# Introduction à la programmation
# Travaux pratiques: séance 1
# INFO0201-1

B. Baert, X. Baumans & F. Ludewig
Bruno.Baert@ulg.ac.be - Xavier.Baumans@ulg.ac.be

# Implication des TP info et Pourquoi programmer ?

- La programmation est importante en physique (analyse expérience, simulations, traitement de données,...)

# Implication des TP info et Pourquoi programmer ?

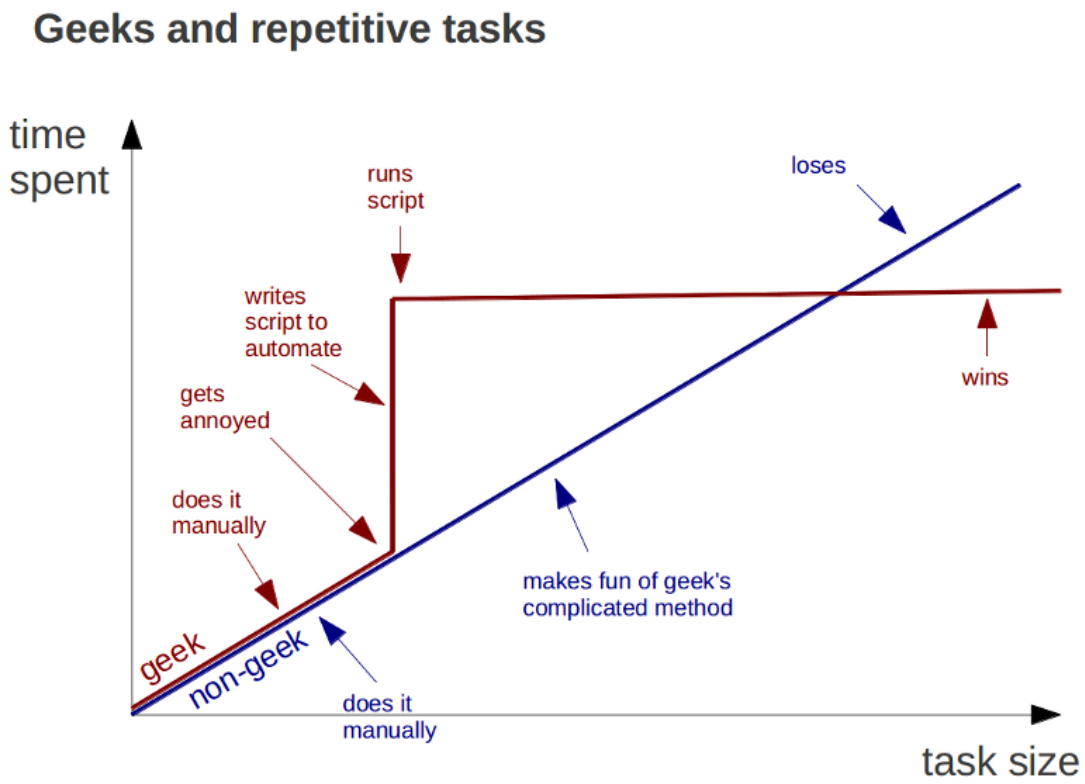
- La programmation est importante en physique (analyse expérience, simulations, traitement de données,...)
- C++ nécessaire pour le cours d'analyse numérique (*Méthodes numériques de la physique*, 2ème BAC)

# Implication des TP info et Pourquoi programmer ?

- La programmation est importante en physique (analyse expérience, simulations, traitement de données,...)
- C++ nécessaire pour le cours d'analyse numérique (*Méthodes numériques de la physique*, 2ème BAC)
- Part importante de la cote finale (70%) et interrogations

# Implication des TP info et Pourquoi programmer ?

- La programmation est importante en physique (analyse expérience, simulations, traitement de données,...)
- C++ nécessaire pour le cours d'analyse numérique (*Méthodes numériques de la physique*, 2ème BAC)
- Part importante de la cote finale (70%) et interrogations



# Comment programmer ?

Résoudre un problème

1. Penser son code (raisonnement logique et intuitif)
   *Détailler les étapes complètes du programme à concevoir - développer l'algorithme correspondant*

# Comment programmer ?

Résoudre un problème

1. Penser son code (raisonnement logique et intuitif)
   *Détailler les étapes complètes du programme à concevoir - développer l'algorithme correspondant*
2. Implémenter (contrôler et commenter chaque étape)
   *Ecrire (traduire) l'algorithme dans le langage de programmation choisi (C/C++)*

# Comment programmer ?

Résoudre un problème

1. Penser son code (raisonnement logique et intuitif)
   *Détailler les étapes complètes du programme à concevoir - développer l'algorithme correspondant*
2. Implémenter (contrôler et commenter chaque étape)
   *Ecrire (traduire) l'algorithme dans le langage de programmation choisi (C/C++)*
3. Compiler / Exécuter
   *Transformer le code en un programme exécutable par l'ordinateur*

# Comment programmer ?

Résoudre un problème

1. Penser son code (raisonnement logique et intuitif)
   *Détailler les étapes complètes du programme à concevoir - développer l'algorithme correspondant*
2. Implémenter (contrôler et commenter chaque étape)
   *Ecrire (traduire) l'algorithme dans le langage de programmation choisi (C/C++)*
3. Compiler / Exécuter
   *Transformer le code en un programme exécutable par l'ordinateur*
4. Debugger si nécessaire (retour au point 1)

# Langages de programmation

Types de langage

- Langage structuré (C,Fortran,...)

