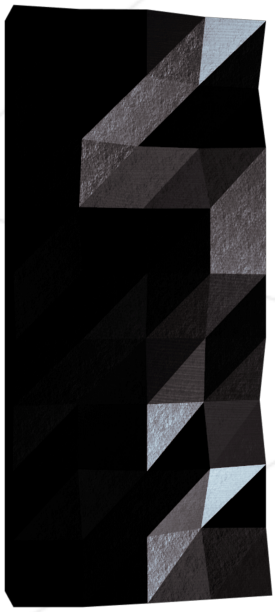

Projet du module Algorithmie I

libft

Résumé: Ce projet a pour but de vous faire coder une bibliothèque de fonctions usuelles que vous pourrez utiliser dans tous vos projets.

Table des matières

Chapitre I

Préambule

Ce premier projet marque le début de votre formation de développeur. profitez-en pour lire cet article et apprenez aujourd'hui que le typage est ce qui différencie le développeur de la bête. Si vous ne comprenez pas tout ce n'est pas grave. Ca viendra avec le temps.

Pour vous accompagner musicalement tout au long de la réalisation de ce projet je vous propose une liste de groupes dignes d'interet. Si vous n'aimez pas, c'est que vous avez visiblement des gouts musicaux pauvres, mais vous avez probablement d'autres qualités comme avoir beaucoup d'amis sur Facebook ou bien pouvoir toucher votre coude avec votre langue. Bref. Les groupes sont listés sans ordre particulier et cette liste n'a pas pour but d'être exhaustive. Les liens proposés sont donnés à titre d'exemple et vous êtes encouragés à explorer vous-même leur riche discographie.

- Between The Buried And Me
- Tesseract
- Chimp Spanner
- Emancipator
- Cynic
- Kalisia
- Porcupine Tree
- Wintersun
- O.S.I
- Dream Theater
- Pain Of Salvation
- Crucified Barbara

Chapitre II

Sujet

II.1 Partie obligatoire

Le projet `libft` reprend le concept du jour 06 de la piscine, à savoir vous faire écrire une bibliothèque de fonctions utiles que vous pourrez ensuite utiliser dans la vaste majorité de vos projets de `C` cette année et ainsi vous faire gagner beaucoup de temps. Ce projet vous demande d'écrire beaucoup de code que vous avez déja réalisé pendant la piscine, ce qui en fait un excellent commencement pour vous remettre sur les rails puisque vous n'y apprendrez rien de nouveau. Voyez ce projet comme le moment ou vous sélectionnez ou équipez votre personnage dans un jeu vidéo.

Les fonctions sont à réaliser dans l'ordre que vous souhaitez et vous êtes très encouragés à utiliser les fonctions déja codées pour réaliser les suivantes. La difficulté n'est pas croissante et l'ordre du sujet parfaitement arbitraire. C'est un peu comme dans un jeu vidéo ou vous pouvez réaliser des quêtes dans l'ordre que vous voulez et utiliser le loot des précédentes pour vous faciliter les suivantes.

II.1.1 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 :

- `memset`
- `bzero`
- `memcpy`
- `memccpy`
- `memmove`
- `memchr`
- `memcmp`
- `strlen`
- `strdup`
- `strcpy`
- `strncpy`
- `strcat`
- `strncat`
- `strlcat`
- `strchr`
- `strrchr`
- `strstr`
- `strnstr`
- `strcmp`
- `strncmp`
- `atoi`
- `isalpha`
- `isdigit`
- `isalnum`
- `isascii`
- `isprint`
- `toupper`
- `tolower`

II.1.2 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.

ft_memalloc	
Prototype	`void * ft_memalloc(size_t size);`
Description	Alloue (avec `malloc(3)`) et retourne une zone de mémoire "fraiche". La mémoire allouée est initialisée à `0`. Si l'allocation échoue, la fonction renvoie `NULL`.
Param. #1	La taille de la zone de mémoire à allouer.
Retour	La zone de mémoire allouée.
Fonctions libc	`malloc(3)`

ft_memdel	
Prototype	`void ft_memdel(void **ap);`
Description	Prend en paramètre l'adresse d'un pointeur dont la zone pointée doit être libérée avec `free(3)`, puis le pointeur est mis à `NULL`.
Param. #1	L'adresse d'un pointeur dont il faut libérer la mémoire puis le mettre à `NULL`.
Retour	Rien.
Fonctions libc	`free(3)`.

