;

messageID = lineFromFile.find("msg"); //search for starting index of 'msg'

docID = lineFromFile.find("</DOCID>"); //search for DOCID string

temp = lineFromFile.substr(messageID + 3, docID - 1);

return temp; //temp is the messageID as a string

}

string get\_cleanString(string strCleanse) {

int position = -1;

do {

position = strCleanse.find('"');

strCleanse.erase(position, 1); //erase single character

} while (strCleanse.find('"') != -1);

return strCleanse;

}

/\* MAIN - Logic, Search for XML tags, perform appropriate action depending on which is read

- If <DOC> is read do nothing

- <DOCID> is read, expect MSGID

- 'Subject' read, do nothing, ignore line

- Else, then we should be reading body tag, perform LOGIC then

- </DOC> closing tag, then justify if the email is spam, perform LOGIC to show states reached

\*/

/\*2. Perform Logic to check if line contains msgID

- If so, then build a substring that contains the ID,

-Convert that substring into an integer and push into vector of SpamIDs

\*/

int main() {

vector < string > vectSpamIDs; //vector will contain the the IDs of the emails that are identified as SPAM

vector < State > vector\_statesReached; //vector to contain all transition functions / states reached, print at end of each body parse

string fileName = "messagefile.txt";

string lineFromFile = "", temp = "", tempString = ""; //tempString will build a string of body lines

int character; //to hold characters

ifstream file;

//START

file.open("messagefile.txt");

if (!file) {

cout << "File with name " + fileName + " does not exist!" << endl << endl;

} else {

while (!file.eof()) {

//1. Read Line from file

getline(file, lineFromFile);

if (contains\_isStartOfEmail(lineFromFile)) {} // DO NOTHING

else {

//Then the line is not wrapped in <DOCID> xml tags, proceed with logic.

if (contains\_MSG\_ID(lineFromFile)) {

//There is a msgID, now retrieve the ID and push into vector

temp = get\_MSG\_ID(lineFromFile);

vectSpamIDs.push\_back(temp);

} //END IF

else if (contains\_isNotSubject(lineFromFile)) {} //DO NOTHING, IGNORE SUBJECT LINE

//Final Check - is line </DOC> closing XML tag

else if (contains\_isEndOfEmail(lineFromFile)) {

int a = 0;

//1. tempString is accumulated Body message, now remove quotes from it then pass into Finite Automaton

//Search the string and remove all double quotes if any

while (contains\_quotes(tempString)) {

tempString = get\_cleanString(tempString);

}

//2. If string passes FA, keep spamID inside vector, else pop it

cout << endl << "Performing Finite Automata..." << endl;

printState(dfaState);

//Pass character into Finite Automaton

for (a = 0; a < tempString.length(); a++) {

FiniteAutomata(tempString[a]);

}

//3. Determine if the Email is SPAM, and keep it in vector or pop it

if (dfaState != state\_final) { //Once FA reaches state\_final, FA stays there, so this will hold and we can determine to pop or push the ID

//NOT SPAM POP FROM VECTOR

vectSpamIDs.pop\_back();

}

//4. reset tempString to be an emptyString for next emailParse

tempString = "";

a = 0;

dfaState = state\_start;

} else {

//Build a single string, this will handle cases where multiple/incompleted lines in the email

tempString += lineFromFile;

}

} //END ELSE

}

file.close(); //close file

}

//DEBUG - PRINT VECTORS

printVectIDs(vectSpamIDs);

//END

//system("pause"); //pause screen to show output

return 0;

}

void printStateTransition(enum State currentState, char c) {

cout << "Transition( q" << currentState << ", " << c << " ) = "; //to be finished within state Function

}

//FINITE AUTOMATA

void FiniteAutomata(char c) {

if (dfaState == state\_start) {

dfaState == start(c);

} else if (dfaState == state\_f) {

dfaState == f(c);

} else if (dfaState == state\_fr) {

dfaState == fr(c);

} else if (dfaState == state\_fre) {

dfaState = fre(c);

} else if (dfaState == state\_free) {

dfaState = free(c);

} else if (dfaState == state\_free\_) {

dfaState = free\_(c);

