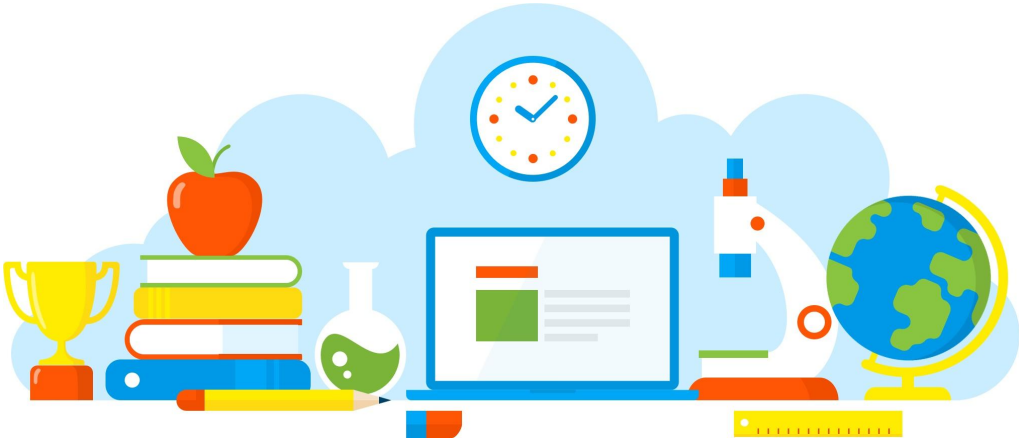
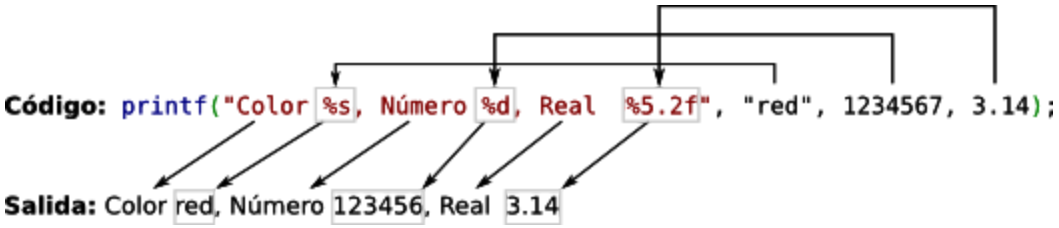


ft_printf

La función **printf** (que deriva su nombre de "*print formatted*") imprime un mensaje por pantalla utilizando un "cadena de formato" que incluye las instrucciones para mezclar múltiples cadenas en la cadena final a mostrar por pantalla.



`printf` es una función especial porque recibe un número variable de parámetros. El primer parámetro es fijo y es la cadena de formato. En ella se incluye texto a imprimir literalmente y ***formateadores*** a reemplazar por el valor de las variables, que se obtiene de los parámetros adicionales. Por tanto, `printf` se llama con tantos parámetros como formateadores haya en la cadena de formato más uno (la propia cadena de formato).

El símbolo "`%`" denota el comienzo del formateador. El formateador "`%d`" se reemplaza por el valor de una variable y se imprime la cadena resultante. La salida, por defecto, se justifica a la derecha del ancho total que le hayamos dado al campo, que por defecto tiene como longitud la longitud de la cadena.

Si en la cadena de formato aparecen varios formateadores, los valores a incluir se toman en el mismo orden en el que aparecen.

https://www.it.uc3m.es/pbasanta/asng/course_notes/input_output_printf_es.html

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 baseft 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.

Funciones Variádicas:

Una **función variádica** es aquella en la que desconocemos la cantidad y el tipo de parámetros con los que vamos a trabajar. Su estructura es la siguiente: tipo nombre(...). La propiedad de poder trabajar con un número indefinido de argumentos viene dada por el operador "**ellipsis (...)**".

