Code 1:

Task1: limits never imported, &s not used during scans

Problematic because INT\_MAX and INT\_MIN are never declared as a result, and ints int1 and int2 are never properly stored given their memory is never referenced.

Task2:

A test case would be trying to store ints in int1 and int2.. but you cant even measure that because before getting to that an error occurs

in which int\_max and min are not declared

test cases therefore assume limits.h has been imported

int 1: 2

int 2: 4

should print 6 when corrected

For task 4, when corrected, the output is 2147483646, which on the lab is listed as the expected result, but it should be an overflow.

//

#include <stdio.h>

int main()

{

    int int1;

    printf("Enter an integer: ");

    scanf("%d", &int1);

    printf("%d\n", int1);

    int int2 = int1 + -2;

    printf("%d", int2);

    return 0;

}

this shows you that 2147483648 is already outside the bounds and already overflows to -2147483648

so when you add -2 to -2147483648, overflow occurs AGAIN, which is why 2147483646 gets output, and not overflow. code2 goes over this issue a bit more and handles the test case correctly I believe.

Code 2:

Task1:

*if* ((int2 > INT\_MAX - int1) || (int2 < INT\_MIN - int1)) {

should be

*if* ((int2 > INT\_MAX - int1) || (int2 < INT\_MIN + int1)) {

int\_min - int1 was corrected to be INT\_MIN + int1, as in the original case you would be subtracting a negative and check against above the intmin. Therefore negative overflow or underflow would not be properly measured

Task2:

test case: int 1 = -2, int2 = -2147483648

Description: Testing overflow

Enter an integer: -2

Enter another integer: -2147483647

both are within integer bounds

Expected result: Integer overflow occurred

For task 4, integer overflow is printed, which I think is correct, given that 2147483648 is out of bounds already so integer overflow already occurred. Because the input was outside the acceptable range, it already wrapped over and caused overflow. I wrote the description for task4 in the code before doing code1, so now I also understand that the overflow occurs very early in the process.