Compiler Design Lab

CEN-692

B.Tech Computer Engineering
VI Semester

Submitted by:
Aiman Fatima
20BCS008

Submitted to:
Dr. Sarfaraz Masood Sir

Department of Computer Engineering,
Faculty of Engineering & Technology,
Jamia Millia Islamia, New Delhi
2023

Index

Deterministic Finite Automaton
Lab 1

Aiman Fatima
20BCS008

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
#include <set>
using namespace std;

int stringToInt(string line)
{
    int i = 0;
    int num = 0;
    while (line[i] != 0)
    {
        if (line[0] == '-')
        {
            return -1;
        }
        num *= 10;
        num += (line[i] - '0');
        i++;
    }
    return num;
}

int main()
{
    ifstream DFA;
    string line;
    DFA.open("l1_inp.txt");
    int i = 0;
    int initial;
    vector<int> final;
    set<int> ff;
    vector<vector<int>> table;

    while (getline(DFA, line))
    {
        if (i == 0)
            initial = stringToInt(line);
        else if (i == 1)
        {
            int x = 0;
            string num = "";
            while (line[x] != 0)
            {
```

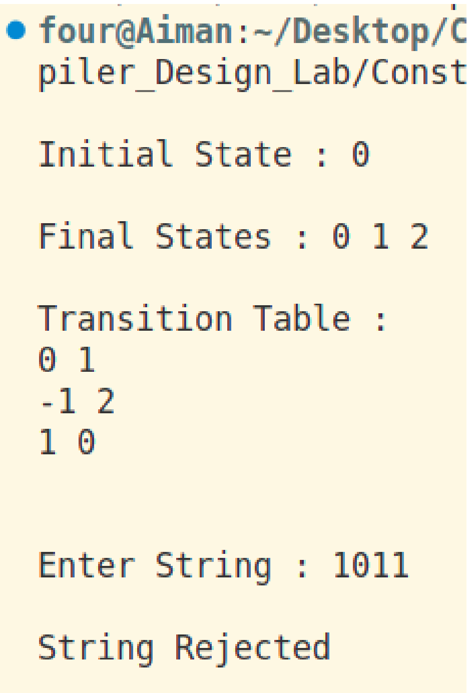
```
            if (line[x] == ' ')
            {
                final.push_back(stringToInt(num));
                num = "";
                x++;
                continue;
            }
            num += line[x++];
        }
        final.push_back(stringToInt(num));
        ff.insert(stringToInt(num));
    }
    else
    {
        vector<int> temp;
        int x = 0;
        string num = "";
        while (line[x] != 0)
        {
            if (line[x] == ' ')
            {
                temp.push_back(stringToInt(num));
                num = "";
                x++;
                continue;
            }
            num += line[x++];
        }
        temp.push_back(stringToInt(num));
        table.push_back(temp);
    }
    i++;
}

DFA.close();

//------------PRINTING THE DFA----------
cout << "\nInitial State : " << initial << "\n";
cout << "\nFinal States : ";
for (int i = 0; i < final.size(); i++)
    cout << final[i] << " ";
cout << "\n\nTransition Table :\n";
for (int i = 0; i < table.size(); i++)
{
    for (int j = 0; j < table[i].size(); j++)
    {
        cout << table[i][j] << " ";
    }
    cout << "\n";
}
cout << "\n";
//-----------------------
```

```
    int curr = initial;
    bool acc = false;
    cout << "\nEnter String : ";
    string s = "";
    getline(cin, s);
    // cin >> s;
    if (s == "" && ff.count(initial))
    {
        cout << "\n|| String Accepted\n\n";
        cout << "Transition State\n\n-> ";
        cout << initial << "\n\n";
        return 0;
    }
    std::vector<int> statess;
    int size = s.size(), k = 0;
    while (curr != -1 && k < size)
    {
        string t = "";
        t += s[k++];
        curr = table[curr][stringToInt(t)];
        statess.push_back(curr);
    }
    for (int i = 0; i < final.size(); i++)
    {
        if (curr == final[i])
        {
            cout << "\nString Accepted\n\n";
            acc = true;
            // return 0;
        }
    }
    if (acc)
    {
        cout << "Transition states";
        cout << "\n";
        for (int i = 0; i < statess.size(); i++)
        {
            cout << " -> " << statess[i];
        }
        cout << "\n\n";
    }
    else
        cout << "\nString Rejected\n\n";
    return 0;
}
```

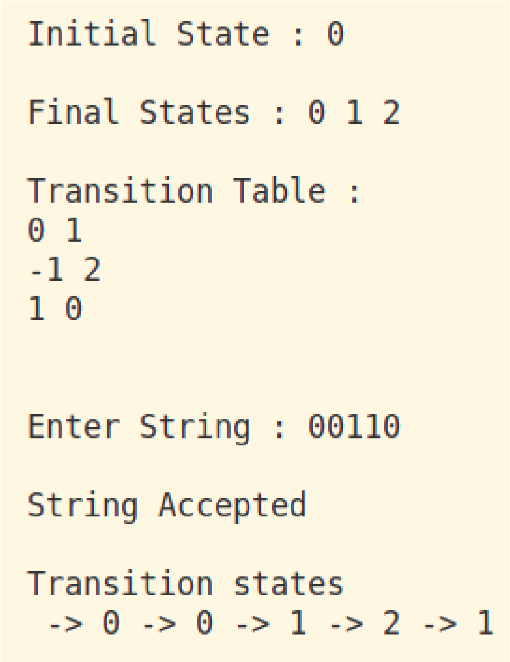
File *input.txt*

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

Screenshots of the output



```
● four@Aiman:~/Desktop/C
  piler_Design_Lab/Const

  Initial State : 0

  Final States : 0 1 2

  Transition Table :
  0 1
  -1 2
  1 0


  Enter String : 1011

  String Rejected
```



```
Initial State : 0

Final States : 0 1 2

Transition Table :
0 1
-1 2
1 0


Enter String : 00110

String Accepted

Transition states
 -> 0 -> 0 -> 1 -> 2 -> 1
```

Mealy Machine
Lab 2

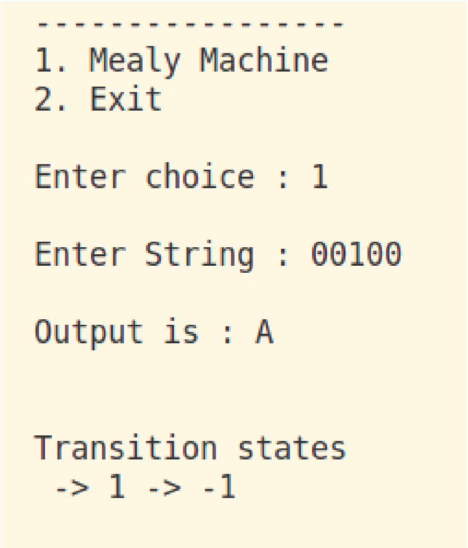
Aiman Fatima
20BCS008

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int initial;
vector<vector<int>> table;
vector<vector<string>> OUT;

int stringToInt(string line)
{
    int i = 0;
    int num = 0;
    while (line[i] != 0)
    {
        if (line[0] == '-')
        {
            return -1;
        }
        num *= 10;
        num += (line[i] - '0');
        i++;
    }
    return num;
}

void mealyMachine()
{
    int curr = initial;
    getchar();
    cout << "\nEnter String : ";
    bool f(true);
    string s = "";
    // cin>>s;
    getline(cin, s);
    if (s == "")
    {
        f = false;
        cout << "\nEpsilon Input\nNo output string\n"
             << endl;
    }
    std::vector<int> states;
    int size = s.size(), k = 0;
    string output = "";
    while (curr != -1 && k < size)
    {
        string t = "";
        t += s[k++];
        if (OUT[curr][stringToInt(t)] != "-1")
        {
            output += OUT[curr][stringToInt(t)];
        }
        curr = table[curr][stringToInt(t)];
        states.push_back(curr);
    }
    if (f)
    {
        cout << "\nOutput is : " << output << endl;
        cout << "\n\nTransition states";
        cout << "\n";
        for (int i = 0; i < states.size(); i++)
            cout << " -> " << states[i];
        cout << "\n\n";
    }
}

int main()
{
    ifstream MEALY;
    string line;
    MEALY.open("inp2m.txt");
    int i = 0;

    while (getline(MEALY, line))
    {
        if (i == 0)
        {
            initial = stringToInt(line);
        }
        else
        {
            vector<int> temp1;
            vector<string> temp2;
            int x = 0;
            int y = 0;
            string num = "";
            string outNum = "";
            while (line[x] != 0)
            {
                if (line[x] == ' ')
                {
                    if (y % 2 == 0)
                    {
                        temp1.push_back(stringToInt(num));
                        num = "";
                    }
                    else
                    {
                        temp2.push_back(outNum);
                        outNum = "";
                    }
                    x++;
                    y++;
                    continue;
                }
                if (y % 2 == 0)
                {
                    num += line[x++];
                }
                else
                {
                    outNum += line[x++];
                }
            }
            temp2.push_back(outNum);
            table.push_back(temp1);
            OUT.push_back(temp2);
        }
        i++;
    }

    MEALY.close();
    while (1)
    {
        cout << "\n-----------------\n";
        cout << "1. Mealy Machine\n";
        cout << "2. Exit\n";
        cout << "\nEnter choice : ";
        int choice;
        cin >> choice;
        switch (choice)
        {
        case 1:
            mealyMachine();
            break;
        case 2:
            cout << "\n--The End--\n\n";
            return 0;
        default:
            cout << "\nWrong Choice\n";
            break;
        }
    }
}
```

