

ABHINAV RANJAN  
RA1911003010003  
CSE A1 SECTION  
SRMIST , KTR

## **COMPILER DESIGN LAB**

### ***EXP 4 -ELIMINATION OF LEFT RECURSION AND LEFT FACTORING***

#### **AIM :**

To create a program which identifies the given grammar as left recursive or not and then perform elimination of the left recursion and left factoring

#### **REQUIREMENTS :**

1. Knowledge of the concepts left and right recursive grammar
2. Knowledge of the methodology to eliminate left recursion and left factoring
3. Online GDB compiler to execute and implement code

#### **THEORY :**

##### **Left Recursive Grammar:**

- A production of grammar is said to have left recursion if the leftmost variable of its RHS is the same as the variable of its LHS.

- A grammar containing a production having left recursion is called as Left Recursive Grammar
- Left recursion is considered to be a problematic situation for Top down parsers.
- Therefore, left recursion has to be eliminated from the grammar.

## Elimination of Left Recursion

Left recursion is eliminated by converting the grammar into a right recursive grammar.

If we have the left-recursive pair of productions-

$$A \rightarrow A\alpha / \beta$$

(Left Recursive Grammar)

where  $\beta$  does not begin with an A.

Then, we can eliminate left recursion by replacing the pair of productions with-

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' / \epsilon$$

(Right Recursive Grammar)

This right recursive grammar functions the same as left recursive grammar.

## Left Factoring :

Left factoring is used so that it makes the grammar useful for top-down parsers. Here we take out the left

factor that appears in 2 productions of the same non-terminal. So, it is done to avoid backtracking by the parser.

### **ALGORITHM :**

1. We declare d,a,b strings and a flag bit
2. Store the parent non-terminal in c var
3. Push c into d string such that  $E \rightarrow$  left production
4. Take the no of productions in. For example take 2
5. And for loop till iterator = 2
6. And take each production and add it to d string var such that after each production it will be  $E \rightarrow E+T$

To detect if the production is left recursive or not:

7. If( $d[0] \neq d[k]$ ) it doesn't have left recursion because  $E \rightarrow E$  here k is the last var E and for left recursion the left variable should be the parent non-terminal.
8. If they have left recursion then push it to A vector array according to the laws of left recursion ie  $TE'$  where  $E' \rightarrow +TE' | \#$  AND  $E \rightarrow TE'$
9. Print the two

### **SOURCE CODE :**

#### **4a - DETECTION AND ELIMINATION OF LEFT RECURSIVE GRAMMAR :**

```
#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 += "|";
        }
    }
    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;

```

```
}
```

#### 4b - LEFT FACTORING ELIMINATION

```
#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
        {

```

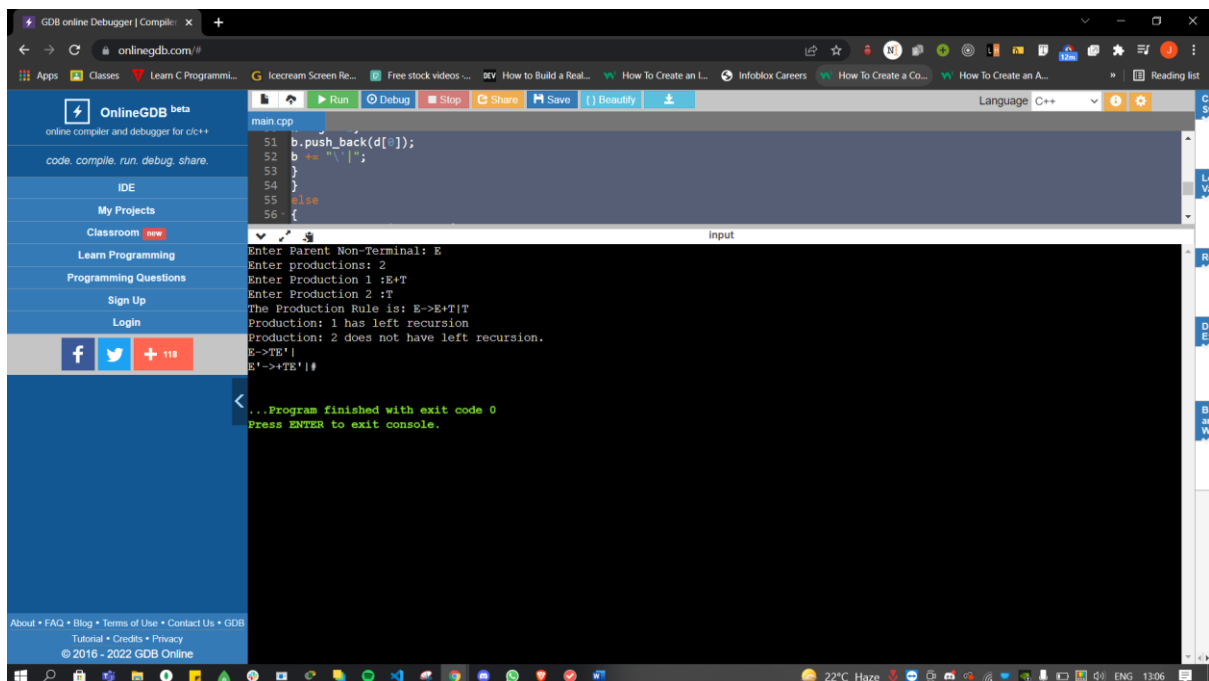
```

        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;
}

