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

X. Baumans
(xavier.baumans@ulg.ac.be)

Programme de la séance

- Déclarations de variables

Programme de la séance

- Déclarations de variables
- Opérateurs de base (arithmétique, logique, comparaison)

Programme de la séance

- Déclarations de variables
- Opérateurs de base (arithmétique, logique, comparaison)
- Entrée/sortie => "cout"/"cin"

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**.
→ Ajouter du texte pour **expliciter** 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

Variables en C/C++

Il existe plusieurs types de variables :

- **int** (entier)

Variables en C/C++

Il existe plusieurs types de variables :

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

Variables en C/C++

Il existe plusieurs types de variables :

- **int** (entier)
- **float**, **double** (réel - avec des précisions différentes)
- **boolean** (booléen, 2 valeurs : True/False)

Variables en C/C++

Il existe plusieurs types de variables :

- **int** (entier)
- **float**, **double** (réel - avec des précisions différentes)
- **boolean** (booléen, 2 valeurs : True/False)
- **char** (caractère)

Variables en C/C++

Il existe plusieurs types de variables :

- **int** (entier)
- **float**, **double** (réel - avec des précisions différentes)
- **boolean** (booléen, 2 valeurs : True/False)
- **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 : 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	>=

Instructions et expressions

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 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 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**.

Instructions et expressions

```
1 a = 2; // la valeur 2 est affectée à la variable a
2 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 a = 2; b = 3;
2 c = (a == b); // --> c = false
```

Les deux premières instructions affectent respectivement les valeurs 2 et 3 aux variables **a** et **b**.

L'instruction suivante nécessite d'évaluer l'expression "$a == b$" qui est une opération de comparaison. **a** et **b** ont des valeurs différentes, l'évaluation de l'expression donne donc un résultat **false**, qui est affecté à la variable **c**.

Retour au type des 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 le type de valeur. Exemple : un **double** dans un **int**.

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

On peut également forcer la conversion d'un type vers un autre. Par exemple pour 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**) = (**int**)
- (**float**) + (**float**) = (**float**)
- (**double**) + (**double**) = (**double**)

Cas de variables de type différent

- (**int**) + (**float**) = (**float**)
- (**int**) + (**double**) = (**double**)
- (**float**) + (**double**) = (**double**)

Il en va de même pour les autres opérateurs.

Entrée / Sortie

Les objets "**cin**" et "**cout**" permettent respectivement de réaliser des opérations d'entrée-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;
```

Exercices (1/3)

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 de type **int**, **float** et **double** ;
   - Affecter aux variables des valeurs saisies au clavier ;
   - Afficher les valeurs des 3 variables avec la précision nécessaire.
2. Calculs arithmétiques sur 2 variables :
   - Déclarer 2 variables de chacun des types **int**, **float** et **double** ;
   - Saisir leurs valeurs au clavier ;
   - Effectuer les opérations +, -, * et / sur chacun des couples de nombre ;
   - Afficher les résultats de toutes les opérations.

   Remarque : pour la division des nombres entiers, on voudrait afficher d'abord le résultat de la division euclidienne puis celui de la division exacte
3. Echanger deux variables :
   - Déclarer 2 variables et saisir leurs valeurs au clavier ;
   - Echanger les valeurs des 2 variables et les afficher.

Exercices (2/3)

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.
5. Calcul 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 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,...)

Exercices (3/3)

6. Aire d'un carré et d'un disque :
   - Saisir au clavier une valeur **x1** ;
   - Calculer l'aire d'un carré dont les côtés mesurent **x1** ;
   - Calculer l'aire d'un disque dont le diamètre est **x1** ;
   - Saisir une autre valeur **x2** ;
   - Calculer les aires des carrés et disques correspondant ;
   - Vérifier si le rapport des aires pour la première valeur **x1** est le même qu'avec la valeur **x2** et l'afficher à l'écran.
7. Table de vérité de la fonction XOR :
   L'opérateur XOR n'est pas défini en C++. Cependant il est équivalent à l'expression A **XOR** $B = (A||B)$ && $(!A||!B)$
   - Afficher dans la console la table de vérité de l'opérateur **XOR**.