siz xº ÍNDICE

APRENDIZAJES DE LA VIDA

Aprendizajes de la vida

- **write y printf**, cuando tenemos una cadena de carácteres con algún '\0' en alguna posición, solo nos imprimirá todo el write, con el printf no, véase ejercicio BZERO.
- **gcc -Wall -Wextra -Werror** -g3 -fsanitize = address ft_memset.c main.c
  Flag para resolver el segmentation fault. No es exacto, es más bien orientativo.
- **Castear**: se utilizan para asegurarse que un dato es de un tipo en concreto. Si es necesario, se convertirá al tipo de dato pedido, pero no sirve en todos los casos, ya que no es un sistema de conversión como tal.

  Ejemplo: Asignarle a una variable `void *` el valor de '0'. No podemos hacer `s[i] = 0` directamente, ya que la variable `void` no puede tomar ningún valor. Para ello hace falta castear y pasarlo a `char *`

Castear con variable auxiliar	Castear directamente

`void *s;`
`char *str;`
 	
`str = s;`
`while (i < n)`
`{`
`  str[i] = 0;`
`  i++;`
`}`
	`void *s;`
	`-`
	
	`-`
	`while (i < n)`
	`{`
	`  ((char *)s)[i] = 0;`
	`  i++;`
	`}`

Si no casteamos no podemos asignarle a una variable `void *` el valor de 0. Para ello me creo una variable auxiliar `char *` y la asigno a la misma dirección de memoria que `void *`	Casteando directamente se puede ahorrar una variable intermedia (str). "Forzamos" a que `void *` sea `char *`, y de esta manera pueda coger valores
No se puede hacer (siendo `s` un `void *`) `s{i] = 0;`	Sí se puede hacer (siendo `s` un `void *`) `((char *)s){i] = 0;`

Relacionado con el tipo de variables:

- **const**: Cuando vemos **const** en una variable, nos indica que es un valor constante, que no se puede modificar. Por lo que para poder operar con ella, tendremos que castearla.
- **unsigned char** (0 , 255)
- **char** (-128 a 127)
- **size_t:** es un `unsigned int long`. Se necesita llamar a la librería `<stdio.h>` para usarlo. El tipo `size_t` es el tipo entero sin signo que es el resultado del operador `sizeof` (y el operador `offsetof`), por lo que se garantiza lo suficientemente grande como para contener el tamaño del objeto más grande que su sistema puede manejar (por ejemplo, una matriz estática de 8 Gb).

  El tipo `size_t` puede ser mayor, igual o menor que un `unsigned int` , y su compilador puede hacer suposiciones sobre él para optimizarlo.

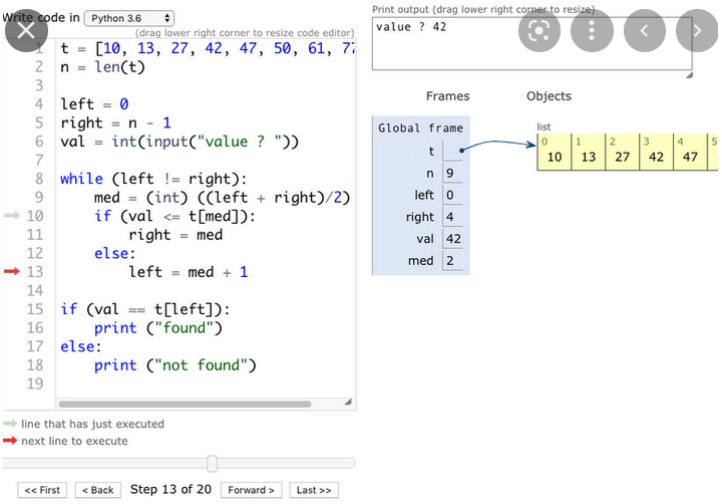
Definiendo el tipo de los subíndices como `size_t` te ***garantiza*** que podrás acceder todos los elementos del arreglo, en todas las plataformas habidas y por haber, si el compilador sigue el estándar ANSI C.

En realidad es solo un alias definido mediante “typedef” que a la hora de compilar será limpiamente sustituido por el compilador por short, int o long, según haga falta - y tu programa compilará igual en todos lados.

- **Restrict**:
    a. La palabra clave `restrict` se usa principalmente en declaraciones de punteros como un calificador de tipo para punteros.
    b. No añade ninguna funcionalidad nueva. Es solo una forma para que el programador informe sobre una optimización que puede hacer el compilador.
    c. Cuando usamos `restrict` con un puntero `ptr`, le dice al compilador que ptr es la única forma de acceder al objeto señalado por él. En otras palabras, no hay otro puntero que apunte al mismo objeto, es decir, la palabra clave `restrict` especifica que un argumento de puntero en particular no es un alias de ningún otro y el compilador no necesita agregar ninguna verificación adicional.
    d. Si un programador usa la palabra clave `restrict` y viola la condición anterior, el resultado es un comportamiento indefinido.
    e. `restrict` no es compatible con C++. Es una palabra clave solo de C.

Web para ver qué ocurre paso a paso

Programa para visualizar las variables que se utilizan y cómo se van rellenando en la memoria. Además se puede ejecutar paso a paso. Muy útil para detectar errores:
https://pythontutor.com/c.html#mode=display



Crear nuestro portfolio: github benefits to create a website

- Pasos: (Learning HTML/CSS to create your web dev portfolio): https://github.com/tanaypratap/io.metastartup.portfolio

- El dominio: https://www.godaddy.com/es-es/offers/domain?isc=esdomEUR3&countryview=1&currencyType=EUR&cdtl=c_14211752584.g_125091993265.k_kwd-10085531.a_562922347897.d_c.ctv_g&bnb=nb&gclid=CjwKCAjw6dmSBhBkEiwA_W-EoPRTH7DGXw16MCSu8EXSyFaHCf6GD1G8K4A36YXiMYYSqu-Z9ntm3RoCZCgQAvD_BwE

Entrega de bonus

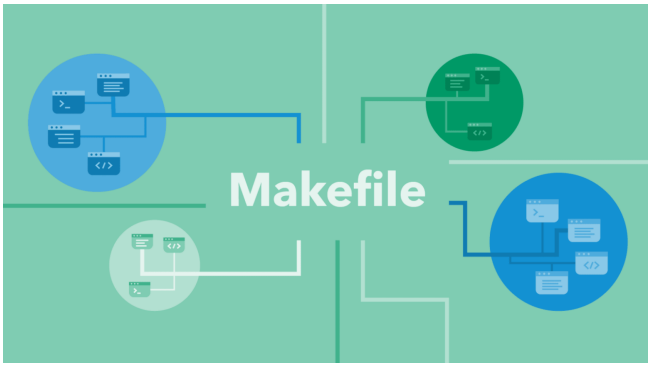
A la hora de entregar los bonus,

- deben **llamarse con el nombre de la función** y
- terminados en **_bonus**.

Únicamente los .c, no las funciones que usamos. Todos los bonus se nombran así y sino pueden ser suspendidos.

Ejemplo: ft_lstclear_bonus.c

MAKEFILE Y LIBFT



2. Makefile0

Para ver las convenciones de make existe este pequeño tutorial:

http://www.gnu.org/prep/standards/html_node/Makefile-Conventions.html#Makefile-Conventions

```
:WQ
NAME = libft.a
SRCS = ft_isalpha.c ft_isdigit.c ft_isalnum.c ft_isascii.c \
ft_isprint.c ft_memset.c ft_bzero.c ft_memcpy.c ft_memmove.c \
ft_strlen.c
OBJECTS = $(SRCS:.c=.o)

CFLAGS = -Wall -Wextra -Werror

all: $(NAME)

$(NAME): $(OBJECTS)
	ar rc $(NAME) $(OBJECTS)

$(OBJECTS): $(SRCS)
	gcc $(CFLAGS) -c $(SRCS)

clean:
	@rm -f $(OBJECTS)

fclean: clean
	@rm -f $(NAME)

re: fclean all

.PHONY: all clean fclean ren.c ft_strlcpy.c ft_strlcat.c ft_tolower.c ft_toupper.c \
```

2.1. Vídeo Youtube

GNU Make – 1. ¿Por qué usar Make?

2.2. Windows

Para poder hacer los MAKEs en Windows, parece que hay que instalar un programa.

- https://parzibyte.me/blog/2020/12/30/instalar-make-windows/
- https://www.iteramos.com/pregunta/22664/como-ejecutar-un-makefile-en-windows

Para pasar los Tester

Cuando instalas el git (original) en el pc, te instala una terminal llamada GIT Bash, la cual permite ejecutar los testers que desde la terminal del Visual daban error. Sigo probando porque el make me daba error (problema de mis funciones), cuando lo soluciones pongo más info.

2.3. Explicación de cada punto

	Explicación
NAME	Define cómo se va a llamar nuestro programa
SRCS	El origen de los archivos que vamos a usar. Se pueden poner en varias líneas, pero para ello es necesario poner al final de la línea, antes de return, una contrabarra.
OBJECTS	A los archivos terminados en *.c, les añade una “copia” que tiene la extensión *.o
CFLAGS	Indica las Flags que queremos usar.
all	Este comando es el que se ejecuta por defecto, es decir, si ponemos “make” en la terminal, se ejecuta el comando “all”. Es equivalente a poner “make all” en la terminal.
clean	Borra los archivos que se han creado en **OBJECTS**. Es decir, borra los archivos terminados en *.o
fclean	Además de borrar los archivos de **clean**, borra el .out o archivo donde se guarde el programa compilado.
re	Usa el **fclean** y además … (todavía no lo sé)
.PHONY	

2.4. NEW MAKE

Este Make es válido para usar más adelante

```
NAME = libft.a
       °

SRCS = (lista de funciones)
OBJECTS = $(SRCS:.c=.o)

CC = gcc

CFLAGS = -Wall -Wextra -Werror
```

```
all: $(NAME)

$(OBJECTS): %.o: %.c
	$(CC) $(CFLAGS) -c $< -o $@

$(NAME): $(OBJECTS)
	ar rcs $@ $^

clean:
	@rm -f $(OBJECTS)

fclean: clean
	@rm -f $(NAME)

re: fclean all

.PHONY: all clean fclean re
```

3. LIBFT (crear una librería para poder usarla. + makefile)

3.1. Explicación de cada punto

#ifndef LIBFT_H	Le estamos preguntando si está definida esta librería, si no le pasamos a indicar que la defina y acabamos cerrando con un **#endif**.
# include	Como todo lo que metemos está dentro de un if, el resto de librerías deben de tener un *espacio*. *Ejemplo:* (**#** *espacio* **include** **<***librería***.h>**).
Prototipados	Debemos incluir todos los prototipados de nuestras funciones.
#endif	Se termina la librería con esta línea.

RECOMENDACIÓN: Eliminar las librerías que no usen.

3.2. Programa

```
#ifndef LIBFT_H
# define LIBFT_H

# include <stdio.h>
# include <unistd.h>
# include <string.h>
# include <stdlib.h>
# include <ctype.h>

int     ft_isalpha(int c);
int     ft_isdigit(int c);
int     ft_isalnum(int c);
int     ft_isascii(int c);
int     ft_isprint(int c);
int     ft_strlen(char *str);
void    *ft_memset(void *str, int c, size_t n);
void    ft_bzero(void *s, size_t n);
void    *ft_memcpy(void *restrict dst, const void *restrict src, size_t n);
void    *ft_memmove(void *dst, const void *src, size_t len);
size_t  ft_strlcpy(char *dst, const char *src, size_t dstsize);
size_t  ft_strlcat(char *dst, const char *src, size_t dstsize);
int     ft_toupper(int c);
int     ft_tolower(int c);

#endif
```

3.3 LIBFT TESTERS

Existen diferentes tester para comprobar si las funciones son correctas. Son una especie de Moulinette.

3.3.1. WAR MACHINE

- **link web**: https://github.com/y3ll0w42/libft-war-machine

- **Pasos**:

- **PASO 1:** Bajamos o clonamos la carpeta (también está en el drive) y la colocamos dentro de la carpeta donde están los ejercicios.
- **PASO 2:** SIEMPRE: deberemos generar el make (escribir en la terminal make en la carpeta de nuestro repositorio), si también queremos que genere los de bonus, debemos escribir make bonus.
- **PASO 3:** Para ejecutar el programa, entramos en la carpeta de libft machine, por la terminal y escribimos:

```
~ % bash grademe.sh
```

- **PASO 4:** Entonces nos dice que si queremos ver si hay actualizaciones, le dais lo que queráis y seguido hace las pruebas y da los resultados que sean.
- **PASO 5:** Además crea un archivo deepthought, para ver los fallos.

