**CSA1443-COMPILER DESIGN FOR INTRAPROCEDURAL ANALYSIS**

**NAME: Ajay Kumar J**

**REG NO: 192372052**

**PROGRAM 10**

**Aim:**

To implement a C program that eliminates **left factoring** from a given context-free grammar (CFG). Left factoring is a technique used to transform grammars that have common prefixes into a form where the choice between alternatives is made after the common prefix is processed.

**Code:**

#include <stdio.h>

#include <string.h>

#define MAX 10

#define MAX\_PROD 100

int isTerminal(char c) {

return !(c >= 'A' && c <= 'Z');

}

void eliminateLeftFactoring(char grammar[MAX\_PROD][MAX], int \*n) {

char newGrammar[MAX\_PROD][MAX];

int newProductionCount = 0;

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

for (int j = i + 1; j < \*n; j++) {

if (grammar[i][0] == grammar[j][0] && grammar[i][2] == grammar[j][2]) {

char prefix[MAX] = {0};

int k = 2;

while (grammar[i][k] == grammar[j][k] && grammar[i][k] != '\0') {

prefix[k - 2] = grammar[i][k];

k++;

}

char newNonTerminal = grammar[i][0] + 1;

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

sprintf(newGrammar[newProductionCount++], "%c→%s%c", grammar[i][0], prefix, newNonTerminal);

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

grammar[i][0] = '\0';

grammar[j][0] = '\0';

}

}

}

printf("\nGrammar after Left Factoring:\n");

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

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

}

}

int main() {

int n;

char grammar[MAX\_PROD][MAX];

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

scanf("%d", &n);

getchar();

printf("Enter the productions in the form A->alpha:\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]);

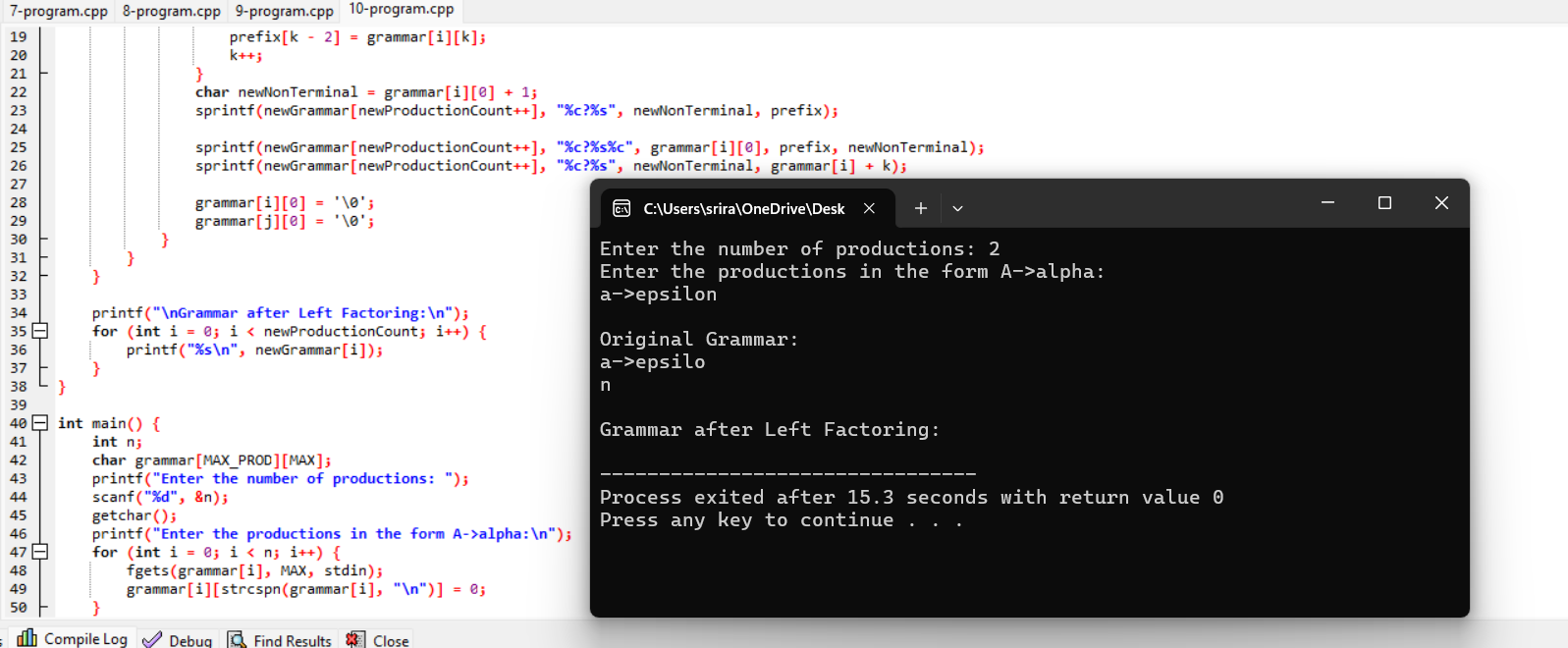
}

eliminateLeftFactoring(grammar, &n);

return 0;

}

**Output:**

****