



## printf

Because putnbr and putstr aren't enough

*Summary: This project is pretty straight forward. You will recode printf. Hopefully you will be able to reuse it in future project without the fear of being flagged as a cheater.*

*You will mainly learn how to use variadic arguments.*

# **Contents**

I	Introduction	2
II	Common Instructions	3
III	Mandatory part	4
IV	Bonus part	5

# Chapter I

## Introduction

The versatility of the `printf` function in C represents a great exercise in programming for us. This project is of moderate difficulty. It will enable you to discover variadic functions in C.

The key to a successful `ft_printf` is a well-structured and good extensible code.

# 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, libraries 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

<b>Program name</b>	libftprintf.a
<b>Turn in files</b>	*.c, /**.c, *.h, /**.h, Makefile
<b>Makefile</b>	all, clean, fclean, re, bonus
<b>External functs.</b>	malloc, free, write, va_start, va_arg, va_copy, va_end
<b>Libft authorized</b>	yes
<b>Description</b>	Write a library that contains ft_printf, a function that will mimic the real printf

- The prototype of ft\_printf should be `int ft_printf(const char *, ...);`
- You have to recode the libc's printf function
- It must not do the buffer management like the real printf
- It will manage the following conversions: `cspdiuxX%`
- It will manage any combination of the following flags: '`-0.*`' and minimum field width with all conversions
- It will be compared with the real printf
- You must use the command `ar` to create your library, using the command `libtool` is forbidden.



`man 3 printf / man 3 stdarg`

# Chapter IV

## Bonus part

- If the Mandatory part is not perfect don't even think about bonuses
- You don't need to do all the bonuses
- Manage one or more of the following conversions: `nfg`
- Manage one or more of the following flags: `l ll h hh`
- Manage all the following flags: `'# +'` (yes, one of them is a space)