



Programmation langage C

Programmation langage C

- ❖ Généralités
- ❖ Types de base
- ❖ Opérateurs et les expressions en C
- ❖ Entrées et sorties conventionnelles
- ❖ Instructions de contrôle
- ❖ La programmation modulaire et les fonctions
- ❖ Les tableaux et les pointeurs
- ❖ Les chaînes de caractères
- ❖ Les structures et les énumérations
- ❖ Les fichiers
- ❖ La gestion dynamique de la mémoire
- ❖ Le préprocesseur

Généralités

- ❖ Avant propos
 - Historique
 - Intérêt du langage C
- ❖ Découverte du langage C
 - Exemple de programme C
 - Structure d'un programme C
 - Règle d'écriture
 - Création d'un programme

Historique

- ❖ Le langage C a été créé en 1972 par **Denis Ritchie** avec un objectif relativement limité : écrire un système d'exploitation (**UNIX**). Mais ses qualités opérationnelles l'ont très vite fait adopter par une large communauté de programmeurs.
- ❖ Il provient de deux langages : **BPCL** développé en 1967 par **Martin Richards** et **Bell** développé en 1970 chez AT&T par **Ken Thompson**.
- ❖ Il fut limité à l'usage interne de **Bell** jusqu'en 1978, date à laquelle **Brian Kernighan** et **Dennie Ritchie** publièrent les spécifications définitives du langage : ***The C programming Language***.
- ❖ Au milieu des années 1980, la popularité du langage était établie. De nombreux compilateurs apparaissent et comportent des incompatibilités. L'*American National Standard Institute* (ANSI) formaient un groupe de travail pour normaliser le langage C qui aboutit en 1988 avec la parution du manuel : ***The C programming Language-2 ème édition***

Intérêt du langage C

- ❖ Universel :
 - Pas orienté sur un domaine d'application spécifique comme **FORTRAN** (applications scientifiques) ou **COBOL** (Applications commerciales ou traitant de grandes quantité de données)
- ❖ Polyvalent:
 - Il permet le développement de systèmes d'exploitation, de programmes scientifiques et de programmes de gestion
- ❖ Langage structuré et évolué :
 - Qui permet néanmoins d'effectuer des opérations de bas niveau
- ❖ Portabilité :
 - Recompilation des sources sur la nouvelle plateforme
- ❖ Rapide :
 - Code très proche du langage machine (suite de bits)
- ❖ Extensible :
 - Nombreuses bibliothèques
- ❖ Père syntaxique du C++, Java, PHP, etc.

Désavantages du langage C

- ❖ Programme difficile à maintenir si
 - La programmation est trop compacte
 - Les commentaires sont insuffisants

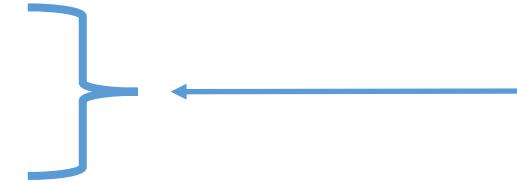
- ❖ Limite de la Portabilité en C
 - Dès que l'on utilise une bibliothèque externe la portabilité devient difficile

Découverte du langage C

- ❖ Un exemple de programme
- ❖ Structure d'un programme C
- ❖ Règles d'écriture
- ❖ Création d'un programme C

Exemple de programme C

```
#include <stdio.h>
#include <math.h>
#define NFOIS 5
```



Directives prises en compte avant la compilation du programme

```
main()
{ int i ;
  float x ;
  float racx ;
```



Déclaration des variables

```
printf ("Bonjour\n") ;
printf ("Je vais vous calculer %d racines carrées\n", NFOIS) ;
```

Affichage d'un message à l'écran

```
for (i=0 ; i<NFOIS ; i++)
{ printf ("Donnez un nombre : ") ;
  scanf ("%f", &x) ;
  if (x < 0.0)
    printf ("Le nombre %f ne possède pas de racine carrée\n", x) ;
  else
    { racx = sqrt (x) ;
      printf ("Le nombre %f a pour racine carrée : %f\n", x, racx) ;
    }
}
printf ("Travail terminé - Au revoir") ;
```

