{EPITECH}
LEARN DIFFERENT*

BOOTSTRAP - ORGANIZED

ELEMENTARY PROGRAMMING IN C

* apprendre autrement

BOOTSTRAP - ORGANIZED

language: C
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).

Reminder

The Organized project involves developing the `add`, `del`, `sort` and `disp` functions of a given shell to store and sort hardware.

That's why this bootstrap covers a lot of different things:
- `void *` pointers
- linked lists manipulation
- sorting algorithms
- …

The boostrap will consist of:

- ✓ In the first part of this bootstrap, we're going to concentrate on `void *` pointer manipulation, so that you can manipulate data without necessarily giving it a type.
- ✓ Then we'll develop a **functions for manipulating linked lists**, which you can reuse on the Organized project and on subsequent ones.
- ✓ Finally, we'll develop and compare **2 different sorting algorithms** to see their strengths for large-scale use.

As a reminder, the final project should look like this:

```
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 »
```

Void * pointers

Step 1

Write a function that displays the value pointed to by the data pointer according to its type. The type is an enum specified in the `bootstrap.h` file.
4 types of variables can be passed:

- ✓ CHAR
- ✓ INTEGER
- ✓ STRING
- ✓ PLAYER (which is a structure specified in `bootstrap.h`.)

The function should be prototyped as follows:

```
int print_typed_value(void *data, type_t type);
```

Take a moment to understand what a `void` pointer is. It's an untyped pointer, which means it's just an address, no matter what it points to.

A possible `main` test function would be:

```
int main(void)
{
    int nb = 3;
    char c = 'k';
    char *str = strdup("Jonathan");
    player_t player = {strdup("Nau"), 98};

    print_typed_value(&nb, INTEGER);
    print_typed_value(&c, CHAR);
    print_typed_value(str, STRING);
    print_typed_value(&player, PLAYER);
    return 0;
}
```

The expected result should be:

```
Terminal
B-CPE-110> ./a.out | cat -e
3$
k$
Jonathan$
Nau: lvl.98$
```

Linked lists library

Step 1

Write a function called `push_to_list` that adds a new node to the chain list given in parameter, with a void pointer to the given data.

```
void push_to_list(linked_list_t **begin, void *data);
```

The functions you're going to develop here are practically those required for the project!

Step 2

Write a function called `display_list` that displays all nodes using a pointer to a function.

```
void display_list(linked_list_t *begin, void (*disp_fct)());
```

Step 3

Write a function named `delete_in_list` that removes all nodes containing data equal to the reference data.

The function should be prototyped as follows:

```
void delete_in_list(linked_list_t **begin, void const *ref, int (*cmp_fct)());
```

A possible `main` test function would be:

```
// Your function to compare:
int compare_function(void *data, void *ref);

// Your function to display:
void display_function(void *data);

int main(void)
{
    linked_list_t *list = NULL;
    int a = 3;
    int b = 4;

    push_to_list(&list, &a);
    push_to_list(&list, &a);
    push_to_list(&list, &b);
    delete_in_list(&list, &a, &compare_function);
    display_list(list, &display_function);
    return 0;
}
```

Sorting algorithms

Step 1

Now that you've mastered void pointers and chained lists, it's time to develop your first sorting algorithm! One of the simplest algorithms to implement, especially in C, is the bubble sort.

With just a few loops, you can sort your data as you like, but it's also the slowest sorting algorithm!

The function should be prototyped as follows:

```
void bubble_sort(int *array, int len);
```

You're sorting an integer array here, but it can be used with anything, comparing IDs, NAMEs…

To find out more...

Bubble sorting is really not the algorithm we're going to ask you to implement. It will have its successes, but on a small scale. Try to find out what's out there :)

I know a great site for comparing the best-known algorithms!

v1

* apprendre autrement