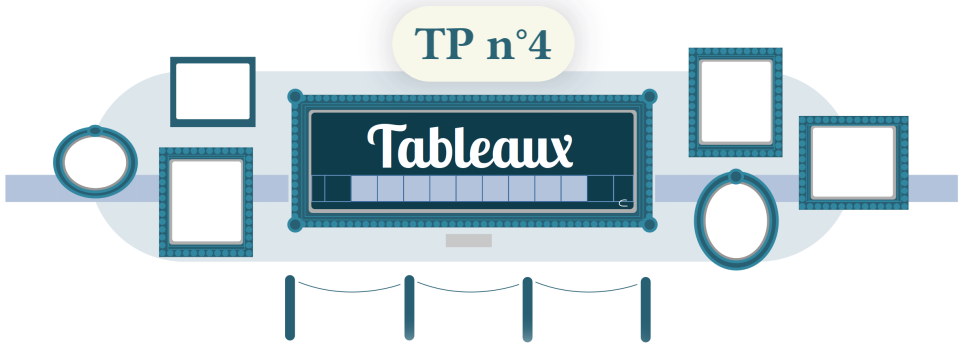


I. Prolégomènes

1 À l'aide d'un terminal de commandes, créer un nouveau répertoire nommé **TP4** dans votre répertoire **TP**, et se déplacer à l'intérieur. Ouvrir un nouveau fichier **tp4.c** avec l'IDE VSCodium. Dans **tp4.c**, écrire le squelette de base d'un code C, en tentant de le faire de mémoire sans regarder.

Tableau

Un **tableau** est une suite de valeurs **stockées consécutivement en mémoire**. Cela permet d'accéder rapidement à la $i$ème valeur : on retrouve l'adresse de la case souhaitée en se décalant de $i-1$ cases par rapport à la première.

II. Définir un tableau en C

Définir un tableau

Voici la syntaxe pour définir un tableau de six entiers en C, ayant pour valeurs 4, 2, 1, 1, 0 et 8 dans cet ordre :

```c
int t[6] = {4, 2, 1, 1, 0, 8};
```
ou plus simplement
```c
int t[] = {4, 2, 1, 1, 0, 8};
```

On peut également définir un tableau de flottants (`float t[] = ...`), de caractères (`char t[] = ...`), de booléens (`bool t[] = ...`) ou n'importe quel autre type. Pour ce TP, nous nous limiterons aux tableaux d'entiers (`int`). **Attention !** Cette syntaxe entre accolades ne peut être utilisée qu'à l'initialisation du tableau, pas plus tard.

Pour accéder à la première case du tableau `t`, on écrit `t[0]`, avec l'indice entre crochets. Cet indice correspond au décalage à réaliser par rapport à la première case. On accède donc à la $i$ème valeur du tableau `t` avec `t[i-1]`.

2 Dans votre fonction `main`, implémenter le tableau donné en exemple, ainsi qu'un autre tableau d'entiers de votre choix de 10 éléments. Combien de place prennent ces tableaux en mémoire ?

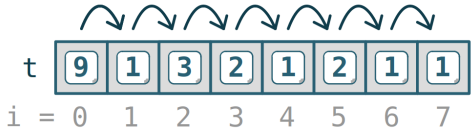
3 Afficher la troisième valeur de votre tableau à l'aide de la fonction `printf`. Peut-on accéder à une valeur hors du tableau ? À retenir : C ne possède pas de manière simple de connaître la taille d'un tableau. **C'est à vous de la stocker, et de prendre soin de ne jamais lire au-delà de cette limite**.

4 Implémenter une fonction de prototype `void swap(int t[], int i, int j)` qui échange les valeurs des cases `i` et `j` du tableau `t` donné en argument. On supposera que `i` et `j` sont des indices valides. Remarquer la syntaxe utlilisée pour qu'un tableau soit paramètre d'une fonction : `int t[]`.