Librería stdarg.h

https://www.youtube.com/watch?v=3iX9a_l9W9Y&list=PLrCDpG6TYaYMkUY2zCdNtxV-x8ih4CZSW&index=6&t=11s

> La librería stdarg.h define un nuevo tipo de variable llamada **va_list** y tres macros que pueden utilizarse para conseguir los argumentos de una función variádica **va_start()**, **va_arg()** y **va_end()**.
>
> **va_list:** Las variables de tipo va_list tienen la capacidad de almacenar la información que necesitan las tres macros siguientes para trabajar. Actúan como punteros a la dirección base de cada parámetro.
>
> Es un conjunto de macros que resuelve el problema del parámetro variable en lenguaje C. El problema del parámetro variable se refiere al número variable de parámetros, que puede ser un parámetro o múltiple; el tipo de cada parámetro en los parámetros variables puede ser diferente , También puede ser el mismo; cada parámetro del parámetro variable no tiene un nombre real correspondiente, es muy flexible de usar.
>
> **Uso de VA_LIST**:
>
> - Primero defina una variable de tipo VA_LIST en la función, esta variable es un puntero al parámetro;
> - Luego use la macro VA_START para inicializar la variable VA_LIST que se acaba de definir;

- Luego use VA_ARG para devolver parámetros variables. El segundo parámetro de VA_ARG es el tipo de parámetro que desea devolver. Obtenga los parámetros);
- Finalmente, use la macro VA_END para finalizar la adquisición de parámetros variables.

Lo anterior es el uso específico de va_list. A continuación se explica el significado de cada declaración de va_list (como se muestra en negrita en el ejemplo anterior) y la implementación de va_list.

.va_list ap; defina una variable va_list ap

va_start (ap, v); ejecutar ap = (va_list) & v + _INTSIZEOF (v), ap apunta a la dirección del parámetro después del parámetro v, es decir, ap apunta al primer parámetro variable La dirección en la pila.

va_arg (ap, t), ((t) ((ap + = _INTSIZEOF (t)) -_ INTSIZEOF (t))) toma el valor señalado por el puntero actual y hace ap señale el siguiente parámetro. ap + = sizeof (tipo t), deje ap apuntar a la dirección del siguiente parámetro. Luego devuelva un puntero de tipo t de ap-sizeof (tipo t), que es la dirección del primer parámetro variable en la pila. Luego, use para obtener el contenido de esta dirección.

va_end (ap); Borrar va_list ap.

Módulo: va_start()

void va_start(va_list ap, last_arg)

Esta macro se encarga de inicializar la variable ap para utilizarse con las macros va_arg y va_end. El parámetro last_arg es el último argumento fijo conocido que se pasa a la función, es decir, el argumento antes de la elipsis.

Módulo: va_arg()

type va_arg(va_list ap, type)

Esta función se encarga de recuperar el siguiente argumento en la lista de parámetros de la función de tipo type. Dicho de otro modo, esta macro lee y devuelve el contenido del argumento donde se encuentra el puntero ap. **Después de terminar esta tarea, el puntero se mueve automáticamente al siguiente argumento de la variable, por lo que debe llamarse solo una vez.** El objetivo de tener como parámetro type es que este sea del mismo tipo del dato al que apunta para poder desrefenciarlo.

El primer uso de esta macro después de la macro va_start devuelve el argumento después de *last_arg*. si no hay siguiente argumento ocurrirá un error aleatorio. Sii *ap* es pasado a una función que utiliza va_arg(ap, type) el valor de *ap* es indefinido después de el resultado de la función.

Módulo: va_end()

type va_end(va_list ap, type);

Esta macro permite devolver un resultado adecuado a las funciones variádicas donde se ha utilizado la macro va_start. Si no se llama a va_end antes de devolver el resultado de la función, este resultado será indefinido, por lo que cada invocación de va_start debe estar ligada por la correspondiente invocación de va_end() en la misma función.

Dicho de otro modo, se encarga de finalizar la variable *ap*. Devuelve *ap* como puntero nulo.

Módulo: va_copy()

void **va_copy** *(va_list dest , va_list src)*

Macro permite copiar objetos de tipo va_list, incluso si este no es un tipo integral. El puntero de argumento en dest se inicializa para apuntar al mismo argumento que el puntero en src .

