Exp 5 - ELIMINATION OF LEFT RECURSION & LEFT FACTORING

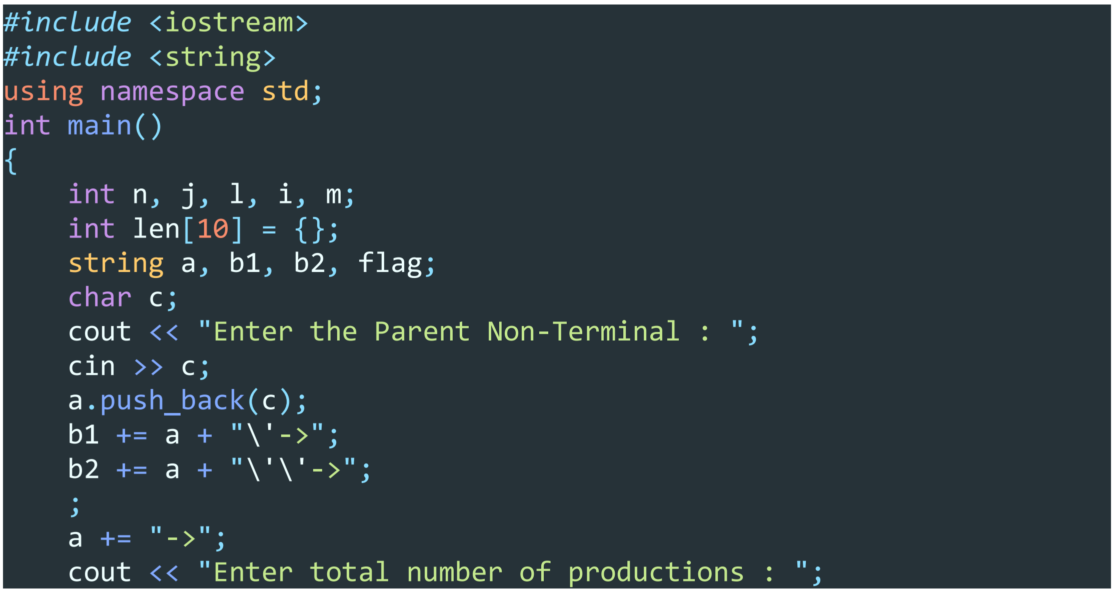
Aim: To write a program for elimination of left recursion and left factoring.

LEFT FACTORING:

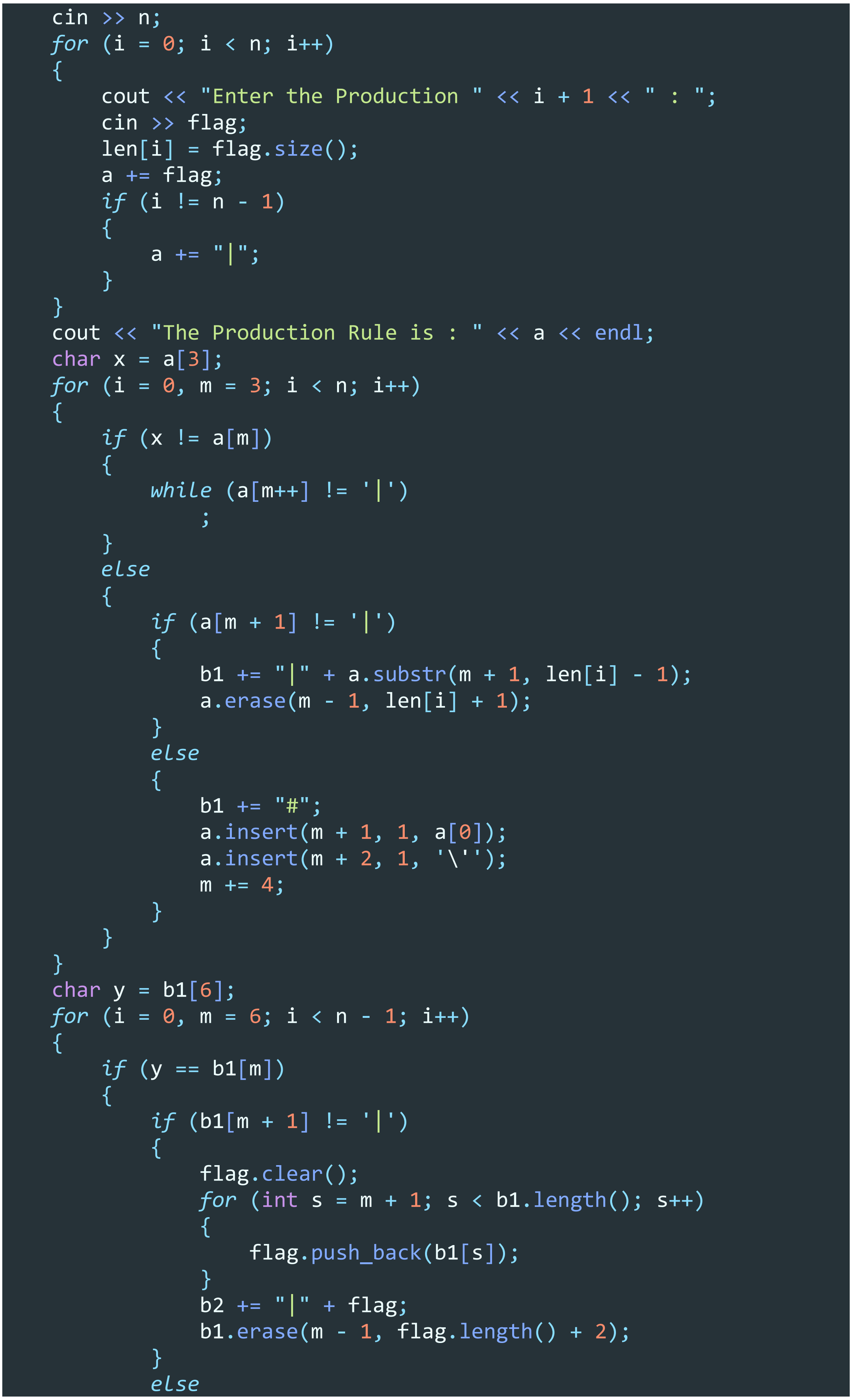
Algorithm:

1. Start
2. For the common prefixes, we make only 1 production.
3. So, here the common prefix can be a terminal or a non-terminal or it can be a combination of both.
4. With the help of new productions, the rest derivation is added.
5. Stop

Program:

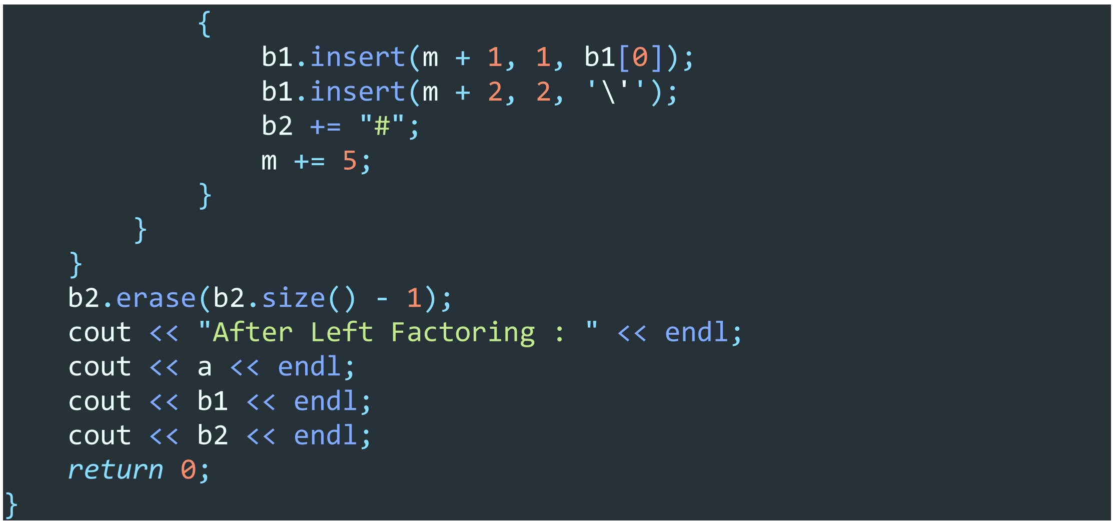


```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int n, j, l, i, m;
    int len[10] = {};
    string a, b1, b2, flag;
    char c;
    cout << "Enter the Parent Non-Terminal : ";
    cin >> c;
    a.push_back(c);
    b1 += a + "\'->";
    b2 += a + "\'\'->";
    ;
    a += "->";
    cout << "Enter total number of productions : ";
```



