

Inheritance, Overriding and Dynamic Binding

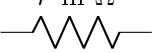
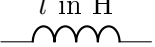
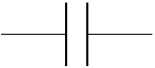

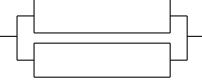
OOP in C++

2017

1 Electric Dipoles

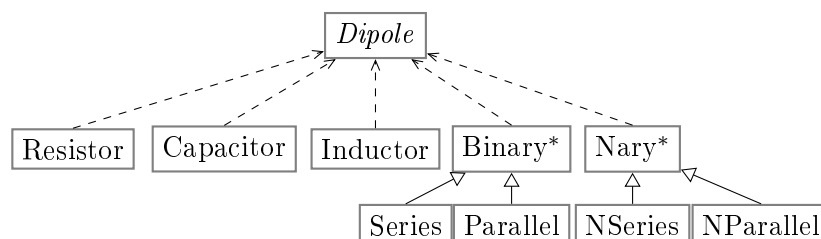
Every electric circuit is composed of differing components such as resistors, capacitors, diodes and electromagnetic coils. They can be assembled in either series or parallel circuits. Depending on their component, each circuit present a specific resistance to the current when a voltage is applied. The **impedance** extends this notion of resistance to alternating currents.

Given ω the angular frequency of the current, the impedance z of the circuit can be computed as follows (with the constant $i = 1\angle\frac{\pi}{2} = e^{j\frac{\pi}{2}}$).

Symbol	Description	Impedance
r in Ω 	A resistor of value r expressed in ohms (noted Ω)	$z = r$
l in H 	An inductor of value l expressed in henries (noted H)	$z = i(\omega * l)$
c in F 	A capacitor of value c expressed in farad (noted F)	$z = i(\frac{-1}{\omega * c})$
	A series circuit with 2 dipoles of impedance z_1 and z_2	$z = z_1 + z_2$
	A parallel circuit with 2 dipoles of impedance z_1 and z_2	$z = \frac{1}{\frac{1}{z_1} + \frac{1}{z_2}}$

1.1 Modeling Dipole

We will use the following class hierarchy to model the electric dipoles.



The *Dipole* class provides a method `virtual Complex impedance(double omega)` implemented below in the hierarchy. The parameter **omega** represents the angular frequency of the current. The impedance is a complex number (only resistors have a real impedance).

1.2 Implementing Dipole

Download the provided code from <https://git.io/vS2Ri>, save it to your local disk and open them with your favorite editor (or **geany** if you have no favorite editor yet) to explore these files.

The archive contains a **Makefile**, so you can rebuild everything by just typing **make** in a terminal. Read **ComplexTest.cpp** as it will show you how to use complex numbers. Try to change this program (eg by adding **c1** and **c2** and displaying the result).

★ Exercise 1: Simple Dipoles.

We will first implement the simple dipoles, ie the **Resistor**, **Inductor** and **Capacitor** classes. For each of them, we want a **cpp** file and a **hpp** file, plus a test file. The files for **Resistor** are already provided: **Resistor.hpp** (contains the declaration), **Resistor.cpp** (contains the actual code) and **ResistorTest.cpp** (contains some test of the class).

▷ **Question 1:** The provided code is not sufficient, as the execution of **./ResistorTest.cpp** leads to an incorrect result. Change what should be in the three **Resistor*** files to fix the problem.

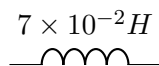
▷ **Question 2:** Copy all **Resistor.cpp** and **Resistor.hpp** files to **Inductor.*** ones, and change what should be in each of them to pass the provided **InductorTest**.

▷ **Question 3:** Implement similarly the **Capacitor** class.



$$(z = 100 \, \Omega)$$

Tested Resistor.



$$(z \approx 22j \, \Omega)$$

Tested Inductor.



$$(z \approx -7.6 \times 10^{-5}j \, \Omega)$$

Tested Capacitor.

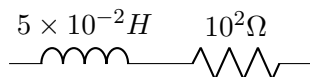
★ Exercise 2: Binary circuits.

We will now implement the binary circuits, starting with the ones built in serie. For that, the private part of the **Serie** class should naturally contain the two dipoles that are connected in parallel. Since we would like these two dipoles to be either resistors, inductors, capacitors, or even binary circuits, the only possibility in C++ is to use generic **Dipole *** pointers, as shown in the provided **Serie.hpp** code.

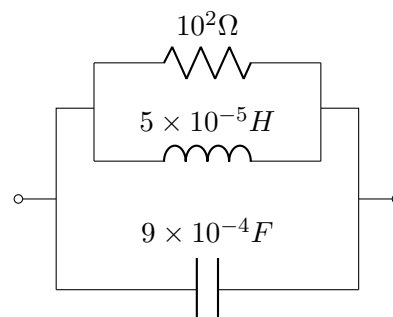
▷ **Question 1:** Complete the **Serie** class so that it passes the provided test.

▷ **Question 2:** Now implement the **Parallel** class similarly to the **Serie** class.

Check your implementation with the following circuits.



$$\text{Serie } (z \approx 100.0 + 15.70j \, \Omega).$$



$$\text{Parallel } (z \approx 0.2079 + -4.55j \, \Omega).$$

▷ **Question 3:** Even if functional, the implementation you probably came up with is still improvable because there is some code in common between your **Serie** and **Parallel** classes. Implement a **Binary** class that would factorize the private fields, and some of the implementation.

★ **Exercise 3: Nary circuits.**

Now, we would like to implement N-ary circuits, that can connect more than 2 dipoles in either parallel or serie. For series circuit: $z = \sum_{i=1}^n \omega_i$. For parallel circuit: $z = \frac{1}{\sum_{i=1}^n \frac{1}{\omega_i}}$.

▷ **Question 1:** Implement the `NSerie`, `NParallel` and also the `Nary` classes. To store the data in the `Nary` class, use a `std::vector` (as described in <http://en.cppreference.com/w/cpp/container/vector>).

▷ **Question 2:** Test your implementation against the following instances:

