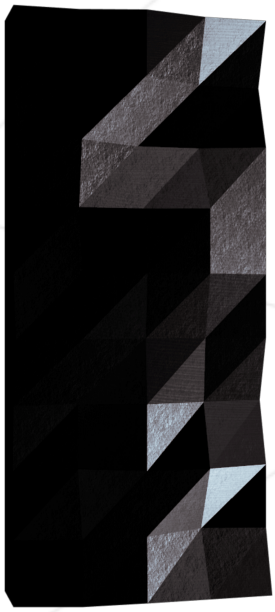

ft_mini_ls

As simple as listing the files in a directory.

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

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.

Chapter II

Common Instructions

- Your project must be written in accordance with the Norm. If you have bonus files/functions, they are included in the norm check and you will receive a `0` if there is a norm error inside.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.
- If the subject requires it, you must submit a `Makefile` which will compile your source files to the required output with the flags `-Wall`, `-Wextra` and `-Werror`, and your Makefile must not relink.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To turn in bonuses to your project, you must include a rule `bonus` to your Makefile, which will add all the various headers, librairies or functions that are forbidden on the main part of the project. Bonuses must be in a different file `_bonus.{c/h}`. Mandatory and bonus part evaluation is done separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` in a `libft` folder with its associated Makefile. Your project's `Makefile` must compile the library by using its `Makefile`, then compile the project.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter III

Mandatory part

Program name	`ft_mini_ls`
Turn in files	`*.c, */*.c, *.h, */*.h, Makefile`
Makefile	`all, clean, fclean, re`
External functs.	`malloc, free, write, opendir, readdir, closedir, stat, lstat, perrror, strerror`
Libft authorized	`yes`
Description	`Write a program similar to ls but with less features`

- Your program should output the same result as this command `ls -1tr`.
- Your program should not handle commandline arguments. If the program is executed with commandline arguments, print a message to stderr. The message should be informative.

```
$> ./ft_mini_ls | cat -e
oldest_file$
not_that_old_file$
most_recent_file$
ft_mini_ls$
$> ls -1tr | cat -e
oldest_file$
not_that_old_file$
most_recent_file$
ft_mini_ls$
```

Chapter IV

Bonus

- Colorize the output (-G).
- Your program can handle commandline arguments.
- Implement additional sorting features. (-S, -u, -U, etc...)
- Add Recursive option (-R).
- Other new features which are useful.
- You are authorized to use command line arguments to add these features.