# Langages de programmation

Types de langage

- Langage structuré (C,Fortran,...)
- Langage orienté objet (C++, C#, Objective-C, Java,...)

# Langages de programmation

Types de langage

- Langage structuré (C,Fortran,...)
- Langage orienté objet (C++, C#, Objective-C, Java,...)

Programmation C++ pour ce cours

- C étendu (langage structuré)
- Notion de classe et objet

# Remarque préliminaire : commenter le code !

Remarque préliminaire : veiller à toujours commenter le code ! ! !
De nombreuses lignes de code à la suite les unes des autres deviennent très rapidement **illisibles**.
→ on ajoute du texte pour **expliquer** à quoi servent les lignes de code.

```
1  #include <iostream> // inclure la librairie pour cout
2
3  using namespace std;// utiliser l'espace de nom 'std'
4
5  int main()
6  {
7      /* La fonction main est la fonction
8      principale du programme */
9      cout << "Hello world!" << endl;
10     return 0; // Tout s'est bien déroulé
11 }
```

# Langage de programmation : C++

- Variables (**int**, **float**, **double**, ...)
  $\rightarrow$ stockage de données

# Langage de programmation : C++

- Variables (**int**, **float**, **double**, ...)
  $\rightarrow$ stockage de données
- Opérateurs arithmétiques (+, -, *, /, ...)
  $\rightarrow$ calculs arithmétiques

# Langage de programmation : C++

- Variables (**int**, **float**, **double**, ...)
  $\rightarrow$ stockage de données
- Opérateurs arithmétiques (+, -, *, /, ...)
  $\rightarrow$ calculs arithmétiques
- Entrées / Sorties (**cin**, **cout**)
  $\rightarrow$ saisie de données au clavier / affichage dans la console

# Langage de programmation : C++

- Variables (**int**, **float**, **double**, ...)
  $\rightarrow$ stockage de données
- Opérateurs arithmétiques (+, -, *, /, ...)
  $\rightarrow$ calculs arithmétiques
- Entrées / Sorties (**cin**, **cout**)
  $\rightarrow$ saisie de données au clavier / affichage dans la console
- Opérateurs de comparaison (<, >, ==, !=,...)
  $\rightarrow$ comparaison de valeurs numériques et d'expressions booléennes

# Langage de programmation : C++

- Variables (**int**, **float**, **double**, ...)
  $\rightarrow$ stockage de données
- Opérateurs arithmétiques (+, -, *, /, ...)
  $\rightarrow$ calculs arithmétiques
- Entrées / Sorties (**cin**, **cout**)
  $\rightarrow$ saisie de données au clavier / affichage dans la console
- Opérateurs de comparaison (<, >, ==, !=,...)
  $\rightarrow$ comparaison de valeurs numériques et d'expressions booléennes
- Structures de contrôle (**if/else**, **for**, **do**, **while**, ...)
  $\rightarrow$ exécution d'instructions conditionnelles ou répétitives

# Programme de la séance

- Les variables
  $\rightarrow$ stockage de données dans le programme

# Programme de la séance

- Les variables
  $\rightarrow$ stockage de données dans le programme
- Opérateurs arithmétiques et de comparaison
  $\rightarrow$ réaliser des calculs simples et des comparaisons

# Programme de la séance

- Les variables
  $\rightarrow$ stockage de données dans le programme
- Opérateurs arithmétiques et de comparaison
  $\rightarrow$ réaliser des calculs simples et des comparaisons
- Entrées/Sorties console
  $\rightarrow$ afficher du texte à l'écran et saisir des données au clavier

# Programme de la séance

- Les variables
  $\rightarrow$ stockage de données dans le programme
- Opérateurs arithmétiques et de comparaison
  $\rightarrow$ réaliser des calculs simples et des comparaisons
- Entrées/Sorties console
  $\rightarrow$ afficher du texte à l'écran et saisir des données au clavier
- Structure de contrôle conditionnelle : l'instruction **if...else**
  $\rightarrow$ exécuter certaines instructions sous condition

# Variables en C/C++

En programmation, une **variable** est un espace de stockage dans lequel on peut placer une **valeur**. On lui donne un nom (un **identifiant**) pour pouvoir y faire référence.

