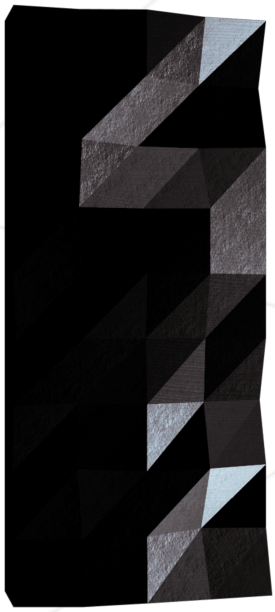

Libft

Ta propre bibliothèque à toi tout seul

Résumé: Ce projet a pour objectif de vous faire coder en `C` une librairie de fonctions usuelles que vous pourrez utiliser dans tous vos prochains projets.

Table des matières

Chapitre I

Introduction

La programmation en `C` est une activité très laborieuse dès lors qu'on a pas accès à toutes ces petites fonctions usuelles très pratiques. C'est pourquoi nous vous proposons à travers ce projet de prendre le temps de récrire ces fonctions, de les comprendre et de vous les approprier. Vous pourrez alors réutiliser votre bibliothèque pour travailler efficacement sur vos projets en `C` suivants.

Ce projet est également pour vous l'occasion d'étendre la liste des fonctions demandées avec les vôtres et ainsi de rendre votre bibliothèque encore plus utile. N'hésitez pas à compléter votre `libft` tout au long de votre scolarité une fois que ce projet ne sera plus qu'un souvenir pour vous.

Chapitre II

Règles communes

- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront inclues dans la vérification de la norme et vous aurez 0 au projet en cas de faute de norme.
- Vos fonctions de doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Toute mémoire allouée sur la heap doit être libéré lorsque c'est nécessaire. Aucun leak ne sera toléré.
- Si le projet le demande, vous devez rendre un Makefile qui compilera vos sources pour créer la sortie demandée, en utilisant les flags `-Wall`, `-Wextra` et `-Werror`. Votre Makefile ne doit pas relink.
- Si le projet demande un Makefile, votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour rendre des bonus, vous devez inclure une règle `bonus` à votre Makefile qui ajoutera les divers headers, librairies ou fonctions qui ne sont pas autorisées dans la partie principale du projet. Les bonus doivent être dans une fichier `_bonus.{c/h}`. L'évaluation de la partie obligatoire et de la partie bonus sont faites séparément.
- Si le projet autorise votre `libft`, vous devez copier ses sources et son Makefile associé dans un dossier libft contenu à la racine. Le Makefile de votre projet doit compiler la librairie à l'aide de son Makefile, puis compiler le projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

II.1 Considérations techniques

- Interdiction d'utiliser des variables globales.
- Si vous avez besoin de fonctions auxiliaires pour l'écriture d'une fonction complexe, vous devez définir ces fonctions auxiliaires en `static` dans le respect de la Norme.

II.2 Part 1 - Fonctions de la libc

Dans cette première partie, vous devez recoder un ensemble de fonctions de la `libc` telles que décrites dans leur `man` respectif sur votre système. Vos fonctions devront avoir exactement le même prototype et le même comportement que les originales. Leur nom devra être préfixé par "`ft_`". Par exemple `strlen` devient `ft_strlen`.

Certains prototypes des fonctions que vous devez recoder utilisent le qualifieur de type "restrict". Ce mot clef fait parti du standard c99, vous devez donc ne pas le mettre dans vos prototypes et ne pas compiler avec le flag -std=c99.

Vous devez recoder les fonctions suivantes. Ces fonctions ne nécessitent aucune fonction externe :

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

Vous devez également recoder ces fonctions, en faisant appel à la fonction "`malloc`" :

- `calloc`
- `strdup`

II.3 Part 2 - Fonctions supplémentaires

Dans cette seconde partie, vous devrez coder un certain nombre de fonctions absentes de la `libc` ou présentes dans une forme différente. Certaines de ces fonctions peuvent avoir de l'intéret pour faciliter l'écriture des fonctions de la première partie.

Function name	`ft_substr`
Prototype	`char *ft_substr(char const *s, unsigned int start, size_t len);`
Fichiers de rendu	`ft_substr.c`

Paramètres	`#1. La chaine de laquelle extraire la nouvelle chaine`
	`#2. L'index de début de la nouvelle chaine`
	`#3. La taille maximale de la nouvelle chaine.`

Valeur de retour	`The nouvelle chaine de caractere. NULL si l'allocation échoue.`
Fonctions externes autorisées	`malloc`
Description	`Alloue (avec malloc(3)) et retourne une chaine de caractères issue de la chaine donnée en argument Cette nouvelle chaine commence à l'index 'start' et a pour taille maximale 'len'`

