Logo

Description automatically generated

JAMIA MILLIA ISLAMIA, NEW DELHI

COMPILER DESIGN LAB

NAME: FAIZAN CHOUDHARY

ROLL NO: 20BCS021

SUBJECT CODE: CEN 692

SEMESTER: 6th

COURSE: B.TECH.(COMPUTER ENGG.)

DEPT: DEPT OF COMPUTER ENGG.

SUBMITTED TO:

DR. SARFARAZ MASOOD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S. NO. | DATE | PROGRAM | PAGE | SIGN |
| 1 | 18/01/2023 | WAP to implement a program that takes an input string from the console and verifies it against a Deterministic Finite Automaton which is given through a separate file. | 3 |  |
| 2 | 01/02/2023 | WAP to implement a Mealy Machine, where the program generates an output corresponding to an input string given thru the console. | 6 |  |
| 3 | 08/02/2023 | WAP to implement a Moore Machine, where the program generates an output corresponding to an input string given thru the console. | 9 |  |
| 4 | 15/02/2023 | WAP to implement the conversion of a NFA to a corresponding DFA. The NFA must be given thru a separate file. | 12 |  |
| 5 | 01/03/2023 | WAP to Evaluate the FIRST & FOLLOW information of a CFG which is given through a file. | 17 |  |
| 6 | 22/03/2023 | WAP to Construct the LL(1) Parsing table for a CFG given through a file. This program should call the FIRST-FOLLOW program to generate the First & Follow information for the given CFG which will be used to generate the LL(1) Table. | 24 |  |
| 7 | 05/04/2023 | WAP to implement LL(1) string checking process where a string, given by the user thru the console, is checked against an LL1 table, given thru a file. | 32 |  |

FAIZAN CHOUDHARY

20BCS021

18th January, 2023

CODE:

#include <iostream>

#include <stdlib.h>

#include <vector>

#include <fstream>

#include <sstream>

#include <string>

using namespace std;

vector< vector<int> > dfa;

int initial\_state;

vector<int> final\_states;

int find (vector<int> &arr, int key) {

    for (int i=0; i<arr.size(); i++)

        if (arr[i] == key)

           return i;

    return -1;

}

string check(vector< vector<int> > &dfa, string input) {

    int curr\_state = initial\_state;

    int i=0;

    cout<<"\nTransitions: ";

    while (i < input.size() && curr\_state != -1) {

        curr\_state = dfa[curr\_state][input[i]-'0'];

        if (curr\_state != -1)

            cout<<"q"<<curr\_state<<" -> ";

        else

            cout<<"Dead state ";

        i++;

    }

    cout<<endl<<endl;

    if (curr\_state == -1)

       return "NOT ACCEPTED: DEAD STATE";

    else if (find(final\_states, curr\_state) == -1)

       return "NOT ACCEPTED: NON FINAL STATE";

    return "ACCEPTED";

}

int main() {

    ifstream fin;

    fin.open("dfa.txt");

    int curr\_line = 0;

    string line;

    cout<<"\n20BCS021\nFAIZAN CHOUDHARY\n\n";

    cout<<"Given DFA: "<<endl;

    // read until EOF

    while (getline(fin, line)) {

        cout<<line<<endl;

    }

    // clearing eof flags and seeking to start of file

    fin.clear();

    fin.seekg(0);

    while (fin) {

        getline(fin, line);

        vector<int> temp;

        if (curr\_line == 0)

           initial\_state = line[0] - '0';

        else if (curr\_line == 1) {

            for (int i=0; i<line.size(); i++) {

                if (line[i] != ',') {

                    final\_states.push\_back(line[i] - '0');

                }

            }

        }

        else {

            for (int i=0; i<line.size(); i++) {

                if (line[i] != ' ') {

                    if (line[i] == '-') {

                        i++;

                        temp.push\_back(-(line[i] - '0'));

                    }

                    else

                        temp.push\_back(line[i]-'0');

                }

            }

            dfa.push\_back(temp);

        }

        curr\_line++;

    }

    string input;

    cout<<"\nEnter input consisting of 0's and 1's: ";

    getline(cin, input);

    // if (input.size() == 0)

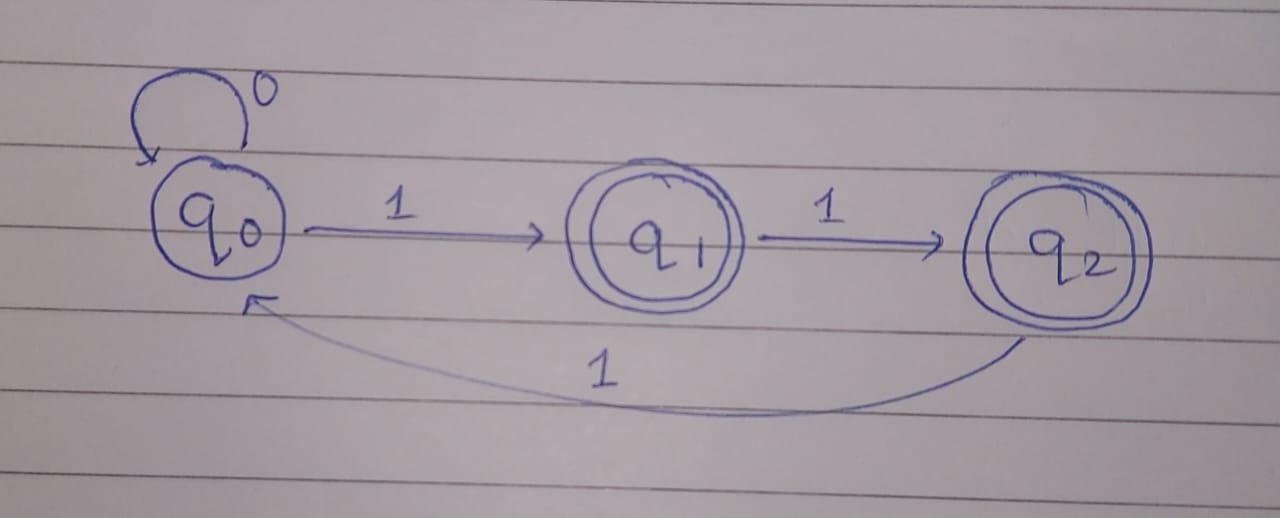
    string ans = check(dfa, input);

    cout<<ans<<endl;

    fin.close();

