

**Cours n° 1**

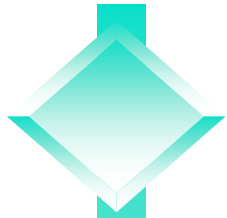
# **Présentation du langage C++**



## **Plan du cours**

---

- 1) Présentation du langage C++**
- 2) Les entrées-sorties**
- 3) Polymorphisme I (ad-hoc, inclusion)**
- 4) Polymorphisme II (patrons de classe et foncteurs)**
- 5) Structures de données abstraites (structures linéaires et graphes)**
- 6) Standard Template Library I (conteneurs séquentiels et itérateurs)**
- 7) Algorithmique du texte**
- 8) Standard Template Library II (relation d'ordre et conteneurs associatifs)**
- 9) Standard Template Library III (algorithmes et string)**
- 10) Introduction à la librairie Boost (Regex, Graph)**
- 11) Interopérabilité logicielle (entre C++, Perl, Python et Java)**
- 12) Heuristiques de conception**



## Sommaire

---

- 1. Déclarations de variables et variables simples**
- 2. Structures de contrôle**
- 3. Flots d'entrée sortie**
- 4. Fonctions et passage de paramètres**
- 5. Classes et fonctions membre**

## Historique et propriétés

---

**1980 Développement dans les laboratoires « AT&T Bell »**  
**1983 Premier compilateur C++ 1.0**  
**1989 C++ 2.0 (héritage multiple)**  
**1993 C++ 3.0 (template)**  
**1994 Bibliothèque de patrons génériques (STL) 1.0**  
**1998 Normalisation de C++ 4.0 (ISO/IEC 14882-1998)**  
**2003 Normalisation de STL (ISO/IEC 14882-2003)**  
**2005 Technical Report on C++ Library Extensions (Boost)**  
**2011 C++11 ISO/IEC 14882-2011 Langage et bibliothèque**  
**2014 C++11 ISO/IEC 14882-2014 (E) Révision mineure**  
**Langage compilé, orienté objet et à typage fort**

- Bibliothèque de composants (réutilisation)
- Vérification des types à la compilation (diminution des erreurs syntaxique)
- Généricité des algorithmes (optimalité)
- Grande importance dans l'industrie du logiciel

## Comparaison avec les langages Java et Perl

---

### **Efficacité (rapidité de traitement)**

- Forte en langage C++ (proche de celle obtenue en langage assembleur),
- Rapport 1 à 10 avec le langage Java,
- Rapport 1 à 100 avec le langage Perl

### **Abstraction (Masquage des couches physiques d'exécution)**

- Possible en langage C++ (déconseillé en règle générale),
- Impossible en langage Java et Perl

### **Expressivité (rapidité de transcription de connaissances métiers)**

- Forte en langage C++ (algorithmique et structures de données)
- Forte en langage Java (pour les interfaces graphiques),
- Forte en langage Perl (pour les expressions régulières)

## Le langage

Bjarne Stroustrup, «Le langage C++», Campus Press

Stanley Lippman, Josée Lajoie, «L'essentiel du C++», Vuibert Informatique

## La programmation

Claude Delannoy, «Programmer en langage C++», Eyrolles

Herb Sutter, «Mieux programmer en C++», Eyrolles

Jean-Bernard Boichat, «Apprendre Java et C++ en parallèle», Eyrolles

## Sites

[www.cppreference.com/](http://www.cppreference.com/)

[www.coursera.org/course/initprogcpp](http://www.coursera.org/course/initprogcpp)

## Différences entre le langage C++ et le langage C

---

### **C++ est un sur-ensemble de C**

- Compatibilité ascendante (quelques exceptions depuis C++ 3.0)

### **Avantages**

- Maintenance d'applications en C
- Interfaces avec des bibliothèques en C

### **Inconvénients (mixage code C/code C++)**

- Diminution drastique de la qualité de programmation
  - Structures de données confuses (utilisation des pointeurs)
  - Effets de bord imprévisibles (faible protection des données)
  - ...