- **Comando shell**:

Para chequear todas las funciones hechas hasta el momento:

```
~ % bash libft-war-machine-master/grademe.sh
```

Para chequear solamente una función:

```
~ % bash libft-war-machine-master/grademe.sh nombre_función

Ejemplo: ~ % bash libft-war-machine-master/grademe.sh ft_isalpha
```

- Ojo: Al pasar el programa desaparecen los archivos creados al hacer make, y a veces, al volverlo a ejecutar puede dar error, si es el caso, hay que volver a hacer make. Y listo.

?????? => ¿Os corrige los bonus?

3.3.2. LIBFT TESTER

- **link web**: https://github.com/Tripouille/libftTester

- **Qué hace:** Este programa funciona como la Moulinette, es decir, cuando encuentra una función que da mal, no sigue corrigiendo.

- **Pasos**:

- **PASO 1:** Bajamos o clonamos la carpeta (también está en el drive) y la colocamos dentro de la carpeta donde están los ejercicios.
- **PASO 2:** No tenemos que generar el MAKE, lo hace directamente
- **PASO 3:** Para ejecutar el programa, entramos en la carpeta de libftTester, por la terminal y escribimos:

```
~ % make m
```

- **Comando shell**:

make m = testeamos las funciones obligatorias

make b = testeamos las funciones bonus

make a = testeamos las funciones obligatorias y bonus

make [function name] = testeamos una función concreta: make calloc

Para un entorno de linux es similar:

make dockerm = testeamos las funciones obligatorias en linux

make dockerb = testeamos las funciones bonus en linux

make dockera = testeamos las funciones obligatorias y bonus en linux

make docker [function name] = testeamos una función concreta en linux: make calloc

Nota: para borrar los archivos .o, si volvemos a ejecutar la librería War Machine, conseguimos que se borren al terminar de ejecutar el test.

- Ojo: Si da error al ejecutar (nos sale letras azules), deberemos meternos en el archivo Makefile de la carpeta libftTester y en la línea:

```
LIBFT_PATH      = $(PARENT_DIR)
```

cambiar la dirección:

```
LIBFT_PATH      = ../
```

3.3.3. LIBFT-UNIT-TEST

- **link web**: https://github.com/alelievr/libft-unit-test

- **Qué hace:** libft-unit-test es un conjunto completo de pruebas para el proyecto libft de 42, que le permite probar su biblioteca, realizar un seguimiento de su progreso y evaluar comparativamente su biblioteca (con la biblioteca del sistema o con otra biblioteca)

3.3.4. LIBTEST

- **link web**: https://github.com/jtoty/Libftest/tree/master/tests

3.3.5. Francinette

Se ejecutan los tests existentes de arriba +1 extra con el comando $paco

(hay instrucciones y más opciones en el enlace debajo!)

- **link web**: https://github.com/xicodomingues/francinette#install

The francinette folder will be under your `$HOME` directory (`/User/<you_username>/`)

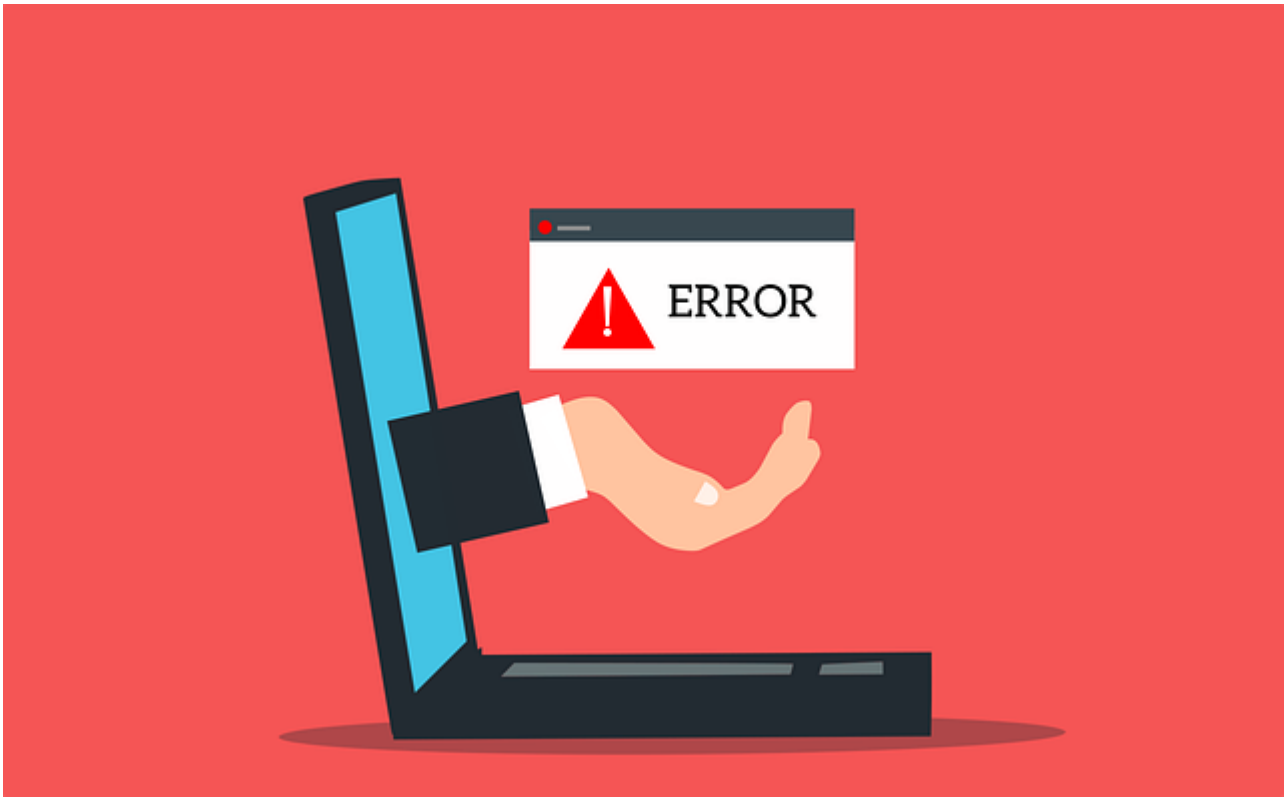
Una vez tenemos la carpeta “francinette” en la carpeta que indica arriba, si estamos en Mac ejecutamos el siguiente comando para instalar el programa.

```
bash -c "$(curl -fsSL
https://raw.github.com/xicodomingues/francinette/master/bin/install.sh)"
```

Ya instalado, poniendo el comando “francinette” o “paco” en la consola ya nos ejecuta el programa

En el canal #general de 42Urduliz hay un hilo también de RECURSOS LIBFT !

ERRORES



4. Errores

(Está cogido con pinzas), pero estaría bien intentar entender los errores. Esto es lo que he sacado de internet.

4.1 zsh: illegal hardware instruction

illegal hardware instruction desde zhs.

Surgen porque no todos los valores de memoria posibles corresponden a una instrucción válida. Por ejemplo, si los códigos de operación de las instrucciones fueran de un byte, habría 256 instrucciones distintas posibles. Un procesador típico en realidad no implementa 256 códigos de operación de instrucciones diferentes, por lo que algo tenía que suceder si el procesador intenta ejecutar un código no válido. A nivel de hardware, la CPU recibe una interrupción. A nivel del sistema operativo, esta interrupción (ahora llamada trampa) detiene el programa y el O/S imprime un breve mensaje.

¿Qué tipo de errores dan lugar a una trampa de instrucción ilegal? Llamar a una biblioteca .net después de descargarla (y reutilizar el almacenamiento), llamar a una biblioteca dinámica a través de un puntero no válido o después de descargar la biblioteca (y reutilizar el almacenamiento), sobrescribir la región de la pila que contiene un dirección de retorno de función, sobrescribir un puntero de función o un puntero de función miembro, en sistemas pequeños sin protección de memoria, sobrescribir la memoria de otro proceso. Los problemas de hardware que dañan la memoria también pueden causar trampas de instrucciones ilegales.

4.2. Bus error

Los errores de bus son raros hoy en día en x86 y se producen cuando el procesador no puede ni siquiera intentar el acceso a la memoria solicitada, normalmente utilizando una instrucción del procesador con una dirección que no satisface sus requisitos de alineación.

Los fallos de segmentación se producen al acceder a la memoria que no pertenece a su proceso. Son muy comunes y suelen ser el resultado de:

- utilizando un puntero a algo que fue reasignado.
- utilizando un puntero no inicializado y por tanto falso.
- utilizando un puntero nulo.
- desbordando un buffer.

PS: Para ser más precisos, no es la manipulación del puntero en sí lo que causará problemas. Es el acceso a la memoria a la que apunta (desreferenciación).

4.3. Segmentation fault

Este error ocurre cuando estamos intentando acceder a una parte de la memoria que corresponde a otra aplicación.

4.4. zsh: permission denied

FUNCIONES

- [x] ~~ISALPHA~~
- [x] ~~ISDIGIT~~
- [x] ~~ISALNUM~~
- [x] ~~ISASCII~~
- [x] ~~ISPRINT~~
- [x] ~~STRLEN~~
- [x] ~~MEMSET~~
- [x] ~~BZERO~~
- [x] ~~MEMCPY~~
- [x] ~~MEMMOVE~~
- [x] ~~STRLCPY~~
- [x] ~~STRLCAT~~
- [x] ~~TOUPPER~~
- [x] ~~TOLOWER~~
- [x] ~~STRCHR~~
- [x] ~~STRRCHR~~
- [x] ~~STRNCMP~~
- [ ] MEMCHR
- [x] ~~MEMCMP~~
- [x] ~~STRNSTR~~
- [x] ~~ATOI~~
- [x] ~~CALLOC (+MALLOC)~~
- [x] ~~STRDUP (+MALLOC)~~

Funciones similares

Comparan si un carácter (int) es	Funciones con **string** char *	Funciones con **memoria** void * (Hay que castear)
ISALPHA: letra ISDIGIT: dígito ISALNUM: letra | | dígito ISASCII: está en ASCII ISPRINT: es printeable	STRLEN (return longitud cadena) STRLCAT (concatena cadenas) STRNSTR (busca subcadena) STRDUP (+MALLOC)	MEMSET (pone un carácter en puntero) BZERO (pone 0 en puntero)
	STRLCPY (copia scr a dst l -1 caracteres)	MEMCPY (copia scr a dst) MEMMOVE (mueve scr a dst, mejor que MEMCPY)
	STRCHR (busca caracter en cadena) STRRCHR (busca caracter en cadena por detrás)	MEMCHR (busca carácter en memoria)
	STRNCMP (compara str1 y str2)	MEMCMP (compara mem1 y mem2)

5. ISALPHA

Qué hace: Comprueba si el valor recibido es un carácter **alfabético** (mayúscula o minúscula).

5.1. Declaración

```
int isalpha(int c)
```

5.2. Parámetros

Valor a configurar	Tipo de variable	Valor a introducir: 2 opciones	
c	int	c = ‘a’	c = 97 su ascii

5.3. Retorno

Devuelve	Tipo de variable	**TRUE**: cumple la condición	**FALSE**: no cumple la condición
un valor (!0): 1 o 0	int	return (1) : Devuelve 1	return (0) : Devuelve 0

5.4. Programa

```c
#include <stdio.h>

int ft_isalpha(int c);

int main(void)
{
   int c;

   c = 'a';
   printf("%d", ft_isalpha(c));
}

int ft_isalpha(int c)
{
   if ((c < 'a' || c > 'z') && (c < 'A' || c > 'Z'))
   {
       return (0);
   }
   return (1);
}
```

6. ISDIGIT

Qué hace: isaln.

