**Sorbonne Université** **M1 Mathématiques**
**2019–2020** **MU4MA056 : Programmation en C++**

**TP noté 1 : empilement de cubes**

IMPORTANT : à l'issue du TP, chaque groupe envoie par email les fichiers `empilementcube.hpp` , `empilementcube.cpp` et `test.cpp` et `bebe.cpp` . **Il est impératif de mettre en commentaire, dans tous les fichiers, les nom, prénom et n° d'étudiant de chacun des membres du groupe.**

Dans ce TP, on se propose d'empiler des cubes les uns sur les autres pour faire une tour la plus haute possible. Malheureusement, on n'est pas très habile et on commet des erreurs de positionnement de chaque nouveau cube. À quelle hauteur moyenne peut-on arriver selon le type de cube et l'erreur commise ?

**Un peu de mathématiques.** Un cube est caractérisé par deux coordonnées dans le plan $(x, y) \in \mathbb{R}$, une ordonnée, une largeur $w \in \mathbb{R}_+$ et une masse $m = w^3$ (intuitivement proportionnelle au volume du cube).

Un empilement de $n$ cubes est une suite finie de cubes $(X_i, Y_i, W_i, M_i)_{0\leq i\leq n-1}$. Un empilement est stable si et seulement si $n \leq 1$ ou bien, pour chaque cube d'indice $0 \leq i \leq n-2$, le barycentre des $n-i-1$ cubes d'indice $j$ tel que $i < j \leq n-1$ (i.e. *au-dessus* de lui) a des coordonnées $(b_i^{(1)}, b_i^{(2)})$ telle que $X_i - W_i/2 < b_i^{(1)} < X_i + W_i/2$ et $Y_i - W_i/2 < b_i^{(2)} < Y_i + W_i/2$.

Nous rappelons que le barycentre d'une collection de $p$ cubes de coordonnées $(x_k, y_k)_{1\leq k\leq p}$ et de masses $(m_k)_{1\leq k\leq p}$ a des coordonnées $(b^{(1)}, b^{(2)})$ donnée par :

$$b^{(1)} = \frac{\sum_{k=1}^{p} x_k m_k}{\sum_{k=1}^{p} m_k} \qquad\qquad b^{(2)} = \frac{\sum_{k=1}^{p} y_k m_k}{\sum_{k=1}^{p} m_k} \tag{1}$$

La hauteur totale d'un empilement de cubes est donnée par $H = \sum_{0\leq i\leq n-1} W_i$.

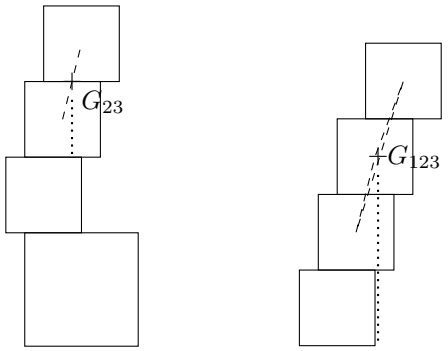


FIGURE 1 – Exemples d'empilements (vus latéralement donc on ne voit pas les coordonnées $y_k$) de cubes issus des fichiers `empilement1.dat` et `empilement2.dat` . Celui de gauche est stable, celui de droite instable : le barycentre $G_{123}$ des trois cubes du haut a une abscisse en dehors du premier cube.

**Implémentation en C++.** Nous imposons les classes suivantes que vous écrirez dans un fichier `empilementcube.hpp` :

```
#ifndef CLASS_EMPILEMENT
#define CLASS_EMPILEMENT
class Cube {
        public:
                .......
                friend class Empilement;
        private:
                double x;//première coordonnée
                double y;//deuxième coordonnée
                double width;//largeur
};
class Empilement {
        public:
                .......
                bool add_new_cube(double posx,double posy,double hw);
                void compute_barycenters();
                bool is_stable() const;
        private:
                std::vector<Cube> vcube;//taille: nb de cubes
                std::vector< std::pair<double,double> > barycenter_above;
                        //taille: nb de cubes -1
};
#endif
```

L'attribut `vcube` stocke les cubes en utilisant la première classe, l'attribut `barycenter_above` les coordonnées des barycentres tels que `barycenter_above[i]` correspond aux coordonnées du barycentres des $n-i-1$ cubes d'indices $j$ tels que $i < j \leq n-1$. Le premier vecteur est de tailles $n$ et le second de taille $n-1$ (ou 0 si le premier est vide).

