Langage C et Programmation Système
TP n° 2 : Chaînes de caractères en C

Exercice 0 : Hello, world !

Cet exercice est optionnel. Il peut être utile si vous ne vous êtes pas encore familiarisé avec le langage C.
Ouvrez un éditeur de texte (comme `emacs` ou `vim`) et entrez le code suivant :

```
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

Enregistrez le fichier sous le nom `hello.c`, compilez-le avec la commande

```
cc hello.c -o hello
```

puis exécutez le programme obtenu en utilisant la commande `./hello` (éventuellement après avoir modifié les droits d'exécution de `hello`).
À quoi sert l'option `-o hello` ? (Cf. `man cc`)

Exercice 1 : Implémentation de `echo`

1. Écrivez en C un programme `myecho` qui affiche le texte qu'il reçoit en paramètre. Plus précisément, le programme va recevoir un nombre arbitraire de chaînes de caractères, et il devra les afficher sur `stdout` (la sortie standard), séparées par des espaces (`" "`) et suivies d'un caractère de fin de ligne (`"\n"`).

La fonction `main`, à mettre dans un fichier `myecho.c`, sera de la forme suivante :

```
int main(int argc, char* argv[]) {
  ...
  return 0;
}
```

Le second paramètre, `argv` (« argument vector »), est un (pointeur sur un) tableau contenant (des pointeurs sur) les chaînes de caractères qui ont été passés au programme. Le premier paramètre, `argc` (« argument count »), est la longueur de ce tableau.

Pour afficher du texte, vous pourrez utiliser `printf` ou `fputs`. Si vous souhaitez lire la documentation de ces fonctions, tapez `man 3 printf` ou `man 3 fputs`, respectivement. (L'argument `3` signifie que l'on veut afficher une page de la section 3 du manuel Unix, qui, sous Linux, correspond aux fonctions de bibliothèque. Dans la suite, nous utiliserons une notation de la forme `printf(3)` pour référer à une telle page.)

2. Vérifiez que votre programme se comporte comme `echo` (sans options). Par exemple, `./myecho *` devrait vous afficher la liste des fichiers dans votre répertoire courant. Est-ce que `echo abc | wc -c` et `./myecho abc | wc -c` affichent la même chose ?

3. Écrivez une fonction `streq` qui teste si deux chaînes de caractères sont égales. Pour cela, créez un fichier d'en-tête `strlib.h` et un fichier de code `strlib.c`. Ajoutez d'abord la déclaration suivante dans `strlib.h` :

```
int streq(char s1[], char s2[]);
```

Puis, écrivez une implémentation de cette fonction dans `strlib.c`, tel que l'appel `streq(s1, s2)` retourne une valeur non nulle si et seulement si les chaînes référencées par `s1` et `s2` sont égales. (En C, les chaînes de caractères sont des tableaux de `char` qui se terminent par le caractère spécial `'\0'` ; en pratique, `'\0' == 0`.)

4. Modifiez votre programme `myecho` de façon à ce qu'il accepte les options `-s` et `-n`. Si ces options sont présentes, elles doivent précéder les autres paramètres passés au programme.

`-s` indique que l'on ne veut pas séparer les chaînes de caractères par des espaces.

`-n` indique que l'on ne veut pas terminer la sortie par un caractère de fin de ligne.

Afin de pouvoir utiliser la fonction `streq` de la question précédente, ajoutez la ligne suivante au début de `myecho.c` :

```
#include "strlib.h"
```

Pour compiler le programme, il faut maintenant ajouter `strlib.c` aux arguments passés au compilateur. (Il faut indiquer au compilateur où se trouve le code de `streq`.)

```
cc myecho.c strlib.c -o myecho
```

5. Testez la nouvelle version de `myecho` en tapant les deux commandes suivantes :

```
./myecho -s -n s i n g > test.txt
./myecho le line >> test.txt
```

Si votre programme fonctionne correctement, le contenu de `test.txt` est :

```
single line
```

Exercice 2 : Chiffre de César

Écrivez le programme `rot13.c` suivant, puis compilez-le, et interagissez avec. (Il attend qu'on entre du texte sur son entrée standard.)

