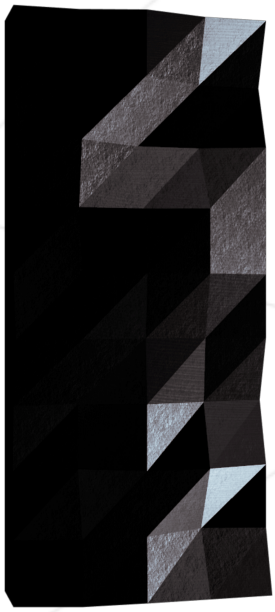

Libft

Ta propre bibliothèque rien que pour toi

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

Version: 15

Table des matières

Chapitre I

Introduction

La programmation en `C` est une activité très laborieuse dès lors que l'on n'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 suivants en `C`.

Prenez le temps d'enrichir votre `libft` tout au long de l'année. Cependant, pour chacun de vos projets futurs, veillez toujours à vérifier quelles sont les fonctions autorisées !

Chapitre II

Règles communes

- Votre projet doit être écrit en C.
- 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 un fichier différent : `_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.

Chapitre III

Partie obligatoire

Nom du programme	`libft.a`
Fichiers de rendu	`Makefile, libft.h, ft_*.c`
Makefile	`NAME, all, clean, fclean, re`
Fonctions externes autorisées	`Détails ci-dessous`
Libft autorisée	`n/a`
Description	`Créez votre propre bibliothèque contenant des fonctions utiles pour la suite de votre cursus.`

III.1 Considérations techniques

- Interdiction d'utiliser des variables globales.
- Si vous avez besoin de fonctions auxiliaires pour réaliser une fonction complexe, vous devez définir ces dernières en `static` dans le respect de la Norme. Ainsi, leur portée sera limitée au fichier concerné.
- Vous devez rendre tous vos fichiers à la racine de votre dépôt.
- Il est interdit de rendre des fichiers non utilisés.
- Chaque fichier `.c` doit être compilé avec les flags `-Wall -Wextra -Werror`.
- Vous devez utiliser la commande `ar` pour créer votre bibliothèque. L'utilisation de la commande `libtool` est interdite.
- Votre `libft.a` doit être créé à la racine de votre dépôt.

III.2 Partie 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. Seule différence, leur nom devra être préfixé par '`ft_`'. Ainsi, `strlen` devient `ft_strlen`.

Certains prototypes des fonctions que vous devez recoder utilisent le qualifieur de type 'restrict'. Ce mot-clé fait parti du standard c99. Par conséquent, vous ne devez pas l'utiliser pour vos prototypes et ne pas compiler votre code avec le flag -std=c99.

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

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

Pour les deux fonctions suivantes, vous pourrez faire appel à la fonction `malloc()` :

- `calloc`
- `strdup`

III.3 Partie 2 - Fonctions supplémentaires

Dans cette seconde partie, vous devrez implémenter un certain nombre de fonctions absentes de la `libc`, ou qui y sont mais sous une forme différente.

Certaines de ces fonctions peuvent faciliter l'écriture des fonctions demandées dans la Partie 1.

Function name	ft_substr
Prototype	char *ft_substr(char const *s, unsigned int start, size_t len);
Fichiers de rendu	-
Paramètres	s: La chaîne de laquelle extraire la nouvelle chaîne. start: L'index de début de la nouvelle chaîne dans la chaîne 's'. len: La taille maximale de la nouvelle chaîne.
Valeur de retour	La nouvelle chaîne de caractères. NULL si l'allocation échoue.
Fonctions externes autorisées	malloc
Description	Alloue (avec malloc(3)) et retourne une chaîne de caractères issue de la chaîne 's'. Cette nouvelle chaîne 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	-
Paramètres	s1: La chaîne de caractères préfixe. s2: La chaîne de caractères suffixe.
Valeur de retour	La nouvelle chaîne de caractères. NULL si l'allocation échoue.
Fonctions externes autorisées	malloc
Description	Alloue (avec malloc(3)) et retourne une nouvelle chaîne, 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	-
Paramètres	s1: La chaîne de caractères à trimmer. set: Le set de référence de caractères à trimmer.
Valeur de retour	La chaîne de caractères trimmée. NULL si l'allocation échoue.
Fonctions externes autorisées	malloc
Description	Alloue (avec malloc(3)) et retourne une copie de la chaîne 's1', sans les caractères spécifiés dans 'set' au début et à la fin de la chaîne de caractères.

