المعهد العالي للدراسات التكنولوجية في المواصلات بتونس
Institut Supérieur des Etudes Technologiques en Communications de Tunis
Iset'Com

Jihen Thabet

# Module:
## Informatique embarquée: Programmation C embarquée (Partie 2)

2017/2018

# OPERATEURS ET EXPRESSIONS

# Généralités sur les opérateurs



- Une expression est un objet syntaxique obtenu en assemblant des constantes, des variables et des opérateurs.

x+3

- Dans le langage C, il y a bien d'autres opérateurs que les opérateurs arithmétiques qu'on a l'habitude de manipuler. Il y en a en fait plus de 40 opérateurs.

# Opérateurs et priorités



- Nous avons l'habitude de manipuler des expressions arithmétiques :

2+3*4*5-2 2-3-4

- On sait que ces expressions sont équivalentes à (2+(3*(4*5)))-2, (2-3)-4.
- Introduire les parenthèses permet de définir sans ambiguïté l'expression que l'on manipule.
- Pour éviter l'usage des parenthèses qui alourdissent la lecture, il existe des règles pour lever toute ambiguïté :
- Exemple :

2+3*4

- La sous expression 3*4 est évaluée en premier et le résultat obtenu est ajouté à la valeur 2 (forme avec parenthèse : 2 + ( 3 * 4)).

# Opérateurs et priorités



- On dit que l'opérateur * possède une priorité supérieure à la priorité de l'opérateur +.

2-3-4

- la sous expression 2-3 est évaluée en premier et au résultat obtenu, on soustrait la valeur 4 (forme avec parenthèse : (2 - 3) - 4).
- On dit que l'*ordre* (d'évaluation) de l'opérateur - est de *gauche à droite*.

**La donnée d'une *priorité* et d'un *ordre d'évaluation* permet de fixer des règles communes d'évaluation des expressions.**

- Ces priorités et ordre d'évaluation ne permettent évidemment pas de se dispenser complètement des parenthèses :

(2+3)*4 à comparer à 2+3*4 ☺

# Opérateurs et priorités



- Caractéristiques de l'opérateur () :

| Opérateur | Nom | Notation | Priorité | Ordre |
|---|---|---|---|---|
| ( ) | parenthèses | (...) | 15 | gauche-droite |

Système de classification des opérateurs C

# L'opérateur d'affectation



- L'opération la plus importante dans un langage de programmation est celle qui consiste à donner une valeur à une variable.
- Cette opération est désignée par le symbole =.
- Comme l'affectation range une valeur dans une variable (une zone mémoire), il est impératif que le membre gauche d'une affectation représente une zone mémoire (*left value*).

# L'opérateur d'affectation



- Une constante n'est pas une *left value* car elle ne désigne pas l'adresse d'une zone mémoire. Elle ne peut donc pas figurer en membre gauche d'une affectation.
- Le membre droit d'une affectation peut désigner soit une constante soit une zone mémoire soit une expression.

| Opérateur | Nom | Notation | Priorité | Ordre |
|---|---|---|---|---|
| = | Affectation | $x = y$ | 2 | droite-gauche |

# Les opérateurs arithmétiques



| Opérateur | Nom | Notation | Priorité | Ordre |
|---|---|---|---|---|
| + | Addition | *x* + *y* | 12 | gauche-droite |
| - | soustraction | *x* - *y* | 12 | gauche-droite |
| * | multiplication | *x* * *y* | 13 | gauche-droite |
| / | division | *x* / *y* | 13 | gauche-droite |
| % | modulo | *x* % *y* | 13 | gauche-droite |

# Les opérateurs arithmétiques



- Les opérateurs +, -, * fonctionnent comme en arithmétique.
- Par contre, l'opérateur / (division) se comporte de manière différente selon que les opérandes sont des entiers ou des nombres flottants :

✓ nombres flottants : le résultat est un nombre flottant obtenu en divisant les deux nombres.

- nombres entiers : **le résultat est un nombre entier obtenu en calculant la division entière ou division euclidienne.** L'opérateur % n'est défini que pour les entiers et le résultat est le reste de la division entière des opérandes.

