Université Pierre et Marie Curie **Master M1 "Mathématiques et Applications"**
Période 2 **Programmation en C/C++**

TP 8 : Classes abstraites, héritage et programmation générique

Le but du présent TP est d'utiliser la notion d'héritage pour manipuler efficacement des systèmes dynamiques ou stochastiques et d'utiliser des *templates* pour pouvoir gérer des espaces d'état aussi généraux que possible. Le dernier exemple de la seconde section permettra de s'initier à la classe `list` de la bibliothèque STL.
Pour chaque question, vous veillerez à tester les différentes classes dans une fonction `main()`.

1 Suites récurrentes à valeurs réelles

Les systèmes dynamiques ou les chaînes de Markov ont la propriété d'être caractérisés par une formule de récurrence et un état initial. Pour décrire un tel système, nous introduisons la classe abstraite :

```
class RealDynSyst{
    public:
        RealDynSyst(double val);
        virtual RealDynSyst & operator++() = 0;
        double eval() const ;
        unsigned niter() const ;
        void reinit() ;
        friend ostream& operator<<(ostream &, const RealDynSyst & S);
    protected:
        double state;
        unsigned iter;
        double init_state;
};
```

La variable `iter` contient l'entier n courant, la variable `init_state` contient la valeur initiale u_0, la variable `state` contient la valeur courante u_n. Le premier constructeur initialise toute la suite à sa valeur initiale ; l'opérateur `++` (préfixé) actualise la suite pour passer de n à $n+1$. Les méthodes `eval` et `niter` renvoient les valeurs de `state` et `iter` et enfin la méthode `reinit` réinitialise tout le système à sa valeur initiale. La méthode `operator++` est virtuelle pure donc la classe `RealDynSyst` est abstraite.

1 *Implémenter toutes ces méthodes (sauf* `operator++()` *qui est virtuelle pure).*

2 *Définir la fonction globale (amie) qui surcharge l'opérateur d'injection* `<<`*.*

3 *Écrire deux classes dérivées* `Arith` *et* `Geom` *de* `RealDynSyst`

```
class Arith : public RealDynSyst{ ________ };
class Geom : public RealDynSyst{ ________ };
```

qui implémentent les suites arithmétiques et géométriques. Réfléchir au préalable à l'information supplémentaire nécessaire.

4 *(suite de Fibonacci) Écrire une classe dérivée* `Fibonacci` *de* `RealDynSyst` *qui implémente la suite suivante :*

$$u_{n+2} = u_{n+1} + u_n \tag{1}$$

avec initialisation de u_0 et u_1. Bien réfléchir aux champs privés à ajouter...

5 *(cas stochastique) Écrire une classes dérivée* `SimpleRandomWalk` *de* `RealDynSyst` *qui implémente la marche aléatoire simple $S_0 = a$ et $S_{n+1} = S_n + U_{n+1}$ où $(U_n)_{n\in\mathbb{N}^*}$ est une suite de variables aléatoires indépendantes et identiquement distribuées telle que $\mathbb{P}(U_k = 1) = \mathbb{P}(U_k = -1) = 1/2$.*

6 *Implémenter une fonction (polymorphe) de prototype*

```
int temps_atteinte(RealDynSyst & S, double L);
```

qui renvoie le premier indice `n` *de la suite vérifiant $u_n > L$ s'il est inférieur à* `1e6` $= 10^6$ *ou* `1e6` *sinon.*

7 *(bonus à faire à la maison) Écrire une classe*

```
class Bessel: public RealDynSyst {
    private:
        SimpleRandomWalk X;
        double min;
}
```

qui elle-même hérite de `RealDynSyst` *et contient deux champ privés* `SimpleRandomWalk X` *et* `double min` *qui implémente la transformation dite de Pitman suivante à partir de la suite* X_n *:*

$$Y_n = X_n - 2 \inf_{0 \leq m \leq n} X_m \tag{2}$$

2 Programmation générique

On veut à présent généraliser l'approche précédente à des systèmes dynamiques à valeurs dans un type quelconque. Pour cela, on veut usage de *templates*.

8 *Définir une classe générique* `DynSyst<T>`

```
template <typename Statespace>
class DynSyst;
```

qui généralise la classe précédente aux suites dont les valeurs ont un type `Statespace` *générique (potentiellement très compliqué !).*

9 *Adapter les classe* `Arith` *et* `Geom` *pour qu'elles héritent de la classe générique* `DynSyst<T>` *instanciée avec le type* `double` *c'est à dire*

```
class Arith2 : public DynSyst<double>{
public:
    Arith2(double val, double increment): DynSyst<double>(val),incr(increment) {};
    _______
}
```

10 *Définir une structure* `RealPair` *qui contient 2 champs privés réels* `a` *et* `b` *de type* `double`*. Ecrire une classe* `Fibo` *qui hérite de la classe générique* `DynSyst<T>` *instanciée avec le type* `RealPair` *c'est à dire*

```
class Fibo : public DynSyst<RealPair>{ ________ };
```

Il faut déclarer `Fibo` *comme classe amie de la classe* `RealPair`*.*

11 *Nous allons à présent modéliser une file d'attente très simple. Nous considérons un guichet où arrive chaque minute un nombre de clients aléatoires de loi binomiale de paramètres* (n, p)*. Chaque minute également, le guichet traite exactement un client (si la file est non-vide). L'état de la file sera ainsi un objet de type* `list<unsigned int>` *(où* `list` *est un conteneur de la STL défini dans la bibliothèque éponyme) : chaque entier positif de la liste correspond à la minute à laquelle le client correspondant est arrivé.*
Pour information, la classe `list` *n'est pas munie de l'opérateur* `[]` *mais les fonctionnalités suivantes sont disponibles :*

1. *les méthodes* `push_front(x)` *et* `push_back(x)` *qui ajoutent un élément au début ou à la fin de la liste ;*
2. *les méthodes* `pop_front()` *et* `pop_back()` *qui retirent un élément au début ou à la fin de la liste ;*
3. *des iterateurs* `list<unsigned it>::iterator` *et* `list<unsigned it>::const_iterator` *qui permettent de pointer vers un élément de la liste et qui sont munis de l'opérateur* `++` *;*
4. *des méthodes* `begin` *et* `end` *qui renvoient un itérateur vers le premier ou le dernier élément de la liste.*
5. *la méthode* `empty` *qui renvoie un* `bool` *selon que la liste est vide ou non.s*

Créer une classe `FileAttente` *qui hérite d'une classe* `DynSyst` *avec le type adéquat, à laquelle on ajoutera deux champs pour les paramètres de la loi binomiale d'arrivée des clients, une surcharge de* `<<` *qui affichera une liste d'attente et enfin une fonction* `attentemoyenne` *qui renvoie le temps d'attente moyen de tous les clients de la file à un moment donné.*

12 *Amusez-vous dans la fonction* `main()`*, à utiliser toutes ces classes et enrichissez-les selon vos idées !*