Exercice langage C, série 3.1

Opérateurs

Exercice 1 : Opérateurs d'assignement

a) Réécrire les instructions suivantes en utilisant des opérateurs d'affectation composés.

```
memoire = memoire + 1;            memoire+=1; memoire++;
mot = mot * 8;                    mot*=8;
monnaie = monnaie-1;              monnaie-=1;
                                  monnaie--; --monnaie;
a = a % b;                        a%=b;
part = part / nbre_personne;      part/= nbre_personne;
```

b) Quelle sont les valeurs des variables `x,y,z`?

```
int x = 10;
int y, z;
x *= y = z = 4
x=40 y=4 z=4
```

Exercice 2 : Opérateurs arithmétiques

a) Addition et division : Quelle est la valeur de la variable x?

```
int n = 5, p = 9;
float x;

x = p / n ;                    -> 1.000000
x = (float) p / n ;            -> 1.800000
x = (p + 0.5) / n ;            -> 1.900000
x = (int) (p + 0.5) / n ;      -> 1.000000
```

b) Modulo : Quelle est la valeur de la variable a?

```
int a;
a = 10 % 10; -> 0
a =  5 % 10; -> 5
a = 10 %  0; -> le programme plante (division par 0)
a =  0 % 10; -> 0
```

Exercice 3 : Opérateurs d'incrémentation

a) Quels sont les valeurs des variables x, y et z?

```
int x = 2, y = 1, z = 3;
z += -x++ + ++y;
 z= 3 + -(2++) + ++(1)
 x++ => x=x+1 => x=3      x affecté avant incrémentation
 ++y => y=y+1 => y=2      y affecté après incrémentation
 z= 3 + ( -2 + (2) )-> 3+0 = 3
 -> x=3 y=2 z=3
```

Exercice 4 : Opérateurs de comparaison et de logique

a) Evaluer les expressions suivantes :

a)	`3 <= 2-1`	**-> false**	d)	`true && false || true`	**-> true**
b)	`3 * (4 > 4) + 2.5`	**-> true**	e)	`3 + (n > n - 1)`	**-> true**
c)	`(4 == 5.1) / 2 + 1`	**-> true**	f)	`!true && false || !false`	**-> true**

b) Assignez à une variable booléenne b. La valeur est vraie si la variable x est comprise strictement entre 5 et 10, fausse sinon.

$$b = \begin{cases} \text{vrai si } x \in [5..10] \\ \text{faux sinon} \end{cases}$$

```
b = (5<=x) && (x<=10);
```

c) Assignez à une variable booléenne b. La valeur est vraie si la variable x n'est pas comprise strictement entre 5 et 10, fausse sinon.

$$b = \begin{cases} \text{vrai si } x \notin [5..10] \\ \text{faux sinon} \end{cases}$$

```
b = (5>x) || (x>10);
```

Exercice 5 : Divers opérateurs

a) Que vaut `b` ?

```
int n = 5, p = 9;
_Bool b;                    int        _Bool
b = n < p ;                 -> 1       -> true
b = n == p ;                -> 0       -> false
b = p % n + (p > n) ;       -> 5       -> true
```

b) Quels résultats fournit le programme suivant ?

```
int   n = 10, p = 5, q = 10, r ;
r = n == (p = q) ;          r: 1   n: 10   p: 10   q: 10
n = p = q = 5;              r: 1   n:  5   p:  5   q:  5
n += p += q ;               r: 1   n: 15   p: 10   q:  5
```

c) Donner les résultats des calculs suivants et en préciser le type :

a)	`3.5 * 6`	**→double 21.00000**	d)	`(int)85.35 * 2`	**→int 170**
b)	`243 * -83`	**→int -20169**	e)	`'w' - 2024`	**→int -1905**
c)	`(int)(20.35 * 24)`	**→int 488**	f)	`(char)(23 / 9)`	**→char 2**

d) Quel doit être le type de **`var`** pour que les affectations suivantes ne provoquent pas d'erreur d'arrondi :

a)	`var = 12 + 'g';`	**int**	d)	`var += 2.5;`	**double**
b)	`var = (1 < 3);`	**_Bool**	e)	`var = 255 + 1;`	**int**
c)	`var = 13 - 274.3;`	**double**	f)	`var = 2 / 7. ;`	double

e) Donner la valeur de la variable **`x`** (de type **`int`**) après chaque instruction de la séquence de programme suivante (pour chaque instruction, la valeur de **`x`** est celle calculée à l'instruction précédente) :

```
x = 2;                          ->   2
x = 3 + (3 > x);                ->   4
x += x -= 2;                    ->   4
x = (++x - 6) * 3;              ->  -3
x *= (5 > x) * (3 + 23);        -> -78
```

f) Dans les expressions suivantes repérer les parenthèses inutiles.

