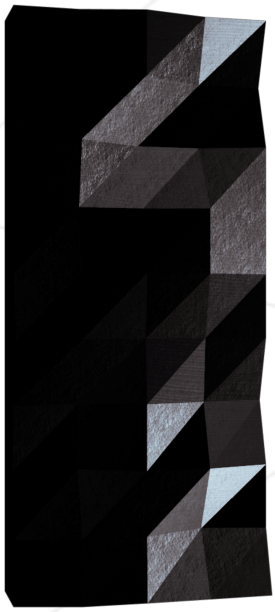

Libft

Tu primera librería

Resumen: Este proyecto tiene como objetivo que escribas las funciones más comunes en `C` *para utilizarlas en todos tus otros proyectos.*

Versión: 15

Índice general

Capítulo I

Introducción

`C` es un lenguaje de programación que puede resultar tedioso si no se dispone de las herramientas adecuadas: las funciones de la librería estándar. Este proyecto te permite reescribir estas funciones, entenderlas, y aprender a utilizarlas. Esta librería te será de utilidad en tus futuros proyectos.

Tómate tu tiempo para expandir tu `libft` a lo largo del año. Por supuesto, asegúrate de comprobar siempre qué funciones se permiten.

Capítulo II

Instrucciones generales

- 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 dentro.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) ni teener comportamientos indefinidos. Si esto pasa tu proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria alocada en heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el subject lo requiere, deberás entregar un `Makefile` que compilará tus archivos fuente al output requerido con las flags `-Wall`, `-Werror` y `-Wextra`, 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 a tu repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus compañeros. Si se encuentra un error durante la evaluación de Deepthought, la evalaución terminará.

Capítulo III

Parte obligatoria

Nombre de programa	`libft.a`
Archivos a entregar	`*.c, libft.h, Makefile`
Makefile	`Sí`
Funciones autorizadas	`Se especifica en cada sección`
Se permite usar libft	`No aplica`
Descripción	`Escribe tu propia librería, que contenga un extracto de todas las funciones importantes para tu cursus.`

III.1. Consideraciones técnicas

- Está prohibido declarar variables globales.
- Si necesitas funciones adicionales para escribir funciones más complejas, deberás definirlas como `static` para evitar publicarlas junto a librería. Es un buen hábito a tener en cuenta para tus futuros proyectos.
- Envía todos tus archivos en la raíz del repositorio.
- Está prohibido entregar archivos no utilizados.
- Cada archivo .c deberá compilar con flags.
- Deberás utilizar el comando `ar` para crear tu librería, el uso del comando `libtool` está prohibido.

III.2. Parte 1 - Funciones de libc

En esta primera parte, deberás programar un conjunto de las funciones de la `libc`, como se define en el `man`. Tus funciones tendrán que presentar el mismo prototipo y comportamiento que las originales. El nombre de tus funciones deberá incluir el prefijo "`ft_`". Por ejemplo, `strlen` se deberá llamar `ft_strlen`.

Algunas de las funciones tienen en su prototipo la palabra reservada "restrict". Esta palabra es parte del estándar c99. Por lo tanto, está prohibido incluirla en tus prototipos y compilar con el flag `-std=c99`.

Deberás programar las siguientes funciones. Estas funciones no dependen de ninguna otra función externa:

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

Deberás programar las siguientes funciones, utilizando la función "`malloc`":

- `calloc`
- `strdup`

III.3. Parte 2 - Funciones adicionales

En esta segunda parte, deberás escribir un conjunto de funciones que, o no están incluidas en `libc`, o lo están en una forma distinta. Algunas de estas funciones se pueden escribir fácilmente utilizando 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	#1. La string de la que formar la nueva. #2. El índice de la string por el que empezar la nueva string. #3. La longitud máxima de la nueva string.
Valor devuelto	La nueva string. NULL si la reserva de memoria falla.
Funciones autorizadas	`malloc`
Descripción	Reserva con malloc(3) memoria para devolver una string nueva basada en la string 's'. La nueva string empieza en 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	#1. La string prefijo. #2. La string sufijo.
Valor devuelto	La nueva string. NULL si la reserva falla.
Funciones autorizadas	`malloc`
Descripción	Reserva con malloc(3) una nueva string, basada en la unión de las dos strings dadas como parámetros.

