# Advanced Programming, Miniproject 2

Marius Mikučionis

May 27, 2015

## 1 General requirements

1. The goal is to design a modern C++ library using C++11 and/or C++14 constructs.
2. The miniproject is an individual assignment and the work should be done independently.
3. Any literature resources are permitted, but the assignment itself should not be discussed (i.e. the work should be independent). Basic discussions about C++ features without hinting about the assignment are allowed. The specific requirements mostly relate (but not limited) to items and topics described in "Efficient Modern C++" by Scott Meyers.
4. Only C++ standard and BOOST libraries can be used.
5. Deliverable should consist of a printed C++ solution with an author name, and an archive uploaded to Moodle with at least the following items:
    (a) Solution as a set of C++ source files.
    (b) The source files should contain comments explaining what requirements are being covered in specific places of the code.
    (c) A separate "read-me.txt" file explaining: 1) what compiler should be used, 2) how the solution should be compiled (i.e. C++11 or C++14?), 3) how to specify inputs and execute the solution, 4) what outputs are expected.
    (d) (Optional) scripts to compile and run the solution (e.g. GNU make file, shell script, VS project files, etc.).
6. The solutions will be evaluated strictly based on the detailed requirements, i.e. the specified features should be present and commented in order to satisfy the requirement.

## 2 Detailed requirements

1. Implement a library to compute operations over polynomials with the following API:
    (a) Default constructor to create a trivial polynomial: `0`.

   (b) Constructor for specific degree term coefficients.
   (c) A method to scale the polynomial, i.e. multiply by a scalar value.
   (d) A method to add a root `r`, i.e. multiply by a term `(x-r)`.
   (e) A method to add several roots at once.
   (f) A method to valuate the polynomial at a given point.
   (g) A method to compute a polynomial which is a derivative of the polynomial.
   (h) A method to compute an integral for given interval bounds.
   (i) A plus operator to return a polynomial equal to a sum of two polynomials.
   (j) A star operator to return a polynomial equal to a product of two polynomials.

2. Make a template class of Polynomial for a given type of coefficients (e.g. floating point types, complex numbers; also support integer types in all operations except integration).

3. Use `auto` where applicable (Item 5). It is enough to mention just once in comments that you are following this requirement, but it should be enforced throughout the source code.

4. Use `const` where applicable (e.g. function arguments and methods, see also Item 13). It is enough to mention just once in comments that you are following this requirement, but it should be enforced throughout the source code.

5. Member functions accepting containers should support any type of container, including array types (Item 13); also support braced initializers (Item 7).

6. Cache the integral data to avoid repetitive integration (Items 16, 40). Make it thread-safe.

7. Implement move semantics (Chapter 5) by using either Pimpl idiom (Items 22,17,8) or smart pointers (Item 17, Chapter 4).

8. Use type traits (see examples in Item 27, e.g. disable or fail assertion for the integration method over integer types).

9. Use lambda expressions (Chapter 6, e.g. when computing sums during evaluation of a polynomial; dispatch asynchronous computation).

10. Use concurrency (Chapter 7, e.g. compute products of polynomials by evaluating each polynomial asynchronously and then multiplying the final product).

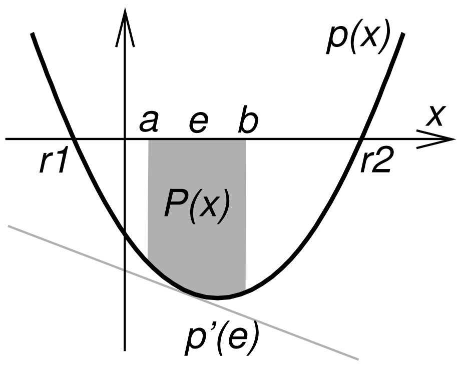

Figure 1: Geometrical interpretation of a polynomial $p(x) = (x - r1)(x - r2)$, its derivative $p'(x)$ as a slope at point $e$ and integral $P(x)|_a^b = \int_a^b p(x)dx$ as an area limited by a polynomial and $x$-axis.