# Variables en C/C++

En programmation, une **variable** est un espace de stockage dans lequel on peut placer une **valeur**. On lui donne un nom (un **identifiant**) pour pouvoir y faire référence.
Il existe plusieurs types de variables :

- **int** (entier)

# Variables en C/C++

En programmation, une **variable** est un espace de stockage dans lequel on peut placer une **valeur**. On lui donne un nom (un **identifiant**) pour pouvoir y faire référence.
Il existe plusieurs types de variables :

- **int** (entier)
- **float**, **double** (réel - avec des précisions différentes)

# Variables en C/C++

En programmation, une **variable** est un espace de stockage dans lequel on peut placer une **valeur**. On lui donne un nom (un **identifiant**) pour pouvoir y faire référence.
Il existe plusieurs types de variables :

- **int** (entier)
- **float**, **double** (réel - avec des précisions différentes)
- **char** (caractère)

# Variables en C/C++

En programmation, une **variable** est un espace de stockage dans lequel on peut placer une **valeur**. On lui donne un nom (un **identifiant**) pour pouvoir y faire référence.
Il existe plusieurs types de variables :

- **int** (entier)
- **float**, **double** (réel - avec des précisions différentes)
- **char** (caractère)

On a de plus, un mot-clé permettant de signaler si le nombre doit être considéré avec un signe (négatif/positif) ou pas :

- **signed** : valeur considérée avec son signe
- **unsigned** : valeur toujours positive

# Déclaration et initialisation de variables

Déclarer une variable $\rightarrow$ **réserver la mémoire** pour sauvegarder une valeur qui correspond au type de variable.
$\longrightarrow$ Il faut toujours déclarer une variable avant de l'utiliser !
On donne un nom (identifiant) à la variable pour pouvoir y faire référence plus tard.

# Déclaration et initialisation de variables

Déclarer une variable $\rightarrow$ **réserver la mémoire** pour sauvegarder une valeur qui correspond au type de variable.
$\longrightarrow$ Il faut toujours déclarer une variable avant de l'utiliser !
On donne un nom (identifiant) à la variable pour pouvoir y faire référence plus tard.

- int a ;

# Déclaration et initialisation de variables

Déclarer une variable $\rightarrow$ **réserver la mémoire** pour sauvegarder une valeur qui correspond au type de variable.
$\longrightarrow$ Il faut toujours déclarer une variable avant de l'utiliser !
On donne un nom (identifiant) à la variable pour pouvoir y faire référence plus tard.

- int a ;
- float a ; double b ;

# Déclaration et initialisation de variables

Déclarer une variable $\rightarrow$ **réserver la mémoire** pour sauvegarder une valeur qui correspond au type de variable.
$\longrightarrow$ Il faut toujours déclarer une variable avant de l'utiliser !
On donne un nom (identifiant) à la variable pour pouvoir y faire référence plus tard.

- int a ;
- float a ; double b ;
- int a,b ;

# Déclaration et initialisation de variables

Déclarer une variable $\rightarrow$ **réserver la mémoire** pour sauvegarder une valeur qui correspond au type de variable.
$\longrightarrow$ Il faut toujours déclarer une variable avant de l'utiliser !
On donne un nom (identifiant) à la variable pour pouvoir y faire référence plus tard.

- int a ;
- float a ; double b ;
- int a,b ;
- unsigned int a ;

# Déclaration et initialisation de variables

Déclarer une variable $\rightarrow$ **réserver la mémoire** pour sauvegarder une valeur qui correspond au type de variable.

$\longrightarrow$ Il faut toujours déclarer une variable avant de l'utiliser !

On donne un nom (identifiant) à la variable pour pouvoir y faire référence plus tard.

- int a ;
- float a ; double b ;
- int a,b ;
- unsigned int a ;

Par défaut, les variables sont signées (le mot-clé **signed** est inutile)

# Déclaration et initialisation de variables

Déclarer une variable $\rightarrow$ **réserver la mémoire** pour sauvegarder une valeur qui correspond au type de variable.
$\longrightarrow$ Il faut toujours déclarer une variable avant de l'utiliser !
On donne un nom (identifiant) à la variable pour pouvoir y faire référence plus tard.

- int a ;
- float a ; double b ;
- int a,b ;
- unsigned int a ;

Par défaut, les variables sont signées (le mot-clé **signed** est inutile)

ATTENTION : initialiser les variables !
En l'absence d'initialisation, la variable prend une valeur indéterminée (pas nécessairement nulle).

# Déclaration et initialisation de variables

Déclarer une variable $\rightarrow$ **réserver la mémoire** pour sauvegarder une valeur qui correspond au type de variable.
$\longrightarrow$ Il faut toujours déclarer une variable avant de l'utiliser !
On donne un nom (identifiant) à la variable pour pouvoir y faire référence plus tard.

- int a ;
- float a ; double b ;
- int a,b ;
- unsigned int a ;

Par défaut, les variables sont signées (le mot-clé **signed** est inutile)

ATTENTION : initialiser les variables !
En l'absence d'initialisation, la variable prend une valeur indéterminée (pas nécessairement nulle).

- int a ;
  a = 0 ;

# Déclaration et initialisation de variables

Déclarer une variable $\rightarrow$ **réserver la mémoire** pour sauvegarder une valeur qui correspond au type de variable.
$\longrightarrow$ Il faut toujours déclarer une variable avant de l'utiliser !
On donne un nom (identifiant) à la variable pour pouvoir y faire référence plus tard.

- int a ;
- float a ; double b ;
- int a,b ;
- unsigned int a ;

Par défaut, les variables sont signées (le mot-clé **signed** est inutile)

ATTENTION : initialiser les variables !
En l'absence d'initialisation, la variable prend une valeur indéterminée (pas nécessairement nulle).