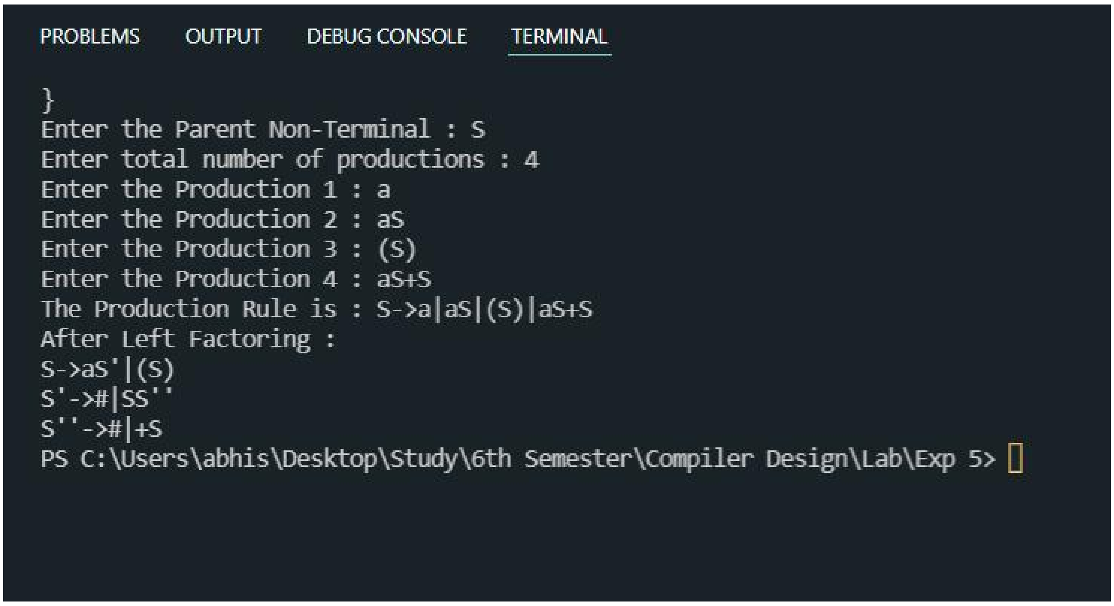
```
    cin >> n;
    for (i = 0; i < n; i++)
    {
        cout << "Enter the Production " << i + 1 << " : ";
        cin >> flag;
        len[i] = flag.size();
        a += flag;
        if (i != n - 1)
        {
            a += "|";
        }
    }
    cout << "The Production Rule is : " << a << endl;
    char x = a[3];
    for (i = 0, m = 3; i < n; i++)
    {
        if (x != a[m])
        {
            while (a[m++] != '|')
                ;
        }
        else
        {
            if (a[m + 1] != '|')
            {
                b1 += "|" + a.substr(m + 1, len[i] - 1);
                a.erase(m - 1, len[i] + 1);
            }
            else
            {
                b1 += "#";
                a.insert(m + 1, 1, a[0]);
                a.insert(m + 2, 1, '\'');
                m += 4;
            }
        }
    }
    char y = b1[6];
    for (i = 0, m = 6; i < n - 1; i++)
    {
        if (y == b1[m])
        {
            if (b1[m + 1] != '|')
            {
                flag.clear();
                for (int s = m + 1; s < b1.length(); s++)
                {
                    flag.push_back(b1[s]);
                }
                b2 += "|" + flag;
                b1.erase(m - 1, flag.length() + 2);
            }
            else
```

Exp 5 - ELIMINATION OF LEFT RECURSION & LEFT FACTORING
RA1911003010143
Abhishek Kumar



```
            {
                b1.insert(m + 1, 1, b1[0]);
                b1.insert(m + 2, 2, '\'');
                b2 += "#";
                m += 5;
            }
        }
    }
    b2.erase(b2.size() - 1);
    cout << "After Left Factoring : " << endl;
    cout << a << endl;
    cout << b1 << endl;
    cout << b2 << endl;
    return 0;
}
```

