MU4MA056 : Programmation en C++

Table des matières :

Organisation générale

- cours **à distance sur Zoom** le lundi matin 9h-10h30 (possibilité de le suivre sur votre ordi ou à l'Atrium à Jussieu si campus ouvert)
- cours **supplémentaire sur Zoom** jeudi 14 janvier, 14h15-16h.
- liens Zoom annoncé sur Moodle.
- documents et annonces sur Moodle.
- Travaux Pratiques : à partir du lundi 18 janvier, **à distance sur `https://repl.it`**. [cf. démonstration en ligne]
- Travaux Pratiques, 4 groupes de 20 : lundi, mardi, jeudi, vendredi, 13h45–16h45.

(Congé paternité de deux semaines à une date aléatoire autour de début mars : vidéos de complément pré-enregistrées sur Moodle)

Évaluation

Habituellement :

$$\textbf{Note sur 100} \stackrel{?}{=} \frac{\text{TP1 sur 25} + \text{TP2 sur 25}}{2} + \text{Examen sur 75}$$

mais cela dépendra des consignes de l'Université sur le contrôle continu intégral ou non. Si changement, passage vers

$$\textbf{Note sur 100} \stackrel{?}{=} \frac{\text{TP1 sur 50} + \text{TP2 sur 50}}{2} + \text{Examen sur 50}$$

Vous serez tenus au courant quand tout le monde en saura plus...

- TP noté 1 : sixième semaine des TP (début mars)
- TP noté 2 : douzième semaine des TP (mi-avril)

Conseils pour réussir cette UE

- *UE traditionnellement difficile* : le C++ est un langage très développé et très puissant et donc très exigeant.
- UE d'informatique au milieu d'UE de mathématiques : autre philosophie. Un nouveau langage avancé en 6 ECTS.

Conseils pour réussir cette UE

- *UE traditionnellement difficile* : le C++ est un langage très développé et très puissant et donc très exigeant.
- UE d'informatique au milieu d'UE de mathématiques : autre philosophie. Un nouveau langage avancé en 6 ECTS.
- Une **règle d'or** :

  Codez, codez, codez ! ! !
- *UE professionnalisante* : ce que demande une entreprise est que vous soyez à l'aise devant un clavier et du C++ (pas que vous soyez superficiellement au courant des derniers gadgets).

Conseils pour réussir cette UE

- *UE traditionnellement difficile* : le C++ est un langage très développé et très puissant et donc très exigeant.
- UE d'informatique au milieu d'UE de mathématiques : autre philosophie. Un nouveau langage avancé en 6 ECTS.
- Une **règle d'or** :

  Codez, codez, codez ! ! !
- *UE professionnalisante* : ce que demande une entreprise est que vous soyez à l'aise devant un clavier et du C++ (pas que vous soyez superficiellement au courant des derniers gadgets).
- *Informatique pour mathématiciens* : pas d'interface, pas de fantaise, mais de la performance numérique. Algorithmique + code optimisé.

Quelques généralités.

Langages informatiques

- Langages interprétés : Python, Ruby, matlab/scilab...
  Très pratique pour petit script, bac à sable, appels à des bibliothèques, syntaxe simplifiée,... mais **lent pour le calcul intensif**

Langages informatiques

- Langages interprétés : Python, Ruby, matlab/scilab...
  Très pratique pour petit script, bac à sable, appels à des bibliothèques, syntaxe simplifiée,... mais **lent pour le calcul intensif**
- Langages compilés : Fortran, C, C++... **Rapide pour le calcul intensif** mais *plus long à écrire, plus de détails à surveiller, plus de subtilités à maîtriser...*

Langages informatiques

- Langages interprétés : Python, Ruby, matlab/scilab...
  Très pratique pour petit script, bac à sable, appels à des bibliothèques, syntaxe simplifiée,... mais **lent pour le calcul intensif**
- Langages compilés : Fortran, C, C++... **Rapide pour le calcul intensif** mais *plus long à écrire, plus de détails à surveiller, plus de subtilités à maîtriser...*
- Interprétation : traduction en langage machine+exécution de chaque ligne, pas de vue globale sur programme. Compilation : traduction en langage machine globale sur l'ensemble du programme puis exécution globale donc possibilité de nombreuses optimisations.

Histoire de C et C++

- 1970 : langage C par Ken Thompson pour écrire Unix. *Avoir un langage adapté pour manipuler la mémoire : pointeurs*.

Histoire de C et C++

- 1970 : langage C par Ken Thompson pour écrire Unix. *Avoir un langage adapté pour manipuler la mémoire : pointeurs*.
- 1980 : développement du C++ par Bjarne Stroustrup. *Augmentation du C pour laprogrammation orientée objet.* Ajout des **classes**.

Histoire de C et C++

- 1970 : langage C par Ken Thompson pour écrire Unix. *Avoir un langage adapté pour manipuler la mémoire : pointeurs*.
- 1980 : développement du C++ par Bjarne Stroustrup. *Augmentation du C pour laprogrammation orientée objet.* Ajout des **classes**.
- **C et C++ :** 2 évolutions mais volonté de rétrocompatibilité.

