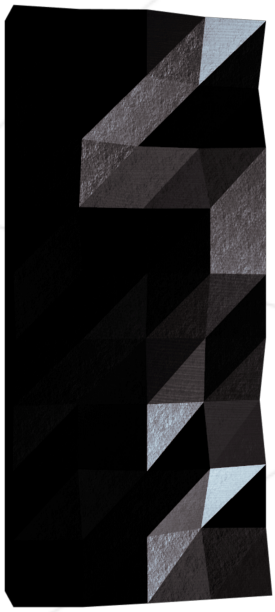

Libft

Tu propia biblioteca, para ti solo

Resumen: Este proyecto tiene como objetivo que desarrolle en `C` une biblioteca de ciertas funciones muy habituales, que así podrá utilizar en todos sus proyectos futuros.

Índice general

Capítulo I

Introducción

La programación en `C` es una actividad muy laboriosa si no tenemos acceso a todas esas pequeñas funciones que se utilizan con frecuencia y son muy prácticas. Por esa razón, a través de este proyecto le proponemos que dedique tiempo a reescribir esas funciones, a comprenderlas y a dominarlas. Así podrá reutilizar su biblioteca para trabajar de manera eficaz en sus próximos proyectos en `C`.

Este proyecto también le va a permitir ampliar la lista de las funciones que le pediremos con las suyas propias, y así hacer que su biblioteca sea aún más útil. No dude en completar su `libft` a lo largo de su escolaridad, cuando este proyecto ya no sea más que un recuerdo.

Capítulo II

Reglas comunes

- Su proyecto debe estar programado respetando la Norma. Si tiene archivos o funciones extras, entrarán dentro de la verificación de la norma y, como haya algún error de norma, tendrá un 0 en el proyecto.
- Sus funciones no pueden pararse de forma inesperada (segmentation fault, bus error, double free, etc.) salvo en el caso de un comportamiento indefinido. Si esto ocurre, se considerará que su proyecto no es funcional y tendrá un 0 en el proyecto.
- Cualquier memoria reservada en el montón (heap) tendrá que ser liberada cuando sea necesario. No se tolerará ninguna fuga de memoria.
- Si el proyecto lo requiere, tendrá que entregar un Makefile que compilará sus códigos fuente para crear la salida solicitada, utilizando los flags `-Wall`, `-Wextra` y `-Werror`. Su Makefile no debe hacer relink.
- Si el proyecto requiere un Makefile, su Makefile debe incluir al menos las reglas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los extras, debe incluir en su Makefile una regla `bonus` que añadirá los headers, bibliotecas o funciones que no estén permitidos en la parte principal del proyecto. Los extras deben estar dentro de un archivo `_bonus.{c/h}`. Las evaluaciones de la parte obligatoria y de la parte extra se hacen por separado.
- Si el proyecto autoriza su `libft`, debe copiar sus códigos fuente y y su Makefile asociado en un directorio libft, dentro de la raíz. El Makefile de su proyecto debe compilar la biblioteca con la ayuda de su Makefile y después compilar el proyecto.
- Le recomendamos que cree programas de prueba para su proyecto, aunque ese trabajo **no será ni entregado ni evaluado**. Esto le dará la oportunidad de probar fácilmente su trabajo al igual que el de sus compañeros.
- Deberá entregar su trabajo en el git que se le ha asignado. Solo se evaluará el trabajo que se suba al git. Si Deepthought debe corregir su trabajo, lo hará al final de las evaluaciones por sus pares. Si surge un error durante la evaluación Deepthought, esta última se parará.

Nombre del programa	`libft.a`
Ficheros de entrega	-
Makefile	Sí
Funciones externas autorizadas	Voir debajo
Libft autorizada	No aplica
Descripción	Escriba su propia librería, que contenga un extracto de las funciones que necesitará más adelante durante su formación.

