Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

TP 16 | bibliothèque `chaines`

Présentation

Dans ce TP, on veut écrire notre propre bibliothèque `chaine`, qui regroupera des fonctions permettant de manipuler des chaînes de caractères. Une bibliothèque est un ensemble de fonctions proches, dans le sens où elles permettent généralement de manipuler les mêmes données. Par exemple, `stdio` se charge des entrées-sorties.

En langage C, une bibliothèque `xxxx` se compose de deux fichiers : le fichier `xxxx.h` contenant les en-têtes des fonctions, les types et les constantes ; et le fichier `xxxx.c` qui contient le corps des fonctions.

Dans les TP où vous devrez définir une bibliothèque, soyez bien attentif à l'emplacement des fonctions, qui vous sera indiqué dans le sujet : soit dans le fichier `main.c`, soit dans la bibliothèque. **Attention :** dans ce dernier cas, cela signifie que les deux fichiers `xxxx.c` et `xxxx.h` doivent être mis à jour. Si l'emplacement n'est pas précisé, alors il faut écrire la fonction dans `main.c`. Dans les deux cas, n'oubliez pas que chaque fonction écrite doit être testée à partir de la fonction `main`.

1 Préparation des fichiers

Exercice 1

Dans votre projet Eclipse, créez les fichiers suivants :

- `main.c` : fichier principal qui contiendra la fonction principale `main`.
- `chaine.h` : fichier d'en-tête (ou *header*) de la bibliothèque.
- `chaine.c` : fichier de corps de la bibliothèque.

Exercice 2

Dans le fichier d'en-tête, copiez-collez le code source suivant, qui permet d'empêcher la bibliothèque d'être chargée plusieurs fois lors de la compilation :

```
#ifndef CHAINE_H_
#define CHAINE_H_

// TODO a completer ici

#endif /* CHAINE_H_ */
```

Dans la suite, ce fichier devra contenir ;

- Les inclusions des bibliothèques utilisées par les fonctions contenue dans notre propre librairie, comme par exemple `#include <stdio.h>` ;
- Les déclarations de constantes (`#define` …), et plus tard de types de données ;
- Les en-têtes des fonctions contenues dans notre bibliothèque.

Le code source correspondant à ces 3 sortes d'éléments sera à rajouter, dans l'ordre indiqué, à la place du commentaire `"// TODO a completer ici"` représenté en rouge dans le code source que vous avez copié-collé.

Exercice 3

Dans le fichier `chaine.c`, rajoutez la ligne suivante :

```
#include "chaine.h"
```

Cette instruction indique que le fichier `chaine.c` est lié au fichier `chaine.h`, et en particulier qu'il utilise les bibliothèques qui y sont incluses, et les constantes et types qui y sont déclarés.

Donc, faites la même chose dans le fichier main.c : cela permettra à votre fonction main d'utiliser les bibliothèques, fonctions, types et constantes utilisées ou définies dans la bibliothèque `chaine`.

Remarque : vous noterez qu'à la différence des utilisations précédentes de `#include`, comme par exemple quand on fait `#include <stdio.h>`, ici le nom de la bibliothèque est entouré de *guillemets* (`"xxx"`) et non pas de *crochets* (`<xxx>`) : cela est dû au fait qu'il s'agit d'une bibliothèque définie *localement* (i.e. dont le code source se trouve dans le projet), et non pas d'une bibliothèque standard localisée au même endroit que le compilateur.

2 Fonctions existantes

Les fichiers sont prêts, et on veut maintenant commencer à rajouter des fonctions à notre bibliothèque. Dans le reste du sujet, pour chaque fonction demandée, vous devez d'abord écrire son en-tête dans `chaine.h`, puis la fonction complète (corps *et* en-tête) dans `chaine.c`. De plus, vous devez tester chaque fonction à partir de la fonction main, comme d'habitude.

Remarque : dans chaine.h, chaque en-têtes doit être suivie d'un *point-virgule* `';'`.

Comme expliqué au début du sujet, une bibliothèque se compose de fonctions traitant toutes du même thème. Ici, il s'agit de fonctions portant sur les chaînes de caractères. Or, nous avons déjà fait un TP sur les chaînes de caractères : nous allons donc commencer par intégrer ce travail précédent à notre bibliothèque. Les fonctions de cette section correspondent à des exercices de ce TP, veuillez consulter sa correction pour les obtenir, et le sujet lui-même si vous voulez plus de détails sur le comportement des fonctions.

