



EXERCISES — Hanoi

version #7be580532266ed398481e31366afcc24b1950c2a



**The way is lit. The path is clear.
We require only the strength to follow it.**

Copyright

This document is for internal use at EPITA ([website](#)) only.

Copyright © 2022-2023 Assistants <assistants@tickets.assistants.epita.fr>

The use of this document must abide by the following rules:

- ▷ You downloaded it from the assistants' intranet.*
- ▷ This document is strictly personal and must **not** be passed onto someone else.
- ▷ Non-compliance with these rules can lead to severe sanctions.

Contents

1	Goal	3
2	Example	4

*<https://intra.assistants.epita.fr>

File Tree

```
hanoi/  
└─ hanoi.c (to submit)
```

Authorized functions : You are only allowed to use the following functions

- putchar(3)
- printf(3)

Authorized headers : You are only allowed to use the functions defined in the following headers

- err.h
- errno.h
- assert.h
- stddef.h

Compilation : Your code must compile with the following flags

- -std=c99 -pedantic -Werror -Wall -Wextra -Wvla

Main function : None

1 Goal

The goal here is to implement a function capable of solving an instance of the [tower of Hanoi problem](#). This function must have the following prototype:

```
void hanoi(unsigned n);
```

In the tower of Hanoi problem, you have three 'towers' on which differently sized rings are stacked on one another. A ring can only be on a larger one (or on the base if it is the first). The problem is solved when all the rings have been moved from the first to the last tower while respecting the previous rule and only moving the rings once at a time.

Your `hanoi` function must display the movements needed to solve the problem. You must display the solution with the least movements.

The `n` parameter of your `hanoi` is the number of rings on the first tower at the beginning. Its output is the sequence of movements needed to solve the problem in the form `X->Y` where `X` is the number of the origin tower and `Y` the number of the destination one.

For instance, `hanoi(3)` prints:

```
1->3  
1->2  
3->2  
1->3  
2->1  
2->3  
1->3
```

2 Example

To help you visualize, here are the steps of the algorithm for `hanoi(2)`:

- Initial state:



- Move 1→2:



- Move 1→3:



- Move 2→3:



The way is lit. The path is clear. We require only the strength to follow it.