ft_printf

Este es un proyecto de programación en C que desafía a los estudiantes de programación a crear su propia versión de la función **printf**, la famosa función de impresión de formato de C.

La idea es recrear el comportamiento de dicha función, que permite imprimir diferentes tipos de datos en la consola, utilizando diversos formatos de especificadores.

La complejidad de este proyecto radica en la gran cantidad de especificadores de formato y funcionalidades que debe manejar **printf**. Algunos de los especificadores más comunes son:

- **%c** Imprime un solo carácter.
- %s Imprime una strig (como se define por defecto en C).
- %p Imprime un puntero void * dado como argumento (formato hexadecimal).
- %d y %i Imprime un número entero (formato decimal).
- %u Imprime un número (sin signo).
- %x Imprime un número (formato hexadecimal en minúsculas).
- %X Imprime un número (formato hexadecimal en mayúsculas).
- %% Imprime el símbolo de porcentaje.

Además de los especificadores de formato, **printf** también admite flags, anchos mínimos, precisiones y longitudes, lo que aumenta la dificultad del proyecto (lo que no se incluye aquí).

El objetivo es que el estudiante adquiera un profundo conocimiento de la función **printf** y de la programación en C, en particular, del manejo de cadenas de caracteres, funciones variádicas, formatos de impresión y conversión de tipos de datos.

En resumen, **ft_printf** es un proyecto estimulante y desafiante que ayuda a los futuros programadores a mejorar sus habilidades en programación en C y a comprender mejor el funcionamiento de la función **printf**.

Si aceptas el desafío … ¡Buena suerte en tu proyecto!

FUNCIONES VARIÁDICAS

Las funciones variádicas, también conocidas como funciones con argumentos variables, son funciones que pueden recibir un número variable de argumentos. Esto significa que la función puede ser llamada con cualquier cantidad de argumentos, desde cero hasta un número ilimitado.

En la mayoría de los lenguajes de programación, las funciones solo pueden recibir un número fijo de argumentos. Por ejemplo, en C, la función **printf** puede tener un número fijo de argumentos, donde el primer argumento es una cadena de formato y los argumentos siguientes son los valores a imprimir.

Sin embargo, en algunos casos, es útil tener una función que pueda recibir un número variable de argumentos. Por ejemplo, en la misma función **printf** en C también puede recibir un número variable de argumentos, lo que permite imprimir una cadena de formato con cualquier cantidad de valores.

Para implementar funciones variádicas en C, se utiliza una convención especial llamada "lista de argumentos variable" o "varargs". Esta convención permite pasar una lista de argumentos variable a una función utilizando un puntero a una lista de argumentos.

La función **printf** en C por tanto, es un claro ejemplo de una función variádica.

La declaración de la función **printf** en C se vería así:

```
int printf(const char *format, ...);
```

El primer argumento de la función **printf** es una cadena de formato, y el segundo argumento es una lista de argumentos variable, representada por los tres puntos (**...**).

Para acceder a los argumentos de la lista variable, se utiliza una función especial llamada **va_start**, que inicializa una variable especial llamada **va_list**. Luego, se utiliza la función **va_arg** para obtener cada argumento de la lista variable.

ft_printf.c

La función **ft_printf** es una implementación personalizada de la función **printf** de la biblioteca estándar de C. Su objetivo es imprimir texto y variables en la pantalla, con formato y opciones personalizables.

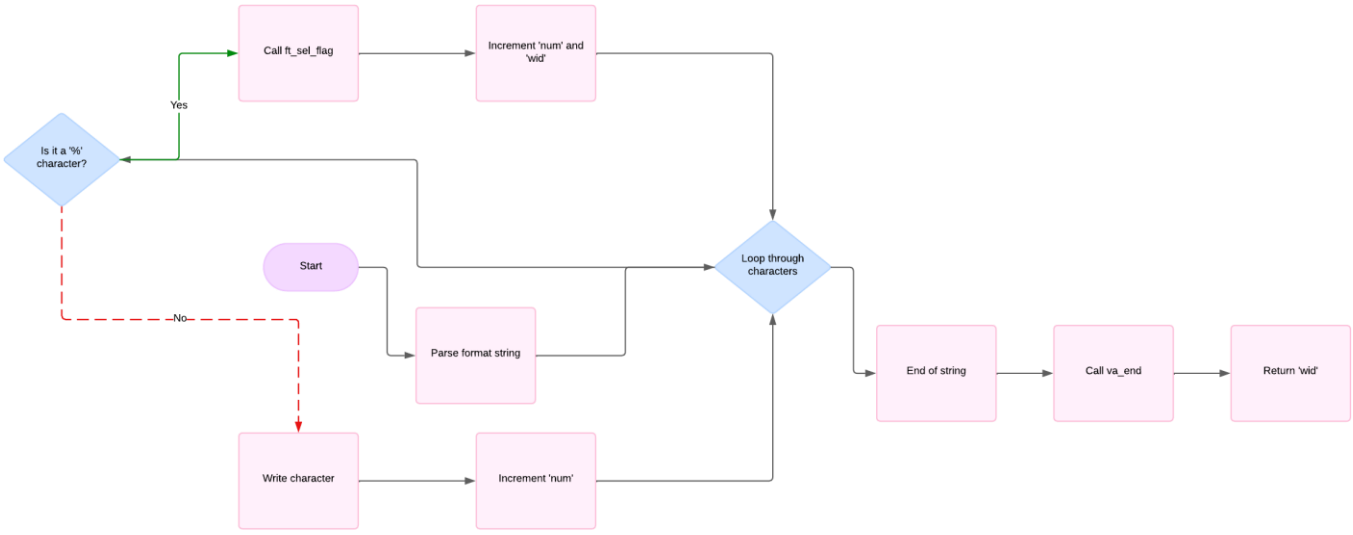
Estructura: La función **ft_printf** tiene dos partes principales:

- La función **ft_printf** en sí misma, que recibe una cadena de caracteres (**str**) y una lista de argumentos variables (**...**).
- La función **ft_sel_flag**, que se encarga de procesar los flags de formato (como **%c**, **%s**, **%d**, etc.) y devuelve el número de caracteres impresos.

Funcionamiento:

ft_printf:

- La función **ft_printf** recibe una cadena de caracteres (**str**) y una lista de argumentos variables (**...**).
- Se inicializan dos variables: **num** para contar el número de caracteres procesados en la cadena **str**, y **wid** para acumular el número de caracteres impresos.
- Se utiliza la función **va_start** para inicializar la lista de argumentos variables **arg**.
- Se itera sobre la cadena **str** caracter por caracter:
    - Si el caracter actual es **%**, se llama a la función **ft_sel_flag** para procesar el flag de formato correspondiente. Se pasa la lista de argumentos **arg** y el caracter siguiente (**str[num + 1]**) como parámetros.
    - Si el caracter actual no es **%**, se imprime directamente en la pantalla utilizando la función **write**.
- Finalmente, se devuelve el número total de caracteres impresos (**wid**).



ft_sel_flag:

- La función **ft_sel_flag** recibe una lista de argumentos variables (**arg**) y un caracter (**wrd**) que representa el flag de formato.
- Se inicializa una variable **wid** para acumular el número de caracteres impresos.
- Se utiliza un conjunto de **if-else** para determinar qué tipo de flag de formato se está procesando:
    - Si el flag es **c**, se llama a la función **ft_write_chr** para imprimir un carácter.
    - Si el flag es **s**, se llama a la función **ft_write_str** para imprimir una cadena de caracteres.
    - Si el flag es **p**, se llama a la función **ft_write_ptr** para imprimir una dirección de memoria.
    - Si el flag es **d** o **i**, se llama a la función **ft_write_nbr** para imprimir un número entero.
    - Si el flag es **u**, se llama a la función **ft_write_dns** para imprimir un número entero sin signo.
    - Si el flag es **x** o **X**, se llama a la función **ft_write_hxl** o **ft_write_hxu** para imprimir un número hexadecimal.
    - Si el flag es **%**, se llama a la función **ft_write_chr** para imprimir el carácter **%**.
    - En cualquier otro caso, se llama a la función **ft_write_chr** para imprimir el carácter correspondiente.
- Se devuelve el número total de caracteres impresos (**wid**).

En resumen, la función **ft_printf** itera sobre una cadena de caracteres y procesa los flags de formato utilizando la función **ft_sel_flag**, que a su vez llama a funciones específicas para imprimir diferentes tipos de variables.

ft_write_chr.c

Como hemos visto anteriormente, la función **ft_printf** recibe como parámetro un puntero a una cadena de caracteres (**char const *str**) que contiene el formato a imprimir y un número variable de argumentos (**...**).

La función itera sobre cada carácter de la cadena de formato. Si el carácter es un **%**, significa que debe imprimir un valor según un formato específico, y por lo tanto llama a la función **ft_sel_flag** para seleccionar el formato y obtener el ancho del resultado. Si el carácter no es un **%**, simplemente imprime el carácter llamando a la función **write** con el descriptor de archivo 1 (que representa la salida estándar).

La función **ft_sel_flag** recibe como parámetro un puntero a una lista de argumentos (**va_list *arg**) y un carácter (**const char wrd**) que representa el formato a seleccionar. La función itera sobre cada formato posible y compara el carácter recibido con el formato. Si hay una coincidencia, la función llama a la función apropiada para imprimir el valor (por ejemplo, **ft_write_chr** para el formato **%c** o **ft_write_str** para el formato **%s**). Después de imprimir el valor, la función devuelve el ancho del resultado.

