Cours de programmation séquentielle

Chaînes de caractères

1 Buts

— Utilisation des tableaux unidimensionnels.
— Utilisation des chaînes de caractères.
— Écriture de librairie de fonctions.

2 Énoncé

Une chaîne de caractère est un tableau unidimensionnel de `char` se terminant par le caractère `'\0'` ou l'entier `0`. Le but de ce travail pratique est d'écrire une librairie `string.h` permettant de manipuler des chaînes de caractères. Puis dans un deuxième temps, il s'agira écrire un programme lisant des chaînes de caractères à la ligne de commande afin de démontrer des cas d'utilisations votre code.

2.1 La librairie `string.h`

Votre librairie doit contenir les fonctions suivantes:

1. Ajoute la chaîne de caractère `src` à la fin de la chaîne de caractères `dest` et retourne un pointeur vers `dest`.
   ```
   char *string_cat(char *dest, const char *src);
   ```
2. Retourne un pointeur vers la première occurrence du caractère `c` dans la chaîne de caractère `s`
   ```
   char *string_chr(const char *s, char c);
   ```
3. Compare deux chaînes de caractères
   ```
   bool string_cmp(const char *s1, const char *s2);
   ```
4. Copie la chaîne de caractères `src` dans la chaîne de caractères `dest` et retourne un pointeur vers le début de `dest`.
   ```
   char *string_cpy(char *dest, const char *src);
   ```
5. Retourne une copie de `s` dans une nouvelle chaîne de caractères allouée avec `malloc()`.
   ```
   char *string_dup(const char *s);
   ```
6. Retourne la longueur de la chaîne de caractères `s`.

```
size_t string_len(const char *s);
```

7. Copie au plus `n` caractères de la chaîne de caractères `src` dans la chaîne de caractères `dest`, et retourne un pointeur au début de `dest`.
   ```
   char *string_ncat(char *dest, const char *src, size_t n);
   ```

8. Compare au plus `n` octets des chaînes de caractères `s1` et `s2`.
   ```
   bool string_ncmp(const char *s1, const char *s2, size_t n);
   ```

9. Copie au plus `n` octets de la chaîne de caractères `src` à la chaîne de caractères `dest` et retourne un pointeur au début de `dest`.
   ```
   char *string_ncpy(char *dest, const char *src, size_t n);
   ```

10. Retourne le nombre d'occurrences du caractère `c` dans la chaîne de caractères `src`.
    ```
    size_t string_cnt_chr(const char *src);
    ```

11. Extrait la première sous-chaîne de caractères de `src`, délimitée par le caractère `delim` et la retourne. La chaîne de caractère `src` est modifiée.
    ```
    char *string_sep(char **src, char delim);
    ```

12. Échange aléatoirement de place les caractères contenus dans la chaîne de caractères `src`
    ```
    void string_fry(char *src);
    ```

2.1.1 Tests unitaires

Vos fonctions doivent toutes être munies de tests unitaires que vous implémenterez à l'aide de la librairie `cunit`. Vous trouverez un exemple d'utilisation sur `cyberlearn`, ainsi que sur le repo `git` https://gitedu.hesge.ch/pierre.kunzli/exemple-cunit.git. Afin de cloner le repo exécuter la fonction

```
git clone https://gitedu.hesge.ch/pierre.kunzli/exemple-cunit.git
```

Pour construire vos tests, vous pouvez, si vous le désirez, utiliser les fonction de la librairie `<string.h>` pour vérifier le bon fonctionnement de vos fonctions. En effet, une très grande partie des fonctions que vous devez écrire existent dans la librairie C, `<string.h>`.

2.1.2 Les pointeurs `const`

Il existe une nouvelle notation dans ce travail pratique. En effet, un certain nombre de fonctions prennent un des arguments de type `const char *`. Une variable

```
const char *p;
```

est un pointeur vers une constante. Cela signifie, que la cible du pointeur ne peut pas être modifiée, le pointeur lui-même peut être modifié.

Par opposition, on peut créer des pointeur constants sur des variables avec la syntaxe

```
char *const c;
```

Ici, le pointeur ne peut être modifié, mais la cible du pointeur peut l'être.

Finalement, on peut rendre constant aussi bien la cible du pointeur que le pointeur avec la syntaxe

```
const char *const c;
```

2.2 Programme

Écrire un programme, `string_manip.c`, qui lit deux chaînes de caractères à la ligne de commande. Vérifier que toutes les fonctions écrites précédemment fonctionnent comme souhaité en affichant les résultats à l'écran.

2.3 Compilation

Pour compiler vous devez écrire un `Makefile` contenant au moins deux cibles: - une cible `string_manip` qui compile votre programme principal et produit l'exécutable `string_manip`. - une cible `tests` qui exécute tous les tests.