# TP 3

## 1 Construction d'une classe d'objets fonctions

`15 minutes, noté (à rendre à votre chargé de TP, mail ou clé USB)`
Construisez une classe de générateurs d'entiers, multiples d'une valeur donnée. Les valeurs fournies par un générateur vont en augmentant au fil des sollicitations. Testez votre classe dans un main affichant les 10 premiers multiples de 3, puis les 20 premiers multiples de 5.

## 2 Fonctions template

Considérons la procédure qui échange les valeurs de 2 variables. Le corps de cette procédure est identique qu'il s'agisse d'entiers, de réels ou de chaînes de caractères (seul le prototype diffère). Dans ce cas, le C++ offre la possibilité de définir un unique patron de fonction (template) qui pourra ensuite être instancié avec des entiers, des réels, des caractères ... (ou tout autre type compatible avec le code générique). Le mécanisme d'instanciation correspond à la génération automatique de code par le compilateur.

- Écrivez la procédure template `void swap(T& a,T& b)` d'échange de 2 éléments. Pourquoi faut-il passer les éléments par adresse non constante ?
- Écrivez une fonction template min qui permet de retourner le minimum de deux éléments de même type ;
- Écrivez une procédure template `void afficheTableau(T tab[], int n)` affichant le contenu du tableau `tab` sur stdout ;
- Écrivez une procédure template `bool isSorted(T tab[], int n)` retournant `vrai` si et seulement si le tableau `tab` est trié ;
- Écrivez une procédure template `void remplisAleatoire(T tab[], int n, T max))` qui va remplir le tableau `tab` de nombres aléatoire entre 0 et max. La procédure générique ne fera rien, seule des spécialisations seront définies. Vous définirez dans un premier temps uniquement la spécialisation pour `int` (en utilisant la fonction `rand`), et s'il vous reste du temps à la fin du TP vous pourrez définir la spécialisation pour d'autres types numériques ;
- Écrivez une procédure template `void heapSort(T tab[], int n)` qui trie les éléments de `tab` par ordre croissant. Cette fonction est templatée par le type des éléments du tableau. On utilisera une implantation de type tri par tas (vous utiliserez donc une procédure annexe `void entasser(T tab[], int rac, int n)` se chargeant d'entasser la valeur `tab[rac]` dans le tableau `tab` étant un tas partout sauf pour cet élément). Testez votre tri sur différents tableaux d'entiers remplis de manière aléatoire. Après chaque tri, vérifiez que le tableau est bien trié ;
- Spécialisez la fonction `min` de manière à étendre sa validité au cas de 2 chaînes de caractères et tester la procédure de tri de la question précédente sur un tableau de chaîne de caractères ;
- S'il vous reste du temps, écrivez une procédure template `void quickSort(T tab[], int n)` implantant le tri rapide en version générique (vous trouverez l'algorithme sur la page wikipedia). Comparez les temps d'exécution de vos deux méthodes de tri sur des tableaux de plus en plus gros.

## Rappel du tri par tas

Un tableau est structuré en tas binaire décroissant entre les indices 0 et $n-1$ (bornes comprises) s'il vérifie la propriété :
$t[i]>=t[2i+1]$ et $t[i]>=t[2i+2]$ pour tout $i$ compris entre 0 et $n/2-1$.
De ce fait, Un tableau de taille $n$ est toujours structuré en tas binaire décroissant entre les indices $n/2$ et $n$.
Soit un tableau étant structuré en tas binaire partout sauf à un indice `rac` donné (avec $0 \leq rac < n$), il est possible de réorganiser localement en $log(n)$ ce tableau afin de lui rendre sa propriété de tas en tout élément. Le principe de l'algorithme consiste à comparer $t[rac]$ avec $t[2rac+1]$ et $t[2rac+2]$ et de faire remonter le plus grand des trois éléments à la place $t[rac]$. En cas d'échange, le tableau est un tas binaire partout sauf à l'indice de la valeur échangée avec $t[rac]$ et on réitère alors le processus. Cet algorithme est implanté dans la fonction `void entasser(T tab[], int rac, int n)`.

Pour structurer un tableau arbitraire en tas binaire, il suffit d'appeler la fonction `entasser` pour tout élément du tableau en commençant à l'indice $n/2-1$ jusqu'à l'indice 0. Ensuite, l'algorithme de tri consiste ensuite à retirer, petit à petit et de manière décroissante, les éléments du tas (à chaque itération échange du premier élément qui est le plus grand élément du tableau et du dernier élément de la partie du tableau correspondant au tas, et restitution de la propriété de tas). La partie du tableau consacrée au tas diminue, ce qui laisse de la place pour stocker les éléments retirés (et triés !). On dit que l'algorithme de tri par tas est un algorithme de tri sur place.