}

//state prototypes handling free\_access

else if (dfaState == state\_free\_a) {

dfaState = free\_a(c);

} else if (dfaState == state\_free\_ac) {

dfaState = free\_ac(c);

} else if (dfaState == state\_free\_acc) {

dfaState = free\_acc(c);

} else if (dfaState == state\_free\_acce) {

dfaState = free\_acce(c);

} else if (dfaState == state\_free\_access) {

dfaState = free\_access(c);

}

//state prototypes handling ree\_software

else if (dfaState == state\_free\_s) {

dfaState = free\_s(c);

} else if (dfaState == state\_free\_so) {

dfaState = free\_so(c);

} else if (dfaState == state\_free\_sof) {

dfaState = free\_sof(c);

} else if (dfaState == state\_free\_soft) {

dfaState = free\_soft(c);

} else if (dfaState == state\_free\_softw) {

dfaState = free\_softw(c);

} else if (dfaState == state\_free\_softwa) {

dfaState = free\_softwa(c);

} else if (dfaState == state\_free\_softwar) {

dfaState = free\_softwar(c);

} else if (dfaState == state\_free\_software) {

dfaState = free\_software(c);

}

//state prototypes handling free\_vacation

else if (dfaState == state\_free\_v) {

dfaState = free\_v(c);

} else if (dfaState == state\_free\_va) {

dfaState = free\_va(c);

} else if (dfaState == state\_free\_vac) {

dfaState = free\_vac(c);

} else if (dfaState == state\_free\_vaca) {

dfaState = free\_vaca(c);

} else if (dfaState == state\_free\_vacat) {

dfaState = free\_vacat(c);

} else if (dfaState == state\_free\_vacati) {

dfaState = free\_vacati(c);

} else if (dfaState == state\_free\_vacatio) {

dfaState = free\_vacatio(c);

} else if (dfaState == state\_free\_vacation) {

dfaState = free\_vacation(c);

}

//state prortypes free\_trials

else if (dfaState == state\_free\_t) {

dfaState = free\_t(c);

} else if (dfaState == state\_free\_tr) {

dfaState = free\_tr(c);

} else if (dfaState == state\_free\_tri) {

dfaState = free\_tri(c);

} else if (dfaState == state\_free\_tria) {

dfaState = free\_tria(c);

} else if (dfaState == state\_free\_trial) {

dfaState = free\_trial(c);

} else if (dfaState == state\_free\_trials) {

dfaState = free\_trials(c);

}

//WIN STATES

else if (dfaState == state\_w) {

dfaState = w(c);

} else if (dfaState == state\_wi) {

dfaState = wi(c);

} else if (dfaState == state\_win) {

dfaState = win(c);

}

//WINNINGS STATES

else if (dfaState == state\_winn) {

dfaState = winn(c);

} else if (dfaState == state\_winni) {

dfaState = winni(c);

} else if (dfaState == state\_winnin) {

dfaState = winnin(c);

} else if (dfaState == state\_winning) {

dfaState = winning(c);

} else if (dfaState == state\_winnings) {

dfaState = winnings(c);

}

//WINNER(S) STATES

else if (dfaState == state\_winne) {

dfaState = winne(c);

} else if (dfaState == state\_winner) {

dfaState = winner(c);

} else if (dfaState == state\_winners) {

dfaState = winners(c);

}

//FINAL

else if (dfaState == state\_final) {

//This was the goal, stop everything, the message has been identified as SPAM

dfaState = final(c);

}

}

//STATE BASED FUNCTIONS

enum State start(char c) {

printStateTransition(dfaState, c);

if (c == 'f') {

dfaState = state\_f;

cout << "q" << state\_f << endl;

} else if (c == 'w') {

dfaState = state\_w;

cout << "q" << state\_w << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl; //no change

}

return dfaState;