File *input.txt*

```
0
1 A -1 -1
-1 -1 2 AB
3 B 11 B
-1 -1 4 AB
5 A -1 -1
-1 -1 6 B
4 A 7 BA
8 A -1 -1
-1 -1 9 B
10 A -1 -1
-1 -1 11 B
0 A -1 -1
```

Screenshot of the output



```
-----------------
1. Mealy Machine
2. Exit

Enter choice : 1

Enter String : 00100

Output is : A


Transition states
 -> 1 -> -1
```

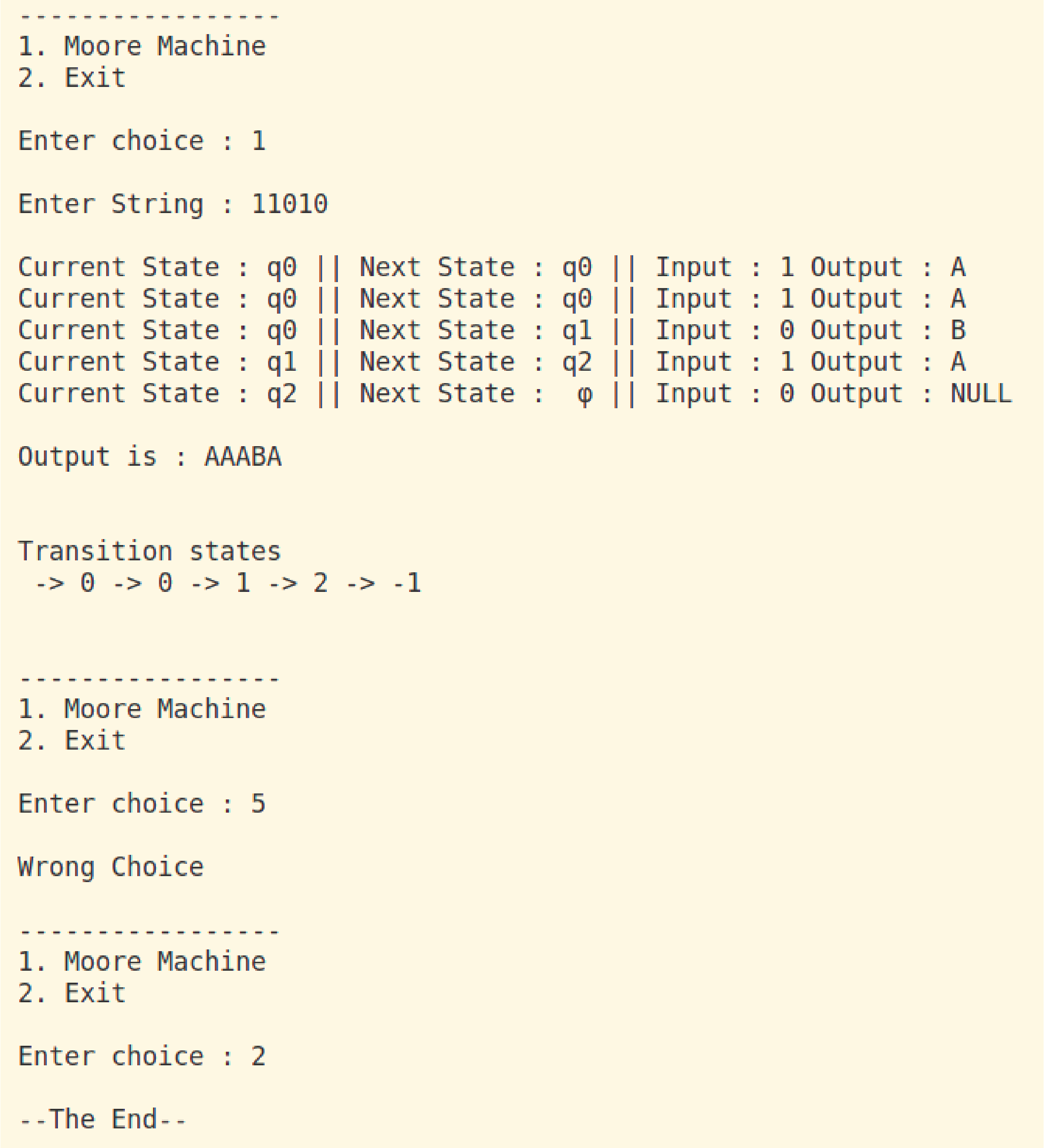
Moore Machine
Lab 3
Aiman Fatima
20BCS008

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int initial;
vector<vector<int>> table;
vector<string> OUT;
int stringToInteger(string line)
{
    int i = 0;
    int num = 0;
    while (line[i] != 0)
    {
        if (line[0] == '-')
        {
            return -1;
        }
        num *= 10;
        num += (line[i] - '0');
        i++;
    }
    return num;
}

void mooreMachine()
{
    int curr = initial;
    cout << "\nEnter String : ";
    bool f(true);
    getchar();
    string s = "";
    getline(cin, s);
    cout << "\n";
    // getline(cin, s);
    if (s == "")
    {
        f = false;
        cout << "\nEpsilon Input\nNo output string\n"
             << endl;
    }
    std::vector<int> states;
    int size = s.size(), k = 0;
    string output = OUT[curr];
    while (curr != -1 && k < size)
    {
        string t = "";
        t += s[k++];
        int x = stringToInteger(t);
        cout << "Current State : q" << curr << " || Next State : ";
        if (table[curr][x] == -1)
            cout << " φ";
        else
            cout << "q" << table[curr][x];
        cout << " || Input : " << x;
        curr = table[curr][x];
        states.push_back(curr);
        if (curr != -1)
            output += OUT[curr];
        cout << " Output : ";
        if (curr == -1)
            cout << "NULL" << endl;
        else
            cout << OUT[curr] << endl;
    }

    if (f)
    {
        cout << "\nOutput is : " << output << endl;
        cout << "\n\nTransition states";
        cout << "\n";
        for (int i = 0; i < states.size(); i++)
            cout << " -> " << states[i];
        cout << "\n\n";
    }
}

int main()
{
    ifstream MOORE;
    string line;
    MOORE.open("l3_inp.txt");
    int i = 0;
    while (getline(MOORE, line))
    {
        if (i == 0)
            initial = stringToInteger(line);
        else
        {
            vector<int> temp1;
            vector<string> temp2;
            int x = 0;
            int y = 0;
            string num = "";
            while (line[x] != 0)
            {
                if (line[x] == ' ')
                {
                    temp1.push_back(stringToInteger(num));
                    num = "";
                    x++;
                    continue;
                }
                num += line[x++];
            }
            table.push_back(temp1);
            OUT.push_back(num);
        }
        i++;
    }

    MOORE.close();

    while (1)
    {
        cout << "\n-----------------\n";
        cout << "1. Moore Machine\n";
        cout << "2. Exit\n";
        cout << "\nEnter choice : ";
        int choice;
        cin >> choice;
        switch (choice)
        {
        case 1:
            mooreMachine();
            break;
        case 2:
            cout << "\n--The End--\n\n";
            return 0;
        default:
            cout << "\nWrong Choice\n";
            break;
        }
    }
}
```

File *input.txt*

```
0
1 A -1 -1
-1 -1 2 AB
3 B 11 B
-1 -1 4 AB
5 A -1 -1
-1 -1 6 B
4 A 7 BA
8 A -1 -1
-1 -1 9 B
10 A -1 -1
-1 -1 11 B
0 A -1 -1
```

Screenshot of the output



```
-----------------
1. Moore Machine
2. Exit

Enter choice : 1

Enter String : 11010

Current State : q0 || Next State : q0 || Input : 1 Output : A
Current State : q0 || Next State : q0 || Input : 1 Output : A
Current State : q0 || Next State : q1 || Input : 0 Output : B
Current State : q1 || Next State : q2 || Input : 1 Output : A
Current State : q2 || Next State :  φ || Input : 0 Output : NULL

Output is : AAABA


Transition states
 -> 0 -> 0 -> 1 -> 2 -> -1



-----------------
1. Moore Machine
2. Exit

Enter choice : 5

Wrong Choice

-----------------
1. Moore Machine
2. Exit

Enter choice : 2

--The End--
```

DFA to NFA
Lab 4
Aiman Fatima
20BCS008

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

// global variables
int initial;
vector<int> final;
vector<vector<vector<int>>> NFA;
vector<vector<int>> DFA;

// string to integer conversion
int stringToInt(string line)
{
    int i = 0;
    int num = 0;
    while (line[i] != 0)
    {
        if (line[0] == '-')
        {
            return -1;
        }
        num *= 10;
        num += (line[i] - '0');
        i++;
    }
    return num;
}

// function converting NFA to DFA
void nfaToDfa()
{
    getchar();
    map<int, vector<int>> state_map;

    int inserted_size = NFA.size();
    int original_size = NFA.size();

    for (int i = 0; i < NFA.size(); i++)
    {
        vector<vector<int>> tempArray;
        vector<vector<int>> temp(NFA[i].size());
```

```
for (int j = 0; j < NFA[i].size(); j++)
{
    if (NFA[i][j].size() > 1)
    {
        // mapping and changing state to new state
        for (auto it : state_map)
        {
            if (it.second == NFA[i][j])
            {
                NFA[i][j].clear();
                NFA[i][j].push_back(it.first);
            }
        }
        state_map.insert(pair<int, vector<int>>(inserted_size, NFA[i][j]));
        NFA[i][j].clear();
        NFA[i][j].push_back(inserted_size++);

        // merging states
        for (auto it : state_map[inserted_size - 1])
        {
            for (int x = 0; x < NFA[it].size(); x++)
            {
                vector<int> s;
                for (int y = 0; y < NFA[it][x].size(); y++)
                {
                    s.push_back(NFA[it][x][y]);
                }
                for (int y = 0; y < s.size(); y++)
                {
                    temp[x].push_back(s[y]);
                }
            }
        }

        // removing duplicates and -1
        for (int x = 0; x < temp.size(); x++)
        {
            vector<int> v;
            set<int> s;
            for (int y = 0; y < temp[x].size(); y++)
            {
                if (temp[x][y] != -1)
                {
                    s.insert(temp[x][y]);
                }
            }
            if (s.empty())
            {
                s.insert(-1);
            }
            for (auto it : s)
            {
                v.push_back(it);
```

