Workshop 06 – Templates: A Generic Complex Class

Objectives

In this workshop, you will practice:

- Converting an existing class into a **class template**
- Implementing template member functions and operators
- Understanding why templates are usually **defined in header files**
- Writing code that works for multiple numeric types (e.g., `int`, `double`)

Starting Point

Use your solution from our third workshop (your `Complex` class with constructors and operator overloads).

Your task is to **modify that code** so that `Complex` becomes a **templated class** named:

```
template <typename T>
class Complex;
```

The template parameter `T` will be the type used for **both** the real and imaginary components.

Problem Description

A complex number is typically written as:

```
a + bi
```

In Workshop 03, you represented `a` and `b` using `double`.

In this workshop, you will generalize the class so that the real and imaginary values can be any type `T`, as long as `T` supports the required arithmetic operations.

Examples of valid usages:

- `Complex<double> z1(3.5, -2.0);`
- `Complex<int> z2(3, 4);`

Requirements

1. Class Template Definition

Create a **class template** named `Complex<T>` with the following **private** data members:

- `T real_`
- `T imag_`

> **Important:** Both members must have the **same type** `T`.

---

2. Constructors

Provide the following constructors for `Complex<T>`:

1. **Default constructor**
   - Initializes the complex number to `0 + 0i` using value-initialization
2. **Overloaded constructor**
   - Receives two values of type `T` and initializes the object

---

3. Operator Overloading (Complex with Complex)

Overload the following operators for **two `Complex<T>` objects**:

- `+` (addition)
- `-` (subtraction)
- `*` (multiplication)

These operators must behave according to standard complex arithmetic:

- `(a + bi) + (c + di) = (a + c) + (b + d)i`
- `(a + bi) - (c + di) = (a - c) + (b - d)i`
- `(a + bi) * (c + di) = (ac - bd) + (ad + bc)i`

---

4. Operations with Scalars of Type `T`

Allow complex numbers to interact with scalar values of type `T`, **regardless of the order**:

Examples:

- `Complex<T> + T`
- `T + Complex<T>`
- `Complex<T> - T`
- `T - Complex<T>`
- `Complex<T> * T`
- `T * Complex<T>`

> **Hint:** Supporting a scalar on the left-hand side will likely require **friend functions** (or non-member overloads).

---

5. Output with `std::cout`

Overload `<<` so that you can print a `Complex<T>`:

Example output format:

```
3 + 4i
1.5 - 2i
```

> **Note:** This requires that `T` can be printed with `std::ostream` (i.e., `operator<<` exists for `T`).

---

6. Templates and File Structure

Because templates must be visible at the point of instantiation, you have two valid approaches:

Option A (recommended for this workshop):

- Put the entire template (declaration + definitions) in `complex.h`.

Option B:

- Put declarations in `complex.h`
- Put definitions in a file like `complex.cpp`
- `#include "complex.cpp"` at the bottom of `complex.h`

> Avoid placing template definitions only in `complex.cpp`, or you may get linker errors.

Submission

Submit:

- `complex.h` (and possibly `complex.cpp`)

Your final code must:

- Compile without warnings
- Use proper const-correctness
- Follow good coding style and readability