}

enum State f(char c) {

printStateTransition(dfaState, c);

if (c == 'r') {

dfaState = state\_fr;

cout << "q" << state\_fr << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State fr(char c) {

printStateTransition(dfaState, c);

if (c == 'e') {

dfaState = state\_fre;

cout << "q" << state\_fre << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State fre(char c) {

printStateTransition(dfaState, c);

if (c == 'e') {

dfaState = state\_free;

cout << "q" << state\_free << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free(char c) {

printStateTransition(dfaState, c);

if (c == ' ') {

dfaState = state\_free\_;

cout << "q" << state\_free\_ << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_(char c) {

printStateTransition(dfaState, c);

if (c == 'a') {

dfaState = state\_free\_a;

cout << "q" << state\_free\_a << endl;

} else if (c == 's') {

dfaState = state\_free\_s;

cout << "q" << state\_free\_s << endl;

} //move onto software test

else if (c == 'v') {

dfaState = state\_free\_v;

cout << "q" << state\_free\_v << endl;

} // move onto vacation test

else if (c == 't') {

dfaState = state\_free\_t;

cout << "q" << state\_free\_t << endl;

} //move onto trials test

else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

} //fail

return dfaState;

}

//state prototypes handling free\_access

enum State free\_a(char c) {

printStateTransition(dfaState, c);

if (c == 'c') {

dfaState = state\_free\_ac;

cout << "q" << state\_free\_ac << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_ac(char c) {

printStateTransition(dfaState, c);

if (c == 'c') {

dfaState = state\_free\_acc;

cout << "q" << state\_free\_acc << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_acc(char c) {

printStateTransition(dfaState, c);

if (c == 'e') {

dfaState = state\_free\_acce;

cout << "q" << state\_free\_acce << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_acce(char c) {

printStateTransition(dfaState, c);

if (c == 's') {

dfaState = state\_free\_access;

cout << "q" << state\_free\_access << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_access(char c) {

printStateTransition(dfaState, c);

//FINAL STATE //DONE

cout << "String 'free access' accepted" << endl;

dfaState = state\_final;

cout << "q" << state\_final << endl;

return dfaState;

}

//state prototypes handling ree\_software

enum State free\_s(char c) {

printStateTransition(dfaState, c);

if (c == 'o') {

dfaState = state\_free\_so;

cout << "q" << state\_free\_so << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_so(char c) {

printStateTransition(dfaState, c);

if (c == 'f') {

dfaState = state\_free\_sof;

cout << "q" << state\_free\_sof << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_sof(char c) {

printStateTransition(dfaState, c);

if (c == 't') {

dfaState = state\_free\_soft;

cout << "q" << state\_free\_soft << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_soft(char c) {

printStateTransition(dfaState, c);

if (c == 'w') {

dfaState = state\_free\_softw;

cout << "q" << state\_free\_softw << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_softw(char c) {

printStateTransition(dfaState, c);

if (c == 'a') {

dfaState = state\_free\_softwa;

cout << "q" << state\_free\_softwa << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_softwa(char c) {

printStateTransition(dfaState, c);

if (c == 'r') {

dfaState = state\_free\_softwar;

cout << "q" << state\_free\_softwar << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_softwar(char c) {

printStateTransition(dfaState, c);

if (c == 'e') {

dfaState = state\_free\_software;

cout << "q" << state\_free\_software << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_software(char c) {

printStateTransition(dfaState, c);

//FINAL STATE DONE

dfaState = state\_final;

cout << "q" << state\_final << endl;

return dfaState;

}

//state prototypes handling free\_vacation

enum State free\_v(char c) {

printStateTransition(dfaState, c);

if (c == 'a') {

dfaState = state\_free\_va;

cout << "q" << state\_free\_va << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_va(char c) {

printStateTransition(dfaState, c);

if (c == 'c') {

dfaState = state\_free\_vac;

cout << "q" << state\_free\_vac << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_vac(char c) {

printStateTransition(dfaState, c);

if (c == 'a') {

dfaState = state\_free\_vaca;

cout << "q" << state\_free\_vaca << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_vaca(char c) {

printStateTransition(dfaState, c);

if (c == 't') {

dfaState = state\_free\_vacat;

cout << "q" << state\_free\_vacat << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_vacat(char c) {

printStateTransition(dfaState, c);

if (c == 'i') {

dfaState = state\_free\_vacati;

cout << "q" << state\_free\_vacati << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_vacati(char c) {

printStateTransition(dfaState, c);

if (c == 'o') {

dfaState = state\_free\_vacatio;

cout << "q" << state\_free\_vacatio << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_vacatio(char c) {

printStateTransition(dfaState, c);

if (c == 'n') {

dfaState = state\_free\_vacation;

cout << "q" << state\_free\_vacation << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_vacation(char c) {

printStateTransition(dfaState, c);

