Langage C

LibString

18 décembre 2022

Version 1

Fabrice BOISSIER <fabrice.boissier@epita.fr>

Copyright

Table des matières

1 Consignes Générales

Les informations suivantes sont très importantes :

Le non-respect d'une des consignes suivantes entraînera des sanctions pouvant aller jusqu'à la multiplication de la note finale par 0.

Ces consignes sont claires, non-ambiguës, et ont un objectif précis. En outre, elles ne sont pas négociables.

N'hésitez pas à demander si vous ne comprenez pas une des règles.

Consigne Générale 0 : Vous devez lire le sujet.
Consigne Générale 1 : Vous devez respecter les consignes.
Consigne Générale 2 : Vous devez rendre le travail dans les délais prévus.

Consigne Générale 3 : Le travail doit être rendu dans le format décrit à la section Format de Rendu.

Consigne Générale 4 : Le travail rendu ne doit pas contenir de fichiers binaires, temporaires, ou d'erreurs (`*~`, `*.o`, `*.a`, `*.so`, `*#*`, `*core`, `*.log`, `*.exe`, binaires, ...).

Consigne Générale 5 : Dans l'ensemble de ce document, la casse (caractères majuscules et minuscules) est très importante. Vous devez strictement respecter les majuscules et minuscules imposées dans les messages et noms de fichiers du sujet.

Consigne Générale 6 : Dans l'ensemble de ce document, **`login.x`** correspond à votre login (donc **`prenom.nom`**).

Consigne Générale 7 : Dans l'ensemble de ce document, **`nom1-nom2`** correspond à la combinaison des deux noms de votre binôme (par exemple pour Fabrice BOISSIER et Mark ANGOUSTURES, cela donnera **`boissier-angoustures`**).

Consigne Générale 8 : Dans l'ensemble de ce document, le caractère ␣ correspond à une espace (s'il vous est demandé d'afficher ␣␣␣, vous devez afficher trois espaces consécutives).

Consigne Générale 9 : Tout retard, même d'une seconde, entraîne la note non négociable de 0.

Consigne Générale 10 : La triche (échange de code, copie de code ou de texte, ...) entraîne **au mieux** la note non négociable de 0.

Consigne Générale 11 : En cas de problème avec le projet, vous devez contacter le plus tôt possible les responsables du sujet aux adresses mail indiquées.

Conseil : N'attendez pas la dernière minute pour commencer à travailler sur le sujet.

2 Format de Rendu

Responsable(s) du projet :	**Fabrice BOISSIER** <fabrice.boissier@epita.fr>
Balise(s) du projet :	**[C][STR]**
Nombre d'étudiant(s) par rendu :	1
Procédure de rendu :	Devoir/Assignment sur Teams
Nom du répertoire :	`login.x-MiniProjet1`
Nom de l'archive :	`login.x-MiniProjet1.tar.bz2`
Date maximale de rendu :	18/12/2022 23h42
Durée du projet :	2 semaines
Architecture/OS :	Linux
Langage(s) :	C
Compilateur/Interpréteur :	**gcc**
Options du compilateur/interpréteur :	**-W -Wall -Werror -std=c99 -pedantic**

Les fichiers suivants sont requis :

`AUTHORS`	contient le(s) nom(s) et prénom(s) de(s) auteur(s).
`README`	contient la description du projet et des exercices, ainsi que la façon d'utiliser le projet.

Votre code sera testé automatiquement, vous devez donc scrupuleusement respecter les spécifications pour pouvoir obtenir des points en validant les exercices. Votre code sera testé en l'intégrant à une série de tests automatisés qui seront fournis un peu plus tard. N'attendez SURTOUT PAS que ces tests soient envoyés pour commencer à produire vos fonctions et vos propres tests.

L'arborescence attendue pour le projet est la suivante :

```
login.x-MiniProjet1/
login.x-MiniProjet1/AUTHORS
login.x-MiniProjet1/README
login.x-MiniProjet1/src/
login.x-MiniProjet1/src/StrBasics.c
login.x-MiniProjet1/src/StrBasics.h
login.x-MiniProjet1/src/StrPart1.c
login.x-MiniProjet1/src/StrPart1.h
login.x-MiniProjet1/src/StrPart2.c
login.x-MiniProjet1/src/StrPart2.h
```

3 Aide Mémoire

Le travail doit être rendu au format ***.tar.bz2***, c'est-à-dire une archive **bz2** compressée avec un outil adapté (voir **`man 1 tar`** et **`man 1 bz2`**).
Tout autre format d'archive (zip, rar, 7zip, gz, gzip, ...) ne sera pas pris en compte, et votre travail ne sera pas corrigé (entraînant la note de 0).