Boucles principales

Lire la valeur entrée au clavier

Impression du résultat

Structure d'un programme C

```
#include<stdio.h>
#define constante
déclaration de fonctions et de variables
main()
{
    déclaration de fonctions et de variables
    instructions
}
fonction_1()
{
    déclaration de fonctions et de variables
    instructions
}
fonction_2()
{
    .....
}
```



Zone des déclarations

Zone des fonctions

Règles d'écriture

❖ Les identificateurs

- Ils permettent de désigner les différents « objets » manipulés par le programme: variables, fonctions, etc.
- un identificateur est composé de lettres et de chiffres, le caractère souligné _ est considéré comme une lettre
- Le premier caractère est une lettre
- Différenciation entre minuscule et majuscule (respect de la casse):
Compte est différent de **compte**

Exemple:

❖ Taux, nom_professeur, nb_mots

Règles d'écriture

Mots clés

auto	default	float	register	struct	volatile
break	do	for	return	switch	while
case	double	goto	short	typedef	
char	else	if	signed	union	
const	enum	int	sizeof	unsigned	
continue	extern	long	static	void	

Les séparateurs

Dans la langue écrite , les différents mots sont séparés par un espace, un signe de ponctuation ou une fin de ligne

- intx, y; /*incorrecte*/
- int x,y,z; /*correcte*/
- int x, y, z; /*correcte et lisible*/

Règles d'écriture

- Une instruction peut s'étendre sur un nombre de lignes quelconque mais attention à conserver une programmation claire !

```
#include <stdio.h>
#include <math.h>
#define NFOIS 5
main() { int i ; float
          x
        ; float racx ; printf ("Bonjour\n") ; printf
          ("Je vais vous calculer %d racines carrées\n", NFOIS) ; for (i=
0 ; i<NFOIS ; i++) { printf ("Donnez un nombre : ") ; scanf ("%f"
, &x) ; if (x < 0.0)
          printf ("Le nombre %f ne possède pas de racine carrée\n", x) ; else
{ racx = sqrt (x) ; printf ("Le nombre %f a pour racine carrée : %f\n",
x, racx) ; }     }     printf ("Travail terminé - Au revoir") ; }
```

Règles d'écriture

Les commentaires

- Il s'agit de textes explicatifs destinés aux lecteurs du programme et qui n'ont aucune incidence sur sa compilation

Texte entre les caractères /* et */.

```
/* commentaire */
*****
commentaires
sur plusieurs lignes
******/
```

```
float tva;      /* tva dépend de la classe du produit */
for (i = debut; i < fin; i++) /* Boucle principale */
```

Création d'un programme C

Edition d'un programme :

- consiste à créer à partir du clavier tout, ou une partie du texte d'un programme (programme source) « **.c** »

La compilation :

- consiste en langage C à traduire le programme source en langage machine en 2 étapes
 - **Traitement par le préprocesseur** : exécution des directives qui le concernent (reconnu par le #)
 - **Compilation** proprement dite càd la traduction en langage machine du texte en langage C fourni par le préprocesseur (résultat : prgm.obj)

L'édition de liens:

- Recherche les modules objet nécessaires dans la bibliothèque standard (printf, scanf, sqrt,...)
- Création de l'exécutable (ensemble d'instruction en langage machine)

Les types de bases du langage C

Les types de base

- Notion de type
 - Le type sert à donner une signification à une information rangée en mémoire
 - Quel que soit le type d'une information, nombre **77** ou la lettre **M**, il est rangé en mémoire sous forme de **bits** (binary digit (0,1)) regroupés en paquets de 8 nommés octets (byte)
 - Exemple : 01001101 → 77 → M
- Les types de base :
 - Entier : **int**
 - Réel, nombre flottant : **float**
 - Caractère : **char**

Les types de base

- **Les types entiers**
 - Le mot clé **int** correspond à la représentation de nombres entiers.
 - **short int** ou **short** (en abrégé)
 - **int**
 - **long int** ou **long** (en abrégé)

Exemple: les **short** et **int** peuvent correspondre à 16 bits (de -32 768 à 32 767) et les **long** correspondent à 32 bits (de -2 147 483 648 à 2 147 483 647).