ft_strnew	
Prototype	`char * ft_strnew(size_t size);`
Description	Alloue (avec `malloc(3)`) et retourne une chaine de caractère "fraiche" terminée par un `'\0'`. Chaque caractère de la chaine est initialisé à `'\0'`. Si l'allocation echoue, la fonction renvoie `NULL`.
Param. #1	La taille de la chaine de caractères à allouer.
Retour	La chaine de caractères allouée et initialisée à `0`.
Fonctions libc	`malloc(3)`

ft_strdel	
Prototype	`void ft_strdel(char **as);`
Description	Prend en paramètre l'adresse d'une chaine de caractères qui doit être libérée avec `free(3)` et son pointeur mis à `NULL`.
Param. #1	L'adresse de la chaine de caractère dont il faut libérer la mémoire et mettre le pointeur à `NULL`.
Retour	Rien.
Fonctions libc	`Free(3)`.

ft_strclr	
Prototype	`void ft_strclr(char *s);`
Description	Assigne la valeur `'\0'` à tous les caractères de la chaine passée en paramètre.
Param. #1	La chaine de caractères à clearer.
Retour	Rien.
Fonctions libc	Aucune.

ft_striter	
Prototype	`void ft_striter(char *s, void (*f)(char *));`
Description	Applique la fonction `f` à chaque caractère de la chaine de caractères passée en paramètre. Chaque caractère est passé par adresse à la fonction `f` afin de pouvoir être modifié si nécéssaire.
Param. #1	La chaine de caractères sur laquelle itérer.
Param. #2	La fonction à appeler sur chaque caractère de `s`.
Retour	Rien.
Fonctions libc	Aucune.

ft_striteri	
Prototype	`void ft_striteri(char *s, void (*f)(unsigned int, char *));`
Description	Applique la fonction `f` à chaque caractère de la chaine de caractères passée en paramètre en précisant son index en premier argument. Chaque caractère est passé par adresse à la fonction `f` afin de pouvoir être modifié si nécéssaire.
Param. #1	La chaine de caractères sur laquelle itérer.
Param. #2	La fonction à appeler sur chaque caractère de `s` et son index.
Retour	Rien.
Fonctions libc	Aucune.

ft_strmap	
Prototype	`char * ft_strmap(char const *s, char (*f)(char));`
Description	Applique la fonction `f` à chaque caractère de la chaine de caractères passée en paramètre pour créer une nouvelle chaine “fraiche” (avec `malloc(3)`) résultant des applications successives de `f`.
Param. #1	La chaine de caractères sur laquelle itérer.
Param. #2	La fonction à appeler sur chaque caractère de `s`.
Retour	La chaine “fraiche” résultant des applications successives de `f`.
Fonctions libc	`malloc(3)`

ft_strmapi	
Prototype	`char * ft_strmapi(char const *s, char (*f)(unsigned int, char));`
Description	Applique la fonction `f` à chaque caractère de la chaine de caractères passée en paramètre en précisant son index pour créer une nouvelle chaine "fraiche" (avec `malloc(3)`) résultant des applications successives de `f`.
Param. #1	La chaine de caractères sur laquelle itérer.
Param. #2	La fonction à appeler sur chaque caractère de `s` en précisant son index.
Retour	La chaine "fraiche" résultant des applications successives de `f`.
Fonctions libc	`malloc(3)`

ft_strequ	
Prototype	`int ft_strequ(char const *s1, char const *s2);`
Description	Compare lexicographiquement `s1` et `s2`. Si les deux chaines sont égales, la fonction retourne `1`, ou `0` sinon.
Param. #1	La première des deux chaines à comparer.
Param. #2	La seconde des deux chaines à comparer.
Retour	`1` ou `0` selon que les deux chaines sont égales ou non.
Fonctions libc	Aucune.