Pour générer une archive *tar* en y mettant les dossiers *folder1* et *folder2*, vous devez taper :

```
tar cvf MyTarball.tar folder1 folder2
```

Pour générer une archive *tar* et la compresser avec GZip, vous devez taper :

```
tar cvzf MyTarball.tar.gz folder1 folder2
```

Pour générer une archive *tar* et la compresser avec BZip2, vous devez taper :

```
tar cvjf MyTarball.tar.bz2 folder1 folder2
```

Pour lister le contenu d'une archive *tar*, vous devez taper :

```
tar tf MyTarball.tar.bz2
```

Pour extraire le contenu d'une archive *tar*, vous devez taper :

```
tar xvf MyTarball.tar.bz2
```

Pour générer des exécutables avec les symboles de debug, vous devez utiliser les flags **`-g -ggdb`** avec le compilateur. N'oubliez pas d'appliquer ces flags sur *l'ensemble* des fichiers sources transformés en fichiers objets, et d'éventuellement utiliser les bibliothèques compilées en mode debug.

```
gcc -g -ggdb -c file1.c file2.c
```

Pour produire des exécutables avec les symboles de debug, il est conseillé de fournir un script **`configure`** prenant en paramètre une option permettant d'ajouter ces flags aux **`CFLAGS`** habituels.

```
./configure
cat Makefile.rules
CFLAGS=-W -Wall -Werror -std=c99 -pedantic
./configure debug
cat Makefile.rules
CFLAGS=-W -Wall -Werror -std=c99 -pedantic -g -ggdb
```

Pour produire une bibliothèque statique *libtest.a* à partir des fichiers *test1.c* et *file.c*, vous devez taper :

```
cc -c test1.c file.c
ar cr libtest.a test1.o file.o
```

Pour produire une bibliothèque dynamique *libtest.so* à partir des fichiers *test1.c* et *file.c*, vous devez taper (pensez aussi à **-fpic** ou **-fPIC**) :

```
cc -c test1.c file.c
cc test1.o file.o -shared -o libtest.so
```

Pour compiler un fichier en utilisant une bibliothèque dont le **.h** se trouve dans un dossier spécifique, par exemple la *libxml2*, vous devez taper (pensez à vous assurer que les *includes* de la bibliothèque ont été entourés de chevrons **< >**) :

```
cc -c -I/usr/include test1.c
```

Pour lier plusieurs fichiers objets ensemble et avec une bibliothèque, par exemple la *libxml2*, vous devez d'abord indiquer dans quel dossier trouver la bibliothèque avec l'option **-L**, puis indiquer quelle bibliothèque utiliser avec l'option **-l** (n'oubliez pas de retirer le préfixe *lib* au nom de la bibliothèque, et surtout, que l'ordre des fichiers objets et bibliothèques est important) :

```
cc -L/usr/lib test1.o -lxml2 file.o -o executable.exe
```

Si la bibliothèque existe en version *dynamique* (**.so**) et en version *statique* (**.a**) dans le système, l'éditeur de lien choisira en priorité la version *dynamique*. Pour forcer la version *statique*, vous devez l'indiquer dans la ligne de commande avec l'option **-static** :

```
cc -static -L/usr/lib test1.o -lxml2 file.o -o executable.exe
```

Dans ce sujet précis, vous ferez du code en C et des appels à des scripts shell qui afficheront les résultats dans le terminal (donc des flux de sortie qui pourront être redirigés vers un fichier texte).

4 Exercice 1

Nom du(es) fichier(s) :	**StrBasics.c StrBasics.h**
Répertoire :	**login.x-MiniProjet1/src/**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions autorisées :	**malloc(3)**, **free(3)**

Objectif : Le but de l'exercice est de recoder les fonctions essentielles permettant de manipuler des chaînes de caractères : **my_strlen**, **my_strdup**, et **my_strcmp**

Vous devez implémenter ces fonctions en respectant strictement les spécifications suivantes. Vous ne devez ni rendre de fonction **main** ni vos tests : vous ne devez rendre *que* les fonctions demandées (et éventuellement celles vous permettant de les faire fonctionner) dans le fichier `StrBasics.c`, ainsi que leurs prototypes dans le fichier `StrBasics.h`.