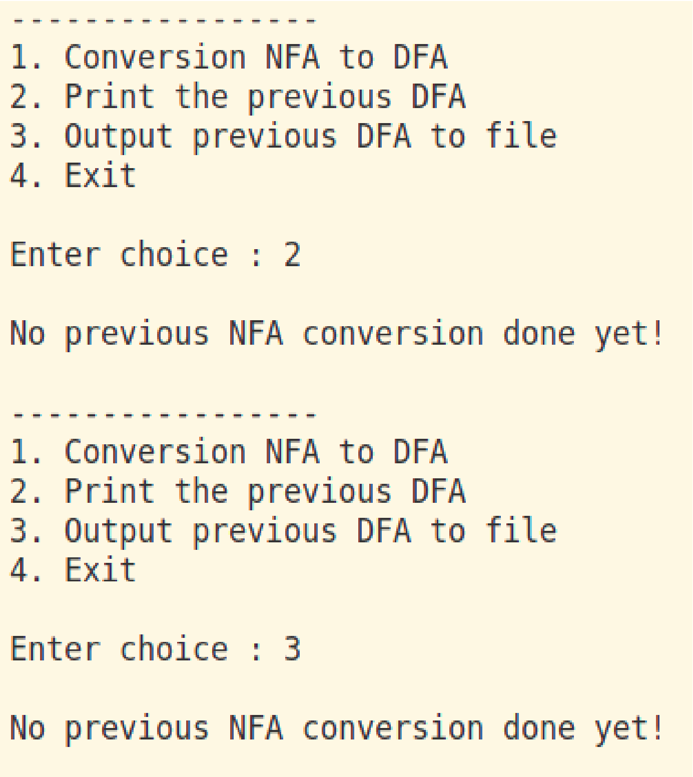
```
                    }
                    temp[x] = v;
                }

                // checking if state already exists in the map
                for (int x = 0; x < temp.size(); x++)
                {
                    if (temp[x].size() > 1)
                    {
                        bool found = false;
                        for (auto it : state_map)
                        {
                            if (it.second == temp[x])
                            {
                                temp[x].clear();
                                temp[x].push_back(it.first);
                                tempArray.push_back(temp[x]);
                                found = true;
                            }
                        }
                        if (found)
                        {
                            continue;
                        }

                        for (int y = temp[x].size() - 1; y >= 0; y--)
                        {
                            // checking if the new create state exists in the temp array and if it does, then merge it with
removing the duplicate
                            if (temp[x][y] >= original_size)
                            {
                                temp[x].erase(temp[x].begin() + y);
                                for (auto it : state_map[temp[x][y]])
                                {
                                    temp[x].push_back(it);
                                }
                                vector<int> v;
                                set<int> s;
                                for (int y = 0; y < temp[x].size(); y++)
                                {
                                    s.insert(temp[x][y]);
                                }
                                for (auto it : s)
                                {
                                    v.push_back(it);
                                }
                                temp[x] = v;
                            }
                        }

                        bool newFound = false;
                        for (auto it : state_map)
                        {
                            if (it.second == temp[x])
                            {
                                temp[x].clear();
                                temp[x].push_back(it.first);
                                tempArray.push_back(temp[x]);
                                newFound = true;
                                break;
                            }
                        }
                        if (newFound)
                        {
                            continue;
                        }
                        state_map.insert(pair<int, vector<int>>(inserted_size, temp[x]));
                        temp[x].clear();
                        temp[x].push_back(inserted_size);
                        tempArray.push_back(temp[x]);
                    }
                    else
                    {
                        tempArray.push_back(temp[x]);
                    }
                }
                NFA.push_back(tempArray);
                tempArray.clear();
            }
        }
    }

    // making final states
    for (auto it : state_map)
    {
        for (auto it1 : it.second)
        {
            for (auto it2 : final)
            {
                if (it1 == it2)
                {
                    final.push_back(it.first);
                }
            }
        }
    }
}

void printDFA()
{
    cout << "\n__Printing the DFA__\n\n";
    //  printing initial state
    cout << "Initial State: " << initial << endl;

    // printing final states
    cout << "Final States: ";
    for (auto it : final)
    {
        cout << it << " ";
    }
    cout << endl;

    // printing DFA
    cout << "State Transitions" << endl;
    for (int i = 0; i < NFA.size(); i++)
    {
        for (int j = 0; j < NFA[i].size(); j++)
        {
            for (int k = 0; k < NFA[i][j].size(); k++)
            {
                cout << NFA[i][j][k] << " ";
            }
        }
        cout << endl;
    }
}

void DFAfile()
{
    // writing to file "Converted_DFA.txt"
    ofstream file1;
    file1.open("Converted_DFA.txt");
    file1 << initial << endl;
    for (auto it : final)
    {
        file1 << it << " ";
    }
    file1 << endl;
    for (int i = 0; i < NFA.size(); i++)
    {
        for (int j = 0; j < NFA[i].size(); j++)
        {
            for (int k = 0; k < NFA[i][j].size(); k++)
            {
                file1 << NFA[i][j][k];
            }
            if (j != NFA[i].size() - 1)
            {
                file1 << " ";
            }
        }
        if (i != NFA.size() - 1)
        {
            file1 << endl;
        }
    }
    file1.close();
}
```

```
int main()
{
    ifstream file; // file's ptr that accepts
    string line;
    file.open("l4_inp.txt"); // opens the file
    int i = 0;

    while (getline(file, line)) // reads the file line by line
    {
        if (i == 0)
        {
            initial = stringToInt(line);
        }
        else if (i == 1)
        {
            int x = 0;
            string num = "";
            while (line[x] != 0)
            {
                if (line[x] == ' ')
                {
                    final.push_back(stringToInt(num));
                    num = "";
                    x++;
                    continue;
                }
                num += line[x++];
            }
            final.push_back(stringToInt(num));
        }
        else
        {
            vector<vector<int>> temp;
            vector<int> temp1;
            int x = 0;
            string num = "";
            while (line[x] != 0)
            {
                if (line[x] == ',')
                {
                    temp1.push_back(stringToInt(num));
                    num = "";
                    x++;
                    continue;
                }
                if (line[x] == ' ')
                {
                    temp1.push_back(stringToInt(num));
                    temp.push_back(temp1);
                    temp1.clear();
                    num = "";
                    x++;
                    continue;
                }
                num += line[x++];
            }
            temp1.push_back(stringToInt(num));
            temp.push_back(temp1);
            NFA.push_back(temp);
        }
        i++;
    }

    file.close();

    // NFA to DFA
    // nfaToDfa();

    bool flag1(false);

    while (1)
    {
        cout << "\n-----------------\n";
        cout << "1. Conversion NFA to DFA\n";
        cout << "2. Print the previous DFA\n";
        cout << "3. Output previous DFA to file\n";
        cout << "4. Exit\n";
        cout << "\nEnter choice : ";
        int choice;
        cin >> choice;
        switch (choice)
        {
        case 1:
            // final.clear();
            // NFA.clear();
            DFA.clear();
            nfaToDfa();
            printDFA();
            DFAfile();
            flag1 = true;
            break;
        case 2:
            if (flag1)
                printDFA();
            else
                cout << "\nNo previous NFA conversion done yet!\n";
            break;
        case 3:
            if (flag1)
                DFAfile();
            else
                cout << "\nNo previous NFA conversion done yet!\n";
            break;
        case 4:
            cout << "\n--The End--\n\n";
            return 0;
        default:
            cout << "\nWrong Choice\n";
            break;
        }
    }

    // Printing the DFA

    return 0;
}
```

