# LIBFT.


jariza-o // Juan Ariza Ordoñez

42 MÁLAGA FUNDACIÓN TELEFONICA

# Indice.

## 1. Crear fichero C.

El fichero se debe crear con la librería incluida, además de añadir todo lo que te pide la norminette.

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

## 2. Fichero libft.h

Para crear el fichero libft.h, hay que tener en cuenta la norminette. El formato de este fichero es el siguiente:

1. Primero van las siguientes 2 líneas, que la primera empieza la librería y la segunda la define.

```
/* ************************************************************************** */
/*                                                                            */
/*                                                        :::      ::::::::   */
/*   libft.h                                            :+:      :+:    :+:   */
/*                                                    +:+ +:+         +:+     */
/*   By: jariza-o <jariza-o@student.42malaga.com>   +#+  +:+       +#+        */
/*                                                +#+#+#+#+#+   +#+           */
/*   Created: 2022/09/27 19:31:59 by jariza-o          #+#    #+#             */
/*   Updated: 2022/09/28 00:14:52 by jariza-o         ###   ########.fr       */
/*                                                                            */
/* ************************************************************************** */

#ifndef LIBFT_H
# define LIBFT_H
```

2. Después irían las librerías que has tenido que usar para realizar las funciones.

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

3. Iría la lista si se hace la parte bonus.

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

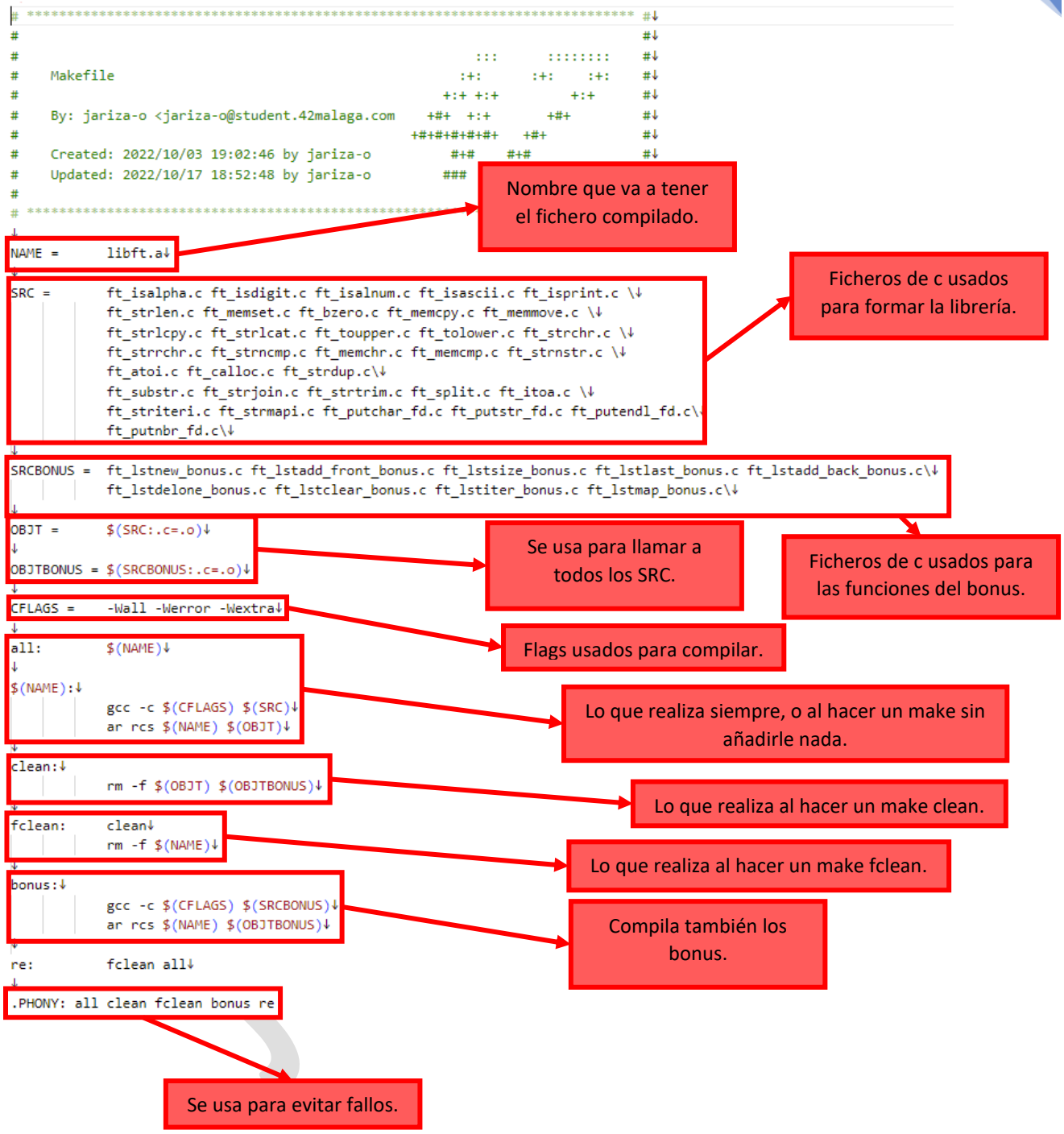
4. Irían todos los archivos C creados, para añadirlos a nuestra librería solo hay que copiar la primera línea de la función.

```
int     ft_isalpha(int c);
int     ft_isdigit(int c);
int     ft_isalnum(int c);
int     ft_isascii(int c);
int     isprint(int c);

size_t  ft_strlen(const char *s);

void    ft_memset(void *b, int c, size_t len);
```