Nous rappelons que l'on peut accéder au premier et au deuxième élément d'une variable `p` de type `std::pair<T1,T2>` par `p.first` et `p.second` (sans parenthèse, ce sont des attributs publics).

***Attention :*** **On veillera à étiqueter `const` toutes les méthodes et les arguments nécessaires.**

**1.** Recopier ce code dans un fichier `empilementcube.hpp` et ajouter les `include` nécessaires. Préparer également un fichier `empilementcube.cpp` avec les inclusions de bibliothèques nécessaires.

**2.** Écrire un constructeur de `Cube` qui prend comme argument trois réels $(x_0, y_0, w_0)$ et crée un cube de coordonnées $(x_0, y_0)$ et de largeur $w_0$.

**3.** Écrire un constructeur par défaut de `Empilement` qui définit un empilement de cube vide.

**4.** Écrire un constructeur de `Empilement` qui prend comme argument un entier $n$ et trois réels $(x_0, y_0, w_0)$ et crée un empilement parfait de cubes, tous aux positions $(x_0, y_0)$ et de largeur $w_0$.

**5.** Ajouter dans `Empilement` un accesseur `size()` au nombre de cubes de l'empilement.

**6.** Ajouter dans `Empilement` des accesseurs `get_x(int i)` , `get_y(int i)` qui renvoient les premières et deuxièmes coordonnées du $i$-ème cube.

**7.** Ajouter une méthode `height()` qui calcule et renvoie la hauteur totale de l'empilement. *Il y aura un bonus pour l'utilisation de* `std::accumulate` *de la STL présent dans* `<numeric>` *.*

**8.** Ajouter un mutateur `width(int i)` qui renvoie l'accès en écriture à la largeur du $i$-ème cube (i.e. `E.width(7)= 3.;` met la position du 7-ème cube de l'empilement `E` à 3). *Vous expliquerez éventuellement en commentaire pourquoi il vaut mieux le mettre en partie privée.*

**9.** Surcharger, pour `Empilement` les opérateurs `<<` et `>>` de telle sorte qu'un empilement soit écrit ou lu avec le format suivant :

```
N
X_0  Y_0  W_0
X_1  Y_1   W_1
...
X_N-1  Y_N-1 W_N-1
```

La première ligne est le nombre de cubes dans l'empilement, les lignes suivantes correspondent aux positions et largeurs de chaque cube du premier au dernier. *Vous pourrez, si vous le souhaitez, tout d'abord surcharger* `<<` *et* `>>` *pour la class* `Cube` *.*

**Pour l'instant, on ne remplit pas encore le vecteur `barycenter_above` .**

**10.** Écrire un programme complet `test.cpp` qui fait les choses suivantes :

- on définit un empilement parfait `E` de 5 cubes identiques de largeur 1 à la même position $(0, 0)$ et on l'affiche dans le terminal ;
- on lit les deux fichiers `example1.dat` et `example2.dat` dans deux empilements `E1` et `E2` ;
- on les affiche dans le terminal *[on doit retrouver la même chose que dans chaque fichier]* ;
- on affiche dans le terminal la hauteur de chaque empilement *[on doit trouver* 5 *pour* `E` *,* 9 *pour* `E1` *et* 8 *pour* `E2` *]* ;

***Attention :*** **tant que vous n'obtenez pas les résultats attendus à la question précédente, il est inutile de continuer.**

**11.** Écrire le code de la méthode `compute_barycenters()` qui redimensionne le vecteur privé `barycenter_above` à la taille $n - 1$ où $n$ est le nombre de cubes puis calcule, pour chaque cube $i$ avec $0 \leq i \leq n - 2$, le barycentre des cubes strictement au-dessus du $i$-ème selon la formule (1).