ft_strnequ	
Prototype	`int ft_strnequ(char const *s1, char const *s2, size_t n);`
Description	Compare lexicographiquement `s1` et `s2` jusqu'à `n` caractères maximum ou bien qu'un `'\0'` ait été rencontré. Si les deux chaines sont égales, la fonction retourne `1`, ou `0` sinon.
Param. #1	La première des deux chaines à comparer.
Param. #2	La seconde des deux chaines à comparer.
Param. #3	Le nombre de caractères à comparer au maximum.
Retour	`1` ou `0` selon que les deux chaines sont égales ou non.
Fonctions libc	Aucune.

ft_strsub	
Prototype	`char * ft_strsub(char const *s, unsigned int start, size_t len);`
Description	Alloue (avec `malloc(3)`) et retourne la copie "fraiche" d'un tronçon de la chaine de caractères passée en paramètre. Le tronçon commence à l'index `start` et à pour longueur `len`. Si `start` et `len` ne désignent pas un tronçon de chaine valide, le comportement est indéterminé. Si l'allocation échoue, la fonction renvoie `NULL`.
Param. #1	La chaine de caractères dans laquelle chercher le tronçon à copier.
Param. #2	L'index dans la chaine de caractères où débute le tronçon à copier.
Param. #3	La longueur du tronçon à copier.
Retour	Le tronçon.
Fonctions libc	`malloc(3)`

ft_strjoin	
Prototype	`char * ft_strjoin(char const *s1, char const *s2);`
Description	Alloue (avec `malloc(3)`) et retourne une chaine de caractères "fraiche" terminée par un `'\0'` résultant de la concaténation de `s1` et `s2`. Si l'allocation echoue, la fonction renvoie `NULL`.
Param. #1	La chaine de caractères préfixe.
Param. #2	La chaine de caractères suffixe.
Retour	La chaine de caractère "fraiche" résultant de la concaténation des deux chaines.
Fonctions libc	`malloc(3)`

ft_strtrim	
Prototype	`char * ft_strtrim(char const *s);`
Description	Alloue (avec `malloc(3)`) et retourne une copie de la chaine passée en paramètre sans les espaces blancs au debut et à la fin de cette chaine. On considère comme espaces blancs les caractères `' '`, `'\n'` et `'\t'`. Si `s` ne contient pas d'espaces blancs au début ou à la fin, la fonction renvoie une copie de `s`. Si l'allocation echoue, la fonction renvoie `NULL`.
Param. #1	La chaine de caractères à trimmer.
Retour	La chaine de caractère "fraiche" trimmée ou bien une copie de `s` sinon.
Fonctions libc	`malloc(3)`

ft_strsplit	
Prototype	`char **  ft_strsplit(char const *s, char c);`
Description	Alloue (avec `malloc(3)`) et retourne un tableau de chaines de caractères “fraiches” (toutes terminées par un `'\0'`, le tableau également donc) résultant de la découpe de `s` selon le caractère `c`. Si l'allocation echoue, la fonction retourne `NULL`. Exemple : `ft_strsplit("*salut*les***etudiants*", '*')` renvoie le tableau `["salut", "les", "etudiants"]`.
Param. #1	La chaine de caractères à découper.
Param. #2	Le caractère selon lequel découper la chaine.
Retour	Le tableau de chaines de caractères “fraiches” résultant de la découpe.
Fonctions libc	`malloc(3)`, `free(3)`

ft_itoa	
Prototype	`char *  ft_itoa(int n);`
Description	Alloue (avec `malloc(3)`) et retourne une chaine de caractères “fraiche” terminée par un `'\0'` représentant l'entier `n` passé en paramètre. Les nombres négatifs doivent être gérés. Si l'allocation échoue, la fonction renvoie NULL.
Param. #1	L'entier à convertir en une chaine de caractères.
Retour	La chaine de caractères représentant l'entier passé en paramètre.
Fonctions libc	`malloc(3)`

ft_putchar	
Prototype	`void  ft_putchar(char c);`
Description	Affiche le caractère `c` sur la sortie standard.
Param. #1	Le caractères à afficher.
Retour	Rien.
Fonctions libc	`write(2)`.