5. Para cerrar la librería escribimos **#endif**.

```
#endif
```

## 3. Fichero Makefile.



# 4. Parte 1 - Funciones de libc.

## ft_isalpha.

Indica si el carácter es alfabético. Devuelve un valor distinto a 0, si el valor está entre a y z o A y Z.

```
#include "libft.h"
↓
int ft_isalpha(int c)
{
    return ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'));
}
```

## ft_isdigit.

Indica si el carácter es numérico. Devuelve un valor distinto a 0, si el valor está entre 0 y 9.

```
#include "libft.h"
↓
int ft_isdigit(int c)
{
    return (c >= '0' && c <= '9');
}
```

## ft_isalnum.

Indica si el carácter es alfanumérico. Devuelve un valor distinto a 0, si el valor está entre 0 y 9, a y z, y A y Z.

```
#include "libft.h"
↓
int ft_isalnum(int c)
{
    return ((c >= '0' && c <= '9')
        || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'));
}
```

## ft_isascii.

Indica si el carácter es un carácter ascci entre el 0 y el 127. Devuelve un valor distinto a 0, si el valor es un carácter ascii entre el 0 y el 127.

```
#include "libft.h"
↓
int ft_isascii(int c)
{
    return (c >= 0 && c <= 127);
}
```

## ft_isprint.

Indica si el carácter es un carácter ascci imprimible. Devuelve un valor distinto a 0, si el valor es un carácter ascii imprimible.

```
#include "libft.h"
↵
int ft_isprint(int c)↵
{↵
    return (c >= 32 && c <= 126);↵
}↵
```

## ft_strlen.

Indica el número de caracteres de un string. Devuelve el número de caracteres que forma el string.

```
#include "libft.h"↓
↓
size_t  ft_strlen(const char *s)↓
{↓
    size_t  n;↓
↓
    n = 0;↓
    while (s[n] != '\0')↓
        n++;↓
    return (n);↓
}↓
```

## ft_memset.

La función memset() copiará c (convertido a un unsigned char) en cada uno de los primeros n bytes del objeto apuntado por s. Devuelve el nuevo string.

```
#include "libft.h"↓
↓
void    *ft_memset(void *b, int c, size_t len)↓
{↓
    unsigned char   *str;↓
↓
    str = (unsigned char *)b;↓
    while (len-- > 0)↓
    {↓
        *(str++) = (unsigned char)c;↓
    }↓
    return (b);↓
}↓
```

