Introduction à l'informatique
Travaux pratiques: séance 3
INFO0205-1

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

Programme de la séance

- Structures de contrôle itératives (**boucles**) :

Programme de la séance

- Structures de contrôle itératives (**boucles**) :
  - **for**

Programme de la séance

- Structures de contrôle itératives (**boucles**) :
  - **for**
  - **while** & **do...while**

Programme de la séance

- Structures de contrôle itératives (**boucles**) :
  - **for**
  - **while** & **do...while**

$\rightarrow$ Très utile (et souvent indispensable) quand il est nécessaire d'**exécuter plusieurs fois** une même instruction (ou bloc d'instructions) !

Structure de contrôle itérative : boucle while (1/4)

Le premier type de boucle est la boucle **while**.

Structure de contrôle itérative : boucle while (1/4)

Le premier type de boucle est la boucle **while**.
Sa syntaxe est la suivante : **while**(*condition*) {*instructions ;*}

Structure de contrôle itérative : boucle while (1/4)

Le premier type de boucle est la boucle **while**.
Sa syntaxe est la suivante : **while**(*condition*) {*instructions ;*}

- *condition* est une **expression booléenne** qui est évaluée ;

Structure de contrôle itérative : boucle while (1/4)

Le premier type de boucle est la boucle **while**.
Sa syntaxe est la suivante : **while**(*condition*) {*instructions ;*}

- *condition* est une **expression booléenne** qui est évaluée ;
- si le résultat est **true**, les *instructions* sont exécutées ;

Structure de contrôle itérative : boucle while (1/4)

Le premier type de boucle est la boucle **while**.
Sa syntaxe est la suivante : **while**(*condition*) {*instructions ;*}

- *condition* est une **expression booléenne** qui est évaluée ;
- si le résultat est **true**, les *instructions* sont exécutées ;
- ensuite, *condition* est à nouveau évaluée ;

Structure de contrôle itérative : boucle while (1/4)

Le premier type de boucle est la boucle **while**.
Sa syntaxe est la suivante : **while**(*condition*) {*instructions ;*}

- *condition* est une **expression booléenne** qui est évaluée ;
- si le résultat est **true**, les *instructions* sont exécutées ;
- ensuite, *condition* est à nouveau évaluée ;
- si le résultat est encore **true**, les *instructions* sont à nouveau exécutées ;

Structure de contrôle itérative : boucle while (1/4)

Le premier type de boucle est la boucle **while**.
Sa syntaxe est la suivante : **while**(*condition*) {*instructions ;*}

- *condition* est une **expression booléenne** qui est évaluée ;
- si le résultat est **true**, les *instructions* sont exécutées ;
- ensuite, *condition* est à nouveau évaluée ;
- si le résultat est encore **true**, les *instructions* sont à nouveau exécutées ;
- cette situation se répète jusqu'à ce que *condition* soit évalué à la valeur **false**.

Structure de contrôle itérative : boucle while (2/4)

Un exemple simple

```
1 int i = 0;
2 while (i < 5)
3 {
4   cout << "La valeur de i est : " << i << endl;
5   i++; // --> i = i + 1;
6 }
```

Ce code va afficher :

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
```

Structure de contrôle itérative : boucle while (2/4)

Un exemple simple

```
1  int i = 0;
2  while (i < 5)
3  {
4    cout << "La valeur de i est : " << i << endl;
5    i++; // --> i = i + 1;
6  }
```

Ce code va afficher :

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
```

Structure de contrôle itérative : boucle while (2/4)

Un exemple simple

```
1  int i = 0;
2  while (i < 5)
3  {
4    cout << "La valeur de i est : " << i << endl;
5    i++; // --> i = i + 1;
6  }
```

Ce code va afficher :

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
```

Structure de contrôle itérative : boucle while (2/4)

Un exemple simple

```
1  int i = 0;
2  while (i < 5)
3  {
4    cout << "La valeur de i est : " << i << endl;
5    i++; // --> i = i + 1;
6  }
```

Ce code va afficher :

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
```

Structure de contrôle itérative : boucle while (2/4)

