IN100 : Cours 2 – Les Opérateurs et les Conditionnelles

Sandrine Vial
sandrine.vial@uvsq.fr

Septembre 2018

Affectation

Opération d'affectation $=$.

- Evaluation du membre droit
- Modification du membre gauche.

Exemples

```
int j;
int p;

j = 2;
p = j;
p = 3;
j = p;
```

Opérateurs Arithmétiques

Opérateurs

- Addition : $+$
- Soustraction : $-$
- Division : $/$
- Multiplication : $*$

Propriétés

- Arguments sont des opérateurs de même type
- Résultat est du même type que les opérandes

Opérateurs Arithmétiques

Exemple

```
int a;
int b;
int c;

    b = 4;
    a = 3 + 1; /* a prend la valeur 4 */
    a = b + 3; /* a prend la valeur 7 */
    b = 2 - a; /* b prend la valeur -5 */
    c = a * b; /* c prend la valeur -35 */
```

commentaire

Et la division ?

La division est aussi entière.

- 4 / 2 vaut 2.
- 5 / 2 vaut 2.

Opérateur modulo : %

- Ne porte que sur des opérandes entières positives.
- Permet de calculer le reste de la division entière.
- 5 / 2 vaut 2 et 5 % 2 vaut 1.
- 11 % 3 vaut 2.

c'est le reste

Permet (entre autre) de tester la parité d'un entier.

Priorité des opérateurs

$$a + b * c$$

- Valeur de cette expression dépend de la priorité des opérateurs.
- Le parenthèsage des expressions permet de forcer l'ordre des opérations.

$$(a + b) * c$$

Du plus prioritaire au moins prioritaire

opérateur	Symbole	Arité
appel de fonction	()	
signes	+ -	1
multiplication, division, modulo	* / %	2
addition, soustraction	+ -	2
affectation	=	2

Priorité des opérateurs

Exemples

- **a + b * c** équivaut à a + (b * c)
- **a * b + c % d** équivaut à (a * b) + (c % d)
- **-c % d** équivaut à (-c) % d
- **-a + c % d** équivaut à (-a) + (c % d)
- **-a / -b + c** équivaut à ((-a) / (-b)) + c

Et l'associativité ?

Que vaut $a/b/c$?

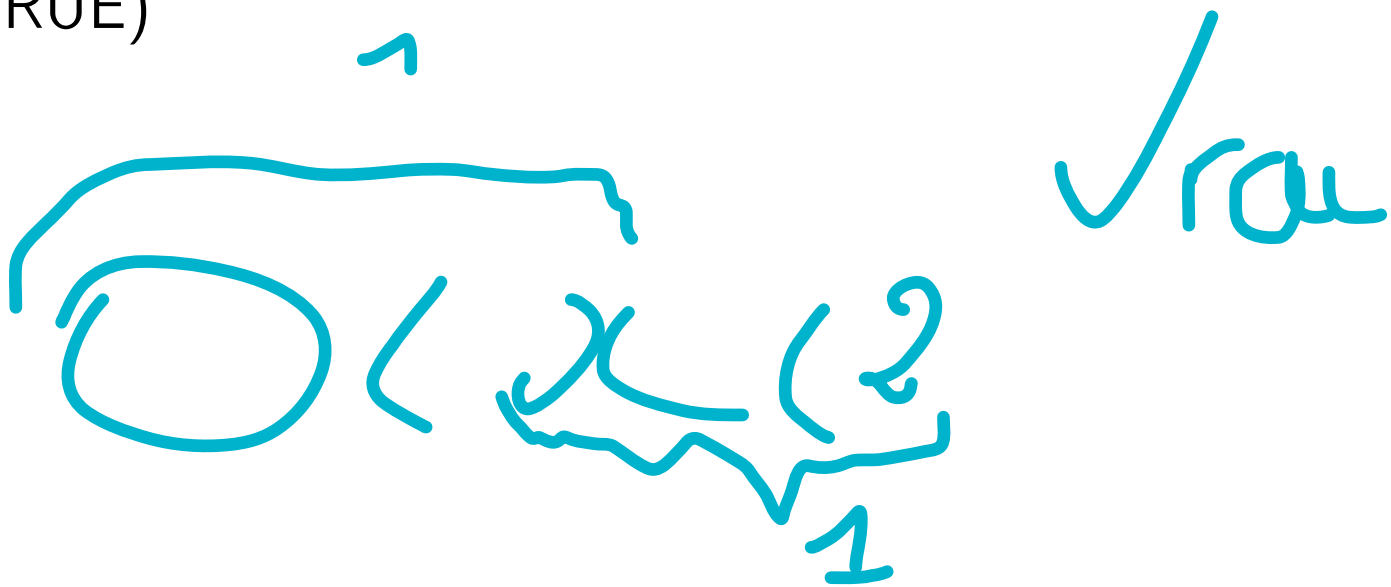
$(a/b)/c$ ou $a/(b/c)$?