## ft_bzero.

Convierte a ‘\0’ los n primeros bytes del string. No devuelve nada.

```
#include "libft.h"↓
↓
void    ft_bzero(void *s, size_t n)↓
{↓
    size_t  counter;↓
↓
    counter = 0;↓
    while (n != 0 && counter++ <= (n - 1))↓
    {↓
        *(char *)s = '\0';↓
        s++;↓
    }↓
}↓
```

## ft_memcpy.

Copia de un string origen a un string destino n bytes. Si ambos string son nulos (0), devuelve nulo (0), sino devuelve la cadena destino.

```
#include "libft.h"
↲
void    *ft_memcpy(void *dst, const void *src, size_t n)↲
{↲
    if (!dst && !src)↲
        return (0);↲
    while (n-- > 0)↲
        *(char *)(dst + n) = *(char *)(src + n);↲
    return (dst);↲
}↲
```

## ft_memmove.

La función memmove() copia n bytes del área de memoria src al área de memoria dest. Las áreas de memoria áreas de memoria pueden superponerse: la copia se realiza como si los bytes de src se copiaran primero en una matriz temporal que no se solapa con src o dest, y los bytes se copian entonces de la matriz temporal a dest. Devuelve nulo si el string destino y origen son nulos, y el string destino en los demás casos.

```
#include "libft.h"↲
↲
void    *ft_memmove(void *dst, const void *src, size_t len)↲
{↲
    char    *d;↲
    char    *s;↲
    size_t  i;↲
↲
    d = (char *)dst;↲
    s = (char *)src;↲
    i = 0;↲
    if (!dst && !src)↲
        return (NULL);↲
    if (d > s)↲
    {↲
        while (len-- > 0)↲
        {↲
            d[len] = s[len];↲
        }↲
    }↲
    else↲
    {↲
        while (i < len)↲
        {↲
            d[i] = s[i];↲
            i++;↲
        }↲
    }↲
    return (dst);↲
}↲
```

## ft_strlcpy.

La función strlcpy copia una cadena src a una dst. Copia hasta el tamaño dstsize -1 caracteres de la cadena terminada en NULL src a dst, terminando en NULL el resultado. Devuelve el tamaño de la cadena de origen.

```c
#include "libft.h"

size_t  ft_strlcpy(char *dst, const char *src, size_t dstsize)
{
    size_t  count;

    count = 0;
    if (dstsize > 0)
    {
        while (count < (dstsize - 1) && src[count] != '\0')
        {
            dst[count] = src[count];
            count++;
        }
        dst[count] = '\0';
    }
    return (ft_strlen(src));
}
```

## ft_strlcat.

Concatena 2 cadenas, la cadena src hasta dst -1. Devuelve la longitud inicial de

inicial de dst más la longitud de src.

```c
#include "libft.h"
↓
size_t  ft_strlcat(char *dst, const char *src, size_t dstsize)
{
    size_t  count;
    size_t  scount;
    size_t  len_d;
↓
    count = 0;
    scount = 0;
    len_d = ft_strlen(dst);
    if (len_d < (dstsize - 1) && dstsize > 0)
    {
        while (dst[count] != '\0')
            count++;
        while (src[scount] != '\0' && count < (dstsize - 1))
        {
            dst[count] = src[scount];
            count++;
            scount++;
        }
        dst[count] = '\0';
    }
    if (len_d >= dstsize)
        len_d = dstsize;
    return (len_d + ft_strlen(src));
}
```

Al devolver el tamaño de src + el tamaño que va a tener la cadena dst, si la cadena dst es más grande o igual que el tamaño que te limita la función, se le indica que el tamaño de dst es dstsize para que sea el de la nueva cadena destino.

