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`
- ■ `&c`
- ☐ `**c`

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

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

Les deux questions qui suivent sont basées sur le programme `prog.c` suivant:

```
#include <stdio.h>

int main(int argc, char** argv) {
  printf("%c\n",
         *(*(argv + 1) + 1));
  printf("%s\n",
         *(argv + 2) + 2);
  return 0;
}
```

On compile le programme et on l'exécute comme suit:

```
$ gcc prog.c -o prog
$ ./prog alpha beta gamma
```

Q3. Qu'est-ce qui sera affiché sur la 1re ligne de la sortie standard?

- ☐ `.`
- ☐ `a`
- ☐ `e`
- ■ `l`

Q4. Qu'est-ce qui sera affiché sur la 2e ligne de la sortie standard?

- ☐ `beta`
- ☐ `lpha`
- ☐ `t`
- ■ `ta`

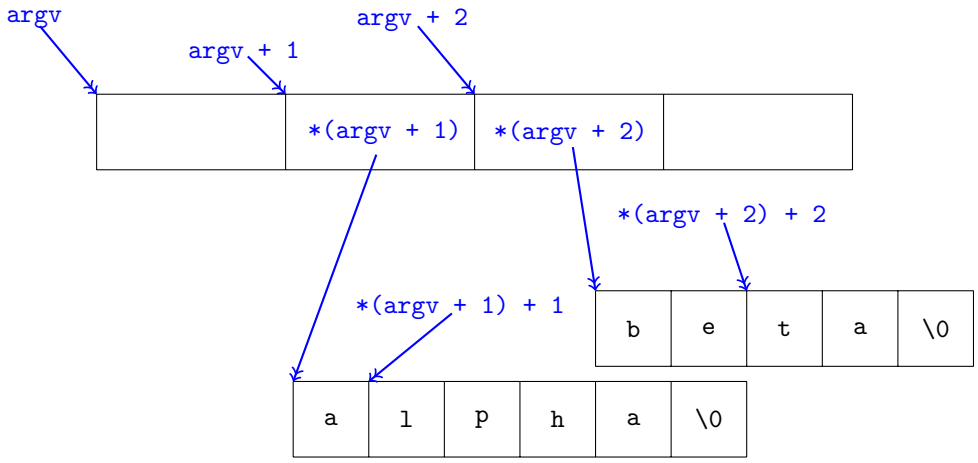
Q5. Qu'est-ce qui sera affiché sur la sortie standard si on exécute le programme suivant?

```
#include <stdio.h>

int x = 2;

int f() {
  static int x = 2;
  --x;
  return x;
}

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 équivalente à l'expression `a`, qui est bien une valeur gauche. L'expression `*b` aussi est une expression gauche, car `b` est un pointeur: on peut modifier la valeur qu'il pointe (dans le cas ici, c'est la valeur de `a` qui sera modifiée). De la même façon, `**c` est une expression gauche, car `c` est un double pointeur (dans le cas ici, c'est la valeur de `a` qui sera modifiée aussi). Finalement, `&c` n'est pas une valeur gauche: il s'agit d'une valeur (une adresse) qu'on peut lire, mais 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 `*b + 1` est une valeur droite valide, qui est la valeur pointée par `b` (c'est-à-dire la valeur contenue dans `a`) à laquelle on ajoute `1`. L'expression `&a + 1` est une valeur droite valide, qui est l'adresse à laquelle se situe la variable `a`, décalée de `sizeof(int)` octets à droite. L'expression `b + 1` est aussi une valeur droite valide, qui est l'adresse contenue dans `b`, décalée de `sizeof(int)` octets à droite. En particulier, `&a + 1` et `b + 1` sont égales. Finalement, l'expression `&(&a)` n'est pas une valeur droite valide, car `&a` n'est pas une valeur gauche: on ne peut donc pas appliquer l'opérateur `&` à `&a`.
Q3 et Q4. Il suffit de se référer au diagramme suivant:



En effet, l'expression `*(*(argv + 1) + 1)` déréférence le pointeur `*(argv + 1) + 1`, dont la valeur associée est le caractère `'l'`, alors que l'expression `*(argv + 2) + 2` contient un pointeur vers le suffixe `"ta"` de la chaîne de caractères `"beta"`.
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, la variable statique `x` est initialisée à `2`. Ainsi, au premier appel de `f`, on décrémente `x` à `1`, puis on retourne la valeur. Au deuxième appel de `f`, la variable vaut toujours `1`, est décrémentée à `0` et est retournée. Par conséquent, ce sont les valeurs `1 0` qui seront affichées sur la sortie standard.