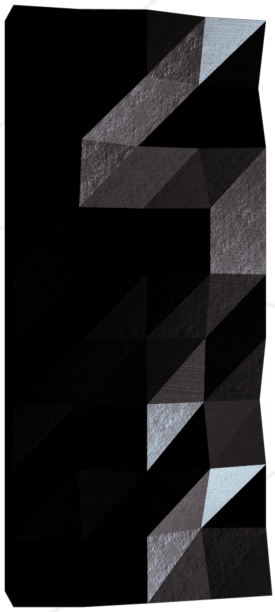

Libft

Your very first own library

Summary:
This project is about coding a `C` *library.*
It will contain a lot of general purpose functions your programs will rely upon.

Version: 15

Contents

Chapter I

Introduction

`C` programming can be very tedious when one doesn't have access to the highly useful standard functions. This project is about understanding the way these functions work, implementing and learning to use them. Your will create your own library. It will be helpful since you will use it in your next `C` school assignments.

Take the time to expand your `libft` throughout the year. However, when working on a new project, don't forget to ensure the functions used in your library are allowed in the project guidelines.

Chapter II

Common Instructions

- Your project must be written in C.
- 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`, use cc, 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}` if the subject does not specify anything else. 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	`libft.a`
Turn in files	`Makefile, libft.h, ft_*.c`
Makefile	`NAME, all, clean, fclean, re`
External functs.	`Detailed below`
Libft authorized	`n/a`
Description	`Write your own library: a collection of functions that will be a useful tool for your cursus.`

III.1 Technical considerations

- Declaring global variables is forbidden.
- If you need helper functions to split a more complex function, define them as `static` functions. This way, their scope will be limited to the appropriate file.
- Place all your files at the root of your repository.
- Turning in unused files is forbidden. (Files containing functions from previous excercises are allowed)
- Every `.c` files must compile with the flags `-Wall -Wextra -Werror`.
- You must use the command `ar` to create your library. Using the `libtool` command is forbidden.
- Your `libft.a` has to be created at the root of your repository.

III.2 Additional functions

In this part, you must develop a set of functions that are either not in the `libc`, or that are part of it but in a different form.

Function name	`ft_strmapi`
Prototype	`char *ft_strmapi(char const *s, char (*f)(unsigned int, char));`
Turn in files	-
Parameters	`s:` The string on which to iterate. `f:` The function to apply to each character.
Return value	The string created from the successive applications of 'f'. Returns NULL if the allocation fails.
External functs.	`malloc`
Description	Applies the function 'f' to each character of the string 's', and passing its index as first argument to create a new string (with malloc(3)) resulting from successive applications of 'f'.

Function name	`ft_striteri`
Prototype	`void ft_striteri(char *s, void (*f)(unsigned int, char*));`
Turn in files	-
Parameters	`s:` The string on which to iterate. `f:` The function to apply to each character.
Return value	None
External functs.	None
Description	Applies the function 'f' on each character of the string passed as argument, passing its index as first argument. Each character is passed by address to 'f' to be modified if necessary.

Function name	`ft_putchar_fd`
Prototype	`void ft_putchar_fd(char c, int fd);`
Turn in files	-
Parameters	`c:` The character to output. `fd:` The file descriptor on which to write.
Return value	None
External functs.	`write`
Description	Outputs the character 'c' to the given file descriptor.

Function name	ft_putstr_fd
Prototype	void ft_putstr_fd(char *s, int fd);
Turn in files	-
Parameters	s: The string to output. fd: The file descriptor on which to write.
Return value	None
External functs.	write
Description	Outputs the string 's' to the given file descriptor.

Function name	ft_putendl_fd
Prototype	void ft_putendl_fd(char *s, int fd);
Turn in files	-
Parameters	s: The string to output. fd: The file descriptor on which to write.
Return value	None
External functs.	write
Description	Outputs the string 's' to the given file descriptor followed by a newline.

Function name	ft_putnbr_fd
Prototype	void ft_putnbr_fd(int n, int fd);
Turn in files	-
Parameters	n: The integer to output. fd: The file descriptor on which to write.
Return value	None
External functs.	write
Description	Outputs the integer 'n' to the given file descriptor.

III.3 Makefile

In this part, you must write a makefile that compiles all of the functions you've written so far from `Libft-00` to `Libft-03` into a binary called `libft.a`

Your turn-in repository should contain all of the functions from `Libft-00`, `Libft-01`, `Libft-02`, and `Libft-03`.

Make sure your makefile follows the guidelines stated in the common instructions.

Chapter IV

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 files to ensure they are correct.

Place all your files at the root of your repository.