## ft_toupper.

Convierte el carácter en mayúscula, si este es minúscula. Devuelve el valor del carácter.

```c
#include "libft.h"
↓
int ft_toupper(int c)
{
    if (c >= 97 && c <= 122)
        c = c - 32;
    return (c);
}
```

## ft_tolower.

Convierte el carácter en minúscula, si este es mayúscula. Devuelve el valor del carácter.

```
#include "libft.h"
↓
int ft_tolower(int c)
{
    if (c >= 65 && c <= 90)
        c = c + 32;
    return (c);
}
```

## ft_strchr.

La función strchr() devuelve un puntero a la primera aparición del carácter c en la cadena s. Si no lo encuentra devuelve NULL.

```
#include "libft.h"
↓
char    *ft_strchr(const char *s, int c)
{
    int count;
↓
    count = 0;
    while (s[count] != '\0')
    {
        if (s[count] == (char)c)
            return ((char *)&s[count]);
        count++;
    }
    if ((char)c == s[count])
        return ((char *)&s[count]);
    return (NULL);
}
```

Se repite dos veces porque el valor de fin de cadena también se tiene en cuenta.

## ft_strrchr.

La función strrchr() devuelve un puntero a la última aparición del carácter c en la cadena cadena s. Si no lo encuentra devuelve NULL.

```
#include "libft.h"
↓
char    *ft_strrchr(const char *s, int c)↓
{↓
    int     count;↓
↓
    count = 0;↓
    while (s[count] != '\0')↓
        count++;↓
    while (count > 0)↓
    {↓
        if (s[count] == (char)c)↓
            return ((char *)&s[count]);↓
        count--;↓
    }↓
    if ((char)c == s[count])↓
        return ((char *)&s[count]);↓
    return (NULL);↓
}↓
```

Se repite dos veces porque el valor de fin de cadena también se tiene en cuenta.

## ft_strncmp.

La función strncmp() debe comparar no más de n bytes (los bytes que siguen a un carácter NUL no se comparan) de la matriz apuntada por s1 a la matriz apuntada por s2. Si llega al final, devuelve la resta del último carácter comparado en ambas cadenas, si n es nulo o si c sobrepasa n -1.

```
#include "libft.h"↵
↵
int ft_strncmp(const char *s1, const char *s2, size_t n)↵
{↵
    size_t  c;↵
↵
    c = 0;↵
    if (n == 0)↵
        return (0);↵
    while (s1[c] == s2[c] && s1[c] != '\0' && s2[c] != '\0')↵
    {↵
        if (c < (n - 1))↵
            c++;↵
        else↵
            return (0);↵
    }↵
    return ((unsigned char)(s1[c]) - (unsigned char)(s2[c]));↵
}↵
```

Se hace n – 1, para tener en cuenta el carácter 0

Si ambos caracteres son iguales, devuelve 0.

## ft_memchr.

La función memchr() localizará la primera aparición de c (convertida en un unsigned char) en los n bytes iniciales (cada uno interpretado como unsigned char) apuntados por s. Devuelve un puntero al byte localizado, o un puntero nulo si el byte no se encuentra.

```
#include "libft.h"
↲
void    *ft_memchr(const void *s, int c, size_t n)↲
{↲
    unsigned char   *ss;↲
    size_t          i;↲
↲
    ss = (unsigned char *)s;↲
    i = 0;↲
    while (i < n)↲
    {↲
        if (ss[i] == (unsigned char)c)↲
            return ((void *)(ss + i));↲
        i++;↲
    }↲
    return (NULL);↲
}↲
```

Se convierte en unsigned char el string recibido, ya que es lo que te pide la función.

Se devuelve como un puntero de void, ya que es lo que te pide la función.

