LIBFT

Este proyecto consiste en programar una librería en C.

LIBFT

Funciones Obligatorias

ft_isalpha
Descripción: Verifica si el carácter es una letra (alfabético).
Prototipo: int ft_isalpha(int c);
Parámetros: '**c**' (carácter a verificar)
Retorno: Uno si **'c'** es una letra; de lo contrario, cero.

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

ft_isdigit
Descripción: Verifica si el carácter es un dígito (0-9).
Prototipo: int ft_isdigit(int c);
Parámetros: '**c'** (carácter a verificar)
Retorno: Uno si '**c'** es un dígito; de lo contrario, cero.

```
int ft_isdigit(int c)
{
    return (c >= '0' && c <= '9');
}
```

ft_isalnum
Descripción: Verifica si el carácter es alfanumérico (letra o dígito).
Prototipo: int ft_isalnum(int c);
Parámetros: 'c' (carácter a verificar)
Retorno: Uno si '**c'** es alfanumérico; de lo contrario, cero.

```
int ft_isalnum(int c)
{
    return (ft_isalpha(c) || ft_isdigit(c));
}
```

ft_isascii
Descripción: Verifica si el carácter es ASCII (0-127).
Prototipo: int ft_isascii(int c);
Parámetros: 'c' (carácter a verificar)
Retorno: Uno si '**c'** es ASCII; de lo contrario, cero.

```
int ft_isascii(int c)
{
	return (c >= 0 && c <= 127);
}
```

ft_isprint
Descripción: Verifica si el carácter es imprimible (incluyendo espacio).
P**rototipo:** int ft_isprint(int c);
Parámetros: 'c' (carácter a verificar)
Retorno: Uno si **'c'** es imprimible; de lo contrario, cero.

```
int ft_isprint(int c)
{
	return (c >= 32 && c <= 126);
}
```

ft_strlen
Descripción: Calcula la longitud de una cadena de caracteres.
Prototipo: size_t ft_strlen(const char *str);
Parámetros: 'str' (cadena de caracteres)
Retorno: Número de caracteres en la cadena excluyendo el carácter nulo.

```
size_t ft_strlen(const char *str)
{
	size_t i;

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

LIBFT

Funciones Obligatorias

ft_memset
Descripción: Llena un bloque de memoria con un valor constante.
Prototipo: void *ft_memset(void *b, int c, size_t len);
Parámetros: 'b' (puntero al bloque de memoria), '**c'** (valor a establecer), **'len'** (número de bytes)
Retorno: Puntero al bloque de memoria **'b'**.

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

	i = 0;
 while (i < len)
 {
	((unsigned char *)b)[i] = (unsigned char)c;
	 i++;
 }
 return (b);
}
```

ft_bzero
Descripción: Inicializa a cero un bloque de memoria.
Prototipo: void ft_bzero(void *s, size_t n);
Parámetros: 's' (puntero al bloque de memoria), **'n'** (número de bytes)
Retorno: Ninguno.

```
void ft_bzero(void *s, size_t n)
{
	size_t i;

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

ft_memcpy
Descripción: Copia un bloque de memoria a otro.
Prototipo: void *ft_memcpy(void *dest, const void *src, size_t n);
Parámetros: 'dst' (puntero al destino), **'src'** (puntero al origen), **'n'** (número de bytes)
Retorno: Puntero al destino **'dst'.**

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

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

ft_memmove

Descripción: Copia un bloque de memoria a otro, asegurando que las áreas se pueden superponer.

Prototipo: void *ft_memmove(void *dest, const void *src, size_t n);

Parámetros: 'dst' (puntero al destino), **'src'** (puntero al origen), **'len'** (número de bytes)

Retorno: Puntero al destino **'dst.'**

```
void *ft_memmove(void *dst, const void *src, size_t len)
{
      size_t i;

      i = 0;
 if (src == dst)
 {
      return (dst);
 }
 if (dst < src)
 {
      while (i < len)
      {
           ((unsigned char *)dst)[i] = ((unsigned char *)src)[i];
           i++;
       }
 }
 if (dst > src)
 {
      while (len > 0)
      {
          ((unsigned char *)dst)[len -1] = ((unsigned char *)src)[len -1];
          len--;
       }
 }
 return (dst);
}
```

ft_strlcpy
Descripción: Copia una cadena a otra asegurando que el resultado sea nulo y evitando desbordamientos de buffer.
Prototipo: size_t ft_strlcpy(char *dest, const char *src, size_t size);
Parámetros: 'dest' (puntero al buffer de destino), **'src'**(puntero a la cadena fuente), **'size'** (tamaño del buffer de destino)
Retorno: Longitud total de la cadena que se intentó crear (longitud de '**src**').

