

# Débuter en C++ moderne - Tome 1

Cours de C++ moderne

Création d'applications en C++ moderne par la pratique

## Avant-propos

- [Introduction](#)
- [Comment suivre ce cours](#)
- [Comment réaliser les exercices de ce cours](#)
- [Demander de l'aide et aider les autres](#)
- [normes et compilateurs](#)

## Premiers pas

### Langage

- [Programme C++ minimal](#)
- [Hello world](#)
- [Les littérales chaînes de caractères](#)
- [Introduction à la qualité logicielle](#)

### Compléments

- [\[Aller plus loin\] Un langage vivant](#)
- [\[Aller plus loin\] Explorer la documentation](#)

# Bases du calcul numérique

## Langage

- Les nombres entiers
- Logique binaire et calcul booléen
- Nombres réels
- string

## Compléments

- [Aller plus loin] L'algèbre de Boole
- complex
- [Aller plus loin] Les nombres à virgule fixe
- [Aller plus loin] Les nombres rationnels

## Conserver les valeurs en mémoire

## Langage

- Utiliser la mémoire avec les variables
- Constantes et énumérations
- La déduction de type
- Manipuler les types
- Obtenir des informations sur les types
- Créer des nouveaux types

## Compléments

- [\[Aller plus loin\] Les nombres pseudo-aléatoires](#)

## **Collections et algorithmes**

### **Langage**

- [Le concept de collections de données](#)
- [Les catégories de collections](#)
- [Les fonctionnalités de base des collections](#)
- [Introduction aux algorithmes standards](#)
- [Les catégories d'algorithmes standards](#)
- [Les itérateurs](#)
- [Les catégories d'itérateurs](#)
- [Les foncteurs et fonctions lambdas](#)

### **Compléments**

- [\[Aller plus loin\] Les tableaux de bits](#)

## **Chaînes avancées et expressions régulières**

### **Compléments**

- [\[Aller plus loin\] Les chaînes de caractères internationales](#)
- [\[Aller plus loin\] Les expressions régulières 1](#)
- [\[Aller plus loin\] Les expressions régulières 2](#)
- [\[Aller plus loin\] Les expressions régulières 3](#)
- [\[Aller plus loin\] Les expressions régulières 4](#)

## Créer des fonctions

Les chapitres suivants sont encore en cours de rédaction, voire à l'état d'ébauche. N'hésitez pas à faire des remarques ou poser des questions sur le forum de [Zeste de Savoir](#) ou de [OpenClassroom](#).

### Langage

- [Créer des fonctions](#)
- [Les paramètres de fonctions](#)
- [Copies, déplacements et indirections](#)
- [Les fonctions génériques](#)
- [fonctions lambdas](#)

### Compléments

- [La surcharge de fonctions et résolution des noms](#)
- type de fonction, `std::function`, `fonctor`, `callable`, `std::bind`, `std::invoke`
- [rvalue, lvalue et leur amis](#)
- [\[Aller plus loin\] Le memoire en detail](#)
- [\[Aller plus loin\] Le perfect forwarding](#)
- [constexpr](#)

## Créer des algorithmes

### Langage

- [Les instructions conditionnelles](#)
- [Les instructions itératives](#)
- [recursivité](#)

## **Compléments**

- [\[Aller plus loin\] Les paires, tuples et structures](#)
- [\[Aller plus loin\] Les classes et fonctions variadiques](#)
- [\[Aller plus loin\] Variables globales et statiques, mémoire static, string table](#)

## **Entrées et sorties**

### **Langage**

- [Introduction sur les entrées et sorties](#)
- [La ligne de commande](#)
- [Entrée console](#)
- [Les fichiers](#)

## **Compléments**

- [cout en détail](#)
- [variables d'environnement](#)
- [signals, abort, exit](#)
- [gestion des buffers avec les streams](#)

## **Concevoir une bibliothèque**

## Langage

- Séparer définition et implémentation
- documentation, commentaire, codes d'exemple
- Gérer les erreurs dans les fonctions
- Contrats et tests unitaires
- Mesurer les performances
- Concevoir une bibliothèque, interface publique, réutilisabilité.  
“Easy to use correctly, hard to use incorrectly” - Scott Meyers.  
Design interface : s'adapter aux conventions qui existent. Etre consistant
- modules, conception en couches, utilisation de n-1 et n+1
- namespace, dépendances. l'espace de noms global et anonyme  
::
- pré-condition et post-condition
- physical design
- variables globales, étude de std::cout
- abi et C
- macro et pré-processeur