## ft_memcmp.

La función memcmp() comparará los primeros n bytes (cada uno interpretado como unsigned char) del objeto apuntado por s1 con los primeros n bytes del objeto apuntado por s2. El dato que devuelve es la resta de los primeros caracteres distintos encontrados entre ambos strings. SI son todos iguales devuelve 0. PORQUE SE USA CON CONST

```
#include "libft.h"↓
↓
int ft_memcmp(const void *s1, const void *s2, size_t n)↓
{↓
    const unsigned char *ss1;↓
    const unsigned char *ss2;↓
    size_t              c;↓
↓
    c = 0;↓
    ss1 = (const unsigned char *)s1;↓
    ss2 = (const unsigned char *)s2;↓
    while (c < n)↓
    {↓
        if (ss1[c] != ss2[c])↓
            return (ss1[c] - ss2[c]);↓
        c++;↓
    }↓
    return (0);↓
}↓
```

## ft_strnstr.

Busca la primera aparición del string neddle en el string haystack, sin pasar de len (busca la aguja en el pajar en los n primeros bytes). Si needle está vacío devuelve haystack, si haystack está vació o no se encuentra needle en haystack devuelve NULL, y en caso de que se encuentre se devuelve un puntero al primer carácter de la primera ocurrencia.

```
#include "libft.h"
↓
char    *ft_strnstr(const char *haystack, const char *needle, size_t len)
{
    size_t  h;
    size_t  n;
↓
    h = 0;
    if (needle[0] == '\0')
        return ((char *)haystack);
    if (haystack[h] == '\0')
        return (NULL);
    while (haystack[h] != '\0')
    {
        n = 0;
        while (haystack[h + n] == needle[n] && (h + n) < len)
        {
            if (haystack[h + n] == '\0' && needle[n] == '\0')
                return ((char *)&haystack[h]);
            n++;
        }
        if (needle[n] == '\0')
            return ((char *)&haystack[h]);
        h++;
    }
    return (0);
}
```

Si needle está vacía devuelve haystack.

Si haystack está vació devuelve nulo.

Primero se hace un bucle while para recorrer haystack. Después tendrás que declarar la variable que va a recorrer needle, y dentro crear otro bucle para que vaya recorriendo ambos string y comparando.

## ft_atoi.

Esta función convierte un char a int. Solo selecciona el primer signo que se encuentre y los primeros números antes de cualquier letra.

```
#include "libft.h"
↓
int ft_atoi(const char *str)
{
    int count;
    int signo;
    int buffer;
↓
    count = 0;
    signo = 1;
    buffer = 0;
    while ((str[count] >= '\t' && str[count] <= '\r')
        || str[count] == 32)
        count++;
    if (str[count] == 43 || str[count] == 45)
    {
        if (str[count] == 45)
            signo *= -1;
        count++;
    }
    while (str[count] >= '0' && str[count] <= '9')
    {
        buffer = (str[count] - '0') + (buffer * 10);
        count++;
    }
    return (signo * buffer);
}
```

Descarta los caracteres que no se necesitan.

Encuentra el primer signo, y determina si el número va a ser positivo o negativo.

Selecciona los números, y con – '0' pasa el char a int, y con el (buffer * 10) determina en cada pasada la posición de un número en el conjunto de números.

## ft_calloc.

La función calloc pasa asigna memoria a 0 a los x primeros bytes (los bytes es la variable count, y syze es el tamaño que ocupa un byte). Devuelve NULL si mallac es nulo, o el puntero si ha salido bien.

```
#include "libft.h"
↓
void    *ft_calloc(size_t count, size_t size)
{
    void    *ptr;
↓
    ptr = malloc(count * size);
    if (ptr == NULL)
        return (NULL);
    ft_bzero(ptr, (count * size));
    return (ptr);
}
```

Usamos la función bzero, ya que esta pasa a 0 los bytes, y multiplicamos count por size para que ponga a 0 el total de bytes necesario.

## ft_strdup.

Esta función pasa el puntero de entrada un nuevo puntero al que se le ha reservado la memoria con malloc. Devuelve el nuevo puntero.

```
#include "libft.h"
↓
char    *ft_strdup(const char *s1)
{
    char    *ss1;
    char    *d;
    size_t  c;
↓
    ss1 = (char *)s1;
    d = (char *)malloc((ft_strlen(ss1) + 1));
    if (d == NULL)
        return (NULL);
    c = 0;
    while (ss1[c] != '\0')
    {
        d[c] = ss1[c];
        c++;
    }
    d[c] = '\0';
    return (d);
}
```