# Les opérateurs arithmétiques



- Exemple

```
int a, b, c ;
float x, y, z;
a = b / c;          division entière
a = x / y;          reste entière
x = y / z;          division réelle
x = 2 / 3;          division entière
x = 2.0/3.0;        division réelle
x = (float) ((float) a)/y ; division réelle
```

L'opérateur unaire - a une priorité supérieure aux opérateurs binaires arithmétiques :

| Opérateur | Nom | Notation | Priorité | Ordre |
|---|---|---|---|---|
| − (unaire) | négation | - $x$ | 14 | droite-gauche |

# Les opérateurs de comparaison



| Opérateur | Nom | Notation | Priorité | Ordre |
|---|---|---|---|---|
| == | test d'égalité | x == y | 9 | gauche-droite |
| != | test de non égalité | x != y | 9 | gauche-droite |
| <= | test inférieur ou égal | x <= y | 10 | gauche-droite |
| >= | test supérieur ou égal | x >= y | 10 | gauche-droite |
| < | test inférieur strict | x < y | 10 | gauche-droite |
| > | test supérieur strict | x > y | 10 | gauche-droite |

- Théoriquement, le résultat d'une comparaison est une valeur booléenne (*vrai* ou *faux*).
- En C, le résultat d'une comparaison est 1 ( != 0) ou 0 selon que cette comparaison est *vraie* ou *fausse*.

# Les opérateurs de comparaison



- Il n'existe pas de type booléen en C :

La valeur entière 0 sera considérée comme équivalente à la valeur *faux* et toute valeur différente de 0 équivalente à la valeur *vrai*.

- Piège :

*Ne pas confondre == (test d'égalité) et = (affectation)* ☺

- ✓ Exemple :

```
if (x == 2) {        ... vs        if (x = 2)
```

- ✓ La première teste l'égalité de la valeur contenue dans la variable x et la valeur 2 , alors que la deuxième teste la valeur de l'affectation x=2 qui vaut 2 quelle que soit la valeur de x (dans cet exemple).

# Les opérateurs logiques



- Une variable booléenne est une variable pouvant prendre la valeur *vrai* ou *faux*.
- La valeur d'une expression booléenne est, comme le résultat des comparaisons, une valeur entière.

| Opérateur | Nom | Notation | Priorité | Ordre |
|---|---|---|---|---|
| `&&` | ET | `x && y` | 5 | gauche-droite |
| `\|\|` | OU | `x \|\| y` | 4 | gauche-droite |
| ! (unaire) | NON | `! x` | 14 | droite-gauche |
| | | | | |

# Les opérateurs de manipulation de bits



| Opérateur | Nom | Notation | Priorité | Ordre |
|---|---|---|---|---|
| & | ET bit à bit | x & y | 8 | gauche-droite |
| \| | OU bit à bit | x \| y | 6 | gauche-droite |
| ^ | OU exclusif bit à bit | x ^ y | 7 | gauche-droite |
| ~ (unaire) | NON bit à bit | ~ x | 14 | droite-gauche |
| | décalage à droite | x | 11 | droite-gauche |
| << | décalage à gauche | << x | 11 | droite-gauche |

# Les opérateurs de manipulation de bits



- Les manipulations de bits sont beaucoup utilisées dans l'embarqué. Pour contrôler un périphérique matériel, on retrouve des registres généralement de 8 bits appelés aussi **ports**. Cela permet généralement d'échanger des données avec le monde extérieur (liaison série, port parallèle…) :
- ✓ Des registres de donnée : **Data Register**. Exemple PADR (module PIT). Ce registre contient la donnée reçue ou à émettre.
- ✓ Des registres d'état : **Status Register**. Exemple PASR. Ce registre donne l'état courant du périphérique.
- ✓ Des registres de contrôle : **Control Register**. Exemple PACR. Ce registre permet de contrôler le périphérique.
- Chaque bit d'un registre SR et CR correspond à une fonctionnalité ou état du périphérique.

# Les opérateurs de manipulation de bits



- Exemple :

Validation d'une fonctionnalité. Mise à 1 du bit 7 du registre CR pour autoriser les interruptions par le périphérique :

```
unsigned char *PACR = (unsigned char *) 0xF500001 ;
*PACR = *PACR  | 0x80; soit aussi *PACR  |= 0x80;
```

Inhibition d'une fonctionnalité. Mise à 0 du bit 7 du registre CR pour interdire les interruptions par le périphérique :

```
unsigned char *PACR = (unsigned char *) 0xF500001 ;
*PACR = *PACR & 0x7f; soit aussi *PACR &= 0x7f;
```

# Les opérateurs de manipulation de bits



- Il faut aussi noter les différents types d'accès à un registre :
- ✓ Type 1 : registre accessible en lecture/écriture (registre R/W) et l'on peut lire exactement la dernière valeur écrite.
- ✓ Type 2 : registre accessible en lecture/écriture (registre R/W) et l'on ne relit pas la dernière valeur écrite. En cas de manipulation de bits sur ce registre, il est obligatoire de connaître la dernière valeur écrite !
- ✓ Type 3 : registre accessible en lecture seulement (registre Ronly).
- ✓ Type 4 : registre accessible en écriture seulement (registre Wonly). En cas de manipulation de bits sur ce registre, il est obligatoire de connaître la dernière valeur écrite !

