Les expressions et l'affectation

Les expressions permettent de préciser les calculs à effectuer à l'aide d'opérateurs à partir des valeurs de variables et de constantes. Les valeurs obtenues peuvent être rangées dans les variables par l'instruction d'affectation. Chaque expression a un type qui dépend des opérandes et l'opérateur en jeu; c'est le type du résultat.
Les expressions peuvent être complexes et utiliser plusieurs opérateurs. Par la priorité d'opérateur le langage précise dans quel ordre les opérations doivent se faire. Quand deux opérateurs ont la même priorité le calcul s'effectue de gauche à droite.

Expressions arithmétiques

Elles produisent un résultat entier ou réel. Les opérateurs sont:

Nom	Opérateur	Priorité	Résultat
Moins unaire	**–**	14	Change le signe de son opérande. Il a le type d'opérande
Multiplication	*	13	Le type est réel, sauf si les deux opérandes sont entiers - alors le type est entier.
Division	/	13	Le type est réel, sauf si les deux opérandes sont entiers - alors le type est entier.
Reste	**%**	13	Les opérandes et le type du résultat sont entiers.
Addition	**+**	12	Le type est réel, sauf si les deux opérandes sont entiers - alors le type est entier.
Soustraction	**–**	12	Le type est réel, sauf si les deux opérandes sont entiers - alors le type est entier.

On peut utiliser des parenthèses () pour gérer l'ordre des opérations: (a+b)*c et a+b*c. résultats. L'appel peut être écrit au lieu d'un opérande. Les arguments sont toujours entre parenthèses.

Exemple: a + sin(b)

Les appels de fonction sont utilisés pour obtenir des résultats. Le (les) paramètre(s) de la fonction est (sont) toujours entre parenthèses.

Liste des fonctions arithmétiques:

abs(int) ou fabs(double)	valeur absolue
sqrt(double≥0)	racine carrée
sin(double)	sinus
cos(double)	cosinus
atan(double)	arcus tangente
exp(double)	exponentielle exp(x)=e^x
pow(double,double)	puissance x^y
log(double>0)	logarithme népérien
floor(double)	Le plus grand entier <= x
ceil(double)	Le plu petit entier >= x
random(i)	Un nombre aléatoire entier entre 0 et `i-1`

Exemples:

$\frac{a+b}{c+d}$ ⇒ (a+b)/(c+d)

$(1+x)\sqrt{\frac{2\sin 2x+\cos^2 x-3tgx^2}{1-e^{(2+x)}}}$ ⇒

```
(1+x)*sqrt((2*sin(2*x)+ pow(cos(x),2) - 3*tan(x*x)/(1-exp(2+x)))
```

Opérateurs incrémentales ++ et --

Les opérateurs incrémentales changes la valeur de leur opérande. Donc l'opérande doit être une Lvalue (v. au dessous).

Ils sont deux types : préfixe et postfixe.

Opérateur	Préfixe	Postfixe	Description
++	++X	X++	augmentation d'une unité
– –	– –X	X– –	diminution d'une unité

Quand la forme est postfixe le résultat rendu est la valeur de l'opérande avant l'opération. Quand la forme est préfixe le résultat rendu est la valeur de l'opérande après l'opération.

Exemples :

Supposons que la valeur de N est égal à 5:

Incrém. postfixe: `X = N++;` Résultat: N=6 et X=5

Incrém. préfixe: `X = ++N;` Résultat: N=6 et X=6

Décalages

`exp1` { `<<` | `>>` } `exp2`

*exp*1 et *exp*2 doivent être d'un type entier (char, short, long, int...). L'expression *exp*1 << *exp*2 (resp. *exp*1 >>*exp*2) représente l'objet de même type que *exp*1 dont la représentation interne est obtenue en décalant les bits de la représentation interne de *exp*1 vers la gauche17 (resp. vers la droite) d'un nombre de positions égal à la valeur de *exp*2.

