



## ft\_printf

Because ft\_putnbr() and ft\_putstr() aren't enough

*Summary: The goal of this project is quite straightforward: you will reimplement printf(). This will primarily teach you how to handle a variable number of arguments. How cool is that? Actually, it's pretty cool! :)*

*Version: 10.2*

# Contents

<b>I</b>	<b>Introduction</b>	<b>2</b>
<b>II</b>	<b>Common Instructions</b>	<b>3</b>
<b>III</b>	<b>Mandatory part</b>	<b>5</b>
<b>IV</b>	<b>Bonus part</b>	<b>7</b>
<b>V</b>	<b>Submission and peer-evaluation</b>	<b>8</b>

# Chapter I

## Introduction

You will explore one of the most popular and versatile functions in C: `printf()`. This exercise provides an excellent opportunity to improve your programming skills. It is considered moderately difficult.

You will discover **variadic functions** in C.

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



Once you have successfully completed this assignment, you will be allowed to add your `ft_printf()` to your `libft`, making it available for use in your school C projects.

# 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.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc.) except for undefined behavior. If this occurs, your project will be considered non-functional and will receive a 0 during the evaluation.
- All heap-allocated memory must be properly freed when necessary. Memory leaks will not be tolerated.
- If the subject requires it, you must submit a **Makefile** that compiles your source files to the required output with the flags **-Wall**, **-Wextra**, and **-Werror**, using **cc**. Additionally, your **Makefile** must not perform unnecessary relinking.
- Your **Makefile** must at contain at least the rules **\$(NAME)**, **all**, **clean**, **fclean** and **re**.
- To submit bonuses for your project, you must include a **bonus** rule in your **Makefile**, which will add all the various headers, libraries, or functions that are not allowed in the main part of the project. Bonuses must be placed in **\_bonus.{c/h}** files, unless the subject specifies otherwise. The evaluation of mandatory and bonus parts is conducted separately.
- If your project allows you to use your **libft**, you must copy its sources and its associated **Makefile** into a **libft** folder. 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 **does not need to be submitted and will not be graded**. It will give you an opportunity to easily test your work and your peers' work. You will find these 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 the assigned Git repository. Only the work in the Git repository will be graded. If Deepthought is assigned to grade your work, it will occur

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>	Makefile, *.h, */*.h, *.c, */*.c
<b>Makefile</b>	NAME, all, clean, fclean, re
<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 original printf()

You have to recode the `printf()` function from `libc`.

The prototype of `ft_printf()` is:

```
int    ft_printf(const char *, ...);
```

Here are the requirements:

- Do not implement the original `printf()`'s buffer management.
- Your function has to handle the following conversions: `cspdiuxX%`
- Your implementation will be evaluated against the behavior of the original `printf()`.
- You must use the command `ar` to create your library.  
The use of the `libtool` command is strictly forbidden.
- `libftprintf.a` must be created at the root of your repository.

You have to implement the following conversions:

- %c Prints a single character.
- %s Prints a string (as defined by the common C convention).
- %p The void \* pointer argument has to be printed in hexadecimal format.
- %d Prints a decimal (base 10) number.
- %i Prints an integer in base 10.
- %u Prints an unsigned decimal (base 10) number.
- %x Prints a number in hexadecimal (base 16) lowercase format.
- %X Prints a number in hexadecimal (base 16) uppercase format.
- %% Prints a percent sign.

# Chapter IV

## Bonus part

You don't have to do all the bonuses.

Bonus list:

- Manage any combination of the following flags: `'-0.'` and the field minimum width under all conversions.
- Manage all the following flags: `'# +'` (Yes, one of them is a space)



If you plan to complete the bonus part, consider the implementation of your additional features from the beginning. This will help you avoid the pitfalls of a naive approach.



The bonus part will only be assessed if the mandatory part is PERFECT. To be considered perfect, the mandatory part must be fully implemented and function correctly without any errors. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.



## Submission and peer-evaluation

Once you have completed this assignment, you will be allowed to add your `ft_printf()` to your `libft`, enabling its use in your school C projects.

