**CSA1443-COMPILER DESIGN FOR INTRAPROCEDURAL ANALYSIS**

**NAME:Ajay Kumar J**

**REG NO: 192372052**

**PROGRAM 9**

**Aim:**

To implement a C program that eliminates left recursion from a given context-free grammar (CFG). Left recursion occurs when a non-terminal on the left-hand side of a production rule appears at the beginning of its own right-hand side, leading to infinite recursion in recursive descent parsers.

**Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX 10

int isTerminal(char c) {

return !isupper(c);

}

void eliminateLeftRecursion(char grammar[MAX][MAX], int \*n, char nonTerminal) {

char newNonTerminal = nonTerminal + '1'

char newGrammar[MAX][MAX];

int newProductionCount = 0;

int i = 0, j = 0;

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

if (grammar[i][0] == nonTerminal) {

if (isTerminal(grammar[i][2])) {

sprintf(newGrammar[newProductionCount++], "%c→%s%c", nonTerminal, grammar[i] + 2, newNonTerminal);

}

} else {

sprintf(newGrammar[newProductionCount++], "%s", grammar[i]);

}

}

sprintf(newGrammar[newProductionCount++], "%c→ε", newNonTerminal);

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

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

}

}

int main() {

int n;

char grammar[MAX][MAX];

printf("Enter number of productions: ");

scanf("%d", &n);

getchar();

printf("Enter the productions (in the form: A->a or A->B):\n");

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

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

grammar[i][strcspn(grammar[i], "\n")] = 0;

}

printf("\nOriginal Grammar:\n");

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

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

}

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

if (isupper(grammar[i][0])) {

printf("\nAfter Eliminating Left Recursion for Non-Terminal %c:\n", grammar[i][0]);

eliminateLeftRecursion(grammar, &n, grammar[i][0]);

}

}

return 0;

}

**Output:**