### **Solution**

- Restriction aux spécificités du langage C++ par rapport au langage C
- Encapsulation des parties de code en langage C

## Types prédéfinis

---

**bool (true, false)**

**char (caractère sur 8 bits)**

**short (entier sur 16 bits, -32768..32767)**

**long (entier sur 32 bits, -21474836478..21474836477)**

**int (entier sur 16 ou 32 bits)**

**float (réel en simple précision)**

**double (réel en double précision)**



## 1. DECLARATIONS DE VARIABLES ET VARIABLES SIMPLES

### Définition et portée d'une variable simple

**type identificateur[=valeur][, identificateur[=valeur][...]];**

- Initialisation automatique à 0

int i=0, j=0;        // Déclaration et initialisation de deux entiers à 0

double somme;    // Déclaration d'une variable réelle

### Structure d'un programme

- Suite de blocs : instruction; ou { suite d'instructions }
- Un bloc peut contenir d'autres blocs

### Portée d'une variable

- Espace de visibilité de la variable
- Partie du bloc suivant la déclaration

{ int i; { int j } i = i+j } // interdit

{ int i; { int j; i = i+ j;} } // autorisé

### Pas de définition de variables globales

## Modificateurs de type

---

### Modificateurs de la représentation des entiers

- signed (par défaut)
- unsigned                      ex : unsigned char (0..255)

### Modificateurs de la durée de vie (présence en mémoire)

- auto (création à la définition, destruction à la fin du bloc)
- static (création au lancement du programme, destruction à la fin du programme)

### Modificateurs des règles de modification

- const (modifications interdites après l'initialisation)
- volatile (modifications autorisées par des instructions extérieures au programme)

## 1. DECLARATIONS DE VARIABLES ET VARIABLES SIMPLES

### Notion de pointeur et adresse-mémoire d'une variable

#### Variable contenant une adresse mémoire

Début d'une zone mémoire  
Adresse mémoire d'une variable  
Structures de données avancées  
(arborescence)

#### Opérateurs

& Adresse mémoire d'une variable  
\* Valeur de la variable pointée  
sizeof Taille mémoire d'une variable

#### Déclaration

```
byte age; float total; char c;  
byte* page = &age;  
float* ptotal = &total;  
char* pc = &c;  
char a = *pc;
```

	Adresses	Mémoire
		.....
age	@000A1267	.....
total	@000A1268	.....
	@000A1269	.....
	@000A126A	.....
	@000A126B	.....
	@000A126C	.....
c	@000A126D	.....
	@000A126E	.....
	@000A126F	.....
	@000A1270	.....
	@000A1271	.....

.....

### Tableaux statiques et allocation dynamique

**type identificateur[taille]([taille](...));**

float vect[100]           // déclaration d'un tableau de 100 réels

bool mat[100][50]       // déclaration d'une matrice 100X50 booléens

#### Réservation mémoire au moment de la compilation

- Pas de modification à l'exécution

**Identificateur = new type [nbelem];**

int i; float\* vect = new float[i] // déclaration d'un vecteur de réel de taille i

// vect est une variable du type référence de float

#### Réservation mémoire au moment de l'exécution

- Adaptation aux données
- Variables statiques

# Expressions logiques (Evaluation à true ou false)

---

## Opérateurs de comparaison

==	égalité
!=	inégalité
<	infériorité
>	supériorité
<=	infériorité ou égalité
>=	supériorité ou égalité

## Opérateurs logiques

&&	et logique
	ou logique
!	négation logique

Exemple : ( (a < b && a > 0) || (a > b && a == 0) )

### Types de structure de contrôle (1)

---

#### **if (ExprL) bloc1 [ else bloc2 ]**

Exécution conditionnelle du bloc suivant ou choix entre deux blocs

```
If (a < b && a > 0) {i = 0;} else {j = 0;}
```

#### **for (instruction1; Expr1; instruction2) blocB;**

Exécution de l'instruction 1 (initialisation)

Test (évaluation de Expr) : sortie du for si false

Exécution du blocB

Exécution de l'instruction2 (itération)

saut à Test

```
for (int i = 0; i < j; i = i+1) k = i;
```

### Types de structure de contrôle (2)

---

#### **for (val: liste) blocB;**

Boucle basée sur une liste ou un tableau

Equivalent du foreach en Java

Initialisation de val au premier élément de la liste ou du tableau

Test (évaluation si dernier élément) : sortie du for si false

Exécution du blocB

Passage de val à l'élément suivant

saut à Test

## 2. STRUCTURES DE CONTROLE

## Types de structure de contrôle (3)

```
#include <string>
#include <iostream>
#include <vector>
using namespace std;
int main (int argc, char* argv[]) {
    vector<string> v = {"hello", "cruel", "world"};
    for (string s: v)
        cout << s << endl;
}
```

```
g++ -std=c++0x -g -c -w boucle.cpp
```

```
g++ -g boucle.o -o boucle.exe
```

**Compilation****Edition de liens**



### Types de structure de contrôle (4)

---

#### **while (ExprL) blocB**

Exécution en boucle de blocB ex: while (a < 10) {i = i + a}

#### **do blocB while (ExprL);**

Exécution en boucle de blocB au moins une fois ex : do {i = i + a} while (a < 10)

