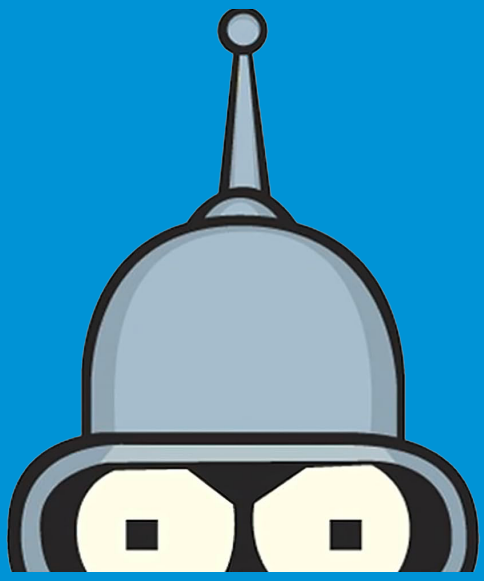
{EPITECH}

BOOTSTRAP - A-MAZE-D

ELEMENTARY PROGRAMMING IN C



BOOTSTRAP - A-MAZE-D

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).

Each exercise is to be submitted in a file called “name_of_the_function.c” at the root of the repository. Each file will be compiled separately with our main function. Also deliver a graph.h file to the root of the directory, which will contain the declaration of your structures.

If you haven't already done so, please read the A-Maze-d project description.
If you didn't come to the A-Maze-d kick-off, well, you should have!

A-Maze-d is a project that includes two new concepts: graphs and pathfinding.
In order to avoid overloading this Bootstrap, we're going to focus on the graph concept.

Step 1

Write a function that creates a new link_t element like the one below:

```
typedef struct link {
    int data;
    struct link *next;
} link_t;
```

This link should be correctly allocated and initialized.

The above example is generally used for linked lists with a next pointer which only allows a pointer toward one, and only one, other link.
As we're working with graphs, **you'll need to modify this structure**, and potentially create another one, so that each link can point to an indefinite number of other links.

```
link_t *create_link(int data);
```

Step 2

Write a function that displays the data value of the element passed as parameter, followed by a `\n`.

```
void print_link(link_t *link);
```

Make sure you created/initialized your link before displaying it!

Step 3

You just created an isolated link. It's through the connection to other links that a graph can be formed. Write a function that links one link to another. In other words, that connects them like we would connect two links in a linked list.
It must take a **link1** (like we generated during the previous step) and create a unidirectional (for now) connection with **link2** (also created with the previous step).

```
void connect_links(link_t *link1, link_t *link2);
```

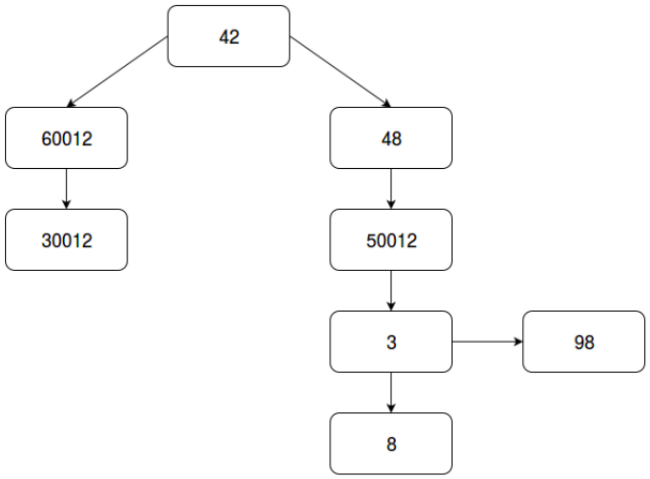
Step 4

Now we connected two links together, let's check the connections are correct. Write a function that takes a link and display the data (followed by a `\n`) of each of the links to which it is connected. The display order isn't important.

```
void print_data_of_connected_links(link_t *link);
```

Step 5

Use the functions that were created during the previous steps to build a graph that corresponds to the following figure:




```
link_t *build_my_graph(void)
```

Step 6

Write a function that takes a pointer on a `link_t` representing a graph (like the one we just created with the previous function) and that displays the values of all of the links on the graph (display order isn't important).

```
void print_my_graph_data(link_t *graph);
```

v1