LABFILE
(CEN 692) COMPILER DESIGN LAB

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

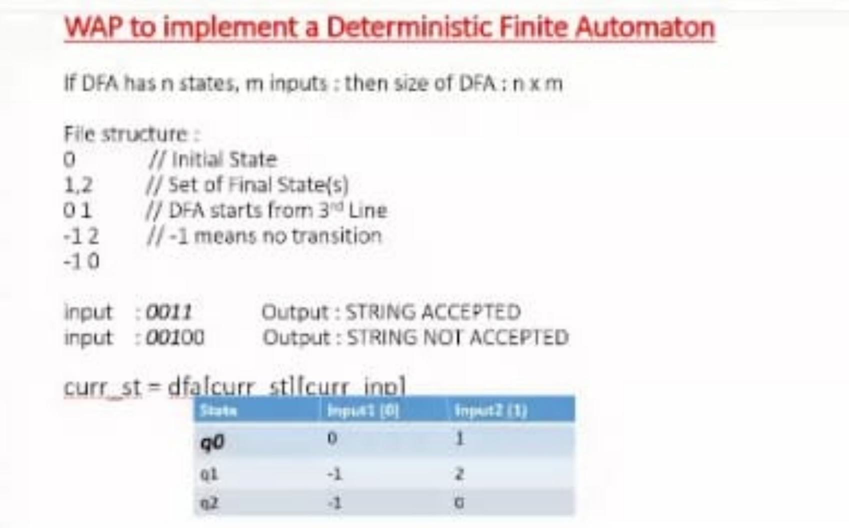
SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD lab Program 1 : Implementation of DFA



WAP to implement a Deterministic Finite Automaton

If DFA has n states, m inputs : then size of DFA : n x m

```
File structure :
0          // Initial State
1,2        // Set of Final State(s)
0 1        // DFA starts from 3rd Line
-1 2       // -1 means no transition
-1 0

input   : 0011       Output : STRING ACCEPTED
input   : 00100      Output : STRING NOT ACCEPTED

curr_st = dfa[curr_st][curr_inp]
```

State	Input1 (0)	Input2 (1)
q0	0	1
q1	-1	2
q2	-1	0

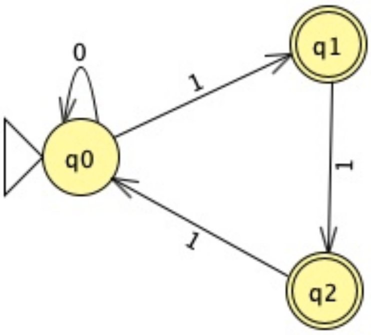
C++ Implementation

```cpp
//
//  main.cpp
//  CD Lab1
//
//  Created by YASH VINAYVANSHI on 19/01/22.
//

#include <iostream>
#include <fstream>
#include <set>
#include <sstream>
#include <tuple>
using std::ifstream;
using std::cerr;
using std::cout;
using std::cin;
using std::endl;
using std::string;
using std::set;
using std::stringstream;
int main(){
    cout<<"Author : YASH VINAYVANSHI\n"<<endl;
    string path;
    cout<<"Enter file path : "; getline(cin, path);
    ifstream file;
    string temp;
    //count no of states
    file.open(path);
    if(!file){
        cerr<<"File not found"<<endl;
        exit(1);
    }
    int count = 0;
    string line;
    while (getline(file, line))
        count++;
    int n = count - 2;
    file.close();

    //extract data and state table
    file.open(path);

    //get first line : assuming only one staring state
    getline(file, temp);
    int initial_state = stoi(temp);

    //get second line
    getline(file, temp);
    stringstream ss;
    ss << temp;
    string temp1;
    set<int> final_states;
    while(!ss.eof()){
        ss >> temp1;
        final_states.insert(stoi(temp1));
    }

    //get state transition diagram
    //assuming input alphabet is (0,1);
    int table[n][2];
    for(int i=0; i<n; i++){
        getline(file, temp);
        ss.clear();
        ss<<temp;
        ss>>temp1; table[i][0] = stoi(temp1);
        ss>>temp1; table[i][1] = stoi(temp1);
    }
    file.close();

    string input;
    while(input != "stop"){
        cout<<"Enter string : ";
        getline(cin, input);
        int current_state = initial_state;
        bool is_dead = false;
        for(int i=0; i<input.length(); i++){
            //check is input is correct
            if(!(input[i] == '0' || input[i] == '1')){
                cout<<"input is invalid"<<endl; break;
            }
            if(table[current_state][input[i] - 48]==-1){
                cout<<"String not accepted\n"<<endl;
                is_dead = true;;
                break;
            }
            current_state = table[current_state][input[i] - 48];
        }
        if(is_dead == true) continue;
        if(final_states.count(current_state) == 0)
            cout<<"String not accepted\n"<<endl;
        else
```

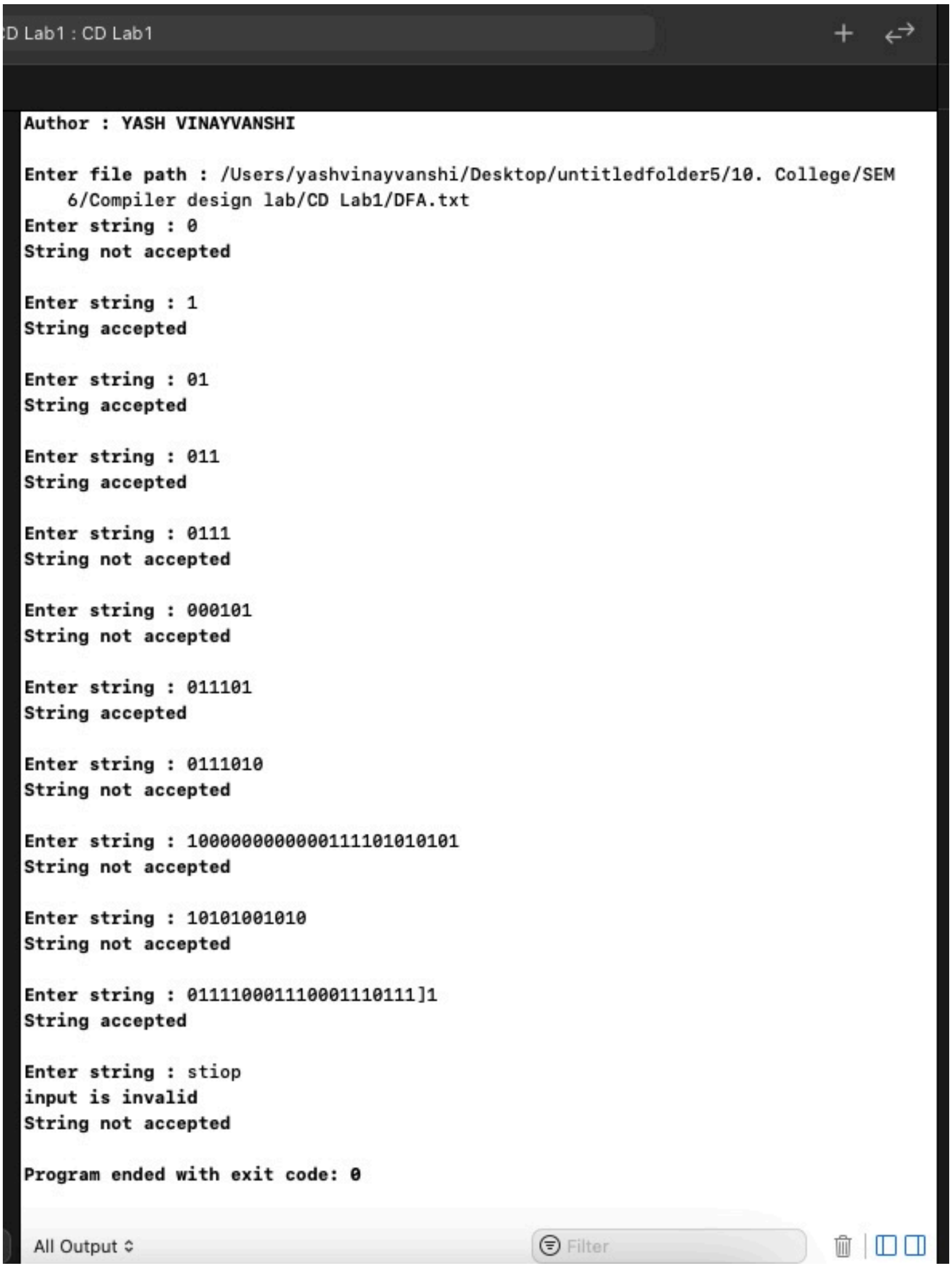
```
                cout<<"String accepted\n"<<endl;
        }
}
```

Run 1

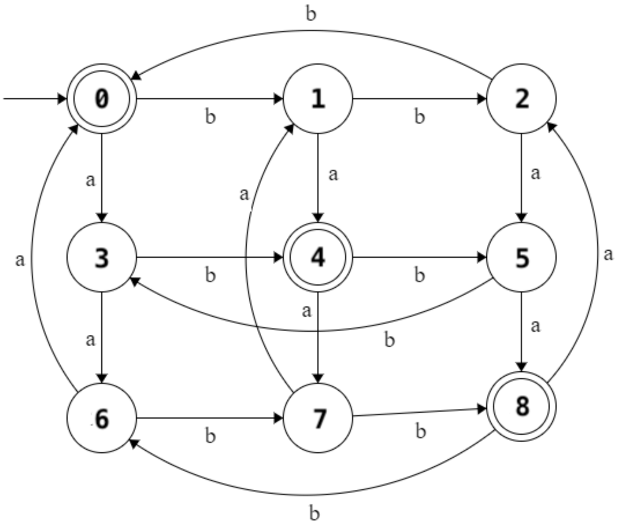
Input
```
0
1 2
0 1
-1 2
-1 0
```



Output



```
Author : YASH VINAYVANSHI

Enter file path : /Users/yashvinayvanshi/Desktop/untitledfolder5/10. College/SEM
    6/Compiler design lab/CD Lab1/DFA.txt
Enter string : 0
String not accepted

Enter string : 1
String accepted

Enter string : 01
String accepted

Enter string : 011
String accepted

Enter string : 0111
String not accepted

Enter string : 000101
String not accepted

Enter string : 011101
String accepted

Enter string : 0111010
String not accepted

Enter string : 1000000000000111101010101
String not accepted

Enter string : 10101001010
String not accepted

Enter string : 011110001110001110111]1
String accepted

Enter string : stiop
input is invalid
String not accepted

Program ended with exit code: 0
```

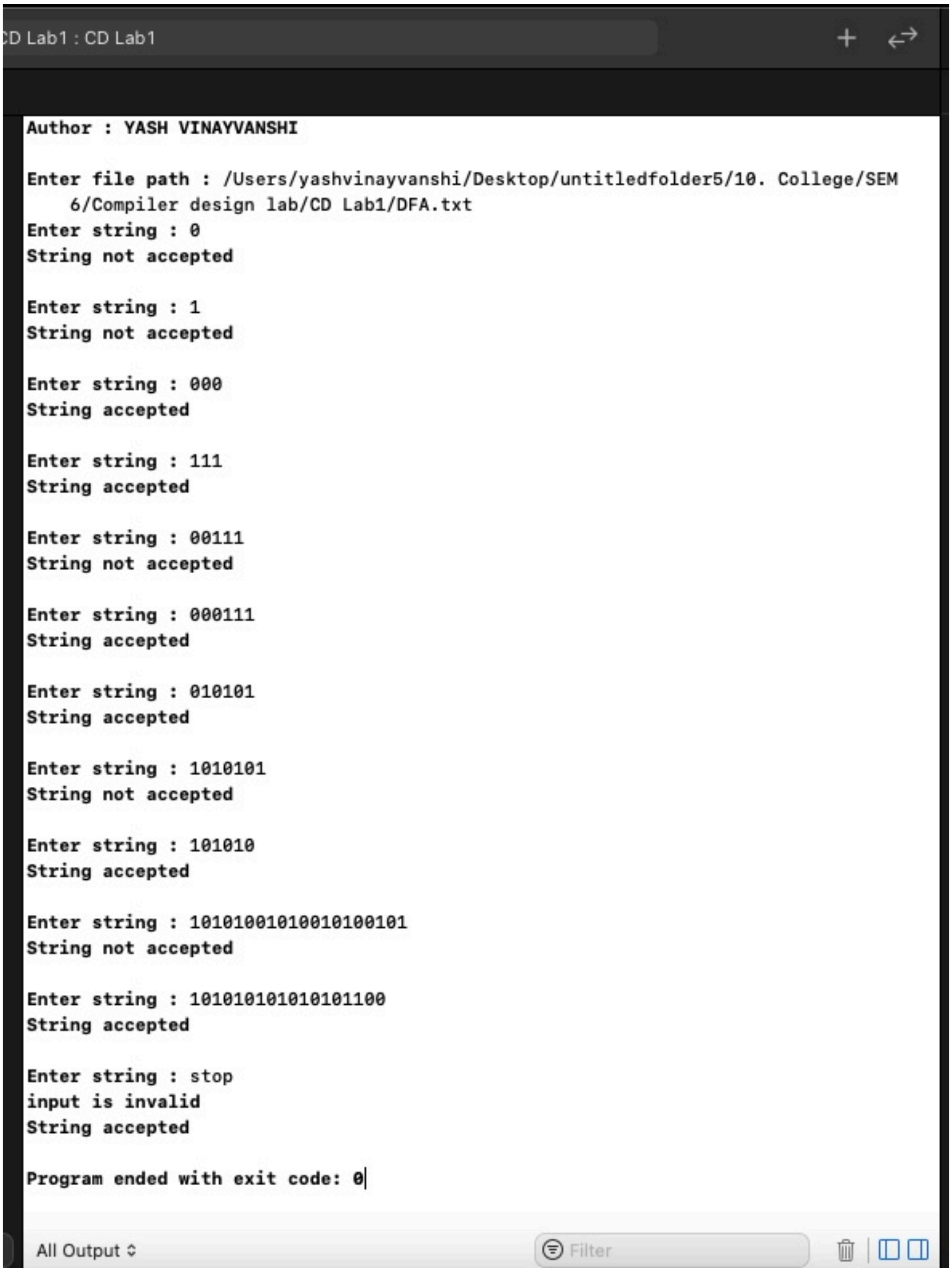


Run 2

Input

```
0
0 4 8
3 1
4 2
5 0
6 4
7 5
8 3
0 7
1 8
2 6
```

Language L = {w | w ∈ {0,1}* and $N_0(w)$ mod 3 = N_1 (w) mod 3}.



```
Author : YASH VINAYVANSHI

Enter file path : /Users/yashvinayvanshi/Desktop/untitledfolder5/10. College/SEM
    6/Compiler design lab/CD Lab1/DFA.txt
Enter string : 0
String not accepted

Enter string : 1
String not accepted

Enter string : 000
String accepted

Enter string : 111
String accepted

Enter string : 00111
String not accepted

Enter string : 000111
String accepted

Enter string : 010101
String accepted

Enter string : 1010101
String not accepted

Enter string : 101010
String accepted

Enter string : 10101001010010100101
String not accepted

Enter string : 101010101010101100
String accepted

Enter string : stop
input is invalid
String accepted

Program ended with exit code: 0
```

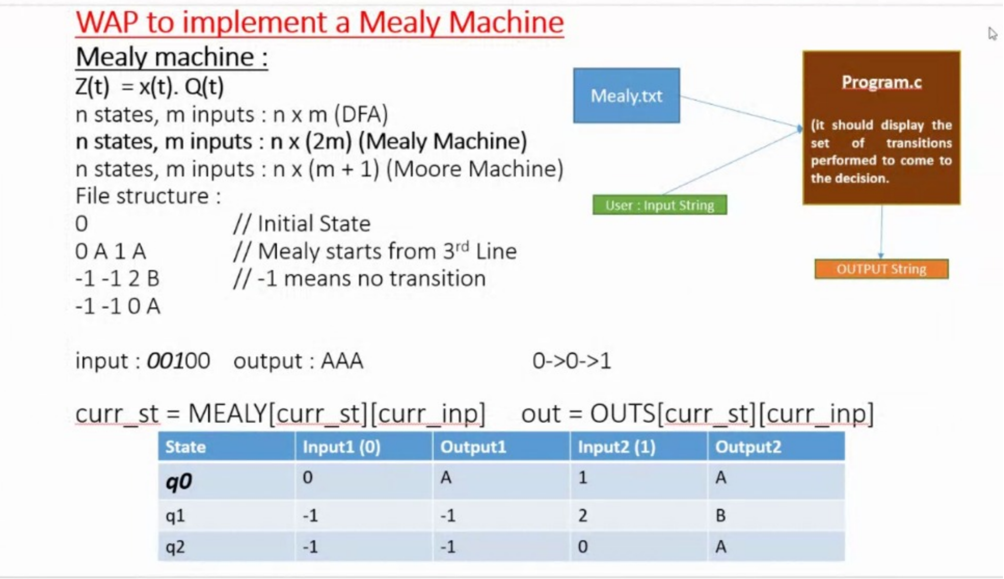
15 FEBRUARY 2022

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD lab Program 2 : Implementation of Mealy Machine



WAP to implement a Mealy Machine

Mealy machine :

Z(t) = x(t). Q(t)

n states, m inputs : n x m (DFA)

n states, m inputs : n x (2m) (Mealy Machine)

n states, m inputs : n x (m + 1) (Moore Machine)

File structure :

```
0          // Initial State
0 A 1 A    // Mealy starts from 3rd Line
-1 -1 2 B  // -1 means no transition
-1 -1 0 A
```

Mealy.txt

User : Input String

Program.c

(it should display the set of transitions performed to come to the decision.

OUTPUT String

input : *001*00 output : AAA 0->0->1

curr_st = MEALY[curr_st][curr_inp] out = OUTS[curr_st][curr_inp]

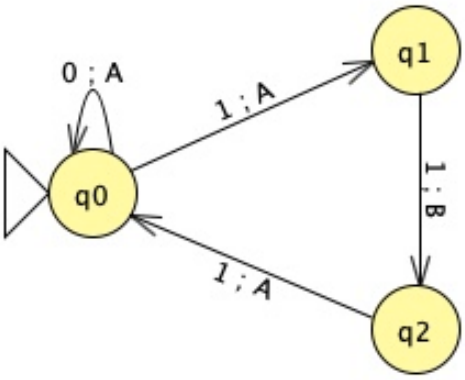
State	Input1 (0)	Output1	Input2 (1)	Output2
q0	0	A	1	A
q1	-1	-1	2	B
q2	-1	-1	0	A

C++ Implementation

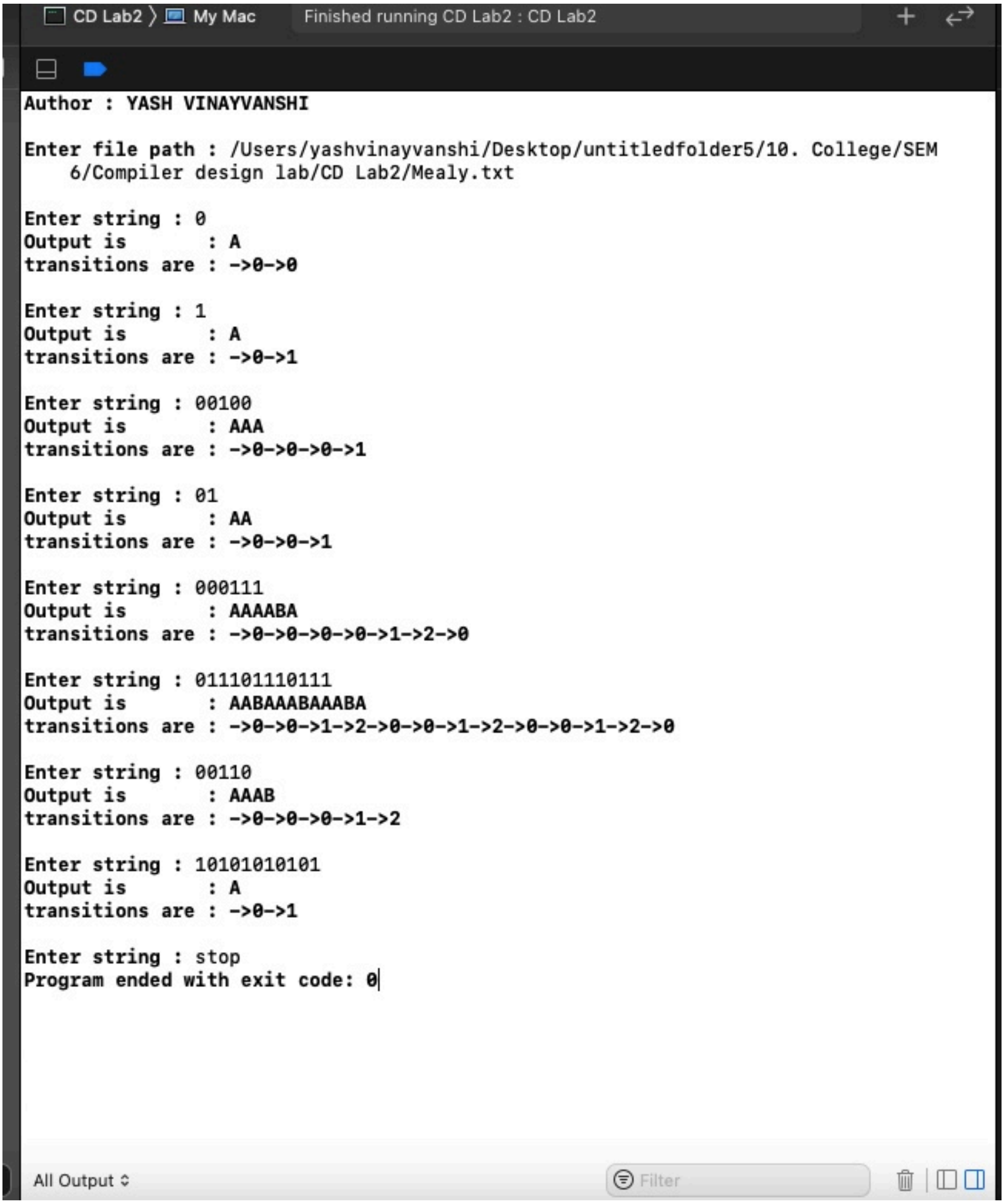
```
//
//  main.cpp
//  CD Lab2
//
//  Created by YASH VINAYVANSHI on 14/02/22.
//

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
using std::ifstream;
using std::cerr;
using std::cout;
using std::cin;
using std::endl;
using std::string;
using std::vector;
using std::to_string;
using std::stringstream;
int main(){
    cout<<"Author : YASH VINAYVANSHI\n"<<endl;
    string path;
    cout<<"Enter file path : "; getline(cin, path);
    ifstream file;
    string temp;
    //count no of states
    file.open(path);
    if(!file){
        cerr<<"File not found"<<endl;
        exit(1);
    }
    int count = 0;
    string line;
    while (getline(file, line))
        count++;
    int n = count - 1;
    file.close();

    //extract data and state table
    file.open(path);

    //get first line : assuming only one starting state
    getline(file, temp);
    int initial_state = stoi(temp);

    //get state transition diagram
    //assuming input alphabet is (0,1);
    int table[n][2];
    char out[n][2];
    stringstream ss; string temp1;
    for(int i=0; i<n; i++){
        getline(file, temp);
        ss.clear();
        ss<<temp;
        ss>>temp1; table[i][0] = stoi(temp1);
        ss>>temp1; out[i][0] = temp1[0];
        ss>>temp1; table[i][1] = stoi(temp1);
        ss>>temp1; out[i][1] = temp1[0];
    }
    file.close();

    string input;
    while(1){
        cout<<"\nEnter string : ";
        getline(cin, input);
        if(input == "stop") return 0;
        string output = "";
        string transition = "->" + to_string(initial_state);
        int current_state = initial_state;
        for(int i=0; i<input.length(); i++){
            //check is input is correct
            if(!(input[i] == '0' || input[i] == '1')){
                cout<<"input is invalid"<<endl; break;
            }
            if(table[current_state][input[i] - 48]==-1){ break; }
            output += out[current_state][input[i] - 48];
            current_state = table[current_state][input[i] - 48];
            transition+="->"+to_string(current_state);
        }
        cout<<"Output is       : "<<output<<endl;
        cout<<"transitions are : "<<transition<<endl;
    }
}
```

Run 1

Input
```
0
0 A 1 A
-1 -1 2 B
-1 -1 0 A
```



Output



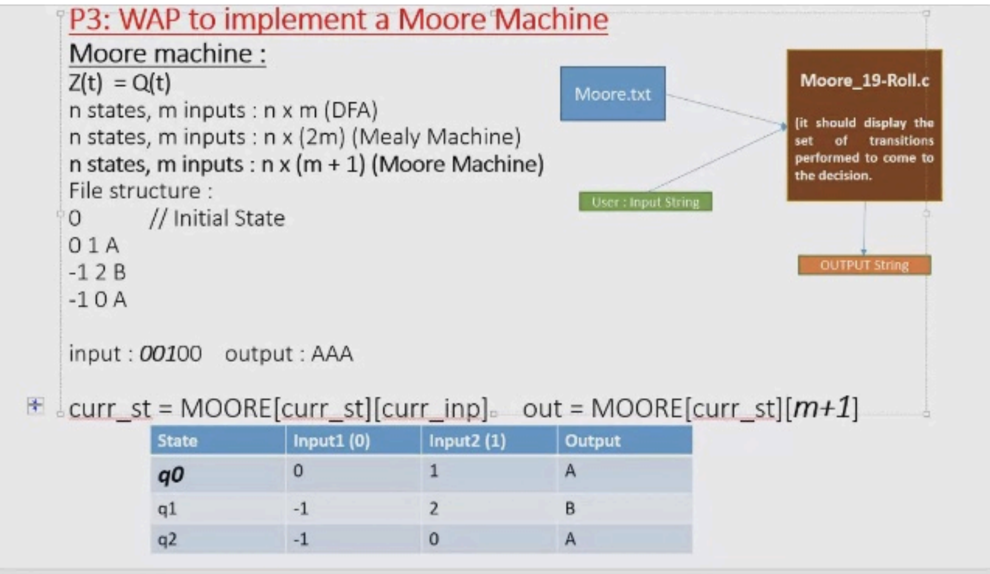
```
Author : YASH VINAYVANSHI

Enter file path : /Users/yashvinayvanshi/Desktop/untitledfolder5/10. College/SEM
    6/Compiler design lab/CD Lab2/Mealy.txt

Enter string : 0
Output is       : A
transitions are : ->0->0

Enter string : 1
Output is       : A
transitions are : ->0->1

Enter string : 00100
Output is       : AAA
transitions are : ->0->0->0->1

Enter string : 01
Output is       : AA
transitions are : ->0->0->1

Enter string : 000111
Output is       : AAAABA
transitions are : ->0->0->0->0->1->2->0

Enter string : 011101110111
Output is       : AABAAABAAABA
transitions are : ->0->0->1->2->0->0->1->2->0->0->1->2->0

Enter string : 00110
Output is       : AAAB
transitions are : ->0->0->0->1->2

Enter string : 10101010101
Output is       : A
transitions are : ->0->1

Enter string : stop
Program ended with exit code: 0
```

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD lab Program 3 : WAP to implement a Moore Machine



P3: WAP to implement a Moore Machine

Moore machine :
Z(t) = Q(t)
n states, m inputs : n x m (DFA)
n states, m inputs : n x (2m) (Mealy Machine)
n states, m inputs : n x (m + 1) (Moore Machine)
File structure :
0 // Initial State
0 1 A
-1 2 B
-1 0 A

input : *00100* output : AAA

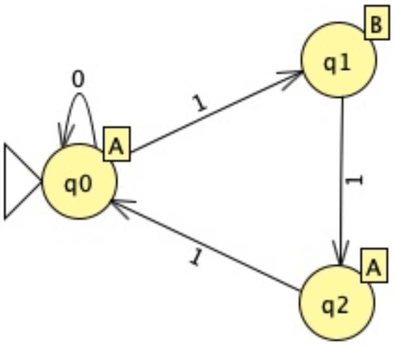
curr_st = MOORE[curr_st][curr_inp] out = MOORE[curr_st][*m+1*]

State	Input1 (0)	Input2 (1)	Output
q0	0	1	A
q1	-1	2	B
q2	-1	0	A

[it should display the set of transitions performed to come to the decision.

