Programmation Orientée Objet

Introduction au C++

charles.meunier@u-bourgogne.fr

VISUAL STUDIO COMMUNITY

charles.meunier@u-bourgogne.fr

ENVIRONNEMENT DE DÉVELOPPEMENT INTÉGRÉ (EDI OU IDE EN ANGLAIS)

OUTIL DE DÉVELOPPEMENT CLÉ EN MAIN

- Editeur de texte
- Coloration syntaxique et auto-complétion
- Gestionnaire de projets
- Compilation
- Débogage
- Prend en charge de nombreux langages (C++, C#, Python, PHP, TypeScript, …)

GUIDE D'INSTALLATION

DISPONIBLE SUR LA MARMOTTE :

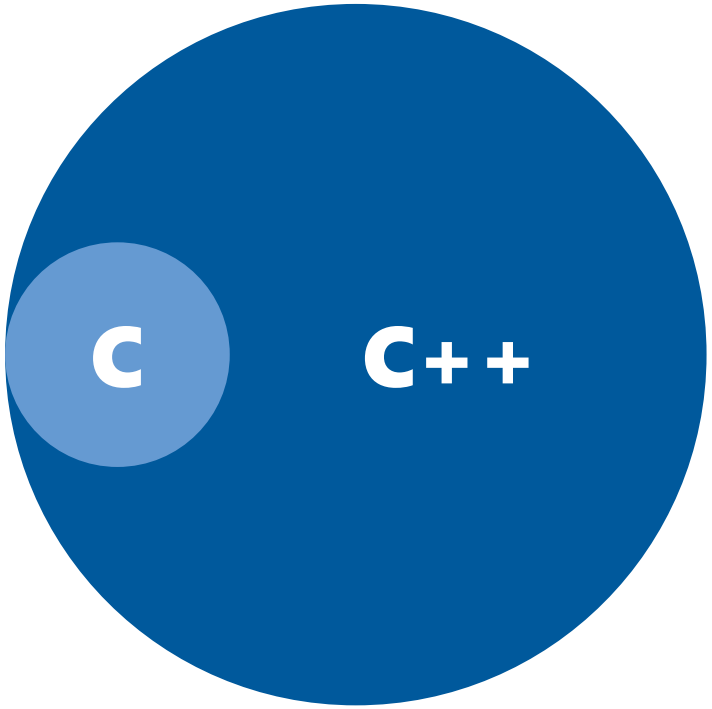
https://www.lamarmotte.info/index.php/2023/02/08/poo-les-guides/

charles.meunier@u-bourgogne.fr

C++

charles.meunier@u-bourgogne.fr

SUR ENSEMBLE DE C



Un code C pourra être compilé avec le compilateur C++,

MAIS PAS L'INVERSE

PAREIL MAIS EN DIFFÉRENT

Malgré de fortes similitudes syntaxiques, le C++ est un langage très différent :

	C	C++
Langage bas niveau	✓	✓
Langage haut niveau		✓
Programmation Procédurale	✓	✓
Programmation Orientée Objet		✓
Programmation Générique		✓
Programmation Fonctionnelle		✓ (partiellement)

PERFORMANCES ET UTILISATIONS

C++ est un langage **COMPILÉ** à **PERFORMANCES ÉLEVÉES**.

Sa **POLYVALENCE** lui vaut d'être utilisé dans le développement de nombreux types d'applications :

- Systèmes embarqués
- Logiciels de rendus 3D
- Jeux
- Logiciels systèmes (pilotes matériels, utilitaires, ...)
- Applications scientifiques traitant un grand volume de données
- ...

NOUVEAUX TYPES

`bool`

Type booléen pouvant prendre les valeurs **`true`** ou **`false`**

```
bool isReady = true;
```

`const`

Permet de définir des constantes

```
const int integer = 19;
integer = 27;                // ERROR!
```

`auto`

Apporte l'inférence de type : le compilateur se charge de déterminer le type de la variable en fonction de la valeur qui va y être stockée.

```
auto character = 'A';
```

NAMESPACE

library-1.h

```
int sum(int a, int b);
```

library-2.h

```
int sum(int a, int b);
```

main.cpp

```
#include "lib-1.h"
#include "lib-2.h"

int main()
{
    return sum(19, 44);
}
```

PROBLÈME : que se passe-t-il si le nom d'une des fonctions est déjà utilisée dans une autre bibliothèque du programme ?

