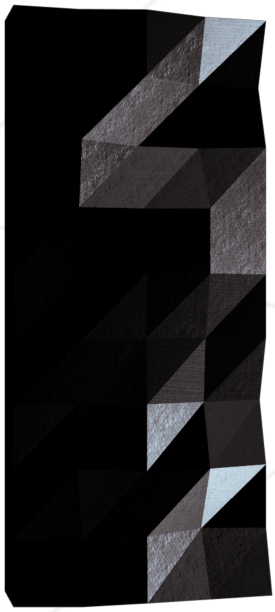

ft_printf

Because ft_putnbr() and ft_putstr() aren't enough

Summary:
The goal of this project is pretty straightforward. You will recode printf().
You will mainly learn about using a variable number of arguments. How cool is that??
It is actually pretty cool :)

Version: 10

Contents

Chapter I

Introduction

You will discover a popular and versatile `C` function: `printf()`. This exercise is a great opportunity to improve your programming skills. It is of moderate difficulty.

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

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

`Once this assignment passed, you will be allowed to add your ft_printf() to your libft so you can use it 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 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	`libftprintf.a`
Turn in files	`Makefile, *.h, */*.h, *.c, */*.c`
Makefile	`NAME, all, clean, fclean, re`
External functs.	`malloc, free, write,` `va_start, va_arg, va_copy, va_end`
Libft authorized	`Yes`
Description	`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:

- Don't implement the buffer management of the original `printf()`.
- Your function has to handle the following conversions: `cspdiuxX%`
- Your function will be compared against the original `printf()`.
- You must use the command `ar` to create your library.
  Using the `libtool` command is forbidden.
- Your `libftprintf.a` has to 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, think about the implementation of your extra features from the start. This way, you will avoid the pitfalls of a naive approach.

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter V

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.

Once this assignment passed, you will be allowed to add your `ft_printf()` to your `libft` so you can use it in your school `C` projects.

```
++++++++++[>+>+++>+++++++>++++++++++<<<<-]>>>.>---.++++++++++++.++.+++
+++.--.<<++.>>------.------------.++++++++++++.<<.>>++++++.------------
.-------.  ++++++++++++++++++.<<.>>----------------.+++++.+++++++++.---
----------.--.+ ++++++++++++++++.--------.++++++++++++.<<.>>----------
-------------.+++.+++ ++++.---.----.++++++++++++++++.---------------
--.-.<<.>>+++++.+++++.<<.>-------...
```