C++ Implementation

```
//
//  main.cpp
//  CD ab3
//
//  Created by YASH VINAYVANSHI on 16/02/22.
//

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
using std::ifstream;
using std::cerr;
using std::cout;
using std::cin;
using std::endl;
using std::string;
using std::vector;
using std::to_string;
using std::stringstream;
int main(){
    cout<<"Author : YASH VINAYVANSHI\n"<<endl;
    string path;
    cout<<"Enter file path : "; getline(cin, path);
    ifstream file;
    string temp;
    //count no of states
    file.open(path);
    if(!file){
        cerr<<"File not found"<<endl;
        exit(1);
    }
    int count = 0;
    string line;
    while (getline(file, line))
        count++;
    int n = count - 1;
    file.close();

    //extract data and state table
    file.open(path);

    //get first line : assuming only one starting state
    getline(file, temp);
    int initial_state = stoi(temp);

    //get state transition diagram
    //assuming input alphabet is (0,1);
    int table[n][2];
    char out[n];
    stringstream ss; string temp1;
    for(int i=0; i<n; i++){
        getline(file, temp);
        ss.clear();
        ss<<temp;
        ss>>temp1; table[i][0] = stoi(temp1);
        ss>>temp1; table[i][1] = stoi(temp1);
        ss>>temp1; out[i] = temp1[0];
    }
    file.close();

    string input;
    while(1){
        cout<<"\nEnter string : ";
        getline(cin, input);
        if(input == "stop") return 0;
        string output = "";
        string transition = "->" + to_string(initial_state);
        int current_state = initial_state;
        for(int i=0; i<input.length(); i++){
            //check is input is correct
            if(!(input[i] == '0' || input[i] == '1')){
                cout<<"input is invalid"<<endl; break;
            }
            if(table[current_state][input[i] - 48]==-1){ break; }
            //initial state output is discarded
            current_state = table[current_state][input[i] - 48];
            output += out[current_state];
            transition+="->"+to_string(current_state);
        }
        cout<<"Output is       : "<<output<<endl;
        cout<<"transitions are : "<<transition<<endl;
    }
}
```

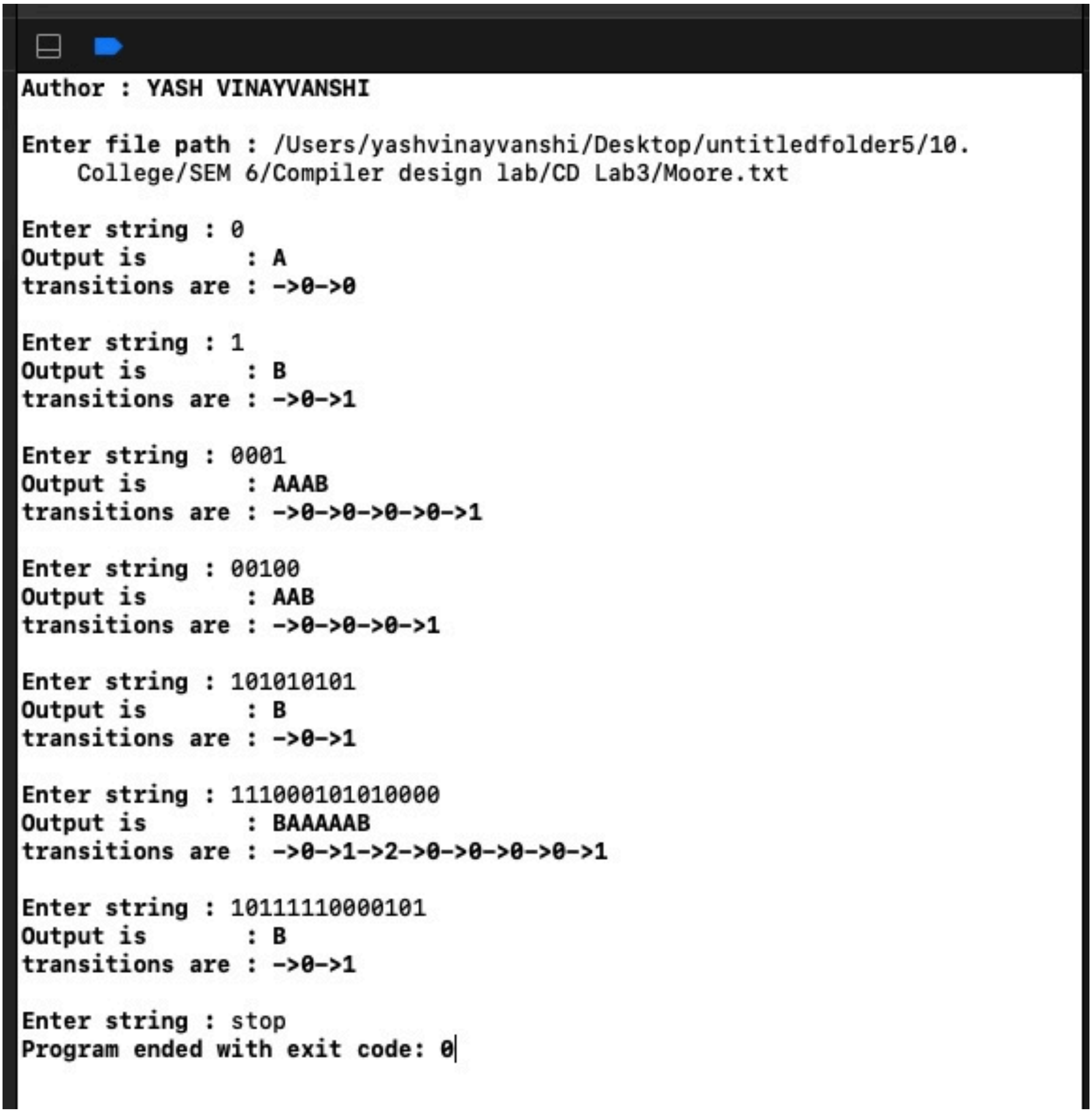


Run 1

Input

```
0
0 1 A
-1 2 B
-1 0 A
```

Output



```
Author : YASH VINAYVANSHI

Enter file path : /Users/yashvinayvanshi/Desktop/untitledfolder5/10.
    College/SEM 6/Compiler design lab/CD Lab3/Moore.txt

Enter string : 0
Output is        : A
transitions are : ->0->0

Enter string : 1
Output is        : B
transitions are : ->0->1

Enter string : 0001
Output is        : AAAB
transitions are : ->0->0->0->0->1

Enter string : 00100
Output is        : AAB
transitions are : ->0->0->0->1

Enter string : 101010101
Output is        : B
transitions are : ->0->1

Enter string : 111000101010000
Output is        : BAAAAAB
transitions are : ->0->1->2->0->0->0->0->1

Enter string : 10111110000101
Output is        : B
transitions are : ->0->1

Enter string : stop
Program ended with exit code: 0
```

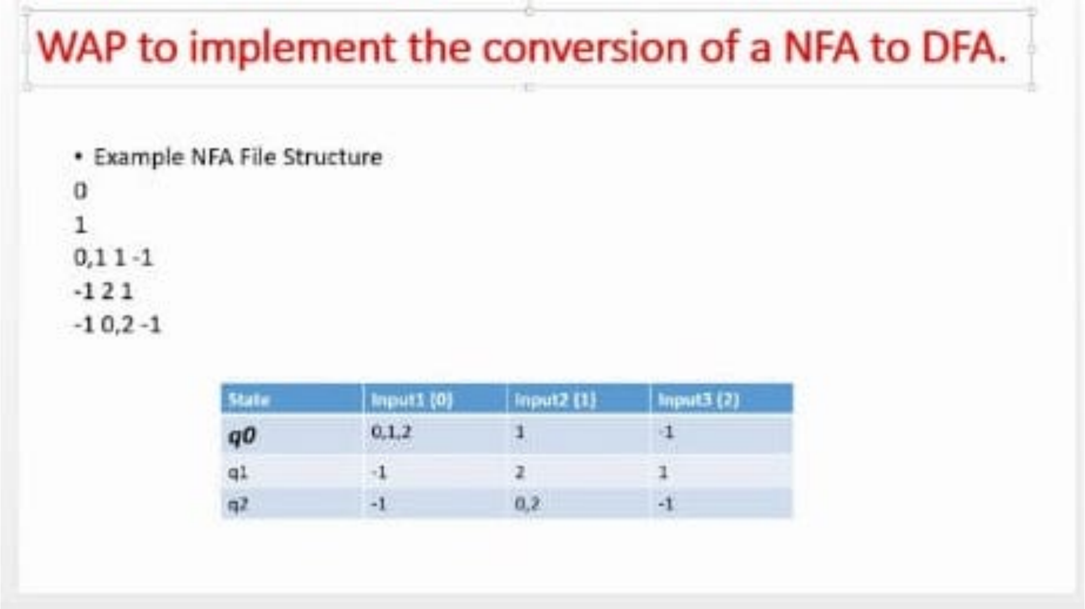
09 MARCH 2022

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD lab Program 4 : WAP to implement the conversion of a NFA to DFA



WAP to implement the conversion of a NFA to DFA.

- Example NFA File Structure

```
0
1
0,1 1 -1
-1 2 1
-1 0,2 -1
```

State	Input1 (0)	Input2 (1)	Input3 (2)
q0	0,1,2	1	-1
q1	-1	2	1
q2	-1	0,2	-1

C++ Implementation

```
//
//  main.cpp
//  NFAtoDFA
//
//  Created by YASH VINAYVANSHI on 09/03/22.
//

#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <set>
#include <string>
#include <map>
using namespace std;

//set<set<int>> Intersection(set<set<int>>, set<set<int>>);
//set<int> Union(set<int>, set<int>);
set<int> Difference(set<int>, set<int>);
```

```
set<int> Union(set<int>, set<int>);
void printSet(set<int>);
void printMap(map<set<int>, int>);
int main(){
    //general input
    string ipath;
    cout<<"Created by YASH VINAYVANSHI"<<endl;
    cout<<"Enter input file path : "; cin>>ipath;
    vector<vector<string>> content;
    vector<string> row;
    string line, word;
    ifstream ifile;
    ifile.open(ipath);
    if(ifile.is_open()){
        while(getline(ifile, line)){
            row.clear();
            stringstream str(line);
            while(getline(str, word, ' ')) row.push_back(word);
            content.push_back(row);
        }
    }
    else{ cout<<"i/p File not opened\n"; return 0;}
    ifile.close();

    /*
    string opath;
    cout<<"Enter output file path : "; cin>>opath;
    ofstream ofile;
    ofile.open(opath);
    if(!ofile.is_open()){
        cout<<"o/p File not opened\n"; return 0;
    }
    */

    //extract start state
    int start_state = stoi(content[0][0]);
    cout<<"start_state : "<<start_state<<endl;

    //extract final states
    set<int> final_states;
    stringstream ss(content[1][0]);
    while (ss.good()){
         string substr;
         getline(ss, substr, ',');
         final_states.insert(stoi(substr));
    }
    cout<<"final_states : "; printSet(final_states);


    //input transitions & make transition list
    //a state is a set of integers
    //below is the set of such states
    int n = content.size();
    int id = 0;
    map<set<int>, int> mapping; //map states to ids
    map<int, set<int>> reverse_mapping;
```

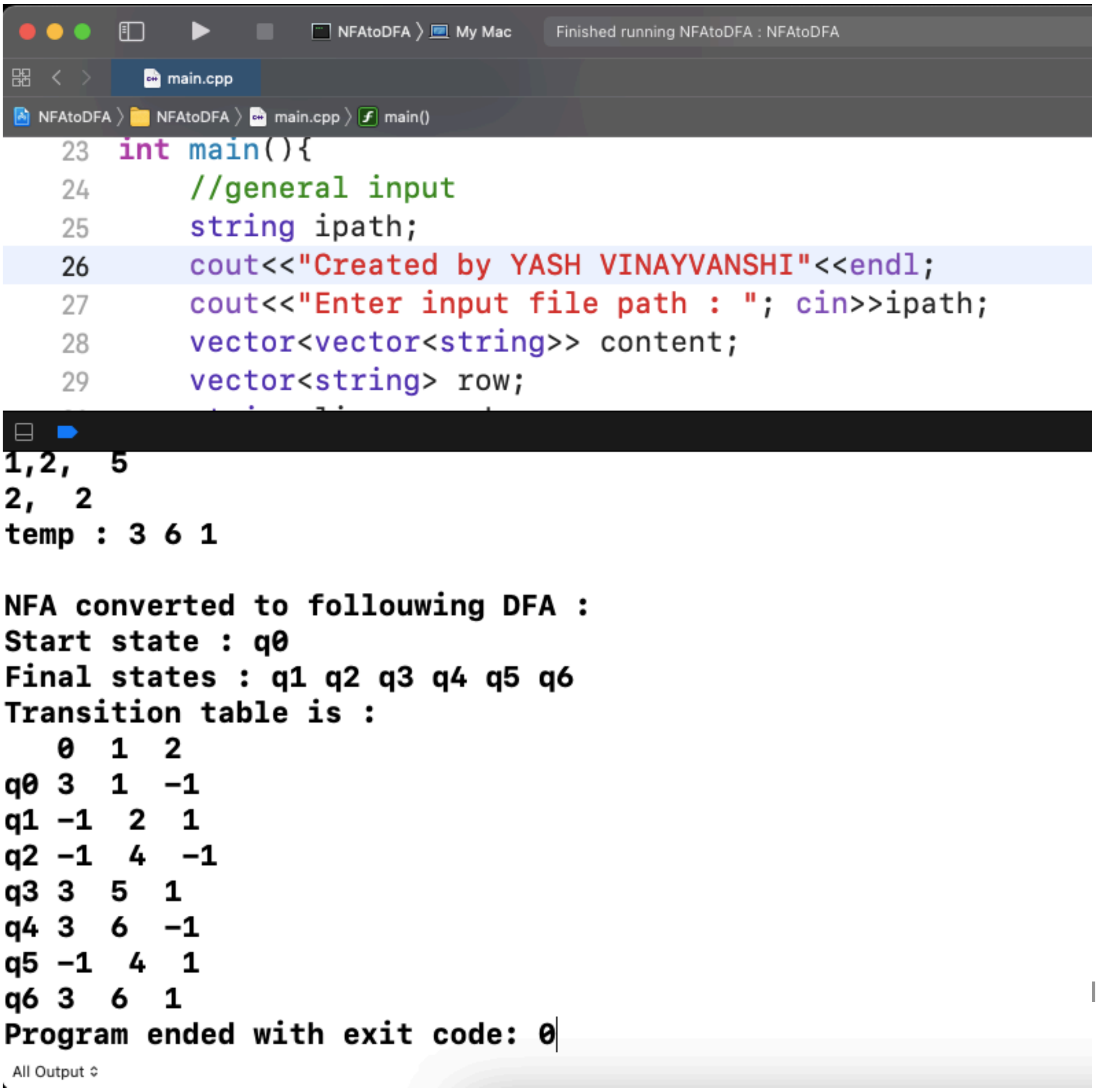
```
set<set<int>> states; //set of states
set<int> states_in_table; //set of ids of states in DFA
set<int> all_states;
vector<vector<int>> DFA;     //to store table of DFA

//init DFA table & states
states.insert({});
mapping[{}] = -1;
reverse_mapping[-1] = {};
for(int i=2; i<n; i++){
    states.insert({i-2});
    mapping[{i-2}] = id;
    states_in_table.insert(id);
    all_states.insert(id);
    reverse_mapping[id]={i-2};
    id++;
}
cout<<"init_states_in_table : "; printSet(states_in_table);
cout<<"init_all_states : "; printSet(all_states);


for(int i=2; i<n; i++){
    //get row of table
    vector<int> temp;
    for(int j=0; j<3; j++){
        string substr;
        stringstream s(content[i][j]);
        set<int> temp1;
        while (s.good()) {
             string substr;
             getline(s, substr, ',');
             temp1.insert(stoi(substr));
        }
        int idtemp;
        if(temp1.count(-1)){
            //no transition
            idtemp=-1;
            mapping[temp1] = -1;
            reverse_mapping[-1] = temp1;
        }
        if(mapping.count(temp1)){
            idtemp = mapping[temp1];
        }
        else{
            //new state found
            //add in all_states
            mapping[temp1] = id;
            reverse_mapping[id] = temp1;
            idtemp = id;
            id++;
            states.insert(temp1);
            all_states.insert(idtemp);
        }
        temp.push_back(idtemp);
    }
    DFA.push_back(temp);
    }
    cout<<"pre_states_in_table : "; printSet(states_in_table);
    cout<<"pre_all_states : "; printSet(all_states);
    cout<<"pre_mapping : "; printMap(mapping);

    //set<int> difference = Difference(states_in_table, all_states);
    set<int> difference = {1};
    cout<<"first_difference : "; printSet(difference);

    //states which are not yet calculates in the table
    while(difference.size()!=0){
        difference = Difference(states_in_table, all_states);
        set<int> new_state;
        for(auto it = difference.begin(); it!=difference.end(); it++){
            //new entry for left out states
            set<int> state = reverse_mapping[*it];
            cout<<*it<<" state : "; printSet(state);
            vector<int> temp;
            for(int j=0;  j<3; j++){
                new_state.clear();
                for(auto it1 = state.begin(); it1 != state.end() ; it1+
+){
                    //take union actrss column
                    new_state = Union(new_state,
reverse_mapping[DFA[*it1][j]]);
                }
                cout<<"New State : "; printSet(new_state);
                if(mapping.count(new_state)){
                    cout<<"state exist"<<endl;
                    temp.push_back(mapping[new_state]);
                    continue;
                }
                else{
                    cout<<"state does not exist"<<endl;
                    mapping[new_state] = id;
                    reverse_mapping[id] = new_state;
                    all_states.insert(id);
                    id++;
                    states.insert(new_state);
                    temp.push_back(mapping[new_state]);
                }
            }
            //row for *it complete
            states_in_table.insert(*it);
            DFA.push_back(temp);
            cout<<"new_states_in_table : "; printSet(states_in_table);
            cout<<"new_all_states : "; printSet(all_states);
            cout<<"new_mapping : "; printMap(mapping);
            cout<<"temp : ";
            for(auto const &it:temp){
                cout<<it<<" ";
            }
            cout<<endl;

        }
    }
    //determine final states
    set<int> final_states_DFA;
    for(auto it : all_states){
        set<int> state = reverse_mapping[it];
        for(auto it1 : state)
            if(final_states.count(it1)) final_states_DFA.insert(it);
    }
    cout<<"\nNFA converted to follouwing DFA : "<<endl;
    cout<<"Start state : q"<<start_state<<endl;
    cout<<"Final states : "; for(auto it : final_states_DFA)
cout<<"q"<<it<<" ";
    cout<<endl;
    cout<<"Transition table is : "<<endl;
    cout<<"   0  1  2"<<endl;
    int count = 0;
    for(auto it=DFA.begin(); it!=DFA.end(); it++){
        cout<<"q"<<count<<" ";
        for(auto it1 = it->begin(); it1!=it->end(); it1++){
            cout<<*it1<<"  ";
        }
        count++;
        cout<<endl;
    }
}
set<int> Difference(set<int> states_in_table, set<int> all_states){
    set<int> difference;
    bool is_found;
    for(auto it=all_states.begin(); it!=all_states.end(); it++){
        is_found = false;
        for(auto it1=states_in_table.begin(); it1!
=states_in_table.end(); it1++){
            if(*it == *it1){
                is_found = true;
                break;
            }
        }
        if(is_found == false){
            difference.insert(*it);
        }
    }
    return difference;
}
set<int> Union(set<int> s1, set<int> s2){
    set<int> uni;
    for(auto it=s1.begin(); it!=s1.end(); it++){
        uni.insert(*it);
    }
    for(auto it=s2.begin(); it!=s2.end(); it++){
        if(*it == -1) continue;
        uni.insert(*it);
    }
    return uni;
}
void printSet(set<int> s){
    for(auto it=s.begin(); it!=s.end(); it++){
        cout<<*it<<" ";
    }
    cout<<endl;
}
void printMap(map<set<int>, int> map){
    for(auto it : map){
        for(auto it1 : it.first){
            cout<<it1<<",";
        }
        cout<<"  "<<it.second<<endl;
    }
}
```

Run 1

Input

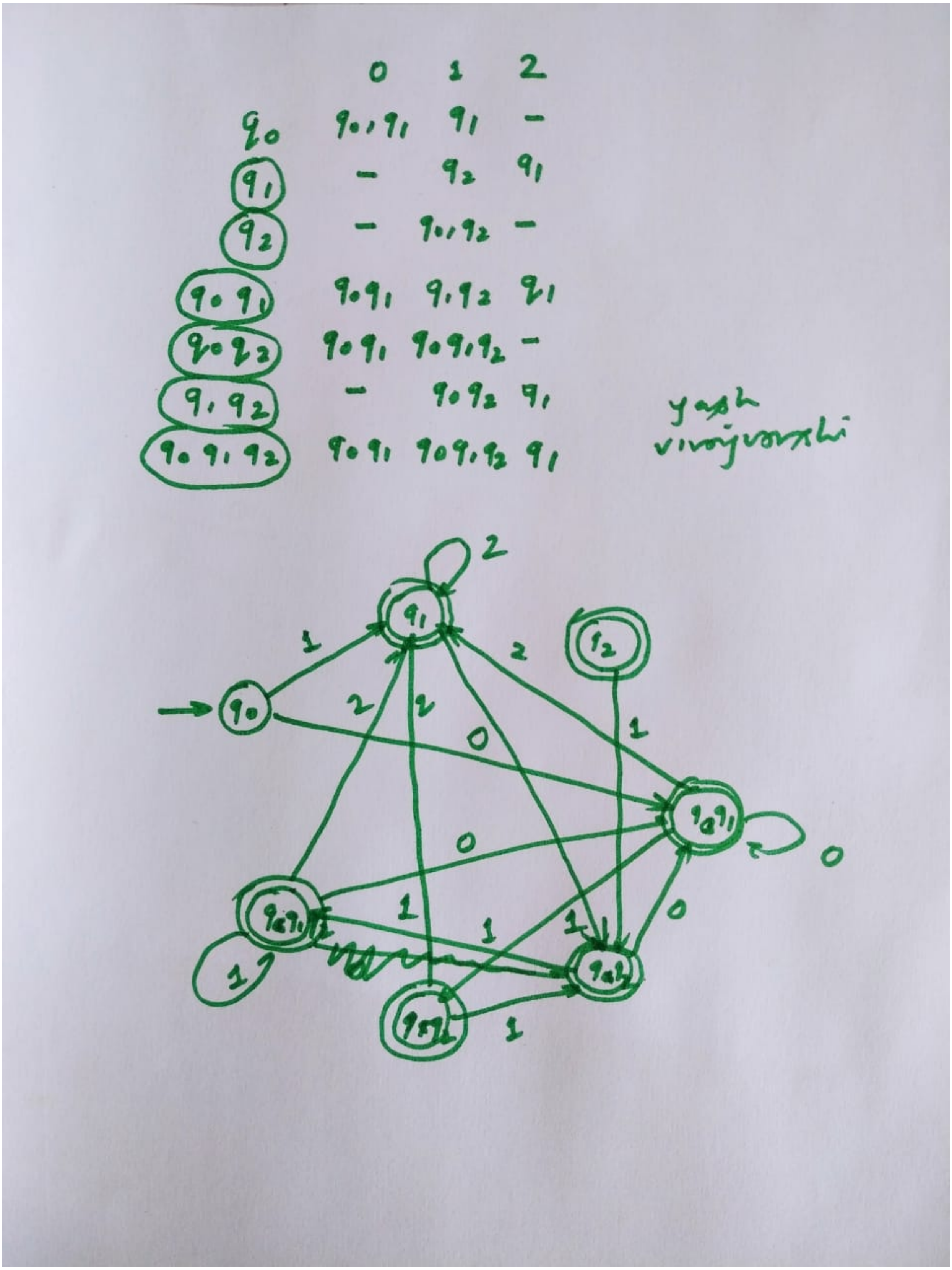
```
0
1,2
0,1 1 -1
-1 2 1
-1 0,2 -1
```

Output


NFAtoDFA 〉 My Mac
Finished running NFAtoDFA : NFAtoDFA
main.cpp
NFAtoDFA 〉 NFAtoDFA 〉 main.cpp 〉 main()
All Output ◇

```
int main(){
    //general input
    string ipath;
    cout<<"Created by YASH VINAYVANSHI"<<endl;
    cout<<"Enter input file path : "; cin>>ipath;
    vector<vector<string>> content;
    vector<string> row;
```

```
1,2,   5
2,   2
temp : 3 6 1

NFA converted to follouwing DFA :
Start state : q0
Final states : q1 q2 q3 q4 q5 q6
Transition table is :
   0  1  2
q0 3  1  -1
q1 -1  2  1
q2 -1  4  -1
q3 3  5  1
q4 3  6  -1
q5 -1  4  1
q6 3  6  1
Program ended with exit code: 0
```



	0	1	2
q_0	q_0, q_1	q_1	-
q_1	-	q_2	q_1
q_2	-	q_0, q_2	-
$q_0 q_1$	$q_0 q_1$	$q_1 q_2$	q_1
$q_0 q_2$	$q_0 q_1$	$q_0 q_1, q_2$	-
$q_1 q_2$	-	$q_0 q_2$	q_1
$q_0 q_1 q_2$	$q_0 q_1$	$q_0 q_1, q_2$	q_1

yash vinayvanshi

The program result for given test case matches the result by manual implementation.