- **LE COMPILATEUR NE SAIT PAS QUELLE FONCTION CHOISIR**

NAMESPACE

library-1.h

```
namespace lib1
{
    int sum(int a, int b);
}
```

library-2.h

```
namespace lib2
{
    int sum(int a, int b);
}
```

main.cpp

```
#include "lib-1.h"
#include "lib-2.h"

int main()
{
    return lib2::sum(19, 44);
}
```

Les **ESPACES DE NOM** (namespace) **ENCAPSULE** les fonctions dans un **BLOC DE CODE NOMMÉ**.

L'opérateur `::` permet ensuite d'indiquer au compilateur à quel bloc de code on souhaite faire référence

BIBLIOTHÈQUE STANDARD

Le C++ apporte peu de types natifs (bool, const, auto) par rapport au C.

Les tableaux et les chaines de caractères **N'EXISTENT PAS** nativement en C++

La bibliothèque standard apporte un ensemble conséquent de fonctionnalités au langage :

- Tableaux et listes (`std::array`, `std::vector`, `std::deque`, ...)
- Chaine de caractères (`std::string`),
- Pointeurs intelligents (`std::uniq_ptr`, `std::shared_ptr`, ...) ,
- Algorithmes (`std::reverse`, `std::sort`, ...),
- ...

Tous les éléments de la bibliothèque standard appartiennent à l'espace de nom **`std`**

ENTRÉE / SORTIES STANDARD

La bibliothèque standard **<iostream>** fournit des outils pour gérer les flux d'entrée / sorties standard (par défaut la console).

std::cin

Lire des données depuis l'entrée standard

```
#include <iostream>

int main()
{
    int integer;
    std::cin >> integer;

    return 0;
}
```

std::cout

Ecrire des données sur la sortie standard

```
#include <iostream>

int main()
{
    int integer = 754;
    std::cout << integer;

    return 0;
}
```

std::cerr

Ecrire des données sur la sortie standard réservée aux erreurs

```
#include <iostream>

int main()
{
    int integer = 754;
    std::cerr << integer;

    return 0;
}
```

CHAINE DE CARACTÈRES

La bibliothèque standard **<string>** fournit un type complet pour la gestion des chaines de caractères.

```
#include <iostream>
#include <string>

int main()
{
    std::string hello = "Hello";
    std::string world = "World"
    std::string helloWorld = hello + " " + world;

    std::cout << helloWorld << " : " << helloWorld.length();

    return 0;
}
```

MATHÉMATIQUES

La bibliothèque standard **<cmath>** fournit tout un ensemble de fonctions mathématiques.

```
#include <iostream>
#include <cmath>

int main()
{
    int integer;
    std::cin >> integer;

    std::cout << std::pow(integer, 2) << ", " << std::sqrt(integer);

    return 0;
}
```

Liste des fonctions disponibles dans cmath : https://cplusplus.com/reference/cmath/

RETOUR SUR LES VARIABLES

charles.meunier@u-bourgogne.fr

PETIT TEST :)

