Université de Nice-Sophia Antipolis
Polytech ELEC4

Jeudi 20 septembre 2018
Durée: 2h

string, vector et valarray
Travaux Dirigés – Séance n. 2

---

1 Objectif

Le langage C++ a été conçu comme un langage à objets. Dans ce TD, vous allez manipuler des objets prédéfinis dans l'environnement de programmation C++, et nous commencerons par les chaînes de caractères et les vecteurs.

2 Notion d'objet

En programmation objet, un objet contient à la fois des *données* et des *méthodes* (fonctions et procédures). Les actions s'appliquent, en général, sur les données de l'objet. Dans les langages à objets de *classe*, comme par exemple C++ ou Java, un objet est décrit de façon *statique* par une *classe*. Nous verrons plus tard, comment déclarer une classe, mais pour l'instant nous allons voir comment appliquer une méthode sur un objet (prédéfini du langage), ou accéder à une de ses données.

Comme dans de nombreux langages, pour appliquer la méthode `m` d'un objet désigné par une variable `o`, on utilise la notation pointée `o.m()`. Si l'objet possède une donnée accessible par une variable `v`, on écrira `o.v` pour accéder à cette donnée.

On n'en dit pas plus pour l'instant car cela est suffisant pour faire les exercices qui suivent.

3 Le type String

.

Nous l'avons vu dans le TD précédent, le type `string` représente les chaînes de caractères. Ce sont des objets, qui possèdent des données, en particulier les caractères de la chaîne, et des méthodes, comme par exemple `length()` qui donne la longueur de la chaîne. On rappelle que leur utilisation nécessite la directive d'inclusion **#include** `<string>`.

exercice 1) Écrivez un programme qui déclare une variable `s` de type `string`. Lisez sur l'entrée standard une chaîne de caractères que vous affecterez à une variable `s`, puis vous écrirez sur la sortie standard sa longueur. Par exemple :

```
Donnez une chaîne de caractères : azerty123
azerty123 : longueur = 9
```

exercice 2) Écrivez un programme qui déclare 3 variables `s1`, `s2` et `s3`, de type `string`. Affectez à `s1` la chaîne `"Bonjour"` et à `s2` la chaîne `" à tous"`. Affectez à `s3` la concaténation de `s1` `s2`. L'opérateur de concaténation est `+`.

exercice 3) Faites la même chose que précédemment, mais en utilisant la fonction `strcat`. Que

constatez-vous ? Comment l'expliquez-vous ?

exercice 4) Ajoutez à votre programme la variable `s4` et affectez `s3` à `s4`. Affichez `s3` et `s4`.

exercice 5) On va modifier le 1er caractère de `s4`. On utilise la notation de tableau, et l'indice du 1er caractère est égal à 0. Modifiez ce caractère, et affichez à nouveau `s3` et `s4` que pouvez-vous en conclure sur l'opération d'affectation ?

exercice 6) Une description complète du type `string` peut être trouvée à la page `http://en.cppreference.com/w/cpp/string/basic_string`. Écrivez un programme qui met en évidence l'utilisation de la méthode `find`.

4 Vector

C++ inclut bien évidemment les tableaux de C. On rappelle qu'en C les tableaux sont statiques, c'est-à-dire que leur taille (leur nombre d'éléments) est déterminée à la compilation.

Le type `vector` propose des tableaux dynamiques, c'est-à-dire de taille variable. Après avoir écrit la directive d'inclusion **#include** `<vector>`, on déclare une variable `v` de type vecteur comme suit :

```
vector <type_élém> v;
```

Ci-dessous, on déclare des vecteurs d'entiers, de chaînes de caractères, et un vecteur de réels.

```
#include <vector>
#include <string>

using namespace std;

vector <int> v1;
vector <string> v2;
vector <vector<double>> v3;
```

Les 3 vecteurs précédents ne contiennent pas d'éléments (leur taille est égale à 0). On peut aussi spécifier un nombre d'éléments à sa construction :

```
vector <int> v4 = vector<int>(10); // vecteur de 10 entiers
```

ou en donnant des valeurs initiales :

```
vector <string> v5 = { "ab", "ef", "azy" }; // vecteur de 3 chaînes
```

exercice 7) Dans un programme, reprenez les déclarations précédentes, et afficher la taille de chacun de ces vecteurs à l'aide de la fonction `size()`.