Vous devez implémenter les fonctions suivantes :

```
int my_strlen(char *str);
char *my_strdup(char *str);
int my_strcmp(char *s1, char *s2);
```

int my_strlen(char *str)

Cette fonction calcule la taille d'une chaîne de caractères. La fonction doit renvoyer un entier qui correspond au nombre de caractères dans la chaîne (sans compter le '**\0**' final). Si la chaîne de caractère est vide (elle ne contient qu'un seul caractère : '**\0**'), alors la fonction doit renvoyer 0. Si le paramètre donné en paramètre est **NULL**, alors la fonction renverra -1.

char *my_strdup(char *str)

Cette fonction duplique la chaîne de caractère donnée en paramètre. La fonction doit renvoyer l'adresse d'un espace mémoire distinct contenant une copie de la chaîne donnée en paramètre. Cet espace mémoire doit avour été alloué avec **malloc(3)** (afin que l'utilisateur de votre fonction puisse le libérer avec **free(3)**). Si la chaîne de caractères donnée en paramètre est vide (le premier caractère est un '**\0**'), alors la fonction doit renvoyer **NULL**. Si **malloc(3)** n'arrive pas à allouer assez de mémoire pour stocker la nouvelle chaîne, alors la fonction doit renvoyer **NULL**.

int my_strcmp(char *s1, char *s2)

Cette fonction compare deux chaînes de caractères entre elles, caractère par caractère. La fonction retourne 0 si les deux chaînes de caractères sont égales. En cas d'inégalité, il s'agit de calculer la différence lexicographique entre **s1** et **s2**.

Ainsi, **`strcmp("ABCDE", "ABCDG")`** renverra -2, car '**`E`**' est deux crans avant '**`G`**', mais, **`strcmp("ABCDG", "ABCDE")`** renverra 2, car '**`G`**' est deux crans après '**`E`**'.
N'oubliez pas que les caractères sont déjà considérés comme des nombres.

5 Exercice 2

Nom du(es) fichier(s) :	**StrPart1.c StrPart1.h**
Répertoire :	**login.x-MiniProjet1/src/**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions autorisées :	**malloc(3)**, **free(3)**

Objectif : Le but de l'exercice est de recoder les fonctions essentielles permettant de manipuler des chaînes de caractères : **my_strchr**, **my_strrchr**, et **my_strcpy**

Vous devez implémenter les fonctions suivantes :

```
char *my_strchr(char *s, int c);
char *my_strrchr(char *s, int c);
char *my_strcpy(char *dest, char *src);
```

char *my_strchr(char *s, int c)

Cette fonction renvoie un pointeur vers la première occurrence d'un caractère dans une chaîne. La fonction doit renvoyer un pointeur vers le premier caractère retrouvé dans la chaîne **s**. Le caractère recherché est donné en paramètre en tant que **c**. Si le caractère n'est pas trouvé, la fonction doit renvoyer **NULL**.

char *my_strrchr(char *s, int c)

Cette fonction renvoie un pointeur vers la dernière occurrence d'un caractère dans une chaîne. La fonction doit renvoyer un pointeur vers le dernier caractère retrouvé dans la chaîne **s**. Le caractère recherché est donné en paramètre en tant que **c**. Si le caractère n'est pas trouvé, la fonction doit renvoyer **NULL**.

char *my_strcpy(char *dest, char *src)

Cette fonction copie une chaîne de caractères vers un espace de taille suffisant déjà alloué. La fonction lit les caractères de la chaîne **src** pour les copier dans l'espace mémoire pointé par **dest**. L'espace mémoire au bout de **dest** doit avoir été préalablement réservé par l'utilisateur appelant, afin de pouvoir y mettre tous les caractères de la chaîne contenue dans **src** ainsi que le '**\0**' final. La fonction doit renvoyer un pointeur vers la chaîne **dest**. *Si l'espace mémoire dans **dest** n'est pas assez grand, vous ne pouvez pas le savoir à l'avance, donc, vous devez essayer de copier normalement les caractères dedans. Si l'utilisateur ne respecte l'exigence de taille, ce n'est pas votre faute en tant que développeur : votre fonction peut échouer dans ce cas.*