Un exemple simple

```
1  int i = 0;
2  while (i < 5)
3  {
4    cout << "La valeur de i est : " << i << endl;
5    i++; // --> i = i + 1;
6  }
```

Ce code va afficher :

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
```

Structure de contrôle itérative : boucle while (2/4)

Un exemple simple

```
1  int i = 0;
2  while (i < 5)
3  {
4    cout << "La valeur de i est : " << i << endl;
5    i++; // --> i = i + 1;
6  }
```

Ce code va afficher :

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
```

Structure de contrôle itérative : boucle while (2/4)

Un exemple simple

```
1  int i = 0;
2  while (i < 5)
3  {
4    cout << "La valeur de i est : " << i << endl;
5    i++; // --> i = i + 1;
6  }
```

Ce code va afficher :

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
```

Structure de contrôle itérative : boucle while (2/4)

Un exemple simple

```
1  int i = 0;
2  while (i < 5)
3  {
4    cout << "La valeur de i est : " << i << endl;
5    i++; // --> i = i + 1;
6  }
```

Ce code va afficher :

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
```

Structure de contrôle itérative : boucle while (2/4)

Un exemple simple

```
1  int i = 0;
2  while (i < 5)
3  {
4    cout << "La valeur de i est : " << i << endl;
5    i++; // --> i = i + 1;
6  }
```

Ce code va afficher :

```
La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
```

Structure de contrôle itérative : boucle while (3/4)

Autre exemple : calculer la série (qui tend vers 2 pour $k \to \infty$)

$$\sum_{k=0}^{\infty} \frac{1}{2^k} \text{ tant que } \frac{1}{2^k} > 0.05$$

Structure de contrôle itérative : boucle while (3/4)

Autre exemple : calculer la série (qui tend vers 2 pour $k \to \infty$)

$$\sum_{k=0}^{\infty} \frac{1}{2^k} \text{ tant que } \frac{1}{2^k} > 0.05$$

```
1  double somme = 0.;
2  double i = 1.;
3  while(i > 0.05)
4  {
5    somme += i;
6    i/=2.;
7    cout << "valeur = " << somme << endl;
8  }
```

```
valeur = 1
valeur = 1.5
valeur = 1.75
```

Structure de contrôle itérative : boucle while (4/4)

ATTENTION : Si la condition est mal exprimée, la boucle peut tourner un nombre infini de fois ! Le programme est alors bloqué.

```
1 int i = 5;
2 while(i != 2) // mal exprimé !
3                // "i <= 2" aurait évité l'erreur
4 {
5  i = i + 1; // Cette boucle va se répéter sans fin
6 }
```

Structure de contrôle itérative : boucle while (4/4)

ATTENTION : Si la condition est mal exprimée, la boucle peut tourner un nombre infini de fois ! Le programme est alors bloqué.

```
1 int i = 5;
2 while(i != 2) // mal exprimé !
3                // "i <= 2" aurait évité l'erreur
4 {
5  i = i + 1; // Cette boucle va se répéter sans fin
6 }
```

Lorsque l'on se trouve dans une telle situation, où le programme est bloqué par une **boucle infinie**, la combinaison de touches **Ctrl-C** permet d'interrompre brutalement le programme et de résoudre le problème.

Structure de contrôle itérative : boucle do...while (1/3)

Exemple : saisir une valeur positive au clavier et le vérifier

```
1  int nombre;
2  cout << "Entrez un nombre positif :" << endl;
3  cin >> nombre;
4  while(nombre < 0)
5  {
6    cout << "Entrez un nombre positif :" << endl;
7    cin >> nombre;
8  }
9  cout << "Le nombre positif est " << nombre << endl;
```

Pour pouvoir écrire la condition “nombre < 0”, il faut d'abord saisir une valeur au clavier, ce qui n'est pas pratique ni élégant. En effet, les mêmes instructions sont écrites deux fois alors que le but des structures de boucles est justement d'éviter cela.

