Chapitre 3: Opérateurs et expressions en C

3. Opérateurs et expressions en C.

3.1. L'affectation(1)

- En C, l'affectation est un opérateur à part entière. Elle est symbolisée par le signe =. Sa syntaxe est la suivante :

variable = expression

- Un opérateur est un symbole qui permet de manipuler une ou plusieurs variables pour produire un résultat.

- Une expression est un calcul qui donne une **valeur** comme **résultat.**
- Il peut être une valeur, une variable ou une opération constituée par des **valeurs**, des **constantes** et des **variables** reliées entre eux par des **opérateurs**.

```
const float PI=3.14;
1, a+ 3*b-c, …
b+2, a= 5*y+PI*(sin(x)-2);
```


- Toute expression suivie d'un **point virgule** devient une **instruction**

3. Opérateurs et expressions en C.

3.1. L'affectation(2)

- L'affectation effectue une conversion de type implicite : la valeur de l'expression (terme de droite) est convertie dans le type du terme de gauche.
- Par exemple, le programme suivant:

```
void main() { int i, j = 2;
      float x = 2.5;
       i = j + x;  /* 4  */
       x = x + i; /* 6.5 imprime pour x la valeur 6.5 (et non 7), car dans
l'instruction  i = j + x;, l'expression  j +x  a été convertie en entier*/
      }
```

- On peut enchaîner des affectations. L'évaluation commence de la **droite** vers la **gauche**.

```
b=(a = 5 + 3)+1  =>  a = 8   et b = 9  ;
a = b = c = d   équivalente :          a = (b= (c = d))
```

3. Opérateurs et expressions en C.

3.1. L'affectation(3) : conversion automatique

- Si un opérateur a des opérandes de différents types, les valeurs des opérandes sont converties automatiquement dans un type commun.
- Règle de conversion, lors d'une opération avec :
  - deux **entiers** : les types ***char*** et ***short*** sont convertis en ***int***. Ensuite, il est choisit le plus large des deux types dans l'échelle : ***int***, ***unsigned int***, ***long***, ***unsigned long***. ***char et short =>int***
  - un ***entier*** et un ***réel*** : le type entier est converti dans le type du ***réel***.
  - deux ***réels*** : il est choisit le plus large des deux types selon l'échelle : ***float***, ***double***, ***long double***. ***int -> long -> float -> double -> long double***
  - Dans une affectation : le résultat est toujours converti dans le type de la ***destination***. Si ce type est plus faible, il peut y avoir une ***perte*** de précision.

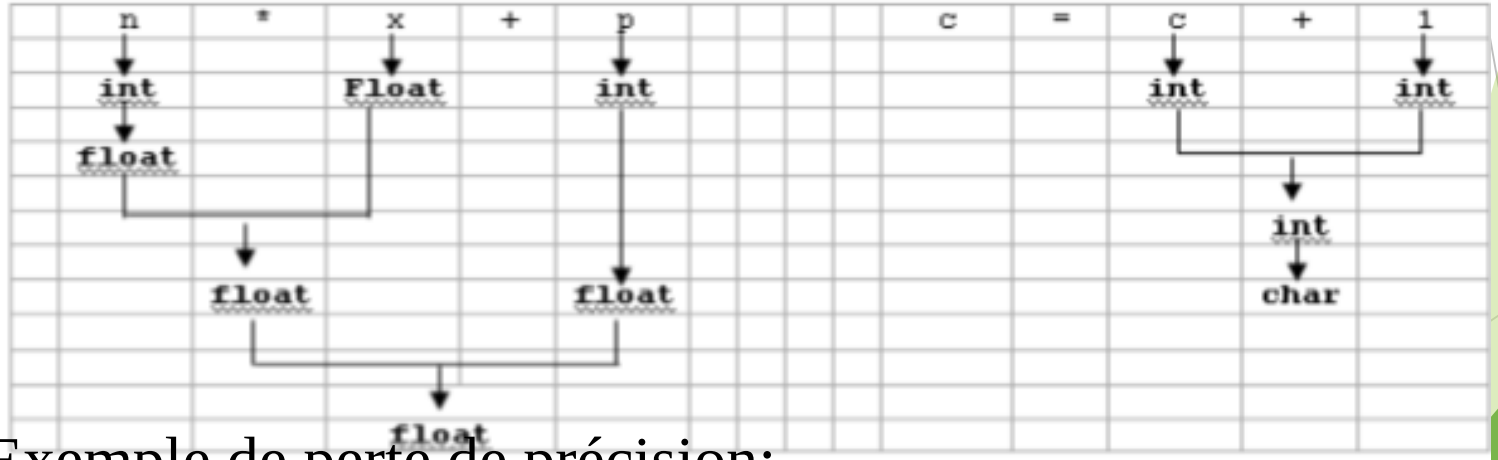
3. Opérateurs et expressions en C.

