

Expressions

Expressions et Opérateurs

A droite du signe égal dans une affectation se trouve une **expression**:

```
nom_de_variable = expression;
```

Une expression calcule une valeur, qui doit être de même type que la variable.

Une expression peut être simplement une valeur littérale:

```
4  
3.14
```

ou une formule qui met en oeuvre des opérateurs:

```
n * n  
n * (n + 1) + 3 * n - 2
```

Les valeurs littérales et leurs types

- `1` est de type `int`;
- `1.0` est de type `double`;
- `1.` est équivalent à `1.0`, et donc de type `double`. On peut écrire:
`double x(1.);`
au lieu de
`double x(1.0);`

Il vaut mieux écrire `1.0` plutôt que `1.` puisque c'est plus lisible
- On peut utiliser la notation scientifique, par exemple écrire `2e3` pour 2×10^3 , c'est-à-dire 2000.
De façon générale: `aeb` vaut $a \times 10^b$. Par exemple:
`double x(1.3e3);`
→ `x` vaut $1.3 \times 10^3 = 1.3 \times 1000 = 1300$
`double y(1.3e-3);`
→ `y` vaut $1.3 \times 10^{-3} = 1.3 \times 0.001 = 0.0013$

Opérateurs

On dispose des 4 opérateurs usuels:

+ pour l'addition;

- pour la soustraction;

* pour la multiplication;

/ pour la division.

Attention: si la division se fait entre `int`, il s'agit de la division entière.

Par exemple:

`1 / 2` vaut 0

`5 / 2` vaut 2

mais

`1 / 2.0` vaut bien 0.5

On dispose aussi des opérateurs `+=`, `-=`, `*=`, `/=`

Par exemple:

```
a += 5;  
est équivalent à  
a = a + 5;
```

```
b *= a;  
est équivalent à  
b = b * a;
```

Opérateurs relatifs au type `int`

Dans le cas des `int`, il existe aussi:

- un opérateur modulo, noté `%`, qui renvoie le reste de la division entière:
`11 % 4` vaut 3
(la division de 11 par 4 a pour reste 3 car $11 = 2 * 4 + 3$).

```
0 % 4 vaut 0 car 0 = 0 * 4 + 0  
1 % 4 vaut 1 car 1 = 0 * 4 + 1  
2 % 4 vaut 2 car 2 = 0 * 4 + 2  
3 % 4 vaut 3 car 3 = 0 * 4 + 3  
4 % 4 vaut 0 car 4 = 1 * 4 + 0  
5 % 4 vaut 1 car 5 = 1 * 4 + 1  
...
```

Opérateurs `++` et `--`

Il existe aussi:

- deux opérateurs notés `++` et `--`, qui permettent respectivement d'incrémenter et de décrémenter, c'est-à-dire d'ajouter et de soustraire 1 à une variable.

Par exemple, l'instruction:

```
++i;  
est équivalente à :  
i = i + 1;
```

Ces deux opérateurs sont souvent utilisés avec l'instruction `for`, que nous verrons par la suite.

Affectation d'une valeur décimale à une variable entière

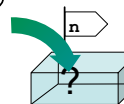
Quand on affecte une valeur décimale à une variable de type `int`, la partie fractionnaire est perdue.

Exemple:

```
double x(1.5);  
int n;
```

```
n = 3 * x;
```

4.5



Le type de la valeur 4.5 est converti de `double` vers `int`.

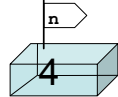
Affectation d'une valeur décimale à une variable entière

Quand on affecte une valeur décimale à une variable de type `int`, la partie fractionnaire est perdue.

Exemple:

```
double x(1.5);  
int n;
```

```
n = 3 * x;  
4.5
```



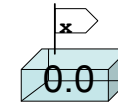
Le type de la valeur 4.5 est converti de `double` vers `int`.

La division entière

```
double x;
```

```
x = 1 / 2;
```

l'expression `1 / 2` est d'abord évaluée
elle vaut 0
la valeur 0 est affectée à `x`



La division entière

Le problème peut se poser par exemple quand on calcule la moyenne de deux valeurs entières:

```
int note1(4), note2(5);
```

```
double moyenne((note1 + note2) / 2);
```

Dans ce cas, `moyenne` vaut 4 au lieu de 4.5

La division entière

Une solution possible:

```
int note1(4), note2(5);
```

```
double moyenne(note1 + note2);  
moyenne /= 2;
```

Fonctions mathématiques

Fonctions mathématiques

La bibliothèque standard du C++ fournit les fonctions mathématiques usuelles.
Par exemple:

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double angle;
    double s;

    angle = 10 * 3.14159 / 180;
    s = sin(angle);
```

il faut ajouter cette ligne pour pouvoir utiliser les fonction mathématiques

→ `sin(angle)` calcule le sinus de la valeur contenue dans la variable `angle`, en radians.

Fonctions mathématiques

- `sin` les fonctions trigonométriques fonctionnent en radians
- `cos`
- `tan`
- `asin` sinus inverse ou arc sinus
- `acos`
- `atan`
- `atan2` `atan2(y, x)` fournit la valeur de l'arc-tangente de y / x
- `sinh` sinus hyperbolique ou *sh*
- `cosh`
- `tanh`
- `exp`
- `log` logarithme népérien ou *ln*
- `log10` logarithme à base 10 ou *log*
- `pow` `pow(x, y)` fournit la valeur de x^y
- `sqrt` racine carrée
- `ceil` `ceil(x)` renvoie le plus petit entier qui ne soit pas inférieur à x : `ceil(2.6) = 3`
- `floor` `floor(x)` renvoie le plus grand entier qui ne soit pas supérieur à x : `floor(2.6) = 2`
- `abs` valeur absolue

Constantes mathématiques

Bien que non standard, les constantes suivantes sont souvent définies par les compilateurs usuels:

- `M_PI` π
- `M_E` e (= 2.71828, base des logarithmes naturels);
- ...

```
#include <iostream>
#include <cmath>
using namespace std;
```

$$\text{angle en radians} = \frac{\pi \text{ angle en degrés}}{180}$$

```
int main()
{
    double angle_en_degres;

    cout << "Entrez un angle en degres: " << endl;
    cin >> angle_en_degres;

    double angle_en_radians(M_PI * angle_en_degres / 180);

    cout << "Sa valeur en radians vaut " << angle_en_radians << endl;
    cout << "Son cosinus vaut " << cos(angle_en_radians) << endl;

    return 0;
}
```