

---

|                                                        |                             |
|--------------------------------------------------------|-----------------------------|
| Project - 1 <b>MA517M-Basic Programming Laboratory</b> | Last Date: 14 November 2026 |
| Name                                                   | Roll No.: MA26M017          |

---

## Discrete Uniform Random Variables using C++ Classes and Operator Overloading

**Objective:** To design a C++ program that implements discrete uniform random variables using classes, computes probability, expectation, variance, and supports operator overloading for comparison.

A discrete uniform random variable  $X$  takes values in a finite set  $\{x_1, x_2, \dots, x_n\}$  with equal probability.

The probability of each value is

$$P(X = x_i) = \frac{1}{n}, \quad i = 1, 2, \dots, n$$

Mean and variance:

$$\mathbb{E}[X] = \frac{1}{n} \sum_{i=1}^n x_i, \quad \text{Var}(X) = \frac{1}{n} \sum_{i=1}^n (x_i - \mathbb{E}[X])^2$$

### Problem Description

Design a class `DiscreteUniformRV` to represent a discrete uniform random variable. The class should allow computation of probability, mean, variance, and comparison with other uniform random variables using operator overloading.

### Class Specification

- **Class Name:** `DiscreteUniformRV`

- **Private Data Members:**

- `vector<int> values;`
  - `int n;`

- **Public Member Functions:**

- `DiscreteUniformRV(const vector<int> &vals);`
  - `double probability(int x) const;`
  - `double mean() const;`
  - `double variance() const;`
  - `void display() const;`

## Operator Overloading

- `operator ==(())`, `operator !=()`, `operator <()`, `operator >()` Compares mean values of two discrete uniform random variables
- `operator <<()` Displays the variable's values, mean, and variance

## Tasks

1. Create a discrete uniform random variable with values  $\{1, 2, 3, 4, 5, 6\}$  (simulating a die)
2. Compute probability of a given value
3. Compute mean and variance
4. Create another variable with a different finite set and compare using overloaded operators
5. Display all results using the overloaded `<<` operator

## Expected Output Example

```
X ~ DiscreteUniform({1,2,3,4,5,6})
```

```
Probability(X=3) = 0.1667
```

```
Mean = 3.5, Variance = 2.9167
```

```
Y ~ DiscreteUniform({2,4,6,8})
```

```
Mean = 5.0, Variance = 5.0
```

```
X < Y : True
```

```
X == Y: False
```

## Project - 2: 2048 Game Using C++ Classes

### Problem Statement

Design and implement the popular **2048 Game** using **C++ classes**. The program should simulate the  $4 \times 4$  sliding tile game where the player combines numbers by sliding them in four directions to reach the tile with value 2048. The project should utilize object-oriented programming concepts such as classes, objects, encapsulation, and methods for handling game logic and user interaction.

### Project Requirements

1. Create a **Board** class to represent the  $4 \times 4$  game grid.
  - (a) Include a method to initialize the board with two random tiles of 2 or 4.
  - (b) Include a method to generate a new random tile in an empty position after each move.
  - (c) Include a method to display the current board in a nicely formatted way.
  - (d) Include a method to check for the game over condition.
2. Create a **Game** class to manage gameplay.

- (a) Display options for the user: **Play** or **Solution**.
  - (b) If the user chooses **Play**, show the navigation commands:
    - W/w for Up
    - S/s for Down
    - A/a for Left
    - D/d for Right
    - Q/q for Quit
  - (c) When the user presses Q/q, confirm before quitting.
  - (d) For each move, merge tiles according to 2048 rules and update the board.
  - (e) Display the board after each move.
3. If the user chooses **Solution**, demonstrate a sequence of moves that leads to a high-value tile (e.g., 2048).
4. Ensure proper encapsulation of game logic and board operations within the respective classes.

## Suggested Class Structure

1. **Board Class:**

- Data member:  $4 \times 4$  integer array representing tiles
- Methods: initializeBoard(), addRandomTile(), displayBoard(), isGameOver(), mergeTiles(char direction)

2. **Game Class:**

- Data member: Board object, user choice, score
- Methods: playGame(), showSolution(), processMove(char move), confirmQuit()