Du plus prioritaire au moins prioritaire

opérateur	Symbole	Arité	Associativité
appel de fonction	()		
signes	+ -	1	$D \Rightarrow G$
multiplication, division, modulo	* / %	2	$G \Rightarrow D$
addition, soustraction	+ -	2	$G \Rightarrow D$
affectation	=	2	$D \Rightarrow G$

VRAI et FAUX

- Le VRAI et FAUX sont représentés par des entiers
- La valeur 0 représente la valeur FAUX (FALSE)
- Toutes les valeurs $\neq 0$ représentent la valeur VRAI (TRUE)



Opérateurs de comparaison

- Permettent de comparer des variables de mêmes types.
- Chaque expression renvoie une valeur.

Opérateurs

- $<$ $<=$ inférieur, inférieur ou égal
- $>$ $>=$ supérieur, supérieur ou égal
- $==$ égalité
- $!=$ différent

Et si on selectionnait ?

- Toutes les lignes de code sont exécutées séquentiellement.
- Embranchement : Permet de choisir les lignes de codes à éxécuter en fonction d'une condition.

`if` simple

- Si une condition est remplie exécuter des instructions
- `if (condition) { un bloc d'instructions }`
- `condition` doit être vraie pour que le bloc d'instructions soit exécuté.

Exemple 1

```
#include "uvsqgraphics.h"
int main()
{
     POINT p;

     init_graphics(900,600);

     p = wait_clic();
     if (p.x < 450)
     {
          draw_fill_circle(p,50,rouge);
     }

     wait_escape();
     exit(1);
}
```

`if ... else`

- Si une condition est remplie exécuter des instructions sinon exécuter d'autres instructions
- `if (condition) { un bloc d'instructions 1 } else { un bloc d'instructions 2 }`
- `condition` doit être vraie pour que le bloc d'instructions 1 soit exécuté et fausse pour que le bloc d'instructions 2 soit exécuté.

Exemple 2

```
#include "uvsqgraphics.h"
int main()
{
     POINT p;

     init_graphics(900,600);

     p = wait_clic();
     if (p.x < 450)
     {
          draw_fill_circle(p,50,rouge);
     }
     else
     {
          draw_fill_circle(p,50,vert);
     }
     wait_escape();
     exit(1);
}
```

Exemple 3

```
#include "uvsqgraphics.h"
int main()
{
     POINT p;

     init_graphics(900,600);

     p = wait_clic();
     if (p.x % 2 == 0)
     {
          draw_circle(p,50,rouge);
     }
     else
     {
          draw_circle(p,100,bleu);
     }
     wait_escape();
     exit(1);
}
```

Exemple 4

```
#include "uvsqgraphics.h"
int main()
{
     int i,j;
     int min,max;

     init_graphics(900,600);
     i = lire_entier_clavier();
     j = lire_entier_clavier();
     if (i > j)
     {
         min = j;
         max = i;
     }
     else
     {
         min = i;
         max = j;
     }
     write_text("Le plus grand entier est"); write_int(max); writeln();
     write_text("Le plus petit entier est"); write_int(min); writeln();

     wait_escape();
     exit(1);
}
```

