EXP4
26/02/21

Elimination of Ambiguity, Left Recursion and Left Factoring

AIM: To write a program to eliminate left recursion and left factoring.

ALGORITHM (left Recursion):
1. Start the program.
2. Initialize the arrays for taking input from the user.
3. Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.
4. Prompt the user to input the right production for non-terminals.
5. Eliminate left recursion using the following rules:- A->Aα1| Aα2 | . . . . . |Aαm A->β1| β2| . . . . .| βn Then replace it by A'-> βi A' i=1,2,3,…..m A'-> αj A' j=1,2,3,…..n A'-> Ɛ
6. After eliminating the left recursion by applying these rules, display the productions without left recursion.
7. Stop.

ALGORITHM (left Factoring):
1. Start the program.
2. Initialize the arrays for taking input from the user.
3. Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.
4. Prompt the user to input the right production for non-terminals.
5.Check for common left factors in the production
6. Group all like productions
7. Simplify original production
8. Create the new production
9. Display all productions
10. Stop

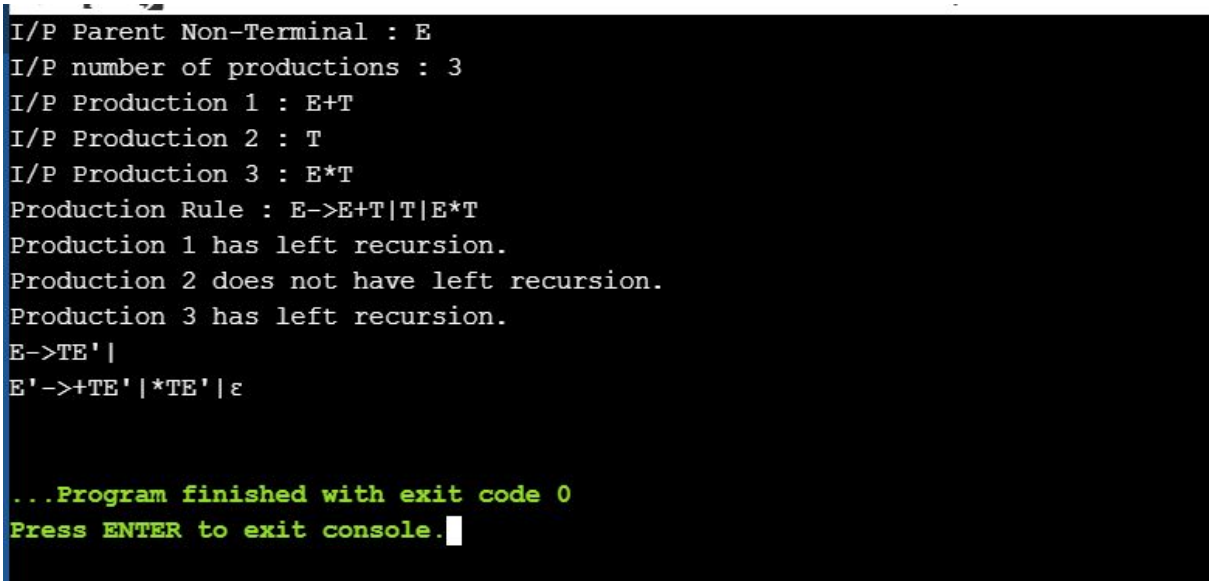
PROGRAM (Left Recursion):

```
#include<iostream>
#include<string>
using namespace std;
int main()
{  string ip,op1,op2,temp;
   int sizes[10] = {};
   char c;
   int n,j,l;
   cout<<"I/P Parent Non-Terminal : ";
   cin>>c;
   ip.push_back(c);
   op1 += ip + "\'->";
   ip += "->";
```

```
    op2+=ip;
    cout<<"I/P number of productions : ";
    cin>>n;
    for(int i=0;i<n;i++)
    {   cout<<"I/P Production "<<i+1<<" : ";
        cin>>temp;
        sizes[i] = temp.size();
        ip+=temp;
        if(i!=n-1)
            ip += "|";
    }
    cout<<"Production Rule : "<<ip<<endl;
    for(int i=0,k=3;i<n;i++)
    {
        if(ip[0] == ip[k])
        {
            cout<<"Production "<<i+1<<" has left recursion."<<endl;
            if(ip[k] != '#')
            {
                for(l=k+1;l<k+sizes[i];l++)
                    op1.push_back(ip[l]);
                k=l+1;
                op1.push_back(ip[0]);
                op1 += "\'|";
            }
        }
        else
        {
            cout<<"Production "<<i+1<<" does not have left recursion."<<endl;
            if(ip[k] != '#')
            {
                for(j=k;j<k+sizes[i];j++)
                    op2.push_back(ip[j]);
                k=j+1;
                op2.push_back(ip[0]);
                op2 += "\'|";
            }
            else
            {
                op2.push_back(ip[0]);
                op2 += "\'";
            }}}
    op1 += "ε";
    cout<<op2<<endl;
    cout<<op1<<endl;
    return 0;

}
```

