Exercice 1 : Arithmétique et expressions

Considérez les déclarations suivantes :

```
char c = 3;     short s = 7;
int i = 3;      long l = 4;
float f = 3.3; double d = 7.7;
```

Pour chacune des expressions ci-dessous, indiquez leur type et leur valeur. Par exemple l'expression `2 / 3 * c` donne `(int)0` car `2 / 3` est égal à `0` en division entière, et comme l'opération passe par l'ALU, le resultat est promu en `int` avant d'être multiplié par `c`.

(a) `c / 2`

(a) « **(int)1** »

(b) `s + c / 10`

(b) « **(int)7** »

(c) `l + i / 2.0`

(c) « **(double)5.5** »

(d) `d + f`

(d) « **(double)11** »

(e) `(int)d + f`

(e) « **(float)10.3** »

(f) `(int)d + l`

(f) « **(long)11** »

(g) `c << 2`

(g) « **(int)12** »

(h) `s & 0xf0`

(h) « **(int)0** »

(i) `s && 0xf0`

(i) « **(int)1** »

(j) `d + f == s + l`

(j) « **(int)0** »

Exercice 2 : Analyse de code

Dans chacune des structures de contrôle ci-dessous, indiquer la nature de l'erreur.

(a)

```
double x = 100.0;
size_t i = 0;
do
    x = x / 2.0;
    i++;
while (x > 1.0);
```

> Il manque les accolades autour du bloc `do..while`.

(b)

```
long x = 100; if (x = 0) printf("Erreur : la valeur 0 est interdite !\n");
```

Le test d'égalité utilise l'opérateur `==`. L'opérateur d'affectation `=` n'est pas valable.

(c)

```
double x = 100.0;
switch (x) {
    case 0:
        printf("x est nul.\n");
        break;
    default:
        print("OK.\n");
}
```

L'instruction `switch` n'est pas applicable à un type à virgule flottante.

(d)

```
for (int i = 0; i < 10; i++); {
    printf("%d\n", i);
}
```

Le point virgule à la fin de l'instruction termine cette dernière. Le bloc formé des accolades n'appartient pas à la boucle.

(e)

```
int i = 0;
while i < 100
{
    printf("%d\n", ++i);
}
```

Il manque des parenthèses autour de la condition de l'instruction `while`.

Exercice 3 : Lecture de code

On s'intéresse ici au passage par adresse. Observez le programme suivant et indiquez ce que vous voyez sur la sortie standard.

```c
#include <stdio.h>
#include <stdlib.h>

int test(int a, int *b, int *c, int *d) {
    a = *b;
    *b = *b + 5;
    *c = a + 2;
    d = c;
    return *d;
}

int main() {
  int a = 0, b = 100, c = 200, d = 300, e = 400;
  e = test(a, &b, &c, &d);
  printf("%05d %d %d %d %d", a, b, c, d, e);
}
```

00000 105 102 300 102

Exercice 4 : Programmation

(a) Écrire une fonction qui reçoit une chaîne de caractère en paramètre et qui retourne vrai si la chaîne est un palindrome, c'est à dire qu'elle se lit de la même manière de gauche à droite et de droite à gauche. Utiliser la syntaxe pointeur pour le paramètre chaîne.

```c
int is_palindrome(const char *str) {
  size_t len = strlen(str);
  for (size_t i = 0; i < len / 2; i++)
      if (str[i] != str[len - 1 - i])
          return 0;
  return 1;
}
```

(b) Écrire une fonction qui reçoit en paramètre un **tableau d'entiers** et qui retourne la position de la **première** occurrence d'une valeur dans ce tableau, ou `-1` si la valeur n'est pas présente. Utiliser la syntaxe pointeur pour le paramètre tableau.

```c
int index(int *array, size_t size, int value) {
  for (int i = 0; i < size; i++)
      if (array[i] == value)
          return i;
  return -1;
}
```

(c) Écrire une fonction qui calcul la longueur totale des segments de droite dont les points sont reçus en paramètre. Les données sont un tableau composé de N par 2. Les indices du sous tableau sont les coordonnées X et Y des points.

```
double distance(double x1, double y1, double x2, double y2) {
  return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}

double length(double array[][2], size_t size) {
  double length = 0;
  for (int i = 0; i < size - 1; i++)
      length += distance(
          array[i][0], array[i][1],
          array[i + 1][0], array[i + 1][1]
      );
  return length;
}
```