# Les opérateurs de manipulation de bits



- **Exemple type 1** :

Validation d'une fonctionnalité. Mise à 1 du bit 7 du registre R/W :

```
unsigned char *PACR = (unsigned char *) 0xF500001 ;
*PACR  |= 0x80;
```

- **Exemple type 2** :

Validation d'une fonctionnalité. Mise à 1 du bit 7 du registre R/W :

```
unsigned char *PACR = (unsigned char *) 0xF500001 ;
unsigned char tmp = 0x0c;
// Initialisation du registre
*PACR = tmp;
…
tmp  |= 0x80;
*PACR = tmp;
```

# Les autres opérateurs binaires d'affectation



- Les opérateurs suivants ne sont que des raccourcis de notation (pas d'optimisation du code généré) :

| Opérateur | équivalent | Notation | Priorité | Ordre |
|---|---|---|---|---|
| += | x = x + y | x += y | 2 | droite-gauche |
| -= | x = x - y | x -= y | 2 | droite-gauche |
| *= | x = x * y | x *= y | 2 | droite-gauche |
| /= | x = x / y | x /= y | 2 | droite-gauche |
| %= | x = x % y | x %= y | 2 | droite-gauche |
| = | x = x y | x = y | 2 | droite-gauche |
| <<= | x = x << y | x <<= y | 2 | droite-gauche |
| &= | x = x & y | x &= y | 2 | droite-gauche |
| \|= | x = x \| y | x \|= y | 2 | droite-gauche |
| ^= | x = x ^ y | x ^= y | 2 | droite-gauche |

# Les autres opérateurs unaires d'affectation



| | | | | |
|---|---|---|---|---|
| ++ | x = x + 1 | x++ ou ++x | 14 | droite-gauche |
| -- | x = x - 1 | x-- ou --x | 14 | droite-gauche |

- y = x++; est le raccourci pour y = x; x = x + 1;
- y = ++x ; est le raccourci pour x = x + 1; y = x;

# Autres opérateurs



- **L'opérateur conditionnel**

Opérateur ternaire (le seul dans C) :

| | | | | |
|---|---|---|---|---|
| ? : | opérateur conditionnel | e ? x : y | 3 | droite-gauche |

Cette expression est une sorte de *si alors sinon* sous forme d'expression : si la condition e est vraie alors cette expression vaut x sinon elle vaut y .

Exemple :

```
a = (v == 2) ? 1 : 2;
```

affecte la variable *a* à la valeur 1 si v vaut 2, sinon affecte la variable *a* à la valeur 2.

# Autres opérateurs



- **L'opérateur séquentiel**

| | | | | |
|---|---|---|---|---|
| , | opérateur séquentiel | `e1, e2` | 1 | droite-gauche |

Cet opérateur permet de regrouper plusieurs expressions en une seule : l'évaluation de l'expression e1, e2 consiste en l'évaluation successives (dans l'ordre) des expressions e1 puis de e2.

# Autres opérateurs



- **L'opérateur de dimension**

Cet opérateur donne l'occupation mémoire (en octets) d'une variable ou d'un type de donné.

| | | | | |
|---|---|---|---|---|
| `sizeof` | opérateur de dimension | `sizeof(e)` | 14 | droite-gauche |

- Exemple :

La valeur de l'expression sizeof(c) est 1 si c est une variable de type char.

L'expression sizeof(char) donne également la valeur 1.

# Autres opérateurs



- **L'opérateur d'adressage**

L'opérateur d'adressage & donne l'adresse d'une variable ou d'une expression.

| | | | | |
|---|---|---|---|---|
| & | opérateur d'adressage | &x | 14 | droite-gauche |

# Autres opérateurs



- **L'opérateur de conversion de type (cast)**

Cet opérateur permet de convertir explicitement le type d'une donnée en un autre type.

Il est à noter que le compilateur C réalise des conversions implicites.

| | | | | |
|---|---|---|---|---|
| `(type)` | opérateur de conversion | `(long int)c` | 14 | droite-gauche |

- Exemple :

```
char i ;
int j ;
j = (int) i ;
```

# Autres opérateurs



- **L'opérateur de parenthèse**

L'opérateur de parenthèse () permet de définir l'ordre d'évaluation d'une expression.

C'est également ce même opérateur qui est utilisé pour encapsuler les paramètres des fonctions.

