**Name:** Bhavya Patel

**Roll No:** 20BCE198

**Course:** Compiler construction

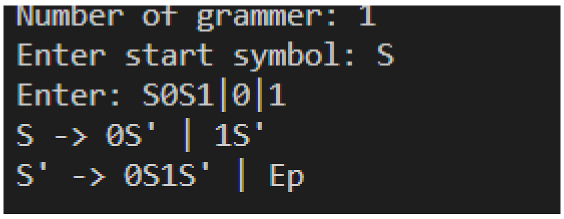
**Practical No:** 4

**Aim: To Implement Left Recursion derivation removal algorithm** : Eliminate direct and indirect Left recursion from given grammar for LL(1) parser

**Methodology followed: direct lr**

```
#include<bits/stdc++.h>

using namespace std;

string grammar;
char startSymbol;

vector<vector<string>> isLeftRecursionExist(string &s){
    vector<vector<string>>v(2);
    for(int i = 0; i < s.length(); ){
        string temp = "";
        while(i < s.length() && s[i] != '|'){
            temp+=s[i++];
        }
        i++;
        if(temp[0] == startSymbol){
            v[0].push_back(temp);
        }
        else{
            v[1].push_back(temp);
        }
    }
    return v;
}

void solveLeftRecursion(){
    auto v = isLeftRecursionExist(grammar);
    if(v[0].size() == 0){
        cout << startSymbol << " -> " << grammar;
        return;
    }
    for(int i=0; i<v[1].size(); i++){
        v[1][i] += "S'";
    }
    for(int i=0; i<v[0].size(); i++){
        v[0][i] = v[0][i].substr(1, v[0][i].length()-1);
        v[0][i] += "S'";
    }
    cout << startSymbol << " -> ";
    for(int i = 0; i < v[1].size(); i++){
        cout << v[1][i];
        if(i != v[1].size()-1){
            cout << " | ";
        }
    }
    cout << '\n';
    cout << "S' -> ";
    for(int i = 0; i < v[0].size(); i++){
        cout << v[0][i] << " | ";
    }

    cout << "Ep\n";
}

int main(){
    cout << "Number of grammer: ";
    int n;
    cin >> n;
    while(n--){
        cout << "Enter start symbol: ";
        cin >> startSymbol;
        cout << "Enter: ";
        cin  >> grammar;
        solveLeftRecursion();
        cout << '\n';
    }
}
```

**Output:**



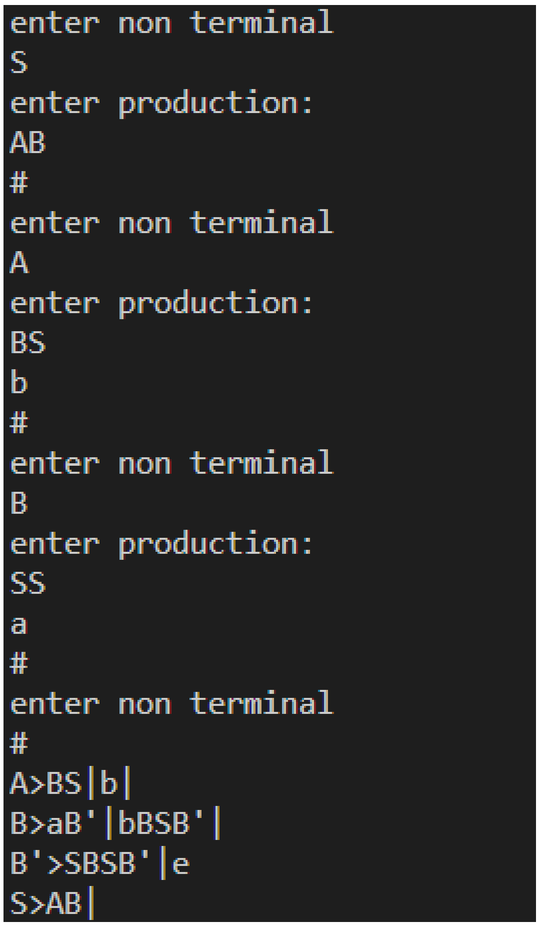
```
Number of grammer: 1
Enter start symbol: S
Enter: S0S1|0|1
S -> 0S' | 1S'
S' -> 0S1S' | Ep
```

**Methodology followed: indirect lr**

```
#include <bits/stdc++.h>
using namespace std;

map<char, vector<string>> production;

map<char, vector<char>> mp;
```

```cpp
map<char, bool> vis;

map<char, bool> pathvis;

void dfs(char cur)
{
    vis[cur] = true;
    pathvis[cur] = true;

    for (auto child : production[cur])
    {
        if (child[0] >= 'A' && child[0] <= 'Z')
        {
            char ch = child[0];
            if (!vis[ch] && ch != cur)
            {
                dfs(ch);
            }
            else if (pathvis[ch] && ch != cur)
            {
                // cycle

                // ch = S,cur=B

                // child = Sd

                production[cur].erase(remove(production[cur].begin(),
production[cur].end(), child), production[cur].end());

                child = child.substr(1, child.length() - 1);
                // cout<<"child is: "<<child<<endl;
                //  production[cur].erase(child);
                // v.erase(std::remove(v.begin(), v.end(), item), v.end());

                // for(auto it : production[cur])
                // {
                //     cout<<"yes "<<it<<endl;
                // }

                for (auto it : production[ch])
                {
                    string temp = it + child;
                    // cout<<"temp is: "<<temp<<endl;
                    production[cur].push_back(temp);
                }
            }
        }
    }

    pathvis[cur] = false;
}

int main()
{
    while (true)
    {
        cout << "enter non terminal" << endl;
        char nt;
        cin >> nt;

        if (nt == '#')
            break;
        cout << "enter production: " << endl;
        while (true)
        {
            string s;
            cin >> s;

            if (s == "#")
                break;

            production[nt].push_back(s);
        }
    }

    dfs('S');

    // for(auto it : production)
    // {
    //     cout<<it.first<<": ";
    //     for(auto i : it.second)
    //     {
    //         cout<<i<<" ";
    //     }
    //     cout<<endl;
    // }

    for (auto cur : production)
    {
        char nt = cur.first;

        vector<string> alpha;
        vector<string> beta;
        bool flag = false;
        for (auto p : cur.second)
        {
            if (p[0] == nt)
            {
                // cout<<p[0]<<" "<<nt<<endl;
                alpha.push_back(p.substr(1, p.length() - 1));
                flag = true;
            }
            else
            {
                beta.push_back(p);
            }
        }

        if (flag)
        {
            cout << nt << ">";
            for (auto it : beta)
            {
                if (it == "e")
                {
                    cout << nt << "'|";
                }
                else
                    cout << it << nt << "'|";
            }
            cout << endl;

            cout << nt << "'>";
            for (auto it : alpha)
            {
                cout << it << nt << "'|";
            }
            cout << "e" << endl;
        }
        else
        {
            cout << nt << ">";
            for (auto it : cur.second)
            {
                cout << it << "|";
            }
            cout << endl;
        }
    }
}
```

**Output:**


```
enter non terminal
S
enter production:
AB
#
enter non terminal
A
enter production:
BS
b
#
enter non terminal
B
enter production:
SS
a
#
enter non terminal
#
A>BS|b|
B>aB'|bBSB'|
B'>SBSB'|e
S>AB|
```