3.1. L'affectation(3) : conversion automatique

```
int n, p;
float x;
char c;
```

Donner le type de chacune des expressions suivantes

```
n*x + p  ;      c = c + 1
```



- Exemple de perte de précision:

```
unsinged int a=5000000000; /* valeur de a sera -589934592
accepté par le compilateur sans avertissement */
```

De 0 à +4 294 967 295

3. Opérateurs et expressions en C.

3.1. Affectation (4) :Conversion forcée(casting)

L'opérateur de conversion de type permet de modifier explicitement le type d'un objet. On écrit ***(type) objet***

Par exemple:

```
 void main()
{
   int i = 3, j = 2;

   printf("%f \n",(float)i/j);/* i devient 3.0
                                 =>3.0/2 => 3.0/2.0 => 1.5 */
}
```

retourne la valeur 1.5.

3. Opérateurs et expressions en C.

3.2. Les opérateurs arithmétiques

❑ L'opérateur **unaire** (– changement de signe) ainsi que les opérateurs **binaires**: (+ addition) (– soustraction) (***** multiplication) (/ division) (**%** reste de la division (modulo)).

❑ Le C ne dispose que de la notation / pour désigner à la fois la division entière et la division entre flottants.

❑ Si les deux opérandes sont de type entier, l'opérateur / produira une division entière (quotient de la division). Par contre, il délivrera une valeur flottante dès que l'un des opérandes est un flottant. Par exemple:

float x;
x = 3 / 2; affecte à x la valeur 1.
Par contre x = 3 / 2. ; affecte à x la valeur 1.5.

3. Opérateurs et expressions en C.

3.3. Les opérateurs relationnels

> strictement supérieur
>= supérieur ou égal
< strictement inférieur
<= inférieur ou égal
== égal
!= différent

Exemple : (3 ==2) retourne la valeur 0.

- Leur syntaxe est: ***expression-1 op expression-2***

Les deux expressions sont évaluées puis comparées. La valeur rendue est de type int (il n'y a pas de type booléen en C); elle vaut 1 si la condition est vraie, et 0 sinon.

➔Attention à ne pas confondre l'opérateur de test d'égalité == avec l'opérateur d'affection=.

3. Opérateurs et expressions en C.

3.4. Les opérateurs logiques booléens

➔ && et logique
➔ || ou logique
➔ ! négation logique

❑ Comme pour les opérateurs de comparaison, la valeur retournée par ces opérateurs est un **int** qui vaut 1 si la condition est vraie et 0 sinon.

❑ Dans une expression de type

expression-1 op-1 expression-2 op-2 ...expression-n

L'évaluation se fait de gauche à droite et s'arrête dès que le résultat final est déterminé.

Exemple: `3 && 4 vaut 1 ; !65.34 vaut 0 ; !!0 vaut 0`

3. Opérateurs et expressions en C.

3.5. Les opérateurs d'affectation composée

❑Les opérateurs d'affectation composée sont:

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

Pour tout opérateur *op, l'expression*

expression-1 op expression-2

est équivalente à

expression-1 = expression-1 op expression-2.

Exemple: `a = a + b` s'écrit `a += b` `n = n << 2` s'écrit `n <<= 2`

3. Opérateurs et expressions en C.

3.6. Les opérateurs d'incrémentation et de décrémentation

Les opérateurs d'incrémentation ++ et de décrémentation -- s'utilisent aussi bien en suffixe (i++) qu'en préfixe (++i). Dans les deux cas la variable i sera incrémentée, la notation suffixe la valeur retournée sera l'ancienne valeur de i alors que dans la notation préfixe se sera la nouvelle. Par exemple,

```
int a = 3, b, c,d;
b = ++a; /* a et b valent 4:   1)incrémente a
                               2)puis passe sa valeur à b*/
c = b++; /* c vaut 4 et b vaut 5:  1)passe la valeur de b à c
                                    2) incrémente b */
c =--a;  /* c=3, a=3*/
d = a--; /*d=3, a=2*/
```

3. Opérateurs et expressions en C.

3.7. L'opérateur virgule

Une expression peut être constituée d'une suite d'expressions séparées par des virgules : ***expression-1, expression-2, ... , expression-n***

➔Cette expression est alors évaluée de gauche à droite. Sa valeur sera la valeur de l'expression de droite.

➔Par exemple, le programme:

```
main()
{
int a, b;
b = ((a = 3), (a + 2));
printf("\n b = %d \n",b);
}
```

imprime b = 5.

3. Opérateurs et expressions en C.

3.8. L'opérateur conditionnel ternaire

L'opérateur conditionnel ? est un opérateur ternaire. Sa syntaxe est la suivante : ***condition ? expression-1 : expression-2***

Cette expression est égale à ***expression-1*** si la condition est satisfaite, et à ***expression-2*** sinon.

Par exemple, l'expression

x >= 0 ? 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.

3. Opérateurs et expressions en C.

3.8. L'opérateur conditionnel ternaire

L'opérateur conditionnel ? est un opérateur ternaire. Sa syntaxe est la suivante : ***condition ? expression-1 : expression-2***

Cette expression est égale à ***expression-1*** si la condition est satisfaite, et à ***expression-2*** sinon.

Par exemple, l'expression

x >= 0 ? 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.

3. Opérateurs et expressions en C.

3.11. Opérateurs de bits

- Les opérateurs de manipulation de bits sont : | , ^ , & , << et >>
- Les opérandes doivent être de type entier (char, short, int, long, signés ou non).

TYPE	OPÉRATEUR	SIGNIFICATION
binaire	&	ET (bit à bit)
	|	OU inclusif (bit à bit)
	^	OU exclusif (bit à bit)
	<<	Décalage à gauche
	>>	Décalage à droite
unaire	~	Complément à un (bit à bit)

Exemple:

	Bin	Hex	Dec
n	0000010101101110	056E	1390
p	0000001110110011	03B3	947
n & p	0000000100100010	0122	290
n | p	0000011111111111	07FF	2047
n ^ p	0000011011011101	06DD	1757
~ n	1111101010010001	FA91	-1391
int n=12;	1100	C	12
n << 2	11	3	3
n >> 3	11000	18	24

3. Opérateurs et expressions en C.

3.12. priorités de tous les opérateurs

CATEGORIE	OPERATEURS	ASSOCIATIVITE
référence	() [] -> .	--->
unaire	+ - ++ -- ! ~ * & (cast) sizeof	<--
arithmétique	* / %	--->
arithmétique	+ -	--->
décalage	<< >>	--->
relationnels	< <= > >=	--->
relationnels	== !=	--->
manip. de bits	&	--->
manip. de bits	^	--->
manip de bits	|	--->
logique	&&	--->
logique	||	--->
conditionnel	? :	--->
affectation	= += -= *= /= %= &= ^= |= <<= >>=	<-- a=b+c
séquentiel	,	--->

Exemple:

- La multiplication a la priorité sur l'addition
- La multiplication et l'addition ont la priorité sur l'affectation.
- Dans une expression, les parenthèses forcent la priorité.

Exercices d'application

1. Déterminer les valeurs de res1, res2 et res3:

```
int n1, n2;
int res1, res2, res3;
n1 = 3;
n2 = 4;
res1 = n2%n1;   n2/n1= 1,
res2 = 45%4;   1
res3 = 45%n1; 0
```

2. Déterminer les valeurs successives de k et x:

```
int i=10, j=4, k;  k ???
float x;  ??
k=i+j;   14
x=i;    10.0
x=i/j;    2.0
x=(float)i/j;   2.5
x=5/i; 0.0
k=i%j;  2
x=105;    i=5;
x= x+i;  110.0
```

3. Chercher et expliquer programme :

```
const int X=10;
int y,z;
y+z = 10;
10 = z;  eerrr
X=100; errr
z==10;
```

Chapitre 4: Les entrées sorties conversationnelles en C.

4. Les entrées sorties conversationnelles en C

Il s'agit des fonctions de la librairie standard **stdio.h** utilisées avec les unités classiques d'entrées-sorties, qui sont respectivement le clavier et l'écran.

- ❑ Lecture des caractères: **getchar.**
- ❑ Ecriture des caractères: **putchar.**
- ❑ Entrée des données: **scanf.**
- ❑ Sortie des données: **printf.**

N.B: Un programme écrit en C s'exécute séquentiellement, c'est à dire instruction après instruction.

