# Introduction à la programmation
# Travaux pratiques: séance 3
# INFO0201-1

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



# Programme de la séance

- Opérateurs d'affectation et unaires (+=, -=, ++, −−, ...)
  → expressions condensées, code plus concis

# Programme de la séance

- Opérateurs d'affectation et unaires (+=, -=, ++, −−, ...)
  → expressions condensées, code plus concis

- Le type **char** et le code ASCII (caractères alphanumériques)

# Programme de la séance

- Opérateurs d'affectation et unaires (+=, -=, ++, −−, ...)
  → expressions condensées, code plus concis

- Le type **char** et le code ASCII (caractères alphanumériques)

- Structure de contrôle conditionnelle : l'instruction **switch**
  → nombreux cas différents (plus concis que le **if**)

# Programme de la séance

- Opérateurs d'affectation et unaires (+=, -=, ++, −−, ...)
  → expressions condensées, code plus concis
- Le type **char** et le code ASCII (caractères alphanumériques)
- Structure de contrôle conditionnelle : l'instruction **switch**
  → nombreux cas différents (plus concis que le **if**)
- Librairies mathématiques
  → utilisation de fonctions mathématiques supplémentaires

# Opérateurs d'affectation condensés

Il existe une série d'opérateurs qui permettent de réaliser une **opération** avec une variable et de remplacer immédiatement sa **valeur** par le **résultat** de l'opération.

# Opérateurs d'affectation condensés

Il existe une série d'opérateurs qui permettent de réaliser une **opération** avec une variable et de remplacer immédiatement sa **valeur** par le **résultat** de l'opération.

Exemple avec l'addition :

```
1 int a = 2;
2 int b = 5;
3 a += b; // --> a = a + b
4 cout << "a = " << a << endl; // affiche 7
```

# Opérateurs d'affectation condensés

Il existe une série d'opérateurs qui permettent de réaliser une **opération** avec une variable et de remplacer immédiatement sa **valeur** par le **résultat** de l'opération.
Exemple avec l'addition :

```
1 int a = 2;
2 int b = 5;
3 a += b; // --> a = a + b
4 cout << "a = " << a << endl; // affiche 7
```

De la même manière, on peut utiliser les opérateurs suivants :

| | | |
|---|---|---|
| += | a+=b | $a = a + b$ |
| -= | a-=b | $a = a - b$ |
| *= | a*=b | $a = a * b$ |
| /= | a/=b | $a = a/b$ |
| %= | a%=b | $a = a\%b$ |

# Opérateurs unaires

Il existe en C++ des **opérateurs** qui ne nécessitent qu'un seul **opérande** : ce sont les **opérateurs unaires** $++$ et $--$.

Ceux-ci servent respectivement à **incrémenter** et **décrémenter** la valeur d'une variable. Ceux-ci ont également un comportement différent selon qu'ils sont utilisés sous leur forme **préfixée** ou **suffixée**.

# Opérateurs unaires

Il existe en C++ des **opérateurs** qui ne nécessitent qu'un seul **opérande** : ce sont les **opérateurs unaires** $++$ et $--$.

Ceux-ci servent respectivement à **incrémenter** et **décrémenter** la valeur d'une variable. Ceux-ci ont également un comportement différent selon qu'ils sont utilisés sous leur forme **préfixée** ou **suffixée**.

```
1 int a = 2, b = 0;
2 a++; // --> a = 3
3 b = a++; // --> b = 3, a = 4
4 b = ++a; // --> b = 5, a = 5
```

# Opérateurs unaires

Il existe en C++ des **opérateurs** qui ne nécessitent qu'un seul **opérande** : ce sont les **opérateurs unaires** $++$ et $--$.

Ceux-ci servent respectivement à **incrémenter** et **décrémenter** la valeur d'une variable. Ceux-ci ont également un comportement différent selon qu'ils sont utilisés sous leur forme **préfixée** ou **suffixée**.

```
1 int a = 2, b = 0;
2 a++; // --> a = 3
3 b = a++; // --> b = 3, a = 4
4 b = ++a; // --> b = 5, a = 5
```

Lorsque l'opérateur est **préfixé**, l'opération est d'abord effectuée sur l'opérande et le résultat est utilisé dans l'expression correspondante.

Lorsque l'opérateur est **suffixé**, la valeur actuelle de l'opérande est utilisée dans l'expression puis l'opération est appliquée à l'opérande.

# Le type char

Le type **char** permet de stocker un **caractère** dans une variable.