Expressions booléennes

Le résultat de ces expressions est de type booléen. C'est un test, dont le résultat peut être vrai ou faux.
La plus simple des expressions booléennes est la relation, dont la syntaxe est:
<expr.arithm.><opérateur de relation><expr.arithm.>
Les opérateurs de relation sont:

<	<	>=	≥
>	>	==	=
<=	≤	!=	≠

Le résultat est vrai ou faux selon que la relation est vraie ou fausse.
a < 5 b>= 9.5 a+b != c/d
La priorité des opérateurs de relation est plus basse (10 et 9).
Opérateurs booléens:

Nom	Opérateur	Priorité	Résultat
Négation	**!**	14	Négation de l'opérande (**!** F = T).
Intersection logique	**&&**	5	Vrai si et seulement si les deux opérandes sont vrais.
Union logique	**||**	4	Faux si et seulement si les deux opérandes sont faux.

Tableau de vérité

a	b	**!** b	a **&&** b	a **||** b
F	F	T	F	F
F	T	F	F	T
T	F	T	F	T
T	T	F	T	T

Note: En C il n y a pas un type logique (booléen). Ce type est présenté par le type int. La valeur FAUX vaut zéro et la valeur VRAI vaut n'importe quelle valeur différente de zéro, mais les expressions booléennes rendent la valeur 1.

Les expressions booléennes en C sont exécutées en raccourci :

- si le premier opérande de l'opérateur && est **faux** il est claire que le résultat sera faux n'importe quelle est la valeur du deuxième opérande et donc le dernier ne sera pas calculé.
- si le premier opérande de l'opérateur || est **vrai** il est claire que le résultat sera vrai n'importe quelle est la valeur du deuxième opérande et donc le dernier ne sera pas calculé.
-

Exercices:

1. Prouvez que:
   a) (a **||** b) **||** c ≡ a **||** (b **||** c)
   b) **!** a **&& !** b ≡ **!** (a **||** b)+
   c)
   d) **!** a **|| !** b ≡ **!** (a **&&** b)
2. Ecrivez une expression booléenne qui est vraie si la valeur de x est dans l'intervalle [3.5,5.2].

Ecrivez une expression booléenne qui est vraie si le point avec des coordonnées (x,y) se trouve dans le cercle de centre (a,b) et radius r.

Opérateurs de manipulation des bits (~ & ^ |)

Ces opérateurs sont les opérations logiques sur les bits correspondants des représentations internes des valeurs des opérandes.

. Le tableau suivant montre les résultats

Bit a	Bit b	~ b	a & b	a ^ b	a | b
0	0	1	0	0	0
0	1	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	1

Ces opérateurs sont utilisés pour tester la valeur de certain(s) bit(s) dans une valeur. Par exemple `(x & 0x4)` teste le 3-ème bit de x, et `(x & 0x3)` rends une valeur différente de `0 si n'importe qui des premiers deux bits de x est différent de 0.`

Expression conditionnelle

Elle a la syntaxe suivante :
condition ? *expression-1* : *expression-2*
Cette expression est égale à **`expression-1`** si **`condition`** est satisfaite, et à **`expression-2`** sinon.

Exemple :

l'expression

```
x ? x : -x
```
correspond à la valeur absolue d'un nombre

De même l'instruction

```
m = ((a > b) ? a : b);
```
affecte à ***m*** le maximum de ***a*** et de ***b*.**

Conversion des types et taille des données

Opérateur *sizeof*

C'est un opérateur unaire qui rend la taille de son opérande en octets. il a deux formes :

```
sizeof <nom d'un objet>
sizeof (<type>)
```
Exemples :

long b ;

```
sizeof (short) // rend 2
sizeof b       // rend 4
```

Conversion des types

Quand les opérandes d'un opérateur binaire sont de différents types on doit convertir le type d'un des opérandes vers l'autre. Cette conversion peut être **implicite** (par défaut) ou **explicite** (à l'aide d'un opérateur de conversion). La conversion implicite obéit certains règles. Dans le cas d'affectation le compilateur essaie se convertir le type de la valeur de l'expression droite vers le type de la Lvalue gauche si c'est possible. Dans certains cas le résultats est une perte d'exactitude ou même des chiffres les plus signifiantes. Pour les autres opérateurs binaires les règles suivantes sont valides :

- Si un des opérandes est de type long double, convertir l'autre dans le type long double ; le type de l'expression sera long double.
- Sinon, si un des opérandes est de type double, convertir l'autre dans le type double ; le type de l'expression sera double.
- Sinon, si un des opérandes est de type float, convertir l'autre dans le type float ; le type de l'expression sera float.