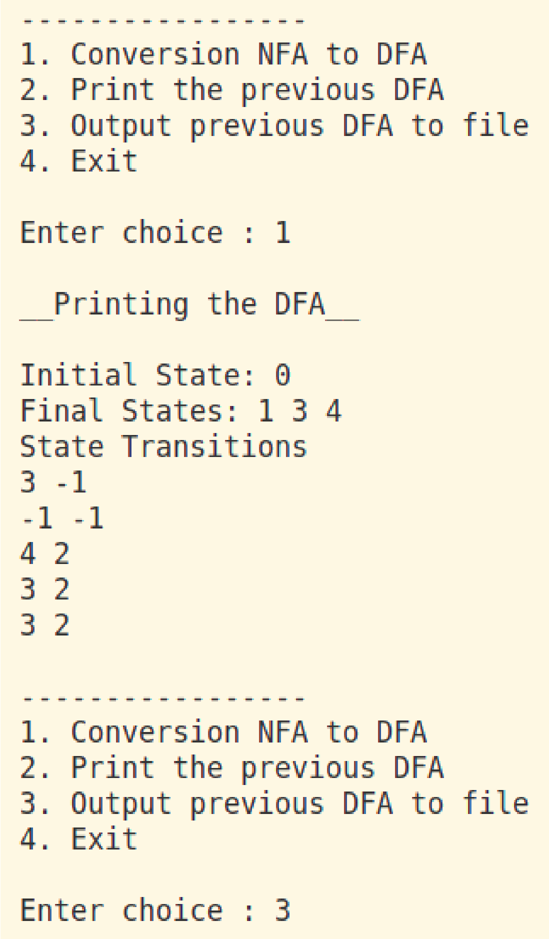
File *input.txt*

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

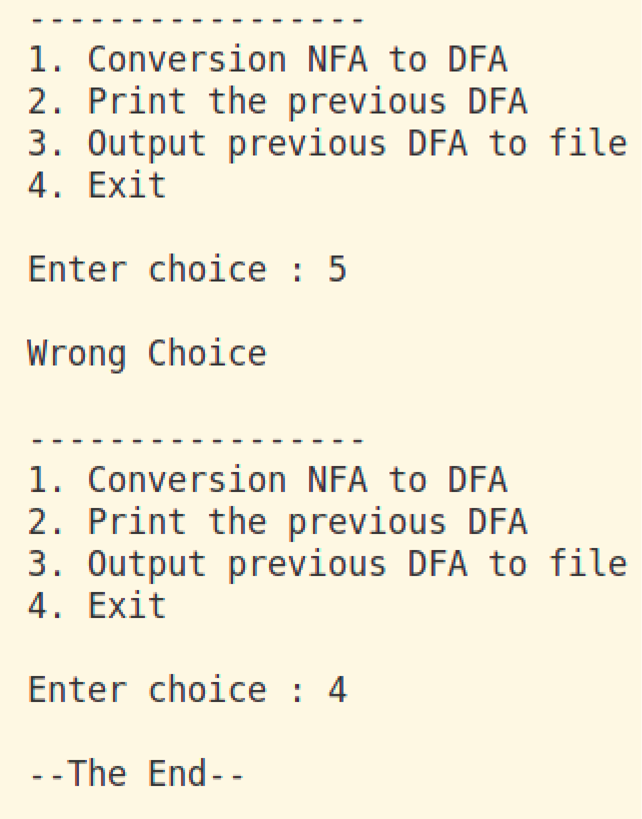
Screenshots of the output



```
-----------------
1. Conversion NFA to DFA
2. Print the previous DFA
3. Output previous DFA to file
4. Exit

Enter choice : 2

No previous NFA conversion done yet!

-----------------
1. Conversion NFA to DFA
2. Print the previous DFA
3. Output previous DFA to file
4. Exit

Enter choice : 3

No previous NFA conversion done yet!
```



```
-----------------
1. Conversion NFA to DFA
2. Print the previous DFA
3. Output previous DFA to file
4. Exit

Enter choice : 1

__Printing the DFA__

Initial State: 0
Final States: 1 3 4
State Transitions
3 -1
-1 -1
4 2
3 2
3 2


-----------------
1. Conversion NFA to DFA
2. Print the previous DFA
3. Output previous DFA to file
4. Exit

Enter choice : 3
```



```
-----------------
1. Conversion NFA to DFA
2. Print the previous DFA
3. Output previous DFA to file
4. Exit

Enter choice : 5

Wrong Choice


-----------------
1. Conversion NFA to DFA
2. Print the previous DFA
3. Output previous DFA to file
4. Exit

Enter choice : 4

--The End--
```

First and Follow of a CFG
Lab 5

Aiman Fatima
20BCS008

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

char Productions_List[100][100][100];
int FollowFound[100] = {0};

// Used to add to both First and Follow based on mode
void Add_Terminal_To_List(char terminal, char List[100][100], int Variable_Number, int mode)
{
    // if(mode)
    //      printf("\tNeed to add %c to Follow[%d]\n",terminal,Variable_Number);
    int len = strlen(List[Variable_Number]);
    int i, found_flag = 0;
    if (mode == 1)
    {
        if (terminal == '#')
            return;
    }
    for (i = 0; i < len; i++)
    {
        if (List[Variable_Number][i] == terminal)
        {
            // printf("\tAlready Present\n");
            found_flag = 1;
            break;
        }
    }
    if (found_flag == 0)
    {
        // printf("\tNot Present so added at pos %d\n",len);
        List[Variable_Number][len] = terminal;
        List[Variable_Number][len + 1] = '\0';
    }
    return;
}

// Used to Make The Production Table
int AddProductionsToList(int Variable_Number, char Production[100])
{
    int i, j, len;
    int Production_Number = 0;
    len = strlen(Production);
    for (i = 0, j = 0; i < len; i++)
    {
        if (Production[i] == '|')
        {
            Productions_List[Variable_Number][Production_Number][j] = '\0';
            Production_Number++;
            j = 0;
        }
        else
        {
            Productions_List[Variable_Number][Production_Number][j] = Production[i];
            j++;
        }
    }
    return (Production_Number + 1);
}

// Used to Display Entire Table
void DisplayTransitions(char Variable_List[100], int Corresponding_Number_of_Productions[100], int N)
{
    int i, j, z;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < Corresponding_Number_of_Productions[i]; j++)
        {
            printf("\t%c -> ", Variable_List[i]);
            for (z = 0; Productions_List[i][j][z] != '\0'; z++)
            {
                printf("%c", Productions_List[i][j][z]);
            }
            printf("\n");
        }
    }
}

// Used to return the location in the list of Variables given the character
int GetVariableNumber(char Variable, char Variable_list[100], int N)
{
    int i = 0, Variable_Number = 0;
    for (i = 0; i < N; i++)
        if (Variable_list[i] == Variable)
            Variable_Number = i;
    return (Variable_Number);
}

// Recursve function to find the First
int Find_First(char First[100][100], int Corresponding_Number_of_Productions[100], char Variable_list[100], char
original, int Found[], int Total)
{
    // printf("In Func for %c\n",original);
    int i, j, N, p, len, precision, Variable_Number;
    int Next_Variable_Number;
    int Nullout = 0, position;
    char t;
    Variable_Number = GetVariableNumber(original, Variable_list, Total);
    if (Found[Variable_Number] == -1)
        return -1;

    if (Found[Variable_Number] == 1)
        return 1;

    position = 0;
    precision = 0;

    N = Corresponding_Number_of_Productions[Variable_Number];
    for (position = 0; position <= precision; position++)
    {
        for (i = 0; i < N; i++)
        {
            // printf("In production:%s\n",Productions_List[Variable_Number][i]);
            t = Productions_List[Variable_Number][i][position];
            if (isupper(t))
            {
                // printf("Is Upper\n");
                Nullout = Find_First(First, Corresponding_Number_of_Productions, Variable_list, t, Found, Total);
                Next_Variable_Number = GetVariableNumber(t, Variable_list, Total);
                len = strlen(First[Next_Variable_Number]);
                for (j = 0; j <= len; j++)
                {
                    Add_Terminal_To_List(First[Next_Variable_Number][j], First, Variable_Number, 0);
                }

                if (Nullout == -1)
                {
                    len = strlen(Productions_List[Variable_Number][i]); // Rule 3 for First
                    if (precision < len)
                    {
                        // printf("\t\tNeed to go 1 more round casue of #");
                        precision += 1;
                    }
                }
            }
            else if (t == '#') // Rule 1 for First
            {
                Nullout = -1;
                Add_Terminal_To_List(t, First, Variable_Number, 0);
                Found[Variable_Number] = Nullout;
            }
            else
            {
                Add_Terminal_To_List(t, First, Variable_Number, 0); // Rule 2 for First
                Found[Variable_Number] = 1;
            }
        }
    }
    return (Nullout);
}
```

