**PROGRAM – 5**

**AIM :** WAP to remove Left recursion and Left factoring from input production rule.

**PROGRAM :**

#include <bits/stdc++.h>

using namespace std ;

vector<string> partProd(string str) {

vector<string> parts ;

string temp = "" ;

for(int i = 3 ; i < str . length() ; i ++) {

if(str[i] == '|') {

parts . push\_back(temp) ;

temp = "" ;

} else {

temp = temp + str[i] ;

}

}

if(temp != "") {

parts . push\_back(temp) ;

}

return parts ;

}

string findMatch(string str1 , string str2) {

string matched = "" ;

for(int i = 0 ; str1[i] != '\0' && str2[i] != '\0' ; i ++) {

if(str1[i] == str2[i]) {

matched = matched + str1[i] ;

} else {

break ;

}

}

return matched ;

}

void removeLeftFact(char start , int &count , vector<string> parted) {

string common = "" ;

vector<int> matched , unmatched ;

int i , j ;

for(i = 0 ; i < parted . size() - 1 ; i ++) {

for(j = i + 1 ; j < parted . size() ; j ++) {

if(findMatch(parted[i] , parted[j]) != "") {

common = findMatch(parted[i] , parted[j]) ;

matched . push\_back(i) ;

matched . push\_back(j) ;

goto out ;

}

}

}

out :

if(common != "") {

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

if(k != i && k != j)

unmatched . push\_back(k) ;

}

for(int k = j + 1 ; k < parted . size() ; k ++) {

if(findMatch(common , parted[k]) != "") {

common = findMatch(common , parted[k]) ;

matched . push\_back(k) ;

} else {

unmatched . push\_back(k) ;

}

}

char start1 = (char)(90 - count) ;

vector<string> next\_parted ;

next\_parted . push\_back(common + (char)(90 - count)) ;

for(int k = 0 ; k < unmatched . size() ; k ++) {

next\_parted . push\_back(parted[unmatched[k]]) ;

}

count ++ ;

removeLeftFact(start , count , next\_parted) ;

next\_parted . clear() ;

int len = common . length() ;

for(int k = 0 ; k < matched . size() ; k ++) {

next\_parted . push\_back(parted[matched[k]] . substr(len , parted[matched[k]] . length() - 1)) ;

}

count ++ ;

removeLeftFact(start1 , count , next\_parted) ;

} else {

cout << start << "->" ;

for(int k = 0 ; k < parted . size() ; k ++) {

if(k != 0)

cout << "|" ;

cout << parted[k] ;

}

cout << "\n" ;

return ;

}

}

vector<string> removeLeftRec(int &count , string str) {

vector<string> afterLeftRecProds ;

string a = "" , b = "" ;

vector<string> leftRecStr , betaStr ;

string temp = "" ;

bool isLeftRec = false ;

vector<string> parted = partProd(str) ;

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

if(parted[i][0] == str[0]) {

leftRecStr . push\_back(parted[i]) ;

isLeftRec = true ;

} else {

betaStr . push\_back(parted[i]) ;

}

}

if(isLeftRec && ! betaStr . empty()) {

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

string temp\_str = "" ;

temp\_str += str[0] ;

temp\_str += "->" ;

for(int j = 0 ; j < betaStr . size() ; j ++) {

temp\_str += betaStr[j] + (char)(90 - count) ;

if(j == betaStr . size() - 1) {

afterLeftRecProds . push\_back(temp\_str) ;

cout << temp\_str << "\n" ;

} else {

temp\_str += "|" ;

}

}

temp\_str = (char)(90 - count) ;

count ++ ;

temp\_str += "->" ;

temp\_str += leftRecStr[i] . substr(1 , leftRecStr[i] . length() - 1) + (char)(90 - i) ;

temp\_str += "|$" ;

cout << temp\_str << "\n" ;

afterLeftRecProds . push\_back(temp\_str) ;

}

} else {

cout << str << "\n" ;

afterLeftRecProds . push\_back(str) ;

if(betaStr . empty())

cout << "Left Recursion can not be removed ...\n" ;

else

cout << "Left Recursion does not exist ...\n" ;

}

return afterLeftRecProds ;

}

int main() {

cout << "Input single line Production rule in form (A->aB|c) -\n" ;

string input ;

cin >> input ;

cout << "\n" ;

cout << "Production rules after removing Left Recursion are -\n" ;

int count = 0 ;

vector<string> strs = removeLeftRec(count , input) ;

int pre\_count = count ;

cout << "\nProduction rules after removing Left Factoring are -\n" ;

bool isLeftFact = false ;

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

removeLeftFact(strs[i][0] , count , partProd(strs[i])) ;

if(count != pre\_count)

isLeftFact = true ;

}

if(! isLeftFact)

cout << "Left factoring does not exist ...\n" ;

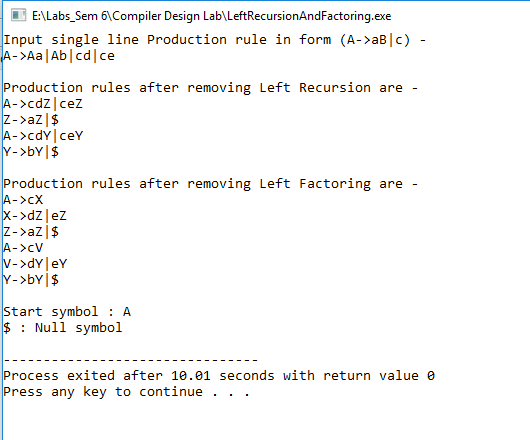
strs . clear() ;

cout << "\nStart symbol : " << input[0] << "\n$ : Null symbol\n" ;

return 0 ;

}

**Output :**

****