- **Même lorsqu'une fonction n'a pas d'arguments, ces parenthèses sont obligatoires.**

| | | | | |
|---|---|---|---|---|
| ( ) | parenthèse | ( ) | 15 | gauche-droite |

# Autres opérateurs



- **L'opérateur de sélection**

Les opérateurs . et - servent à sélectionner des champs de données structurées.

| | | | | |
|---|---|---|---|---|
| . | opérateur de sélection | `x.info` | 15 | gauche-droite |
| -> | opérateur de sélection | `x->info` | 15 | gauche-droite |
| [] | opérateur de sélection | `x[3]` | 15 | gauche-droite |

# Conversion de types



- Le problème de la conversion des types se pose lorsqu'une expression est composée de données de nature différentes. Par exemple, quel sens donner à une addition d'un entier et d'un nombre flottant ?
- Le langage de C définit précisément quels sont les types de données compatibles et quel type de conversion est effectuée.

# Les conversions implicites



- Les conversion implicites sont celles faites automatiquement par le compilateur C lors de l'évaluation d'une expression (et donc également d'une affectation).
- **Cas de l'évaluation d'une expression :**

Lorsqu'il est nécessaire de procéder à des conversions, le compilateur C effectue tout d'abord les conversions puis évalue l'expression. Les règles de conversion suivantes s'appliquent :

- ✓ Si l'un des opérandes est de type long double, conversion de l'autre expression en long double; le résultat est un long double.
- ✓ Si l'un des opérandes est de type double, conversion de l'autre expression en double; le résultat est un double.

# Les conversions implicites



- Cas de l'affectation :

Lors d'une affectation, le type de l'expression membre droit est converti (si nécessaire) vers le type de l'expression en membre gauche. Par exemple, les char sont convertis en int.

Cas où l'on affecte une expression plus ``longue'' à une expression plus ``courte'' ?

```
int i;

char j;

j = i;
```

La valeur affectée à la variable *j* sera tronquée des bits de poids faibles.

La conversion d'un float en un int tronque la partie fractionnaire.

Dans tous les cas, il vaut mieux avoir une conversion explicite sinon se référer à la documentation du compilateur C !

# Les conversions explicites



- Le programmeur peut effectuer des conversions grâce à l'opérateur de conversion de type (*cast*).

**Exemple :**

L'expression (int) 1.225 convertit le réel de type double en un entier. il en découle évidemment une perte de précision.

La conversion de (int) 1.225 donne comme résultat la valeur entière 1.

# Récapitulatif



| Opérateur | Nom | Priorité | Ordre |
|---|---|---|---|
| `[]` | Elément de tableau | 15 | gauche-droite |
| `()` | Parenthèse | 15 | gauche-droite |
| `.` | Sélection | 15 | gauche-droite |
| `->` | Sélection | 15 | gauche-droite |
| `&` | Adressage | 15 | gauche-droite |
| `sizeof` | Dimension | 14 | droite-gauche |
| `(type)` | Conversion | 14 | droite-gauche |
| ! (unaire) | NON | 14 | droite-gauche |
| ~ (unaire) | NON bit à bit | 14 | droite-gauche |
| `++` | Incrément | 14 | droite-gauche |
| `--` | Décrément | 14 | droite-gauche |
| `*` | Multiplication | 13 | gauche-droite |

# Récapitulatif



| | | | |
|---|---|---|---|
| / | Division | 13 | gauche-droite |
| % | Modulo | 13 | gauche-droite |
| + | Addition | 12 | gauche-droite |
| - | Soustraction | 12 | gauche-droite |
| | Décalage à droite | 11 | droite-gauche |
| << | Décalage à gauche | 11 | droite-gauche |
| <= | Test d'inférieur ou égal | 10 | gauche-droite |
| = | Test supérieur ou égal | 10 | gauche-droite |
| < | Test inférieur strict | 10 | gauche-droite |
| > | Test supérieur strict | 10 | gauche-droite |
| == | Test d'égalité | 9 | gauche-droite |
| != | Test de non-égalité | 9 | gauche-droite |
| & | ET bit à bit | 8 | gauche-droite |

# Récapitulatif



| | | | |
|---|---|---|---|
| ^ | OU exclusif bit à bit | 7 | gauche-droite |
| \| | OU bit à bit | 6 | gauche-droite |
| && | ET | 5 | gauche-droite |
| \|\| | OU | 4 | gauche-droite |
| ?: | Opérateur conditionnel | 3 | droite-gauche |
| = | Affectation | 2 | droite-gauche |
| += | Affectation | 2 | droite-gauche |
| -= | Affectation | 2 | droite-gauche |
| .*= | Affectation | 2 | droite-gauche |
| /= | Affectation | 2 | droite-gauche |
| %= | Affectation | 2 | droite-gauche |
| = | Affectation | 2 | droite-gauche |
| <<= | Affectation | 2 | droite-gauche |
| &= | Affectation | 2 | droite-gauche |

# Récapitulatif



| | | | |
|---|---|---|---|
| \|= | Affectation | 2 | droite-gauche |
| ^= | Affectation | 2 | droite-gauche |
| , | Séquentiel | 1 | droite-gauche |