Sorbonne Université **M1 Mathématiques**
2018–2019 **4M056 : Programmation en C++**

TP noté 1 : un semblant de calcul formel

Le but de ce TP est de proposer une version jouet de ce que pourrait être un système de calcul formel pour dériver des fonctions.

Chaque binome rendra trois fichiers appelés respectivement `fonction.hpp`, `fonction.cpp` et `test_fonction.cpp` dont les premières lignes seront des commentaires indiquants les noms et numéros d'étudiants des deux membres du binôme. Le code demandé necessite le standard C++11. On compilera donc avec l'option `--std=c++11`.

L'objet central sera la classe `Fonction` dont les instances représentront les fonctions que l'on veut dériver. Cette classe contiendra deux champs `fct` et `drv` de type chaîne de caractères, qui encoderont l'expression de la fonction et de sa dérivée respectivement. Afin de faciliter l'implémentation de la composition de fonctions, les endroits où une variable peut être placée sera indiquée par le caractère `'.'` (point). Par exemple, la chaîne de caractère pour la fonction exponentielle sera `"exp(.)"`.

On voudra construire dans un premier temps des fonctions de trois manières différentes, suivant les paramètres du constructeur :

— en donnant une seule chaîne de caractères, pour représenter une fonction dont on ne connaît que le nom. Par exemple, en passant le paramètre `"f"`, on représente la fonction `f(.)`, dont la dérivée est `f'(.)`.

— en donnant deux chaînes de caractères, pour représenter une fonction usuelle dont on veut préciser l'expression de la dérivée. Par exemple, la fonction cos sera représentée par l'appel `Fonction("cos(.)","-sin(.)")`.

— en donnant un entier n, pour représenter la fonction $x \mapsto x^n$.

On pourra effectuer des opérations arithmétiques sur ces fonctions ainsi que les composer.

1 Déclaration de la classe et premières opérations

1. Dans le fichier `fonction.hpp`, déclarer une classe `Fonction` contenant deux champs privés de type chaîne de caractères, appelés `fct` et `drv`. Le champ `fct` représente la fonction à proprement parler, et `drv` sa dérivée.

2. Écrire le constructeur prenant en argument une chaîne de caractères. Attention, pour ce constructeur, il n'y a pas de partie `"(.)"` dans les arguments. En revanche, elle doit apparaître dans les champs `fct` et `drv`.

3. Écrire une méthode `derivee()` qui sera un accesseur pour le champ `drv`.

4. Écrire une fonction globale `operator<<` pour l'affichage d'objects de type `Fonction`. Cette fonction doit fournir le résultat suivant pour l'affichage de `Fonction("f")` :

```
La fonction f(.) a pour dérivée f'(.)
```

et terminer par un saut de ligne.

5. Écrire le constructeur pour la fonction puissance. L'affichage de `Fonction(5)` devrait afficher :

```
La fonction (.)^5 a pour dérivée 5*(.)^4
```

On pourra utiliser la fonction `std::to_string` qui permet en particulier de convertir un entier en chaîne de caractère. On optimisera le code pour simplifier `fct` et `drv` lorsque $n = 0$ (pour ne donner que `1` et `0` respectivement) et lorsque $n = 1$ (pour donner `(.)` et `1` respectivement).

6. Déclarer dans la fonction `main` du fichier `test_fonction.cpp` les fonctions `f` et `pow_5` des deux exemples précédents, ainsi que la fonction `cos` et écrire les instructions pour tester leur affichage.

7. Écrire les opérateurs d'addition et de multiplication, sous forme de fonctions globales amies. On veillera à ajouter des parenthèses autour des sommes et des produits pour s'assurer que les opérations seront effectuées dans le bon ordre. Ajouter les lignes de code dans `test_fonction.cpp` pour afficher les (uniquement) les dérivées de $f + g$ et $f * g$.

8. Définir une fonction globale amie `operator*(double a, const Fonction & f)` qui renvoie la fonction donnée par le produit de `f` avec le scalaire `a` .

Afin de définir la composition, on souhaite maintenant écrire une fonction globale

```
std::string subst_dot(const std::string & fct, const std::string & nvar)
```

Cette fonction remplace dans la chaîne `fct` chaque occurence du caractère `'.'` par le texte de `nvar` . Le champ `fct` pour $f \circ g$ pourra donc être obtenu par `subst(f.fct, g.fct)` .

9. Une chaîne de caractères `s` se comporte un peu comme un vecteur et on peut accéder au i^{e} caractère avec `s[i]` , ajouter à la fin un caractére `c` avec `s.push_back(c)` ou une autre chaine `t` avec `s.append(t)` . Écrire une première version de la fonction `subst` , renvoyant la chaîne `s` obtenue à partir de `fct` après substitution.

10. Afin de rendre plus explicite le code, écrire une deuxième version `subst2` , en utilisant `std::for_each` . Au lieu de la chaîne `s` , on utilisera un objet `ss` de type `std::stringstream` (défini dans l'entête `<sstream>`), qui se manipule comme un flux de sortie : pour ajouter du contenu (nombre, caractère, etc.) à la fin de `ss` , on utilise l'opérateur `<<` . On peut accéder à la chaîne de caractères ainsi construite avec la méthode `str()` . Dans le corps de `subst2` , définir une lambda-fonction `replace` qui capture les variables du contexte `ss` et `n_var` en référence, et prend en argument un caractère `c` et qui ajoute à `ss` soit `n_var` , soit `c` suivant que `c` est un point ou non. On rappelle que `for_each` prend trois arguments : l'itérateur de début et de fin de l'intervalle de travail, et la fonction (ou lambda-fonction) à appliquer à chaque élément de l'intervalle.

11. Écrire la méthode `Fonction::rond(const Fonction &) const` qui renvoie la composée.

12. Dans la fonction `main` , afficher la dérivée de la fonction $x \mapsto f(\cos(x)x^5)$.

13. On veut ajouter un constructeur à la classe pour construire plus facilement les fonctions polynômes. En passant un vecteur de doubles en référence constante $a_0, ..., a_{n-1}$, on veut obtenir l'objet représentant la fonction $x \mapsto \sum_{i=0}^{n-1} a_i x^i$. Á l'aide des opérations de somme et de multiplication par un scalaire, écrire un tel constructeur. Indication : on aimerait peut-être construire comme étape intermédiaire un vecteur avec les fonctions monomes, mais ce n'est pas possible avec le code tel quel. Pourquoi ? Modifier afin que ce soit possible. Ensuite faire la somme soit à la main, soit en utilisant `std::accumulate` (entête `<numeric>`).

14. Écrire dans le fichier f.txt l'affichage de la fonction $(1 - 3\cos + 2\cos^3)^5 * f(x^5)$.

15. (Bonus) Compléter les opérations arithmétique pour la soustraction et la division, ainsi que la négation unaire ($f \mapsto -f$).