Introduction à l'informatique
Travaux pratiques: séance 4
INFO0205-1

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

Programme de la séance

- Rappels structures de contrôle itératives
  $\rightarrow$ **while**, **do...while**, **for**

Programme de la séance

- Rappels structures de contrôle itératives
  $\rightarrow$ **while**, **do...while**, **for**
- Rappel et synthèse des règles de bonne pratique

Programme de la séance

- Rappels structures de contrôle itératives
  $\rightarrow$ **while**, **do...while**, **for**
- Rappel et synthèse des règles de bonne pratique
- Nombres aléatoires
  $\rightarrow$ Génération de nombres pseudo-aléatoires

Structure de contrôle itérative : boucle while

```
1 while(condition)
2 {
3   instructions;
4 }
```

Boucle réalisée "tant que" la condition est vérifiée (vraie).
Attention à bien influer sur la condition dans les instructions sinon : **boucle infinie !**
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 }
```

Structure de contrôle itérative : boucle while

```
1 while(condition)
2 {
3   instructions;
4 }
```

Boucle réalisée "tant que" la condition est vérifiée (vraie).
Attention à bien influer sur la condition dans les instructions sinon : **boucle infinie !**
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 }
```

Structure de contrôle itérative : boucle while

```
1 while(condition)
2 {
3   instructions;
4 }
```

Boucle réalisée "tant que" la condition est vérifiée (vraie).
Attention à bien influer sur la condition dans les instructions sinon : **boucle infinie !**
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 }
```

Structure de contrôle itérative : boucle while

```
1 while(condition)
2 {
3   instructions;
4 }
```

Boucle réalisée "tant que" la condition est vérifiée (vraie).
Attention à bien influer sur la condition dans les instructions sinon : **boucle infinie !**
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 }
```

Structure de contrôle itérative : boucle while

```
1 while(condition)
2 {
3   instructions;
4 }
```

Boucle réalisée "tant que" la condition est vérifiée (vraie).

Attention à bien influer sur la condition dans les instructions sinon : **boucle infinie !**

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 }
```

Structure de contrôle itérative : boucle while

```
1 while(condition)
2 {
3   instructions;
4 }
```

Boucle réalisée "tant que" la condition est vérifiée (vraie).
Attention à bien influer sur la condition dans les instructions sinon : **boucle infinie !**
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 }
```

Structure de contrôle itérative : boucle while

```
1 while(condition)
2 {
3   instructions;
4 }
```

Boucle réalisée "tant que" la condition est vérifiée (vraie).
Attention à bien influer sur la condition dans les instructions sinon : **boucle infinie !**
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 }
```

Structure de contrôle itérative : boucle while

```
1 while(condition)
2 {
3   instructions;
4 }
```

Boucle réalisée "tant que" la condition est vérifiée (vraie).

Attention à bien influer sur la condition dans les instructions sinon : **boucle infinie !**

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 }
```

Structure de contrôle itérative : boucle while

```
1 while(condition)
2 {
3   instructions;
4 }
```

Boucle réalisée "tant que" la condition est vérifiée (vraie).
Attention à bien influer sur la condition dans les instructions sinon : **boucle infinie !**
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 }
```

Structure de contrôle itérative : boucle do...while

```
1 do
2 {
3   instructions;
4
5 }while(condition);
```

Structure indiquée quand on veut exécuter le bloc d'instruction au moins une fois avant de vérifier la condition :

```
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;
```

Structure de contrôle itérative : boucle for

