Fiche de TP : Prélude au Projet

Cette fiche porte sur la mise au point d'un outil indispensable à la réalisation efficace du projet.

Elle porte sur l'écriture d'une classe **`Tableau`** extensible, dont la vocation sera de stocker des données avec accès direct de manière contiguë. On pourra **sans perte de données** lui ajouter ou lui supprimer des éléments.

Besoin - 1 *Générécité*

La classe sera *générique*, avec comme argument *template* le type de données à stocker.

Besoin - 2 *Données membres*

Cette classe disposera de 3 données membres

`m_valeurs` un pointeur sur le type générique permettant de stocker les données du tableau,

`m_capacite` un entier qui contient la capacité actuelle du tableau,

`m_taille` un entier qui indique la taille occupée par les données.

Besoin - 3 *Fonctions membres*

Cette classe fournira :

Cycle de vie
- Un constructeur utilisant en argument optionnel la capacité du tableau (par défaut 0)
- Un constructeur par copie.
- Un destructeur.

Opérateurs
- Une surcharge de l'opérateur = qui permettra de recopier des Tableaux. Lors de la copie la capacité sera la capacité minimale pour contenir une copie.
- Deux fonctions d'accès (lecture seule et lecture/écriture) aux données par surcharge de l'opérateur `[]`.

Opérations
- Une fonction **`void reserve(const int& nouvelle_capacite)`** qui permettra d'augmenter la capacité sans perte de données. Si la capacité proposée est plus petite que l'ancienne, aucune action n'est effectuée.
- Une fonction **`void retaille(const int& nouvelle_taille)`** qui change la taille des données utilisées. La capacité doit être augmentée au besoin.
- Une fonction **`void ajoute(const T& t)`** qui permettra d'ajouter une élément à la fin du tableau. La taille sera mise à jour et si la capacité du tableau n'est pas suffisante, elle sera augmentée (doubler la capacité est une bonne stratégie).
- Une fonction **`void supprime(const int& i)`** qui supprimera la case `i` du tableau. Pour cela, la dernière valeur contenue dans le tableau sera copiée en case `i` et la taille sera réduite de 1.

Accesseurs
- Une fonction retournant la taille des données stockées.
- Une fonction retournant la capacité

Autres besoins pour le projet

Afin de faciliter la réalisation du projet, il serait judicieux de fournir une interface pour les classes suivantes :

Besoin - 4 ***template class TinyVector<typename T, int N>***

Classe modélisant un tableau static de **`N`** objets de type **`T`**.

Cette classe doit surcharger les opérateurs nécessaires pour fournir une algèbre raisonnable. Des opérateurs d'accès surchargeant l'opérateur **`[]`** doivent être programmés pour accéder aux données de manière sécurisée.

On surchargera l'opérateur chevron (**`<<`**) pour permettre l'écriture de TinyVector sur la console.

Besoin - 5 ***N2***

En se servant de la classe précédente, on définira dans le fichier `N2.hpp` le type **`N2`** qui modélisera $\mathbb{N}^2$.

Besoin - 6 ***class Pixel : public N2***

Cette classe servira à décrire un pixel de l'image originale. La classe mère **`N2`** permettra de stocker les coordonnées du pixel et on ajoutera un membre entier qui contiendra la couleur.

On définira les fonctions membres **`x()`** et **`y()`** qui permettront d'accéder aux coordonnées ainsi qu'une fonction **`couleur()`** qui donnera accès à la couleur du **`Pixel`**. On surchargera l'opérateur de copie et on définira les constructeurs.

Enfin, on permettra l'écriture de pixels sur la sortie standard.

Besoin - 7 *Essayer de vous renseigner sur la notion de* ***`Quadtree`***