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD lab Program 5 : write a program to implement a Regular Grammar, the Program should read an R.G through a file & should check whether a string given from the console is acceptable by the given R.G or not.

Approach
We have built a C++ recursive parser generator, this program reads the grammar from a file and outputs a C++ recursive descent parser for this grammar. The recursive descent parser program takes as input a sentence and outputs if it can be parsed with the grammar or not.

Input File format
1. All capital letters are considered variables
2. All other letters are considered terminals
3. epsilon is denoted by @
4. The RHS of first production is considered to be start variable
Example
S -> Ab|Bb
A -> Aa|a
B -> b|epsilon
Above grammar will be inputted as
S Ab Bb
A Aa a
B b @

C++ Implementation : Recursive descent parser generator

```
//
//  main.cpp
//  recursive descent parser builder
//
//  Created by YASH VINAYVANSHI on 14/03/22.
//

#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <fstream>
#include <sstream>
#include <cstring>
using namespace std;

int main(){
```

```
string ipath;
cout<<"Created by YASH VINAYVANSHI"<<endl;
cout<<"Enter input file path : "; cin>>ipath;
vector<vector<string>> content;
vector<string> row;
string line, word;
ifstream ifile;
ifile.open(ipath);
if(ifile.is_open()){
    while(getline(ifile, line)){
        row.clear();
        stringstream str(line);
        while(getline(str, word, ' ')) row.push_back(word);
        content.push_back(row);
    }
}
else{ cout<<"i/p File not opened\n"; return 0;}
ifile.close();

char startvar='S';
set<char> variables;
set<char> terminals;
map<char, vector<vector<char>>> productions;

int n = content.size();
for(int i=0; i<n; i++){
    char RHS[2]; strcpy(RHS, content[i][0].c_str());
    variables.insert(RHS[0]);
    if(i==0){ startvar = RHS[0]; }
    int n1 = content[i].size();
    vector<vector<char>> prod;
    for(int j=1; j<n1; j++){
        int n2 = content[i][j].size();
        char ar[n2+1]; strcpy(ar, content[i][j].c_str());
        vector<char> rule;
        for(int k=0; k<n2; k++){rule.push_back(ar[k]);}
        prod.push_back(rule);
        for(auto it : rule){
            if(it>='A' && it<='Z') variables.insert(it);
            else{ if(it != '@') terminals.insert(it);}
        }

    }
    productions[RHS[0]] = prod;
}

cout<<"#include <iostream>\n";
cout<<"using namespace std;\n";

//global declarations
cout<<"char l;\n";
cout<<"int i = 0;\n";
cout<<"string sentence;\n";

//no. of functions = no. of variables
//function protype declatation
```

```
    cout<<"void match(char);\n";
    for(auto it : variables){
        cout<<"void "<<it<<"(void);\n";
    }

    //main() function
    cout<<"int main(){\n";
    cout<<"cout<<\"Enter string to be parsed : \";";
    cout<<"cin>>sentence;\n";
    cout<<"l = sentence[i];\n";
    cout<<startvar<<"();\n";
    cout<<"if(l=='$') cout<<\" Parsing successful \";\n";
    cout<<"}\n";

    //match function
    cout<<"void match(char t){\n";
    cout<<"if(l==t) l=sentence[++i];";
    cout<<"else {cout<<\" parsing unsuccessful \"; exit(0);}\n";
    cout<<"}\n";

    //variable functions
    for(auto it : variables){
        cout<<"void "<<it<<"(){\n";
        for(auto it1 : productions[it]){
            cout<<"if(l=='"<<it1[0]<<"'){\n";
            if(it1[0]=='@')
                cout<<"return;\n";
            else{
                for(auto it2 : it1){
                    if(terminals.find(it2) != terminals.end())
                        cout<<"match('"<<it2<<"');\n";
                    if(variables.find(it2) != variables.end())
                        cout<<it2<<"();\n";
                }
            }
            cout<<"}\n";
        }
        cout<<"}\n";
    }
}
```

Run 1

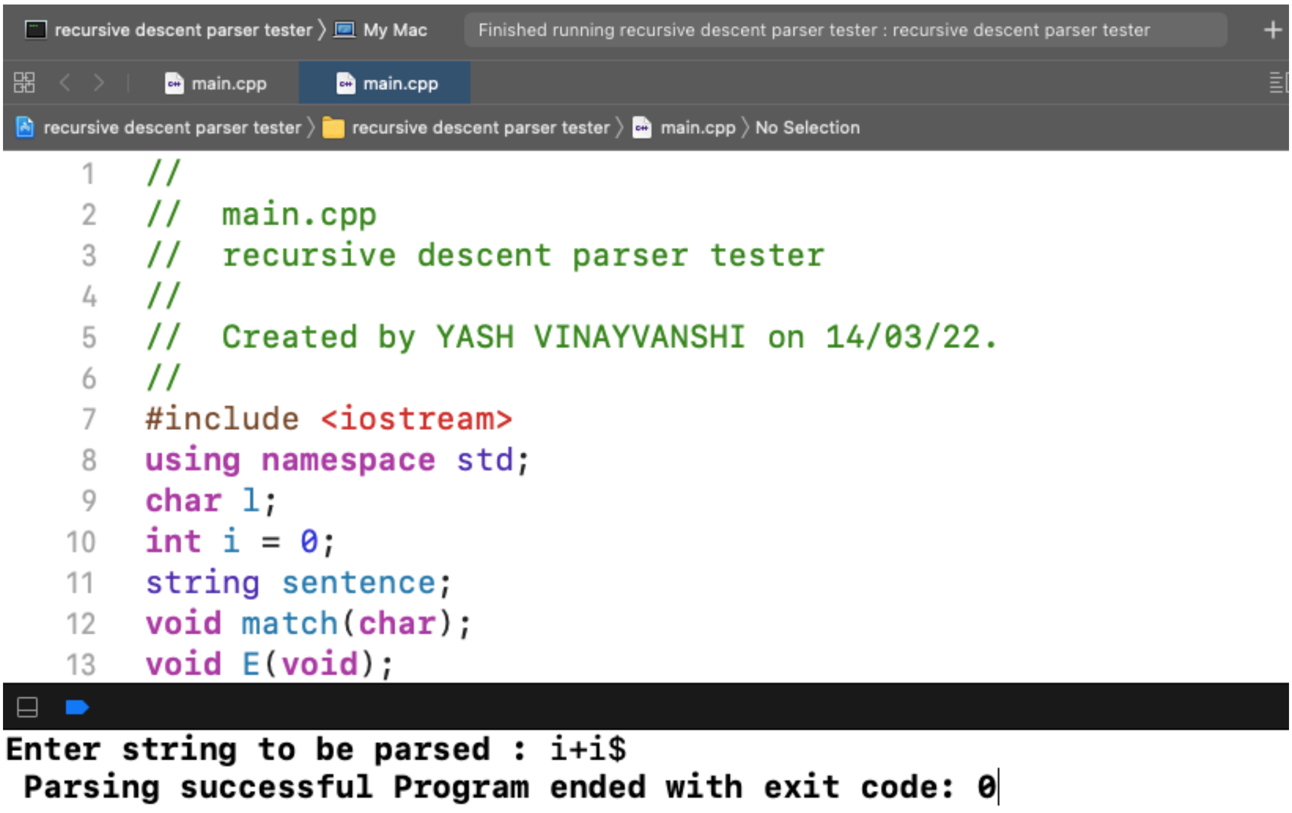
Input

```
E iF
F +iF @
```

Output : Recursive descent parser for above grammar

```
#include <iostream>
using namespace std;
char l;
int i = 0;
string sentence;
void match(char);
void E(void);
void F(void);
int main(){
cout<<"Enter string to be parsed : ";cin>>sentence;
l = sentence[i];
E();
if(l=='$') cout<<" Parsing successful ";
}
void match(char t){
if(l==t) l=sentence[++i];else {cout<<" parsing unsuccessful "; exit(0);}
}
void E(){
if(l=='i'){
match('i');
F();
}
}
void F(){
if(l=='+'){
match('+');
match('i');
F();
}
if(l=='@'){
return;
}
}
```

The above parser is executed


recursive descent parser tester
My Mac
Finished running recursive descent parser tester : recursive descent parser tester
main.cpp
main.cpp
recursive descent parser tester
recursive descent parser tester
main.cpp
No Selection

```
//
//  main.cpp
//  recursive descent parser tester
//
//  Created by YASH VINAYVANSHI on 14/03/22.
//
#include <iostream>
using namespace std;
char l;
int i = 0;
string sentence;
void match(char);
void E(void);
```

```
Enter string to be parsed : i+i$
 Parsing successful Program ended with exit code: 0
```

Run 2

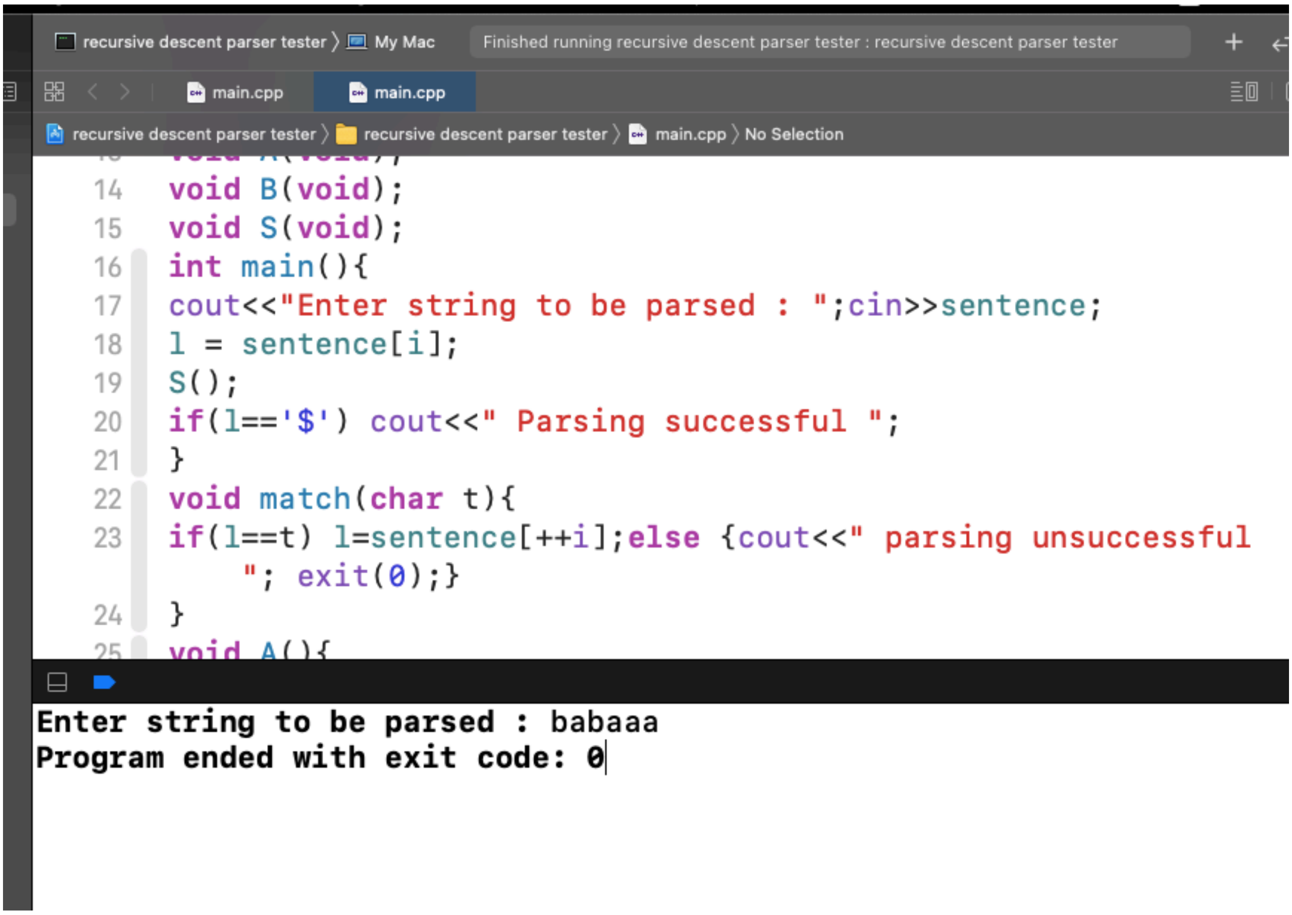
Input

```
S Ab Bb
A Aa a
B b
```

Output : Recursive descent parser for above grammar

```
#include <iostream>
using namespace std;
char l;
int i = 0;
string sentence;
void match(char);
void A(void);
void B(void);
void S(void);
int main(){
cout<<"Enter string to be parsed : ";cin>>sentence;
l = sentence[i];
S();
if(l=='$') cout<<" Parsing successful ";
}
void match(char t){
if(l==t) l=sentence[++i];else {cout<<" parsing unsuccessful "; exit(0);}
}
void A(){
if(l=='A'){
A();
match('a');
}
if(l=='a'){
match('a');
}
}
void B(){
if(l=='b'){
match('b');
}
}
void S(){
if(l=='A'){
A();
match('b');
}
if(l=='B'){
B();
match('b');
}
}
```

The above parser is executed


recursive descent parser tester 〉 My Mac
Finished running recursive descent parser tester : recursive descent parser tester
main.cpp
main.cpp
recursive descent parser tester 〉 recursive descent parser tester 〉 main.cpp 〉 No Selection

```
void A(void);
void B(void);
void S(void);
int main(){
cout<<"Enter string to be parsed : ";cin>>sentence;
l = sentence[i];
S();
if(l=='$') cout<<" Parsing successful ";
}
void match(char t){
if(l==t) l=sentence[++i];else {cout<<" parsing unsuccessful
    "; exit(0);}
}
void A(){
```

```
Enter string to be parsed : babaaa
Program ended with exit code: 0
```

Note : since the sentence does not belong to the grammar, it cannot be parsed and thus the program does not return anything.

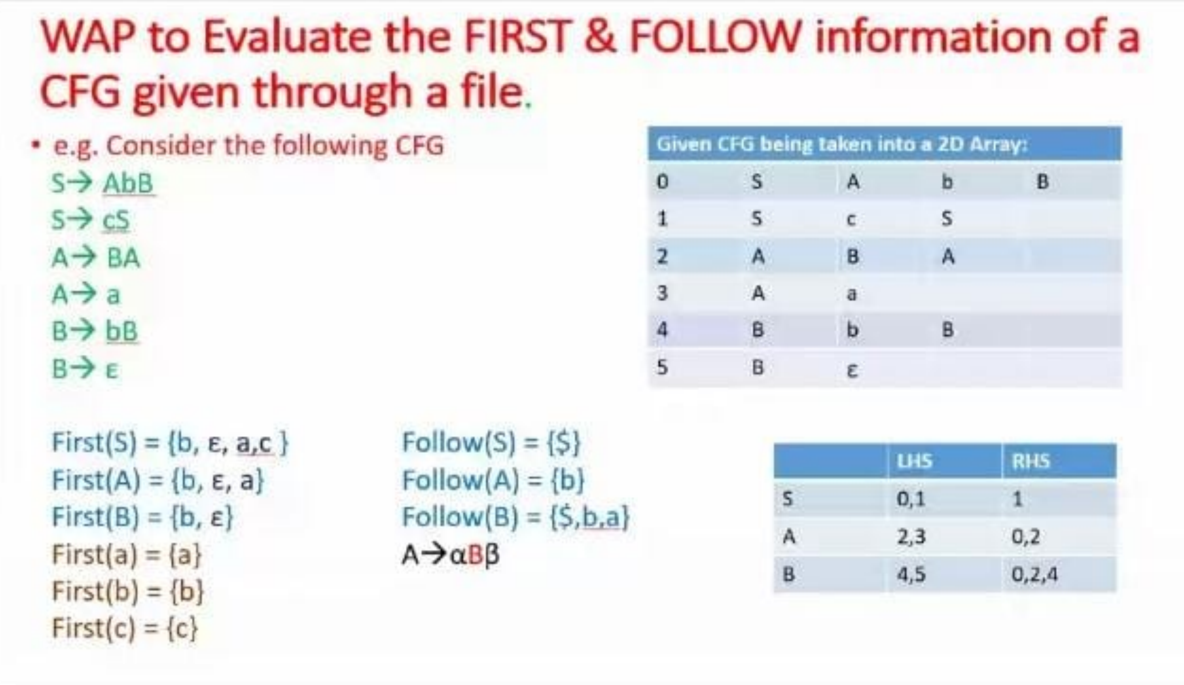
13 APRIL 2022

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD Lab program 6 :



WAP to Evaluate the FIRST & FOLLOW information of a CFG given through a file.

- e.g. Consider the following CFG

S→ AbB
S→ cS
A→ BA
A→ a
B→ bB
B→ ε

Given CFG being taken into a 2D Array:				
0	S	A	b	B
1	S	c	S	
2	A	B	A	
3	A	a		
4	B	b	B	
5	B	ε		

First(S) = {b, ε, a,c }
First(A) = {b, ε, a}
First(B) = {b, ε}
First(a) = {a}
First(b) = {b}
First(c) = {c}

Follow(S) = {$}
Follow(A) = {b}
Follow(B) = {$,b,a}
A→αBβ

	LHS	RHS
S	0,1	1
A	2,3	0,2
B	4,5	0,2,4

Input File format
1. All capital letters are considered variables
2. All other letters are considered terminals
3. epsilon is denoted by #
4. The RHS of first production is considered to be start variable
Example
S -> Ab|Bb
A -> Aa|a
B -> b|epsilon
Above grammar will be inputted as
S Ab Bb
A Aa a
B b #

C++ Implementation : first() and follow() sets of a grammar

```
//
//  main.cpp
//  first and follow
```

```
//
//  Created by YASH VINAYVANSHI on 12/04/22.
//

#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <stack>
#include <fstream>
#include <sstream>
#include <cstring>
using namespace std;

//global variables to avoid too much passing
map<char, vector<vector<char>>> productions;
set<char> variables;
set<char> terminals;
char startvar;
map<char, set<char>> firsts;
map<char, set<char>> follows;
set<char> this_first;
vector<vector<vector<char>>> parse_table;

//for grammar
void get_grammar(string);
void show_grammar();

//for firsts
void find_firsts();
bool dfs(char, char, char);
void show_firsts();

//for follows
void find_follows();
void show_follows();

int main(){
    string ipath;
    cout<<"Created by YASH VINAYVANSHI"<<endl;
    cout<<"Enter input file path : "; cin>>ipath;
    get_grammar(ipath);
    show_grammar();
    find_firsts();
    show_firsts();
    find_follows();
    show_follows();
    return 0;
}

void get_grammar(string ipath){
    vector<vector<string>> content;
    vector<string> row;
    string line, word;
    ifstream ifile;
    ifile.open(ipath);
    if(ifile.is_open()){
        while(getline(ifile, line)){
            row.clear();
            stringstream str(line);
            while(getline(str, word, ' ')) row.push_back(word);
            content.push_back(row);
        }
    }
    else{ cout<<"i/p File not opened\n"; exit(0);}
    ifile.close();

    int n = content.size();
    for(int i=0; i<n; i++){
        char RHS[2]; strcpy(RHS, content[i][0].c_str());
        variables.insert(RHS[0]);
        if(i==0){ startvar = RHS[0]; }
        int n1 = content[i].size();
        vector<vector<char>> prod;
        for(int j=1; j<n1; j++){
            int n2 = content[i][j].size();
            char ar[n2+1]; strcpy(ar, content[i][j].c_str());
            vector<char> rule;
            for(int k=0; k<n2; k++){rule.push_back(ar[k]);}
            prod.push_back(rule);
            for(auto it : rule){
                if(it>='A' && it<='Z') variables.insert(it);
                else{ if(it != '#') terminals.insert(it);}
            }

        }
        productions[RHS[0]] = prod;
    }
}
void show_grammar(){
    cout<<"Grammar is : "<<endl;
    for(auto it : variables){
        cout<<it<<" -> ";
        for(auto it1 : productions[it]){
            for(auto it2 : it1){
                cout<<it2;
            }
            cout<<" ";
        }
        cout<<endl;
    }
    cout<<"Start Variable is : "<<startvar<<endl;
}

void find_firsts(){
    for(auto it : productions){
        this_first.clear();
        dfs(it.first, it.first, it.first);
        for(auto it1 : this_first)
            firsts[it.first].insert(it1);
    }
}
```

```
bool dfs(char current, char origin, char last){
    /*
     find firsts
     1. if X is terminal -> First(X) = {X} [Base case]
     2. if X is non terminal & X -> Y1Y2...Yk
        Firsts of upto Yi contains epsilon
        2.1 i < k -> First(X) = First(Y1) U ... First(Yi)
        2.2 i == k -> First(X) = First(Y1) U ... First(Yk) U epsilon
     3. if X -> epsilon -> include epsilon in First(X)

     A -> BC
     B -> df | #
     C -> eg | #

            A           (A, A, A)
          /   \      case3 /  \
         B     C    (B, A, C) (C, A, C) (current = last)
        / \   / \
       df  # eg  #

     */
    bool takefurther = false;
    for(auto it : productions[current]){
        bool take = true;
        for(auto it1 : it){
            if(it1 == current) break;
            if(!take) break;
            //case1 : if terminal
            if(!(it1>='A'&&it1<='Z')&&it1!='#'){
                this_first.insert(it1);
                break;
            }
            //case 2 : production of V contains epsilon
            else if(it1 == '#'){
                //origin = current means cycle traced
                //i = last means reached end of production
                //if last variable contains epsilon, add epsilon
                if(origin == current||current == last)
                    this_first.insert(it1);
                takefurther = true;
                break;
            }
            //case 3
            else{
                take = dfs(it1, origin, it[it.size()-1]);
                takefurther |= take;
            }
        }
    }
    return takefurther;
}
void show_firsts(){
    cout<<"\nFirsts are : "<<endl;
    for(auto it : variables){
        cout<<it<<" : { ";
        for(auto it1 : firsts[it]){
            cout<<it1<<" ";
        }
        cout<<"}"<<endl;
    }
}

void find_follows(){
    int i;
    //satart variable has $ in its follow set
    follows[startvar].insert('$');
    terminals.insert('$');
    int count = 10;
    /*
     Rules
     1. Place $ in Follow(S)
     2. A -> alphaBbeta,
        2.1 if beta is a terminal
                Follow(B) = First(beta) = {beta}
        2.2 if First(beta) do not contiain epsilon
                Follow(B) = First(beta)
        2.3 if First(beta) contain epsilon
                Follow(B) = First(beta) U Follow(A) - epsilon
     */
    while(count--){
        for(auto q : productions){
            //for each production of variable q
            for(auto r : q.second){
                //for each char in production r of q
                for(i=0;i<r.size()-1;i++){
                    //if char is a variable ie B in A -> alphaBbeta
                    if(r[i]>='A'&&r[i]<='Z'){
                        //if next char is terminal ie case 2.1
                        if(!(r[i+1]>='A'&&r[i+1]<='Z'))
follows[r[i]].insert(r[i+1]);
                        else {
                            //temp contains first char of beta
                            char temp = r[i+1];
                            int j = i+1;
                            while(temp>='A'&&temp<='Z'){
                                //# first in sorted order in firsts set
                                //case 2.3
                                if(*firsts[temp].begin()=='#'){
                                    for(auto g : firsts[temp]){
                                        if(g=='#') continue;
                                        follows[r[i]].insert(g);
                                    }
                                    j++;
                                    if(j<r.size()){
                                        temp = r[j];
                                        if(!(temp>='A'&&temp<='Z')){
                                            follows[r[i]].insert(temp);
                                            break;
                                        }
                                    }
                                    else{
                                        for(auto g : follows[q.first])
follows[r[i]].insert(g);
                                        break;
                                    }
                                }
                                //case 2.2
                                else{
                                    for(auto g : firsts[temp]){
                                        follows[r[i]].insert(g);
                                    }
                                    break;
                                }
                            }
                        }
                    }
                }
                //if last char if production is variable
                //case 2.3
                if(r[r.size()-1]>='A'&&r[r.size()-1]<='Z'){
                    for(auto g : follows[q.first])
follows[r[i]].insert(g);
                }
            }
        }
    }
}
void show_follows(){
    cout<<"\nFollows are : "<<endl;
    for(auto it : variables){
        cout<<it<<" : { ";
        for(auto it1 : follows[it]){
            cout<<it1<<" ";
        }
        cout<<"}"<<endl;
    }
}
```

