

# Une brève introduction à C++

Fabian Bastin

Hiver 2011

La majeure partie des outils repris dans le répertoire COIN-OR sont écrits en C++. C++ en deux mots.

- langage dérivé du C ; le code C peut le plus souvent être directement utilisé avec un compilateur C++ (quelques précautions doivent toutefois être prises) ;
- langage orienté-objet (comme Java).

Un *Hello World* en C++.

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello \ World! \ " << endl;

    return 0;
}
```

# Hello World !

- Syntaxe similaire à `Java`.
- Le programme commence par une série de déclarations, puis suit le code principal.
- Tout programme doit contenir une fonction `main`, qui est le point d'entrée du programme. A la différence de `Java`, cette fonction n'est pas à l'intérieur. `Java` ne travaille qu'avec des objets, `C++` non. On peut écrire un programme `C++` sans objets.
- `iostream` est une librairie servant aux entrées-sorties.
- Le mot-clé `#include` sert à indiquer que les fonctions utilisées sont décrites dans le fichier `iostream.h` (`.h` pour *header*).
- `using namespace std;` sert à indiquer que les entrées-sorties se feront sur le canal standard (i.e. le clavier et l'écran).

# Types de variables prédéfinis

`void` : type vide ;

`char` : caractère ;

`int` : entier ;

`bool` : booléen ;

`float` : réel simple précision ;

`double` : réel double précision ;

`wchar_t` : caractère long.

Les types entiers (`int`) peuvent être caractérisés d'un des mots clés `long` ou `short`. Ces mots clés permettent de modifier la taille du type, c'est-à-dire la plage de valeurs qu'ils peuvent couvrir. De même, les réels en double précision peuvent être qualifiés du mot clé `long`, ce qui augmente leur plage de valeurs. Note, sur les architectures habituelles, `double` et `long double` ont même place de valeur.

```
#include <iostream>
using namespace std;

class CRectangle {
    int x, y;
    public:
        void set_values (int, int);
        int area () {return (x*y);}
};

void CRectangle::set_values (int a, int b) {
    x = a;
    y = b;
}
```

```
int main () {  
    CRectangle rect;  
    rect.set_values (3,4);  
    cout << "area:_" << rect.area ();  
    return 0;  
}
```

- Syntaxe. Chaque instruction est terminée par le symbole ; et un bloc d'instructions est délimité par une accolade ouvrante ( { ) et une accolade ( } ) fermante.
- Une **classe** est un objet, qui contiendra des variables et des fonctions, appelées **méthodes**. Similaire à Java.
- On construit ici une classe rectangle, qui sera spécifié en indiquant sa largeur et sa longueur, et qui dispose d'une méthode permettant d'en calculer l'aire.

# On ne partage pas tout !

- Les éléments (variables et méthodes) peuvent être privés, publics ou protégés :
  - private** les membres privés d'une classe sont seulement accessible à partir des autres membres de la même classe ou de leurs amis ;
  - protected** les membres protégés sont accessibles pour les membre de la même classe et de leurs amis, mais aussi pour les membres des classes dérivées.
  - public** les membres publics sont accessible par n'importe qui pouvant voir l'objet.
- Dans l'exemple, les variables sont privées (comportement par défaut), et on utilise le "setter" `set_values` pour les initialiser.

# Classes : constructeurs et destructeurs

On voudrait pouvoir initialiser certains éléments de l'instance d'une classe lors de sa création.

C'est possible grâce à la fonction spéciale appelée **constructeur**, qui est automatiquement appelée quand une nouvelle instance est créée. Cette fonction doit avoir même nom que la classe, et ne peut avoir aucun type de retour, même `void`.

```
#include <iostream>
using namespace std;

class CRectangle {
    int width, height;
public:
    CRectangle (int, int);
    int area () {return (width*height);}
};
```

# Constructeurs : exemple

```
CRectangle::CRectangle (int a, int b) {  
    width = a;  
    height = b;  
}  
  
int main () {  
    CRectangle rect (3,4);  
    CRectangle rectb (5,6);  
    cout << "rect_area:_" << rect.area() << endl;  
    cout << "rectb_area:_" << rectb.area() << endl;  
    return 0;  
}
```

Dans l'exemple, on passe la largeur et la longueur comme arguments lors de la création d'une instance.

- Le destructeur remplit la fonctionnalité opposée : il est automatiquement appelé quand un objet est détruit.
- Le destructeur doit avoir le même nom que la classe, mais précédé d'un signe tilde (~), et il ne doit retourner aucune valeur.
- L'utilisation de destructeurs est particulièrement adaptée quand un objet affecte de la mémoire dynamique au cours de durée de vie, et qu'au moment de sa destruction, nous souhaitons libérer la mémoire qui avait été allouée.
- Allocation dynamique ? Utilisateur de **pointeurs**. Chaque variable est stockée en mémoire. On peut connaître l'**adresse mémoire** grâce à l'**opérateur de référence** : &.

- De la même manière qu'on peut connaître mémoire associée à une variable, on peut créer une variable dont le contenu est une adresse : **pointeur**.
- La déclaration d'un pointeur se fait en plaçant une astérisque entre le type et le nom de variable. Exemple :

