Chapter III

Objectives

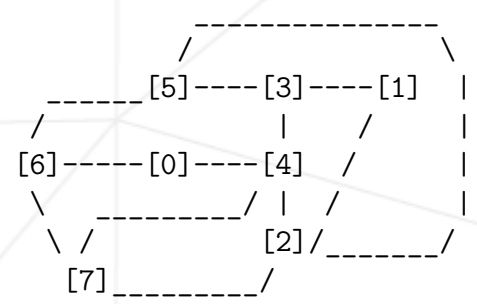
- Your program will receive the data describing the ant farm from the standard output in the following format:

```
number_of_ants
the_rooms
the_links
```

- The ant farm is defined by the following links:

```
##start
1 23 3
2 16 7
#comment
3 16 3
4 16 5
5 9 3
6 1 5
7 4 8
##end
0 9 5
0-4
0-6
1-3
4-3
5-2
3-5
#another comment
4-2
2-1
7-6
7-2
7-4
6-5
```

- Which corresponds to the following representation:



- There are two parts:
  - The rooms, which are defined by: `name coord_x coord_y`
  - The links, which are defined by: `name1-name2`
  - All of it is broken by comments, which start with `#`

The rooms' names won't necessarily be numbers, and they won't necessarily be in the right and continuous order (we will find rooms with names such as zdfg, qwerty, etc...) But most importantly, a room will never start with the character L nor the character #

The rooms' coordinates will always be integers.

- Lines that start with `##` are commands modifying the properties of the line that comes right after.
- For example, `##start` signals the ant farm's entrance and `##end` its exit.

Any unknown command will be ignored.

- Any non compliant or empty lines will automatically stop the ant farm's reading as well as the orderly processing of the acquired data.

- If there isn't enough data to process normally you must display `ERROR`

Chapter IV

General Instructions

- This project will only be corrected by actual human beings. You are therefore free to organize and name your files as you wish, although you need to respect some requirements listed below.
- The executable file must be named `lem-in`.
- You must submit a `Makefile`. That `Makefile` needs to compile the project and must contain the usual rules. It can only recompile the program if necessary.
- If you are clever, you will use your library for your `lem-in`. Also submit your folder `libft` including its own `Makefile` at the root of your repository. Your `Makefile` will have to compile the library, and then compile your project.
- Your project must be written in `C` in accordance with the Norm.
- You have to handle errors in a sensitive manner. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- Your program cannot have memory leaks.
- You'll have to submit at the root of your folder, a file called `author` containing your username followed by a '\n'

```
$>cat -e author
xlogin$
```

- Within your mandatory part you are allowed to use the following functions:
  - `malloc`
  - `free`
  - `read`
  - `write`
  - `strerror`
  - `perror`

- `exit`

- You are allowed to use other functions to carry out the bonus part as long as their use is justified during your defence.

- You can ask questions on the forum & Slack.

Chapter V

Mandatory part

- The goal of this project is to find the quickest way to get `n` ants across the farm.
- Obviously, there are some basic constraints. To be the first to arrive, ants will need to take the shortest path (and that isn't necessarily the simplest). They will also need to avoid traffic jams as well as walking all over their fellow ants.
- At the beginning of the game, all the ants are in the room `##start`. The goal is to bring them to the room `##end` with as few turns as possible. Each room can only contain one ant at a time. (except at `##start` and `##end` which can contain as many ants as necessary.)
- We consider that all the ants are in the room `##start` at the beginning of the game.
- At each turn you will only display the ants that moved.
- At each turn you can move each ant only once and through a tube (the room at the receiving end must be empty).
- You must to display your results on the standard output in the following format:

```
number_of_ants
the_rooms
the_links

Lx-y Lz-w Lr-o ...
```

`x, z, r` represents the ants' numbers (going from 1 to number_of_ants) and `y, w, o` represents the rooms' names.

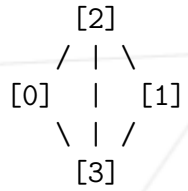
- Example 1 :

```
[0]-[2]-[3]-[1]
```

```
zaz@blackjack /tmp/lem-in $ ./lem-in < subjet1.map
3
##start
0 1 0
##end
1 5 0
2 9 0
3 13 0
0-2
2-3
3-1

L1-2
L1-3 L2-2
L1-1 L2-3 L3-2
L2-1 L3-3
L3-1
zaz@blackjack /tmp/lem-in $
```

- Example 2 :




```
zaz@blackjack /tmp/lem-in $ ./lem-in < subjet2.map
3
2 5 0
##start
0 1 2
##end
1 9 2
3 5 4
0-2
0-3
2-1
3-1
2-3

L1-3 L2-2
L1-1 L2-1 L3-3
L3-1
zaz@blackjack /tmp/lem-in $
```

This is not as simple as it seems

- Now, it would have been interesting to find out more about the type of operations those students of the School of Wizardry could conduct with such a computer. All that we know is that electricity is way more reliable today.

Chapter VI

Bonus part

We will look at your bonuses if and only if your mandatory part is EXCELLENT. This means that your must complete the mandatory part, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. If that's not the case, your bonuses will be totally IGNORED.

Find below a few ideas of interesting bonuses you could create. Some could even be useful. You can, of course, invent your own, which will then be evaluated by your correctors according to their own taste.

- As a bonus, why not code an ant farm vizualizer?
    - Either in 2 dimensions, seen from the "top". Or even better from the perspective of an ant in the corridors of the farm in 3D.
    - To use it, we could write: `./lem-in < ant_farm_map.txt | ./visu-hex`
    - Please note that because the commands and comments also appear on the standard output, it is possible to pass specific commands to the visualizer (such as various colors or levels)
    - You should have noticed that the room's coordinates will only be useful here.