**Aim:-**

To write a program to eliminate left recursion and left factoring in grammar.

**Algorithm:-**

**Left Recursion:-**



**Left Factoring:-**



**Code:-**

**Left Recursion:-**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

int main()

{

int n;

cout<<"\nEnter number of non terminals: ";

cin>>n;

cout<<"\nEnter non terminals one by one: ";

int i;

vector<string> nonter(n);

vector<int> leftrecr(n,0);

for(i=0;i<n;++i) {

cout<<"\nNon terminal "<<i+1<<" : ";

cin>>nonter[i];

}

vector<vector<string> > prod;

cout<<"\nEnter '^' for null";

for(i=0;i<n;++i) {

cout<<"\nNumber of "<<nonter[i]<<" productions: ";

int k;

cin>>k;

int j;

cout<<"\nOne by one enter all "<<nonter[i]<<" productions";

vector<string> temp(k);

for(j=0;j<k;++j) {

cout<<"\nRHS of production "<<j+1<<": ";

string abc;

cin>>abc;

temp[j]=abc;

if(nonter[i].length()<=abc.length()&&nonter[i].compare(abc.substr(0,nonter[i].length()))==0)

leftrecr[i]=1;

}

prod.push\_back(temp);

}

for(i=0;i<n;++i) {

cout<<leftrecr[i];

}

for(i=0;i<n;++i) {

if(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()&&nonter[i].compare(prod[i][j].substr(0,nonter[i].length()))==0) {

string abc=prod[i][j].substr(nonter[i].length(),prod[i][j].length()-nonter[i].length())+nonter[i]+"'";

temp.push\_back(abc);

prod[i].erase(prod[i].begin()+j);

--j;

}

else {

prod[i][j]+=nonter[i]+"'";

}

}

temp.push\_back("^");

prod.push\_back(temp);

}

cout<<"\n\n";

cout<<"\nNew set of non-terminals: ";

for(i=0;i<nonter.size();++i)

cout<<nonter[i]<<" ";

cout<<"\n\nNew set of productions: ";

for(i=0;i<nonter.size();++i) {

int j;

for(j=0;j<prod[i].size();++j) {

cout<<"\n"<<nonter[i]<<" -> "<<prod[i][j];

}

}

return 0;

}

**Output:-**

**Left Recursion:-**





**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<<"Enter the Parent Non-Terminal : ";

cin>>c;

ip.push\_back(c);

op1 += ip + "\'->";

op2 += ip + "\'\'->";;

ip += "->";

cout<<"Enter the number of productions : ";

cin>>n;

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

{

cout<<"Enter 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:-**

The program for the elimination of left recursion and left factoring was successfully implemented.