Université de Nice-Sophia Antipolis
Polytech ELEC4

Lundi 9 septembre 2019
Durée: 2h

C++ introduction
Travaux Dirigés – Séance n. 1

---

1 Objectif

Le but de ce premier TD est la prise en main du langage C++. Il s'agit, dans un contexte procédural (pour l'instant) de mettre en évidence les premières différences entre les langages C et C++.

2 Introduction

Le langage C++ est un langage à objets inventé par le danois Bjarne Stroustrup dont la première version est sortie en 1983. C'est un langage normalisé, et sa norme actuelle est celle de 2017. Depuis sa conception, le langage a évolué et propose aujourd'hui plusieurs paradigmes de programmation (procédural, objet, fonctionnel). Pour une bref historique du langage, vous pouvez consulter la page wikipédia `https://fr.wikipedia.org/wiki/C%2B%2B#Histoire`.

Ce langage est une extension du langage C (d'où le nom, `++` est l'opérateur d'incrémentation en C). C'est un sur-ensemble (toutefois pas à 100%) de C. Un compilateur C++ compile un programme écrit en C. On rappelle que le langage C est un langage procédural. C++ ajoute à C, en particulier, la notion de *classe* pour définir les *objets*, mais nous verrons cela lors d'une prochaine leçon.

3 Premier programme C++

Comme d'habitude, voici le programme traditionnel d'accueil écrit en C++.

```
/*
 *  Ce programme écrit « bonjour à tous » sur la sortie standard
 */
#include <cstdlib>
#include <iostream>

int main() {
  std::cout << "Bonjour à tous" << std::endl;
  return EXIT_SUCCESS;
}
```

Ce programme se compile à l'aide du compilateur `g++` qui produit l'exécutable `a.out`. Comme avec `gcc`, vous pouvez utiliser l'option `-o` pour changer le nom de l'exécutable produit.

La 1ère directive d'inclusion est utilisée pour obtenir l'accès à `EXIT_SUCCESS`. Le fichier `cstdlib` est l'équivalent de `stdlib.h` dans l'environnement C.

La 2ème directive d'inclusion est utilisée pour obtenir l'accès aux éléments d'E/S, ici `cout` (la sortie standard) et `endl` (la fin de ligne). Le fichier `iostream` est l'équivalent de `stdio.h` dans l'environnement C.

Notez l'absence de `.h` dans les noms de fichiers à inclure.

`cout` et `endl` sont définis dans un *espace de noms*, `std`. Pour accéder à ces 2 noms, il faut alors les qualifier par le nom de l'espace suivi de `::`, *e.g.* `std::cout`.

L'opérateur `<<` permet l'écriture de son opérande droit sur `std::stdout`, *i.e.* la sortie standard.

Enfin, l'exécution du programme débute par celle de fonction `main` dont la déclaration est semblable à celle du langage C.

exercice 1) Écrivez le programme dans le fichier `bonjour.cpp`, puis compilez-le et exécutez-le.

La directive **using namespace** *nom_espace* permet d'utiliser les noms définis dans l'espace sans avoir à les préfixer par le nom de l'espace.

exercice 2) Récrivez le programme précédent en utilisant la directive **using namespace**. Testez-le.

L'opérateur de lecture est `>>` et l'entrée standard est `std::cin`. Par exemple, si `x` est une variable entière, l'instruction `cin >> x` lira un entier sur l'entrée standard et l'affectera à la variable x.

exercice 3) Écrivez un programme qui demande votre année de naissance, et qui affiche sur la sortie standard votre âge. Vous définirez une constante `ANNEE_COURANTE` égale à 2019.

exercice 4) Écrivez un programme qui lit deux réels qui représentent, respectivement, la largeur et la longueur d'un rectangle, et qui écrit sa surface sur la sortie standard. L'exécution de votre programme ressemblera à :

```
largeur : 3.2
longueur : 5.1
la surface du rectangle 3.2 × 5.1 est 16.32
```

Remarquez que les instructions d'entrée/sortie ne font pas intervenir le type des valeurs à lire ou à écrire.

4 Fonctions

En C++, la déclaration d'une fonction est similaire à celle faite en C. Nous ne reviendrons pas sur cette notion vue l'an passé.

Toutefois, C++ n'impose pas d'ordre entre les déclarations et les instructions, et il introduit un second mode de transmission des paramètres et permet la surcharge et les paramètres par défaut.

