## Semaine 7 – Tableaux dynamiques

Dans ce TP vous allez implémenter une structure de donnée similaire aux vecteurs du C++ permettant d'utiliser faiclement des tableaux dynamiques dont la taille n'est pas connue au moment de la compilation mais évolue au fur et à mesure que l'on ajoute des éléments. Cette fois-ci, il vous est demandé de faire attention au erreurs d'allocation mémoire et de les gérer en terminant proprement le programme avec un message d'erreur.

# 1 Structure de base

Commencer par définir les éléments de base permettant la gestion d'un vecteur dynamique :

1. Tout d'abord, définissez la structure qui va constituer un vecteur, celle-ci doit contenir ce qui est nécessaire afin de mémoriser toutes les informations contenues dans le tableau.
2. Ensuite définissez les fonction permettant d'allouer et de liberer un vecteur. Attention à correctement gérer les erreur d'allocation mémoire.

   ```
   vecteur *nouveau_vecteur();

   void liberer_vecteur();
   ```

3. Enfin, pour faciliter la mise au point des fonctions suivantes, implémentez une fonction permettant l'affichage d'un vecteur.

   ```
   void afficher_vecteur();
   ```

# 2 Ajout et suppression en queue

Les vecteurs en C++ proposent des fonction spéciale afin d'ajouter et de supprimer un élément en dernière position. Ces fonction sont proposées car elle peuvent être implémenter de manière très efficace par rapport à celles que nous implémenterons dans la section suivante du TP.

1. Commencer par définir une fonction permettant d'augmenter la taille réelle d'un vecteur. Cette fonction sera très utile pour les suivantes qui l'appellerons lors de l'ajout d'une valeur à un tableau déjà plein.

   ```
   void augmenter_taille(vecteur *v);
   ```

2. Implémentez maintenant les fonction d'ajout et de suppression en queue, attention de bien gérer les différents cas d'erreurs. (la fonction de suppression va retirer la valeur et la renvoyer)

   ```
   void ajouter_en_queue(vecteur *v, int val);

   int retirer_en_queue(vecteur *v);
   ```

Pour rendre notre structure de donnée utilisable, d'autres fonction sont nécessaires. Celle-ci ont aussi une imlémentation simple et efficace.

1. Implémentez une fonction permettant d'obtenir le taille d'un vecteur, c'est-à-dire le nombre de valeur réellement stockés dans ce vacteur.

   ```
   int taille_vecteur(vecteur *v);
   ```

2. Implémenter maintenant deux fonction permettant de lire et de modifier la valeur à un indice donné du tableau :

   ```
   int lire_valeur(vecteur *v, int idx, int val);
   ```

   ```
   void modifier_valeur(vecteur *v, int idx);
   ```

# 3 Insertion et suppression

Il nous reste à implémenter les fonctions plus générales permettant l'utilisation de nos vecteurs au prix de performances plus faibles que les précédentes.

1. Implémentez la fonction permettant d'ajouter une valeur à une poistion quelconque du vecteur. Vous retrouvez ici une difficulté similaire à celle rencontrée précédement lors du tri par insertion.

   ```
   void inserer(vecteur *v, int idx, int val);
   ```

2. Et enfin, implémentez la fonction permettant de supprimer une valeur quelconque du vecteur.

   ```
   void supprimer(vecteur *v, int idx);
   ```

Afin de tester dans des conditions un peu plus réelles votre code, vous pouvez réimplémenter le tri par insertion sur les vecteurs que vous venez de définir. Attention de choisir une variante utilisant au mieux les opération disponibles.