**Name: Sparsh Karna**

**Reg Number: 23BDS1172**

**Lab Excercise: 5**

**Aim:**

To develop a C++ program that simulates a Deterministic Finite Automaton (DFA). The program will evaluate whether a given input string is accepted by the DFA, based on its defined states, transition function, initial state, and set of accepting (final) states.

**Procedure:**

1. Define the DFA with the following components:
   1. A finite set of states.
   2. An input alphabet.
   3. A transition function (state, symbol) → next\_state.
   4. An initial state.
   5. A set of accepting states.
2. Represent the transition function using a suitable data structure (e.g., map or table).
3. Read an input string from the user.
4. Start from the initial state and process the string symbol by symbol:
   1. For each symbol, look up the next state using the transition function.
   2. If no valid transition exists, reject the string.
5. After processing the entire string, check if the current state is one of the accepting states.
6. Print whether the string is **ACCEPTED** or **REJECTED**.

**Algorithm:**

1. Define the DFA with states, start state, final states, and transitions.
2. Take an input string.
3. Initialize the current state as the start state.
4. For each character in the string:
5. If a transition exists for (current\_state, character), move to the next state.
6. Else, reject the string.
7. After the string is processed, if the current state ∈ final states, accept the string.
8. Otherwise, reject it.

**Program:**

#include <bits/stdc++.h>

using namespace std;

class DFA

{

int numStates; // Total number of states

int initialState; // Starting state

vector<int> finalStates; // Accepting states

map<pair<int, char>, int> transition; // Transition function

public:

DFA(int states, int start, vector<int> finals)

{

numStates = states;

initialState = start;

finalStates = finals;

}

void addTransition(int from, char symbol, int to)

{

transition[{from, symbol}] = to;

}

bool isAccepted(const string &input)

{

int currentState = initialState;

for (char symbol : input)

{

if (transition.find({currentState, symbol}) == transition.end())

{

return false; // No valid transition → reject

}

currentState = transition[{currentState, symbol}];

}

// Check if current state is in final states

return find(finalStates.begin(), finalStates.end(), currentState) != finalStates.end();

}

};

int main()

{

// Example DFA: Accepts strings over {a,b} ending with "ab"

int numStates = 3;

int startState = 0;

vector<int> finalStates = {2};

DFA dfa(numStates, startState, finalStates);

// Define transitions

dfa.addTransition(0, 'a', 1);

dfa.addTransition(0, 'b', 0);

dfa.addTransition(1, 'a', 1);

dfa.addTransition(1, 'b', 2);

dfa.addTransition(2, 'a', 1);

dfa.addTransition(2, 'b', 0);

string input;

cout << "Enter input string (over {a,b}): ";

cin >> input;

if (dfa.isAccepted(input))

{

cout << "✅ String ACCEPTED by DFA" << endl;

}

else

{

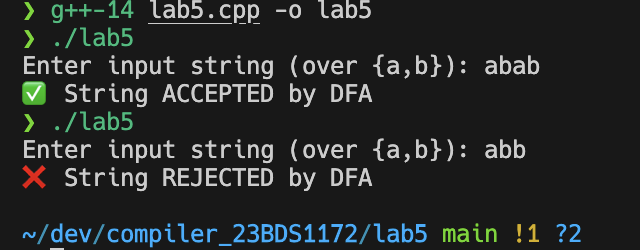
cout << "❌ String REJECTED by DFA" << endl;

}

return 0;

}

**Output:**



**Result:**

The program successfully simulates a DFA and validates input strings based on the defined states, transitions, initial state, and accepting states. It correctly identifies whether the given string is accepted or rejected by the automaton.