Informatique et systèmes de communication

Programmation 1

Chapitre 1
Introduction

Contributeurs

- Ont contribué à l'élaboration des transparents de ce cours :
  (dans l'ordre alphabétique)

- BREGUET Guy-Michel
- CUISENAIRE Olivier
- POPESCU-BELIS Andrei
- RENTSCH René

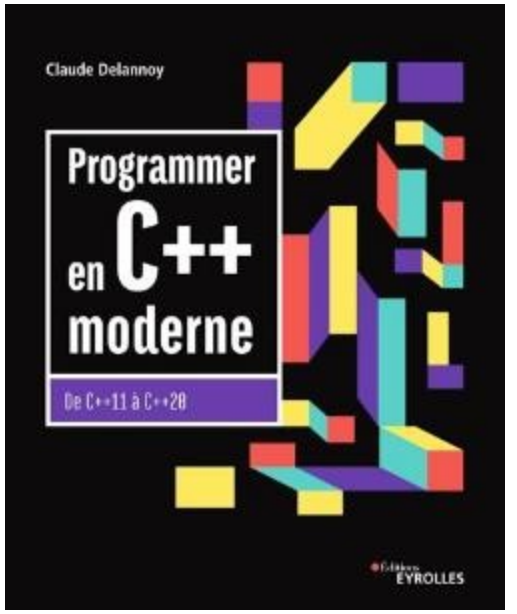
- Révision
  - TAILLARD Éric

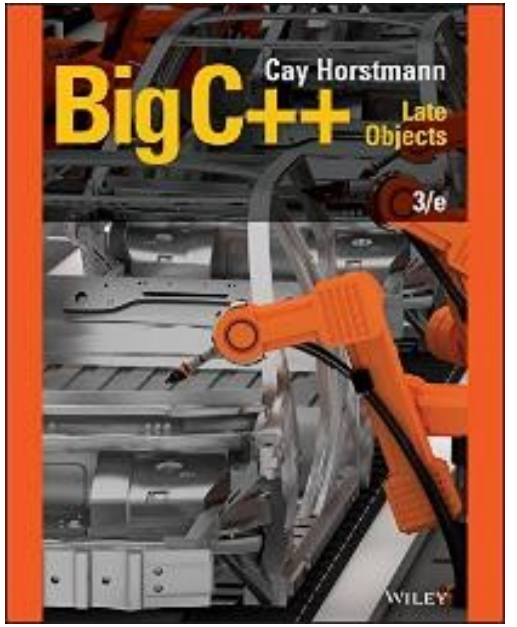
Sources

- Les transparents de ce cours sont principalement inspirés des supports de cours suivants :
  - Evan Gallagher, en support du livre "*C++ for Everyone"* de Cay Horstmann
  - INF1 / INF2, HEIG-VD, 2008-2014
  - INF1 / INF2, HEIG-VD, 2015-2020

Références

- *Programmer en C++ moderne*, Claude Delannoy
- *Big C++*, Cay S. Horstmann
- http://www.cplusplus.com
- http://fr.cppreference.com





Plan du chapitre 1

1. Qu’est-ce que la programmation ?
2. Langages de programmation de bas vs haut niveau
3. Compilation
4. Premier programme C++
5. Erreurs de compilation, d’édition de liens et d’exécution
6. Algorithmes et pseudo-code
7. Résumé

1. Qu’est-ce que la programmation ?

Qu'est-ce que la programmation ?

