{EPITECH}
LEARN DIFFERENT*

BOOTSTRAP - SECURED

ELEMENTARY PROGRAMMING IN C

BOOTSTRAP - SECURED

language: C
Authorized functions: For this bootstrap, the only authorized functions are those of the standard `libc`.

- ✓ 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).

{EPITECH}

Reminder

The Secured project involves developing your own hash table and subsequent hash function. In this bootstrap, we're going to concentrate exclusively on the hash table part, the hash function being mainly personal research work.

> That's why this bootstrap covers a lot of different things:
> - pointers to functions
> - array structure
> - manipulation of linked lists
> - ...

Together, we'll try to develop a system that resembles a telephone directory! In a way, this echoes the hash table system:

- ✓ **Allocate**, create a phone book
- ✓ **Fill** it with contacts
- ✓ **Manipulating** your phone book with function pointers

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

```
B-CPE-110> ./a.out
[0]:
> 13116 - +33 6 31 45 61 23 71
[1]:
> 1769199557 - ./Documents/Tournament/Modules/Vision
[2]:
> 1952572858 - ./Trash/Hollidays_Pics/.secret_folder/kratos.ai
[3]:
```

{EPITECH}

Hash table

Step 1

Before you start, don't forget to download the `bootstrap.h` file appended to the bootstrap project. This file contains the `entry_t` structure we're going to manipulate:

```
B-CPE-110> cat bootstrap.h
#ifndef BOOTSTRAP_H
        #define BOOTSTRAP_H

typedef struct entry_s {
        char *name;
        char *phone_number;
} entry_t;

#endif /* BOOTSTRAP_H */
```

This `entry_t` structure will be used as an entry in the phone book, a name and a phone number! The first step will be to develop a function that will create our phone book, i.e., **an array of this structure** and write in the standard output that it has created your address book!

The function should be prototyped as follows:

```
entry_t **create_address_book(int len);

int main(void)
{
    int size = 5;
    entry_t **address_book = create_address_book(size);

    return 0;
}
```

```
B-CPE-110> ./a.out
address book created
```

It might be interesting to set each of the indexes in this table to `NULL`!

3

Step 2

To insert new contacts into this directory, we need a function that **creates a new entry**, a pointer to an `entry_t`, filled with the contact's information.

The function should be prototyped as follows:

```
entry_t *create_address(const char *name, const char *phone_number);
```

> Allocating our `entry_t **` first, then each of our `entry_t *` echoes what you've been doing up to now to allocate a `char **` in your `str_to_word_array` for example!

{EPITECH}

Step 3

Now that we have an address book and a function to generate new entries for us, let's create a function that will call `create_address` and **insert our directory entry** at the first available index. What happens if there's no more room? It would be better to write an error message about this in the standard output.

The function should be prototyped as follows:

```c
void add_address(entry_t **address_book, int len, const char *name, const char *
    phone_number);

int main(void)
{
    int size = 5;
    entry_t **address_book = create_address_book(size);

    add_address(address_book, size, "Kevin", "+33 6 45 12 87 11 02");
    add_address(address_book, size, "Jonathan", "+33 6 74 91 48 12 04");
    add_address(address_book, size, "Cyril", "+33 7 64 99 01 00 06");
    add_address(address_book, size, "Leo", "+33 7 65 19 98 71 08");
    add_address(address_book, size, "Gildas", "+33 6 45 67 89 90 10");
    add_address(address_book, size, "<3", "+33 6 07 08 09 00 12"); // There's no slot left
        for you </3
    return 0;
}
```

```
▽                                   Terminal                              –  +  X
B-CPE-110> ./a.out
address book created
new address added
new address added
new address added
new address added
new address added
add_address: not enough capacity to store new address
```

{EPITECH}

Step 4

Before we finish with this directory, I'd like you to develop a function that takes a **pointer to function** as parameter! The function pointer will be used on each entry.

It could be a function that simply displays each address or a function that would change the names of people with phone numbers starting with "+33 6"!
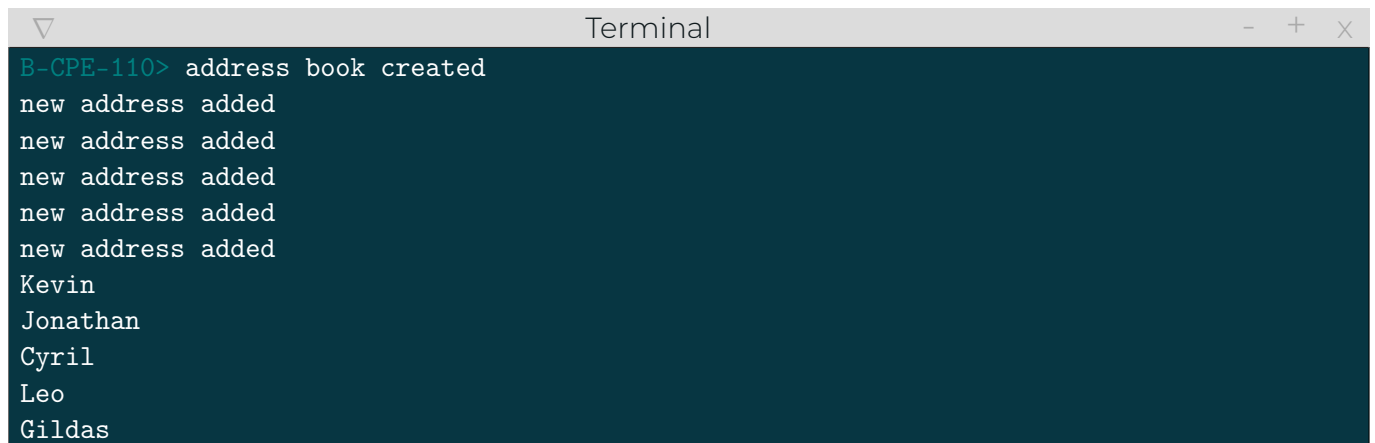
The function should be prototyped as follows:

```c
void execute_on_address_book(entry_t **address_book, int len, void (*execute)(entry_t *));

void dummy_function(entry_t *address)
{
    write(1, address->name, strlen(address->name));
    write(1, "\n", 1);
}

int main(void)
{
    int size = 5;
    entry_t **address_book = create_address_book(size);

    add_address(address_book, size, "Kevin", "+33 6 45 12 87 11 02");
    add_address(address_book, size, "Jonathan", "+33 6 74 91 48 12 04");
    add_address(address_book, size, "Cyril", "+33 7 64 99 01 00 06");
    add_address(address_book, size, "Leo", "+33 7 65 19 98 71 08");
    add_address(address_book, size, "Gildas", "+33 6 45 67 89 90 10");
    execute_on_address_book(address_book, size, &dummy_function);
    return 0;
}
```

```
                                    Terminal                              –  +  X
B-CPE-110> address book created
new address added
new address added
new address added
new address added
new address added
Kevin
Jonathan
Cyril
Leo
Gildas
```
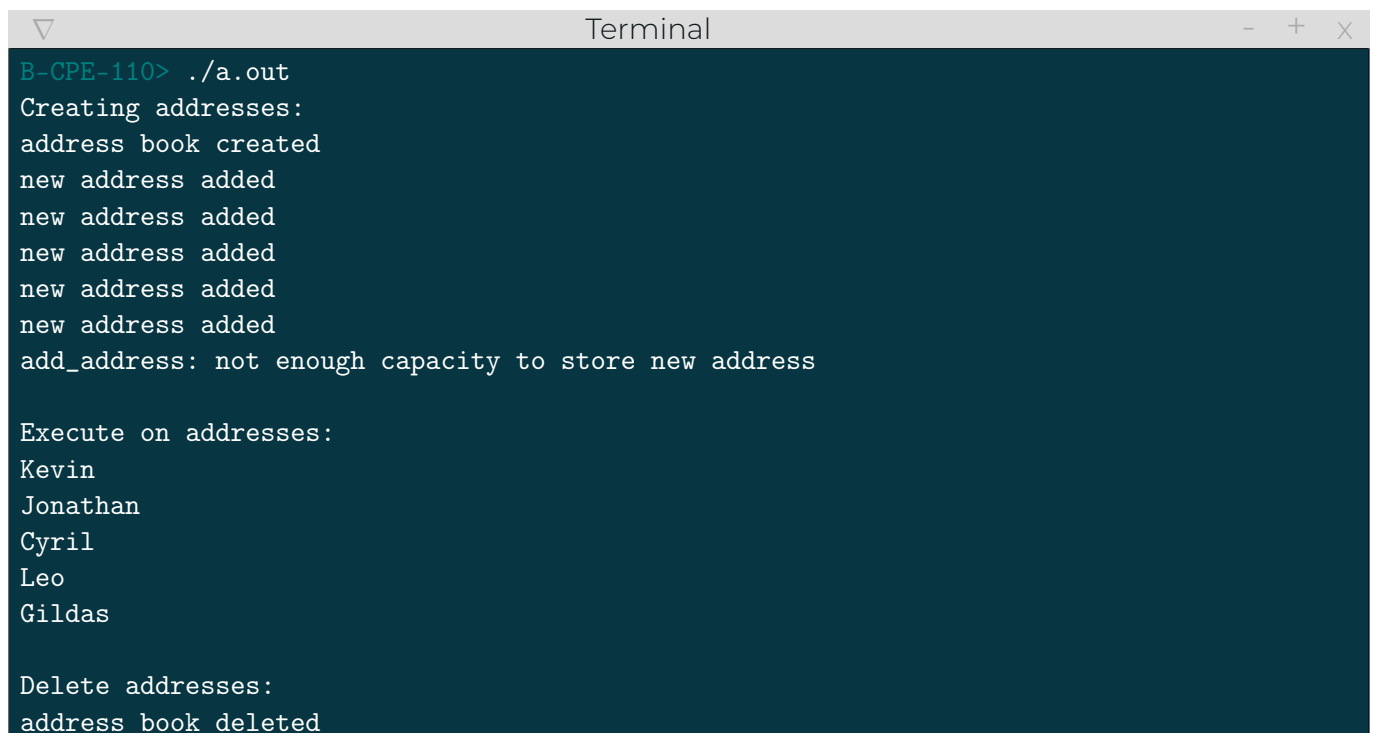
{EPITECH}

Step 5

Finally, I'm going to ask you to develop a function to delete your address book and free up the allocated memory!

```c
void delete_address_book(entry_t **address_book, int len);

int main(void)
{
    int size = 5;
    entry_t **address_book;

    write(STDOUT_FILENO, "Creating addresses:\n", strlen("Creating addresses:\n"));
    address_book = create_address_book(size);
    add_address(address_book, size, "Kevin", "+33 6 45 12 87 11 02");
    add_address(address_book, size, "Jonathan", "+33 6 74 91 48 12 04");
    add_address(address_book, size, "Cyril", "+33 7 64 99 01 00 06");
    add_address(address_book, size, "Leo", "+33 7 65 19 98 71 08");
    add_address(address_book, size, "Gildas", "+33 6 45 67 89 90 10");
    add_address(address_book, size, "<3", "+33 6 07 08 09 00 12"); // There's no slot left
        for you </3
    write(STDOUT_FILENO, "\nExecute on addresses:\n", strlen("\nExecute on addresses:\n"))
        ;
    execute_on_address_book(address_book, size, &dummy_function);
    write(STDOUT_FILENO, "\nDelete addresses:\n", strlen("\nDelete addresses:\n"));
    delete_address_book(address_book, size);
    return 0;
}
```

```
▽                                    Terminal                                  –  +  X
B-CPE-110> ./a.out
Creating addresses:
address book created
new address added
new address added
new address added
new address added
new address added
add_address: not enough capacity to store new address

Execute on addresses:
Kevin
Jonathan
Cyril
Leo
Gildas

Delete addresses:
address book deleted
```

{ EPITECH }

Linked lists

If you're feeling a little rusty on the notion since your last project, or if you haven't opted for linked lists on the `Organized` project, I can only advise you to get back on the subject of Bootstrap for the `Organized` project or day 11 of the C Pool.

But now ?

I hope the bootstrap has given you a few ideas and leads for the project! I'll leave you with three things to think about:
- It's not very practical to have to move the `len` variable around.
- In the subject, entries must be inserted in a specific index, not the first one that comes along.
- When an index is already in use, don't simply say that it's impossible to insert it, as in this bootstrap.

> Remember the tool shared at the last bootstrap to visualize sorting algorithms? It also has a hash table viewer ;)

{ EPITECH }

{EPITECH}
LEARN DIFFERENT*