# 5. Parte 2 - Funciones adicionales.

## ft_substr.

```
#include "libft.h"
↓
char    *ft_substr(char const *s, unsigned int start, size_t len)
{
    char    *ss;
    size_t  c;
↓
    if (!s)
        return (NULL);
    if ((size_t)start > ft_strlen(s))
        return (ft_strdup (""));
    if (len > (ft_strlen(s) - start))
        len = (ft_strlen(s) - start);
    ss = (char *)malloc(sizeof(char) * (len + 1));
    if (!ss)
        return (NULL);
    c = 0;
    while (c < len)
    {
        ss[c] = *(s + start + c);
        c++;
    }
    ss[c] = '\0';
    return (ss);
}
```

Si el string de entrada no existe se devuelve NULL.

Si el valor start, que es desde donde empieza a pasarse la substring, es mayor que el tamaño de la string, se devuelve ft_strdup de nada.

Si el tamaño que se le asigna al substring, es mayor al tamaño del string menos la posición a donde va a empezar, se le asigna a len este tamaño para que no añada nada de más

Esto asigna a la posición del substring la posición del string a partir de donde debe empezar más el contador que se compara con el tamaño que va a tener.

## ft_strjoin.

```
#include "libft.h"
↓
char    *ft_strjoin(char const *s1, char const *s2)
{
    char    *str;
    size_t  c1;
    size_t  c2;
↓
    if (!s1 || !s2)
        return (0);
    str = (char *)malloc(sizeof(char) * (ft_strlen(s1) + ft_strlen(s2) + 1));
    if (!str)
        return (0);
    c1 = 0;
    while (s1[c1] != '\0')
    {
        str[c1] = s1[c1];
        c1++;
    }
    c2 = 0;
    while (s2[c2] != '\0')
    {
        str[c1] = s2[c2];
        c1++;
        c2++;
    }
    str[c1] = '\0';
    return (str);
}
```

Si alguno de los strings es NULL devuelve NULL.

## ft_strtrim.

```
#include "libft.h"
↓
char    *ft_strtrim(char const *s1, char const *set)
{
    size_t      c;
↓
    if (!s1 || !set)
        return (0);
    while (*s1 && ft_strchr(set, *s1))
        s1++;
    c = ft_strlen(s1);
    while (c && ft_strchr(set, s1[c]))
        c--;
    return (ft_substr(s1, 0, c + 1));
}
```

Si alguno de los strings es NULL devuelve NULL.

Mientras este en el string s1 y coincida el string set en la posición que está s1, suma una posición a donde apunta el string,

Mientras este en el countador c y coincida el string set en la posición que está s1, resta uno al contador c que empieza siendo el tamaño del string s1. Esto determina el tamaño que va a tener el string de salida.

Devuelve un nuevo string, que empieza a crearlo a partir de la posición 0 de s1, y el tamaño es de c más el carácter final.

## ft_split.

```c
#include "libft.h"

static size_t	ft_wordcounter(char const *s, char c)
{
	size_t	i;
	size_t	x;

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

char	**ft_split(char const *s, char c)
{
	char	**str;
	size_t	len;
	size_t	n;

	str = malloc(sizeof(char *) * (ft_wordcounter(s, c) + 1));
	if (!str)
		return (0);
	n = 0;
	while (*s)
	{
		if (*s != c)
		{
			len = 0;
			while (*s && *s != c)
			{
				++len;
				++s;
			}
			str[n++] = ft_substr(s - len, 0, len);
		}
		else
			++s;
	}
	str[n] = 0;
	return (str);
}
```

Esta función cuneta el número de palabras que hay, teniendo en cuenta que en la posición donde está el string, no es el carácter delimitador, y el siguiente carácter del string si es, o es nulo porque es la última palabra.

Entra en un bucle mientras el srting s no llege a NULL.

Si la posición del string es distinta a al carácter delimitador, entra en el if y empieza a calcular el tamaño de la palabra, y cuando se llega al carácter delimitador, se devuelve la palabra con la función substr, que devuelve a partir del primer carácter de la palabra, y devuelve solo la palabra. Si en la posición que está el string en el carácter delimitado pasa a la siguiente posición y empieza a buscar una nueva palabra.

