Analyse du tableau dynamique (vector): fonctionalités, points forts /faibles

```
#include <vector>
vector<double> tab(5);
tab.size()
tab[i]
```

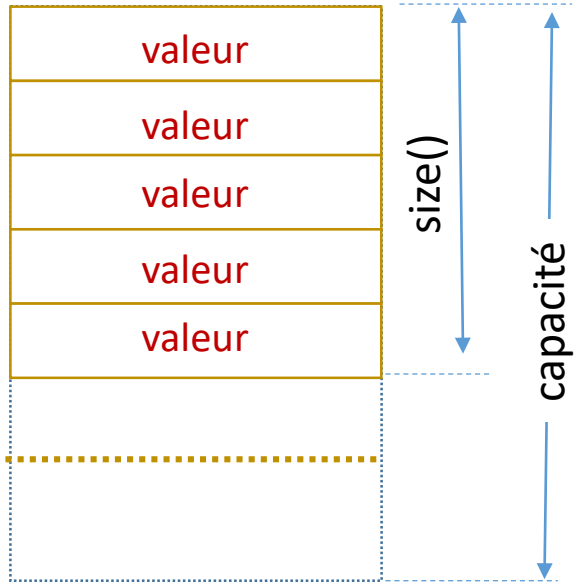
Fait partie des outils de type ***container***

Initialisation avec le motif binaire nul

Un vector connaît sa taille

Accès en **O(1)** à tout élément du vector

Extrait du cours Topic12 sem1



Un vector est particulièrement efficace pour ajouter/enlever **le dernier élément** (coté ***back***) avec **`push_back`** et **`pop_back`**.

Cette fonctionalité permet facilement de mettre en œuvre le concept de *pile* (LIFO = Last In, First Out)

```
tab.push_back(x); // ajoute la valeur x ici
tab.pop_back();   // enlève cette dernière valeur
```

Point faible1: une méthode **insert** permet d'insérer *un ou plusieurs* élément à l'intérieur du vecteur MAIS cela implique un coût de décalage de tous les éléments qui suivent. Même problème avec la méhode **erase** pour enlever *un ou plusieurs* éléments.

Point faible2: il n'existe PAS de méthode pour ajouter/enlever le premier élément (coté ***front***). => Utiliser le ***container*** **deque** (double-ended queue). Cet outil permet de mettre en œuvre le concept de *file d'attente* (**queue**).

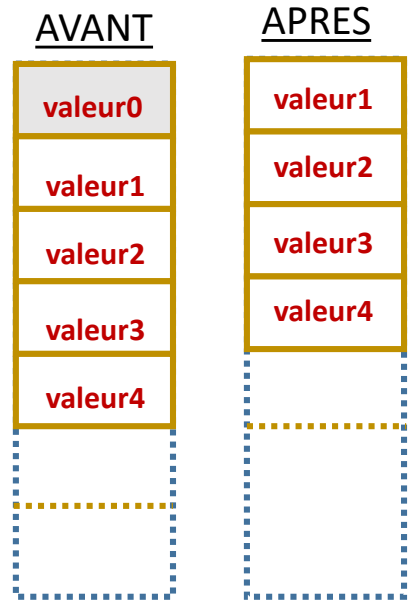
Manipulation sur un tableau dynamique non trié (vector)

Complément à la fiche points forts/faibles

Observation1: les méthodes **insert** et **erase** ne doivent pas modifier l'ordre des éléments déjà présent dans un vector. C'est pourquoi elles induisent un coût de décalage des éléments qui suivent dans le vector.

Conséquence négative: coût linéaire => **O(vector_size)**

Ex: utilisation de **erase** pour supprimer le premier élément de **valeur0**

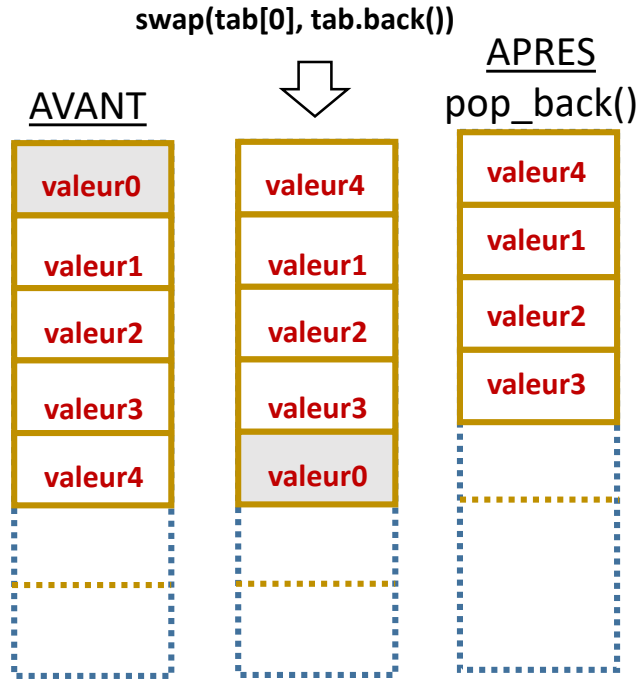


Observation2: si le tableau dynamique n'a ***pas besoin d'être trié***, alors on peut réduire drastiquement l'ordre de complexité des opérations *d'ajout* et de *suppression* d'un élément.

Pour la ***suppression***, il suffit d'utiliser la méthode **swap** pour échanger la valeur de l'élément à supprimer avec celle du dernier élément, puis d'appeler **pop_back().**

Conséquence positive: coût constant => **O(1)**

Ex: suppression du premier élément du vector **tab** de valeur **valeur0**



Pour ***l'ajout***, il suffit d'appeler **push_back(valeur).**

Conséquence positive: *généralement* à coût constant => **O(1)**

Mise en œuvre de file d'attente avec deque (double ended queue)

```
#include <deque>
```
Fait partie des outils de type «container»

```
deque<int> q;
```
Déclaration d'un deque vide

```
q.size()
```
Un deque connaît sa taille

```
q[i]
```
Accès avec un indice à tout élément du deque

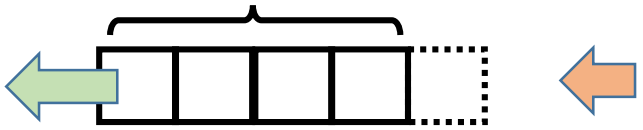
```
q.empty()
```
Renvoie *true* s'il est vide

Un **deque** permet facilement de mettre en œuvre le **concept** de *file d'attente* (FIFO = First In, First Out).

Exemple représentant une file d'attente à une caisse de supermarché ; les clients arrivent par la droite.

On obtient l'information du client en tête de file avec **q.front()** *et on l'enlève avec* **q.pop_front()**

Clients déjà présents dans la file



On ajoute un client en queue de file avec **q.push_back(val)**

Un deque peut ajouter/enlever **le premier élément** (coté ***front***) avec `push_front` et `pop_front`.

Comme vector, il peut ajouter/enlever **le dernier élément** (coté ***back***) avec `push_back` et `pop_back`. Cependant vector est plus performant que deque pour cette opération.

Note: C++ offre un «container adaptor» **queue** dont l'interface est restreinte aux opérations autorisées sur une file d'attente (méthodes **push(val)**, **front()** et **pop()**). Cet aspect est vu en série4.