Alors qu'en C, seul le mode de transmission *par valeur* existe, C++ ajoute la transmission *par référence* qui permet d'avoir des paramètres résultats sans avoir à simuler ce mode de transmission à l'aide des pointeurs.

Dans l'en-tête de la fonction, on indique un paramètre formel par référence en plaçant devant son nom le symbole `&`.

exercice 5) Écrivez la procédure `echanger` qui échange les valeurs de ses 2 paramètres de type **int**. Comparez la avec celle écrite en C l'an passé. Testez votre procédure C++.

exercice 6) Écrivez la fonction `div` à 4 paramètres : deux paramètres *données* a et b, et deux paramètres *résultats*, q et r. Cette fonction calcule le quotient et le reste de la division euclidienne de a par b. Vous procéderez par soustractions successives selon l'algorithme vu l'an passé.

Vous testerez votre fonction `div` dans une fonction `main`. Vous lirez les valeurs de a et b sur l'entrée standard. L'exécution de votre programme ressemblera à :

```
a ? 45
```

```
b ? 7
q = 6
r = 3
45 = 6 × 7 + 3
```

Une autre différence importante avec C est la possibilité de *surcharger* les fonctions. Le concept de *surcharge* est la possibilité de déclarer plusieurs fonctions de *même nom* mais différenciées par le type et le nombre de leur paramètres.

exercice 7) L'an passé, nous avions écrit deux fonctions `max2` et `max3` qui renvoyaient respectivement le maximum de 2 et 3 entiers (**int**). Récrivez en C++ ces 2 fonctions, mais avec le nom unique `max`. Testez votre fonction `max` avec 2 et 3 paramètres.

Les deux fonctions précédentes sont distinguées par leur nombre de paramètres. Le compilateur C++ distingue aussi les fonctions selon le type des paramètres (mais *pas* le type de retour de la fonction).

exercice 8) Ajoutez une fonction `max` à deux paramètres, avec des types différents de **int**. Compilez et testez votre programme.

Une autre différence est la possibilité de donner une valeur *par défaut* à un paramètre formel lorsque le paramètre effectif correspondant est absent. Dans l'exemple suivant, le paramètre `y` prendra par défaut la valeur 12.1 et `z` la valeur `'z'`.

```
void p(int x, double y=12.1, char z='z') {
  cout << "x=" << x << " y=" << y << " ; z = " << z << endl;
}
```

exercice 9) Testez la fonction précédente avec les appels suivants :

```
p(9, 1.0,'a');
p(3, 'a');
p(5, 2.3);
p(19);
```

Que constatez-vous ?

C++ définit aussi la notion de fonction *inline*. Sa déclaration se fait en plaçant le mot-clé **inline** devant l'en-tête.

Contrairement à l'appel classique d'une fonction, lors de l'appel d'une fonction *inline*, le corps de la fonction se substitue à l'appel. L'intérêt premier est un code produit plus efficace.

Notez qu'une fonction *inline* est similaire à une macro C avec paramètres, toutefois la gestion des paramètres est correctement assurée.

exercice 10) Déclarez la macro `CARRE` et une fonction *inline* `carre` qui renvoie le carré (donc le produit par lui-même) de son paramètre et testez le code suivant :

```
int a = 3;
cout << CARRE(a++) << endl;

a = 3;
cout << carre(a++) << endl;
```


5 Les types de base

Les types de bases sont ceux du langage C. Toutefois, C++ introduit le type booléen et de nouveaux types caractères.

Le type prédéfini **bool** est un ensemble à deux valeurs **true** et **false**.

exercice 11) Écrivez un programme qui déclare un variable booléenne `p` initialisée à **true** et qui écrit sa valeur sur la sortie standard. Que constatez-vous ?

exercice 12) Modifiez votre programme en ajoutant la déclaration d'un seconde variable booléenne `q` initialisée à 0 que vous afficherez. Que constatez-vous et qu'en déduisez-vous ?

exercice 13) Écrivez l'instruction `cout << boolalpha;` et affichez à nouveau les valeurs des variables `p` et `q`.

exercice 14) Écrivez l'instruction `cout << noboolalpha;` et affichez à nouveau les valeurs des variables `p` et `q`.

