Date

# EXPERIMENT - 4

08-02-23 **ELIMINATION OF LEFT RECURSION AND LEFT FACTORING**

Faculty : Mr. M. Anand DO : 1

**AIM:** To remove left recursion and left factoring and execute the program.

**ALGORITHM:**

1. Elimination of left recursion.
2. We need to check if the grammar contains left recursion. If present we need to seperate and start working.
   [ $S \rightarrow Sa | Sb | c | d$ ]
3. Introduce new non-terminal and write it at the least of every terminal. We produce new non-terminal $S'$ & write production as,
   $S \rightarrow cS' | dS'$
4. So far equivalent production is $S \rightarrow cS' | S'$
   $S' \rightarrow ? | S' | bS'$
5. To remove left factoring
   - for common prefix use only production.
   Common prefix can be terminal, non terminal or combination of both.
6. The result that is obtained after this process of left recursion and left factoring is known as left factor grammar.

→ Left Recursion

$A \rightarrow A\alpha | \beta$

↓ Removal

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' | \varepsilon$

→ Left factoring

$A \rightarrow \alpha\beta_1 | \alpha\beta_2$

↓ Removal

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 | \beta_2$

Output

A → And | a

Enter number of non terminals : 1
Enter non terminals one by one :
Non-terminal 1 : A
Enter 'esp' for null
Number of A productions : 2
One by one enter all A productions
RHS of production 1 : And
RHS of production 2 : a
|

New set of non-terminals : A A'

New set of productions :

A → aA'

A' → nAA'

A' → esp

o/p

Input & Output : L → L;S|S

Enter no of non terminals : 2
Non terminal 1 : L
Number of L productions : 2
RHS of production 1 : L;S
RHS of production 2 : S

o/p

New set of non terminals : L S L'
New set of productions : L → SL'

7. Algorithm for left factoring given grammar:

- For each nonterminal A, find longest prefix $\alpha$ common to 2 or more of its alternative.
- If $\alpha \neq \epsilon$, there is non trivial common prefix, replace all A productions
- A' is new non terminal. Repeatedly apply this until no 2 alternatives for non terminal have common prefix.

PROGRAM CODE:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int main()
{ int n;
  cout<<"\n Enter no of non terminals : ";
  cin >>n;
  cout<<"\nEnter nonterminal one by one: ";
  int i;
  vector <string> nonter(n);
  vector <int> leftrecr(n,0);
  for (i=0;i<n;++i) { cout<<"\n Non termal "<<
   i+1<<" : ";
  cin >>nonter[i]; }
Vector <vector <string >> prod;
 cout <<"\n Enter 'esp' for null ";
 for (i=0;i<n;++i) {
 cout<<"\n\nNumber of "<<nonter[i]<<" productions:
 int k;
 cin >>k;
 int j;
Cout <<"One by one enter all "<<nonter[i]<<"
```

