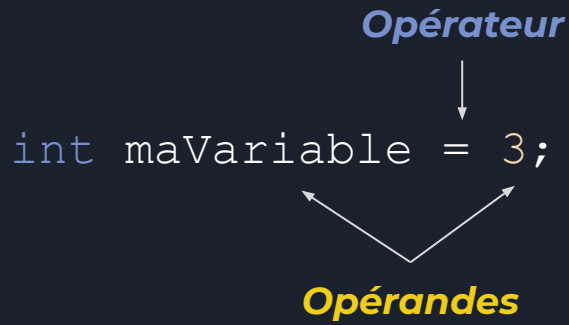
Opérateurs

Opérateur

Un **opérateur** est un symbole permettant d'effectuer une **opération** sur un ou plusieurs **opérandes** (variables et valeurs).



Evaluation de variable

Lorsqu'une variable est utilisée dans **une expression**, elle est **évaluée**. Autrement dit :

Elle est substituée par sa valeur stocké en mémoire.

```
int a = 3;
int b = 6;
std::cout << (a + b + 1);
\\ Evalué en std::cout << (3 + 6 + 1);
\\ Evalué en std::cout << 10;
\\ Affiche "10"
```

Opération d'affectation

L’**opérateur d'affectation "="** permet d'affecter une valeur à une variable (de droite à gauche).

L’ancienne valeur de la variable est écrasée par la nouvelle.

```
int a = 3;
// a vaut 3
int b = 6;
// b vaut 6
b = a;
// b vaut 3
```

Opérateurs arithmétiques

Les **opérateurs arithmétiques** effectuent des calculs arithmétiques entre deux opérandes.

Opérateur	Description	Exemple
+	Addition	a+b
-	Soustraction	a-b
*	Multiplication	a*b
/	Division	a/b
%	Modulo	a%b

Le **modulo %** est le reste de la division entière.

`7 % 3` vaut `1` (7 par 3 = 2, il me reste 1)

Opérateurs arithmétiques

Lorsque l'opérateur de division est utilisé sur deux opérandes de type entier, la **division est entière**.

Sinon, la **division est décimale**.

`(7 / 3)` vaut `2`

`(7.0f / 3)` vaut `3.5f`

Affectations composées

Les **affectations composées** sont des raccourcis pour écrire une opération arithmétique suivie d'une assignation sur un même opérande.

Opérateur	Exemple	Equivalence
+=	a+=b	a = a + b
-=	a-=b	a = a - b
*=	a*=b	a = a * b
/=	a/=b	a = a / b
%=	a%=b	a = a % b

Incrémentation

Incrémentation et décrémentation : Augmente ou diminue la valeur d'une variable de 1.

Opérateur	Description	Exemple
++n	Pré-incrémentation	++a
--n	Pré-décrémentation	--a
n++	Post-incrémentation	a++
n--	Post-incrémentation	a--

Incrémentation

Les **post-incrémentations** et **post-décrémentations** sont appliqués **après avoir évalué** la valeur de la variable sur la ligne d'instruction.

```
int i = 8;
int j = ++i; // pre-incremment => i = i + 1; int j = 9;
// i et j valent 9

int x = 8;
int y = x++; // post-incremment => int y = 8; x = x + 1;
// x vaut 9 et y vaut 8
```

Opérateurs relationnels

Les **opérateurs relationnels comparent deux valeurs** de même type et résultent par une valeur booléenne, “true” ou “false” :

Opérateur	Description	Exemple
==	Strictement égal	a == b
!=	Différent	a != b
>	Supérieur	a > b
<	Inférieur	a < b
>=	Supérieur ou égal	a >= b
<=	Inférieur ou égal	a <= b

Opérateurs logiques

Les **opérateurs logiques** s'appliquent entre booléens.

Opérateur	Description	Exemple
&&	ET : a et b sont vrais ?	true && true => true true && false => false false && true => false false && false => false
||	OU : a ou b est vrai ?	true || true => true true || false => true false || true => true false || false => false
!	NON : Donne l'inverse de a	! true => false ! false => true

Opérateurs bit à bit

Les opérateurs “bit à bit” s'appliquent sur chaques bits des opérandes de même type.

Opérateur	Description	Exemple
~	Complément	`0010100 => 1101011`

&	Bit AND : bit à 1 dans les deux opérandes	`  0010100`
		`& 0010010`
		`=> 0010000`

|	Bit OR : bit à 1 dans au moins un des deux opérandes	`  0010100`
		`| 0010010`
		`=> 0010110`

^	Bit XOR : bit à 1 dans un seul des deux opérandes	`  0010100`
		`^ 0010010`
		`=> 0000110`

Opérateurs de décalage de bits

Les **décalages de bits** sur valeurs entières sont équivalents à des multiplications et divisions par une puissance de 2.

Opérateur	Description	Exemple
<<	Décalage de n bits vers la gauche. Multiplie par 2^n.	`0010100 << 1 => 0101000`
>>	Décalage de n bits vers la droite. Divise par 2^n.	`0010100 >> 2 => 0000101`

Priorités

Chaque opérateur a son niveau de **priorité**.

Si deux opérateurs ont le même niveau de priorité, ils sont lus de gauche à droite.

Il est possible de forcer la priorité entre opérateurs à l'aide de **parenthèses** :

```
true || true && false => false
true || (true && false) => true
```

Evaluer une expression conditionnelle

`(... && ...)`
“Est-ce que ... est vrai **et** ... est vrai ?”

`(... || ...)`
“Est-ce que ... est vrai **ou** ... est vrai ?”

```
(x == 5 && y > 2)
```
“Est-ce que x est égal à 5 et y est supérieur à 2 ?”

Réponses :

- Oui, c’est vrai !
- Non, c’est faux !