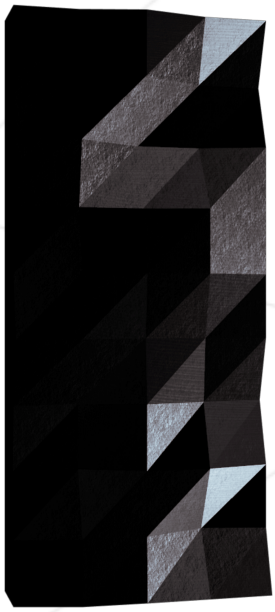

Libft

La tua prima libreria

Sommario: Lo scopo di questo progetto è di programmare una libreria in `C` raggruppando le solite funzioni che sarai in grado di utilizzare in tutti i tuoi progetti futuri.

Indice

Capitolo I

Introduzione

Programmare in `C` può essere molto difficile quando non si ha accesso a queste funzioni standard molto utili. Questo progetto ti dà l'opportunità di riscrivere queste funzioni, capirle e imparare ad utilizzale. Questa libreria ti aiuterà in tutti i tuoi progetti `C` futuri.

Attraverso questo progetto, ti diamo anche l'opportunità di espandere la lista delle funzioni con le tue stesse. Prenditi il tempo per espandere il tuo `libft` durante l'anno.

Capitolo II

Common Instructions

- Il tuo progetto deve essere scritto seguendo le regole imposte dalla Norma. Se hai dei file o funzioni bonus, saranno inclusi nel controllo della Norma e riceverai uno `0` se vi sono errori.
- Le tue funzioni non devono terminare inaspettatamente(segementation fault, bus error, doppio free, etc) a meno che non si tratti di un comportamento non definito. Se ciò dovesse accadere, il tuo progetto sarà considerato come non funzionante e riceverai uno `0` durante la valutazione.
- Tutta la memoria allocata sull'heap deve essere liberata correttamente quando necessario. Non sarà tollerato nessun leak.
- Dovrai consegnare un `Makefile` quando richiesto dal subject. Dovrà compilare i tuoi file sorgente nell;output richiesto utilizzando le flag `-Wall`, `-Wextra` e `-Werror`. Non deve ricollegare.
- Il tuo `Makefile` deve contenere almeno le regole `$(NAME)`, `all`, `clean`, `fclean` e `re`.
- Per consegnare dei bonus, devi includere la regola `bonus` nel tuo Makefile, che aggiungerà tutti gli header, librerie o funzioni proibite nella parte obbligatoria del progetto. I Bonus devono essere in dei file `_bonus.{c/h}`. La valutazione della parte obbligatoria e dei bonus saranno condotte separatamente.
- Se il progetto consente l'utilizzo della tua `libft`, devi copiare i suoi sorgente ed il suo `Makefile` in una cartella `libft`. Il `Makefile` del tuo progetto deve compilare la libreria usando il suo `Makefile`, per poi compilare il progetto.
- Ti incoraggiamo a creare i tuoi programmi di test per il progetto anche se questi programmi **non dorvanno essere consegnati e non saranno valutati**. Ti darà la possibilità di testare facilmente il tuo lavoro e quello dei tuoi peer. Troverai questi programmi specialmente utili durante la tua difesa, durante la quale sarai libero di utilizzare i tuoi, o quelli del tuo peer, file di test.
- Consegna il tuo lavoro nella repository git che ti è stata assegnata. Verrà valutato solo il lavoro presente nella repository git. Se Deepthought dovrà valutare il tuo lavoro, lo farà solo dopo le valutazioni tra pari. Se Deepthought incontra un errore in qualsiasi sezione del tuo porgetto, terminerà la valutazione immediatamente.

