2CS701- Compiler Construction

Practical 4

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

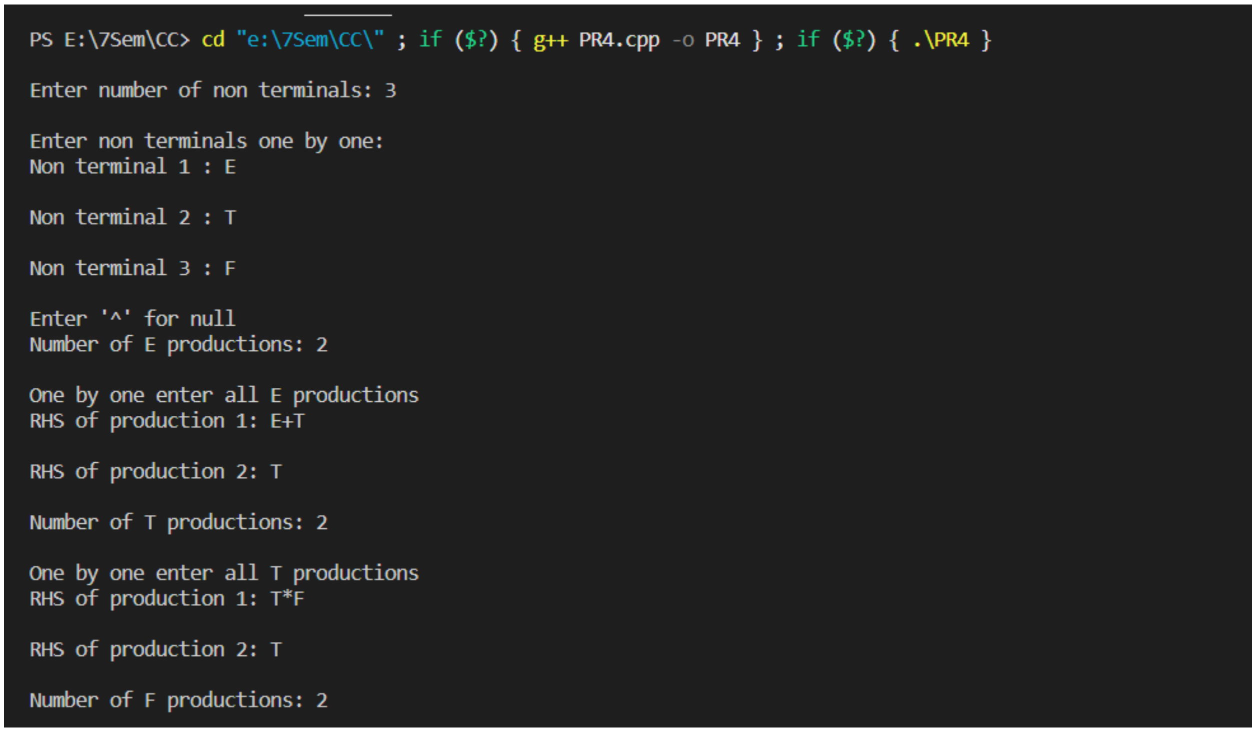
Author: Darshil Maru 20BCE514

Guide: Prof. Deepti Saraswat

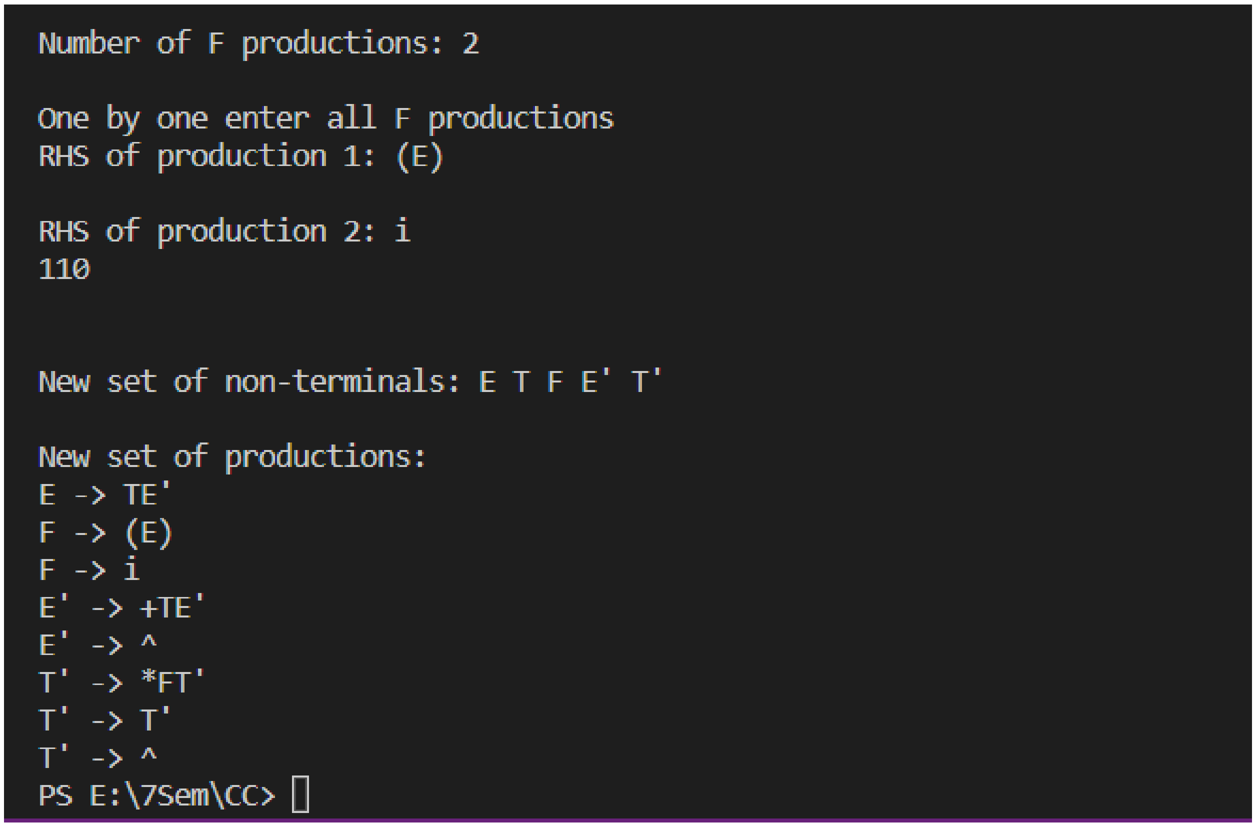
Code:

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int main()
{
    int n;
    cout << "\nEnter number of non terminals: ";
    cin >> n;
    cout << "\nEnter non terminals one by one: ";
    int i;
    vector<string> nonter(n);
    vector<int> leftrecr(n, 0);
    for (i = 0; i < n; ++i)
    {
        cout << "\nNon terminal " << i + 1 << " : ";
        cin >> nonter[i];
    }
    vector<vector<string>> prod;
    cout << "\nEnter '^' for null";
    for (i = 0; i < n; ++i)
    {
        cout << "\nNumber of " << nonter[i] << " productions: ";
        int k;
        cin >> k;
        int j;
        cout << "\nOne by one enter all " << nonter[i] << " productions";
        vector<string> temp(k);
        for (j = 0; j < k; ++j)
        {
            cout << "\nRHS of production " << j + 1 << ": ";
            string abc;
            cin >> abc;
            temp[j] = abc;
            if (nonter[i].length() <= abc.length() &&
nonter[i].compare(abc.substr(0, nonter[i].length())) == 0)
                leftrecr[i] = 1;
        }
        prod.push_back(temp);
    }
    for (i = 0; i < n; ++i)
    {
        cout << leftrecr[i];
    }
    for (i = 0; i < n; ++i)
    {
        if (leftrecr[i] == 0)
            continue;
        int j;
        nonter.push_back(nonter[i] + "'");
        vector<string> temp;
        for (j = 0; j < prod[i].size(); ++j)
        {
            if (nonter[i].length() <= prod[i][j].length() &&
nonter[i].compare(prod[i][j].substr(0, nonter[i].length())) == 0)
            {
                string abc = prod[i][j].substr(nonter[i].length(),
prod[i][j].length() - nonter[i].length()) + nonter[i] + "'";
                temp.push_back(abc);
                prod[i].erase(prod[i].begin() + j);
                --j;
            }
            else
            {
                prod[i][j] += nonter[i] + "'";
            }
        }
        temp.push_back("^");
        prod.push_back(temp);
    }
    cout << "\n\n";
    cout << "\nNew set of non-terminals: ";
    for (i = 0; i < nonter.size(); ++i)
        cout << nonter[i] << " ";
    cout << "\n\nNew set of productions: ";
    for (i = 0; i < nonter.size(); ++i)
    {
        int j;
        for (j = 0; j < prod[i].size(); ++j)
        {
            cout << "\n"
                 << nonter[i] << " -> " << prod[i][j];
        }
    }
    return 0;
}
```

Output:



```
PS E:\7Sem\CC> cd "e:\7Sem\CC\" ; if ($?) { g++ PR4.cpp -o PR4 } ; if ($?) { .\PR4 }

Enter number of non terminals: 3

Enter non terminals one by one:
Non terminal 1 : E

Non terminal 2 : T

Non terminal 3 : F

Enter '^' for null
Number of E productions: 2

One by one enter all E productions
RHS of production 1: E+T

RHS of production 2: T

Number of T productions: 2

One by one enter all T productions
RHS of production 1: T*F

RHS of production 2: T

Number of F productions: 2
```



```
Number of F productions: 2

One by one enter all F productions
RHS of production 1: (E)

RHS of production 2: i
110


New set of non-terminals: E T F E' T'

New set of productions:
E -> TE'
F -> (E)
F -> i
E' -> +TE'
E' -> ^
T' -> *FT'
T' -> T'
T' -> ^
PS E:\7Sem\CC> 
```