

Project - 1    **MA517M-Basic Programming Laboratory**    Last Date: 14 November 2026  
Name \_\_\_\_\_ Roll No.: **MA26M019**

---

## Quaternion Arithmetic using C++ Classes and Operator Overloading

**Objective:** To design a C++ program that implements quaternions using classes, supports arithmetic operations, and demonstrates operator overloading.

A quaternion is an extension of complex numbers:

$$q = a + bi + cj + dk$$

where  $a, b, c, d \in \mathbb{R}$ , and  $i, j, k$  satisfy

$$i^2 = j^2 = k^2 = ijk = -1, \quad ij = k, \quad ji = -k, \dots$$

Quaternions are used in 3D rotations and physics, and their multiplication is non-commutative.

### Problem Description

Design a class `Quaternion` that represents quaternions. The class should support addition, subtraction, multiplication, division (by the norm squared), conjugation, and equality comparison using operator overloading.

### Class Specification

- **Class Name:** `Quaternion`
- **Private Data Members:**
  - `double a, b, c, d;` // real and imaginary components
- **Public Member Functions:**
  - `Quaternion(double a=0, double b=0, double c=0, double d=0);` Constructor initializes the quaternion
  - `double norm() const;` Computes  $\|q\| = \sqrt{a^2 + b^2 + c^2 + d^2}$
  - `Quaternion conjugate() const;` Returns  $\bar{q} = a - bi - cj - dk$
  - `void display() const;` Displays quaternion in  $a + bi + cj + dk$  form

## Operator Overloading

- operator `+` , `-` Addition and subtraction component-wise
- operator `*` Quaternion multiplication using the rules:

$$ij = k, ji = -k, jk = i, \dots$$

- operator `/` Division using inverse:  $q_1/q_2 = q_1 * q_2^{-1}$
- operator `==` Checks the equality of two quaternions

## Tasks

1. Create quaternions  $q_1 = 1 + 2i + 3j + 4k$  and  $q_2 = 2 - i + k$
2. Compute  $q_1 + q_2$ ,  $q_1 - q_2$ ,  $q_1 * q_2$ ,  $q_1/q_2$
3. Compute norms and conjugates of  $q_1$  and  $q_2$
4. Compare  $q_1$  and  $q_2$  using equality operator

## Expected Output Example

`q1 = 1 + 2i + 3j + 4k`

`q2 = 2 - i + 0j + 1k`

`q1 + q2 = 3 + 1i + 3j + 5k`

`q1 - q2 = -1 + 3i + 3j + 3k`

`q1 * q2 = -12 + 4i + 10j + 4k`

`q1 / q2 = 0.44 + 1.11i + 0.67j + 0.22k`

`Norm of q1 = 5.477`

`Conjugate of q1 = 1 - 2i - 3j - 4k`

`q1 == q2 : False`

## Project - 2: Conway's Game of Life Using C++ Classes

### Problem Statement

Design and implement **Conway's Game of Life** using **C++ classes**. This is a zero-player game where the evolution of a 2D grid is determined by its initial state and simple rules. Each cell in the grid can be *alive* or *dead*, and the next generation of the grid is calculated based on the number of alive neighbors. The project should utilize object-oriented programming concepts such as classes, objects, encapsulation, and methods for handling game logic and grid updates.

## Project Requirements

1. Create a **Cell** class to represent a single cell in the grid.
  - (a) Data member: current state (alive or dead)
  - (b) Methods: `setState()`, `getState()`
2. Create a **Grid** class to represent the game board (matrix).
  - (a) Initialize the grid with random alive and dead cells.
  - (b) Include a method to display the current state of the grid.
  - (c) Include a method to count alive neighbors for each cell.
  - (d) Include a method to update the grid according to the following rules:
    - Any live cell with fewer than two live neighbors dies (underpopulation).
    - Any live cell with two or three live neighbors lives on.
    - Any live cell with more than three live neighbors dies (overpopulation).
    - Any dead cell with exactly three live neighbors becomes alive (reproduction).
3. Create a **Game** class to manage simulation.
  - (a) Display options for the user: **Start**, **Next Generation**, or **Quit**.
  - (b) Allow the user to advance the simulation one generation at a time.
  - (c) Allow the user to quit and confirm before exiting.
  - (d) Optionally, allow the user to specify the number of generations to simulate automatically.
4. Ensure proper encapsulation of cell and grid operations within the respective classes.

## Suggested Class Structure

1. **Cell Class**:
  - Data member: `bool isAlive`
  - Methods: `setState(bool state)`, `getState()`
2. **Grid Class**:
  - Data member: 2D array of **Cell** objects
  - Methods: `initializeGrid()`, `displayGrid()`, `countAliveNeighbors(int row, int col)`, `updateGrid()`
3. **Game Class**:
  - Data member: **Grid** object, user choice, generation counter
  - Methods: `startGame()`, `nextGeneration()`, `confirmQuit()`

## Reference

For more details about Conway's Game of Life, visit: [https://en.wikipedia.org/wiki/Conway's\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway's_Game_of_Life)