## ❑ Exercice 1 : *Pointeurs*

On considère le programme suivant :

```
int a = 4;
int b = 1;
int c = 2;
int* p;
int* q;
p = &a;
q = &c;
*p = *q + 1;
p = q;
q = &b;
*p = *p - *q;
*q = *q + 1;
*p *= *q;
```

Compléter le tableau suivant afin de donner l'état des variables au cours de l'exécution du programme :

| | a | b | c | p | q |
|---|---|---|---|---|---|
| initialisation | 4 | 1 | 2 | ? | ? |
| `p = &a;` | 4 | 1 | 2 | &a | ? |
| `q = &c;` | | | | | |
| `*p = *q + 1;` | | | | | |
| `p = q;` | | | | | |
| `q = &b;` | | | | | |
| `*p = *p - *q;` | | | | | |
| `*q = *q + 1;` | | | | | |
| `*p *= *q;` | | | | | |

## ❑ Exercice 2 : `printf` *et* `scanf`

1. Ecrire l'instruction permettant d'afficher une variable `n` de type entier avec `printf`
2. Ecrire l'instruction permettant de saisir au clavier une variable `n` de type entier avec `scanf`
3. Expliquer la différence entre le mode de passage de `n` dans ces deux fonctions

## ❑ Exercice 3 : *Pointeurs*

On considère le programme suivant :

```
#include <stdio.h>

int main()
{
    int n = 7;
    int *p;
    *p = n;
    printf("Valeur de n =%d \n", *p);
    printf("Valeur de p= %p \n", p);
}
```

1. Ce programme est-il correct ?
2. Proposer une correction.

## ❑ Exercice 4 : *Incrémenter une variable*

La fonction suivante doit incrémenter la variable `n` donnée en argument :

```
void incremente(int x) {
    x = x + 1;
}
```

1. Commenter

2. Proposer une correction.

❑ **Exercice 5** : *Fonction modifiant un paramètre*
Ecrire en C une fonction `inverse` qui prend en argument un pointeur vers un booléen, ne renvoie rien et inverse la valeur de ce booléen (`true` devient `false` et inversement).

❑ **Exercice 6** : *Renvoyer un tableau*

```c
// Renvoie un tableau contenant les entiers de 0 à n-1
int *cree_tab_entiers(int n)
{
    int tab_entiers[n];
    for (int i = 0; i < n; i++)
    {
        tab_entiers[i] = i;
    }
    return tab_entiers;
}
```

1. Lors de la compilation de la fonction ci-dessus, on obtient l'avertissement (*warning*) suivant : « *function returns address of local variable* ». Expliquer cet avertissement.
2. Dans quelle partie de la mémoire est stockée le tableau `tab_entiers` défini à la ligne 6 ?
3. Remplacer la ligne 6 par une allocation sur le tas.

❑ **Exercice 7** : *Deux plus grandes valeurs*
On souhaite écrire une fonction en C qui prend en argument un tableau d'entiers (de taille $n \geqslant 2$) et renvoie les deux plus grandes valeurs de ce tableau.

1. Proposer une solution utilisant un type structuré que l'on définira et donner alors la signature de la fonction.
2. Proposer une solution avec une fonction ne renvoyant rien mais modifiant deux paramètres passés par adresse. Donner la signature de la fonction dans ce cas.
3. Ecrire les deux implémentations.

❑ **Exercice 8** : *Passer un pointeur*
On considère le programme suivant :

```c
#include <stdio.h>

void init_pointer(int *p, int *adr)
{
    //Initialise le pointeur p à l'adresse de val
    p = adr;
}

int main()
{
    int a = 42;
    int *p = NULL;
    init_pointer(p, &a);
    printf("Valeur pointée par p : %d\n", *p);
}
```

Ce programme produit un *warning* à la compilation : *parameter 'p' set but not used* et une erreur de segmentation à l'exécution.

1. Expliquer ces deux résultats
2. Proposer une correction afin que `init_pointer` soit conforme à sa spécification.