

Crash course de C++

Pour programmeurs en n'importe quel langage raisonnable

beOI Training



OLYMPIADE BELGE D'INFORMATIQUE
BELGISCHE INFORMATICA-OLYMPIADE

Table des matières

Les bases

La librairie standard

Pointeurs et références

Hello world !

```
// Ceci est un commentaire.  
#include <bits/stdc++.h> // importe toute la STL  
using namespace std; // évite de taper std::  
  
// Déclaration de fonction : type nom() { [...] }  
int main() {  
    // Toutes les instructions terminent par ";"  
    cout << "Hello World!" << endl;  
    // endl = retour à la ligne  
}
```

- ▶ La fonction **int** main() est appelée au lancement.
- ▶ Le **int** veut dire que main() renvoie un entier, mais main() renvoie 0 automatiquement (raisons historiques).
- ▶ Le sens des chevrons << indique que les mots vont vers cout, la "sortie standard".

Compilation et lancement

Les programmes en C++ doivent d'abord être transformés dans un langage compréhensible par l'ordinateur.

Compilation (en ligne de commande)

```
g++ -std=c++11 -Wall -Wextra -Wshadow hello.cpp
```

- ▶ `g++` : nom du compilateur
- ▶ `-std=c++11` : version de C++ utilisée
- ▶ `-Wall -Wextra -Wshadow` : active plein de warnings très utiles pour écrire des programmes sans bugs
- ▶ `hello.cpp` : code source

Lancement : `./a.out`

- ▶ `./` : dossier dans lequel on se trouve
- ▶ `a.out` : nom du fichier exécutable produit par `g++`.

Variables et opérations

Une variable doit avoir un type spécifié

```
int i = 5; // entiers dans  $[-2^{31}, 2^{31}[$ 
double j = 5.4; // nombres à virgules
bool b = true; // booléen (vrai ou faux)
char ch = 'D'; // caractères seuls
string s = "abcd"; // chaîne de caractères

// On peut les initialiser plus tard
int k, l; // plusieurs variables du même type

k = i + 2;
l = 7 / 3; // division entière  $\Rightarrow l = 2$ 
s += ch; // ajout d'un caractère
j /= 3; // divise j par 3
i++; l--; // ajoute 1, enlève 1
```

Conditions

```
int age;
cout << "Quel est votre âge ? ";
cin >> age; // lecture d'input
if (age < 18)
    cout << "Vous êtes mineur." << endl;
else if (age <= 120)
    cout << "Vous êtes majeur." << endl;
else {
    int a=3, b=4, c=5;
    bool rectangle = (a*a + b*b == c*c);
    if (rectangle && !(a == 0 || b == 0))
        cout << "Hypoténuse = " << c << endl;
}
```

- ▶ Les accolades {} sont facultatives pour une seule ligne.
- ▶ Le sens des chevrons >> indique que l'entier vient de cin, "l'entrée standard".

Boucles

```
// Imprime les nombres de 1 à 5
for (int i=1; i<=5; i++)
    cout << i << endl;

string s; // initialement vide
while (s != "oui") { // pas égal
    cout << "Aimez-vous programmer ? ";
    cin >> s;
}
```

- ▶ Les boucles **for** (;;) ont trois parties :
 - ▶ Initialisation : initialise une ou plusieurs variables
 - ▶ Condition : la boucle s'arrête quand elle est fausse
 - ▶ Incrémentation : exécutée à la fin de chaque itération

Fonctions

```
// Type obligatoire pour résultat et paramètres
int square(int x) {
    return x*x;
}
void sayHello(string s) { // void = ne renvoie rien
    cout << "Bonjour " << s << endl;
}
int main() {
    int y = square(4); // y = 16
    sayHello("Victor");
}
```

- ▶ Pas imbriquables, et toujours placées *avant* leur appel. Sinon il faut les déclarer ainsi : **void** sayHello(string s); et les implémenter après.
- ▶ Quand une fonction n'est pas **void**, toutes les exécutions possibles doivent terminer avec un **return**.

Tableaux

Toutes les cases d'un tableau doivent avoir le même type.

```
int maxi(int tab[], int n) { // le [] est toujours
    int ma = 0;              // après le nom
    for (int i=0; i<n; i++)
        ma = max(ma, tab[i]);
    return ma; // renvoie le maximum de a
}
int main() {
    int a[5], b[4][3]; // 4 lignes et 3 colonnes
    for (int i=0; i<5; i++)
        cin >> a[i];
    cout << maxi(a, 5) << endl;
}
```

- ▶ Le premier élément est à l'indice [0].
- ▶ La taille ne peut pas être modifiée.
- ▶ Un tableau ne connaît pas sa taille ! Il faut la donner à part quand on l'envoie à une fonction.

Table des matières

Les bases

La librairie standard

Pointeurs et références

STL et conteneurs

La librairie standard (STL) contient un tas de structures et fonctions très utiles.