```
#include <stdio.h>
#include <string.h>

int main() {
    int i;
    char s[64];
    fgets(s, 64, stdin);
    for(i = 0; i < strlen(s); i++) {
        if (97 <= s[i] && s[i] <= 122)
            s[i] = (((s[i] - 97) + 13) % 26) + 97;
    }
    fputs(s, stdout);
    return 0;
}
```

1. Exponentielle et Logarithme vont ensemble au restaurant. Lequel invite l'autre ?

   ```
   rkcbaragvryyr, pne ybtnevguzr aécéevra.
   ```

2. Trouvez dans la page man `ascii(7)` les codes ASCII décimaux pour les lettres 'a' et 'z'.
3. Expliquez ligne par ligne ce que fait le programme. Rendez la condition d'arrêt de la boucle efficace.
4. Améliorez le programme pour qu'il transforme aussi les majuscules.
5. Que se passe-t-il si on entre plus de 64 caractères ? Améliorez le programme pour qu'il fonctionne sur des textes de longueur arbitraire. (Piste : Quelle est la valeur de retour de `fgets` ? Cf. la page `gets(3)`.)
6. Produisez une version chiffrée de la « GNU General Public License » (`http://www.gnu.org/licenses/gpl.txt`), qui, au passage, est la licence d'utilisation du compilateur `gcc`. (Sur la plupart des installations Linux, `cc` est un lien symbolique vers `gcc`.)

*. *Exercice optionnel* : En vous inspirant du code de `rot13.c`, écrivez une implémentation (éventuellement partielle) de la commande Unix `tr` qui permet de transcoder un alphabet en un autre (cf. la page `tr(1)`). Comment peut-on simuler `rot13` avec `tr` ?

Exercice 3 : Tri rapide

Dans cet exercice, vous allez écrire un programme qui trie les caractères de chaque ligne de texte de son entrée standard.

1. Implémentez l'algorithme de tri rapide (« quicksort ») dans le fichier `strlib.c` que vous avez créé dans l'exercice 1. Le résultat devra être une fonction avec la signature suivante :

   ```
   void quicksort(char s[], int left, int right)
   ```

   Après l'appel de fonction `quicksort(s, left , right)`, les caractères de la chaîne `s` devront être triés par ordre croissant de leur code ASCII entre les positions `left` et `right`. Vous pouvez écrire des fonctions auxiliaires si cela vous semble utile.
2. Servez-vous de `quicksort` pour écrire une fonction `void strsort(char s[])` qui trie les caractères d'une ligne de texte contenue dans le tableau `s`. Que faut-il faire avec les caractères `'\n'` et `'\0'` ? Mettez le code de `strsort` dans `strlib.c` et une déclaration correspondante dans `strlib.h`.
3. Compilez `strlib.c` en utilisant la commande

   ```
   cc -c strlib.c
   ```

   qui vous crée le fichier objet `strlib.o`. Il s'agit d'un fichier intermédiaire contenant la traduction en code machine de votre code C. Vous ne pouvez pas l'exécuter directement.
4. Créez un fichier `csort.c` (« character sort ») qui inclut `strlib.h` et dont la fonction `main` procède de la manière suivante : Chaque ligne de `stdin` est lue individuellement, puis triée avec `strsort`, et finalement affichée sur `stdout`. Vous pouvez supposer que la longueur de chaque ligne est inférieure à 1024 caractères.

5. Compilez votre programme avec la commande

```
cc csort.c strlib.o -o csort
```

et testez-le sur des exemples (essayez aussi avec des chiffres). Remarquez que vous avez utilisé le code machine de `strlib.o`, au lieu de le recompiler à partir du code source de `strlib.c`.