Nombre de función	ft_strtrim
Prototipo	char *ft_strtrim(char const *s1, char const *set);
Archivos a entregar	-
Parámetros	#1. La string a recortar. #2. El conjunto de caracteres utilizado como referencia para el recorte.
Valor devuelto	La string recortada. NULL si falla la reserva.
Funciones autorizadas	malloc
Descripción	Reserva con malloc(3) y devuelve una copia de 's1' con los caracteres dados en 'set' eliminados tanto del principio como del final.

Nombre de función	ft_split
Prototipo	char **ft_split(char const *s, char c);
Archivos a entregar	-
Parámetros	#1. La string a separar. #2. El caracter delimitador.
Valor devuelto	El array de strings resultante. NULL si la reserva falla.
Funciones autorizadas	malloc, free
Descripción	Reserva con malloc(3) y devuelve un array de strings obtenido al separar 's' con el caracter 'c' como delimitador. El array debe terminar en NULL.

Nombre de función	ft_itoa
Prototipo	char *ft_itoa(int n);
Archivos a entregar	-
Parámetros	#1. El entero a convertir.
Valor devuelto	La string que represente el número. NULL si falla la reserva.
Funciones autorizadas	malloc
Descripción	Reserva con malloc(3) y devuelve una string que representa el número dado como argumento. Los números negativos deben gestionarse correctamente.

Nombre de función	`ft_strmapi`
Prototipo	`char *ft_strmapi(char const *s, char (*f)(unsigned int, char));`
Archivos a entregar	-
Parámetros	#1. La string que iterar. #2. Un puntero a la función que aplicar a cada caracter.
Valor devuelto	La string resultante de aplicar sucesivas veces 'f' a cada caracter. NULL si falla la reserva.
Funciones autorizadas	`malloc`
Descripción	Aplica la función 'f' a cada caracter de la string 's' para crear la nueva string, resultado de aplicar sucesivas veces 'f' (utilizando malloc(3)). A esta función se le pasará el índice del caracter iterado.

Nombre de función	`ft_striteri`
Prototipo	`void ft_striteri(char *s, void (*f)(unsigned int, char*));`
Archivos a entregar	-
Parámetros	#1. La string que iterar. #2. La función a aplicar a cada caracter.
Valor devuelto	Nada
Funciones autorizadas	Ninguna
Descripción	Aplica la función 'f' a cada caracter de la string dada como argumento, pasando su índice como primer argumento. Cada caracter se pasa como una dirección a 'f', por si hace falta modificarlo.

Nombre de función	ft_putchar_fd
Prototipo	void ft_putchar_fd(char c, int fd);
Archivos a entregar	-
Parámetros	#1. El caracter a enviar. #2. El file descriptor sobre el que escribir.
Valor devuelto	Nada
Funciones autorizadas	write
Descripción	Envía el caracter 'c' al file descriptor dado.

Nombre de función	ft_putstr_fd
Prototipo	void ft_putstr_fd(char *s, int fd);
Archivos a entregar	-
Parámetros	#1. La string que imprimir. #2. El file descriptor sobre el que escribir.
Valor devuelto	Nada
Funciones autorizadas	write
Descripción	Escribe la string 's' en el file descriptor indicado.

Nombre de función	ft_putendl_fd
Prototipo	void ft_putendl_fd(char *s, int fd);
Archivos a entregar	-
Parámetros	#1. La string que escribir. #2. El file descriptor sobre el que escribir.
Valor devuelto	Nada
Funciones autorizadas	write
Descripción	Escribe la string 's' en el file descriptor indicado, 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	#1. El número 'n' a escribir. #2. El file descriptor en el que escribir.
Valor devuelto	Nada
Funciones autorizadas	`write`
Descripción	Escribe el número 'n' al file descriptor dado.

Capítulo IV

Parte bonus

Si has disfrutado completando la parte obligatoria, te gustará ir más allá. Puedes ver esta última sección como puntos bonus.

Sí, está bien tener funciones para manipular memoria y strings... Pero pronto te darás cuenta de que aún mejor es tener funciones para manipular listas.

`make bonus` debe añadir las funciones bonus a tu libft.a.

Utilizarás la siguiente estructura para representar elementos de tu lista. Esta estructura debe añadirse a tu `libft.h`.

