

# Niveau 4

TP  
S4. Développement logiciel  
S4.1. Principes de base : Gestion mémoire, allocation dynamique

## Le mobile d'Otto

### TABLE DES MATIÈRES

1 - Tableaux dynamiques.....	2
1.1 - Allocation.....	2
1.1.1 - Tableau 1D.....	2
1.1.2 - Tableau 2D.....	3
1.2 - Dés-allocation.....	3
2 - Travail à réaliser.....	4
2.1 - Gestion dynamique d'un tableau 1D.....	4
2.2 - Bonus.....	6
2.3 - Tri à bulle.....	6



# 1 - TABLEAUX DYNAMIQUES

## 1.1 - ALLOCATION

Très fréquemment, nous sommes amenés à stocker des données dans un tableau, nous l'avons déjà vu. Mais dans certains cas, le nombre de données à gérer n'est pas connu lors de la déclaration du tableau. C'est notamment le cas lorsqu'il s'agit de saisir le nombre de données à stocker...

Il convient alors de créer un tableau de taille parfaitement adaptée. Pour cela, la syntaxe `nTab[x]` n'est pas valide, car `x` n'est pas connu au moment de la déclaration du tableau.

En effet, selon le type des éléments du tableau, le compilateur n'est pas toujours capable de créer correctement ce tableau. Les types simples les plus classiques ne semblent pas poser de problème la plupart du temps, mais ce n'est pas toujours le cas.

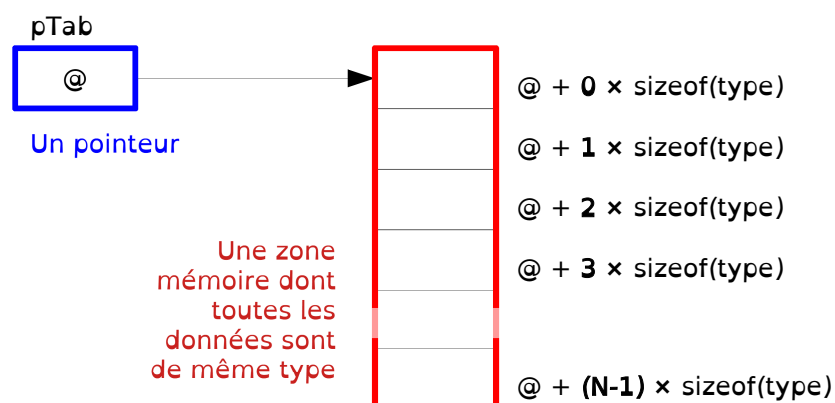
N'importe quel type de données, simples et complexes, peut être utilisé lors d'une déclaration dynamique de tableau !

Il faut donc, dans une situation pareille, **allouer dynamiquement** le tableau, en demandant au système d'exploitation de réserver une plage de mémoire, qui va rester disponible durant le reste de l'exécution, grâce à son adresse de début, et jusqu'à ce qu'on demande au système de libérer cet espace.

Attention, si l'espace n'est pas rendu explicitement au système avant la fin du programme, cet espace reste réservé mais ne sera plus accessible pour aucune autre application... Elle devient une zone fantôme en mémoire !

### 1.1.1 - TABLEAU 1D

Souvenez-vous la structure réelle d'un tableau en mémoire :



L'allocation dynamique d'un tableau 1D reprend cette structure et s'effectue par l'opérateur **new**. Par exemple, pour créer un tableau de `NbreValeur` réelles, on écrira :

```
float * pTab      (nullptr) ; // Pointeur sur une zone mémoire contenant des réels
int     NbreValeur (37) ;

...
pTab = new float [NbreValeur] ; // Création de la zone mémoire de NbreValeur réels
                                   // et affectation du pointeur
```

Par la suite, ce tableau est utilisable de façon classique :

```
pTab[3] = 24.65 ;
```

## 1.1.2 - TABLEAU 2D

Un tableau 2D peut être vu comme un tableau 1D dont chaque case contient un autre tableau 1D, ne l'oubliez pas !

Dans le cas d'un tableau 2D créé dynamiquement, on doit utiliser un pointeur (première dimension) qui pointe sur un autre pointeur (seconde dimension) sur des réels. La déclaration d'un tel tableau dynamique, de 18 × 43, serait la suivante :

```
float * * pTab (nullptr) ; // Déclaration d'un pointeur sur des pointeurs de réels
int NbreDim_1 (18) ;
int NbreDim_2 (43) ;
```

La demande d'allocation du tableau en 2D doit donc être réalisée en deux phases :

