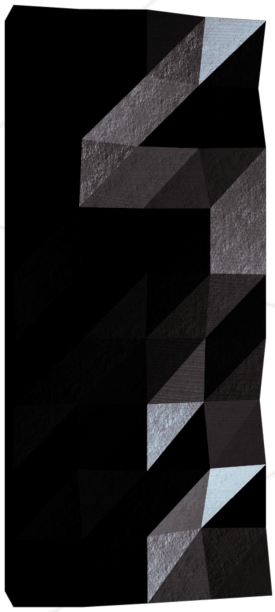

printf

Porque putnbr() y putstr() no son suficientes

Resumen:
El objetivo de este proyecto es bastante sencillo. Deberás reprogramar printf(). Aprenderás, principalmente, a utilizar un número variable de argumentos.

Versión: 10

Índice general

Capítulo I

Introducción

Vas a descubrir una función de `C` muy famosa y versátil: `printf()`. Este ejercicio es una grán oportunidad para mejorar tus habilidades de programación. Es un ejercicio de dificultad moderada

También descubrirás las **funciones variádicas** en `C`.

La clave para superar `ft_printf` es tener un código bien estructurado y extensible.

`Una vez que superes este ejercicio, podrás incluir ft_printf() en tu libft, por lo que podrás utilizarla en futuros proyectos en C.`

Capítulo II

Instrucciones generales

- Tu proyecto deberá estar escrito en C.
- Tu proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas están incluidas en la verificación de la Norma y tendrás un `0` si hay algún error de norma en cualquiera de ellos.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) ni tener comportamientos indefinidos. Si esto pasa tu proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria asignada en el heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el enunciado lo requiere, deberás entregar un `Makefile` que compilará tus archivos fuente al output requerido con las flags `-Wall`, `-Werror` y `-Wextra` y por supuesto tu `Makefile` no debe hacer relink.
- Tu `Makefile` debe contener al menos las normas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los bonus de tu proyecto deberás incluir una regla `bonus` en tu `Makefile`, en la que añadirás todos los headers, librerías o funciones que estén prohibidas en la parte principal del proyecto. Los bonus deben estar en archivos distintos `_bonus.{c/h}`. La parte obligatoria y los bonus se evalúan por separado.
- Si tu proyecto permite el uso de la `libft`, deberás copiar su fuente y sus `Makefile` asociados en un directorio `libft` con su correspondiente `Makefile`. El `Makefile` de tu proyecto debe compilar primero la librería utilizando su `Makefile`, y después compilar el proyecto.
- Te recomendamos crear programas de prueba para tu proyecto, aunque este trabajo **no será entregado ni evaluado**. Te dará la oportunidad de verificar que tu programa funciona correctamente durante tu evaluación y la de otros compañeros. Y sí, tienes permitido utilizar estas pruebas durante tu evaluación o la de otros compañeros.
- Entrega tu trabajo en tu repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus com-

pañeros. Si se encuentra un error durante la evaluación de Deepthought, esta habrá terminado.

Capítulo III

Parte obligatoria

Nombre de programa	`libftprintf.a`
Archivos a entregar	`Makefile, *.h, */*.h, *.c, */*.c`
Makefile	`NAME, all, clean, fclean, re`
Funciones autorizadas	`malloc, free, write,` `va_start, va_arg, va_copy, va_end`
Se permite usar libft	`Yes`
Descripción	`Escribe una librería que contenga la función` `ft_printf(), que imite el printf() original`

Debes reprogramar la función `printf()` de la `libc`.

El prototipo de `ft_printf()` es:

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

Aquí tienes los requisitos:

- No implementes la gestión del buffer del `printf()` original.
- Deberás implementar las siguientes conversiones: `cspdiuxX%`
- Tu función se comparará con el `printf()` original.
- Tienes que usar el comando `ar` para crear tu librería. El uso de `libtool` command is forbidden.
- Tu archivo `libftprintf.a` deberá ser creado en la raiz de tu repositorio.

Tienes que implementar las siguientes conversiones:

- `%c` Imprime un solo carácter.
- `%s` Imprime una string (como se define por defecto en `C`).
- `%p` El puntero `void *` dado como argumento se imprime en formato hexadecimal.
- `%d` Imprime un número decimal (base 10).
- `%i` Imprime un entero en base 10.
- `%u` Imprime un número decimal (base 10) sin signo.
- `%x` Imprime un número hexadecimal (base 16) en minúsculas.
- `%X` Imprime un número hexadecimal (base 16) en mayúsculas.
- `%%` para imprimir el símbolo del porcentaje.

Capítulo IV

Parte bonus

No es necesario hacer todos los bonus.

Lista de bonus:

- Gestiona cualquier combinación de los siguientes flags: '`-0.`' y el ancho mínimo (field minimum width) bajo todas las conversiones posibles.
- Gestiona todos los siguientes flags: '`# +`' (sí, uno de ellos es un espacio).

Si quieres completar la parte bonus, piensa en la implementación de las características extras desde el primcipio. De esta forma, evitarás los peligros de un enfoque ingenuo.

La parte bonus solo podrá evaluarse si la parte obligatoria está PERFECTA. Perfecta significa que la parte obligatoria funciona al completo, sin ningún error ni comportamiento inesperado. Si no has superado todos los requisitos de la parte obligatoria, no se evaluará nada de la parte bonus.

Capítulo V

Entrega y evaluacion

Entrega tu proyecto en tu repositorio `Git` como de costumbre. Sólo será evaluado el contenido de dentro de tu repositorio `Git`. Asegúrate de comprobar dos veces los nombres de tus archivos para que sean correctos.

Una vez que superes este ejercicio, podrás incluir `ft_printf()` en tu `libft`, por lo que podrás utilizarla en futuros proyectos en `C`.

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