En plus du type **char**, C++ définit 3 nouveaux types caractères pour représenter des caractères du jeu unicode : `char16_t` dont la représentation est au moins sur 16 bits, `char32_t` dont la représentation est au moins sur 32 bits, et **wchar_t** (caractères étendus) qui permet de représenter le jeu de caractères le plus grand mise en œuvre sur la machine.

exercice 15) Écrivez un programme qui écrit sur la sortie standard la taille (en octets) de chaque de ces 4 types sur votre machine.

Les constantes caractères littérales unicode codées en hexa doivent être précédées par le caractères `u` (pour UTF-8) et `U` (pour UTF-16) , et le caractère par un `L`. Par exemple, dans la déclaration :

```
wchar_t c = L'\u0190';
```

le caractère affecté à `c` est le caractère unicode ϵ (epsilon) d'ordinal 0190 (en hexa).

Pour écrire des caractères unicode sur la sortie standard, on utilise `std::wcout` à la place de `std::cout`. De même pour la lecture de caractères unicode, il existe `std::wcin`.

exercice 16) Écrivez un programme qui affiche le caractère unicode Â. Que constatez-vous ? Expliquez.

6 Chaînes de caractères

On peut toujours représenter les chaînes de caractères par des tableaux de caractères avec `'\0'` comme délimiteur de fin de chaîne, et on pourra utiliser les fonctions classiques de manipulation de chaînes de caractères (`strlen`, `strcpy`) après avoir inclus le fichier `cstring`.

exercice 17) Écrivez un programme qui déclare un tableau de caractères et l'initialise avec une chaîne de caractères. Ensuite, vous afficherez cette chaîne avec sa longueur.

Toutefois, C++ définit le type `string` pour représenter les chaînes de caractères de type **char**. Il définit aussi les types `std::u16string`, `std::u32string` et `wstring` pour représenter des chaînes de caractères de type `char16_t`, `char32_t` et **wchar_t**. Dans ce dernier cas, les constantes littérales chaînes de caractères doivent être préfixées par un `L`.

exercice 18) Avec le type `string`, écrivez un programme qui lit sur l'entrée standard votre prénom et qui affiche sur la sortie standard *« Bonjour »* suivi de votre prénom. L'exécution de votre

programme donnera par exemple :

```
Quel est votre prénom ? Maud
Bonjour Maud !
```

Les chaînes de caractères sont des objets, au sens de la *programmation objet*, et nous verrons lors de la prochaine leçon comment utiliser ces objets.

exercice 19) Une chaîne de caractères de type `string` peut contenir des caractères unicode. Déclarez et afficher sur la sortie standard une chaîne de caractères qui contient le caractère unicode Â.

Enfin, C++ définit également le type `wstring` pour les chaînes qui contiennent des caractères étendus (type **wchar_t**).

7 For

L'énoncé **for** de C++ est semblable à celui de C. Comme dans la version 99 de C, il permet de déclarer la variable de boucle *locale* à la boucle, et il faudra utiliser systématiquement cette possibilité. Par exemple :

```
for (int i=0; i<MAX; i++) {
   ....
}
```

Depuis sa norme 2011, C++ propose aussi une version généralisée de l'énoncé *pour*, similaire à ce que l'on trouve dans le langage Java. On l'appelle aussi « énoncé *foreach* » . Sa forme est la suivante :

```
for (T var : ens_val) {
   ....
}
```

ens_val définit un ensemble de valeurs de type *T*. Le nombre d'itérations est égal au cardinal de cet ensemble. À chaque itération, la variable *var* prend une nouvelle valeur de cet ensemble. Cet ensemble peut être, par exemple, une chaîne de caractères ou un tableau.

exercice 20) Écrivez une programme qui déclare un tableau `t` de **double**, que vous initialiserez avec les valeurs de votre choix, puis utilisez l'énoncé **for** généralisé pour afficher sur la sortie standard chacune des valeurs du tableau.

exercice 21) Écrivez la procédure `printStr` qui prend une chaîne de caractères (type `string`) en paramètre, et qui écrit sur sur la sortie standard chacun de ses caractères suivi d'un point (`'.'`).

exercice 22) Écrivez une fonction `main` pour tester la procédure précédente.

C++ peut aussi faire de l'*inférence de type*, c'est-à-dire, que dans le cas de l'énoncé **for** généralisé, il peut déterminer le type des éléments de l'ensemble sans avoir à le spécifier dans la déclaration de la variable de boucle. Dans ce cas, on remplace ce type, par le mot clé **auto**.

exercice 23) Modifiez votre procédure `printStr` pour utiliser cette possibilité.