Une structure très utile est le `vector<>` :

```
vector<int> v(3,-1); // 3 éléments initialisés à -1
v.push_back(7); // ajoute 7 au vecteur
v[1] = 5; // accès comme un tableau
for (int i : v) // prend les éléments un par un
    cout << i << endl; // imprime -1, 5, -1, 7
```

- ▶ On met le type des éléments dans les chevrons : `<int>`.
- ▶ Les structures ont des constructeurs qui les initialisent (ici `(3,-1)`) et beaucoup de méthodes (ici `.push_back()`).
- ▶ Plus d'infos sur les conteneurs :
<http://en.cppreference.com/w/cpp/container>

Algorithmes STL

La STL contient aussi beaucoup d'algorithmes prêts à l'emploi :

```
vector<int> v{4,-1,3,2}; // initialisé avec liste  
sort(v.begin(), v.end()); // trie tout le vecteur  
swap(v[0], v[1]); // échange les contenus  
reverse(v.begin(), v.end()); // renverse le vecteur  
  
for (int i : v)  
    cout << v << endl; // imprime 4, 3, -1, 2
```

- ▶ Et bien d'autres : copies, recherche binaire, matching, sélection du i -ème plus petit élément, ...
- ▶ Plus d'infos sur les algorithmes :
<http://en.cppreference.com/w/cpp/algorithm>

Table des matières

Les bases

La librairie standard

Pointeurs et références

Disclaimer

Attention, à partir de maintenant ça devient hard.

Âmes sensibles s'abstenir !

Mémoire et adresses

La mémoire d'un PC est comme un grand tableau, rempli de cases pour stocker les variables.

Programme

```
...  
int a = 2, b = 3;  
char ch[] { 'h', 'e', 'y' };  
...
```

Mémoire RAM

...	...
mem[12]	a = 2
mem[8]	b = 3
mem[4]	'h' 'e' 'y'
...	...

- ▶ Chaque case peut contenir un byte (8 bits), une variable peut être étalée sur plusieurs cases (**int** 4, **double** 8)
- ▶ Chaque case a une *adresse* (4, 8, 12, ...).
- ▶ L'adresse permet d'accéder à la variable

Pointeurs

- ▶ Un pointeur est une variable qui contient l'adresse d'une autre variable.
- ▶ On peut lire ou modifier une variable grâce à son pointeur.

Programme

```
int a = 2;  
int *b = &a; // type = (int *)  
cout << b << endl; // 12  
cout << *b << endl; // 2  
*b = 5; // modifie a, pas b  
cout << a << endl; // 5
```

Mémoire RAM

...	...
mem[12]	a = 2
mem[8]	b = 12
...	...

- ▶ `&a` = "indice de a dans mem[]"
- ▶ `*b` = mem[b] (et donc on peut lire et écrire à cet indice)

Passage par pointeur

- ▶ Quand on envoie une variable à une fonction, elle est copiée (passée *par valeur*).
- ▶ *Si on la modifie, ça ne change pas la variable de base.*

Passage par valeur

```
void add3(int a) {  
    a += 3;  
} // a = copie de i  
int main() {  
    int i=2;  
    add3(i);  
    cout << i << endl;  
} // affiche 2
```

Passage par pointeur

```
void add3(int *p) {  
    *p += 3;  
} // "mem[p] += 3"  
int main() {  
    int i=2;  
    add3(&i);  
    cout << i << endl;  
} // affiche 5
```

À droite, l'adresse `p` est aussi passée par valeur, *mais on modifie la valeur de `i` directement dans la mémoire avec `*p`.*

Références

Les références permettent d'utiliser un pointeur comme une variable normale (sans écrire & et * tout le temps).

```
int a = 2;
int &b = a; // type = (int &)
cout << b << endl; // 2
b = 5; // modifie a à travers b
cout << a << ' ' << b << endl; // 5 5
```

Différences avec les pointeurs :

- ▶ Une référence est liée à une seule variable, pour toujours.
- ▶ b est comme un autre nom pour a, c'est un alias.
- ▶ On n'a pas accès à l'adresse de a.
- ▶ Le & des références n'a **rien à voir** avec le & des pointeurs.

Passage par référence

Quand on passe une variable *par référence*, elle n'est pas copiée, c'est la "même variable".

Passage par valeur

```
void add3(int a) {  
    a += 3;  
} // a = copie de i  
int main() {  
    int i=2;  
    add3(i);  
    cout << i << endl;  
} // affiche 2
```

Passage par référence

```
void add3(int &a) {  
    a += 3;  
} // a = alias de i  
int main() {  
    int i=2;  
    add3(i);  
    cout << i << endl;  
} // affiche 5
```

- ▶ La seule différence c'est le & devant le nom de la variable.
- ▶ À droite, vu que i et a sont deux noms pour une même variable, quand on modifie a, ça modifie i aussi.
- ▶ Ça évite de copier la variable \Rightarrow plus rapide.