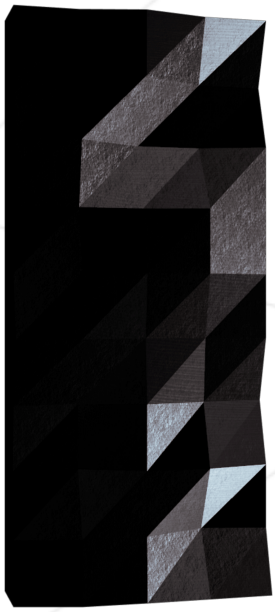

Libft

Tu primera librería

Resumen: Este proyecto consiste en programar una librería en `C`*.*
Tu librería tendrá un montón de funciones de propósito general en las que se apoyarán tus programas.

Versión: 16.2

Índice general

Capítulo I

Introducción

Programar en `C` puede ser aburrido cuando uno no tiene acceso a las funciones comunes más utilizadas. Este proyecto te permitirá entender la forma en la que estas funciones funcionan, cómo implementarlas y aprender a utilizarlas. Crearás una librería propia, que será muy útil ya que la utilizarás en los siguientes proyectos de `C`.

Asegúrate de ir enriqueciendo tu `libft` a lo largo de tu cursus. Sin embargo, cuando utilices tu librería asegúrate de que todas las funciones utilizadas por tu librería respetan las permitidas por cada proyecto.

Capítulo II

Instrucciones generales

- Tu proyecto deberá estar escrito en C.
- Tu proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas están incluidas en la verificación de la Norma y tendrás un `0` si hay algún error de norma en cualquiera de ellos.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) excepto en el caso de comportamientos indefinidos. Si esto sucede, tu proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria asignada en el heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el enunciado lo requiere, deberás entregar un `Makefile` que compilará tus archivos fuente al output requerido con las flags `-Wall`, `-Werror` y `-Wextra`, utilizar cc y por supuesto tu `Makefile` no debe hacer relink.
- Tu `Makefile` debe contener al menos las normas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los bonus de tu proyecto deberás incluir una regla `bonus` en tu `Makefile`, en la que añadirás todos los headers, librerías o funciones que estén prohibidas en la parte principal del proyecto. Los bonus deben estar en archivos distintos `_bonus.{c/h}`. La parte obligatoria y los bonus se evalúan por separado.
- Si tu proyecto permite el uso de la `libft`, deberás copiar su fuente y sus `Makefile` asociados en un directorio `libft` con su correspondiente `Makefile`. El `Makefile` de tu proyecto debe compilar primero la librería utilizando su `Makefile`, y después compilar el proyecto.
- Te recomendamos crear programas de prueba para tu proyecto, aunque este trabajo **no será entregado ni evaluado**. Te dará la oportunidad de verificar que tu programa funciona correctamente durante tu evaluación y la de otros compañeros. Y sí, tienes permitido utilizar estas pruebas durante tu evaluación o la de otros compañeros.
- Entrega tu trabajo en tu repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus com-

pañeros. Si se encuentra un error durante la evaluación de Deepthought, esta habrá terminado.

Capítulo III

Parte obligatoria

Nombre de programa	`libft.a`
Archivos a entregar	`Makefile, libft.h, ft_*.c`
Makefile	`NAME, all, clean, fclean, re`
Funciones autorizadas	`Detalles debajo`
Se permite usar libft	`todavía no la tienes`
Descripción	`Escribe tu propia librería: un conjunto de funciones que será una herramienta muy útil a lo largo del cursus.`

III.1. Consideraciones técnicas

- *Declarar* variables globales está prohibido.
- Si necesitas separar una función compleja en varias, asegúrate de utilizar la palabra `static` para ello. De esta forma, las funciones se quedarán en el archivo apropiado.
- Pon todos tus archivos en la raíz de tu repositorio.
- Se prohibe entregar archivos no utilizados.
- Todos los archivos `.c` deben compilarse con las flags `-Wall -Werror -Wextra`.
- Debes utilizar el comando `ar` para generar la librería. El uso de `libtool` queda prohibido.
- Tu `libft.a` tiene que ser creado en la raíz del repositorio.

III.2. Parte 1 - Funciones de libc

Para empezar, deberás rehacer algunas funciones de la `libc`. Tus funciones tendrán los mismos prototipos e implementarán los mismos comportamientos que las funciones originales. Deberán ser tal y como las describe el `man`. La única diferencia será la nomenclatura. Empezarán con el prefijo `"ft_"`. Por ejemplo, `strlen` se convertirá en `ft_strlen`.

Algunas funciones tienen en sus prototipos la palabra "restrict". Esta palabra forma parte del estándar de c99. Por lo tanto, está prohibido incluirla en tus propios prototipos, así como compilar tu código con la flag -std=c99.