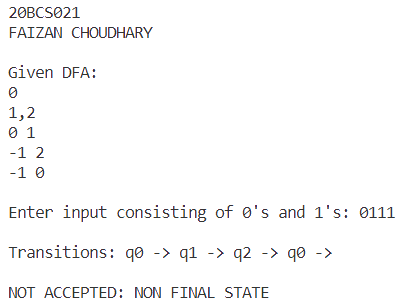
    return 0;

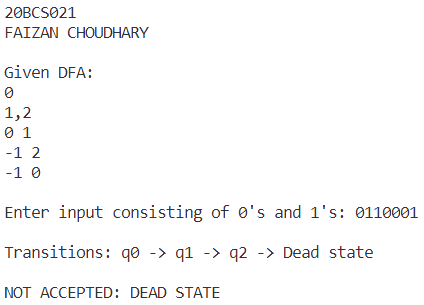
}



DFA used

OUTPUT:





Text

Description automatically generated

FAIZAN CHOUDHARY

20BCS021

1st February, 2023

CODE:

#include <iostream>

#include <stdlib.h>

#include <vector>

#include <fstream>

#include <sstream>

#include <string>

using namespace std;

vector< vector<pair<int, char> > > mealy;

int initial\_state;

string check(string input) {

    string out;

    pair<int, char> t = {initial\_state, ' '};

    int i=0;

    cout<<"\nTransitions: ";

    while (i < input.size() && t.first != -1) {

        t = mealy[t.first][input[i]-'0'];

        if (t.first != -1)

            cout<<"q"<<t.first<<" -> ";

        if (t.first == -1)

            break;

        else

            out += t.second;

        i++;

    }

    cout<<endl;

    return out;

}

int main() {

    ifstream fin;

    fin.open("mealy.txt");

    int curr\_line = 0;

    string line;

    cout<<"\n20BCS021\nFAIZAN CHOUDHARY\n\n";

    cout<<"Given Mealy: "<<endl;

    // read until EOF

    while (getline(fin, line)) {

        cout<<line<<endl;

    }

    // clearing eof flags and seeking to start of file

    fin.clear();

    fin.seekg(0);

    while (fin) {

        int state;

        char output;

        vector< pair<int, char> > temp;

        getline(fin, line);

        if (curr\_line == 0)

           initial\_state = line[0] - '0';

        // no final states in mealy

        else {

            for (int i=0; i<line.size(); ) {

                if (line[i] != ' ') {

                    if (line[i] == '-') {

                        i++;

                        state = (-(line[i] - '0'));

                        output = (' ');

                        temp.push\_back({state, output});

                        i+=5;

                    }

                    else {

                        state = (line[i]-'0');

                        output = (line[i+2]);

                        temp.push\_back({state, output});

                        i+=4;

                    }

                }

            }

            mealy.push\_back(temp);

        }

        curr\_line++;

    }

    string input;

    cout<<"\nEnter input consisting of 0's and 1's: ";

    getline(cin, input);

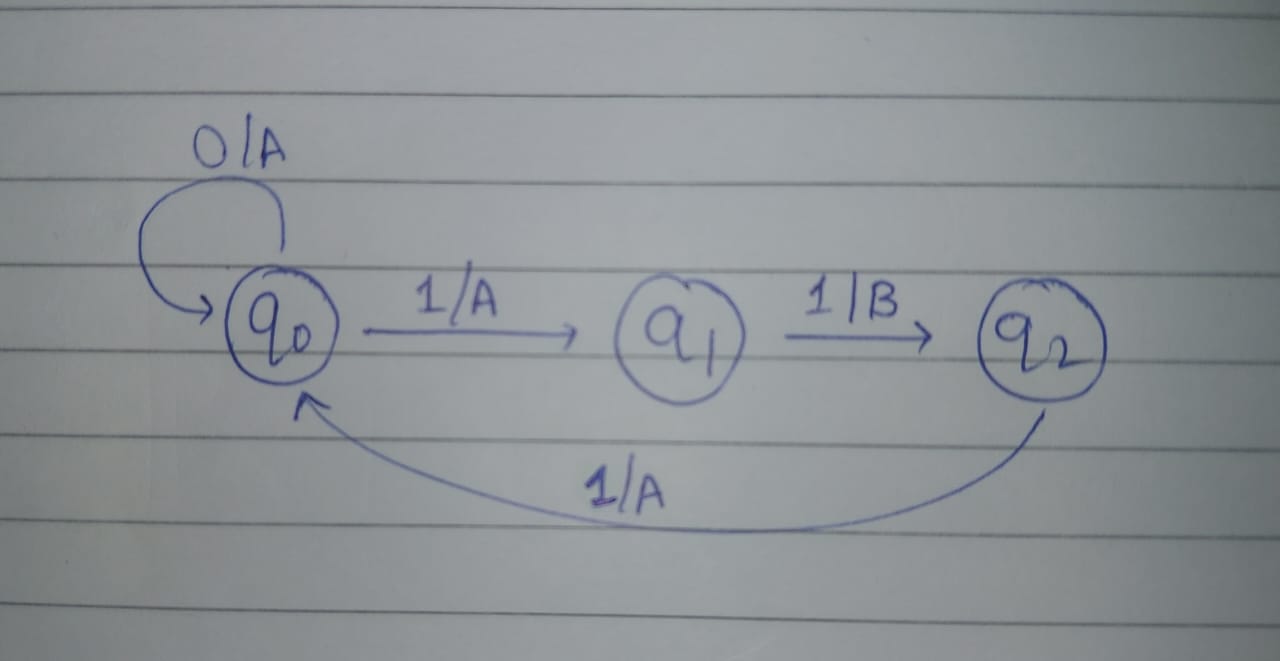
    string ans = check(input);

    cout<<"Output of Mealy machine: "<<ans<<endl;

    fin.close();

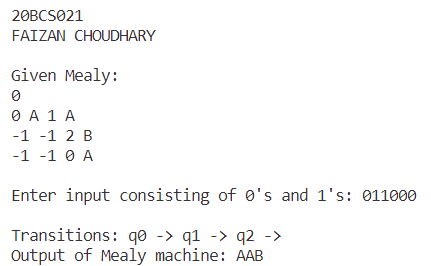
    return 0;

