# Aim

Write a program to implement left recursion.

# Program logic

Check if the given grammar contains left recursion, if present then separate the production and start working on it.  
In our example,

S-->S a/

S b

/c

/ d

## Introduce a new nonterminal and write it at the last of every terminal. We produce a new nonterminal S’and write new production as,

S-->cS' / dS'

## Write newly produced nonterminal in LHS and in RHS it can either produce or it can produce new production in which the terminals or non-terminals which followed the previous LHS will be replaced by new nonterminal at last.

S'-->?

/ aS'

/ bS'

## So, after conversion the new equivalent production is

S-->cS' / dS'

S'-->? / aS' / bS'

### Step by step elimination of this indirect left recursion

A -> Cd

B -> Ce

C -> A | B | f

### In this case order would be A < B < C, and possible paths for recursion of non-terminal C would be

C=> A => Cd

and

C=> B => Ce

so new rules for C would be

C=> Cd | Ce | f

now you can simply just remove direct left recursion:

C=> fC'

C'=> dC' | eC' | eps

and the resulting non-recursive grammar would be:

A => Cd

B => Ce

C => fC'

C' => dC' | eC' | eps

# Lab Assignment

## What is Left Recursion?

* A production of grammar is said to have left recursion if the leftmost variable of its RHS is same as variable of its LHS.
* A grammar containing a production having left recursion is called as Left Recursive Grammar.

Example-

S → Sa / ∈

**(Left Recursive Grammar)**

* Left recursion is a problematic situation for Top-down parsers.
* Therefore, left recursion must be eliminated from the grammar.

## What is right recursion?

* A production of grammar is said to have **right recursion** if the rightmost variable of its RHS is same as variable of its LHS.
* A grammar containing a production having right recursion is called as Right Recursive Grammar.

Example-

S → aS / ∈

**(Right Recursive Grammar)**

* Right recursion does not create any problem for the Top-down parsers.
* Therefore, there is no need of eliminating right recursion from the grammar.

## Why to remove left recursion?

Left recursion is a problematic situation for Top-down parsers. Therefore, left recursion has to be eliminated from the grammar.

## Define algorithm for left recursion

Left recursion is eliminated by converting the grammar into a right recursive grammar.

If we have the left-recursive pair of productions-

A → Aα / β

(Left Recursive Grammar)

where β does not begin with an A.

Then, we can eliminate left recursion by replacing the pair of productions with-

A → βA’

A’ → αA’ / ∈

(Right Recursive Grammar)

This right recursive grammar functions same as left recursive grammar.

## What are different rules for left Recursion.

The production is left-recursive if the leftmost symbol on the right side is the same as the non-terminal on the left side. For example,  
            expr → expr + term.

If one were to code this production in a recursive-descent parser, the parser would go in an infinite loop.

We can eliminate the left-recursion by introducing new nonterminal and new productions rules.

# Lab Assignment Program

Write a program to implement left recursion.

## Code

gram = {}

def add(str):                               #to rules together

    x = str.split("->")

    y = x[1]

    x.pop()

    z = y.split("|")

    x.append(z)

    gram[x[0]]=x[1]

def removeDirectLR(gramA, A):

    """gramA is dictonary"""

    temp = gramA[A]

    tempCr = []

    tempInCr = []

    for i in temp:

        if i[0] == A:

            #tempInCr.append(i[1:])

            tempInCr.append(i[1:]+[A+"'"])

        else:

            #tempCr.append(i)

            tempCr.append(i+[A+"'"])

    tempInCr.append(["e"])

    gramA[A] = tempCr

    gramA[A+"'"] = tempInCr

    return gramA

def checkForIndirect(gramA, a, ai):

    if ai not in gramA:

        return False

    if a == ai:

        return True

    for i in gramA[ai]:

        if i[0] == ai:

            return False

        if i[0] in gramA:

            return checkForIndirect(gramA, a, i[0])

    return False

def rep(gramA, A):

    temp = gramA[A]

    newTemp = []

    for i in temp:

        if checkForIndirect(gramA, A, i[0]):

            t = []

            for k in gramA[i[0]]:

                t=[]

                t+=k

                t+=i[1:]

                newTemp.append(t)

        else:

            newTemp.append(i)

    gramA[A] = newTemp

    return gramA

def rem(gram):

    c = 1

    conv = {}

    gramA = {}

    revconv = {}

    for j in gram:

        conv[j] = "A"+str(c)

        gramA["A"+str(c)] = []

        c+=1

    for i in gram:

        for j in gram[i]:

            temp = []

            for k in j:

                if k in conv:

                    temp.append(conv[k])

                else:

                    temp.append(k)

            gramA[conv[i]].append(temp)

    #print(gramA)

    for i in range(c-1,0,-1):

        ai = "A"+str(i)

        for j in range(0,i):

            aj = gramA[ai][0][0]

            if ai!=aj :

                if aj in gramA and checkForIndirect(gramA,ai,aj):

                    gramA = rep(gramA, ai)

    for i in range(1,c):

        ai = "A"+str(i)

        for j in gramA[ai]:

            if ai==j[0]:

                gramA = removeDirectLR(gramA, ai)

                break

    op = {}

    for i in gramA:

        a = str(i)

        for j in conv:

            a = a.replace(conv[j],j)

        revconv[i] = a

    for i in gramA:

        l = []

        for j in gramA[i]:

            k = []

            for m in j:

                if m in revconv:

                    k.append(m.replace(m,revconv[m]))

                else:

                    k.append(m)

            l.append(k)

        op[revconv[i]] = l

    return op

n = int(input("Enter No of Production: "))

for i in range(n):

    txt=input()

    add(txt)

result = rem(gram)

for x,y in result.items():

    print("The output after Left Reccursion is ::")

    print(f'{x} -> {y}')

## Output

Text

Description automatically generated

# Conclusion

Hence, we were able to implement left recursion.