/\*

Name: Karl Anthony James

E-Mail: kjames21@uco.edu

Assignment: a02 (TG)

Due: 9/25/2013

Class Time: Theory of Computing M/W 5:45 PM

\*/

#include <cstdlib>

#include <cstring>

#include <iostream>

#include <iomanip>

#include <string>

#include <fstream>

using namespace std;

#include "TG.h"

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

try{

std::string path;

std::string outputPath;

if(argc <= 1){

cout << "Enter the tg file path: ";

cin >> path;

}

else{

path = argv[1];

}

if(argc > 2){

outputPath = argv[2];

}

else{

outputPath = "tgOutput.dat";

}

TG tgMachine;

ifstream inputFile(path.c\_str());

if(!inputFile)

throw Exception("The tg file " + path + " could not be found");

ofstream outputFile(outputPath.c\_str());

if(!outputFile)

throw Exception("Could not open output file " + outputPath + ".");

tgMachine.ReadTG(inputFile);

tgMachine.EvaluateTG();

tgMachine.PrintResult(cout);

tgMachine.PrintResult(outputFile);

inputFile.close();

outputFile.close();

}

catch(...){

cout << endl << "An unexpected error occured. Ending program." << endl;

exit(EXIT\_FAILURE);

}

return 0;

}

/\*

Name: Karl Anthony James

E-Mail: kjames21@uco.edu

Assignment: a02 (TG)

Due: 9/25/2013

Class Time: Theory of Computing M/W 5:45 PM

\*/

#include <cstdlib>

#include <iostream>

#include <iomanip>

#include <fstream>

#include <map>

#include <vector>

using namespace std;

struct TGState {

map<int,int> stateMap;

};

struct TGMetadata {

string comment;

string alphabet;

int startState;

map<int,bool> stateSuccess;

map<string, TGState> tgMap;

};

struct TGEvaluation {

string input;

int state;

bool success;

};

struct Exception{

Exception(string message){

cout << endl << message << endl;

}

};

class TG {

public:

void ReadTG(ifstream& inputFile);

void EvaluateTG();

void PrintResult(ostream& output);

private:

string path;

TGMetadata tgMetadata;

vector<TGEvaluation> evaluations;

int evaluatePortion(int start, int end, int state, string& evaluation);

void FixNewLine(string &input);

bool WindowsSafeEmpty(string input);

};

/\*

Name: Karl Anthony James

E-Mail: kjames21@uco.edu

Assignment: a02 (TG)

Due: 9/25/2013

Class Time: Theory of Computing M/W 5:45 PM

\*/

#include <cstdlib>

#include <cstring>

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

#include <stdio.h>

using namespace std;

#include "TG.h"

void TG::ReadTG(ifstream& inputFile){

getline(inputFile, this->tgMetadata.comment);

getline(inputFile, this->tgMetadata.alphabet);

FixNewLine(this->tgMetadata.alphabet);//I've got to take into account the \r used in windows files.

int stateCount = 0;

int edgeCount = 0;

string stateMapLine;

inputFile >> stateCount;

getline(inputFile, stateMapLine);

inputFile >> edgeCount;

getline(inputFile, stateMapLine);

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

getline(inputFile, stateMapLine);

int alphaIndex = 0;

int s1,s2,j=0;

string triggerString = "";

string stateString = "";

while(stateMapLine[j] != ' ' && j < stateMapLine.size()){

stateString += stateMapLine[j];

j++;

}

FixNewLine(stateString);

if(!WindowsSafeEmpty(stateString)){

s1 = atoi(stateString.c\_str());

}

j++;

stateString = "";

while(stateMapLine[j] != ' ' && j < stateMapLine.size()){

stateString += stateMapLine[j];

j++;

}

FixNewLine(stateString);

if(!WindowsSafeEmpty(stateString)){

s2 = atoi(stateString.c\_str());

}

triggerString = stateMapLine.substr(++j);

FixNewLine(triggerString);

if(this->tgMetadata.tgMap.find(triggerString) == this->tgMetadata.tgMap.end()){

TGState state;

state.stateMap.insert(pair<int,int>(s1,s2));

this->tgMetadata.tgMap.insert(pair<string,TGState>(triggerString,state));

