Information, Calcul et Communication

Composante Pratique: Programmation C++

MOOC semaine 2: instruction conditionnelle

Force d'un type et conversion ; conversion automatique

Booléen : opérateurs de comparaison, conversions

Instruction conditionnelle / instruction contrôlée

Opérateurs logiques / évaluation paresseuse

Choisir entre if-else ou switch

Opérateur ternaire

Force d'un type et conversion ; conversion automatique

Types élémentaires

Constantes littérales

`int`	*entier signé en complément à 2*	*32 bits*	**`26`**
`float`	*virgule flottante IEEE 754 simple précision*	*32 bits*	
`double`	*virgule flottante IEEE 754 double précision*	*64 bits*	**`3.`**
`char`	*un seul caractère alphanumérique*	*8 bits*	'**`x`**'
`bool`	**`true`** *ou* **`false`** *(1 bit)*	*mais 8 bits en mémoire centrale*	

Les opérateurs arithmétiques et logiques qui travaillent avec deux opérandes doivent avoir **le même type pour ces 2 opérandes**

Conversions automatiques pour opérateurs arithmétiques:
`short`, **`char`** et **`bool`** → **`int`**

Le type *faible* → type *fort.* *Critère = domaine couvert le plus grand*
`int → long → long long → float → double → long double`

Booléen : opérateurs de comparaison, conversions

Un booléen ne peut prendre que 2 valeurs: `true` ou `false`

not ! ++ --	* / %	+ -	< <= > >=	== !=	and &&	or ||	? :

On dispose **d'opérateurs de comparaison** et **d'opérateur logique** pour construire une expression conditionnelle

Conversion entre les types de données et le type booléen :

- Dans une expression logique:
  - Motif binaire nul (0 , 0.0, …) → booléen `false`
  - Tous les autres motifs binaires → booléen `true`

```
int p(0);
cin >> p;
if(p) …   ⇔ if(p != 0) …
```

- Dans une expression arithmétique:

`false` → `0` `true` → `1`

Instruction conditionnelle / instruction contrôlée

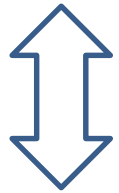
```
if( condition )
    une seule instruction contrôlée ;
else // optionnel
    une seule instruction contrôlée ;
```

1) Utiliser un bloc **{…} pour contrôler plus d'une instruction**
   - l'indentation ne suffit PAS !!
   - Si on dispose d'espace, toujours utiliser un bloc même avec une seule instruction contrôlée

2) **Ne PAS ajouter de point virgule en fin de ligne du if ou du else**
   le point virgule représente l'instruction nulle qui ne fait rien !!

```
if( n == 33 );
    cout << " n vaut 33 !" << endl;
```

```
if( n == 33 );
    cout << " n vaut 33 !" << endl;
```



```
if( n == 33 )
     ;  // l'instruction nulle est contrôlée !!

cout << " n vaut 33 !" << endl; // toujours exécutée
```

Instruction conditionnelle / instruction contrôlée (suite)

3) **Le piège classique**: utiliser l'opérateur d'affectation
au lieu de l'opérateur de d'égalité

```
if( n = 0 )
      cout << " n est nul " << endl;
else
      cout << " n n'est pas nul " << endl;
```