ft_putstr	
Prototype	`void  ft_putstr(char const *s);`
Description	Affiche la chaine `s` sur la sortie standard.
Param. #1	La chaine de caractères à afficher.
Retour	Rien.
Fonctions libc	`write(2)`.

ft_putendl	
Prototype	`void ft_putendl(char const *s);`
Description	Affiche la chaine `s` sur la sortie standard suivi d'un `'\n'`.
Param. #1	La chaine de caractères à afficher.
Retour	Rien.
Fonctions libc	`write(2)`.

ft_putnbr	
Prototype	`void ft_putnbr(int n);`
Description	Affiche l'entier `n` sur la sortie standard.
Param. #1	L'entier à afficher.
Retour	Rien.
Fonctions libc	`write(2)`.

ft_putchar_fd	
Prototype	`void ft_putchar_fd(char c, int fd);`
Description	Ecrit le caractère `c` sur le descripteur de fichier `fd`.
Param. #1	Le caractères à écrire.
Retour	Rien.
Fonctions libc	`write(2)`.

ft_putstr_fd	
Prototype	`void ft_putstr_fd(char const *s, int fd);`
Description	Ecrit la chaine `s` sur le descripteur de fichier `fd`.
Param. #1	La chaine de caractères à écrire.
Retour	Rien.
Fonctions libc	`write(2)`.

ft_putendl_fd	
Prototype	`void ft_putendl_fd(char const *s, int fd);`
Description	Ecrit la chaine `s` sur le descripteur de fichier `fd` suivi d'un `'\n'`.
Param. #1	La chaine de caractères à écrire.
Retour	Rien.
Fonctions libc	`write(2)`.

ft_putnbr_fd	
Prototype	`void ft_putnbr_fd(int n, int fd);`
Description	Ecrit l'entier `n` sur le descripteur de fichier `fd`.
Param. #1	L'entier à écrire.
Retour	Rien.
Fonctions libc	`write(2)`.

II.2 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 pack d'extension pour un jeu vidéo. Les bonus seront comptabilisés si vous obtenez au moins `18/20` à la partie obligatoire.

Avoir des fonction de manipulation de mémoire brute et de chaines 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;
    size_t          content_size;
    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.
- `content_size` : La taille de la donnée stockée. Le type `void *` ne permettant pas de connaitre la taille de la donnée pointée, il est nécessaire d'en sauvegarder la taille. Par exemple la chaine de caractères `"42"` a une taille de `3 octets` et l'entier 32bits `42` a une taille de `4 octets`.
- `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.

ft_lstnew	
Prototype	`t_list * ft_lstnew(void const *content, size_t content_size);`
Description	Alloue (avec `malloc(3)`) et retourne un maillon "frais". Les champs `content` et `content_size` du nouveau maillon sont initialisés par **copie** des paramètres de la fonction. Si le paramètre `content` est nul, le champs `content` est initialisé à `NULL` et le champs `content_size` est initialisé à `0` quelque soit la valeur du paramètre `content_size`. Le champ `next` est initialisé à `NULL`. Si l'allocation échoue, la fonction renvoie `NULL`.
Param. #1	Le contenu à ajouter au nouveau maillon.
Param. #2	La taille du contenu à ajouter au nouveau maillon.
Retour	Le nouveau maillon.
Fonctions libc	`malloc(3)`, `free(3)`

ft_lstdelone	
Prototype	`void ft_lstdelone(t_list **alst, void (*del)(void *, size_t));`
Description	Prend en paramètre l'adresse d'un pointeur sur un maillon et libère la mémoire du contenu de ce maillon avec la fonction `del` passée en paramètre puis libère la mémoire du maillon en lui même avec `free(3)`. La mémoire du champ `next` ne doit en aucun cas être libérée. Pour terminer, le pointeur sur le maillon maintenant libéré doit être mis à `NULL` (de manière similaire à la fonction `ft_memdel` de la partie obligatoire).
Param. #1	L'adresse d'un pointeur sur le maillon à libérer.
Retour	Rien.
Fonctions libc	`free(3)`