- la première phase consiste à allouer dynamiquement la première dimension, comme indiqué précédemment, mais en utilisant le type **pointeur sur réel** (`float *`) plutôt que `float` seulement :  

```
pTab = new float * [NbreDim_1] ;
```
- une fois la première dimension allouée, chaque case doit être passée en revue, au moyen d'une boucle **POUR**, pour y placer dynamiquement un autre tableau dynamique 1D, contenant, lui, des réels (`float`), comme indiqué au chapitre 1.1.1 - Tableau 1D :  

```
for(unsigned int i (0) ; i < NbreDim_1 ; i++)
{
    pTab[i] = new float [NbreDim_2] ;
}
```

Par la suite, ce tableau est utilisable de façon classique, en prenant bien garde de respecter les limites données à chacune des dimensions :

```
pTab[3][19] = 3,14159 ;
```

## 1.2 - DÉS-ALLOCATION

Le problème dans l'allocation dynamique de mémoire, est que, si l'on ne libère pas la mémoire explicitement, c'est-à-dire si on ne demande pas au système de libérer la mémoire qui n'est plus utilisée et qu'on lui avait demandé de réserver, cette mémoire restera bloquée et inutilisable dans le système.

Il faut donc toujours demander, au plus tard en fin du programme principal, de libérer la mémoire que l'on a demandé d'allouer.

L'opération s'effectue toujours dans l'ordre inverse de la demande : la seconde dimension doit être libérée en premier, puis la première dimension sera dés-allouée.

Pour libérer une allocation dynamique réalisée par `new`, on utilise l'opérateur **delete** !

Pour reprendre l'exemple précédent, pour libérer l'espace du tableau `Tab`, on écrira simplement :

```
if(pTab != nullptr)    // Vérifie que la zone mémoire a bien été allouée précédemment
{
    // Ré-initialisation du tableau à 0 pour laisser la mémoire propre
    InitTab1D(pTab, nTaille) ;
    delete [] pTab ;    // Dés-allocation de la zone mémoire réservée par le new []
    pTab = nullptr ;    // Remise à nullptr du pointeur du tableau pour éviter une
                        // seconde (voire plus) demande de dés-allocation.
}
```

Notez les `[]` qui indiquent au système qu'il s'agit de dés-allouer un tableau. La taille de la dimension n'est pas à préciser... Et remettre le pointeur à `nullptr` est une importante sécurité.

Pour un tableau à 2D, il faut dés-allouer le tableau en deux phases :

- la première consiste à scruter par une boucle **POUR** chaque case de la première dimension, et de libérer le tableau de la seconde dimension alloué pour cette case-là ;
- la seconde phase est de libérer la première dimension.

## 2 - TRAVAIL À RÉALISER

### 2.1 - GESTION DYNAMIQUE D'UN TABLEAU 1D

Monsieur Root doit réaliser l'acquisition du nombre de véhicule qui passent sur une partie de voirie pendant une certaine période de temps et à une fréquence à définir. Ces données seront ensuite triées pour que Otto en connaisse la valeur médiane.

Il vous demande de l'aider à résoudre son problème sachant que la durée des acquisitions peut être comprise entre 1 h et 48 h (inclus), et que la période d'acquisition peut varier entre 1 min et 15 min (inclus). Ces deux paramètres sont renseignés par l'utilisateur au lancement du programme d'acquisition.

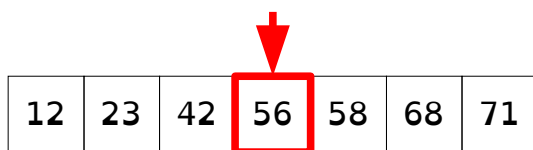
- 1) Dans un nouveau projet, déclarez dans le `main()`, les variables nécessaires pour gérer un tableau dynamique (`pTabOtto`) contenant des entiers non signés, la `nDuree` totale d'acquisition, la `nPeriode` d'acquisition et le `nNbAcquisition` total que devra contenir le tableau. La durée et la période seront à demander à l'utilisateur. Le nombre de données acquises devra être calculé en fonction des données saisies.
- 2) Ajoutez le couple de fichiers `.h` et `.cpp` à votre projet.
- 3) Écrivez un module `CreerTab1D()` qui va créer **dynamiquement** le tableau `pTabOtto` (déclaré dans le `main()` sous forme de pointeur), ayant une dimension de taille adaptée au `nNbAcquisition`, passé en paramètre. La fonction utilise une variable locale, `pTab`, pointeur elle aussi, lui permettant l'allocation de la zone mémoire désirée. Une fois allouée à `pTab`, le tableau ainsi formé est initialisé à 0 puis **retourné** au `main()`. Le type de retour est donc un pointeur.  
***Vous pourrez réutiliser des modules d'initialisation déjà réalisés !***  
 Voici le prototype à respecter :  

```
unsigned int * CreerTab1D(unsigned int nNbAcquisition) ;
```

 Appelez cette fonction dans le `main()`.

