Exercice 1 : Opérateurs de relation et opérateurs logiques

Considérez l'expression suivante :

```
double x, y;
int z = x >= 0 && x <= 20 && y > x || y == 50 && x == 1 * 2 || y == 30 + 30;
```

(a) Ajouter à l'expression suivante toutes les parenthèses montrant l'ordre d'exécution des opérations.

```
double x, y;
int z = ((((x >= 0) && (x <= 20)) && (y > x)) ||
((y == 50) && (x == (1 * 2))) || (y == (30 + 30)));
```

(b) Quelle est la valeur de `z` évaluées avec les valeurs suivantes ?

i. `x = -1.0; y = 60.0;`

i. « `(int)1` »

ii. `x = 0.0 ; y = 1.0;`

ii. « `(int)1` »

iii. `x = 19.0 ; y = 1.0;`

iii. « `(int)0` »

iv. `x = 0.0 ; y = 50.0;`

iv. « `(int)1` »

v. `x = 2.0 ; y = 50.0;`

v. « `(int)1` »

vi. `x = -10.0 ; y = 60.0;`

vi. « `(int)1` »

Exercice 2 : Cas particuliers

(a) Que vaut `i` ?

(a) « 32768 »

```
uint16_t i = 32767;
i++;
```

(b) Que vaut `i` ?

(b) « 0. »

```
int16_t i = 0;
--i; i--; i++; ++i;
```

(c) Que vaut `i` ?

(c) « D68 »

```
short i = 'A' > 'B' ? 'C' : 'D';
```

(d) Que valent `i`, `j` et `k` ?

(d) « i1j0k5 »

```
short i = 0, j = 1, k;
k = (k = 5, i++) >= j ? i++ : --j;
```

(e) Que valent `i`, `j` et `k` ?

(e) « **i3j1k8** »

```
short i = 2, j = 1, k;
k = i >= j << 1 ? i++ << 2 : --j << 3;
```

Exercice 3 : Calcul de masques

Que vaut `m` en binaire dans les cas suivants ?

```
char m, n = 2;
char d = 0x55, e = 0xaa;
```

(a) `m = 1 « n;` — []{w=5cm}

(a) « 0b00000100 »

(b) `m = ~(1 « n);`

(b) « 0b11111011 »

(c) `m = d | (1 « n);`

(c) « 0b01010101 »

(d) `m = e | (1 « n);`

(d) « 0b10101110 »

(e) `m = d ^(1 « n);`

(e) « 0b01010001 »

(f) `m = e ^(1 « n);`

(f) « 0b10101110 »

(g) `m = d & ~(1 « n);`

(g) « 0b01010001 »

(h) `m = e & ~(1 « n);`

(h) « 0b10101010 »

Exercice 4 : Oneliners

Résoudre les problèmes suivant avec une seule ligne de code.

(a) Mettre à 1 (*set*) le n-ième bit de la variable `x`.

```
x |= 1 « n;
```

(b) Mettre à 0 (*clear*) le n-ième bit de la variable `x`.

```
x &= ~(1 << n);
```

(c) Inverser (*toggle*) le n-ième bit de la variable `x`.

```
x ^= 1 << n;
```

Exercice 5 : Programmation

Écrire une fonction en C qui inverse les chiffres d'un entier. Utilisez le prototype suivant :

```
int32_t reverse(int32_t num);
```

Exemple : `123` donne `321` et `208478933` donne `339874802`.

```
int32_t reverse(int32_t num) {
  int32_t reversed = 0;

  while (num != 0) {
    // Extraire le dernier chiffre
    int32_t digit = num % 10;

    // Ajouter ce chiffre à la valeur retournée
    reversed = reversed * 10 + digit;

    // Retirer le dernier chiffre
    num /= 10;
  }

  return reversed;
}
```