Remarque : attention aux en-têtes du présent TP, qui sont légèrement différentes de celles du TP passé.

Exercice 4

Écrivez une fonction `int mesure_chaine(char* chaine)` qui reçoit une chaîne de caractères `chaine` et calcule sa longueur, sans compter le caractère de fin de chaîne `'\0'`.

Exercice 5

Écrivez une fonction `void copie_chaine(char* chaine1, char* chaine2)` qui recopie la chaine de `chaine1` dans `chaine2`. On supposera que `chaine2` correspond à un tableau dont la taille est suffisante pour y recopier `chaine1`.

Exercice 6

Écrivez une fonction `int compare_chaines(char* chaine1, char* chaine2)` qui reçoit deux chaînes et les compare en utilisant l'ordre lexicographique de la table ASCII. La fonction doit renvoyer -1 si la `chaine1` est située avant `chaine2` dans l'ordre lexicographique, 0 si `chaine1` et `chaine2` sont exactement les mêmes, et $+1$ si `chaine1` est situé après `chaine2` dans l'ordre lexicographique.

Exercice 7

Écrivez une fonction `void inverse_chaine(char* chaine)` qui inverse la chaîne de caractère chaine passée en paramètre. Le traitement doit être réalisé *sur place*. Autrement dit : vous devez travailler directement dans `chaine`.

3 Nouvelles fonctions

Exercice 8

Écrivez une fonction `void supprime_majuscules(char* chaine)` qui prend en argument une chaine de caractères `chaine` et qui supprime toutes les majuscules qu'elles contient. Les caractères qui ne sont pas des lettres ne sont pas modifiés. On suppose que la chaine originale ne contient pas d'accents.

exemple : pour `chaine="afegAEfd CDghj!"`, après avoir appelé la fonction, on obtient la chaine `"afegfd ghj!"`.

Remarque : votre fonction ne doit pas utiliser de tableau supplémentaire lors du traitement. Autrement dit, aucun tampon (buffer) n'est autorisé, vous devez travailler directement sur la chaine).

Exercice 9

Écrivez une fonction `void remplace_majuscules(char* chaine)` similaire à `supprime_majuscules`, avec la différence qu'au lieu de supprimer les majuscules, elle les *remplace* par des minuscules.

exemple : pour `chaine="afegAEfd CDghj!"`, après avoir appelé la fonction, on obtient la chaine `"afegaefd cdghj!"`.

Exercice 10

Écrivez une fonction `int compte_espaces(char *chaine)` qui calcule le nombre d'espaces `' '` contenus dans la chaîne de caractères passée en paramètre.

exemple : pour `chaine="un deux trois"`, la fonction doit retourner 2.

Exercice 11

Écrivez une fonction `int compte_mots(char *chaine)` qui calcule le nombre de mots contenus dans la chaîne de caractères `chaine`. On supposera que les mots peuvent être séparés par un ou plusieurs espaces, et qu'il peut y avoir des espaces au début et à la fin de la chaîne.

exemples :

- `compte_mots("un deux trois")` retournera la valeur 3 ;
- `compte_mots("un  deux  trois")` retournera aussi la valeur 3.
- `compte_mots("  un deux  trois")` retournera aussi la valeur 3.
- `compte_mots("  un deux  trois  ")` retournera aussi la valeur 3.

Exercice 12

On dit qu'une chaîne de caractères `chaine1` est un *préfixe* d'une autre chaîne `chaine2` ssi la chaîne `chaine1` correspond exactement au début de la chaîne `chaine2`.

exemples :

- `"bon"` est un préfixe de `"bonjour"`.
- `"bonne"` n'est pas un préfixe de `"bonjour"`.
- `"bonne"` n'est pas un préfixe de `"bon"`.
- `""` (chaîne vide) est préfixe de n'importe qu'elle chaîne.

Écrivez une fonction `int est_prefixe(char* chaine1, char* chaine2)` qui renvoie 1 si `chaine2` est un préfixe de `chaine1`, et 0 sinon.