Structure de contrôle itérative : boucle do...while (2/3)

Pour remédier à ce problème, on peut utiliser une structure de boucle légèrement différente : la boucle **do...while**

Sa syntaxe est la suivante : **do** {*instructions ;* } **while** (*condition*) ;

Structure de contrôle itérative : boucle do...while (2/3)

Pour remédier à ce problème, on peut utiliser une structure de boucle légèrement différente : la boucle **do...while**

Sa syntaxe est la suivante : **do** {*instructions ;* } **while** (*condition*) ;

Avec cette structure de boucle, les *instructions* sont d'abord exécutées. Ensuite la *condition* est évaluée, et si elle est évaluée à la valeur **true**, les *instructions* sont exécutées à nouveau.

Structure de contrôle itérative : boucle do...while (2/3)

Pour remédier à ce problème, on peut utiliser une structure de boucle légèrement différente : la boucle **do...while**

Sa syntaxe est la suivante : **do** {*instructions ;* } **while** (*condition*) ;

Avec cette structure de boucle, les *instructions* sont d'abord exécutées. Ensuite la *condition* est évaluée, et si elle est évaluée à la valeur **true**, les *instructions* sont exécutées à nouveau.

Cette structure implique que les *instructions* sont toujours exécutées **au moins une fois**.

Structure de contrôle itérative : boucle do...while (3/3)

L'exemple précédent peut ainsi se ré-écrire

```
1  int nombre;
2  do
3  {
4    cout << "Entrez un nombre positif :" << endl;
5    cin >> nombre;
6  } while(nombre < 0);
7  cout << "Le nombre positif est " << nombre << endl;
```

Les instructions effectuant la saisie du nombre au clavier sont ainsi exécutées une première fois avant l'évaluation de la condition.

Ce code a exactement la même fonction que le précédent, mais il est plus concis et plus lisible.

Structure de contrôle itérative : boucle for (1/2)

Dernier type de boucle : la **boucle for** est utilisée pour répéter des instructions un certain nombre déterminé de fois.

Son utilisation (contrairement aux autre boucles **while** et **do...while**) est particulièrement indiquée lorsque le nombre d'itérations est connu ou peut être calculé facilement.

Structure de contrôle itérative : boucle for (1/2)

Dernier type de boucle : la **boucle for** est utilisée pour répéter des instructions un certain nombre déterminé de fois.

Son utilisation (contrairement aux autre boucles **while** et **do...while**) est particulièrement indiquée lorsque le nombre d'itérations est connu ou peut être calculé facilement.

Sa syntaxe est la suivante :
for(*initialisation* ; *condition* ; *itération*) { *instructions ;* }

Structure de contrôle itérative : boucle for (1/2)

Dernier type de boucle : la **boucle for** est utilisée pour répéter des instructions un certain nombre déterminé de fois.

Son utilisation (contrairement aux autre boucles **while** et **do...while**) est particulièrement indiquée lorsque le nombre d'itérations est connu ou peut être calculé facilement.

Sa syntaxe est la suivante :
for(*initialisation* ; *condition* ; *itération*) { *instructions ;* }

La séquence d'évènements se déroule ainsi :

Structure de contrôle itérative : boucle for (1/2)

Dernier type de boucle : la **boucle for** est utilisée pour répéter des instructions un certain nombre déterminé de fois.

Son utilisation (contrairement aux autre boucles **while** et **do...while**) est particulièrement indiquée lorsque le nombre d'itérations est connu ou peut être calculé facilement.

Sa syntaxe est la suivante :
for(*initialisation* ; *condition* ; *itération*) { *instructions ;* }

La séquence d'évènements se déroule ainsi :

1. *initialisation* est évaluée ;

Structure de contrôle itérative : boucle for (1/2)

Dernier type de boucle : la **boucle for** est utilisée pour répéter des instructions un certain nombre déterminé de fois.

Son utilisation (contrairement aux autre boucles **while** et **do...while**) est particulièrement indiquée lorsque le nombre d'itérations est connu ou peut être calculé facilement.

