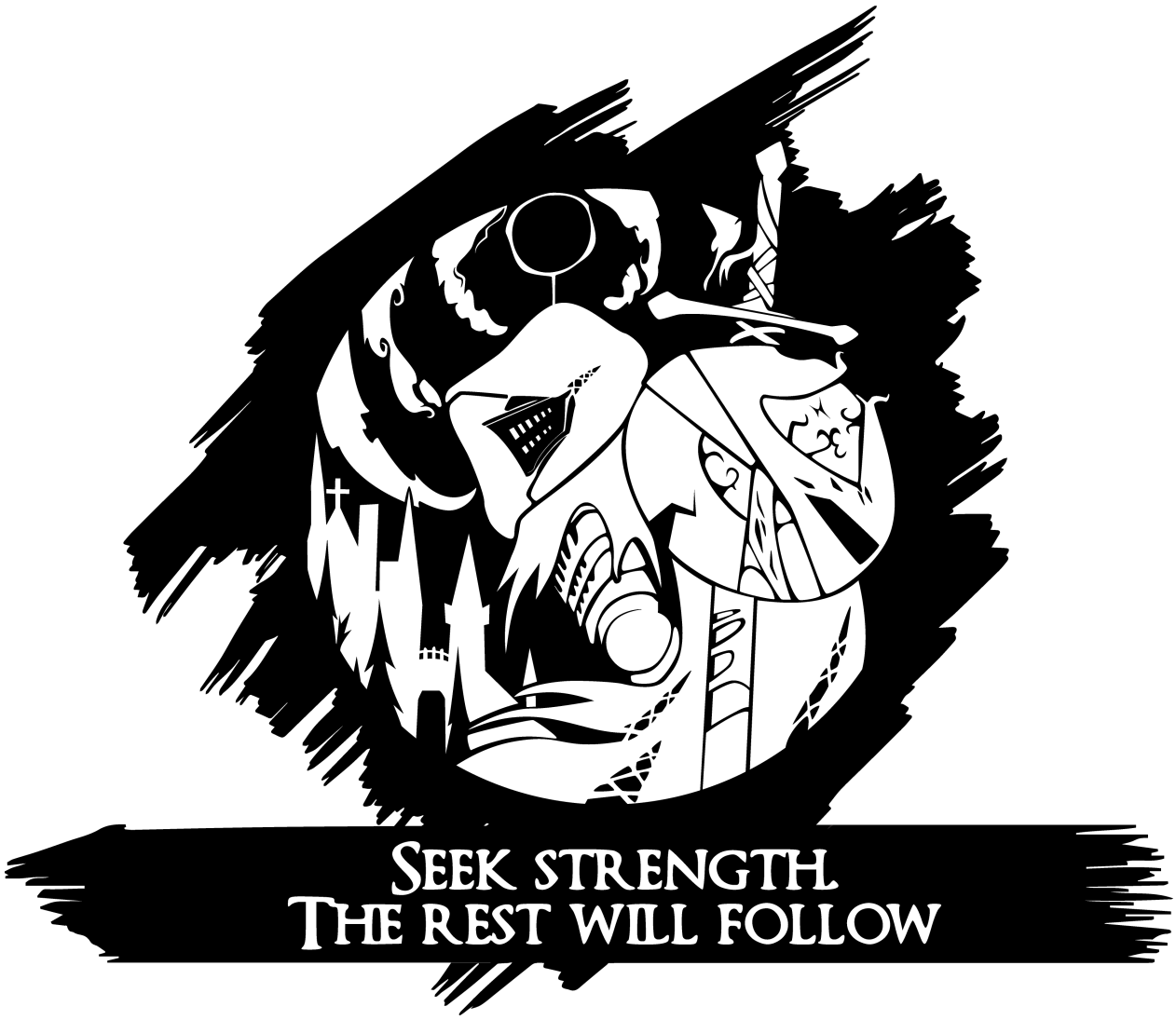
Exercise — Alignment

version **#bfdaecd01cbf44dfbd7800b940343997d9ad7d73**



Assistants C/Unix 2025 <assistants@tickets.assistants.epita.fr>

Copyright

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

Contents

*https://intra.forge.epita.fr

File Tree

```
alignment/
├── alignment.c   (to submit)
└── alignment.h
```

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

1 Definition

Tips

A memory block that is aligned on `n` have its starting address as a multiple of `n`. Using this information to your advantage will save you a lot of time when working on malloc.

Alignement is an implementation detail of `malloc(3)`, which makes some memory operations more efficient. In fact, on some architectures (e.g. x86), certain memory operations can only be performed on memory addresses that are a multiple of the size of the data type being operated on.

For example, if you want to perform a 64-bit integer addition on two values that are stored in memory, they must be aligned on a 8-byte boundary in order for the operation to be possible. Similarly, if you want to load a double-precision floating point value from memory into a register, it must be aligned on a 4-byte boundary. And if you want to load a `long double` value from memory, it must be aligned on an 8-byte boundary.

Therefore, by aligning the memory returned by `malloc(3)` on the size of a `long double`, it ensures that the memory can be used for any type of data, without the need to perform any extra memory operations to align the data first.

If you want more information you can see the x86_64 ABI and the Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 1.

Tips

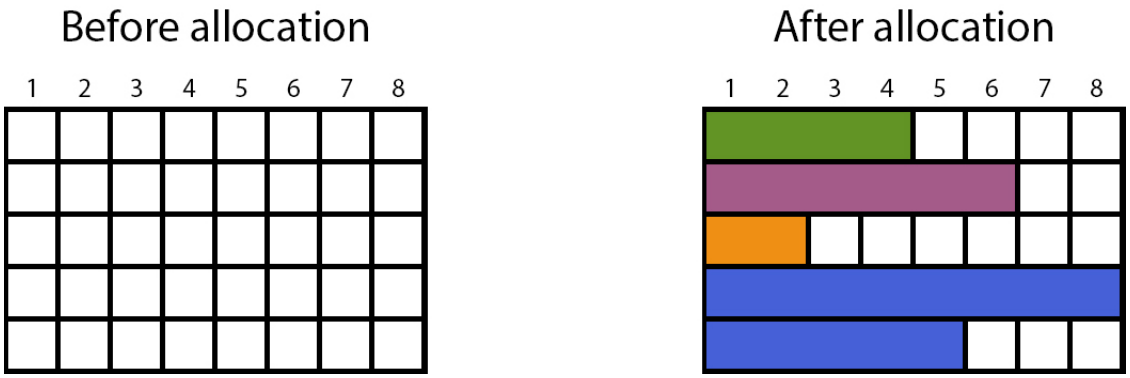
We use padding to align the addresses thus not affecting the allocated size. Padding is also used in malloc to add space for metadata to the beginning or end of the allocated memory.

2 Example

For a better understanding, assuming that we want to align our memory on `8`, if we execute the following code:

```
void *green = malloc(4);
void *purple = malloc(6);
void *orange = malloc(2);
void *blue = malloc(13);
```

here is what the memory will looks like:



3 Goal

The goal of this exercise is to align the size of the memory we want to allocate to follow the above-mentioned specification. In order to do that, you have to implement the following function:

```
size_t align(size_t size);
```

The new size returned must be the closest greater number aligned on `sizeof(long double)`. Moreover, you have to check any potential overflow. For this purpose you must use the gcc builtin functions. The online documentation is available here, read it and choose the right function to use. In case of overflow you must return 0.

Seek strength. The rest will follow.