}



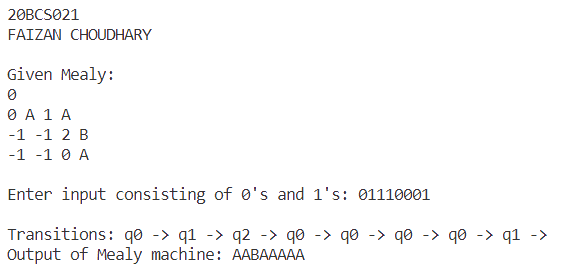
Mealy Machine used

OUTPUT:



Text, letter

Description automatically generated



FAIZAN CHOUDHARY

20BCS021

8th February, 2023

CODE:

#include <iostream>

#include <stdlib.h>

#include <vector>

#include <fstream>

#include <sstream>

#include <string>

#include <algorithm>

using namespace std;

vector< vector<int> > moore;

int initial\_state;

vector<string> stateOutputs;

int convertToInt(string s) {

    int num = 0;

    for (int i=0; i<s.size(); i++) {

        num = num\*10 + (s[i]-'0');

    }

    return num;

}

string check(string input) {

    string out;

    int t = initial\_state;

    int i=0;

    cout<<"\nTransitions: ";

    while (i <= input.size() && t != -1) {

        if (t != -1)

            cout<<"q"<<t<<" -> ";

        if (t == -1)

            break;

        // else

            out += stateOutputs[t];

        t = moore[t][input[i]-'0'];

        i++;

    }

    cout<<endl;

    return out;

}

int main() {

    ifstream fin;

    fin.open("moore.txt");

    int curr\_line = 0;

    string line;

    cout<<"\n20BCS021\nFAIZAN CHOUDHARY\n\n";

    cout<<"Given Moore: "<<endl;

    // read until EOF

    while (getline(fin, line)) {

        cout<<line<<endl;

    }

    // clearing eof flags and seeking to start of file

    fin.clear();

    fin.seekg(0);

    while (getline(fin, line)) {

        int state;

        string output;

        vector<int> temp;

        if (curr\_line == 0)

           initial\_state = convertToInt(line);

        // no final states in moore

        else {

            int i = 0;

            int j = line.size()-1;

            while (line[j] != ' ')

                output += line[j--];

            reverse(output.begin(), output.end());

            // cout<<"OUTPUT: "<<output<<endl;

            while (i < line.size()-output.size()) {

                string s;

                while (line[i] != ' ' && i < line.size())

                    s += line[i++];

                if (s == "-1") {

                    state = -1;

                    temp.push\_back(state);

                    i++;

                    continue;

                }

                else if (isdigit(s[0]))

                    state = convertToInt(s);

                // cout<<"State: "<<state<<" "<<output<<endl;

                if (output != "")

                    temp.push\_back(state);

                i++;

            }

            stateOutputs.push\_back(output);

            moore.push\_back(temp);

        }

        curr\_line++;

    }

    string input;

    cout<<"\nEnter input consisting of 0's and 1's: ";

    getline(cin, input);

    string ans = check(input);

    cout<<"Output of Moore machine: "<<ans<<endl;

    // for (int i=0; i<moore.size(); i++) {

    //  for (int j=0; j<moore[0].size(); j++)

    //      cout<<moore[i][j]<<" "<<stateOutputs[i]<<"  ";

    //  cout<<endl;

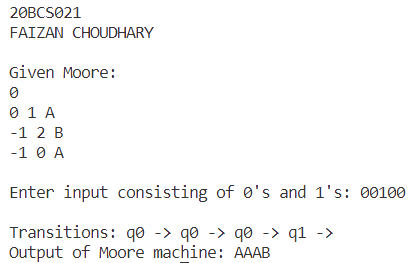
    // }

    fin.close();

    return 0;

}

OUTPUT:



FAIZAN CHOUDHARY

20BCS021

15th February, 2023

CODE:

#include <iostream>

#include <stdlib.h>

#include <vector>

#include <fstream>

#include <sstream>

#include <string>

#include <set>

#include <map>

#include <queue>

#include <algorithm>

using namespace std;

// set to have non-duplicate elements

vector< vector< set<int> > > nfa;

vector< vector<int> > dfa;

int initial\_state;

vector<int> final\_states;

// states = no of rows, inputCount = no of columns

int states = 0, inputCount = 0;

// map and reverse map to replace combined states like q0q1 to another named state like q4

map<set<int>, int> stateMap;

map<int, set<int>> stateMap2;

// queue to keep track of the states which are not yet processed but can be reached by the current state by transitions

queue<int> q;

int convertToInt(string s) {

    int num = 0;

    for (int i=0; i<s.size(); i++) {

        num = num\*10 + (s[i]-'0');

    }

    return num;

}

void helper() {

    // initially no of states = no of rows in the nfa

    int n = states;

    // iterating through the nfa

    for (int i=0; i<states; i++) {

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

            // checking for absence of -1's in the set and checking for the absence of the set in the map

            if (!nfa[i][j].count(-1) && stateMap.find(nfa[i][j]) == stateMap.end()) {

                // making a new state(set) by merging the old states, but before doing this renaming the state (like q0q1 -> q4)

                stateMap[nfa[i][j]] = n;

                stateMap2[n] = nfa[i][j];

                // pushing the set in the queue for further processing

                q.push(n);

                // incrementing no of states

                n++;

            }

        }

    }

    // processing while the queue is not empty

    while (!q.empty()) {

        // no of elements in the queue at the current time to be processed

        int k = q.size();

        for (int i=0; i<k; i++) {

            // popping

            int curr = q.front();

            q.pop();

            vector<set<int>> temp;

            // running a loop through the no of cols for each element in the queue

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

                set<int> t;

                // finding out the states for which the current state is renamed to

                for (auto state : stateMap2[curr]) {

                    // if the state doesnt contain any -1's then insert the elements of the set in the new set

                    if (!nfa[state][j].count(-1)) {

                        t.insert(nfa[state][j].begin(), nfa[state][j].end());

                    }

                }

                // if there are elements in the new set and if the map doesnt contain the set then make a new state

                if (!t.empty() && stateMap.find(t) == stateMap.end()) {

                    stateMap[t] = n;

                    q.push(n);

                    stateMap2[n] = t;

                    n++;

                }

                // otherwise insert a -1

                if (t.empty()) {

                    t.insert(-1);

                }

                temp.push\_back(t);

            }

            nfa.push\_back(temp);

        }

    }

    // updating final states due to merging of states

    set<int> final(final\_states.begin(), final\_states.end());