//FINAL STATE, prompt

dfaState = state\_final;

cout << "q" << state\_final << endl;

return dfaState;

}

//state prortypes free\_trials

enum State free\_t(char c) {

printStateTransition(dfaState, c);

if (c == 'r') {

dfaState = state\_free\_tr;

cout << "q" << state\_free\_tr << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_tr(char c) {

printStateTransition(dfaState, c);

if (c == 'i') {

dfaState = state\_free\_tri;

cout << "q" << state\_free\_tri << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_tri(char c) {

printStateTransition(dfaState, c);

if (c == 'a') {

dfaState = state\_free\_tria;

cout << "q" << state\_free\_tria << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_tria(char c) {

printStateTransition(dfaState, c);

if (c == 'l') {

dfaState = state\_free\_trial;

cout << "q" << state\_free\_trial << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_trial(char c) {

printStateTransition(dfaState, c);

if (c == 's') {

dfaState = state\_free\_trials;

cout << "q" << state\_free\_trials << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State free\_trials(char c) {

printStateTransition(dfaState, c);

//FINAL PROMPT

cout << "q" << "String 'free trials' accepted" << endl << endl;

dfaState = state\_final;

cout << "q" << state\_final << endl;

return dfaState;

}

/\*

- STATES free-win

\*/

enum State w(char c) {

printStateTransition(dfaState, c);

if (c == 'i') {

dfaState = state\_wi;

cout << "q" << state\_wi << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State wi(char c) {

printStateTransition(dfaState, c);

if (c == 'n') {

dfaState = state\_win;

cout << "q" << state\_win << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State win(char c) {

printStateTransition(dfaState, c);

if (c == 'n') {

dfaState = state\_winn;

cout << "q" << state\_winn << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

/\*

- STATES free-winnings

\*/

enum State winn(char c) {

printStateTransition(dfaState, c);

if (c == 'i') {

dfaState = state\_winni;

cout << "q" << state\_winni << endl;

} else if (c == 'e') {

dfaState = state\_winne;

cout << "q" << state\_winne << endl;

} else {

dfaState = state\_start;

}

return dfaState;

}

enum State winni(char c) {

printStateTransition(dfaState, c);

if (c == 'n') {

dfaState = state\_winnin;

cout << "q" << state\_winnin << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State winnin(char c) {

printStateTransition(dfaState, c);

if (c == 'g') {

dfaState = state\_winning;

cout << "q" << state\_winning << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State winning(char c) {

printStateTransition(dfaState, c);

if (c == 's') {

dfaState = state\_winnings;

cout << "q" << state\_winnings << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State winnings(char c) {

printStateTransition(dfaState, c);

if (c == ' ') {

dfaState = state\_final;

cout << "q" << state\_final << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

/\*

- STATES free-winner(s)

\*/

enum State winne(char c) {

printStateTransition(dfaState, c);

if (c == 'r') {

dfaState = state\_winner;

cout << "q" << state\_winner << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State winner(char c) {

printStateTransition(dfaState, c);

if (c == ' ') {

dfaState = state\_final;

cout << "q" << state\_final << endl;

} else if (c == 's') {

dfaState = state\_winners;

cout << "q" << state\_winners << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

enum State winners(char c) {

printStateTransition(dfaState, c);

if (c == ' ') {

dfaState = state\_final;

cout << "q" << state\_final << endl;

} else {

dfaState = state\_start;

cout << "q" << state\_start << endl;

}

return dfaState;

}

//FINAL

enum State final(char c) {

printStateTransition(dfaState, c);

cout << "qFINAL or q" << state\_final << endl << endl;

dfaState = state\_final;

return dfaState;

}