a) `a = (x * w) + 3;` **Inutiles**: *, puis + et = en dernier

b) `f *= 15 - (3 + f);` **Nécessaires,** sinon 15-3 est fait avant
c) `g = (g++ + g) * 3;` **Nécessaires,** sinon g*3 est fait en premier
d) `m = k > (b || 1);` **Nécessaires:** > a une priorité plus grande que ||
e) `h = (n += 12);` **Inutiles**: associativité de droite à gauche
f) `b *= (20 + 3);` **Inutiles** + à une priorité plus grande que *=

pour (a), (e) et (f), la parenthèse ne change pas l'ordre découlant des priorités.

Exercice 6 : Opérateurs bit à bit

a) Soit la variable x ci-dessous

```
unsigned char x=0xAA; // 1010 1010
```

a. Ecrivez un programme qui met les 2 bits de poids forts à 1 et les deux bits de poids faible à 0 de la variable x. Le résultat est :

1	**1**	**1**	**0**	1	0	**0**	**0**

```
unsigned char x=0xAA; // 1010 1010

x= x | 0xC0;  // Mise à 1 des 2 bits de poids fort
x= x & 0xFC;  // Mise à 0 des 2 bits de poids faible
```

ou

```
x |= 0xC0;
x &= 0xFC
```

b. Ecrivez un programme qui affiche le contenu des bits 3, 4 et 5 de la variable x

1	0	**1**	**0**	**1**	0	1	0

```
printf("%X\n", (x >> 3) & 0x07 );
```

b) Soit la déclaration de variable suivante

```
unsigned int x=0x03020100;
```

Décomposez la variable x en 4 variables `b0`, `b1`, `b2`, `b3` contenant les 4 bytes de `x` et affichez-les.

```
unsigned int x=0x03020100; // 1010 1010
unsigned char b0,b1,b2,b3;
unsigned int mask = 0xFF;

b0 = ( ( x & mask ) );

mask <<= 8;
b1 = ( ( x & mask ) >> 8 );

mask <<= 8;
b2 = ( ( x & mask ) >> 16 );

mask <<= 8;
b3 = ( ( x & mask ) >> 24 );
```

Exercice 7 : Débordement (**AVANCÉ**)

On demande de décrire le programme suivant.

```
printf("INT_MAX: %d  INT_MAX+1: %d\n",INT_MAX,INT_MAX+1); /* Débordement */
printf("DBL_MAX: %g\n",      DBL_MAX);
printf("DBL_MAX+1000: %g\n", DBL_MAX+1000); /*NE FAIT RIEN */
printf("DBL_MAX*2   : %g\n", DBL_MAX*2);    /*OVERFLOW */
printf("q = 1/0: %f  \n", 1/0); /* LE PROGRAMME PLANTE */
float q  =  1.0f/0;
printf("q = 1.0f/0: %f  \n", q); /* VALEUR INFINIE */
printf("1/q       : %f  \n", 1/q); /*1/INFINI =0 */
printf("q/q       : %f  \n", q/q); /*INFINI/INFINI =INDERTERMINE */
printf("sqrt(-1)  : %f  \n", sqrt(-1)); /*VALEUR INDERERMINEE */
```

```
INT_MAX     : 2147483647  INT_MAX+1: -2147483648
DBL_MAX     : 1.79769e+308
DBL_MAX+1000: 1.79769e+308
DBL_MAX*2   : 1.#INF
q = 1.0f/0  : 1.#INF00
1/q         : 0.000000
q/q         : -1.#IND00
sqrt(-1)    : -1.#IND00
```