```
size_t ft_strlcpy(char *dest, const char *src, size_t size)
{
	size_t i;

	i = 0;
 if (size > 0)
 {
	 while (src[i] && i < (size -1))
	 {
		dest[i] = src[i];
		i++;
	  }
	  dest[i] = 0;
 }
 return (ft_strlen(src));
}
```

ft_strlcat
Descripción: Concatena dos cadenas asegurando que el resultado sea nulo y evitando desbordamientos de buffer.
Prototipo: size_t ft_strlcat(char *dst, const char *src, size_t size);
Parámetros: 'dst' (puntero al buffer de destino), **'src'** (puntero a la cadena fuente), **'size'** (tamaño del buffer de destino)
Retorno: Longitud total de la cadena que se intentó crear (longitud de **'dst'** + longitud de **'src'**).

```
size_t ft_strlcat(char *dst, const char *src, size_t size)
{
    size_t len_d;
    size_t len_s;
    size_t i;

    len_d = ft_strlen(dst);
    len_s = ft_strlen(src);
    if (len_d >= size)
    {
        return (len_s + size);
    }
    else if (len_d + 1 < size)
    {
         i = 0;
         while (src[i] && (len_d + i + 1 < size))
    {
    dst [len_d + i] = src[i];
    i++;
    }
    dst [len_d + i] = 0;
    }
 return (len_d + len_s);
}
```

ft_toupper
Descripción: Convierte un carácter a mayúscula.
Prototipo: int ft_toupper(int c);
Parámetros: ‘c’ (carácter a convertir)
Retorno: El carácter convertido a mayúscula si es una letra minúscula; de lo contrario, devuelve **‘c’** sin cambios.

```
int ft_toupper(int c)
{
    if (c >= 'a' && c <= 'z')
    {
        c = c -32;
    }
    return (c);
}
```

ft_tolower
Descripción: Convierte un carácter a minúscula.
Prototipo: int ft_tolower(int c);
Parámetros: ‘c’ (carácter a convertir)
Retorno: El carácter convertido a minúscula si es una letra mayúscula; de lo contrario, devuelve **‘c’** sin cambios.

```
int ft_tolower(int c)
{
    if (c >= 'A' && c <= 'Z')
    {
        c = c + 32;
    }
 return (c);
}
```

ft_strchr
Descripción: Encuentra la primera aparición de un carácter en una cadena.
Prototipo: char *ft_strchr(const char *str, int c);
Parámetros: 'str' (cadena de caracteres), **'c'** (carácter a buscar)
Retorno: Un puntero a la primera aparición del carácter **'c'** en la cadena **'str'**, o **'NULL'** si el carácter no se encuentra.

```
char *ft_strchr(const char *str, int c)
{
    while (*str != (char)c)
    {
        if (*str == 0)
        {
            return (NULL);
        }
        str++;
    }
    return ((char *)str);
}
```

ft_strrchr
Descripción: Encuentra la última aparición de un carácter en una cadena.
Prototipo: char *ft_strrchr(const char *str, int c);
Parámetros: '**str**' (cadena de caracteres), **'c'** (carácter a buscar)
Retorno: Un puntero a la última aparición del carácter **'c'** en la cadena '**str**', o '**NULL**' si el carácter no se encuentra.

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

	i = ft_strlen(str) + 1;
	while (i--)
	{
		if (str[i] == (char)c)
		{
			return ((char *) &str[i]);
		}
	 }
	 return (NULL);
}
```

ft_strncmp
Descripción: Compara dos cadenas hasta un número especificado de caracteres.
Prototipo: int ft_strncmp(const char *s1, const char *s2, size_t size);
Parámetros: 's1' (primera cadena), **'s2'** (segunda cadena), **'size'** (número máximo de caracteres a comparar)
Retorno: Un entero menor que, igual a, o mayor que cero si **'s1'** es menor que, igual a, o mayor que **'s2'**, respectivamente.

```
int ft_strncmp(const char *s1, const char *s2, size_t size)
{
	size_t i;

	i = 0;
	while (size--)
	{
		if (s1[i] != s2[i] || s1[i] == 0 || s2[i] == 0 || size <= 0)
		{
			return ((unsigned char)s1[i] - (unsigned
char)s2[i]);
		}
		i++;
	}
	return (0);
}
```