    // for each state in the map

    for (auto it : stateMap2) {

        // check if it is a final state

        if (!final.count(it.first)) {

            // if it is not a final state then check if any of the states in the set is a final state

            for (auto i : it.second) {

                if (final.count(i)) {

                    // if any of the states in the set is a final state then insert the set in the final states

                    final.insert(it.first);

                    break;

                }

            }

        }

    }

    ofstream fout;

    fout.open("convertedDFA.txt");

    fout<<initial\_state<<endl;

    for (auto itr = final.begin(); itr != final.end(); itr++) {

        if (next(itr) == final.end())

            fout<<\*itr;

        else

            fout<<\*itr<<",";

    }

    fout<<endl;

    for (int i=0; i<n; i++) {

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

            if (stateMap.find(nfa[i][j]) != stateMap.end()) {

                fout<<stateMap[nfa[i][j]]<<" ";

            }

            else {

                fout<<"-1 ";

            }

        }

        fout<<endl;

    }

    fout.close();

}

int main() {

    ifstream fin;

    fin.open("nfa.txt");

    int curr\_line = 0;

    string line;

    cout<<"\n20BCS021\nFAIZAN CHOUDHARY\n\n";

    cout<<"Given NFA: "<<endl;

    // read until EOF

    while (getline(fin, line)) {

        cout<<line<<endl;

    }

    // clearing eof flags and seeking to start of file

    fin.clear();

    fin.seekg(0);

    while (getline(fin, line)) {

        int state;

        vector<set<int>> temp;

        if (curr\_line == 0)

           initial\_state = convertToInt(line);

        else if (curr\_line == 1) {

            for (int i=0; i<line.size(); i++) {

                if (line[i] != ',') {

                    final\_states.push\_back(convertToInt(line.substr(i, 1)));

                }

            }

        }

        else {

            string s;

            stringstream ss(line);

            while (getline(ss, s, ' ')) {

                string k;

                set<int> t;

                // cout<<s<<endl;

                for (int i=0; i<s.size(); i++) {

                    if (s[i] != ',')

                        k += s[i];

                    else {

                        t.insert(convertToInt(k));

                        k = "";

                    }

                }

                if (k != "-1")

                    t.insert(convertToInt(k));

                else

                    t.insert(-1);

                temp.push\_back(t);

            }

            nfa.push\_back(temp);

        }

        curr\_line++;

    }

    // updating no of rows and cols

    states = nfa.size();

    inputCount = nfa[0].size();

    // inserting the state values in the two maps

    for (int i=0; i<states; i++) {

        stateMap.insert({{i}, i});

        stateMap2.insert({i, {i}});

    }

    helper();

    fin.close();

    fin.open("convertedDFA.txt");

    cout<<"\nConverted DFA: "<<endl;

    while (getline(fin, line)) {

        cout<<line<<endl;

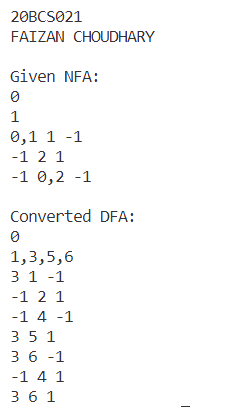
    }

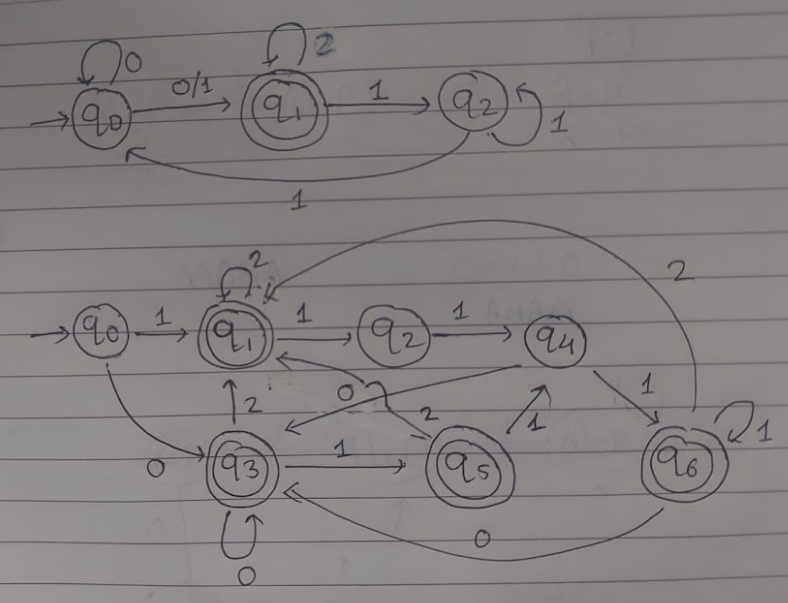
    fin.close();

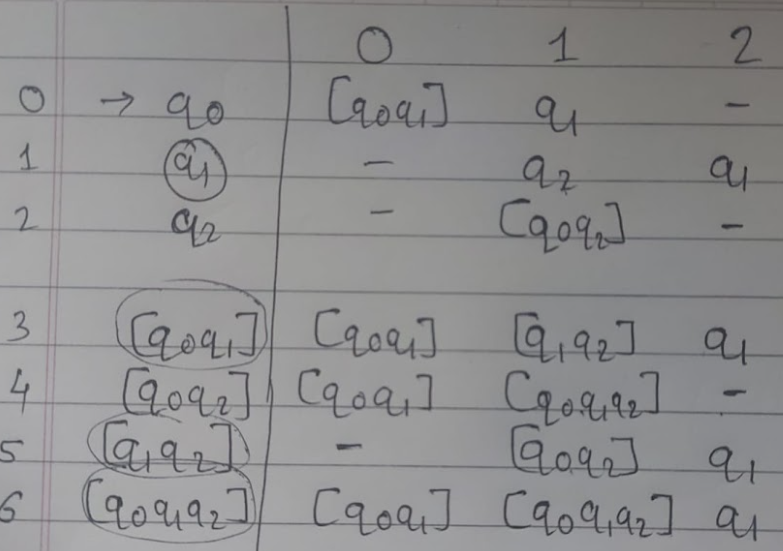
    return 0;

}

OUTPUT:







FAIZAN CHOUDHARY

20BCS021

1st March, 2023

CODE:

#include <iostream>

#include <stdlib.h>

#include <vector>

#include <fstream>

#include <sstream>

#include <string>

#include <set>

#include <map>

#include <queue>

#include <algorithm>

using namespace std;

vector< vector<char> > CFG;

int noOfNTs = 0;

map <string, bool> isVisited;

set <char> NonTerminals;

set <char> Terminals;

map <char, set<char>> first;

map <char, set<char>> follow;

map <char, vector<string>> productions;

bool isUpper(string s) {

    for (int i=0; i<s.size(); i++)

        if (islower(s[i]))

            return false;

    return true;

}

bool checkEpsilon(string p, char n) {

    bool allEpsilon = true;

    if (isUpper(p)) {

        for (char c: p) {

            if (find(productions[c].begin(), productions[c].end(), "#") == productions[c].end()) {

                allEpsilon = false;

                break;

            }

        }

        if (allEpsilon)

            first[n].insert('#');

    }

    else

        return allEpsilon;

}

// to find first of a non-terminal recursively

void findFirst(string p, char n) {

    // cout<<n<<": "<<p<<endl;

    isVisited[p] = true;

    if (islower(p[0])) {

        // if first of p is terminal, then add it to first of n

        first[n].insert(p[0]);

        return;

    }

    else {

        for (int i=0; i<p.size(); i++) {

            if (isupper(p[i]) || !isalpha(p[i]) && p[i] != '#') {

                // if first of p is epsilon, then add first of next symbols in p until first of a non-epsilon symbol is found, else add epsilon to first of n

                for (string x: productions[p[i]]) {

                    if (!isVisited[x])

                        findFirst(x, n);

                }

                first[n].insert(first[p[i]].begin(), first[p[i]].end());

                if (find(productions[p[i]].begin(), productions[p[i]].end(), "#") == productions[p[i]].end())

                    break;

                else {

                    // if all productions of n are epsilon, then add epsilon to first of n

                    if (i != p.size()-1 && checkEpsilon(p, n))

                        first[n].erase('#');

                    // continue;

                }

            }

            else if (i > 0 && islower(p[i])) {

                if (find(productions[p[i-1]].begin(), productions[p[i-1]].end(), "#") != productions[p[i-1]].end()) {

                    // if first of previous symbol in p is epsilon, then add first of p to first of n

                    first[n].insert(p[i]);

                    break;

                }

            }

        }

        if (p == "#")

            first[n].insert('#');

    }

}

void findFollow(char c) {

    // cout<<c<<endl;

    isVisited[to\_string(c)] = true;

    for (auto x: productions) {

        for (string p: x.second) {

            for (int i=0; i<p.size(); i++) {

                if (p[i] == c) {

                    if (i == p.size()-1) {

                        // if c is last symbol in p, then add follow of x.first to follow of c

                        if (isupper(x.first) && x.first != c)

                            if (!isVisited[to\_string(x.first)])

                                findFollow(x.first);

                        follow[c].insert(follow[x.first].begin(), follow[x.first].end());

                    }

                    else {

                        // if c is not last symbol in p, then add first of next symbol in p to follow of c

                        for (int j=i+1; j<p.size(); j++) {

                            if (isupper(p[j])) {

                                if (!isVisited[to\_string(p[j])])

                                    findFollow(p[j]);

                                // if first of p[j] is epsilon, then add first of next symbols in p until first of a non-epsilon symbol is found, else add epsilon to follow of c

                                follow[c].insert(first[p[j]].begin(), first[p[j]].end());

                                if (find(productions[p[j]].begin(), productions[p[j]].end(), "#") != productions[p[j]].end()) {

                                    follow[c].erase('#');

                                    if (j == p.size()-1)

                                        follow[c].insert(follow[x.first].begin(), follow[x.first].end());

                                    // else

                                    //     continue;

                                    // break;

                                }

                                else

                                    break;

                            }

                            else {

                                follow[c].insert(p[j]);

                                break;

                            }

                        }

                    }

                }

            }

        }

    }

}

int main() {

    ifstream fin;

    fin.open("CFG.txt");

    int curr\_line = 0;

    string line;

    cout<<"\n20BCS021\nFAIZAN CHOUDHARY\n\n";

    cout<<"Given CFG: "<<endl;

    // read until EOF

    while (getline(fin, line)) {

        cout<<line<<endl;

    }

    // clearing eof flags and seeking to start of file

    fin.clear();

    fin.seekg(0);

    while (getline(fin, line)) {

        char s;

        stringstream ss(line);

        vector<char> temp;

        int col = 0;

        while (ss >> s) {

            if (isupper(s)) {

                noOfNTs++;

                NonTerminals.insert(s);

                // first NT of first production is start symbol and its follow is $

                if (curr\_line == 0 && col == 0)

                    follow[s].insert('$');

            }

            else if (s == '#') {

                // epsilon

                first[temp.back()].insert('#');

            }

            else {

                Terminals.insert(s);

                // first of the terminal is the terminal itself

                first[s].insert(s);

            }

            col++;

            temp.push\_back(s);

        }

        // adding production to map

        string k;

        for (int i=1; i<temp.size(); i++) {

            k += temp[i];

        }

        productions[temp[0]].push\_back(k);

        CFG.push\_back(temp);

        curr\_line++;

    }

    for (auto x: NonTerminals)

        isVisited[to\_string(x)] = false;

    // FIRST

    for (char n: NonTerminals) {

        // p is a string of production

        for (string p: productions[n]) {

            findFirst(p, n);

        }

    }

    for (auto x: NonTerminals)

        isVisited[to\_string(x)] = false;

    // FOLLOW

    for (char n: NonTerminals) {

        findFollow(n);

    }

    // for (auto x: productions) {

    //     cout<<x.first<<" : ";

    //     for (auto s: x.second)

    //         cout<<s<<" ";

    //     cout<<endl;

    // }

    cout<<endl<<"Firsts:\n";

    for (auto x: first) {

        cout<<"First("<<x.first<<") : ";

        for (auto s: x.second)

            cout<<s<<" ";

        cout<<endl;

    }

    cout<<endl<<"Follows:\n";

    for (auto x: follow) {

        cout<<"Follow("<<x.first<<") : ";

        for (auto s: x.second)

            cout<<s<<" ";

        cout<<endl;

    }

    // PARSED CFG:

    // cout<<endl;

    // for (auto c: CFG) {

    //     for (auto x: c)

    //         cout<<x<<" ";

    //     cout<<endl;

