

Exercice sur l'héritage

Programmation C++ avancée

1 Enoncé

- Créer une classe **Fonction** avec 3 méthodes principales :
 1. `float operator()(float x) const` qui évalue la fonction en `x`
 2. `Fonction* derivee() const` retournant la fonction dérivée (voir section 2)
 3. `float inverse(float y) const` calculant l'antécédent de `y` par la méthode de Newton (voir section 3)
- Les deux premières sont virtuelles pures, car on ne sait pas quoi faire concrètement pour une fonction générique. Donc **Fonction** est une classe abstraite.
- Écrire une classe **Polynome**, enfant de **Fonction**, et une classe **Affine** enfant de **Polynome**. Il suffit pour la classe affine (fonction $x \rightarrow ax + b$) de faire un constructeur spécifique prenant les paramètres a et b plutôt qu'un tableau de coefficients.
- Écrire une classe **Trigo**, enfant de **Fonction**, prenant comme paramètre un chaîne de caractères ("`cos`", "`sin`" ou "`tan`").
- Utiliser ces fonctions pour calculer $27^{1/3} = 3$ (inverse du polynôme $x \rightarrow x^3$) et $4 * \text{atan}(1) = \pi$ (inverse de la fonction **Trigo** associée à "`tan`").

2 Calcul de la dérivée

- Pour la classe **Polynome**, la dérivée est à nouveau un polynôme, et il est facile d'implémenter sa méthode `derivee`.
- Pour la classe **Trigo**, on a un problème car la dérivée de la fonction `tan` n'est ni une fonction trigonométrique ni un polynôme¹. Plus généralement, pour les classes dont on ne peut pas représenter la dérivée, on procède de la façon suivante :
 - On crée une sous-classe **Derivee** de **Fonction**.
 - On ajoute un champ `Fonction* primitive` à la classe **Derivee**.

1. mais c'est un polynôme trigonométrique !

- `Derivee::operator()` renvoie la dérivée calculée par différence finie :

$$f'(x) \sim \frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon}$$

avec $\epsilon = 10^{-2}$. La fonction f de cette formule est le champ `primitive`.

- `Trigo::derivee` doit donc retourner une nouvelle fonction de type `Derivee` dont le champ `primitive` est une copie de l'objet appelé.
- Pour éviter qu'une `Derivee` soit dépendante d'une fonction intégrale dont elle n'a pas le contrôle de la durée de vie, on préfère faire des copies de fonctions. On va donc ajouter un "constructeur virtuel" `clone` au niveau de `Fonction` (abstrait) et au niveau de ses sous-classes, qui renvoie une copie (allouée par `new`) de l'objet.
- La `Fonction` qui sera passée au constructeur de `Derivee` sera donc immédiatement clonée pour remplir le champ `primitive` et la `Derivee` va pouvoir vivre sa vie sans dépendre de l'objet `Fonction` avec lequel on l'a créée.
- Noter que la bonne définition de `Fonction* Derivee::derivee() const` (une ligne de code) permet même de calculer des dérivées seconde et supérieures ! Faire calculer par exemple $\tan'(\pi/4) = 2$ et $\tan''(\pi/4) = 4$.

3 Méthode de Newton

- On fait au plus 100 itérations :

$$x_{i+1} = x_i + \frac{y - f(x_i)}{f'(x_i)}$$

tant que $|x_{i+1} - x_i| > 10^{-5}$.

- Pour éviter la dérivée nulle en 0 de la fonction cube, partir de $x_0 = 1$.
- À noter que la classe `Affine` peut avoir sa propre implémentation de `inverse`, car le calcul est trivial. On n'y gagne cependant pas grand chose, car la méthode de Newton converge en une seule itération dans ce cas !