#### **switch (valeur) {**

**case constante1: bloc1 break;**

**..**

**default: blocd break; }**

Branchement conditionnel

## Affichage Ecran – Lecture Clavier

---

### Flot de sortie prédéfini cout

- Affichage de variables Ex : `cout << var`
  - Affichage de suite de caractères Ex : `cout << "La valeur de"`
  - Affichage de caractère de contrôle Ex : `cout << endl` (passage à la ligne)
- `cout << "la valeur de var est égale à " << var << endl`

### Flot d'entrée prédéfini cin

- Lecture de variables Ex : `int i; float f; cin >> i >> f`

### Déclaration des flots standard

Ajout en début de programme

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

### Définition d'une fonction

---

**typeR identificateurF(paramF) blocF**

- TypeR est le type de la valeur renvoyée par la fonction (résultat)
- identificateurF est le nom de la fonction
- paramF définit les paramètres de la fonction
- BlocF correspondant aux déclarations, aux structures de contrôle et aux instructions

**type1 var1 [= val1] [, type2 var2 [= val2] [...]]**

Définition des paramètres d'une fonction,

Type1 est le type de la variable val1 initialisée par défaut à la valeur val1

**int main(int argc, char \*argv[])**

Fonction point d'entrée d'un programme

## 4. FONCTIONS ET PASSAGE DE PARAMETRES

### Définition d'une fonction (entête)

```
// déclaration des flots standard
#include <iostream>
using namespace std;

// prototype de la fonction prod
// deuxième paramètre par défaut
int prod(int n, int i = 1);
```

#### 4. FONCTIONS ET PASSAGE DE PARAMETRES

### Définition d'une fonction (code)

```
#include "factorielle.h"

// point d'entrée de l'exécutable
int main (int argc, char* argv[]) {
    int n; cin >> n; // lecture de la variable n
    cout << "prod(" << n << ")=" << prod(n) << endl;
    cout << prod(n, 0) << endl;
    return(0); }

int prod(int n, int i) {
    int res = 1;
    while (i <= n) {res = res * i; i++;}
    return(res); }
```

## 4. FONCTIONS ET PASSAGE DE PARAMETRES

### Surcharge de fonctions (code)

#### Possibilité d'avoir plusieurs fonctions de même nom

- Choix du compilateur en fonction des paramètres d'appel (nombre et type)

```
#include "minimum.h"

// minimum de 2 entiers
int min(int x, int y) { if (x < y) return x; else return y; }

// minimum de 3 entiers
int min(int x, int y, int z) {
    if (x < y) { if (x < z) return x; else return z; }
    else { if (y < z) return y; else return z; } }

int main (int argc, char* argv[]) {
    int i , j, k;
    cin >> i >> j >> k;
    cout << min(i,j) << min (i,j,k); return 0; }
```

## 4. FONCTIONS ET PASSAGE DE PARAMETRES

## Surcharge de fonctions (entête)

```
// minimum de 2 entiers  
int min(int x, int y);  
  
// minimum de 3 entiers  
int min(int x, int y, int z);
```

### Passage par valeur ou par référence

---

#### Passage par valeur

- Mode de passage par défaut
- Recopie de la variable (coût en mémoire et en temps calcul)
- Elimination des effets de bord

#### Passage par référence

- Ajout du symbole & avant le nom de la variable
- Utilisation de la même variable dans la programme appelant et la fonction appelée
- Elimination des effets de bord par l'utilisation du modificateur const



## 4. FONCTIONS ET PASSAGE DE PARAMETRES

**Passage par référence &**

```
#include "factorielle2.h"

int main (int argc, char* argv[]) {
    int n;
    cin >> n;
    cout << "la " << fact(n) << endl;
    return(0);}

int fact(const int& n) { // protection de la variable n
    int res = 1, i;
    for (i = 1; i <= n; i++)
        res = res * i;
    return(res); }
```

### Introduction

---

#### **Encapsulation des données**

- Réduction des possibilités d'accès aux variables
- Contrôle de l'accès aux variables
- Association de variables de même comportement
- Définition d'ensemble de variables

#### **Encapsulation des traitements**

- Association de chaque fonction à un des ensembles de variables

#### **Buts**

- Minimisation des erreurs de programmation
- Réutilisation dans d'autres programmes
- Aide à la conception

