

# Séance 1

## Programmation C++

### Notions de Base

UE Physique numérique. M1 Physique.

M. Ismail, PHITEM, Université Grenoble Alpes

1.1

### Table des matières

<b>1</b>	<b>Organisation de l'UE</b>	<b>1</b>
<b>2</b>	<b>Un premier programme</b>	<b>2</b>
<b>3</b>	<b>Identificateurs et mots-clés</b>	<b>3</b>
<b>4</b>	<b>Portée de variables</b>	<b>4</b>
<b>5</b>	<b>Expressions et opérateurs</b>	<b>5</b>
<b>6</b>	<b>Les instructions</b>	<b>6</b>
<b>7</b>	<b>Le type tableau</b>	<b>7</b>
<b>8</b>	<b>Pointeurs</b>	<b>8</b>
<b>9</b>	<b>Tableaux et pointeurs</b>	<b>8</b>
<b>10</b>	<b>Les références</b>	<b>9</b>
<b>11</b>	<b>Allocation dynamique</b>	<b>9</b>

1.2

## 1 Organisation de l'UE


UE Physique numérique

### Volume horaire

- 12h CM C++ (8 séances d'1h30)
- 12h TP C++ (8 séances d'1h30 en binômes) sous Linux

### évaluation. UE coefficient 1 : 3 ECTS

- Examen devant PC en individuel. Duree de lexamen = 2h

 **Pas de rattrapage.**

1.3

## Équipe pédagogique

- **Cours et TPs : Mourad Ismail** <Mourad.Ismail@univ-grenoble-alpes.fr> Laboratoire Interdisciplinaire de Physique, bâtiment E, bureau 311
- **Cours et TPs : Emilie Despiau-Pujo** <Emilie.Despiau-Pujo@univ-grenoble-alpes.fr> LTM (UMR 5129 CNRS/UGA), 17 av. des Martyrs, 38054 Grenoble

1.4

## Quelques mots sur le langage C++

- 2<sup>ème</sup> langage le plus utilisé au monde (1er si on le regroupe avec C). D'après <http://www.tiobe.com> (janvier 2025) : 1. Python, 2. C++, 3. C et 4. java
- Multiples paradigmes : programmation procédurale, programmation orientée objet, programmation générique, etc ...
- Pas de droits d'auteurs ©
- Développé par Bjarne Stroustrup dans les années 80 (*pour améliorer le langage C*)

1.5

## Support de cours et bibliographie

### Lien utile : supports de cours et TPs

- <https://cours.univ-grenoble-alpes.fr/course/view.php?id=7725>


### Bibliographie

- Claude Delannoy, Programmer en c++ moderne. De c++11 à c++20, 10<sup>ème</sup> édition, 848 pages
- Bjarne Stroustrup, Le langage C++ - 4e édition, 1098 pages
- Tutoriel en ligne : <http://www.cplusplus.com>

1.6





# 2 Un premier programme

## Hello World!!

-  Le plus petit programme en C++

```
int main () // fonction principale
{
    return 0; // retour de la fonction main
}
```

ltoto.CI.cpp1

- Il s'agit d'un exemple de la fonction principale (`main`) qui ne prend aucun argument et n'exécute aucune instruction !
- Tout ce qui est entre `//` et EOL est un commentaire ignoré par le compilateur
- Toutes les lignes entre `/*` et `*/` sont des commentaires
-  Tout programme C++ doit contenir une fonction nommée `main` (et une seule !)
-  Cette fonction doit retourner un entier (`int`).
-  Une valeur de retour non nulle signale un échec
-  Une valeur de retour nulle ou l'absence de valeur de retour signalent une exécution réussie

```
int main ()
{
    return 0; // retour de la fonction main
}
```

ltotoI.CI.cpp1

## 🔗 « Hello World »

```
#include <iostream> // directive du preprocesseur
                  // gestion des entrees-sorties

using namespace std; // utilisation de l'espace de nom std

int main ()
{
    cout << "Hello World" << endl; // operateur de flux. Utilisation
    return 0;                      // de la sortie standard
}
```

ou bien

```
#include <iostream>

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

`helloWorld1.C1.cpp`

## 🔗 Compilation avec g++ et clang++ sous Linux

```
g++ helloWorld.C1.cpp -o helloWorld
clang++ helloWorld.C1.cpp -o helloWorld
```

## 🔗 Exécution sous Linux

```
./helloWorld
```

Hello World

1.7

# 3 Identificateurs et mots-clés

## Déclaration et types

### Déclaration

