

Project - 1 MA517M-Basic Programming Laboratory Last Date: 14 November 2026
Name _____ Roll No.: MA26M009

Euler's Totient Function using C++ Classes and Operator Overloading

Objective: To design a C++ program that implements Euler's totient function $\phi(n)$ using classes, computes it based on prime factorization, and supports comparison with other integers using operator overloading.

Euler's totient function $\phi(n)$ counts the number of positive integers up to n that are relatively prime to n . For example, $\phi(9) = 6$ since $\{1, 2, 4, 5, 7, 8\}$ are coprime to 9.

Problem Description

Design a class `Totient` to represent an integer n and compute $\phi(n)$ using prime factorization. The class should support arithmetic or comparison operations involving the totient value.

Class Specification

- **Class Name:** `Totient`
- **Private Data Members:**
 - `int n;`
 - `int phiValue;`
 - `void computePhi();`
- **Public Member Functions:**
 - `Totient(int x);`
 - `int getValue() const;`
 - `void display() const;`

Operator Overloading

- `operator ==(()), !=(), <(), >()` Compares the totient values of two objects
- `operator +()` Adds the totient values of two objects
- `operator <<()` Displays n and $\phi(n)$

Tasks

1. Create Totient objects for $n = 9$ and $m = 12$
2. Compute $\phi(9)$ and $\phi(12)$ using prime factorization
3. Compare totient values using overloaded comparison operators
4. Compute sum of $\phi(9) + \phi(12)$
5. Display results using the overloaded $<<$ operator

Expected Output Example

`n = 9, phi(n) = 6`

`m = 12, phi(m) = 4`

`phi(n) == phi(m) : False`

`phi(n) < phi(m) : False`

`phi(n) + phi(m) = 10`

Project - 2: 2048 Game Using C++ Classes

Problem Statement

Design and implement the popular **2048 Game** using **C++ classes**. The program should simulate the 4×4 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 4×4 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: 4×4 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()`