En concreto, la función **ft_write_chr** recibe como parámetro un entero (**int chr**) y escribe un solo carácter con el valor de **chr** en la salida estándar llamando a la función **write** con el descriptor de archivo 1 y el puntero a **chr** y el tamaño de 1. La función devuelve el tamaño escrito (que será 1 si se escribió correctamente).

ft_write_dns.c

Objetivo: La función **ft_write_dns** tiene como objetivo imprimir un número entero (**unsigned int**) en la pantalla, utilizando la función **write**. El nombre "dns" se refiere a "decimal number string" (cadena de número decimal).

Estructura: La función tiene la siguiente estructura:

- Incluye dos archivos de cabecera (**ft_printf.h**) que contienen definiciones y funciones relacionadas con la impresión de texto.
- La función **ft_write_dns** toma un parámetro **n** de tipo **unsigned int**, que es el número que se quiere imprimir.
- La función devuelve un valor de tipo **int**, que representa el número de caracteres impresos.

Funcionamiento: Aquí es donde las cosas se ponen interesantes. La función utiliza una técnica llamada "recursividad" para imprimir el número. La recursividad es cuando una función se llama a sí misma repetidamente hasta que se cumple una condición.

Aquí se explica cómo funciona:

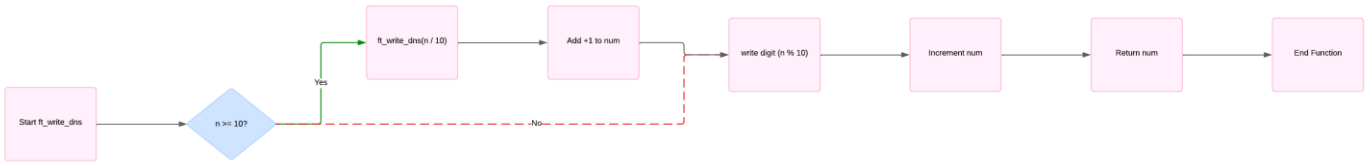
1. La función inicializa una variable **num** a 0. Esta variable se utilizará para contar el número de caracteres impresos.
2. La función verifica si el número **n** es mayor o igual a 10. Si es así, llama a sí misma con el valor de **n** dividido entre 10. Esto se llama "llamada recursiva".
3. La función escribe en la pantalla el carácter correspondiente al resto de la división de **n** entre 10, utilizando la función **write**. El carácter se selecciona de una cadena de caracteres que contiene los dígitos del 0 al 9 (**"0123456789"**). El índice del carácter se calcula con **n % 10**.
4. La función incrementa la variable **num** en 1, ya que se ha impreso un carácter.
5. La función devuelve el valor de **num**, que representa el número de caracteres impresos.

Ejemplo: Supongamos que llamamos a la función con el valor **n = 123**. Aquí's cómo se ejecutaría:

1. La función se llama a sí misma con **n = 12** (123 / 10).
2. La función se llama a sí misma con **n = 1** (12 / 10).
3. La función escribe el carácter '1' en la pantalla (1 % 10).
4. La función devuelve 1 y se vuelve a llamar a sí misma con **n = 12**.
5. La función escribe el carácter '2' en la pantalla (12 % 10).

6. La función devuelve 2 y se vuelve a llamar a sí misma con **n = 123**.
7. La función escribe el carácter '3' en la pantalla (123 % 10).
8. La función devuelve 3 y termina.

Al final, la función habrá impreso el número "123" en la pantalla y devuelto el valor 3, que es el número de caracteres impresos.



ft_write_hxl.c & ft_write_hxu.c

Objetivo: La función **ft_write_hxl** tiene como objetivo imprimir un número entero sin signo (**unsigned int**) en hexadecimal en la salida estándar (la pantalla) y devuelve el número de caracteres impresos.

Estructura: La función tiene la siguiente estructura:

- Incluye el archivo de cabecera **ft_printf.h**, que probablemente contiene funciones relacionadas con la impresión de cadenas de caracteres.
- La función **ft_write_hxl** toma un parámetro **n** de tipo **unsigned int**, que es el número que se quiere imprimir en hexadecimal.
- La función devuelve un valor de tipo **int**, que representa el número de caracteres impresos.

