[LC022]

Remarques sur les expressions

L'ordre d'évaluation des arguments d'une expression est quelconque

(non imposé par le langage)

Exemple

Considérons l'expression suivante : `i = 4 ; x = ++i + (i+=3) .`

13 → ++i puis +=3 et enfin +
15 → +=i puis ++i et enfin +
16 → i'+i' avec i' = i+3+1

Exceptions

- Pour les opérateurs booléens && et || l'ordre d'évaluation est et vérifie la règle du " coupe circuit ".

  exemple :

```
if (i<lim-1 && (s[i]=getchar())!=c) {....}
```

- Pour les expressions composées :

  expression1 , expression2 , ... , expressionk

  l'ordre d'évaluation est aussi de gauche à droite. Le type et la valeur du résultat sont en fait ceux de l'expression qui est évaluée en dernier (c'est à dire la plus à droite). Les évaluations des autres expressions ne sont donc en fait utilisées que pour leurs .

  exemple :

```
z = ( t = 3 , t + 2 );
```

Remarque 1 : Les parenthèses sont nécessaires ici car l'opérateur "," est le moins prioritaire de tous.

Remarque 2 : Dans le cas de l'exemple précédent, on pourrait aussi écrire :

```
z = (( t = 3 ) + 2 );
```

Ne pas confondre && || avec & | ^

Exemple : Supposons **x = 21** = 00010101_B, **y = 25** = 00011001_B et **z = 36** = 00100100_B alors :

x && y vaut 1	x & y vaut 00010001 = 17	x | y vaut 00011101 = 29	x ^ y vaut 0C
x && z vaut 1	x & z vaut 00000100 = 4	x | z vaut 00110101 = 53	x ^ z vaut 31
x || y vaut 1	y & z vaut 00000000 = 0	y | z vaut 00111101 = 0x35	y ^ z vaut 3D

Expressions conditionnelles

Exemple :

```
/* max de a et b */
if (a>b) z = a;
else z=b;
```

z = a > b ? a : b;

Exemple :

```
for (i=0 ; i<N ; i++)  /* pour imprimer N éléments */
printf ("%6d%c", a[i], i%10 == 9 || i==N-1?'\n':' ');
```

Affectations - Valeurs gauches

Définitions

Une affectation simple (=) ou composée (+=, *=, ...) est un opérateur (binaire) utilisable dans des expressions. Il a donc deux opérandes (le gauche et le droit). Mais l'opérande gauche d'une affectation doit toujours désigner un objet modifiable (soit une zone mémoire, soit un registre). On trouve les noms suivants pour cet opérande gauche :

l-value ou **left-value** (en français : **valeur gauche**) ou encore **left-half-side** (**lhs**)

La condition de modifiabilité ci-dessus doit être aussi vérifiée pour les arguments (uniques) des opérateurs ++ (i = i+1 ⇔ i++) et -- (i = i-1 ⇔ i--). Pour cette raison ces arguments sont eux aussi considérés comme des "valeurs gauches".
Mise à part la contrainte d'être modifiable, une valeur gauche peut être n'importe quel objet.

Par exemples :

- ***v***

```
k += 3;          z = exp(x) + 2;          ++i;
```

- ***objets***

```
*(ptr + 5) = 3.14/t;          (*p)++;
```

- ***éléments de tableaux***

```
tab[7] = 49;          tt[312] *= 5;
```

- ***membres de structure ou d'union .***

```
ville.nom = "CLERMONT";          ptstar ->mgn = 2.3;
```

- ***structures .***

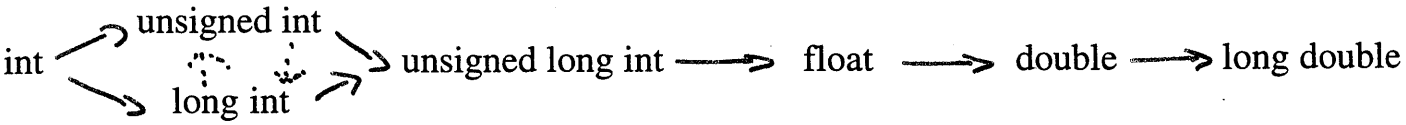
```
aujourd = demain;          const= constell[117];
```

Conversions de types

Conversions implicites dans une expression

- **promotion entière**

- **pour un opérateur binaire**



Conversions implicites lors d'une affectation

Il y a adaptation du type du résultat au type de l'objet désigné par la lvalue.

Conversions implicites lors d'un appel de fonction sans prototype

On retrouve la promotion entière (comme ci-dessus) mais aussi la promotion de tout **float** en **double.**

Conversions explicites ou conversions par "casting"

(encore appelé : adaptation forcée ou coercition -)

Principe : On précise le type entre parenthèses devant l'expression à convertir.

(type) expression

Ne pas perdre de vue en outre la grande priorité de cet opérateur unaire

Exemple : Quelles sont les conversions effectuées dans le calcul de **z** ci-dessous ?

```
{
 double z; int x; char y;
 ....;
 z = (float) x /y;
}
```

Exercice (tiré du DS du 13/11/98) :

1) Réécrire chacune des expressions suivantes en supprimant les parenthèses inutiles (c'est à dire les parenthèses que l'ont peut supprimer sans changer l'interprétation de l'expression) :

`k = ( ( 5 * x ) / ( k * db ) ) * ( z - w )`	k = 5 * x / (k * db) * (z - w)
`z = * ( tpt [ 7 ] )`	z = *tpt[7]
`w = ( y *= ( * ( * tpt ) ) , ( 3 + x ) )`	w = y *= (**tpt , 3+x)
`x = ( ( * a ) [ 2 ] ) ( 3 - ( s << 2 ) )`	x = (*a)[2](3 - s << 2)
`t = ( u += ( & ( ( ( st -> b ) [ 7 ] ) [ 2 ] ) ) )`	

2) On considère les déclarations suivantes pour les variables de la première expression :

```
int k; char w,z; float x; double db;
```

Décrire les conversions de type effectuées lors du calcul de cette expression :

3) Comment pourrait on modifier cette première expression pour supprimer l'une de ces conversions (qui est inutile) ?

Exercice (tiré du DS du 1/12/97) :

On considère les déclarations suivantes :

```
int y=2,x=1245;     long int t=5,k=3;     short int z=12;
float a=1.2,c;      double b=2.3;
```

Et l'expression suivante :

```
k *= c = ( k = x >> t & y + z ) / a * b ;
```

1) Mettre des parenthèses pour indiquer dans quel ordre sont faites les différentes opérations.
(indication : voir plus bas)

2) Préciser en quels types sont converties les différentes variables avant d'être utilisées. Préciser de même, s'il en existe, les conversions de sous-expressions :

variable	type
x	
t	
y	
z	
a	
b	
k (1ère fois)	
c	
k (2ème fois)	

ss-expressions	type

3) Quelle est la valeur finale de **k** ?

INDICATION pour tout l'exercice : La valeur finale de **c** est **11,5** .