Deberás escribir tus propias funciones implementando las siguientes funciones originales. No requieren de funciones autorizadas:

- isalpha
- isdigit
- isalnum
- isascii
- isprint
- strlen
- memset
- bzero
- memcpy
- memmove
- strlcpy
- strlcat
- toupper
- tolower
- strchr
- strrchr
- strncmp
- memchr
- memcmp
- strnstr
- atoi

Para implementar estas otras dos funciones, tendrás que utilizar `malloc()`:

- calloc
- strdup

Dependiendo de tu sistema operativo actual, la página del manual de calloc y el comportamiento de la función pueden diferir. La siguiente instrucción sustituye lo que puedes encontrar en la página del manual: Si nmemb o size es 0, entonces calloc() devuelve un valor de puntero único que más tarde puede pasarse con éxito a free().

III.3. Parte 2 - Funciones adicionales

En esta segunda parte, deberás desarrollar un conjunto de funciones que, o no son de la librería `libc`, o lo son pero de una forma distinta.

Algunas de las siguientes funciones pueden ser útiles para hacer las funciones de la parte 1.

Nombre de función	ft_substr
Prototipo	char *ft_substr(char const *s, unsigned int start, size_t len);
Archivos a entregar	-
Parámetros	s: La string desde la que crear la substring. start: El índice del caracter 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.
Funciones autorizadas	malloc
Descripción	Reserva (con malloc(3)) y devuelve una substring de la string 's'. La substring empieza desde el índice 'start' y tiene una longitud máxima 'len'.

Nombre de función	ft_strjoin
Prototipo	char *ft_strjoin(char const *s1, char const *s2);
Archivos a entregar	-
Parámetros	s1: La primera string. s2: La string a añadir a 's1'.
Valor devuelto	La nueva string. NULL si falla la reserva de memoria.
Funciones autorizadas	malloc
Descripción	Reserva (con malloc(3)) y devuelve una nueva string, formada por la concatenación de 's1' y 's2'.

Nombre de función	`ft_strtrim`
Prototipo	`char *ft_strtrim(char const *s1, char const *set);`
Archivos a entregar	-
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.
Funciones autorizadas	`malloc`
Descripción	Elimina todos los caracteres de la string 'set' desde el principio y desde el final de 's1', hasta encontrar un caracter no perteneciente a 'set'. La string resultante se devuelve con una reserva de `malloc(3)`

Nombre de función	`ft_split`
Prototipo	`char **ft_split(char const *s, char c);`
Archivos a entregar	-
Parámetros	`s`: 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.
Funciones autorizadas	`malloc, free`
Descripción	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`.

Nombre de función	ft_itoa
Prototipo	char *ft_itoa(int n);
Archivos a entregar	-
Parámetros	n: el entero a convertir.
Valor devuelto	La string que represente el número. NULL si falla la reserva de memoria.
Funciones autorizadas	malloc
Descripción	Utilizando malloc(3), genera una string que represente el valor entero recibido como argumento. Los números negativos tienen que gestionarse.

Nombre de función	ft_strmapi
Prototipo	char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
Archivos a entregar	-
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.
Funciones autorizadas	malloc
Descripción	Aplica la función 'f' a cada carácter de la cadena 's', pasando su índice como primer argumento y el propio carácter como segundo argumento. Se crea una nueva cadena (utilizando malloc(3)) para recoger los resultados de las sucesivas aplicaciones de 'f'.

Nombre de función	ft_striteri
Prototipo	void ft_striteri(char *s, void (*f)(unsigned int, char*));
Archivos a entregar	-
Parámetros	s: La string que iterar. f: La función a aplicar sobre cada carácter.
Valor devuelto	Nada
Funciones autorizadas	Ninguna
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.

Nombre de función	ft_putchar_fd
Prototipo	void ft_putchar_fd(char c, int fd);
Archivos a entregar	-
Parámetros	c: El carácter a enviar. fd: El file descriptor sobre el que escribir.
Valor devuelto	Nada
Funciones autorizadas	write
Descripción	Envía el carácter 'c' al file descriptor especificado.

Nombre de función	ft_putstr_fd
Prototipo	void ft_putstr_fd(char *s, int fd);
Archivos a entregar	-
Parámetros	s: La string a enviar. fd: El file descriptor sobre el que escribir.
Valor devuelto	Nada
Funciones autorizadas	write
Descripción	Envía la string 's' al file descriptor especificado.

Nombre de función	`ft_putendl_fd`
Prototipo	`void ft_putendl_fd(char *s, int fd);`
Archivos a entregar	-
Parámetros	`s: La string a enviar.` `fd: El file descriptor sobre el que escribir.`
Valor devuelto	`Nada`
Funciones autorizadas	`write`
Descripción	`Envía la string 's' al file descriptor dado,` `seguido de un salto de línea.`

Nombre de función	`ft_putnbr_fd`
Prototipo	`void ft_putnbr_fd(int n, int fd);`
Archivos a entregar	-
Parámetros	`n: El número que enviar.` `fd: El file descriptor sobre el que escribir.`
Valor devuelto	`Nada`
Funciones autorizadas	`write`
Descripción	`Envía el número 'n' al file descriptor dado.`

Capítulo IV

Parte bonus

Si completas la parte obligatoria, no dudes en llevarla más lejos haciendo esta parte extra. Te dará puntos adicionales si la completas correctamente.