- int a ;
  a = 0 ;
- int a = 42 ;

# Affectation de variables

- Assigner une valeur fixée
  a = 7 ;
- Assigner la valeur d'une autre variable
  a = b ;
  → l'ancienne valeur de a est perdue

# Variables : type constant

Comme leur nom l'indique, les valeurs des *variables* peuvent être modifiées. On peut cependant vouloir déclarer des variables particulières, dont la valeur est fixée à la première affectation et ne peut plus être modifiée. Il existe pour cela le type spécifique **const**. Une variable qui est déclarée de type **const** doit être initialisée immédiatement (lors de la déclaration) et sa valeur ne pourra être modifée dans le programme, sous peine de générer une erreur dès la compilation.

# Variables : type constant

Comme leur nom l'indique, les valeurs des *variables* peuvent être modifiées. On peut cependant vouloir déclarer des variables particulières, dont la valeur est fixée à la première affectation et ne peut plus être modifiée. Il existe pour cela le type spécifique **const**. Une variable qui est déclarée de type **const** doit être initialisée immédiatement (lors de la déclaration) et sa valeur ne pourra être modifée dans le programme, sous peine de générer une erreur dès la compilation.

```
1  int main()
2  {
3    const double PI = 3.14159265358979323846;
4    const int N = 1000;
5
6    N = 2000; // ERREUR !!!
7  }
```

# Variables : int (integer)

Dans une implémentation typique (32bits), on trouve des types entiers dont les caractéristiques sont :

**int**

- **4 octets** (taille du stockage dans l'ordinateur)
- valeurs allant de **-2 147 483 648** à **2 147 483 647**

**unsigned int**

- **4 octets** (taille du stockage dans l'ordinateur)
- valeurs allant de **0** à **4 294 967 295**

# Variables : float/double (virgule flottante)

**float**

- **4 octets** (taille du stockage dans l'ordinateur)
- valeurs de $-\mathbf{3.4\ 10^{-38}}$ à $\mathbf{3.4\ 10^{38}}$
- **6** chiffres significatifs

**double**

- **8 octets** (taille du stockage dans l'ordinateur)
- valeurs de $-\mathbf{1.7\ 10^{-308}}$ à $\mathbf{1.7\ 10^{308}}$
- **14** chiffres significatifs

# Opérateurs arithmétiques et de comparaison

## Opérateurs arithmétiques

| | |
|---|---|
| Addition | + |
| Soustraction | − |
| Multiplication | * |
| Division | \ |
| Modulo | % |

## Opérateurs logiques

| | |
|---|---|
| NOT | ! |
| AND | && |
| OR | \|\| |

## Opérateurs de comparaison

| | |
|---|---|
| Egalité | == |
| Différent de | != |
| Strictement inférieur à | < |
| Inférieur à | <= |
| Strictement supérieur à | > |
| Supérieur à | >= |

## Expressions

Une **expression** est une combinaison de valeurs, variables, opérateurs et fonctions. L'expression est **évaluée** lorsqu'elle apparait dans le programme et se réduit à une **valeur**.

Par exemple, "$5 + 4$" et "$(x + 1) < 3 * 4$" sont des expressions. Lorsqu'elles sont évaluées, elles prennent respectivement pour valeurs :

- $5 + 4 \rightarrow 9$
- $(x + 1) < 3 * 4 \rightarrow$ **true** si $x < 11$ ou **false** si $x >= 11$

# Instructions et expressions

Un programme est composé d'une suite d'instructions.

## Instructions

L'instruction est le plus petit élément indépendant d'un langage de programmation.
Une **instruction** contient généralement des **expressions**, qui sont **évaluées** pour pouvoir **exécuter** l'instruction. L'instruction peut elle-même être une expression.

Une instruction est toujours clôturée par un point virgule " ;" ! ! !
Exemples :

```
1 9*3;
```

est une instruction composée d'une seule expression. L'évaluation de cette expression donne "27", mais le résultat de cette instruction n'a aucune utilité puisque le résultat n'est pas utilisé (pour une affectation dans une variable par exemple).

# Instructions et expressions

```
1 int a = 5 + 2;
```

est une instruction composée d'une expression ("$5 + 2$") et d'un opérateur d'affectation ("$=$") qui affecte la valeur de l'expression (ici "7") dans la variable nommée **a**. Le tout ("$a = 5 + 2$") est également une expression qui vaut"7".

Cela permet de réaliser des opérations d'affectation en chaîne :

```
1 int a,b;
2 a = b = 24/3; // --> a = 8, b = 8
```

L'expression "$24/3$" est évaluée et vaut 8. L'expression "$b = 24/3$" est ainsi réduite à "$b = 8$" qui affecte la valeur 8 à la variable **b**. Cette expression vaut elle-même 8, ce qui implique que l'expression "$a = b = 24/3$" se réduit à "$a = 8$" et affecte la valeur 8 à la variable **a**.

```
1 int a = 2; // la valeur 2 est affectée à la variable a
2 int b = (a = 3) + 2; // --> a = 3 et b = 5
```