4. Les entrées sorties conversationnelles en C

- **4.1.Lecture des caractères:** La fonction externe **getchar** a pour rôle de lire un caractère en entrée(clavier) et fournir le caractère après confirmation par la touche "entrée ". Syntaxe : var_car =**getchar()**

Exemple:
```
char a;
a=getchar();      /* équivalent à  scanf("%c",&a) ;  /*
```

- **4.2Ecriture des caractères**: le rôle de la fonction externe **putchar** est de permettre l'affichage des caractères à l'écran. Syntaxe: **putchar(var_car)**

Exemple:
```
char c='S';  putchar(c); /* printf(« %c »,c);*/
putchar(65); //  affiche  A
```

RQ: ***getchar*** resp(***putchar***) retourne le caractère(entier entre 0 et 255) lu resp(écrit), ou bien la valeur -1 (EOF) en cas d'erreur.

4. Les entrées sorties conversationnelles en C

4.3. Entrée des données (Lire): le rôle de la fonction externe scanf est de permettre la lecture des données en entrée. Ces données peuvent être de différents type: numériques, caractères simples, chaînes de caractères…

- Syntaxe: ***scanf(chaîne de format, arg1, arg2,….,argn)***
- La chaîne de format se compose de groupes de caractères de format associés individuellement à chacune des données en entrée.
- Chaque groupe de caractère de format est constitué de % suivi d'une spécification de forme indiquant le type de la donnée à laquelle elle est associée(voir tableau).

4. Les entrées sorties conversationnelles en C

4.3. Entrée des données (Lire):

Format	Type de correspondant	Signification
%c	char	Caractère simple
%d ou %i	int	Entier décimal
%hd	short int	Entier court
%hu	unsigned short	Entier court non signé
%ld	long int	Entier long
%lu	unsigned long	Entier long non signé
%f	float	Virgule flottante
%lf	double	Virgule flottante
%e	float ou double	Notation exponentielle(e)
%E	float ou double	Notation exponentielle (E)
%s	chaîne de caractères	Chaine de caractères
%u ou %U	unsigned int	Entier décimal non signé
%X ou %x	int	Entier hexadécimal (A-F ou a-f)
%0	int	Entier octal

N.B: Le type données argument doit être cohérent avec la chaîne de format associée.

4. Les entrées sorties conversationnelles en C

4.3. Entrée des données (lire) :

- arg1, arg2,…,argn représentent les différentes données figurent en entrée. En fait, ces arguments indiquent les adresses des données en mémoires (&variable).

- Tous les noms de variables à l'exception des tableaux doivent être précédés par le symbole &.

4. Les entrées sorties conversationnelles en C

4.3. Entrée des données (lire) :

Exemple 1:

```
int a;
scanf("%d", &a);
```

Exemple 2:

```
float x;
scanf("%f", &x);
```

Type et Variable	Affichage	Format

short i;
int j;
long k;
	scanf("%hd",&i);
	scanf("%d",&j);
	scanf("%ld",&k);
		décimal

short i;
int j;
long k;
	scanf("%ho",&i);
	scanf("%o",&j);
	scanf("%lo",&k);
		octal

short i;
int j;
long k;
	scanf("%hx",&i);
	scanf("%x",&j);
	scanf("%lx",&k);
		hexadécimal

float x;
double y;
long double z;
	scanf("%f",&x);
	scanf("%lf",&y);
	scanf("%Lf",&z);
		décimal (avec point)

float x;
double y;
long double z;
	scanf("%e",&x);
	scanf("%le",&y);
	scanf("%Le",&z);
		exponentiel

char c;
char m[10];
	scanf("%c",&c);
	scanf("%s",m);
	scanf("%s",&m[0]);
		caractère
		chaîne de caractères

N.B: Lors de l'évaluation des données, scanf s'arrête si la chaine de format a été traitée jusqu' la fin ou si une donnée ne correspond pas au format indiqué. scanf retourne comme résultat le nombre(entier) d'arguments correctement reçus et affectés, qui peut être récupéré ou non.

k = scanf("valeurs %d et %d",&i,&j); la valeur de **k**, vaut le nombre de variables scannées.

4. Les entrées sorties conversationnelles en C

4.4. Sortie des données(Ecrire): La fonction **printf** permet d'écrire des données en provenance de l'ordinateur sur l'unité de sortie standard(Ecran). Les données peuvent être de différents types: numérique, caractères simples ou chaînes…

- **Syntaxe: int printf(chaîne de format, arg1, arg2,…,argn);**
- Chaîne de format fait référence à une chaine contenant des informations précisant les caractéristiques d'affichage
- arg1, arg2,…,argn représentent les différentes données à afficher: constantes, variables, tableaux, expressions.
- Contrairement à la fonction scanf, les arguments ne représentent pas des adresses mémoire.