II.1. Consideraciones técnicas

- Está prohibido utilizar variables globales.
- Si necesita funciones auxiliares para escribir una función compleja, tendrá que definir esas funciones auxiliares en `static`, respetando la Norma.

II.2. Parte 1 - Funciones de la libc

En esta primera parte, tendrá que volver a programar un conjunto de funciones de la `libc`, tal y como vienen descritas en el `man` respectivo de su sistema. Sus funciones tendrán que tener exactamente el mismo prototipo y el mismo comportamiento que las originales. Sus nombres tendrán que tener el prefijo "`ft_`". Por ejemplo, `strlen` se convierte en `ft_strlen`.

Algunos de los prototipos de las funciones que tiene que volver a programar utilizan el calificador de tipo "restrict". Esta palabra clave pertenece al estándar c99, por lo tanto no debe incluirlo en sus prototipos y no debe compilar con el flag -std=c99.

Tiene que volver a programar las funciones siguientes. Estas funciones no necesitan ninguna función externa:

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

También tendrá volver programar estas funciones llamando a la función "`malloc`":

- `calloc`
- `strdup`

II.3. Parte 2 - Funciones adicionales

En esta segunda parte, tendrá que volver a programar algunas funciones que no se encuentran en la `libc` o que lo están en una forma distinta. Algunas de estas funciones pueden ser interesantes para facilitar la escritura de las funciones de la primera parte.

Nombre de la función	`ft_substr`
Prototipo	`char *ft_substr(char const *s, unsigned int start, size_t len);`
Ficheros a entregar	-

Parámetros	`#1. La cadena de la que se extrae la nueva cadena`
	`#2. El índice del principio de la nueva cadena`
	`#3. El tamaño máximo de la nueva cadena.`

Valor de retorno	`La nueva cadena de caracteres. NULL si falla la reserva de memoria.`
Funciones externas autorizadas	`malloc`
Descripción	`Reserva memoria (con malloc(3)) para la cadena de caracteres que va a devolver, y que proviene de la cadena pasada como argumento.` `Esta nueva cadena comienza en el índice 'start' y tiene como tamaño máximo 'len'`

Nombre de la función	`ft_strjoin`
Prototipo	`char *ft_strjoin(char const *s1, char const *s2);`
Ficheros a entregar	-
Parámetros	`#1. La cadena de caracteres prefijo.` `#2. La cadena de caracteres sufijo.`
Valor de retorno	`La nueva cadena de caracteres. NULL si falla la reserva de memoria.`
Funciones externas autorizadas	`malloc`
Descripción	`Reserva memoria (con malloc(3)) para la cadena de caracteres que va a devolver, y que resulta de la concatenación de s1 y s2.`

Nombre de la función	`ft_strtrim`
Prototipo	`char *ft_strtrim(char const *s1, char const *set);`
Ficheros a entregar	-
Parámetros	#1. La cadena de caracteres que hay que depurar. #2. El set de referencia de caracteres que hay que retirar.
Valor de retorno	La cadena de caracteres depurada. NULL si falla la reserva de memoria.
Funciones externas autorizadas	`malloc`
Descripción	Reserva memoria (con malloc(3)) para la cadena de caracteres que va a devolver, que es una copia de la cadena de caracteres pasada como argumento, sin los caracteres indicados en el set pasado como argumento al principio y al final de la cadena de caracteres.

Nombre de la función	`ft_split`
Prototipo	`char **ft_split(char const *s, char c);`
Ficheros a entregar	-
Parámetros	#1. La cadena de caracteres que hay que trocear. #2. El carácter delimitador.
Valor de retorno	La tabla con las nuevas cadenas de caracteres que resulten del troceado. NULL si falla la reserva de memoria.
Funciones externas autorizadas	`malloc, free`
Descripción	Reserva memoria (con malloc(3)) para la tabla de cadena de caracteres que va a devolver, obtenida separando s con el carácter c, que se utiliza como delimitador. La tabla debe terminar con NULL.