Casos Paranormales para anormales

Para que el printf funcione correctamente, mínimo debe de tener un parámetro, que es el texto que queramos que aparezca printado en la pantalla.

```
printf("Hola Mundo");
Hola Mundo
```

f

Si le metemos formateadores “%c”, tenemos que introducir tantos parámetros de más como formateadores hayamos indicado.

```
printf("Caracter: %c\nString: %s\nNumero: %d", 'r', "Hola", 5);
Caracter: r
String: Hola
Numero: 5
```

El orden en que el programa interpreta los datos es de forma ordenada, por ejemplo. El primer formateador con la primera variable, el segundo con la segunda, etc…

En este apartado vamos a recoger los casos en los que no cumplen esta regla.

Menos formateadores que datos.

- En este caso hemos eliminado el primer formateador.

```
printf("Caracter: \nString: %s\nNumero: %d", 'r', "Hola", 5);

ft_printf.c:24:47: error: format specifies type 'char *' but the argument has type
'char' [-Werror,-Wformat]
```

```
       printf("Caracter: \nString: %s\nNumero: %d", 'r', "Hola", 5);
                                  ~~               ^~~
                                  %c

ft_printf.c:24:52: error: format specifies type 'int' but the argument has type 'char
*' [-Werror,-Wformat]
       printf("Caracter: \nString: %s\nNumero: %d", 'r', "Hola", 5);
                                                 ~~        ^~~~~
                                                 %s

ft_printf.c:24:60: error: data argument not used by format string
[-Werror,-Wformat-extra-args]
       printf("Caracter: \nString: %s\nNumero: %d", 'r', "Hola", 5);
              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~               ^
```

Al interpretar los formateadores en orden, recoge como primero un string, pero como hemos mantenido los parámetros y el primer es un carácter, nos indica error. Que no coinciden los formateadores con sus valores dados, es decir el segundo pasa a ser el primero, el tercero el segundo y así sucesivamente.

- En el caso de eliminar el último formateador, nos indica que hay un parámetro que no hemos usado y nos muestra un error.

```
printf("Caracter: %c\nString: %s\nNumero:", 'r', "Hola", 5);

ft_printf.c:24:59: error: data argument not used by format string
[-Werror,-Wformat-extra-args]
       printf("Caracter: %c\nString: %s\nNumero:", 'r', "Hola", 5);
              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~               ^
```

Menos datos que formateadores

- Si eliminamos el primer dato, nos indica el error de que hay más formateadores que datos.

```
printf("Caracter: %c\nString: %s\nNumero: %d", "Hola", 5);

ft_printf.c:24:45: error: more '%' conversions than data arguments [-Werror,-Wformat]
       printf("Caracter: %c\nString: %s\nNumero: %d", "Hola", 5);
                                                 ~^
```

- Si eliminamos el último dato, nos indica lo mismo, que hay más formateadores que datos.

Metemos datos sin formateadores

```
printf("Caracter: \nString: \nNumero: ", 'r', "Hola", 5);

ft_printf.c:24:43: error: data argument not used by format string
[-Werror,-Wformat-extra-args]
        printf("Caracter: \nString: \nNumero: ", 'r', "Hola", 5);
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  ^
```

- Nos dice que los datos no se van a usar en el string y da error.

Tipos de variables

En la parte obligatoria de ft_printf utilizamos: c, s, p, d, i, u, x, X (abajo en verde)

Specifiers							
length	**d i**	**u o x X**	**f F e E g G a A**	**c**	**s**	**p**	**n**
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar_t*		long int*
ll	long long int	unsigned long long int					long long int*
j	intmax_t	intmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

DESARROLLO

Como hemos explicado antes, esta función lo que tiene que hacer básicamente es mostrar en pantalla un string, que puede contener uno o más parámetros. Para esto usaremos la **va_list**. Que lo que hace es reproducirse mientras haya parámetros.