ft_lstdel	
Prototype	`void ft_lstdel(t_list **alst, void (*del)(void *, size_t));`
Description	Prend en paramètre l'adresse d'un pointeur sur un maillon et libère la mémoire de ce maillon et celle de tous ses successeurs l'un après l'autre avec `del` et `free(3)`. Pour terminer, le pointeur sur le premier maillon maintenant libéré doit être mis à `NULL` (de manière similaire à la fonction `ft_memdel` de la partie obligatoire).
Param. #1	L'adresse d'un pointeur sur le premier maillon d'une liste à libérer.
Retour	Rien.
Fonctions libc	`free(3)`

ft_lstadd	
Prototype	`void ft_lstadd(t_list **alst, t_list *new);`
Description	Ajoute l'élément `new` en tête de la liste.
Param. #1	L'adresse d'un pointeur sur le premier maillon d'une liste.
Param. #2	Le maillon à ajouter en tête de cette liste.
Retour	Rien.
Fonctions libc	Aucune.

ft_lstiter	
Prototype	`void ft_lstiter(t_list *lst, void (*f)(t_list *elem));`
Description	Parcourt la liste `lst` en appliquant à chaque maillon la fonction `f`.
Param. #1	Pointeur sur le premier maillon d'une liste.
Param. #2	L'adresse d'une fonction à laquelle appliquer chaque maillon de la liste.
Retour	Rien.
Fonctions libc	Aucune.

ft_lstmap	
Prototype	`t_list * ft_lstmap(t_list *lst, t_list *(*f)(t_list *elem));`
Description	Parcourt la liste `lst` en appliquant à chaque maillon la fonction `f` et crée une nouvelle liste "fraiche" avec `malloc(3)` résultant des applications successives. Si une allocation échoue, la fonction renvoie `NULL`.
Param. #1	Pointeur sur le premier maillon d'une liste.
Param. #2	L'adresse d'une fonction à appliquer à chaque maillon de la liste pour crér une nouvelle liste.
Retour	La nouvelle liste.
Fonctions libc	`malloc(3)`, `free(3)`.

Si vous réussissez parfaitement la partie obligatoire et la partie bonus, vous êtes encouragés à ajouter d'autres fonctions qui vous paraissent utiles pour agrandir votre bibliothèque. Si votre correcteur les juge pertinentes, vous pourriez avoir des points supplémentaires. Exemples : une version de `ft_strsplit` qui renvoie une liste de chaines au lieu d'un tableau de chaines, la fonction `ft_lstfold` similaire à la fonction `reduce` de `Python` et à la fonction `List.fold_left` d'`OCaml` (attention aux fuites mémoires !), des fonctions de manipulation de tableaux, de piles, de files, de maps, de tables de hash, etc. La limite est votre imagination.

II.3 Rendu

- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
```

- Vous devez rendre un fichier C par fonction à réaliser ainsi qu'un fichier `libft.h` qui contiendra tous leurs prototypes ainsi que les `macros` et les `typedefs` dont vous pourriez avoir besoin. Tous ces fichiers devront se trouver à la racine de votre dépot.
- Vous devez rendre un `Makefile` qui compilera vos sources vers une bibliothèque statique nommée `libft.a`.
- Votre `Makefile` doit au moins proposer les règles `$(NAME)`, `all`, `clean`, `fclean` et `re` dans l'ordre qui vous paraîtra le plus adapté.
- Votre `Makefile` doit compiler votre travail avec les flags de compilation `-Wall`, `-Wextra` et `-Werror`.
- Afin de faciliter votre soutenance, vous devez également réaliser un ou des programmes de test pour votre bibliothèque. Bien que ce travail ne soit **pas à rendre sur votre dépot et ne sera pas évalué**, il vous permettra de tester facilement votre travail et celui des personnes dont vous ferez passer les soutenances. Vous êtes libres d'utiliser vos tests ou ceux du souteneur/soutenu voire même les deux si cela vous fait plaisir et la logistique sous-jacente est à votre discrétion. Il n'y a rien de pire que de ne pas avoir tous les points que vous méritez en soutenance parceque le correcteur n'a pas eu le temps de tout évaluer dans le temps imparti, non ? C'est votre travail, votre responsabilitée.
- Seul le contenu présent sur votre dépot sera évalué en soutenance.

II.4 Considérations techniques

- Votre fichier `libft.h` peut contenir des `macros` et des `typedefs` selon vos besoins.
- Une chaine de caractères est **TOUJOURS** terminée par un `'\0'`, même si cela a été omis dans la description d'une fonction. Dans le cas contraire, cela serait explicitement indiqué.
- 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.

`Savoir ce qu'est une fonction statique est un bon début :` http://codingfreak.blogspot.com/2010/06/static-functions-in-c.html

- Vous devez prêter attention à vos types et utiliser judicieusement les casts quand c'est nécéssaire, en particulier lorsqu'un type `void *` est impliqué. Dans l'absolu, évitez les casts implicites, quels que soient les types concernés. Exemple :