```
type identificateur;
```

#### Exemples de types d'entiers :

- `short int`
- `int`
- `long int`

La taille de chacun de ces types dépend de la machine. Par exemple : `short int` (16 bits), `int` (32 bits), `long int` (64 bits).

## 🔗 Vérification. Fonction `sizeof`

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

int main ()
{
    cout << "nombre d'octets d'un char = " << sizeof(char) << endl;
    cout << "nombre d'octets d'un short int = " << sizeof(short int) << endl;
    cout << "nombre d'octets d'un int = " << sizeof(int) << endl;
    cout << "nombre d'octets d'un long int = " << sizeof(long int) << endl;
    cout << "nombre d'octets d'un float = " << sizeof(float) << endl;
    cout << "nombre d'octets d'un double = " << sizeof(double) << endl;
    cout << "nombre d'octets d'un long double = " << sizeof(long double) << endl;
    return 0;
}
```

### 🔗 Exécution :

```
Nbr d'oct. d'un char = 1
Nbr d'oct. d'un short int = 2
Nbr d'oct. d'un int = 4
Nbr d'oct. d'un long int = 8
Nbr d'oct. d'un float = 4
Nbr d'oct. d'un double = 8
Nbr d'oct. d'un long double = 16
```

— Dans ce cas, un `int` occupe 32 bits ce qui permet le stockage de :

$$-2^{31}, \dots, -1, 0, 1, \dots, 2^{31} - 1.$$

— Quelques types réels :

- `float`
- `double`
- `long double`

— Le type booléen :

- `bool` : `true` ou `false`

🛑 `true` possède la valeur 1

🛑 `false` possède la valeur 0

### 🔗 Exemple :

```
bool b = 5; // 5 est converti en true, donc b = true
int i = true; // true est converti en entier, donc i=1
int j = b+i; // j= 1+1 = 2
```

1.8

## Mot-clé `const`

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

int main ()
{
    const double pi = 3.1415926535897;
    const int dimension = 10;
    const int nombre_elements = 100 * dimension;

    pi = 3.1; //illegal, une constante ne peut etre modifiee
    const int size; //illegal, une constante doit etre initialisee
    return 0;
}
```

lconst.C1.cpp1

1.9

## 4 Portée de variables

### Notions de blocs, de portées et déclarations

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

int main ()
{
    int n;
    {
        //cette paire d'accolade introduit une nouvelle portee
        int m = 10; // m est accessible uniquement a l'interieur
                    // de cette portee
        n = m + 1; // OK, n est accessible
    }
}
```

```

n = m ; // Erreur de compilation, m est en dehors de la
        // presente portee
cout << " la valeur de n est " << n << endl;
return 0;
}

```

|portee.C1.cpp|

```

#include <iostream>
using namespace std;

int main ()
{
    int n;
    { //cette paire d'accolade introduit une nouvelle portee
        int m = 10; // m est accessible uniquement a l'interieur
                    // de cette portee
        n = m + 1; // OK, n est accessible
    }
    int m = 1 ; //OK car declaration d'une nouvelle variable
    n = m ;
    cout << " la valeur de n est " << n << endl;
    return 0;
}

```

|porteeBis.C1.cpp|

1.10

## 5 Expressions et opérateurs

### Opérateur d'affectation. Opérateurs arithmétiques

- `i = j` désigne une expression qui réalise une action : affecte la valeur de `j` à `i`
- addition `+`, soustraction `-`, multiplication `*`, division `/`, l'opérateur modulo `%`

#### Exemples :

- ✗ `double x = 3/4;`  $\Rightarrow x = 0$ , le reste de la division est ignoré
- ✓ `double y = double(3)/4;`  $\Rightarrow y = 0.75$ , l'entier 3 est converti en 3. au format `double`
- ✓ `double y = 3/4.;`  $\Rightarrow y = 0.75$
- ✓ `double y = 3./4.;`  $\Rightarrow y = 0.75$
- `int n = 4%3;`  $\Rightarrow n = 1$ , le reste de la division euclidienne de 4 par 3 est 1

1.11

### Les opérateurs d'affectation élargie

- `i += k;` équivalent à `i = i + k;`
- `i -= 2*n;` équivalent à `i = i - 2*n;`
- `a *= b;` équivalent à `a = a * b;`
- `a /= (b+1);` équivalent à `a = a/(b+1);`

1.12

### Les opérateurs d'incrément et de décrémentation

- `int i = 4, j = 4;`
- `i++;` `i = i + 1;` ; Soit `i = 5`
- `++j;` `j = j + 1;` ; Soit `j = 5`
- `int m = i++;`
  - 1) `m = i`,
  - 2) `i = i + 1`.

