- Fiche de TP1 : Implémentation d'un tas -

On propose de réaliser les tps en langage C, mais vous pouvez choisir un autre langage de programmation si vous le souhaitez. Une trame de programme est disponible pour certains exercices. Tout le code est en C sauf les primitives d'entrées/sorties (`cin` et `cout`) qui sont en C++. Il faudra compiler avec **g++**.

Même si ce n'est pas vraiment nécessaire pour les tas, il est pratique pour les files de priorité de travailler avec des tableaux de taille fixée grande et de taille utile plus petite. Par exemple on définira la taille fixée par

```
#define N 1000
```

Et on déclarera des tableaux de taille `N` avec une taille utile `n` plus petite.

- Tas -

- Exercice 1 - Entrées/sorties sur les tableaux -

Écrire une fonction `AfficherTableau(int n, int T[])` qui affiche le tableau T de taille utile n.
Écrire une fonction `RemplitTableau(int n, int T[])` qui remplit le tableau T de taille utile n, soit en demandant chaque valeur à l'utilisateur, soit en générant aléatoirement une valeur entre 1 et 100 pour chaque case. On pourra utiliser les appels :

- `srand (time(NULL))` Initialise la graine (seed) de la fonction rand sur l'horloge.
- `rand()%k` Retourne un entier entre 0 et $k-1$.

- Exercice 2 - Création de tas -

Implémenter la fonction `void Entasser(int n, int T[], int i)` qui entasse le nœud i dans le tableau T de taille utile n. Tester votre fonction, notamment sur l'exemple $T1 = [7, 12, 4, 8, 10, 2, 1, 1, 3, 9]$, avec $n = 10$ et $i = 0$. La solution devra être $T1 = [12, 10, 4, 8, 9, 2, 1, 1, 3, 7]$.
Utiliser cette fonction pour implémenter `void Tas(int n, int T[])` qui transforme le tableau T de taille utile n en un tas. Tester votre fonction, notamment sur l'exemple $T2 = [5, 1, 6, 8, 2, 10]$, avec $n = 6$. La solution devra être $T2 = [10, 8, 6, 1, 2, 5]$.

- Exercice 3 - Tri par tas -

Écrire la fonction `void Trier(int n, int T[], int Ttrie[])` qui implémente l'algorithme TRI-PAR-TAS du cours où $Ttrie$ contiendra les valeurs de T triées en ordre croissant. Tester votre algorithme sur les tableaux $T1$, $T2$ et sur des tableaux générés aléatoirement.
Réaliser une implémentation `Trier2` de TRI-PAR-TAS qui trie sur place, c'est-à-dire sans l'utilisation du tableau $Ttrie$ annexe.

- Files de priorité -

- Exercice 4 - File de priorité -

Proposer une implémentation d'une file de priorité par un tas qui contiendra notamment les fonctions :

- `void AjoutElem(int n, int T[], int val)` qui ajoute un élément de valeur *val* à T (et conserve la structure de tas de T).
- `int ExtraitMax(int n, int T[])` qui supprime l'élément de T de valeur maximale et retourne cette valeur (et conserve la structure de tas de T).
- `AugmentePriorite(int n, int T[], int i, int gain)` qui augmente de *gain* la valeur de l'élément i de T (et conserve la structure de tas de T).
- `DiminuePriorite(int n, int T[], int i, int perte)` qui diminue de *perte* la valeur de l'élément i de T (et conserve la structure de tas de T).