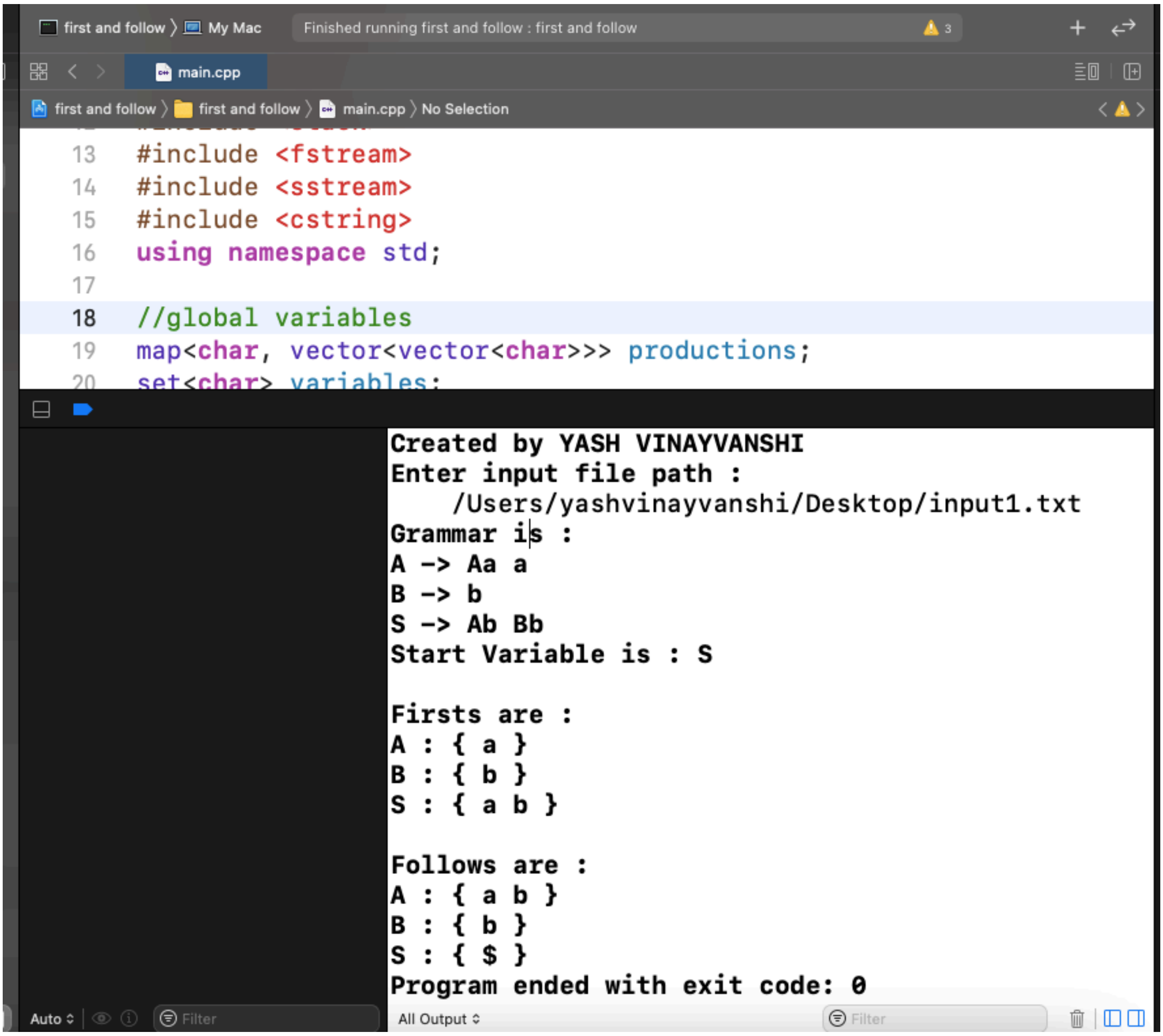
13 APRIL 2022

Run 1

Input

```
S Ab Bb
A Aa a
B b
```

Output



```
#include <fstream>
#include <sstream>
#include <cstring>
using namespace std;

//global variables
map<char, vector<vector<char>>> productions;
set<char> variables;
```

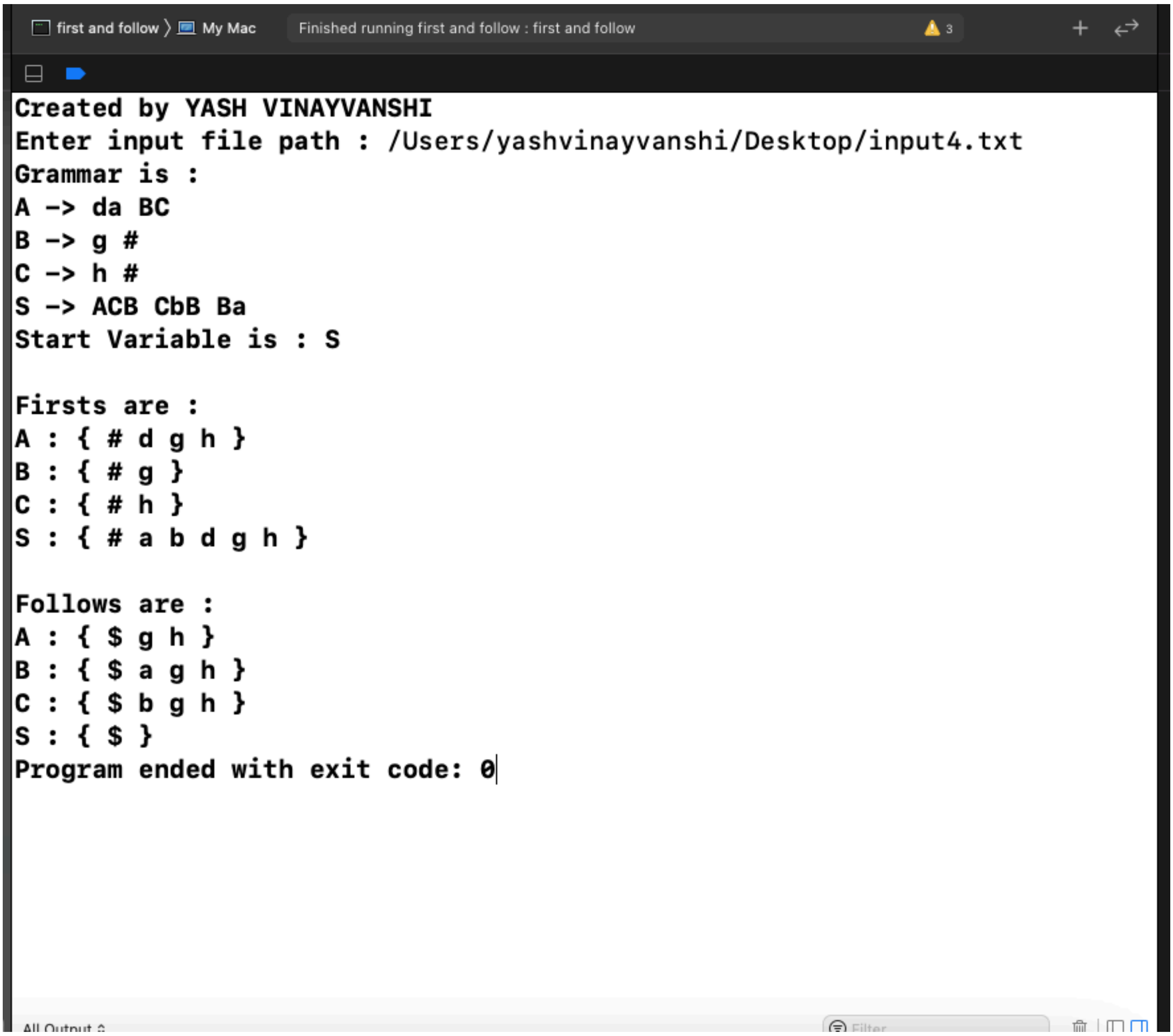
```
Created by YASH VINAYVANSHI
Enter input file path :
    /Users/yashvinayvanshi/Desktop/input1.txt
Grammar is :
A -> Aa a
B -> b
S -> Ab Bb
Start Variable is : S

Firsts are :
A : { a }
B : { b }
S : { a b }

Follows are :
A : { a b }
B : { b }
S : { $ }
Program ended with exit code: 0
```

Run 2

Input

```
S ACB CbB Ba
A da BC
B g #
C h #
```

Output



first and follow 〉 My Mac
Finished running first and follow : first and follow
3
All Output
Filter

```
Created by YASH VINAYVANSHI
Enter input file path : /Users/yashvinayvanshi/Desktop/input4.txt
Grammar is :
A -> da BC
B -> g #
C -> h #
S -> ACB CbB Ba
Start Variable is : S

Firsts are :
A : { # d g h }
B : { # g }
C : { # h }
S : { # a b d g h }

Follows are :
A : { $ g h }
B : { $ a g h }
C : { $ b g h }
S : { $ }
Program ended with exit code: 0
```

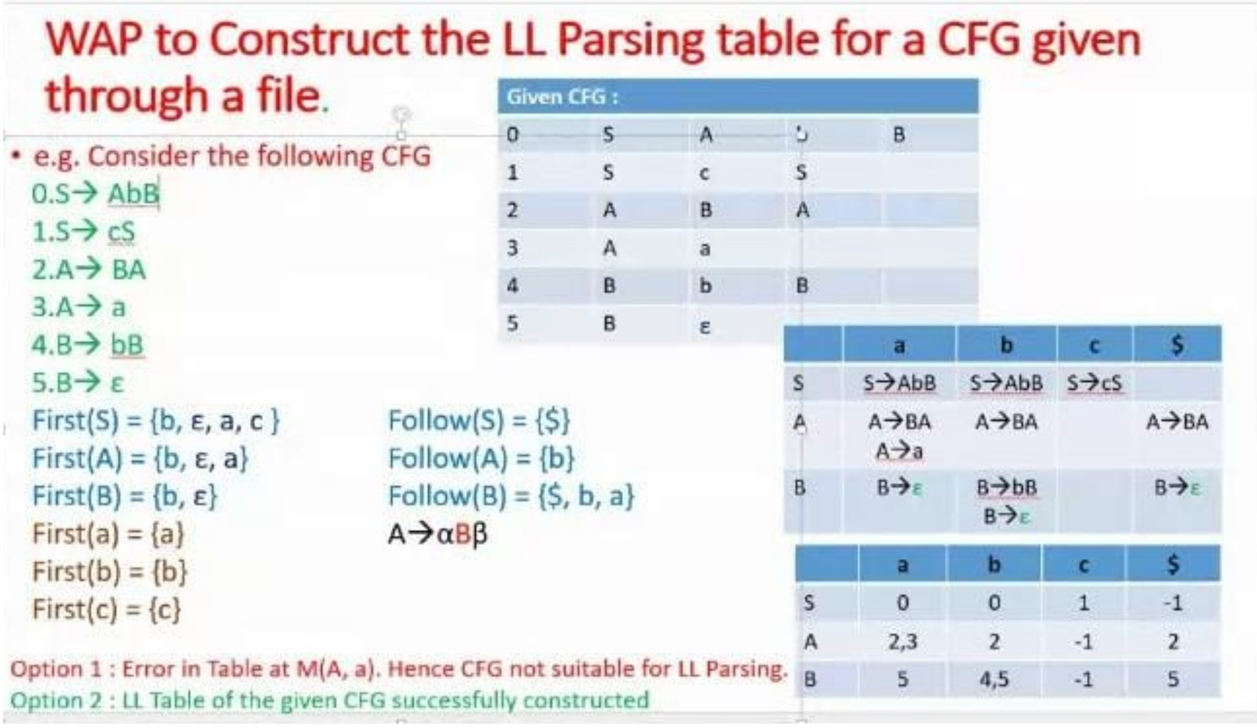
13 APRIL 2022

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD Lab program 7 :



WAP to Construct the LL Parsing table for a CFG given through a file.

- e.g. Consider the following CFG

0.S→ AbB
1.S→ cS
2.A→ BA
3.A→ a
4.B→ bB
5.B→ ε

First(S) = {b, ε, a, c }
First(A) = {b, ε, a}
First(B) = {b, ε}
First(a) = {a}
First(b) = {b}
First(c) = {c}

Follow(S) = {$}
Follow(A) = {b}
Follow(B) = {$, b, a}
A→αBβ

Given CFG :				
0	S	A	b	B
1	S	c	S	
2	A	B	A	
3	A	a		
4	B	b	B	
5	B	ε		

	a	b	c	$
S	S→AbB	S→AbB	S→cS	
A	A→BA A→a	A→BA		A→BA
B	B→ε	B→bB B→ε		B→ε

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

Option 1 : Error in Table at M(A, a). Hence CFG not suitable for LL Parsing.
Option 2 : LL Table of the given CFG successfully constructed

Input File format
1. All capital letters are considered variables
2. All other letters are considered terminals
3. epsilon is denoted by #
4. The RHS of first production is considered to be start variable
Example
S -> Ab|Bb
A -> Aa|a
B -> b|epsilon
Above grammar will be input as :
S Ab Bb
A Aa a
B b #

C++ Implementation : LL(1) parsing table of a grammar

```
//
//  main.cpp
//  build LL(1) parsing table of a grammar
//
//  Created by YASH VINAYVANSHI on 12/04/22.
//

#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <stack>
#include <fstream>
#include <sstream>
#include <cstring>
using namespace std;

//global variables to avoid too much passing
map<char, vector<vector<char>>> productions;
set<char> variables;
set<char> terminals;
char startvar;
map<char, set<char>> firsts;
map<char, set<char>> follows;
set<char> this_first;
//map<char, vector<vector<char>>> parse_table;
vector<vector<vector<char>>> parse_table;

//for grammar
void get_grammar(string);
void show_grammar();

//for firsts
void find_firsts();
bool dfs(char, char, char);
void show_firsts();

//for follows
void find_follows();
void show_follows();

//for parsing table
void build_LL1_parse_table();
void show_parse_table();

int main(){
    string ipath;
    cout<<"Created by YASH VINAYVANSHI"<<endl;
    cout<<"Enter input file path : "; cin>>ipath;
    get_grammar(ipath);
    show_grammar();
    find_firsts();
    show_firsts();
    find_follows();
    show_follows();
    build_LL1_parse_table();
    show_parse_table();
    return 0;
}

void get_grammar(string ipath){
    vector<vector<string>> content;
    vector<string> row;
    string line, word;
    ifstream ifile;
    ifile.open(ipath);
    if(ifile.is_open()){
        while(getline(ifile, line)){
            row.clear();
            stringstream str(line);
            while(getline(str, word, ' ')) row.push_back(word);
            content.push_back(row);
        }
    }
    else{ cout<<"i/p File not opened\n"; exit(0);}
    ifile.close();

    int n = content.size();
    for(int i=0; i<n; i++){
        char RHS[2]; strcpy(RHS, content[i][0].c_str());
        variables.insert(RHS[0]);
        if(i==0){ startvar = RHS[0]; }
        int n1 = content[i].size();
        vector<vector<char>> prod;
        for(int j=1; j<n1; j++){
            int n2 = content[i][j].size();
            char ar[n2+1]; strcpy(ar, content[i][j].c_str());
            vector<char> rule;
            for(int k=0; k<n2; k++){rule.push_back(ar[k]);}
            prod.push_back(rule);
            for(auto it : rule){
                if(it>='A' && it<='Z') variables.insert(it);
                else{ if(it != '#') terminals.insert(it);}
            }

        }
        productions[RHS[0]] = prod;
    }
}
void show_grammar(){
    cout<<"Grammar is : "<<endl;
    for(auto it : variables){
        cout<<it<<" -> ";
        for(auto it1 : productions[it]){
            for(auto it2 : it1){
                cout<<it2;
            }
            cout<<" ";
        }
        cout<<endl;
    }
    cout<<"Start Variable is : "<<startvar<<endl;
}
```

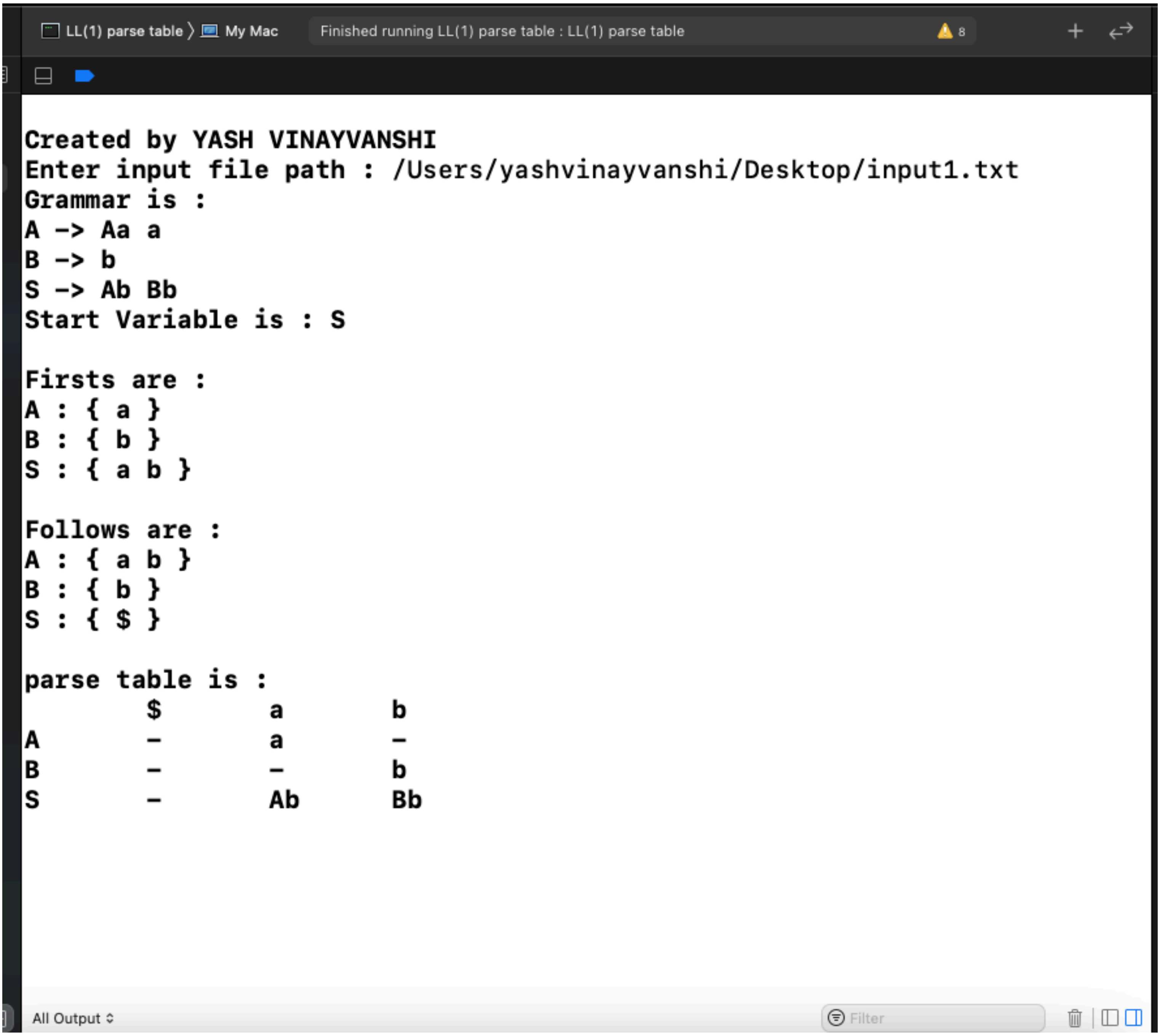
```
void find_firsts(){
    for(auto it : productions){
        this_first.clear();
        dfs(it.first, it.first, it.first);
        for(auto it1 : this_first)
            firsts[it.first].insert(it1);
    }
}
bool dfs(char current, char origin, char last){
    /*
     find firsts
     1. if X is terminal -> First(X) = {X} [Base case]
     2. if X is non terminal & X -> Y1Y2...Yk
        Firsts of upto Yi contains epsilon
        2.1 i < k -> First(X) = First(Y1) U ... First(Yi)
        2.2 i == k -> First(X) = First(Y1) U ... First(Yk) U epsilon
     3. if X -> epsilon -> include epsilon in First(X)

     A -> BC
     B -> df | #
     C -> eg | #

            A           (A, A, A)
          /   \      case3 /  \
         B     C    (B, A, C) (C, A, C) (current = last)
        / \   / \
       df  # eg  #

     */
    bool takefurther = false;
    for(auto it : productions[current]){
        bool take = true;
        for(auto it1 : it){
            if(it1 == current) break;
            if(!take) break;
            //case1 : if terminal
            if(!(it1>='A'&&it1<='Z')&&it1!='#'){
                this_first.insert(it1);
                break;
            }
            //case 2 : production of V contains epsilon
            else if(it1 == '#'){
                //origin = current means cycle traced
                //i = last means reached end of production
                //if last variable contains epsilon, add epsilon
                if(origin == current||current == last)
                    this_first.insert(it1);
                takefurther = true;
                break;
            }
            //case 3
            else{
                take = dfs(it1, origin, it[it.size()-1]);
                takefurther |= take;
            }
        }
    }
    return takefurther;
}
void show_firsts(){
    cout<<"\nFirsts are : "<<endl;
    for(auto it : variables){
        cout<<it<<" : { ";
        for(auto it1 : firsts[it]){
            cout<<it1<<" ";
        }
        cout<<"}"<<endl;
    }
}

void find_follows(){
    int i;
    //satart variable has $ in its follow set
    follows[startvar].insert('$');
    terminals.insert('$');
    int count = 10;
    /*
     Rules
     1. Place $ in Follow(S)
     2. A -> alphaBbeta,
        2.1 if beta is a terminal
                Follow(B) = First(beta) = {beta}
        2.2 if First(beta) do not contiain epsilon
                Follow(B) = First(beta)
        2.3 if First(beta) contain epsilon
                Follow(B) = First(beta) U Follow(A) - epsilon
     */
    while(count--){
        for(auto q : productions){
            //for each production of variable q
            for(auto r : q.second){
                //for each char in production r of q
                for(i=0;i<r.size()-1;i++){
                    //if char is a variable ie B in A -> alphaBbeta
                    if(r[i]>='A'&&r[i]<='Z'){
                        //if next char is terminal ie case 2.1
                        if(!(r[i+1]>='A'&&r[i+1]<='Z'))
follows[r[i]].insert(r[i+1]);
                        else {
                            //temp contains first char of beta
                            char temp = r[i+1];
                            int j = i+1;
                            while(temp>='A'&&temp<='Z'){
                                //# first in sorted order in firsts set
                                //case 2.3
                                if(*firsts[temp].begin()=='#'){
                                    for(auto g : firsts[temp]){
                                        if(g=='#') continue;
                                        follows[r[i]].insert(g);
                                    }
                                    j++;
                                    if(j<r.size()){
                                        temp = r[j];
                                        if(!(temp>='A'&&temp<='Z')){
                                            follows[r[i]].insert(temp);
                                            break;
                                        }
                                    }
                                    else{
                                        for(auto g : follows[q.first])
follows[r[i]].insert(g);

                                        break;
                                    }
                                }
                                //case 2.2
                                else{
                                    for(auto g : firsts[temp]){
                                        follows[r[i]].insert(g);
                                    }
                                    break;
                                }
                            }
                        }
                    }
                }
                //if last char if production is variable
                //case 2.3
                if(r[r.size()-1]>='A'&&r[r.size()-1]<='Z'){
                    for(auto g : follows[q.first])
follows[r[i]].insert(g);
                }
            }
        }
    }
}
void show_follows(){
    cout<<"\nFollows are : "<<endl;
    for(auto it : variables){
        cout<<it<<" : { ";
        for(auto it1 : follows[it]){
            cout<<it1<<" ";
        }
        cout<<"}"<<endl;
    }
}

void build_LL1_parse_table(){
    /*
     rule
     for each production of type A -> alpha
        put A -> alpha in columns of First(alpha)
     for each production of type A -> epsilon
        put A -> epsilon in Follow(A)
     */
    //#rows = #variables
    //#columns = #terminals
    terminals.erase('#');
    int terminals_count = terminals.size();
    int variables_count = variables.size();
    //cout<<variables_count<<endl;
    vector<char> temp; temp.push_back('-');
    vector<vector<char>> temp1;
    for(int i=0; i<terminals_count; i++)
        temp1.push_back(temp);
    for(int j=0; j<variables_count; j++)
        parse_table.push_back(temp1);
    //show_parse_table();
    //exit(0);
    set<char> to_place_in_cols;
    int varindex = 0;
    for(auto it : variables){
        for(auto it1 : productions[it]){
            to_place_in_cols.clear();
            //if first char of RHS is variable
            if(it1[0]>='A' && it1[0]<='Z'){
                for(auto it2 : firsts[it1[0]])
                    to_place_in_cols.insert(it2);
            }
            //if it is epsilon : cols = follow(LHS)
            else if(it1[0] == '#'){
                for(auto it2 : follows[it])
                    to_place_in_cols.insert(it2);
            }
            //if it is terminals : cols = terminal
            else
                to_place_in_cols.insert(it1[0]);
            //cout<<"variable"<<it<<endl;
            //for(auto it3 : to_place_in_cols) cout<<it3<<" ";
cout<<endl;
            int termindex = 0;
            for(auto it2 : terminals){
                //cout<<it2<<" "<<to_place_in_cols.count(it2)<<endl;
                if(to_place_in_cols.count(it2) > 0){
                    parse_table[varindex][termindex] = it1;
                }
                termindex++;
            }
        }
        varindex++;
    }
}
void show_parse_table(){
    int terminals_count = terminals.size();
    int variables_count = variables.size();
    cout<<"\nparse table is : "<<endl;
    for(auto it : terminals)
        cout<<"\t\t"<<it;
    cout<<endl;

    int varindex = 0;
    for(auto it : variables){
        cout<<it<<"\t\t";
        for(int j=0; j<terminals_count; j++){
            for(auto it1 : parse_table[varindex][j]){
                cout<<it1;
            }
            cout<<"\t\t";
        }
        cout<<endl;
        varindex++;
    }
}
```