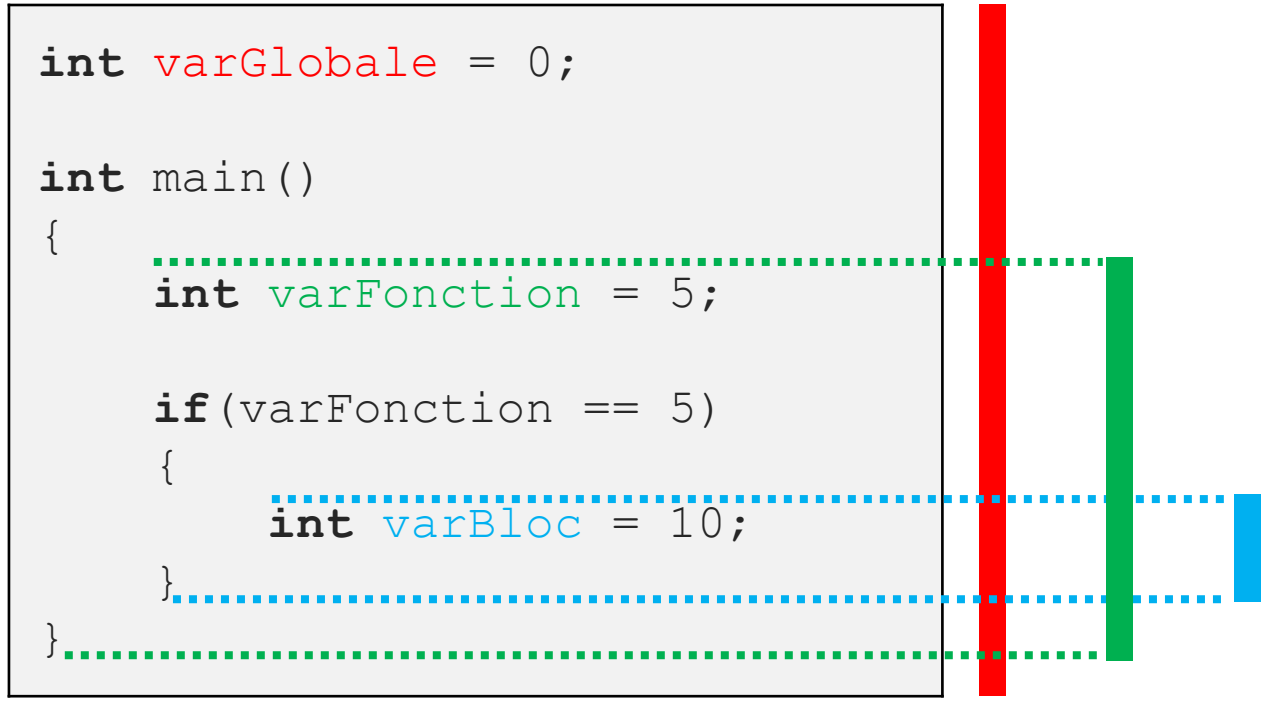
`int unEntier;`	OK	Erreur
`unEntier = 34;`	✓	
`unEntier = 8.69;`	✓	
`unEntier = 0xFF;`	✓	
`unEntier = 0b11001010;`	✓	
`unEntier = 'A';`	✓	
`unEntier = "Bonjour !";`		✗

PORTÉE DES VARIABLES

```
int varGlobale = 0;

int main()
{
    int varFonction = 5;

    if(varFonction == 5)
    {
        int varBloc = 10;
    }
}
```



PORTÉE DES VARIABLES

VALEUR, POINTEURS, RÉRÉRENCES

charles.meunier@u-bourgogne.fr

VALEURS

```
int integer1 = 5;
int integer2 = 5;
bool boolean = false;
std::string str = "Yolo !!";
```

Ces quatre éléments sont des **VALEURS**

- Ils occupent chacun leur propre espace en mémoire
- `integer1 = integer2` : recopie la valeur de `integer2` dans `integer1`
- `integer1 == integer2` : compare les valeurs de `integer1` et `integer2`

VALEURS

AVANTAGES

- **ALLOCATION STATIQUE**
- Création automatique
- Destruction automatique

INCONVÉNIENTS

- Passage de paramètres par recopie
- Paramètres en sortie impossibles
- **DURÉE DE VIE ET PORTÉE IDENTIQUE**

POINTEURS

```
int integer1 = 5;
int integer2 = 5;
int* ptr1 = &integer1;
int* ptr2 = &integer2;
```

Ces quatre éléments sont encore des **VALEURS**

Mais `ptr1` et `ptr2` font référence à `integer1` et `integer2` car ils contiennent leur adresse.