NB : Tous les entiers n'ont pas la même taille selon les machines !!!

Les types de base

Représentation des entiers en mémoire

□ Représentation des entiers	1	0000 0001	01
	2	0000 0010	02
	127	0111 1111	7F
	0	0000 0000	00
	-1	1111 1111	FF
	-2	1111 1110	FE
	-127	1000 0001	81
	-128	1000 0000	80

- les négatifs sont représentés en "**complément à deux**", ce qui signifie que pour un nombre négatif, on prend la valeur absolue calculée en binaire puis on inverse tous les bits (**0 => 1**, **1 => 0**), puis on ajoute **1**.

Les types de base

Constantes entières

- La façon la plus naturelle de déclarer une constante entière est de l'écrire sous forme de décimal, octal ou hexadécimal
- Les constantes décimales : +92 -46

Notation octale

précédé du chiffre **0** **92 = 0134**

Notation hexadécimale

précédé du chiffre **0x** ou **0X** **92 = 0x5C**

- Une constante entière est par défaut de type **int**.

Elle est de type **long** si elle est suffixée par les lettres **l** ou **L**, et non signée lorsqu'elle est suffixée par les lettres **u** ou **U**

Base 10 : **12lu** Base octale : **014u** Base hexadécimale: **0xCl**

Les types de base

Les types flottants

Les nombres réels sont représentés de manière approchée grâce au type ***float***

- **float**
- **double**
- **long double**

Notation scientifique, flottant, virgule flottante : M * B^e

Exemple: 124.765 ----> 1.24756 * 10²

Un réel peut être représenté par une mantisse **M** et un exposant **e**; dans notre exemple, la mantisse est **1.24756** et l'exposant est **2**.

Attention: La représentation des réels par les flottants n'est qu'une approximation.

Les types de base

Les types flottants

Précision de calcul

- **Précision :** La limitation du nombre de décimales impose une erreur dite de troncature.
Cette erreur est de 10^{-6} pour le type **float** et 10^{-10} pour le type **double**.

- **Domaine de couvert :** l'ensemble des nombres représentables à l'erreur de troncature près.

On est assuré qu'il s'étendra de 10^{-37} à 10^{+37}

- **Les constantes :**

Notation décimale et exponentielle

125.3	-0.089	-.3 .34
2.34E4	2.34E+4	23.4E+3
5.678E-29	56.78E-30	5678E-32

NB: Par défaut, les constantes sont créées en type **double**; on peut forcer le type des constantes en faisant suivre la constante par **f** ou **F** pour **float** et **l** ou **L** pour long double.

Les types caractère, char

- **char**

char mon_char = 'A';

- Les caractères disponibles dépendent de l'environnement mais on dispose des caractères avec lesquels on écrit les programmes : les minuscules, les majuscules, les chiffres, les séparateurs, des signes de ponctuations.
- Les caractères sont stockés sur un octet : **'A' '+'?'**
- On peut noter les caractère en code ASCII octal \et hexadécimal \x

'A' '\x41' '\101'

Les types de base

Séquences d'échappement	Notation en C	code ASCII	Abréviation
tabulation horizontale	\t	9	HT
retour à la ligne	\n	10	LF
Guillemets	\ "	34	«
apostrophe	\ '	39	'
point d 'interrogation	\?	63	?
anti-slash	\ \	92	\

Les types de base

Les constantes

◆ Grâce au préprocesseur

- `#define NB 15`
- `int i = NB;`

◆ les constantes symboliques : `const int nb = 15;`

- `const int constante_symbolique = 15;`
- `int i = constante_symbolique;`
- ne peuvent faire partie d'une expression constante (*les indices de tableaux*)

◆ les valeurs numériques

`15` ==> `int`

`15L` ==> `long int`

`15.0` ==> `double` (les variables réelles sont par défaut doubles)

