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

FAIZAN CHOUDHARY

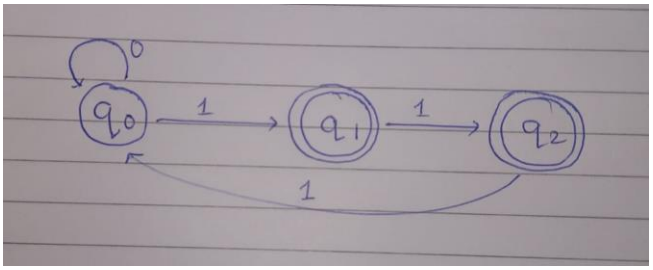
20BCS021

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:

```
20BCS021
FAIZAN CHOUDHARY

Given DFA:
0
1,2
0 1
-1 2
-1 0

Enter input consisting of 0's and 1's: 0111

Transitions: q0 -> q1 -> q2 -> q0 ->

NOT ACCEPTED: NON FINAL STATE
```

```
20BCS021
FAIZAN CHOUDHARY

Given DFA:
0
1,2
0 1
-1 2
-1 0

Enter input consisting of 0's and 1's: 0110001

Transitions: q0 -> q1 -> q2 -> Dead state

NOT ACCEPTED: DEAD STATE
```

```
20BCS021
FAIZAN CHOUDHARY

Given DFA:
0
1,2
0 1
-1 2
-1 0

Enter input consisting of 0's and 1's: 01111

Transitions: q0 -> q1 -> q2 -> q0 -> q1 ->

ACCEPTED
```

FAIZAN CHOUDHARY

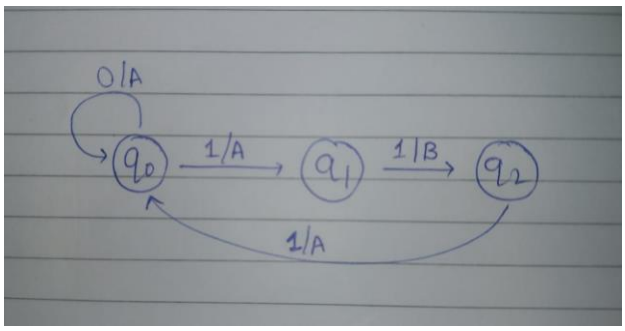
20BCS021

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:

```
20BCS021
FAIZAN CHOUDHARY

Given Mealy:
0
0 A 1 A
-1 -1 2 B
-1 -1 0 A

Enter input consisting of 0's and 1's: 011000

Transitions: q0 -> q1 -> q2 ->
Output of Mealy machine: AAB
```

```
20BCS021
FAIZAN CHOUDHARY

Given Mealy:
0
0 A 1 A
-1 -1 2 B
-1 -1 0 A

Enter input consisting of 0's and 1's: 10010100

Transitions: q1 ->
Output of Mealy machine: A
```

```
20BCS021
FAIZAN CHOUDHARY

Given Mealy:
0
0 A 1 A
-1 -1 2 B
-1 -1 0 A

Enter input consisting of 0's and 1's: 01110001

Transitions: q0 -> q1 -> q2 -> q0 -> q0 -> q0 -> q0 -> q1 ->
Output of Mealy machine: AABAAAAA
```

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:

```
20BCS021
FAIZAN CHOUDHARY

Given Moore:
0
0 1 A
-1 2 B
-1 0 A

Enter input consisting of 0's and 1's: 00100

Transitions: q0 -> q0 -> q0 -> q1 ->
Output of Moore machine: AAAB
```

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
```

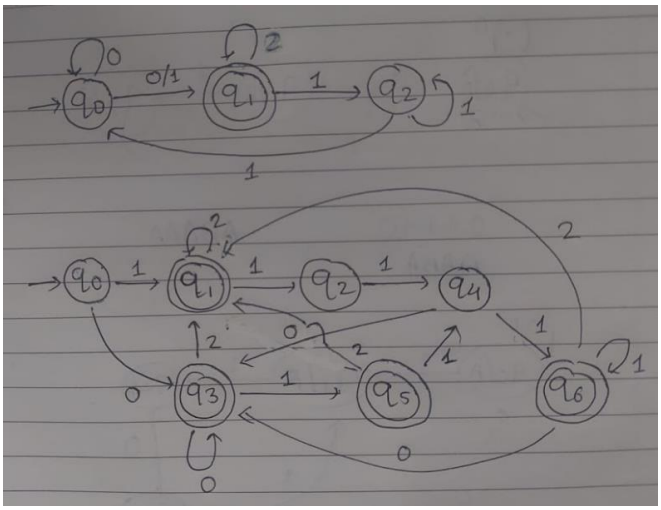
```
    for (auto it : stateMap2) {
        // check if it is a final state
        if (!final.count(it.first)) {
            // if it is not a final state then check if any of the states in the set is a
final state
            for (auto i : it.second) {
                if (final.count(i)) {
                    // 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:

```
20BCS021
FAIZAN CHOUDHARY

Given NFA:
0
1
0,1 1 -1
-1 2 1
-1 0,2 -1

Converted DFA:
0
1,3,5,6
3 1 -1
-1 2 1
-1 4 -1
3 5 1
3 6 -1
-1 4 1
3 6 1
```



