

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

Índice general

I.	In	ntroducción 2
II.	In	nstrucciones generales 3
III.	Pa	arte obligatoria 5
I	II.1.	Consideraciones técnicas
Ι	II.2.	Parte 1 - Funciones de libc
I	II.3.	Parte 2 - Funciones adicionales
IV.	Pa	arte bonus 12
V.	Eı	ntrega y evaluación 17

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) ni tener comportamientos indefinidos. Si esto pasa 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 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-

Libft	Tu primera librería
pañeros. Si se en terminado.	ncuentra un error durante la evaluación de Deepthought, esta habrá
	4

Capítulo III Parte obligatoria

Nombre de pro-	libft.a
grama	
Archivos a entre-	Makefile, libft.h, ft_*.c
gar	
Makefile	NAME, all, clean, fclean, re
Funciones autor-	Detalles debajo
izadas	
Se permite usar	todavía no la tienes
libft	
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	•	toupper
•	isdigit	< .	tolower
•	isalnum		
•	isascii	•	strchr
•	isprint	•	strrchr
•	strlen		
•	memset	•	$\operatorname{strncmp}$
•	bzero	•	memchr
•	memcpy		memcmp
•	memmove		
/•	strlcpy	•	strnstr
•	strlcat		atoi

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

- calloc
- strdup

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 se pueden hacer de forma más fácil si utilizas funciones de la parte 1.

Nombre de fun-	ft_substr
ción	
Prototipo	<pre>char *ft_substr(char const *s, unsigned int start,</pre>
	size_t len);
Archivos a entre-	- /
gar	
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 autor-	malloc
izadas	
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 fun-	ft_strjoin
ción	
Prototipo	<pre>char *ft_strjoin(char const *s1, char const *s2);</pre>
Archivos a entre-	-
gar	
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 autor-	malloc
izadas	
Descripción	Reserva (con malloc(3)) y devuelve una nueva
	string, formada por la concatenación de 's1' y
/	's2'.

Nombre de fun-	ft_strtrim
ción	
Prototipo	<pre>char *ft_strtrim(char const *s1, char const *set);</pre>
Archivos a entre-	- /
gar	
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 autor-	malloc
izadas	
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 fun-	ft_split
ción	
Prototipo	<pre>char **ft_split(char const *s, char c);</pre>
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 autor-	malloc, free
izadas	
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 fun-	ft_itoa
ción	
Prototipo	<pre>char *ft_itoa(int n);</pre>
Archivos a entre-	- /
gar	
Parámetros	n: el entero a convertir.
Valor devuelto	La string que represente el número.
	NULL si falla la reserva de memoria.
Funciones autor-	malloc
izadas	
Descripción	Utilizando malloc(3), genera una string que
	represente el valor entero recibido como argumento.
	Los números negativos tienen que gestionarse.

Nombre de fun-	ft_strmapi
ción	15_SSIMAPI
Prototipo	<pre>char *ft_strmapi(char const *s, char (*f)(unsigned</pre>
	int, char));
Archivos a entre-	-
gar	
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 autor-	malloc
izadas	
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'

Nombre de fun- ción	ft_striteri
Prototipo	<pre>void ft_striteri(char *s, void (*f)(unsigned int,</pre>
	char*));
Archivos a entre-	
gar	
Parámetros	s: La string que iterar.
	f: La función a aplicar sobre cada carácter.
Valor devuelto	Nada
Funciones autor-	Ninguna
izadas	
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 fun-	ft_putchar_fd
ción	
Prototipo	<pre>void ft_putchar_fd(char c, int fd);</pre>
Archivos a entre-	-
gar	
Parámetros	c: El carácter a enviar.
	fd: El file descriptor sobre el que escribir.
Valor devuelto	Nada
Funciones autor-	write
izadas	
Descripción	Envía el carácter 'c' al file descriptor
/	especificado.

