UNIVERSITÉ DE MONTPELLIER

FACULTÉ DES SCIENCES MONTPELLIER

M2 Modélisation et Analyse Numérique
UE Programmation II - 2025/2026

CC 1 - polynomials - dense storage

The mandatory deliveries filenames are specified below in red color. *Some provided files are named in* blue color *below, and are available in the* CC1/src *folder. Please, respect the naming convention. The use of any LLM chatbot or IA co-pilot is strictly forbidden.*

This evaluation relies on the use and extension of the class_list.hpp studied in PW1.

Exercice 1. A template class polynomial

Polynomials are a basic component of several numerical methods. Your work is to design and code a polynomial class, stored in a file called class_polynomial.hpp. Storing a polynomial reduces to store its coefficients: your polynomial class should **publicly derive** from the class list<T>: the polynomial's coefficients are stored in the item_ field and the polynomial's degree is related to the number_ field. For example, the polynomial

$$p(x) = x^4 + 2x^2 - 3,$$

should result in the following values: item_ = [–3., 0., 2., 0., 1.] and number_ = 5.

Here are your guidelines and questions:

(1) two constructors should be provided, corresponding to the two constructors of the base class list, with the following prototypes:

```
polynomial(size_t degree);
polynomial(size_t degree, T value);
```

(2) a method with the following prototype

```
inline size_t degree() const
```

should return the degree of the calling polynomial,

(3) a common task involving the polynomial p is to calculate its value $p(b)$ at a given point $x = b$. Define an overloaded **operator**(), such that the command p(b) returns the value of the polynomial p evaluated at any given number b
(the straightforward evaluation of the sum $p(b) = \sum_{i=0}^{n} a_i b^i$ should be preferred, as the more elaborated Horner algorithm is studied at Exercice 3)

(4) define an overloaded **operator**<<() such that the commands:

```
polynomial<double> p(3, 10.);
cout << p ;
```

result in the following pretty print:

```
10. x^3 + 10. x^2 + 10. x + 10.
```

Test your class with the provided main_ex1.cpp program, which should compile and provide the correct results (note that you can obviously comment some lines of this main, if needed, during your validations steps).

Exercice 2. Algebraic operators

Two operators should also be overloaded to manipulate polynomials, and implemented as **nonmember functions**:

(4) define two versions of **operator***() that returns the product of a scalar and a polynomial, with the following prototypes:

```
const polynomial<S> operator*(S, polynomial<S> const&);
const polynomial<S> operator*(polynomial<S> const&, S);
```

(5) define a binary **operator**+() which returns the sum of two polynomials, with the following prototype:

```
const polynomial<S> operator+(polynomial<S> const&, polynomial<S> const&);
```

It should handle the addition of polynomials of different degrees.

Test your class with the provided main_ex2.cpp program, which should compile and provide the correct results.

Exercice 3. Evaluation of polynomials (again)

One of the common principles of efficient algorithm is to avoid recalculating data that can be easily fetched from the memory. Here, we consider an algorithm that uses this principle in the present problem of calculating the value of a polynomial at a given point, leading to a more efficient implementation. Indeed, let consider this simple formula:

$$A(B + C) = AB + AC.$$

In the right expression, only one addition and two products are needed, while in the left expression only one addition and one product are needed. This idea is used in the Horner algorithm for polynomials. Let p be defined as follows

$$p(x) = \sum_{i=0}^{n} a_i x^i.$$

The direct evaluation of the right-hand side requires $2n$ multiplications ($n-1$ multiplications to calculate the powers x^i, and another $n+1$ multiplications to multiply them by the coefficients a_i) and n additions. The Horner algorithm uses the following formula:

$$p(x) = (\ldots(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-3})\ldots)x + a_0,$$

which requires only a total of n multiplications and n additions. Implement this more efficient algorithm in a method declared as follows:

```
const S horner_polynomial(polynomial<S> const& , S);
```

Test your class with the provided main_ex3.cpp program, which should compile and provide the correct results.