6.1. Declaración

```
int isdigit(int c)
```

6.2. Parámetros

Valor a configurar	Tipo de variable	Valor a introducir: 2 opciones	
c	int	c = ‘3’	c = 51 su ascii

6.3. Retorno

Devuelve	Tipo de variable	**TRUE**: cumple la condición	**FALSE**: no cumple la condición
un valor (!0): 1 o 0	int	return (1) : Devuelve 1	return (0) : Devuelve 0

6.4. Programa

```c
#include <stdio.h>

int ft_isdigit(int c);

int main(void)
{
   int c;

   c = 's';
   printf("%d", ft_isdigit(c));
}

int ft_isdigit(int c)
{
   if (c < '0' || c > '9')
   {
       return (0);
   }
   return (1);
}
```

7. ISALNUM

Qué hace: Comprueba si el valor introducido es un **alfanumérico**. Se pueden hacer con las funciones ISALPHA || ISDIGIT, ya que si cumple una de ellas nos devuelve (!0)

7.1. Declaración

```
int isalnum(int c)
```

7.2. Parámetros

Valor a configurar	Tipo de variable	Valor a introducir: 2 opciones	
c	int	c = '3'	c = 51 su ascii

7.3. Retorno

Devuelve	Tipo de variable	**TRUE**: cumple la condición	**FALSE**: no cumple la condición
un valor (!0): 1 o 0	int	return (1) : Devuelve 1	return (0) : Devuelve 0

7.4. Programa

```c
#include <stdio.h>

int ft_isalnum(int c);

int main(void)
{
   int c;

   c = '!';
   printf("%d", ft_isalnum(c));
}

int ft_isalnum(int c)
{
   if ((c < 'a' || c > 'z') && (c < 'A' || c > 'Z')
       && (c < '0' || c > '9'))
   {
       return (0);
   }
   return (1);
}
```

8. ISASCII

Qué hace: Comprueba si el valor indicado se encuentra en la **tabla ASCII**. Entre el 0 y 127.

8.1. Declaración

```
int isascii(int c)
```

8.2. Parámetros

Valor a configurar	Tipo de variable	Valor a introducir: 2 opciones	
c	int	c = '3'	c = 51 su ascii

8.3. Retorno

Devuelve	Tipo de variable	**TRUE**: cumple la condición	**FALSE**: no cumple la condición
un valor (!0): 1 o 0	int	return (1) : Devuelve 1	return (0) : Devuelve 0

8.4. Programa

```c
#include <stdio.h>

int ft_isascii(int c);

int main(void)
{
   int c;

   c = 129;
   printf("%d", ft_isascii(c));
}

int ft_isascii(int c)
{
   if (c < 0 || c > 127)
   {
       return (0);
   }
   return (1);
}
```

9. ISPRINT

Qué hace: Mediante una variable int recogemos un valor y tenemos que resolver si es printable. Los valores printables son >= 32 && < 127 (el 127 es el SUPRIMIR y no es printable)

9.1. Declaración

```
int isprint(int c)
```

9.2. Parámetros

Valor a configurar	Tipo de variable	Valor a introducir: 2 opciones	
c	int	c = '3'	c = 51 su ascii

9.3. Retorno

Devuelve	Tipo de variable	**TRUE**: cumple la condición	**FALSE**: no cumple la condición
un valor (!0): 1 o 0	int	return (1) : Devuelve 1	return (0) : Devuelve 0

9.4. Programa

```c
#include <stdio.h>

int ft_isprint(int c);

int main(void)
{
   int c;

   c = 126;
   printf("Función original: %d\n", isprint(c));
   printf("Función propia:   %d\n", ft_isprint(c));
}

int ft_isprint(int c)
{
   if (c >= 32 && c < 127)
   {
       return (1);
   }
   return (0);
}
```

10. STRLEN

Qué hace: Nos pide un string en una variable char. Mediante un while sacamos la longitud del string que devolvemos mediante un int.

10.1. Declaración

```
size_t strlen(const char *str)
```

10.2. Parámetros

Valor a configurar	Tipo de variable	Explicación	Tenemos que
str	char *	La variable recoge el string	Recorrer y obtener su valor

10.3. Retorno

Devuelve	Tipo de variable
un valor = es la longitud de la cadena recibida	size_t

10.4. Programa

```c
#include <stdio.h>

size_t ft_strlen(const char *str);

int main(void)
{
  char    *str =  "hola";
  printf("Función original: %lu\n", strlen(str));
  printf("Función propia:   %lu\n", ft_strlen(str));
}

size_t ft_strlen(const char *str)
{
   size_t count;

   count = 0;
   while (str[count])
   {
       count ++;
   }
   return (count);
}
```

11. MEMSET

Qué hace: Esta función copia el carácter c (un carácter sin signo) en los primeros n caracteres de la cadena a la que apunta el argumento str.

¿?Ojo: Tenemos que **castear** el parámetro str que es void, por char, ya que no podemos operar con un void. Aunque después devolvamos el void, ya que el valor está modificado mediante el char. Las dos variables están apuntando al mismo valor.

m11.1. Declaración

```
void *memset(void *str, int c, size_t n)
```

11.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
str	char *	Es un puntero al bloque de memoria para llenar.
c	int	Es el valor a configurar. El valor se pasa como un int, pero la función llena el bloque de memoria utilizando la conversión de caracteres sin firmar de este valor.
n	size_t	Es el número de bytes que se establecerá en el valor.

11.3. Retorno

Devuelve	Tipo de variable
devuelve un puntero al área de memoria str	int

11.4. Programa

```c
#include <stdio.h>

void    *ft_memset(void *str, int c, size_t n)

int main(void)
{
   char str[50] = "Esta es una funcion muy bonica";

   printf("String original:             %s\n", str);
   memset(str + 5,'$', 4);
   printf("Función original:    %s\n", str);

   char str1[50] = "Esta es una funcion muy bonica";

   ft_memset(str1 + 5,'$', 4);
   printf("Función propia: %s\n", str1);
   return(0);
}

void    *ft_memset(void *str, int c, size_t n)
{
```

```
    size_t  i;
    char    *dest;

    dest = str;
    i = 0;
    while (i < n)
    {
        dest[i] = c;
        i++;
    }
    return (str);
}
```

12. BZERO

Qué hace: Esta función sobreescribe 0 en los primeros n caracteres de la cadena a la que apunta el argumento s.

Ojo: Tenemos que **castear** el parámetro s que es void, por char, ya que no podemos operar con un void. Aunque después devolvamos el void, ya que el valor está modificado mediante el char. Las dos variables están apuntando al mismo valor.

Ojo: Para poder ejecutar esta función debemos usar el write, ya que el printf cuando encuentra un nulo, no sigue leyendo.

12.1. Declaración

```
void  bzero(void *s, size_t n)
```

12.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
s	char *	Este es un puntero que apunta al bloque de memoria en el que tenemos el string
n	size_t	Es el número de bytes que se establecerá en el valor 0

12.3. Retorno

Devuelve	Tipo de variable
devuelve un puntero al área de memoria s	void *

12.4 Programa

```c
#include <string.h>
# include <unistd.h>
# include <stdlib.h>

void    ft_bzero(void *s, size_t n);

int main(void)
{
   char str[50] = "Esta es la funcion bzero";

   printf("String original:  %s\n", str);
   bzero(str, 2);
   write(1, "Función original: ", 20);
   write(1, &str, 30);
   write(1, "\n", 1);
```

```
    char str1[50] = "Esta es la funcion bzero";

    ft_bzero(str1, 2);
    write(1, "Función propia:   ", 20);
    write(1, &str1, 30);
    write(1, "\n", 1);
    return(0);
}

void    ft_bzero(void *s, size_t n)
{
    unsigned char   *str;
    size_t          i;

    i = 0;
    str = s;
    while (i < n)
    {
        str[i] = 0;
        i++;
    }
}
```

13. MEMCPY

Qué hace: Esta función sobreescribe n caracteres de la cadena src a la cadena dst.

Ojo: Tenemos que castear el parámetro src y dst, que es void, por char.

Además, aunque se puede obviar, no es necesario poner el restrict. En este caso nos vemos obligados a devolver el valor desde el void, ya que el restrict nos obliga a ello.

Ojo: En este caso casteamos las variables directamente al operar, lo que nos ahorra crear dos variables para almacenar el valor.

13.1. Declaración

```
void  memcpy(void *restrict dst, const void *restrict src, size_t n)
```

13.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
dst	void *restrict	Este es un puntero que apunta al bloque de memoria en el que tenemos que copiar los n caracteres del puntero src.
src	const void *restrict	Este es un puntero que apunta al bloque de memoria en el que tenemos el string
n	size_t	Es el número de bytes que se establecerá en el valor 0

13.3. Retorno

Devuelve	Tipo de variable
devuelve un puntero al área de memoria dst	void *

13.4 Programa

```c
#include <stdio.h>
#include <string.h>

void    *ft_memcpy(void *restrict dst, const void *restrict src, size_t n);

int main(void)
{
   char src[50] = "mahmudulhasanjony";
   char dest[50];
   char dest1[50];

   memcpy(dest, src, 25);
   printf("Funcion original: %s\n", dest);
   ft_memcpy(dest1, src, 25);
   printf("Funcion propia:   %s\n", dest1);
   return (0);
}
```

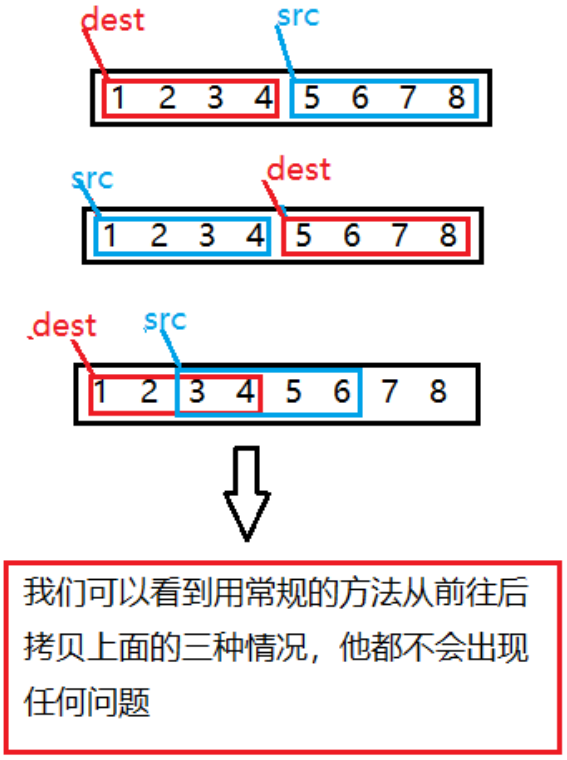
```c
void    *ft_memcpy(void *restrict dst, const void *restrict src, size_t n)
{
   size_t  i;

   i = 0;
   while (i < n)
   {
       ((unsigned char *)dst)[i] = ((const char *)src)[i];
       i++;
   }
   return (dst);
}
```

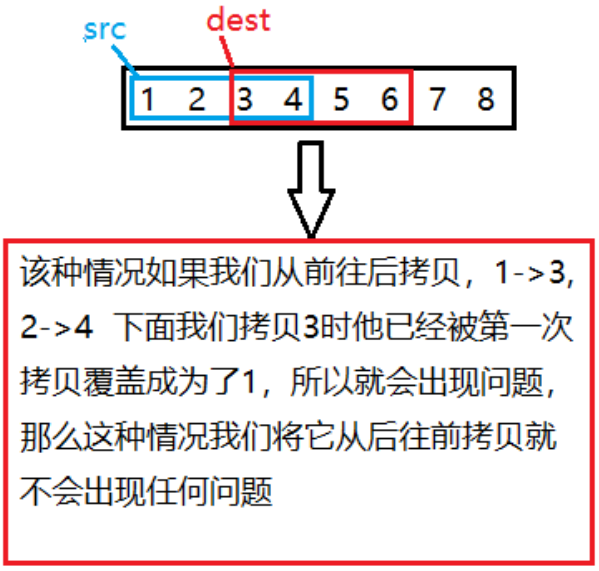
size_t -> <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, <wchar.h>

14. MEMMOVE

Qué hace: Esta función copia n caracteres de str2 a str1, pero para bloques de memoria superpuestos(overlap), memmove() es un enfoque más seguro que memcpy().

No es un caso común, pero puede caber la posibilidad de que la dirección de memoria de ambas cadenas se sobrepongan.





En este caso, no tendríamos problema, podemos sobreescribir ya que al estar el dest antes en la memoria, no sobreescribimos los valores.	En este caso, sí tendríamos problema, ya que al sobreescribir los índice 0 y 1 de la string original a los índices 0 y 1, del string de destino, estamos sobreescribiendo los datos originales de los índices 2 y 3 de la original y los perdemos. Para solucionar esto, tenemos que sobreescribir desde el final del string hacia delante.