Run 1

Input

```
S Ab Bb
A Aa a
B b
```

Output



LL(1) parse table 〉 My Mac
Finished running LL(1) parse table : LL(1) parse table
8
All Output ⇕
Filter

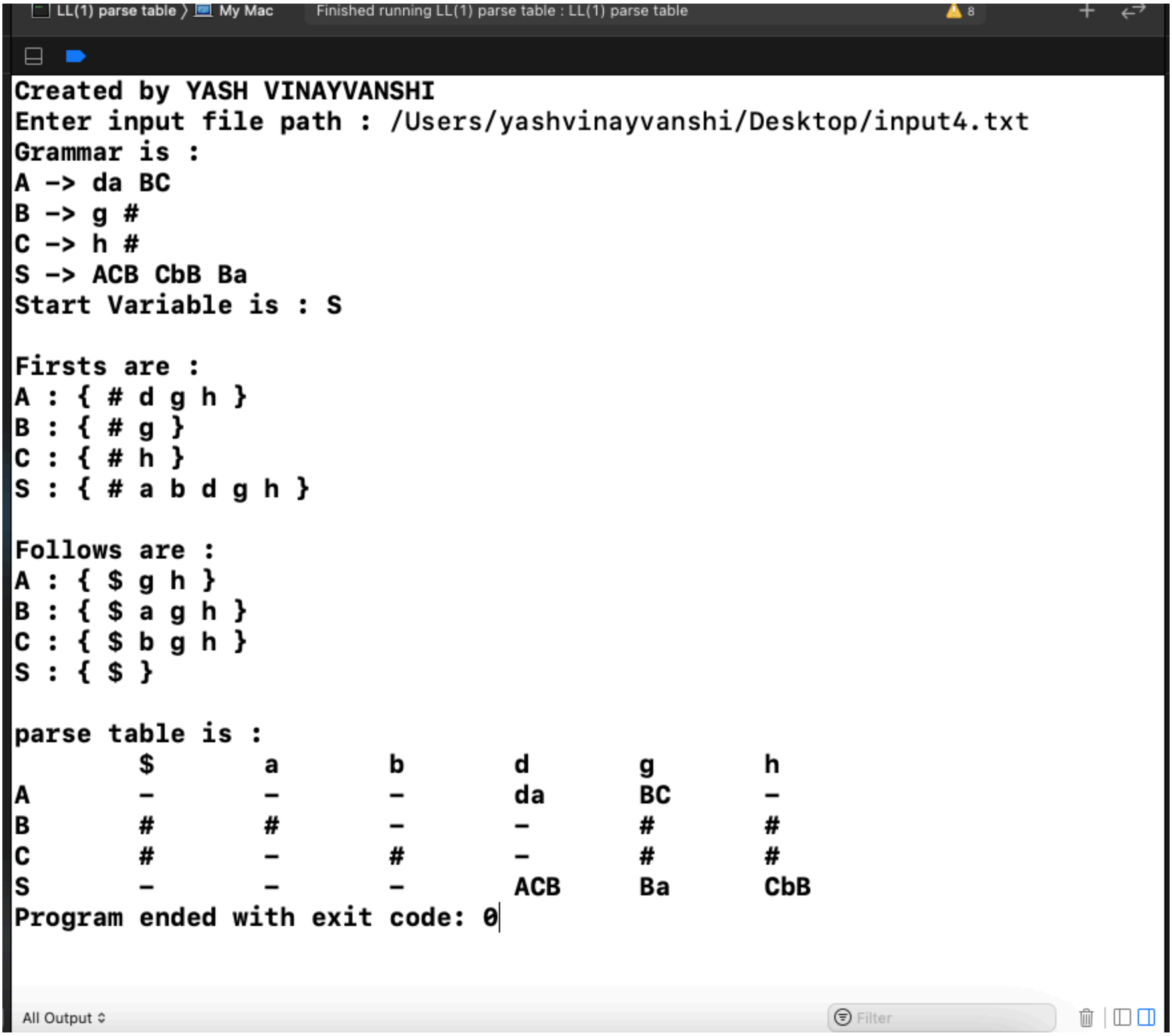
```
Created by YASH VINAYVANSHI
Enter input file path : /Users/yashvinayvanshi/Desktop/input1.txt
Grammar is :
A -> Aa a
B -> b
S -> Ab Bb
Start Variable is : S

Firsts are :
A : { a }
B : { b }
S : { a b }

Follows are :
A : { a b }
B : { b }
S : { $ }

parse table is :
        $       a       b
A       -       a       -
B       -       -       b
S       -       Ab      Bb
```

Run 2

Input

```
S ACB CbB Ba
A da BC
B g #
C h #
```

Output



LL(1) parse table 〉 My Mac
Finished running LL(1) parse table : LL(1) parse table
⚠ 8
All Output ⇕
Filter

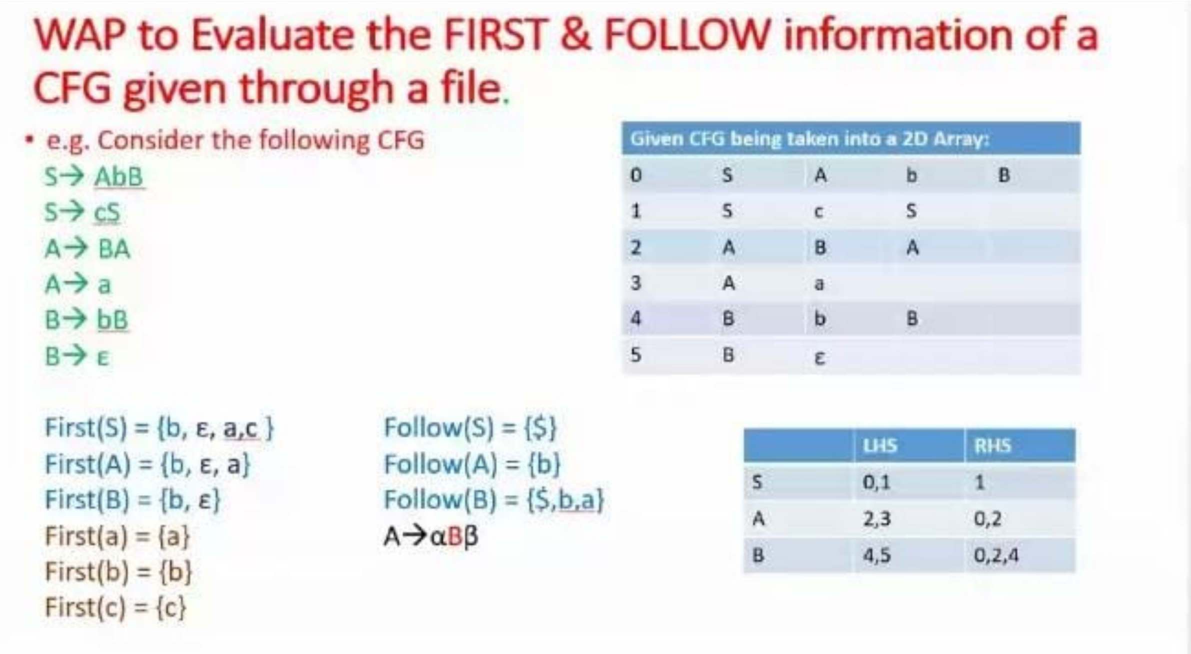
```
Created by YASH VINAYVANSHI
Enter input file path : /Users/yashvinayvanshi/Desktop/input4.txt
Grammar is :
A -> da BC
B -> g #
C -> h #
S -> ACB CbB Ba
Start Variable is : S

Firsts are :
A : { # d g h }
B : { # g }
C : { # h }
S : { # a b d g h }

Follows are :
A : { $ g h }
B : { $ a g h }
C : { $ b g h }
S : { $ }

parse table is :
        $       a       b       d       g       h
A       -       -       -       da      BC      -
B       #       #       -       -       #       #
C       #       -       #       -       #       #
S       -       -       -       ACB     Ba      CbB
Program ended with exit code: 0
```

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD Lab program 8



WAP to Evaluate the FIRST & FOLLOW information of a CFG given through a file.

- e.g. Consider the following CFG

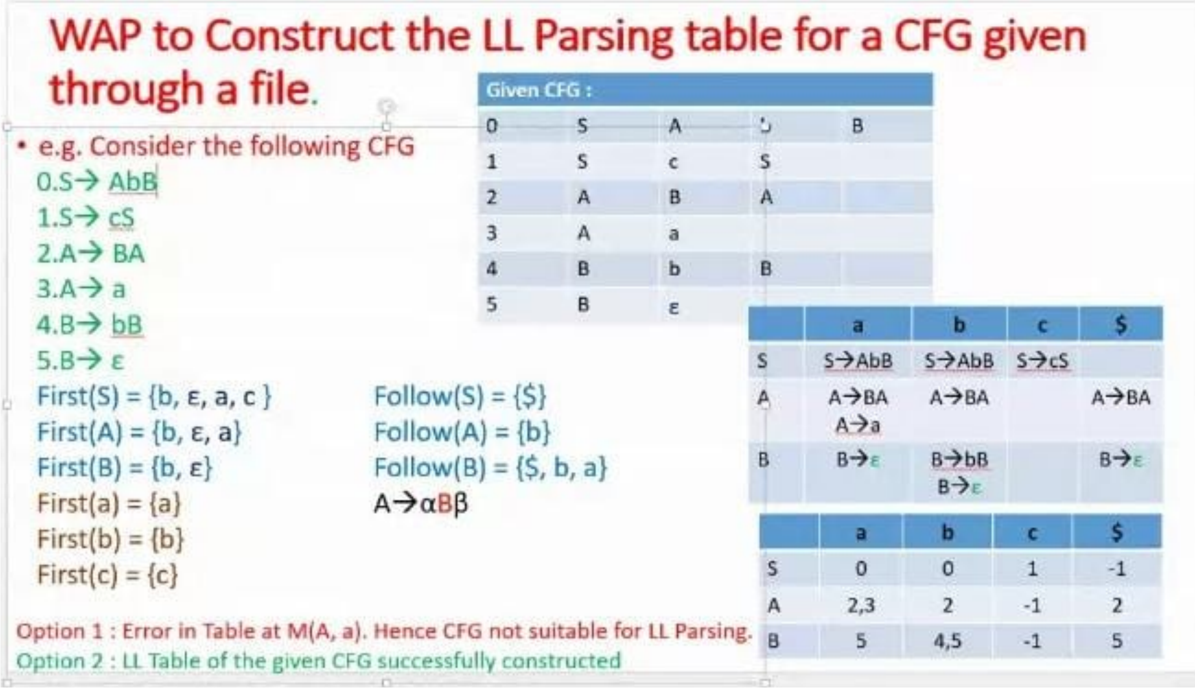
S→ AbB
S→ cS
A→ BA
A→ a
B→ bB
B→ ε

Given CFG being taken into a 2D Array:				
0	S	A	b	B
1	S	c	S	
2	A	B	A	
3	A	a		
4	B	b	B	
5	B	ε		

First(S) = {b, ε, a,c }
First(A) = {b, ε, a}
First(B) = {b, ε}
First(a) = {a}
First(b) = {b}
First(c) = {c}

Follow(S) = {$}
Follow(A) = {b}
Follow(B) = {$,b,a}
A→αBβ

	LHS	RHS
S	0,1	1
A	2,3	0,2
B	4,5	0,2,4



WAP to Construct the LL Parsing table for a CFG given through a file.

- e.g. Consider the following CFG

0.S→ AbB
1.S→ cS
2.A→ BA
3.A→ a
4.B→ bB
5.B→ ε

Given CFG :				
0	S	A	b	B
1	S	c	S	
2	A	B	A	
3	A	a		
4	B	b	B	
5	B	ε		

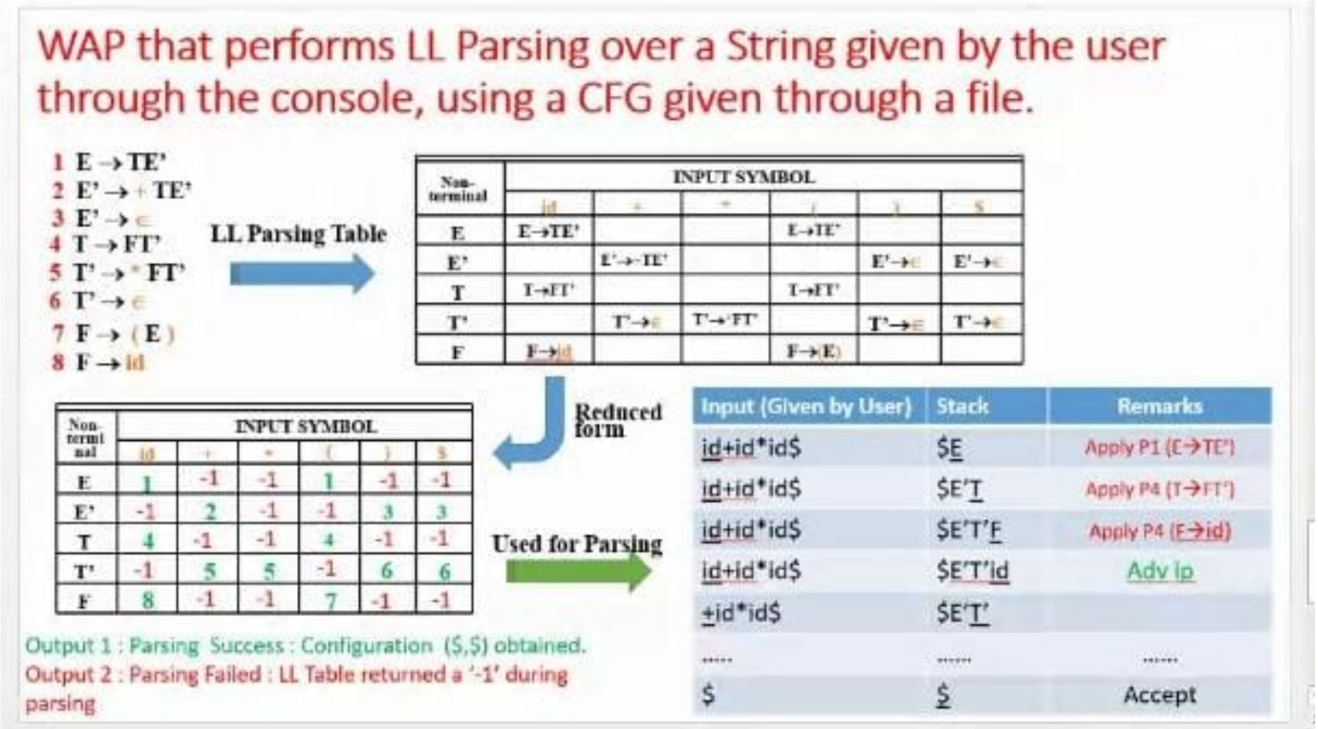
First(S) = {b, ε, a, c }
First(A) = {b, ε, a}
First(B) = {b, ε}
First(a) = {a}
First(b) = {b}
First(c) = {c}

Follow(S) = {$}
Follow(A) = {b}
Follow(B) = {$, b, a}
A→αBβ

	a	b	c	$
S	S→AbB	S→AbB	S→cS	
A	A→BA A→a	A→BA		A→BA
B	B→ε	B→bB B→ε		B→ε

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

Option 1 : Error in Table at M(A, a). Hence CFG not suitable for LL Parsing.
Option 2 : LL Table of the given CFG successfully constructed



WAP that performs LL Parsing over a String given by the user through the console, using a CFG given through a file.

1 E → TE'
2 E' → + TE'
3 E' → ∈
4 T → FT'
5 T' → * FT'
6 T' → ∈
7 F → (E)
8 F → id

LL Parsing Table

Non-terminal	INPUT SYMBOL					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→∈	E'→∈
T	T→FT'			T→FT'		
T'		T'→∈	T'→*FT'		T'→∈	T'→∈
F	F→id			F→(E)		

Reduced form

Non-terminal	INPUT SYMBOL					
	id	+	*	(	)	$
E	1	-1	-1	1	-1	-1
E'	-1	2	-1	-1	3	3
T	4	-1	-1	4	-1	-1
T'	-1	5	5	-1	6	6
F	8	-1	-1	7	-1	-1

Used for Parsing

Input (Given by User)	Stack	Remarks
id+id*id$	$E	Apply P1 (E→TE')
id+id*id$	$E'T	Apply P4 (T→FT')
id+id*id$	$E'T'F	Apply P4 (F→id)
id+id*id$	$E'T'id	Adv ip
+id*id$	$E'T'	
......	......	......
$	$	Accept

Output 1 : Parsing Success : Configuration ($,$) obtained.
Output 2 : Parsing Failed : LL Table returned a '-1' during parsing

The below program performs following tasks :

1. **Takes an (assumed to be unambiguous) context free grammar (CFG) in format specified below from a file.**
2. **Removes left recursion in given CFG.**
3. **Finds First and follow sets of each non terminal symbol.**
4. **Builds the LL(1) Parsing table**
5. **Takes as input from console a sentence and perform LL(1) parsing on it and outputs if the sentence is successfully parsed or not.**

Input File format
1. All capital letters are considered variables
2. All other letters are considered terminals
3. epsilon is denoted by #
4. The RHS of first production is considered to be start variable
Example
S -> Ab|Bb
A -> Aa|a
B -> b|epsilon
Above grammar will be inputted as
S Ab Bb
A Aa a
B b #

