Exercises — Functional Programming

version **#7be580532266ed398481e31366afcc24b1950c2a**

Assistants C/Unix 2023 <assistants@tickets.assistants.epita.fr>

Copyright

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

Contents

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

File Tree

```
functional_programming/
├── foldl.c  (to submit)
├── foldr.c  (to submit)
├── functional_programming.h
└── map.c  (to submit)
```

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

In this exercise you will write your first basic functional functions.

1.1 `map`

Write the `map` function, to apply a function (`func`) to every element of an `int` array.

```
void map(int *array, size_t len, void (*func)(int *));
```

For example:

```
void times_two(int *a)
{
    *a *= 2;
}

int main(void)
{
    int arr[] = {1, 4, 7};
    map(arr, 3, times_two);
    // arr == {2, 8, 14}
}
```

1.2 `foldr`

As for the previous function, `foldr` takes a function to apply to each element of the array. However, the function also takes an accumulator as parameter, initially 0. For example, on the array `{1, 2, 3}`:

`foldr(sum, {1, 2, 3}) <==> sum(1, sum(2, sum(3, 0)))`

The call to `sum(3, 0)` is the new accumulator for `sum(2, 3)`. The function finally returns the last value of the accumulator (`6` in our case).

```
int foldr(int *array, size_t len, int (*func)(int, int));
```

1.3 `foldl`

`foldl` is similar to `foldr`, but traverses the array backwards:

`foldl(sum, {1, 2, 3}) <==> sum(sum(sum(0, 1), 2), 3)`

Here is the prototype:

```
int foldl(int *array, size_t len, int (*func)(int, int));
```

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