Soit  $m = 5, i = 6$ .

— `int n = ++j;`

1)  $j = j + 1$ ,

2)  $n = j$ .

Soit  $j = 6, n = 6$ .

1.13

## Les opérateurs relationnels

— `>` (strictement supérieur à)

— `<` (strictement inférieur à)

— `>=` (supérieur ou égal à)


— `<=` (inférieur ou égal à)

— `==` (égal à)

— `!=` (différent de)

### ATTENTION!

Il convient de distinguer l'opérateur d'affectation `=` de l'opérateur de test à l'égalité `==`. Voir `/affectation.C1.cpp/`

```
 int x = 1;
int y = 0;
if (y==x)
{
    cout << "Test verifie"<< endl;
}
else
{
    cout << "Test non verife" << endl;
}
```

`/affectation.C1.cpp/`

### Exécution :

Test verifie

1.14

## Les opérateurs logiques

— `&&` : opérateur *et*

— `||` : opérateur *ou*

— `!` : opérateur *non*

1.15

# 6 Les instructions

## Les instructions conditionnelles

### L'instruction *if-else*

```
if(condition_1) instruction_1
else
{
    if(condition_2) instruction_2
    else
    {
        if(condition_3) instruction_3
        else instruction_4
    }
}
```

## équivalence avec une instruction conditionnelle ternaire

```
i = a > b ? a : b;
```

```
if (a > b) i = a;
else      i = b;
```

## L'instruction *switch*

```
int i, j, k
// suite des instructions attribuant,
// entre autres, une valeur à i
switch(i)
{
    case 0 :      //exécution si i = 0
        j = 1;
        k = 2;
        break; // sortie du switch
    case 2 :      //exécution si i = 0
        j = 3;
        k = 4;
        break; // sortie du switch
    default : //exec si i diff. de 0 et 2
        j = 0;
        k = 0;
        break; // sortie du switch
}
```

---

1.16

## Les instructions d'itérations

### L'instruction *for*

```
for(expr_1 ; expr_2 ; expr_3) instruction
```

### L'instruction *while*. Faire tant que

```
while(expression) instruction
```

### L'instruction *do . . . while*. Faire jusqu'à

```
do
instruction
while(expression);
```

---

1.17

# 7 Le type tableau

## Tableaux statiques

- `double vec[10] = {0};` : définit un vecteur de doubles de taille 10 dont toutes les composantes sont initialisées à 0
- `vec[0] = 1.0;` : affectation sur le 1er élément de `vec`
- `vec[9] = 11.0;` : affectation du dernier élément de `vec`
- `const int dim = 5; int a[2*dim] = {};` : définit un vecteur d'entiers de taille 10 dont toutes les composantes sont initialisées à 0
- `for(int i = 0; i < 2*dim; i++) a[i] = i*i;` : remplissage de `a` composante par composante

---

1.18

## 8 Pointeurs

### Définition et exemple

- Une variable de type **T\*** est une variable automatique destinée à recevoir l'adresse d'une variable de type **T**

🔗 Exemple :

```
double x = 0., y = 1.;
double* ptr = 0; //ptr est initialisé par le pointeur NULL
                //il ne pointe sur rien.

ptr = &x; //on affecte l'adresse de x au pointeur ptr
cout << "je pointe sur l'objet d'adresse "<< ptr
    << " et ma valeur est "<< *ptr << endl;

double z = *ptr;
cout << "je suis un double d'adresse "<< &z
    << " et de valeur " << z << endl;

ptr = &y;
z = *ptr;
cout << "je pointe sur l'objet d'adresse "<< ptr
    << " et ma valeur est " << *ptr << endl;

/*
 * Notons que les deux dernières opérations sont équivalentes
 * à z = y mais elles ont été réalisées via un pointeur
 */
```

|pointeurs.C1.cpp|

🔗 Le résultat de l'exécution :

```
je pointe sur l'objet d'adresse 0xbfedbe20 et ma valeur est 0
je suis un double d'adresse 0xbfedbe10 et de valeur 0
je pointe sur l'objet d'adresse 0xbfedbe18 et ma valeur est 1
```

1.19

## 9 Tableaux et pointeurs

### Deux règles

- 🛑 Une expression désignant un objet de type tableau est convertie par le compilateur en un pointeur constant sur son premier élément
- 🛑 L'opérateur `[]` est défini de sorte que si `exp1` et `exp2` désignent des expressions, la première de type pointeur et la seconde de type entier. Alors les expressions suivantes sont équivalentes :

```
exp1[exp2] et *(exp1+exp2)
```