```
// Recursive Function to find the Follow
void Follow_Of(char Follow[100][100], char First[100][100], int original, char Variable_List[100], int N, int
Corresponding_Number_of_Productions[100], int Found[100])
{
    // printf("In Follow(%c)\n",Variable_List[original]);
    if (FollowFound[original] != 0)
    {
        return;
    }
    else
    {
        FollowFound[original] = 1;
    }

    char Variable = Variable_List[original];
    int Character_Number, i, j, Variable_Number;
    char character, next_character, temp;
    int Production_Number, Number_Of_Productions;
    int next_char_number, nflag, fflag, temp_number;
    // for(i=0;i<N;i++)
    //     printf("_%d",Found[i]);
    for (Variable_Number = 0; Variable_Number < N; Variable_Number++)
    {
        Number_Of_Productions = Corresponding_Number_of_Productions[Variable_Number];
        for (Production_Number = 0; Production_Number < Number_Of_Productions; Production_Number++)
        {
            Character_Number = 0;
            character = Productions_List[Variable_Number][Production_Number][Character_Number];
            for (; character != '\0'; Character_Number++)
            {
                character = Productions_List[Variable_Number][Production_Number][Character_Number];
                if (character == Variable)
                {
                    // printf("\tfound Variable in %c -> %s\
n",Variable_List[Variable_Number],Productions_List[Variable_Number][Production_Number]);
                    next_character = Productions_List[Variable_Number][Production_Number][Character_Number + 1];
                    if (next_character == '\0')
                    {
                        // printf("Is last character\nSo Follow(%c)==Follow(%c)\
n",Variable,Variable_List[Variable_Number]);
                        Follow_Of(Follow, First, Variable_Number, Variable_List, N,
Corresponding_Number_of_Productions, Found); // Rule 2 of Follow
                        for (i = 0; i < strlen(Follow[Variable_Number]); i++)
                        {
                            Add_Terminal_To_List(Follow[Variable_Number][i], Follow, original, 1);
                        }
                        // printf("Done\n");
                        break;
                    }
                    else
                    {
                        if (isupper(next_character))
                        {
```

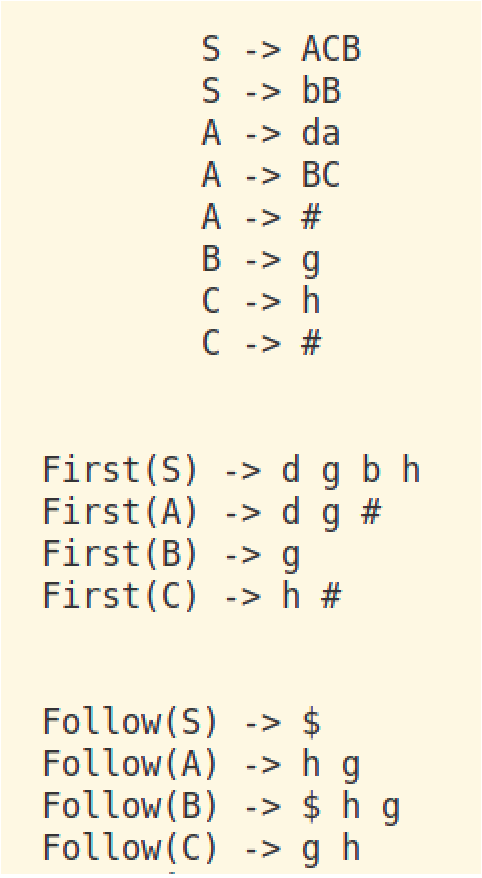
```
// printf("\tNext Character is Variable: %c\n",next_character);
next_char_number = GetVariableNumber(next_character, Variable_List, N);
// printf("number = %d\n",next_char_number);
if (Found[next_char_number] != -1) // Rule 3a of Follow
{
    // printf("\t%c Doesnt contain '#' as %d\n",next_character,Found[next_char_number]);
    for (i = 0; i < strlen(First[next_char_number]); i++)
    {
        Add_Terminal_To_List(First[next_char_number][i], Follow, original, 1);
    }
}
else
{
    // printf("Has '#' as %d so-\n",Found[next_char_number]);
    nflag = 0;
    fflag = strlen(Productions_List[Variable_Number][Production_Number]);
    fflag = fflag - (Character_Number + 1);

    temp = Productions_List[Variable_Number][Production_Number][Character_Number + 1];
    for (j = Character_Number + 1; temp != '\0'; j++)
    {
        temp = Productions_List[Variable_Number][Production_Number][j];
        if (temp == '\0')
            break;
        // printf("\t\tchecking %c",temp);
        if (islower(temp) || !isalnum(temp))
        {
            // printf("\t\tIs a Terminal\n");
            Add_Terminal_To_List(temp, Follow, original, 1);
            break;
        }
        else
        {
            // printf("\t\tIs a Variable\n");
            temp_number = GetVariableNumber(temp, Variable_List, N);
            if (Found[temp_number] == -1)
            {
                nflag += 1;
                for (i = 0; i < strlen(First[temp_number]); i++)
                {
                    Add_Terminal_To_List(First[temp_number][i], Follow, original, 1);
                }
            }
            else
            {
                for (i = 0; i < strlen(First[temp_number]); i++)
                    Add_Terminal_To_List(First[temp_number][i], Follow, original, 1);
                break;
            }
            if (fflag == nflag)
            {
                // printf("Rest is equivalent to '#'");
```

```
                                            Follow_Of(Follow, First, Variable_Number, Variable_List, N,
Corresponding_Number_of_Productions, Found);
                                            for (i = 0; i < strlen(Follow[Variable_Number]); i++)
                                            {
                                                Add_Terminal_To_List(Follow[Variable_Number][i], Follow, original, 1);
                                            }
                                        }
                                    }
                                }
                            }
                        }
                        else
                        {
                            Add_Terminal_To_List(next_character, Follow, original, 1);
                            break;
                        }
                    }
                }
            }
        }
    }
    // printf("\nEnd_OF_Function!!\n");
}

void main(void)
{
    int N = 0, i, j, k, z;
    int Number_of_Productions = 0, Variable_Number;
    char Variable, Start_Variable = 'S', Production[100];
    char Variable_List[100];
    char First[100][100];
    char Follow[100][100];
    int Found[100], Nullout;
    int Corresponding_Number_of_Productions[100];
    FILE *fp;
    fp = fopen("input.txt", "r");
    fscanf(fp, "\n%c -> %s", &Start_Variable, Production);
    N++;
    Variable_Number = 0;
    Variable_List[Variable_Number] = Start_Variable;
    Number_of_Productions = AddProductionsToList(Variable_Number, Production);
    Corresponding_Number_of_Productions[Variable_Number] = Number_of_Productions;
    Variable_Number++;
    while (!feof(fp))
    {
        // fscanf(fp, "%d", &N);
        fscanf(fp, "\n%c -> %s", &Variable, Production);
        Variable_List[Variable_Number] = Variable;
        Number_of_Productions = AddProductionsToList(Variable_Number, Production);
        Corresponding_Number_of_Productions[Variable_Number] = Number_of_Productions;
        Variable_Number++;
        N++;
    }
    printf("\n\n");
    DisplayTransitions(Variable_List, Corresponding_Number_of_Productions, N);
    printf("\n\n");
    // Find_First()
    for (i = 0; i < N; i++)
    {
        strcpy(First[i], "");
        Found[i] = 0;
    }
    for (i = 0; i < N; i++)
    {
        if (Found[i] == 0)
            Nullout = Find_First(First, Corresponding_Number_of_Productions, Variable_List, Variable_List[i], Found,
N);
    }
    for (i = 0; i < N; i++)
    {
        printf("First(%c) -> ", Variable_List[i]);
        for (j = 0; j < strlen(First[i]); j++)
        {
            if (Variable_List[i] == 'S')
            {
                if (First[i][j] != '#')
                {
                    printf("%c ", First[i][j]);
                }
                if (First[i][j] == '#')
                    Found[i] = -1;
            }
            else
            {
                printf("%c ", First[i][j]);
                if (First[i][j] == '#')
                    Found[i] = -1;
            }
        }
        printf("\n");
    }
    // Find Follow
    strcpy(Follow[0], "$"); // Rule 1 of Follow
    for (i = 1; i < N; i++)
    {
        strcpy(Follow[i], "");
    }
    for (Variable_Number = 0; Variable_Number < N; Variable_Number++)
    {
        // printf("Main::");
        Follow_Of(Follow, First, Variable_Number, Variable_List, N, Corresponding_Number_of_Productions, Found);
    }
    printf("\n\n");
    for (i = 0; i < N; i++)
    {
        printf("Follow(%c) -> ", Variable_List[i]);
        for (j = 0; j < strlen(Follow[i]); j++)
            printf("%c ", Follow[i][j]);
        printf("\n");
    }
}
```

File *input.txt*

```
S->ACB|bB
A->da|BC|#
B->g
C->h|#
```

Screenshot of the output



```
        S -> ACB
        S -> bB
        A -> da
        A -> BC
        A -> #
        B -> g
        C -> h
        C -> #


First(S) -> d g b h
First(A) -> d g #
First(B) -> g
First(C) -> h #


Follow(S) -> $
Follow(A) -> h g
Follow(B) -> $ h g
Follow(C) -> g h
```

LL(1) Parser

Lab 6

Aiman Fatima
20BCS008