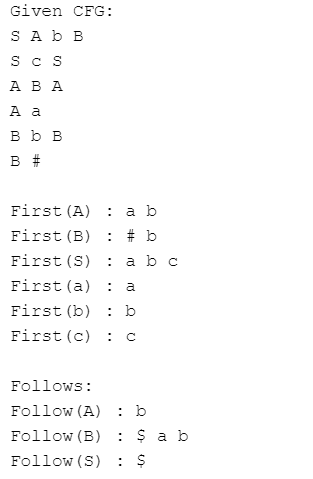
    // }

    fin.close();

    return 0;

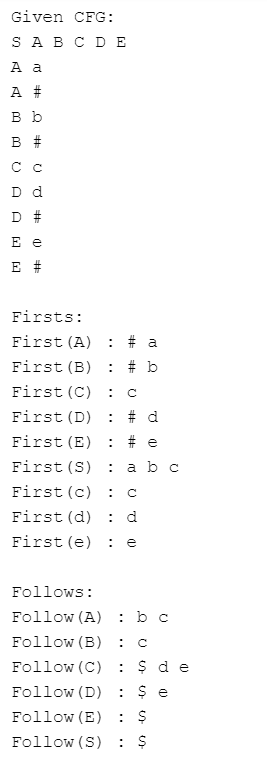
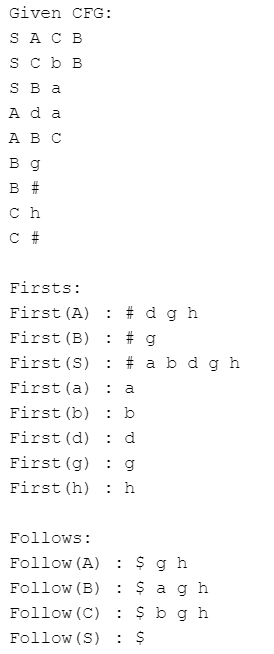
}

OUTPUT:



A screenshot of a computer program

Description automatically generated with low confidence

FAIZAN CHOUDHARY

20BCS021

22nd March, 2023

CODE:

#include "firstAndFollow.h"

using namespace std;

int main() {

    ifstream fin;

    fin.open("CFG1.txt");

    int curr\_line = 0;

    string line;

    cout<<"\n20BCS021\nFAIZAN CHOUDHARY\n\n";

    cout<<"Given CFG: "<<endl;

    // read until EOF

    while (getline(fin, line)) {

        cout<<curr\_line<<" "<<line<<endl;

        char s;

        stringstream ss(line);

        vector<char> temp;

        int col = 0;

        while (ss >> s) {

            if (isupper(s)) {

                noOfNTs++;

                NonTerminals.insert(s);

                // first NT of first production is start symbol and its follow is $

                if (curr\_line == 0 && col == 0)

                    follow[s].insert('$');

            }

            else if (s == '#') {

                // epsilon

                first[temp.back()].insert('#');

            }

            else {

                Terminals.insert(s);

                // first of the terminal is the terminal itself

                first[s].insert(s);

            }

            col++;

            temp.push\_back(s);

        }

        // adding production to map

        string k;

        for (int i=1; i<temp.size(); i++) {

            k += temp[i];

        }

        // productions[temp[0]].first = curr\_line;

        productions[temp[0]].push\_back(k);

        prodNo[temp[0] + k] = curr\_line;

        CFG.push\_back(temp);

        curr\_line++;

    }

    for (auto x: NonTerminals)

        isVisited[to\_string(x)] = false;

    // FIRST

    for (char n: NonTerminals) {

        // p is a string of production

        for (string p: productions[n]) {

            findFirst(p, n);

        }

    }

    for (auto x: NonTerminals)

        isVisited[to\_string(x)] = false;

    // FOLLOW

    for (char n: NonTerminals) {

        findFollow(n);

    }

    cout<<endl<<"Firsts:\n";

    for (auto x: first) {

        cout<<"First("<<x.first<<") : ";

        for (auto s: x.second)

            cout<<s<<" ";

        cout<<endl;

    }

    cout<<endl<<"Follows:\n";

    for (auto x: follow) {

        cout<<"Follow("<<x.first<<") : ";

        for (auto s: x.second)

            cout<<s<<" ";

        cout<<endl;

    }

    fin.close();

    // inserting $ into terminals

    vector<char> NTs(NonTerminals.begin(), NonTerminals.end());

    vector<char> Ts(Terminals.begin(), Terminals.end());

    Ts.push\_back('$');

    for (int i=0; i<NTs.size(); i++)

        mp[NTs[i]] = i;

    for (int i=0; i<Ts.size(); i++)

        mp[Ts[i]] = i;

    vector <vector<set<string>>> LL (NTs.size(), vector<set<string>> (Ts.size()));

    for (auto p: productions) {

        char NT = p.first;

        // int prodNo = p.second.first;

        for (string prod: p.second) {

            // string s(1, NT);

            // string k = s + "->" + prod;

            if (prod == "#" || !checkEpsilon(prod, NT)) {

                // if the whole string derives epsilon

                for (char t: follow[NT])

                    LL[mp[NT]][mp[t]].insert(prod);

            }

            // else {

                for (int i=0; i<prod.size(); i++) {

                    if (islower(prod[i])) {

                        LL[mp[NT]][mp[prod[i]]].insert(prod);

                        break;

                    }

                    // if first of prod[i] contains epsilon

                    if (find(first[prod[i]].begin(), first[prod[i]].end(), '#') != first[prod[i]].end()) {

                        for (char t: first[prod[i]]) {

                            if (t != '#')

                                LL[mp[NT]][mp[t]].insert(prod);

                        }

                    }

                    // no epsilon in first of prod[i]

                    else {

                        for (char t: first[prod[i]])

                            LL[mp[NT]][mp[t]].insert(prod);

                        break;

                    }

                }

            // }

        }

    }

    // LL table

    ofstream fout;

    fout.open("LLtable.txt");

    cout<<endl<<"LL Table:\n\n";

    fout<<"\t\t";

    cout<<"\t\t";

    for (char t: Ts) {

        cout<<t<<"\t\t";

        fout<<t<<"\t\t";

    }

    fout<<endl;

    cout<<endl;

    for (int i=0; i<NTs.size(); i++) {

        cout<<NTs[i]<<"\t\t";

        fout<<NTs[i]<<"\t\t";

        for (int j=0; j<Ts.size(); j++) {

            for (int k=0; k<LL[i][j].size(); k++) {

                cout<<prodNo[NTs[i] + \*next(LL[i][j].begin(), k)];

                fout<<prodNo[NTs[i] + \*next(LL[i][j].begin(), k)];

                if (k != LL[i][j].size()-1) {

                    cout<<",";

                    fout<<",";

