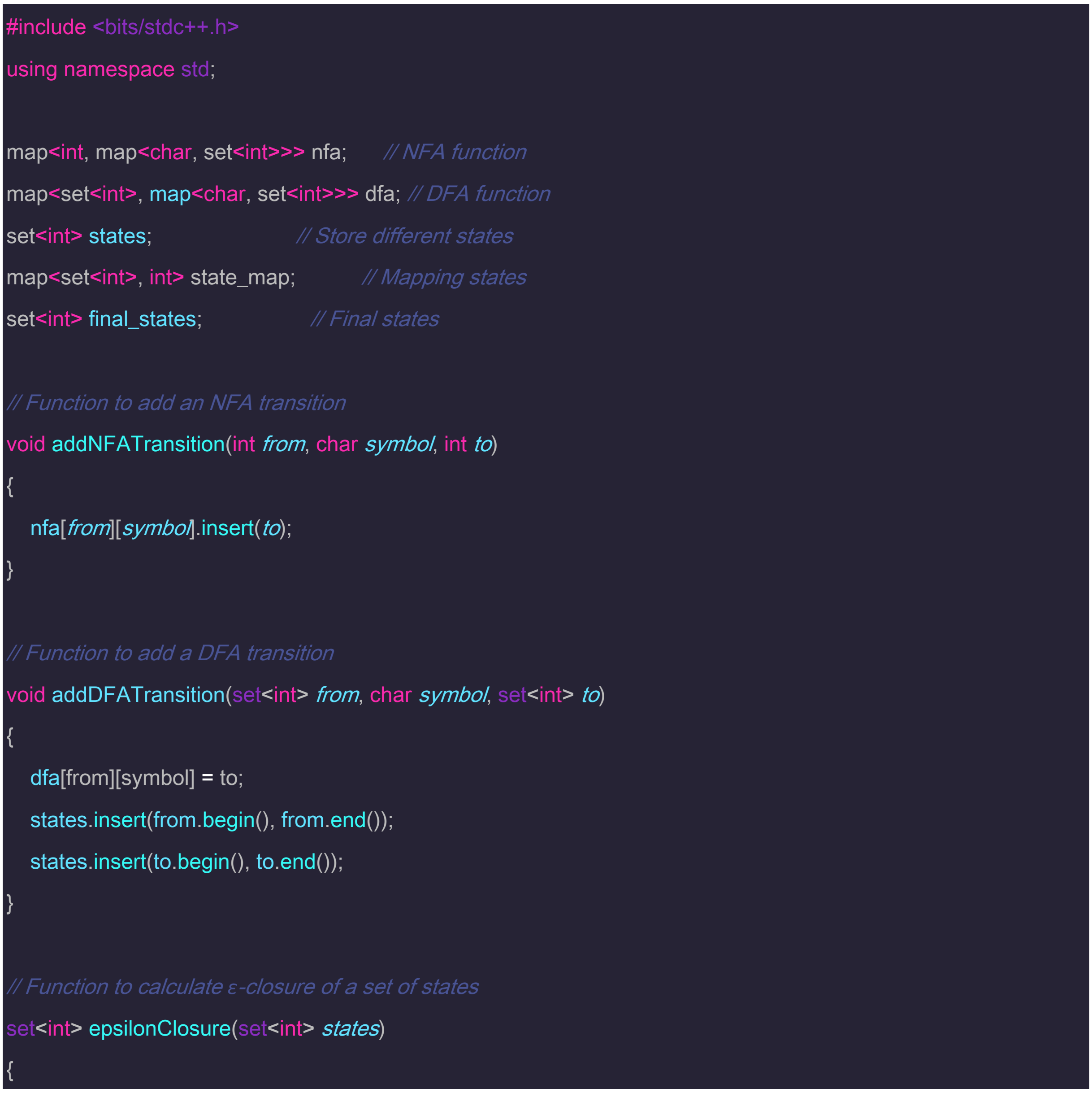
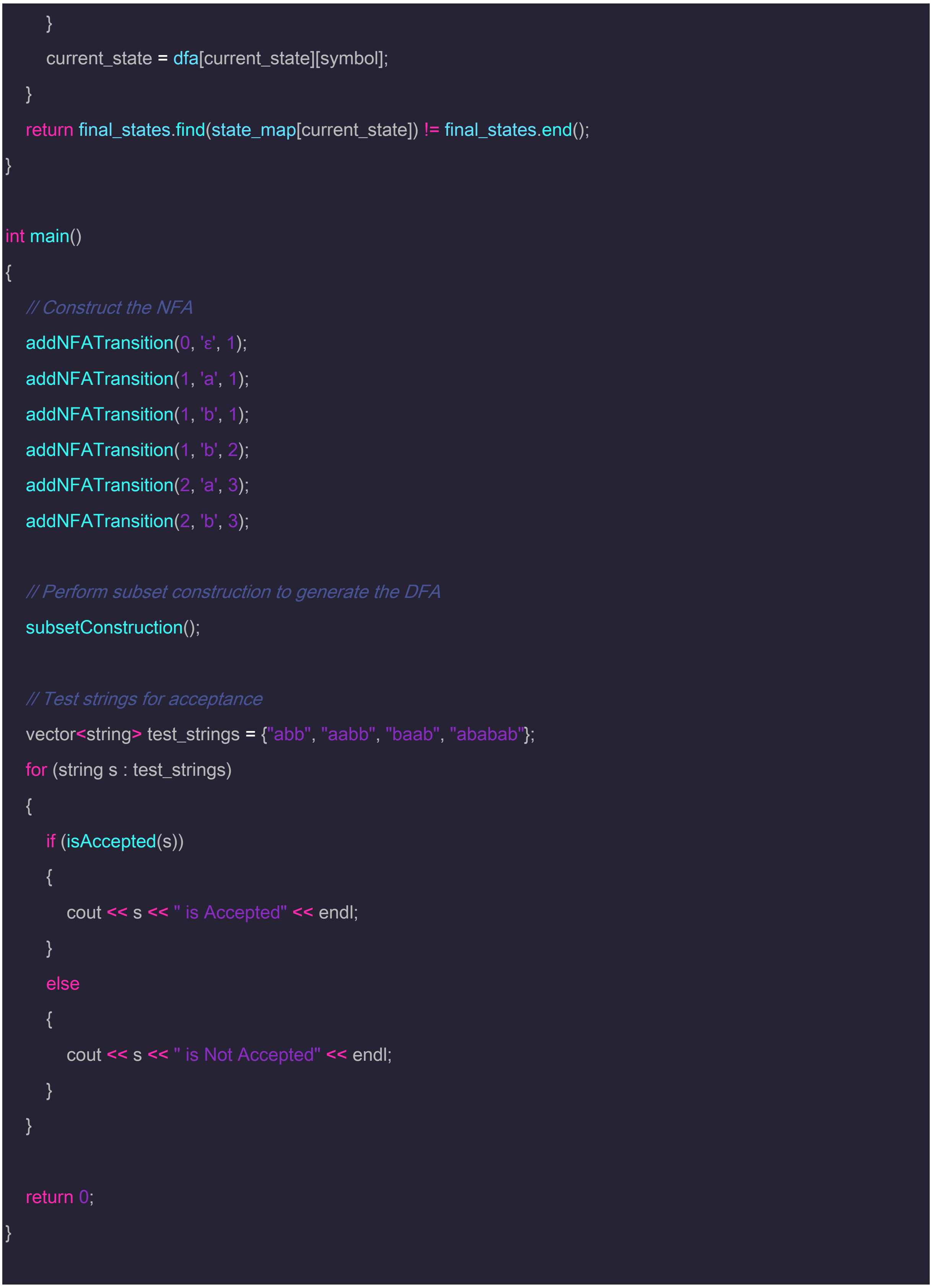
COMPILER CONSTRUCTION (UCS802)
ASSIGNMENT 1

RISHAB CHAKRABARTI
4CO26
102003688

Q1. Design a Minimized DFA for the Regular Expression (a/b)*abb i.e. All strings ending with abb.