}

else{

if(this->tgMetadata.tgMap[triggerString].stateMap.find(s1) == this->tgMetadata.tgMap[triggerString].stateMap.end()){

this->tgMetadata.tgMap[triggerString].stateMap.insert(pair<int,int>(s1,s2));

}

}

}

string startStateString;

getline(inputFile, startStateString);

this->tgMetadata.startState = atoi(startStateString.c\_str());

this->tgMetadata.stateSuccess = map<int,bool>();

for(int i = 1; i <= stateCount; i++){

this->tgMetadata.stateSuccess.insert(std::pair<int,bool>(i,false));

}

string numGoodStatesString;

int numGoodStates;

string goodStateDesc;

getline(inputFile, goodStateDesc);

for(int i = 0; i < goodStateDesc.find(' '); i++){

numGoodStatesString += goodStateDesc[i];

}

numGoodStates = atoi(numGoodStatesString.c\_str());

for(int i = goodStateDesc.find(' ') + 1; i < goodStateDesc.size(); i++){

string stateString = "";

while(goodStateDesc[i] != ' ' && i < goodStateDesc.size()){

stateString += goodStateDesc[i];

i++;

}

if(!WindowsSafeEmpty(stateString)){

int state = atoi(stateString.c\_str());

this->tgMetadata.stateSuccess[state] = true;

}

}

string testLine;

while(getline(inputFile, testLine)){

FixNewLine(testLine);

if(WindowsSafeEmpty(testLine)) continue;

TGEvaluation eval;

eval.input = testLine;

eval.state = this->tgMetadata.startState;

eval.success = false;

this->evaluations.push\_back(eval);

}

}

void TG::EvaluateTG(){

for(int i = 0; i < this->evaluations.size(); i++){

TGEvaluation &eval = this->evaluations[i];

eval.state = this->tgMetadata.startState;

int evaluatedState = this->tgMetadata.startState;

for(int j = 1; j <= eval.input.size(); j++){

evaluatedState = this->evaluatePortion(0, j, eval.state, eval.input);

if(this->tgMetadata.stateSuccess[evaluatedState])

break;

}

if(evaluatedState > 0){

eval.success = this->tgMetadata.stateSuccess[evaluatedState] ? true : false;

}

else{

eval.success = false;

}

}

}

int TG::evaluatePortion(int start, int end, int state, string& evaluation){

if(start >= evaluation.size())

return state;

string snippet = evaluation.substr(start, end-start);

if(this->tgMetadata.tgMap.find(snippet) != this->tgMetadata.tgMap.end()

&& this->tgMetadata.tgMap[snippet].stateMap.find(state) != this->tgMetadata.tgMap[snippet].stateMap.end()){

int newState = this->tgMetadata.tgMap[snippet].stateMap[state];

int childState = newState;

for(int i = end+1; i <= evaluation.size(); i++){

childState = this->evaluatePortion(end, i, newState, evaluation);

if(childState > 0 && this->tgMetadata.stateSuccess[childState])

return childState;

}

return childState;

}

else{

return 0;

}

}

void TG::PrintResult(ostream &output){

for(int i = 0; i < this->evaluations.size(); i++){

output << this->evaluations[i].input << endl;

output << (this->evaluations[i].success ? "ACCEPT" : "REJECT") << endl;

output << endl;

}

}

void TG::FixNewLine(string &input){

if(input != ""){

//Check for the newline character

if(input[0] == '\n') input = input.substr(1);

if(input[input.size()-1] == '\n') input = input.substr(0, input.size()-1);

//Check for the \r (sometimes used in windows)

if(input[0] == '\r') input = input.substr(1);

if(input[input.size()-1] == '\r') input = input.substr(0, input.size()-1);

}

}

bool TG::WindowsSafeEmpty(string input){

if(input.empty() || input == "" || input.size() == 0 || input == "\n" || input == "\r")

return true;

else

return false;

}

Code located at: /home/ws/ws044/AssignmentTwo