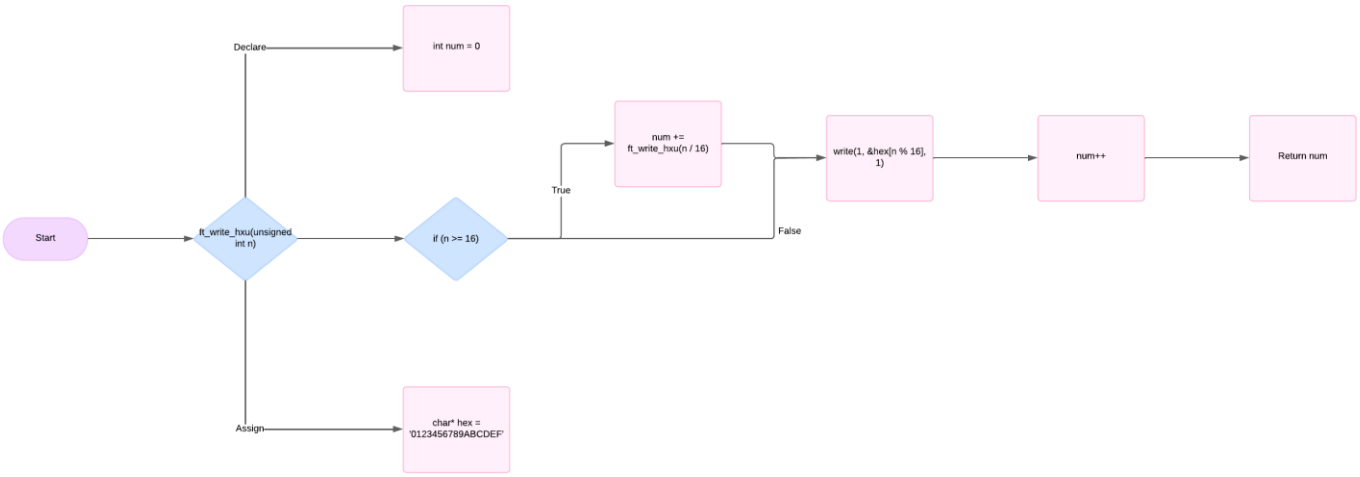
Funcionamiento: Esta función utiliza también la técnica de recursividad para imprimir el número en hexadecimal. Recordemos que la recursividad es cuando una función se llama a sí misma.

Aquí hay un paso a paso de su funcionamiento:

- Se inicializan dos variables: **num** se establece en 0 y **hex** se establece en una cadena de caracteres que contiene los dígitos hexadecimales de 0 a 9 y las letras de a a f (en minúscula).
- Se verifica si el número **n** es mayor o igual a 16. Si es así, se llama a la función **ft_write_hxl** nuevamente, pero esta vez con el valor de **n** dividido entre 16. Esto es la parte recursiva de la función.
- Luego, se utiliza la función **write** para imprimir un solo carácter en la salida estándar. El carácter se selecciona de la cadena **hex** utilizando el operador **%** (módulo), que devuelve el resto de la división de **n** entre 16. Por ejemplo, si **n** es 20, **n % 16** sería 4, por lo que se imprimiría el carácter en la posición 4 de la cadena **hex**, que es el carácter '4'.
- Se incrementa el valor de **num** en 1, ya que se ha impreso un carácter.
- Finalmente, se devuelve el valor de **num**, que representa el número de caracteres impresos.

Ejemplo: Si se llama a la función **ft_write_hxl** con el valor 20, se imprimiría la cadena "14" en la salida estándar, ya que 20 en hexadecimal es 14. La función devolvería el valor 2, ya que se han impreso 2 caracteres.

En cuanto a la función ft_write_hxu.c ésta es exactamente igual a ft_write_hxl.c pero estableciendo la cadena de caracteres hexadecimales de 0 a 9 y las letras de A a F en mayúsculas.



ft_write_nbr.c

Objetivo: Las funciones incorporadas aquí, están diseñadas para imprimir números enteros y unsigned (sin signo) en la pantalla utilizando una función llamada **ft_write_chr** y **ft_write_str**, que probablemente escriben un carácter o una cadena en la pantalla, respectivamente. La función **ft_write_uns** se encarga de imprimir números unsigned, mientras que la función **ft_write_nbr** se encarga de imprimir números enteros (con signo).

Estructura: Ambas funciones tienen una estructura similar. Tienen un parámetro de entrada, que es el número que se quiere imprimir, y devuelven un valor que indica la cantidad de caracteres impresos.

Funcionamiento:

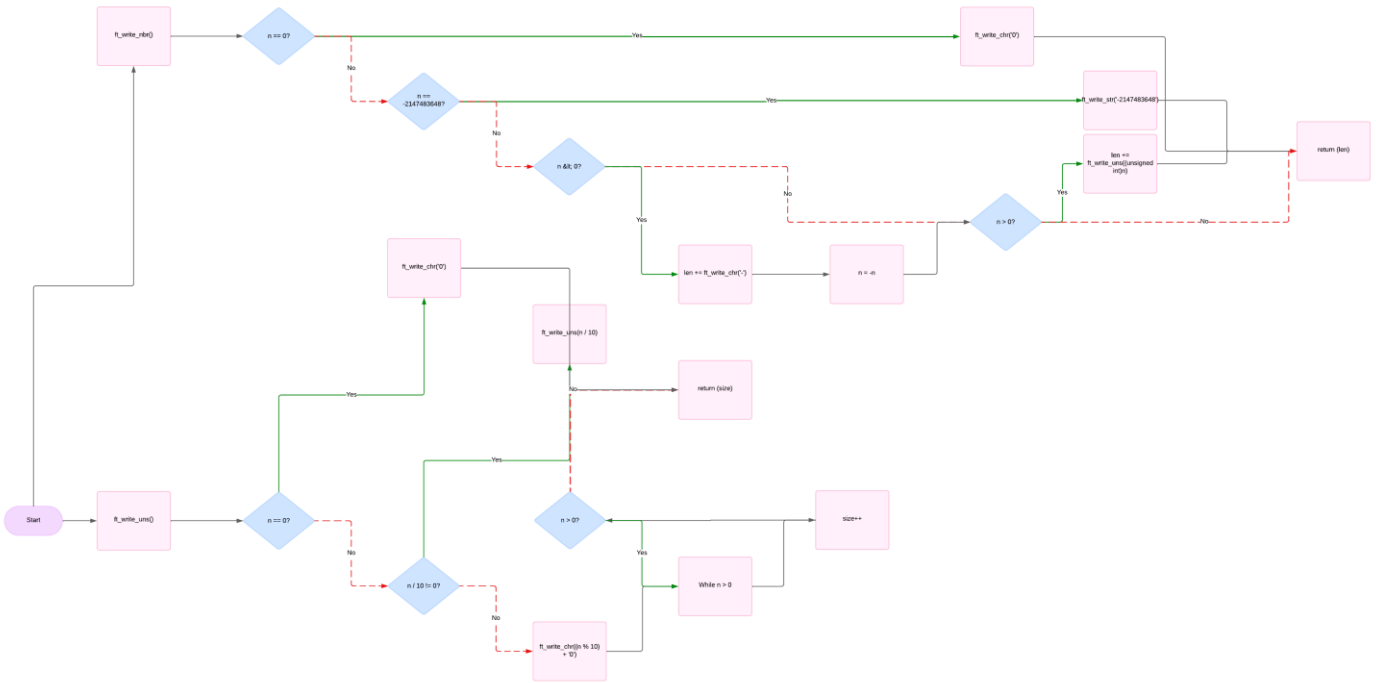
ft_write_uns:

1. La función recibe un número unsigned **n** como parámetro.
2. Se inicializa una variable **size** en 0, que se utilizará para contar la cantidad de caracteres impresos.
3. Si **n** es 0, se imprime el carácter '0' utilizando **ft_write_chr** y se incrementa **size** en 1.
4. Si **n** es mayor que 0, se utiliza una técnica de recursividad para imprimir el número. La recursividad es cuando una función se llama a sí misma. En este caso, se llama a **ft_write_uns** con el valor de **n** dividido entre 10, lo que significa que se está quitando el último dígito del número.
5. Luego, se imprime el último dígito del número utilizando **ft_write_chr** y se incrementa **size** en 1.
6. Se utiliza un bucle while para contar la cantidad de dígitos del número original y se incrementa **size** en cada iteración.
7. Finalmente, se devuelve el valor de **size**, que indica la cantidad de caracteres impresos.

ft_write_nbr:

1. La función recibe un número entero **n** como parámetro.
2. Se inicializa una variable **len** en 0, que se utilizará para contar la cantidad de caracteres impresos.
3. Si **n** es 0, se imprime el carácter '0' utilizando **ft_write_chr** y se incrementa **len** en 1.
4. Si **n** es -2147483648 (el menor número entero posible), se imprime la cadena "-2147483648" utilizando **ft_write_str** y se devuelve **len**.
5. Si **n** es negativo, se imprime el carácter '-' utilizando **ft_write_chr** y se incrementa **len** en 1. Luego, se convierte **n** en su valor absoluto (positivo).
6. Si **n** es positivo, se llama a la función **ft_write_uns** con el valor de **n** convertido a unsigned int, y se devuelve el valor de **len**, que indica la cantidad de caracteres impresos.

En resumen, estas funciones se encargan de imprimir números enteros y unsigned en la pantalla utilizando recursividad y llamadas a otras funciones para imprimir caracteres y cadenas.



ft_write_ptr.c

Objetivo: La función **ft_write_ptr** se encarga de imprimir la representación en hexadecimal de un puntero (dirección de memoria) en la consola. Recordemos que un puntero es una variable que almacena la dirección de memoria de otra variable. En este caso, el puntero se pasa como parámetro a la función y se imprime su valor en hexadecimal, precedido de "0x" para indicar que es un valor hexadecimal.

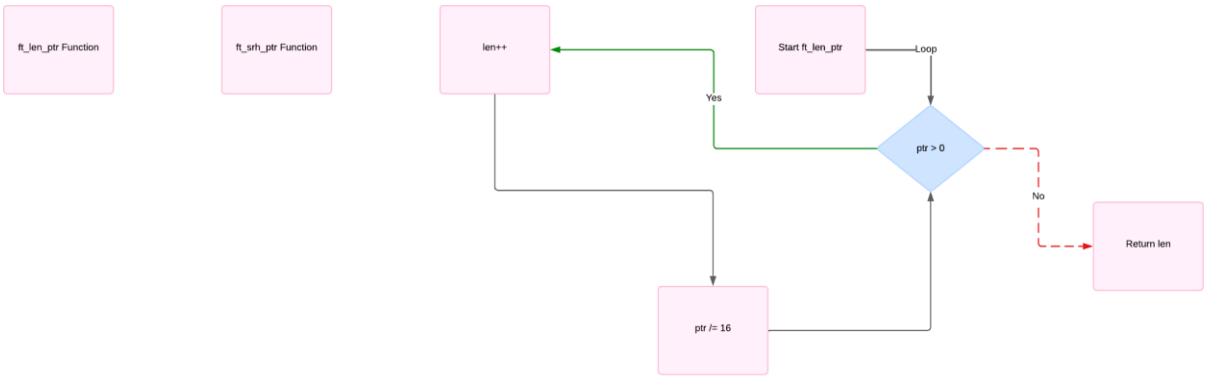
Estructura: La función **ft_write_ptr** se compone de tres partes:

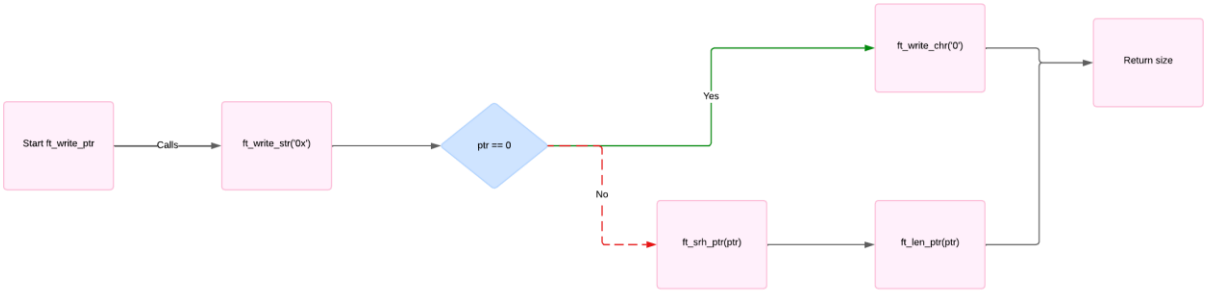
1. La función **ft_write_ptr** en sí misma, que se encarga de imprimir la representación en hexadecimal del puntero.
2. La función **ft_len_ptr**, que calcula la longitud del puntero en hexadecimal.
3. La función **ft_srh_ptr**, que imprime el puntero en hexadecimal de manera recursiva.

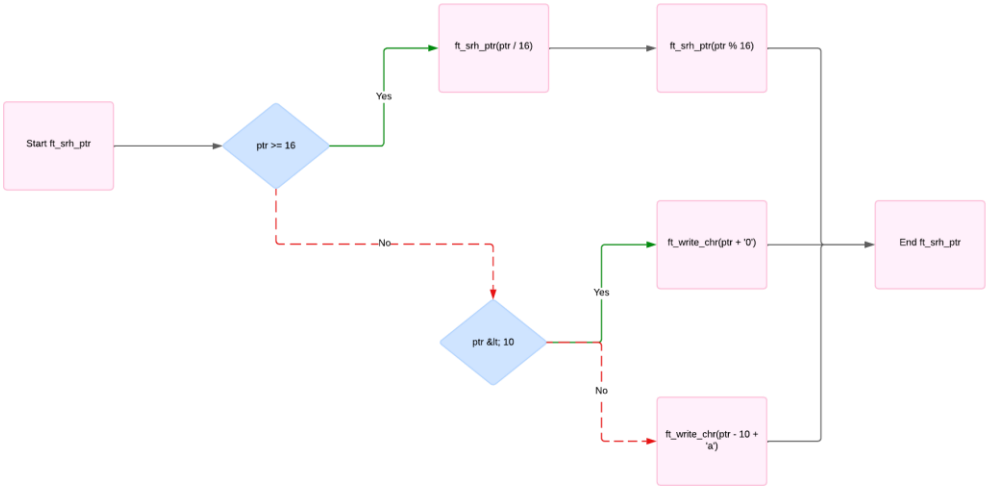
Funcionamiento:

1. La función **ft_write_ptr** recibe un puntero **ptr** como parámetro y devuelve la cantidad de caracteres impresos.
2. Primero, se imprime la cadena "0x" para indicar que el valor que sigue es hexadecimal.
3. Luego, se verifica si el puntero es nulo (0). Si es así, se imprime un solo carácter '0'. Si no es nulo, se llama a la función **ft_srh_ptr** para imprimir el puntero en hexadecimal.
4. La función **ft_len_ptr** se encarga de calcular la longitud del puntero en hexadecimal. Lo hace mediante un bucle que divide el puntero entre 16 hasta que sea 0, y cuenta la cantidad de veces que se realiza la división. Esto se debe a que cada dígito hexadecimal ocupa 4 bits, y al dividir entre 16 se puede determinar la cantidad de dígitos necesarios para representar el puntero.
5. La función **ft_srh_ptr** se encarga de imprimir el puntero en hexadecimal de manera recursiva. Si el puntero es mayor o igual a 16, se llama a sí misma con el valor del puntero dividido entre 16, y luego con el resto de la división entre 16. De esta manera, se imprime cada dígito hexadecimal de manera recursiva. Si el puntero es menor que 10, se imprime como un carácter numérico (0-9). Si es mayor o igual a 10, se imprime como un carácter alfabético (a-f).