6 Exercice 3

Nom du(es) fichier(s) :	**StrPart2.c StrPart2.h**
Répertoire :	**login.x-MiniProjet1/src/**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions autorisées :	**malloc(3)**, **free(3)**

Objectif : Le but de l'exercice est de recoder les fonctions essentielles permettant de manipuler des chaînes de caractères : **my_strcat**, **my_strtok_simple**, et **my_strstr**

Vous devez implémenter les fonctions suivantes :

```
char *my_strcat(char *dest, char *src);
char **my_strtok_simple(char *str, char delim);
char *my_strstr(char *haystack, char *needle);
```

char *my_strcat(char *dest, char *src)

Cette fonction ajoute une chaîne de caractère à la fin d'une autre chaîne de caractères (elles sont concaténées). La fonction lit les caractères de la chaîne **src** pour les copier à la fin de la chaîne **dest** (en écrasant son '**\0**'). L'espace mémoire au bout de **dest** doit avoir été préalablement réservé par l'utilisateur appelant, afin de pouvoir y ajouter tous les caractères de la chaîne contenue dans **src** ainsi que le '**\0**' final. La fonction doit renvoyer un pointeur vers la chaîne **dest**. *Si l'espace mémoire dans **dest** n'est pas assez grand, vous ne pouvez pas le savoir à l'avance, donc, vous devez essayer de copier normalement les caractères dedans. Si l'utilisateur ne respecte l'exigence de taille, ce n'est pas votre faute en tant que développeur : votre fonction peut échouer dans ce cas.*

char **my_strtok_simple(char *str, char delim)

Cette fonction découpe une chaîne de caractère selon un carcatère séparateur. La fonction doit renvoyer un tableau contenant des pointeurs vers des chaînes de caractères (comme **argv**), sachant que le dernier élément de ce tableau sera **NULL** (qui agira comme un '**\0**' dans une chaîne de caractères). Si la chaîne de caractère en entrée est vide (elle ne contient qu'un seul caractère : '**\0**'), alors la fonction doit renvoyer un tableau contenant une seule case qui elle-même contiendra la valeur **NULL**. Si le paramètre donné en paramètre est **NULL**, alors la fonction renverra **NULL**.
*Cette implémentation diffère de la fonction **strtok** implémentée dans la bibliothèque standard.*

`char *my_strstr(char *haystack, char *needle)`

Cette fonction recherche une sous-chaîne dans une chaîne de caractère. La fonction doit renvoyer l'adresse du premier caractère de la première sous-chaîne (**`needle`**) trouvée dans la chaîne principale (**`haystack`**). Si la sous-chaîne n'est pas trouvée, alors la fonction doit renvoyer **`NULL`**.
Cette implémentation diffère de la fonction ***`strtok`*** *implémentée dans la bibliothèque standard.*

7 Exercice 4 - BONUS

Nom du(es) fichier(s) :	**StrBonus.c StrBonus.h**
Répertoire :	**login.x-MiniProjet1/src/**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions autorisées :	**malloc(3)**, **free(3)**

Objectif : Le but de l'exercice est de recoder les fonctions vues jusqu'à maintenant dans leurs versions *n* (**strnlen**, **strndup**, **strncpy**, ...) en respectant cette fois les spécifications du manuel officiel, mais vous devez également implémenter les véritables spécifications de **strstr** et **strtok**.

Vous devez implémenter les fonctions suivantes :

```
int my_strnlen(char *str, int n);
char *my_strndup(char *str, int n);
int my_strncmp(char *s1, char *s2, int n);
char *my_strncpy(char *dest, char *src, int n);
char *my_strncat(char *dest, char *src, int n);

char *my_strtok(char *str, char *delim);
char *my_strstr(char *haystack, char *needle);
```

Accès aux exigences

Les exigences des fonctions se trouvent dans les *manuels* de documentation. Différentes sections existent dans les manuels, vous devrez utiliser la section 3 qui sert à décrire les bibliothèques. Ainsi, pour obtenir des informations sur **strnlen**, vous devrez entrer dans votre terminal la commande suivante : **man 3 strnlen**
Pour quitter un manuel, il suffit d'appuyer sur la touche **q** .