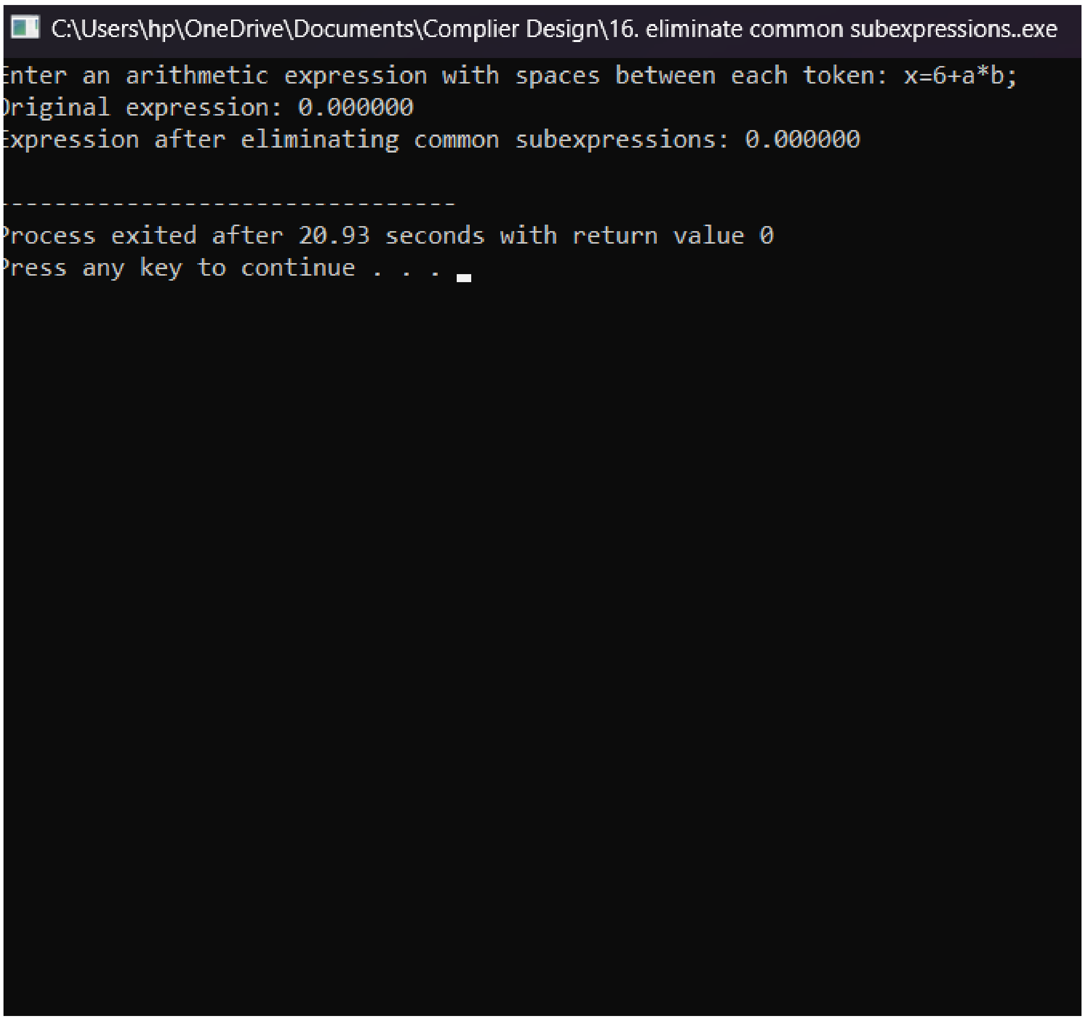
16. eliminate common subexpressions.

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

#define MAX_EXPR_LENGTH 100

typedef struct {
    char op;
    double value;
    bool isValue;
} Token;

typedef struct {
    char expression[MAX_EXPR_LENGTH];
    Token tokens[MAX_EXPR_LENGTH];
    int tokenCount;
} Expression;

void tokenize(Expression *exp) {
    char *token = strtok(exp->expression, " ");
    while (token != NULL) {
        if (token[0] == '+' || token[0] == '-' || token[0] == '*' || token[0] == '/') {
            exp->tokens[exp->tokenCount].op = token[0];
            exp->tokens[exp->tokenCount].isValue = false;
        } else {
            exp->tokens[exp->tokenCount].value = atof(token);
            exp->tokens[exp->tokenCount].isValue = true;
        }
        exp->tokenCount++;
        token = strtok(NULL, " ");
    }
}

void eliminateCommonSubexpressions(Expression *exp) {
    for (int i = 0; i < exp->tokenCount; i++) {
        if (exp->tokens[i].isValue) {
            for (int j = i + 2; j < exp->tokenCount; j += 2) {
                if (exp->tokens[j].isValue && exp->tokens[j - 2].isValue) {
                    // Found a common subexpression, eliminate it
                    double result = 0.0;
                    switch (exp->tokens[j - 1].op) {
                        case '+':
                            result = exp->tokens[j - 2].value + exp->tokens[j].value;
                            break;
                    case '-':
                        result = exp->tokens[j - 2].value - exp->tokens[j].value;
                        break;
                    case '*':
                        result = exp->tokens[j - 2].value * exp->tokens[j].value;
                        break;
                    case '/':
                        result = exp->tokens[j - 2].value / exp->tokens[j].value;
                        break;
                }
                exp->tokens[j].isValue = true;
                exp->tokens[j].value = result;
                memmove(&exp->tokens[j - 2], &exp->tokens[j + 1], (exp->tokenCount -
j - 1) * sizeof(Token));
                exp->tokenCount -= 2;
                j -= 2;
            }
        }
    }
  }
}

void printExpression(Expression *exp) {
    for (int i = 0; i < exp->tokenCount; i++) {
        if (exp->tokens[i].isValue) {
            printf("%lf", exp->tokens[i].value);
        } else {
            printf(" %c ", exp->tokens[i].op);
        }
    }
    printf("\n");
}

int main() {
    Expression exp;
    printf("Enter an arithmetic expression with spaces between each token: ");
    fgets(exp.expression, MAX_EXPR_LENGTH, stdin);
    exp.tokenCount = 0;

    tokenize(&exp);
    printf("Original expression: ");
    printExpression(&exp);

    eliminateCommonSubexpressions(&exp);
    printf("Expression after eliminating common subexpressions: ");
    printExpression(&exp);
    return 0;
}
```

Output:



C:\Users\hp\OneDrive\Documents\Complier Design\16. eliminate common subexpressions..exe

```
Enter an arithmetic expression with spaces between each token: x=6+a*b;
Original expression: 0.000000
Expression after eliminating common subexpressions: 0.000000

--------------------------------
Process exited after 20.93 seconds with return value 0
Press any key to continue . . .
```