Le résultat de cette instruction est multiple. L'expression entre parenthèses est d'abord évaluée : elle vaut 3 et a pour effet de modifier la valeur de **a** à cette même valeur puisque l'opérateur d'affectation a été utilisé. L'expression résultante "$3 + 2$" est ensuite évaluée, et le résultat ("5") est placé dans la variable **b**.

```
1 int a = 2, b = 3;
2 int c = (a = a + b) + 1; // --> c = 6, a = 5
```

Les deux premières instructions affectent respectivement les valeurs 2 et 3 aux variables **a** et **b**.
L'instruction suivante nécessite d'évaluer l'expression "$a = a + b$" qui est une opération d'affectation. Le résultat vaut 5 auquel est ajouté 1 et le résultat final, 6 est affecté à la variable **c** ;

# Conversion entre types de variables

On ne peut affecter aux variables que des valeurs du même type. Exemple : un entier dans un **int**.

```
1 int a = 2;
```

Si on essaye d'affecter une valeur d'un type différent du type de la variable, le langage C++ tente de convertir *implicitement* le type de valeur. Exemple : un **double** dans un **int**.

```
1 double a = 5.2;
2 int b = a; // --> b = 5;
```

On peut aussi forcer la conversion *explicite* d'un type vers un autre. Par exemple, mettre un entier dans une variable de type double.

```
1 double a = 5.2;
2 double b = a; // --> 5.2
3 double b = (int)a; // --> 5 car 'a' a d'abord été
      converti en entier
```

# Type de variable et expressions

Lors de l'évaluation d'une expression, le compilateur doit déterminer le **type** de l'expression résultante.
Il faut pour cela considérer le type des variables qui interviennent dans l'expression.

Cas de variables de même type

- (**int**) + (**int**) $\Longrightarrow$ (**int**)
- (**float**) + (**float**) $\Longrightarrow$ (**float**)
- (**double**) + (**double**) $\Longrightarrow$ (**double**)

Cas de variables de types différents

- (**int**) + (**float**) $\Longrightarrow$ (**float**)
- (**int**) + (**double**) $\Longrightarrow$ (**double**)
- (**float**) + (**double**) $\Longrightarrow$ (**double**)

Il en va de même pour les autres opérateurs.

# La bibliothèque standard du C++

Le langage C++ définit une série de fonctions et classes standardisées qui constituent la **bibliothèque standard du C++**.

Ces fonctions sont supposées être toujours fournies avec le compilateur et peuvent donc être utilisées dans tous les programmes.

# La bibliothèque standard du C++

Le langage C++ définit une série de fonctions et classes standardisées qui constituent la **bibliothèque standard du C++**.

Ces fonctions sont supposées être toujours fournies avec le compilateur et peuvent donc être utilisées dans tous les programmes.

Elles sont contenues dans plusieurs fichiers et doivent être inclues au début du programme grâce à l'instruction pré-processeur

```
1 #include <librairie>
```

# La bibliothèque standard du C++

Le langage C++ définit une série de fonctions et classes standardisées qui constituent la **bibliothèque standard du C++**.

Ces fonctions sont supposées être toujours fournies avec le compilateur et peuvent donc être utilisées dans tous les programmes.

Elles sont contenues dans plusieurs fichiers et doivent être inclues au début du programme grâce à l'instruction pré-processeur

```
1 #include <librairie>
```

On trouve des fonctions et classes pour :

- Gérer l'affichage et la saisie au clavier (**cout**, **cin**)
  $\rightarrow$ contenu dans **iostream**
- Utiliser des fonctions mathématiques avancées
  $\rightarrow$ contenu dans **cmath**
- Traiter du texte et des chaînes de caractères
  $\rightarrow$ contenu dans **string**

# La bibliothèque standard du C++ : espace de nom 'std'

Toutes les fonctions de la librairie standard sont contenues dans un **espace de nom** appelé "**std**".

Il est donc nécessaire de précéder toutes les fonctions qu'elle contient par "**std : :**"

```
1 std::cout << "texte" << std::endl;
```

# La bibliothèque standard du C++ : espace de nom 'std'

Toutes les fonctions de la librairie standard sont contenues dans un **espace de nom** appelé "**std**".

Il est donc nécessaire de précéder toutes les fonctions qu'elle contient par "**std : :**"

```
1 std::cout << "texte" << std::endl;
```

Ou d'utiliser au préalable l'instruction

```
1 using namespace std;
```

qui implique que l'espace de nom **std** est utilisé par défaut et permet ainsi de se dispenser de la notation "**std : :**".

# Entrée / Sortie

Les objets "**cin**" et "**cout**" permettent respectivement de réaliser des opérations d'entrée et de sortie.
Pour pouvoir les utiliser, il faut inclure la bibliothèque **iostream** au moyen de l'instruction pré-processeur

```
1 #include <iostream>
```

- **cout** : affichage dans la console à l'écran (d'où le **c** de **c**out)
- **cin** : lecture de données du clavier

Il est également possible de rediriger ces objets vers des fichiers pour écrire ou lire dans ceux-ci.