Si no hay más, intuimos que solo va a ser un texto lo que tengamos que imprimir, esta parte es la fácil, pues metiendo un write en un while o con la función putchar, lo tenemos listo. Ahora bien, si dentro de este string encontramos algún %c,%s, %p, %d, %i, %u, %x, %X, y no tenemos más parámetros debería mostrar un error. Excepto que sea %%, que en este caso nos mostrará un %.

Salida de printf

Printf tiene como salida un int, que es la longitud total del string que va a imprimir en pantalla.

Por ejemplo:

```
printf("Hola %s %i Agur\n", "mundo", 1540);
```

Imprime en pantalla: Hola mundo 1540 Agur

La cadena tiene una longitud de: Hola (4) + espacio (1) + mundo (5) + espacio (1) + 1540 (4) + espacio (1) + Agur (4) + \n (1) = 21

Por lo que la salida de printf es el número 21.

Si defino una variable m integer, en la siguiente línea m coge el valor de 21.

```
m = printf("Hola %s %i Agur\n", "mundo", 1540);

printf("El valor de m es %i \n", m);
```

MAKEFILE

Opción 1	Opción 2 (este programa utiliza un archivo ft_utils.c donde se meten las funciones auxiliares)

```
NAME =  libftprintf.a

SRCS =  ft_printf.c

OBJECTS = $(SRCS:.c=.o)

CC = gcc

CFLAGS = -Wall -Wextra -Werror

AR = ar rcs

RM = rm -f

all: $(NAME)

$(OBJECTS): %.o: %.c

	$(CC) $(CFLAGS) -c $< -o $@

$(NAME):    $(OBJECTS)

	$(AR) $@ $^

clean:

	@$(RM) $(OBJECTS)

fclean:    clean

	@$(RM) $(NAME)

re:         fclean all

.PHONY:     all clean fclean re
```

```
NAME = libftprintf.a

SRCS = ft_printf.c\

      ft_utils.c

OBJS = $(SRCS:.c=.o)

all : $(NAME)

$(NAME): $(OBJS)

   ar rcs $(NAME) $(OBJS)

$(OBJS): $(SRCS)

   gcc -Wall -Wextra -Werror -c $(SRCS)

clean:

   rm -f $(OBJS)

fclean : clean

   rm -f $(NAME)

re: fclean all

.PHONY : all clean fclean re
```

ft_puthexa.C

```c
int ft_puthexa_fd(unsigned long int n, char c, int fd)
{
   int     long_n;

   long_n = 1;
   if (c == 'p' || c == 'x')
   {
       if (n >= 16)
           long_n = long_n + ft_puthexa_fd(n / 16, c, fd);
       ft_putchar_fd("0123456789abcdef"[n % 16], fd);
   }
   else if (c == 'X')
   {
       if (n >= 16)
           long_n = long_n + ft_puthexa_fd(n / 16, c, fd);
       ft_putchar_fd("0123456789ABCDEF"[n % 16], fd);
   }
   return (long_n);
}
```

Ojo:

```c
ft_puthexa_fd(n / 16, c, fd)
ft_putchar_fd("0123456789ABCDEF"[n % 16], fd)
```

Ejemplo: El número 2619 decimal en hexadecimal es = A3B. ¿Cómo se calcula el hexadecimal?

Hexadecimal puede tener estos valores: 0123456789ABCDEF. Donde A=10, B=11, C=12, D=13, E=14, F=15.

Si nuestro número n = 2619 (decimal)

2619 / 16 = 163 (2619 % 16 = módulo = 11, donde en hexadecimal 11 = B)

163 / 16 = 10 (163 % 16 = módulo = 3)

10 / 16 = 0 (10 %16 = módulo = 10, donde en hexadecimal 10 = A)

Resultado: A3B

ft_putnbr.C

```c
int ft_putnbr_fd(int n, int fd)
{
   unsigned int    num;
   int             long_n;

   num = 0;
   long_n = 1;
   if (n == INT_MIN)
   {
       write(fd, "-2147483648", 11);
       return (11);
   }
   if (n < 0)
   {
       ft_putchar_fd('-', fd);
       num = -n;
       long_n++;
   }
   else
       num = n;
   if (num > 9)
   {
       long_n = long_n + ft_putnbr_fd(num / 10, fd);
   }
   ft_putchar_fd((char)(num % 10 + '0'), fd);
   return (long_n);
}
```