ft_memchr
Descripción: Encuentra la primera aparición de un carácter en un bloque de memoria.
Prototipo: void *ft_memchr(const void *s, int c, size_t n);
Parámetros: 's' (puntero al bloque de memoria), **'c'** (carácter a buscar), **'n'** (número de bytes a examinar)
Retorno: Un puntero a la primera aparición del carácter **'c'** en el bloque de memoria **'s',** o **'NULL'** si el carácter no se encuentra.

```
void *ft_memchr(const void *s, int c, size_t n)
{
	char *str;
	char target;
	size_t i;

	str = ((char *)s);
	target = ((char)c);
	i = 0;
	while (i < n)
	{
		if (str[i] != target)
		{
			i++;
		}
		else
		{
			return ((char *)&s[i]);
		}
	}
	return (0);
}
```

ft_memcmp
Descripción: Compara dos bloques de memoria.
Prototipo: int ft_memcmp(const void *s1, const void *s2, size_t n);
Parámetros: 's1' (primer bloque de memoria), **'s2'** (segundo bloque de memoria), **'n'** (número de bytes a comparar)
Retorno: Un entero menor que, igual a, o mayor que cero si **'s1'** es menor que, igual a, o mayor que **'s2'**, respectivamente.

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

	 i = 0;
	str1 = (unsigned char *)s1;
	str2 = (unsigned char *)s2;
	while (i < n)
	{
	    if (str1[i] != str2[i])
	    {
	        	return (str1[i] - str2[i]);
	    }
	    i++;
	}
	return (0);
}
```

ft_strnstr

Descripción: Encuentra la primera aparición de una subcadena en otra cadena, limitando la búsqueda a un número específico de caracteres.

Prototipo: char *ft_strnstr(const char *haystack, const char *needle, size_t len);

Parámetros: 'haystack' (cadena en la que buscar), **'needle'** (subcadena a buscar), **'len'** (número máximo de caracteres a buscar)

Retorno: Un puntero a la primera aparición de **'needle'** en **'haystack'**, o **'NULL'** si la subcadena no se encuentra dentro de los primeros **'len'** caracteres.

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

	if (*needle == 0)
		return ((char *)haystack);
	needle_len = ft_strlen(needle);
	while (*haystack && len >= needle_len)
	{
		i = 0;
		while (haystack[i] == needle[i] && haystack[i] && needle[i])
			i++;
		if (i == needle_len)
			return ((char *)haystack);
		haystack++;
		len--;
	}
	return (NULL);
}
```

ft_atoi
Descripción: Convierte una cadena de caracteres a un entero.
P**rototipo:** int ft_atoi(const char *str);
Parámetros: 'str' (cadena de caracteres que representa un número entero)
Retorno: El valor entero convertido a partir de la cadena, o cero si la conversión no es posible.

```
int ft_atoi(const char *str)
{
	long int i;
	long int sign;
	long int number;

	i = 0;
	sign = 1;
	number = 0;
	while (str[i] == 32 || (str[i] >= 9 && str[i] <= 13))
		i++;
	if (str[i] == '-' || str[i] == '+')
	{
		if (str[i] == '-')
			sign *= -1;
			i++;
	}
	while (str[i] != '\0' && str[i] >= '0' && str[i] <= '9')
	{
		number = number * 10 + (str[i] - 48);
		i++;
	}
	number *= sign;
	return (number);
}
```

ft_calloc
Descripción: Asigna memoria para un arreglo de elementos y la inicializa a cero.
Prototipo: void *ft_calloc(size_t count, size_t size);
Parámetros: 'count' Número de elementos a asignar, '**size'** Tamaño en bytes de cada elemento.
Retorno: Un puntero al bloque de memoria asignado, que se inicializa a cero. Si no se puede asignar la memoria, devuelve **'NULL'.**

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

	ptr = malloc(count * size);
	if (ptr == 0 || size < 0)
		return (0);
	ft_bzero (ptr, (count * size));
	return (ptr);
}
```

ft_strdup
Descripción: Duplica una cadena de caracteres, asignando memoria para la nueva cadena.
Prototipo: char *ft_strdup(const char *s1);
Parámetros: 's1' Cadena de caracteres a duplicar.
Retorno: Un puntero a una nueva cadena, que es una copia de **'s1'**. La memoria para la nueva cadena se asigna con malloc. Si no se puede asignar la memoria, devuelve **'NULL'.**

