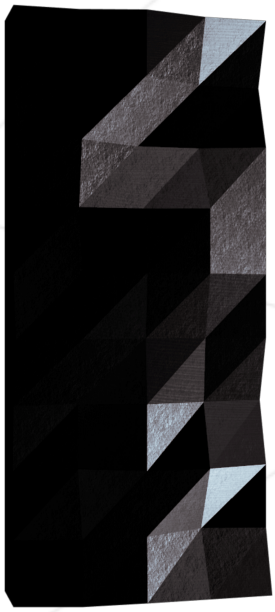

Libft

Your very first own library

Summary: The aim of this project is to code a `C` library regrouping usual functions that you'll be allowed to use in all your other projects.

Contents

Chapter I

Introduction

`C` programming can be very tedious when one doesn't have access to those highly useful standard functions. This project makes you to take the time to re-write those functions, understand them, and learn to use them. This library will help you for all your future `C` projects.

Through this project, we also give you the opportunity to expand the list of functions with your own. Take the time to expand your `libft` throughout the year.

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	`libft.a`
Turn in files	`Makefile, libft.h, and all the .c files required.`
Makefile	`Yes`
External functs.	`Detailed below`
Libft authorized	`Non-applicable`
Description	`Write your own library, containing an extract of important functions for your cursus.`

III.1 Technical considerations

- It is forbidden to use global variables.
- If you need sub-functions to write a complex function, you must define these sub-functions as `static` as stipulated in the Norm.

III.2 Part 1 - Libc functions

In this first part, you must re-code a set of the `libc` functions, as defined in their `man`. Your functions will need to present the same prototype and behaviors as the originals. Your functions' names must be prefixed by "`ft_`". For instance `strlen` becomes `ft_strlen`.

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

You must re-code the following functions. These function do not need any external functions:

- `memset`
- `bzero`
- `memcpy`
- `memccpy`
- `memmove`
- `memchr`
- `memcmp`
- `strlen`
- `strlcpy`
- `strlcat`
- `strchr`
- `strrchr`
- `strnstr`
- `strncmp`
- `atoi`
- `isalpha`
- `isdigit`
- `isalnum`
- `isascii`
- `isprint`
- `toupper`
- `tolower`

You must also re-code the following functions, using the function "`malloc`":

- `calloc`
- `strdup`

III.3 Part 2 - Additional functions

In this second part, you must code a set of functions that are either not included in the `libc`, or included in a different form. Some of these functions can be useful to write Part 1's functions.

Function name	`ft_substr`
Prototype	`char *ft_substr(char const *s, unsigned int start, size_t len);`
Turn in files	`ft_substr.c`

Parameters	`#1. The string from which create the substring.`
	`#2. The start index of the substring in the string.`
	`#3. 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 given in argument.` `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	`ft_strjoin.c`
Parameters	`#1. The prefix string.` `#2. 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, 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	`ft_strtrim.c`
Parameters	`#1. The string to be trimmed.` `#2. 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 the string given as argument without the characters specified in the set argument at the beginning and the end of the string.`

Function name	`ft_split`
Prototype	`char **ft_split(char const *s, char c);`
Turn in files	`ft_split.c`
Parameters	#1. The string to be split. #2. The delimiter character.
Return value	The array of new strings result of 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 be ended by a NULL pointer.

Function name	`ft_itoa`
Prototype	`char *ft_itoa(int n);`
Turn in files	`ft_itoa.c`
Parameters	#1. 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.

Function name	`ft_strmapi`
Prototype	`char *ft_strmapi(char const *s, char (*f)(unsigned int, char));`
Turn in files	`ft_strmapi.c`
Parameters	#1. The string on which to iterate #2. 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 passed as argument to create a new string (with malloc(3)) resulting from successive applications of f.

Function name	`ft_putchar_fd`
Prototype	`void ft_putchar_fd(char c, int fd);`
Turn in files	`ft_putchar_fd.c`
Parameters	`#1. The character to output` `#2. The file descriptor on which to write.`
Return value	`None`
External functs.	`write`
Description	`Outputs the character c to given file descriptor.`

Function name	`ft_putstr_fd`
Prototype	`void ft_putstr_fd(char *s, int fd);`
Turn in files	`ft_putstr_fd.c`
Parameters	`#1. The string to output` `#2. The file descriptor on which to write.`
Return value	`None`
External functs.	`write`
Description	`Outputs the string c to given file descriptor.`

Function name	`ft_putendl_fd`
Prototype	`void ft_putendl_fd(char *s, int fd);`
Turn in files	`ft_putendl_fd.c`
Parameters	`#1. The string to output` `#2. The file descriptor on which to write.`
Return value	`None`
External functs.	`write`
Description	`Outputs the string c to given file descriptor,` `followed by a newline.`

Function name	`ft_putnbr_fd`
Prototype	`void ft_putnbr_fd(int n, int fd);`
Turn in files	`ft_putnbr_fd.c`
Parameters	`#1. The integer to output` `#2. The file descriptor on which to write.`
Return value	`None`
External functs.	`write`
Description	`Outputs the integer n to given file descriptor,` `followed by a newline.`

Chapter IV

Bonus part

If you successfully completed the mandatory part, you'll enjoy taking it further. You can see this last section as Bonus Points.