Function name	`ft_strjoin`
Prototype	`char *ft_strjoin(char const *s1, char const *s2);`
Fichiers de rendu	`ft_strjoin.c`
Paramètres	`#1. La chaine de caractères préfixe.` `#2. La chaine de caractères suffixe.`
Valeur de retour	`La nouvelle chaine de caractères. NULL si l'allocation échoue.`
Fonctions externes autorisées	`malloc`
Description	`Alloue (avec malloc(3)) et retourne une nouvelle chaine, résultat de la concaténation de s1 et s2.`

Function name	ft_strtrim
Prototype	char *ft_strtrim(char const *s1, char const *set);
Fichiers de rendu	ft_strtrim.c
Paramètres	#1. La chaine de caractères à trimmer. #2. Le set de reference de caractères à trimmer.
Valeur de retour	La chaine de caractères trimmée. NULL si l'allocation échoue.
Fonctions externes autorisées	malloc
Description	Alloue (avec malloc(3)) et retourne une copie de la chaine de caractères donnée en argument, sans les caractères spécifiés dans le set donné en argument au début et à la fin de la chaine de caractères.

Function name	ft_split
Prototype	char **ft_split(char const *s, char c);
Fichiers de rendu	ft_split.c
Paramètres	#1. La chaine de caractères à découper. #2. Le caractère délimitant.
Valeur de retour	Le tableau de nouvelles chaines de caractères, résultant du découpage. NULL si l'allocation échoue.
Fonctions externes autorisées	malloc, free
Description	Alloue (avec malloc(3)) et retourne un tableau de chaines de caracteres obtenu en séparant s à l'aide du caractère c, utilisé comme délimiteur. Le tableau doit être terminé par NULL.

Function name	ft_itoa
Prototype	char *ft_itoa(int n);
Fichiers de rendu	ft_itoa.c
Paramètres	#1. l'integer à convertir.
Valeur de retour	La chaine de caractères représentant l'integer. NULL si l'allocation échoue.
Fonctions externes autorisées	malloc
Description	Alloue (avec malloc(3)) et retourne une chaine de caractères représentant l'integer reçu en argument. Les nombres négatifs doivent être gérés.

Function name	`ft_strmapi`
Prototype	`char *ft_strmapi(char *s, void (*f)(unsigned int, char));`
Fichiers de rendu	`ft_strmapi.c`
Paramètres	`#1. La chaine de caractères sur laquelle itérer` `#2. La fonction à appliquer à chaque caractère.`
Valeur de retour	`La chaine de caractères résultant des applications successives de f. Retourne NULL si l'allocation échoue.`
Fonctions externes autorisées	`malloc`
Description	`Applique la fonction f à chaque caractère de la chaine de caractères passée en argument pour créer une nouvelle chaine de caractères (avec malloc(3)) résultant des applications successives de f.`

Function name	`ft_putchar_fd`
Prototype	`void ft_putchar_fd(char c, int fd);`
Fichiers de rendu	`ft_putchar_fd.c`
Paramètres	`#1. Le caractère à écrire` `#2. Le file descriptor sur lequel écrire.`
Valeur de retour	`None`
Fonctions externes autorisées	`write`
Description	`Écrit le caractère c sur le file descriptor donné.`

Function name	`ft_putstr_fd`
Prototype	`void ft_putstr_fd(char *s, int fd);`
Fichiers de rendu	`ft_putstr_fd.c`
Paramètres	`#1. La chaine de caractères à écrire` `#2. Le file descriptor sur lequel écrire.`
Valeur de retour	`None`
Fonctions externes autorisées	`write`
Description	`Écrit la chaine de caractères c sur le file descriptor donné.`

Function name	`ft_putendl_fd`
Prototype	`void ft_putendl_fd(char *s, int fd);`
Fichiers de rendu	`ft_putendl_fd.c`
Paramètres	`#1. La chaine de caractères à écrire` `#2. Le file descriptor sur lequel écrire.`
Valeur de retour	`None`
Fonctions externes autorisées	`write`
Description	`Écrit La chaine de caractères c sur le file descriptor donné, suivi d'un retour chariot.`

Function name	`ft_putnbr_fd`
Prototype	`void ft_putnbr_fd(int n, int fd);`
Fichiers de rendu	`ft_putnbr_fd.c`
Paramètres	`#1. L'integer à écrire` `#2. Le file descriptor sur lequel écrire.`
Valeur de retour	`None`
Fonctions externes autorisées	`write`
Description	`Écrit the integer n sur le file descriptor donné.`

Chapitre III

Partie bonus

Si vous avez réussi parfaitement la partie obligatoire, cette section propose quelques pistes pour aller plus loin. Un peu comme quand vous achetez un DLC pour un jeu vidéo.