- Un **programme informatique** indique à un ordinateur, dans les moindres détails, la séquence d'étapes qui sont nécessaires pour lire des données, les traiter et restituer de l'information

- Terminologie
  - **Matériel** (***hardware***)
    - L'ordinateur physique et ses périphériques sont collectivement appelés le matériel
  - **Logiciel** (***software***)
    - Les programmes que l'ordinateur exécute sont appelés le logiciel
- La programmation…
  est l'art de **concevoir** et de **mettre en œuvre**
  des programmes informatiques

2. Langages de programmation de bas vs haut niveau

Code machine et processeur

- Les programmes sont stockés sous la forme d'**instructions machine** dans un code qui dépend du type de **processeur**
- Une séquence typique d'instructions machine serait :
  1. Déplacer le contenu de l'emplacement mémoire 40000 vers le **CPU**[1].
  2. Si cette valeur est supérieure à 100, continuer avec l'instruction stockée à l'emplacement mémoire 11280.
- Ces instructions sont codées sous forme de nombres pour être stockables en mémoire

[1] Le processeur ou CPU (*Central Processing Unit*) est le cœur de l'ordinateur

Langage de bas vs haut niveau, compilateur

- Les **langages de haut niveau** comme le **C++** sont indépendants du type de processeur et du type de matériel. Ils fonctionnent tout aussi bien :
  - sur un ordinateur avec un processeur Intel ou AMD
  - ou sur un téléphone portable
- Le **compilateur** est un programme spécial qui traduit la description de haut niveau (le code C++) en instructions machine pour un processeur particulier
- **Langage de bas niveau** : instructions machine pour un CPU particulier
  - Le code machine généré par le compilateur sera différent selon le CPU visé, mais le programmeur qui utilise un langage de haut niveau ne doit pas s'en inquiéter.

C++ - aujourd'hui

C++ dérive du langage **C** ; ces deux langages coexistent et **évoluent** toujours

- C++ permet la programmation sous de multiples **paradigmes** comme
  - la programmation **procédurale**
  - la programmation **orientée objet**
  - la programmation **générique**
- C++ est l'un des langages de programmation les plus **populaires**, avec une grande variété de plateformes matérielles et de systèmes d'exploitation
- D'autres langages populaires comme **Java** ou **C#** s'en sont largement inspirés

L'apprentissage de la programmation en TIC

- **Première année**
  - **Semestre 1**
    - PRG1 – C++
  - **Semestre 2**
    - ASD – C++
    - PRG2 – C
- **Deuxième année**
  - **Semestre 1**
    - POO1 – Java
  - **Semestre 2**
    - POO2 – C++

3. Compilateur

Compilateurs

- Il existe de nombreux **compilateurs C++**
- Nous utiliserons dans ce cours un **compilateur compatible avec la norme 2020**
- En PRG1, nous utiliserons le compilateur g++
  - Installé d'office dans linux et Mac OS
  - Installable en version 32 ou 64 bits sous Windows avec **Mingw-w64**
    - **https://sourceforge.net/projects/mingw-w64/files/latest/download**

Cycle de développement

- Écrire le programme avec un éditeur de texte
- Compiler le programme
- Exécuter le programme et vérifier son bon fonctionnement
- Corriger les éventuelles erreurs et recommencer

Environnement de développement intégré (IDE)

- Réunit en une seule fenêtre les outils de navigation dans les fichiers, d'édition du code, de compilation et d'exécution dans une console

IDE pour C++

- Il existe de nombreux IDE pour C++. On peut citer, entres autres :
  - CLion
  - Code::Blocks
  - CodeLite
  - Dev-C++
  - Eclipse
  - NetBeans
  - Visual Studio
  - Xcode

4. Premier programme C++

Hello World!

- Traditionnellement, l'étude d'un nouveau langage de programmation commence toujours par le même exemple appelé « **Hello, World!** »
- Son but : afficher le texte « Hello, World! » à l'écran
- En voici le code source en C++

```
#include <iostream>

using namespace std;

int main() {
   cout << "Hello, World!" << endl;
   return 0;
}
```

La syntaxe d'Hello World!