# Entrée / Sortie : cout

Fonctionnement de **cout** :

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

Si **x** est une variable, on peut également l'afficher :

```
1 cout << "x =" << x << endl;
```

- Le retour à la ligne est provoqué par "**endl**" ;
- Une tabulation est insérée par le caractère spécial "\t" où "\" est un **caractère d'échappement** permettant de donner une signification différente au caractère qui le suit (ici "t") ;
- Le retour à la ligne peut également être provoqué par le caractère spécial "\n".
- On modifie la précision de l'affichage avec

```
1 cout.precision(x);
```

# Entrée / Sortie : cin

Fonctionnement de **cin** :
Si **x** est une variable d'un certain type, on peut lui affecter une valeur à partir du clavier :

```
1 int x;
2 cin >> x;
```

Le programme est alors bloqué et attend qu'une valeur soit saisie au clavier et que l'utilisateur presse la touche **Enter**.

Attention à la direction des signes "»" et "«" !

```
1 cin >> x;
2 cout << x;
```

# La bibliothèque “cmath” (1/2)

La bibliothèque **cmath** contient des fonctions mathématiques avancées. Parmi celles-ci, on trouve

# La bibliothèque “cmath” (1/2)

La bibliothèque **cmath** contient des fonctions mathématiques avancées. Parmi celles-ci, on trouve

- les fonctions trigonométriques : **cos, sin, tan, acos,...**

# La bibliothèque “cmath” (1/2)

La bibliothèque **cmath** contient des fonctions mathématiques avancées. Parmi celles-ci, on trouve

- les fonctions trigonométriques : **cos, sin, tan, acos,...**
- les fonctions exponentielles et logarithmiques : **exp, log,...**

# La bibliothèque “cmath” (1/2)

La bibliothèque **cmath** contient des fonctions mathématiques avancées. Parmi celles-ci, on trouve

- les fonctions trigonométriques : **cos, sin, tan, acos,...**
- les fonctions exponentielles et logarithmiques : **exp, log,...**
- les puissances : **pow, sqrt,...**

# La bibliothèque “cmath” (1/2)

La bibliothèque **cmath** contient des fonctions mathématiques avancées. Parmi celles-ci, on trouve

- les fonctions trigonométriques : **cos, sin, tan, acos,...**
- les fonctions exponentielles et logarithmiques : **exp, log,...**
- les puissances : **pow, sqrt,...**

```
1  #include <cmath>
2
3  using namespace std;
4
5  int main()
6  {
7      double a = cos(2.*3.1415926535); // --> a = 1
8      double b = sqrt(16); // --> b = 4
9      double c = pow(2, b); // --> c = 2 ^ b = 16
10     return 0;
11 }
```

# La bibliothèque “cmath” (2/2)

Quelques exemples :

- **pow(x,y)** : calcule $x$ à la puissance $y$ ($x^y$)

# La bibliothèque “cmath” (2/2)

Quelques exemples :

- **pow(x,y)** : calcule $x$ à la puissance $y$ ($x^y$)
- **sqrt(x)** : calcule la racine carrée de $x$ ($\sqrt{x}$)

# La bibliothèque “cmath” (2/2)

Quelques exemples :

- **pow(x,y)** : calcule $x$ à la puissance $y$ ($x^y$)
- **sqrt(x)** : calcule la racine carrée de $x$ ($\sqrt{x}$)
- **log(x)** : calcule le logarithme naturel de $x$ ($\ln(x)$)

# La bibliothèque “cmath” (2/2)

Quelques exemples :

- **pow(x,y)** : calcule $x$ à la puissance $y$ ($x^y$)
- **sqrt(x)** : calcule la racine carrée de $x$ ($\sqrt{x}$)
- **log(x)** : calcule le logarithme naturel de $x$ ($\ln(x)$)
- **log10(x)** : calcule le logarithme en base 10 de $x$ ($\log(x)$)

# La bibliothèque “cmath” (2/2)

Quelques exemples :

- **pow(x,y)** : calcule $x$ à la puissance $y$ ($x^y$)
- **sqrt(x)** : calcule la racine carrée de $x$ ($\sqrt{x}$)
- **log(x)** : calcule le logarithme naturel de $x$ ($\ln(x)$)
- **log10(x)** : calcule le logarithme en base 10 de $x$ ($\log(x)$)
- **fabs(x)** : calcule la valeur absolue de $x$ ($|x|$)

# La bibliothèque “cmath” (2/2)

Quelques exemples :

- **pow(x,y)** : calcule $x$ à la puissance $y$ ($x^y$)
- **sqrt(x)** : calcule la racine carrée de $x$ ($\sqrt{x}$)
- **log(x)** : calcule le logarithme naturel de $x$ ($\ln(x)$)
- **log10(x)** : calcule le logarithme en base 10 de $x$ ($\log(x)$)
- **fabs(x)** : calcule la valeur absolue de $x$ ($|x|$)
- **floor(x)** : calcule la valeur entière directement inférieure à $x$

# La bibliothèque “cmath” (2/2)

Quelques exemples :

- **pow(x,y)** : calcule $x$ à la puissance $y$ ($x^y$)
- **sqrt(x)** : calcule la racine carrée de $x$ ($\sqrt{x}$)
- **log(x)** : calcule le logarithme naturel de $x$ ($\ln(x)$)
- **log10(x)** : calcule le logarithme en base 10 de $x$ ($\log(x)$)
- **fabs(x)** : calcule la valeur absolue de $x$ ($|x|$)
- **floor(x)** : calcule la valeur entière directement inférieure à $x$