Una vez encontradas todas las palabras, se le pone a el ultimo carácter que es el final.

## Ft_itoa.

```
#include "libft.h"
↓
static size_t	ft_intlen(long x)
{
	size_t	c;
↓
	c = 0;
	if (x == 0)
	{
		c++;
		return (c);
	}
	if (x < 0)
	{
		x = x * -1;
		c++;
	}
	while (x > 0)
	{
		x = x / 10;
		c++;
	}
	return (c);
}
↓
char	*ft_itoa(int n)
{
	char	*str;
	size_t	len;
	long	nl;
↓
	nl = n;
	len = ft_intlen(nl);
	str = (char *)malloc(sizeof(char) * (len + 1));
	if (!str)
		return (0);
	str[len--] = '\0';
	if (n == 0)
		str[0] = '0';
	if (nl < 0)
	{
		str[0] = '-';
		nl = nl * -1;
	}
	while (nl > 0)
	{
		str[len] = (nl % 10) + '0';
		nl = nl / 10;
		len--;
	}
	return (str);
}
```

Calcula el número de cifras que tiene el número. Te devuelve este valor.

Si el número introducido es igual a 0, suma 1 al contador y lo devuelve para indicar que la longitud del número es de una cifra.

SI el número es negativo, lo pasa a positivo para poder calcular la longitud.

Mientras el número sea mayor a 0, se divide entre 10 y se suma uno al contador.

Se pasa el número de entrada que es un int a un long, porque el número puede ocupar muchos bytes y así evitas fallos con números grandes.

Se define el byte final, que siempre va a ser el tamaño -1.

Si el número es 0, el primer byte será 0.

Si el número es negativo, el primer bytes es el signo -, y se pasa el número a positivo para operar con él.

Mientras el número sea mayor que 0, se saca el resto y se le suma el carácter 0 para pasarlo a char (no empieza por el carácter final porque con anterioridad a len ya se le ha restado 1).

Se divide el número entre 10 para poder seguir obtenido los demás.

## Ft_strmapi.

```
#include    "libft.h"
↓
char    *ft_strmapi(char const *s, char (*f)(unsigned int, char))
{
    size_t  n;
    char    *str;
↓
    if (s == NULL && f == NULL)
        return (NULL);
    n = ft_strlen((char *)s);
    str = (char *)malloc(sizeof(*s) * (n + 1));
    if (!str)
        return (NULL);
    n = 0;
    while (s[n] != '\0')
    {
        str[n] = f(n, s[n]);
        n++;
    }
    str[n] = '\0';
    return (str);
}
```

Si no existen ambas entradas devuelve NULL.

Se calcula el tamaño del string para reservar el espacio de memoria con malloc.

Se vuelve a poner n como 0 para poder operar con el contador.

Mientras esté recorriendo el string, añade al byte del string el resultado de la función f.

## Ft_striteri.

```
#include    "libft.h"
↓
void    ft_striteri(char *s, void (*f)(unsigned int, char*))
{
    size_t  n;
↓
    n = 0;
    if (!(s == NULL && f == NULL))
    {
        while (s[n] != '\0')
        {
            f(n, &s[n]);
            n++;
        }
    }
}
```

Si el string s y la función f son distintos a nulo, entra en el bucle, que recorre el string ejecuta la función.

## Ft_putchar_fd.

```
#include "libft.h"
↓
void    ft_putchar_fd(char c, int fd)
{
    write(fd, &c, 1);
}
```

Envía el carácter 'c' al file descriptor.

## Ft_putstr_fd.

```
#include "libft.h"
↓
void    ft_putstr_fd(char *s, int fd)
{
    write(fd, s, ft_strlen(s));
}
```

Envía la string 's' al file descriptor especificado. Para ver el tamaño del string se usa strlen.