Function name	ft_split
Prototype	char **ft_split(char const *s, char c);
Fichiers de rendu	-
Paramètres	s: La chaîne de caractères à découper. c: Le caractère délimiteur.
Valeur de retour	Le tableau de nouvelles chaînes 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 chaînes de caractères 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	-
Paramètres	n: L'entier à convertir.
Valeur de retour	La chaîne de caractères représentant l'entier. NULL si l'allocation échoue.
Fonctions externes autorisées	malloc
Description	Alloue (avec malloc(3)) et retourne une chaîne de caractères représentant l'entier 'n' reçu en argument. Les nombres négatifs doivent être gérés.

Function name	ft_strmapi
Prototype	char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
Fichiers de rendu	-
Paramètres	s: La chaîne de caractères sur laquelle itérer. f: La fonction à appliquer à chaque caractère.
Valeur de retour	La chaîne 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 chaîne de caractères passée en argument pour créer une nouvelle chaîne de caractères (avec malloc(3)) résultant des applications successives de 'f'.

Function name	ft_striteri
Prototype	void ft_striteri(char *s, void (*f)(unsigned int, char*));
Fichiers de rendu	-
Paramètres	s: La chaîne de caractères sur laquelle itérer. f: La fonction à appliquer à chaque caractère.
Valeur de retour	Aucune
Fonctions externes autorisées	Aucune
Description	Applique la fonction 'f' à chaque caractère de la chaîne de caractères transmise comme argument, et en passant son index comme premier argument. Chaque caractère est transmis par adresse à 'f' afin d'être modifié si nécessaire.

Function name	ft_putchar_fd
Prototype	void ft_putchar_fd(char c, int fd);
Fichiers de rendu	-
Paramètres	c: Le caractère à écrire. fd: Le descripteur de fichier sur lequel écrire.
Valeur de retour	Aucune
Fonctions externes autorisées	write
Description	Écrit le caractère 'c' sur le descripteur de fichier donné.

Function name	`ft_putstr_fd`
Prototype	`void ft_putstr_fd(char *s, int fd);`
Fichiers de rendu	-
Paramètres	`s: La chaîne de caractères à écrire.` `fd: Le descripteur de fichier sur lequel écrire.`
Valeur de retour	`Aucune`
Fonctions externes autorisées	`write`
Description	`Écrit la chaîne de caractères 's' sur le descripteur de fichier donné.`

Function name	`ft_putendl_fd`
Prototype	`void ft_putendl_fd(char *s, int fd);`
Fichiers de rendu	-
Paramètres	`s: La chaîne de caractères à écrire.` `fd: Le descripteur de fichier sur lequel écrire.`
Valeur de retour	`Aucune`
Fonctions externes autorisées	`write`
Description	`Écrit La chaîne de caractères 's' sur le descripteur de fichier donné suivie d'un retour à la ligne.`

Function name	`ft_putnbr_fd`
Prototype	`void ft_putnbr_fd(int n, int fd);`
Fichiers de rendu	-
Paramètres	`n: L'entier à écrire.` `fd: Le descripteur de fichier sur lequel écrire.`
Valeur de retour	`Aucune`
Fonctions externes autorisées	`write`
Description	`Écrit l'entier 'n' sur le descripteur de fichier donné.`

Chapitre IV

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. Toutefois, vous vous rendrez vite compte qu'avoir des fonctions de manipulation de listes est encore plus pratique.

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

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

Les membres de la structure `t_list` sont les suivants :

- `content` : La donnée contenue dans le maillon.
  `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 suivant est le dernier.

Dans votre Makefile, une règle `make bonus` vous permettra d'ajouter les fonctions demandées à votre `libft.a`.

Vous ne devez pas suffixer vos fichiers `.c` et vos fichiers d'en-tête avec `_bonus`. En effet, ajoutez le suffixe `_bonus` seulement aux fichiers supplémentaires réalisés exclusivement pour la partie bonus.