```
char *ft_strdup(const char *s1)
{
	char *string;
	int  i;

	i = 0;
	string = (char *)malloc(sizeof(char) * ft_strlen(s1) +1);
	if (!string)
		return (NULL);
	while (s1[i] != 0)
	{
		string[i] = s1[i];
		i++;
	 }
	 string[i] = 0;
	 return (string);
}
```

ft_substr

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

Parámetros:

‘**s’:** La string desde la que crear la substring.

‘**start’**: El índice del carácter en **‘s’** desde el que empezar la substring.

‘**len’**: La longitud máxima de la substring.

Valor devuelto: La substring resultante. **‘NULL’** si falla la reserva de memoria.

Descripción: Reserva (con malloc) y devuelve una substring de la string **‘s’**. La substring empieza desde el índice **‘start’** y tiene una longitud máxima **‘len’.**

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

	if (s == NULL)
		return (NULL);
	s_len = ft_strlen(s);
	if (start >= s_len)
	{
		substr = (char *)malloc(1);
		if (substr == NULL)
			return (NULL);
		substr[0] = 0;
		return (substr);
	 }
	 if (len > s_len - start)
	 {
		len = s_len - start;
	 }
	substr = (char *)malloc(len + 1);
	if (substr == NULL)
	{
		 return (NULL);
	}
	ft_strlcpy(substr, s + start, len + 1);
	return (substr);
}
```

ft_strjoin

Prototipo: char *ft_strjoin(char const *s1, char const *s2);
Parámetros:
's1': La primera string.
'**s2'**: La string a añadir a **'s1'**.
Valor devuelto: La nueva string. **'NULL'** si falla la reserva de memoria.
Descripción: Reserva (con malloc) y devuelve una nueva string, formada por la concatenación de **'s1'** y **'s2'**.

```
char *ft_strjoin(char const *s1, char const *s2)
{
	char *str;
	int  i;
	int  j;
	int  len1;
	int  len2;

	len1 = ft_strlen(s1);
	len2 = ft_strlen(s2);
	str = (char *)malloc(((len1 + len2) * sizeof(char)) + 1);
	if (str == 0)
	{
		return (NULL);
	}
	i = -1;
	while (s1[++i])
	{
		str[i] = s1[i];
	}
	j = 0;
	while (s2[j])
	{
		str[i++] = s2[j++];
	}
	str[i] = 0;
	return (str);
}
```

ft_strtrim
Prototipo: char *ft_strtrim(char const *s1, char const *set);
Parámetros:
‘**s1':** La string que debe ser recortada.
‘**set’:** Los caracteres a eliminar de la string.
Valor devuelto: La string recortada. **‘NULL’** si falla la reserva de memoria.
Descripción: 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.

```
char *ft_strtrim(const char *s1, const char *set)
{
	char *ptr;
	int  start;
	int  end;

	start = 0;
	if (s1[start] == 0)
		return (ft_strdup(""));
	end = ft_strlen(s1);
	while (ft_strchr (set, s1[start]))
		start++;
	while (ft_strchr (set, s1[end]))
		end--;
	ptr = ft_substr (s1, start, (end - start) + 1);
	return (ptr);
}
```

ft_split
Prototipo: char **ft_split(char const *str, char c);
Parámetros:
‘**str’:** La string a separar.
‘**c’:** El carácter delimitador.
Valor devuelto: El array de nuevas strings resultante de la separación. ‘**NULL’** si falla la reserva de memoria.
Descripción: Reserva (utilizando malloc) un array de strings resultante de separar la string ‘**str’** en substrings utilizando el carácter ‘**c’** como delimitador. El array debe terminar con un puntero ‘**NULL**.’

```
static char **ft_free(char **str, int i)
{
	while (--i >= 0)
		free(str[i]);
	free(str);
	return (NULL);
}
```

```
static int ft_countwords(const char *str, char c)
{
	int i;
	int words;

	 i = 0;
	words = 0;
	while (str[i] != '\0')
	{
		if (str[i] != c && (str[i + 1] == c || str[i + 1] == '\0'))
			words++;
		i++;
	 }
	 return (words);
}
```

```
 static int ft_wordlen(const char *str, char c)
 {
	 int len;

	 len = 0;
	 while (str[len] && str[len] != c)
		 len++;
	 return (len);
 }
```

```
char **ft_split(const char *str, char c)
{
	char **array;
	int  i;
	int  start;
	int  words;

	if (!str)
		return (NULL);
	words = ft_countwords(str, c);
	array = (char **)malloc(sizeof(char *) * (words + 1));
	if (!array)
		return (NULL);
	i = 0;
	start = 0;
	while (i < words)
	{
		while (str[start] == c)
			start++;
		array[i] = ft_substr(str, start, ft_wordlen(str + start, c));
		if (!array[i])
			return (ft_free(array, i));
		start += ft_wordlen(str + start, c);
		i++;
	}
	array[i] = NULL;
	return (array);
}
```

