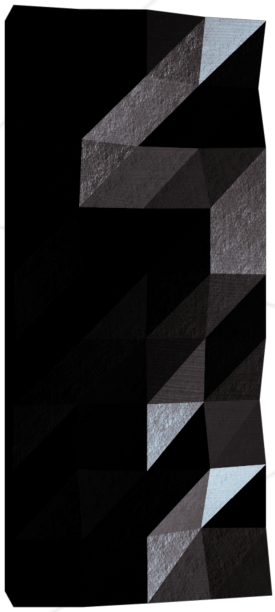

Libft-02

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.
- 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

Turn in files	`libft.h, ft_*.c`
External functs.	`Detailed below`
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.
- Every `.c` files must compile with the flags `-Wall -Wextra -Werror`.

III.2 Libc functions

To begin, you must redo a set of functions from the `libc`. Your functions will have the same prototypes and implement the same behaviors as the originals. They must comply with the way they are defined in their `man`. The only difference will be their names. They will begin with the '`ft_`' prefix. For instance, `strlen` becomes `ft_strlen`.

Some of the functions' prototypes you have to redo use the 'restrict' qualifier. This keyword is part of the c99 standard. It is therefore forbidden to include it in your own prototypes and to compile your code with the -std=c99 flag.

You must write your own function implementing the following original ones. In order to implement the two following functions, you will use `malloc()`:

- `calloc`
- `strdup`

III.3 Additional functions

In this second 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_substr
Prototype	char *ft_substr(char const *s, unsigned int start, size_t len);
Turn in files	-
Parameters	s: The string from which to create the substring. start: The start index of the substring in the string 's'. len: The maximum length of the substring.
Return value	The substring. NULL if the allocation fails.
External functs.	malloc
Description	Allocates (with malloc(3)) and returns a substring from the string 's'. The substring begins at index 'start' and is of maximum size 'len'.

Function name	ft_strjoin
Prototype	char *ft_strjoin(char const *s1, char const *s2);
Turn in files	-
Parameters	s1: The prefix string. s2: The suffix string.
Return value	The new string. NULL if the allocation fails.
External functs.	malloc
Description	Allocates (with malloc(3)) and returns a new string, which is the result of the concatenation of 's1' and 's2'.

Function name	ft_strtrim
Prototype	char *ft_strtrim(char const *s1, char const *set);
Turn in files	-
Parameters	s1: The string to be trimmed. set: The reference set of characters to trim.
Return value	The trimmed string. NULL if the allocation fails.
External functs.	malloc
Description	Allocates (with malloc(3)) and returns a copy of 's1' with the characters specified in 'set' removed from the beginning and the end of the string.

Function name	`ft_split`
Prototype	`char **ft_split(char const *s, char c);`
Turn in files	-
Parameters	`s: The string to be split.` `c: The delimiter character.`
Return value	`The array of new strings resulting from the split.` `NULL if the allocation fails.`
External functs.	`malloc, free`
Description	`Allocates (with malloc(3)) and returns an array of strings obtained by splitting 's' using the character 'c' as a delimiter. The array must end with a NULL pointer.`

Function name	`ft_itoa`
Prototype	`char *ft_itoa(int n);`
Turn in files	-
Parameters	`n: the integer to convert.`
Return value	`The string representing the integer.` `NULL if the allocation fails.`
External functs.	`malloc`
Description	`Allocates (with malloc(3)) and returns a string representing the integer received as an argument. Negative numbers must be handled.`

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.