Histoire de C et C++

- 1970 : langage C par Ken Thompson pour écrire Unix. *Avoir un langage adapté pour manipuler la mémoire : pointeurs*.
- 1980 : développement du C++ par Bjarne Stroustrup. *Augmentation du C pour laprogrammation orientée objet.* Ajout des **classes**.
- **C et C++ :** 2 évolutions mais volonté de rétrocompatibilité.
- C++, depuis 1990 : ajout de la **programmation générique** et début de la **bibliothèque standard**. Fortes disparités avec le C malgré le passé commun.

Histoire de C et C++

- 1970 : langage C par Ken Thompson pour écrire Unix. *Avoir un langage adapté pour manipuler la mémoire : pointeurs*.
- 1980 : développement du C++ par Bjarne Stroustrup. *Augmentation du C pour laprogrammation orientée objet.* Ajout des **classes**.
- **C et C++ :** 2 évolutions mais volonté de rétrocompatibilité.
- C++, depuis 1990 : ajout de la **programmation générique** et début de la **bibliothèque standard**. Fortes disparités avec le C malgré le passé commun.
- C++, 2011 : nouveau standard majeur. Nouvelles fonctionnalités et *forte influence sur l'écriture du code*.

  C++, standards 2014 et 2017 : poursuite de cette évolution.

Histoire de C et C++

- 1970 : langage C par Ken Thompson pour écrire Unix. *Avoir un langage adapté pour manipuler la mémoire : pointeurs*.
- 1980 : développement du C++ par Bjarne Stroustrup. *Augmentation du C pour laprogrammation orientée objet.* Ajout des **classes**.
- **C et C++ :** 2 évolutions mais volonté de rétrocompatibilité.
- C++, depuis 1990 : ajout de la **programmation générique** et début de la **bibliothèque standard**. Fortes disparités avec le C malgré le passé commun.
- C++, 2011 : nouveau standard majeur. Nouvelles fonctionnalités et *forte influence sur l'écriture du code*.

  C++, standards 2014 et 2017 : poursuite de cette évolution.
- C++, standard 2020 : nouveau standard majeur (pas encore sur tous les compilateurs !).

Histoire de C et C++

- 1970 : langage C par Ken Thompson pour écrire Unix. *Avoir un langage adapté pour manipuler la mémoire : pointeurs*.
- 1980 : développement du C++ par Bjarne Stroustrup. *Augmentation du C pour laprogrammation orientée objet.* Ajout des **classes**.
- **C et C++ :** 2 évolutions mais volonté de rétrocompatibilité.
- C++, depuis 1990 : ajout de la **programmation générique** et début de la **bibliothèque standard**. Fortes disparités avec le C malgré le passé commun.
- C++, 2011 : nouveau standard majeur. Nouvelles fonctionnalités et *forte influence sur l'écriture du code*.

  C++, standards 2014 et 2017 : poursuite de cette évolution.
- C++, standard 2020 : nouveau standard majeur (pas encore sur tous les compilateurs !).

Conclusion : même si on peut encore écrire du C++ un peu comme du C, les deux langages n'ont plus grand chose à voir.

Attention, si vous connaissez déjà C ! ! !

Langage informatique : de quoi parle-t-on ?

Dans un ordinateur :

- processeur(s) (CPU)
- mémoire vive (RAM)

- carte(s) graphique(s) (GPU) avec leur processeurs et leurs mémoires

- mémoire morte (disque dur)
- périphériques (clavier, écran, souris...)

- tout ça branché sur la carte-mère.

Langage informatique : de quoi parle-t-on ?

Dans un ordinateur :

- processeur(s) (CPU)
- mémoire vive (RAM)
- carte(s) graphique(s) (GPU) avec leur processeurs et leurs mémoires
- mémoire morte (disque dur)
- périphériques (clavier, écran, souris...)
- tout ça branché sur la carte-mère.

Une **variable** en informatique ($\neq$ variable en mathématique) :

- un emplacement dans la RAM (avec adresse et taille)
- un type (manière d'encoder en une suite de bits 0 ou 1)
- un nom dans votre programme
- une *valeur*

Calculs faits par le CPU ; la valeur ensuite sauvegardée dans le disque dur.

Les outils du programmeur

1. Un éditeur de code (Geany, XCode, Code : :blocks, etc.)

Les outils du programmeur

1. Un éditeur de code (Geany, XCode, Code : :blocks, etc.)
2. Un compilateur (g++, clang++, etc.)

Les outils du programmeur

1. Un éditeur de code (Geany, XCode, Code : :blocks, etc.)
2. Un compilateur (g++, clang++, etc.)
3. quelques bibliothèques (libeigen3 pour nous pour l'algèbre linéaire).