Des informations très complètes concernant l'utilisation de ces fonctions et de nombreuses autres peuvent être obtenues à l'adresse http://www.cplusplus.com/reference/

# Structures de contrôle : if

Pour construire un programme plus complexe
$\rightarrow$ exécuter certaines instructions sous conditions

Première structure de contrôle : le "**if**"

Sa syntaxe est la suivante : **if**(*condition*) *instruction* ;

*condition* est une **expression booléenne** qui sera évaluée. Si le résultat est **true**, l'instruction sera exécutée. Si le résultat est **false**, elle sera ignorée.

```
1 int a = 3;
2 cout << "La valeur de a est " << a << endl;
3 if(a > 2)
4   cout << "a est plus grand que 2" << endl;
```

# Structures de contrôle : if...else

On peut également exécuter une autre instruction si la condition évaluée n'est pas vérifiée. Cette instruction est introduite par le mot-clé **else** :

```
1 if(a > 2)
2   cout << "a est plus grand que 2" << endl;
3 else
4   cout << "a n'est pas plus grand que 2" << endl;
```

On peut même faire suivre plusieurs conditions :

```
1 if(a > 5)
2   cout << "a est plus grand que 5" << endl;
3 else if(a < 2)
4   cout << "a est plus petit que 2" << endl;
5 else
6   cout << "a est compris entre 2 et 5" << endl;
```

# Blocs d'instructions

Dans une structure de contrôle telle que **if...else**, on ne peut exécuter qu'une seule instruction pour chaque condition.

Pour remédier à cette limitation, on utilise des **blocs d'instructions**. Ceux-ci sont constitués de plusieurs instructions placées entre parenthèses **{...}** : ils sont considérés comme une seule instruction à exécuter.

```
1 if(a > 5)
2 {
3     cout << "a est plus grand que 5" << endl;
4     int b = 5*a;
5     cout << "5 x a = " << b << endl;
6 }
```

# Indentation

Pour améliorer la lisibilité du code source, on aligne les blocs de code en accord avec la structure du programme.

Règle de base : toute section de code qui dépend hiérarchiquement d'une autre est décalée d'une tabulation vers la droite par rapport à celle-ci.

```
1 if(a > 2)
2 {
3     cout << "a est plus grand que 2" << endl;
4     if(a > 4)
5         cout << "a est plus grand que 4" << endl;
6 }
```

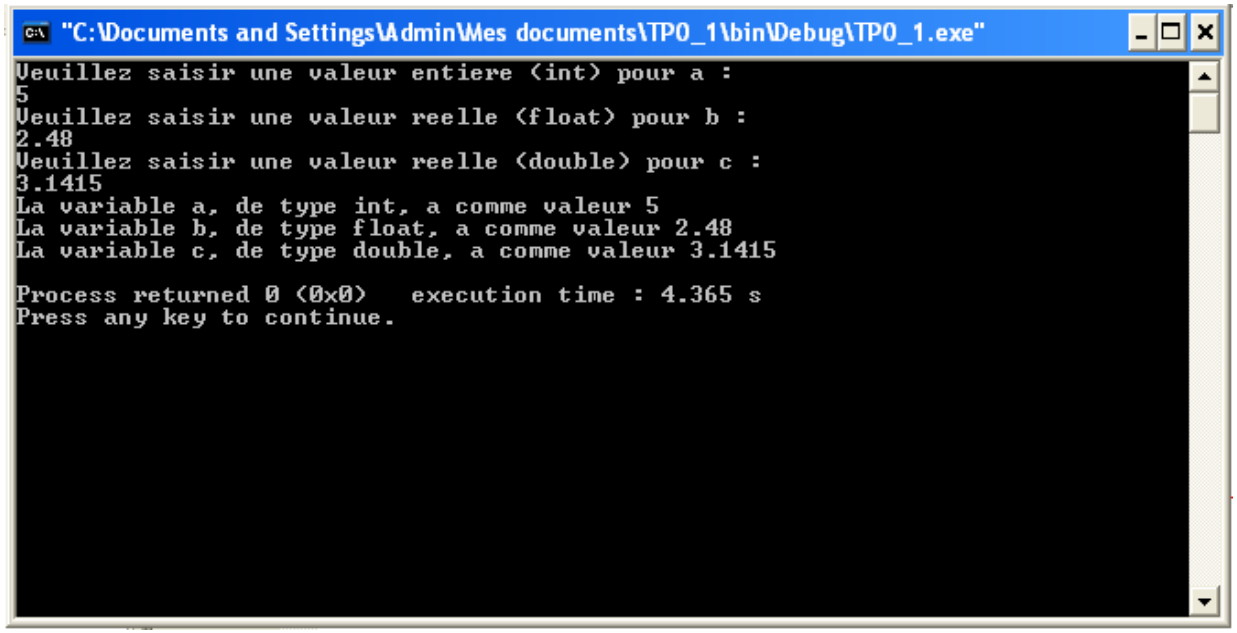
Dans le cas d'un bloc d'instruction, on aligne les parenthèses sur l'instruction précédente et on n'indente que le code situé entre les parenthèses.