`15.0F` ==> `float`

Les opérateurs

Les opérateurs

- ◆ Notion d'instructions et d'expressions
- ◆ Les opérateurs arithmétiques
- ◆ Les priorités et associations
- ◆ Les conversions
- ◆ Les opérateurs relationnels
- ◆ Les opérateurs logiques
- ◆ Les opérateurs d'affectation (Lvalue)
- ◆ Les opérateurs d'incrémentation
- ◆ Les opérateurs d'affectation d'élargie
- ◆ L'opérateur CAST
- ◆ L'opérateur conditionnel
- ◆ L'opérateur séquentiel
- ◆ L'opérateur SIZEOF
- ◆ Tableau récapitulatif des priorités des opérateurs

Les opérateurs

Dans les langages en général :

Notion d'expression: Les expressions sont formées entre autres à partir d'opérateurs et possèdent une valeur mais n'effectuent rien

Notion d'instruction: les instructions peuvent contenir des expressions et effectuent une action et ne possèdent pas de valeur

Par exemple, l'instruction d'affectation =====> $a = b * c + d + 8;$

En général, dans les langages, les notions **d'expression** et **d'instruction** sont distinctes.

Les opérateurs

Dans le langage C :

`k = i = 5;` interprétée comme `k = (i = 5);`
`i = 5` est une expression et renvoie 5

Il existe une instruction qui est une expression terminée par un point virgule
`a = 5;` est une instruction

Par exemple, l'instruction d'incrémantation ======> `a = i++;`

Autre exemple :

```
int resultat_du_test;  
resultat_du_test = ( i < 0 );
```

`resultat_du_test` vaut vrai (tout sauf zéro 0) ou faux (0 zéro) selon la valeur de `i`.

Les opérateurs

Les opérateurs arithmétiques

◆ Les opérateurs arithmétiques :

- ◆ L'addition : $a + b$
- ◆ La multiplication : $a * b$
- ◆ La division : a / b
- ◆ Modulo : $a \% b$, **a** et **b** doivent être entiers, modulo renvoie le *reste de la division* de **a** par **b**;

◆ par exemple :

$$a = c * b + a$$

Les opérateurs

Les priorités

- ◆ comme dans l'algèbre traditionnelle * et / ont une priorité supérieure à + et -
- ◆ les parenthèses permettent d'outrepasser les règles de priorité.

$$\begin{aligned} 3 * 4 + 5 &\quad \text{ \iff } \quad (3 * 4) + 5 \\ 3 * (4 + 5) &\quad \text{ \iff } \quad 3 * (4 + 5) \end{aligned}$$

- ◆ Les priorités sont définies et sont utilisées pour fournir ***un sens à une expression.***

Les opérateurs

Association

- ◆ En cas de priorités identiques, les calculs s'effectuent de **gauche à droite**, "**associativité gauche droite**" sauf pour l'affectation et quelques autres opérateurs.

- ◆ **Association droite de l'affectation :**
 $a = b = c = d = 5 \iff (a = (b = (c = (d = 5))))$

Les opérateurs

Ordre d'évaluation des opérandes

- ◆ Par ailleurs, l'ordre d'évaluation des opérandes d'un opérateur n'est pas imposé par la norme.

$$a = f() + g()$$

- ◆ **Attention !** aux effets de bord, dans notre exemple l'ordre d'évaluation de ***f()*** et ***g()*** n'est pas déterminé.
- ◆ En revanche l'évaluation de gauche à droite est garantie par la norme pour les opérateurs logiques "**&&**" "**||**" et pour l'opérateur virgule **,**".

