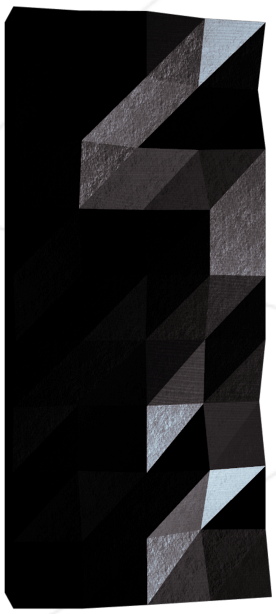

ft_ls

As simple as listing the files in a directory.

Summary: In short: This project will make you recode the command "ls".

Version: 3.1

Contents

Chapter I

Introduction

The `ls` command is one of the first commands you have learned to use with shell. It is also one you are using the most. Perhaps you have already asked yourself how is this function coded? Thanks to this project, you will soon find out.

To Recode `ls` and some of its options will allow you to find out how to interact with the file system using `C`. After all, you already know how to open, read, write and close a file. But, what about the directories? Special files? Rights, dates or sizes of the files?

And while I am on the topic, the quality of your `libft` will make the difference between a pleasant project experience and an abominable one. For example, if you add `ft_printf` to your `libft`, your life will be more enjoyable. It is possible to complete the `ft_ls` project without the `ft_printf` function. By the same token, you can easily eat a yogurt with your fingers. A spoon would still make your experience more pleasant

Chapter II

Objectives

The project `ft_ls` opens the path to the `Unix` branch of the sphere system. For the first time, you will have to face the one `libc` functions that will allow you to do other things than just read or write on a file descriptor (this is to simplify of course). You will discover a sub-system of functions of operating system's API, the associated data structures, as well as the management of memory allocation and the associated data.

`ft_ls` is also a great opportunity to think about the structure of your code before you even start writing your code. `ft_ls`' bad reputation is due to students discovering too late in the game that their (lack of) initial design is preventing them from finishing their project without refactorizing a great part of their code. I admit that it can be frustrating...

To conclude, `ft_ls` is another opportunity to add to your `libft` new practical functions. Browsing a file and identifying directories is quite common in programming. Remember that you you will have to do it on many future occasions. Improving your `libft` today will save you time tomorrow

Chapter III

General Instructions

- This project will only be evaluated 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.
- Your project must be written in **C**.
- The executable file must be named `ft_ls`.
- You must submit a Makefile. That Makefile will have 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 `ft_ls`, submit also 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.
- 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). If you are unsure, handle the errors like `ls`.
- Your program cannot have memory leaks.

- Within your mandatory part you are allowed to use the following functions:
    - `write`
    - `opendir`
    - `readdir`
    - `closedir`
    - `stat`
    - `lstat`
    - `getpwuid`
    - `getgrgid`
    - `listxattr`
    - `getxattr`
    - `time`
    - `ctime`
    - `readlink`
    - `malloc`
    - `free`
    - `perror`
    - `strerror`
    - `exit`
- You are allowed to use other functions to carry out the bonus part as long as their use is justified during your defence. For example, to use `tcgetattr` is justified in certain case, to use `printf` because you are lazy isn't. Be smart!

Chapter IV

Mandatory part

- You must recode the system's command `ls`.
- Its behavior must be identical to the original `ls` command with the following variations:
  - Amongst the numerous options available, we are asking you to create the following: `-l`, `-R`, `-a`, `-r` and `-t`.
  - **We strongly recommend that you account for the implications of the option `-R` from the very beginning of your code...**
  - You do not have to deal with the multiple column format for the exit when the option `-l` isn't in the arguments.
  - You are not required to deal with `ACL` and extended attributes.
  - The overall display, depending on each option, must stay as identical as possible to the system command. We will be cordial when grading either the padding or the pagination, but no information can be missing.

```
man ls
```

```
For your evaluation:  no particular management of the locals is
required i.e LC_ALL=C.
```

Chapter V

Bonus part

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 evaluators according to their own taste.

- Management of ACL and extended attributes.
- Management of the columns without the option `-l`. (man 4 tty)
- Management of options `-u`, `-f`, `-g`, `-d`, ...
- Management of views in colors (Similar to option `-G`)
- Optimization of your code (What is the response time of your ls on a BIG ls -lR for example?)

```
The bonus part will only be assessed if the mandatory part is
PERFECT. Perfect means the mandatory part has been integrally done
and works without malfunctioning.  If you have not passed ALL the
mandatory requirements, your bonus part will not be evaluated at all.
```

Chapter VI

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.