```
char    *str;

str = malloc(42 * sizeof(*str));            /* Wrong ! Malloc retourne un void * (cast implicite) */
str = (char *) malloc(42 * sizeof(*str));   /* Right ! (cast explicite) */
```

II.5 Fonctions autorisées

- malloc(3)
- free(3)
- write(2)

Vous devez bien entendu inclure l'`include` système nécessaire pour utiliser l'une ou l'autre des 3 fonctions autorisées dans votre fichier `.c` concerné. Le seul `include` système que vous êtes autorisés à utiliser en plus est `string.h` pour avoir accès à la constante `NULL` et au type `size_t`. Tout le reste est interdit.

Chapitre III

Consignes

- Vous pouvez coder les fonctions dans l'ordre que vous voulez et ne pas réussir une fonction n'empêche pas d'avoir les points pour les suivantes si celles-ci sont bonnes.
- Vous êtes vivement encouragés à faire appel à des fonctions que vous avez déja écrites pour écrire les suivantes.
- Votre projet doit être à la Norme. La Norminette ne sera pas utilisée pour vérifier la Norme qui s'applique donc dans son ensemble et sera vérifiée par un humain lors de la soutenance. Une faute de norme donne la note de `0` en soutenance.
- En aucun cas vos fonctions ne doivent quitter de façon inattendue (Segmentation fault, bus error, double free, etc) en dehors des comportements indéterminés. Votre projet serait alors considéré comme non fonctionnel et recevra la note de `0` en soutenance.
- Toute mémoire allouée sur le tas doit être libérée proprement quand nécéssaire.
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
```

- Vous ne devez jamais rendre de code que vous n'avez pas écrit vous-même. En cas de doute, vous serez invités à une séance de recode au bocal pour juger de votre bonne foi.

Chapitre IV

Notation

Le projet `libft` s'effectue sur deux semaines et s'effectuera en deux temps.

En premier lieu, une moulinette passera de temps en temps sur vos rendus afin d'évaluer la partie 1 de la partie obligatoire (page 4 - `libc`). Cette moulinette ne vous enverra pas de traces détaillées mais uniquement le nombre de fonctions réussies/échoués. Lorsque nous constaterons qu'un nombre que nous jugerons satisfaisant d'etudiants aura réussi toutes les fonctions de la partie 1, nous révèlerons à l'ensemble de la promotion le sujet du projet `Filler` censé se dérouler sur les deux même semaines que le projet `libft`.

Cela a plusieurs conséquences :

- Plus vous mettrez de temps à être suffisamment nombreux à avoir réussi la partie 1, moins vous aurez de temps pour réaliser le `Filler`.
- Vous n'aurez aucune information sur vos erreurs de la part de la moulinette. Vous devrez donc préparer vos propres ensembles de tests que nous vous encourageons à partager pour trouver vous-même vos erreurs et faciliter la soutenance.
- Vous n'aurez aucune aide de la part du staff.

En second lieu, vous aurez une soutenance classique à la fin du projet pour obtenir votre note. Lors de cette soutenance, vos bonus ne seront pris en compte que si vous obtenez au moins 18/20 à la partie obligatoire. L'optimisation de la qualité de certains éléments de votre code sera évaluée et pourrait donner lieu à des points supplémentaires dans cette partie bonus.

La partie obligatoire est sur 20 points et la partie bonus sur 22 points pour un total de 42 points maximum.

Bon courage à tous !