III. Parcours linéaire d'un tableau

On dit que l'on **parcourt** le tableau lorsque l'on visite chacun de ses éléments, généralement dans l'ordre.

```c
void efface(int t[], int taille) {
    for (int i=0; i<taille; i++) {
        t[i] = 0;
    }
}
```



Cette fonction parcourt le tableau `t` et écrit la valeur 0 dans chacune de ses cases. Pour cela, une boucle `for` est très utile pour pouvoir visiter chaque indice valide, entre 0 et `taille-1` inclus. Dans le même esprit, les fonctions à implémenter dans cette partie se basent toutes sur un parcours de tableau. On donne toujours sa taille en argument.

5 Implémenter une fonction de prototype `void affiche_tableau(int t[], int taille)` qui affiche chaque entier présent dans le tableau `t`. Proposer un affichage propre commençant par le symbole `'['` et finissant par le symbole `']'`. Indice : utiliser une boucle `for` qui parcourt les indices de 0 à `taille-1`, et appeler la fonction `printf` dans son bloc d'instructions pour afficher l'élément.

6 Implémenter une fonction de prototype `bool mem(int t[], int taille, int element)` qui renvoie `true` si l'entier `element` est présent dans une des cases du tableau, et `false` sinon. Indice : utiliser une boucle `for` et une conditionnelle (`if`) à l'intérieur pour tester si l'élément vaut la bonne valeur.

7 Implémenter une fonction de prototype `void tests_parcours()` qui vérifie que les résultats renvoyés par la fonction précédente sont ceux attendus grâce au mot-clef `assert`. Pour chaque nouvelle question traitée dans cette partie, rajouter au moins deux tests supplémentaires dans `tests_parcours`, pour vérifier que le comportement est celui attendu.

8 Implémenter une fonction de prototype `int nb_occurences(int t[], int taille, int element)` qui renvoie le nombre d'occurences de l'entier `element` dans le tableau `t`.

9 Implémenter une fonction de prototype `int somme(int t[], int taille)` qui renvoie la somme des éléments du tableau `t`. On pourra utiliser une variable pour stocker la somme partielle.

10 Implémenter une fonction de prototype `int argmax(int t[], int taille)` qui renvoie l'indice de l'entier le plus grand présent dans le tableau `t`.

11 Implémenter une fonction de prototype `bool est_croissant(int t[], int taille)` qui vérifie si les éléments du tableau `t` sont ordonnés dans l'ordre croissant, en un seul parcours.

12 Implémenter une fonction de prototype `bool tab_egaux(int t1[], int t2[], int taille)` qui vérifie si deux tableaux de même taille contiennent les mêmes éléments dans le même ordre.

IV. Tri par sélection

Dans cette partie, nous nous intéressons à un **algorithme de tri**, c'est-à-dire un algorithme qui permute les éléments du tableau pour qu'ils soient ordonnées dans l'ordre croissant. Le **tri par sélection** commence par chercher la plus petite valeur du tableau : on sait qu'il s'agit de la première case du tableau trié. On échange donc cette case minimale avec la première case du tableau. Ensuite, on recherche de nouveau la case la plus petite parmi les cases restantes, et on vient l'échanger avec la deuxième case du tableau pour la mettre à sa place. On continue ainsi jusqu'à avoir mis chaque case du tableau à sa place.



13 En suivant l'algorithme, continuer l'exemple proposé en dessinant les étapes omises par les points de suspension au brouillon.

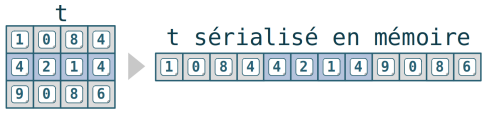
14 Implémenter une fonction de prototype `void tri_selection(int t[], int taille)` qui suit cette approche pour trier le tableau `t`.

15 Implémenter une nouvelle fonction de tests qui permette de vérifier le bon comportement de cette fonction de tri. Compiler et vérifier que tout se passe bien.