---

exercice 8) Écrivez `v5` sur `cout`. Que constatez-vous ?

exercice 9) Écrivez la procédure *afficherVecteur* qui affiche les éléments du vecteur passé en paramètre. Vous utiliserez la notation `[]` ;

exercice 10) Récrivez la procédure précédente avec l'énoncé *foreach* vu la semaine dernière.

---

exercice 11) Écrivez la fonction `main` qui déclare deux vecteurs `v1` et `v2` de `string`. Vous initialiserez `v1` avec les 3 valeurs `"a"`, `"b"` et `"c"`. Affectez `v1` à `v2`. Affichez le contenu de `v1` et `v2`. Modifiez

`v2[0]` et affichez le contenu de `v1`. Que pouvez-vous en déduire ?

Les vecteurs sont des structures dynamiques et les méthodes `push_back` et `pop_back` permettent d'ajouter, respectivement supprimer, un élément en fin de tableau.

exercice 12) Écrivez la procédure `vecteurPair`, qui initialise un vecteur v avec les n premières valeurs paires positives (v et n sont les 2 paramètres de la procédure).

La notation `v[i]` permet d'accéder ou modifier un élément d'un vecteur à l'indice i qui doit être, *nécessairement* compris entre 0 et `v.size()-1`.

exercice 13) Dans la fonction `main`, déclarez un vecteur `v1` et testez vos procédures pour afficher les éléments de `v1`. Par exemple, l'appel de la fonction `vecteurPair(v1, 6)` produira `0 2 4 6 8 10`.

Les fonctions `insert` et `erase`, permettent d'insérer, respectivement supprimer, un (ou plusieurs) éléments à partir d'une *position*. La position n'est pas un indice, mais un *itérateur* (nous reviendrons sur cette notion ultérieurement). Les itérateurs `begin()` et `end()` donnent respectivement la position du premier élément, et celle qui suit le dernier. Ainsi, `v.insert(v.begin()+i, x)` insère dans le vecteur `v` l'élément `x` *avant* l'élément d'indice i.

exercice 14) Écrivez la procédure `vecteurPairImpair` qui insère dans un vecteur `v` qui contient une suite de nombres pairs, les nombres impairs pour former un suite croissante d'entiers naturels. Testez votre procédure. Avec le vecteur précédent, la suite formée sera donc `0 1 2 3 4 5 6 7 8 9 10 11`.

5 Valarray

Le type `valarray` permet de représenter des tableaux dont les éléments sont des valeurs numériques, et de faire des opérations mathématiques de façon efficace. Il permet aussi de gérer des sous-ensembles d'éléments du tableaux. Pour utiliser ces tableaux, la directive **#include** `<valarray>` sera nécessaire.

L'exemple suivant déclare deux tableaux `valarray`, le premier de 5 entiers, intiallisés à 0, et le second de 3 réels intialisé aux valeurs 1.2, 2.2 et 3.0.

```
valarray <int> v1(5);
valarray <double> v2 = { 1.2, 2.2, 3.0 };
```

La notation `t[i]` permet d'accéder à l'élément d'indice `i`.

La taille d'un `valarray` ne peut croître automatiquement avec des fonctions d'insertion, comme pour `vector`. La modification de la taille du tableau doit être faite explicitement à l'aide de la fonction `resize`.

exercice 15) Un vecteur (au sens mathématique) est représenté par un `valarray` de **double**. Écrivez la fonction `produitScalaire` qui renvoie le produit scalaire de deux vecteurs transmis en paramètre. On vérifiera que les deux vecteurs possèdent bien la même dimension. La signature de cette méthode est la suivante :

```
/*
 * Rôle : renvoie le produit scalaire des vecteurs v1 et v2.
 */
double produitScalaire(valarray<double> v1, valarray<double> v2)
```


Rappel :

$$v'.v'' = \sum_{i=1}^{n} x'_i * x''_i$$

exercice 16) Écrivez la fonction `main` pour tester votre fonction.

---

6 Vecteur de vecteurs

On veut représenter un matrice $m \times n$ comme un vecteur (`vector`) de m vecteurs (`vector`) de dimension n.

exercice 17) Écrivez la fonction `initMatrice` qui renvoie une matrice $m \times n$ initialisée aléatoirement. Cette fonction possède deux paramètres entiers m et n qui sont les 2 dimensions de la matrice.

exercice 18) Écrivez la procédure `afficherMatrice` qui écrit sur la sortie standard la matrice passée en paramètre.

exercice 19) Écrivez la fonction `main` pour tester les deux fonctions précédentes.

exercice 20) Recommencez en remplaçant les `vector` par des `valarray`.