Nombre de la función	`ft_itoa`
Prototipo	`char *ft_itoa(int n);`
Ficheros a entregar	-
Parámetros	#1. El integer que hay que convertir.
Valor de retorno	La cadena de caracteres que representa al integer. NULL si falla la reserva de memoria.
Funciones externas autorizadas	`malloc`
Descripción	Reserva memoria (con malloc(3)) para la cadena de caracteres que va a devolver, que representa el integer pasado como argumento. Se deben gestionar los números negativos.

Nombre de la función	`ft_strmapi`
Prototipo	`char *ft_strmapi(char *s, void (*f)(unsigned int, char));`
Ficheros a entregar	-
Parámetros	#1. La cadena de caracteres sobre la que hay que realizar la iteración #2. La función que hay que aplicar a cada carácter.
Valor de retorno	La cadena de caracteres que resulte de las aplicaciones sucesivas de f. Devuelve NULL si falla la reserva de memoria.
Funciones externas autorizadas	`malloc`
Descripción	Aplica la función f a cada carácter de la de cadena de caracteres pasada como argumento para crear una nueva cadena de caracteres (con malloc (3)) que resulte de las aplicaciones sucesivas de f.

Nombre de la función	`ft_putchar_fd`
Prototipo	`void ft_putchar_fd(char c, int fd);`
Ficheros a entregar	-
Parámetros	#1. El carácter que hay que escribir #2. El file descriptor sobre el que hay que escribir.
Valor de retorno	None
Funciones externas autorizadas	`write`
Descripción	Escribe el carácter c sobre el file descriptor proporcionado.

Nombre de la función	`ft_putstr_fd`
Prototipo	`void ft_putstr_fd(char *s, int fd);`
Ficheros a entregar	-
Parámetros	#1. La cadena de caracteres que hay que escribir #2. El descriptor de fichero sobre el que hay que escribir.
Valor de retorno	None
Funciones externas autorizadas	`write`
Descripción	Escribe la cadena de caracteres c sobre el descriptor de fichero proporcionado.

Nombre de la función	`ft_putendl_fd`
Prototipo	`void ft_putendl_fd(char *s, int fd);`
Ficheros a entregar	-
Parámetros	#1. La cadena de caracteres que hay que escribir #2. El file descriptor sobre el que hay que escribir.
Valor de retorno	None
Funciones externas autorizadas	`write`
Descripción	Escribe la cadena de caracteres s sobre el file descriptor proporcionado, seguida de un salto de línea.

Nombre de la función	`ft_putnbr_fd`
Prototipo	`void ft_putnbr_fd(int n, int fd);`
Ficheros a entregar	`-`
Parámetros	`#1. El entero que hay que escribir` `#2. El file descriptor sobre el que hay que escribir.`
Valor de retorno	`None`
Funciones externas autorizadas	`write`
Descripción	`Escribe el entero n sobre el file descriptor proporcionado.`

Capítulo III

Parte extra

Si ha logrado realizar perfectamente la parte obligatoria, esta sección propone algunas pistas para ir más lejos. Es un poco como cuando compra un DLC para un videojuego.

Disponer de funciones de manipulación de memoria bruta y de cadenas de caracteres es muy útil, pero enseguida va a comprender que disponer de funciones de manipulación de listas es todavía más útil.

Utilizará la estructura siguiente para representar los nodos de su lista. Deberá añadir esta estructura a su fichero `libft.h`.

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

La descripción de los campos de la estructura `t_list` es la siguiente:

- `content`: El dato contenido en el nodo. El `void *` permite almacenar un dato de cualquier tipo.
- `next`: La dirección del siguiente nodo de la lista o `NULL`, si se trata del último nodo.

Las funciones siguientes le permitirán manipular sus listas con facilidad.