		0	1	2
0	→ q_0	$[q_0q_1]$	q_1	-
1	q_1	-	q_2	q_1
2	q_2	-	$[q_0q_2]$	-
3	$[q_0q_1]$	$[q_0q_1]$	$[q_1q_2]$	q_1
4	$[q_0q_2]$	$[q_0q_1]$	$[q_0q_1q_2]$	-
5	$[q_1q_2]$	-	$[q_0q_2]$	q_1
6	$[q_0q_1q_2]$	$[q_0q_1]$	$[q_0q_1q_2]$	q_1

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:

```
Given CFG:
S A b B
S c S
A B A
A a
B b B
B #

First(A) : a b
First(B) : # b
First(S) : a b c
First(a) : a
First(b) : b
First(c) : c

Follows:
Follow(A) : b
Follow(B) : $ a b
Follow(S) : $
```

```
Given CFG:
E T G
G + T G
G #
T F U
U * F U
U #
F ( E )
F i

Firsts:
First(() : (
First()) : )
First(*) : *
First(+) : +
First(E) : ( i
First(G) : # +
First(T) : ( i
First(U) : # *
First(i) : i

Follows:
Follow(E) : $ )
Follow(F) : $ ) * +
Follow(G) : $ )
Follow(T) : $ ) +
Follow(U) : $ ) +
```

```
Given CFG:
S A B C D E
A a
A #
B b
B #
C c
D d
D #
E e
E #

Firsts:
First(A) : # a
First(B) : # b
First(C) : c
First(D) : # d
First(E) : # e
First(S) : a b c
First(c) : c
First(d) : d
First(e) : e

Follows:
Follow(A) : b c
Follow(B) : c
Follow(C) : $ d e
Follow(D) : $ e
Follow(E) : $
Follow(S) : $
```

```
Given CFG:
S A C B
S C b B
S B a
A d a
A B C
B g
B #
C h
C #

Firsts:
First(A) : # d g h
First(B) : # g
First(S) : # a b d g h
First(a) : a
First(b) : b
First(d) : d
First(g) : g
First(h) : h

Follows:
Follow(A) : $ g h
Follow(B) : $ a g h
Follow(C) : $ b g h
Follow(S) : $
```

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:

```
Given CFG:
0 S A b B
1 S c S
2 A B A
3 A a
4 A #
5 B b B
6 B #

Firsts:
First(A) : # a b
First(B) : # b
First(S) : a b c
First(a) : a
First(b) : b
First(c) : c

Follows:
Follow(B) : $ a b
Follow(S) : $

LL Table:

        a       b       c       $
A       2,3     4,2,3
B       6,5     6,5             6,5
S       0       0       1       0,1
```

20BCS021
FAIZAN CHOUDHARY

Given CFG:
0 E T G
1 G + T G
2 G #
3 T F U
4 U * F U
5 U #
6 F (E)
7 F i

Firsts:
First(() : (
First()) :)
First(*) : *
First(+) : +
First(E) : (i
First(F) : (i
First(G) : # +
First(T) : (i
First(U) : # *
First(i) : i

Follows:
Follow(E) : $)
Follow(F) : $) * +
Follow(G) : $)
Follow(T) : $) +
Follow(U) : $) +

	(	)	*	+	i	$
E	0				0	
F	6				7	
G		2		1		2
T	3				3	
U		5	4	5		5

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:

```
20BCS021
FAIZAN CHOUDHARY

Given CFG:
0 S a A C
1 S B b
2 A e D
3 B f
4 B g
5 C h
6 C i
7 D b E
8 D #
9 E e D
10 E d D


Given LL(1) parsing table:

              a             b             d             e             f             g             h             i             $
A                                                       2
B                                                                     3             4
C                                                                                                 5             6
D                           7                                                                     8             8
E                                         10            9
S             0                                                       1             1

Enter string to check: aebde

Top of Stack    Input String    Production applied
S               aebde$          S->aAC
A               ebde$           A->eD
D               bde$            D->bE
E               de$             E->dD
D               e$

String is not accepted!



Enter string to check: aebdh

Top of Stack    Input String    Production applied
S               aebdh$          S->aAC
A               ebdh$           A->eD
D               bdh$            D->bE
E               dh$             E->dD
D               h$              D->#
C               h$              C->h
$               $

String is accepted
```