C++ Implementation : LL(1) Parser

```
//
//  main.cpp
//  LL(1) Parser
//
//  Created by YASH VINAYVANSHI on 12/04/22.
//
```

```
#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <stack>
#include <fstream>
#include <sstream>
#include <cstring>
using namespace std;

//global variables to avoid too much passing
map<char, vector<vector<char>>> productions;
set<char> variables;
set<char> terminals;
int variables_count;
int terminals_count;
char startvar;
bool is_suitable_for_LL1 = true;
map<char, set<char>> firsts;
map<char, set<char>> follows;
set<char> this_first;
//map<char, vector<vector<char>>> parse_table;
vector<vector<vector<char>>> parse_table;

//for grammar
void get_grammar(string);
void show_grammar();
void convert_to_LL1();

//for firsts
void find_firsts();
bool dfs(char, char, char);
void show_firsts();

//for follows
void find_follows();
void show_follows();

//for parsing table
void build_LL1_parse_table();
void show_parse_table();

//for parsing
void parse_sentence(vector<char>);

//others
void print_stack(stack<char>);
void print_vector(vector<char>);

int main(){
    string ipath;
    cout<<"Created by YASH VINAYVANSHI"<<endl;
    cout<<"Enter input file path : "; cin>>ipath;
    get_grammar(ipath);
    show_grammar();
    convert_to_LL1();
    find_firsts();
    show_firsts();
    find_follows();
    show_follows();
    build_LL1_parse_table();
    show_parse_table();

    vector<char> input;
    cout<<"\nEnter string to be parsed : ";
    string temp; cin>>temp;
    for(int i=0; i<temp.size(); i++)
        input.push_back(temp[i]);

    parse_sentence(input);
    return 0;
}

void get_grammar(string ipath){
    vector<vector<string>> content;
    vector<string> row;
    string line, word;
    ifstream ifile;
    ifile.open(ipath);
    if(ifile.is_open()){
        while(getline(ifile, line)){
            row.clear();
            stringstream str(line);
            while(getline(str, word, ' ')) row.push_back(word);
            content.push_back(row);
        }
    }
    else{ cout<<"i/p File not opened\n"; exit(0);}
    ifile.close();

    /*
    char startvar='S';
    set<char> variables;
    set<char> terminals;
    map<char, vector<vector<char>>> productions;
     */

    int n = content.size(); //cout<<n<<endl;
    for(int i=0; i<n; i++){
        char RHS[2]; strcpy(RHS, content[i][0].c_str());
        variables.insert(RHS[0]);
        if(i==0){ startvar = RHS[0]; }
        int n1 = content[i].size();
        vector<vector<char>> prod;
        for(int j=1; j<n1; j++){
            int n2 = content[i][j].size();
            char ar[n2+1]; strcpy(ar, content[i][j].c_str());
            vector<char> rule;
            for(int k=0; k<n2; k++){rule.push_back(ar[k]);}
            prod.push_back(rule);
            for(auto it : rule){
                if(it>='A' && it<='Z') variables.insert(it);
                else{ if(it != '#') terminals.insert(it);}
            }

        }
        productions[RHS[0]] = prod;
    }
    //int variables_count = variables.size();
    //int terminals_count = terminals.size();
}
void show_grammar(){
    cout<<"Grammar is : "<<endl;
    for(auto it : variables){
        cout<<it<<" -> ";
        for(auto it1 : productions[it]){
            for(auto it2 : it1){
                cout<<it2;
            }
            cout<<" ";
        }
        cout<<endl;
    }
    cout<<"Start Variable is : "<<startvar<<endl;
}
void convert_to_LL1(){
    /*
     Removing left recursion
     A -> Aalpha | beta
     replaced with
     A -> betaA'
     A'-> alphaA' | epsilon

     Let factoring not yet applied

     */
    int flag2 = 0;
    vector<char> del;
    map<char,vector<vector<char>>> add;
    vector<int> viz(100,0);
    for(auto q : productions){
        int one = 0;
        char c;
        for(auto r : q.second){
            //A -> Aalpha check
            if(q.first == r[0]){
                one = 1;
                flag2 = 1;
                //put A -> Aalpha in deletion row
                del.push_back(q.first);
                c = 'A';
                //find a new free character to be assigned as A'
                while(productions.count(c)||viz[c-'A']) c++;
                vector<char> temp;
                //push alpha leave behind A
                for(int i=1;i<r.size();i++)
                    temp.push_back(r[i]);
                temp.push_back(c);
                //A' -> alphaA' | epsilon added
                add[c].push_back(temp);
                add[c].push_back({'#'});
                //variables.insert(c);
            }
        }
        if(one){
            for(auto r : q.second){
                if(q.first != r[0]){
                    vector<char> temp;
                    for(int i=0;i<r.size();i++)
                        temp.push_back(r[i]);
                    temp.push_back(c);
                    add[q.first].push_back(temp);
                }
            }
            //variable marked to be occupied
            viz[c-'A'] = 1;
        }
    }
    for(auto q : del) productions.erase(q);
    for(auto q : add) productions[q.first] = q.second;
    if(flag2){
        cout<<"\nGiven CFG is not suitable for LL1"<<endl;
        cout<<"Converted new Grammar is :"<<endl;
        for(auto q : productions){
            string ans = "";
            ans+=q.first;
            ans += " -> ";
            for(auto r : q.second){
                for(auto s : r) ans+=s;
                ans+=' ';
            }
            ans.pop_back();
            cout<<ans<<'\n';
        }
    }
    else{
        cout<<"Given CFG is suitable for LL1"<<'\n';
        //is_suitable_for_LL1 = false;
    }

    //recalculate variables and terminals
    variables.clear();
    terminals.clear();
    for(auto q : productions){
        variables.insert(q.first);
        for(auto r : q.second){
            for(auto s : r){
                //for variables which only occur in LHS
                if(s>='A' && s<='Z') variables.insert(s);
                else{ if(s != '#') terminals.insert(s);}
            }
        }
    }
}
```

```
void find_firsts(){
    for(auto it : productions){
        this_first.clear();
        dfs(it.first, it.first, it.first);
        for(auto it1 : this_first)
            firsts[it.first].insert(it1);
    }
}
bool dfs(char current, char origin, char last){
    /*
     find firsts
     1. if X is terminal -> First(X) = {X} [Base case]
     2. if X is non terminal & X -> Y1Y2...Yk
        Firsts of upto Yi contains epsilon
        2.1 i < k -> First(X) = First(Y1) U ... First(Yi)
        2.2 i == k -> First(X) = First(Y1) U ... First(Yk) U epsilon
     3. if X -> epsilon -> include epsilon in First(X)

     A -> BC
     B -> df | #
     C -> eg | #

            A           (A, A, A)
          /   \      case3 /  \
         B     C    (B, A, C) (C, A, C) (current = last)
        / \   / \
       df  # eg   #

     */
    bool takefurther = false;
    for(auto it : productions[current]){
        bool take = true;
        for(auto it1 : it){
            if(it1 == current) break;
            if(!take) break;
            //case1 : if terminal
            if(!(it1>='A'&&it1<='Z')&&it1!='#'){
                this_first.insert(it1);
                break;
            }
            //case 2 : production of V contains epsilon
            else if(it1 == '#'){
                //origin = current means cycle traced
                //i = last means reached end of production
                //if last variable contains epsilon, add epsilon
                if(origin == current||current == last)
                    this_first.insert(it1);
                takefurther = true;
                break;
            }
            //case 3
            else{
                take = dfs(it1, origin, it[it.size()-1]);
                takefurther |= take;
            }
        }
    }
    return takefurther;
}
void show_firsts(){
    cout<<"\nFirsts are : "<<endl;
    for(auto it : variables){
        cout<<it<<" : { ";
        for(auto it1 : firsts[it]){
            cout<<it1<<" ";
        }
        cout<<"}"<<endl;
    }
}

void find_follows(){
    int i;
    //satart variable has $ in its follow set
    follows[startvar].insert('$');
    terminals.insert('$');
    int count = 10;
    /*
     Rules
     1. Place $ in Follow(S)
     2. A -> alphaBbeta,
        2.1 if beta is a terminal
                Follow(B) = First(beta) = {beta}
        2.2 if First(beta) do not contiain epsilon
                Follow(B) = First(beta)
        2.3 if First(beta) contain epsilon
                Follow(B) = First(beta) U Follow(A) - epsilon
     */
    while(count--){
        for(auto q : productions){
            //for each production of variable q
            for(auto r : q.second){
                //for each char in production r of q
                for(i=0;i<r.size()-1;i++){
                    //if char is a variable ie B in A -> alphaBbeta
                    if(r[i]>='A'&&r[i]<='Z'){
                        //if next char is terminal ie case 2.1
                        if(!(r[i+1]>='A'&&r[i+1]<='Z'))
follows[r[i]].insert(r[i+1]);
                        else {
                            //temp contains first char of beta
                            char temp = r[i+1];
                            int j = i+1;
                            while(temp>='A'&&temp<='Z'){
                                //# first in sorted order in firsts set
                                //case 2.3
                                if(*firsts[temp].begin()=='#'){
                                    for(auto g : firsts[temp]){
                                        if(g=='#') continue;
                                        follows[r[i]].insert(g);
                                    }
                                    j++;
                                    if(j<r.size()){
                                        temp = r[j];
                                        if(!(temp>='A'&&temp<='Z')){
                                            follows[r[i]].insert(temp);
                                            break;
                                        }
                                    }
                                    else{
                                        for(auto g : follows[q.first])
follows[r[i]].insert(g);

                                        break;
                                    }
                                }
                                //case 2.2
                                else{
                                    for(auto g : firsts[temp]){
                                        follows[r[i]].insert(g);
                                    }
                                    break;
                                }
                            }
                        }
                    }
                }
                //if last char if production is variable
                //case 2.3
                if(r[r.size()-1]>='A'&&r[r.size()-1]<='Z'){
                    for(auto g : follows[q.first])
follows[r[i]].insert(g);
                }
            }
        }
    }
}
void show_follows(){
    cout<<"\nFollows are : "<<endl;
    for(auto it : variables){
        cout<<it<<" : { ";
        for(auto it1 : follows[it]){
            cout<<it1<<" ";
        }
        cout<<"}"<<endl;
    }
}

void build_LL1_parse_table(){
    /*
     rule
     for each production of type A -> alpha
        put A -> alpha in columns of First(alpha)
     for each production of type A -> epsilon
        put A -> epsilon in Follow(A)
     */
    //#rows = #variables
    //#columns = #terminals
    terminals.erase('#');
    int terminals_count = terminals.size();
    int variables_count = variables.size();
    //cout<<variables_count<<endl;
    vector<char> temp; temp.push_back('-');
    vector<vector<char>> temp1;
    for(int i=0; i<terminals_count; i++)
        temp1.push_back(temp);
    for(int j=0; j<variables_count; j++)
        parse_table.push_back(temp1);
    //show_parse_table();
    //exit(0);
    set<char> to_place_in_cols;
    int varindex = 0;
    for(auto it : variables){
        for(auto it1 : productions[it]){
            to_place_in_cols.clear();
            //if first char of RHS is variable
            if(it1[0]>='A' && it1[0]<='Z'){
                for(auto it2 : firsts[it1[0]])
                    to_place_in_cols.insert(it2);
            }
            //if it is epsilon : cols = follow(LHS)
            else if(it1[0] == '#'){
                for(auto it2 : follows[it])
                    to_place_in_cols.insert(it2);
            }
            //if it is terminals : cols = terminal
            else
                to_place_in_cols.insert(it1[0]);
            //cout<<"variable"<<it<<endl;
            //for(auto it3 : to_place_in_cols) cout<<it3<<" ";
cout<<endl;
            int termindex = 0;
            for(auto it2 : terminals){
                //cout<<it2<<" "<<to_place_in_cols.count(it2)<<endl;
                if(to_place_in_cols.count(it2) > 0){
                    parse_table[varindex][termindex] = it1;
                }
                termindex++;
            }
        }
        varindex++;
    }
}
void show_parse_table(){
    int terminals_count = terminals.size();
    int variables_count = variables.size();
    cout<<"\nparse table is : "<<endl;
    for(auto it : terminals)
        cout<<"\t\t"<<it;
    cout<<endl;

    int varindex = 0;
    for(auto it : variables){
        cout<<it<<"\t\t";
        for(int j=0; j<terminals_count; j++){
            for(auto it1 : parse_table[varindex][j]){
                cout<<it1;
            }
            cout<<"\t\t";
        }
        cout<<endl;
        varindex++;
    }
}

void parse_sentence(vector<char> input){
    cout<<"stack\t\t   input\t\t   action"<<endl;
    stack<char> Stack;
    Stack.push('$');
    Stack.push(startvar);

    vector<char> action;
    print_stack(Stack);
    print_vector(input);
    cout<<endl;

    //int currentsym = 0;
    while(!(Stack.top() == '$')){
        char A = Stack.top();
        char r = input[0/*currentsym++*/];
        action.clear();

        //cout<<"A = "<<A<<", r = "<<r<<endl;
        //stack.top() is a terminal or input finished
        if(!(A>='A' && A<='Z') || A =='$'){
            //if it matches with current input symbol : match it
            if(A == r){
                Stack.pop();
                input.erase(input.begin());
                //print_vector(input); cout<<endl;
                action = {'p', 'o', 'p'};
            }
            //if doesnt match return error
            else{
                cout<<"string not parsable"<<endl;
                exit(0);
            }
        }
        //if stack top is variable & input symbol is terminal
        else if(A>='A' && A<='Z'){
            //find variable index in variable set
            int varindex = 0;
            for(auto it : variables){
                if(it == A) break;
                varindex++;
            }
            //find terminal index in terminal set
            int terindex = 0;
            for(auto it : terminals){
                if(it == r) break;
                terindex++;
            }
            //if a non empty entry found in parse table
            if(parse_table[varindex][terindex][0] != '-'){
                Stack.pop();
                vector<char> prod = parse_table[varindex][terindex];

                //insert RHS in reverse order
                stack<char> rev;
                for(auto it : prod) rev.push(it);
                while(!rev.empty()){Stack.push(rev.top()); rev.pop();}
                //e is not added to stack
                if(Stack.top()=='#') Stack.pop();

                action = {A, ' ', '-', '>', ' '};
                for(auto it : prod) action.push_back(it);
                //print_stack(Stack); cout<<endl;
            }
            //if no corresponding production return error
            else{
                cout<<"string not parsable"<<endl;
                exit(0);
            }
        }
        print_stack(Stack);
        print_vector(input);
        print_vector(action);
        cout<<endl;
    }
    //input should be exhausted except of end marker
    if(input[0] != '$'){
        cout<<"string not parsable"<<endl;
        exit(0);
    }
    cout<<"string successfully parsed"<<endl;
}
void print_stack(stack<char> S){
    stack<char> rev;
    int n = S.size();
    while(!S.empty()){ rev.push(S.top()); S.pop(); }
    while(!rev.empty()){ cout<<rev.top(); rev.pop();}
    for(int i=0;  i<15-n; i++)
        cout<<" ";
}
void print_vector(vector<char> V){
    int n = V.size();
    for(auto it : V)
        cout<<it;
    for(int i=0;  i<15-n; i++)
        cout<<" ";
}
```

Run 1

Input

```
S Ab Bb
A Aa a
B b

sentence : aab$
```

Output

```
Created by YASH VINAYVANSHI
Enter input file path : /Users/yashvinayvanshi/Desktop/input1.txt
Grammar is :
A -> Aa a
B -> b
S -> Ab Bb
Start Variable is : S

Given CFG is not suitable for LL1
Converted new Grammar is :
A -> aC
B -> b
C -> aC #
S -> Ab Bb

Firsts are :
A : { a }
B : { b }
C : { # a }
S : { a b }

Follows are :
A : { b }
B : { b }
C : { b }
S : { $ }

parse table is :
       $             a             b
A      -             aC            -
B      -             -             b
C      -             aC            #
S      -             Ab            Bb

Enter string to be parsed : aab$
stack     input                 action
$S             aab$
$bA            aab$           S -> Ab
$bCa           aab$           A -> aC
$bC            ab$            pop
$bCa           ab$            C -> aC
$bC            b$             pop
$b             b$             C -> #
$              $              pop
string successfully parsed
Program ended with exit code: 0
```

Run 2

Input

```
S Ab Bb
A Aa a
B b

sentence : aabb$
```

Output : Recursive descent parser for above grammar

```
Created by YASH VINAYVANSHI
Enter input file path : /Users/yashvinayvanshi/Desktop/input1.txt
Grammar is :
A -> Aa a
B -> b
S -> Ab Bb
Start Variable is : S

Given CFG is not suitable for LL1
Converted new Grammar is :
A -> aC
B -> b
C -> aC #
S -> Ab Bb

Firsts are :
A : { a }
B : { b }
C : { # a }
S : { a b }

Follows are :
A : { b }
B : { b }
C : { b }
S : { $ }

parse table is :
       $             a             b
A      -             aC            -
B      -             -             b
C      -             aC            #
S      -             Ab            Bb

Enter string to be parsed : aabb$
stack     input                action
$S             aabb$
$bA            aabb$          S -> Ab
$bCa           aabb$          A -> aC
$bC            abb$           pop
$bCa           abb$           C -> aC
$bC            bb$            pop
$b             bb$            C -> #
$              b$             pop
string not parsable
Program ended with exit code: 0
```

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

cd lab 9 : WAP that performs LR Parsing over a String given by the user through the console, using an LR Parsing Table (for a CFG) given through a file.

Input File format

1. All capital letters are considered variables
2. All other letters are considered terminals
3. epsilon is denoted by #
4. OR productions are space separated
5. The SLR parsing table is written after 1 blank line from grammar
6. The first line of SLR table are space separated columns
7. next n rows contain space separated entries for each n state starting from 0.
8. shift entries are represented by Si where i is the state from 0 to n-1
9. reduce entries are represented by Ri where i is the production used to reduces from 1 to m(no. of productions)
10. Accept entry written as A
11. No entry written as -
12. Goto entries written as numbers.

The grammar

```
E -> E+T | T
T -> T*F | F
F -> (E) | i
```

and its corresponding SLR table are laid in file as

Example:

```
E E+T T
T T*F F
F (E) i

id + * ( ) $ E T F
0 S5 - - S4 - - 1 2 3
1 - S6 - - - A - - -
2 - R2 S7 - R2 R2 - - -
3 - R4 R4 - R4 R4 - - -
4 S5 - - S4 - - 8 2 3
5 - R6 R6 - R6 R6 - - -
6 S5 - - S4 - - - 9 3
7 S5 - - S4 - - - - 10
8 - S6 - - S11 - - - -
9 - R1 S7 - R1 R1 - - -
10 - R3 R3 - R3 R3 - - -
```

11 – R5 R5 – R5 R5 – – –

C++ Implementation : SLR(1) Parser

```
//
//  main.cpp
//  SLR parse
//
//  Created by YASH VINAYVANSHI on 26/04/22.
//

#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <stack>
#include <fstream>
#include <sstream>
#include <cstring>
using namespace std;

vector<vector<char>> productions;
vector<char> column;
map<char, int> col_index;
int states = 0;
vector<vector<int>> slr_table;

void get_grammar(string);
void show_grammar();
void show_slr_table();
void slr_parse(vector<char>);
void print_input_vector(vector<char>);
void print_state_stack(stack<int>);
void print_symbol_stack(stack<char>);

int main(int argc, const char * argv[]) {
    string ipath;
    cout<<"Created by YASH VINAYVANSHI"<<endl;
    cout<<"Enter input file path : "; cin>>ipath;
    get_grammar(ipath);
    show_grammar();
    show_slr_table();

    char temp = ' ';
    vector<char> input;
    cout<<"\nEnter input : "<<endl;
    while(temp != '$'){
        cin>>temp;
        input.push_back(temp);
    }
    slr_parse(input);
}

void get_grammar(string ipath){
    vector<vector<string>> content;
    vector<string> row;
    string line, word;
    ifstream ifile;
    ifile.open(ipath);
```

```
if(ifile.is_open()){
    while(getline(ifile, line)){
        row.clear();
        stringstream str(line);
        while(getline(str, word, ' ')) row.push_back(word);
        content.push_back(row);
    }
}
else{ cout<<"i/p File not opened\n"; exit(0);}
ifile.close();

/*
char startvar='S';
set<char> variables;
set<char> terminals;
map<char, vector<vector<char>>> productions;
 */

int size = content.size(); //cout<<n<<endl;
int n = 0;
for(int i=0; i<size; i++){
    if(content[i].size() == 0) break;
    n++;
}

for(int i=0; i<n; i++){
    char RHS[2];
    strcpy(RHS, content[i][0].c_str());
    int n1 = content[i].size();
    vector<vector<char>> prod;
    for(int j=1; j<n1; j++){
        int n2 = content[i][j].size();
        char ar[n2+1];
        strcpy(ar, content[i][j].c_str());
        vector<char> rule;
        rule.push_back(RHS[0]);
        rule.push_back('-');
        rule.push_back('>');
        for(int k=0; k<n2; k++){rule.push_back(ar[k]);}
        productions.push_back(rule);
    }
}

//int count = 1;
char C[2];
for(auto it : content[n+1]){
    strcpy(C, it.c_str());
    column.push_back(C[0]);
    //column[C[0]] = count;
    //count++;
}

for(int i=n+2; i<size; i++){
    int n1 = content[i].size();
    states++;
    vector<int> rule;
    for(int j=1; j<n1; j++){
        int n2 = content[i][j].size();
        if(content[i][j][0] == 'S')
            rule.push_back((int)(content[i][j][1]-48));
            else if(content[i][j][0] == 'R')
                rule.push_back(-1*(int)(content[i][j][1]-48));
            else if(content[i][j][0] == 'A')
                rule.push_back(100);
            else if(content[i][j][0] >= '0' && content[i][j][0] <= '9')
                rule.push_back(stoi(content[i][j]));
            else
                rule.push_back(0);
        }
        slr_table.push_back(rule);
    }

    int count = 0;
    for(auto it : column){
        col_index[it]=count;
        count++;
    }
}
void show_grammar(){
    int count = 1;
    for(auto it : productions){
        cout<<count<<". ";
        for(auto it1 : it){
            cout<<it1;
        }
        cout<<endl;
        count++;
    }
}
void show_slr_table(){
    int count = 0;
    cout<<"\nParsing table is : "<<endl;
    cout<<"\t";
    for(auto it : column)
        cout<<it<<"\t";
    cout<<endl;
    for(auto it : slr_table){
        cout<<count<<"\t";
        for(auto it1 : it){
            cout<<it1<<"\t";
        }
        cout<<endl;
        count++;
    }
}
void slr_parse(vector<char> input){
    //algorithm
    /*
     let a be first symbol of w$ (input)
     while(1){
        let s be the state on top of stack
        if(ACTION[s, a] = shift t){
            push t onto state stack
            let a be next nymbol of input
        }
        else if(ACTION[s, a] = reduce A -> beta){
            pop |beta| symbols of the stack
            let state t now be on top of the stack
            push GOTO[t, A] onto the stack
            output the production A -> beta
        }
        else if(ACTION[s, a] = Accept) break;
        else call error-recovery routine;
     }
    */
    cout<<"\nstate     symbol    input     action"<<endl;
    int ip_ptr = 0;
    stack<int> state;
    stack<char> symbol;
    char a = input[ip_ptr];
    state.push(0);
    while(1){
        print_state_stack(state);
        print_symbol_stack(symbol);
        print_input_vector(input);
        string action;
        int s = state.top();
        int todo = slr_table[s][col_index[a]];
        //cout<<s<<" "<<a<<" "<<todo;
        if(todo > 0 && todo != 100){//ie shift
            state.push(todo);
            input.erase(input.begin());
            symbol.push(a);
            a = input[0];
            action = "shift";
        }
        else if(todo < 0){
            todo = -1*todo;
            vector<char> reduce = productions[todo-1];
            int len = reduce.size();
            len = len - 3;
            for(int i=0; i<len; i++){
                state.pop();
                symbol.pop();
            }
            symbol.push(reduce[0]);
            int t = state.top();
            state.push(slr_table[t][col_index[reduce[0]]]);

            action = "reduce by ";
            for(auto it : reduce) action.push_back(it);
        }
        else if(todo == 100){
            cout<<"accept"<<endl;
            cout<<"parsing successful"<<endl;
            break;
        }
        else{
            cout<<"Error"<<endl;
            cout<<"sentence not parsable"<<endl;
            break;
        }
        cout<<action;
        cout<<endl;
    }
}
void print_input_vector(vector<char> ip){
    for(auto it : ip)
        cout<<it;
    int n = 10-ip.size();
    if(n>0) for(int i=0; i<n; i++) cout<<" ";
}
void print_state_stack(stack<int> st){
    vector<int> temp;
    while(!st.empty()){
        temp.push_back(st.top());
        st.pop();
    }
    int s = temp.size();
    for(int i=s-1; i>=0; i--) cout<<temp[i]<<" ";
    int n = 15-2*temp.size();
    if(n>0) for(int i=0; i<n; i++) cout<<" ";
}
void print_symbol_stack(stack<char> st){
    vector<char> temp;
    while(!st.empty()){
        temp.push_back(st.top());
        st.pop();
    }
    int s = temp.size();
    for(int i=s-1; i>=0; i--) cout<<temp[i]<<" ";
    int n = 15-2*temp.size();
    for(int i=0; i<n; i++) cout<<" ";
}
```

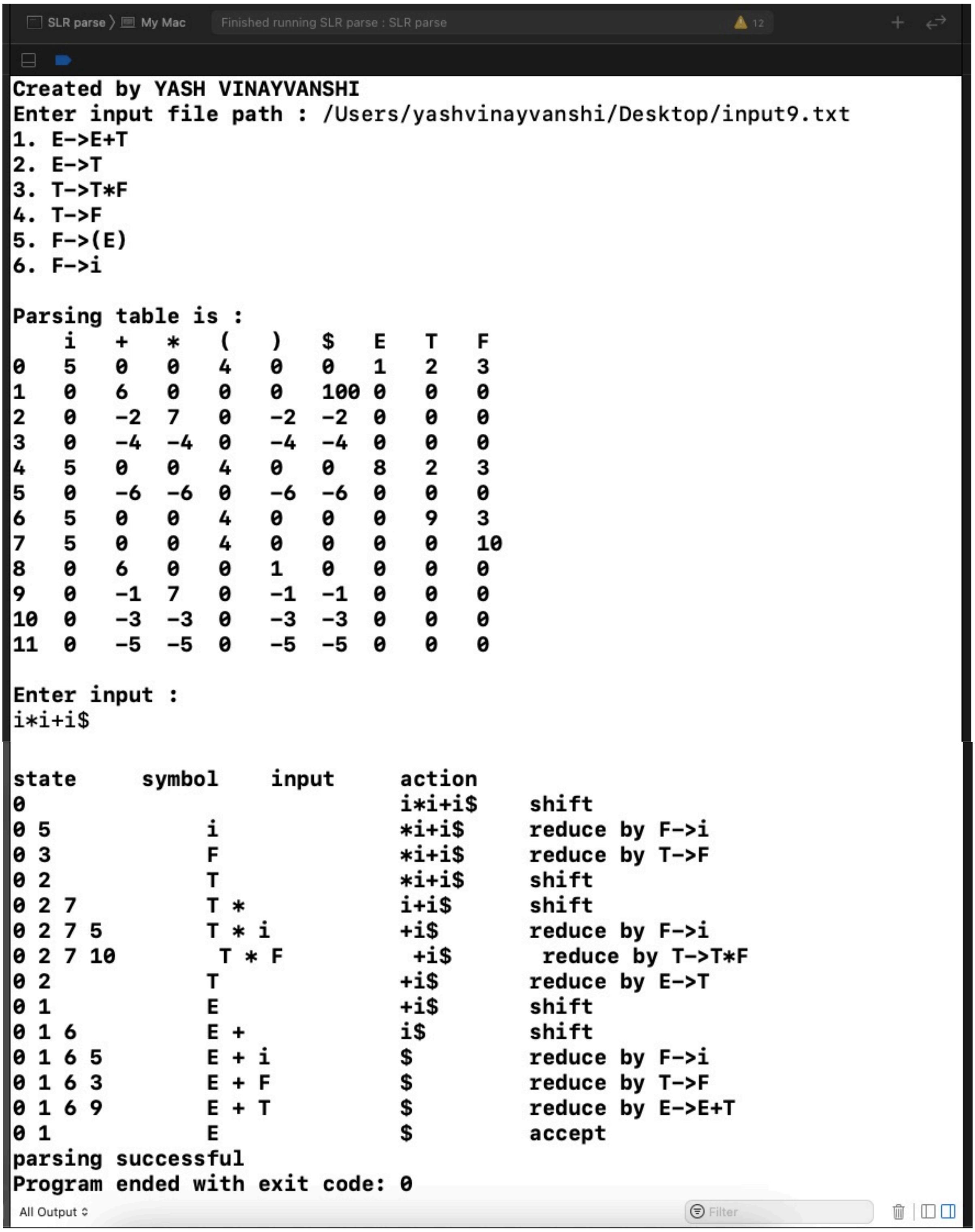
Run 1

Input

```
E E+T T
T T*F F
F (E) i

id + * ( ) $ E T F
0 S5 - - S4 - - 1 2 3
1 - S6 - - - A - - -
2 - R2 S7 - R2 R2 - - -
3 - R4 R4 - R4 R4 - - -
4 S5 - - S4 - - 8 2 3
5 - R6 R6 - R6 R6 - - -
6 S5 - - S4 - - - 9 3
7 S5 - - S4 - - - - 10
8 - S6 - - S11 - - - -
9 - R1 S7 - R1 R1 - - -
10 - R3 R3 - R3 R3 - - -
11 - R5 R5 - R5 R5 - - -
```

Console input : i*i+i$

Output



```
Created by YASH VINAYVANSHI
Enter input file path : /Users/yashvinayvanshi/Desktop/input9.txt
1. E->E+T
2. E->T
3. T->T*F
4. T->F
5. F->(E)
6. F->i

Parsing table is :
    i   +   *   (   )   $   E   T   F
0   5   0   0   4   0   0   1   2   3
1   0   6   0   0   0   100 0   0   0
2   0   -2  7   0   -2  -2  0   0   0
3   0   -4  -4  0   -4  -4  0   0   0
4   5   0   0   4   0   0   8   2   3
5   0   -6  -6  0   -6  -6  0   0   0
6   5   0   0   4   0   0   0   9   3
7   5   0   0   4   0   0   0   0   10
8   0   6   0   0   1   0   0   0   0
9   0   -1  7   0   -1  -1  0   0   0
10  0   -3  -3  0   -3  -3  0   0   0
11  0   -5  -5  0   -5  -5  0   0   0

Enter input :
i*i+i$

state     symbol    input     action
0                             i*i+i$     shift
0 5            i              *i+i$      reduce by F->i
0 3            F              *i+i$      reduce by T->F
0 2            T              *i+i$      shift
0 2 7          T *            i+i$       shift
0 2 7 5        T * i          +i$        reduce by F->i
0 2 7 10        T * F          +i$        reduce by T->T*F
0 2            T              +i$        reduce by E->T
0 1            E              +i$        shift
0 1 6          E +            i$         shift
0 1 6 5        E + i          $          reduce by F->i
0 1 6 3        E + F          $          reduce by T->F
0 1 6 9        E + T          $          reduce by E->E+T
0 1            E              $          accept
parsing successful
Program ended with exit code: 0
```