Ojo:

```c
ft_putchar_fd((char)(num % 10 + '0'), fd);
```

MAIN.C

Programa main para chequear que printf funciona bien

```c
#include <stdio.h>

#include <limits.h>
```

```c
int main()

{

    int len = 0, len1 = 0;



    printf("\n---NO %% CASE---\n");

    len = printf("Printf: Aqui no pasamos ningún parámetro adicional\n");

    len1 = ft_printf("Ft_printf: Aqui no pasamos ningún parámetro adicional\n");

    printf("Printf = %d\tFt_printf = %d\n", len, len1 - 3);

    printf("\n---%% CASE---\n");

    len = printf("Printf: %%\n");

    len1 = ft_printf("Ft_printf: %%\n");

    printf("%d \t%d\n", len, len1 - 3);

    ft_printf("\n---CHARACTER CASE---\n");

    len = printf("Printf: Estos son varios caracteres: %c, %c, %c, %c, %c, %c, %c.\n",
'a','1','\0','Z',  '0' - 256, '0' + 256, ' ');

    len1 = ft_printf("Ft_printf: Estos son varios caracteres: %c, %c, %c, %c, %c, %c, %c.\n",
'a','1','\0','Z', '0' - 256, '0' + 256, ' ');

    printf("Printf = %d\tFt_printf = %d\n", len, len1 - 3);

    ft_printf("\n---SRING CASE---\n");

    len = printf("Printf: Estas son varias cadenas: %s, %s, %s, %s, %s.\n", "", "c", "4",
"cadena ", NULL);

    len1 = ft_printf("Ft_printf: Estas son varias cadenas: %s, %s, %s, %s, %s.\n", "", "c",
"4", "cadena ", NULL);

    printf("Printf = %d\tFt_printf = %d\n", len, len1 - 3);

    ft_printf("\n---POINTER CASE---\n");

    len = printf("Printf: Estas son varias direcciones: %p, %p, %p, %p, %p, %p, %p, %p, %p,
%p, %p, %p, %p, %p, %p.\n", 0, 0, NULL, NULL, -1, 1, 15, 16, 17, LONG_MAX, LONG_MIN, INT_MAX,
INT_MIN, ULLONG_MAX, -ULONG_MAX);

    len1 = ft_printf("Ft_printf: Estas son varias direcciones: %p, %p, %p, %p, %p, %p, %p, %p,
%p, %p, %p, %p, %p, %p, %p.\n", 0, 0, NULL, NULL, -1, 1, 15, 16, 17, LONG_MAX, LONG_MIN,
INT_MAX, INT_MIN, ULLONG_MAX, -ULONG_MAX);

    printf("Printf = %d\tFt_printf = %d\n", len, len1 - 3);

    ft_printf("\n---LOWERHEX CASE---\n");
```

```
    len = printf("Printf: Estos son varios numeros en hexadecimal txiki: %x, %x, %x, %x, %x,
%x, %x, %x, %x, %x, %x, %x, %x, %x, %x.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101, -15, -16, -17,
-99, -100, -101);

    len1 = ft_printf("Ft_printf: Estos son varios numeros en hexadecimal txiki: %x, %x, %x,
%x, %x, %x, %x, %x, %x, %x, %x, %x, %x, %x, %x.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101, -15,
-16, -17, -99, -100, -101);

    len += printf("Printf: Estos son varios numeros en hexadecimal txiki: %x, %x, %x, %x, %x,
%x, %x.\n", INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX, 9223372036854775807LL);

    len1 += ft_printf("Ft_printf: Estos son varios numeros en hexadecimal txiki: %x, %x, %x,
%x, %x, %x, %x.\n", INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX,
9223372036854775807LL);

    printf("Printf = %d\tFt_printf = %d\n", len, len1 - 6);

    ft_printf("\n---UPPERHEX CASE---\n");

    len = printf("Printf: Estos son varios numeros en hexadecimal mayor: %X, %X, %X, %X, %X,
%X, %X, %X, %X, %X, %X, %X, %X, %X, %X.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101, -15, -16,
-17, -99, -100, -101);

    len1 = ft_printf("Ft_printf: Estos son varios numeros en hexadecimal mayor: %X, %X, %X,