## 5. CLASSES ET FONCTIONS MEMBRES

### Définition d'une classe

```
class Ident {  
  private : // variables et fonctions privées (visibilité réduite aux fonctions de la classe)  
  type var; // attribut de la classe  
  typeR identificateurF(paramF); // prototypes  
  
  public : // fonctions publiques  
  typeR identificateurF(paramF); // prototypes  
};  
  
Ident::typeR identificateurF(paramF) blocF // définition des fonctions
```

### Fonctions membres usuelles

---

#### Fonctions accesseurs

- Lecture et écriture des attributs de la classe (contrôle d'accès)

#### Fonctions canoniques

**Ident()**

Ident (const Ident &)

~ Ident ()

Ident & operator = (const Ident &)

**Constructeur de classe**

Constructeur de copie

Destructeur de classe

Affectation de classe

# Gestion dynamique de la mémoire

---

## Avantages

- Adaptation dynamique aux données

## Inconvénients

- Proches des couches physiques d'exécution
- Pas de contrôle à l'exécution (erreur d'accès mémoire)
- Fragmentation de la mémoire (pas de garbage collector)

## Opérateurs new et delete

- Identificateur = **new** type [ [nbelem] ]

Création de l'objet (ou d'un tableau d'objets) de classe « type » et renvoi d'une référence sur cet objet

- delete Identificateur

Destructeur de l'objet (ou du tableau d'objets)

## 5. CLASSES ET FONCTIONS MEMBRES

## Définition de la classe Etudiant (entête)

```
class Etudiant {  
private :  
    int m_NombreNote; // nombre de notes  
    const static int m_NombreNoteDef = 10; // nombre de notes par défaut  
    int* m_Tnote; // Tableau dynamique des notes  
public :  
    Etudiant(); // constructeur vide (m_NombreNoteDef notes)  
    Etudiant(int n); // constructeur non vide  
    ~Etudiant(); // destructeur  
    void putnote (int e, int n); // ajouter une note  
    int getnote(int e) const; // lire une note avec protection  
};
```

## 5. CLASSES ET FONCTIONS MEMBRES

## Définition des méthodes (1/2)

```
#include "Etudiant.h"

// constructeur vide (m_NombreNoteDef notes)
Etudiant::Etudiant () {
    m_NombreNote = m_NombreNoteDef;
    m_Tnote = new int [m_NombreNote];
}

// constructeur non vide
Etudiant::Etudiant (int n) {
    m_NombreNote = n;
    m_Tnote = new int [m_NombreNote];
}

// destructeur
Etudiant::~Etudiant () { delete m_Tnote; }
```

## 5. CLASSES ET FONCTIONS MEMBRES

## Définition des méthodes (2/2)

```
// ajouter une note
void Etudiant::putnote (int e, int n) {
    if (e < m_NombreNote)
m_Tnote[e] = n;
}

// lire une note avec protection
int Etudiant::getnote (int e) const {
    if (e >= m_NombreNote) return -1;
    else return m_Tnote[e];
}
```



## 5. CLASSES ET FONCTIONS MEMBRES

## Utilisation de la classe Etudiant (source)

```
#include "testEtudiant.h"
#include "Etudiant.h"

int main() {
    // construction de deux objets de la classe etudiant
    etudiant Pascal, Pierre (15);

    // appel par l'objet Pascal à la méthode putnote
    Pascal.putnote(3, 15);

    // appel par l'objet Pascal à la méthode getnote
    cout << Pascal.getnote(3);
    return 0; }
```

**Eclipse avec le plugin cdt (C/C++ Development Tooling)**  
**Compilateur C++ gcc**  
**Création d'un projet C++ du type Makefile project vide**  
**Utilisation d'un fichier de projet makefile (voir en atelier)**

### **Installation sous Linux (fedora)**

`dnf install eclipse-cdt gcc`

### **Installation sous Windows**

ajouter le plugin cdt

installer le compilateur gcc mingw ([www.mingw.org/](http://www.mingw.org/))

## Normes de programmation

---

### Objectifs

- Elimination d'erreurs classiques de programmation
- Maintenance possible par plusieurs programmeurs
- Amélioration de la portabilité
- Facilité de lecture et de compréhension
- Cohérence du style de programmation

### Exemples

- Chaque fichier de code source doit contenir un entête décrivant son contenu.
  - Les sections public, protected et private d'une classe devraient apparaître dans cet ordre.
  - Une fonction membre d'une classe qui n'altère en rien l'état de l'objet doit être déclarée const.
  - Une fonction ne doit jamais retourner une référence ou un pointeur sur une variable locale.

### Site

[google.github.io/styleguide/cppguide.html](https://google.github.io/styleguide/cppguide.html)