Les outils du programmeur

1. Un éditeur de code (Geany, XCode, Code : :blocks, etc.)
2. Un compilateur (g++, clang++, etc.)
3. quelques bibliothèques (libeigen3 pour nous pour l'algèbre linéaire).
4. **du papier et des crayons ! ! !**

Les outils du programmeur

1. Un éditeur de code (Geany, XCode, Code : :blocks, etc.)
2. Un compilateur (g++, clang++, etc.)
3. quelques bibliothèques (libeigen3 pour nous pour l'algèbre linéaire).
4. **du papier et des crayons ! ! !**
5. des documentations :

   `https://en.cppreference.com/w/`

   et

   `http://www.cplusplus.com/reference/`

   savoir les lire fait partie du cours !

Buts du cours

- connaître et *être à l'aise* avec *toute* la syntaxe de base, de la plus ancienne à la plus moderne.

Buts du cours

- connaître et *être à l'aise* avec *toute* la syntaxe de base, de la plus ancienne à la plus moderne.
- savoir définir et utiliser des **classes** (POO)

Buts du cours

- connaître et *être à l'aise* avec *toute* la syntaxe de base, de la plus ancienne à la plus moderne.
- savoir définir et utiliser des **classes** (POO)
- savoir définir et utiliser des **templates** (programmation générique)

Buts du cours

- connaître et *être à l'aise* avec *toute* la syntaxe de base, de la plus ancienne à la plus moderne.
- savoir définir et utiliser des **classes** (POO)
- savoir définir et utiliser des **templates** (programmation générique)
- savoir lire la documentation

Buts du cours

- connaître et *être à l'aise* avec *toute* la syntaxe de base, de la plus ancienne à la plus moderne.
- savoir définir et utiliser des **classes** (POO)
- savoir définir et utiliser des **templates** (programmation générique)
- savoir lire la documentation
- connaître les parties les plus utilisées la bibliothèque standard.

Buts du cours

- connaître et *être à l'aise* avec *toute* la syntaxe de base, de la plus ancienne à la plus moderne.
- savoir définir et utiliser des **classes** (POO)
- savoir définir et utiliser des **templates** (programmation générique)
- savoir lire la documentation
- connaître les parties les plus utilisées la bibliothèque standard.

NE FAIT PAS PARTIE DU COURS :

les tout derniers standards C++17 et C++20, l'intégralité de la bibliothèque standard, les subtilités fines de l'héritage de classes, les notions les plus récentes des templates, les espaces de noms, le C.

Syntaxe élémentaire du C++.

Premier programme

```cpp
#include <iostream>
int main() {
    std::cout << "Bienvenue en MU4MA056.\n";
    return 0;
}
```

Compilation (dans le terminal) avec :

```
g++ essai1.cpp -o essai1.exe
```

Lancement (dans le terminal) avec

```
./essai1.exe
```

Fonction, entrée/sortie

```
#include <iostream>
#include <cmath>
double circle_area(double r) {
    return M_PI*r*r;
}
int main() {
    std::cout << "Entrez le rayon du cercle:\n";
    double x;
    std::cin  >> x;
    std::cout << "L'aire du cercle est "
          << circle_area(x) << "\n";
         return 0; }
```

Bibliothèque externe

```
#include <iostream>
#include <Eigen/Dense>// chemin vers bibliothèque
int main()
{
  Eigen::Matrix<double,4,4> A;
  A << 1, 2 , 3, 4 , 1,-1,1,-1,4,3,2,1,1,-1,0,0 ;
  std::cout << "La matrice A est:\n" << A << "\n";
  std::cout << "Son determinant est: "
            << A.determinant() << "\n";
  std::cout << "Son inverse est:\n"
        << A.inverse() << "\n";
  return 0;}
```

Compilation avec :

```
g++ -I /usr/include/eigen3 essai3.cpp -o essai3.exe
```

L'affichage produit

```
La matrice A est:
 1  2  3  4
 1 -1  1 -1
 4  3  2  1
 1 -1  0  0
Son determinant est: 40
Son inverse est:
-0.075 -0.125  0.175    0.5
-0.075 -0.125  0.175   -0.5
 0.175  0.625 -0.075   -0.5
 0.175 -0.375 -0.075    0.5
```

Déclaration des variables

Langage fortement typé : déclaration avec type.

```
// Nombres réels:
   double x=4.; //initialisé à 4
// Nombres entiers:
   int n=-2;
   long int m=-45654553365433556;
// Tableaux :
   std::vector<int> v(50,3);//50 entiers égaux à 3
   std::vector<double> w{2.3,-4.,1.1,0.};//4 réels
// Chaîne de caractères:
   std::string S("Le C++, c'est top.");
// Syntaxe alternative > C++11 préférée désormais:
   auto v = vector<int>(50,3);
   auto w = vector<double>{2.3, 4.2, 1.1, 0.};
   auto S = "Le C++, c'est top"s;
```

Variables

1. un nom (dans le code, disparaît dans l'exécutable)
2. un type (donne la taille de l'emplacement mémoire **et** spécifie le codage binaire)
3. un emplacement mémoire (une adresse dans la mémoire vive RAM)
4. une valeur (stockée à l'emplacement mémoire via le codage du type)

mais aussi :

- une durée de vie : la variable est automatiquement supprimée à la fin du bloc où elle a été déclarée.
- toutes les fonctions et opérations qui sont définis sur le type de la variable