Run 2

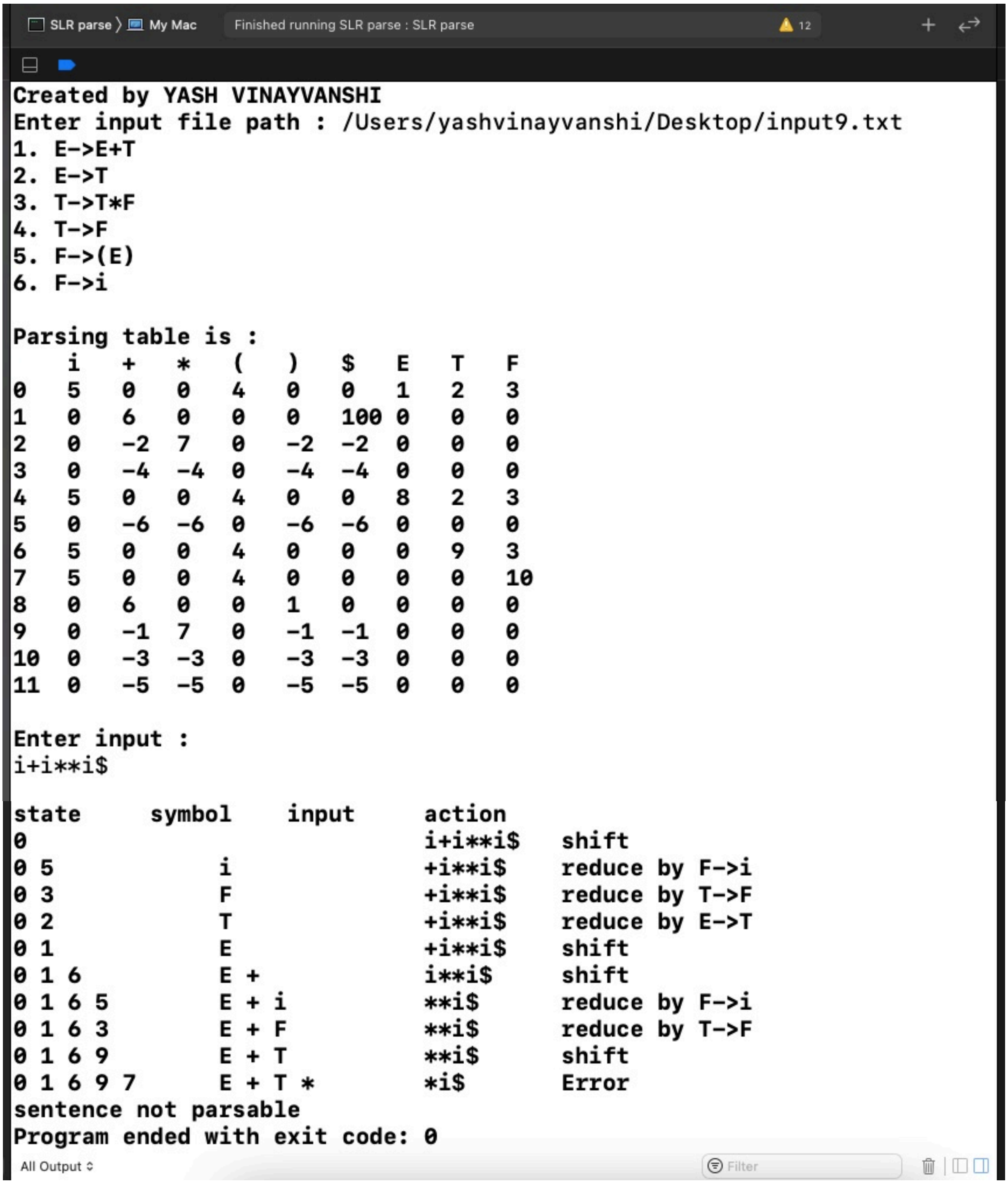
Input

```
E E+T T
T T*F F
F (E) i

id + * ( ) $ E T F
0 S5 - - S4 - - 1 2 3
```

```
1 - S6 - - - A - - -
2 - R2 S7 - R2 R2 - - -
3 - R4 R4 - R4 R4 - - -
4 S5 - - S4 - - 8 2 3
5 - R6 R6 - R6 R6 - - -
6 S5 - - S4 - - - 9 3
7 S5 - - S4 - - - - 10
8 - S6 - - S11 - - - -
9 - R1 S7 - R1 R1 - - -
10 - R3 R3 - R3 R3 - - -
11 - R5 R5 - R5 R5 - - -
```

Console input : i+i**i$

Output



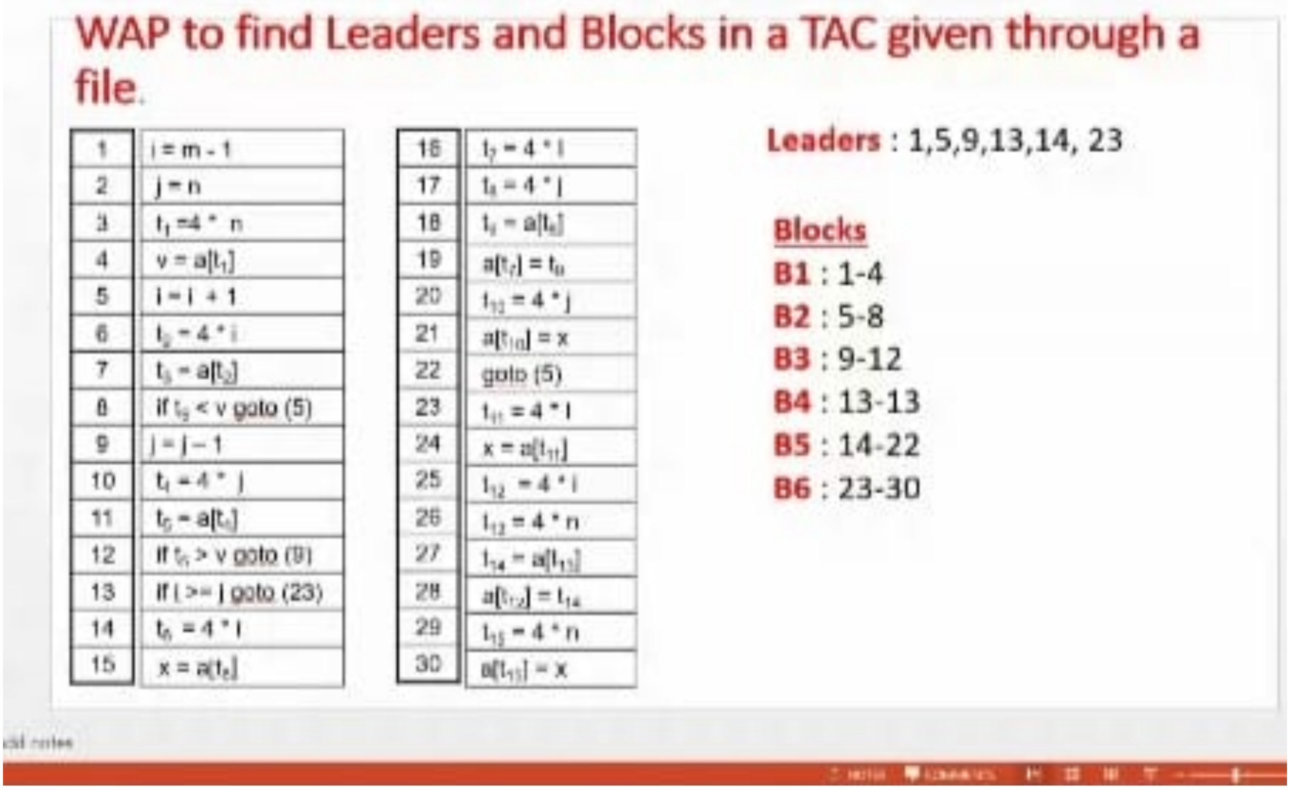
```
Created by YASH VINAYVANSHI
Enter input file path : /Users/yashvinayvanshi/Desktop/input9.txt
1. E->E+T
2. E->T
3. T->T*F
4. T->F
5. F->(E)
6. F->i

Parsing table is :
    i   +   *   (   )   $   E   T   F
0   5   0   0   4   0   0   1   2   3
1   0   6   0   0   0   100 0   0   0
2   0   -2  7   0   -2  -2  0   0   0
3   0   -4  -4  0   -4  -4  0   0   0
4   5   0   0   4   0   0   8   2   3
5   0   -6  -6  0   -6  -6  0   0   0
6   5   0   0   4   0   0   0   9   3
7   5   0   0   4   0   0   0   0   10
8   0   6   0   0   1   0   0   0   0
9   0   -1  7   0   -1  -1  0   0   0
10  0   -3  -3  0   -3  -3  0   0   0
11  0   -5  -5  0   -5  -5  0   0   0

Enter input :
i+i**i$

state     symbol    input     action
0                   i+i**i$   shift
0 5           i     +i**i$    reduce by F->i
0 3           F     +i**i$    reduce by T->F
0 2           T     +i**i$    reduce by E->T
0 1           E     +i**i$    shift
0 1 6         E +   i**i$     shift
0 1 6 5       E + i **i$      reduce by F->i
0 1 6 3       E + F **i$      reduce by T->F
0 1 6 9       E + T **i$      shift
0 1 6 9 7     E + T * *i$     Error
sentence not parsable
Program ended with exit code: 0
```

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD Lab program 10



WAP to find Leaders and Blocks in a TAC given through a file.

1	i = m - 1	16	t_7 = 4 * i
2	j = n	17	t_8 = 4 * j
3	t_1 =4 * n	18	t_9 = a[t_8]
4	v = a[t_1]	19	a[t_7] = t_9
5	i = i + 1	20	t_{10} = 4 * j
6	t_2 = 4 * i	21	a[t_{10}] = x
7	t_3 = a[t_2]	22	goto (5)
8	if t_3 < v goto (5)	23	t_{11} = 4 * i
9	j = j – 1	24	x = a[t_{11}]
10	t_4 = 4 * j	25	t_{12} = 4 * i
11	t_5 = a[t_4]	26	t_{13} = 4 * n
12	if t_5 > v goto (9)	27	t_{14} = a[t_{13}]
13	if i >= j goto (23)	28	a[t_{12}] = t_{14}
14	t_6 = 4 * i	29	t_{15} = 4 * n
15	x = a[t_6]	30	a[t_{15}] = x

Leaders : 1,5,9,13,14, 23

Blocks

B1 : 1-4
B2 : 5-8
B3 : 9-12
B4 : 13-13
B5 : 14-22
B6 : 23-30

The below program performs following tasks :

1. **Takes a three TAC address code from a file**
2. **Find leaders in TAC**
3. **Finds Blocks in TAC**

C++ Implementation : leaders and blocks in TAC

```
//
//  main.cpp
//  cd lab 10
//
//  Created by YASH VINAYVANSHI on 01/05/22.
//

#include <iostream>
#include <vector>
```

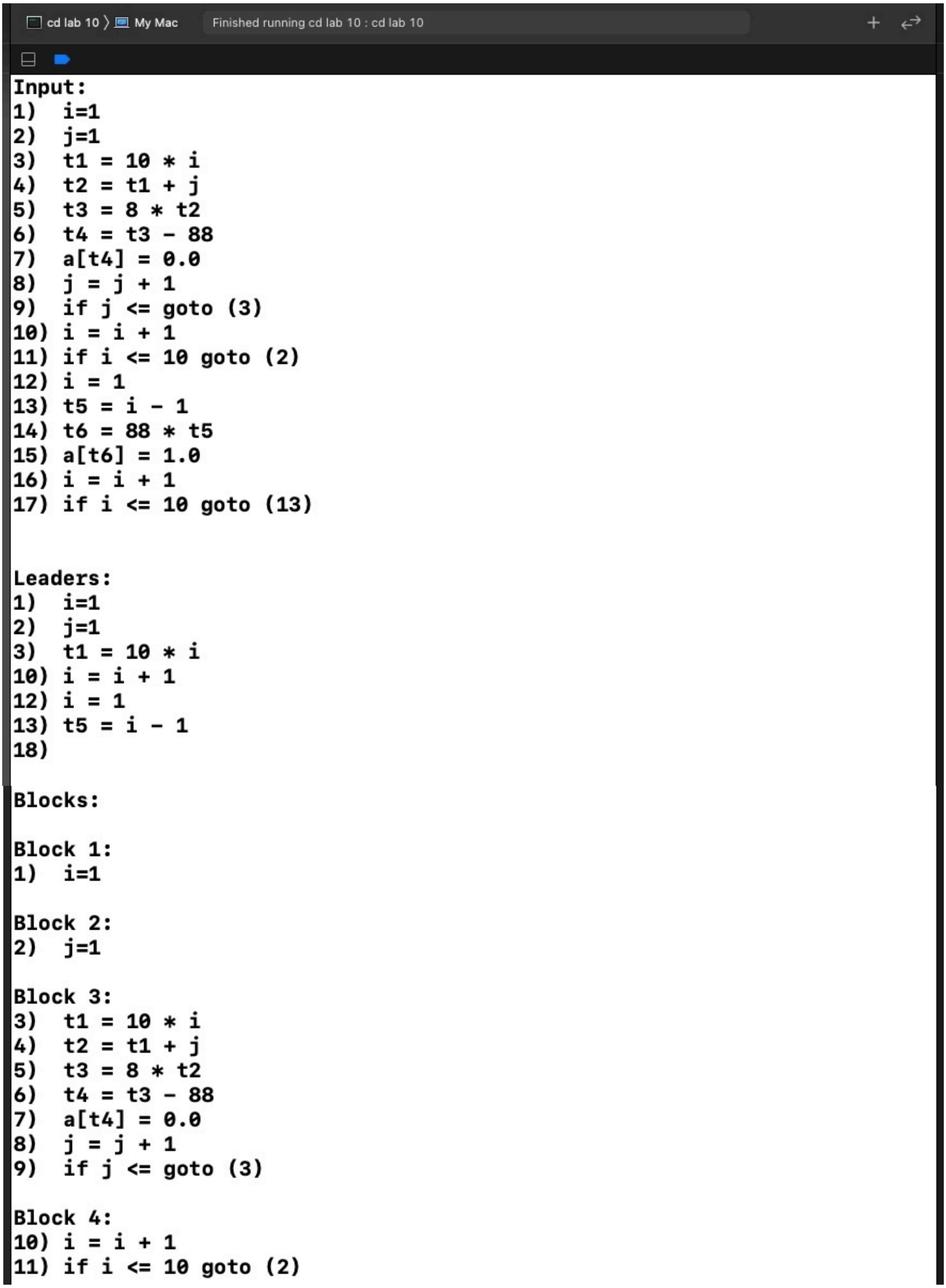
```
#include <map>
#include <fstream>

/*
 Rules to find Leaders
 Rule1 : first instruction of tac is a leader
 Rule2 :Instructions that are targets of unconditional or conditional
jump/goto statements are leaders.
 Rule3 : Instructions that immediately follow unconditional or
conditional jump/goto statements are considered leaders.
 */
using namespace std;
int main(){
    int i;
    ifstream fin("/Users/yashvinayvanshi/Desktop/leaders.txt");
    string num;
    cout<<"Input: "<<'\n';
    int count = 1;
    vector<string> a;
    //read file line by line in vector of strings a
    while(getline(fin,num)){
        //print the TAC input being taken
        cout<<count<<")\t"<<num<<'\n';
        a.push_back(num);
        count++;
    }

    map<int,int> mp;
    mp[0] = 0; //rule1
    int curr = 0;
    for(auto q : a){
        //for each line of TAC
        //pattern match goto or GOTO naively
        for(i=0;i<q.size()-3;i++){
            if((q[i] == 'g'&&q[i+1] == 'o'&&q[i+2] == 't'&&q[i+3] ==
'o')||(q[i] == 'G'&&q[i+1] == 'O'&&q[i+2] == 'T'&&q[i+3] == 'O')){
                mp[curr+1] = 1;
                string num = "";
                //to which line goto redirects
                int j = i+6;
                //the line number can be multi digit
                while(q[j]>='0'&&q[j]<='9'&&j<q.size()){ num+=q[j];
                    j++;
                }
                //convert line number string to integer
                int temp = stoi(num);
                //Rule 2
                mp[temp-1] = 1;
            }
        }
        curr++;
    }


    //leader : line no -> TAC on this line no
    vector<pair<int,string>> leaders;
    //block : vector of leaders
    vector<vector<pair<int,string>>> blocks;

    for(auto q : mp){
        leaders.push_back({q.first+1,a[q.first]});
    }
    vector<pair<int,string>> temp;
    for(i=0;i<a.size();i++){
        if(mp.count(i)){
            blocks.push_back(temp);
            temp.clear();
        }
        temp.push_back({i+1,a[i]});
    }
    blocks.push_back(temp);
    cout<<"\n\nLeaders:"<<'\n';
    for(auto q : leaders) cout<<q.first<<")\t"<<q.second<<'\n';
    cout<<"\nBlocks:\n"<<'\n';
    for(i=1;i<blocks.size();i++){
        cout<<"Block "<<i<<": "<<'\n';
        for(auto q : blocks[i]) cout<<q.first<<")\t"<<q.second<<'\n';
        cout<<'\n';
    }
    return 0;
}
```

Run

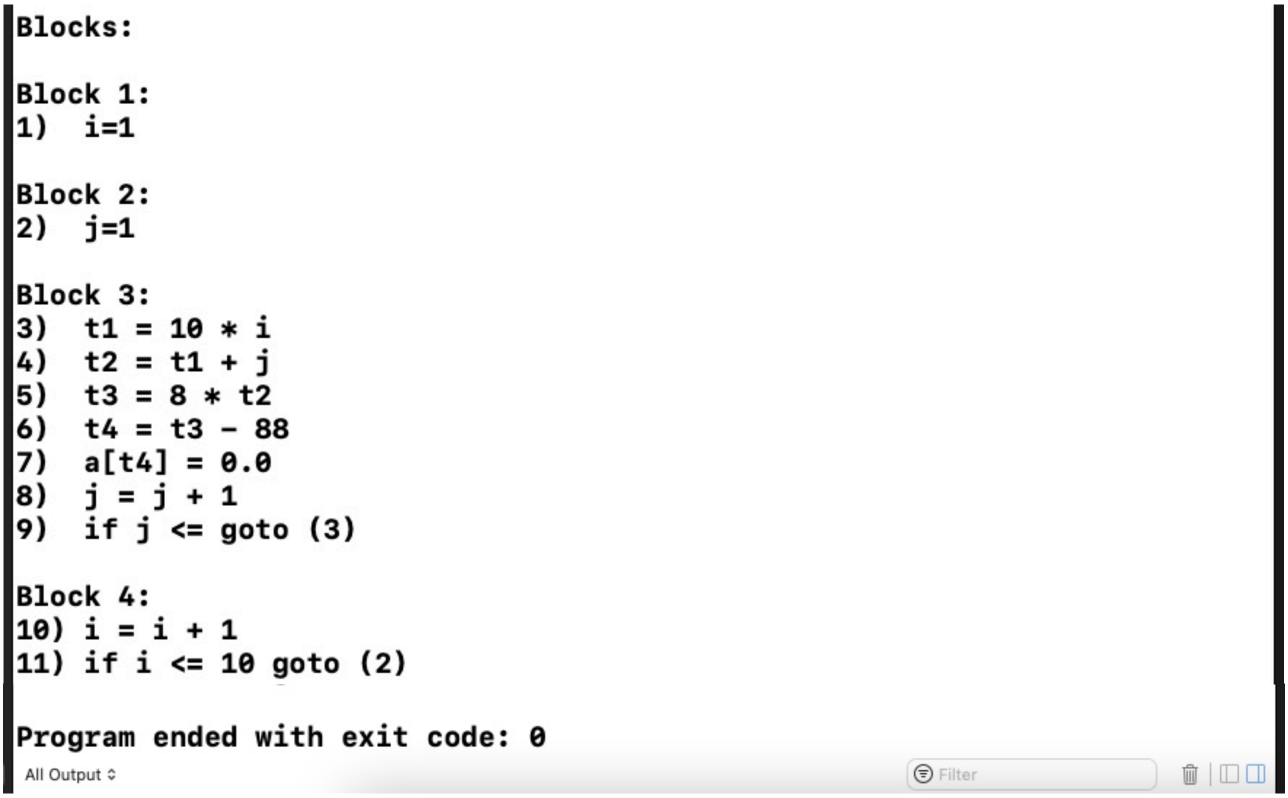
Input

```
i=1
j=1
t1 = 10 * i
t2 = t1 + j
t3 = 8 * t2
t4 = t3 - 88
a[t4] = 0.0
j = j + 1
if j <= goto (3)
i = i + 1
if i <= 10 goto (2)
i = 1
t5 = i - 1
t6 = 88 * t5
a[t6] = 1.0
i = i + 1
if i <= 10 goto (13)
```

Output



```
Input:
1)  i=1
2)  j=1
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j + 1
9)  if j <= goto (3)
10) i = i + 1
11) if i <= 10 goto (2)
12) i = 1
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i = i + 1
17) if i <= 10 goto (13)


Leaders:
1)  i=1
2)  j=1
3)  t1 = 10 * i
10) i = i + 1
12) i = 1
13) t5 = i - 1
18)

Blocks:

Block 1:
1)  i=1

Block 2:
2)  j=1

Block 3:
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j + 1
9)  if j <= goto (3)

Block 4:
10) i = i + 1
11) if i <= 10 goto (2)
```



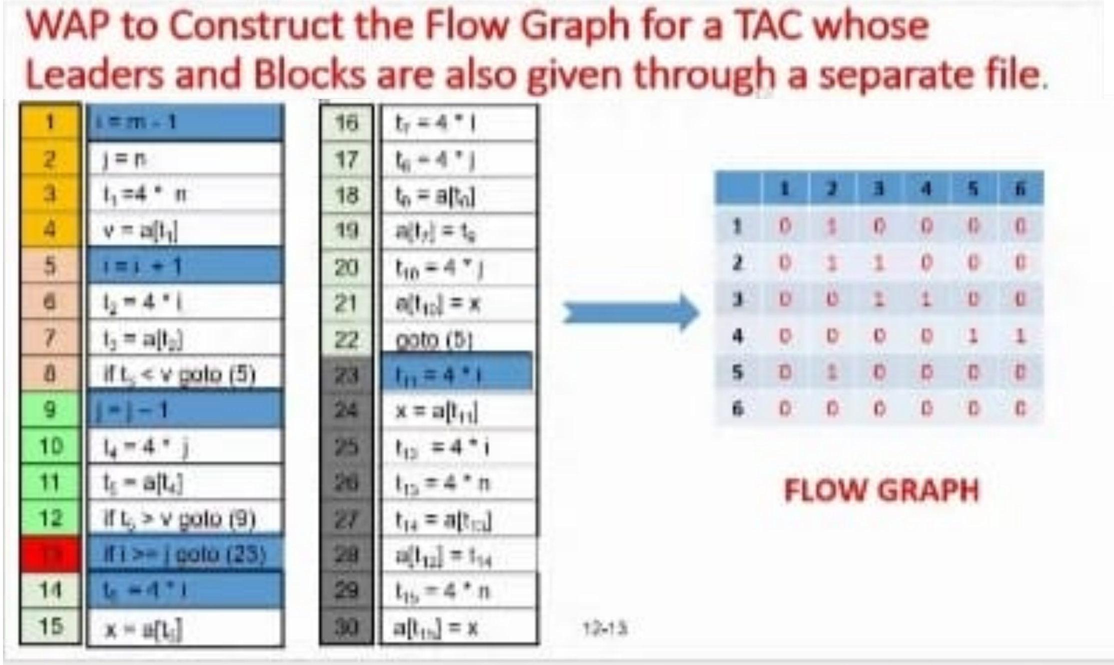
```
Blocks:

Block 1:
1)  i=1

Block 2:
2)  j=1

Block 3:
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j + 1
9)  if j <= goto (3)

Block 4:
10) i = i + 1
11) if i <= 10 goto (2)

Program ended with exit code: 0
```

COMPILER DESIGN LAB : CEN 692

SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

CD Lab program 11



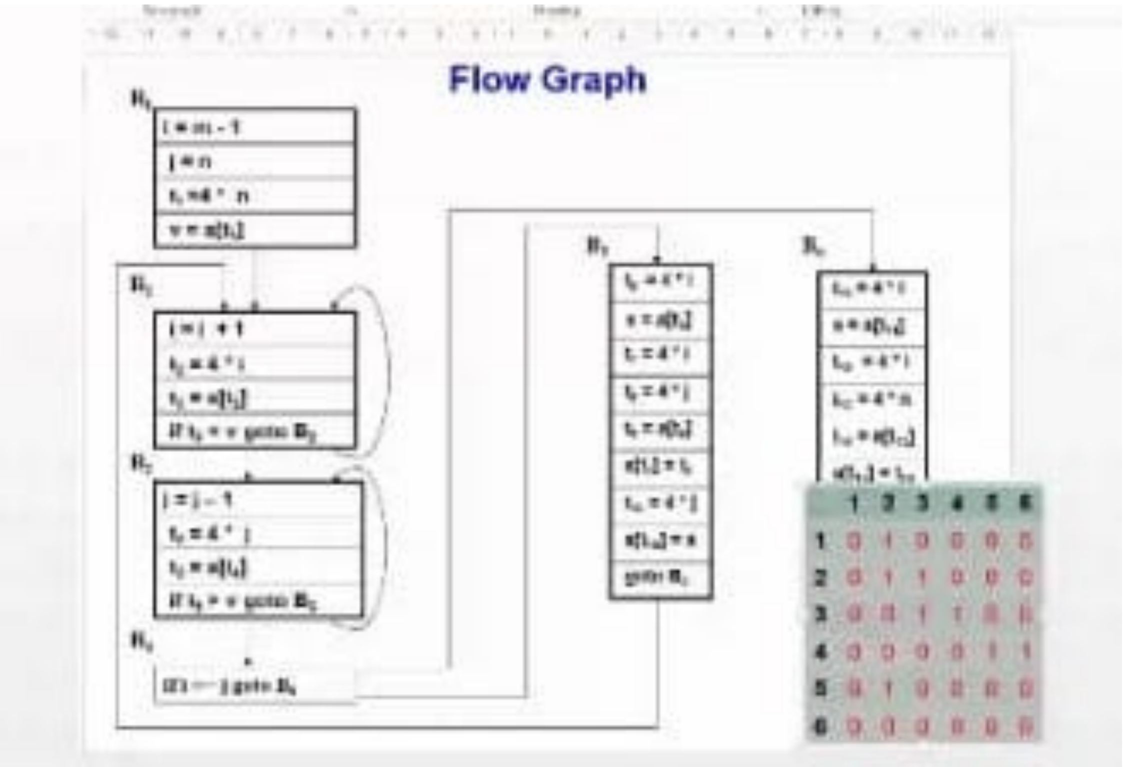
WAP to Construct the Flow Graph for a TAC whose Leaders and Blocks are also given through a separate file.