14.1. Declaración

```
void  memmove(void *dst, const void *src, size_t len)
```

14.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
dst	void *	Este es un puntero que apunta al bloque de memoria en el que tenemos que copiar los n caracteres del puntero src.
src	const void *	Este es un puntero que apunta al bloque de memoria en el que

		tenemos el string
len	size_t	Es el número de bytes que se copiarán

14.3. Retorno

Devuelve	Tipo de variable
devuelve un puntero al área de memoria dst	void *

14.4 Programa

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char str1[50];
   char str2[50] = "Esta es la funcion memmove";
   char *ret;

   //ret = memmove(str1, str2, 11);
   ret = ft_memmove(str1, str2, 11);
   printf("%s", ret);
   return (0);
}

void    *ft_memmove(void *dst, const void *src, size_t len)
{
   unsigned char       *dest;
   unsigned const char *origen;
   size_t              i;

   i = 0;
   dest = dst;
   origen = src;
   if (dest > origen)
   {
       while (len--)
       {
           dest[len] = origen[len];
       }
   }
   else if (dest < origen)
   {
       while (i < len)
       {
           ((unsigned char *)dst)[i] = ((const char *)src)[i];
           i++;
       }
   }
   return (dst);
}
```

15. MEMMOVE vs MEMCPY

El memmove siempre es correcto y se usa más que memcpy.

El memcpy es más rápido, pero no siempre es correcto porque puede sobrescribir ubicaciones de memoria si el origen y el destino se superponen.

15.1. Programa MEMMOVE con MEMCPY

```c
#include "libft.h"

void    *ft_memmove(void *dst, const void *src, size_t len)
{
   char    *dest;
   char    *origen;

   dest = (char *)dst;
   origen = (char *)src;
   if (dest > origen)
   {
       while (len--)
       {
           dest[len] = origen[len];
       }
   }
   else if (dest < origen)
   {
       ft_memcpy(dst, src, len);
   }
   return (dest);
}
```

16. STR STRLCPY

→Esta función toma el tamaño de dst como parámetro y no escribirá más de esa cantidad de bytes para evitar el desbordamiento del búfer (suponiendo que el tamaño sea correcto). Además, siempre escribe un solo byte '\0' en el destino (si el tamaño no es cero). Se garantiza que la cadena resultante terminará en '\0' incluso si se trunca. Además, no pierde el tiempo escribiendo múltiples bytes '\0' para llenar el resto del búfer.

16.1. Declaración

```
size_t  strlcpy(char *dst, const char *src, size_t dstsize)
```

16.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
dst	char*	Este es un puntero que apunta al bloque de memoria en el que tenemos que copiar los n caracteres del puntero src.
src	const char*	Este es un puntero que apunta al bloque de memoria en el que tenemos el string
dstsize	size_t	Es el número de bytes que copiaremos del src a dst, teniendo en cuenta que **el último carácter siempre es '\0'**, es decir, si dstsize = 1; nos copiará sólo el '\0'.

16.3. Retorno

Devuelve	Tipo de variable
devuelve la longitud de la cadena de origen (src)	size_t

Esta longitud se puede comparar con el tamaño del búfer de destino para verificar si se truncó y para evitar el truncamiento.

Además, la función modifica dst, copiando dstsize - 1 caracteres de scr en él.

16.4. Programa

```c
size_t  ft_strlcpy(char *dst, const char *src, size_t dstsize)
{
   size_t  i;
   size_t  j;

   i = 0;
   j = 0;
   if (src == NULL || dst == NULL)
   {
       return (0);
   }
   while (src[i] != '\0')
   {
       i++;
   }
   if (dstsize != 0)
```

```
    {
        while (j < (dstsize - 1) && src[j] != '\0')
        {
            dst[j] = src[j];
            j++;
        }
        dst[j] = '\0';
    }
    return (i);
}
```

17. STRLCAT

→Esta función agrega una cadena al final de la otra. Ofrece un argumento adicional, dstsize. Este argumento establece la longitud de la cadena de destino, de hecho igual a su tamaño de búfer.

17.1. Declaración

```
size_t  strlcat(char *dst, const char *src, size_t dstsize)
```

17.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
dst	char*	Este es un puntero que apunta al bloque de memoria en el que tenemos que copiar los caracteres del puntero src, según el valor del dstsize, ya que dst puede contener algún valor
src	const char*	Este es un puntero que apunta al bloque de memoria en el que tenemos el string
dstsize	size_t lo	Este parámetro nos da el valor de la cadena de dst más los valores que vamos a concatenar de src

17.3. Retorno

Condición	Devuelve	Tipo de variable
dstsize <= longitud de dst dstsize = 0	devuelve dstsize + longitud cadena src	size_t
dstsize > longitud de dst	devuelve longitud cadena dst + longitud cadena src	size_t

17.4. Programa

```c
#include "libft.h"

size_t  ft_strlcat(char *dst, const char *src, size_t dstsize)
{
   size_t  dest;
   size_t  orig;
   size_t  i;

   i = 0;
   orig = ft_strlen(src);
   dest = ft_strlen(dst);
   if (dstsize == 0 || dstsize <= dest)
   {
       return (dstsize + orig);
```

```
    }
    while (i < (dstsize - dest - 1) && src[i])
    {
        dst[dest + i] = src[i];
        i++;
    }
    dst[dest + i] = '\0';
    return (dest + orig);
}
```

a. TOUPPER

→ La función convierte una letra minúscula en la letra mayúscula correspondiente. El argumento debe poder representarse como un unsigned char o el valor de EOF (End Of File)

18.1. Declaración

```
int  toupper(int c)
```

18.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
c	int	Este parámetro recoge el valor de la letra que debemos convertir en su mayúscula

18.3. Retorno

Devuelve	Tipo de variable
devuelve la mayúscula correspondiente del carácter que hemos recibido. Si no, el argumento se devuelve sin cambios	int

18.4. Programa

```c
#include <stdio.h>

int ft_toupper(int c);

int main(void)
{
   int c;

   c = 'a';
   c = ft_toupper(c);
   printf("%c", c);
   return (0);
}

int ft_toupper(int c)
{
   if (c >= 97 && c <= 122)
       c = c - 32;
   return (c);
}

/*OTRA OPCIÓN*/
{
   if (c >= 'a' && c <= 'z')
       return (c - ('a' - 'A'));
   return (c);
}c
```

18. TOLOWER

→ La función convierte una letra mayúscula en la letra minúscula correspondiente. El argumento debe poder representarse como un unsigned char o el valor de EOF (End Of File)

19.1. Declaración

```
int  tolower(int c)
```

19.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
c	int	Este parámetro recoge el valor de la letra que debemos convertir en su minúscula

19.3. Retorno

Devuelve	Tipo de variable
devuelve la minúscula correspondiente del carácter que hemos recibido. Si no, el argumento se devuelve sin cambios	int

19.4. Programa

```c
#include <stdio.h>

int ft_tolower(int c);

int main(void)
{
   int c;

   c = 'Q';
   c = ft_tolower(c);
   printf("%c", c);
   return (0);
}

int ft_tolower(int c)
{
   if (c >= 65 && c <= 90)
       c = c +32;
   return (c);
}

/*OTRA OPCIÓN*/
{
   if (c >= 'A' && c <= 'Z')
       return (c + ('a' - 'A'));
   return (c);
}
```

19. STRCHR

→ Busca cierto carácter dentro de una cadena de texto.

20.1. Declaración

```
char  *strchr(const char *s, int c)
```

20.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
s	char*	String a analizar
c	char	Carácter a buscar dentro del string

20.3. Retorno

Devuelve	Tipo de variable
devuelve la primera posición o un puntero a la posición `&str[i]` , donde se encuentra el carácter buscado. Si no lo encuentra, devuelve NULL.	char *

Para poder mostrar el NULL, te pide una librería (“libft.h”)

20.4. Programa

```c
#include "libft.h"

char    *ft_strchr(const char *s, int c)
{
   int     i;
   char    *str;

   str = (char *)s;
   i = 0;
   while (str[i] != '\0')
   {
       if (str[i] == (unsigned char)c)
           return (&str[i]);
       i++;
   }
   if ((unsigned char)c == '\0')
       return (&str[i]);
   return (NULL); //o return (0);
}
```

20.

21. STRRCHR

→ Busca cierta letra dentro de una cadena de texto empezando desde detrás de la cadena.

f 21.1. Declaración

```
char  *strrchr(const char *s, int c)
```

21.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
s	char*	String a analizar
c	char	Carácter a buscar dentro del string

21.3. Retorno

Devuelve	Tipo de variable
devuelve la primera posición o un puntero a la posición `&str[i]` , donde se encuentra el carácter buscado. Si no lo encuentra, devuelve NULL.	char *

Para poder mostrar el NULL, te pide una librería (“libft.h”)

21.4. Programa

```c
#include "libft.h"

char    *ft_strrchr(const char *s, int c)
{
   char    *str;
   int     i;

   i = 0;
   str = (char *)s;
   while (str[i] != '\0')
       i++;
   while (i >= 0)
   {
       if (str[i] == (unsigned char)c)
           return (&str[i]);
       i--;
   }
   return (NULL);
}
```

22. STRNCMP

→ La función compara el string 1 y el string 2 n número de caracteres. En este caso he almacenado los const char en unsigned char.

22.1. Declaración

```
int  strncmp(const char *str1, const char *str2, size_t n)
```

22.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
str1	const char*	La primera cadena para comparar
str2	const char*	La segunda cadena para comparar
n	size_t	Número máximo de caracteres a comparar

22.3. Retorno

Condición	Devuelve	Tipo de variable
str1 < str2 (ASCII)	devuelve str1 - str2 (valor negativo)	int
str2 > str1 (ASCII)	devuelve str1 - str2 (valor positivo)	int
str1 = str2	devuelve 0	int

Excepto, en el caso de que las dos string sean iguales, el valor devuelto es la diferencia del valor, del siguiente carácter al que es idéntico, y el valor de la misma posición del segundo carácter.

22.4. Programa

```
int ft_strncmp(const char *s1, const char *s2, size_t n)
{
   size_t          i;
   unsigned char   *str1;
   unsigned char   *str2;

   i = 0;
   str1 = (unsigned char *)s1;
   str2 = (unsigned char *)s2;
   while ((str1[i] != '\0' || str2[i] != '\0') && (i < n))
   {
       if (str1[i] < str2[i])
       {
           return (str1[i] - str2[i]);
       }
       else if (str1[i] > str2[i])
       {
```

```
            return (str1[i] - str2[i]);
        }
        i++;
    }
    return (0);
}
```

23. MEMCHR

→ Esta función localiza la primera posición de c en los n caracteres iniciales del objeto al cual señala s.

23.1. Declaración

```
void *memchr (const void *s, int c, size_t n)
```

23.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
s	const void*	Este parámetro recoge el valor del string en el que tenemos que buscar el valor de c
c	int	Este parámetro recoge el valor del carácter que debemos buscar en s. Al ser un int hay que castearla a unsigned char
n	size_t	Este parámetro recoge el valor de la letra que debemos convertir en su minúscula

23.3. Retorno

Condición	Devuelve	Tipo de variable
Encuentra el carácter	devuelve la primera posición o puntero a la posición donde se encuentra el carácter buscado	void *
No encuentra el carácter	devuelve NULL	void *

Para que pueda devolver NULL hay que declarar la librería <libft.h>

23.4. Programa

```
#include "libft.h"

void    *ft_memchr(const void *s, int c, size_t n)
ft_mem:w,
```

24. MEMCMP

→ La función compara el string 1 y el string 2 n número de caracteres.

24.1. Declaración

```
int memcmp(const void *s1, const void *s2, size_t n);
```

24.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
str1	const void*	La primera cadena para comparar
str2	const void*	La segunda cadena para comparar
n	size_t	Número máximo de caracteres a comparar

24.3. Retorno

Condición	Devuelve	Tipo de variable
str1 < str2 (ASCII)	devuelve str1 - str2 (valor negativo)	int
str2 > str1 (ASCII)	devuelve str1 - str2 (valor positivo)	int
str1 = str2	devuelve 0	int

24.4. Programa

```c
#include "libft.h"

int ft_memcmp(const void *s1, const void *s2, size_t n)
{
   while (n > 0)
   {
       if (*(unsigned char *)s1 != *(unsigned char *)s2)
       {
           return (*(unsigned char *)s1 - *(unsigned char *)s2);
       }
       n--;
       s1++;
       s2++;
   }
   return (0);
}
```

25. ºSTRNSTR

→ Busca una subcadena (`needle`) en una cadena de texto (`haystack`) en los primeros `len` caracteres. Hace referencia a buscar una aguja (`needle`) en un pajar (`haystack`).

25.1. Declaración

