```c
#include <stdio.h>

int main() {
        // Let us first check for associativity of the subtraction operator
    int a, b = 5, c = 3, d = 2;
    a = b - c - d;
    printf("%d\n",a);
        // Output is 0 which means the above expression is bracketed as
        // a = (b - c) - d
        // The +, -, *, /, % operators are all left associative
        // We can explicitly bracket ourselves in a different manner as shown below
        // a = b - (c - d)

    int e = 10 - 6 + 5 * 4 % 2;
    // The above expression is bracketed as
        // e = ((10 - 6) + ((5 * 4) % 2))
    printf("%d\n",e);

        // The unary negation operator is right associative
    int f = -(-(-e));
    printf("%d\n",f);

        // The assignment operator is right associative
    a = b = c = 9;
        // The above expression is bracketed as
        // (a = (b = (c = 9)))
    printf("%d\n",a);
        // We get a = 9
        // If = had been left associative, we would have had a = 5 since b = 5

        // Note that the assignment operator = also produces a side value, the same as it just
assigned
        // f = 10 assigns f to 10 but also generates a side value 10
        // In g = f = 10, it is this side value that gets assigned to g
    printf("%d\n",f = 10);

        // Be careful about mixed type, mixed operator expressions
    printf("%f\n",1/3 * 3.0);
        // The expression 1/3 * 3.0 is evaluated, according to BODMAS and associativity rules, as
        // ((1/3) * 3.0) which becomes
        // (0 * 3.0) since 1/3 is integer/integer division so no automatic typecasting. This
becomes
        // (0.0 * 3.0) due to automatic typecasting which becomes
        // 0.0
    return 0;
}
```