



printf

Va siendo hora de reemplazar putnbr y putstr

Resumen: Este proyecto va bastante al grano. Deberás programar la función printf. Por suerte para ti, podrás reutilizarla en tus futuros proyectos como una función legítima de tu libft. Aprenderás principalmente a utilizar funciones variádicas.

Versión: 9

Índice general

I.	Introducción	2
II.	Instrucciones generales	3
III.	Parte obligatoria	4
IV.	Parte bonus	6
V.	Entrega y evaluación	7

Capítulo I

Introducción

La versalidad de la función `printf` en `C` hace de este un buen reto de programación. Se puede calificar como un proyecto de dificultad moderada. Te permitirá descubrir las funciones variádicas en `C`.

Un consejo: la clave para un buen `ft_printf` es un código bien estructurado y extensible.

Capítulo II

Instrucciones generales

- 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 dentro.
- 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 alocada en heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el subject lo requiere, deberás entregar un **Makefile** que compilará tus archivos fuente al output requerido con las flags **-Wall**, **-Werror** y **-Wextra**, 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 a tu repositorio **Git** asignado. Solo el trabajo de tu repositorio **Git** será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus compañeros. Si se encuentra un error durante la evaluación de Deepthought, la evaluación terminará.

Capítulo III

Parte obligatoria

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

Debes programar la función `printf` de la `libc`.

Para ello, el prototipo utilizado será:

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

Además, deberás respetar las siguientes consideraciones técnicas:

- No es necesario gestionar el buffer como lo hace el `printf` original.
- Deberá implementar las siguientes conversiones: `cspdiuxX%`.
- Su funcionamiento se contrastará con el `printf` original.
- El uso de `libtool` está prohibido. En cambio, deberás utilizar el comando `ar` para generar la librería.
- Tu archivo `libftprintf.a` deberá generarse en la raíz de tu repositorio.

Una pequeña y simple descripción de las conversiones que se te piden:

- `%c` para imprimir un solo carácter.
- `%s` para imprimir una string (como se define por convención en `C`).
- `%p` el puntero `void *` dado como argumento se imprimirá en hexadecimal.

- **%d** para imprimir un número decimal (de base 10).
- **%i** para imprimir un entero en base 10.
- **%u** para imprimir un número decimal (de base 10) sin signo.
- **%x** para imprimir un número hexadecimal (de base 16), en minúscula.
- **%X** para imprimir un número hexadecimal (de base 16), en mayúscula.
- **%%** para imprimir el signo del porcentaje.

Capítulo IV

Parte bonus

No es necesario hacer todos los bonus.

Lista de bonus:

- Gestiona todas las combinaciones de las siguientes flags para todas las conversiones: “-0.” y el flag de longitud mínima (minimum field width).
- Gestiona todas las combinaciones de las siguientes flags para todas las conversiones: “# +” (sí, una es un espacio).



Si tienes intención de hacer los bonus, deberías pensar en cómo hacerlos desde el principio para evitar enfocar el proyecto incorrectamente.



La parte bonus solo podrá evaluarse si la parte obligatoria está perfecta. Perfecta quiere decir que la parte obligatoria ha conseguido todos los puntos.

1. Minimum Field Width:

Asegura al menos x caracteres de espacios de ancho después del porcentaje

```
x = 12;  
y = 12345;  
printf(“%5d\n”, x)  
output: 12  
printf(“%05\n”, x)  
output: 00012  
printf(“%2\n”, y)  
output: 12345;
```

2. Flags # (espacio) +

+ : imprime el signo del numero
#: para números reales imprime ceros detrás y printa la coma.
Para números que no son base 10, añaden el prefijo de la base.
‘ ‘: printa un espacio delante de los números positivos que no printa +.

Capítulo V

Entrega y evaluación

Entrega tu proyecto en tu repositorio `Git` como de costumbre. Asegúrate de que el nombre de los archivos es correcto. Asegúrate también de que todo tu trabajo está dentro del repositorio, si algo no está dentro no será evaluado.

Una vez completes tu `ft_printf`, podrás incorporarlo a tu `libft` para su posterior utilización en todos los proyectos de `C`; sin miedo a ser considerado un tramposo.