```
#include <iostream>
#include <map>
#include <set>
#include <vector>
#include <string>
#include <deque>
#include <sstream>
#include <regex>
#include <iomanip>

using namespace std;
multimap<string, deque<string>> m;
map<string, bool> Noterm;
set<string> Term;
map<string, int> PosTerm, PosNoTerm;
map<string, vector<string>> First;
map<string, set<string>> Follow;
vector<string> string_test;
void ReadGrammar()
{
    string s, flecha;
    string k = "@", ini;
    while (getline(cin, s))
    {
        if (s == "string_test:")
            break;
        stringstream in(s);
        in >> ini >> flecha;
        if (k == "@")
            Noterm[ini] = 1;
        else
            Noterm[ini] = 0;
        deque<string> valores;
        while (in >> k)
        {
            if (k == "|")
                m.insert({ini, valores}), valores.clear();
            else
            {
                bool ok = 1;
                for (auto ch : k)
                    if (ch >= 'A' && ch <= 'Z')
                        ok = 0;
                valores.push_back(k);
                if (ok)
                    Term.insert(k);
            }
        }
        m.insert({ini, valores});
    }
    getline(cin, s);
    stringstream in(s);
    string_test = {};
    while (in >> k)
        string_test.push_back(k);
    string_test.push_back("$");
    reverse(string_test.begin(), string_test.end());
}
void Recursion()
{
    multimap<string, deque<string>> New;
    set<string> NewNoterm;
    for (auto e : Noterm)
    {
        bool ok = 0;
        for (auto val : m)
            if (val.first == e.first && e.first == val.second.front())
                ok = 1;
        if (ok)
        {
            string ini = e.first + "'";
            while (Noterm.find(ini) != Noterm.end())
                ini += "'";
            NewNoterm.insert(ini);
            deque<string> d;
            for (auto val : m)
            {
                if (val.first == e.first)
                {
                    d = val.second;
                    if (e.first != d.front())
                    {
                        d.push_back(ini);
                        New.insert({e.first, d});
                    }
                    else
                    {
                        d.pop_front();
                        d.push_back(ini);
                        New.insert({ini, d});
                    }
                }
            }
            d = {"Ɛ"};
            New.insert({ini, d});
        }
        else
        {
            for (auto val : m)
```

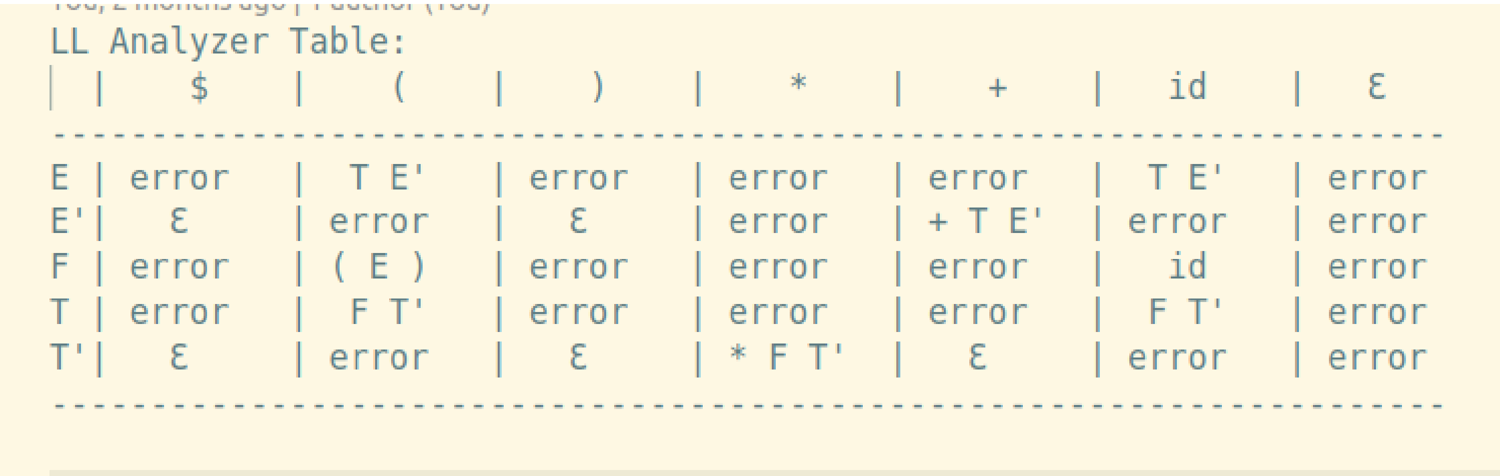
```
                if (val.first == e.first)
                    New.insert(val);
        }
    }
    for (auto e : NewNoterm)
        Noterm[e] = 0;
    m = New;
}
void Ambiguity()
{
    multimap<string, deque<string>> New;
    set<string> NewNoterm;
    for (auto e : Noterm)
    {
        map<string, int> cnt;
        for (auto val : m)
            if (val.first == e.first)
            {
                cnt[val.second.front()]++;
            }
        int mx = 0;
        string mxs;
        for (auto ele : cnt)
            if (ele.second > mx)
                mx = ele.second, mxs = ele.first;
        if (mx <= 1)
        {
            for (auto val : m)
                if (val.first == e.first)
                    New.insert(val);
            continue;
        }
        string ini = e.first + "'";
        while (Noterm.find(ini) != Noterm.end())
            ini += "'";
        NewNoterm.insert(ini);
        deque<string> d;
        for (auto val : m)
        {
            if (val.first == e.first)
            {
                d = val.second;
                if (mxs == d.front())
                {
                    d.pop_front();
                    if (!d.size())
                        d = {"Ɛ"};
                    New.insert({ini, d});
                }
                else
                    New.insert({e.first, d});
            }
        }
        d = {mxs, ini};
        New.insert({e.first, d});
    }
    for (auto e : NewNoterm)
        Noterm[e] = 0;
    m = New;
}
map<string, int> vis;
vector<string> DfsFirst(string e)
{
    vis[e] = 1;
    if (!m.count(e))
    {
        First[e] = {e};
        return {e};
    }
    vector<string> res;
    for (auto val : m)
    {
        if (val.first == e)
        {
            vector<string> ter;
            ter = DfsFirst(val.second.front());
            for (auto u : ter)
                res.push_back(u);
        }
    }
    First[e] = res;
    return res;
}
void CalcFirst()
{
    for (auto e : Term)
        First[e] = {e};
    for (auto e : Noterm)
    {
        if (!vis[e.first])
            First[e.first] = DfsFirst(e.first);
    }
}
map<string, int> visg;
void DfsFollow(string e)
{
    map<string, int> used;
    vector<string> st;
    st.push_back(e);
    while (st.size())
    {
        e = st.back();
        if (visg[e])
        {
            st.pop_back();
            if (st.size())
            {
                string v = st.back();
                for (auto ele : Follow[e])
                    Follow[v].insert(ele);
            }
            used[e] = 0;
            continue;
        }
        visg[e] = used[e] = 1;
        for (auto val : m)
        {
            deque<string> d = val.second;
            for (int i = 0; i < d.size(); i++)
            {
                if (e == d[i])
                {
                    bool ok = 0;
                    if (i + 1 < d.size())
                    {
                        vector<string> res = First[d[i + 1]];
                        for (auto v : res)
                        {
                            if (v == "Ɛ")
                                ok = 1;
                            else
                                Follow[e].insert(v);
                        }
                    }
                    if (i + 1 >= d.size() || ok)
                    {
                        if (used[val.first])
                        {
                            for (int j = 0; j < st.size(); j++)
                            {
                                if (st[j] == val.first)
                                {
                                    st.insert(st.begin() + j, e);
                                    break;
                                }
                            }
                        }
                        else
                            st.push_back(val.first);
                    }
                }
            }
        }
    }
}
void CalcFollow()
{
    for (auto e : Noterm)
    {
        if (e.second)
            Follow[e.first].insert("$");
        if (!visg[e.first])
            DfsFollow(e.first);
    }
}
deque<string> table[22][22];
void TableLL()
{
    for (auto e : First)
    {
        for (auto v : e.second)
        {
            deque<string> d;
            if (v == "Ɛ")
            {
                d = {"Ɛ"};
                for (auto val : Follow[e.first])
                {
                    table[PosNoTerm[e.first]][PosTerm[val]] = d;
                }
            }
            else
            {
                for (auto val : m)
                {
                    if (val.first == e.first)
                    {
                        bool ok = 0;
                        for (auto u : First[val.second.front()])
                        {
                            if (v == u)
                                ok = 1;
                        }
                        if (ok)
                        {
                            d = val.second;
                            table[PosNoTerm[e.first]][PosTerm[v]] = d;
                            break;
                        }
                    }
                }
            }
        }
    }
}
void valid()
{
    string ini;
    for (auto e : Noterm)
        if (e.second)
            ini = e.first;
    vector<string> pila;
```

```
pila.push_back("$");
pila.push_back(ini);
bool ok = 1;
string line;
line = "stack";
cout << line << string(20 - line.size(), ' ');
line = "string";
cout << string(30 - line.size(), ' ') << line;
line = "Action";
cout << string(40 - line.size(), ' ') << line;
cout << endl;
while (pila.size() && string_test.size())
{
    line = "";
    for (int i = pila.size() - 1; i >= 0; i--)
        line += pila[i] + " ";
    cout << line << string(20 - line.size(), ' ');
    if (!ok)
    {
        break;
    }
    line = "";
    for (int i = string_test.size() - 1; i >= 0; i--)
        line += string_test[i] + " ";
    cout << string(30 - line.size(), ' ') << line;

