

# TP 3 — rappels de C

Master première année, option EEA

<http://magphy.ujf-grenoble.fr/ratchov/eea/>

14 septembre 2004

## 1 Pointeurs vers des fonctions

À titre d'exemple d'application des pointeurs vers des fonctions nous allons construire un programme qui trie des nombres entiers. Voici le schéma d'une fonction de tri :

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N - 1; j++) {
        if (tab[j] > tab[j + 1]) {
            temp = tab[j + 1];
            tab[j + 1] = tab[j];
            tab[j] = temp;
        }
    }
}
```

tab est un tableau contenant les N entiers qu'on veut trier.

**Exercice 1** *Comprendre (et expliquer brièvement) comment marche l'algorithme de tri puis définir une fonction :*

```
void tri(int *tab, unsigned N);
```

*qui trie par ordre croissant le tableau d'entiers passé en argument. Essayez la dans un programme.*

**Exercice 2** *Faire une fonction qui trie des entiers par ordre décroissant.*

On constate que lorsqu'on change l'ordre du tri, la fonction reste pratiquement la même. Uniquement la condition du “if” a été modifiée. Il est maladroit de faire une fonction pour chaque ordre de tri. L'idée sera de passer la condition du “if” en argument à la fonction `tri`.

**Exercice 3** *Créer une fonction :*

```
int ordre_crois(int x, int y);
```

*qui retourne 1 si  $x \leq y$  et 0 sinon. Adaptez la fonction `tri` pour qu'elle prenne en argument un pointeur vers une fonction du même type que `ordre_crois`. Par exemple :*

```
void tri(int *tab, unsigned N, int (*f)(int, int));
```

Remplacer la condition du “if” dans `tri` par un appel à la fonction `f` passée en argument. Assurez-vous que tout marche bien.

**Exercice 4** Créer les fonctions du même type que `ordre_crois` telles que le tri se fasse par ordre :

- décroissant
- croissant, mais juste en tenant compte du dernier chiffre
- croissant, mais avec les entiers pairs en premier

## 2 Types

**Exercice 5** Quel est le type de `x` dans les définitions suivantes (à faire ensemble) :

- `int x;`
- `int *x;`
- `int x[10];`
- `int *x[10];`
- `int (*x)[10];`
- `int (*x)(int);`
- `int (*x)(int *);`
- `int (*x)(int (*)());`
- `int (*x[10])(int);`

Donner une méthode systématique (un “algorithme”) qui permet de trouver le type d’une définition aussi compliqué qu’on veut.

## 3 Structures

Considérons la structure suivante :

```
struct point {
    int x, y;
};
```

**Exercice 6** Faire un programme qui définit un objet `point` et qui donne :

- son adresse
- l’adresse des champs `x` et `y`
- la taille d’un `int`
- la taille d’une `struct point`

Faire un dessin de ce qui se passe dans la mémoire.

**Exercice 7** Faire un programme qui définit un tableau contenant 10 pointeurs vers des `struct point`. Faire pointer chaque élément du tableau vers un objet `struct point` alloué dynamiquement avec `malloc`. Initialisez les champs `x` et `y` de tous ces objets ainsi alloués. Faites une fonction qui affiche tous les objets.

**Exercice 8** *Modifiez le programme de tri de façon à ce que la fonction `tri` prenne comme argument un tableau de pointeurs vers des `struct point`. Faites les fonctions `ordre_*` pour que le tri se fasse selon :*

- les `x` croissants*
- la distance du point à l'origine*