CSE23CT302- THEORY OF COMPUTATION AND COMPILER DESIGN GitHub Assignment 1.
Implement a program that detects redundant computations, dead code, and strength reduction.

Github link: https://github.com/abeejay13/Abeejay

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>


#define MAX_LINES 100

#define MAX_LEN 256


void trim(char *str) {

    char *start = str;

    while (isspace((unsigned char)*start)) start++;

    if (start != str) memmove(str, start, strlen(start) + 1);


    if (*str == 0) return;
```

```
    char *end = str + strlen(str) - 1;
    while (end > str && isspace((unsigned char)*end)) end--;
    *(end + 1) = 0;
}

int evaluate(const char *expr, int *result) {
    int a, b;
    char op;
    if (sscanf(expr, "%d %c %d", &a, &op, &b) == 3) {
        switch (op) {
            case '+': *result = a + b; return 1;
            case '-': *result = a - b; return 1;
            case '*': *result = a * b; return 1;
            case '/': if (b != 0) { *result = a / b; return 1; }
        }
    }
    return 0;
}

int main() {
    char code[MAX_LINES][MAX_LEN];
```

```
char optimized[MAX_LINES][MAX_LEN];
char final[MAX_LINES][MAX_LEN];
char vars[MAX_LINES][MAX_LEN];
char vals[MAX_LINES][MAX_LEN];
int used[MAX_LINES] = {0};
int n = 0, opt_n = 0, final_n = 0;

printf("Enter code lines (e.g., x = 2 * 8). Type 'done' to finish:\n");
while (n < MAX_LINES) {
    fgets(code[n], MAX_LEN, stdin);
    trim(code[n]);
    if (strcmp(code[n], "done") == 0) break;
    n++;
}

for (int i = 0; i < n; i++) {
    char lhs[MAX_LEN], rhs[MAX_LEN];
    if (sscanf(code[i], "%s = %[^\n]", lhs, rhs) == 2) {
        trim(rhs);

        for (int j = 0; j < opt_n; j++) {
            char *pos = strstr(rhs, vars[j]);
```

```
                if (pos) {
                    char temp[MAX_LEN];
                    snprintf(temp, MAX_LEN, "%.*s%s%s", (int)(pos - rhs), rhs,
vals[j], pos + strlen(vars[j]));
                    strcpy(rhs, temp);
                }
            }

            int val;
            if (evaluate(rhs, &val)) {
                sprintf(rhs, "%d", val);
            }

            if (strstr(rhs, "* 1")) {
                char *pos = strstr(rhs, "* 1");
                *pos = '\0';
                trim(rhs);
            } else if (strstr(rhs, "1 *")) {
                char *pos = strstr(rhs, "1 *");
                strcpy(rhs, pos + 3);
                trim(rhs);
            } else if (strstr(rhs, "+ 0")) {
```

```
            char *pos = strstr(rhs, "+ 0");
            *pos = '\0';
            trim(rhs);
        } else if (strstr(rhs, "0 +")) {
            char *pos = strstr(rhs, "0 +");
            strcpy(rhs, pos + 3);
            trim(rhs);
        } else if (strstr(rhs, "- 0")) {
            char *pos = strstr(rhs, "- 0");
            *pos = '\0';
            trim(rhs);
        }

        strcpy(vars[opt_n], lhs);
        strcpy(vals[opt_n], rhs);
        sprintf(optimized[opt_n], "%s = %s", lhs, rhs);
        opt_n++;
    }
}

if (opt_n > 0) used[opt_n - 1] = 1;
for (int i = opt_n - 1; i >= 0; i--) {
```

```
        if (used[i]) {
            strcpy(final[final_n++], optimized[i]);
            for (int j = 0; j < opt_n; j++) {
                if (strstr(vals[i], vars[j])) used[j] = 1;
            }
        }
    }

    printf("\nOptimized code:\n");
    for (int i = final_n - 1; i >= 0; i--) {
        printf("%s\n", final[i]);
    }
    return 0;
}
```