Identification /5

Prénom: .....

Nom: .....

Code permanent: .....

Veuillez choisir **une seule réponse** par question, avec une croix (×) ou un crochet (✓).
Les deux premières questions sont basées sur les déclarations suivantes:

```
int a;
int* b = &a;
int** c = &b;
```

Q1. Parmi les expressions suivantes, laquelle n'est pas une valeur gauche (*left value*) valide à la compilation?

- ☐ `*(&a)`
- ■ `b + 1`
- ☐ `*(b + 1)`
- ☐ `*(c + 1)`

Q2. Parmi les expressions suivantes, laquelle n'est pas une valeur droite (*right value*) valide à la compilation?

- ☐ `&a + 1`
- ☐ `*b + 1`
- ■ `*(a + 1)`
- ☐ `b + 1`

Les deux questions qui suivent sont basées sur le programme suivant:

```
#include <stdio.h>

int main(void) {
  int a[] = {0, 1, 2, 3};
  int* b[] = {a + 2, a, a + 1};
  printf("%lu %p\n", sizeof(int), a);
  printf("%d\n", *(*(b + 2) + 1));
  printf("%p\n", *(b + 1) + 2);
  return 0;
}
```

Supposons qu'à l'exécution, le programme affiche

```
4 0x7ffe67040ab0
```

sur la première ligne de la sortie standard.

Q3. Qu'est-ce qui sera affiché sur la 2e ligne de la sortie standard?

- ☐ `0`
- ☐ `1`
- ■ `2`
- ☐ `3`

Q4. Qu'est-ce qui sera affiché sur la 3e ligne de la sortie standard?

- ☐ `0x7ffe67040ab0`
- ■ `0x7ffe67040ab8`
- ☐ `0x7ffe67040ab4`
- ☐ `0x7ffe67040abc`

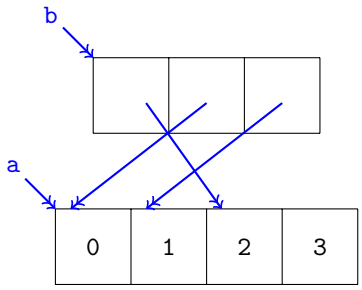
Q5. Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>

int f() {
  static int x = 0, y = 1;
  if (x % 2 == 0)
    ++x;
  else
    --y;
  return x + y;
}

int main(void) {
  for (int i = 0; i < 2; ++i)
    printf("%d ", f());
  return 0;
}
```

- ☐ `1 0`
- ☐ `1 1`
- ■ `2 1`
- ☐ `2 2`

Voir explications page suivante

Q1. Une valeur gauche est une valeur qui peut apparaître à gauche de l'opération d'affectation. L'expression `*(&a)` est bien une valeur gauche: elle est même équivalente à `a`. L'expression `*(b + 1)` aussi est une valeur gauche, dont l'adresse se trouve à `sizeof(int)` octets à droite de celle de `a`. De la même façon, `*(c + 1)` est une valeur gauche, dont l'adresse se trouve à `sizeof(int)` octets à droite de celle de `b`. Par contre, `b + 1` n'est pas une valeur gauche, car elle désigne une adresse spécifique qu'on ne peut pas modifier.
Q2. Une valeur droite est une valeur qui peut apparaître à droite de l'opération d'affectation. L'expression `&a + 1` est bien une valeur droite: elle est égale à l'adresse de `a` décalée de `sizeof(int)` octets vers la droite. L'expression `*b + 1` est aussi une valeur droite: elle est égale à la valeur contenue dans `a` à laquelle on ajoute `1`. Quant à `b + 1`, elle est aussi une valeur droite valide: elle est même égale à `&a + 1`. Finalement, `*(a + 1)` n'est pas une valeur droite valide, car `a + 1` est de type `int` et donc l'opérateur de déréférencement ne peut pas s'y appliquer.
Q3 et Q4. Il suffit de se référer au diagramme suivant:



D'une part, l'expression `*(b + 2)` est équivalente à `a + 1`, ce qui entraîne que `*(*(b + 2) + 1)` est équivalente à `*(a + 2)`, qui correspond à la valeur `2` du tableau `a`. D'autre part, de façon semblable, puisque `*(b + 1)` est équivalente à `a`, l'expression `*(b + 1) + 2` est équivalente à `a + 2`, qui contient donc l'adresse de `a` décalée de `2 * sizeof(int)` = `2 * 4` = `8` octets vers la droite. On trouve donc `0x7ffe67040ab0 + 8` = `0x7ffe67040ab8`.
Q5. Le mot réservé `static` permet d'attacher de la mémoire à une fonction qui va durer pendant toute l'exécution du programme. Au début du programme, les variables statiques `x` et `y` sont initialisées à `0` et `1`, respectivement. Au premier appel de la fonction `f`, comme `x % 2 == 0` est vraie, on incrémente donc `x` à `1`, puis on retourne la valeur `1 + 1 = 2`. Lorsque la fonction `f` est appelée une deuxième fois, les variables `x` et `y` valent toujours `1` chacune. La condition `x % 2 == 0` devient fausse, de sorte que `y` est décrémentée à `1`. On retourne donc la valeur `1 + 0 = 1`. Par conséquent, ce sont les valeurs `2 1` qui seront affichées sur la sortie standard.