En resumen, la función **ft_write_ptr** se encarga de imprimir la representación en hexadecimal de un puntero, precedida de "0x", y devuelve la cantidad de caracteres impresos.


ft_len_ptr Function
ft_srh_ptr Function
len++
Start ft_len_ptr
Loop
Yes
ptr > 0
No
Return len
ptr /= 16


Start ft_write_ptr
Calls
ft_write_str('0x')
ptr == 0
Yes
ft_write_chr('0')
Return size
No
ft_srh_ptr(ptr)
ft_len_ptr(ptr)


Start ft_srh_ptr
ptr >= 16
Yes
ft_srh_ptr(ptr / 16)
ft_srh_ptr(ptr % 16)
No
ptr < 10
Yes
ft_write_chr(ptr + '0')
End ft_srh_ptr
No
ft_write_chr(ptr - 10 + 'a')

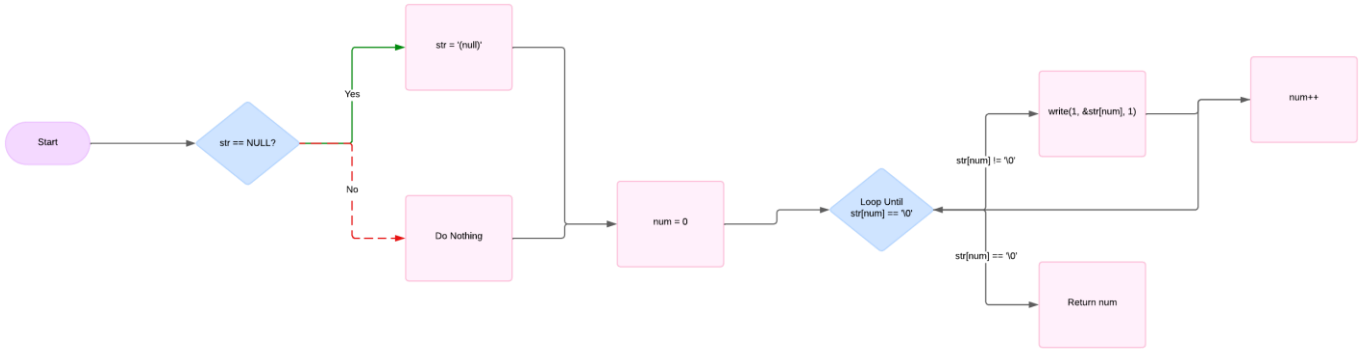
ft_write_str.c

Objetivo: La función **ft_write_str** tiene como objetivo escribir una cadena de caracteres (un string) en la pantalla y devolver la cantidad de caracteres escritos.

Funcionamiento:

- **Declaración de variables:** Se declara una variable **num** de tipo entero (**int**) que se utilizará para contar la cantidad de caracteres escritos.
- **Inicialización de variables:** Se inicializa la variable **num** con el valor 0.
- **Verificación de si la cadena es nula:** Se verifica si la cadena **str** es nula (es decir, si no se ha proporcionado una cadena). Si es así, se asigna la cadena **"(null)"** a **str**. Esto se hace para evitar errores y mostrar un mensaje claro en caso de que no se proporcione una cadena.
- **Bucle para escribir cada carácter de la cadena:** Se utiliza un bucle **while** que se ejecutará mientras el carácter actual de la cadena (**str[num]**) no sea el carácter nulo (**'\0'**). En cada iteración del bucle:
  - Se escribe el carácter actual en la pantalla utilizando la función **write**. El primer parámetro **1** indica que se escribe en la salida estándar (la pantalla). El segundo parámetro **&str[num]** es la dirección del carácter actual, y el tercer parámetro **1** indica que se escribe un solo carácter.
  - Se incrementa la variable **num** en 1 para avanzar a la siguiente posición en la cadena.
- **Devolución del número de caracteres escritos:** Finalmente, se devuelve el valor de **num**, que representa la cantidad de caracteres escritos en la pantalla.

En resumen, esta función escribe una cadena de caracteres en la pantalla y devuelve la cantidad de caracteres escritos. Si no se proporciona una cadena, se muestra el mensaje **"(null)"**.



ft_printf.h

Esta sección de código define una serie de funciones y estructuras en un archivo de encabezado de C (**.h**). El objetivo principal de este módulo es proporcionar una implementación personalizada de la función **printf()**, llamada **ft_printf()**. La función **printf()** es una función de librería de C que permite imprimir formateado texto y valores en la salida estándar.

La estructura **s_flags** contiene diferentes campos que representan diversos formatos de conversión, como **%c**, **%s**, **%p**, **%d**, **%i**, **%u**, **%x**, **%X**, **%%**, y se utilizan para almacenar los argumentos variables de **ft_printf()**.