Capitolo III

Parte obbligatoria

Nome del programma	`libft.a`
File da consegnare	`*.c, libft.h, Makefile`
Makefile	`Si`
Funzioni esterne permesse	`Dettagliato sotto`
Libft permessa	`Non applicabile`
Descrizione	`Scrivi la tua libreria contenendo un estratto delle funzioni importanti per il tuo cursus.`

III.1 Considerazioni tecniche

- E' vietato utilizzare le variabili globali.
- Se hai bisogno delle sotto funzioni per scrivere una funzione complessa, devi definire queste sotto funzioni come un `static` per evitare di pubblicarli nella tua libreria. Sarebbe una buona abitudine fare questo anche nei tuoi progetti futuri.
- Presenta i tuoi file nella radice della tua repository
- Devi utilizzare il comando `ar` per creare la tua libreria, usare il comando `libtool` è vietato.

III.2 Prima parte - funzioni Libc

In questa prima parte, devi riprogrammare un set delle funzioni `libc`, come definiti nel loro `man`. Le tue funzioni devono avere lo stesso prototipo ed gli stessi comportamenti di quelle originali. I nomi delle funzioni devono essere predefiniti dal "`ft_`". Per esempio `strlen` diventa `ft_strlen`.

Devi riprogrammare alcuni dei prototipi delle funzioni usando il qualificatore "restrict". Questa parola chiave è parte del c99 standard. E' quindi vietato includerlo nei tuoi prototipi e compilarlo con le tue flag -std=c99.

Devi riprogrammare le seguenti funzioni. Queste funzioni non hanno bisogno di nessuna funzione esterna:

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

Devi anche riprogrammare le seguenti funzioni, usando la funzione "`malloc`":

- `calloc`
- `strdup`

III.3 Seconda parte- funzioni aggiuntive

In questa seconda parte, devi programmare un set di funzioni che o non sono incluse nel `libc`, o sono incluse in una forma diversa. Alcune di queste funzioni possono essere utili per scrivere le funzioni della prima parte.

Nome della funzione	`ft_substr`
Prototipo	`char *ft_substr(char const *s, unsigned int start, size_t len);`
File da consegnare	-
Parametri	#1. La stringa dalla quale crei la sotto stringa. #2. L'index di inizio della sotto stringa nella stringa 's'. #3. La massima lunghezza della sotto stringa.
Valore di ritorno	La sotto stringa. NULLO se l'assegnazione fallisce
Funzioni esterne permesse	malloc
Descrizione	Assegna (con malloc(3)) e restituisce una sotto stringa dalla stringa 's'. La sotto stringa inizia all'indice 'start' ed è di massima taglia 'len'.

Nome della funzione	`ft_strjoin`
Prototipo	`char *ft_strjoin(char const *s1, char const *s2);`
File da consegnare	-
Parametri	#1. Il prefisso della stringa. #2. Il suffisso della stringa.
Valore di ritorno	La nuova stringa. NULLO se l'assegnazione fallisce.
Funzioni esterne permesse	malloc
Descrizione	Distribuisce (con malloc(3)) e restituisce una nuova stringa, che è il risultato della concatenazione di 's1' ed 's2'.

Nome della funzione	ft_strtrim
Prototipo	char *ft_strtrim(char const *s1, char const *set);
File da consegnare	-
Parametri	#1. La stringa da essere accorciata. #2. La referenza di caratteri da accorciare.
Valore di ritorno	La stringa accorciata. NULLO se l'assegnazione fallisce.
Funzioni esterne permesse	malloc
Descrizione	Distribuisce (con malloc(3)) e restituisce una copia di 's1' con i caratteri specificati in "set" rimossi dall'inizio e la fine della stringa

Nome della funzione	ft_split
Prototipo	char **ft_split(char const *s, char c);
File da consegnare	-
Parametri	#1. La stringa deve essere separata. #2. Il carattere delimitante.
Valore di ritorno	L'array delle nuove stringe che risulta dalla separazione. NULLO se l'assegnazione fallisce.
Funzioni esterne permesse	malloc, libero
Descrizione	Distribuisce (con malloc(3)) e restituisce un array di stringhe ottenute attraverso la separazione 's' usando il carattere 'c' come un delimitatore. L'array deve finire da un puntatore NULLO.

