{EPITECH}
LEARN DIFFERENT*

ORGANIZED

ELEMENTARY PROGRAMMING IN C

* apprendre autrement

ORGANIZED

binary name: organized
language: C
compilation: via Makefile, including re, clean and fclean rules
Authorized functions: open, read, write, close, malloc, free

- ✓ The totality of your source files, except all useless files (binary, temp files, objfiles,...), must be included in your delivery.
- ✓ All the bonus files (including a potential specific Makefile) should be in a directory named bonus.
- ✓ Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

> Having set up your laboratory infrastructure, you're now faced with a bunch of tools, electronic components and documents scattered across your brand-new work surfaces. This is your development arsenal, but in its current state, it's quite a mess.
>
> Imagine that every tool is an idea, every component is an opportunity, and every document is an inspiration. If you don't organize them, how can you make the most of these resources? Genius without order is just wasted potential.
> Your mission is to create a system that not only sorts these items, but also makes it easy to find them. This is where linked lists come in. Think of them as the shelves of your virtual workspace, where each item has a place and a connection to the next.
>
> Next, you need a way to sort efficiently. Whether it's size, type, utility or some other criterion you deem relevant, your sorting algorithm will need to be up to the task of organizing this chaos.
> By mastering these tools, you'll not only streamline your workspace, but also refine your algorithmic thinking. It's a valuable skill that will serve you well throughout your career as a software engineer.
>
> Are you up to the challenge of bringing the necessary order so that your creativity can flourish unhindered? Your journey into the world of algorithms continues. Take the tools of disorder and forge order!
>
> **— Narrator —**

The project

Objectives

You need to be able to store and sort all the materials you have at your fingertips.
Your laboratory is really upside down and you need to organize it. This project can be divided into 2 parts:

- ✓ Storing and handling hardware
- ✓ Sorting your stuff according to tags

In fact, there's a third challenge! You're going to be handling **a lot of hardware** and your sorting algorithm needs to be able to **process a large quantity** of hardware quickly and efficiently.

Materials

Your laboratory may be a mess, but not just anything is lying around. All the materials to be organized is divided into 5 categories, mainly for robot design:

- ✓ Actuators: buttons, levers, …
- ✓ Devices: radios, watches, recorders, …
- ✓ Processors: intel, amd, …
- ✓ Sensors : movement sensor, sound sensor, thermal sensor, …
- ✓ Wires : type-c, hmdi, jack, …

In addition to a type, these materials will always have a **name** and a **unique id**.

The first id in the program must be set to 0, and incremented with each new material registered.

Shell

To do this, you'll need to manipulate your workshop through a shell, so that you can type your commands to it! The shell itself doesn't need to be developed, it's provided for you, with access to a `libshell.a` and a `shell.h`.
The `libshell.a` library contains a function to call to launch the shell:

```
int workshop_shell(void *data);
```

This function will then call the functions corresponding to the commands available in the shell. The `exit` command is already implemented but there are 4 commands to implement:

- ✓ `add`: to add new hardware
- ✓ `del`: to delete a hardware
- ✓ `disp`: to display the contents of your workshop
- ✓ `sort`: to sort the materials present in the workshop

You need to push the `libshell.a` library and the `shell.h` header, the tester will not provide them.

These 4 commands are functions that you will have to implement with the following prototypes (indicated in the `shell.h` file):

Terminal

```
B-CPE-110> cat shell.h
/*
** EPITECH PROJECT, 2023
** B-CPE-110 : Setting Up Shell
** File description:
** shell.h
*/

#ifndef SHELL_H
        #define SHELL_H

int add(void *data, char **args);
int del(void *data, char **args);
int sort(void *data, char **args);
int disp(void *data, char **args);
int workshop_shell(void *data);

#endif /* SHELL_H */
```

Storing and handling materials

For the storage and handling part, it is the development of the commands: `add`, `del` and `disp`:

add command:

```
Terminal
B-CPE-110> ./organized
Workshop » add WIRE usb
WIRE n°0 - ‘‘usb’’ added.
Workshop » add ACTUATOR button, DEVICE recorder
ACTUATOR n°1 - ‘‘button’’ added.
DEVICE n°2 - ‘‘recorder’’ added.
Workshop »
```

Hardwares are treated as in a linked list, and must be added to your list in reverse. If you add hardwares #0, #1 and #2, then they will be stored in the list as: #2, #1, #0.

del command:

```
Terminal
B-CPE-110> ./organized
Workshop » add WIRE usb, ACTUATOR button, DEVICE recorder
WIRE n°0 - ‘‘usb’’ added.
ACTUATOR n°1 - ‘‘button’’ added.
DEVICE n°2 - ‘‘recorder’’ added.
Workshop » del 1
ACTUATOR n°1 - ‘‘button’’ deleted.
Workshop » del 0, 2
WIRE n°0 - ‘‘usb’’ deleted.
DEVICE n°2 - ‘‘recorder’’ deleted.
Workshop »
```

The `args` parameter of the functions to be implemented contains an array of the command arguments. It’s up to you to deal with errors: if a command is not correct, you must stop the program and return 84.

disp command:

```
Terminal
B-CPE-110> ./organized
Workshop » add WIRE usb, ACTUATOR button, DEVICE recorder
WIRE n°0 - ‘‘usb’’ added.
ACTUATOR n°1 - ‘‘button’’ added.
DEVICE n°2 - ‘‘recorder’’ added.
Workshop » disp
DEVICE n°2 - ‘‘recorder’’
ACTUATOR n°1 - ‘‘button’’
WIRE n°0 - ‘‘usb’’
Workshop » del 1
ACTUATOR n°1 - ‘‘button’’ deleted.
Workshop » disp
DEVICE n°2 - ‘‘recorder’’
WIRE n°0 - ‘‘usb’’
Workshop »
```

Sorting your stuff

The last command to be implemented, sort, is a command for sorting all the hardwares already present. Materials can be sorted according to 3 tags:

- ✓ TYPE (ascii)
- ✓ NAME (ascii)
- ✓ ID (numerically ascending)

Each of these tags can be assigned a "-r" flag, sorting the materials in reverse.

sort command:

```
Terminal
B-CPE-110> ./organized
Workshop » add WIRE usb, ACTUATOR button, DEVICE recorder
WIRE n°0 - ‘‘usb’’ added.
ACTUATOR n°1 - ‘‘button’’ added.
DEVICE n°2 - ‘‘recorder’’ added.
Workshop » sort NAME
Workshop » disp
ACTUATOR n°1 - ‘‘button’’
DEVICE n°2 - ‘‘recorder’’
WIRE n°0 - ‘‘usb’’
Workshop » sort NAME -r
Workshop » disp
WIRE n°0 - ‘‘usb’’
DEVICE n°2 - ‘‘recorder’’
ACTUATOR n°1 - ‘‘button’’
Workshop »
```

Try to find out about all the sorting algorithms out there! Bubble sorts are simple to implement and common, but there are far more efficient ones.

Finally, you can sort your hardware with **several sorting tags**. For example, if you sort by NAME and ID, then hardware will be sorted alphabetically by NAME, then by ID if two pieces of hardware have the same name.

```
Terminal
B-CPE-110> ./organized
Workshop » add WIRE usb, WIRE type-c, WIRE usb, ACTUATOR button
WIRE n°0 - ‘‘usb’’ added.
WIRE n°1 - ‘‘type-c’’ added.
WIRE n°2 - ‘‘usb’’ added.
ACTUATOR n°3 - ‘‘button’’ added.
Workshop » sort TYPE -r NAME ID -r
Workshop » disp
WIRE n°1 - ‘‘type-c’’
WIRE n°2 - ‘‘usb’’
WIRE n°0 - ‘‘usb’’
ACTUATOR n°3 - ‘‘button’’
Workshop »
```

How to test quickly?

Our test server will not type the commands one by one, it will run tests using pre-filled prompt files like this one:

```
Terminal
B-CPE-110> cat -e example_test
add WIRE a, WIRE b, WIRE c, WIRE d, SENSOR e, SENSOR f, WIRE a$
disp$
sort TYPE -r$
disp$
exit$
```

To test, all you have to do is :

```
Terminal
B-CPE-110> cat example_test | ./organized
```

v1

* apprendre autrement