Sa syntaxe est la suivante :
for(*initialisation* ; *condition* ; *itération*) { *instructions ;* }

La séquence d'évènements se déroule ainsi :

1. *initialisation* est évaluée ;
2. *condition* est évaluée. Si le résultat est **false**, fin de la boucle ;

Structure de contrôle itérative : boucle for (1/2)

Dernier type de boucle : la **boucle for** est utilisée pour répéter des instructions un certain nombre déterminé de fois.

Son utilisation (contrairement aux autre boucles **while** et **do...while**) est particulièrement indiquée lorsque le nombre d'itérations est connu ou peut être calculé facilement.

Sa syntaxe est la suivante :
for(*initialisation* ; *condition* ; *itération*) { *instructions ;* }

La séquence d'évènements se déroule ainsi :

1. *initialisation* est évaluée ;
2. *condition* est évaluée. Si le résultat est **false**, fin de la boucle ;
3. sinon, les *instructions* sont exécutées ;

Structure de contrôle itérative : boucle for (1/2)

Dernier type de boucle : la **boucle for** est utilisée pour répéter des instructions un certain nombre déterminé de fois.

Son utilisation (contrairement aux autre boucles **while** et **do...while**) est particulièrement indiquée lorsque le nombre d'itérations est connu ou peut être calculé facilement.

Sa syntaxe est la suivante :
for(*initialisation* ; *condition* ; *itération*) { *instructions ;* }

La séquence d'évènements se déroule ainsi :

1. *initialisation* est évaluée ;
2. *condition* est évaluée. Si le résultat est **false**, fin de la boucle ;
3. sinon, les *instructions* sont exécutées ;
4. *itération* est évaluée ;

Structure de contrôle itérative : boucle for (1/2)

Dernier type de boucle : la **boucle for** est utilisée pour répéter des instructions un certain nombre déterminé de fois.

Son utilisation (contrairement aux autre boucles **while** et **do...while**) est particulièrement indiquée lorsque le nombre d'itérations est connu ou peut être calculé facilement.

Sa syntaxe est la suivante :
for(*initialisation* ; *condition* ; *itération*) { *instructions ;* }

La séquence d'évènements se déroule ainsi :

1. *initialisation* est évaluée ;
2. *condition* est évaluée. Si le résultat est **false**, fin de la boucle ;
3. sinon, les *instructions* sont exécutées ;
4. *itération* est évaluée ;
5. retour au point 2.

Structure de contrôle itérative : boucle for (2/2)

Exemple : compter jusque 10

```
1 for(int c=0; c <= 10; c++)
2 {
3   cout << "La valeur de c est " << c << endl;
4 }
```

Structure de contrôle itérative : boucle for (2/2)

Exemple : compter jusque 10

```
1 for(int c=0; c <= 10; c++)
2 {
3   cout << "La valeur de c est " << c << endl;
4 }
```

Autre exemple : calculer la factorielle d'un nombre

```
1 int factorielle = 1;
2 int nombre = 5; // calculer la factorielle de 5
3 for(int i=nombre; i > 1; i--)
4 {
5   factorielle *= i;
6 }
7 cout << "La factorielle de " << nombre;
8 cout << " vaut " << factorielle << endl;
```

Equivalence des types de boucles

Toutes les structures itératives (**while**, **do..while** et **for**) sont interchangeables. Certaines sont simplement plus faciles ou intuitives à utiliser dans certains cas.
Exemple : compter de 2 à 7

Equivalence des types de boucles

Toutes les structures itératives (**while**, **do..while** et **for**) sont interchangeables. Certaines sont simplement plus faciles ou intuitives à utiliser dans certains cas.

Exemple : compter de 2 à 7

```
1 for(int a=2; a < 8; a++)
2 {
3   cout << a << endl;
4 }
```

Equivalence des types de boucles

Toutes les structures itératives (**while**, **do..while** et **for**) sont interchangeables. Certaines sont simplement plus faciles ou intuitives à utiliser dans certains cas.