Nome della funzione	ft_itoa
Prototipo	char *ft_itoa(int n);
File da consegnare	-
Parametri	#1. Il numero intero da convertire.
Valore di ritorno	La stringa che rappresenta il numero intero. NULLO se l'assegnazione fallisce.
Funzioni esterne permesse	malloc
Descrizione	Distribuisce (con malloc(3)) and restituisce una stringa che rappresenta il numero intero ricevuto come un parametro. I numeri negativi devono essere gestiti.

Nome della funzione	ft_strmapi
Prototipo	char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
File da consegnare	-
Parametri	#1. La stringa sulla quale reiterare. #2. La funzione da applicare a ciascun carattere.
Valore di ritorno	La stringa creata delle applicazioni di 'f'. Restituisce NULLO se l'assegnazione fallisce
Funzioni esterne permesse	malloc
Descrizione	Applica la funzione 'f' per ciascun carattere della stringa 's' per creare una nuova stringa (con malloc(3)) che risulta dalle applicazioni di 'f'.

Nome della funzione	ft_putchar_fd
Prototipo	void ft_putchar_fd(char c, int fd);
File da consegnare	-
Parametri	#1. Il carattere da dare come output. #2. Il descrittore del file sul quale scrivere
Valore di ritorno	Nessuna
Funzioni esterne permesse	Scrivi
Descrizione	La funzione ha come output il carattere 'c' al canale indicato nel descrittore di file ricevuto

Nome della funzione	ft_putstr_fd
Prototipo	void ft_putstr_fd(char *s, int fd);
File da consegnare	-
Parametri	#1. La stringa da dare come output. #2. Il descrittore del file sul quale scrivere.
Valore di ritorno	Nessuno
Funzioni esterne permesse	Scrivi
Descrizione	La funzione ha come output la stringa 's' del canale indicato nel descrittore di file ricevuto.

Nome della funzione	`ft_putendl_fd`
Prototipo	`void ft_putendl_fd(char *s, int fd);`
File da consegnare	-
Parametri	#1. La stringe per l'output #2. Il descrittore del file sul quale scrivere.
Valore di ritorno	Nessuno
Funzioni esterne permesse	Scrivi
Descrizione	La funzione ha come output la stringe 's' del canale indicato nel descrittore di file ricevuto, seguito da una nuova riga.

Nome della funzione	`ft_putnbr_fd`
Prototipo	`void ft_putnbr_fd(int n, int fd);`
File da consegnare	-
Parametri	#1. Il numero intero per l'output. #2. Il file di descrizione sul quale scrivere.
Valore di ritorno	Nessuno
Funzioni esterne permesse	Scrivi
Descrizione	La funzione ha come output il numero intero 'n' per il descrittore del file dato.

Capitolo IV

parte Bonus

Se completi la parte obbligatoria con successo, avrai piacere ad andare avanti. Puoi vedere questa ultima parte come Punti Bonus.

Avere funzioni che manipolano la memoria e le stringhe è molto utile, ma presto scoprirai che avere le funzioni per manipolare le liste è ancora più utile.

Userai la seguente struttura per rappresentare gli elementi della tua lista. Questa struttura deve essere aggiunta al tuo `libft.h` file.

`make bonus` aggiungerà le funzioni bonus alla libreria libft.a.

In questa parte, non hai bisogno di aggiungere `_bonus` ai files di .c ed all'header
Aggiungi solo `_bonus` ai file che contengono le tue stesse funzioni bonus.

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

Questa è la descrizione dei campi del `t_list` struct:

- `contenuto` : I dati contenuti nell'elemento. Il `void *` consente conservare ogni tipo di dati.
- `next` : L'indirizzo del prossimo elemento `NULL` o se è l'ultimo elemento.

Le seguenti funzioni ti consentiranno di utilizzare facilmente le tue liste.

