

**Premier University Chittagong**

Course Title : Compiler Construction Laboratory

Course Code : CSE 454

Department : Computer Science and Engineering

Report No : 05

Report Name : Write a program to check and

remove the left factoring.

Date of Submission : 27/09/2020

Marks

Name : Joyanta Dutta

ID : 1402710200740

Sec : C8A

Batch : 27th

**Objectives:** To check and remove the left factoring.

**Introduction:** Left factoring is taking out the regular left factor that shows up in two productions of the equivalent non-terminal. It is done to keep away from back-tracking by the parser.

Considering an example below

A -> αP/αQ

where A, P, Q are non-terminals and α is a common factor

after left factoring the grammar will be:

A -> αS'

S' -> P/Q

**Left Factoring** is basically a grammar transformation technique. It has "factoring out" prefixes which are common to two or more productions or in other words Left factoring is a process of transformation, in which the grammar turns from a left-recursive form to an equivalent non-left-recursive form.

**Software requirements:**

1. Laptop
2. IDE (CodeBlocks)
3. CodeBlocks

**Methodology:**

Enter the number of production

Set the word in the row of parent non terminal (`) then |

For i = 0 to n-1

Enter the production

Expected Output

For i = 0 to n-1

If common prefix

NO

YES

Set the prefix with the (parent non terminal)` then |

Fig 1: Flowchart for the code of checking and removing the left factoring.

**Description:** First of all, took the products, then checked if there are common prefix or not, if those were, then uncommon all were removed from this row. Then the common prefix’s were been set up with the parent non-terminal(with `) in the row of parent non-terminal. Then those uncommon words were been set up in the row of parent non-terminal(with `).

**Source Code:**

#include <bits/stdc++.h>

using namespace std;

int main()

{

long long int i,j,k,l,n,m=9999999999,mini,ma=0;

string s[100],st,ch,sc="",result,fs,maxi,rs="";

vector<string>ss;

vector<string>sp;

cin>>n;

for(i=1;i<=n;i++)

{

cin>>s[i];

}

for(i=1;i<=n;i++)

{

st=s[i];

sc="";

for(j=0;j<st.length();j++)

{

if(i==1)

{

fs=st[0];

}

if(st[j]=='=')

{

l=j;

}

}

if(i==1)

{

for(k=l+1;k<st.length();k++)

{

if(st[k]=='|')

{

ss.push\_back(sc);

sc="";

}

if(st[k]!='|')

{

ch=st[k];

sc=sc+ch;

}

}

ss.push\_back(sc);

}

}

for(k=0;k<ss.size();k++)

{

mini=ss[k].size();

m=min(m,mini);

maxi=ss[k];

}

for (int i=0; i<m; i++)

{

char current = ss[0][i];

for (int j=1 ; j<ss.size(); j++)

{

if (ss[j][i] != current)

{

break;

}

result.push\_back(current);

}

}

for(j=0;j<ss.size();j++)

{

maxi=ss[j];

for(k=0;k<maxi.length();k++)

{

if(k>=result.length())

{

rs=rs+maxi[k];

}

}

if(j!=ss.size()-1)

{

rs=rs+'|';

}

}

cout<<endl;

cout<<"Result :"<<endl;

cout<<endl;

cout<<fs<<"="<<result<<fs<<"'"<<endl;

cout<<fs<<"'"<<"="<<rs<<endl;

for(i=2;i<=n;i++)

{

cout<<s[i]<<endl;

}

return 0;

}

**Output:**

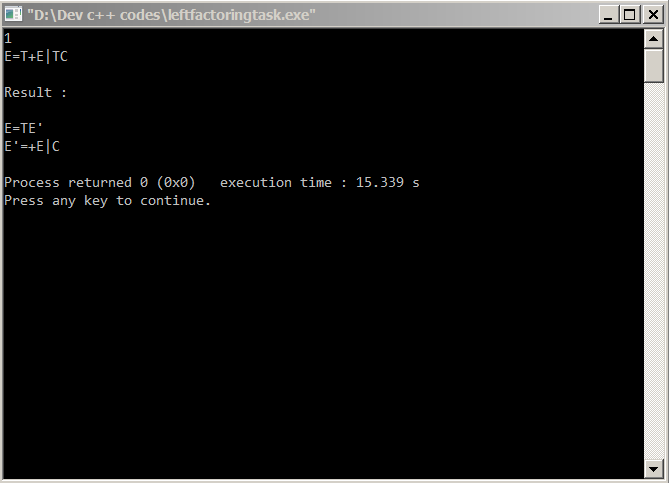


Fig 2: Output of the program

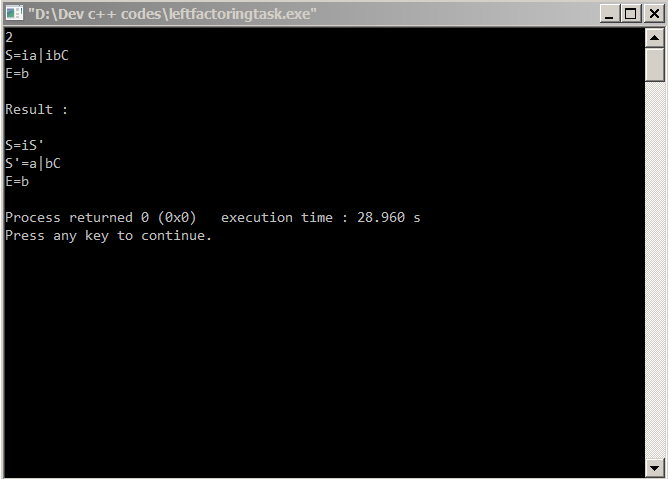


Fig 3: 2nd output of the program

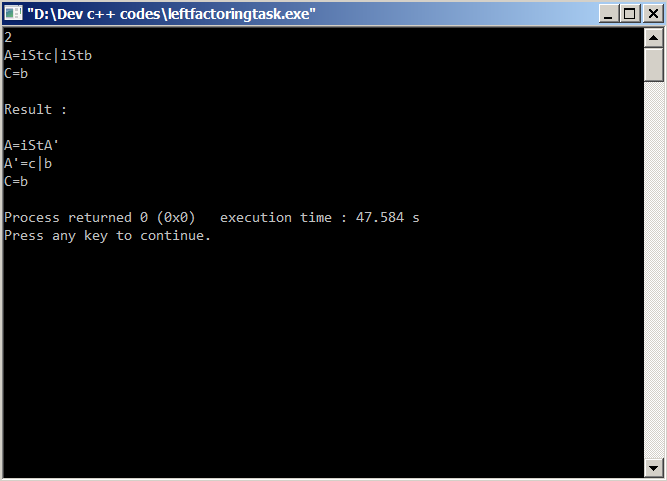
****

Fig 4: 3rd output of the program

**Discussion:** Here, I have learnt, it is a grammar transformation that is useful for producing a grammar suitable for predictive or top down parsing. In the other command, it’s a process of transformation, turning the grammar from a left-recursive form to an equivalent non-left-recursive form.