Introduction à la programmation
Travaux pratiques: séances 7 & 8
INFO0201-1

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

26-28/03/2014

Programme des séances : récapitulation et révisions

Récapitulation des notions de base :

- Variables
  $\rightarrow$ déclaration, initialisation et portée

Programme des séances : récapitulation et révisions

Récapitulation des notions de base :

- Variables
  $\rightarrow$ déclaration, initialisation et portée
- Structures de contrôle
  $\rightarrow$ **if**...**else**, **switch case**(),...

Programme des séances : récapitulation et révisions

Récapitulation des notions de base :

- Variables
  $\rightarrow$ déclaration, initialisation et portée
- Structures de contrôle
  $\rightarrow$ **if**...**else**, **switch case**(),...
- Structures de contrôle itératives
  $\rightarrow$ **while**, **do**...**while**, **for**

Programme des séances : récapitulation et révisions

Récapitulation des notions de base :

- Variables
  $\rightarrow$ déclaration, initialisation et portée
- Structures de contrôle
  $\rightarrow$ **if**...**else**, **switch case**(),...
- Structures de contrôle itératives
  $\rightarrow$ **while**, **do**...**while**, **for**
- Nombres aléatoires
  $\rightarrow$ Génération de nombres pseudo-aléatoires

Programme des séances : récapitulation et révisions

Récapitulation des notions de base :

- Variables
  $\rightarrow$ déclaration, initialisation et portée
- Structures de contrôle
  $\rightarrow$ **if**...**else**, **switch case**(),...
- Structures de contrôle itératives
  $\rightarrow$ **while**, **do**...**while**, **for**
- Nombres aléatoires
  $\rightarrow$ Génération de nombres pseudo-aléatoires
- Tableaux statiques
  $\rightarrow$ déclaration, initialisation, indices,...

Programme des séances : récapitulation et révisions

Récapitulation des notions de base :

- Variables
  $\rightarrow$ déclaration, initialisation et portée
- Structures de contrôle
  $\rightarrow$ **if**...**else**, **switch case**(),...
- Structures de contrôle itératives
  $\rightarrow$ **while**, **do**...**while**, **for**
- Nombres aléatoires
  $\rightarrow$ Génération de nombres pseudo-aléatoires
- Tableaux statiques
  $\rightarrow$ déclaration, initialisation, indices,...
- Erreurs courantes et débogage

Variables : déclaration, initialisation et portée

- Déclaration : toute variable utilisée dans le programme doit **d'abord** être déclarée !
  Une bonne pratique qui permet de **structurer** un programme consiste à déclarer toutes les variables principales en début de programme, dans une "zone de déclaration". Les variables qui ne sont nécessaires que de manière temporaire continueront évidemment à être déclarées là où elles sont nécessaires.
- Initialisation : toute variable non initialisée possède une valeur indéterminée (pas nécessairement zéro)
- Portée des variables : une variable déclarée entre un groupe d'accolades {...} n'existe qu'entre ces accolades. Elle ne peut être utilisée ni avant ni après ces accolades !
  Par exemple, dans la boucle for

```
1 for(int i=0; i < 10; i++)
```

la variable *i* n'existe qu'à l'intérieur des accolades {...} de la boucle, et ne peut pas être utilisée après

Structures de contrôle : if...else, switch case

- Structure **if...else** :

```
1  if(condition1)
2  {
3    // instructions exécutées si condition1 est VRAI
4  }
5  else if(condition2)
6  {
7    // instructions exécutées si
8    // condition2 est VRAI ET condition1 est FAUX
9  }
10 else
11 {
12   // instructions exécutées dans les autres cas
13 }
```

Structures de contrôle : if...else, switch case

- Structure **switch case** :

```
1  switch(une_variable)
2  {
3    case 1:
4      // instructions exécutées si une_variable == 1
5      break;
6    case 3:
7    case 4:
8      // instructions exécutées si une_variable == 3
9      // OU une_variable == 4
10     break;
11   default:
12     //instructions exécutées dans les autres cas
13 }
```

Structures de contrôle : if...else, switch case

- Bloc d'instructions : lorsque plusieurs instructions doivent être exécutées dans une structure **if**, il faut entourer ces instructions avec des accolades **{...}**
- Expression de la condition dans les structures **if...else** : ATTENTION à la différence entre opérateur d'**affectation** et opérateur de **comparaison** !

```
1 if(a == 2) // On compare les valeurs 'a' et '2'
```

```
1 if(a = 2) // On change la valeur de a !!!!!
```

Structures de contrôle itératives : while, do...while, for

- Structure de boucle **while** :

```
1 while(condition_pour_continuer)
2 {
3   // instructions exécutées tant que
      'condition_pour_continuer' est VRAI
4 }
```

- Structure de boucle **do...while** :

```
1 do
2 {
3   // instructions exécutées tant que
4   // 'condition_pour_continuer' est VRAI
5 }while(condition_pour_continuer);
```

ATTENTION : dans la boucle **do...while**, la condition est vérifiée **après** chaque exécution de la série d'instructions.

Structures de contrôle itératives : while, do...while, for

- Structure de boucle **for** :

```
1 for(int i=debut; i < fin; i++)
2 {
3   // instructions exécutées (fin-debut-1) fois,
      avec des valeurs de i allant de 'debut' à
      'fin'-1
4 }
```

ATTENTION : **pas** de point-virgule après

```
1 for(int i=debut; i < fin; i++)
```

ni après

```
1 while(condition_pour_continuer)
```

Structures de contrôle itératives : while, do...while, for

Quand utiliser chaque type de boucle ?