    auto u = pila.back();
    pila.pop_back();
    if (u == string_test.back() || u == "Ɛ")
    {
        line = u;
        if (u == string_test.back())
        {
            string_test.pop_back();
            if (!string_test.size())
                line = "Accepted";
            else
                line = "match";
        }
        cout << string(40 - line.size(), ' ') << line;
    }
    else
    {
        deque<string> d;
        d = table[PosNoTerm[u]][PosTerm[string_test.back()]];
        line = "";
        for (auto e : d)
            line += e;
        reverse(d.begin(), d.end());
        for (auto e : d)
        {
            if (e == "error")
                ok = 0;
                pila.push_back(e);
            }
            cout << string(40 - line.size(), ' ') << line;
        }
        cout << endl;
        if (!ok || !string_test.size())
            break;
    }
}
void validShowGrammar()
{
    Ambiguity();
    Recursion();
    cout << "Rules after resolving ambiguity and Left Recursion: \n";
    for (auto e : m)
    {
        cout << e.first << " -> ";
        for (auto val : e.second)
            cout << val << " ";
        cout << endl;
    }
    cout << endl;
}
void ShowFirst()
{
    cout << "First: \n";
    for (auto e : First)
    {
        cout << e.first << " -> ";
        for (auto v : First[e.first])
            cout << v << " ";
        cout << endl;
    }
    cout << endl;
}
void ShowFollow()
{
    cout << "Follow: \n";
    for (auto e : Follow)
    {
        cout << e.first << " -> ";
        for (auto v : Follow[e.first])
            cout << v << " ";
        cout << endl;
    }
    cout << endl;
}
void ShowTable()
{
    cout << "LL Analyzer Table:\n";
    int w = 9;
    cout << string(2, ' ');
    for (auto v : Term)
    {
        int l = (w - v.size()) / 2;
        int r = w - l - v.size();
        cout << "|" << string(l, ' ') << v << string(r, ' ');
    }
    cout << endl;
    for (auto v : Term)
        cout << string(w + 1, '-');
    cout << endl;
    for (auto u : First)
    {
        if (!m.count(u.first))
            continue;
        cout << setw(2) << left << u.first;
        for (auto v : Term)
        {
            string line;
            for (auto e : table[PosNoTerm[u.first]][PosTerm[v]])
                line += e + " ";
            int l = (w - (int)line.size()) / 2;
            int r = w - l - line.size();
            if (line == "Ɛ ")
                r++;
            cout << "|" << string(l, ' ') << line << string(r, ' ');
        }
        cout << endl;
    }
    for (auto v : Term)
        cout << string(w + 1, '-');
    cout << "\n\n";
}
int main()
{
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    ReadGrammar();
    // validShowGrammar();
    CalcFirst();
    CalcFollow();
    Term.insert("$");
    int cntTerm = 0;
    for (auto e : Term)
        PosTerm[e] = cntTerm++;
    int cntNoTerm = 0;
    for (auto e : Noterm)
        PosNoTerm[e.first] = cntNoTerm++;
    /*  Show First*/
    // ShowFirst();
    /*  Show Follow*/
    // ShowFollow();
    for (int i = 0; i < cntNoTerm; i++)
        for (int j = 0; j < cntTerm; j++)
        {
            table[i][j] = {"error"};
        }
    TableLL();
    // Show Table///
    ShowTable();
    // valid();
}
```

File *input.txt*

```
E -> T E'
E' -> + T E'
E' -> Ɛ
T -> F T'
T' -> * F T'
T' -> Ɛ
F -> id
F -> ( E )
string_test:
id + id * id
```

Screenshot of the output



```
You, 2 months ago | 1 author (You)
LL Analyzer Table:
  |    $    |    (    |    )    |    *    |    +    |   id    |    Ɛ
----------------------------------------------------------------------
E | error   |  T E'   | error   | error   | error   |  T E'   | error
E'|   Ɛ     | error   |   Ɛ     | error   | + T E'  | error   | error
F | error   | ( E )   | error   | error   | error   |   id    | error
T | error   |  F T'   | error   | error   | error   |  F T'   | error
T'|   Ɛ     | error   |   Ɛ     | * F T'  |   Ɛ     | error   | error
----------------------------------------------------------------------
```

LL(1) String Checking
Lab 7
Aiman Fatima
20BCS008

```
#include <iostream>
#include <map>
#include <set>
#include <vector>
#include <string>
#include <deque>
#include <sstream>
#include <regex>
#include <iomanip>

using namespace std;
multimap<string, deque<string>> m;
map<string, bool> Noterm;
set<string> Term;
map<string, int> PosTerm, PosNoTerm;
map<string, vector<string>> First;
map<string, set<string>> Follow;
vector<string> string_test;
void ReadGrammar()
{
    string s, flecha;
    string k = "@", ini;
    while (getline(cin, s))
    {
        if (s == "string_test:")
            break;
        stringstream in(s);
        in >> ini >> flecha;
        if (k == "@")
            Noterm[ini] = 1;
        else
            Noterm[ini] = 0;
        deque<string> valores;
        while (in >> k)
        {
            if (k == "|")
                m.insert({ini, valores}), valores.clear();
            else
            {
                bool ok = 1;
                for (auto ch : k)
                    if (ch >= 'A' && ch <= 'Z')
                        ok = 0;
                valores.push_back(k);
                if (ok)
                    Term.insert(k);
            }
        }
        m.insert({ini, valores});
    }
    getline(cin, s);
    stringstream in(s);
    string_test = {};
    while (in >> k)
        string_test.push_back(k);
    string_test.push_back("$");
    reverse(string_test.begin(), string_test.end());
}
void Recursion()
{
    multimap<string, deque<string>> New;
    set<string> NewNoterm;
    for (auto e : Noterm)
    {
        bool ok = 0;
        for (auto val : m)
            if (val.first == e.first && e.first == val.second.front())
                ok = 1;
        if (ok)
        {
            string ini = e.first + "";
            while (Noterm.find(ini) != Noterm.end())
                ini += "";
            NewNoterm.insert(ini);
            deque<string> d;
            for (auto val : m)
            {
                if (val.first == e.first)
                {
                    d = val.second;
                    if (e.first != d.front())
                    {
                        d.push_back(ini);
                        New.insert({e.first, d});
                    }
                    else
                    {
                        d.pop_front();
                        d.push_back(ini);
                        New.insert({ini, d});
                    }
                }
            }
            d = {"Ɛ"};
            New.insert({ini, d});
        }
        else
        {
            for (auto val : m)
```

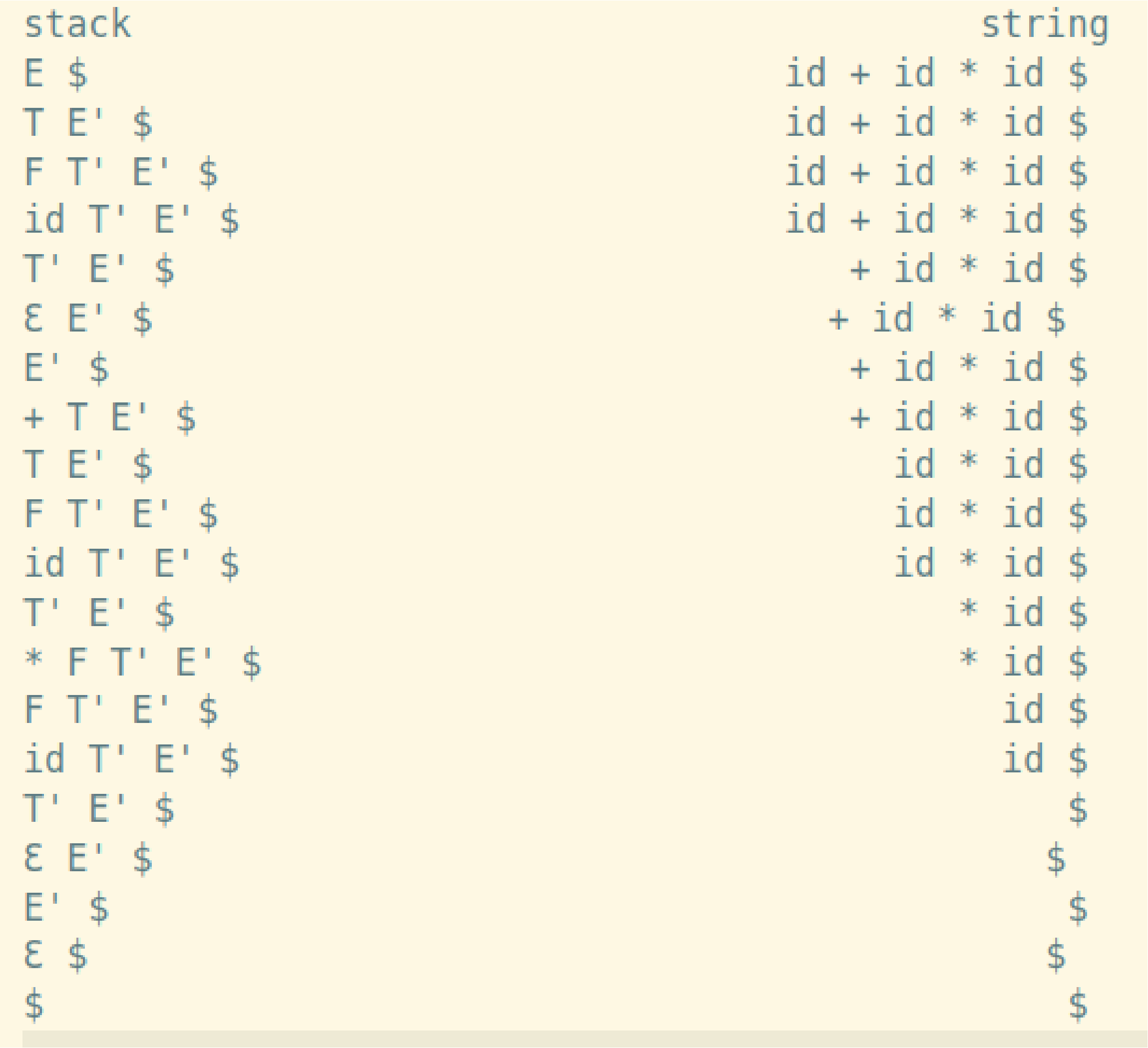
```
                if (val.first == e.first)
                    New.insert(val);
        }
    }
    for (auto e : NewNoterm)
        Noterm[e] = 0;
    m = New;
}
void Ambiguity()
{
    multimap<string, deque<string>> New;
    set<string> NewNoterm;
    for (auto e : Noterm)
    {
        map<string, int> cnt;
        for (auto val : m)
            if (val.first == e.first)
            {
                cnt[val.second.front()]++;
            }
        int mx = 0;
        string mxs;
        for (auto ele : cnt)
            if (ele.second > mx)
                mx = ele.second, mxs = ele.first;
        if (mx <= 1)
        {
            for (auto val : m)
                if (val.first == e.first)
                    New.insert(val);
            continue;
        }
        string ini = e.first + "'";
        while (Noterm.find(ini) != Noterm.end())
            ini += "'";
        NewNoterm.insert(ini);
        deque<string> d;
        for (auto val : m)
        {
            if (val.first == e.first)
            {
                d = val.second;
                if (mxs == d.front())
                {
                    d.pop_front();
                    if (!d.size())
                        d = {"Ɛ"};
                    New.insert({ini, d});
                }
                else
                    New.insert({e.first, d});
            }
        }
        d = {mxs, ini};
        New.insert({e.first, d});
    }
    for (auto e : NewNoterm)
        Noterm[e] = 0;
    m = New;
}
map<string, int> vis;
vector<string> DfsFirst(string e)
{
    vis[e] = 1;
    if (!m.count(e))
    {
        First[e] = {e};
        return {e};
    }
    vector<string> res;
    for (auto val : m)
    {
        if (val.first == e)
        {
            vector<string> ter;
            ter = DfsFirst(val.second.front());
            for (auto u : ter)
                res.push_back(u);
        }
    }
    First[e] = res;
    return res;
}
void CalcFirst()
{
    for (auto e : Term)
        First[e] = {e};
    for (auto e : Noterm)
    {
        if (!vis[e.first])
            First[e.first] = DfsFirst(e.first);
    }
}
map<string, int> visg;
void DfsFollow(string e)
{
    map<string, int> used;
    vector<string> st;
    st.push_back(e);
    while (st.size())
    {
        e = st.back();
        if (visg[e])
        {
            st.pop_back();
            if (st.size())
            {
                string v = st.back();
                for (auto ele : Follow[e])
                    Follow[v].insert(ele);
            }
            used[e] = 0;
            continue;
        }
        visg[e] = used[e] = 1;
        for (auto val : m)
        {
            deque<string> d = val.second;
            for (int i = 0; i < d.size(); i++)
            {
                if (e == d[i])
                {
                    bool ok = 0;
                    if (i + 1 < d.size())
                    {
                        vector<string> res = First[d[i + 1]];
                        for (auto v : res)
                        {
                            if (v == "Ɛ")
                                ok = 1;
                            else
                                Follow[e].insert(v);
                        }
                    }
                    if (i + 1 >= d.size() || ok)
                    {
                        if (used[val.first])
                        {
                            for (int j = 0; j < st.size(); j++)
                            {
                                if (st[j] == val.first)
                                {
                                    st.insert(st.begin() + j, e);
                                    break;
                                }
                            }
                        }
                        else
                            st.push_back(val.first);
                    }
                }
            }
        }
    }
}
void CalcFollow()
{
    for (auto e : Noterm)
    {
        if (e.second)
            Follow[e.first].insert("$");
        if (!visg[e.first])
            DfsFollow(e.first);
    }
}
deque<string> table[22][22];
void TableLL()
{
    for (auto e : First)
    {
        for (auto v : e.second)
        {
            deque<string> d;
            if (v == "Ɛ")
            {
                d = {"Ɛ"};
                for (auto val : Follow[e.first])
                {
                    table[PosNoTerm[e.first]][PosTerm[val]] = d;
                }
            }
            else
            {
                for (auto val : m)
                {
                    if (val.first == e.first)
                    {
                        bool ok = 0;
                        for (auto u : First[val.second.front()])
                        {
                            if (v == u)
                                ok = 1;
                        }
                        if (ok)
                        {
                            d = val.second;
                            table[PosNoTerm[e.first]][PosTerm[v]] = d;
                            break;
                        }
                    }
                }
            }
        }
    }
}
void valid()
{
    string ini;
    for (auto e : Noterm)
        if (e.second)
            ini = e.first;
    vector<string> pila;
```

