EXP NO : 11 **DATE : 28.04.25**

IMPLEMENT CODE OPTIMIZATION TECHNIQUES LIKE DEAD CODE AND COMMON EXPRESSION ELIMINATION

AIM:

The aim is to implement code optimization techniques such as Dead Code Elimination (DCE) and Common Subexpression Elimination (CSE) on an intermediate representation of a program (such as Three-Address Code (TAC)). These optimization techniques help reduce the size of the code, improve runtime performance, and eliminate redundant computations during the compilation process.

ALGORITHM:

- Start
- Create the input file which contains three address code.
- Open the file in read mode.
- If the file pointer returns NULL, exit the program else go to 5.
- Scan the input symbol from left to right.
- Store the first expression in a string.
- Compare the string with the other expressions in the file.
- If there is a match, remove the expression from the input file.
- Perform these steps 5-8 for all the input symbols in the file.
- Scan the input symbol from the file from left to right.
- Get the operand before the operator from the three address code.
- Check whether the operand is used in any other expression in the three address code.
- If the operand is not used, then eliminate the complete expression from the three address code else go to 14.
- Perform steps 11 to 13 for all the operands in the three address code till end of the file is reached.
- Stop.

PROGRAM:

```
#include  <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_CODE_LINES 100
#define MAX_LINE_LENGTH 100
#define MAX_VAR_LENGTH 20
typedef struct {
    char lhs[MAX_VAR_LENGTH];
    char op1[MAX_VAR_LENGTH];
    char operator;
    char op2[MAX_VAR_LENGTH];
    int isDead;
} TAC;
```

```c
void parseTACLine(char *line, TAC *tac) {
    sscanf(line, "%s = %s %c %s", tac->lhs, tac->op1, &tac->operator, tac->op2);
    tac->isDead = 0;
}

void performDCE(TAC tac[], int n) {
    int used[MAX_CODE_LINES] = {0};

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (strcmp(tac[i].lhs, tac[j].op1) == 0 || strcmp(tac[i].lhs, tac[j].op2) == 0) {
                used[i] = 1;
                break;
            }
        }
    }
    for (int i = 0; i < n; i++) {
        if (!used[i]) {
            tac[i].isDead = 1;
        }
    }
}

void performCSE(TAC tac[], int n) {
    for (int i = 0; i < n; i++) {
        if (tac[i].isDead) continue;
        for (int j = i + 1; j < n; j++) {
            if (tac[j].isDead) continue;
            if (strcmp(tac[i].op1, tac[j].op1) == 0 &&
                strcmp(tac[i].op2, tac[j].op2) == 0 &&
                tac[i].operator == tac[j].operator) {
                // Replace the second occurrence with the first
                strcpy(tac[j].op1, tac[i].lhs);
                tac[j].operator = '\0';
                strcpy(tac[j].op2, "");
                tac[j].isDead = 1;
            }
        }
    }
}
void printOptimizedTAC(TAC tac[], int n) {
    printf("Optimized Three-Address Code:\n");
    for (int i = 0; i < n; i++) {
        if (!tac[i].isDead) {
            printf("%s = %s", tac[i].lhs, tac[i].op1);
            if (tac[i].operator != '\0') {
                printf(" %c %s", tac[i].operator, tac[i].op2);
            }
            printf("\n");
        }
```

```
    }
}

int main() {
    char *code[] = {
        "t1 = a + b",
        "t2 = a + b",
        "t3 = t1 * c",
        "t4 = t2 * c",
        "d = t3 + t4",
        "e = t5 - t6"
    };
    int n = sizeof(code) / sizeof(code[0]);
    TAC tac[MAX_CODE_LINES];
    for (int i = 0; i < n; i++) {
        parseTACLine(code[i], &tac[i]);
    }
    performCSE(tac, n);
    performDCE(tac, n);
    printOptimizedTAC(tac, n);
    return 0;
}
```

OUTPUT :

```
Optimized Three-Address Code:
t1 = a + b
t3 = t1 * c
t4 = t2 * c
```

RESULT:

Thus The Above Program To Implement Code Optimization Techniques Like Dead Code And Common Expression Elimination Is Executed And Implemented Successfully.