## Ft_putendl_fd.

```
#include "libft.h"
↓
void    ft_putendl_fd(char *s, int fd
{
    char    nl;
↓
    nl = '\n';
    write(fd, s, ft_strlen(s));
    write(fd, &nl, 1);
}
```

Envía la string 's' al file descriptor dado, seguido de un salto de línea.
Para ver el tamaño del string se usa strlen.

## Ft_putnbr_fd.

```
#include "libft.h"
↓
void    ft_putnbr_fd(int n, int fd)
{
    long int    copia;
↓
    copia = n;
    if (copia < 0)
    {
        copia = (copia * -1);
        write(fd, "-", 1);
    }
    if (copia > 9)
    {
        ft_putnbr_fd(copia / 10, fd);
        ft_putchar_fd((copia % 10) + '0', fd);
    }
    else
        ft_putchar_fd(copia + '0', fd);
}
```

Se convierte el int a long int por si el número tiene muchas cifras, no tener problemas.

Si el número es negativo, se pasa a positivo para poder operar con el y se escribe en fd el signo -.

Escribe el último digito en fd.

# 6. Parte Bonus.

## Ft_lstnew.

```
#include "libft.h"
↓
t_list  *ft_lstnew(void *content)
{
    t_list  *node;
↓
    node = malloc(sizeof(t_list));
    if (node == NULL)
        return (NULL);
    node->content = content;
    node->next = NULL;
    return (node);
}
```

## Ft_lstadd_front.

```
#include "libft.h"
↓
void    ft_lstadd_front(t_list **lst, t_list *new)
{
    if (lst && new)
    {
        new->next = *lst;
        *lst = new;
    }
}
```

## Ft_lstsize.

```
#include "libft.h"
↓
int ft_lstsize(t_list *lst)
{
    size_t  c;
↓
    c = 0;
    while (lst)
    {
        lst = lst->next;
        c++;
    }
    return (c);
}
```

### Ft_lstlast.

```
#include "libft.h"
↓
t_list  *ft_lstlast(t_list *lst)↓
{↓
    if (!lst)↓
        return (NULL);↓
    while (lst)↓
    {↓
        if (lst->next == NULL)↓
            return (lst);↓
        lst = lst->next;↓
    }↓
    return (lst);↓
}↓
```

### Ft_lastadd_back.

```
#include "libft.h"↓
↓
void    ft_lstadd_back(t_list **lst, t_list *new)↓
{↓
    t_list  *last;↓
↓
    last = ft_lstlast(*lst);↓
    if (last)↓
        last->next = new;↓
    else↓
        *lst = new;↓
}↓
```

### Ft_lstdelone.

```
#include "libft.h"↓
↓
void    ft_lstdelone(t_list *lst, void (*del)(void *))↓
{↓
    if (lst)↓
    {↓
        del(lst->content);↓
        free(lst);↓
    }↓
}↓
```

## Ft_lstclear.

```
#include "libft.h"↓
↓
void    ft_lstclear(t_list **lst, void (*del)(void *))↓
{↓
    t_list  *aux;↓
↓
    if (*lst)↓
    {↓
        while (*lst)↓
        {↓
            aux = (*lst)->next;↓
            ft_lstdelone(*lst, del);↓
            *lst = aux;↓
        }↓
    }↓
    *lst = NULL;↓
}↓
```

## Ft_lsttiter.

```
#include "libft.h"↓
↓
void    ft_lstiter(t_list *lst, void (*f)(void *))↓
{↓
    if (lst)↓
    {↓
        while (lst)↓
        {↓
            f(lst->content);↓
            lst = lst->next;↓
        }↓
    }↓
}↓
```

## Ft_lstmap.

```c
#include "libft.h"

t_list  *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *))
{
    t_list  *ret;
    t_list  *aux;

    ret = 0;
    while (lst)
    {
        aux = ft_lstnew(f(lst->content));
        if (!aux)
        {
            ft_lstclear(&ret, del);
            return (0);
        }
        ft_lstadd_back(&ret, aux);
        lst = lst->next;
    }
    return (ret);
}
```