Les opérateurs

Ordre d'évaluation des opérateurs commutatifs

- ◆ **Attention !** L'ordre d'évaluation de 2 opérateurs "commutatifs" n'est pas défini et ne peut être forcé par les parenthèses.

$$\begin{aligned} a + b + c &\Longrightarrow a + b \text{ puis } +c \\ \text{ou} \quad a + b + c &\Longrightarrow b + c \text{ puis } +a \end{aligned}$$

Les opérateurs

Les conversions

◆ Les conversions d'ajustement de type :

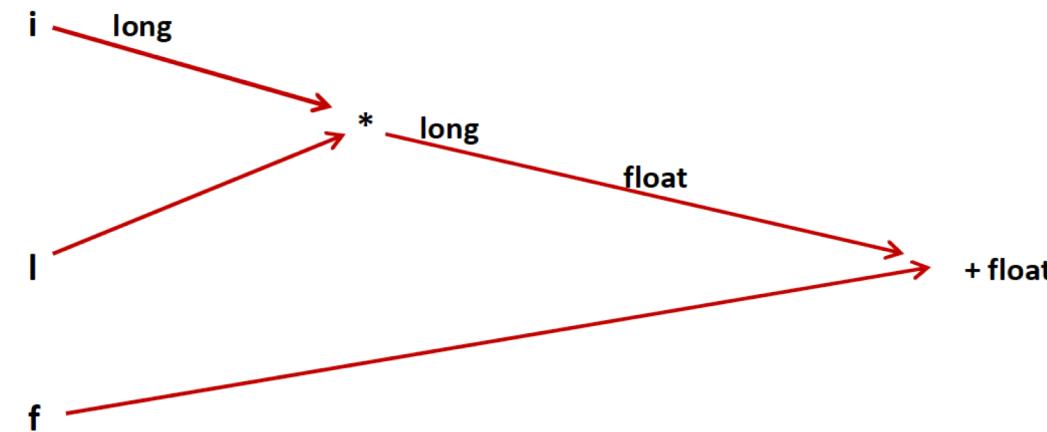
Les conversions se font selon une hiérarchie qui permet de ne pas dénaturer la valeur.

int > long > float > double > long double

Exemple :

$i * l + f$

i est un **int**, l est un **long**, f est un **float**



Les opérateurs

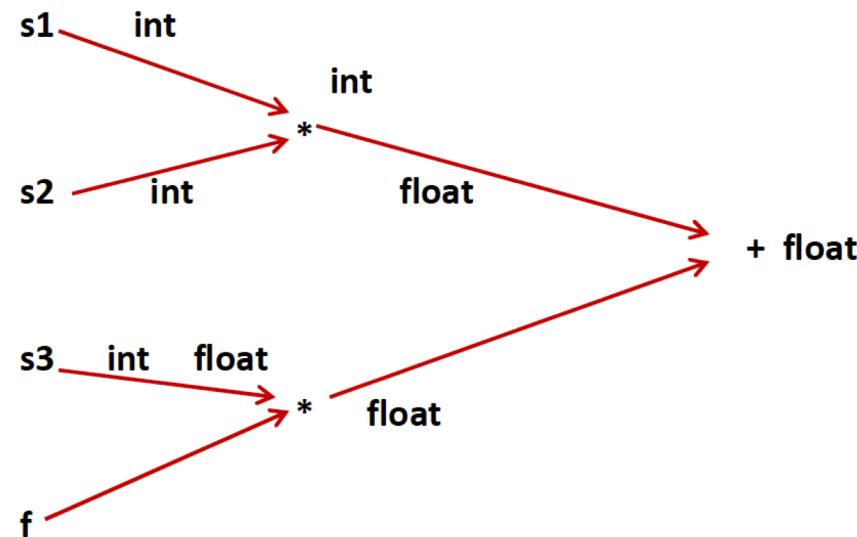
Les promotions numériques

- Les opérateurs arithmétiques ne sont pas définis pour les types **short** et **char**.
Le langage **C** convertit automatiquement ces types en **int**.

Exemple :

$s1 * s2 + s3 * f$

$s1, s2, s3$ sont des **short**, f est un **float**.



Les opérateurs

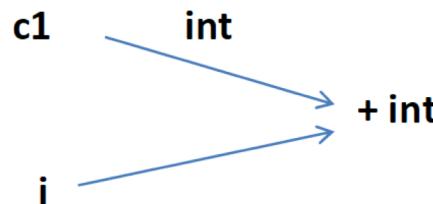
Les promotions numériques des caractères

L'entier associé à un caractère donné n'est pas toujours le même.

