

TP : structures et chaînes de caractères

Mots et textes de taille bornée

En C, il est d'usage de terminer une chaîne de caractères par `'\0'`. Cela permet notamment de l'afficher avec `printf` et de l'utiliser avec les fonctions de `<string.h>`.

Nous conservons cette convention, tout en mémorisant la longueur du mot dans une structure. Le champ `size` permet d'obtenir cette longueur en $O(1)$, sans parcourir le mot comme le fait `strlen`.

```
1 #include <stddef.h>
2
3 #define MAX_SIZE 100
4 #define MAX_TXT_SIZE 20
5
6 struct string {
7     size_t size;
8     char mot[MAX_SIZE + 1];
9 };
10 typedef struct string string;
11
12 typedef struct {
13     size_t size;
14     string mots[MAX_TXT_SIZE];
15 } text;
```

Dans une `string`, le tableau `mot` peut contenir au plus `MAX_SIZE` caractères, suivis de `'\0'`. On supposera toujours que `s.size <= MAX_SIZE` et que `s.mot[s.size] == '\0'`.

De même, les mots d'un `text` occupent les cases d'indices 0 à `size - 1` du tableau `mots`. On supposera que cette taille ne dépasse pas `MAX_TXT_SIZE`.

Après avoir écrit chaque fonction, écrire une fonction de test nommée `test_f` pour une fonction nommée `f`. La fonction principale ne contiendra que des appels à ces tests, suivis de `return 0;`.

Question 1.

Écrire une fonction `string creer(const char mot[])` qui construit une `string` à partir d'une chaîne C. On supposera que celle-ci contient au plus `MAX_SIZE` caractères.

Par exemple, `creer("chat")` construit une structure dont le champ `size` vaut 4 et dont le tableau contient `"chat"`.

```
mot : chat
taille : 4
```

Question 2.

Écrire une fonction `void afficher(const string *s)` qui affiche le mot avec `printf` et le spécificateur `%s`. La fonction n'ajoute pas de retour à la ligne.

```
chat
```

Question 3.

Écrire une fonction `int egales(const string *a, const string *b)` qui renvoie 1 si les mots sont égaux et 0 sinon. Commencer par comparer leurs longueurs, puis comparer leurs caractères. Ne pas utiliser `strcmp`.

```
string chat = creer("chat"), chut = creer("chut");
egales(&chat, &chat) = 1
egales(&chat, &chut) = 0
```

Question 4.

Écrire une fonction `int comparer(const string *a, const string *b)` qui renvoie un entier négatif si *a* précède *b* dans l'ordre lexicographique, nul s'ils sont égaux, et positif si *a* suit *b*. Comparer les caractères un à un, sans utiliser `strcmp`. Si un mot est un préfixe de l'autre, le plus court précède le plus long.

```
string chat = creer("chat"), chatte = creer("chatte"), chien=creer("chien");
comparer(&chat, &chien) < 0
comparer(&chat, &chat) = 0
comparer(&chatte, &chat) > 0
```

Pour cette question, on utilise l'ordre des codes des caractères, sans traitement particulier des majuscules ou des accents.

Question 5.

Écrire une fonction `int contient(const string *s, char c)` qui renvoie 1 si le caractère *c* apparaît dans le mot, et 0 sinon.

```
contient(&chat, 'a') = 1
contient(&chat, 'z') = 0
```

Question 6.

Écrire une fonction `void inverser(string *s)` qui inverse sur place l'ordre des caractères du mot. Sa longueur ne change pas et le mot reste terminé par `'\0'`.

```
Avant : chat
Après : tahc
```

Question 7.

Écrire une fonction `int palindrome(const string *s)` qui renvoie 1 si le mot se lit de la même façon dans les deux sens, et 0 sinon. Elle ne doit pas modifier son argument.

```
string chat = creer("chat"), radar = creer("radar");
palindrome(&radar) = 1
palindrome(&chat) = 0
```

Question 8.

Écrire une fonction `int ajouter(string *s, const string *suite)` qui ajoute *suite* à la fin de *s*. Elle renvoie 1 si l'ajout a été effectué, et 0 si le mot obtenu dépasserait `MAX_SIZE` caractères. En cas d'échec, *s* doit rester inchangé. On supposera que *s* et *suite* désignent deux objets distincts.

```
Avant : bon
Suite : jour
Après : bonjour
Retour : 1
```

Question 9.

Écrire une fonction `void afficher_texte(const text *t)` qui affiche les mots d'un texte, séparés par des espaces, puis un retour à la ligne.

Question 10.

Écrire une fonction `void trier(text *t)` qui trie les mots d'un texte dans l'ordre lexicographique par tri à bulle. Utiliser la fonction `comparer` pour décider si deux mots doivent être échangés.

```
Contenu du texte avant : chien chat arbre chats
Contenu du texte après : arbre chat chats chien
```