Printf

> `PRINTF`

```c
//Main function.
int ft_printf(const char *str, ...)
{
    int     i;
    int     len;
    int     total;
    va_list param;

    i = 0;
    len = 0;
    total = 0;
    va_start(param, str);
    while (str[i] != '\0')
    {
        if (str[i] == '%')
            len = ft_format_check(str[++i], param);
        else
            len = write (1, &str[i], 1);
        if (len == -1)
            return (-1);
        total += len;
        i++;
    }
    va_end(param);
    return (total);
}
/*
int main(void)
{
//  printf("%i\n", ft_printf("Hola %s\n", "Borja"));
        printf("%i\n", printf(" NULL %s NULL ", NULL));
        printf("%i\n", ft_printf(" NULL %s NULL ", NULL));
    return (0);
}*/
```

- **Definición de la función**:

La función `ft_printf` toma como argumentos un puntero a una cadena de caracteres `str` y una cantidad variable de argumentos.

- **Iniciaización de variables enteras:**

    - `i`: itera sobre la cadena `str`.
    - `len`: almacena la longitud de los caracteres escritos por cada llamada a write.
    - `total`: mantiene un seguimiento de la cantidad total de caracteres escritos.
- **Inicializacion lde la lista de argumentos:**

Se declara una variable de tipo `va_list` llamada `param` para trabajar con la lista variable de argumentos.

    - Se inicializa utilizando `va_start`, pasando la lista de argumentos (param) y la cadena de formato (str).
    - Se inicia un bucle `while` que itera str hasta que se alcance el carácter nulo ('\0').
    - En el `while`, se comprueba si el carácter actual es un signo de porcentaje ('%'). Si es así, llama a la función `ft_format_check` pasando el siguiente carácter después del signo de porcentaje y la lista de argumentos (param) almacenando el valor devuelto en `len`.
    - Si el carácter actual no es %, se escribe ese carácter directamente en la salida estándar utilizando la función `write`.
- **Verificación de errores**:

Se comprueba si len es igual a -1. En ese caso, se devuelve -1 para indicar un error.

- **Actualización del contador total**:

Se actualiza el contador total sumando el valor de `len`, que representa la cantidad de caracteres escritos en esta iteración.

- **Iteración sobre la cadena**:

Se incrementa `i` para pasar al siguiente carácter de la cadena str.

- **Finalización de va_list**:

Después de iterar, se finaliza el manejo de la lista de argumentos con `va_end`.

- **Retorno del total de caracteres escritos**:

Se devuelve total, que representa la cantidad total de caracteres escritos en la salida estándar.

`FORMAT_CHECK`

```c
//Check if its alpha,num,unsigned int,hexad[16]+,hexad[16]-,ptr(0x).Ret len.
int ft_format_check(const char format, va_list param)
{
    int len;

    len = 0;
    if (format == 'c')
        len = ft_putchar(va_arg(param, int));
    else if (format == 's')
        len = ft_putstr(va_arg(param, char *));
    else if (format == 'p')
        len = ft_puthex(va_arg(param, unsigned long), "0123456789abcdef", 1);
    else if (format == 'd' || format == 'i')
        len = ft_putnbr(va_arg(param, int));
    else if (format == 'u')
        len = ft_putnbr(va_arg(param, unsigned int));
    else if (format == 'x')
        len = ft_puthex(va_arg(param, unsigned int), "0123456789abcdef", 0);
    else if (format == 'X')
        len = ft_puthex(va_arg(param, unsigned int), "0123456789ABCDEF", 0);
    else if (format == '%')
        len = ft_putchar('%');
    if (len == -1)
        return (-1);
    return (len);
}
```

- **Inicialización de variables**: `len` = 0 para almacenar la longitud del resultado formateado.
- **Selección de formato**: Se verifica el valor de `format` para determinar qué tipo de formato se está procesando.
- **Formato 'c'**: imprime un solo carácter utilizando `ft_putchar` y se asigna la longitud devuelta a `len`.
- **Formato 's'**: imprime una cadena de caracteres utilizando `ft_putstr` y se asigna la longitud devuelta a `len`.
- **Formato 'p'**: imprime un número hexadecimal utilizando `ft_puthex` y se asigna la longitud devuelta a `len`.
- **Formatos 'd' e 'i'**: imprime un número entero utilizando `ft_putnbr` y se asigna la longitud devuelta a `len`.
- **Formato 'u'**: imprime un número entero sin signo utilizando `ft_putnbr` y se asigna la longitud devuelta a `len`.

- **Formatos 'x' y 'X'**: imprime un número hexadecimal en minúsculas o mayúsculas utilizando `ft_puthex` y se asigna la longitud devuelta a `len`.
- **Formato '%'**: imprime el carácter '%' y se asigna la longitud devuelta a `len`.
- **Verificación de errores**: comprueba si `len` es igual a -1 después de cada operación para detectar errores. Si es -1, se devuelve -1; de lo contrario, se devuelve len.

PUTCHAR

```c
//Print alpha. Return len.
int ft_putchar(char c)
{
    if (write(1, &c, 1) == -1)
        return (-1);
    return (1);
}
```

- **Entrada**: Toma un carácter `c` como entrada.
- **Acción**: Escribe el carácter en la salida estándar (stdout).
- **Retorno**:
  - Si tiene éxito, devuelve 1.
  - Si hay un error, devuelve -1.

