

Programmation C

TP n° 3 : Pointeurs

Exercice 1 : Swap

1. Écrire une fonction `void swap (int *p, int *q)` échangeant les valeurs entières stockées aux adresses `p` et `q`. Testez-la à l'aide des instructions suivantes dans `main` :

```

1  int x, y;
2  x = 5;
3  y = 6;
4  printf("(avant) x : %d, y : %d\n", x, y); // => (avant) 5 6
5  swap (&x, &y);
6  printf("(apres) x : %d, y : %d\n", x, y); // => (apres) 6 5
7

```

2. Créez maintenant un tableau d'entiers avec au moins deux éléments et utilisez `swap` pour échanger les premières et dernières valeurs du tableau. Testez votre code de manière similaire.

Exercice 2 : Retour de pointeur

Dans cet exercice et les suivants, les fonctions demandées supposeront que `t` est l'adresse d'un tableau d'entiers à `nbr` éléments. N'oubliez pas de tester vos fonctions.

1. Ecrire une fonction `int occ (int nbr, int t[], int v)` renvoyant : la position de la première occurrence de `v` dans le tableau d'adresse `t` si elle existe ; -1 sinon.
2. Ecrire une fonction `int *occ_ptr (int nbr, int t[], int v)` renvoyant : l'adresse de la première case du tableau contenant `v` si elle existe ; `NULL` sinon.

Exercice 3 : Paramètres de sortie

Un "paramètre de sortie" (output parameter) est un paramètre de type pointeur qui permet de fournir à une fonction une adresse à laquelle elle peut écrire un certain résultat.

1. Écrire une fonction `int nbr_occ (int nbr, int t[], int v)` renvoyant le nombre d'occurrences de la valeur `v` dans le tableau d'adresse `t`.
2. Écrire une fonction `void nbr_occ_op (int nbr, int t[], int v, int *pnv)` *inscrivant à l'adresse `pnv` le nombre d'occurrences de `v` dans le tableau.*
3. Écrire une fonction `void min_max_op (int nbr, int t[], int *pmin, int *pmax)`.
 - En un seul parcours du tableau d'adresse `t`, la fonction devra déterminer la plus petite et la plus grande de ses valeurs.
 - Elle devra ensuite écrire la plus petite à l'adresse `pmin`, et écrire la plus grande à l'adresse `max`

On supposera `nbr > 0`.

N'oubliez (toujours) pas de tester vos fonctions.

Exercice 4 : Boucles sur des pointeurs

1. Écrire une fonction

```
1 void print_tab (int t[], int start, int end)
```

En supposant que `start` est la position d'une case dans le tableau d'adresse `t`, et que `(end - 1)` est celle d'une case de position supérieure ou égale à `start`, la fonction doit afficher, pour chaque case depuis la première jusqu'à la seconde incluse :

- (a) l'adresse de cette case (utilisez `%p` pour insérer cette valeur dans une chaîne),
- (b) la valeur dans cette case.

L'affichage pourra par exemple ressembler à ceci :

```
0x7ffd6e633d60 : 1
0x7ffd6e633d64 : 5
0x7ffd6e633d68 : 2
0x7ffd6e633d6c : 1
```

2. Commenter la fonction précédente, et en écrire une nouvelle version.

```
1 void print_tab (int *pstart, int *pend)
```

En supposant cette fois que `pstart` est l'adresse d'une case d'un tableau quelconque, et que `(pend - 1)` est l'adresse d'une case de position supérieure ou égale à celle d'adresse `pstart`, la fonction doit afficher, pour chaque case depuis la première jusqu'à la seconde incluse, les mêmes informations que précédemment.

Indication. Vous n'avez pas à utiliser la notation indexée, et on peut incrémenter un pointeur.

Exercice 5 : (★) Tri à bulles

Le tri à bulles (ou tri par propagation) est un algorithme de tri lent mais peu gourmand en mémoire. Son principe est le suivant. Étant donné un tableau `t` de taille `N` :

- Pour chaque `n` allant de $(N - 1)$ inclus à 1 exclu :
 - Pour `i` allant de 0 inclus à `n` exclu :
 - Si $t[i] > t[i + 1]$, échanger les contenus de positions `i` et `i + 1`.

Noter qu'à chaque étape de la boucle en `n`, et après la boucle en `i`, le maximum de $t[0 \dots n]$ est en $t[n]$, d'où la correction du traitement.

1. Écrire une fonction `void sort (int t[], int start, int end)` qui trie les entiers stockés entre les positions `start` (inclus) et `end` (exclu) du tableau d'adresse `t`. Pensez à utiliser la fonction codée à l'exercice 1.
2. Plutôt que des valeurs de positions, on peut utiliser des pointeurs sur les cases correspondantes. Écrire une fonction `void sort_ptr (int *start, int *end)` qui trie les entiers situés en mémoire entre l'adresse `start` (incluse) et l'adresse `end` (exclue). L'algorithme doit être le même qu'à la question précédente.

Indication. Remplacez les boucles sur les positions par des boucles sur des pointeurs.

3. Tester votre fonction avec un tableau non trié, initialisé à la main. Si un tableau `tab` est de taille `N`, son tri peut se faire par un appel de la forme `sort_ptr (tab, tab + N)` ; Affichez le contenu du tableau pour vérifier le résultat.