Date: 15/09/2025

Experiment No: 07

Aim: To implement Symbol Table.

Code:

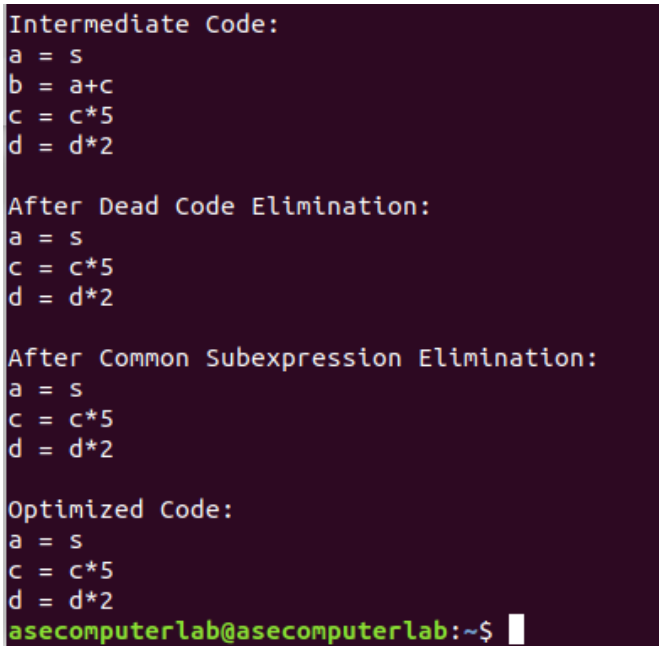
```c
#include <stdio.h>
#include <string.h>

struct op {
    char l;         // Left variable
    char r[20];     // Right expression
} op[10], pr[10];

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

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

    // Input expressions
    for (i = 0; i < n; i++) {
        printf("left: ");
        scanf(" %c", &op[i].l);        // read left variable
        printf("right: ");
        scanf("%s", op[i].r);          // read right expression (string)
    }

    // Print initial intermediate code
    printf("\nIntermediate Code:\n");
    for (i = 0; i < n; i++) {
        printf("%c = %s\n", op[i].l, op[i].r);
    }

    // Dead code elimination:
    // Keep expressions whose left variables are used on RHS of later expressions
    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++;
                break;  // variable used; keep and stop searching further
            }
```

```
        if (p) {
            pr[z].l = op[i].l;
            strcpy(pr[z].r, op[i].r);
            z++;
            break;  // variable used; keep and stop searching further
        }
    }
}
// Always keep last expression (assuming it has side effect)
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:
// Replace variables in RHS with their equivalent left variable if RHS matches

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;
            // Replace all occurrences of 't' with pr[m].l in RHS expressions
            for (i = 0; i < z; i++) {
                l = strchr(pr[i].r, t);
                while (l) {
                    int a = l - pr[i].r;
                    pr[i].r[a] = pr[m].l;
                    l = strchr(pr[i].r + a + 1, t);  // find next occurrence
                }
            }
        }
    }
}

printf("\nAfter Common Subexpression Elimination:\n");
for (i = 0; i < z; i++) {
```

```
    printf("\nAfter Common Subexpression Elimination:\n");
    for (i = 0; i < z; i++) {
        printf("%c = %s\n", pr[i].l, pr[i].r);
    }

    // Final optimization - remove duplicate expressions with same lhs and rhs
    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 == 0)) {
                pr[i].l = '\0';  // mark as deleted
            }
        }
    }

    printf("\nOptimized 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:

```
asecomputerlab@asecomputerlab:~$ gcc ex7.c
asecomputerlab@asecomputerlab:~$ ./a.out
Enter the Number of Values: 4
left: a
right: s
left: b
right: a+c
left: c
right: c*5
left: d
right: d*2

Intermediate Code:
a = s
b = a+c
```



Results: The program in YACC for parser generation has been executed successfully