Avoir des fonctions de manipulation de mémoire brute et de chaînes de caractères est très pratique, mais vous vous rendrez vite compte qu'avoir des fonctions de manipulation de liste est encore plus pratique.

Vous utiliserez la structure suivante pour représenter les maillons de votre liste. Cette structure est à ajouter à votre fichier `libft.h`.

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

La description des champs de la structure `t_list` est la suivante :

- `content` : La donnée contenue dans le maillon. Le `void *` permet de stocker une donnée de n'importe quel type.
- `next` : L'adresse du maillon suivant de la liste ou `NULL` si le maillon est le dernier.

Les fonctions suivantes vous permettront de manipuler vos listes aisément.

Function name	ft_lstnew
Prototype	t_list ft_lstnew(void const *content);
Fichiers de rendu	ft_lstnew.c
Paramètres	#1. Le contenu du nouvel élément.
Valeur de retour	Le nouvel element
Fonctions externes autorisées	malloc
Description	Alloue (avec malloc(3)) et renvoie un nouvel élément. la variables content est initialisée par copie du paramètre de la fonction. La variable next est initialisée à NULL.

Function name	ft_lstadd_front
Prototype	void ft_lstadd_front(t_list **alst, t_list *new);
Fichiers de rendu	ft_lstadd_front.c
Paramètres	#1. L'adresse du pointeur vers le premier élément de la liste. #2. L'adresse du pointeur vers l'élément à rajouter à la liste.
Valeur de retour	None
Fonctions externes autorisées	None
Description	Ajoute l'élément new au début de la liste

Function name	ft_lstsize
Prototype	int ft_lstsize(t_list *lst);
Fichiers de rendu	ft_lstsize.c
Paramètres	#1. Le début de la liste.
Valeur de retour	Taille de la liste.
Fonctions externes autorisées	None
Description	Compte le nombre d'éléments de la liste.

Function name	ft_lstlast
Prototype	t_list *ft_lstlast(t_list *lst);
Fichiers de rendu	ft_lstlast.c
Paramètres	#1. Le début de la liste.
Valeur de retour	Dernier élément de la liste
Fonctions externes autorisées	None
Description	Renvoie le dernier élément de la liste.

Function name	ft_lstadd_back
Prototype	void ft_lstadd_back(t_list **alst, t_list *new);
Fichiers de rendu	ft_lstadd_back.c
Paramètres	#1. L'adresse du pointeur vers le premier élément de la liste. #2. L'adresse du pointeur vers l'élément à rajouter à la liste.
Valeur de retour	None
Fonctions externes autorisées	None
Description	Ajoute l'élément new à la fin de la liste.

Function name	ft_lstdelone
Prototype	void ft_lstdelone(t_list *lst, void (*del)(void *);
Fichiers de rendu	ft_lstdelone.c
Paramètres	#1. The adress of a pointer to a element that needs to be freed. #2. The adress of the function used to delete the content of l'élément.
Valeur de retour	None
Fonctions externes autorisées	free
Description	Libère la mémoire de l'élément passé en argument en utilisant la fonction del puis avec free(3). La mémoire de "next" ne doit pas être free. Le pointeur vers l'élément doit ensuite être passé à NULL

Function name	ft_lstclear
Prototype	void ft_lstclear(t_list **lst, void (*del)(void *);
Fichiers de rendu	ft_lstclear.c
Paramètres	#1. L'adresse du pointeur vers un élément. #2. L'adresse de la fonction permettant de supprimer le contenu d'un élément.
Valeur de retour	None
Fonctions externes autorisées	free
Description	Supprime et libère la mémoire de l'élément passé en paramètre, et de tous les élements qui suivent, à l'aide de del et de free(3) Enfin, le pointeur initial doit être mis à NULL.

Function name	`ft_lstiter`
Prototype	`void ft_lstiter(t_list *lst, void (*f)(t_list *);`
Fichiers de rendu	`ft_lstiter.c`
Paramètres	`#1 L'adresse du pointeur vers un élément.` `#2. L'adresse de la fonction à appliquer.`
Valeur de retour	`None`
Fonctions externes autorisées	`None`
Description	`Itère sur la list lst et applique la fonction f à chaque élément.`

Function name	`ft_lstmap`
Prototype	`t_list *ft_lstmap(t_list *lst, t_list *(*f)(t_list *);`
Fichiers de rendu	`ft_lstmap.c`
Paramètres	`#1. L'adresse du pointeur vers un élément.` `#2. L'adresse de la fonction à appliquer.`
Valeur de retour	`La nouvelle liste. NULL si l'allocation échoue.`
Fonctions externes autorisées	`malloc`
Description	`Itère sur la liste lst et applique la fonction f à chaque élément. Crée une nouvelle liste résultant des applications successives de f.`

Vous êtes libres d'ajouter des fonctions à votre libft à votre guise.