```
vector < string > temp(k);
for (j=0 ; j< k ;++j ) {
cout <<"\n RHS of production "<<j+1<<" productions ";
vector <string> temp(k);
for (j=0 ; j< k ;++j ) {
string abc;
cin >> abc;
temp[j] = abc;
if (nonter[i].length()<=abc.length ()&& nonter[i].
    compare(abc.substr(0, nonter[i].length()))==0)
    left recr[i] = 1; }
  prod.push_back (temp);
}
for( i=0 ; i<n; i ++) {
      cout << 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() )
   string abc = prod[i][j].substr (nonter[i].length(), )
   temp.push_back(abc);
   prod[i].erase (prod[i].begin ()+j);
    --j;
  }
  else {
    prod [i][j] += nonter[i]+"' ";
  }
}
```

output: Left factoring

Enter the grammar: A → iEts | iEtSeS | a

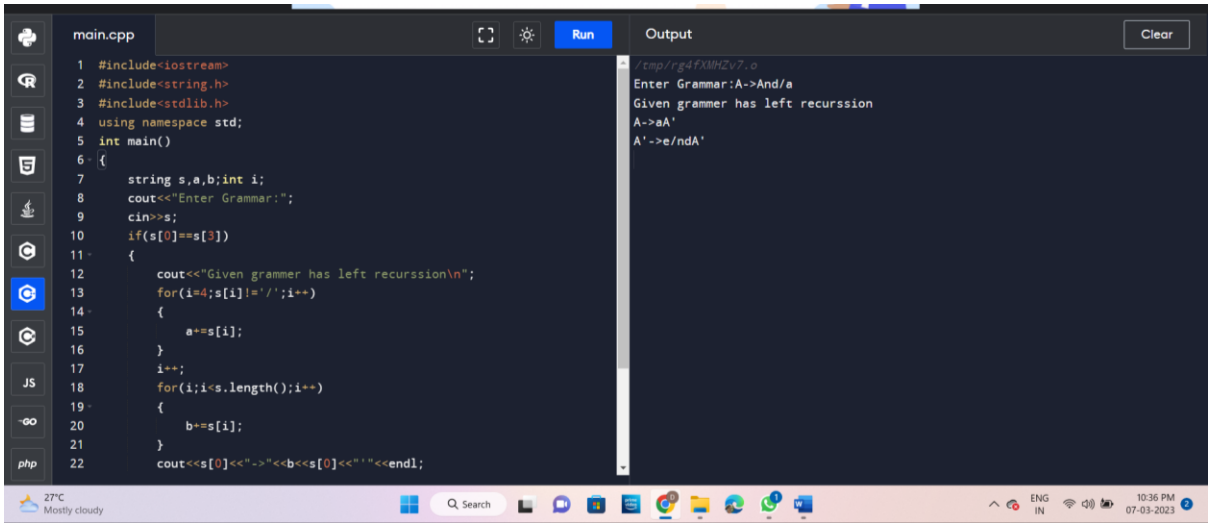
A → iEtSA' | a

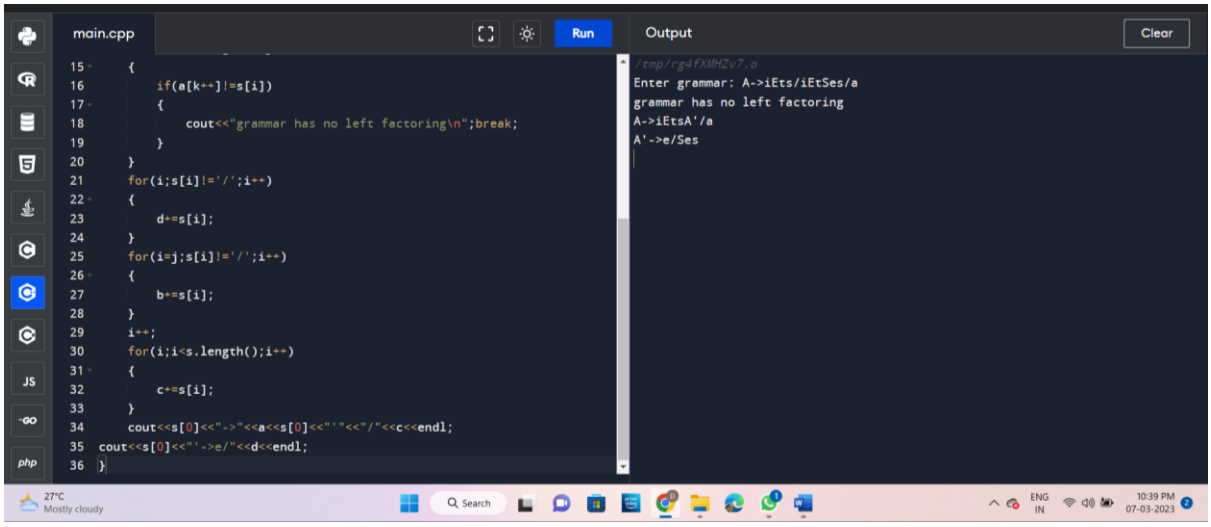
A' → e | SeS

o/p

Enter Grammar : A → aAB

Grammar has no left factoring


main.cpp
Run
Output
Clear
#include<iostream>
#include<string.h>
#include<stdlib.h>
using namespace std;
int main()
{
string s,a,b;int i;
cout<<"Enter Grammar:";
cin>>s;
if(s[0]==s[3])
{
cout<<"Given grammer has left recurssion\n";
for(i=4;s[i]!='/';i++)
{
a+=s[i];
}
i++;
for(i;i<s.length();i++)
{
b+=s[i];
}
cout<<s[0]<<"->"<<b<<s[0]<<"'"<<endl;
/tmp/rg4fXMHZv7.o
Enter Grammar:A->And/a
Given grammer has left recurssion
A->aA'
A'->e/ndA'
27°C
Mostly cloudy
Search
ENG
IN
10:36 PM
07-03-2023


main.cpp
Run
Output
Clear
{
if(a[k++]!=s[i])
{
cout<<"grammar has no left factoring\n";break;
}
}
for(i;s[i]!='/';i++)
{
d+=s[i];
}
for(i=j;s[i]!='/';i++)
{
b+=s[i];
}
i++;
for(i;i<s.length();i++)
{
c+=s[i];
}
cout<<s[0]<<"->"<<a<<s[0]<<"'"<<"/"<<c<<endl;
cout<<s[0]<<"'->e/"<<d<<endl;
}
/tmp/rg4fXMHZv7.o
Enter grammar: A->iEts/iEtSes/a
grammar has no left factoring
A->iEtsA'/a
A'->e/Ses
27°C
Mostly cloudy
Search
ENG
IN
10:39 PM
07-03-2023

```
temp.push_back("eap");
prod.push_back(temp);
}
cout<<"\n\n";
cout<<"New set of nonterminals : ";
  for (i=0;i<nonth.size();++i)
    cout<<nonth[i]<<" ";
  cout<<"\n\nNew set of productions : ";
  for (i=0;i<nonth.size();++i)
  { int j;
    for (j=0;j<prod[i].size();++j)
  { cout<<"\n "<<nonth[i]<<" -> "<<prod[i][j];
  }
return 0;
}
```

RESULT: Hence left recursion and left factoring is done for the given grammar.