Programmation Orientée Objet - C++ - TP 1
Premier pas en C++

On va commencer en douceur... Ce TP a pour but de vous faire découvrir le C++. La syntaxe est proche du C, mais comme on a vu en cours, il y a beaucoup de différences. Écoutez bien votre chargé de TP concernant la compilation !

TP à faire en monôme ou binôme et à déposer sur elearning : un dossier nommé par votre/vos nom(s) ne contenant **QUE** les fichiers sources (un exercice = un fichier).
Vous pouvez coder avec vos machines persos, sous l'OS que vous voulez et avec l'IDE que vous voulez. Par contre, *in fine*, le code doit être compilable et exécutable sur les machines de la fac ! Attention !

Exercice 1 : IMAC Power

Objectifs : compiler un programme ; afficher un message sur la sortie standard (`std::cout`).

Créez un fichier `TP1_ex1.cpp` contenant la fonction `main` classique. La fonction `main` doit afficher « Les IMACs sont des brutes de C++ » et ne retourner aucune erreur.

Exercice 2 : Tu sors ou je te sors ?

Objectif : découvrir les flux d'entrée/sorties (`std::cout`, *etc.*).

D'après vous, quel nom doit porter le fichier que vous devez créer ?

Codez un programme qui demande un entier à l'utilisateur et qui affiche :
— « Parfait » sur la sortie standard si l'entier est égal à 79 ;
— « Négatif » sur la sortie standard si l'entier est négatif ;
— « Strictement positif » sur la sortie d'erreur si l'entier est strictement positif.

Exercice 3 : Une chaîne, comme un bon camembert (de caractère(s))

Objectifs : manipuler des chaînes de caractères (`std::string`) ; utiliser la documentation (apprendre en codant).

1. Demandez une chaîne de caractères à l'utilisateur.
2. En n'utilisant qu'un seul `std::cout`, affichez le nombre de caractères et le dernier caractère tel que :
   `String size : X`
   `Last element : Y`.
3. Effacez le dernier caractère et vérifiez le résultat.
4. Ajoutez « IMAC » au début de la chaîne.

Pour aller plus loin et faire un minimum d'algo (faites ça une fois le TP terminé) :

5. Inversez la chaîne de caractères (abcd → dcba). L'inversion doit se faire « en place » *i.e.* n'utilisez pas d'autre(s) chaîne(s) de caractères. Proposez deux versions, l'une utilisant `std::swap`, l'autre non.
6. Mettez les voyelles en majuscule et les consonnes en minuscule. Utilisez une fonction codée par vos soins qui vérifie si le caractère est une voyelle et retourne un booléen (`bool`).
7. Parcourez la doc et amusez vous !

Exercice 4 : Soyons dynamiques !

Objectifs : découvrir les mécanismes d'allocation en C++ (`new`, `delete`) ; manipuler des adresses/pointeurs ; éviter les problèmes avec Chuck Norris.

1. Créez un tableau d'entiers dont la taille et les données sont définies par l'utilisateur.
2. Affichez l'adresse mémoire des trois premiers éléments ainsi que la taille d'un élément. Concluez sur la manière dont le tableau est stocké en mémoire.
3. Affichez le premier élément (sans utiliser l'opérateur `[]`).
4. Affichez le cinquième élément (comme dans le film, mais toujours sans utiliser l'opérateur `[]`).
5. Codez une fonction `maxValue` permettant de déterminer le plus grand élément du tableau.
6. N'oubliez pas de libérer la mémoire !

Exercice 5 : Les vecteurs, ça change la vie !

Objectifs : découvrir les espaces de noms ; découvrir les vecteurs de la STL (`std::vector`) ; passer un paramètre par référence ; comprendre l'intérêt des références.

1. Téléchargez le fichier `TP1_ex5.cpp`. À partir de maintenant nous coderons uniquement dans l'espace de nom `TP_CPP_IMAC2`.
2. Dans la fonction `TP_CPP_IMAC2::main`, créez un vecteur d'entiers vide.
3. Demandez à l'utilisateur la taille du vecteur puis ajoutez autant d'éléments (avec la fonction `push_back`). L'élément `i` doit avoir pour valeur `i % 10`.
4. Affichez la taille du vecteur.
5. Affichez l'adresse mémoire des trois premiers éléments ainsi que la taille d'un élément. Concluez sur la manière dont le tableau est stocké en mémoire.
6. Supprimez le dernier élément.
7. Effacez le contenu du vecteur.

Exercice 6 : De l'utilité des références

1. Reprenez le code de l'exercice précédent.
2. Codez une fonction `mean` permettant de calculer la moyenne des éléments du vecteur. Vous passerez le vecteur par référence.
3. Codez une fonction `meanCopy` permettant de calculer la moyenne des éléments du vecteur. Vous passerez le vecteur par copie.
4. Utilisez la classe `Chrono` du fichier `chrono.hpp` pour mesurer les temps d'exécution des deux fonctions précédentes. Quelle est la différence de mémoire utilisée ? Concluez sur l'intérêt d'utiliser des références pour passer des paramètres. Voici comment utiliser la classe `Chrono` :

```
Chrono chrono;
chrono.start();
// code
chrono.stop();
std::cout << "Time: " << chrono.timeSpan() << " s" << std::endl;
```

N.B. : Pour bien mesurer la différence, donnez une taille suffisamment grande. Attention à ne pas dépasser la mémoire disponible sur votre machine. Pour éviter tout problème, vous pouvez calculer la mémoire utilisée (en MB) pour un vecteur de taille `N` contenant des éléments de type `T` tel que : `(N * sizeof(T)) >> 20`.