Le programme inclut un ou plusieurs fichiers d'en-têtes pour pouvoir utiliser des services nécessaires tels que les entrées/sorties

Cette instruction est discutée slide 22

```
#include <iostream>

using namespace std;

int main() {
   cout << "Hello, World!" << endl;
   return 0;
}
```

Chaque programme a une fonction principale `main`

Les instructions d'une fonction forment un bloc et sont placées entre accolades { … }

Chaque instruction se termine par un point-virgule

Hello World! – fichiers d'en-tête

- La première ligne indique au compilateur d'inclure un service pour les « flux d'entrées /sorties »
- Plus tard nous en apprendrons davantage à ce sujet mais, pour l'instant, il suffit de savoir que c'est nécessaire pour écrire à l'écran

```
#include <iostream>

using namespace std;

int main() {
   cout << "Hello, World!" << endl;
   return 0;
}
```

Hello World! – espace de noms

- La deuxième ligne indique au compilateur d'utiliser l'**espace de noms `std`** (`std` pour « standard »). Nous reviendrons plus tard sur cette notion d'espace de noms.
- Cette ligne n'est pas obligatoire… mais sans elle, il aurait fallu écrire `std::cout` et `std::endl` dans notre code.

```
#include <iostream>

using namespace std;

int main() {
   cout << "Hello, World!" << endl;
   return 0;
}
```

Hello World! – fonction principale

- Les lignes suivantes définissent une **fonction**
- Le nom de la fonction est `main`
  - tout programme doit avoir une fonction principale nommée `main`
- Elle **retourne** un entier (ce type est noté **int** en C++)
  - la valeur 0 indique que le programme se termine avec succès

```
#include <iostream>

using namespace std;

int main() {
   cout << "Hello, World!" << endl;
   return 0;
}
```

Hello World! – affichage sur la sortie standard

- Pour afficher la sortie à l'écran, nous utilisons une entité appelée **`cout`**
- Ce que vous voulez voir apparaître à l'écran est « envoyé » vers l'entité `cout` (*console out*) en utilisant l'opérateur **`<<`**
- Cette entité fait le nécessaire pour afficher à l'écran la chaîne de caractères "Hello, World!"

```
#include <iostream>

using namespace std;

int main() {
   cout << "Hello, World!" << endl;
   return 0;
}
```

Hello World! – affichage sur la sortie standard

```
cout << "Hello, World!" << endl;
```

- On peut afficher plus d'une chose en utilisant plusieurs fois l'**opérateur <<**
- **"Hello, World!"** est une chaîne de caractères, qu'on appelle **string** en C++
- **endl** est le symbole de retour à la ligne : il fait passer le curseur à la ligne suivante

5. Erreurs de compilation, d'édition de liens et d'exécution

Erreur classique – oublier un point-virgule

- Le point-virgule est un terminateur d'instruction

```
#include <iostream>

using namespace std;

int main() {
   cout << "Hello, World!" << endl
   return 0;
}
```

Oups !

Erreur de compilation (*syntax error*)

Sans le point-virgule, nous avons en fait écrit

```
cout << "Hello, World!" << endl return 0;
```

… ce qui constitue une **erreur de syntaxe**

- Cela entraîne une **erreur de compilation**
- Une **erreur de compilation** survient lorsque le compilateur ne peut plus, à partir de l'endroit où se trouve l'erreur, traduire votre code source en un code machine

Erreur de compilation

- Supposons que vous écriviez malencontreusement

```
cot << "Hello, World!" << endl;
```

Cela entraîne une erreur lexicale

- Le compilateur se plaindra en vous avertissant qu'il ne sait pas ce que `cot` signifie

Le message exact dépend du compilateur, mais il ressemblera à :
Undefined symbol cot ou encore: ***'cot' was not declared in this scope***

Combien d'erreurs ?

- Le compilateur ne s'arrête pas de compiler après une erreur mais continue tant qu'il peut (parfois en faisant des hypothèses sur l'intention du programmeur)
- Il est probable qu'il affiche de nombreuses erreurs, qui sont souvent des conséquences de la première erreur rencontrée
- Dès lors, on ne corrigera que les erreurs qui nous parlent (qui correspondent à des messages d'erreurs compréhensibles), et surtout la première erreur

Ⓟ Puis on relance la compilation, jusqu'à ne plus avoir d'erreur

Erreur d'exécution (*runtime error*)

- Considérons ce code

```
cout << "Hollo, World!" << endl;
```

- Une **erreur d'exécution** (ou **erreur logique**) est une erreur dans un programme qui compile (la syntaxe est correcte) mais qui n'exécute pas l'action attendue.