                }

            }

            cout<<"\t\t";

            fout<<"\t\t";

        }

        cout<<endl<<endl;

        fout<<endl;

    }

    fout.close();

    return 0;

}

firstAndFollow.h

#include <iostream>

#include <stdlib.h>

#include <vector>

#include <fstream>

#include <sstream>

#include <string>

#include <set>

#include <map>

#include <queue>

#include <algorithm>

using namespace std;

vector< vector<char> > CFG;

int noOfNTs = 0;

map <string, bool> isVisited;

set <char> NonTerminals;

set <char> Terminals;

map <char, set<char>> first;

map <char, set<char>> follow;

map <char, vector<string>> productions;

map <char, int> mp;

map <string, int> prodNo;

bool isUpper(string s) {

    for (int i=0; i<s.size(); i++)

        if (islower(s[i]))

            return false;

    return true;

}

bool checkEpsilon(string p, char n) {

    bool allEpsilon = true;

    if (isUpper(p)) {

        for (char c: p) {

            if (find(productions[c].begin(), productions[c].end(), "#") == productions[c].end()) {

                allEpsilon = false;

                break;

            }

            // else

            //     allEpsilon = true;

        }

        if (allEpsilon)

            first[n].insert('#');

    }

    else

        return allEpsilon;

}

// to find first of a non-terminal recursively

void findFirst(string p, char n) {

    // cout<<n<<": "<<p<<endl;

    isVisited[p] = true;

    if (islower(p[0])) {

        // if first of p is terminal, then add it to first of n

        first[n].insert(p[0]);

        return;

    }

    else {

        for (int i=0; i<p.size(); i++) {

            if (isupper(p[i]) || !isalpha(p[i]) && p[i] != '#') {

                // if first of p is epsilon, then add first of next symbols in p until first of a non-epsilon symbol is found, else add epsilon to first of n

                for (string x: productions[p[i]]) {

                    if (!isVisited[x])

                        findFirst(x, n);

                }

                first[n].insert(first[p[i]].begin(), first[p[i]].end());

                if (find(productions[p[i]].begin(), productions[p[i]].end(), "#") == productions[p[i]].end())

                    break;

                else {

                    // if all productions of n are epsilon, then add epsilon to first of n

                    if (i != p.size()-1 && checkEpsilon(p, n))

                        first[n].erase('#');

                    // continue;

                }

            }

            else if (i > 0 && islower(p[i])) {

                if (find(productions[p[i-1]].begin(), productions[p[i-1]].end(), "#") != productions[p[i-1]].end()) {

                    // if first of previous symbol in p is epsilon, then add first of p to first of n

                    first[n].insert(p[i]);

                    break;

                }

            }

        }

        if (p == "#")

            first[n].insert('#');

    }

}

void findFollow(char c) {

    // cout<<c<<endl;

    isVisited[to\_string(c)] = true;

    for (auto x: productions) {

        for (string p: x.second) {

            for (int i=0; i<p.size(); i++) {

                if (p[i] == c) {

                    if (i == p.size()-1) {

                        // if c is last symbol in p, then add follow of x.first to follow of c

                        if (isupper(x.first) && x.first != c)

                            if (!isVisited[to\_string(x.first)])

                                findFollow(x.first);

                        follow[c].insert(follow[x.first].begin(), follow[x.first].end());

                    }

                    else {

                        // if c is not last symbol in p, then add first of next symbol in p to follow of c

                        for (int j=i+1; j<p.size(); j++) {

                            if (isupper(p[j])) {

                                if (!isVisited[to\_string(p[j])])

                                    findFollow(p[j]);

                                // if first of p[j] is epsilon, then add first of next symbols in p until first of a non-epsilon symbol is found, else add epsilon to follow of c

                                follow[c].insert(first[p[j]].begin(), first[p[j]].end());

                                if (find(productions[p[j]].begin(), productions[p[j]].end(), "#") != productions[p[j]].end()) {

                                    follow[c].erase('#');

                                    if (j == p.size()-1)

                                        follow[c].insert(follow[x.first].begin(), follow[x.first].end());

                                }

                                else

                                    break;

                            }

                            else {

                                follow[c].insert(p[j]);

                                break;

                            }

                        }

                    }

                }

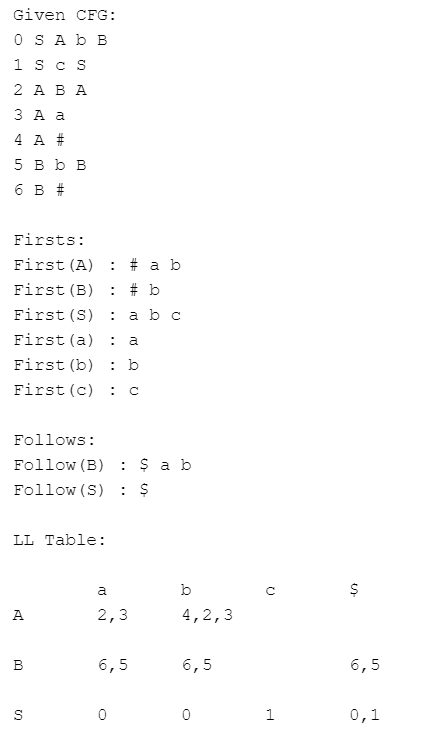
            }

        }

    }

}

OUTPUT:



A picture containing text, screenshot, font

Description automatically generated

A picture containing sketch, pattern

Description automatically generated

FAIZAN CHOUDHARY

20BCS021

5th April, 2023

CODE:

#include <iostream>

#include <stdlib.h>

#include <vector>

#include <fstream>

#include <sstream>

#include <string>

#include <set>

#include <map>

#include <stack>

#include <algorithm>

using namespace std;

set <char> NonTerminals;

set <char> Terminals;

map <char, vector<string>> productions;

map <int, string> prodNo;

map <char, int> mp;

char start;

int main() {

    ifstream fin;

    fin.open("CFG1.txt");

    int curr\_line = 0;

    string line;

    cout<<"\n20BCS021\nFAIZAN CHOUDHARY\n\n";

    cout<<"Given CFG: "<<endl;

    // read until EOF

    while (getline(fin, line)) {

        cout<<curr\_line<<" "<<line<<endl;

        char s;

        stringstream ss(line);

        vector<char> temp;

        int col = 0;

        while (ss >> s) {

            if (isupper(s)) {

                if (curr\_line == 0 && col == 0)

                    start = s;