Exemple :

c1 + i

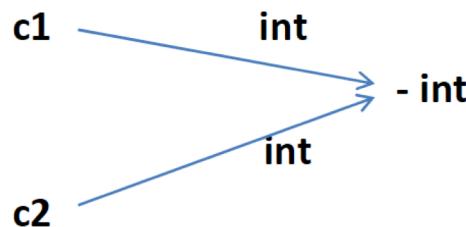
c1 est un **char**, **i** est un **entier**.



Exemple :

c1 - c2

c1 et **c2** sont des **char**.



Les opérateurs

Les opérateurs conditionnels

- ◆ Le résultat d'une comparaison est un entier et non un booléen
 - ◆ 0 si le résultat de la comparaison est **faux**.
 - ◆ 1 si le résultat de la comparaison est **vrai**, tout autre nombre que 0 est considéré comme **vrai**

Opérateur	signification
<	inférieur à
<=	inférieur ou égal à
>	supérieur à
>=	supérieur ou égal à
==	égal à
!=	différent de

Les opérateurs

Les opérateurs logiques

❖ **(condition1)&&(condition2)** condition1 et condition2

ET	Faux	Vrai
Faux	Faux	Faux
Vrai	Faux	Vrai

Les opérateurs

Les opérateurs logiques

◆ **(condition1) || (condition2)** condition1 **ou** condition2

OU	Faux	Vrai
Faux	Faux	Vrai
Vrai	Vrai	Vrai

.!(condition) not condition

Not	
Faux	Vrai
Vrai	Faux

Les opérateurs

Les opérateurs logiques

- ◆ Les opérateurs logiques acceptent n'importe quel opérande numérique.

- ◆ `int a, b, n;` `a && b` `a || b` `!n`
- ◆ `if(n == 0)` peut s'écrire `if(!n)`

◆ Vrai et faux

- ◆ `0` correspond à faux
- ◆ toute valeur non nulle correspond à vrai

Les opérateurs

Les opérateurs logiques

- ◆ L'évaluation des conditions se fait de gauche à droite et seulement si cela est nécessaire.

$(a > b) \&\& (c > d)$

- ◆ L'expression $(c > d)$ n'est pas évaluée si l'expression $(a > b)$ est fausse
- ◆ Attention aux effets de cette règle, par exemple :

```
if ((i < NBFOIS) && (( n = get_int()) > 0))
    res = sqrt(n);
else
    {
        ...
    }
```

Les opérateurs

L'opérateur d'affectation

X = 1

A gauche de l'affectation, on peut mettre une expression; mais à droite de l'affectation, il faut placer un objet qui est une référence en mémoire, **une Lvalue**.

3 + X = 12 incorrect

Lvalue:

C'est la partie gauche d'une affectation qui possède une adresse mémoire : un identificateur, membre d'un tableau etc ...

L'associativité de l'affectation est de droite à gauche :

X = Y = 5

Conversion

Si le résultat de l'évaluation de la partie droite de l'affectation est de type différent de celui de la **lvalue** alors il y a conversion imposée dans le type de la **lvalue**, d'où le risque de perte d'information.

Attention lors d'une conversion **float/int**, on perd la partie décimale !

Les opérateurs

L'opérateur d'incrémentation

- ◆ Les opérateurs d'incrémentation «`++` » et de décrémentation «`--`» permettent de remplacer les deux instructions suivantes :

`i = i + 1; <====> i++;`

`j = j - 1; <====> j--;`

- ◆ `++` est :

- ◆ un opérateur de **pré-incrémantation** si la lvalue est placée à droite.
- ◆ un opérateur de **post-incrémantation** si la lvalue est placée à gauche.

- ◆ `--`est

- ◆ un opérateur de **pré-décrémantation** si la lvalue est placée à droite.
- ◆ un opérateur de **post-décrémantation** si la lvalue est placée à gauche.

- ◆ Exemple : que vaut `i` et `j` ?

