**Aim**

To develop a **C program** to eliminate **left recursion** from a given **context-free grammar (CFG)**.

**Concept of Left Recursion Elimination**

A grammar has **left recursion** if a **non-terminal** has a production of the form:

less

CopyEdit

A → Aα | β

where α is a sequence of symbols, and β does not start with A.

**Steps to Remove Left Recursion**

1. Rewrite the grammar by introducing a new non-terminal A':

vbnet

CopyEdit

A → βA'

A' → αA' | ε

1. Repeat for all non-terminals.

**Example Grammar (with Left Recursion)**

r

CopyEdit

S → Sa | Sb | c

**After Removing Left Recursion**

nginx

CopyEdit

S → cS'

S' → aS' | bS' | ε

**C Program to Eliminate Left Recursion**

c

CopyEdit

#include <stdio.h>

#include <string.h>

#define MAX 10

char productions[MAX][MAX]; // Stores input grammar

char newProductions[MAX][MAX]; // Stores transformed grammar

int numProductions; // Number of productions

// Function to eliminate left recursion

void eliminateLeftRecursion() {

int newProdCount = 0; // Tracks new production count

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

char nonTerminal = productions[i][0]; // Get LHS (e.g., 'S')

char alpha[MAX][MAX], beta[MAX][MAX]; // To store α and β

int alphaCount = 0, betaCount = 0;

// Process RHS of the production

char \*rhs = strchr(productions[i], '>') + 1;

char \*token = strtok(rhs, "|"); // Split by '|'

while (token != NULL) {

if (token[0] == nonTerminal) {

// If production is left-recursive (A → Aα)

strcpy(alpha[alphaCount++], token + 1);

} else {

// Otherwise, store β

strcpy(beta[betaCount++], token);

}

token = strtok(NULL, "|");

}

if (alphaCount > 0) {

// New non-terminal for recursion removal

char newNonTerminal = nonTerminal + '\'';

// Construct new rules

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

sprintf(newProductions[newProdCount++], "%c->%s%c", nonTerminal, beta[j], newNonTerminal);

}

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

sprintf(newProductions[newProdCount++], "%c->%s%c", newNonTerminal, alpha[j], newNonTerminal);

}

// Add epsilon production for new non-terminal

sprintf(newProductions[newProdCount++], "%c->ε", newNonTerminal);

} else {

// If no left recursion, keep production unchanged

strcpy(newProductions[newProdCount++], productions[i]);

}

}

numProductions = newProdCount;

}

// Function to print the updated grammar

void printNewGrammar() {

printf("\nGrammar after eliminating left recursion:\n");

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

printf("%s\n", newProductions[i]);

}

}

int main() {

printf("Enter the number of productions: ");

scanf("%d", &numProductions);

getchar(); // Consume newline

printf("Enter the productions (e.g., S->Sa|Sb|c):\n");

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

fgets(productions[i], MAX, stdin);

productions[i][strcspn(productions[i], "\n")] = '\0'; // Remove newline

}

eliminateLeftRecursion();

printNewGrammar();

return 0;

}

**Example Input and Output**

**Input**

r

CopyEdit

Enter the number of productions: 1

Enter the productions (e.g., S->Sa|Sb|c):

S->Sa|Sb|c

**Output**

rust

CopyEdit

Grammar after eliminating left recursion:

S->cS'

S'->aS'

S'->bS'

S'->ε