INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Project - 1 **MA517M-Basic Programming Laboratory** Last Date: 09 November 2025
Name **Roll No.: MA25M101**

Finite Field Construction using C++ Classes and Operator Overloading

Objective: To design a C++ program that implements arithmetic in a *finite field* (Galois Field $GF(p)$) using classes, encapsulation, and operator overloading.

> A *finite field* $GF(p)$ is a set of integers $\{0, 1, 2, \ldots, p-1\}$ with arithmetic operations of addition and multiplication modulo a prime number p. Every nonzero element has a multiplicative inverse, and the field satisfies the usual algebraic properties: closure, associativity, commutativity, distributivity, identity elements, and inverses.

Problem Description

Design a class `FiniteFieldElement` to represent elements of $GF(p)$. The class should support arithmetic operations, inverses, and comparisons using operator overloading.

Class Specification

- **Class Name:** `FiniteFieldElement`
- **Private Data Members:**
  - `int value;` // element value
  - `int prime;` // modulus of the field
- **Public Member Functions:**
  - `FiniteFieldElement(int v, int p);` Constructor initializes element modulo p.
  - `FiniteFieldElement inverse() const;` Returns multiplicative inverse (throws error if value = 0)
  - `FiniteFieldElement pow(int n) const;` Returns exponentiation modulo p
  - `void display() const;` Displays the element

Operator Overloading

- `operator +()` Addition modulo p:

$$a + b \equiv (a + b) \mod p$$

- `operator -()` Subtraction modulo p:

$$a - b \equiv (a - b) \mod p$$

- `operator *()` Multiplication modulo p:

$$a * b \equiv (a \cdot b) \mod p$$

- `operator /()` Division modulo p using multiplicative inverse:

$$a/b \equiv a * b^{-1} \mod p$$

- `operator ==()` Checks the equality of two field elements

Tasks

1. Construct the finite field $GF(7)$
2. Create field elements $a = 3$ and $b = 5$
3. Demonstrate arithmetic operations: $a + b$, $a - b$, $a * b$, a/b
4. Compute multiplicative inverses for all nonzero elements
5. Verify field properties: closure, associativity, commutativity, distributivity, and identity elements

Expected Output Example

```
GF(7) Elements: 0, 1, 2, 3, 4, 5, 6

a = 3, b = 5
a + b = 1 (mod 7)
a - b = 5 (mod 7)
a * b = 1 (mod 7)
a / b = 2 (mod 7)

Multiplicative inverses:
1^-1 = 1
2^-1 = 4
3^-1 = 5
4^-1 = 2
5^-1 = 3
6^-1 = 6
```

Project - 2: Connect Four Using C++ Classes

Problem Statement

Design and implement **Connect Four** using **C++ classes**. Connect Four is a two-player game where players take turns dropping colored discs into a vertical 6×7 grid. The goal is to connect four discs in a row, column, or diagonal before the opponent does. The project should utilize object-oriented programming concepts such as classes, objects, encapsulation, and methods for handling game logic and player interaction.

Project Requirements

1. Create a `Disc` class to represent an individual disc.
   (a) Data member: color or player identifier.
   (b) Methods: getColor(), setColor().
2. Create a `Board` class to represent the game grid.
   (a) Initialize a 6×7 grid with empty slots.
   (b) Display the current state of the grid in a readable format.
   (c) Include a method to drop a disc into a specified column.
   (d) Include a method to check for a winning condition (four connected discs horizontally, vertically, or diagonally).
   (e) Include a method to check if the board is full (draw condition).
3. Create a `Game` class to manage gameplay.
   (a) Allow two players to take turns dropping discs.
   (b) Display options for the user: `Play` or `Solution`.
   (c) After each move, update the board and display it.
   (d) Allow the user to quit and confirm before exiting.
   (e) Keep track of player turns and optionally score.
4. Ensure proper encapsulation of disc and board operations within the respective classes.

Suggested Class Structure

1. `Disc` Class:
   - Data member: `char color` or `int playerID`
   - Methods: getColor(), setColor()
2. `Board` Class:
   - Data member: 2D array of `Disc` objects
   - Methods: initializeBoard(), displayBoard(), dropDisc(int column, Disc disc), checkWin(), isFull()
3. `Game` Class:
   - Data member: `Board` object, currentPlayer, user choice
   - Methods: playGame(), showSolution(), switchPlayer(), confirmQuit()

Reference

For more details about Connect Four, visit: `https://en.wikipedia.org/wiki/Connect_Four`