Name: Nimish Kushwaha

Reg. No.: CH.EN.U4CSE22073

Lab Exp.: 07

Aim: To implementation of Code Optimization Techniques.

Code:

```
#include <stdio.h>
#include <string.h>

// Structure to represent an intermediate code statement
struct op {
   char l;    // Left part (the variable being assigned to)
   char r[20]; // Right part (the expression)
} op[10], pr[10]; // op[] stores the original code, pr[] stores the optimized code

int main() {
   int a, i, k, j, n, z = 0, m, q;
   char *p, *l;
   char temp, t;
   char *tem;

   printf("Enter the Number of Values: ");
   scanf("%d", &n); // Read the number of statements (n)

   // Input the intermediate code statements
   for (i = 0; i < n; i++) {
      printf("left: ");
      // Assuming single character variable name on the left side
      scanf(" %c", &op[i].l);
      printf("right: ");
      // Reading the expression on the right side
      scanf("%s", op[i].r);
   }

   printf("\nIntermediate Code\n");
   for (i = 0; i < n; i++) {
      printf("%c = %s\n", op[i].l, op[i].r);
   }
```

```
// Dead code elimination part
// Copy only statements where the assigned variable (op[i].l) is used in a later statement's right side.
for (i = 0; i < n - 1; i++) {
   temp = op[i].l; // Variable assigned in current statement

   // Check if the variable 'temp' is used in any subsequent statement's right side
   for (j = i + 1; j < n; j++) {
      p = strchr(op[j].r, temp); // Search for 'temp' in op[j].r
      if (p) {
         // If found, this statement is NOT dead. Copy it to the 'pr' array.
         pr[z].l = op[i].l;
         strcpy(pr[z].r, op[i].r);
         z++;
         break; // Once found, no need to add duplicates for this statement (op[i])
      }
   }
}

// Add last statement as it is (it's assumed the result of the last statement is used/printed outside)
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);
}
// Common subexpression elimination
for (m = 0; m < z; m++) {
   tem = pr[m].r; // Right side of the current statement

   // Compare with all subsequent statements
   for (j = m + 1; j < z; j++) {
      p = strstr(tem, pr[j].r); // Check if pr[j].r is a common subexpression in pr[m].r
      if (p) {
         t = pr[j].l; // Variable assigned by the later common subexpression
         pr[j].l = pr[m].l; // Replace the variable of the later statement with the earlier one

         // The following inner loop seems intended to update the right sides of other statements
         // that might use the later common subexpression (pr[j].l), replacing it with the earlier one (pr[m].l)
         for (i = 0; i < z; i++) {
            l = strchr(pr[i].r, t); // Search for the eliminated variable 't' in pr[i].r
```

```
                if (l) {
                    a = l - pr[i].r; // Position of the character 't'
                    pr[i].r[a] = pr[m].l; // Replace it with the earlier variable 'pr[m].l'
                }
            }
        }
    }

    printf("\nAfter Eliminating Common Expressions\n");
    for (i = 0; i < z; i++) {
        printf("%c = %s\n", pr[i].l, pr[i].r);
    }
    // Remove duplicates by marking them '\0' (This step cleans up the result of CSE)
    for (i = 0; i < z; i++) {
        // Compare statement i with all following statements j
        for (j = i + 1; j < z; j++) {
            // Compare the right sides
            q = strcmp(pr[i].r, pr[j].r);

            // If right sides are the same AND the left sides are the same (e.g., a=b+c and a=b+c)
            if ((pr[i].l == pr[j].l) && (q == 0)) {
                pr[i].l = '\0'; // Mark the earlier duplicate statement's left variable as '\0' for removal
            }
        }
    }
    printf("\nOptimized Code\n");
    for (i = 0; i < z; i++) {
        // Print only statements that haven't been marked for removal
        if (pr[i].l != '\0') {
            printf("%c = %s\n", pr[i].l, pr[i].r);
        }
    }
    return 0;
}
```

Output:

```
asecomputerlab@asecomputerlab-hp-prodesk-400-g7-micrtower-pc:~/Desktop/lab$ nano dead_code_elimination.c
asecomputerlab@asecomputerlab-hp-prodesk-400-g7-micrtower-pc:~/Desktop/lab$ gcc -o dead_code dead_code_elimination.c
asecomputerlab@asecomputerlab-hp-prodesk-400-g7-micrtower-pc:~/Desktop/lab$ ./dead_code
Enter the Number of Values: 4
left: a
right: b+c
left: d
right: a+e
left: f
right: d+g
left: h
right: a+i

Intermediate Code
a = b+c
d = a+e
f = d+g
h = a+i

After Dead Code Elimination
a = b+c
d = a+e
h = a+i

After Eliminating Common Expressions
a = b+c
d = a+e
h = a+i

Optimized Code
a = b+c
d = a+e
h = a+i
```

Result: Thus, the program to implement Code Optimization Techniques has been executed successfully.