Output:



```
}
Enter the Parent Non-Terminal : S
Enter total number of productions : 4
Enter the Production 1 : a
Enter the Production 2 : aS
Enter the Production 3 : (S)
Enter the Production 4 : aS+S
The Production Rule is : S->a|aS|(S)|aS+S
After Left Factoring :
S->aS'|(S)
S'->#|SS''
S''->#|+S
PS C:\Users\abhis\Desktop\Study\6th Semester\Compiler Design\Lab\Exp 5> 
```

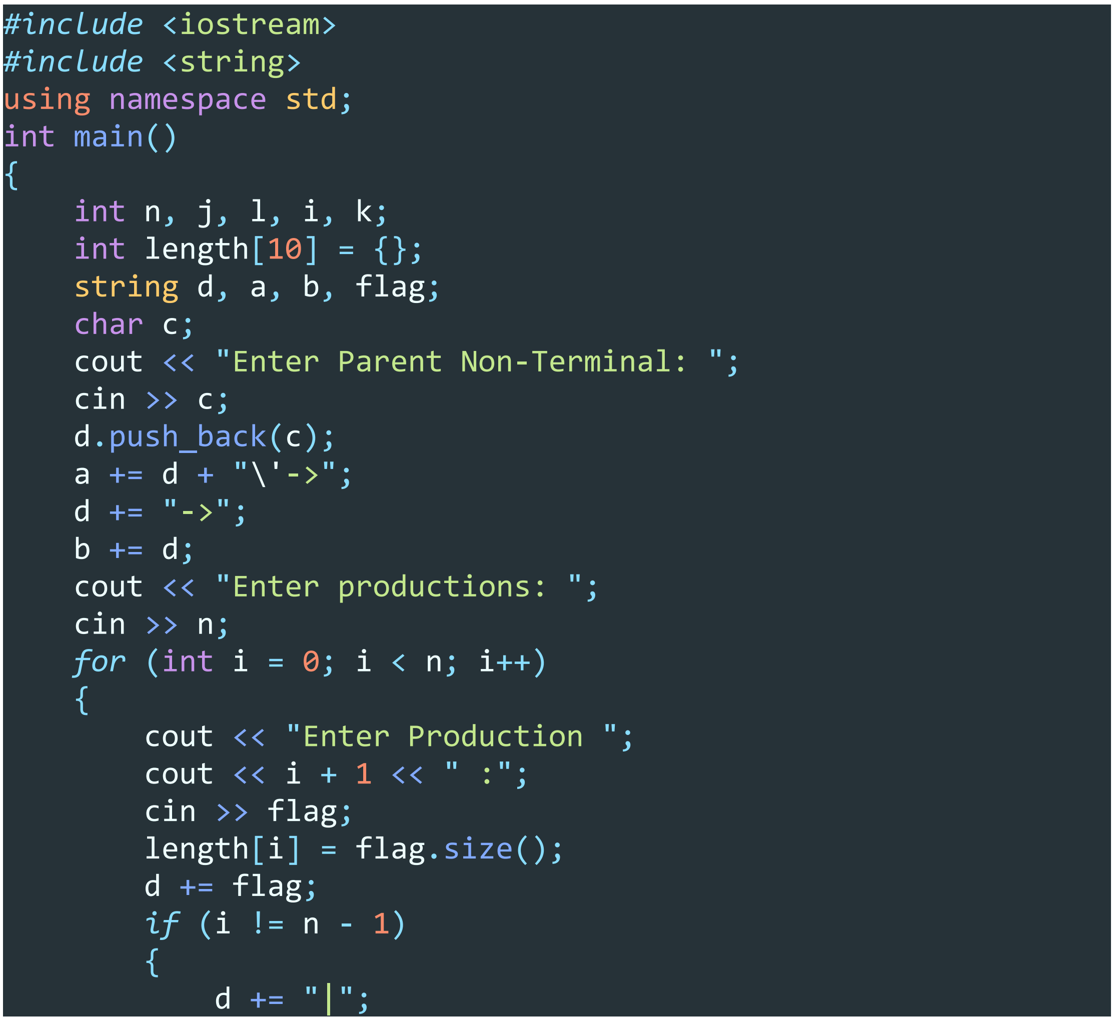
Exp 5 - ELIMINATION OF LEFT RECURSION & LEFT FACTORING
RA1911003010143
Abhishek Kumar

LEFT FACTORING:

Algorithm:

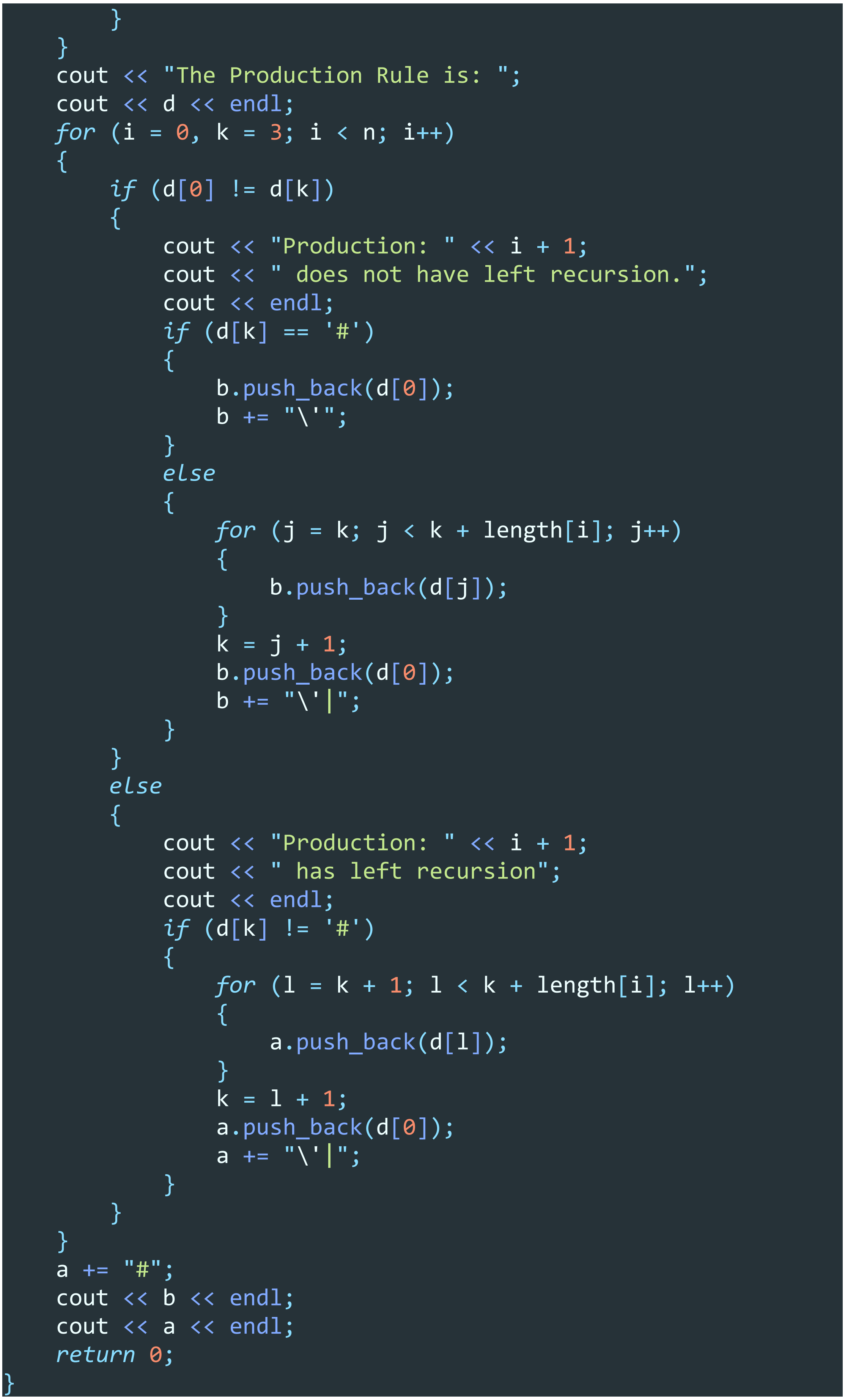
1. Start
2. Consider, S->S+A, E=a, T=b
3. In it's parse tree S will grow left indefinitely, so to remove it S=Sa | b
4. We take as S=bS', S'= aS'|S
5. Stop

Program:



```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int n, j, l, i, k;
    int length[10] = {};
    string d, a, b, flag;
    char c;
    cout << "Enter Parent Non-Terminal: ";
    cin >> c;
    d.push_back(c);
    a += d + "\'->";
    d += "->";
    b += d;
    cout << "Enter productions: ";
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cout << "Enter Production ";
        cout << i + 1 << " :";
        cin >> flag;
        length[i] = flag.size();
        d += flag;
        if (i != n - 1)
        {
            d += "|";
```