Les bonus ne seront évalués que si la partie obligatoire est PARFAITE. Par parfaite, nous entendons complète et sans aucun dysfonctionnement. Si vous n'avez pas réussi TOUS les points de la partie obligatoire, votre partie bonus ne sera pas prise en compte.

Implémentez les fonctions suivantes afin de manipuler vos listes aisément.

Function name	ft_lstnew
Prototype	t_list *ft_lstnew(void *content);
Fichiers de rendu	-
Paramètres	content: Le contenu du nouvel élément.
Valeur de retour	Le nouvel élément
Fonctions externes autorisées	malloc
Description	Alloue (avec malloc(3)) et renvoie un nouvel élément. La variable membre 'content' est initialisée à l'aide de la valeur du paramètre 'content'. La variable 'next' est initialisée à NULL.

Function name	ft_lstadd_front
Prototype	void ft_lstadd_front(t_list **lst, t_list *new);
Fichiers de rendu	-
Paramètres	lst: L'adresse du pointeur vers le premier élément de la liste. new: L'adresse du pointeur vers l'élément à rajouter à la liste.
Valeur de retour	Aucune
Fonctions externes autorisées	Aucune
Description	Ajoute l'élément 'new' au début de la liste.

Function name	ft_lstsize
Prototype	int ft_lstsize(t_list *lst);
Fichiers de rendu	-
Paramètres	lst: Le début de la liste.
Valeur de retour	Taille de la liste
Fonctions externes autorisées	Aucune
Description	Compte le nombre d'éléments de la liste.

Function name	ft_lstlast
Prototype	t_list *ft_lstlast(t_list *lst);
Fichiers de rendu	-
Paramètres	lst: Le début de la liste.
Valeur de retour	Dernier élément de la liste
Fonctions externes autorisées	Aucune
Description	Renvoie le dernier élément de la liste.

Function name	ft_lstadd_back
Prototype	void ft_lstadd_back(t_list **lst, t_list *new);
Fichiers de rendu	-
Paramètres	lst: L'adresse du pointeur vers le premier élément de la liste. new: L'adresse du pointeur vers l'élément à rajouter à la liste.
Valeur de retour	Aucune
Fonctions externes autorisées	Aucune
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	-
Paramètres	lst: L'élément à free del: L'adresse de la fonction permettant de supprimer le contenu de l'élément.
Valeur de retour	Aucune
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.

Function name	ft_lstclear
Prototype	void ft_lstclear(t_list **lst, void (*del)(void *));
Fichiers de rendu	-
Paramètres	lst: L'adresse du pointeur vers un élément. del: L'adresse de la fonction permettant de supprimer le contenu d'un élément.
Valeur de retour	Aucune
Fonctions externes autorisées	free
Description	Supprime et libère la mémoire de l'élément passé en paramètre, et de tous les éléments 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)(void *));`
Fichiers de rendu	-
Paramètres	`lst: L'adresse du pointeur vers un élément.` `f: L'adresse de la fonction à appliquer.`
Valeur de retour	`Aucune`
Fonctions externes autorisées	`Aucune`
Description	`Itère sur la liste 'lst' et applique la fonction 'f' au contenu chaque élément.`

Function name	`ft_lstmap`
Prototype	`t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));`
Fichiers de rendu	-

Paramètres	`lst: L'adresse du pointeur vers un élément.`
	`f: L'adresse de la fonction à appliquer.`
	`del: L'adresse de la fonction permettant de supprimer le contenu d'un élément.`

Valeur de retour	`La nouvelle liste.` `NULL si l'allocation échoue`
Fonctions externes autorisées	`malloc, free`
Description	`Itère sur la liste 'lst' et applique la fonction 'f 'au contenu de chaque élément. Crée une nouvelle liste résultant des applications successives de 'f'. La fonction 'del' est là pour détruire le contenu d'un élément si nécessaire.`

Chapitre V

Rendu et peer-evaluation

Rendez votre travail sur votre dépot `Git` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vous devez rendre tous vos fichiers à la racine de votre dépôt.