```
pila.push_back("$");
pila.push_back(ini);
bool ok = 1;
string line;
line = "stack";
cout << line << string(20 - line.size(), ' ');
line = "string";
cout << string(30 - line.size(), ' ') << line;
line = "Action";
cout << string(40 - line.size(), ' ') << line;
cout << endl;
while (pila.size() && string_test.size())
{
    line = "";
    for (int i = pila.size() - 1; i >= 0; i--)
        line += pila[i] + " ";
    cout << line << string(20 - line.size(), ' ');
    if (!ok)
    {
        break;
    }
    line = "";
    for (int i = string_test.size() - 1; i >= 0; i--)
        line += string_test[i] + " ";
    cout << string(30 - line.size(), ' ') << line;

    auto u = pila.back();
    pila.pop_back();
    if (u == string_test.back() || u == "Ɛ")
    {
        line = u;
        if (u == string_test.back())
        {
            string_test.pop_back();
            if (!string_test.size())
                line = "Accepted";
            else
                line = "match";
        }
        cout << string(40 - line.size(), ' ') << line;
    }
    else
    {
        deque<string> d;
        d = table[PosNoTerm[u]][PosTerm[string_test.back()]];
        line = "";
        for (auto e : d)
            line += e;
        reverse(d.begin(), d.end());
        for (auto e : d)
        {
            if (e == "error")
                ok = 0;
                pila.push_back(e);
            }
            cout << string(40 - line.size(), ' ') << line;
        }
        cout << endl;
        if (!ok || !string_test.size())
            break;
    }
}
void validShowGrammar()
{
    Ambiguity();
    Recursion();
    cout << "Rules after resolving ambiguity and Left Recursion: \n";
    for (auto e : m)
    {
        cout << e.first << " -> ";
        for (auto val : e.second)
            cout << val << " ";
        cout << endl;
    }
    cout << endl;
}
void ShowFirst()
{
    cout << "First: \n";
    for (auto e : First)
    {
        cout << e.first << " -> ";
        for (auto v : First[e.first])
            cout << v << " ";
        cout << endl;
    }
    cout << endl;
}
void ShowFollow()
{
    cout << "Follow: \n";
    for (auto e : Follow)
    {
        cout << e.first << " -> ";
        for (auto v : Follow[e.first])
            cout << v << " ";
        cout << endl;
    }
    cout << endl;
}
void ShowTable()
{
    cout << "LL Analyzer Table:\n";
    int w = 9;
    cout << string(2, ' ');
    for (auto v : Term)
    {
        int l = (w - v.size()) / 2;
        int r = w - l - v.size();
        cout << "|" << string(l, ' ') << v << string(r, ' ');
    }
    cout << endl;
    for (auto v : Term)
        cout << string(w + 1, '-');
    cout << endl;
    for (auto u : First)
    {
        if (!m.count(u.first))
            continue;
        cout << setw(2) << left << u.first;
        for (auto v : Term)
        {
            string line;
            for (auto e : table[PosNoTerm[u.first]][PosTerm[v]])
                line += e + " ";
            int l = (w - (int)line.size()) / 2;
            int r = w - l - line.size();
            if (line == "Ɛ ")
                r++;
            cout << "|" << string(l, ' ') << line << string(r, ' ');
        }
        cout << endl;
    }
    for (auto v : Term)
        cout << string(w + 1, '-');
    cout << "\n\n";
}
int main()
{
    freopen("string_check_input.txt", "r", stdin);
    freopen("string_check_output.txt", "w", stdout);
    ReadGrammar();
    // validShowGrammar();
    CalcFirst();
    CalcFollow();
    Term.insert("$");
    int cntTerm = 0;
    for (auto e : Term)
        PosTerm[e] = cntTerm++;
    int cntNoTerm = 0;
    for (auto e : Noterm)
        PosNoTerm[e.first] = cntNoTerm++;
    /*  Show First*/
    // ShowFirst();
    /*  Show Follow*/
    // ShowFollow();
    for (int i = 0; i < cntNoTerm; i++)
        for (int j = 0; j < cntTerm; j++)
        {
            table[i][j] = {"error"};
        }
    TableLL();
    // Show Table///
    // ShowTable();
    valid();
}
```

File *input.txt*

```
E -> T E'
E' -> + T E'
E' -> Ɛ
T -> F T'
T' -> * F T'
T' -> Ɛ
F -> id
F -> ( E )
string_test:
id + id * id
```

Screenshot of the output



```
stack                                       string
E $                                id + id * id $
T E' $                             id + id * id $
F T' E' $                          id + id * id $
id T' E' $                         id + id * id $
T' E' $                               + id * id $
Ɛ E' $                               + id * id $
E' $                                  + id * id $
+ T E' $                              + id * id $
T E' $                                  id * id $
F T' E' $                               id * id $
id T' E' $                              id * id $
T' E' $                                    * id $
* F T' E' $                                * id $
F T' E' $                                    id $
id T' E' $                                   id $
T' E' $                                         $
Ɛ E' $                                         $
E' $                                            $
Ɛ $                                            $
$                                               $
```