Aquí hay una descripción breve de las funciones definidas:

- **ft_printf(char const *str, ...)**: Emula el comportamiento de la función **printf()** original. Acepta un formato de cadena y una lista variable de argumentos y devuelve el número de caracteres impresos.
- **ft_strlen(const char *str)**: Calcula la longitud de una cadena pasada como argumento.
- **ft_write_chr(int chr)**: Imprime un solo carácter.
- **ft_write_dns(unsigned int n)**: Imprime un número en formato decimal sin signo.
- **ft_write_hxl(unsigned int n)**: Imprime un número en formato hexadecimal con letras minúsculas.
- **ft_write_hxu(unsigned int n)**: Imprime un número en formato hexadecimal con letras mayúsculas.
- **ft_write_nbr(int n)**: Imprime un número en formato decimal, con signo positivo o negativo.
- **ft_write_ptr(unsigned long long ptr)**: Imprime un puntero en formato hexadecimal.
- **ft_write_str(char *str)**: Imprime una cadena de caracteres.

Como sabemos, este código está diseñado para proporcionar una funcionalidad similar a **printf()**, pero con su propia implementación personalizada. En resumen: al utilizar esta biblioteca, se puede aprovechar la funcionalidad de formateo de **printf()** con las funciones personalizadas definidas en este archivo de encabezado.

Makefile

¡Ojo, con M en MAYÚSCULA!

Objetivo: El objetivo de este código es crear una biblioteca de funciones llamada **libftprintf.a** que contiene implementaciones de funciones para imprimir texto y números en la consola, similar a la función **printf** de la biblioteca estándar de C. La biblioteca se llama **libftprintf** y se compila en un archivo llamado **libftprintf.a**.

Estructura: El código es un archivo de makefile, que es un archivo de configuración para el programa **make**. El archivo de makefile se utiliza para automatizar la compilación y construcción de proyectos de programación.

Este archivo de makefile se divide en varias secciones:

- **Variables**: En esta sección se definen variables que se utilizarán en el archivo de makefile. En este caso, se definen las siguientes variables:
    - **NAME**: El nombre de la biblioteca que se va a crear, que es **libftprintf.a**.

    - **SRC**: Una lista de archivos fuente (**.c**) que se utilizarán para compilar la biblioteca.
    - **OBJS**: Una lista de archivos objeto (**.o**) que se generarán al compilar los archivos fuente.
    - **CC**: El compilador de C que se utilizará para compilar los archivos fuente.
    - **CFLAGS**: Las opciones de compilación que se utilizarán para compilar los archivos fuente.
- **Reglas**: En esta sección se definen las reglas para compilar y construir la biblioteca. Hay varias reglas:
    - **all**: La regla principal que se utiliza para compilar y construir la biblioteca.
    - **$(NAME)**: La regla que se utiliza para crear la biblioteca **libftprintf.a**.
    - **%.o: %.c**: La regla que se utiliza para compilar un archivo fuente (**%.c**) y generar un archivo objeto (**%.o**).
    - **clean**: La regla que se utiliza para eliminar los archivos objeto y la biblioteca.
    - **fclean**: La regla que se utiliza para eliminar todos los archivos generados, incluyendo la biblioteca.
    - **re**: La regla que se utiliza para reconstruir la biblioteca desde cero.
- **Comandos**: En esta sección se definen los comandos que se utilizarán para compilar y construir la biblioteca. Por ejemplo, el comando **$(AR) -csr $@ $?** se utiliza para crear la biblioteca **libftprintf.a** a partir de los archivos objeto.

Funcionamiento: Aquí hay un resumen del funcionamiento del archivo de makefile:

1. Cuando se ejecuta el comando **make**, el programa **make** busca el archivo de makefile y lee sus instrucciones.
2. La regla **all** se utiliza para compilar y construir la biblioteca. Esta regla depende de la regla **$(NAME)**, que a su vez depende de los archivos objeto (**$(OBJS)**).
3. Para compilar los archivos fuente, se utiliza la regla **%.o: %.c**. Esta regla utiliza el compilador **CC** con las opciones **CFLAGS** para compilar cada archivo fuente y generar un archivo objeto correspondiente.
4. Una vez que se han compilado todos los archivos fuente, se utiliza el comando **$(AR) -csr $@ $?** para crear la biblioteca **libftprintf.a** a partir de los archivos objeto.
5. Si se ejecuta el comando **make clean**, se eliminan todos los archivos objeto y la biblioteca.
6. Si se ejecuta el comando **make fclean**, se eliminan todos los archivos generados, incluyendo la biblioteca.
7. Si se ejecuta el comando **make re**, se reconstruye la biblioteca desde cero, eliminando todos los archivos generados y volviendo a compilar todos los archivos fuente.

En resumen, este archivo de makefile se utiliza para automatizar la compilación y construcción de la biblioteca **libftprintf.a**, que contiene implementaciones de funciones para imprimir texto y números en la consola.