Name: Niall Francis Ajeet Dcunha

Reg No: 23BCE1985

Topic: Compiler Design Lab1

**Experiment-1**    Implementation of Deterministic Finite Automaton (DFA) from regular grammar using C language.

**Input:** The set of states (Q), set of input alphabet (T), set of final states (F) and the set

  of transitions (Delta) and a terminal string (w).

**Output:** Transition table and the sequence of transition to accept w.

**Code:** #include <stdio.h>

#include <string.h>

#define MAX 100

int numStates, numSymbols;

char states[MAX][10], symbols[MAX];

char transition[MAX][MAX][10];

char finalStates[MAX][10];

char startState[10];

int getStateIndex(char \*state) {

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

if (strcmp(states[i], state) == 0)

return i;

}

return -1;

}

int getSymbolIndex(char symbol) {

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

if (symbols[i] == symbol)

return i;

}

return -1;

}

int isFinalState(char \*state) {

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

if (strcmp(finalStates[i], state) == 0)

return 1;

}

return 0;

}

int main() {

char w[100];

char currentState[10];

scanf("%d", &numStates);

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

scanf("%s", states[i]);

}

scanf("%d", &numSymbols);

scanf("%s", symbols);

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

for (int j = 0; j < numSymbols; j++) {

scanf("%s", transition[i][j]);

}

}

int numFinal;

scanf("%d", &numFinal);

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

scanf("%s", finalStates[i]);

}

scanf("%s", startState);

scanf("%s", w);

strcpy(currentState, startState);

printf("\nTransition sequence:\n%s", currentState);

for (int i = 0; i < strlen(w); i++) {

int symbolIndex = getSymbolIndex(w[i]);

if (symbolIndex == -1) {

printf("\nInvalid input symbol: %c\n", w[i]);

return 1;

}

int stateIndex = getStateIndex(currentState);

if (stateIndex == -1) {

printf("\nInvalid current state: %s\n", currentState);

return 1;

}

strcpy(currentState, transition[stateIndex][symbolIndex]);

printf(" -> %s", currentState);

}

if (isFinalState(currentState)) {

printf("\n\nResult: String accepted!\n");

} else {

printf("\n\nResult: String rejected.\n");

}

// Print transition table

printf("\nTransition Table:\n ");

for (int j = 0; j < numSymbols; j++)

printf(" %c", symbols[j]);

printf("\n");

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

printf("%s ", states[i]);

for (int j = 0; j < numSymbols; j++) {

printf(" %3s", transition[i][j]);

}

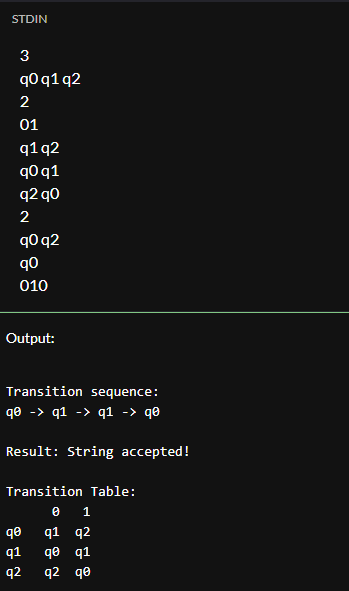
printf("\n");

}

return 0;

}

**Output:**



**Experiment-2**    Implement a C program to find a regular grammar from Deterministic Finite Automaton (DFA).

**Input:** The set of states (Q), set of input alphabet (T), set of final states (F) and    Transition table.

**Output:** The grammar G =(N, T, P , S).

**Code:**

#include <stdio.h>

#include <string.h>

#define MAX 100

int numStates, numSymbols;

char states[MAX][10];

char alphabet[MAX];

char transitions[MAX][MAX][10]; // transitions[state][symbol] = next\_state

char finalStates[MAX][10];

char startState[10];

int getStateIndex(char \*state) {

for (int i = 0; i < numStates; i++)

if (strcmp(states[i], state) == 0)

return i;

return -1;

}

int isFinalState(char \*state) {

for (int i = 0; i < numStates; i++)

if (strcmp(finalStates[i], state) == 0)

return 1;

return 0;

}

int main() {

int numFinal;

scanf("%d", &numStates);

for (int i = 0; i < numStates; i++)

scanf("%s", states[i]);

scanf("%d", &numSymbols);

for (int i = 0; i < numSymbols; i++)

scanf(" %c", &alphabet[i]);

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

for (int j = 0; j < numSymbols; j++) {

// printf("δ(%s, %c) = ", states[i], alphabet[j]);

scanf("%s", transitions[i][j]);

}

}

scanf("%d", &numFinal);

for (int i = 0; i < numFinal; i++)

scanf("%s", finalStates[i]);

scanf("%s", startState);

printf("\nRegular Grammar G = (N, T, P, S)\n");

printf("N = { ");

for (int i = 0; i < numStates; i++)

printf("%s ", states[i]);

printf("}\n");

printf("T = { ");

for (int i = 0; i < numSymbols; i++)

printf("%c ", alphabet[i]);

printf("}\n");

printf("S = %s\n", startState);

// Production rules P

printf("P = {\n");

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

for (int j = 0; j < numSymbols; j++) {

char \*from = states[i];

char symbol = alphabet[j];

char \*to = transitions[i][j];

printf(" %s -> %c%s", from, symbol, to);

if (isFinalState(to)) {

printf(" | %c", symbol); // Add terminal-only production for final state

}

printf("\n");