```
char *strnstr(const char *haystack, const char *needle, size_t len);
```

25.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
haystack	const void*	Este parámetro recoge el valor de la cadena en la que tenemos que buscar la subcadena (pajar)
needle	const void*	Este parámetro recoge el valor de la subcadena que tenemos que buscar en la cadena (aguja)
len	size_t	Número de caracteres a buscar

25.3. Retorno

Condición	Devuelve	Tipo de variable
needle = NULL	devuelve la cadena haystack	char *
needle no se encuentra en haystack	devuelve NULL	char *
needle se encuentra en haystack	devuelve un puntero al primer carácter encontrado	char *ft

25.4. Programa

```c
#include "libft.h"

char    *ft_strnstr(const char *haystack, const char *needle, size_t len)
{
   size_t  i;
   size_t  j;

   i = 0;
   if (needle[i] == '\0')
   {
       return ((char *)haystack);
   }
   while (haystack[i] && i < len)
   {
       j = 0;
       while (haystack[i + j] == needle[j] && i + j < len)
       {
           if (needle[j + 1] == '\0')
```

```
            {
                return ((char *)haystack + i);
            }
            j++;
        }
        i++;
    }
    return (NULL);
}
```

26. ATOI

→ La función busca en una cadena de diferentes caracteres un número, solo busca hasta que el siguiente carácter sea algo distinto de un número.

26.1. Declaración

```
int atoi(const char *str);
```

26.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
str	const char*	Este parámetro recoge la cadena de caracteres en la que debemos buscar el número.

26.3. Retorno

Devuelve	Tipo de variable
devuelve cualquier número convertido en int hasta que encuentra algún carácter no numérico	int

Si es antes del número y del primer signo “\f”, “\t”, “ “, “\n”, “\r”, “\v” lo ignora, si después hay un signo si es negativo el resultado dará negativo con signo, si es positivo el resultado dará positivo sin signo. Si después del signo no hay un número, devuelve cero como resultado.

Ejemplos:

Entrada	str	“123”	“-123”	“+123”	“ 123”	“ - -123”	“ -123p2”
Salida	int	123	-123	123	123	0	-123

26.4. Programa

```c
#include "libft.h"

/*Esta es una recreación de la función atoi en C. Tomamos una cadena de
* caracteres que se supone que son un número convertido en un int.
* Según el hombre, "La función atoi convierte la porción inicial de
* la cadena apunta a la representación de by str a int."*/

int         ft_atoi(const char *str)
{

/*Comenzamos creando tres variables. El primero es lo que sostendrá nuestro
* resultado que se va a devolver. Usamos largo porque está garantizado para
* ser capaz de almacenar, al menos, valores que se encuentran dentro del rango de
* -2147483647 y 2147483647. Sign será lo que usemos para convertir el int
* negativo en el caso de que sea un número negativo el que se pone en el
```

```
* string. Lo tenemos como largo, por lo que podemos multiplicar nuestro resultado por él
en el final. Luego tenemos un int sin signo i, que será el contador de nuestro string.
Para poder compensar una cadena increíblemente larga usamos un iunsigned int para poder
usar su rango positivo extendido tiene más de un signed int. Los pondremos todos a 0
excepto nuestro signo que establecemos en 1 para usar en función de la aparición de un
negativo símbolo en nuestra cadena*/
   long            res;
   long            sign;
   unsigned int    i;

   res = 0;
   sign = 1;
   i = 0;
/*Lo primero que queremos que haga nuestra función es asegurarnos de omitir cualquier
* tipo de espacio que se puede encontrar al principio del string.*/
   while (str[i] == ' ' || str[i] == '\t' || str[i] == '\n'
       || str[i] == '\r' || str[i] == '\v' || str[i] == '\f')
       i++;
/*Una vez pasado el espacio adicional, si existe, estamos comprobando si hay
* es un símbolo negativo al comienzo del número que convertiremos.
 si vemos un símbolo negativo o un símbolo positivo, ajustamos en consecuencia. Si
 es negativo establecemos nuestro signo igual a -1 para multiplicar nuestro resultado
cuando
 lo devolvemos.*/
   if (str[i] == '-' || str[i] == '+')
   {
       if (str[i] == '-')
           sign = -1;
       i++;
   }
/*Aquí convertimos nuestra cadena de caracteres de char a int siempre y cuando
* son números. Si el personaje en el que nos encontramos actualmente es un número,
* lo convirtió a su valor numérico ascii. Para el primer carácter res
* siempre se establece actualmente en 0. Multiplicamos 10 inmediatamente por nuestro res
para
* configure la ubicación de los dígitos donde debería estar. Luego restamos el
* valor numérico del carácter 0 en la tabla ascii de nuestro actual
* número de carácter. Esto lo establece en su valor numérico ascii. Nosotros entonces
* Comience nuestro bucle de nuevo y continúe hasta que lleguemos a un carácter que no
sea un
* número.*/
   while (str[i] >= '0' && str[i] <= '9')
   {
       res = res * 10 + str[i] - '0';
       i++;
   }
/*Por último, devolvemos el valor res multiplicado por el valor del signo para devolver
el
* número en función de si fue negativo o no. NOTA: Tenemos int en
* paréntesis para emitir el resto * inicie sesión en un int para que pueda devolverse
como
* un int.*/
   return ((int)(res * sign));
}
int main(void)
{
   char *s = " 98764 987";
   printf("%d %d", ft_atoi(s), atoi(s));
   return (0);
```

```
}
```

27. MALLOC(3)

→ Esta función asigna de manera contigua suficiente espacio para contar objetos que son bytes de tamaño de memoria cada uno y devuelve un puntero a la memoria asignada.

El (3) hace referencia a la hora de buscar información en el man y aunque en este caso nos ofrece la misma, para buscar debemos indicar `man 3 malloc`

28. CALLOC (+MALLOC)

→ Esta función asigna de manera contigua suficiente espacio para contar objetos que son bytes de tamaño de memoria cada uno y devuelve un puntero a la memoria asignada. La memoria asignada se llena con bytes de valor cero.

28.1. Declaración

```
void *calloc(size_t count, size_t size);
```

28.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
count	size_t	Este parámetro recoge el número de elementos que se van a guardar en la memoria
size	size_t	Este es el tamaño de los elementos. Se puede usar la función sizeof () para saber el tamaño del elemento. Por ejemplo: sizeof (char) = 1 byte, sizeof (int) = 4 byte

28.3. Retorno

Condición	Devuelve	Tipo de variable
Se asigna memoria suficiente	devuelve un puntero a la memoria asignada	void *
No se asigna memoria	devuelve NULL	void *

Si tiene éxito, malloc(), realloc(), reallocf(), valloc() y aligned_alloc() devuelven un puntero a la memoria asignada. Si hay un error, devuelven un puntero NULL y establecen errno en ENOMEM.

28.4. Programa

```
#include "libft.h"
```

```
void    *ft_calloc(size_t count, size_t size)
{
   void    *dest;

   dest = malloc(sizeof(size) * count); sizeof, proporciona la cantidad
                                        de almacenamiento necesaria
   if (!dest)
   {
       return (NULL);
   }
   ft_memset(dest, 0, size * count);
   return (dest);
}

int main(void)
{
   int *pData;

   int i, n;
   printf("El número de dígitos que se introducirán:");
   scanf("%d", &i);
   pData = (int *)calloc(i, sizeof(int));
   if (pData == NULL)
       exit(1);
   for (n = 0; n < i; n++)
   {
       printf("ingresa un número #% d:", n + 1);
       scanf("%d", &pData[n]);
   }
   printf("El número que ingresaste es:");
   for (n = 0; n < i; n++)
       printf("%d ", pData[n]);
   free(pData);
   system("pause");
   return (0);
}
```

estructura necesaria de Malloc:

```
if (!dest)  si el destino está vacío
   {
       return (NULL);  devuelve NULL
   }
```

Alternativa de Main

```
int main(void)
{
   printf("\n---- calloc ----\n");
   int *calloc_test;
   calloc_test = (int *)ft_calloc(6, sizeof(int));
   printf("Calloc an array of 6 int\n    ");
   for(int i = 0; i < 6; i++)
       printf("%d ", calloc_test[i]);
   printf("\n");
   free(calloc_test);
```

```
}
```

29. ºººSTRDUP(+MALLOC)

→ Asigna memoria suficiente para una copia de la cadena s1, hace la copia y le devuelve un puntero. El puntero se puede utilizar posteriormente como argumento de la función free(3).

Si no hay suficiente memoria disponible, se devuelve NULL y errno se establece en ENOMEM.

29.1. Declaración

```
char *strdup(const char *s1);
```

29.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
s1	const char*	Este parámetro recoge el valor de la cadena de origen que se va a duplicar

29.3. Retorno

Condición	Devuelve	Tipo de variable
Se asigna memoria suficiente	devuelve un puntero a la cadena recién copiada	char *
No se asigna memoria	devuelve NULL	char *

29.4. Programa

```
#include "libft.h"

char    *ft_strdup(const char *s1)
{
   int cont; /* contador */
   char *s1_new;  /* nuevo puntero  */
   int size;  /* valor de la cadena */

   size = ft_strlen(s1); /* meto el valor de la cadena con un programa contador */
  s1_new = malloc(sizeof(char)  * size + 1); /* guardo el espacio necesario con malloc
(+1= contar el null)*/
   if (!s1_new)  /*función necesaria para evitar errores en el malloc*/
       return (NULL);
   cont = 0; /* inicio el contador al principio */
   while(s1[cont] != '\0') /* mientras el contador no esté en el final */
   {
       s1_new[cont] = s1[cont];  /*igualo las cadenas*/
       cont++;    /* sumo el contador, que corra!!*/
   }
   s1_new[cont] = s1[cont]; /* para leer el total de los caracteres incluido el null */
   return (s1_new); /* retorno mi cedena */
}
```

```c
int main(void)
{
   printf("%s\n", ft_strdup("hola Mundo!!")); /* mi función */
   printf("%s\n", strdup("hola Mundo!!"));  /* función original */
   return (0);
}
 /* tras compilar aplico ./a.out | cat -e para estar seguro del mismo resultado */

 /* El operador sizeof proporciona la cantidad de almacenamiento,
en bytes, necesaria para almacenar un objeto del tipo del operando.
Este operador permite no tener que especificar tamaños de datos dependientes
del equipo en los programas */
```

FUNCIONES ADICIONALES

- [x] ~~ft_substr~~
- [x] ~~ft_strjoin~~
- [ ] ft_strtrim
- [ ] ft_split
- [x] ~~ft_itoa~~
- [ ] ft_strmapi
- [ ] ft_striteri
- [x] ~~ft_putchar_fd~~
- [x] ~~ft_putstr_fd~~
- [x] ~~ft_putendl_fd~~
- [x] ~~ft_putnbr_fd~~

30. SUBSTR (+MALLOC)

→ Qué hace: Coge n `(len)` caracteres de un string (`s`) desde la posición inicial (`start`) que le indiques.

30.1. Declaración

```
char    *ft_substr(char const *s, unsigned int start, size_t len);
```

30.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
s	const char*	Este parámetro recoge el string original
start	unsigned int	Aquí recogemos el valor inicial, desde la posición que queremos empezar
len	size_t	Cantidad de caracteres que queremos coger

30.3. Retorno

Condición	Devuelve	Tipo de variable
String no está vacío	devuelve los n caracteres desde la que hemos iniciado	char *
String está vacío	devuelve NULL	char *

30.4. Programa

```c
#include "libft.h"

char                                                                *ft_
(char const *s, unsigned int start, size_t len)
{
   char            *s_new;
   unsigned int    i;

   /*función para seguridad para substr*/
   if (s == NULL)
       return (NULL);
   /* función de seguridad para substr*/
   if (len > ft_strlen(s))
       /* función normal */
       len = ft_strlen(s);
   i = 0;
   /* casteo malloc y len y los convierto en char,
       a len le sumo 1 para el '\0' */
   s_new = (char *)malloc(sizeof(char) * (len + 1));
   /* función de seguridad para el malloc */
   if (!s_new)
       return (NULL);
   /* cuando desde donde comienzo es mayor que el string */
```

```
    while (start > ft_strlen(s))
    {
        /* llevo mi puntero hasta el final y lo retorno */
        *s_new = '\0';
        return (s_new);
    }
    /* cuando el texto que quiero pintar(osea hay texto) */
    while (len > 0)
    {
        /* la i esta donde marca start(el comienzo) */
        s_new[i] = s[start];
        /* hago sumar la i que corra por el string desde el comienzo que he marcado
        hago que corra también el comienzo(start) y voy restando lo que me va quedando
        de lo que quiero que me pinte(len) */
        i++;
        start++;
        len--;
    }
    /* llevo mi string hasta el final y lo retorno completado */
    s_new[i] = '\0';
    return (s_new);
}

int main(void)
{
    char    *s;

    s = "hola buenas tardes";
    /* comienza en el lugar 5(start) y píntame 8 letras del string(len) */
    printf("%s\n", ft_substr(s, 5, 8));
}
```

Versión recortada

Modo recortado (chequeado con paco), usando ft_strlen y ft_strlcpy. Se puede utilizar el mismo main que el de arriba.