Opérateurs logiques / évaluation paresseuse

```
bool a, b;
```

a and b	b: false	b: true
a: false	false	false
a: true	false	true

ET-LOGIQUE

and &&

`false and b = false`

`true and b = b`

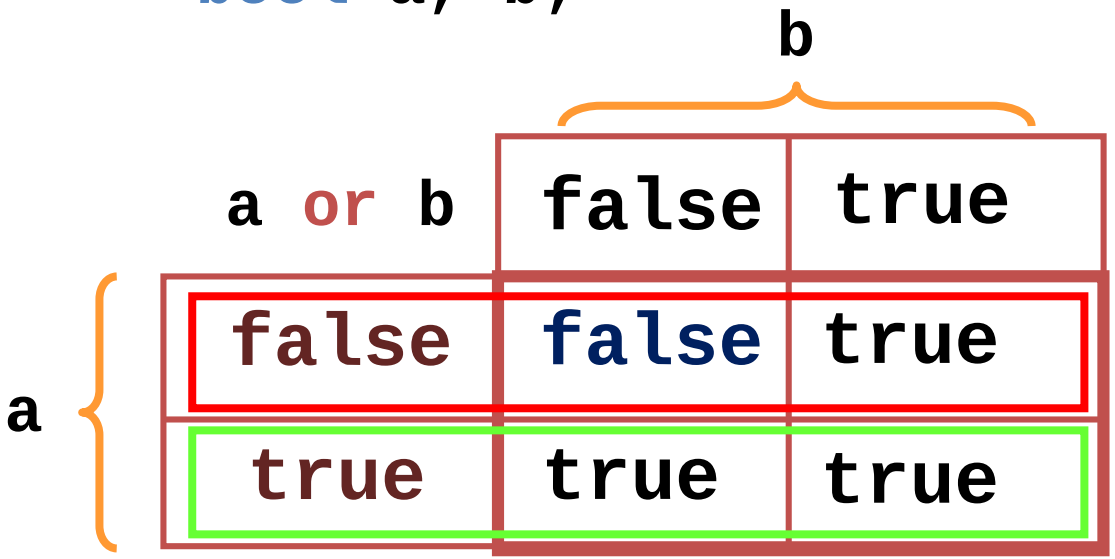
Evaluation paresseuse du ET logique:

L'opérande **droit** n'est PAS évalué si celui de **gauche** vaut **false**

En effet c'est inutile car on connaît déjà le résultat

Opérateurs logiques / évaluation paresseuse (suite)

```
bool a, b;
```



OU-LOGIQUE

or ||

`false or b =` **b**

`true or b =` **true**

Evaluation paresseuse du OU logique:

L'opérande **droit** n'est PAS évalué si celui de **gauche** vaut **true**

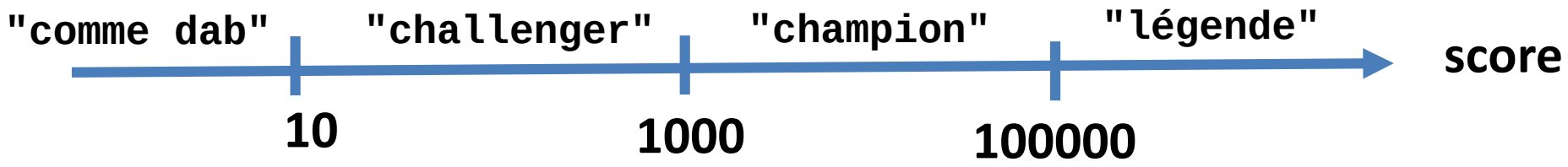
En effet c'est inutile car on connaît déjà le résultat

Opérateurs logiques / évaluation paresseuse (suite)

Lois de de Morgan : relation entre not, and, or

```
not(A or  B) <-> (not A) and (not B)
not(A and B) <-> (not A) or  (not B)
```

Cascade de if-else



Switch sert à tester des valeurs entières (dont char)

```
1   int numero_mois(0);
2   cin >> numero_mois;
3
4   switch (numero_mois)
5   {
6   case 1:
7   case 3:
8   case 5:
9   case 7:
10  case 8:
11  case 10:
12  case 12: cout << "ce mois a 31 jours"          << endl; break;
13  case 4:
14  case 6:
15  case 9:
16  case 11: cout << "ce mois a 30 jours"          << endl; break;
17  case 2:  cout << "février a 28 ou 29 jours"    << endl; break;
19  default: cout << "Numéro de mois incorrect !« << endl; break;
20 }
```

Opérateur conditionnel ternaire (faible priorité)

`condition ? opérande2 : opérande3`

Si **`condition`** (**`opérande1`**) vaut **`true`**

Alors la valeur de l'expression est **`opérande2`**

Sinon la valeur de l'expression est **`opérande3`**

```
int max(0), a(77), b(88) ;
max = (a > b) ? a : b ;
```