*[Il existe une solution naïve avec complexité $O(n^2)$ ; vous aurez un bonus si vous trouver une solution avec complexité $O(n)$.]*

**12.** Écrire le code de la méthode `is_stable()` qui teste si un empilement est stable ou non. On supposera que cette méthode s'utilisera *toujours* après `compute_barycenters()` : il n'y a donc rien à recalculer du côté des barycentres.

**13.** Mettre à jour la surcharge de l'opérateur `>>` en ajoutant le calcul des barycentres après la lecture des informations. Mettre à jour le fichier `test.cpp` en affichant si les empilements `E1` et `E2` sont stables ou non. *[oui pour le premier, non pour le second].*

**14.** Écrire le code de la méthode `add_new_square(x,y,w)` qui :

- ajoute un nouveau cube de taille `w` avec les coordonnées `x` et `y`
- recalcule les barycentres,
- teste la stabilité et renvoie le résultat.

On utilisera bien évidemment les questions précédentes.

**15.** *(ce n'est pas nécessaire pour la suite)* Au lieu d'une méthode `add_new_square` comme à la question précédente, écrire une surcharge de l'opérateur `+=` tel que `E+= Cube(x,y,z)` soit équivalent à `E.add_new_square(x,y,z)` .

**16.** Compléter le programme `test.cpp` en vérifiant que l'ajout d'un premier cube à la position $(0.5, 1)$ et de largeur 1 sur `E1` le laisse stable et que l'ajout d'un deuxième cube à la position $(0.5, 1.)$ et de largeur 2 le rend instable.

***Attention :*** **tant que vous n'obtenez pas les résultats attendus à la question précédente, il est inutile de continuer.**

**17.** On s'intéresse à présent à la hauteur moyenne d'un empilement fait par un bébé aux capacités motrices encore balbutiantes. Le premier modèle est le suivant : tous les cubes sont supposés de largeur 1. Lors de l'ajout d'un nouveau cube, le bébé essaie de le mettre exactement sur le précédent mais avec une erreur due à sa maladresse. Mathématiquement, cela correspond à des largeurs $W_i = 1$ constantes et des positions données par $X_{i+1} = X_i + U_{i+1}$ et $Y_{i+1} = Y_i + V_{i+1}$ où les $U_i$ et les $V_i$ sont des variables i.i.d. centrées uniformes sur $[-1/4, 1/4]$.

Écrire un programme `bebe.cpp` qui calcule, sur une moyenne de 100 échantillons, la hauteur moyenne de l'empilement au moment où il s'effondre *[vous devriez trouver autour de 18]*. **L'aléatoire sera géré avec la bibliothèque `<random>` de C++11 et un générateur de type `std::mt19937` .**

**18.** Supposons que nous ayons à présent deux types de cubes de largeurs $0,9$ et $1,1$ et que le bébé choisisse à chaque fois un cube au hasard uniformément. Compléter le programme précédent pour étudier à nouveau la hauteur moyenne de l'empilement au moment où il s'effondre.

**19.** Surcharger l'opérateur `+` sous forme d'une fonction (et non d'une méthode) qui correspond à la superposition verticale de deux empilements de cube. Pour l'empilement final, on recalculera les barycentres avec de renvoyer le résultat. Ajouter des lignes de test dans `test.cpp` .

**20.** *Bonus : nous avons supposé que les cubes ne pouvaient pas tourner sur eux-mêmes. Ce n'est pas le cas dans la réalité car les cubes peuvent tourner sur eux-mêmes. Écrire une classe* `EmpilementRotation` *inspirée de* `Empilement` *qui permette cette description. Vous êtes entièrement libres des outils utilisés.*