***EXPERIMENT NO. 07***

***Aim:*** Write a program for left recursion direct as well as indirect given the set of production rule.

***Theory:***

In the formal language theory of computer science, left recursion is a special case of recursion where a string is recognized as part of a language by the fact that it decomposes into a string from that same language (on the left) and a suffix (on the right). For instance, can be recognized as a sum because it can be broken into, also a sum, and , a suitable suffix.

In terms of context-free grammar, a nonterminal is left-recursive if the leftmost symbol in one of its productions is itself (in the case of direct left recursion) or can be made itself by some sequence of substitutions (in the case of indirect left recursion). A grammar is left-recursive if and only if there exists a nonterminal symbol  that can derive to a sentential form with itself as the leftmost symbol.[[1]](https://en.wikipedia.org/wiki/Left_recursion#cite_note-1) Symbolically,

,

where  indicates the operation of making one or more substitutions, and  is any sequence of terminal and nonterminal symbols.

**Direct left recursion**

Direct left recursion occurs when the definition can be satisfied with only one substitution. It requires a rule of the form



where  is a sequence of nonterminals and terminals. For example, the rule



is directly left-recursive. A left-to-right recursive descent parser for this rule might look like

function Expression()

{

Expression(); match('+'); Term();

}

and such code would fall into infinite recursion when executed.

**Indirect left recursion**

Indirect left recursion occurs when the definition is satisfied via several substitutions. It entails a set of rules following the pattern





where  are sequences that can each yield the empty string, while  may be any sequences at all. The derivation



then gives  as leftmost in its final sentential form.

**Removing left recursion**

Left recursion often poses problems for parsers, either because it leads them into infinite recursion (as in the case of most top-down parsers) or because they expect rules in a normal form that forbids it (as in the case of many bottom-up parsers, including the CYK algorithm). Therefore a grammar is often preprocessed to eliminate the left recursion.

**Removing direct left recursion**

The general algorithm to remove direct left recursion follows. Several improvements to this method have been made.[[2]](https://en.wikipedia.org/wiki/Left_recursion#cite_note-Moore2000-2) For a left-recursive nonterminal , discard any rules of the form  and consider those that remain:



where:

each  is a nonempty sequence of nonterminals and terminals, and

each  is a sequence of nonterminals and terminals that does not start with .

Replace these with two sets of productions, one set for :



and another set for the fresh nonterminal  (often called the "tail" or the "rest"):

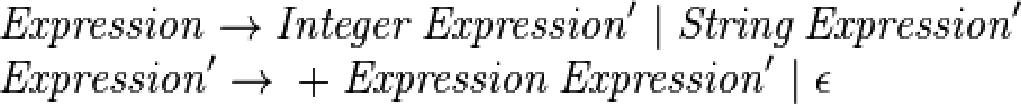


Repeat this process until no direct left recursion remains.

As an example, consider the rule set



This could be rewritten to avoid left recursion as



**Removing all left recursion**

By establishing a topological ordering on nonterminals, the above process can be extended to also eliminate indirect left recursion

inputs *A grammar: a set of nonterminals  and their productions* Output *A modified grammar generating the same language but without left recursion*

*For each nonterminal :*

*Repeat until an iteration leaves the grammar unchanged:*

*For each rule ,  being a sequence of terminals and nonterminals:*

*If  begins with a nonterminal  and :*

*Let  be  without its leading .*

*Remove the rule .*

*For each rule :*

*Add the rule .*

*Remove direct left recursion for  as described above.*

***Program:***

#include<stdio.h>

#include<string.h>

#define SIZE 10

int main () {

char non\_terminal;

char beta,alpha,g;

int num;

char production[10][SIZE];

int index=3; /\* starting of the string following "->" \*/

printf("Enter Number of Production : ");

scanf("%d",&num);

printf("Enter the grammar as E->E-A :\n");

for(int i=0;i<num;i++){

scanf("%s",production[i]);

}

//Direct Rec

for(int i=0;i<num;i++){

printf("\nGRAMMAR : : : %s",production[i]);

non\_terminal=production[i][0];

if(non\_terminal==production[i][index]) {

alpha=production[i][index+1];

g= production[i][index+2];

printf(" is left recursive.\n");

while(production[i][index]!=0 && production[i][index]!='|')

index++;

if(production[i][index]!=0) {

beta=production[i][index+1];

printf("Grammar without left recursion:\n");

printf("%c->%c%c\'",non\_terminal,beta,non\_terminal);

printf("\n%c\'->%c%c%c\'|&\n",non\_terminal,alpha,g,non\_terminal);

}

else

printf(" can't be reduced\n");

}

else

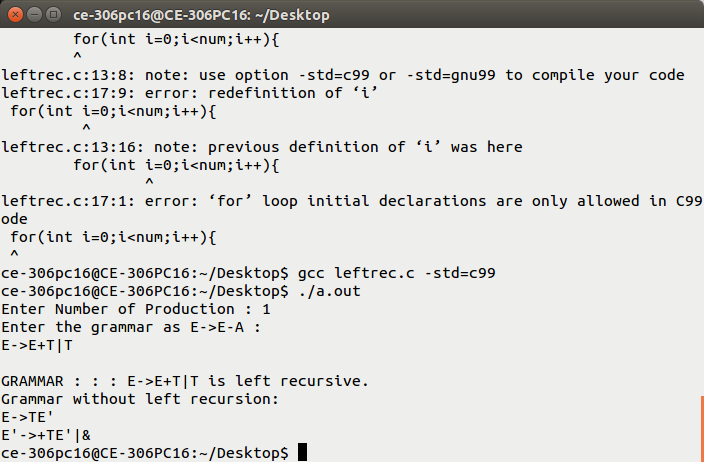
printf(" is not left recursive.\n");

index=3;

}

}

***Output:***

******

***Conclusion:*** We study recursion and Implemented Program for Removal of Left Recursion using direct and indirect.