%X, %X, %X, %X, %X, %X, %X, %X, %X, %X, %X, %X.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101,
-15, -16, -17, -99, -100, -101);

    len += printf("Printf: Estos son varios numeros en hexadecimal mayor: %X, %X, %X, %X, %X,
%X, %X.\n", INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX, 9223372036854775807LL);

    len1 += ft_printf("Ft_printf: Estos son varios numeros en hexadecimal mayor: %X, %X, %X,
%X, %X, %X, %X.\n", INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX,
9223372036854775807LL);

    printf("Printf = %d\tFt_printf = %d\n", len, len1 - 6);

    ft_printf("\n---DECIMAL CASE---\n");

    len = printf("Printf: Estos son varios numeros decimales: %d, %d, %d, %d, %d, %d, %d, %d,
%d, %d, %d, %d, %d, %d, %d, %d, %d.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101, -15, -16, -17, -99,
-100, -101);

    len1 = ft_printf("Ft_printf: Estos son varios numeros decimales: %d, %d, %d, %d, %d, %d,
%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101, -15, -16, -17,
-99, -100, -101);

    len += printf("Printf: Estos son varios numeros decimales: %d, %d, %d, %d, %d, %d, %d.\n",
INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX, 9223372036854775807LL);

    len1 += ft_printf("Ft_printf: Estos son varios numeros decimales: %d, %d, %d, %d, %d, %d,
%d.\n", INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX, 9223372036854775807LL);

    printf("Printf = %d\tFt_printf = %d\n", len, len1 - 6);

    ft_printf("\n---INTEGER CASE---\n");
```

```
    len = printf("Printf: Estos son varios numeros enteros: %i, %i, %i, %i, %i, %i, %i, %i,
%i, %i, %i, %i, %i, %i, %i.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101, -15, -16, -17, -99, -100,
-101);

    len1 = ft_printf("Ft_printf: Estos son varios numeros enteros: %i, %i, %i, %i, %i, %i, %i,
%i, %i, %i, %i, %i, %i, %i, %i.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101, -15, -16, -17, -99,
-100, -101);

    len += printf("Printf: Estos son varios numeros enteros: %i, %i, %i, %i, %i, %i, %i.\n",
INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX, 9223372036854775807LL);

    len1 += ft_printf("Ft_printf: Estos son varios numeros enteros: %i, %i, %i, %i, %i, %i,
%i.\n", INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX, 9223372036854775807LL);

    printf("Printf = %d\tFt_printf = %d\n", len, len1 - 6);

    ft_printf("\n---UNSIGNED INT CASE---\n");

    len = printf("Printf: Estos son varios numeros enteros sin signo: %u, %u, %u, %u, %u, %u,
%u, %u, %u, %u, %u, %u, %u, %u, %u.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101, -15, -16, -17,
-99, -100, -101);

    len1 = ft_printf("Ft_printf: Estos son varios numeros enteros sin signo: %u, %u, %u, %u,
%u, %u, %u, %u, %u, %u, %u, %u, %u, %u, %u.\n", 0, -1, 1, 15, 16, 17, 99, 100, 101, -15, -16,
-17, -99, -100, -101);

    len += printf("Printf: Estos son varios numeros enteros sin signo: %u, %u, %u, %u, %u, %u,
%u.\n", INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX, 9223372036854775807LL);

    len1 += ft_printf("Ft_printf: Estos son varios numeros enteros sin signo: %u, %u, %u, %u,
%u, %u, %u.\n", INT_MAX, INT_MIN, LONG_MAX, LONG_MIN, UINT_MAX, ULONG_MAX,
9223372036854775807LL);

    printf("Printf = %d\tFt_printf = %d\n", len, len1 - 6);

}
```