- Effectuer la *promotion entière* : convertir les char, les short, les énumérations et les champs de bits en des int. Si l'une des valeurs ne peut pas êetre repréesentée dans le type int, les convertir *toutes* dans le type unsigned int.
- Ensuite, si un des opérandes est unsigned long, convertir l'autre en unsigned long ; le type de l'expression sera unsigned long.
- Sinon, si un des opérandes est long et l'autre unsigned int :
  - si un long peut représenter toutes les valeurs unsigned int, alors convertir l'opérande de type unsigned int en long. Le type de l'expression sera long ;
  - sinon, convertir les deux opérandes en unsigned long. Le type de l'expression sera unsigned long.
- Sinon, si un des opérandes est de type long, convertir l'autre en long ; le type de l'expression sera long.
- Sinon, si un des opérandes est de type unsigned int, convertir l'autre en unsigned int ; le type de l'expression sera unsigned int.
- Sinon, et si l'expression est correcte, c'est que les deux opérandes sont de type int ; le type de l'expression sera int.

- Exemples :

```
int a = 2,b =4,c = -3;
short i = 4; long j=35897
float x =3.2; double z = -4.5;
char d = ‘A’;
2/3. // 2 est  converti en float et le résultat e 1.5
a = j // le deuxième octet de j est coupé et à a est affecté -29912
b = x; // la fraction de x est coupée et la partie entière 3 est affectée à b
c = z; // -4 est affecté à c
x = 5/a; // le résultat 2 est converti en float et est affecté à x
c = d;  // la valeur de char est convertie en entier et 65 est affecté à c
d = 66; // la valeur 66 est convertie en char et ‘B’ est affecté à d
```

Conversion explicite

L’opérateur (<*type*>) s’appelle **cast** en anglais. Il convertit la valeur de son opérande en type dont le nom est entre les parenthèses. Dans certain cas ça exige la modification de la valeur elle-même.

Exemples :

```
int a = 2,b =4;
unsigned c = -3;
short i = 4; long j=35897
float x =3.2; double z = -4.5;
char d = ‘A’;
a = (float)c/2; // résultat a= -1.5
b = (int)x;  // résultat b=3
c = (unsigned int) j; // résultat c=35897
a = (int)x/2 ; // résultat a = 1
```

Opérateurs sur les pointeurs

* – Indirection (déréférence) – c'est un opérateur qui, appliqué à une valeur de type pointeur, délivre la valeur pointée. Par exemple si le pointeur *pi* désigne la variable *i*, **pi* délivre la valeur de *i*.

& – Adresse – Cet opérateur, appliqué sur une variable, délivre son adresse. Par exemple *&x* rend l'adresse de la variable *x,* qui peut plut tard être affecté à un pointeur *p* déclaré sur le même type que *x*.

Exemple:

Si nous avons les définitions:

```
int j, *pi; /* j est une variable du type int, pi est un pointeur sur int.
                  Leurs valeur sont indéfinies */
j = 10; /* la valeur de j devienne 10 */
pi = &j; /*  maintenant la valeur de pi = l'adresse de j */
printf("%d",*pi); /* ça va afficher 10 */
*pi = 12;  /* maintenant la valeur de j est 12 */
```

