TP 2 - Polynômes

Le but de ce TP est de créer une class `Polynome` en `C++` permettant de représenter les polynôme de $\mathbb{R}[X]$. Nous rappelons que pour tout polynôme P de $\mathbb{R}[X]$ il existe un entier $n \in \mathbb{N}$ et des réels $a_0, ..., a_n$ tels qu'on ait

$$P(X) = \sum_{i=0}^{n} a_i X^i.$$

On peut donc naturellement représenter le polynôme P par le tableau (de taille $n+1$) de coefficients $[a_0, ..., a_n]$. Par convention le polynôme nulle sera représentée par un tableau vide. Voici le début de notre classe `Polynome` :

```
class Polynome{
private:
  size_t taille;
  double* coeffs;
public:
  ...
};
```

Exercice 1. Ajoutons quelques fonctionnalités de base à notre classe.

- **a.** Ecrire le constructeur `Polynome::Polynome()` qui crée le polynôme nul.
- **b.** Ecrire le constructeur `Polynome::Polynome(size_t n)` qui crée le polynôme X^n.
- **c.** Ecrire le destructeur `Polynome::~Polynome()`.
- **d.** Ecrire une fonction `int Polynome::degre()` retournant le degré du polynôme. Le polynôme nul étant de degré $-\infty$ on retounera -1.
- **e.** Ecrire la fonction `double& Polynome::operator[](size_t i)` permettant d'accéder au coefficient a_i du polynôme. On vérifiera que ce coefficient existe bien à l'aide de la commande `assert` disponible dans la librairie `cassert`
- **f.** Ecrire la fonction `Polynome& Polynome::operator=(const Polynome& P)` assignant P au polynôme courant. On retournera une référence sur le polynôme courant ainsi modifié.
- **g.** Ecrire le constructeur `Polynome::Polynome(initializer_list<double> l)` créant un polynôme à partir d'une liste de coefficients. On chargera la librairie `initializer_list`.
- **h.** Ecrire une fonction `double Polynome::operator()(double x)` retournant le polynôme courant évalué en x.
- **i.** En dehors de la classe `Polynome` écrire une fonction d'affichage `ostream& operator<<(ostream& os,const Polynome& P)` affichant le polynôme dans le flux `os`. Le rendu de cette fonction pourra être amélioré ultérieurement.

Exercice 2.

a. Ecrire une fonction `Polynome Polynome::derive()` qui retourne le polynôme dérivé du polynôme courant.

b. Ecrire une fonction `Polynome Polynome::integre()` qui retourne un polynôme primitive du polynôme courant.

c. Ecrire une fonction `void Polynome::racines()` qui affiche lesracines d'un polynôme de degré au plus 2 et affiche une erreur sinon. On chargera la librairie `cmath` pour accéder à la fonction `double srqt(double a)` retournant la raciné carrée de a.

Exercice 3.

a. Ecrire une fonction `Polynome Polynome::operator*(double a)` qui retourne le polynôme courant multiplié par a.

b. Ecrire une fonction `Polynome Polynome::normalise()` qui détermine le degré réel du polynôme courant en tenant compte des zeros devant les coefficients dominants.

c. Ecrire une fonction `Polynome Polynome::operator+(Polynome& P)` qui retourne le polynôme courant auquel on a additionné P.

d. Ecrire une fonction `Polynome Polynome::operator-(Polynome& P)` qui retourne le polynôme courant auquel on a soustrait P.

e. Ecrire une fonction `Polynome Polynome::operator*(Polynome& P)` qui retourne le polynôme courant auquel on a multiplié P.