Having functions to manipulate memory and strings is very useful, but you'll soon discover that having functions to manipulate lists is even more useful.

You'll use the following structure to represent the elements of your list. This structure must be added to your `libft.h` file.

```
typedef struct      s_list
{
    void            *content;
    struct s_list   *next;
}                   t_list;
```

Here is a description of the fields of the `t_list` struct:

- `content` : The data contained in the element. The `void *` allows to store any kind of data.
- `next` : The next element's address or `NULL` if it's the last element.

The following functions will allow you to easily use your lists.

Function name	`ft_lstnew`
Prototype	`t_list *ft_lstnew(void *content);`
Turn in files	`ft_lstnew_bonus.c`
Parameters	`#1. The content to create the new element with.`
Return value	`The new element`
External functs.	`malloc`
Description	`Allocates (with malloc(3)) and returns a new element. The variable content is initialized with the value of the parameter content. The variable next is initialized to NULL.`

Function name	`ft_lstadd_front`
Prototype	`void ft_lstadd_front(t_list **alst, t_list *new);`
Turn in files	`ft_lstadd_front_bonus.c`
Parameters	`#1. The address of a pointer to the first link of a list.` `#2. The address of a pointer to the element to add to the list.`
Return value	`None`
External functs.	`None`
Description	`Adds the element new at the beginning of the list`

Function name	`ft_lstsize`
Prototype	`int ft_lstsize(t_list *lst);`
Turn in files	`ft_lstsize_bonus.c`
Parameters	`#1. The beginning of the list.`
Return value	`Length of the list.`
External functs.	`None`
Description	`Counts the number of elements int a list.`

Function name	`ft_lstlast`
Prototype	`t_list *ft_lstlast(t_list *lst);`
Turn in files	`ft_lstlast_bonus.c`
Parameters	`#1. The beginning of the list.`
Return value	`Last element of the list`
External functs.	`None`
Description	`Returns the last element of the list.`

Function name	ft_lstadd_back
Prototype	void ft_lstadd_back(t_list **alst, t_list *new);
Turn in files	ft_lstadd_back_bonus.c
Parameters	#1. The address of a pointer to the first link of a list. #2. The address of a pointer to the element to add to the list.
Return value	None
External functs.	None
Description	Adds the element new at the end of the list.

Function name	ft_lstdelone
Prototype	void ft_lstdelone(t_list *lst, void (*del)(void *));
Turn in files	ft_lstdelone_bonus.c
Parameters	#1. The address of the pointer to an element #2. The address of the function to delete the content.
Return value	None
External functs.	free
Description	Takes as a parameter a element's pointer address and frees the memory of the element's content using the function del given as a parameter, then frees the element's memory using free(3). The memory of next must not be freed under any circumstance.

Function name	ft_lstclear
Prototype	void ft_lstclear(t_list **lst, void (*del)(void *));
Turn in files	ft_lstclear_bonus.c
Parameters	#1. The adress of a pointer to a element. #2. The adress of the function used to delete the content of l'élément.
Return value	None
External functs.	free
Description	Deletes and frees the given element and every successor of that element, using the function del and free(3) Finally, the pointer to the list must be set to NULL.

Function name	`ft_lstiter`
Prototype	`void ft_lstiter(t_list *lst, void (*f)(void *));`
Turn in files	`ft_lstiter_bonus.c`
Parameters	`#1. The adress of a pointer to a element.` `#2. The adress of the function to iterate on the list.`
Return value	`None`
External functs.	`None`
Description	`Iterates the list lst and applies the function f to each element.`

Function name	`ft_lstmap`
Prototype	`t_list *ft_lstmap(t_list *lst, t_list *(*f)(void *));`
Turn in files	`ft_lstmap_bonus.c`
Parameters	`#1. The adress of a pointer to a element.` `#2. The adress of the function to iterate on the list.`
Return value	`The new list. NULL if the allocation fails.`
External functs.	`malloc`
Description	`Iterates the list lst and applies the function f to each element. Creates a new list resulting of the successive applications of the function f.`

You are free to add any function to your libft as you see fit.