UNIVERSITÉ
CÔTE D'AZUR

Université Côte d'Azur
Programmation C

PO2 SITE

Nom
Prénom
Né(e) à
Le

Il est de votre responsabilité de rabattre le triangle grisé et de le cacheter au moyen de colle, agrafes ou papier adhésif. Si ne vous le faites pas, vous acceptez implicitement que votre copie ne soit pas anonyme.

# Seconde session : Programmation C

## 12 juin 2023

**Durée :** 1 heure
Tous documents autorisés. Il est interdit d'accéder à internet.

| Note |
| --- |
| |

Toutes les questions sont indépendantes.
Tous les codes devront être écrits en **Langage C** ANSI. La notation est donnée à titre indicatif.
Nombre de pages : 6

## Exercice 1 : sur les chaînes et les tableaux (8 points)

1. Écrire une fonction `longueur` qui renvoie la longueur d'une chaîne donnée en paramètre.

2. Écrire une fonction `indice_plus_longue` qui prend en paramètre un tableau non vide de chaîne de caractères et un entier représentant la longueur de ce tableau et qui renvoie l'indice du tableau contenant la plus longue chaîne.

3. Écrire une fonction `echange` qui prend en argument un tableau de chaînes de caractères ainsi que deux indices et échange dans ce tableau les deux chaînes correspondantes aux indices. Par exemple, avec un tableau `tab` contenant cinq valeurs : | `"Boby"` | `"Kiky"` | `"Yuki"` | `"Maxy"` | `"Doly"` |, l'appel de `echange(tab,1,3)` sur ce dernier le transfome en | `"Boby"` | `"Maxy"` | `"Yuki"` | `"Kiky"` | `"Doly"` |. La fonction ne renverra rien.

4. Que fait le code suivant ? À quoi correspond `tab` et `*tab` ?

```c
int mystere(int *tab, int n){
  int i; int bouledegomme = 0;
  for (i=0 ; i<n ; i++) {
    bouledegomme += *tab;
    tab++;
  }
  return bouledegomme;
}
```

5. En utilisant les fonctions précédentes, écrire une fonction `tri` prenant deux arguments : un tableau de chaînes de caractères et un entier représentant la taille du tableau et le trie selon la longueur des chaînes. Le tableau en argument devra être modifié et la fonction ne devra rien renvoyer.

## Exercice 2 : sur les multensembles (12 points)

Un multensemble est une collection dans laquelle l'ordre des éléments n'importe pas mais où une même valeur peut apparaître plusieurs fois. Par exemple $\{3, 2, 1\} = \{1, 3, 2\}$ mais $\{1, 2, 3, 3\} \neq \{1, 2, 2, 3\} \neq \{1, 2, 3\}$. Pour coder un multensemble, nous allons stocker les valeurs dans un tableau en ajoutant leur multiplicité. Par exemple le multensemble {15.5, 3.25, 15.5, 1.0} sera stocké comme le tableau

| `(15.5,2)` | `(3.25,1)` | `(1.0,1)` |
|---|---|---|

; effectivement, la valeur `15.5` apparait 2 fois, `3.25` et `1.0` n'apparaissent qu'une seule fois.

Le problème, c'est que les couples n'existent pas en C. Nous allons commencer par contruire des couples dont le premier élément est un flottant et le second un entier.

1. Créer un nouveau type `couple`. Cette structure aura deux champs : un champs **`valeur`** de type flottant et un champs **`nombre`** de type entier. Par exemple si `valeur` vaut `15.5` et `nombre` vaut 3, cela signifiera que le nombre `15.5` apparaît trois fois dans notre multensemble.

2. Écrire une fonction `nouveau_couple` qui prend en paramètre un flottant `x` et renvoie un `couple` de valeur `x` . Par défaut, le champs **`nombre`** sera initialisé à `1`.

3. Créer maintenant un nouveau type `multensemble`. Un `multensemble` sera défini à partir d'une structure contenant trois champs :
   - **`mem`** : un tableau de `couple`. Le tableau sera alloué sur le tas.
   - **`taille`** : un entier correspondant à la taille du tableau.
   - **`n`** : un entier correspondant au nombre de cases du tableau effectivement utilisées.

4. Créer maintenant une fonction `nouveau_multensemble` qui ne prend aucun paramètre et renvoie un pointeur vers un `multensemble` alloué sur le tas. Le tableau contiendra dix cases (**`taille`**=10) mais restera vide (**`n`**=0).

5. Si on souhaite ajouter un élément dans le tableau et qu'il n'y a plus de place (lorsque le champs **`n`** est égal au champs **`taille`**), il faut augmenter la taille du tableau en réallouant de l'espace sur le tas. Écrire une fonction `agrandir` prenant un pointeur de `multensemble` en paramètre et qui remplace l'ancien tableau pointé par **`mem`** par un nouveau deux fois plus grand (tout en conservant les données). On fera bien attention à libérer la mémoire de l'ancien tableau qui ne sera plus utilisée.

6. Écrire une fonction `est_present` qui prend en paramètre un pointeur vers un `multensemble` `e` et un flottant `x` et qui renvoie `1` si un élément de `e` possède `x` comme valeur. Sinon la fonction renverra `0`.

7. Écrire une fonction `ajout` qui prend en paramètre un pointeur vers un `multensemble` et un flottant `x` et ajoute ce dernier au `multensemble`.
   - Si un couple de **`valeur`** égale à `x` se trouve déjà dans le tableau, on incrémente le champs **`nombre`** correspondant.
   - Sinon, on ajoute un nouveau couple correspondant à `x` à la fin du tableau. On pourra utiliser la fonction de la question 2. On fera bien attention d'avoir la place nécessaire pour ajouter le nouveau couple, quitte à utiliser la fonction `agrandir` (uniquement si nécessaire).

8. Écrire une fonction `nettoyage` qui prend en argument un pointeur vers un `multensemble` et libère toute la mémoire correspondante.

9. En utilisant les fonctions précédentes, écrire une fonction `main` dans laquelle vous créerez un multensemble contenant les valeurs de `0.0` à `20.0` avec un pas de `0.5` (c'est-à-dire `0.0, 0.5, 1.0, 1.5, …, 19.5, 20`) et où les valeurs entières (`0.0, 1.0, 2.0, …`) sont présentes chacune deux fois (les autres ne seront présentes qu'une fois). On libérera proprement la mémoire avant de quitter le programme avec un code de retour indiquant un succès.