En interne, il s'agit en fait d'un petit entier, permettant de stocker des valeurs comprises entre $-128$ et $+127$. Les valeurs numériques correspondent au code ASCII du caractère et peuvent ainsi représenter les différents caractères.
Exemple : le caractère "A" possède le code ASCII 64.

```
1 char caractere = 'A'; // caractère A == 64
2 char c = 97; // caractère a == 97
```

ATTENTION : L'affectation à un type **char** s'effectue avec des guillemets simples ('), contrairement au texte (chaînes de caractères) qui s'écrit avec des guillemets doubles (").

```
1 char g = "B"; // ERREUR, cela ne compilera pas !
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (1/4)

Test plus concis qu'avec de nombreux **if...else** imbriqués.

```
1  int a = 3;
2  switch(a)
3  {
4    case 1: // si a == 1
5      cout << "a vaut 1" << endl;
6      break;
7    case 2: // si a == 2
8    case 3: // si a == 3
9      cout << "a vaut 2 ou 3" << endl;
10     break;
11   default: // dans tous les autres cas
12     cout << "a ne vaut pas 1, 2 ou 3" << endl;
13     break;
14 }
```

# Structure de contrôle conditionnelle : switch...case (2/4)

- Chaque valeur à tester est introduite par le mot-clé **case** :

```
1   case 5:
```

# Structure de contrôle conditionnelle : switch...case (2/4)

- Chaque valeur à tester est introduite par le mot-clé **case** :

```
1   case 5:
```

- La fin des instructions à exécuter après vérification d'un cas est signalée par le mot-clé **break**

```
1   case 5:
2     // instructions
3     break;
```

# Structure de contrôle conditionnelle : switch...case (2/4)

- Chaque valeur à tester est introduite par le mot-clé **case** :

```
1   case 5:
```

- La fin des instructions à exécuter après vérification d'un cas est signalée par le mot-clé **break**

```
1   case 5:
2     // instructions
3     break;
```

En l'absence de **break**, l'exécution des instructions se poursuit.

```
1   case 5:
2     // instructions
3   case 6:
4     // instructions exécutées dans les cas 6 ET 5
5     break;
```

# Structure de contrôle conditionnelle : switch...case (3/4)

Un mot-clé spécifique, **default**, permet de couvrir tous les cas restants :

```
1  switch(a)
2  {
3    case 1:
4      // instructions pour a == 1
5      break;
6    case 2:
7      // instructions pour a == 2
8      break;
9    default:
10     // instructions pour les autres cas
11     // (i.e. a!=1 && a!=2)
12     break;
13 }
```

# Structure de contrôle conditionnelle : switch...case (4/4)

On peut aussi tester des caractères alphanumériques (type **char**).

```
1  switch(a) // a est de type 'char'
2  {
3    case 'a':
4      cout << "ceci est un a" << endl;
5      break;
6    case 'b':
7      cout << "ceci est un b" << endl;
8      break;
9    default:
10     cout << "une lettre indéterminée" << endl;
11     break;
12 }
```

# 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 : :**".

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

# 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/

# Exercices (1/2)

1. Détection de lettres et voyelles :
   - Saisir un caractère alphanumérique (type **char**) au clavier ;
   - S'il s'agit d'un "a", afficher "première lettre de l'alphabet" ;
   - S'il s'agit d'un "z", afficher "dernière lettre de l'alphabet" ;
   - S'il s'agit d'une voyelle, afficher "ceci est une voyelle" ;
   - Sinon, afficher "ceci est une consonne".

2. Calculatrice avancée
   - Afficher à l'écran une liste de fonctions à sélectionner par un numéro ou un caractère alphanumérique ;
   - Saisir au clavier l'opération et les opérandes ;
   - A l'aide d'une structure **switch...case**, réaliser l'opération demandée ;
   - Afficher le résultat à l'écran.

   Remarque : vérifier la cohérence des données entrées (pas de racine carrée négative, pas de division par zéro, etc...)

# Exercices (2/2)

Code ASCII

3. Conversion majuscule <=> minuscule
   - Entrer au clavier un des 26 caractères alphabétiques ;
   - Si c'est une majuscule, faire en sorte de renvoyer la minuscule correspondante ;
   - Si c'est une minuscule, renvoyer la majuscule correspondante ;

   Remarque : Faire attention à la gestion des caractères non alphabétiques en renvoyant, par exemple, un message à l'attention de l'utilisateur distrait...

Erreurs numériques avec des nombres réels :

4. Calcul de $[(10/3) - 3]\ x\ 3 - 1$ en plusieurs étapes :
   - Calculer a = 10.0/3.0 ;
   - Calculer b = a - 3. ;
   - Calculer c = b*3. ;
   - Calculer d = c - 1 ;

   Quel est le résultat ? Est-il conforme à celui attendu ?