                NonTerminals.insert(s);

                // first NT of first production is start symbol and its follow is $

            }

            else {

                Terminals.insert(s);

                // first of the terminal is the terminal itself

            }

            col++;

            temp.push\_back(s);

        }

        // adding production to map

        string k;

        for (int i=1; i<temp.size(); i++) {

            k += temp[i];

        }

        // productions[temp[0]].first = curr\_line;

        productions[temp[0]].push\_back(k);

        prodNo[curr\_line] = k;

        curr\_line++;

    }

    fin.close();

    vector<char> NTs(NonTerminals.begin(), NonTerminals.end());

    vector<char> Ts(Terminals.begin(), Terminals.end());

    Ts.push\_back('$');

    fin.open("LLtable.txt");

    vector<vector<string>> table (NTs.size(), vector<string> (Ts.size(), " "));

    curr\_line = 0;

    int c = 0;

    cout<<"\n\nGiven LL(1) parsing table: "<<endl<<endl;

    // read until EOF

    while (getline(fin, line)) {

        cout<<line<<endl;

        if (curr\_line == 0) {

            string s;

            stringstream ss(line);

            while (ss >> s) {

                mp[s[0]] = c++;

            }

            c = 0;

        }

        else {

            int counter = 0, prevC = 0;

            mp[line[0]] = c++;

            char NT = line[0];

            for (int i=1; i<line.size(); i++) {

                if (line[i] != '\t') {

                    // cout<<line[i]<<endl;

                    string k;

                    while (line[i] != '\t')

                        k += line[i++];

                    if (counter % 2 == 0) {

                        if (prevC != counter)

                            // there exists an entry for the cell given by line[i] and col (counter/2)-1 row is mp[NT]

                            table[mp[NT]][(counter/2)-1] = k;

                    }

                }

                else {

                    prevC = counter;

                    counter++;

                }

                // cout<<counter<<endl;

            }

        }

        curr\_line++;

    }

    // for (auto c: mp)

    //     cout<<c.first<<" "<<c.second<<endl;

    // for (int i=0; i<NTs.size(); i++) {

    //     for (int j=0; j<Ts.size(); j++)

    //         cout<<table[i][j]<<" ";

    //     cout<<endl;

    // }

    string check;

    cout<<"\nEnter string to check: ";

    cin>>check;

    check += '$';

    stack <char> st;

    st.push('$');

    st.push(start);

    int i = 0;

    cout<<"\nTop of Stack\tInput String\tProduction applied\n";

    while (st.top() != '$') {

        char top = st.top();

        st.pop();

        // cout<<top<<endl;

        if (top == check[i]) {

            i++;

        }

        else if (isupper(top)) {

            int r = mp[top];

            int c = mp[check[i]];

            // cout<<r<<" "<<c<<endl;

            // cout<<table[r][c]<<endl;

            if (table[r][c] == " ") {

                cout<<top<<"\t\t"<<check.substr(i)<<endl;

                cout<<"\nString is not accepted!"<<endl;

                return 0;

            }

            string prod = prodNo[stoi(table[r][c])];

            if (prod[0] == check[i] || prod == "#")

                cout<<top<<"\t\t"<<check.substr(i)<<"\t\t"<<top<<"->"<<prod<<endl;

            else

                cout<<top<<"\t\t"<<check.substr(i)<<endl;

            if (prod != "#") {

                for (int i=prod.size()-1; i>=0; i--)

                    st.push(prod[i]);

            }

        }

        else {

            cout<<top<<"\t\t"<<check.substr(i)<<endl;

            cout<<"\nString not accepted!"<<endl;

            return 0;

        }

    }

    cout<<st.top()<<"\t\t"<<check.substr(i)<<endl;

    if (st.top() == '$' && check[i] == '$')

        cout<<"\nString is accepted"<<endl;

    else

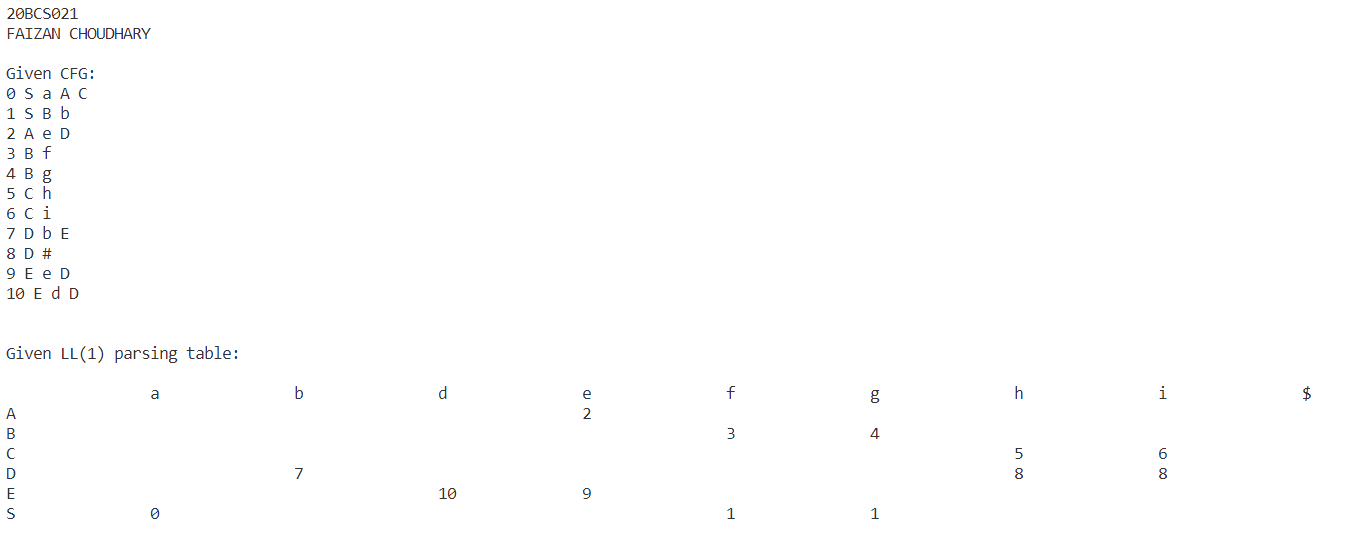
        cout<<"\nString is not accepted!"<<endl;

    fin.close();

    return 0;

}

OUTPUT:



A picture containing text, screenshot, font, number

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence