Travaux dirigés C++ n°1

Informatique

—IMAC 2e année—

Introduction à la syntaxe du C++

Ce TD a pour but de rappeler et d'introduire la syntaxe générale d'un programme C++ ainsi que sa compilation. TP à faire en monôme ou binôme.
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.** ***Compilation et entrée/sorties***

Objectif de l'exercice : pouvoir compiler un programme C++ sans dépendances en ligne de commande. Revoir les entrées sorties clavier.

1. *Créez un fichier* `TP1_ex1.cpp` *contenant la fonction* `main` *classique. Cette fonction* `main` *doit afficher "Les IMACs sont des brutes de C++" et ne retourner aucune erreur.*

2. *Compilez avec la commande :* `g++ -Wall -O2 TP1_ex1.cpp -o TP1_ex1`*, ou bien avec votre IDE en vérifiant que ces options sont bien activées.*

3. *Ajoutez au programme une variable de type* `int` *et modifiez vorte programme pour qu'il affiche :*

   - *le nombre en question*
   - *"Parfait" sur la sortie standard si l'entier est égal à 42 ;*
   - *"Strictement positif" sur la sortie standard si l'entier est strictement positif et différent de 42.*
   - *"Négatif" sur la sortie d'erreur si l'entier est strictement négatif.*

   Est-ce que le nom de la variable que vous avez choisi est un nom explicite (quand on le lit, on imagine ce que peut contenir cette variable)?

▸ **Exercice 2.** ***Chaine de caractères***

Objectif de l'exercice : manipuler des chaînes de caractères (`std::string`) ; utiliser la documentation (apprendre en codant).

1. *D'après vous, quel nom doit porter le fichier que vous devez créer ?*
2. *Inclure* `#include<string>`.
3. *Dans le* `main`*, déclarez une chaîne de caractères et affichez la avec l'opérateur* `std::cout`.
4. *En n'utilisant qu'un seul* `std::cout`*, affichez le nombre de caractères de votre chaîne et le dernier caractère tel que la sortie soit :*
   ```
   String size  :  X
   Last element :  Y
   ```
   Si vous utilisez l'operateur `std::string::back()`*, pensez à ajouter* `-std=c++11` *à votre commande de compilation.*
5. *Effacez le dernier caractère et vérifiez le résultat.*
6. *Ajoutez "IMAC" au début de la chaîne.*
7. *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`). Vous pourrez utiliser des fonctions comme* `std::tolower`.
8. *Inversez la chaîne de caractères (abcd → dcba). Vous pouvez regarder la fonction* `std::swap`.

▸ **Exercice 3.** ***Les vectors***

Objectif de l'exercice : découvrir les vecteurs de la STL (`std::vector`).

1. *Créez un nouveau fichier, y inclure* `#include<vector>`*. Dans la fonction* `main`*, créez un vecteur d'entiers vide.*
2. *Ajoutez une variable de type* `size_t` *représantant la taille du vecteur souhaité, puis ajoutez autant d'éléments à votre vecteur en utilisant la fonction* `std::vector::push_back` *dans une boucle. Le i-ème élément doit avoir pour valeur* `2*i`.
3. *Affichez la taille du vecteur.*
4. *Affichez tout les éléments du vecteur avec l'opérateur* `[ ]`.
5. *Affichez l'adresse mémoire des trois premiers éléments ainsi que la taille d'un élément (avec la fontion* `sizeof`*). Concluez sur la manière dont le tableau est stocké en mémoire.*
6. *Supprimez le dernier élément.*
7. *Effacez le contenu du vecteur.*

▸ **Exercice 4.** ***Les références***

Objectif de l'exercice : comprendre l'intérêt des références.

1. *Créez un nouveau fichier dans lequel vous commencez, comme dans l'exercice précédent, par créer une variable* `my_vector_size` *de type* `size_t` *ayant pour valeur 20. Créez ensuite un vecteur* `my_vector` *de taille* `my_vector_size` *et remplissez le tel que* `my_vector[i]` *ait pour valeur* `i % 10`*. Affichez les éléments de votre vecteur.*

2. *Ajouter une fonction* `void add_ten(std::vector<int> vec)` *qui ajoute 10 à chaque élément du vecteur. Appelez cette fonction depuis de le main puis affichez le contenu du vecteur. A-t-il changé?*

3. *Modifiez votre fonction de telle sorte qu'elle prenne en parametre une référence sur un vector :* `void add_ten(std::vector<int> &vec)`*. Appelez cette fonction depuis de le main puis affichez le contenu du vecteur. A-t-il changé?*

4. *Commentez l'affichage du contenu du vecteur (car on va travailler sur des gros vecteurs). Codez une fonction* `mean_reference` *permettant de calculer la moyenne des éléments du vecteur. Vous passerez le vecteur par référence. Est-ce que la référence doit être constante?*

5. *Codez une fonction* `mean_copy` *permettant de calculer la moyenne des éléments du vecteur. Vous passerez le vecteur par copie. Est-ce que la copie doit être déclarée constante?*

6. *Dans deux boucles séparées, appelez 10 000 de fois chacune des fonctions avec des vecteurs de taille 10 000 :*

```
  const unsigned int nbIter = 1000000;

  std::cout << "with␣reference␣..." << std::endl;
  for(unsigned int i=0; i< nbIter; ++i)
    mean_reference(my_vector);

  std::cout << "with␣copy␣...␣" << std::endl;
  for(unsigned int i=0; i< nbIter; ++i)
    mean_copy(my_vector);
  std::cout << "done" << std::endl;

  return 0;
}
```

   Est-ce que l'une de ces boucles vous paraît plus lente? (compilez avec `-O2`*, si l'operation est trop rapide, mettez des vecteurs plus volumineux).*

7. *que se passe-t-il si on enlève les* `const` *dans les fonctions* `mean_reference` *et* `copy_reference`*?*