YunHo(Louis) Law

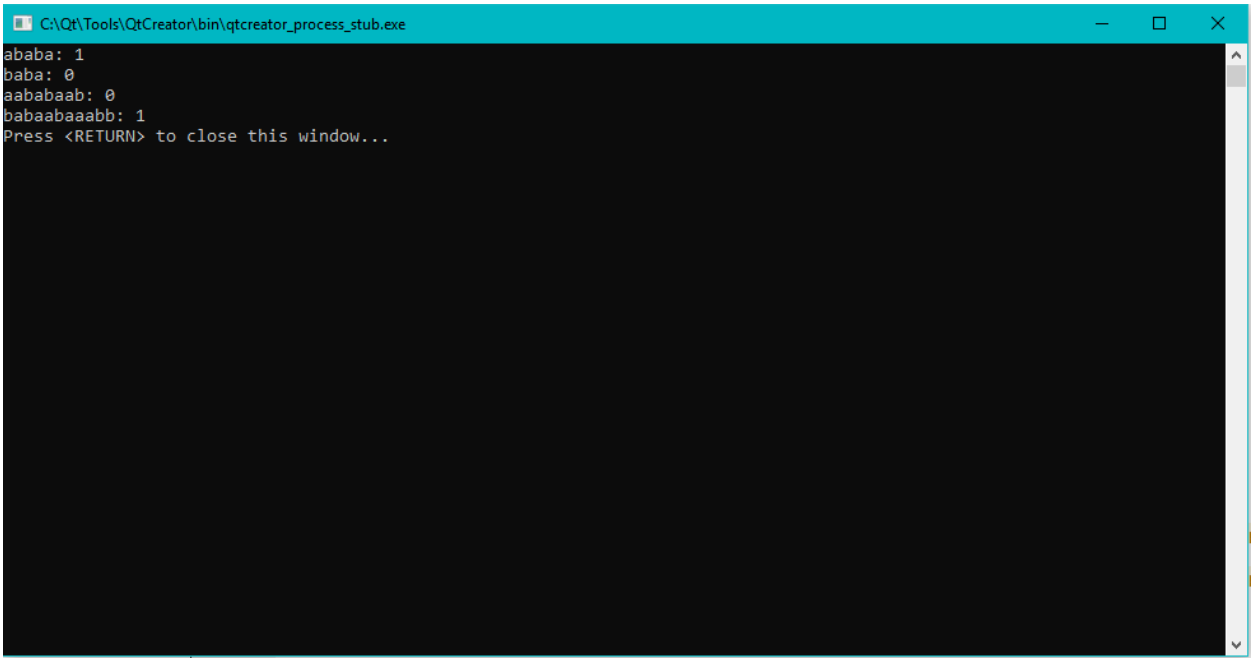
CS 361

Lab 4 report

DFA Code:

```
class DFA{
private:
    // five tuples
    vector<int> state;
    int startState ;
    vector<int> acceptState;
    vector<char> input;
    int** location;
public:
    //constructor takes 5 tuples input
    DFA(vector<int> stateSet, int s, vector<int> endStates, vector<char> i, int** transition){
        this->state = stateSet;
        this->startState = s;
        this->acceptState = endStates;
        this->input = i;
        this->location = new int* [state.size()];
        // initialize the table
        for(int j = 0; j < stateSet.size(); j++){                    comparison of integers of different signs: '
            location[j] = new int [i.size()];
            for(int k = 0; k < i.size(); k++){                       comparison of integers of different signs: '
                location[j][k] = transition[j][k];
            }
        }
    }
    bool DFAmatcher(char pattern[], int patternSize){
        int currentstate = startState;
        for(int i = 0; i < patternSize; i++){ // check the entire string by character
            int charlocation;
            for(int a = 0; a < input.size(); a++){ //check the column location of the transition table by looping the whole input set
                if(input[a] == pattern[i]){                          implicit conversion changes signedness:
                    charlocation = a;
                    currentstate = location[currentstate][charlocation]; //move state
                }
            }
        }
        for(int j = 0; j < acceptState.size(); j++){ //check if current state is on any of the accept state    comparison of integers
            if(currentstate == acceptState.at(j)){                   implicit conversion changes signedness:
                return true;
            }
        }
        return false; //return false w/ no matches found
    }
};
```

DFA Output:



I used the pseudocode from the book as reference to write my DFA class.

I found this algorithm is very difficult to implement in C++ since there the transition table plus the input set, states set, and the accept states set make it really complexed. I assume there are some libraries and function that would help me to optimize the code and make it look way better than this.

Bellman-Ford Code:

```cpp
#include <iostream>
#include <list>
#include <vector>
#include <set>

using namespace std;

struct vertex{
    int name;
    int distance;
    int predecessor;
};

class graph{
    vector<vertex> v;
    int** adjacencyMatrix;
public:
    graph(vector<vertex> ver, int** adj){
        this->v = ver;
        this->adjacencyMatrix = new int*[ver.size()];
        for(int i = 0; i ≤ ver.size(); i++){
            adjacencyMatrix[i] = new int[ver.size()];
            for(int j = 0; j ≤ ver.size(); j++){
                adjacencyMatrix[i][j] = adj[i][j];
            }
        }
    }
    void relax(vertex u, vertex v){
        if(v.distance > u.distance + adjacencyMatrix[u.name][v.name]){
            v.distance = u.distance + adjacencyMatrix[u.name][v.name];
            v.predecessor = u.name;
        }
    }
    bool BellmanFord(vertex start){
        start.distance = 0;
        for(int a = 0; a ≤ v.size(); a++){
            for(int i = 0; i ≤ v.size(); i++){
                for(int j = 0; j ≤ v.size(); j++){
                    if(adjacencyMatrix[i][j] != INT_MAX){
                        relax(v[i],v[j]);    implicit conversion changes signedness: 'int'
                    }
                }
            }
```

```
    }
    void relax(vertex u, vertex v){
        if(v.distance > u.distance + adjacencyMatrix[u.name][v.name]){
            v.distance = u.distance + adjacencyMatrix[u.name][v.name];
            v.predecessor = u.name;
        }
    }
    bool BellmanFord(vertex start){
        start.distance = 0;
        for(int a = 0; a ≤ v.size(); a++){                                  ⚠ comparis
            for(int i = 0; i ≤ v.size(); i++){                              ⚠ comparis
                for(int j = 0; j ≤ v.size(); j++){                          ⚠ comparis
                    if(adjacencyMatrix[i][j] != INT_MAX){
                        relax(v[i],v[j]);     ⚠ implicit conversion changes signedness: 'int' to 'st
                    }
                }
            }
        }
        for(int i = 0; i ≤ v.size(); i++){                                  ⚠ comparis
            for(int j = 0; j ≤ v.size(); j++){                              ⚠ comparis
                if(adjacencyMatrix[i][j] != INT_MAX){
                    if(v[i].distance > v[j].distance + adjacencyMatrix[i][j]){     ⚠ implicit conver
                        return false;
                    }
                }
            }
        }
        return true;
    }
    void printGraph(){
        for(auto it = v.begin(); it != v.end(); ++it){
            cout << it->name << endl;
            cout << "distance: " << it->distance << endl;
            cout << "predecessor: " << it->predecessor << endl;
        }
    }
};
```