```
1 for(initialisation; condition; itération)
2 {
3   instructions;
4 }
```

Son utilisation est particulièrement indiquée lorsque le nombre d'itérations est connu ou peut être calculé facilement.

Exemple simple : compter jusque 10

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

Règles de bonne pratique

Règles de bonne pratique :

- Réfléchir à la structure du programme avant de coder !
  Réfléchir d'abord aux différentes étapes nécessaires pour résoudre le problème : quelles variables seront nécessaires, faut-il des boucles, des tests conditionnels, etc...
  Idéalement, écrire le plan du programme sur papier !

Règles de bonne pratique

Règles de bonne pratique :

- Réfléchir à la structure du programme avant de coder !
  Réfléchir d'abord aux différentes étapes nécessaires pour résoudre le problème : quelles variables seront nécessaires, faut-il des boucles, des tests conditionnels, etc...
  Idéalement, écrire le plan du programme sur papier !
- Indenter correctement le code !
  L'alignement des blocs de code rend le programme beaucoup plus lisible et structuré. Il permet aussi de repérer facilement des accolades "{" manquantes. Utiliser la commande automatique "Plugins/Source code formatter" de Code::Blocks.
  Cela est valable pour l'examen aussi !

Règles de bonne pratique

- Commenter le code un maximum !
  Tout le monde doit pouvoir comprendre à quoi sert le code sans refaire le raisonnement complet.
  Cela est valable pour l'examen aussi !

Règles de bonne pratique

- Commenter le code un maximum !
  Tout le monde doit pouvoir comprendre à quoi sert le code sans refaire le raisonnement complet.
  Cela est valable pour l'examen aussi !
- Lire les erreurs du compilateur
  Lorsqu'une erreur se produit à la compilation, le compilateur renvoie un message d'erreur contenant la ligne du code source à laquelle l'erreur a été détectée et la raison de l'erreur.

  Attention de toujours commencer par la première erreur, la corriger, puis tenter de recompiler. En effet, les erreurs suivantes peuvent découler de la première et disparaître lorsque celle-ci a été corrigée.
  $\rightarrow$ La moindre des choses est de rendre un programme qui compile !

Nombres aléatoires

Il est fréquent en programmation d'avoir besoin de générer des **nombres aléatoires**. Pour ce faire, il existe une fonction

rand() ;

Cette fonction retourne un entier positif pseudo-aléatoire compris dans l'intervalle [0 :RAND_MAX].

Pour l'utiliser, il faut inclure la bibliothèque **cstdlib** :

```
1 #include <cstdlib>
```

Nombres aléatoires : manipulations

- Pour définir les limites de l'intervalle des nombres générés
  $\rightarrow$ Utilisation de la fonction modulo

```
1 int a = rand()%100; // nombre entre 0 et 99
```

Nombres aléatoires : manipulations

- Pour définir les limites de l'intervalle des nombres générés
  $\rightarrow$ Utilisation de la fonction modulo

```
1 int a = rand()%100; // nombre entre 0 et 99
```

- Obtenir un nombre réel
  $\rightarrow$ Convertir en réel et diviser

```
1 // nombre entre 0 et 1
2 double a = (double)rand()/RAND_MAX;
3
4 // nombre entre 10 et 20
5 double b = 10. + 10.*(double)rand()/RAND_MAX;
```

Nombres aléatoires : initialisation du générateur

Le générateur produira toujours la même suite de nombres (pseudo-aléatoires). Pour obtenir des nombres aléatoires qui ne soient pas identiques d'une exécution à l'autre, on utilise la fonction **srand(x)** qui permet de commencer la série à un endroit déterminé.

Cette valeur de départ x est appelée ***seed*** (graine) et doit être différente à chaque exécution.
On utilise généralement pour cette valeur le nombre de secondes écoulées depuis le 1^{er} Janvier 1970 (qui est donc différent à chaque exécution). Ce nombre est obtenu par la fonction **time(NULL)**.

```
1 // initialisation:
2 srand(time(NULL));
3 // nombre aléatoire différent à chaque exécution:
4 int a = rand();
```

La fonction **time** nécessite d'inclure la libraire <**ctime**>

Exercices

1. Terminer les exercices (au moins le 4 et le 5) de la séance précédente...
2. Suite de nombres aléatoires
   - Écrire un petit programme qui affiche dans la console une suite de nombres aléatoires tous compris entre 0 et 6 ;
   - Faire de même, mais en s'arrangeant pour afficher cette fois-ci des nombres compris entre 1 et 6.
3. Lancés d'un dé
   - Écrire un petit programme qui simule un grand nombre (choisi au clavier) de lancés d'un dé et qui calcule, sur cette base, la probabilité d'obtenir chacune des faces.