Exemple : compter de 2 à 7

```
1 for(int a=2; a < 8; a++)
2 {
3   cout << a << endl;
4 }
```

```
1 int a = 2;
2 while(a < 8)
3 {
4   cout << a << endl;
5   a++;
6 }
```

Equivalence des types de boucles

```
1 int a = 2;
2 do{
3   cout << a << endl;
4   a++;
5 } while (a < 8);
```

Equivalence des types de boucles

```
1 int a = 2;
2 do{
3   cout << a << endl;
4   a++;
5 } while (a < 8);
```

Ces trois exemples de code aboutissent au même résultat, pour autant que l'initialisation et les conditions soient adaptées au type de boucle utilisée.

- La **boucle for** est particulièrement indiquée lorsque l'on connait le nombre d'itérations, puisque les initialisations et incrémentations sont prévues ;
- La boucle **do...while** permet de pallier à certaines difficultés qui peuvent être rencontrées avec la boucle **while** lorsqu'une première itération est nécessaire avant de pouvoir évaluer la condition.

Exercices (1/5)

Erreurs numériques avec des réels

1. Boucles de comptage : répéter les exercices suivants en utilisant successivement les 3 types de boucles.
   - Ecrire un programme qui compte de 0 à 100 par incréments de 1. Vérifier que le résultat final est bien égal à 100.
   - Ecrire un programme qui compte de 0 à 1 par pas de 0.1. Vérifier que le résultat final est bien égal à 1.
2. Précision numérique :
   - Déclarer 3 nombres $a = 10^{30}$, $b = -10^{30}$ et $c = 1$;
   - Calculer $(a + b) + c$ et $a + (b + c)$;
   - Comparer les résultats.

Exercices (2/5)

3. Diviseurs d'un entier
   - Demander à l'utilisateur de rentrer un nombre entier dont il souhaite connaître les diviseurs
   - Lui donner la possibilité de se voir afficher la liste complète des diviseurs de l'entier d'intérêt
   - Lui donner comme autre possibilité de tester des valeurs de diviseur à la main (autant d'essais qu'il le désire)

Exercices (3/5)

4. Suite de Fibonacci et nombre d'or :
   La suite de Fibonacci est une suite de nombres entiers dont chaque terme est la somme des deux précédents :
   $1, 1, 2, 3, 5, 8, 13, 21, \ldots$ Le rapport de deux termes consécutifs de cette suite tend vers le nombre d'or Φ.
   - Saisir au clavier la précision e avec laquelle on veut calculer Φ ;
   - Calculer les termes de la suite de Fibonacci jusqu'à ce que la différence entre deux valeurs calculées consécutives de Φ soit inférieure à la précision e. Tester avec $e = 0.0001$;
   - Afficher le nombre d'or Φ calculé et le nombre d'itérations nécessaires pour l'obtenir.

   Pour ce faire :
   - Soient a et b les deux derniers termes actuels de la suite de Fibonacci. La valeur actuelle de Φ est donc b/a ;
   - Calculer le terme suivant de la suite : $c = a + b$;
   - Calculer la nouvelle valeur de Φ_{new} : celle-ci vaut c/b ;
   - Continuer à procéder ainsi TANT QUE $|\Phi_{new} - \Phi| > e$ avec e la précision voulue.

Exercices (4/5)

5. Le juste prix :
   - Demander au premier joueur de saisir une valeur au clavier ;
   - Demander au second joueur d'essayer de trouver cette valeur ;
   - Si la valeur donnée est la bonne, féliciter le joueur !
   - Si la valeur est erronée, dire au joueur si la valeur cherchée est plus petite ou plus grande que la proposition et recommencer ;
   - Si le joueur n'a pas trouvé la bonne solution après 10 essais, lui annoncer qu'il a perdu et terminer le programme.

Exercices (5/5)

6. Mastermind avec des nombres :
   - Demander au premier joueur de choisir une suite de 4 chiffres ;
   - Demander au second joueur de proposer une suite de 4 chiffres ;
   - Si la proposition est correcte (chiffres et positions identiques), féliciter le joueur !
   - Si la proposition est incorrecte
     - Mentionner le nombre de chiffres corrects mais mal placés ;
     - Mentionner le nombre de chiffres correctement placés ;
     - Demander au joueur de ré-essayer.
   - Après 10 essais infructueux, annoncer au joueur qu'il a perdu et terminer le programme.