**Ex No: 10**

**Date:**

**IMPLEMENT CODE OPTIMIZATION TECHNIQUES**

**DEAD CODE AND COMMON SUB EXPRESSION ELIMINATION**

**AIM:**

To write a C program to implement the dead code elimination and common sub expression elimination (code optimization) techniques.

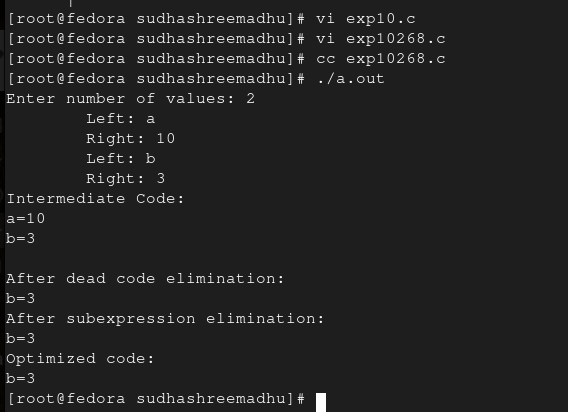
**ALGORITHM:**

* 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<string.h>  
  
struct op {  
    char l;  
    char r[20];  
} op[10], pr[10];  
  
int main() {  
    int a, i, k, j, n, z = 0, m, q;  
    char \*p, \*l;  
    char temp, t;  
    char \*tem;  
  
    printf("Enter number of values: ");  
    scanf("%d", &n);  
  
    for (i = 0; i < n; i++) {  
    printf("\tLeft: ");  
    scanf(" %c", &op[i].l); // added space before %c to clear input buffer  
        printf("\tRight: ");  
        scanf("%s", op[i].r);  
    }  
  
    printf("Intermediate Code:\n");  
    for (i = 0; i < n; i++) {  
        printf("%c=%s\n", op[i].l, op[i].r);  
    }  
  
    // Dead code elimination  
   for (i = 0; i < n - 1; i++) {  
      temp = op[i].l;  
       for (j = 0; j < n; j++) {  
        p = strchr(op[j].r, temp);  
          if (p) {  
           pr[z].l = op[i].l;  
          strcpy(pr[z].r, op[i].r);  
            z++;  
            }  
        }  
    }  
    pr[z].l = op[n - 1].l;  
    strcpy(pr[z].r, op[n - 1].r);  
    z++;  
  
    printf("\nAfter dead code elimination:\n");  
    for (k = 0; k < z; k++) {  
        printf("%c=%s\n", pr[k].l, pr[k].r);  
    }  
  
    // Subexpression elimination  
    for (m = 0; m < z; m++) {  
        tem = pr[m].r;  
        for (j = m + 1; j < z; j++) {  
            p = strstr(tem, pr[j].r);  
            if (p) {  
                t = pr[j].l;  
                pr[j].l = pr[m].l;  
                for (i = 0; i < z; i++) {  
                    l = strchr(pr[i].r, t);  
                    if (l) {  
                        a = l - pr[i].r;  
                        pr[i].r[a] = pr[m].l;  
                    }  
                }  
            }  
        }  
    }  
  
    printf("After subexpression elimination:\n");  
    for (i = 0; i < z; i++) {  
        printf("%c=%s\n", pr[i].l, pr[i].r);  
    }  
  
    // Duplicate production elimination  
    for (i = 0; i < z; i++) {  
        for (j = i + 1; j < z; j++) {  
            q = strcmp(pr[i].r, pr[j].r);  
            if ((pr[i].l == pr[j].l) && !q) {  
                pr[i].l = '\0';  
                strcpy(pr[i].r, "\0");  
            }  
        }  
    }  
  
    printf("Optimized code:\n");  
    for (i = 0; i < z; i++) {  
        if (pr[i].l != '\0') {  
            printf("%c=%s\n", pr[i].l, pr[i].r);  
        }  
    }  
  
    return 0;  
}

**OUTPUT:**



**RESULT:**