Nome della funzione	`ft_lstnew`
Prototipo	`t_list *ft_lstnew(void *content);`
File da consegnare	-
Parametri	#1. Il contenuto per create un nuovo elemento con.
Valore di ritorno	Il nuovo elemento
Funzioni esterne permesse	malloc
Descrizione	Applica (con malloc(3)) e restituisci un nuovo elemento. La variabile 'contenuto' è inizializzato con il valore del parametro 'contenuto'. La variabile 'next' è inizializzata al NULLO

Nome della funzione	`ft_lstadd_front`
Prototipo	`void ft_lstadd_front(t_list **lst, t_list *new);`
File da consegnare	-
Parametri	#1. L'indirizzo del puntatore al primo link della lista. #2. L'indirizzo del puntatore all'elemento che deve essere aggiunto alla lista.
Valore di ritorno	Nessuno
Funzioni esterne permesse	Nessuno
Descrizione	Aggiunge il 'nuovo' elemento all'inizio della lista.

Nome della funzione	`ft_lstsize`
Prototipo	`int ft_lstsize(t_list *lst);`
File da consegnare	-
Parametri	#1. L'inizio della lista
Valore di ritorno	La lunghezza della lista.
Funzioni esterne permesse	Nessuna
Descrizione	Conta i numeri degli elementi nella lista.

Nome della funzione	ft_lstlast
Prototipo	t_list *ft_lstlast(t_list *lst);
File da consegnare	-
Parametri	#1. L'inizio della lista.
Valore di ritorno	L'ultimo elemento della lista.
Funzioni esterne permesse	Nessuno
Descrizione	Ritorna all'ultimo elemento della lista.

Nome della funzione	ft_lstadd_back
Prototipo	void ft_lstadd_back(t_list **lst, t_list *new);
File da consegnare	-
Parametri	#1. L'indirizzo del pointer per il primo link della lista. #2. L'indirizzo del pointer per l'elemento da aggiungere alla lista.
Valore di ritorno	Nessuno
Funzioni esterne permesse	Nessuno
Descrizione	Aggiungi l'elemento 'nuovo' all'inizio della lista.

Nome della funzione	ft_lstdelone
Prototipo	void ft_lstdelone(t_list *lst, void (*del)(void *));
File da consegnare	-
Parametri	#1. L'elemento da liberare. #2. L'indirizzo della funzione usata per cancellare il contenuto.
Valore di ritorno	Nessuno
Funzioni esterne permesse	libero
Descrizione	Prende come parametro un elemento e congela la memoria del contenuto degli elemento usando la funzione 'del' dato come un parametro e libera l'elemento. La memoria di 'next' non deve essere liberata.

Nome della funzione	`ft_lstclear`
Prototipo	`void ft_lstclear(t_list **lst, void (*del)(void *));`
File da consegnare	-
Parametri	#1. L'indirizzo del puntatore per l'elemento. #2. L'indirizzo della funzione usato per cancellare il contenuto dell'elemento.
Valore di ritorno	Nessuno
Funzioni esterne permesse	Libero
Descrizione	Cancella e libera gli elementi dati e ogni successore di quell'elemento , usando la funzione 'del'). Infine, il puntatore della lista deve essere fissato su NULLO.

Nome della funzione	`ft_lstiter`
Prototipo	`void ft_lstiter(t_list *lst, void (*f)(void *));`
File da consegnare	-
Parametri	#1. L'indirizzo del puntatore per un elemento. #2. L'indirizzo della funzione usata per iterare nella lista.
Valore di ritorno	Nessuna
Funzioni esterne permesse	Nessuna
Descrizione	Itera la lista 'lst' e applica la funzione f' per il contenuto di ciascun elemento.

Nome della funzione	`ft_lstmap`
Prototipo	`t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));`
File da consegnare	-
Parametri	#1. L'indirizzo del puntatore di un elemento. #2. L'indirizzo di una funzione usato per iterare sulla lista. #3. L'indirizzo di una funzione usato per cancellare il contenuto di un elemento se necessario.
Valore di ritorno	La nuova lista. NULLA se l'assegnazione fallisce.
Funzioni esterne permesse	malloc, libero
Descrizione	Itera la list I 'lst' e applica la funzione 'f' per il contenuto di ogni elemento. Crea una nuova lista che risulta nelle successive applicazioni della funzione 'f'. La funzione 'del' è usata per cancellare il contenuto di un elemento se necessario.

Sei libero di aggiungere qualsiasi funzione al tuo libft che ti sembra essere adatta.