Pas vraiment une erreur, alors ?

Erreur d'exécution

```
cout << "Hollo, World!" << endl;
```

- **Si, c'est une erreur,** car le programmeur est responsable de **l'inspection** et du **test** de son programme pour éliminer les erreurs d'exécution

Erreurs – génération d'exceptions

- Certaines erreurs d'exécution sont si graves qu'elles génèrent une **exception** : un signal envoyé par le processeur qui, s'il n'est pas géré (voir chapitre 9), arrête le programme et génère un message d'erreur

- Par exemple, si votre programme contient l'énoncé :

```
cout << 1/0;
```

  votre programme peut se terminer par une exception ***divide by zero***

Erreur d'édition des liens (*link error*)

- Chaque programme C++ doit avoir une et une seule fonction principale nommée **`main`**
- La plupart des programmes C++ contiennent d'autres fonctions en plus de **`main`** (nous reviendrons sur les fonctions plus tard)
- Attention, **le C++ est sensible à la casse** (MAJUSCULE 📽 minuscule)

Ecrire

```
int Main() {
   return 0;
}
```

compile mais produit une **erreur de lien**

Erreur d'édition des liens (link error)

- Une erreur de lien se produit **ici** parce que l'éditeur de liens ne trouve pas la fonction principale **`main`** (ou plus généralement parce qu'une fonction appelée `main` n'est définie dans aucun des fichiers d'un projet)
  - En effet, vous n'avez pas défini de fonction nommée **`main`**

- Vous avez le droit de nommer une fonction **`Main`**
  - même si c'est une mauvaise idée
  - mais ce n'est pas la même fonction que **`main`**
  - et il doit y avoir une fonction **`main`** quelque part dans votre code pour que l'édition des liens fonctionne.

6. Algorithmes et pseudo-code

Le processus de développement

- *Comprendre le problème*
- *Développer et décrire un algorithme*
- *Tester l'algorithme avec des entrées simples*
- *Mettre en œuvre l'algorithme en C++*
- *Compiler et tester le programme*

Pour chaque problème,
le programmeur passe par ces étapes

Décrire un algorithme en pseudo-code

- Le **pseudo-code** est
  - une description informelle
  - pas un langage que l'ordinateur comprend
  - mais qu'on peut aisément traduire en un langage de haut niveau comme le C++
- La méthode décrite en pseudo-code doit
  - **être non ambigüe**, i.e. indiquer précisément
    - que faire à chaque étape
    - quelle est l'étape suivante
  - **être exécutable**, i.e. chaque étape peut être mise en œuvre
  - **se terminer**, i.e. son exécution amène à une étape finale

Décrire un algorithme en pseudo-code

- Considérons le problème suivant :

  - Vous hésitez entre acheter deux voitures.
  - L'une d'elles est plus chère à l'achat, mais consomme moins d'essence.
  - Vous connaissez le prix d'achat (en CHF) et la consommation (en litres aux 100 km) de chaque voiture.
  - Vous espérez utiliser votre voiture pendant 10 ans.
  - On suppose que l'essence coûte 1.45 CHF par litre et que l'on parcourt 10'000 km par an.
  - On achète la voiture cash et on ignore la complexité du financement.

- Quelle voiture est la moins chère globalement ?

