Exercise session 05

Functions. Templates and generic programming in C++.

Advanced Programming - SISSA, UniTS, 2025-2026

Pasquale Claudio Africa

29 Oct 2025

Exercise 1: lambda functions

Starting from the `hints/ex1.cpp` source file, create a C++ program that calculates the total cost of a given list of products.

1. Use `std::accumulate` to calculate the total cost of the products, passing a custom lambda function.
2. Display the results after each step (partial sums) and the total cost to check for correctness.

Exercise 2: function pointers, functors, lambdas

Starting from the `hints/ex2.cpp` source file, develop a library management system with search capabilities using lambdas and functors.

1. Using `std::sort`, sort the books:
    - In ascending order based on year, using a **function pointer** as a comparator.
    - In descending order based on year, using a **lambda function** as a comparator.
    - In ascending order based on the author name, using a **functor** as a comparator.
2. Using `std::copy_if`, fill a new vector `filtered_books` by extracting from `books` only the books written by a specific author. Implement the search functional using lambdas.
3. Display the results after each step to check for correctness.

Exercise 3: function wrappers, templates (1/2)

The `hints/ex3/` folder provides a partial C++ implementation of the Newton method to approximate the root(s) of a function f, i.e., to solve the problem $f(x) = 0$.

Newton's method in a nutshell

Starting with an initial guess for the root(s) of the function, denoted as $x^{(0)}$, repeatedly refine the estimate using the formula

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})},$$

where $f'(x)$ is the derivative of $f(x)$.

The iterations continue until the difference between two consecutive estimates, $\left|x^{(k+1)} - x^{(k)}\right|$ is smaller than a predefined tolerance. If the condition is not met within a maximum number of iterations, the algorithm failed to reach converge and the solver returns `NaN`.

Exercise 3: function wrappers, templates (2/2)

1. Fill in the missing parts to make the program work with real-valued scalar functions.
2. Use the program to solve $f(x) = x^2 - 1 = 0$, starting from $x^{(0)} = 0.5$.
3. Templatize the solver to be able to deal with more general functions, such as complex-valued functions.
   Use the program to solve $f(x) = x^2 + 1 = 0$, starting from $x^{(0)} = 0.5 + 0.5i$.
4. How would you organize the project files? Is it better to keep everything in header files, or splitting declarations and definitions in header and source files by providing explicit instantiations? Try both alternatives.

A note on the use of template arguments as policies

⚠️ **In C++, template arguments can be used as *policies*.**

This allows you to customize the behavior of a class or function without changing its core implementation. As templates provide a way to write generic code that can work with different data types, policies provide a way to write generic code that can behave according to different algorithms or strategies.

When you use a template argument as a policy, you are essentially saying, "Here is a piece of code, and I want to let users decide certain aspects of its behavior." This can include things like how elements are compared, how data is stored, which specific algorithm to apply, or how certain operations are performed.

See the example included in the `examples` folder.

Exercise 4: template metaprogramming

Use template metaprogramming to calculate the factorial of an integer at compile time.

1. Define a template class for calculating the factorial of an integer.
2. Instantiate the template for the integers 5, 7, and 10.
3. Use `static_assert` to ensure that values are computed at compile time rather than runtime.
4. Print the results of the factorials.