V. Tableaux bidimensionnels

En C, il est également possible de manipuler des tableaux bidimensionnels (voire multidimensionnels), par exemple pour implémenter des matrices. Dans ce cas, les valeurs sont tout de même stockées à la suite en mémoire, ligne par ligne. Pour définir un tel tableau, on utilise une syntaxe similaire aux tableaux unidimensionnels, mais avec plusieurs indices entre crochets. Par exemple pour définir le tableau suivant de 3 lignes et 4 colonnes on note :

```
int mat[3][4] = {{1, 0, 8, 4},
                 {4, 2, 1, 4},
                 {9, 0, 8, 6}};
```



16 Comment passer des indices (i, j) d'une matrice à l'indice k du tableau sérialisé en mémoire ? Implémenter cela dans deux fonctions de conversion, et vérifier sur un exemple.

17 Un tableau 2D est un **tableau de Young** si chaque ligne et chaque colonne est ordonnée par ordre croissant. Implémenter une fonction de prototype `bool est_young(int mat[5][5])` qui décide si le tableau `mat` de dimension 5 x 5 est un tableau de Young. On retiendra qu'il faut toujours indiquer la dimension d'un tableau multidimensionnel dans un prototype de fonction, ici par exemple on a précisé `int mat[5][5]`. Pourquoi le compilateur en a-t-il besoin ?

VI. Collection de problèmes plus ouverts

Plus longue répétition consécutive

On souhaite connaître la taille de la plus longue suite d'éléments adjacents de même valeur dans `t`. Dans l'exemple ci-contre, cette valeur est 3.



18 Implémenter une fonction de prototype `int plus_longue_repet(int t[], int taille)` qui renvoie le nombre maximum d'éléments égaux adjacents dans `t`.

Mêmes éléments, mais dans le désordre

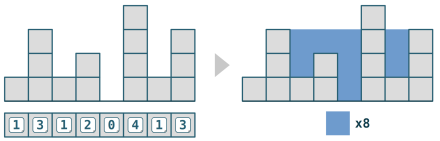
On souhaite savoir si deux tableaux `t1` et `t2` de même taille possèdent les mêmes éléments, avec le même nombre d'occurences, mais pas forcément dans le même ordre.



19 Implémenter une fonction de prototype `bool memes(int t1[], int t2[], int taille)` qui réponde à cette question. On supposera que les tableaux n'ont que des entiers positifs.

Averse

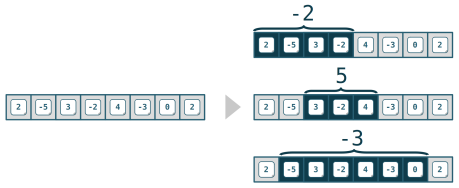
On suppose que le tableau `t` d'entiers positifs représente une structure de blocs comme illustré ci-contre : la case d'indice i du tableau indique la hauteur de la $i$ème pile de blocs. Une averse inonde alors la structure. Dans l'exemple, la structure retient 8 blocs d'eau.



20 Implémenter une fonction de prototype `int averse(int t[], int taille)` qui renvoie le nombre de cases d'eau piégées après inondation de la structure par l'averse.

Sous-tableau de somme maximum

Un sous-tableau de `t` est une suite adjacente de cases de `t`, éventuellement vide. On souhaite déterminer la plus grande valeur possible en faisant la somme des éléments d'un sous-tableau de `t`. Sur l'exemple ci-contre, cette valeur vaut 5. On pourra supposer qu'il y a toujours au moins un entier positif dans `t`.



21 Implémenter une fonction de prototype `int sous_tableau_max(int t[], int taille)` qui renvoie la somme maximum que l'on puisse obtenir en prenant un sous-tableau de `t`. Implémenter une autre fonction mais qui ne visite qu'une seule fois chaque élément du tableau. Autrement dit, une seule boucle `for` doit suffire.