Algorithmes

Etape 1 – déterminer les entrées / sorties

- Dans notre exemple, les **entrées** sont
  - `prixAchat1`, le prix (en CHF) de la première voiture
  - `consommation1`, la consommation (en litres aux 100 km) de la première voiture
  - `prixAchat2`, le prix (en CHF) de la seconde voiture
  - `consommation2`, la consommation (en litres aux 100 km) de la seconde voiture

- Quant à la **sortie**, nous désirons simplement savoir
  - quelle est la voiture la plus économique

Etape 2 – décomposer en sous-problèmes

- Pour chaque voiture (N vaudra 1 ou 2) :

  1. Le coût total de la voiture sera
     `prixAchatN (entrée) + coutUtilisationN`

  2. En supposant une utilisation constante et un prix de l'essence stable, le prix d'utilisation sera
     `nombreAnnees (10) x coutAnnuelEssenceN`

  3. Le coût annuel de l'essence sera
     `prixParLitre (1.45) x quantiteAnnuelleEssenceN`

  4. Enfin, la quantité annuelle d'essence consommée sera
     `nombreDeKmRoules (10000) x consommationN (entrée) / 100`

Etape 3 – décrire les sous-probl. en pseudo-code

- On doit organiser les étapes pour que chaque valeur intermédiaire nécessaire à une étape soit calculée avant d'être utilisée

```
pour chaque voiture N (N = 1 ou 2)
   calculer quantiteAnnuelleEssenceN = nombreDeKmRoules (10000) x consommationN / 100
   calculer coutAnnuelEssenceN = prixParLitre (1.45) x quantiteAnnuelleEssenceN
   calculer coutUtilisationN = nombreAnnees (10) x coutAnnuelEssenceN
   calculer coutTotalN = prixAchatN + coutUtilisationN

si coutTotal1 < coutTotal2
   choisir la voiture 1
sinon
   choisir la voiture 2
```

Etape 4 – tester le pseudo-code

- Testons le pseudo-code avec les valeurs suivantes :
  - Voiture 1 : 25'000 CHF, 5.6 litres aux 100 km
  - Voiture 2 : 22'000 CHF, 7.8 litres aux 100 km
- Pour la voiture 1 :
  - quantiteAnnuelleEssence1 = 10000 * 5.6 / 100 = 560
  - coutAnnuelEssence1 = 1.45 x 560 = 812
  - coutUtilisation1 = 10 x 812 = 8120
  - coutTotal1 = 25000 + 8120 = **33120**
- Pour la voiture 2, on trouve : coutTotal2 = **33310**
- Conclusion : **la voiture 1 est la plus économique**

7. Résumé

Résumé

- **Qu'est-ce que la programmation ?**
  - Les ordinateurs exécutent des instructions simples très rapidement
  - Un programme est une séquence d'instructions et de décisions
  - **Programmer est l'art de concevoir et mettre en œuvre des programmes**
- **Quel langage utiliser pour programmer ?**
  - Les programmes sont stockés sous forme d'instructions machine dont le codage dépend du processeur
  - **C++ est un langage de haut niveau** largement répandu et permettant de programmer selon divers paradigmes
    - Les langages de haut niveau sont indépendants du type de processeur. C'est le compilateur qui se charge de les traduire en code machine.

Résumé

- **Quels sont les éléments d'un programme C++ simple ?**
  - Tout programme C++ contient une fonction **`main`**, qui retourne une valeur entière.
  - On utilise **`cout`** et l'opérateur **`<<`** pour afficher des valeurs à l'écran.
    Envoyer **`endl`** à **`cout`** passe à ligne suivante.
  - Les instructions se terminent par un point-virgule.

Résumé

- **Quelles erreurs peut-on rencontrer ?**
  - Erreurs lexicales, de syntaxe et de sémantique à la compilation
  - Erreurs de lien à l'édition des liens
  - Erreurs d'exécution
    - Sortie fausse
    - Crash du programme qui lance une exception

Résumé

- **Comment aborder un problème de programmation?**
  - Décomposer en sous-problèmes plus simples tant que nécessaire
  - Décrire informellement la méthode choisie en utilisant du pseudo-code
  - Un algorithme est une séquence d'étapes non ambigües, exécutables et se terminant
  - Tester la méthode choisie avec des exemples
  - Traduire cette méthode en C++
  - Compiler, corriger, recompiler, recorriger, … jusqu'à résoudre toutes les erreurs de compilation ou d'édition de liens
  - Tester le programme avec des entrées bien choisies. Corriger les erreurs d'exécution si nécessaire.