14.Write a C Program for code optimization to eliminate common subexpression.

**PROGRAM:**

#include <stdio.h>

#include <string.h>

typedef struct {

char op;

char arg1[10];

char arg2[10];

char result[10];

} Triple;

Triple triples[50];

int nextTriple = 0;

void generateTriple(char op, char arg1[], char arg2[], char result[]) {

Triple newTriple;

newTriple.op = op;

strcpy(newTriple.arg1, arg1);

strcpy(newTriple.arg2, arg2);

strcpy(newTriple.result, result);

triples[nextTriple++] = newTriple;

}

int findCommonSubexpression(char op, char arg1[], char arg2[]) {

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

if (triples[i].op == op && strcmp(triples[i].arg1, arg1) == 0 && strcmp(triples[i].arg2, arg2) == 0) {

return i;

}

}

return -1;

}

int main() {

generateTriple('+', "a", "b", "t1");

generateTriple('-', "t1", "c", "t2");

generateTriple('\*', "a", "b", "t3");

generateTriple('+', "t2", "t3", "t4");

generateTriple('/', "t4", "t1", "result");

generateTriple('+', "a", "b", "t5");

generateTriple('-', "t5", "c", "t6");

generateTriple('\*', "a", "b", "t7");

generateTriple('+', "t6", "t7", "t8");

generateTriple('/', "t8", "t5", "result");

printf("Original Triples:\n");

printf("Op\tArg1\tArg2\tResult\n");

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

printf("%c\t%s\t%s\t%s\n", triples[i].op, triples[i].arg1, triples[i].arg2, triples[i].result);

}

int changed = 0;

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

int commonIndex = findCommonSubexpression(triples[i].op, triples[i].arg1, triples[i].arg2);

if (commonIndex != -1 && commonIndex < i) {

strcpy(triples[i].result, triples[commonIndex].result);

changed = 1;

}

}

if (changed) {

printf("\nAfter Common Subexpression Elimination:\n");

printf("Op\tArg1\tArg2\tResult\n");

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

printf("%c\t%s\t%s\t%s\n", triples[i].op, triples[i].arg1, triples[i].arg2, triples[i].result);

}

} else {

printf("\nNo common subexpressions eliminated.\n");

}

return 0;

}

**OUTPUT:**