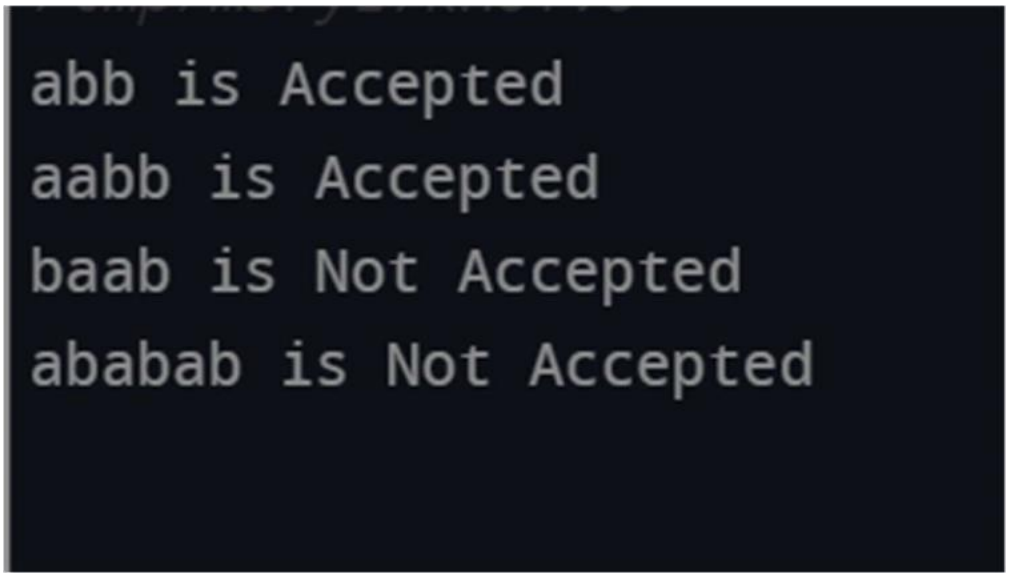


```
#include <bits/stdc++.h>
using namespace std;

map<int, map<char, set<int>>> nfa;      // NFA function
map<set<int>, map<char, set<int>>> dfa; // DFA function
set<int> states;                        // Store different states
map<set<int>, int> state_map;           // Mapping states
set<int> final_states;                  // Final states

// Function to add an NFA transition
void addNFATransition(int from, char symbol, int to)
{
    nfa[from][symbol].insert(to);
}

// Function to add a DFA transition
void addDFATransition(set<int> from, char symbol, set<int> to)
{
    dfa[from][symbol] = to;
    states.insert(from.begin(), from.end());
    states.insert(to.begin(), to.end());
}

// Function to calculate ε-closure of a set of states
set<int> epsilonClosure(set<int> states)
{
    set<int> result = states;
    queue<int> q;
    for (int state : states)
    {
        q.push(state);
    }
    while (!q.empty())
    {
        int current = q.front();
        q.pop();
        for (int next : nfa[current]['ε'])
        {
            if (result.find(next) == result.end())
            {
                result.insert(next);
                q.push(next);
            }
        }
    }
    return result;
}

// Function to perform subset construction to convert NFA to DFA
void subsetConstruction()
{
    queue<set<int>> unmarked_states;
    set<int> start_state = epsilonClosure({0});
    state_map[start_state] = 0;
    unmarked_states.push(start_state);

    while (!unmarked_states.empty())
    {
        set<int> current_state = unmarked_states.front();
        unmarked_states.pop();
        for (char symbol : {'a', 'b'})
        {
            set<int> next_state;
            for (int state : current_state)
            {
                for (int next : nfa[state][symbol])
                {
                    next_state.insert(next);
                }
            }
            next_state = epsilonClosure(next_state);
            if (next_state.empty())
                continue;
            if (state_map.find(next_state) == state_map.end())
            {
                int state_id = state_map.size();
                state_map[next_state] = state_id;
                unmarked_states.push(next_state);
            }
            addDFATransition(current_state, symbol, next_state);
        }
    }

    // Identify final states in the DFA
    for (auto entry : state_map)
    {
        for (int state : entry.first)
        {
            if (state == 3)
            { // 3 is the accepting state in the NFA
                final_states.insert(entry.second);
                break;
            }
        }
    }
}

// Function to check if a string is accepted by the DFA
bool isAccepted(string input)
{
    set<int> current_state = epsilonClosure({0});
    for (char symbol : input)
    {
        if (dfa[current_state].find(symbol) == dfa[current_state].end())
        {
            return false;
```



```
        }
        current_state = dfa[current_state][symbol];
    }
    return final_states.find(state_map[current_state]) != final_states.end();
}

int main()
{
    // Construct the NFA
    addNFATransition(0, 'ε', 1);
    addNFATransition(1, 'a', 1);
    addNFATransition(1, 'b', 1);
    addNFATransition(1, 'b', 2);
    addNFATransition(2, 'a', 3);
    addNFATransition(2, 'b', 3);

    // Perform subset construction to generate the DFA
    subsetConstruction();

    // Test strings for acceptance
    vector<string> test_strings = {"abb", "aabb", "baab", "ababab"};
    for (string s : test_strings)
    {
        if (isAccepted(s))
        {
            cout << s << " is Accepted" << endl;
        }
        else
        {
            cout << s << " is Not Accepted" << endl;
        }
    }

    return 0;
}
```

OUTPUT



```
abb is Accepted
aabb is Accepted
baab is Not Accepted
ababab is Not Accepted
```