<https://www.famkruihof.net/uuid/uuidgen>

## Les outils de développement

- Mettre en place une chaîne de compilation
- Les environnement de développement
- Les compilateurs
- Les débogueurs
- La validation statique du code
- La validation du style du code
- Créer une documentation
- Les gestionnaires de versions

- Les tests
- Déployer une application
- Le suivi des problèmes

## Les classes à sémantique de valeur

- `poo`

## Langage

- Créer des classes
- Créer et copier des objets
- Surcharger les opérateurs
- Conversion de types  
<http://cpptruths.blogspot.de/2015/11/covariance-and-contravariance-in-c.html>
- \* allocation et libération de ressources, `raii`

## Compléments

- [Aller plus loin] `Swap`
- [Aller plus loin] Les principes SOLID
- [Aller plus loin] Les design patterns
- [Aller plus loin] Les objets-fonctions
- `chaining methods`

## Pratiquer

- Classe à sémantique de collection
- Les ranges et vues sur les collections

- [Itérateur personnalisé](#)
- [Les design patterns Wrapper et Facade](#)
- traits : adapter les `type_traits` pour fonctionner avec vos classes

## Les classes à sémantique d'entité

### Langage

- [La sémantique d'entité](#)
- [L'héritage](#)
- [Polymorphisme et substitution](#)
- [Fonctions virtuelles](#)
- [invariant de classe](#)

### Compléments

- [La Pile et le Tas](#), création et destruction
- théorie des graphes, arbres
- héritage multiple
- pourquoi les manipuler via indirections ? aliasing, smart ptr
- points de variation : comment faire varier le comportement d'un programme ? (directive de compilation, template police/traits, héritage, DP stratégie) Quand utiliser quelle technique ? Impact sur la qualité du code (maintenabilité, évolutivité, etc)
- durée de vie et propriétaire des objets

### Pratiquer

- jeu d'échec

- évaluation de script (if, for, etc)
- article citation format (type medline)
- <http://progdupeu.pl/forums/sujet/205/banque-dexercices>
- factory. Créer une fonction factory avec switch, avec create, avec allocator
- système de gestion d'événements

## Bibliothèques externes

- [Utiliser une bibliothèque](#)
  - internationalisation : ICU
  - interface utilisateur : Qt, SFML
  - réseau : boost.asio, POCO, QtNetwork
  - XML : QtXml
  - base de données : QtSql
  - web : wt
  - script : boost.python, QtScript
  - <https://cpp.libhunt.com/>
  - <http://fffaraz.github.io/awesome-cpp/>
  - <http://en.cppreference.com/w/cpp/links/libs>
- 

## Le C++03

- [Le C++03](#)
- [Les chaînes de caractères style C](#)
- [tableaux style C](#)
- [Indirections dans les fonctions](#), surcharge opérateurs d'accès → \* &, etc + surcharge opérateur new et delete, pointeur de fonction

# Pratiquer (A trier)

- Analyse lexicale et syntaxique
- Évaluation d'expressions
- Gérer des intervalles de valeurs
- Zip et unzip
- Jouons avec les chiffres
- Créer un modèle de document
- XML
- Lire et écrire des feuilles de calculs
- Lire et enregistrer des images
- Sampler
- Jouer au Poker
- Algorithmes d'extraction de sous-chaînes
- Résoudre des intégrammes
- Jouer au Scrabble
- Analyseur simple d'expressions régulières
- imagerie : génération d'arbres avec L-System
- implémenter des générateurs aléatoires cryptographiques
- créer un système d'allocation mémoire (tas). Allocator, utilisation des espaces libres, défragmentation
- buffer avec vector
- <http://martinfowler.com/bliki/CQRS.html>
- [https://en.wikipedia.org/wiki/Fizz\\_buzz](https://en.wikipedia.org/wiki/Fizz_buzz)

Notes:

- conversion implicite et explicite
- indirections, adaptateurs, raw ptr/ref
- “Make interfaces easy to use correctly and hard to use incorrectly” - Scott Meyers

<https://sciencetonnante.wordpress.com/2015/10/16/la-machine-a-inventer-des-mots-video/>      <https://github.com/fffaraz/awesome-cpp>

<https://github.com/rigtorp/awesome-modern-cpp>  
<https://www.reddit.com/r/dailyprogrammer/>

## **À suivre...**

[À suivre...](#)