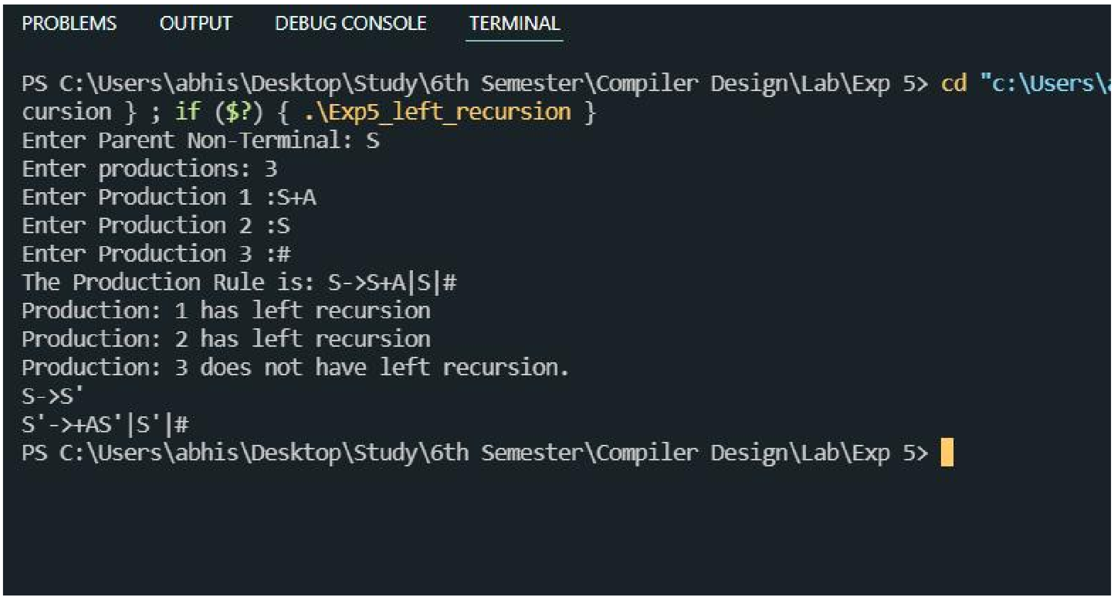
Exp 5 - ELIMINATION OF LEFT RECURSION & LEFT FACTORING
RA1911003010143
Abhishek Kumar



```
        }
    }
    cout << "The Production Rule is: ";
    cout << d << endl;
    for (i = 0, k = 3; i < n; i++)
    {
        if (d[0] != d[k])
        {
            cout << "Production: " << i + 1;
            cout << " does not have left recursion.";
            cout << endl;
            if (d[k] == '#')
            {
                b.push_back(d[0]);
                b += "\'";
            }
            else
            {
                for (j = k; j < k + length[i]; j++)
                {
                    b.push_back(d[j]);
                }
                k = j + 1;
                b.push_back(d[0]);
                b += "\'|";
            }
        }
        else
        {
            cout << "Production: " << i + 1;
            cout << " has left recursion";
            cout << endl;
            if (d[k] != '#')
            {
                for (l = k + 1; l < k + length[i]; l++)
                {
                    a.push_back(d[l]);
                }
                k = l + 1;
                a.push_back(d[0]);
                a += "\'|";
            }
        }
    }
    a += "#";
    cout << b << endl;
    cout << a << endl;
    return 0;
}
```

Exp 5 - ELIMINATION OF LEFT RECURSION & LEFT FACTORING
RA1911003010143
Abhishek Kumar

Output:



```
PS C:\Users\abhis\Desktop\Study\6th Semester\Compiler Design\Lab\Exp 5> cd "c:\Users\a
cursion } ; if ($?) { .\Exp5_left_recursion }
Enter Parent Non-Terminal: S
Enter productions: 3
Enter Production 1 :S+A
Enter Production 2 :S
Enter Production 3 :#
The Production Rule is: S->S+A|S|#
Production: 1 has left recursion
Production: 2 has left recursion
Production: 3 does not have left recursion.
S->S'
S'->+AS'|S'|#
PS C:\Users\abhis\Desktop\Study\6th Semester\Compiler Design\Lab\Exp 5>
```

Result: Elimination of Left factoring and Left recursion was successfully performed using C++.