Exercises — list

version #

Assistants C/Unix 2022 <assistants@tickets.assistants.epita.fr>

Copyright

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

Contents

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

1 list

Files to submit:

- list/list.c
- list/list_print.c
- list/list.h

Main function: None

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

- malloc(3)
- free(3)
- printf(3)

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

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

1.1 Goal

You have to implement the behavior of a simple linked-list.

You will use the following structure:

```
struct list
{
    int data;
    struct list *next;
};
```

1.1.1 Insertion

This function adds the `e` element in front of the list `l` and returns the head of the list after the insertion. If any error occurs, you have to return `l` unchanged.

```
struct list *list_add(struct list *l, int e);
```

1.1.2 Print

This function prints every element of the list `l`, each separated from the previous one by a space, followed by a line feed.

```
void list_print(struct list *l);
```

You should print a line feed if there is no element in the list.

```
42sh$ ./list_print | cat -e
1 2 3 4 5 6 7 8 9 10 11 12$
```

1.1.3 Find

This function returns a pointer to the first link that contains the element `e` or `NULL` if this element is not found. It also returns `NULL` if `l` is `NULL`.

```
struct list *list_find(struct list *l, int e);
```

1.1.4 Remove

This function removes the first occurrence of `e` in `l` and returns `l`'s head after the removal. If `l` is `NULL`, `list_remove` returns `NULL`.

```
struct list *list_remove(struct list *l, int e);
```

1.1.5 Reverse sorted insertion

This function inserts the `e` element to its rightful place in `l`, sorted in a **decreasing** order. It returns the head of the list after the insertion. If any error occurs, you have to return `l` as it was when your function was called.

```
struct list *list_reverse_sorted_add(struct list *l, int e);
```

1.1.6 Remove if

This function removes from the list all elements for which the predicate returns `1`. It returns `l`'s head after removal. If `l` is `NULL`, `list_remove_if` returns `NULL`.

```
struct list *list_remove_if(struct list *l, int (*predicate)(int));
```

It is my job to make sure you do yours.