# Exercices (1/5)

Pour chaque exercice, tester le programme et les effets d'une erreur de l'utilisateur ou de valeurs extrêmes.

1. Déclaration et affichage de variables :
   - Déclarer 3 variables *a*, *b* et *c* de type **int**, **float** et **double** ;
   - Affecter aux variables des valeurs saisies au clavier ;
   - Afficher les valeurs des 3 variables avec la précision nécessaire, sous la forme :
     *"La variable a, de type int, a comme valeur ...*
     *La variable b, de type float, a comme valeur...*
     *La variable c, de type double, a comme valeur..."*

   Remarque : Tester la saisie d'une valeur décimale pour la variable entière.



# Exercices (2/5)

2. Calculs arithmétiques sur 2 variables :
   - Déclarer 4 variables *i1*, *i2* de type **int** et *d1*, *d2* de type **double** ;
   - Saisir leurs valeurs au clavier ;
   - Effectuer les opérations +, -, * et / sur chaque couple de nombres ;
   - Afficher les résultats de toutes les opérations à l'écran.

   Remarque : pour la division des nombres entiers, on voudrait afficher d'abord le résultat de la division euclidienne puis celui de la division exacte (utiliser l'opérateur *modulo*).

3. Echanger deux variables :
   - Déclarer 2 variables *a* et *b* et saisir leurs valeurs au clavier ;
   - Dans le programme, placer la valeur de *a* dans *b* et celle de *b* dans *a* ;
   - Afficher les valeurs des variables *a* et *b* à l'écran (qui devraient donc avoir pris les valeurs '*b*' et '*a*', respectivement).

   Remarque : Trouver une manière de ne pas perdre la valeur de 'a' en écrivant a = b ;

# Exercices (3/5)

4. Séparation de la partie entière et non entière d'un réel :
   - Saisir au clavier un nombre réel ;
   - Afficher successivement la partie entière et la partie non entière de ce nombre (sans utiliser la fonction *floor()*)

5. Vérification d'un numéro de compte en banque :

   Soit le numéro de compte en banque '210546876857'. Les deux derniers chiffres servent à vérifier l'absence de faute de frappe et sont égaux au reste de la division entière des 10 premiers chiffres du numéro de compte par 97.
   - A partir du numéro de compte ci-dessus, calculer quels devraient être les 2 derniers chiffres du numéro de compte ;
   - Calculer les 2 derniers chiffres d'un autre numéro de compte (le vôtre, le numéro de compte pour payer votre minerval à l'ULg,...)

   Remarque : Pour pouvoir stocker un nombre entier de 10 chiffres ($\equiv 10^{11} - 1 > 2^{32} = 4\ 294\ 967\ 296 \approx 10^{9.6}$), déclarer une variable de type **int64_t** qui permet d'utiliser des entiers codés sur 64 bits ;

# Exercices (4/5)

6. Calcul d'une distance de tir :
   Un joueur de basket à l'arrêt tente un tir à distance en direction du panier. Déterminer à quelle distance $d$ du panier le lancé doit être effectué en fonction des paramètres tels que l'angle $\alpha$ du lancé, la hauteur $h$ du panier et la vitesse $v$ initiale du ballon. La relation entre tous ces éléments est la suivante :

$$d = \frac{v\cos(\alpha)}{g}\left(v\sin(\alpha) + \sqrt{(v\sin(\alpha))^2 - 2gh}\right)$$

   - Saisir au clavier les valeurs de $\alpha$ en degrés, $h$ en m et $v$ en m/s ;
   - Calculer la distance $d$ à laquelle le lancer doit être effectué.

   Remarque : Au basket, le panier est généralement placé à une hauteur de 3.05 m et on choisira $g = 9.81\text{m/s}^2$).

7. Tests conditionnels multiples :
   - Saisir un nombre entier au clavier ;
   - Déterminer si le nombre est divisible par 2 et afficher s'il l'est ou pas ;
   - Déterminer si le nombre est divisible par 3 et afficher s'il l'est ou pas ;
   - Déterminer si le nombre est divisible par 5 et afficher s'il l'est ou pas ;
   - Déterminer si la racine carrée du nombre peut-être calculée, et l'afficher le cas échéant.

# Exercices (5/5)

8. Conversion format temporel :
   - Saisir un nombre entier au clavier. Celui-ci correspond à une durée exprimée en secondes ; *(par exemple : 2262 secondes)*
   - Décomposer ce nombre en heures, minutes et secondes :
     - le nombre d'heures est le multiple entier de 3600 contenu dans le nombre total (1h = 3600s)
     - le nombre de minutes est le multiple entier de 60 (1 minute = 60s) contenu dans le restant (le nombre total moins les secondes correspondant aux heures complètes)
     - le nombre de secondes est finalement le nombre restant après avoir retiré toutes celles correspondant aux heures et aux minutes complètes.
   - Afficher à l'écran la durée au format "x heure(s) y minute(s) et z seconde(s)".

Remarque : n'afficher "heure(s)" et "minute(s)" que si cela est nécessaire.