}

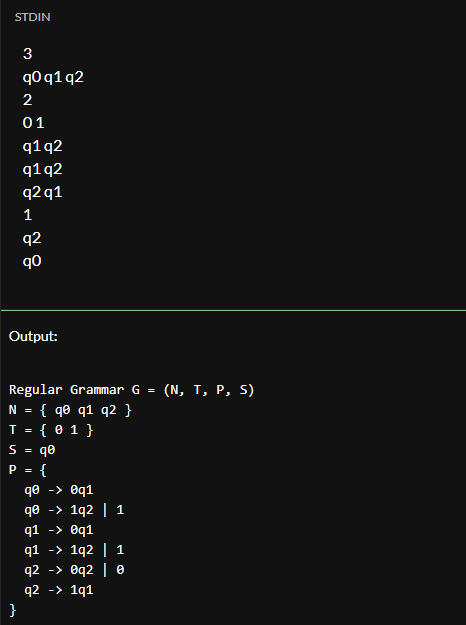
}

printf("}\n");

return **0;**

**}**

**Output:**



**Experiment-3**    Implementation of Deterministic Finite Automaton (DFA) from Non-deterministic Finite Automata (NFA) without ε-edges   using C language.

**Code:**

#include <iostream>

#include <map>

#include <set>

#include <vector>

#include <queue>

#include <sstream>

using namespace std;

map<string, map<char, set<string>>> nfa;

map<set<string>, map<char, set<string>>> dfa;

vector<char> input\_symbols;

set<string> final\_states;

string start\_state;

string setToString(set<string> states) {

stringstream ss;

ss << "{";

for (auto it = states.begin(); it != states.end(); ++it) {

if (it != states.begin()) ss << ",";

ss << \*it;

}

ss << "}";

return ss.str();

}

bool isFinalState(const set<string>& states) {

for (const string& s : states) {

if (final\_states.count(s)) return true;

}

return false;

}

void convertNFAtoDFA() {

queue<set<string>> q;

set<set<string>> visited;

set<string> start = {start\_state};

q.push(start);

visited.insert(start);

while (!q.empty()) {

set<string> current = q.front();

q.pop();

for (char symbol : input\_symbols) {

set<string> next;

for (const string& state : current) {

if (nfa[state].count(symbol)) {

next.insert(nfa[state][symbol].begin(), nfa[state][symbol].end());

}

}

dfa[current][symbol] = next;

if (!visited.count(next) && !next.empty()) {

visited.insert(next);

q.push(next);

}

}

}

}

bool testStringOnDFA(const string& input) {

set<string> current = {start\_state};

for (char symbol : input) {

current = dfa[current][symbol];

if (current.empty()) break;

}

return isFinalState(current);

}

int main() {

int num\_states;

vector<string> states;

cin >> num\_states;

for (int i = 0; i < num\_states; ++i) {

string s; cin >> s;

states.push\_back(s);

}

int num\_symbols;

cin >> num\_symbols;

for (int i = 0; i < num\_symbols; ++i) {

char c; cin >> c;

input\_symbols.push\_back(c);

}

cin >> start\_state;

int num\_finals;

cin >> num\_finals;

for (int i = 0; i < num\_finals; ++i) {

string f; cin >> f;

final\_states.insert(f);

}

int num\_transitions;

cin >> num\_transitions;

for (int i = 0; i < num\_transitions; ++i) {

string from, to;

char symbol;

cin >> from >> symbol >> to;

nfa[from][symbol].insert(to);

}

convertNFAtoDFA();

cout << "\nDFA Transition Table:\n";

for (const auto& [state\_set, transitions] : dfa) {

cout << setToString(state\_set);

for (char symbol : input\_symbols) {

cout << " --" << symbol << "--> " << setToString(transitions.at(symbol));

}

cout << '\n';

}

string test;

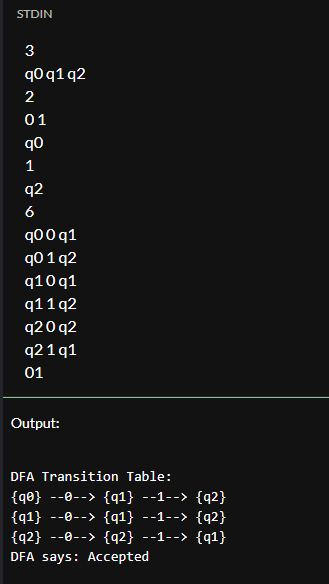
cin >> test;

cout << "DFA says: " << (testStringOnDFA(test) ? "Accepted" : "Rejected") << endl;

return 0;

}

**Output:**

****

**Experiment-4**

1. Implement a  DFA in LEX code which accepts Odd number of a’s and even number of b’s.

**Code:**

%{

#include <stdio.h>

int a\_count = 0, b\_count = 0;

%}

%%

a { a\_count++; }

b { b\_count++; }

[^ab\n] ; // Ignore other characters

\n {

if (a\_count % 2 == 1 && b\_count % 2 == 0)

printf("Accepted\n");

else

printf("Rejected\n");

// Reset counts for next input

a\_count = b\_count = 0;

}

%%

int main() {

yylex();

return 0;

}

b) Implement a DFA in LEX code which accepts strings over {a, b, c} having bca as substring.

**Code:**

%{

#include <stdio.h>

%}

%%

.\*bca.\* { printf("Accepted\n"); }

[a-c\n]+ { printf("Rejected\n"); }

. ; // Ignore any other character

%%

int main() {

yylex();

return 0;

}