Nombre de fun-	ft_putstr_fd
ción	
Prototipo	<pre>void ft_putstr_fd(char *s, int fd);</pre>
Archivos a entre-	-
gar	
Parámetros	s: La string a enviar.
	fd: El file descriptor sobre el que escribir.
Valor devuelto	Nada
Funciones autor-	write
izadas	
Descripción	Envía la string 's' al file descriptor
	especificado.

Nombre de fun-	ft_putendl_fd	/
ción		
Prototipo	<pre>void ft_putendl_fd(char *s, int</pre>	fd);
Archivos a entre-	- /	
gar		
Parámetros	s: La string a enviar.	
	fd: El file descriptor sobre el	que escribir.
Valor devuelto	Nada	
Funciones autor-	write	
izadas		
Descripción	Envía la string 's' al file desc	criptor dado,
	seguido de un salto de línea.	

Nombre de fun-	ft putnbr fd
ción	
Prototipo	<pre>void ft_putnbr_fd(int n, int fd);</pre>
Archivos a entre-	_
gar	
Parámetros	n: El número que enviar.
	fd: El file descriptor sobre el que escribir.
Valor devuelto	Nada
Funciones autor-	write
izadas	
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:

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 fun-	ft_lstnew
ción	
Prototipo	t_list *ft_lstnew(void *content);
Archivos a entre-	-
gar	
Parámetros	content: el contenido con el que crear el nodo.
Valor devuelto	El nuevo nodo
Funciones autor-	malloc
izadas	
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 fun-	ft_lstadd_front
ción	
Prototipo	<pre>void ft_lstadd_front(t_list **lst, t_list *new);</pre>
Archivos a entre-	-
gar	
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 autor-	Ninguna
izadas	
Descripción	Añade el nodo 'new' al principio de la lista 'lst'.

Nombre de fun-	ft_lstsize
ción	
Prototipo	<pre>int ft_lstsize(t_list *lst);</pre>
Archivos a entre-	-
gar	
Parámetros	lst: el principio de la lista.
Valor devuelto	La longitud de la lista.
Funciones autor-	Ninguna
izadas	
Descripción	Cuenta el número de nodos de una lista.

Nombre de fun-	ft_lstlast
ción	
Prototipo	t_list *ft_lstlast(t_list *lst);
Archivos a entre-	- /
gar	
Parámetros	lst: el principio de la lista.
Valor devuelto	Último nodo de la lista.
Funciones autor-	Ninguna
izadas	
Descripción	Devuelve el último nodo de la lista.

Nombre de fun-	ft_lstadd_back
ción	
Prototipo	<pre>void ft_lstadd_back(t_list **lst, t_list *new);</pre>
Archivos a entre-	- /
gar	
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 autor-	Ninguna
izadas	
Descripción	Añade el nodo 'new' al final de la lista 'lst'.

Nombre de fun-	ft_lstdelone
ción	
Prototipo	<pre>void ft lstdelone(t list *lst, void (*del)(void</pre>
/	*));
Archivos a entre-	- / / /
gar	
Parámetros	lst: el nodo a liberar.
	del: un puntero a la función utilizada para liberar
	el contenido del nodo.
Valor devuelto	Nada
Funciones autor-	free
izadas	
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.

77 1 1 0	
Nombre de fun-	ft_lstclear
ción	
Prototipo	<pre>void ft lstclear(t list **lst, void (*del)(void</pre>
Trococipo	*));
Archivos a entre-	
gar	
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 autor-	free
izadas	
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 fun-	ft_lstiter
ción	
Prototipo	<pre>void ft_lstiter(t_list *lst, void (*f)(void *));</pre>
Archivos a entre-	-
gar	
Parámetros	lst: un puntero al primer nodo.
	f: un puntero a la función que utilizará cada nodo.
Valor devuelto	Nada
Funciones autor-	Ninguna
izadas	
Descripción	Itera la lista 'lst' y aplica la función 'f' en el
	contenido de cada nodo.

Nombre de fun-	ft_lstmap
ción	
Prototipo	t_list *ft_lstmap(t_list *lst, void *(*f)(void *),
	<pre>void (*del)(void *));</pre>
Archivos a entre-	
gar	
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 autor-	malloc, free
izadas	
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