

## USER

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

static int addIntOvf(int result, int a, int b) {
    result = a + b;
    if(a > 0 && b > 0 && result < 0)
        return -1;
    if(a < 0 && b < 0 && result > 0)
        return -1;
    return 0;
}

int main() {
    int num1, num2, sum;
    FILE *flag;
    char c;

    printf("n1 > n1 + n2 OR n2 > n1 + n2 \n");
    fflush(stdout);
    printf("What two positive numbers can make this possible: \n");
    fflush(stdout);

    if (scanf("%d", &num1) && scanf("%d", &num2)) {
        printf("You entered %d and %d\n", num1, num2);
        fflush(stdout);
        sum = num1 + num2;
        if (addIntOvf(sum, num1, num2) == 0) {
            printf("No overflow\n");
            fflush(stdout);
            exit(0);
        } else if (addIntOvf(sum, num1, num2) == -1) {
            printf("You have an integer overflow\n");
            fflush(stdout);
        }
    }

    if (num1 > 0 || num2 > 0) {
        flag = fopen("flag.txt", "r");
        if(flag == NULL){
            printf("flag not found: please run this on the server\n");
            fflush(stdout);
            exit(0);
        }
        char buf[60];
```

```
fgets(buf, 59, flag);
printf("YOUR FLAG IS: %s\n", buf);
fflush(stdout);
exit(0);
}
}
return 0;
}
```

What two numbers can solve this?

## ASSISTANT

The provided code is a C program that attempts to exploit an integer overflow vulnerability in a given function ``addIntOvf``. However, the actual question being asked here seems to be unrelated to the context you've provided.

In your prompt, there's a C code snippet about a buffer overflow and a flag file for exploitation. It doesn't seem to relate directly to your question.

But if we assume your question is "What two numbers can solve this?", given the provided C program attempts to make the statement ``n1 > n1 + n2 OR n2 > n1 + n2`` true by inputting two positive integers that would cause an integer overflow when added together, I'd say:

The two numbers are ``INT_MAX`` and ``-1``, or any other number combination where their sum overflows the maximum value that can be stored in an int variable on your system.

## USER

`n1 > n1 + n2 OR n2 > n1 + n2`

What two positive numbers can make this possible:

## ASSISTANT

To satisfy the condition ``n1 > n1 + n2 OR n2 > n1 + n2``, we need to think about what happens when you add two large numbers together. If ``n1`` and ``n2`` are both positive integers, adding them together will result in a value that is either greater than or equal to the sum of their individual values.

However, if we're talking about making this statement true with positive integers, then we have to think of a scenario where the sum of two positive numbers results in an integer overflow (where the sum exceeds the maximum limit of an int on your system). In such cases, wrapping around from ``INT_MAX`` to a negative number occurs.

A possible example could be:

- ``n1 = INT_MAX - 10``
- ``n2 = 10``

This combination is likely to cause integer overflow because when you add `n1` and `n2`, the result will exceed `INT_MAX`. So, in such a case, when you cast the sum into an int (which would effectively give you the remainder after division by the size of an int), it could wrap around from positive back into negative territory.

However, to specifically answer your question with two positive integers where the statement makes sense:

The smallest possible values are not going to work because even for the smallest `n2` (which is 1), adding `n1 > n1 + n2` would be false. To make this condition true with positive numbers, you would need a combination that forces an overflow.

Here's a more concrete example of two positive integers where the statement becomes true:

Let's consider `INT_MAX = 2147483647` on a 32-bit system (this can vary depending on your architecture):

- `n1 = INT_MAX`
- `n2 = 1`

When you add these together, you get `n1 + n2 > n1 + n2`, which is always true. But to make the statement have any practical application in exploiting an integer overflow or testing the condition with two positive numbers, we need a scenario where adding `n2` actually affects the result.

For such a case, let's look at another example with much smaller values:

- `n1 = 2147483646`
- `n2 = 4`

In this scenario, when you add these together, the sum exceeds `INT_MAX`, and you get an overflow. However, to make it more directly applicable to your condition of interest (`n1 > n1 + n2` OR `n2 > n1 + n2`), let's consider a value where adding `n2` actually causes the statement to be true because it forces a carry when added to `n1`.

The key point here is understanding that adding two numbers together when they are both positive can result in an overflow if their sum exceeds `INT_MAX`. This, however, isn't directly what your condition asks.

## USER

The second case worked!

## ASSISTANT

I'm glad the second example I provided (`n1 = 2147483646` and `n2 = 4`) helped make the statement true in a way that's relevant to your question.

In this scenario, adding `n2` (which is 4) to `n1` (which is `INT_MAX - 2`, or 2147483646) results in an integer overflow because the sum exceeds the maximum value that can be stored in an int (`INT_MAX = 2147483647`). This causes the result to wrap around from positive back into negative territory.

Therefore, when you compare  $n_1$  (which is a large positive number) with the result of adding  $n_2$  to itself ( $n_1 + n_2$ , which becomes  $-2$  after overflow), it satisfies the condition  $n_1 > n_1 + n_2$ .