

Information, Calcul, Communication (partie programmation) : Structures de contrôle en C++ (1) : branchements conditionnels

Jean-Cédric Chappelier

Faculté I&C

Objectifs de la leçon d'aujourd'hui

- ▶ Ce qu'il faut savoir sur les premières **structures de contrôle** en C++
 - ▶ branchements
 - ▶ conditions
 - ▶ type `bool`
- ▶ Etude(s) de cas
- ▶ Réponses aux questions

Rappel du calendrier

	MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 8-10	cours prog. 45 min. Jeudi 10-11
1 10.09.26	--	-1	prise en main	Bienvenue/Introduction
2 17.09.26	1. variables	0	variables / expressions	variables / expressions
3 24.09.26	2. if	0	if – switch	if – switch
4 01.10.26	3. for/while	0	for / while	for / while
5 08.10.26	4. fonctions	0	fonctions (1)	fonctions (1)
6 15.10.26		1	fonctions (2)	fonctions (2)
- 22.10.26				
7 29.10.26	5. tableaux (vector)	1	vector	vector
8 05.11.26	6. string + struct	1	array / string	array / string
9 12.11.26		2	structures	structures
10 19.11.26	7. pointeurs	2	pointeurs	pointeurs
11 26.11.26		-	entrées/sorties	entrées/sorties
12 03.12.26		-	erreurs / exceptions	erreurs / exceptions
13 10.12.26		-	révisions	théorie : sécurité
14 17.12.26	8. étude de cas	-	révisions	Révisions

Les différentes structures de contrôle

On distingue 3 types de structures de contrôle :

les branchements conditionnels : *si ... alors ...*

Si $\Delta = 0$

| $x \leftarrow -\frac{b}{2}$

Sinon

| $x \leftarrow \frac{-b-\sqrt{\Delta}}{2}, y \leftarrow \frac{-b+\sqrt{\Delta}}{2}$

les boucles conditionnelles : *tant que ...*

Tant que pas arrivé

| avancer d'un pas

Répéter

| poser la question
jusqu'à réponse valide

les itérations : *pour ... allant de ... à ... , pour ... parmi ...*

$$x = \sum_{i=1}^5 \frac{1}{i^2}$$

$x \leftarrow 0$

Pour i de 1 à 5

| $x \leftarrow x + \frac{1}{i^2}$

Branchement conditionnel

Le branchement conditionnel permet d'exécuter des traitements selon certaines conditions.

La syntaxe générale d'un branchement conditionnel est

```
if (condition)  
    Instructions 1  
else  
    Instructions 2
```

La condition est tout d'abord évaluée puis, si le résultat de l'évaluation est vrai alors la séquence d'instructions 1 est exécutée, sinon la séquence d'instructions 2 est exécutée.

Instructions 1 et *Instructions 2* sont soit une **instruction élémentaire**, soit un **bloc d'instructions**.



Choix multiples



En C++, on peut écrire de façon plus synthétique l'enchaînement de plusieurs conditions dans le cas où l'on teste différentes valeurs d'une *même* expression :

```
if (i == 1)
    Instructions 1
else if (i == 12)
    Instructions 2
...
else if (i == 36)
    Instructions N
else
    Instructions N+1
```



```
switch (i) {
    case 1:
        Instructions 1;
        break;

    case 12:
        Instructions 2;
        break;

    ...

    case 36:
        Instructions N;
        break;

    default:
        Instructions N+1;
        break;
}
```

4 contraintes :

- ▶ que des `if`
- ▶ que des tests d'égalité de valeur (`==`)
- ▶ ne teste que la *même* expression
- ▶ que des valeurs entières connues à la compilation



Exemple plus complexe



Si on ne met pas de `break`, l'exécution ne passe pas à la fin du `switch`, mais continue l'exécution des instructions du `case` suivant :

```
switch (a+b) {  
  case 2:  
  case 8: instruction2; // lorsque (a+b) vaut 2 ou 8  
  case 4:  
  case 3: instruction3; // lorsque (a+b) vaut 2, 3, 4 ou 8  
    break;  
  case 0: instruction1; // exécuté uniquement lorsque  
    break;           // (a+b) vaut 0  
  default: instruction4; // dans tous les autres cas  
    break;  
}
```

ATTENTION PIÈGE !



Ne pas confondre l'opérateur de test d'égalité `==` et l'opérateur d'affectation `=` !

`x = 3` : affecte la valeur 3 à la variable `x`
(et donc modifie cette dernière)

`x == 3` : teste la valeur de la variable `x`, renvoie `true` si elle vaut 3 et `false` sinon
(et donc ne modifie pas la valeur de `x`)



Évaluation « paresseuse »



Les opérateurs logiques `and` (ou `&&`) et `or` (ou `||`) effectuent une **évaluation « paresseuse »** (« *lazy evaluation* ») de leurs arguments :

l'évaluation des arguments se fait de la gauche vers la droite et seuls les arguments strictement nécessaires à la détermination de la valeur logique sont évalués.

Ainsi, dans `X1 and X2 and ... and Xn`, les arguments `Xi` ne sont évalués que *jusqu'au 1er argument faux* (s'il existe, auquel cas l'expression est fausse, sinon l'expression est vraie) ;

Exemple : dans `(x != 0.0) and (3.0/x > 12.0)` le second terme ne sera effectivement évalué uniquement si `x` est non nul. La division par `x` ne sera donc jamais erronée.

Et dans `X1 or X2 or ... or Xn`, les arguments ne sont évalués que *jusqu'au 1er argument vrai* (s'il existe, auquel cas l'expression est vraie, sinon l'expression est fausse).

Exemple : dans `(x == 0.0) or (3.0/x <= 12.0)` le second terme ne sera effectivement évalué uniquement si `x` est non nul.



Opérateurs



Opérateurs arithmétiques

*	multiplication
/	division
%	modulo
+	addition
-	soustraction
++	incrément (1 opérande)
--	décrément (1 opérande)

Opérateurs de comparaison

==	teste l'égalité logique
!=	non égalité
<	inférieur
>	supérieur
<=	inférieur ou égal
>=	supérieur ou égal

Opérateurs logiques

and	&&	« et » logique
or		ou
not	!	négation (1 opérande)

Priorités (par ordre décroissant, tous les opérateurs d'un même groupe sont de priorité égale) :

not ++ --, * / %, + -, < <= > >=, == !=, and, or

Etude(s) de cas

- ▶ reprendre l'équation du second degré
 👉 cf « exercice 0 »
- ▶ calculer (sans produire d'erreur) des valeurs de la fonction

$$f(x) = \frac{\sqrt{20 + 7x - x^2} \log\left(\frac{1}{x+5}\right)}{\frac{x}{10} - \sqrt{\log(x^3 - 3x + 7)} - \frac{x^2}{5}}$$