ISIMA 3ème année - MODL/C++
TP 2 : Généricité

Pré-requis

On reprend le premier exercice du TP précédent. On suppose que les classes **Point**, **Cartésien** et **Polaire** ont été correctement définies et sont fonctionnelles, et que les moyens de conversion entre **Cartésien** et **Polaire** existent. Copiez le code source de ces classes dans le répertoire tp_2/src et modifiez CMakeLists.txt pour que CMake les prenne en compte.

Ce TP vous fera également découvrir un autre outil de génie logiciel : **Git**. Il s'agit d'un **système de gestion de version** permettant notamment de conserver un historique des modifications d'un projet à travers l'identification de versions, et donc de pouvoir revenir en arrière si nécessaire. Il permet également un travail collaboratif sur un projet. Un document introductif à Git vous est fourni : il vous explique les principes et les commandes principales de l'outil.

Exercice 1 - Nuage

Il vous est demandé d'utiliser l'outil Git pour accompagner le développement de cet exercice. Vous devez donc créer un dépôt (*repository*) en local sur votre compte, et valider une nouvelle version à chaque fin de question (*commit*).

1) Définir la classe **Nuage**, générique sur le type de points à contenir. Utiliser la classe std::vector pour stocker les points. Cette classe devra fournir des itérateurs, à la manière STL (méthodes begin() et end(), type iterator). Tests 1-2

2) Définir une **fonction** générique barycentre_v1() prenant en argument un nuage de points et retournant le barycentre, supposé du même type que les points. Proposer une solution générique qui s'appuie sur la formule connue pour les cartésiens (utiliser des conversions). Tests 3-4a

3) Connaissant la formule du barycentre en coordonnées polaires (si vous ne l'avez pas, faites une moyenne des angles et des distances), proposer une spécialisation pour les polaires de la fonction générique barycentre_v1(). Test 4b

Dans la prochaine question, vous devez proposer une solution alternative à celle des questions précédentes, il vous est donc demandé de créer une nouvelle branche dans votre dépôt, afin de bien identifier les deux possibilités.

4) On souhaite que la fonction de barycentre s'affranchisse aussi du type du conteneur de points. Pour cela, la classe **Nuage** propose une interface similaire aux conteneurs STL, à savoir des méthodes begin() et end(), ainsi qu'un type interne iterator. Proposer une fonction barycentre_v2() qui puisse fonctionner indifféremment sur les principaux conteneurs STL et la classe **Nuage**. Tests 5-7

Exercice 2 - Métaprogrammation

Un calcul peut parfois être évalué en partie à la compilation (par exemple la somme de deux constantes), le reste étant évalué à l'exécution du programme. Les génériques peuvent être utilisés pour augmenter et automatiser la part de calcul évaluée à la compilation dans une expression, et de ce fait, accélérer son évaluation à l'exécution. On appelle cette technique l'évaluation partielle.

1) Factorielle et fonctions récursives

   Certains algorithmes demandent parfois de manipuler des valeurs constantes non-triviales qu'il est maladroit de cacher sous la forme d'une simple constante nommée. Par exemple, le calcul de la série de Taylor d'une fonction peut faire appel à la série de constantes *1!*, *2!*, *3!*, …, *n!*. Une solution classique revient à définir des constantes :

```
static const int FACT2 = 2;
static const int FACT3 = 6;
static const int FACT4 = 24;
static const int FACT5 = 120;
```

   A la place, proposez une structure générique contenant un membre de classe de type unsigned long et qui permet d'effectuer le calcul de *n!* de manière statique. Au final, l'utilisation de cette structure se fera de la manière suivante :

```
unsigned long fact20 = Factorielle<20>::valeur;
```
Test 8

2) Calcul de séries de Taylor

   L'expression d'une fonction *f(x)* sous la forme d'une série de Taylor fait intervenir l'ordre *N* et la valeur du point *x* où l'on évalue la fonction. L'idée est de fournir une structure générique qui va construire à la compilation et de manière récursive la série de Taylor de la fonction à l'ordre voulu. À l'exécution, ce développement sera évalué en un point quelconque. On a donc une partie statique et une partie dynamique. Ainsi, par exemple, l'évaluation en *1.5* de la série de Taylor à l'ordre *5* d'exponentielle sera donné par :

```
double val = Exponentielle<5>(1.5);
```

   Fournir la série de Taylor des fonctions exponentielle, sinus et cosinus. On rappelle que :

$$\begin{aligned}
\exp(x) &\approx \sum_{k=0}^{N} \frac{x^k}{k!} \\
\cos(x) &\approx \sum_{k=0}^{N} (-1)^k \frac{x^{2k}}{(2k)!} \\
\sin(x) &\approx \sum_{k=0}^{N} (-1)^k \frac{x^{2k+1}}{(2k+1)!}
\end{aligned}$$

   Proposer les structures génériques nécessaires à ces calculs. Test 9-12