Las funciones para manipular memoria y strings son muy útiles... Pero pronto descubrirás que la manipulación de listas lo es incluso más.

Deberás utilizar la siguiente estructura para representar un nodo de tu lista. Añade la declaración a tu archivo `libft.h`:

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

Los miembros de la estructura `t_list` son:

- `content`: la información contenida por el nodo.
  `void *`: permite guardar cualquier tipo de información.
- `next`: la dirección del siguiente nodo, o `NULL` si el siguiente nodo es el último.

En tu Makefile, añade una regla `make bonus` que incorpore las funciones bonus a tu `libft.a`.

La parte bonus será exclusivamente evaluada si la parte obligatoria está perfecta. ¿Perfecta? Sí: todos los requisitos de la parte obligatoria deben estar correctamente completados. De otro modo, tus bonus no serán evaluados en absoluto.

Implementa las siguientes funciones para utilizar fácilmente tus listas.

Nombre de función	`ft_lstnew`
Prototipo	`t_list *ft_lstnew(void *content);`
Archivos a entregar	-
Parámetros	`content: el contenido con el que crear el nodo.`
Valor devuelto	`El nuevo nodo`
Funciones autorizadas	`malloc`
Descripción	`Crea un nuevo nodo utilizando malloc(3). La variable miembro 'content' se inicializa con el contenido del parámetro 'content'. La variable 'next', con NULL.`

Nombre de función	`ft_lstadd_front`
Prototipo	`void ft_lstadd_front(t_list **lst, t_list *new);`
Archivos a entregar	-
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`
Funciones autorizadas	`Ninguna`
Descripción	`Añade el nodo 'new' al principio de la lista 'lst'.`

Nombre de función	`ft_lstsize`
Prototipo	`int ft_lstsize(t_list *lst);`
Archivos a entregar	-
Parámetros	`lst: el principio de la lista.`
Valor devuelto	`La longitud de la lista.`
Funciones autorizadas	`Ninguna`
Descripción	`Cuenta el número de nodos de una lista.`

Nombre de función	ft_lstlast
Prototipo	t_list *ft_lstlast(t_list *lst);
Archivos a entregar	-
Parámetros	lst: el principio de la lista.
Valor devuelto	Último nodo de la lista.
Funciones autorizadas	Ninguna
Descripción	Devuelve el último nodo de la lista.

Nombre de función	ft_lstadd_back
Prototipo	void ft_lstadd_back(t_list **lst, t_list *new);
Archivos a entregar	-
Parámetros	lst: el puntero al primer nodo de una lista. new: el puntero a un nodo que añadir a la lista.
Valor devuelto	Nada
Funciones autorizadas	Ninguna
Descripción	Añade el nodo 'new' al final de la lista 'lst'.

Nombre de función	ft_lstdelone
Prototipo	void ft_lstdelone(t_list *lst, void (*del)(void *));
Archivos a entregar	-
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.

Nombre de función	`ft_lstclear`
Prototipo	`void ft_lstclear(t_list **lst, void (*del)(void *));`
Archivos a entregar	-
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.`

Nombre de función	`ft_lstiter`
Prototipo	`void ft_lstiter(t_list *lst, void (*f)(void *));`
Archivos a entregar	-
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.`

Nombre de función	`ft_lstmap`
Prototipo	`t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));`
Archivos a entregar	-

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.`

Capítulo V

Entrega y evaluación

Entrega tu proyecto en tu repositorio `Git` como de costumbre. Solo el trabajo entregado en el repositorio será evaluado durante la defensa. No dudes en comprobar varias veces los nombres de los archivos para verificar que sean correctos.

Deja todos tus archivos en la raíz del repositorio.

Rnpu cebwrpg bs gur 97 Pbzzba Pber pbagnvaf na rapbqrq uvag. Sbe rnpu pvepyr, bayl bar cebwrpg cebivqrf gur pbeerpg uvag arrqrq sbe gur arkg pvepyr. Guvf punyyratr vf vaqvivqhny, gurer vf bayl n cevmr sbe bar fghqrag jvaare cebivqvat nyy qrpbqrq zrffntrf. Nal nqinagntrq crbcyr pna cynl, yvxr pheerag be sbezre fgnss, ohg gur cevmr jvyy erznva flzobyvp. Gur uvag sbe guvf svefg cebwrpg vf:
Ynetr pbjf trarebfvgl pbzrf jvgu punegf naq sbhe oybaqr ungf gb qrsl hccre tenivgl ureb