Dirk Praetorius
Michele Ruggeri

Summer semester 2019
May 17, 2019

Scientific programming in mathematics

Exercise sheet 9

References, keyword `const`, operator overloading, dynamic memory allocation

Exercise 9.1. Write a class `Polynomial` to store polynomials of degree $n \in \mathbb{N}_0$ represented with respect to the monomial basis, i.e.,

$$p(x) = \sum_{j=0}^{n} a_j x^j.$$

The class contains the polynomial degree $n \in \mathbb{N}_0$ (`int`) and the dynamic vector $(a_0, \dots, a_n) \in \mathbb{R}^{n+1}$ of the coefficients (`double*`). Implement the following features:

- The constructor, which, given $n \in \mathbb{N}_0$, allocates a polynomial of degree $n \in \mathbb{N}_0$ and initializes all entries of its coefficient vector with zero. Use `assert` to ensure that $n \geq 0$.
- The destructor.
- The copy constructor (note that you have to allocate new dynamic memory for the coefficient vector of the output, why?).
- The assignment operator.
- A method `int degree() const` to get the polynomial degree $n \in \mathbb{N}_0$.
- The possibility to print a polynomial `p` to the screen via the syntax `cout << p`.

Templates for the structure of your implementation are provided by the classes `Complex` and `Vector` presented in the lecture notes (slides 276–279 and slides 292–296, respectively). Test your implementation appropriately!

Exercise 9.2. Extend the class `Polynomial` from Exercise 9.1 by capability of accessing the coefficients of the polynomials via `[ ]`, i.e., for $0 \leq j \leq n$, the j-th coefficient a_j of a polynomial stored in an object `p` can be obtained by typing `p[j]`. Implement this feature for both `const` and 'normal' objects, i.e., in the class definition use the signatures

```
const double& operator [](int j) const;
double& operator [](int j);
```

Use `assert` to ensure that $0 \leq j \leq n$. Test your implementation appropriately!

Exercise 9.3. The sum of two polynomials $p(x) = \sum_{j=0}^{n} a_j x^j$ and $q(x) = \sum_{k=0}^{m} b_k x^k$ is still a polynomial. What is the degree of the polynomial sum $p+q$ in terms of the degrees n and m of the polynomial addends p and q? What are the coefficients of the polynomial $p+q$? Extend the class `Polynomial` from Exercise 9.1 by the feature of adding two polynomials `p` and `q` by typing `r=p+q`. Note that the input polynomials might have different degrees. Moreover, implement the opportunity to add a number $a \in \mathbb{R}$ stored as `double` or `int` to a polynomial p in an appropriate way via `r=a+p` and `r=p+a`. Test your implementation appropriately!

Exercise 9.4. The product of two polynomials $p(x) = \sum_{j=0}^{n} a_j x^j$ and $q(x) = \sum_{k=0}^{m} b_k x^k$ is still a polynomial. What is the degree of the polynomial product pq in terms of the degrees n and m of the polynomial factors p and q? Derive a formula for the coefficients of the polynomial $p+q$. Extend the class `Polynomial` from Exercise 9.1 by the feature of multiplying two polynomials `p` and `q` by typing `r=p*q`. Moreover, implement the opportunity to multiply a polynomial p by a number $a \in \mathbb{R}$ stored as `double` or `int` in an appropriate way via `r=a*p` and `r=p*a`. Test your implementation appropriately! What is the computational complexity of your implementation for two polynomials p and q of same degree n? If the computation of the product of two polynomials p and q of same degree for $n = 10^2$ has a runtime of 1 second with your implementation, which runtime do you expect for $n = 10^3$? Justify your answers!

Exercise 9.5. Two polynomials coincide if and only if they have the same degree and all coefficients coincide. Extend the class `Polynomial` from Exercise 9.1 by the feature of checking two polynomials `p` and `q` by typing `p==q`. Do not use `==` to compare the coefficients of the polynomials (which have type `double`), but rather check whether the difference of the coefficients is smaller than a given (small) tolerance. Why is this the correct strategy? Test your implementation appropriately!

Exercise 9.6. For each $k \in \mathbb{N}_0$ the k-th derivative $p^{(k)}$ of a polynomial p is still a polynomial. Extend the class `Polynomial` from Exercise 9.1 by the following features:

- The possibility to evaluate the k-th derivative of a polynomial p via `p(k,x)`, where $x \in \mathbb{R}$ (`double`) and $k \in \mathbb{N}_0$ (`int`);
- For $k = 0$ the call `p(x)` must be also possible, i.e., the calls `p(0,x)` and `p(x)` are equivalent;
- The possibility to obtain the k-th derivative of a polynomial p via `p(k)`, where $k \in \mathbb{N}_0$ (`int`).

Test your implementation appropriately!

Exercise 9.7. Extend the class `Polynomial` from Exercise 9.1 by the method `double computeIntegral(double alpha, double beta)`, which computes and returns the integral of a polynomial p. More precisely, given $\alpha, \beta \in \mathbb{R}$ with $\alpha < \beta$, the method should compute and return the integral

$$\int_\alpha^\beta p(x)\,dx.$$

Note that, for $p(x) = \sum_{j=0}^{n} a_j x^j$, it holds that

$$\int_\alpha^\beta p(x)\,dx = \sum_{j=0}^{n} \frac{a_j(\beta^{j+1} - \alpha^{j+1})}{j+1}.$$

Where does this formula come from? Use `assert` to check that $\alpha < \beta$. Test your implementation appropriately!

Exercise 9.8. Given an initial guess $x_0 \in \mathbb{R}$, the Newton method determines an approximation of the root of a polynomial p according to the following procedure: Starting from x_0, define inductively the sequence $\{x_k\}$ by

$$x_k = x_{k-1} - p(x_{k-1})/p'(x_{k-1}) \quad \text{for } k \geq 1,$$

where p' denotes the derivative of p. Under appropriate assumptions, the sequence $\{x_k\}$ then converges towards a zero of p. Extend the class `Polynomial` from Exercise 9.1 by the method `double computeZero(double x0, double tau)`, which, given $x_0 \in \mathbb{R}$, computes and returns an approximation of a zero of a polynomial obtained with the Newton method. The Newton iteration is performed until, given a tolerance $\tau > 0$, the following inequalities are satisfied:

$$|p(x_k)| \leq \tau \quad \text{and} \quad |x_k - x_{k-1}| \leq \tau.$$

In this case, the function returns the last value x_k as an approximation of the zero. Use `assert` to ensure that $\tau > 0$. You should not store all the elements of the sequence, since every step of the algorithm requires only x_{k-1} and x_k. Test your implementation appropriately!