ft_itoa
Prototipo: char *ft_itoa(int n);
Parámetros:
'n': El entero a convertir.
Valor devuelto: La string que representa el número. '**NULL**' si falla la reserva de memoria.
Descripción: Utilizando malloc, genera una string que representa el valor entero recibido como argumento. Los números negativos tienen que gestionarse.

```
static  int ft_sizenum(long number)
{
	int i;

	i = 1;
	if (number < 0)
	{
		i++;
		number = number * -1;
	 }
	 while (number > 9)
	 {
		 number = number / 10;
		 i++;
	  }
	 return (i);
}
```

```
char *ft_itoa(int n)
{
	char *string;
	int  size;
	long long_n;

	long_n = (long)n;
	size = ft_sizenum(long_n);
	string = (char *)malloc(sizeof(char) * size + 1);
	if (!string)
		return (NULL);
	string[size] = '\0';
	if (long_n < 0)
	{
		string[0] = '-';
		long_n = long_n * -1;
	}
	if (long_n == 0)
		string[0] = '0';
	 while (long_n > 9)
	 {
		string[--size] = (long_n % 10) + '0';
		long_n /= 10;
	 }
	 if (long_n > 0)
		string[--size] = long_n + '0';
	return (string);
}
```

ft_strmapi
Prototipo: char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
Parámetros:
‘**s’:** La string que iterar.
‘**f ’:** La función a aplicar sobre cada carácter.
Valor devuelto: La string creada tras el correcto uso de ‘ **f ’** sobre cada carácter. ‘**NULL’** si falla la reserva de memoria.
Descripción: 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 ‘.**

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

	i = 0;
	string = (char *) malloc((ft_strlen(s) + 1) * sizeof(char));
	if (string == 0)
		return (NULL);
	while (s[i] != 0)
	{
		string[i] = (*f)(i, s[i]);
		i++;
	 }
	 string[i] = 0;
	 return (string);
}
```

ft_striteri
Prototipo: void ft_striteri(char *s, void (*f)(unsigned int, char*));
Parámetros:
‘**s’:** La string que iterar.
‘**f ‘:** La función a aplicar sobre cada carácter.
Valor devuelto: Nada.
Descripción: 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.

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

	i = 0;
	while (s[i] != 0)
	{
		(*f)(i, &s[i]);
		i++;
	}
}
```

ft_putchar_fd
Prototipo: void ft_putchar_fd(char c, int fd);
Parámetros:
'**c':** El carácter a enviar.
'**fd':** El file descriptor sobre el que escribir.
Valor devuelto: Nada.
Descripción: Envía el carácter '**c'** al **file descriptor** especificado.

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

ft_putstr_fd
Prototipo: void ft_putstr_fd(char *s, int fd);
Parámetros:
'**s':** La string a enviar.
'**fd':** El file descriptor sobre el que escribir.
Valor devuelto: Nada.
Descripción: Envía la string '**s'** al **file descriptor** especificado.

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

	i = 0;
	while (s[i] != 0)
	{
		write(fd, &s[i], 1);
		i++;
	 }
 }
```

ft_putendl_fd
Prototipo: void ft_putendl_fd(char *s, int fd);
Parámetros:
‘**s’:** La string a enviar.
‘**fd’:** El **file descriptor** sobre el que escribir.
Valor devuelto: Nada.
Descripción: Envía la string **‘s’** al **file descriptor** dado, seguido de un salto de línea.

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

	i = 0;
	while (s[i] != 0)
	{
		write(fd, &s[i], 1);
		i++;
	}
	write(fd, "\n", 1);
}
```