OUTPUT (Left Recursion):



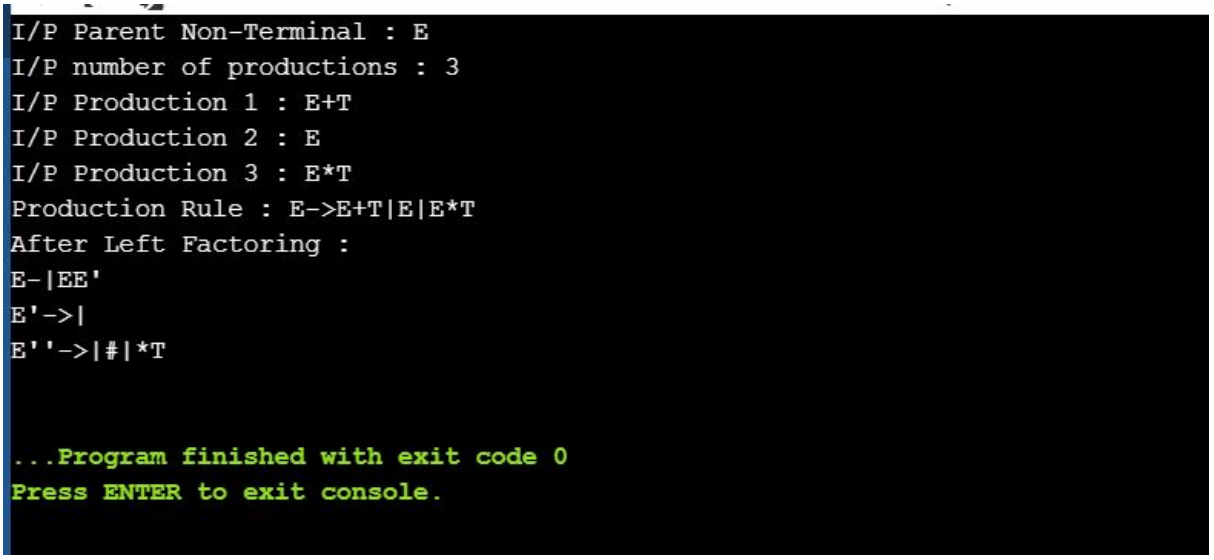
PROGRAM (Left factoring):

```
#include<iostream>
#include<string>
using namespace std;
int main()
{  string ip,op1,op2,temp;
   int sizes[10] = {};
   char c;
   int n,j,l;
   cout<<"I/P Parent Non-Terminal : ";
   cin>>c;
   ip.push_back(c);
   op1 += ip + "\'->";
   op2 += ip + "\'\'->";;
   ip += "->";
   cout<<"I/P number of productions : ";
   cin>>n;
   for(int i=0;i<n;i++)
   {
      cout<<"I/P Production "<<i+1<<" : ";
      cin>>temp;
      sizes[i] = temp.size();
      ip+=temp;
      if(i!=n-1)
         ip += "|";
   }
   cout<<"Production Rule : "<<ip<<endl;
```

```
char x = ip[3];
for(int i=0,k=3;i<n;i++)
{
   if(x == ip[k])
   {
         if(ip[k+1] == '|')
         {
            op1 += "#";
            ip.insert(k+1,1,ip[0]);
            ip.insert(k+2,1,'\'');
            k+=4;
         }
         else
         {
            op1 += "|" + ip.substr(k+1,sizes[i]-1);
            ip.erase(k-1,sizes[i]+1);
         }
   }
   else
   {
      while(ip[k++]!='|');
   }
}
char y = op1[6];
for(int i=0,k=6;i<n-1;i++)
{
   if(y == op1[k])
   {
         if(op1[k+1] == '|')
         {
            op2 += "#";
            op1.insert(k+1,1,op1[0]);
            op1.insert(k+2,2,'\'');
            k+=5;
         }
         else
         {
            temp.clear();
            for(int s=k+1;s<op1.length();s++)
               temp.push_back(op1[s]);
            op2 += "|" + temp;
            op1.erase(k-1,temp.length()+2);
         } }}
op2.erase(op2.size()-1);
cout<<"After Left Factoring : "<<endl;
cout<<ip<<endl;
cout<<op1<<endl;
cout<<op2<<endl;
```

```
    return 0;
}
```

OUTPUT (Left factoring):



RESULT:program to eliminate left recursion and left factoring was successfully implemented.