Chapitre 5

Opérateurs, expressions et conversions

1. Expressions

- Une expression est la composition d'opérateurs, de variables, de constantes, de fonctions et de parenthèses.
- Une expression retourne toujours un résultat, sa valeur.
- La valeur de l'expression a un type et, elle peut-être utilisée dans une autre expression

Dans cet exemple :

```
int x = 3 + sin(z) ;
```

```
Variable : Z

Constante : 3

Fonction : sin

Valeur de retour : x , type « int »
```

2. Opérateurs

- Le langage C++ est riche en opérateurs.

Priorité et associativité des opérateurs	
Opérateur	**Associativité**
:: (portée globale) :: (portée classe)	**Gauche – Droite**
() [] -> . (postfixe)++ (postfixe)-- sizeof typedef	**Gauche – Droite**
++(préfixe) --(prefixe) ! ~ &(adresse) +(unaire) -(unaire) *(indirection) delete new casts	***Droite - Gauche***
.* ->*	**Gauche – Droite**
* / %	**Gauche – Droite**
+ -	**Gauche – Droite**
<< >>	**Gauche – Droite**
< <= > >=	**Gauche – Droite**
== !=	**Gauche – Droite**
&	**Gauche – Droite**
^	**Gauche – Droite**
|	**Gauche – Droite**
&&	**Gauche – Droite**
||	**Gauche – Droite**
?:	***Droite – Gauche***
= += -= *= /= %= >>= <<= &= ^= |=	***Droite – Gauche***
, (opérateur virgule)	**Gauche – Droite**

2.1 Opérateurs arithmétiques

Expression arithmétique (par ordre de priorité)	Commentaire
-i +j	moins unaire, plus unaire
i*j i/j i%10	multiplication division modulo
i+j i-j	addition soustraction
i = 3/2.0	a prend la valeur 1.5
i = 3/2	a prend la valeur 1

- Le symbole « % » représente l'opérateur modulo.
- Il fournit le reste de la division entière de son premier opérande par son second.
- Remarque : la division entière n'est réalisée que sur des nombres entiers. En Java, elle peut être utilisée aussi sur des réels (float).

```
11%4 vaut 3
12%4 vaut 0
11%2 vaut 1
```

2.2 Opérateurs de manipulation de bits

Manipulation de bits	Commentaire
`~i` `i=0x5F ~i=10100000⇔ 160`	Négation bit à bit. Il inverse un à un tous les bits de son unique opérande

`i << n`
`i = -100`
`i << 2 ⇔ i = -400`
	Décalage à gauche de n rangs. Les bits sortants sont perdus. Ils sont remplacés par des 0. Si la variable est signée, le bit de signe est conservé. Multiplication par 2.

`i >> n`
`i = -100`
`i >> 2 ⇔ i = -25`
	Décalage à droite de n rangs. Les bits sortants sont perdus. Ils sont remplacés par des 0. Si la variable est signée, le bit de signe est conservé. Division par 2.

`&`	ET bit à bit entre les valeurs de 2 expressions
`|`	OU bit à bit entre les valeurs de 2 expressions
`^`	OU exclusif bit à bit entre les valeurs de 2 expressions

Op1	Op2	Op1&Op2	Op1|Op2	Op1^Op2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

2.3 Opérateurs d'incrément et de décrément

- Les opérateurs d'incrémentation ou décrémentation sont placés soit avant (pré) soit après (post) la variable.

Incrément et décrément	Expression équivalente
`j = ++i ;`	**D'abord** `i = i+1 ;` **puis** `j = i ;`
`j = i++ ;`	**D'abord** `j = i ;` **puis** `i = i+1 ;`
`j = --i ;`	**D'abord** `i = i-1 ;` **puis** `j = i ;`
`j = i-- ;`	**D'abord** `j = i ;` **puis** `i = i-1 ;`

- Pré : L'incrémentation/décrémentation est effectuée puis la variable est utilisée.
- Post : L'utilisation de la variable est effectuée avant l'incrémentation/décrémentation.

2.4 Opérateurs d'affectation

- L'élément à gauche du « = » se voit affecter la valeur retournée par l'expression de droite.
- Les conversions éventuelles sont prises en considération lors de l'affectation.

```
a = b = 0 ;
a = b + (c=3) ; ⇔ c=3 ; a = b + c ;
a op= b ; ⇔ a = a op b ; // Affectation combinée
a += 3 ; a = a + 3
op : * / % + - << >> & ^ |
```


2.5 Opérateurs relationnels et booléens

- Le résultat de la comparaison de deux expressions vaut :
  - false (0) si le résultat de la comparaison est FAUX.
  - true (1) si le résultat de la comparaison est VRAI.

Type	Opération	Symbole
Relationnel	Inférieur que	<
	Supérieur que	>
	Inférieur que ou égal	<=
	Supérieur que ou égal	>=
Égalité	Égal	==
	Différent	!=
Logique	Négation (unaire)	!
	ET booléen	&&
	OU booléen	||

- Une expression est vraie si elle est non nulle.
- Une expression est fausse si elle est égale à zéro.