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

Aquí tienes una breve descripción sobre cada campo de la estructura `t_list`:

- `content`: La información de cada elemento. Al ser de tipo `void *` puede contener todo tipo de información.
- `next`: Un puntero al siguiente elemento, o NULL en caso de ser el último.

¿Suena complicado? Las siguientes funciones te ayudarán en el futuro a utilizar las listas eficientemente.

Nombre de función	`ft_lstnew`
Prototipo	`t_list *ft_lstnew(void *content);`
Archivos a entregar	-
Parámetros	#1. El contenido sobre el que crear un nuevo elemento.
Valor devuelto	El nuevo elemento.
Funciones autorizadas	`malloc`
Descripción	Reserva con malloc(3) y devuelve un elemento nuevo. La variable 'content' se inicializa con el valor del parámetro 'content'. La variable 'next' se inicializa a NULL.

Nombre de función	`ft_lstadd_front`
Prototipo	`void ft_lstadd_front(t_list **lst, t_list *new);`
Archivos a entregar	-
Parámetros	#1. La dirección de un puntero al primer elemento de una lista. #2. La dirección de un puntero al elemento a añadir a la lista.
Valor devuelto	Nada
Funciones autorizadas	Ninguna
Descripción	Añade el elemento 'new' al principio de la lista.

Nombre de función	`ft_lstsize`
Prototipo	`int ft_lstsize(t_list *lst);`
Archivos a entregar	-
Parámetros	#1. El principio de una lista.
Valor devuelto	Longitud de la lista.
Funciones autorizadas	Ninguna
Descripción	Cuenta el número de elemento de una lista.

Nombre de función	ft_lstlast
Prototipo	t_list *ft_lstlast(t_list *lst);
Archivos a entregar	-
Parámetros	#1. El principio de una lista.
Valor devuelto	Último elemento de una lista.
Funciones autorizadas	Ninguna
Descripción	Devuelve el último elemento de una lista.

Nombre de función	ft_lstadd_back
Prototipo	void ft_lstadd_back(t_list **lst, t_list *new);
Archivos a entregar	-
Parámetros	#1. La dirección de un puntero al primer elemento de una lista. #2. Un puntero al elemento nuevo que añadir a la lista.
Valor devuelto	Nada
Funciones autorizadas	Ninguna
Descripción	Añade el elemento 'new' al final de una lista.

Nombre de función	ft_lstdelone
Prototipo	void ft_lstdelone(t_list *lst, void (*del)(void *));
Archivos a entregar	-
Parámetros	#1. El elemento a liberar. #2. La dirección de la función utilizada para eliminar el contenido.
Valor devuelto	Nada
Funciones autorizadas	free
Descripción	Toma como parámetro un elemento y libera la memoria del contenido del elemento utilizando la función 'del' dada como parámetro, por último libera el elemento. 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	#1. La dirección del puntero a un elemento. #2. Un puntero a la función utilizada para eliminar el contenido de cada elemento.
Valor devuelto	Nada
Funciones autorizadas	`free`
Descripción	Elimina y libera cada uno de los elementos de la lista dada, utilizando la función 'del' y free(3). Por último, el puntero a la lista debe ponerse a NULL.

Nombre de función	`ft_lstiter`
Prototipo	`void ft_lstiter(t_list *lst, void (*f)(void *));`
Archivos a entregar	-
Parámetros	#1. Un puntero al primer elemento de una lista. #2. Un puntero a la función que se aplicará a cada elemento de la lista.
Valor devuelto	Nada
Funciones autorizadas	Ninguna
Descripción	Itera la lista 'lst' y aplica la función 'f' al contenido de cada elemento.

Nombre de función	`ft_lstmap`
Prototipo	`t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));`
Archivos a entregar	-
Parámetros	#1. La dirección de un puntero a un elemento. #2. La dirección a un puntero a función utilizada para iterar la lista. #3. La dirección a un puntero a función utilizado para eliminar el contenido de un elemento en caso de requerirse.
Valor devuelto	La nueva lista. NULL si la reserva falla.
Funciones autorizadas	`malloc, free`
Descripción	Itera la lista 'lst' y aplica la función 'f' al contenido de cada elemento. La aplicación correcta de la función 'f' sobre cada elemento genera una nueva lista con estos. La función 'del' se utilizará para eliminar el contenido de un elemento en caso de necesitarse.