Exercises — Functional Programming Advanced

version **#7be580532266ed398481e31366afcc24b1950c2a**

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

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

Copyright

Contents

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

File Tree

```
functional_programming_advanced/
├─ all.c   (to submit)
├─ any.c   (to submit)
├─ filter.c   (to submit)
├─ functional_programming_advanced.h
├─ max.c   (to submit)
└─ print_even.c   (to submit)
```

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

- malloc(3)
- calloc(3)
- free(3)
- realloc(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

Be careful!

This exercise is the next step after having completed the *functional_programming* exercise. Make sure you have already completed it, because you will need some of the functions you wrote in the first part.

In this exercise you will continue to implement functional functions.

1.1 `all`

Write the `all` function, that iterates over the list, and checks whether the predicate (function returning `true` or `false`) is `true` for every element.

For example, if you have the predicate `is_even` (which returns whether or not a number is even), `all(is_even, {1, 2, 3})` returns `false` while `all(is_even, {2, 4, 6})` returns `true`.

```
bool all(int *array, size_t len, bool (*func)(int));
```

1.2 `any`

Write the `any` function that traverses the list, and returns `true` if the predicate returns `true` for at least one element of the array.

For example, `any(is_even, {1, 2, 3})` is `true`, while `any(is_even, {1, 5, 7})` is `false`.

```
bool any(int *array, size_t len, bool (*func)(int));
```

1.3 `filter`

Write the `filter` function. It takes an `int` array `array` as parameter, the number of elements `len`, a pointer to the `int` array `out_array` that will contain the results, and the predicate `func`. The array `out_array` must contain every element of `array` for which `func` returns `true`. `filter` returns the size of the new array.

As `out_array` is a pointer to an array, it is your responsibility to allocate `out_array` in the `filter` function. Do not forget to `free` the result!

```
size_t filter(int *array, size_t len, int **out_array, bool (*func)(int));
```

1.4 `print_even`

This function is a good example use of the previous functions.

Write `print_even`, that will display every even element of the array, one per line. For this, you will need to use `map` and `filter`.

```
void print_even(int *array, size_t len);
```

1.5 `max`

Write a function that will find the maximum value of an `int` array. For this, you will need to use either `foldl` or `foldr`.

```
int max(int *array, size_t len);
```

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