ft_putnbr_fd
Prototipo: void ft_putnbr_fd(int n, int fd);
P**arámetros:**
‘**n’:** El número que enviar.
‘**fd’:** El **file descriptor** sobre el que escribir.
Valor devuelto: Nada.
Descripción: Envía el número **‘n’** al **file descriptor** dado.

```
void ft_putnbr_fd(int n, int fd)
{
	if (n == -2147483648)
	{
		ft_putchar_fd('-', fd);
		ft_putchar_fd('2', fd);
		ft_putnbr_fd(147483648, fd);
	 }
	 else if (n > 9)
	 {
		ft_putnbr_fd(n / 10, fd);
		ft_putchar_fd(n % 10 + '0', fd);
	 }
	 else if (n < 0)
	 {
		ft_putchar_fd('-', fd);
		ft_putnbr_fd(-n, fd);
	 }
	 else
	 {
		ft_putchar_fd(n + '0', fd);
	 }
}
```

ft_lstnew
Prototipo: t_list *ft_lstnew(void *content);
Parámetros:
'**content':** El contenido con el que crear el nodo.
Valor devuelto: El nuevo nodo.
Descripción: Crea un nuevo nodo utilizando malloc. La variable miembro '**content'** se inicializa con el contenido del parámetro '**content'**. La variable '**next'**, con '**NULL'**.

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

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

ft_lstadd_front
Prototipo: void ft_lstadd_front(t_list **lst, t_list *new);
Parámetros:
'**lst':** La dirección de un puntero al primer nodo de una lista.
'**new':** Un puntero al nodo que añadir al principio de la lista.
Valor devuelto: Nada.
Descripción: Añade el nodo '**new'** al principio de la lista '**lst'.**

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

ft_lstsize

Prototipo: int ft_lstsize(t_list *lst);
Parámetros:
‘lst’: El principio de la lista.
Valor devuelto: La longitud de la lista.
Descripción: Cuenta el número de nodos de una lista.

```
int ft_lstsize(t_list *lst)
{
	size_t i;

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

ft_lstlast
Prototipo: t_list *ft_lstlast(t_list *lst);
Parámetros:
‘**lst’:** El principio de la lista.
Valor devuelto: El último nodo de la lista.
Descripción: Devuelve el último nodo de la lista.

```
t_list *ft_lstlast(t_list *lst)
{
	if (!lst)
		return (NULL);
	while (lst->next)
	lst = lst->next;
	return (lst);
}
```

ft_lstadd_back
Prototipo: void ft_lstadd_back(t_list **lst, t_list *new);
Parámetros:
‘**lst’:** El puntero al primer nodo de una lista.
‘**new’:** El puntero a un nodo que añadir a la lista.
Valor devuelto: Nada.
Descripción: Añade el nodo ‘**new’** al final de la lista ‘**lst’.**

```
void ft_lstadd_back(t_list **lst, t_list *new)
{
	if (!lst || !new)
	return ;
	if (*lst)
		(ft_lstlast(*lst))->next = new;
	else
	*lst = new;
}
```

ft_lstdelone

Prototipo: void ft_lstdelone(t_list *lst, void (*del)(void *));
Parámetros:
'l**st':** El nodo a liberar.
'**del':** Un puntero a la función utilizada para liberar el contenido del nodo.
Valor devuelto: Nada.
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.

```
void ft_lstdelone(t_list *lst, void (*del)(void *))
{
	if (!lst || !del)
		return ;
	if (lst && del)
	{
		del (lst->content);
		free(lst);
	 }
}
```

ft_lstclear

Prototipo: void ft_lstclear(t_list **lst, void (*del)(void *));
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.
Descripción: Elimina y libera el nodo ‘**lst’** dado y todos los consecutivos de ese nodo, utilizando la función ‘**del’** y ‘**free’**. Al final, el puntero a la lista debe ser ‘**NULL’**.

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

	if (!lst || !del)
	return ;
	while (*lst)
	{
		aux = (*lst)->next;
		ft_lstdelone(*lst, del);
		*lst = aux;
	}
}
```

ft_lstiter
Prototipo: void ft_lstiter(t_list *lst, void (*f)(void *));
Parámetros:
lst: Un puntero al primer nodo.
f: Un puntero a la función que utilizará cada nodo.
Valor devuelto: Nada.
Descripción: Itera la lista **lst** y aplica la función **f** en el contenido de cada nodo.

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

ft_lstmap

Prototipo: t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));
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.
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.

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

	new_lst = NULL;
	while (lst)
	{
		new_node = ft_lstnew(NULL);
		if (!new_node)
		{
			ft_lstclear(&new_lst, del);
			return (NULL);
		}
		new_node ->content = f(lst->content);
		ft_lstadd_back(&new_lst, new_node);
		lst = lst->next;
	}
	return (new_lst);
}
```

Notas

Notas

Notas