### Pointeur versus tableau

```
int tabEntiers[20];
int* ptab;
ptab = tabEntiers;
*tabEntiers = 1;
ptab[0] = 1;
*(tabEntiers+1) = 2;
ptab[1] = 2;
```

1.20

### Tableaux à 2 dimensions

- 🔗 Exemple de déclaration d'un tableau statique en 2D : `double a[5][7];`
- Le tableau `a` est stocké en mémoire ligne par ligne : `a[0][0]`, `a[0][1]`, `a[0][2]`, `a[0][3]`, `a[0][4]`, `a[0][5]`, `a[0][6]`, `a[1][0]`, `a[1][1]` . . .
- 🛑 `a` pointe vers `a[0][0]`
- 🛑 `a+7*i+j` pointe vers `a[i][j]`
- 🛑 Le tableau `a` aurait pu être déclaré comme un pointeur de pointeur : `double** a;`

1.21



## 10 Les références

### Définition et exemple

- une référence est un alias pour un objet déjà déclaré
- T&** désigne une référence sur un objet de type **T**
- Syntaxe :** `T& ref_v = v;`

### Exemple

```
int n = 5;
int& r = n; // r est une réf. sur un entier.
           // r réfère à n
int m = r; // m = 5 puisque r est un alias de n
r = 1; // n = 1 puisque r est un synonyme de n
```

1.22

### Pointeurs et Références

- Un objet et sa référence ont la même taille
- Un pointeur est toujours de taille fixe
- Exemple :

```
struct myStruct
{
    double X[10];
    double Y;
    long int Ref[150];
};

int main ()
{
    myStruct a ;
    myStruct * aPtr = &a;
    myStruct & aRef = a;
    cout << "size of myStruct a : " << sizeof(a) << endl;
    cout << "size of myStruct* aPtr : " << sizeof(aPtr) << endl;
    cout << "size of myStruct& aRef : " << sizeof(aRef) << endl;
    return 0;
}
```

lsizPointRef.C1.cpp1

### Le résultat de l'exécution :

```
size of myStruct a : 1288
size of myStruct* aPtr : 8
size of myStruct& aRef : 1288
```

1.23

## 11 Allocation dynamique

### Les opérateurs **new** et **delete**

```
T* ptr = 0; // déclaration d'un pointeur sur un
           // type T et son initialisation à 0
ptr = new T; // demande d'allocation de mémoire
           // pour un type T
// forme condensée : T* ptr = new T;
*ptr = 1; // utilisation de la variable dynamique
          // par le biais du pointeur
delete ptr; // libération de l'espace mémoire
```

### Recommandation

```
delete ptr;
ptr = 0;
```

1.24

## Les opérateurs `new[]` et `delete[]`. Tableaux dynamiques

```
T* ptr = 0;      // déclaration d'un pointeur
                //sur un type T
ptr = new T[n];  // allocation de mémoire pour
                //un tableau de type T
                // et de taille n
// forme condensée : T* ptr = new T[n];
delete[] ptr;    // libération de l'espace mémoire
ptr = 0;
```

1.25

### Exemple

#### Application : calcul des nombres premiers

```
#include <iostream>
#include <iomanip> // pour setw()
#include <cstdlib> // pour exit()
using namespace std;
int main()
{
    int max = 0;    // Nbr de nombres premiers demand\es
    int count = 3;  // Nbr de nombres premiers trouv\es
    long int test = 5; // Candidat à être un nbr premier
    bool est_premier = true; // indique qu'un nbr premier est trouv\e

    cout << "Entrez le nbr de nombres premiers d\\'esir\\'es > 3" << endl ;
    cin >> max;
    if (max < 4) exit(0);
    long* premiers = new long[max]; // allocation dynamique du tableau
                                    // des nbrs premiers

    *premiers = 2;
    *(premiers+1) = 3;
    *(premiers+2) = 5;

    do
    {
        test += 2;
        int i = 0;

        do
        {
            est_premier = (test % premiers[i]) > 0;
        } while ( (++i < count) && est_premier);

        if (est_premier)
            premiers[count++] = test;
    } while (count < max);

    for (int i = 0; i<max; i++)
    {
        if(i%5 == 0) cout << endl;
        cout << setw(10) << *(premiers+i);
    }
    cout << endl;
    delete[] premiers;
    premiers = 0;
    return 0;
}
```

lnbrPremiers.C1.cpp

#### Exécution :

Entrez le nbr de nombres premiers désirés > 3				
10				
2	3	5	7	11
13	17	19	23	29

1.26