1	i = m - 1	16	t_7 = 4 * i
2	j = n	17	t_8 = 4 * j
3	t_1 =4 * n	18	t_9 = a[t_8]
4	v = a[t_1]	19	a[t_7] = t_9
5	i = i + 1	20	t_{10} = 4 * j
6	t_2 = 4 * i	21	a[t_{10}] = x
7	t_3 = a[t_2]	22	goto (5)
8	if t_3 < v goto (5)	23	t_{11} = 4 * i
9	j = j – 1	24	x = a[t_{11}]
10	t_4 = 4 * j	25	t_{12} = 4 * i
11	t_5 = a[t_4]	26	t_{13} = 4 * n
12	if t_5 > v goto (9)	27	t_{14} = a[t_{13}]
13	if i >= j goto (23)	28	a[t_{12}] = t_{14}
14	t_6 = 4 * i	29	t_{15} = 4 * n
15	x = a[t_6]	30	a[t_{15}] = x

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	1	1	0	0
4	0	0	0	0	1	1
5	0	1	0	0	0	0
6	0	0	0	0	0	0

FLOW GRAPH

12-13



Flow Graph

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	1	1	0	0
4	0	0	0	0	1	1
5	0	1	0	0	0	0
6	0	0	0	0	0	0

The below program performs following tasks :

1. **Takes a three TAC address code from a file**
2. **Find leaders in TAC**
3. **Finds Blocks in TAC**
4. **Build its flow graph in represented in adjacency matrix**

C++ Implementation : flow graph

```
//
//  main.cpp
//  cd lab 11
//
//  Created by YASH VINAYVANSHI on 04/05/22.
//
#include <iostream>
#include <vector>
#include <map>
#include <fstream>

/*
 Rules to find Leaders
 Rule1 : first instruction of tac is a leader
 Rule2 :Instructions that are targets of unconditional or conditional
jump/goto statements are leaders.
 Rule3 : Instructions that immediately follow unconditional or
conditional jump/goto statements are considered leaders.
 */
using namespace std;
int getblock(int, vector<int>, int);
int main(){
    int i;
    ifstream fin("/Users/yashvinayvanshi/Desktop/cd lab 10/
leaders.txt");
    string num;
    cout<<"Input: "<<'\n';
    int count = 1;
    vector<string> a;
    //read file line by line in vector of strings a
    while(getline(fin,num)){
        //print the TAC input being taken
        cout<<count<<")\t"<<num<<'\n';
        a.push_back(num);
        count++;
    }

    int N = a.size();
    map<int,int> mp;
    vector<int> lead;
    map<int,int> gotos;
    map<int,int> unconditionalgotos;
    lead.push_back(0);//rule1
    int curr = 0;
    for(auto q : a){
        //for each line of TAC
        //pattern match goto or GOTO naively
        for(i=0;i<q.size()-3;i++){
            if((q[i] == 'g'&&q[i+1] == 'o'&&q[i+2] == 't'&&q[i+3] ==
'o')||(q[i] == 'G'&&q[i+1] == 'O'&&q[i+2] == 'T'&&q[i+3] == 'O')){
                if(curr < N-1) lead.push_back(curr+1);//rule 3
                string num = "";
                //to which line goto redirects
                int j = i+6;
                //the line number can be multi digit
                while(q[j]>='0'&&q[j]<='9'&&j<q.size()){ num+=q[j];
                    j++;
                }
                //convert line number string to integer
                int temp = stoi(num);
                if(i == 0)
                    unconditionalgotos[curr+1] = temp;
                gotos[curr+1] = temp;
                //Rule 2
                lead.push_back(temp-1);
            }
        }
        curr++;
    }
    sort(lead.begin(), lead.end());
    for(auto it : unconditionalgotos)
        cout<<it.first<<" "<<it.second<<endl;


    //leader : line no -> TAC on this line no
    vector<pair<int,string>> leaders;
    //block : vector of leaders
    vector<vector<pair<int,string>>> blocks;

    for(auto q : lead){
        leaders.push_back({q+1,a[q]});
    }

    vector<pair<int,string>> temp;
    for(i=0;i<a.size();i++){
        if(find(lead.begin(), lead.end(), i)!=lead.end()){
            blocks.push_back(temp);
            temp.clear();
        }
        temp.push_back({i+1,a[i]});
    }
    blocks.push_back(temp);

    cout<<"\n\nLeaders:"<<'\n';
    for(auto q : leaders) cout<<q.first<<")\t"<<q.second<<'\n';
    cout<<"\nBlocks:\n"<<'\n';
    //0th block not used
    for(i=1;i<blocks.size();i++){
        cout<<"Block "<<i<<": "<<'\n';
        for(auto q : blocks[i]) cout<<q.first<<")\t"<<q.second<<'\n';
        cout<<'\n';
    }
```

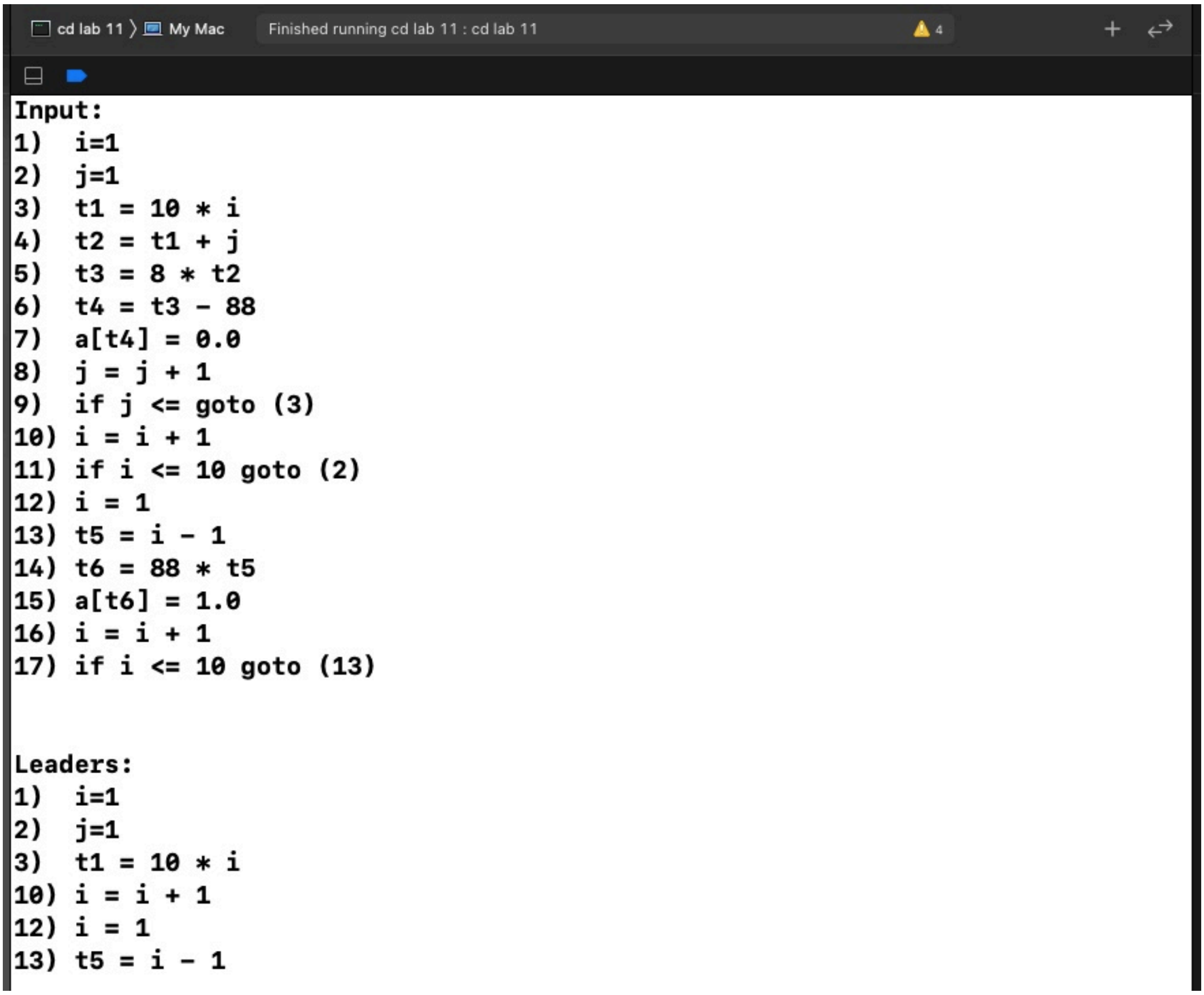
```cpp
    int n = blocks.size()-1;
    int A[n][2]; int k = 0;
    for(auto it : mp){
        A[k][0] = it.first;
    }
    int flowgraph[n][n];
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            flowgraph[i][j] = 0;
    //lead is now being used for range
    lead.push_back(N);
    int s = lead.size();
    for(auto it : gotos){
        int n1 = it.first;
        int n2 = it.second;
        int frmblk = getblock(n1, lead, s);
        int toblk = getblock(n2, lead, s);
        flowgraph[frmblk-1][toblk-1] = 1;
    }
    int blk = 0;
    for(auto it : lead){
        if(!unconditionalgotos.count(it)){
            flowgraph[blk][blk+1] = 1;
            blk++;
        }
    }

    cout<<"  ";
    for(int i=0; i<n; i++)
        cout<<i+1<<" ";
    cout<<endl;
    for(int i=0; i<n; i++){
        cout<<i+1<<" ";
        for(int j=0; j<n; j++){
            cout<<flowgraph[i][j]<<" ";
        }
        cout<<endl;
    }
    return 0;
}
int getblock(int line, vector<int>lead, int s){
    int blk = 0;
    int inblk;
    for(int i=0; i<s-1; i++){
        if(line >= lead[i] && line <= lead[i+1]){
            inblk = blk+1;
            break;
        }
        blk++;
    }
    return inblk;
}
```

Run

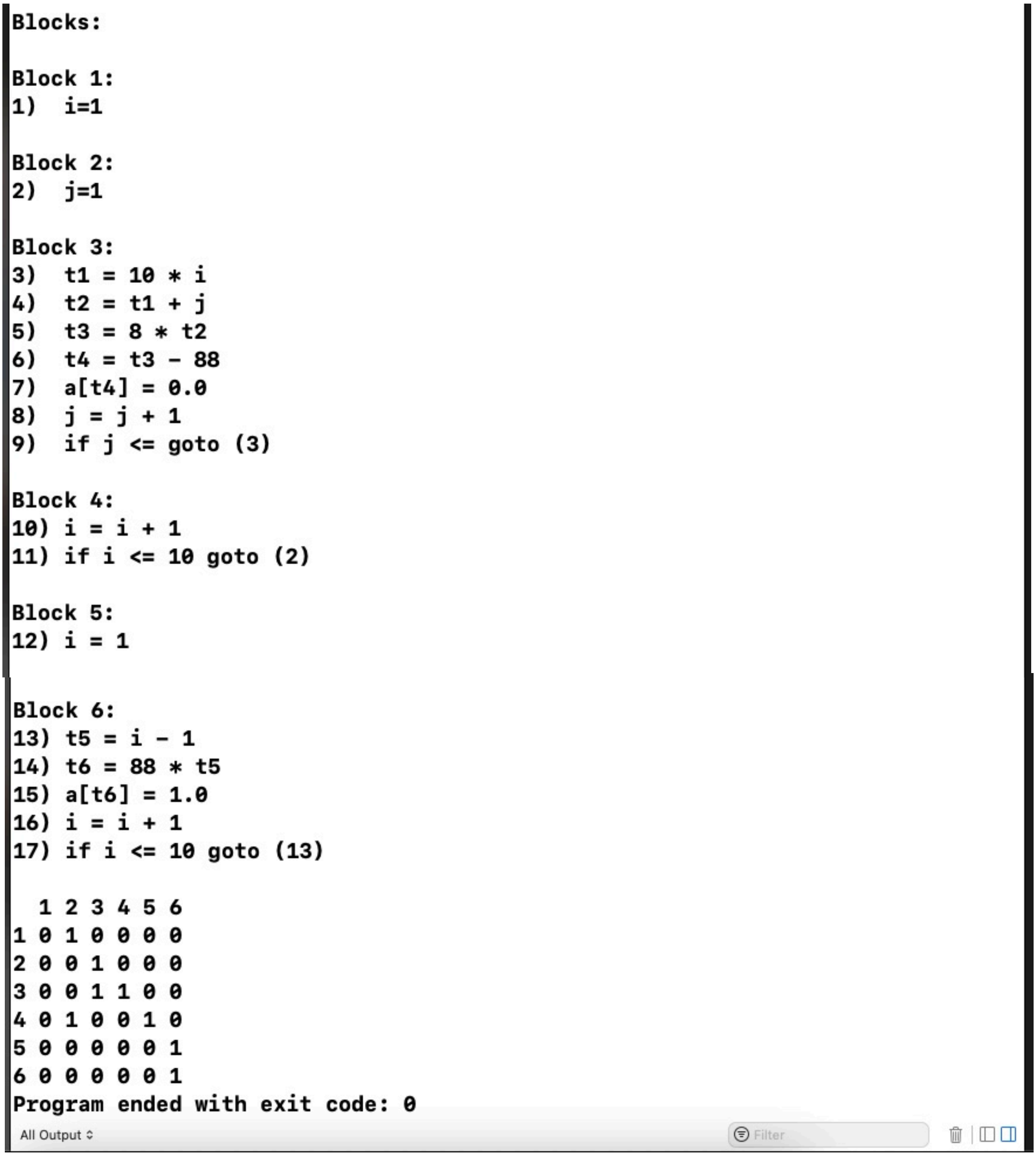
Input

```
i=1
j=1
t1 = 10 * i
t2 = t1 + j
t3 = 8 * t2
t4 = t3 - 88
a[t4] = 0.0
j = j + 1
if j <= goto (3)
i = i + 1
if i <= 10 goto (2)
i = 1
t5 = i - 1
t6 = 88 * t5
a[t6] = 1.0
i = i + 1
if i <= 10 goto (13)
```

Output



```
Input:
1)  i=1
2)  j=1
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j + 1
9)  if j <= goto (3)
10) i = i + 1
11) if i <= 10 goto (2)
12) i = 1
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i = i + 1
17) if i <= 10 goto (13)


Leaders:
1)  i=1
2)  j=1
3)  t1 = 10 * i
10) i = i + 1
12) i = 1
13) t5 = i - 1
```



```
Blocks:

Block 1:
1)  i=1

Block 2:
2)  j=1

Block 3:
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j + 1
9)  if j <= goto (3)

Block 4:
10) i = i + 1
11) if i <= 10 goto (2)

Block 5:
12) i = 1

Block 6:
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i = i + 1
17) if i <= 10 goto (13)

  1 2 3 4 5 6
1 0 1 0 0 0 0
2 0 0 1 0 0 0
3 0 0 1 1 0 0
4 0 1 0 0 1 0
5 0 0 0 0 0 1
6 0 0 0 0 0 1
Program ended with exit code: 0
```

COMPILER DESIGN LAB : CEN 692

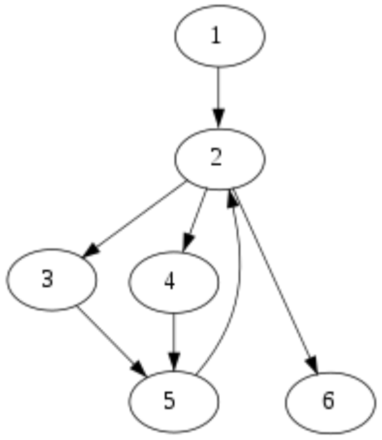
SUBMITTED BY : **YASH VINAYVANSHI**
B.TECH COMPUTER ENGINEERING (6th SEMESTER)
ROLL NO. 19BCS081
JAMIA MILLIA ISLAMIA FET, NEW DELHI

SUBMITTED TO : **DR. SARFARAZ MASOOD**
PROFESSOR
DEPARTMENT OF COMPUTER ENGINEERING
JAMIA MILLIA ISLAMIA FET, NEW DELHI

P12. WAP to Construct the Domination List & the Dominator List for a TAC whose FLOW GRAPH is given through a file. Only the FLOW GRAPH in the form of a matrix must be given thru the file.

C++ Implementation : dominator list

```
//
//  main.cpp
//  dominator list
//
//  Created by YASH VINAYVANSHI on 09/05/22.
//

/*
 Algorithm used : Naive
 
 for each vertex v != s
    delete v
    find all vertices still reachable by s (by BFS or DFS)
    vertex v dominates all unreachable vertices
 
 BFS used here : T(V) = O(V^2) as adj matrix is used
 one BFS for each vertex -> T(V) = O(V^3)
 */

#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
#include <queue>
#include <unordered_set>
using namespace std;

void readAdjMat(string &, vector<vector<int>>&);
void printAdjMat(vector<vector<int>>);
void printDomList(vector<vector<int>>);
vector<int> findUnreachable(vector<vector<int>>, int, int, int);
int main(int argc, const char * argv[]) {
    string filepath;
    vector<vector<int>> adjmat;
    cout<<"Enter filepath : "; getline(cin, filepath);
    readAdjMat(filepath, adjmat);
    printAdjMat(adjmat);

    //assuming source is 0
    int n = adjmat[0].size();
    vector<vector<int>> domination_list;
    vector<vector<int>> dominator_list(n);
    for(int i=0; i<n; i++){
        vector<int> temp = findUnreachable(adjmat, 0, i, n);
        domination_list.push_back(temp);
        for(auto it : temp)
            dominator_list[it].push_back(i);
    }

    cout<<"Domination list is : (x -> y => y is dominated by 
x)"<<endl; printDomList(domination_list);
    cout<<"Dominator list is : (x -> y => x is dominated by 
y)"<<endl; printDomList(dominator_list);
    return 0;
}

void readAdjMat(string &filepath, vector<vector<int>> &adjmat){
    vector<vector<string>> content;
    vector<string> row;
    string line, word;
    ifstream ifile;
    ifile.open(filepath);
    if(ifile.is_open()){
        while(getline(ifile, line)){
            row.clear();
            stringstream str(line);
            while(getline(str, word, ' ')) row.push_back(word);
            content.push_back(row);
        }
    }
    else{ cout<<"i/p File not opened\n"; exit(0);}
    ifile.close();

    vector<int> temp;
    for(auto it : content){
        temp.clear();
        for(auto it1 : it){
            temp.push_back(stoi(it1));
        }
        adjmat.push_back(temp);
    }
}

void printAdjMat(vector<vector<int>> adjmat){
    cout<<"Adjacency matrix of Directed flow graph is : "<<endl;
    int count = 1;
    cout<<"  ";
    for(auto it : adjmat){
        cout<<count<<" ";
        count++;
    }
    cout<<endl;
    count = 1;
    for(auto it : adjmat){
        cout<<count<<" ";
        for(auto it1 : it){
            cout<<it1<<" ";
        }
        count++;
        cout<<endl;
    }
}

vector<int> findUnreachable(vector<vector<int>> adjmat, int src,
int v, int n){
    unordered_set<int> visited;
    queue<int> BFS;
    if(v!=src){
        BFS.push(src);
        visited.insert(src);
    }
    while(!BFS.empty()){
        int curr = BFS.front(); BFS.pop();
        for(int i=0; i<n; i++){
            if(adjmat[curr][i] == 1 &&
visited.find(i)==visited.end() && i!=v){
                BFS.push(i);
                visited.insert(i);
            }
        }
    }
    vector<int> unvisited;
    for(int i=0; i<n; i++){
        if(visited.find(i)==visited.end())
            unvisited.push_back(i);
    }
    return unvisited;
}

void printDomList(vector<vector<int>> domlist){
    int count = 1;
    for(auto it : domlist){
        cout<<count<<" -> ";
        for(auto it1 : it){
            cout<<it1+1<<" ";
        }
        count++;
        cout<<endl;
    }
}
```

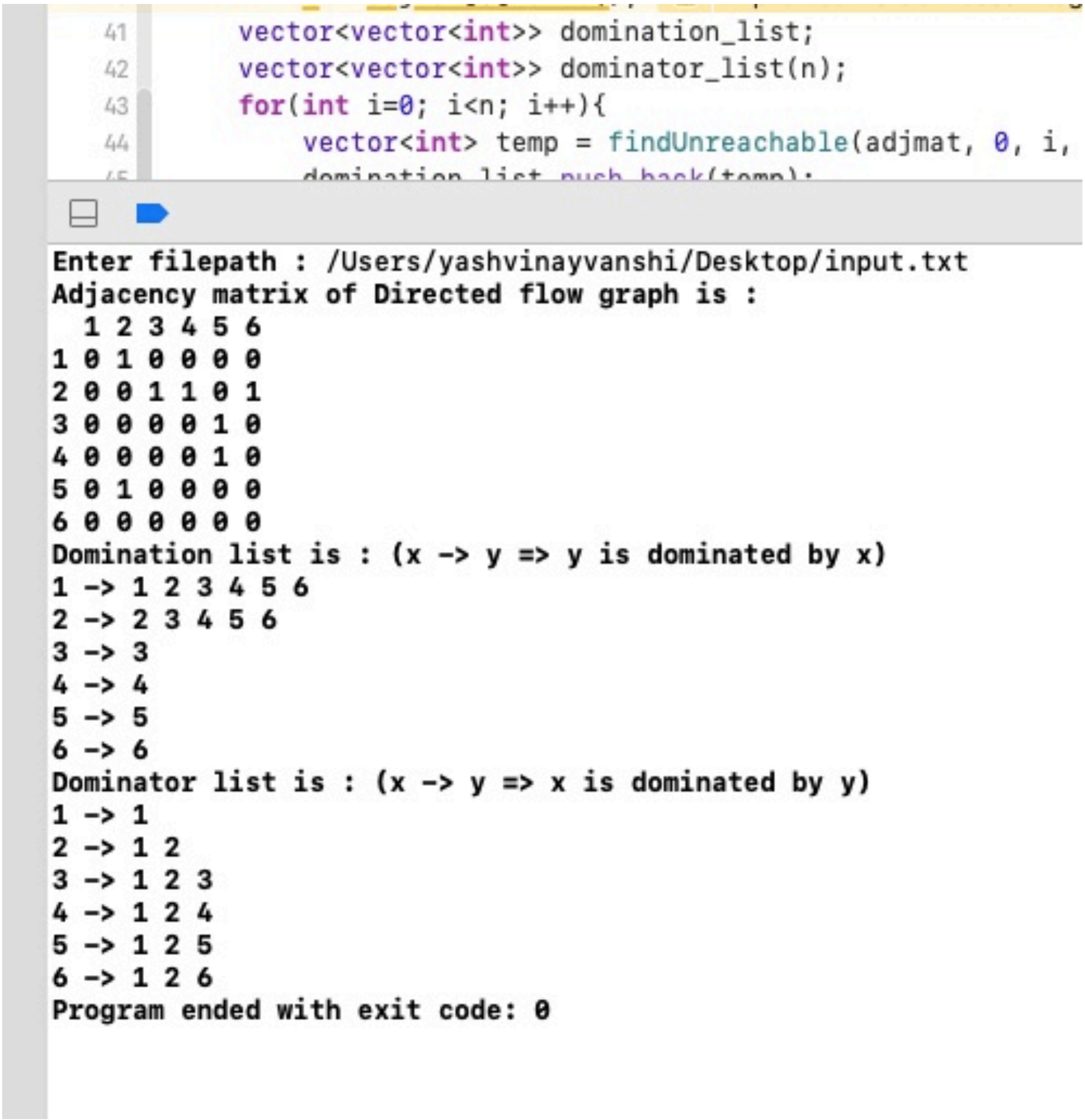


Run

Input

```
0 1 0 0 0 0
0 0 1 1 0 1
0 0 0 0 1 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 0
```

Output



```
    vector<vector<int>> domination_list;
    vector<vector<int>> dominator_list(n);
    for(int i=0; i<n; i++){
        vector<int> temp = findUnreachable(adjmat, 0, i,
        domination_list.push_back(temp);
```

```
Enter filepath : /Users/yashvinayvanshi/Desktop/input.txt
Adjacency matrix of Directed flow graph is :
  1 2 3 4 5 6
1 0 1 0 0 0 0
2 0 0 1 1 0 1
3 0 0 0 0 1 0
4 0 0 0 0 1 0
5 0 1 0 0 0 0
6 0 0 0 0 0 0
Domination list is : (x -> y => y is dominated by x)
1 -> 1 2 3 4 5 6
2 -> 2 3 4 5 6
3 -> 3
4 -> 4
5 -> 5
6 -> 6
Dominator list is : (x -> y => x is dominated by y)
1 -> 1
2 -> 1 2
3 -> 1 2 3
4 -> 1 2 4
5 -> 1 2 5
6 -> 1 2 6
Program ended with exit code: 0
```