Université Pierre et Marie Curie
Master 2 IFMA
Programmation en C++

Examen du 22 décembre 2017
Durée : 2 heures

— *Tous les fichiers rendus devront contenir vos* **nom et prénoms**.
— *Le fait d'envoyer des fichiers qui ne se compilent pas correctement sera sanctionné.*
— *Les deux parties sont indépendantes entre elles.*

1 Temps de sortie pour une marche aléatoire

On considère une marche aléatoire sur $\mathbb{Z}^2$: plus précisément, on considère la suite de variables aléatoires à valeurs dans $\mathbb{Z}^2$ définie par

$$X_n = \sum_{k=1}^{n} U_k,$$

où les U_k sont indépendants et de loi uniforme sur $\{(1,0),(0,1),(-1,0),(0,-1)\}$. En particulier, X_0 vaut $(0,0)$. On s'intéresse au couple (X_τ, τ) où τ est le premier temps de sortie de l'ensemble $E_N = \{-N+1,\dots,N-1\}^2$. Dans le programme, on choisira $N = 10$. Noter que la variable X_τ prend nécessairement une valeur de la forme $(\pm N, k)$ ou $(k, \pm N)$ avec $-N < k < N$.

1. Dans des fichiers `marche.hpp` et `marche.cpp`, écrire une classe `marche` contenant en membre privé :
   — l'entier `N` correspondant à la taille du domaine considéré ;
   — les deux coordonnées `x` et `y` de la position de la marche ;
   — une variable `t` correspondant au nombre de pas déjà effectués par la marche.
   — une variable `histo` de type `std::vector<std::vector<int> >` qui servira à stocker l'historique des points/temps de sortie de la marche.
2. Écrire un constructeur initialisant la position au point $(0,0)$ et le temps à 0. La taille du domaine est fournie en argument. Le vecteur `histo` doit être de taille `N`, et les `histo[i]` doivent être vides.
3. Faut-il récrire le constructeur par copie, l'opérateur `=` et le destructeur pour cette classe ? Si oui, le faire.
4. Écrire une méthode `un_pas` faisant avancer la marche (et le temps) d'un pas si la marche n'est pas encore sortie de E_N, et qui ne fait rien sinon.
5. Écrire une méthode `r_a_z` qui remet la marche à zero : la marche repart du point $(0,0)$, au temps 0.
6. Écrire une méthode `sortie` qui simule des pas de la trajectoire jusqu'à ce que la marche soit sortie de E_N.

7. Écrire une méthode `ajoute_histo` qui, quand la marche est situé en un point de la forme $(\pm N, \pm k)$ ou $(\pm k, \pm N)$, avec $0 \leq k < N$, ajoute à `histo[k]` le temps mis pour atteindre ce point.

   Écrire un accesseur pour la variable `histo`.
8. Écrire un programme qui calcule un histogramme de 10000 simulations de X_τ. Pour $X_\tau = (\pm N, \pm k)$ ou $X_\tau = (\pm k, \pm N)$, avec $0 \leq k < N$, on ne retiendra que la valeur de k.
9. Écrire un programme qui calcule par la méthode de Monte Carlo une approximation de $\mathbb{E}(\tau | X_\tau = (N, k))$, pour $0 \leq k < N$.

2 Le maçon aléatoire

Soit $(X_n)_{n\in\mathbb{N}}$ une marche aléatoire simple sur $\mathbb{Z}$. Plus précisément, on considère $(U_n)_{n\geq 1}$ une suite de variables aléatoires indépendantes de loi uniforme sur $\{-1, 1\}$, et on pose, pour $n \geq 0$

$$X_n = \sum_{k=1}^{n} U_k.$$

De plus la marche construit un "mur" en déposant à chaque pas une brique sur sa position actuelle. Si il y a déjà des briques sur la position de la marche, la nouvelle brique est posée par dessus les autres. La hauteur du mur au point $a \in \mathbb{Z}$ au temps $n \geq 0$ est donc

$$H_n^a = \sum_{k=0}^{n} \mathbf{1}_{X_k = a}.$$

On veut tracer la suite des couples $(X_n, H_n^{X_n})_{n\in\mathbb{N}}$, qui correspond aux positions des briques successivement déposées.

Pour représenter informatiquement ce processus, on va avoir besoin à chaque instant n de connaître la famille $(H_n^a)_{a\in\mathbb{Z}}$, qui est une suite indexée par les entiers positifs *et* négatifs. Pour cela, on va écrire une classe `hauteurs`[1] qui représente les suites doubles.

1. Dans des fichiers `hauteurs.hpp` et `hauteurs.cpp` écrire le code d'une classe `hauteurs` qui contient comme membres privés :
   — deux pointeurs `gauche` et `droite` qui vont pointer vers les adresses de deux tableaux d'entiers ;
   — deux entiers `lg` et `ld` qui vont représenter le nombre d'entiers stockés respectivement dans les deux tableaux `gauche` et `droite`.
2. Écrire un constructeur par défaut pour cette classe, initialisant les deux tableaux avec une taille de `1` et contenant chacun la valeur `0`.
3. Faut-il récrire le constructeur par copie, l'opérateur `=` et le destructeur pour cette classe ? Si oui, le faire.
4. Écrire une surcharge de l'opérateur `[]` permettant d'accéder aux éléments des deux tableaux `gauche` et `droite`. Si `X` est de type `hauteurs`, on veut que les éléments de `X.droite` soient accessible par `X[0]`, `X[1]`, `X[2]`, etc., et que ceux de `X.gauche` le soient par `X[-1]`, `X[-2]`, `X[-3]`, etc. Si `n` correspond à un entier qui est hors de la capacité mémoire de `gauche` et `droite`, `X[n]` renverra `0`.

1. Le but est ici d'écrire un équivalent de la classe de la bibliothèque standard `std::deque`. On ne va donc as utiliser cette classe, même si cela serait la solution naturelle en pratique.

5. Écrire une méthode `ajout` qui prend en argument un entier `n` et telle que `X.ajout(n)` augmente de `1` la valeur de `X[n]`. Si besoin, on augmentera la taille du tableau `gauche` ou `droite` en la *multipliant par deux*. En particulier, les tailles de `gauche` et `droite` seront toujours des puissances de 2.
6. Écrire un programme qui utilise la classe `hauteurs` pour afficher sur la sortie standard les couples $(X_n, H_n^{X_n})$. Une représentation graphique de ces couples ressemblera à la figure suivante (à gauche 100 pas, à droite 1000 pas) :