**Quest06**

Remember to git add && git commit && git push each exercise!

We will execute your function with our test(s), please DO NOT PROVIDE ANY TEST(S) in your file

For each exercise, you will have to create a folder and in this folder, you will have additional files that contain your work. Folder names are provided at the beginning of each exercise under submit directory and specific file names for each exercise are also provided at the beginning of each exercise under submit file(s).

**My Iterative Pow**

* Submit directory: ex00
* Submit file: ["my\_iterative\_pow.c"]

2 ^ 2 => 4 2 ^ 3 => 8 2 ^ 4 => 16

Let's create a function to calculate the pow of a number!

Create an iterated function that returns the value of a power applied to a number. An power lower than 0 returns 0. Overflows don't have to be handled. First parameter is the number, second parameter is the power You have to use a loop (for/while/...) to perform this exercise

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_iterative\_pow

\*\*

\*\* @param {int} param\_1

\*\* @param {int} param\_2

\*\*

\*\* @return {int}

\*\*

\*/

int my\_iterative\_pow(int param\_1, int param\_2)

{

}

**Example 00**

Input: 2 && 2

Output:

Return Value: 4

**Example 01**

Input: 2 && 3

Output:

Return Value: 8

**Example 02**

Input: 2 && 4

Output:

Return Value: 16

**My Recursive Pow**

* Submit directory: ex01
* Submit file: ["my\_recursive\_pow.c"]

2 ^ 2 => 4 2 ^ 3 => 8 2 ^ 4 => 16

Let's create a function to calculate the pow of a number!

Create an iterated function that returns the value of a power applied to a number. An power lower than 0 returns 0. Overflows don't have to be handled. First parameter is the number, second parameter is the power You have to use the recursive method to perform this exercise. While / for / any loop are forbidden.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_recursive\_pow

\*\*

\*\* @param {int} param\_1

\*\* @param {int} param\_2

\*\*

\*\* @return {int}

\*\*

\*/

int my\_recursive\_pow(int param\_1, int param\_2)

{

}

**Example 00**

Input: 2 && 2

Output:

Return Value: 4

**Example 01**

Input: 2 && 3

Output:

Return Value: 8

**Example 02**

Input: 2 && 4

Output:

Return Value: 16

*Tip* Google the following: recursive programming

**My Iterative Factorial**

* Submit directory: ex02
* Submit file: ["my\_iterative\_factorial.c"]

2! => 2 x 1 => 2 3! => 3 x 2 x 1 => 6 4! => 4 x 3 x 2 x 1 => 24

Let's create a function to calculate the factorial of a number!

Create an iterated function that returns a number. This number is the result of a factorial operation based on the number given as a parameter.

If there's an error, the function should return 0. You have to use a loop (for/while/...) to perform this exercise

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_iterative\_factorial

\*\*

\*\* @param {int} param\_1

\*\*

\*\* @return {int}

\*\*

\*/

int my\_iterative\_factorial(int param\_1)

{

}

**Example 00**

Input: 2

Output:

Return Value: 2

**Example 01**

Input: 3

Output:

Return Value: 6

**Example 02**

Input: 4

Output:

Return Value: 24

**My Recursive Factorial**

* Submit directory: ex03
* Submit file: ["my\_recursive\_factorial.c"]

2! => 2 x 1 => 2 3! => 3 x 2 x 1 => 6 4! => 4 x 3 x 2 x 1 => 24

Let's create a function to calculate the factorial of a number!

Create an iterated function that returns a number. This number is the result of a factorial operation based on the number given as a parameter.

If there's an error, the function should return 0. You have to use the recursive method to perform this exercise. While / for / any loop are forbidden.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_recursive\_factorial

\*\*

\*\* @param {int} param\_1

\*\*

\*\* @return {int}

\*\*

\*/

int my\_recursive\_factorial(int param\_1)

{

}

**Example 00**

Input: 2

Output:

Return Value: 2

**Example 01**

Input: 3

Output:

Return Value: 6

**Example 02**

Input: 4

Output:

Return Value: 24

*Tip* Google the following: recursive programming

**My Atoi**

* Submit directory: ex04
* Submit file: ["my\_atoi.c"]

The atoi() function in C takes a string (which represents an integer) as an argument and returns its value of type int. So basically the function is used to convert a string argument to an integer. Syntax:

int atoi(const char strn)

Parameters: The function accepts one parameter strn which refers to the string argument that is needed to be converted into its integer equivalent.

Return Value: If strn is a valid input, then the function returns the equivalent integer number for the passed string number. If no valid conversion takes place, then the function returns zero.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_atoi

\*\*

\*\* @param {char\*} param\_1

\*\*

\*\* @return {int}

\*\*

\*/

int my\_atoi(char\* param\_1)

{

}

**Example 00**

Input: "2"

Output:

Return Value: 2

**Example 01**

Input: "123"

Output:

Return Value: 123

**Example 02**

Input: "-10"

Output:

Return Value: -10

*Tip* (In C) Split the number by dividing it, and to get the rest are you aware of the mod operator? You should google it :)

**My Fibonacci**

* Submit directory: ex05
* Submit file: ["my\_fibonacci.c"]

Create a function my\_fibonacci that returns the n-th element of the Fibonacci sequence, the first element being at the 0 index. We'll consider that the Fibonacci sequence starts like this: 0, 1, 1, 2.

If the value is less than 0, the function should return -1.

It should be prototyped:

Recursive will be helpful here.

Google fibanacci.

**Function prototype** (c)

/\*

\*\*

\*\* QWASAR.IO -- my\_fibonacci

\*\*

\*\* @param {int} param\_1

\*\*

\*\* @return {int}

\*\*

\*/

int my\_fibonacci(int param\_1)

{

}

**Example 00**

Input: 2

Output:

Return Value: 1

**Example 01**

Input: 3

Output:

Return Value: 2

**Example 02**

Input: 4

Output:

Return Value: 3

*Tip* Google the following: recursive programming