

*Documents de cours, de TD et de TP autorisés. Autres documents interdits.
La notation tiendra compte de la lisibilité du code et de la clarté des explications.
La barème n'est donné qu'à titre indicatif.*

Exercice 1 (5pts) Répondre par vrai ou faux aux questions suivantes. Barème: 0,5 pour une réponse correcte, -0,5 pour une réponse incorrecte et 0 en cas d'absence de réponse. Dans cet exercice, il n'est pas demandé de justification. Vrai signifie « toujours vrai », sauf lorsque la question exprime explicitement une possibilité.

Vrai Faux

1. Les deux séquences suivantes

Séquence 1: `static int x;
 x = 1;`

Séquence 2: `static int x = 1;`

sont équivalentes si elles sont placées au début d'une définition de fonction.

2. Si la taille d'un `char` est 1 octet et la taille d'un `short` est 2 octets, le bloc

```
{
    struct {char a; short b;} x;
    printf("%d\n", sizeof(x));
}
```

affiche 3.

3. Les fichiers manipulés par les fonctions de la bibliothèque standard sont terminés par un caractère spécial EOF.

4. Dans un tableau de caractères, il est possible d'écrire au delà du dernier '\0'. ...

5. L'expression `f(i++,i++)` s'évalue comme `f(1,2)` lorsque `i` vaut initialement 1. .

6. L'expression `(++i + i++)` s'évalue en 2 si `i` vaut initialement 0.

7. La séquence suivante est incorrecte en C: `char c;
 char p[10], *q;
 p = q = &c;`

8. L'expression `(f() && g())` est équivalente à l'expression `(g() && f())`.

9. Dans la fonction `main()`, une instruction `return 0;` est toujours équivalente à une instruction `exit(0);`.

10. En C ANSI, pour créer le fichier objet `essai.o` à partir du fichier source `essai.c`, il est nécessaire que toutes les fonctions utilisées dans `essai.c` soient définies avant leur utilisation ou soient des fonctions de bibliothèque.

Exercice 2 (3 pts) Écrire un programme qui lit un texte sur l'entrée standard et le réécrit sur la sortie standard, en doublant toutes les voyelles (`a` est remplacé par `aa`, `e` par `ee`, etc.).

Exercice 3 (6pts) Dans cet exercice, on demande d'écrire des *fonctions*. On ne demande pas d'écrire un programme complet, en particulier, pas de fonction `main()`.

1. Écrire une version *itérative* d'une fonction de prototype

```
int somme_cube(int n, int *tab);
```

qui calcule la somme des cubes des n premières valeurs de type `int` rangées à partir de l'adresse `tab`, c'est-à-dire $\sum_{i=0}^{n-1}(\text{tab}[i])^3$.

2. Écrire une version *réursive* de la fonction précédente.

3. Écrire une fonction `appliquer_f_sur_tabint` qui prend en argument

- un entier n ,
- un pointeur `tab` sur une zone supposée contenir n entiers (type `int`) consécutifs,
- un pointeur sur une fonction `f` prenant en argument un `int` et retournant un `int`,

et qui retourne $\sum_{i=0}^{n-1} f(\text{tab}[i])$. On pourra écrire, au choix, une version itérative ou une version réursive de cette fonction.

Exercice 4 (6pts) Détailler très précisément le comportement du programme suivant.

```
#include <malloc.h>
#include <stdio.h>

void f(int **p)
{
    **p = 2;
    *p = (int*) malloc(sizeof(int));
    **p = 3;
    p = (int**) malloc(sizeof(int*));
    *p = (int*) malloc(sizeof(int));
    **p = 4;
    // Fin de f
}

int main(void)
{
    int **p,***q,**r,*s,t;

    t = 5;
    p = (int**) malloc(sizeof(int*));
    *p = &t;
    **p = 1;
    q = &p;
    r = p;
    s = *p;

    printf("%d %d %d %d %d \n",
           ***q,**p,**r,*s,t);
    f(p);
    printf("%d %d %d %d %d \n",
           ***q,**p,**r,*s,t);
    exit(0);
}
```

Dessiner l'état de la mémoire avant chaque appel à `printf()` et expliquer son évolution au cours du programme.