`i = 6;`

`j = --i + 4;`

`j = i++ + 4;`

Les opérateurs

L'affectation élargie

$X = X + 7 <==> X += 7$

encore plus fort

$X = X * 7 <==> X *= 7$

de manière générale

Lvalue = Lvalue **opérateur** expression $<==>$ Lvalue **opérateur** =expression

Les opérateurs concernés sont :

+= -= *= /= %= |= ^= &= <<= >>=

Les opérateurs

L'opérateur cast

- ◆ On peut forcer la conversion d'une expression grâce à l'opérateur **cast**
(type) expression_à_caster

- ◆ Permissivité excessive du C.

- ◆ **Exemple :**

```
double d;  
int i, j;  
i = 20; j = 7;  
d = i / j;          /* d vaut 2 */  
d = (double)(i/j); /* d vaut 2.0 */  
d = (double)(i)/j; /* d vaut 2.857143 */  
d = 20 / 7;         /* d vaut 2 */  
d = 20.0 / 7.0;    /* d vaut 2.857143 */
```

Les opérateurs

L'opérateur conditionnel

si a > b alors max = a;
sinon max = b;

peut s'écrire :

max = a > b ? a : b

Format de l'opérateur conditionnel

cond ? valeur de retour si cond est vraie : valeur de retour si cond est fausse

Attention aux priorités !

max = a > b ? a : b;

fonctionne mais pour éviter les confusions :

max = (a > b ? a : b);

est plus claire.

Les opérateurs

L'opérateur séquentiel

- ◆ L'opérateur séquentiel est la virgule.

a * b , c + d

- ◆ **a * b** est évalué d'abord puis **c + d**. Cette expression renvoie la valeur de **c + d**.
- ◆ if (**i++**, a > b) ... <==> **i++**; if (a > b)...
- ◆ **for(i = 0, j = 10; . . . ; . . .)** <==> **i = 0**; **for(j = 10; . . . ; . . .)**

Les opérateurs

L'opérateur sizeof()

- ◆ L'opérateur **sizeof()** renvoie la taille en octet de l'opérande ou du type.

int **i**;

float **j**;

sizeof(**i**); ==> **2**

sizeof(**j**); ==> **4**

- ◆ On peut utiliser **sizeof** avec le nom des types :

sizeof(**int**); ==> **2**

sizeof(**float**); ==> **4**

Les opérateurs

Récapitulatif des priorités

Catégorie

Référence

Unaire

Arithmétique

Décalage

Relationnel

Manipulation de bits

Logique

Conditionnel

Affectation

Séquentiel

Opérateurs

`() [] ! .`

`+ - ++ -- ! ~ * & (cast) sizeof`

`+ - * / %`

`<< >>`

`< <= > >= == !=`

`& ^ |`

`&& ||`

`? :`

`= += -= *= /= %= &= ^= |= <<= >>=`

`,`

Exos d'application

Soit les déclarations suivantes :

int n = 10 , p = 4 ;

long q = 2 ;

float x = 1.75 ;

Donner le type et la valeur de chacune des expressions suivantes :

- a) n + q
- b) n + x
- c) n % p +q
- d) n < p
- e) n >= p
- f) n > q
- g) q + 3 * (n > p)
- h) q && n
- i) (q-2) && (n-10)
- j) x * (q==2)
- k) x *(q=5)

Exos d'application

2) n étant de type int, écrire une expression qui prend la valeur :

- 1 si n est négatif,
- 0 si n est nul,
- 1 si n est positif.

Quels résultats fournit le programme suivant ?

```
#include <stdio.h>
main()
{
    int n=10, p=5, q=10, r ;
    r = n == (p = q) ;
    printf ("A : n = %d p = %d q = %d r = %d\n", n, p, q, r) ;
    n = p = q = 5 ;
    n += p += q ;
    printf ("B : n = %d p = %d q = %d\n", n, p, q) ;
    q = n < p ? n++ : p++ ;
    printf ("C : n = %d p = %d q = %d\n", n, p, q) ;
    q = n > p ? n++ : p++ ;
    printf ("D : n = %d p = %d q = %d\n", n, p, q) ;
}
```