- `ptr1` contient l'adresse de `integer1`
- `ptr1 = ptr2` : recopie la valeur de `ptr2` dans `ptr1`, c-à-d l'adresse de `integer2`
- `ptr1 == ptr2` : compare les valeurs de `ptr1` et `ptr2`, c-à-d les adresses de `integer1` et `integer2`

POINTEURS

AVANTAGES

- Font référence à d'autres valeurs via leur adresse
- Utilisés pour l'allocation dynamique
- Optimisent les passages en paramètres de structures volumineuses

INCONVÉNIENTS

- Opérateur `->` pour accéder aux attributs d'une structure pointée
- Opérateur `*` pour obtenir la valeur pointée
- Pas compliqué, mais pas simple non plus

RÉFÉRENCES

```
int integer1 = 5;
int integer2 = 5;

int& ref1 = integer1;
int& ref2 = integer2;
```

Un type suivi d'un **&** indique que l'on déclare une référence.

Une référence est un alias vers une valeur.

- `ref1` est une référence vers `integer1`
- Manipuler `ref1` c'est la même chose que manipuler `integer1` directement
- `ref1 = ref2` : recopie la valeur de `ref2` dans `ref1`, c-à-d `integer2` dans `integer1`
- `ref1 == ref2` : compare les valeurs de `integer1` et `integer2`

RÉFÉRENCES

- Un pointeur qui se manipule comme une valeur (opérateur . pour accéder aux attributs d'une structure référencée)
- **DOIT ÊTRE INITIALISÉE À SA CRÉATION**
- Ne peut pas référencer une valeur qui n'existe pas
- **NE PEUT PAS CHANGER DE VALEUR RÉFÉRENCÉE**
- Optimise les passages en paramètres de structures volumineuses

RÉFÉRENCES NON CONSTANTES

```cpp
#include <string>
#include <iostream>

void addWorld(std::string& str)
{
    str += " World";
}

int main()
{
    std::string hello = "Hello";

    addWorld(hello);
    addWorld("World !");

    return 0;
}
```

La fonction `display` prend en paramètre une **RÉFÉRENCE NON CONSTANTE** sur une `string`. Cela veut dire que le code de `display` a le droit de modifier la valeur référencé par `str`.

ATTENTION : `"World !"` est de type `const char[8]` et **NE PEUT PAS ÊTRE RÉFÉRENCÉ PAR UNE RÉFÉRENCE NON CONSTANTE**

RÉFÉRENCES CONSTANTES

```cpp
#include <string>
#include <iostream>

void display(const std::string& str)
{
    std::cout << str << std::endl;
}

int main()
{
    std::string hello = "Hello";

    display(hello);
    display("World !");

    return 0;
}
```

La fonction `display` prend en paramètre une **RÉFÉRENCE CONSTANTE** sur une `string`.

Cela veut dire que le code de `display` ne peut pas modifier la valeur référencé par `str`.

`"World !"` est de type `const char[8]` et sera implicitement converti en `const string` avant d'être référencé par `str`.

RÉFÉRENCES ZOMBIES

```cpp
#include <string>
#include <iostream>

std::string& newHello()
{
    std::string hello = "Hello";

    return hello;
}

int main()
{
    std::string& ref = newHello();

    std::cout << ref;

    return 0;
}
```

La fonction `newHello` renvoie une référence sur sa variable statique `hello`.

Or, une fois la fonction `newHello` terminée, sa variable `hello` est détruite.

La référence `ref` de la fonction `main` ne pointe donc sur rien ce qui produira une erreur à l'exécution.

POUR TERMINER…

charles.meunier@u-bourgogne.fr

PETIT EXERCICE

DONNÉES EN ENTRÉE
tableau : un tableau d'entiers aléatoires
taille : un entier représentant le nombre d'entiers dans le tableau

VARIABLES
i, **j**, **nombre** : trois entiers

DÉBUT

```
POUR CHAQUE i allant de 1 à taille
FAIRE
    nombre <- tableau[i]
    j = i - 1
    TANT QUE j >= 0 ET nombre < tableau[j]
    FAIRE
        tableau[j + 1] = tableau[j]
        j = j - 1
    FIN TANT QUE
    tableau[j + 1] = nombre
FIN POUR
```

RÉALISEZ UN JEU D'ESSAI AVEC LES DONNÉES D'ENTRÉE SUIVANTES ET DÉTERMINEZ CE QUE FAIT CET ALGORITHME

- tableau : [5, 2, 9, 1]
- taille : 4