# 6

# Erreurs couramment commises par les débutants

**Opérateurs de comparaison**

Une condition simple compare deux expressions en utilisant les opérateurs suivant :

```
<      inférieur à
>      supérieur à
<=     inférieur ou égal à
>=     supérieur ou égal à
==     égal à
!=     différent de
```

L'opérateur pour tester si deux valeurs sont égales s'écrit avec deux signes égal (`==`). un seul signe égal (`=`) représente l'affectation. Par exemple, si on veut tester si une variable `a` est égale à 7, il faut écrire :

`if (a == 7){ ... }` et non pas `if (a = 7){ ...}`

Cette dernière écriture est correcte et considérée comme licite par les compilateurs. Voyons pourquoi?

En fait, chaque expression *fait* quelque chose et *vaut* quelque chose (retourne une valeur).

**Exemple :** Soit `a` une variable entière : `int a = 3;`
L'expression : `a + 5`

- fait : addition
- vaut (retourne comme valeur) : `8`
  `a + 5 = 3 + 5 = 8`

L'expression : `a * 3`

- fait : multiplication
- vaut (retourne comme valeur) : `9`
  `a * 3 = 3 * 3 = 9`

L'expression : `a > 1`

- fait : comparaison
- vaut (retourne comme valeur) : `true`
  `a > 1, i.e., 3 > 1`
  Une condition est une expression qui retourne deux valeurs: `true` ou `false`. Une condition vaut `true` si elle est vraie, et vaut `false` quand elle est fausse.

L'expression : `a = 7`

- fait : affectation
- vaut (retourne comme valeur) : `7`
  Elle retourne la valeur affectée à la variable `a`.

En fait, on peut écrire : `int a, b; b = a = 7;`

```
int main(){
  int a, b;
  b = a = 7;

  cout << "a = " << a << endl;
  cout << "b = " << b << endl;
}
```

L'instruction `b = a = 7;` peut être écrite de cette façon : `b = (a = 7);`

Ce qui veut dire qu'on exécute tout d'abord l'instruction `a = 7`, puis on affecte la valeur retournée par cette expression (c'est-à-dire `7`) à la variable `b`.

- ⇨ Cette écriture `(b = a = 7)` est licite mais déconseillée.

Revenons à ce que nous disions : pour voir si une variable `a` est égale à une valeur (`7` par exemple), alors on doit écrire :

`if (a == 7){ ... }` et non pas `if (a = 7){ ... }`

Parce que comme nous avions vu, l'expression `a = 7` retourne comme valeur `7` (l'expression retourne la valeur affectée à la variable). Donc, écrire :

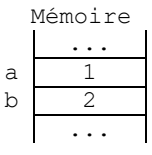
`if (a = 7){ ... }` revient à écrire :

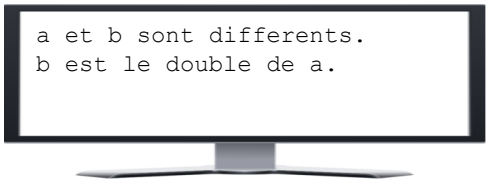
```
a = 7;
if (7){ ... }
```

En réalité, toute valeur nulle (0) est considérée comme `false` et toute autre valeur positive ou négative est considérée comme `true`. `7` est une valeur non nulle alors les instructions qui se trouvent dans le bloc '`if`' seront toujours exécutées.

**Exercice :** Qu'affiche le programme C++ suivant?

```
int main(){
       int a = 1;
       int b = 2;

       if  (a == b) { cout << "a et b sont egaux." << endl; }
       else { cout << "a et b sont differents." << endl; }

       if (2 * a == b){ cout << "b est le double de a." << endl; }
}
```



Pour rendre le programme beaucoup plus lisible, on écrit l'expression `(2 * a)` entre parenthèses:

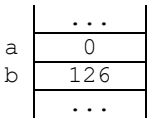
```
if ((2 * a) == b){ cout << "b est le double de a." << endl; }
```

**Exercice:** Voici un petit programme :

```
int main(){
	int a = 0;
	int b = 126;

	if (a = b) {
		cout << "a = " << a << endl;
		cout << "b = " << b << endl;
		cout << "a et b sont egaux." << endl;
	}
	else {
		cout << "a = " << a << endl;
		cout << "b = " << b << endl;
		cout << "a et b sont differents." << endl;
	}
}
```

- Que renvoie (affiche) ce programme?
- Modifier le programme pour qu'il renvoie le résultat voulu.

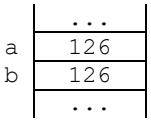
Au début, on a :

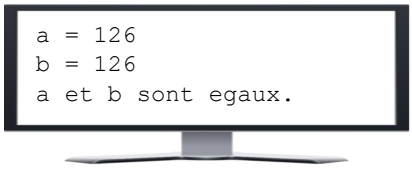



Après exécution de :

```
	if (a = b) {
		cout << "a = " << a << endl;
		cout << "b = " << b << endl;
		cout << "a et b sont egaux." << endl;
	}
	else {
		cout << "a = " << a << endl;
		cout << "b = " << b << endl;
		cout << "a et b sont differents." << endl;
	}
```

On aura :



Pourquoi? Parce qu'au lieu d'écrire `if(a == b)`, on a écrit : `if (a = b) { ... }`.
Que s'est-il passé? Rappelez-vous, l'expression `a = b` :

- fait : affectation,
  elle affecte la valeur de `b (126)` à la variable `a`
- vaut (retourne comme valeur) : `126`
  retourne la valeur affectée à la variable `a`

Donc, écrire `if (a = b){ ... } else{ ... }` reviens à écrire :

```
a = b;
if (b){ ... } else { ... }
```

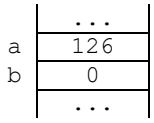
Et comme `b` est non nulle alors la partie '`if`' sera exécutée.

**Exercice :** Voici un autre programme :

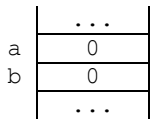
```
int main(){
	int a = 126;
	int b = 0;

	if (a = b) {
		cout << "a = " << a << endl;
		cout << "b = " << b << endl;
		cout << "a et b sont egaux." << endl;
	}
	else {
		cout << "a = " << a << endl;
		cout << "b = " << b << endl;
		cout << "a et b sont differents." << endl;
	}
}
```

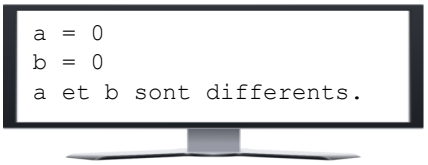
- Que renvoie (affiche) ce programme?

Au début, on a :




Après exécution du bloc : `if (a = b) { ... } else { ... }`, on aura :




L'expression `a = b` :

- fait : affectation,
  elle affecte la valeur de `b` (i.e., `0`) à la variable `a`
- vaut (retourne comme valeur) : `0`
  retourne la valeur affectée à la variable `a`

Donc, écrire `if (a = b){ ... } else { ... }`, reviens à écrire :

```
a = b;
if  (b){ ... } else { ... }
```

Et comme `b` est nulle `(b == 0)` alors la partie `'else'` sera exécutée.