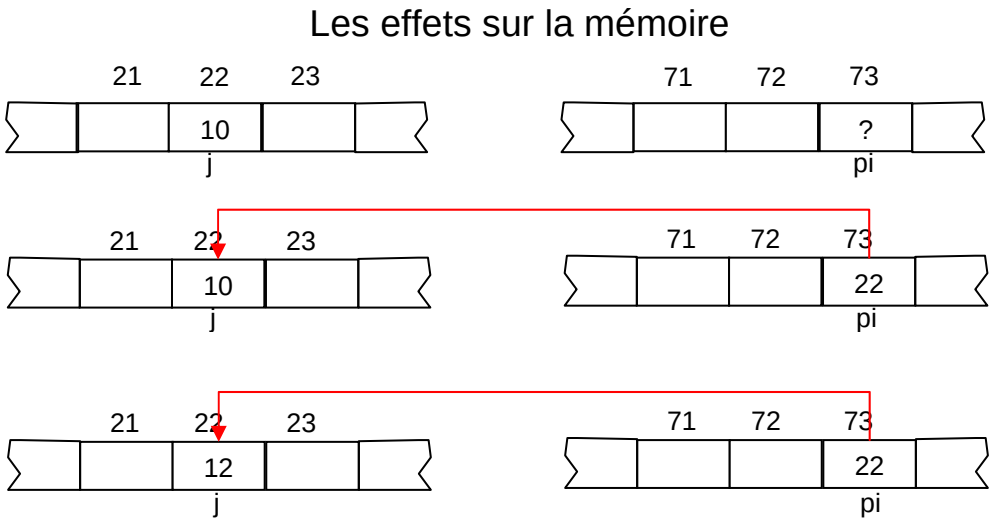

Fig.2.1 L'état de la mémoire

Précédence des opérateurs

Priorité	Opérateurs	Sens de l'associativité
15	**() [] -> .**	gauche à droit
14	**! ~ +(un) -(un) ++ -- &(un) *(un) (type) sizeof**	droit à gauche
13	***(bin) / %**	gauche à droit
12	**+(bin) -(bin)**	gauche à droit
11	**<< >>**	gauche à droit
10	**< <= > >=**	gauche à droit
9	**== !=**	gauche à droit
8	**&(bin)**	gauche à droit
7	**^**	gauche à droit
6	**|**	gauche à droit
5	**&&**	gauche à droit
4	**||**	gauche à droit
3	**?:**	droit à gauche
2	**= *= /= %= += -= &= ^= != <<= >>=**	droit à gauche
1	**,**	gauche à droit

L'instruction et l'opérateur d'affectation

Lvalue et Rvalue

Ce sont deux très importants concepts du langage C.

Lvalue – expressions signifiant « le contenu de... ». C'est une expression qui doit délivrer une variable (par opposition à une constante). Par exemple : le nom d'une variable, le nom d'un champs de structure, l'élément d'un tableau, la valeur désigné par un pointeur.

Rvalue – expressions signifiant « la valeur de... ». C'est une valeur – résultat d'une expression, mais elle n'est pas liée avec une location de la mémoire.

L'affectation simple

La syntaxe est:
<*Lvalue*> = <*expression*>
L'expression est calculée et sa valeur est rangée dans la Rvalue. L'expression doit être de type compatible avec celui de la Rvalue. C'est le cas quand elles sont du même type ou quand l'expression est entière et la variable réelle. Mais quand

la variable est entière et l'expression réelle il faut utiliser "floor" ou "ceil" pour transformer la valeur réelle à une valeur entière.

Exemples:

Si on a les déclarations:

```
double x,a; int d,b;  char k : char;
ce sont des affectations légales:
x = a + 5*b/(x-1);
d  = b==0;
L'affectation est aussi un opérateur dont le résultat et la valeur affectée. De tell
façon l'opérateur de l'affectation peut participer dans une expression. Exemples :
k = b =  ';' /* ';' est affectée à  k */
(x =  b – 4) == 0 /* cette expression vérifie si à x est affectée la valeur 0 */
```

L'affectation est un opérateur en C et son résultat et la valeur affectéeé De telle façon on peut utiliser cette valeur dans autre expressions

Exemples :

```
x = y = 5 ; /* y accepte la valeur de 5 et x prends aussi la m¸eme valeur */
x = (a = x+2) + 2 ; /*Ici a prends la valeur de 7 (x vaut 5) mais l'expession utilise
                      cette valeur et la valeur 9 est affectée à x */
(x = 5*b) > 4 /*A x est affectée une valeur et elle est testée à la fois */
```

Affectation étendue (élargie ou complexe)

$$\text{Lvalue} \left\{ \begin{array}{l} += \\ -= \\ *= \\ /= \\ \%= \\ >>= \\ <<= \\ \&= \\ \wedge= \\ \mid= \end{array} \right\} \text{exp}2$$

ou

```
exp1 •= exp2
```

Cette forme peut être vue comme

```
exp1 = exp1 • exp2
```

mais avec une seule évaluation de exp1. L'expression résultante n'est pas une *lvalue.*

```
Exemples
```

x +=2 ; */* Equivalent à x=x+2 */*

a *= b+3 ; */* Equivalent à a= a*(b+3) */*