Les Conditions

- Les conditions testées dans le `if` sont booléennes.

```
int i,j;

i = lire_entier_clavier();
j = lire_entier_clavier();

if (i < j)
{
    write_text("i est plus petit que j");
}
```

Les Conditions

- Les conditions testées dans le `if` sont booléennes.

```
int i;

i = lire_entier_clavier();

if (i%2 == 0)
{
    write_text("i est ....");
}
```

Les Conditions

- Les conditions testées dans le `if` sont booléennes.

```
int i;

i = lire_entier_clavier();

if (i%2 == 0)
{
    write_text("i est pair");
}
```

Les Conditions

- Les conditions testées dans le `if` sont booléennes.

```
int i,b;

i = lire_entier_clavier();
b = (i % 2 == 0);

if (b)
{
    write_text("i est pair");
}
```

Egalité à `vrai` n'est pas nécessaire

Tests Simples

- Tous les opérateurs de comparaisons sont utilisables dans une condition
- `if (a < 10) ...`
- `if (b >= 12) ...`
- `if (i == j) ...`

Attention ! Piège !

- L'expression `if (a = valeur_non_nulle) ...` est une expression toujours **vraie**

```
POINT p;

p.x = 100;
p.y = 200;

if (p.x = 200)
  {
    draw_fill_circle(p,50,vert);
  }
```

Attention ! Piège !

- L'expression `if (a = valeur_non_nulle) ...` est une expression toujours **vraie**
- L'expression `if (a = 0) ...` est une expression toujours **fausse**

```
POINT p;

p.x = 0;
p.y = 200;

if (p.x = 0)
  {
    draw_fill_circle(p,50,vert);
  }
```

Conditions plus complexes

- On peut combiner les conditions
  - Les deux conditions sont vraies en même temps : `ET` noté `&&`
  - Une des deux conditions soit vraie : `OU` noté `||`

Conditions plus complexes

- On peut combiner les conditions
  - Les deux conditions sont vraies en même temps : `ET` noté `&&`
  - Une des deux conditions soit vraie : `OU` noté `||`

A && *B*	TRUE	FALSE
TRUE	**TRUE**	*FALSE*
FALSE	*FALSE*	*FALSE*

Conditions plus complexes

- On peut combiner les conditions
  - Les deux conditions sont vraies en même temps : `ET` noté `&&`
  - Une des deux conditions soit vraie : `OU` noté `||`

A && *B*	TRUE	FALSE
TRUE	**TRUE**	*FALSE*
FALSE	*FALSE*	*FALSE*

A || *B*	TRUE	FALSE
TRUE	**TRUE**	**TRUE**
FALSE	**TRUE**	*FALSE*

Exemple 5

```
#include "uvsqgraphics.h"
int main()
{
     POINT p;

     init_graphics(900,600);

     p = wait_clic();
     if (p.x < 450 && p.y > 300)
     {
          draw_fill_circle(p,50,rouge);
     }
     else
     {
          draw_fill_circle(p,50,vert);
     }
     wait_escape();
     exit(1);
}
```

Exemple 6

```
#include "uvsqgraphics.h"
int main()
{
     POINT p;

     init_graphics(900,600);

     p = wait_clic();
     if (p.x < 450 && p.y > 300)
     {
          draw_fill_circle(p,50,rouge);
     }
     else
     {
          if (p.y < 300)
              draw_fill_circle(p,50,vert);
          else
              draw_fill_circle(p,100,bleu);
     }
     wait_escape();
     exit(1);
}
```

Exemple 7

```
#include "uvsqgraphics.h"
int main()
{
     POINT p;

     init_graphics(900,600);

     p = wait_clic();
     if (p.x < 450 || p.y > 300)
     {
          draw_fill_circle(p,50,rouge);
     }
     else
     {
         draw_fill_circle(p,50,vert);
     }
     wait_escape();
     exit(1);
}
```