- 4) Profitez d'avoir alloué le tableau dynamiquement pour rédiger le module qui va permettre sa dés-allocation comme indiqué au chapitre 1.2 - Dés-allocation.  
le résultat de la dés-allocation est retourné au `main()` : VRAI si tout c'est bien passé, FAUX sinon !  
Voici le prototype à respecter :  
`bool LibérerTab1D(unsigned int * pTab, unsigned int nTaille) ;`  
Appelez cette fonction à la toute fin du `main()`.
- 5) Ajoutez à votre programme un module d'affichage de tableau 1D... vous pouvez récupérer et adapter légèrement un module déjà réalisé.  
Le module doit porter le nom `AfficherTab1D`.  
Appelez cette procédure dans le `main()`. Le programme ne doit pas planter !
- 6) Réutilisez et incorporez un module d'acquisition afin de remplir le tableau. Vous pouvez soit faire des saisies manuelles, soit une saisie automatique aléatoire. Dans les deux cas, on considère qu'une acquisition doit être comprise entre 100 et 1000.  
Ce module porte le nom `AcquérirTab1D`.  
Appelez ce module dans le `main()` et affichez le contenu du tableau.
- 7) Poursuivez votre programme en ajoutant un module permettant de trier le tableau par ordre croissant (les petites valeurs dans les indices faibles et les plus grandes valeurs à la fin du tableau).  
Vous trouverez une technique de tri de tableau au chapitre 2.3 - Tri à bulle. à la page 6.  
Ce module s'appelle `TrierTab1D`.  
Appelez ce module dans le `main()` et ré-affichez le tableau une fois trié.
- 8) Terminez le programme en écrivant un module permettant de trouver la médiane des valeurs acquises. La valeur médiane est celle qui est au milieu du tableau. Le tableau et sa taille sont passés au module qui va retourner cette médiane.  
Si le nombre d'acquisition est impair, pas de problème : la médiane est la valeur centrale.  
Si le nombre est pair, la médiane est la moyenne des deux valeurs centrales, arrondie à l'inférieur...  
Ce module porte le nom `TrouverMedianeTab1D`.  
Appelez ce module dans le `main()` et affichez la médiane pour Otto !

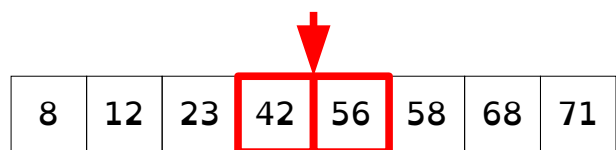
Médiane = 56



12	23	42	56	58	68	71
----	----	----	----	----	----	----

Nbre d'acquisitions impair

Médiane =  $(42+56)/2 = 49$



8	12	23	42	56	58	68	71
---	----	----	----	----	----	----	----

Nbre d'acquisitions pair

### Indices de validation :

Avec une acquisition automatique aléatoire, réglez la graine avec la valeur 2023...

Durée	5	2
Période	11	10
Nombre d'acquisitions	27	12
Valeur médiane	540	541
Calcul	<code>pTab[13]</code>	<code>(pTab[5] + pTab[6])/2</code>

## 2.2 - BONUS

- 9) Reprenez les questions 1 à 6 en adaptant les modules à la gestion d'un tableau à deux dimensions d'entiers non signés.  
Les noms seront à adapter avec 2D à la place de 1D !  
Les tailles des deux dimensions devront être passées en paramètres ainsi que le double pointeur du tableau.

## 2.3 - TRI À BULLE.

Le tri est réalisé en propageant par permutations successives le plus grand élément du tableau vers la fin de celui-ci et donc le plus petit se déplace vers le début. De la même façon que les bulles d'air montent à la surface de l'eau, le plus petit élément, "plus léger" que les autres, gagne de proche en proche la « surface » (extrémité du tableau d'indice 0).

Tout le tableau est parcouru au moins une fois. L'élément courant est comparé avec son voisin suivant immédiat. Si ces deux valeurs sont classées entre-elles, il ne se passe rien et on passe à l'élément suivant. Si par contre, elles ne sont pas bien ordonnées, on les permute et on passe à l'élément suivant du tableau. Un élément plus lourd que les autres s'enfonce vers la fin du tableau jusqu'à ce qu'il rencontre un élément encore plus grand.

Une fois que le tableau a été parcouru en entier de cette façon, on pourrait croire qu'il est trié. Et bien pas forcément : s'il y a eu une permutation, il se peut que l'élément qui est monté (si le plus grand s'enfonce, le plus petit, lui, va monter) ne soit pas bien placé. Il faut donc tout reprendre depuis le début... Et ceci jusqu'à ce qu'aucune permutation ne soit réalisée dans le parcours du tableau.

Exemple :

