

Bases de la Programmation

TP 4 – Fonctions

V. Padovani, PPS - IRIF

Nous avons vu en cours qu'il est possible de définir une fonction invocable sur un nom de tableau, à l'aide un paramètre de la forme `int *t`. Dans le corps d'une telle fonction, la notation `t[i]` permet d'accéder en lecture et en écriture au contenu du tableau argument. Il est courant d'ajouter à la fonction un paramètre supplémentaire indiquant la taille du tableau – on supposera celle-ci toujours non nulle, sans vérification.

Les fonctions ci-dessous seront rassemblées dans un seul fichier. Votre fonction `main` devra tester chacune des fonctions écrites sur un ou plusieurs tableaux déclarés dans `main` :

```
// ...
#define NBR 10

void print_array (int *t, int taille) {
    // ...
}

void read_array (int *t, int taille) {
    //...
}

int main () {
    int tab[NBR];

    read_array (tab, NBR);
    print_array (tab, NBR);
    // ...
    return 0;
}
```

Exercice 1 : Préliminaires

1. Ecrire une fonction `void print_array(int *t, int taille)` affichant la suite des valeurs du tableau argument, suivies d'un retour à la ligne.
2. Ecrire une fonction `void read_array(int *t, int taille)` lisant successivement `taille` entiers au clavier, en les stockant dans le tableau argument.
3. Ecrire une fonction `void rand_array(int *t, int taille, int min, int max)` tirant successivement au hasard `taille` nombres compris entre `min` et `max`, en les stockant dans le tableau argument.
4. Ecrire `void sorted_rand_array(int *t, int taille, int min, int step)`. Cette fonction devra remplir le tableau argument de la manière suivante : la valeur de `t[0]` sera égale à `min`; pour toute autre position `i`, la valeur de `t[i]` sera celle de `t[i-1]` plus un nombre aléatoire compris entre 0 et `step` inclus.

Exercice 2 : Recherche linéaire

Ecrire `int linear_search(int *t, int taille, int v)` renvoyant la première position `i` telle que `t[i]` soit égal à `v`, ou `-1` si cette position est introuvable.

Exercice 3 : Test de tri

Ecrire une fonction `int is_sorted(int *t, int taille)` renvoyant 1 si le tableau argument est trié par ordre croissant, et 0 sinon.

Exercice 4 : Max

Ecrire une fonction `int max_pos(int *t, int taille)` renvoyant la position de la plus grande valeur du tableau argument. Attention, les valeurs de ce tableau sont quelconques, elles peuvent ne pas être toutes positives.

Exercice 5 : Recherche par dichotomie

La recherche par *dichotomie* est un algorithme permettant de rechercher une valeur dans un tableau *trié*, en temps logarithmique¹. La recherche par dichotomie d'une valeur `v` manipule trois entiers `p ≤ m ≤ g` (pour « petit », « moyen » et « grand »)².

Initialement, `p` vaut 0 et `g` vaut la taille du tableau. À chaque étape du calcul :

1. si `p` est égal à `q`, l'algorithme termine en indiquant que la valeur n'a pas été trouvée,
2. sinon, `m` est (re)défini comme la partie entière de $(p + g) / 2$,
3. si `t[m]` est égal à `v`, l'algorithme termine en indiquant que la valeur a été trouvée à la position `m`,
4. sinon et si `t[m] < v`, `p` est redéfini comme $(m + 1)$, et l'on revient à l'étape 2,
5. sinon, `g` est redéfini comme `m`, et l'on revient à l'étape 1.

Ecrire une fonction `binary_search` de mêmes paramètres que la fonction `linear_search` de l'exercice 2 mais supposant que le tableau argument est trié, et effectuant une recherche par dichotomie. Que se passe-t-il si on lui passe un tableau non-trié ?

Exercice 6 : Tri par sélection

Le *tri par sélection* est un algorithme très simple³ permettant de trier un tableau sur place, c'est-à-dire sans se servir d'un tableau additionnel. Son principe est le suivant. Pour trier un tableau `t` à `taille` éléments :

- Pour chaque valeur de `i` comprise entre 0 et $(taille - 2)$,
 - on cherche la position `j` du plus petit élément du tableau à partir de la position `i`⁴;
 - on échange les valeurs aux positions `i` et `j`.

Ecrire une fonction `void selection_sort(int *t, int taille)`, triant son tableau argument par sélection. Noter que cette fonction peut être écrite en un seul bloc, mais aussi à l'aide de fonctions auxiliaires, eg. `swap` permettant d'échanger des valeurs à deux positions données, et `min_pos_from` renvoyant la plus petite valeur à partir d'une position donnée.

1. Informellement, de manière beaucoup plus efficace que la recherche linéaire.
2. Comparer cette méthode avec celle du dernier exercice du TP 2.
3. Et inefficace.
4. Si la plus petite valeur du tableau apparaît plusieurs fois, on peut par exemple choisir la première position de cette valeur minimale.

	0	1	2	3	4
$i = 0$	5	3	1	2	4
$i = 1$	<u>1</u>	3	5	2	4
$i = 2$	<u>1</u>	<u>2</u>	5	3	4
$i = 3$	<u>1</u>	<u>2</u>	<u>3</u>	5	4
	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>

FIGURE 1 – Un tri par sélection. Les valeurs soulignées sont à leurs positions correctes. La valeur en gras de chaque ligne est la plus petite valeur de la partie non triée du tableau, avant échange.