4. Les entrées sorties conversationnelles en C

4.4. Sortie des données(Ecrire):

- La chaîne de format *contient :*
  - des constantes chaines comme A, B, ….,Z, a, b, …, z, é, ç, à, è, …,0, 1, …9
  - des séquences d'échappement comme \n, \t,\b, \a, \r, \v, \', \\, … qui représentent des caractères spéciaux (cf. le type caractère).
  - Des codes de format comme %d, %i, %e, %f, %c, %s, %ld, %lf,…
  - Le code de format se compose de groupes de caractères commençant par % et suivi de spécification de format(voir tableau ci-dessous)

Format	Type de correspondant	Signification
%c	char	Caractère simple
%d ou %i	int	Entier décimal
%hd	short int	Entier court
%hu	unsigned short	Entier court non signé
%ld	long int	Entier long
%lu	unsigned long	Entier long non signé
%f	float	Virgule flottante
%lf	double	Virgule flottante
%e	float ou double	Notation exponentielle(e)
%E	float ou double	Notation exponentielle (E)
%s	chaîne de caractères	Chaine de caractères
%u ou %U	unsigned int	Entier décimal non signé
%X ou %x	int	Entier hexadécimal (A-F ou a-f)
%0	int	Entier octal

4. Les entrées sorties conversationnelles en C

4.4. Sortie des données(Ecrire):

- **printf("%4d et %5d",i,j);** affiche i et j en insérant, devant, des espaces de façon à ce que les espaces plus l'affichage du i (resp. du j) prennent 4 caractères (resp. 5 caractères).

- **printf("%6.2f %.3Lf",x,y);** affiche x avec 2 chiffres après la virgule, mais en insérant des espaces devant afin que le tout fasse 6 caractères. Tandis que le variable y de type long double sera simplement affichée avec 3 chiffres après la virgule, sans spécification de la longueur totale.

Type et Variable	Affichage	Format
`short i; int j; long k;`	`printf("%hd %d %ld",i,j,k);`	décimal
`short i; int j; long k;`	`printf("%ho %o %lo",i,j,k);`	octal
`short i; int j; long k;`	`printf("%hx %x %lx",i,j,k);`	hexadécimal
`float x; double y;`	`printf("%f %lf",x,y);`	décimal (avec point)
`float x; double y;`	`printf("%e %le",x,y);`	exponentiel
`float x; double y;`	`printf("%g %lg",x,y);`	le plus court des deux
`long double z;`	`printf("%Lf",z);`	décimal (avec point)
`long double z;`	`printf("%Le",z);`	exponentiel
`long double z;`	`printf("%Lg",z);`	le plus court des deux
`char c;`	`printf("%c",c);`	caractère
`char m[10];`	`printf("%s",m);`	chaîne de caractères

4. Les entrées sorties conversationnelles en C

4.4. Sortie des données :

```
printf("%4d", 12);     ==> __12
printf("%4d", 1234);   ==> 1234
printf("%4d", 1234567);          ==> 1234567
printf("%4c", 65);     ==> ___A

printf("%.1f", 1.0);             ==> 1.0
printf("%.2f", 1.0);             ==> 1.00
printf("%.3f", 1.0);             ==> 1.000
printf("%.3f", 35.47825);        ==> 35.478
```

Caractère de séparation en lecture :
Espace, tabulation '\t' ou retour chariot '\n'

```
printf("%5.1f", 1.0);            ==> __1.0
printf("%5.2f", 1.0);            ==> _1.00
printf("%5.3f", 1.0);            ==> 1.000
printf("%5.3f", 5542.45310);     ==> 5542.453
```

❑ ***Exemple de scanf :***

```
int jour, mois, annee, recu ;
recu = scanf("%i %i %i", &jour, &mois, &annee) ;
```

Donnée introduite	Affichage à l'éctran			
	recu	jour	mois	annee
11 6 1987	3	11	06	1978
11/6/1978	1	11	indéfinie	indéfinie
11.4 1978	1	11	indéfinie	indéfinie
11 4 19.78	3	11	4	19

instructions	Données saisies	Résultats
int A,B; scanf("%4d %2d", &A, &B);	1234567	A=1234 B=56

Le chiffre 7 sera gardé pour la prochaine instruction de lecture.