```
#include "libft.h"

char    *ft_substr(char const *s, unsigned int start, size_t len)
{
    char    *strsub;

    if (len > ft_strlen(s))
    {
        len = ft_strlen(s);
    }
    strsub = (char *)malloc(len + 1);
    if (start > ft_strlen(s))
    {
        *strsub = '\0';
        return (strsub);
    }
    if (!strsub)
    {
        return (0);
```

```
    }
    ft_strlcpy(strsub, &s[start], len + 1);
    return (strsub);
}
```

31. STRJOIN

Qué hace: Une la string `s1` junto a la `s2`. Primero coloca la `s1` y posteriormente la `s2`, finalizando con un '\0'.

31.1. Declaración

```
char *ft_strjoin(char const *s1, char const *s2);
```

31.2. Parámetros

Valor a configurar	Tipo de variable	Explicación
s1	char const *	La primera string
s2	char const *	La string a añadir a s1

31.3. Retorno

Condición	Devuelve	Tipo de variable
s1 o s2 está vacío	devuelve NULL	char *
s1 y s2 no están vacíos	devuelve la cadena juntando s1 y s2	char *

Funcion ft_strjoin.c

31.4. Programa

```c
#include "libft.h"

char    *ft_strjoin(char const *s1, char const *s2)
{
   unsigned int    y1;
   unsigned int    y2;  /* creo variables */
   unsigned int    i;
   unsigned int    x;
   char            *str1;

   if (!s1 || !s2)       /* función de seguridad */
       return (NULL);
   y1 = ft_strlen(s1);
   y2 = ft_strlen(s2); /* pongo contador en los 2 strings */
   i = 0;   /* doy valores a las variables para recorrer el string */
   x = -1;   /* la comienzo por -1 porque de 0 me da error */
   str1 = malloc(sizeof(char) * (y1 + y2 + 1)); /* creo el malloc */
   if (!str1)        /* función de seguridad del malloc */
       return (NULL);
```

```
   while (s1[i])   /* mientras el string *s1 partiendo de la posición 0*/
   {
       str1[i] = s1[i];  /* igualo mi string(malloc) con el string de la función */
       i++;  /* y que corra i */
   }
   while (s2[++x]) /* mientras el string s2 poniendo la x a 0 */
       str1[i++] = s2[x]; /* igualo mi string y que corra la i con s2 */
   str1[i] = '\0'; /* cierro mi string */
   return (str1); /* y lo retorno */
}
/* la función suma los 2 strings, s1 + s2  y da como resultado el mio str1 */


/*
```

Versión recortada

Modo recortado (chequeado con paco), usando ft_strlen, ft_strlcpy y ft_strlcat. Se puede utilizar el mismo main que el de arriba.

```
#include "libft.h"

char    *ft_strjoin(char const *s1, char const *s2)
{
   char    *strs1s2;
   size_t  strs1s2_size;

   strs1s2_size = ft_strlen(s1) + ft_strlen(s2) + 1;
   strs1s2 = (char *)malloc(strs1s2_size);
   if (!strs1s2)
       return (0);
   ft_strlcpy(strs1s2, s1, ft_strlen(s1) + 1);
   ft_strlcat(strs1s2, s2, strs1s2_size);
   return (strs1s2);
}
```

32. STRTRIM

Qué hace: Elimina todos los caracteres de la string `set` desde el principio y desde el final de `s1`, hasta encontrar un carácter no perteneciente a `set`. La string resultante se devuelve con una reserva de malloc(3).

32.1. Declaración

```
char    *ft_strtrim(char const *s1, char const *set);
```

32.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
s1	char const *	Es la cadena que se va a recortar
set	char const *	Son los caracteres que, si aparecen en s1, van a cortarla por delante y por detrás

32.3. Retorno

Condición	Devuelve	Tipo de variable
Si falla reserva memoria	devuelve NULL	char *
s1 y s2 no están vacíos	devuelve la cadena juntando s1 y s2	char *

32.4. Programa

```c
#include "libft.h"

static size_t   ft_check_char(char c, char const *set)
{
   size_t  i;

   i = 0;
   while (set[i])
   {
       if (set[i] == c)
           return (1);
       i++;
   }
   return (0);
}

char    *ft_strtrim(char const *s1, char const *set)
{
   char    *str;
   size_t  i;
   size_t  start;
   size_t  end;

   if (!s1 || !set)
```

```
        return (NULL);
    start = 0;
    while (s1[start] && ft_check_char(s1[start], set))
    {
        start++;
    }
    end = ft_strlen(s1);
    while (end > start && ft_check_char(s1[end - 1], set))
    {
        end--;
    }
    str = (char *)malloc(sizeof(*s1) * (end - start + 1));
    if (!str)
        return (NULL);
    i = 0;
    while (start < end)
        str[i++] = s1[start++];
    str[i] = 0;
    return (str);
}

/* int  main(void)
{
    char    s1[] = "lorem ipsum dolor sit amet";
    char    *set;
    size_t  i;
    size_t  j;
    char    *strtrim;

    set = "l ";
    i = ft_sizefront(s1, set);
    j = ft_sizeend(s1, set);
    printf("Valor Start: %zu\n", i);
    printf("Valor End: %zu\n", j);
    strtrim = ft_strtrim(s1, set);
    printf("%s\n", strtrim);
}
*/IT
```

Versión recortada

Modo recortado (No pasa un tester usando francinette (paco), hay que mirarlo), usando ft_strlen, ft_strchr y ft_strsub. Se puede utilizar el mismo main que el de arriba.

```
#include "libft.h"

char    *ft_strtrim (char const *s1, char const *set)
{
    char    *str_trim;
    size_t  str_trim_size;
    size_t  cnt_start;

    cnt_start = 0;
    while (ft_strchr(set, s1[cnt_start]))
        cnt_start++;
    str_trim_size = ft_strlen(s1);
    while (ft_strchr(set, s1[str_trim_size]))
```

```
        str_trim_size--;
    str_trim = (char *)malloc(sizeof(char) * (str_trim_size - cntstart + 1));
    str_trim = ft_strsub(s1, cnt_start, str_trim_size - cnt_start + 1);
    return(str_trim);
}
```

33. SPLIT

Qué hace: Reserva (utilizando malloc(3)) un array de strings resultante de separar la string `s` en substrings utilizando el caracter `c` como delimitador. El array debe terminar con un puntero NULL.

33.1. Declaración

```
char    **ft_split(char const *s, char c);0
```

33.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
s	char const *	Es la cadena que se va a cortar en cadenas más pequeñas
c	char c	Es el carácter que va a delimitar cada subcadena

33.3. Retorno

Condición	Devuelve	Tipo de variable
Si falla reserva memoria	devuelve NULL	char **
Si c no aparece en s	devuelve la cadena s en la posición [0] de la matriz	char **
Si c aparece en s	devuelve una matriz de los segmentos que haya	char **

Ejemplos:

s	c	return (char **)
“Hola me llamo Ander”	‘a’	Matriz de 3 cadenas de caracteres “Hol” | “ me ll” | “mo Ander”
“Hola soy Ander”	‘ ‘ (espacio)	Matriz de 3 cadenas de caracteres “Hola” | “soy” | “Ander”

33.4. Programa

```
#include "libft.h"

static size_t	ft_size_substring(char const *s, char c)
{
	size_t	i;
```

```
    size_t  numsubstr;

    i = 0;
    numsubstr = 0;
    while (s[i])
    {
        if (s[i] != c)
        {
            numsubstr++;
        }
        while (s[i] && s[i] != c)
        {
            i++;
        }
        if (s[i])
        {
            i++;
        }
    }
    return (numsubstr);
}

char    *ft_substring(char const *s, char c, size_t j)
{
    size_t  i;
    size_t  count;
    char    *str;

    str = NULL;
    i = -1;
    count = -1;
    while (++i < (size_t)ft_strlen(s) && s[i])
    {
        if (s[i] != c)
            count++;
        if (count == j && s[i] != c)
        {
            count = i;
            while (s[count] && s[count] != c)
                count++;
            str = ft_substr(s, i, count - i);
            if (!str)
                return (NULL);
        }
        while (s[i] && s[i] != c)
            i++;
    }
    return (str);
}

char    **ft_split(char const *s, char c)
{
    char    **strings;
    size_t  substring;
    size_t  i;

    if (!s)
        return (NULL);
    substring = ft_size_substring(s, c);
```

```
    if (c == '\0' && ft_strlen(s) > 0)
        substring = 1;
    strings = (char **)ft_calloc(substring + 1, sizeof(char *));
    if (strings == NULL)
        return (NULL);
    i = 0;
    while (i < substring)
    {
        strings[i] = ft_substring(s, c, i);
        if (strings[i] == NULL)
            return (NULL);
        i++;
    }
    strings[i] = NULL;
    return (strings);
}

/* int  main(void)
{
    char    *s = "      gfdgf      ";
    char    c = ' ';
    size_t  count;
    char    **tab = ft_split(s, c);
    size_t  i;

    i = 0;
    count = ft_size_substring(s, c);
    printf("s[0]: %c\n", s[0]);
    printf("COUNT en main: %zu\n", count);
    while (i < count)
    {
        printf("%s\n", tab[i]);
        i++;
    }
}
*/
```

34. ITOA

Qué hace: Convierte un número `int` en una cadena de caracteres. Se hace una reserva en la memoria con malloc (3) del espacio que vamos a necesitar.

34.1. Declaración

```
char    *ft_itoa(int n);
```

34.2. Parámetros

Valor a configurar	Tipo de variable	Valor a introducir
n	int	n = 1456

34.3. Retorno

Devuelve	Tipo de variable	Número positivo ejemplo, n = 146	Número negativo ejemplo, n = -146
String	char *	String de tamaño 4: 1 | 4 | 6 | \0	String de tamaño 5: - | 1 | 4 | 6 | \0

34.4. Programa

```
#include "libft.h"

static int  ft_length(int n_length)
{
   int length;

   length = 1;
   while (n_length / 10 != 0)
   {
       length++;
       n_length = n_length / 10;
   }
   if (n_length < 0)
       length = length + 1;
   return (length);
}

char    *ft_itoa(int n)
{
   char    *strnum;
   int     counter;
   int     str_length;
   long    nb;

   nb = n;
   str_length = ft_length(nb);
```

```
    strnum = (char *)malloc(sizeof(char) * (str_length + 1));
    if (!(strnum))
        return (NULL);
    strnum[str_length] = '\0';
    counter = 0;
    if (nb < 0)
    {
        strnum[counter] = '-';
        nb = nb * -1;
    }
    if (nb == 0)
        strnum[0] = '0';
    while (nb > 0)
    {
        strnum[str_length - counter++ - 1] = nb % 10 + '0';
        nb = nb / 10;
    }
    return (strnum);
}

/* int  main(void)
{
    int     num;
    int     i;
    char    *output;

    num = 123456;
    i = 0;
    output = ft_itoa(num);
    while (output[i])
    {
        write(1, &output[i], 1);
        i++;
    }
    free(output);
    return (0);
} */
```

35. STRMAPI

Qué hace: A cada carácter de la string `s`, aplica la función `f` dando como parámetros el índice de cada carácter dentro de `s` y el propio carácter. Genera una nueva string con el resultado del uso sucesivo de `f`

35.1. Declaración

```
char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
```

35.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
s	const char	String a la que se le aplica la función f
f	char	Función a aplicar a cada carácter

35.3. Retorno

Devuelve	Tipo de variable	String original	String devuelta al aplicarle la función f = - 32
String	char *	**“hola”**	“HOLA”

35.4. Programa

```c
#include "libft.h"

char    *ft_strmapi(char const *s, char (*f)(unsigned int, char))
{
   unsigned int    i;
   char            *str;

   if (!s)
       return (NULL);
   i = 0;
   str = (char *)malloc(sizeof(char) * (ft_strlen(s)) + 1);
   if (str == NULL)
       return (NULL);
   while (s[i] != '\0')
   {
       str[i] = f(i, s[i]);
       i++;
   }
   str[i] = '\0';
   return (str);
}
/* char mi_funcion(unsigned int i, char str)
{
   i = 32;
   return (str - i);
```

```
}
int main(void)
{
   char    *str;
   char    *resultado;

   str = "?hola?guapa.?";
   printf("el resultado es: %s\n", str);
   resultado = ft_strmapi(str, mi_funcion);
   printf("el resultado es: %s\n", resultado);
   return (0);
}
*/
```

36. STRITERI

Qué hace: A cada carácter de la string `s`, aplica la función `f` dando como parámetros el índice de cada carácter dentro de `s` y la dirección del propio carácter, que podrá modificarse si es necesario.

36.1. Declaración

```
void ft_striteri(char *s, void (*f)(unsigned int, char*));
```

36.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
s	const char	String a la que se le aplica la función f
f	char	Función a aplicar a cada carácter

36.3. Retorno

Devuelve	Tipo de variable
Nada	void

36.4. Programa

```c
#include "libft.h"

void    ft_striteri(char *s, void (*f)(unsigned int, char*))
{
    size_t  i;

    i = 0;
    if (!s || !f)
        return;
    while(s[i] != '\0')
    {
        f(i, &s[i]);
        i++;
    }
}


void    ft_change(unsigned int i, char *str)
{
        *str = *str + i;
}

int main(void)
{
```

```
    char    string[] = "Holaaaagurrrr";
    printf("string sin modificar = %s\n", string);
    ft_striteri(string, ft_change);
    printf("string modificada = %s\n", string);
    return(0);
}
```

37. PUTCHAR_FD

Qué hace: Esta función envía el carácter `c` al file descriptor `fd` especificado.

37.1. Declaración

```
void    ft_putchar_fd(char c, int fd);
```

37.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
c	const char	El carácter a enviar
fd	char	El file descriptor sobre el que escribir fd = 1 (Muestra en pantalla)

37.3. Retorno

NO devuelve nada.

37.4. Programa

```
#include "libft.h"

void    ft_putchar_fd(char c, int fd)
{
   write(fd, &c, 1);
}

int main(void)
{
   char    c;
   int     fd;

   c = 'M';
   fd = 1;
   ft_putchar_fd(c, fd);
}
```

38. PUTSTR_FD

Qué hace: Esta función envía la string `s` al file descriptor `fd` especificado.

38.1. Declaración

```
void    ft_putstr_fd(char *s, int fd);
```

38.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
c	const char	El carácter a enviar
fd	char	El file descriptor sobre el que escribir fd = 1 (Muestra en pantalla)

38.3. Retorno

NO devuelve nada.

38.4. Programa

```
#include "libft.h"

void    ft_putstr_fd(char *s, int fd)
{
   unsigned int    len;
   unsigned int    i;

   i = 0;
   len = ft_strlen(s);
   while (i < len)
   {
       write(fd, &s[i], 1);
       i++;
   }
}
int main(void)
{
   char    *c;
   int     fd;

   c = "MAI";
   fd = 1;
   ft_putstr_fd(c, fd);
}
```

39. PUTENDL_FD

Qué hace: Esta función envía la string `s` al file descriptor `fd` dado, seguido de un salto de línea.

39.1. Declaración

```
void    ft_putendl_fd(char *s, int fd);
```

39.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
c	const char	El carácter a enviar
fd	char	El file descriptor sobre el que escribir fd = 1 (Muestra en pantalla)

39.3. Retorno

NO devuelve nada.

39.4. Programa

```
#include "libft.h"

void    ft_putendl_fd(char *s, int fd)
{
   unsigned int    len;
   unsigned int    i;
   i = 0;
   len = ft_strlen(s);
   while (i < len)
   {
       write(fd, &s[i], 1);
       i++;
   }
   write(fd, "\n", 1);
}
int main(void)
{
   char    *c;
   int     fd;
   c = "MAI";
   fd = 1;
   ft_putendl_fd(c, fd);
}
```

40. PUTNBR_FD

Qué hace: Esta función envía el número `n` al file descriptor `fd` dado.

40.1. Declaración

```
void    ft_putnbr_fd(int n, int fd);
```

40.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
c	const char	El carácter a enviar
fd	char	El file descriptor sobre el que escribir fd = 1 (Muestra en pantalla)

40.3. Retorno

NO devuelve nada.

40.4. Programa

```
#include "libft.h"

void    ft_putnbr_fd(int n, int fd)
{
   unsigned int    nb;

   if (n == -2147483648)
   {
       write(fd, "-2147483648", 11);
       return ;
   }
   if (n < 0)
   {
       ft_putchar_fd('-', fd);
       nb = (unsigned int)(n * -1);
   }
   else
   {
       nb = (unsigned int)(n);
   }
   if (nb > 9)
   {
       ft_putnbr_fd(nb / 10, fd);
   }
   ft_putchar_fd((char)(nb % 10 + '0'), fd);
}

int main(void)
{
int     nmb = -2147483648;
ft_putnbr_fd(nmb, 1);
```

```
return (0);
}
```

BONUS

- [x] ~~LSTNEW~~
- [x] ~~LSTADD_FRONT~~
- [x] ~~LSTSIZE~~
- [x] ~~LSTLAST~~
- [ ] LSTADD_BACK
- [ ] LSTDELONE
- [ ] LSTCLEAR
- [ ] LSTITER
- [ ] LSTMAP

LISTAS

Un estudiante del 42 (sperez-p) nos ha pasado este enlace para ver videos sobre listas. Nos comenta lo siguiente: *“Hasta el video 13 son todo linked lists. También tiene punteros, y de algoritmos y de todo. Casi todo lo que he aprendido en 42 viene de este muchacho.”*

https://www.youtube.com/watch?v=NobHlGUjV3g&list=PL2_aWCzGMAwI3W_JlcBbtYTwiQSsOTa6P&index=3

Typedef y struct

La palabra reservada typedef nos permite definir tipos de variables. En este caso, creamos un tipo de variable llamado struct, donde podremos definir nuestras listas.

Una forma alternativa de definir una estructura es:

```
typedef struct
{
    char    c;
    int     i;
```

5
4
3
2
1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

Resumen

Esquema

xº ÍNDICE

APRENDIZAJES DE LA VIDA

Aprendizajes de la vida

Web para ver qué ocurre paso a paso

Crear nuestro portfolio: github benefits to create a website

Entrega de bonus

MAKEFILE Y LIBFT

Makefile

2.1. Vídeo Youtube

2.2. Windows

Para pasar los Tester

2.3. Explicación de cada punto

2.4. NEW MAKE

LIBFT (crear una librería para poder usarla. + makefile)

3.1. Explicación de cada punto

RECOMENDACIÓN: Eliminar las librerías que no usen.

3.2. Programa

3.3 LIBFT TESTERS

3.3.1. WAR MACHINE

3.3.3. LIBFT-UNIT-TEST

3.3.4. LIBTEST

3.3.5. Francinette

ERRORES

Errores

4.1 zsh: illegal hardware instruction

4.2. Bus error

4.3. Segmentation fault

4.4. zsh: permission denied

FUNCIONES

Funciones similares

ISALPHA

5.1. Declaración

5.2. Parámetros

5.3. Retorno

5.4. Programa

ISDIGIT

6.1. Declaración

6.2. Parámetros

6.3. Retorno

6.4. Programa

ISALNUM

7.1. Declaración

7.2. Parámetros

7.3. Retorno

7.4. Programa

ISASCII

8.1. Declaración

8.2. Parámetros

8.3. Retorno

8.4. Programa

ISPRINT

9.1. Declaración

9.2. Parámetros

9.3. Retorno

9.4. Programa

STRLEN

10.1. Declaración

10.2. Parámetros

10.3. Retorno

10.4. Programa

MEMSET

m11.1. Declaración

11.2. Parámetros

11.3. Retorno

11.4. Programa

BZERO

12.1. Declaración

12.2. Parámetros

12.3. Retorno

12.4 Programa

MEMCPY

13.1. Declaración

13.2. Parámetros

13.3. Retorno

13.4 Programa

MEMMOVE

14.1. Declaración

14.2. Parámetros

14.3. Retorno

14.4 Programa

FUNCIONES ADICIONALES

SUBSTR (+MALLOC)

30.1. Declaración

30.2. Parámetros

30.3. Retorno

30.4. Programa

Versión recortada

STRJOIN

31.1. Declaración

31.2. Parámetros

31.3. Retorno

31.4. Programa

Versión recortada

STRTRIM

32.1. Declaración

32.2. Parámetros

32.3. Retorno

32.4. Programa

Versión recortada

SPLIT

33.1. Declaración

33.2. Parámetros

33.3. Retorno

33.4. Programa

ITOA

34.1. Declaración

34.2. Parámetros

34.3. Retorno

34.4. Programa

STRMAPI

35.1. Declaración

35.2. Parámetros

35.3. Retorno

35.4. Programa

STRITERI

36.1. Declaración

36.2. Parámetros

36.3. Retorno

36.4. Programa

PUTCHAR_FD

37.1. Declaración

37.2. Parámetros

37.3. Retorno

37.4. Programa

PUTSTR_FD

38.1. Declaración

38.2. Parámetros

38.3. Retorno

38.4. Programa

PUTENDL_FD

39.1. Declaración

39.2. Parámetros

39.3. Retorno

39.4. Programa

PUTNBR_FD

40.1. Declaración

40.2. Parámetros

40.3. Retorno

40.4. Programa

BONUS

LISTAS

Typedef y struct

LSTNEW

https://github.com/edramir18/ft_list

41.1. Declaración

41.2. Parámetros

41.3. Retorno

41.4. Programa

LSTADD_FRONT

46.4. Programa

LSTCLEAR

47.1. Declaración

47.2. Parámetros

47.3. Retorno

47.4. Programa

LSTITER

48.1. Declaración

48.2. Parámetros

48.3. Retorno

48.4. Programa

LSTMAP

49.1. Declaración

49.2. Parámetros

49.3. Retorno

49.4. Programa

Introduction to linked list

```
}                    Ejemplo;
```

Podemos ahora utilizar `Ejemplo` para declarar variables del tipo `struct`, por ejemplo:

```
Ejemplo a[10];
```

41. LSTNEW

Para agregar nuevos elementos a la lista debemos crear un nuevo nodo por medio de la función `ft_lstnew`, a la cual pasaremos un puntero al dato que deseamos almacenar.

Las listas son una estructura de datos que nos permite almacenar una colección de elementos, sin conocer previamente el número total de ellos. Se recomienda cuando es necesario trabajar con un conjunto de elementos a los cuales estamos añadiendo y eliminando elementos de forma continua.

La **estructura** que nos permitirá trabajar con la lista sería la siguiente (esta la ponemos en nuestro libft.h:

```
typedef struct s_list
{
    void            *content;
    struct s_list   *next;
}                   t_list;
```

La estructura contiene 2 apuntadores (*pointer*):

1. `content (void*)`: Es un apuntador sin tipo que nos permitirá almacenar cualquier elemento en esta lista sin necesidad de redefinir el tipo de la variable utilizada.
2. *`next (s_list *)`:* indica el siguiente elemento de la lista. Como el siguiente elemento es otra lista, el tipo de esta variable es un puntero a la lista (`struct s_list *`).

A cada elemento de una lista se le suele denominar **node/nodo**, para indicar que es una estructura de datos que contiene la información necesaria para trabajar con las distintas funciones de listas.

Para el manejo de la lista utilizaremos un puntero de tipo *`t_list`*, el cual debe ser inicializado en **NULL** si no es necesario reservar memoria con *`malloc`* o crear una estructura adicional para manejar la lista, pero en nuestro caso sí vamos a reservar memoria con malloc:

```
    t_list  *list;

    list = malloc(sizeof(*list));
```

https://github.com/edramir18/ft_list

41.1. Declaración

```
t_list  *ft_lstnew(void *content);
```

Crea un nuevo nodo utilizando malloc(3). La variable miembro `content` se inicializa con el contenido del parámetro `content`. La variable `next`, con `NULL`.

41.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
content	void *	El contenido con el que crear el nodo

41.3. Retorno

Condición	Devuelve	Tipo de variable
Si falla reserva memoria	devuelve NULL	t_list
Si OK la reserva de memoria	devuelve el nuevo nodo	t_list

41.4. Programa

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct s_list
{
    void               *content;
    struct s_list      *next;
}                      t_list;

t_list  *ft_lstnew(void *content)
{
    t_list  *list;

    list = malloc(sizeof(*list));
    if (!list)
        return (NULL);
    list->content = content;
    list->next = NULL;
    return (list);
}

int main(void)
{
    char    str[] = "lorem ipsum dolor sit";

    t_list  *elem;

    elem = ft_lstnew((void *)str);
    printf("Contenido (content) del nodo creado: %s\n", elem->content);
    printf("Siguiente nodo (next) del nodo creado: %s\n", elem->next);
}
```

Asigna (con malloc(3)) y devuelve un nuevo elemento.