```
int a;  
int *b = &a;
```

- Pour connaître le contenu de la mémoire dont l'adresse est donnée par le pointeur, on effectuera une **opération de déréférencement**, en plaçant le symbole astérisque devant le nom de variable :

```
a = *b;
```

# Allocation mémoire

Le mot-clé `new` permet de requérir de la mémoire dynamique. `new` sera suivi de l'identifiant de type, et éventuellement du nombre d'éléments entre crochets (`[]`) pour allouer un tableau. L'appel à `new` retourne un pointeur au début du bloc mémoire nouvellement alloué. Par exemple,

```
int *semaine ;  
semaine = new int [7];
```

Le premier élément pointé par `semaine` peut être accédé soit avec l'expression `semaine[0]` ou l'expression `*semaine`. Le second élément peut être accédé avec `semaine[1]` ou `*(semaine + 1)`, et ainsi de suite.

Contrairement à `Java`, il n'y a pas de "garbage collector" : la mémoire alloué restera alloué tant qu'aucune instruction indiquant qu'elle doit être libérée n'est rencontrée.

# Désallocation mémoire

En C++, il est à la charge du programmeur de libérer le mémoire allouée dynamiquement pour la rendre à nouveau disponible. Cela se fait avec l'instruction `delete`, dont le format est :

```
delete pointer ;  
delete [] pointer ;
```

La première expression devrait être utilisée pour supprimer la mémoire alloué à un seul élément, et la seconde pour la mémoire allouée à un tableau d'éléments.

Note : en C, on utilise les instructeurs de plus bas niveau `malloc` et `free`. Les instructions `new` et `delete` permettent cependant de profiter des constructeurs et destructeurs.

Revenons aux destructeurs...

# Destructeur : exemple

```
#include <iostream>
using namespace std;

class CRectangle {
    int *width, *height;
public:
    CRectangle (int, int);
    ~CRectangle ();
    int area () {return (*width * *height);}
};
```

```
CRectangle::CRectangle (int a, int b) {
    width = new int;
    height = new int;
    *width = a;
    *height = b;
}
```

## Destructeur : exemple (2)

```
CRectangle::~CRectangle () {  
    delete width;  
    delete height;  
}
```

```
int main () {  
    CRectangle rect (3,4), rectb (5,6);  
    cout << "rect_area:_" << rect.area() << endl;  
    cout << "rectb_area:_" << rectb.area() << endl;  
    return 0;  
}
```

Plutôt que de déclarer l'instance `CRectangle` de cette manière, on peut utiliser un pointeur :

```
CRectangle *rect = new CRectangle;  
[...]  
delete rect;
```

Différents environnements sont disponibles.

Solution basique, mais efficace : un éditeur de texte (par exemple `emacs` ou `vim`), compilateur gnu `g++`.

Environnement de développement. Un exemple open-source (téléchargeable gratuitement) : Eclipse (<http://www.eclipse.org>). Supporte à la base le Java, mais gère le C/C++ grâce au plugin CDT (<http://www.eclipse.org/cdt/>).

Un tutorial d'utilisation d'Eclipse CDT : <http://www.ibm.com/developerworks/opensource/library/os-eclipse-stlcdt/index.html>

# Pour aller plus loin...

Ce cours n'étant pas destiné à apprendre un langage de programmation, il n'en sera pas dit d'avantage sur le C++.

Différents tutoriaux existent sur internet, par exemple

<http://www.cplusplus.com>

Vous pouvez aussi consulter les notes et exemples du cours

IFT1166 : <http://www.iro.umontreal.ca/~dift1166/>

Un livre de référence (parmi d'autres) :

- Stanley B. Lippman, Josée LaJoie, Barbara E. Moo, *C++ Primer*, 4th Edition, Addison Wesley, 2005.

Leitmotiv : "*Read the Source, Luke*". On apprend énormément en lisant du code source. Pour Smi, profitez des exemples repris dans le code distribué !!!