## 3 Background and Tips

1. A few useful facts about polynomials (see also Fig.1):

   - Any $n$-degree polynomial can be expressed as a product: $p(x) = (x - r_1) \cdot (x - r_2) \cdot \ldots \cdot (x - r_n)$, where $r_i$ is called a root of a polynomial ($p(r_i) = 0$), for $i \in \{1, 2, \ldots, n\}$. $n$-degree polynomial always has $n$ roots, some of them may repeat and some can be complex numbers.
   - Polynomial of degree $n$ can be expressed as a sum: $p(x) = c_0 + c_1 \cdot x + c_2 \cdot x^2 + \ldots + c_n \cdot x^n$, where $c_i$ is a coefficient for an $i^{\text{th}}$ degree term. This is a more convenient form for our purposes due to easy conversion, derivation and integration (see the next facts).
   - Distributivity law to convert a product into a sum:

     $$\begin{aligned} p_n(x) \cdot (x - r_{n+1}) &= (c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n) \cdot (x - r_{n+1}) \\ &= -c_0 r_{n+1} + (c_0 - c_1 r_{n+1})x + (c_1 - c_2 r_{n+1})x^2 + \ldots + c_n x^{n+1} \end{aligned}$$

   - Formula to compute a derivative of a polynomial:

     $$\begin{aligned} p'(x) &= (c_0 + c_1 \cdot x + c_2 \cdot x^2 + \ldots + c_n \cdot x^n)' \\ &= c_1 + 2 \cdot c_2 \cdot x + 3 \cdot c_3 \cdot x^2 + \ldots + n \cdot c_n \cdot x^{n-1} \end{aligned}$$

     which is the tangent (slope) of $p$ at point $x$ (can be computed for specific $x$).

- Formula to compute an integral of a polynomial over an interval $[a, b]$:

$$\begin{aligned}\int_a^b p(x)dx &= \int_a^b (c_0 + c_1 \cdot x + c_2 \cdot x^2 + \ldots + c_n \cdot x^n)dx \\ &= \left(c_0 x + \frac{c_1}{2}x^2 + \frac{c_2}{3}x^3 + \ldots + \frac{c_n}{n+1} \cdot x^{n+1}\right)\Big|_a^b \\ &= \left(c_0 b + \frac{c_1}{2}b^2 + \frac{c_2}{3}b^3 + \ldots + \frac{c_n}{n+1}b^{n+1}\right) \\ &- \left(c_0 a + \frac{c_1}{2}a^2 + \frac{c_2}{3}a^3 + \ldots + \frac{c_n}{n+1}a^{n+1}\right)\end{aligned}$$

- A few ideas for testing (not for library implementation!):

$$\begin{aligned}(p_1 + p_2)(x) &= p_1(x) + p_2(x). \\ (p_1 \cdot p_2)(x) &= p_1(x) \cdot p_2(x). \\ \text{if}\, p(x) &= (x - r_1) \cdot (x - r_2), \text{then}\, p'((r_1 + r_2)/2) = 0. \\ \text{if}\, p(x) &= (x - r) \cdot (x + r), \text{then} \int_{-r}^{r} p(x)dx = -4/3 \cdot r^3. \\ p'(x) &= \lim_{d \to 0} \frac{p(x+d) - p(x)}{d}. \\ \int_a^b p(x)dx &= \lim_{d \to 0} \sum_{i=0}^{(b-a)/d} p(a + i \cdot d) \cdot d.\end{aligned}$$

2. Chapter 1, 2 and 5 are essential to understanding and writing C++ efficiently.

3. Avoid code duplication: distill reusable parts into separate functions and reuse them by function calls.

4. Use standard library: ⟨iostream⟩, ⟨vector⟩/⟨list⟩/⟨deque⟩, ⟨iterator⟩, ⟨utility⟩, ⟨algorithm⟩, ⟨functional⟩, ⟨memory⟩, ⟨type_traits⟩, ⟨future⟩, ⟨complex⟩.

5. Perform unit testing to gain confidence while developing incrementally (e.g. write a test before implementing a function, implement the function and then run the test).

6. C++14 is easier to work with than C++11, but it can be difficult to setup the compiler for C++14. C++11 is also fine, except a few workarounds may be needed (see EMC++ and/or lecture slides). Some workarounds are also good examples of how to program, hence they are useful in learning too.