- Boucle **for** : Quand on connaît le nombre d'itérations (parcourir un tableau, répéter des instructions un nombre précis de fois, ...)
- Boucle **while** : Quand les instructions doivent se répéter tant qu'une condition logique n'est pas fausse (convergence d'une solution, etc...)
- Boucle **do...while** : Quand on devrait utiliser une boucle **while** mais que la condition de la boucle ne peut être évaluée avant d'avoir exécuté les instructions une première fois.

Les tableaux statiques

- Déclaration = réserver la place en mémoire

```
1 // tableau de 'taille' entiers
2 int nom_tableau[taille];
3 // tableau de taille_1 * taille_2 élements double
4 double nom_tableau_2D[taille_1][taille_2];
```

- Initialisation = donner des valeurs de départ

```
1 int maurice[5] = {19, 42, 785, 361, 7};
```

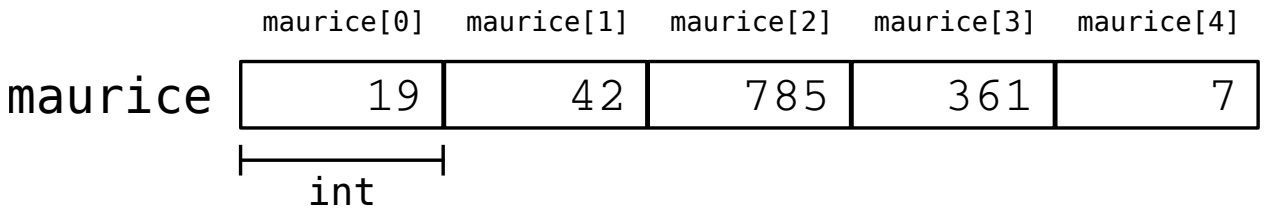
ATTENTION : l'*initialisation* peut uniquement se faire avec des valeurs 'constantes', connues avant l'exécution du programme. Donc **pas des variables** !
Pour utiliser des variables, il faut le faire élément par élément :

```
1 maurice[0] = a;
2 maurice[1] = b;
3 ...
```

Les tableaux statiques

- Numérotation des éléments : les **indices** pour accéder aux éléments du tableau vont de 0 à $N - 1$, où N est le nombre d'éléments du tableau.
  ATTENTION : ne pas accéder à un élément hors du tableau !

```
1 int maurice[5] = {19, 42, 785, 361, 7};
```



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.

Erreurs courantes

- Diviser deux entiers : quand on divise deux **int** l'un par l'autre, on obtient un nombre entier, même si l'on place le résultat dans un **double**. Pour obtenir un résultat non-entier, il faut écrire

```
1 int a;
2 int b;
3 double c = (double)a/b;
```

Erreurs courantes

- Diviser deux entiers : quand on divise deux **int** l'un par l'autre, on obtient un nombre entier, même si l'on place le résultat dans un **double**. Pour obtenir un résultat non-entier, il faut écrire

```
1 int a;
2 int b;
3 double c = (double)a/b;
```

- Expression de comparaison : utiliser = (affectation) à la place de == (comparaison)

Erreurs courantes

- Diviser deux entiers : quand on divise deux **int** l'un par l'autre, on obtient un nombre entier, même si l'on place le résultat dans un **double**. Pour obtenir un résultat non-entier, il faut écrire

```
1 int a;
2 int b;
3 double c = (double)a/b;
```

- Expression de comparaison : utiliser = (affectation) à la place de == (comparaison)

- Oublier un point-virgule : toutes les instructions se terminent par un point-virgule.

Exercices de synthèse (1/4)

Problème : écrire un programme grâce auquel l'utilisateur peut saisir une série de données expérimentales. Le programme calcule ensuite la moyenne et l'écart-type de ces données expérimentales.

$$\bar{y} = \frac{1}{N} \sum_{i}^{N} y_i \qquad \sigma = \sqrt{\overline{y^2} - \bar{y}^2}$$

Exercices de synthèse (2/4)

Problème : écrire un programme qui calcule le nombre d'Euler e avec une précision de 10^{-9}. Utiliser, par exemple, le fait que ce nombre peut être calculé grâce à la série suivante :

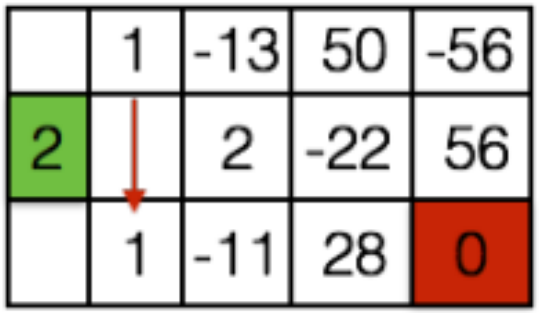
$$\sum_{i=0}^{+\infty} \frac{1}{i!} = 1 + \frac{1}{1} + \frac{1}{1 \times 2} + \frac{1}{1 \times 2 \times 3} + \ldots$$

Exercices de synthèse (3/4)

Problème : écrire un programme qui réalise un grand nombre de tirages aléatoires de deux dés (valeurs de 1 à 6).
Calculer la probabilité d'obtenir chacune des valeurs possibles pour la somme des deux dés.

Exercices de synthèse (4/4)

Problème : écrire un programme qui factorise un polynôme de degré trois grâce à la méthode de Horner. Demander les coefficients, trouver le diviseur du terme indépendant pour lequel le polynôme s'annule. Réaliser la méthode de Horner et écrire le polynôme comme le produit des deux polynômes obtenus.
Ex : $P(x) = x^3 - 13x^2 + 50x - 56, P(2) = 0$



Solution : $P(x) = (x - 2)(x^2 - 11x + 28)$
Tester le programme avec : $P(x) = x^3 - 2x^2 + 8x - 16$