La variable 'content' se inicializa con el valor del parámetro 'content'.

La variable 'next' se inicializa en NULL.

Crear nueva lista.

En lenguaje C, el operador -> se utiliza después de una variable de tipo puntero que apunta a una estructura de datos, para indicar a qué campo de la estructura queremos acceder. Se indica que accedemos a su campo dato.

Una estructura de datos, es una variable que puede contener otras variables.

42. LSTADD_FRONT

42.1. Declaración

```
void ft_lstadd_front(t_list **lst, t_list *new);
```

Añade el nodo ’new’ al principio de la lista ’lst’.

42.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
lst	t_list **	Es la dirección de un puntero al primer nodo de una lista. Es doble puntero porque el primer nodo es un puntero, por lo que para apuntar a un puntero tengo que definir doble puntero
new	t_list *	Es un puntero al nodo que añadir al principio de la lista

42.3. Retorno

No retorna nada

42.4. Programa

```c
#include "libft.h"

void    ft_lstadd_front(t_list **lst, t_list *new)
{
   if (!new)
       return ;
   if (!lst)
   {
       *lst = new;
       return ;
   }
   new->next = *lst;
   *lst = new;
}

int main(void)
{
    t_list  *new_list;
    char    str_new[] = "Kaixo";
    t_list  *previous_list;
    char    str_previous[] = "Agur";

    previous_list = malloc(sizeof(*previous_list));
    new_list = malloc(sizeof(*new_list));
    previous_list->content = (void *)str_previous;
    new_list->content = (void *)str_new;

    // Defino dos lista independientes. Ahora voy a añadir new_list a previous_list
```

```
    ft_lstadd_front(&previous_list, new_list);
    printf("El contenido del nodo nuevo es: %s", previous_list->content);
    // Ahora el puntero previous_list apunta al nodo new, de manera que el content será
"Kaixo"
}
```

43. LSTSIZE

Qué hace: Cuenta el número de nodos de una lista. Misma filosofía que `strlen`.

43.1. Declaración

```
int ft_lstsize(t_list *lst);
```

43.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
lst	t_list *	Es el principio de la lista

43.3. Retorno

Condición	Devuelve	Tipo de variable
Si falla reserva memoria	devuelve NULL	int
Si OK la reserva de memoria	devuelve el número de elementos/nodos de una lista	int

43.4. Programa

```
#include "libft.h"

int ft_lstsize(t_list *lst)
{
   size_t  count;

   count = 0;
   while (lst != NULL)
   {
       lst = lst->next;
       count++;
   }
   return (count);
}

int main(void)
{
    t_list  *lista;
    t_list  *lista1;
    t_list  *lista2;

    lista = malloc(sizeof(*lista));
    lista1 = malloc(sizeof(*lista1));
    lista2= malloc(sizeof(*lista2));

    lista->next = lista1;
    lista1->next = lista2;
```

```
    lista2->next = NULL;

    printf("El size de la lista es de: %d\n", ft_lstsize(lista));

    free(lista);
    free(lista1);
    free(lista2);
    return (0);
}
```

44. LSTLAST

Qué hace: Devuelve el último nodo de la lista.

44.1. Declaración

```
t_list *ft_lstlast(t_list *lst);
```

44.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
lst	t_list *	Es el principio de la lista

44.3. Retorno

Condición	Devuelve	Tipo de variable
Si falla reserva memoria	devuelve NULL	t_list
Si OK la reserva de memoria	devuelve el último elemento/nodo de la lista	t_list

44.4. Programa

```c
#include "libft.h"

t_list  *ft_lstlast(t_list *lst)
{
   t_list  *end;

   if (!(lst))
   {
       return (NULL);
   }
   end = lst;
   while (end->next != NULL)
   {
       end = end->next;
   }
   return (end);
}

int main(void)
{
    t_list  *lista;
    t_list  *lista1;
    t_list  *lista2;
    t_list  *last;
    char    str[] = "Soy el ultimo";
```

```
    lista = malloc(sizeof(*lista));
    lista1 = malloc(sizeof(*lista1));
    lista2= malloc(sizeof(*lista2));
    last= malloc(sizeof(*last));

    lista->next = lista1;
    lista1->next = lista2;
    lista2->content = (void *)str;
    lista2->next = NULL;

    last = ft_lstlast(lista);
    printf("El ultimo nodo de la lista contiene: %s\n", last->content);

    free(lista);
    free(lista1);
    free(lista2);
    free(last);
    return (0);
}
```

45. LSTADD_BACK

Qué hace: Añade el nodo `new` al final de la lista `lst`.

45.1. Declaración

```
void ft_lstadd_back(t_list **lst, t_list *new);
```

45.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
lst	t_list **	Es el puntero al primer nodo de una lista
new	t_list *	Es el puntero a un nodo que añadir a la lista

45.3. Retorno

No retorna nada

45.4. Programa

```c
#include "libft.h"

void    ft_lstadd_back(t_list **lst, t_list *new)
{
   t_list  *last;

   last = ft_lstlast(*lst);
   if (!new)
   {
       return ;
   }
   if (!*lst)
       *lst = new;
   else
   {
       if (last == NULL)
           return ;
       last->next = new;
   }
}

int main(void)
{
    t_list  *lista;
    t_list  *lista1;
    t_list  *lista2;
    t_list  *last;
    char    str[] = "Soy el ultimo";

    lista = malloc(sizeof(*lista));
    lista1 = malloc(sizeof(*lista1));
```

```
    lista2 = malloc(sizeof(*lista2));
    last = malloc(sizeof(*last));

    lista->next = lista1;
    lista1->next = lista2;
    lista2->next = NULL;
    last->content = (void *)str;
    last->next = NULL;

    // Las listas están separadas. Primero las unimos y despues chequeamos que esta OK
    ft_lstadd_back(&lista, last);
    last = ft_lstlast(lista);
    printf("El ultimo nodo de la lista unida contiene: %s\n", last->content);

    free(lista);
    free(lista1);
    free(lista2);
    free(last);
    return (0);
}
```

46. LSTDELONE

Qué hace: Toma como parámetro un nodo `lst` y libera la memoria del contenido utilizando la función `del` dada como parámetro, además de liberar el nodo. La memoria de `next` no debe liberarse.

46.1. Declaración

```
void ft_lstdelone(t_list *lst, void (*del)(void *));
```

46.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
lst	t_list *	Es el elemento a liberar la memoria
del	función	Es un puntero a la función para liberar el contenido del nodo

46.3. Retorno

No retorna nada

46.4. Programa

```
#include "libft.h"
#include <stdlib.h>

void    ft_lstdelone(t_list *lst, void (*del)(void *))
{
   if (!lst)     SI NO EXISTE
   {
       return ;     NO HAGAS NADA, sal de la función
   }
   (*del)(lst->content);  borra el parámetro indicado
   free(lst);
}

/* Parámetros lst:  el nodo a liberar.
del:             un puntero a la función utilizada para liberar
                el contenido del nodo. */
/* Valor devuelto:          Nada
Funciones autorizadas:   free
Descripción:              Toma como parámetro un nodo 'lst'
y libera la memoria del contenido utilizando la función 'del'
dada como parámetro, además de liberar el nodo. La
memoria de 'next' no debe liberarse. */
```

47. LSTCLEAR

Qué hace: Elimina y libera el nodo `lst` dado y todos los consecutivos de ese nodo, utilizando la función `del` y `free(3)`. Al final, el puntero a la lista debe ser `NULL`.

47.1. Declaración

```
void ft_lstclear(t_list **lst, void (*del)(void*));
```

47.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
lst	t_list **	Es el elemento y sus consecutivos a liberar la memoria
del	función	Es un puntero a la función para liberar el contenido del nodo

47.3. Retorno

No retorna nada

47.4. Programa

```c
#include "libft.h"

void    ft_lstclear(t_list **lst, void (*del)(void *))
{
   t_list  *new;

   if (!*lst)
   {
       return ;
   }
   if (del && lst)
   {
       while (*lst)
       {
           new = (*lst)->next;       // creo un bucle que va borrando
           ft_lstdelone(*lst, del);  // toda mi lista
           *lst = new;
       }
   }
   lst = NULL;
}

/* Parámetros lst:            la dirección de un puntero a un nodo.
del:                       un puntero a función utilizado para eliminar
                          el contenido de un nodo.
Valor devuelto Nada
Funciones autorizadas:     free
Descripción:               Elimina y libera el nodo 'lst' dado y todos los
                          consecutivos de ese nodo, utilizando
                           la función 'del' y free(3).
```

```
Al final, el puntero a la lista debe ser NULL. */

/*SINOPSIS: borrar secuencia de elementos de lista desde un punto de partida
DESCRIPCIÓN:
Elimina y libera el elemento dado y cada sucesor de ese elemento,
usando la función 'del' y free(3). Finalmente, el apuntador a la lista debe
establecerse en NULL.
*/
```

48. LSTITER

Qué hace: Itera la lista `lst` y aplica la función `f` en el contenido de cada nodo.

48.1. Declaración

```
void ft_lstiter(t_list *lst, void (*f)(void *));
```

48.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
lst	t_list *	Es un puntero al primer nodo
f	función	Es un puntero a la función que utilizará cada nodo

48.3. Retorno

No retorna nada

48.4. Programa

```
#include "libft.h"

void    ft_lstiter(t_list *lst, void (*f)(void *))
{
   if (!lst || !f)
   {
       return ;
   }
   while (lst)
   {
       (*f)(lst->content);
       lst = lst->next;
   }
}

/* Parámetros lst:      un puntero al primer nodo.
f:                      un puntero a la función que utilizará cada nodo.
Valor devuelto:         Nada
Funciones autorizadas:  Ninguna
Descripción:            Itera la lista ’lst’ y aplica la función ’f’ en el
                       contenido de cada nodo. */
/* SINOPSIS: aplica la función al contenido de todos los elementos de la lista
DESCRIPCIÓN: Itera la lista 'lst' y aplica la función 'f' al contenido de
            cada elemento. */
```

49. LSTMAP

Qué hace: Itera la lista `lst` y aplica la función `f` al contenido de cada nodo. Crea una lista resultante de la aplicación correcta y sucesiva de la función `f` sobre cada nodo. La función `del` se utiliza para eliminar el contenido de un nodo, si hace falta.

49.1. Declaración

```
t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));
```

49.2. Parámetros

Valor a configurar	Tipo de variable	Descripción
lst	t_list *	Es un puntero a un nodo
f	función	Es la dirección de un puntero a la función usada en la iteración de cada elemento de la lista
del	función	Es un puntero a una función utilizado para eliminar el contenido de un nodo, si es necesario

49.3. Retorno

Condición	Devuelve	Tipo de variable
Si falla reserva memoria	devuelve NULL	t_list
Si OK la reserva de memoria	devuelve la nueva lista	t_list

49.4. Programa

```c
#include "libft.h"

t_list  *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *))
{
   t_list  *result;
   t_list  *new;

   if (!lst && !*del && !*f)
   {
       return (NULL);
   }
   result = NULL;
   while (lst)
   {
       new = ft_lstnew((*f)(lst->content));
       ft_lstadd_back(&result, new);
       lst = lst->next;
   }
   return (result);
```

```
}

/* Parámetros lst: un puntero a un nodo.
f:                 la dirección de un puntero a una función usada
                   en la iteración de cada elemento de la lista.
del:               un puntero a función utilizado para eliminar
                   el contenido de un nodo, si es necesario.
Valor devuelto La nueva lista.
NULL si falla la reserva de memoria.

Funciones autorizadas:  malloc, free

Descripción:       Itera la lista ’lst’ y aplica la función ’f’ al
                   contenido de cada nodo. Crea una lista resultante
                   de la aplicación correcta y sucesiva de la función
                   ’f’ sobre cada nodo. La función ’del’ se utiliza
                   para eliminar el contenido de un nodo, si hace
                   falta. */

/* SINOPSIS: aplica la función al contenido de todos los elementos de la lista en una
nueva lista

DESCRIPCIÓN:
Itera la lista 'lst' y aplica la función 'f' al contenido de
cada elemento. Crea una nueva lista resultante de las sucesivas aplicaciones de
la función 'f'. La función 'del' se utiliza para borrar el contenido de un
elemento si es necesario. */
```