Nombre de la función	`ft_lstnew`
Prototipo	`t_list ft_lstnew(void const *content);`
Ficheros a entregar	-
Parámetros	#1. El contenido del nuevo elemento.
Valor de retorno	El elemento nuevo
Funciones externas autorizadas	`malloc`
Descripción	Reserva memoria (con malloc(3)) para un nuevo elemento que devuelve. La variable content se inicializa mediante el valor del parámetro content. La variable next se inicializa con NULL.

Nombre de la función	`ft_lstadd_front`
Prototipo	`void ft_lstadd_front(t_list **alst, t_list *new);`
Ficheros a entregar	-
Parámetros	#1. La dirección del puntero al primer elemento de la lista. #2. La dirección del puntero al elemento que hay que añadir a la lista.
Valor de retorno	None
Funciones externas autorizadas	None
Descripción	Añade el elemento new al principio de la lista

Nombre de la función	`ft_lstsize`
Prototipo	`int ft_lstsize(t_list *lst);`
Ficheros a entregar	-
Parámetros	#1. Principio de la lista.
Valor de retorno	Tamaño de la lista.
Funciones externas autorizadas	None
Descripción	Cuenta el número de elementos de la lista.

Nombre de la función	ft_lstlast
Prototipo	t_list *ft_lstlast(t_list *lst);
Ficheros a entregar	-
Parámetros	#1. Principio de la lista.
Valor de retorno	Último elemento de la lista
Funciones externas autorizadas	None
Descripción	Devuelve el último elemento de la lista.

Nombre de la función	ft_lstadd_back
Prototipo	void ft_lstadd_back(t_list **alst, t_list *new);
Ficheros a entregar	-
Parámetros	#1. La dirección del puntero al primer elemento de la lista. #2. La dirección del puntero al elemento que hay que añadir a la lista.
Valor de retorno	None
Funciones externas autorizadas	None
Descripción	Añade el elemento new al final de la lista.

Nombre de la función	ft_lstdelone
Prototipo	void ft_lstdelone(t_list *lst, void (*del)(void *));
Ficheros a entregar	-
Parámetros	#1. El elemento cuya memoria se debe liberar #2. La dirección de una función utilizada para borrar el contenido del elemento.
Valor de retorno	None
Funciones externas autorizadas	free
Descripción	Libera la memoria del elemento pasado como argumento utilizando la función del y después free(3). No se debe liberar la memoria de "next". A continuación, el puntero al elemento debe pasar a NULL

Nombre de la función	`ft_lstclear`
Prototipo	`void ft_lstclear(t_list **lst, void (*del)(void *);`
Ficheros a entregar	-
Parámetros	#1. La dirección del puntero a un elemento. #2. La dirección de la función que permite suprimir el contenido de un elemento.
Valor de retorno	None
Funciones externas autorizadas	`free`
Descripción	Suprime y libera la memoria del elemento pasado como parámetro y de todos los elementos siguientes, con del y free (3) Por último, el puntero inicial debe pasar a NULL.

Nombre de la función	`ft_lstiter`
Prototipo	`void ft_lstiter(t_list *lst, void (*f)(void *));`
Ficheros a entregar	-
Parámetros	#1 La dirección del puntero a un elemento. #2. La dirección de la función que hay que aplicar.
Valor de retorno	None
Funciones externas autorizadas	None
Descripción	Realiza una iteración sobre la lista lst y aplica la función f al contenido de cada elemento.

Nombre de la función	`ft_lstmap`
Prototipo	`t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));`
Ficheros a entregar	-
Parámetros	#1. La dirección del puntero a un elemento. #2. La dirección de la función que hay que aplicar.
Valor de retorno	La nueva lista. NULL si falla la asignación de memoria.
Funciones externas autorizadas	`malloc`
Descripción	Realiza una iteración sobre la lista lst y aplica la función f al contenido de cada elemento. Crea una nueva lista que resulta de las aplicaciones sucesivas de f. Disponemos de la función del si hay que elminar el contenido de algún elemento.

Puede añadir funciones a su libft a su gusto.