Déclaration et initialisation		
`int a = -5, b=3, c=0 ;`		
Expression	**Équivalence**	**Valeur**
`a + 5 && b`	`((a + 5) && b)`	**false ou 0**
`!(a < b) && c`	`(( !(a < b)) && c)`	**false ou 0**
`1 || (a != 7)`	`(1 || (a != 7))`	**true ou 1**

```
(( !(a < b)) && c)
(a<b) ⇔ (-5<3) ? => vrai
!(vrai) => faux
c = 0 ⇔ c a une valeur nulle => faux
(( !(vrai)) && faux) => faux (false)
```

Op1	Op2	Op1&&Op2	Op1||Op2	!Op1
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

2.6 Opérateur de séquence

- L'opérateur de séquence est représenté par le symbole « , ».
- Il est le moins prioritaire dans la table des opérateurs.
- Il permet d'évaluer les différentes opérations dans l'ordre.
- Dans une liste d'expressions séparées par « , », chaque expression est évaluée en premier avant de passer à la suivante qui se trouve à sa droite.

```
a = (b=10 , b+20) ;
b=10 ; a=b+20=10+20=30 ;

x = 10, i=2, z=3 ;
```

2.7 Opérateur conditionnel

- Il est appelé aussi opérateur ternaire.
- Il prend comme opérandes 3 expressions.
- La syntaxe de l'opérateur conditionnel est comme suit :

```
(expression1) ? expression2 : expression3 ;
```

- « expression1 » est évaluée en premier.
- Si « expression1 » est vraie alors :
  - « expression2 » est évaluée.
  - Le résultat est celui fourni par « expression2 ».
- Si « expression1 » est fausse alors :
  - « expression3 » est évaluée.
  - Le résultat est fourni par « expression3 ».


```
int a = -3;
int n = (a < 0) ? -a : a;
a = -3
est-ce que a < 0 ?
vrai => n = -a = -(-3) = 3;
Dans cet exemple, n est égale à la valeur absolue de a.
```

2.8 Opérateur de taille

- L'opérateur de taille est représenté par le mot clé réservé « sizeof ».
- Il permet de déterminer la taille en octets d'une variable ou d'un type ou d'une expression.
- Cette taille est calculée en fonction de l'architecture interne de la machine.

Déclaration et initialisation	
`int a, b[10] ;`	
Expression	**Valeur gcc sur windows**
`sizeof(a)`	`4` (la taille d'un int)
`sizeof(b)`	`40` (10 x la taille d'un int = 40)
`sizeof(b[2])`	`4` (la taille d'un int)

3. Conversions

- Le langage C++ a deux types de conversion : implicite et explicite.

3.1 Conversions implicites

- Les conversions implicites peuvent avoir lieu dans le calcul d'une expression quand on passe directement un argument à une fonction ou lors du retour d'une valeur par une fonction.
- Ces conversions implicites facilitent la tâche du programmeur.
- Cependant elles risquent d'être potentiellement dangereuses si l'on ne garde pas un œil ouvert.
- En effet, elles peuvent générer des bogues lors de l'exécution d'un programme. Des bogues qui sont parfois difficiles à cerner pour le commun des mortels !

- Les règles de conversion sont comme suit :
    - Si l'opérande est du type « char », « wchar_t », « short », « bool » ou « enum », elle est convertie en « int ».
    - Si l'opérande est un nombre entier tellement grand que l'on ne peut pas représenter par un « int », il sera représenté par un « unsigned int ».
    - Après ces deux premières étapes de conversion, si une expression contient un ensemble de types différents, la hiérarchie à suivre lors de la conversion est comme suit :
        - `« int » < « unsigned » < « long » < « unsigned long » < « float » < « double » < « long double »`.
        - L'opérande, ayant un type moins élevé dans la hiérarchie, est promue à un type plus élevé et l'expression récupère ce type.

3.2 Conversions explicites

- On peut demander explicitement une conversion d'un opérande dans un type désiré.
- Cette opération s'appelle « casting » ou « transtypage ».
- Les règles de forçage de conversion de type du langage C peuvent être utilisées aussi en C++.

Casts	Commentaires
`x = float(y) ;`	`Notation C++`
`x = (float) y ;`	`Notation C, ok aussi en C++`

- 4 nouveaux opérateurs ont été introduits en C++, pour forcer la conversion de type.

Opérateurs	Commentaires
`static_cast`	Cet opérateur est utilisé pour effectuer les opérations de conversion standard, par exemple, de `int` en `float`, un `float` en `char` etc.
`const_cast`	Cet opérateur permet uniquement la conversion de types `const` vers non `const`.
`reinterpret_cast`	Cet opérateur est utilisé essentiellement pour la conversion de types de relations différentes (non standard).
`dynamic_cast`	Cet opérateur est utilisé en programmation polymorphique où la conversion est différée au moment de l'exécution du programme.

Opérations	Commentaires
`double x = 5.89;`	
`int y = static_cast <int>(x);`	ANSI C++ : double vers int.
`double z = static_cast <double>(y);`	ANSI C++ : int vers double.
`int i = 7 ;`	
`float w = (float) i ;`	C
`w = float(i) ;`	C++
`w = static_cast <float>(i) ;`	ANSI C++
`char a = static_cast<char> ('B'+5.0) ;`	ANSI C++

Les opérateurs « const_cast », « reinterpret_cast » et « dynamic_cast » seront étudiés plus tard dans le cours.