
Project - 1 MA517M-Basic Programming Laboratory	Last Date: 14 November 2026
Name	Roll No.: MA26M005

Geometric Random Variables using C++ Classes and Operator Overloading

Objective: To design a C++ program that implements geometric random variables using classes, computes PMF, CDF, mean, variance, and supports operator overloading for comparison and arithmetic.

A geometric random variable $X \sim \text{Geom}(p)$ counts the number of Bernoulli trials until the first success, with success probability p .

The probability mass function (PMF) is

$$P(X = k) = (1 - p)^{k-1}p, \quad k = 1, 2, 3, \dots$$

The cumulative distribution function (CDF) is

$$F(k) = P(X \leq k) = 1 - (1 - p)^k$$

Mean and variance:

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

Problem Description

Design a class `GeometricRV` to represent a geometric random variable. The class should allow computation of PMF, CDF, mean, variance, comparison with other geometric random variables, and arithmetic operations (such as addition of independent variables).

Class Specification

- **Class Name:** `GeometricRV`
- **Private Data Members:**
 - `double p;`
- **Public Member Functions:**
 - `GeometricRV(double prob);`
 - `double pmf(int k) const;`
 - `double cdf(int k) const;`
 - `double mean() const;`
 - `double variance() const;`
 - `void display() const;`

Operator Overloading

- `operator +()` Adds two independent geometric random variables (sum of trials until first success for each)
- `operator ==(())`, `operator !=()`, `operator <()`, `operator >()` Compares mean or variance of two geometric random variables
- `operator <<()` Displays the geometric variable's parameters and statistics

Tasks

1. Create geometric random variables $X \sim \text{Geom}(0.3)$ and $Y \sim \text{Geom}(0.5)$
2. Compute PMF and CDF for selected values
3. Compute mean and variance
4. Compute $X + Y$ assuming independence
5. Compare X and Y using overloaded comparison operators
6. Display all results using the overloaded `<<` operator

Expected Output Example

```
X ~ Geom(0.3)
PMF P(X=3) = 0.147
CDF P(X<=3) = 0.657
Mean = 3.333, Variance = 7.778
```

```
Y ~ Geom(0.5)
PMF P(Y=2) = 0.25
CDF P(Y<=2) = 0.75
Mean = 2.0, Variance = 2.0
```

```
X + Y (independent) = Geometric sum with mean 5.333, variance 9.778
```

```
X > Y : True
X == Y: False
```

Project - 2: Match-3 Game (Candy Crush Variant) Using C++ Classes

Problem Statement

Design and implement a **Match-3 Game** using **C++ classes**. The game consists of a grid of colored tiles (or symbols), where the player swaps adjacent tiles to form a sequence of three or more identical tiles in a row or column. When a match is formed, the tiles disappear, points are awarded, and new

tiles fall from the top to fill empty spaces. The project should utilize object-oriented programming concepts such as classes, objects, encapsulation, and methods to handle grid updates, scoring, and user interaction.

Project Requirements

1. Create a **Tile** class to represent an individual tile in the grid.
 - (a) Data member: type or color of the tile.
 - (b) Methods: `getType()`, `setType()`.
2. Create a **Board** class to represent the game grid.
 - (a) Initialize the grid with random tiles.
 - (b) Display the current state of the grid in a readable format.
 - (c) Detect and remove matches of three or more tiles in a row or column.
 - (d) Drop new tiles from the top to fill empty spaces.
 - (e) Include a method to check if possible moves exist.
3. Create a **Game** class to manage gameplay.
 - (a) Display options for the user: **Play** or **Solution**.
 - (b) Allow the user to swap adjacent tiles using input commands.
 - (c) Keep track of the player's score.
 - (d) Allow the user to quit and confirm before exiting.
 - (e) Optionally, allow the user to reset the board or play multiple rounds.
4. Ensure proper encapsulation of tile and board operations within the respective classes.

Suggested Class Structure

1. **Tile Class**:
 - Data member: `int type` or `char color`
 - Methods: `getType()`, `setType()`
2. **Board Class**:
 - Data member: 2D array of **Tile** objects
 - Methods: `initializeBoard()`, `displayBoard()`, `detectMatches()`, `removeMatches()`, `dropTiles()`, `hasPossibleMoves()`
3. **Game Class**:
 - Data member: **Board** object, user choice, score
 - Methods: `playGame()`, `showSolution()`, `processSwap(int row1, int col1, int row2, int col2)`, `confirmQuit()`, `resetBoard()`

Reference

For more details about Match-3 games and mechanics, visit: https://en.wikipedia.org/wiki/Tile-matching_video_game