PUTSTR

```c
//Print str. Return len.
int ft_putstr(char *str)
{
    int i;

    i = 0;
    if (!str)
    {
        str = "(null)";
    }
    while (str[i] != '\0')
    {
        if (write (1, &str[i], 1) == -1)
            return (-1);
        i++;
    }
    return (i);
}
```

- **Inicialización**: indice i = 0.
- **Verificación de nulidad**: Si el puntero str es nulo, se asigna la cadena "(null)".
- **Impresión de caracteres**: Se imprime cada carácter de la cadena str uno por uno.
- **Manejo de errores**: Se verifica si hay errores al escribir en la salida estándar.
- **Incremento del índice**: Se incrementa el índice para avanzar en la cadena.
- **Retorno**: Se devuelve la cantidad de caracteres impresos o -1 en caso de error.

> PUTNBR

```c
//Print number. Return len.
long    ft_putnbr(long nb)
{
    int     len;
    int     check;

    len = 0;
    if (nb < 0)
    {
        if (ft_putchar('-') == -1)
            return (-1);
        len++;
        nb = -nb;
    }
    if (nb > 9)
    {
        check = ft_putnbr(nb / 10);
        if (check == -1 || ft_putchar((nb % 10) + 48) == -1)
            return (-1);
        len += check + 1;
    }
    else
    {
        if (ft_putchar(nb + 48) == -1)
            return (-1);
        len++;
    }
    return (len);
}
```

- **Declaración de variables:**

Se declaran dos variables de tipo entero:

  - `len`: cuenta los caracteres impresos.
  - `check`: almacena el resultado de llamadas recursivas a `ft_putnbr`.

- **Comprobación de signo**:

Si es negativo, se imprime el signo "-" y se incrementa el contador `len`. Luego, se convierte el número `nb` a positivo multiplicándolo por -1.

- **División y módulo**:
  - Se divide el número `nb` entre 10 para obtener el cociente.
  - Si el cociente > 9, el número tiene más de un dígito, se llama recursivamente a `ft_putnbr` con el cociente como argumento. El resultado se almacena en `check`.

- Se toma el módulo de `nb` con 10 para obtener el último dígito del número y se imprime sumándole 48, (valor ASCII de '0') para convertirlo a char.
- Se incrementa el contador `len` según la cantidad de caracteres impresos en esta llamada recursiva.

- **Caso base y salida recursiva**:

  Si `nb` <=9, significa que solo tiene un dígito. Se imprime ese dígito sumándole 48, y se incrementa `len`. Esto actúa como el caso base para la recursión.

- **Retorno**:

La función devuelve len, que es la cantidad total de caracteres impresos.

PUTHEX

```c
//Check hex with base[16] && put 0x if its pointer. Return len
long    ft_puthex(unsigned long nb, char *base, int isptr)
{
    int len;
    int check;

    len = 0;
    if (isptr == 1)
    {
        if (ft_putstr("0x") == -1)
            return (-1);
        len += 2;
    }
    if (nb > 15)
    {
        check = ft_puthex(nb / 16, base, 0);
        if (check == -1 || ft_putchar(base[(nb % 16)]) == -1)
            return (-1);
        len += check + 1;
    }
    else
    {
        if (ft_putchar(base[nb]) == -1)
            return (-1);
        len++;
    }
    return (len);
}
```

- **Inicialización de variables**:

Se inicializa la variable `len` a 0 para llevar la cuenta de la longitud total de la cadena hexadecimal.

- **Comprobación de la bandera isptr**:

Si `isptr` (puntero) es 1, se imprime el prefijo "0x" utilizando `ft_putstr` y se incrementa `len` en 2 para contar los dos caracteres agregados ("0x").

- **Conversión del número a hexadecimal**:
  - Si `nb` > 15, se realiza la conversión dividiendo `nb` por 16 y llamando recursivamente a la función `ft_puthex` con el cociente.
  - El resultado de la llamada recursiva se almacena en `check`.
  - Se imprime el dígito hexadecimal correspondiente al residuo de la división (nb % 16) utilizando `ft_putchar`.
  - Se suma 1 a `check` (la longitud de la parte convertida) y se agrega a `len`.
- **nb <= 15**:
  - Si `nb` <= 15, se imprime directamente el dígito hexad. usando `ft_putchar`.
  - Se incrementa len en 1.
- **Retorno de longitud total**:

La función devuelve `len`, que representa la longitud total de la cadena hexadecimal generada e impresa.

> VA_

- **va_list**:
  - Tipo de dato utilizado para declarar una lista de argumentos variables en funciones.
  - Almacenar los argumentos que se pasan a una función con un número variable de parámetros.
  - Declarar una variable de tipo va_list proporciona un mecanismo para acceder a los argumentos variables.
- **va_start**:
  - Macro utilizada para inicializar el uso de una lista de argumentos variables.
  - Toma dos argumentos: la lista de argumentos (va_list) y el último argumento conocido en la función.

    - Se utiliza para establecer el punto de inicio desde el cual se pueden acceder los argumentos variables dentro de la función.
- **va_arg**:
    - Macro utilizada para acceder a los argumentos individuales en una lista de argumentos variables.
    - Toma dos argumentos: la lista de argumentos (va_list) y el tipo del argumento que se desea recuperar.
    - Devuelve el siguiente argumento de la lista con el tipo especificado, avanzando la posición de la lista.
- **va_end**:
    - Macro utilizada para limpiar y cerrar una lista de argumentos variables una vez que ya no se necesite.
    - Toma un único argumento que es la lista de argumentos (va_list).
    - Se utiliza para realizar las operaciones de limpieza necesarias y liberar los recursos asociados a la lista de argumentos.