```

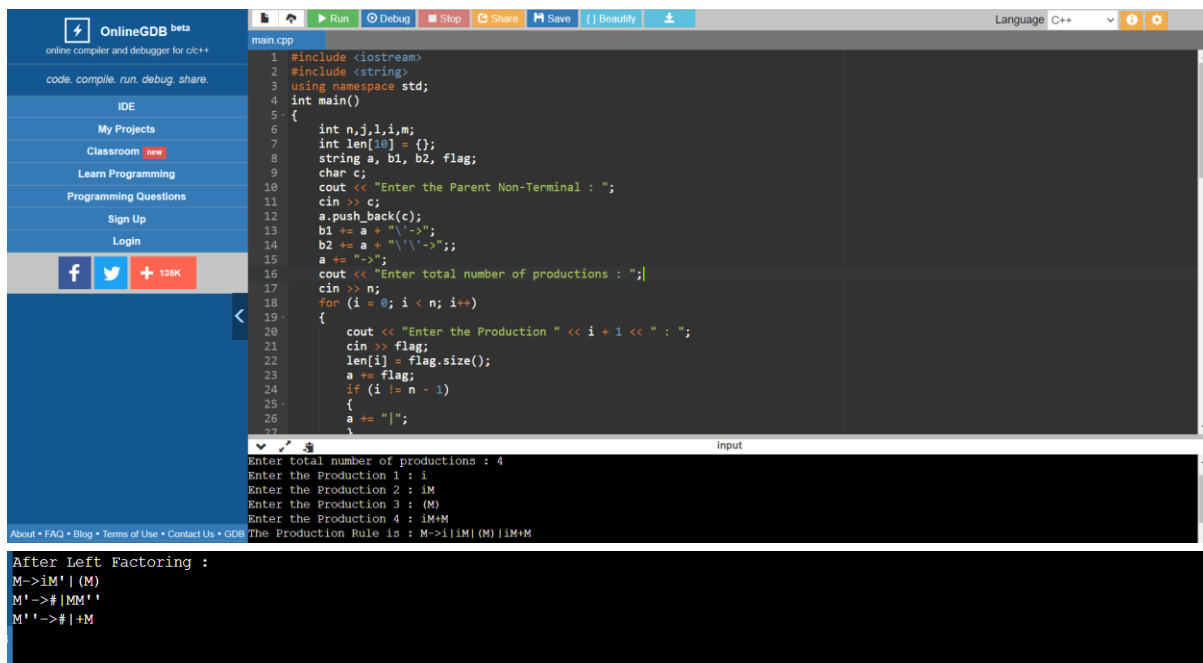
## SCREENSHOT OF OUTPUT :

### 4a -DETECTION AND ELIMINATION OF LEFT RECURSIVE GRAMMAR





## 4b - ELIMINATION OF LEFT FACTORING



The screenshot displays the OnlineGDB IDE interface. The main editor shows a C++ program that prompts the user to enter the total number of productions, followed by each production rule. The program then processes these rules to identify and eliminate left recursion and left factoring. The output window shows the results of this process.

```
main.cpp
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main()
5 {
6     int n,j,l,i,m;
7     int len[10] = {};
8     string a, b1, b2, flag;
9     char c;
10    cout << "Enter the Parent Non-Terminal : ";
11    cin >> c;
12    a.push_back(c);
13    b1 += a + "\\->";
14    b2 += a + "\\'->";
15    a += "->";
16    cout << "Enter total number of productions : ";
17    cin >> n;
18    for (i = 0; i < n; i++)
19    {
20        cout << "Enter the Production " << i + 1 << " : ";
21        cin >> flag;
22        len[i] = flag.size();
23        a += flag;
24        if (i != n - 1)
25        {
26            a += "|";
27        }
28    }
29    cout << "The Production Rule is : M->i|M| (M) iM+M";
30}
```

Input

```
Enter total number of productions : 4
Enter the Production 1 : i
Enter the Production 2 : iM
Enter the Production 3 : (M)
Enter the Production 4 : iM+M
The Production Rule is : M->i|M| (M) iM+M
```

After Left Factoring :

```
M->iM'| (M)
M' ->#|MM''
M''->#|+M
```

### OBSERVATION :

The grammars given as input were accurately detected as left recursive or not . Those identified as left recursive underwent left recursion elimination and left factoring elimination .The output was verified.

### RESULT :

Thus we have successfully implemented a program to detect and eliminate left recursive grammar.