Write a function that returns the factorial of a given number.

Prototype: int factorial(int n);
If n is lower than 0, the function should return -1 to indicate an error
Factorial of 0 is 1
julien@ubuntu:~/0x08. Recursion$ cat 3-main.c

```
#include "main.h"
#include <stdio.h>

/**
* main - check the code
*
* Return: Always 0.
*/
int main(void)
{
int r;

r = factorial(1);
printf("%d\n", r);
r = factorial(5);
printf("%d\n", r);
r = factorial(10);
printf("%d\n", r);
r = factorial(-1024);
printf("%d\n", r);
return (0);
}
```

julien@ubuntu:~/0x08. Recursion$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 3-main.c 3-factorial.c -o 3-factorial
julien@ubuntu:~/0x08. Recursion$ ./3-factorial
1
120
3628800
-1

my solution

```
int factorial(int n)
{
if (n < 0)
{
return (-1);
}
if (n == 0)
{
```

```
return (1);
}

return (n * factorial(n - 1));
}
```

Your solution is correct! The function `factorial` you've provided is a recursive implementation that computes the factorial of a given integer `n`.

Here's a breakdown of how the function works:

1. If `n` is negative, it returns `-1` to indicate an error.
2. If `n` is `0`, it returns `1` because the factorial of `0` is `1`.
3. Otherwise, it computes the factorial of `n` by multiplying `n` with the factorial of `n-1`. The function then calls itself with `n-1` as the argument to compute the factorial of `n-1`, and so on until `n` is `0`.

The test cases you've provided demonstrate the function's correctness for various inputs:

1. The factorial of `1` is `1`.
2. The factorial of `5` is `5 * 4 * 3 * 2 * 1 = 120`.
3. The factorial of `10` is `10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 3,628,800`.
4. For negative input `-1024`, the function returns `-1` as an error.

So, your solution is correct and works as expected!

brack it down with all details n = 3

Of course! Let's break down the factorial calculation for $n = 3$n = 3 using your recursive function.

When $n = 3$n = 3:

```
int factorial(int n)
{
        if (n < 0)    // Not triggered since 3 is not less than 0
        {
                return (-1);
        }
        if (n == 0)   // Not triggered since 3 is not 0
        {
                return (1);
        }

        return (n * factorial(n - 1));  // 3 * factorial(3 - 1)
}
```

On the line `return (n * factorial(n - 1));`, nn is 3. So the function will return $3 \times \text{factorial}(3 - 1)$3 × factorial(3 − 1). Now, $\text{factorial}(3 - 1)$factorial(3 − 1) needs to be evaluated.

For $n = 2$n = 2:

```
int factorial(int n)
{
        if (n < 0)    // Not triggered since 2 is not less than 0
        {
                return (-1);
        }
        if (n == 0)   // Not triggered since 2 is not 0
        {
                return (1);
        }

        return (n * factorial(n - 1));  // 2 * factorial(2 - 1)
}
```

On the line `return (n * factorial(n - 1));`, nn is 2. So this will return $2 \times \text{factorial}(2 - 1)$ 2 × factorial(2 − 1). Now, $\text{factorial}(2 - 1)$factorial(2 − 1) needs to be evaluated.

For $n = 1$n = 1:

```
int factorial(int n)
{
        if (n < 0)    // Not triggered since 1 is not less than 0
        {
                return (-1);
        }
        if (n == 0)   // Not triggered since 1 is not 0
        {
                return (1);
        }

        return (n * factorial(n - 1));  // 1 * factorial(1 - 1)
}
```

On the line `return (n * factorial(n - 1));`, nn is 1. So this will return $1 \times \text{factorial}(1 - 1)$ 1 × factorial(1 − 1). Now, $\text{factorial}(1 - 1)$factorial(1 − 1) needs to be evaluated.

Finally, for $n = 0$n = 0:

```
int factorial(int n)
{
        if (n < 0)    // Not triggered since 0 is not less than 0
        {
                return (-1);
        }
        if (n == 0)   // This is triggered since n is 0
        {
                return (1);
        }
        // This part will not be executed for n = 0
        return (n * factorial(n - 1));
}
```

For $n = 0$n = 0, the function directly returns 1 because of the condition `if (n == 0)`.

Combining all these recursive returns:

1. $\text{factorial}(3) = 3 \times \text{factorial}(2)$factorial(3) = 3 × factorial(2)
2. $\text{factorial}(2) = 2 \times \text{factorial}(1)$factorial(2) = 2 × factorial(1)
3. $\text{factorial}(1) = 1 \times \text{factorial}(0)$factorial(1) = 1 × factorial(0)

4. $\text{factorial}(0) = 1$factorial(0) = 1

The calculation becomes:

1. $3 \times (2 \times (1 \times 1))$3 × (2 × (1 × 1))
2. $3 \times (2 \times 1)$3 × (2 × 1)
3. $3 \times 2$3 × 2
4. $6$6

Thus, $\text{factorial}(3) = 6$factorial(3) = 6.

Is this conversation helpful so far?

From ChatGPT, designed with FancyGPT