class DFA\_inputs

public:

// variables

int no\_of\_states, no\_of\_inputs, no\_of\_final\_states;

char start\_state;

vector<char> final\_states;

vector<char> inputs;

vector<char> states;

map<char, map<char, char>> dfa; //to create table

vector<vector<char>> equilance\_sets; //set of non-final states and final states

map<char, int> state\_to\_index; // map of state to index

//constructor

DFA\_inputs(int n, int m); //constructor

//method

void NonFinal\_States(); //set of non-final states

void add\_transition(char s, char i, char d); //add transition to dfa

void print\_dfa(); //print initial dfa

Publically inherited

Class DFA\_minimisation : public DFA\_inputs

public:

//variables

bool matrix[N\_state][N\_state]; //matrix of equivalence

int no\_of\_false\_blocks; //number of false blocks

set<string> optimise\_states;

vector<string> s2;

set<char> s;

vector<string> union\_of\_dfa\_minimised;

//constructor

DFA\_minimisation(int n, int m) : DFA\_inputs(n, m) {};

//method

bool DFA\_Equivalent\_States(char s, char i); //check if two states are equivalent

void get\_equilance\_classes(vector<char> states); //get equivalence classes

void optimize\_states();

void union\_of\_optimise\_states(); //union to merge equivalent states

void final\_minimised\_dfa();

void final\_minimised\_dfa2(); //for printing the final dfa

void print\_union\_of\_dfa\_minimised(); //for printing the minimise final dfa

void final\_minimised\_transition\_table(); //for printing the final transition table

void print\_final\_states();

main()

int n, m;

Enter the number of states: = n

Enter the number of inputs: = m

cout << "\nEnter the states: ";

DFA\_minimisation dfa(n, m); //object of DFA\_minimisation

Use this object to call the variables and method’s of the class

for (i,0,n) //enter states

{

dfa.states.push\_back(c);

}

for (i,0,m) // enter inputs

{

dfa.inputs.push\_back(c);

}

Enter the start state: = dfa.start\_state

Enter the number of final states: = dfa.no\_of\_final\_states

for (i,0,dfa.no\_of\_final\_states) // enter final states

{

dfa.final\_states.push\_back(c);

}

for ( i, 0,n) // add transitions

{

for ( j, 0, m)

{

dfa.add\_transition(dfa.states[i], dfa.inputs[j], d);

}

}

dfa.NonFinal\_States();

dfa.print\_dfa();

dfa.get\_equilance\_classes(dfa.states);

dfa.optimize\_states();

dfa.union\_of\_optimise\_states();

dfa.final\_minimised\_dfa();

dfa.final\_minimised\_dfa2();

dfa.print\_union\_of\_dfa\_minimised();

dfa.final\_minimised\_transition\_table();

dfa.print\_final\_states();