
Project - 1 MA517M-Basic Programming Laboratory	Last Date: 14 November 2026
Name	Roll No.: MA26M102

Prime Factorization and the Unique Factorization Theorem

Objective: To design a C++ program using **classes** and **operator overloading** to demonstrate the *Unique Factorization Theorem* for integers.

Every integer $n > 1$ can be represented uniquely (up to the order of factors) as a product of prime powers:

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \dots \cdot p_k^{a_k},$$

where p_i are primes and $a_i \geq 1$.

Problem Description: Design a C++ class `PrimeDecomposition` that stores an integer and its unique prime factorization, verifies arithmetic relations between integers, and allows operations between factorizations using operator overloading.

Class Specification

- **Class Name:** `PrimeDecomposition`
- **Private Data Members:**
 - `int number;` // stores the integer
 - `std::map<int, int> factors;` // key: prime, value: exponent
- **Public Member Functions:**
 - `PrimeDecomposition(int n);` Constructor to initialize the number and compute its prime factors.
 - `void display() const;` Displays the factorization in readable form. Example: $360 = 2^3 \cdot 3^2 \cdot 5^1$
 - `bool isPrime(int n);` Helper function to test primality.
 - `std::map<int,int> factorize(int n);` Returns a map of prime factors and their powers.

Operator Overloading

- `operator *()` Combines two prime decompositions to represent multiplication of integers:

$$(2^3 \cdot 3^2) * (2 \cdot 5) = 2^4 \cdot 3^2 \cdot 5^1$$

- `operator ==()` Checks whether two decompositions represent the same integer.
- `operator <()` and `operator >()` Compare two integers based on their numerical values.

Tasks

1. Create objects for the following integers:

$$n_1 = 360, \quad n_2 = 84, \quad n_3 = 1260$$

2. Display their unique prime factorizations:

$$360 = 2^3 \cdot 3^2 \cdot 5^1, \quad 84 = 2^2 \cdot 3^1 \cdot 7^1, \quad 1260 = 2^2 \cdot 3^2 \cdot 5^1 \cdot 7^1$$

3. Verify the relation:

$$(360 * 84) == (1260 * 24)$$

and display the resulting factorization.

4. Test comparison operations:

$$360 < 1260, \quad 84 > 24$$

Expected Output Example

```
360 = 2^3 * 3^2 * 5^1
```

```
84 = 2^2 * 3^1 * 7^1
```

```
1260 = 2^2 * 3^2 * 5^1 * 7^1
```

```
(360 * 84) = 2^5 * 3^3 * 5^1 * 7^1
```

```
(1260 * 24) = 2^5 * 3^3 * 5^1 * 7^1
```

```
The factorizations are equal.
```

```
360 < 1260 : True
```

```
84 > 24 : True
```

Concepts Covered

- Classes and encapsulation
- Operator overloading: `*`, `==`, `<`, `>`
- Prime factorization algorithm
- Implementation of the Unique Factorization Theorem
- Use of STL containers (`std::map`)

Project - 2 : Word Shuffle Game Using C++ Classes

Problem Statement

Design and implement a **Word Shuffle Game** using **C++ classes**. The program should allow the user to unscramble letters to form meaningful words. The project should utilize object-oriented programming concepts such as classes, objects, encapsulation, and methods to handle word selection, shuffling, and user interaction.

Project Requirements

1. Create a **Word** class to represent a word in the game.
 - (a) Include a method to randomly select a word from a predefined list.
 - (b) Include a method to shuffle the letters of the word.
 - (c) Include a method to display the shuffled word to the user.
2. Create a **Game** class to manage gameplay.
 - (a) Display options for the user: **Play** or **Solution**.
 - (b) If the user chooses **Play**, allow them to enter guesses.
 - (c) Validate user input and provide feedback if the guess is correct or incorrect.
 - (d) Keep track of the number of attempts.
 - (e) Allow the user to quit by entering a special command (e.g., **Q/q**) and confirm before exiting.
3. If the user chooses **Solution**, display the original word and the correct sequence of letters.
4. Ensure proper encapsulation of word logic and gameplay operations within the respective classes.

Suggested Class Structure

1. **Word Class**:
 - Data member: `string originalWord`, `string shuffledWord`
 - Methods: `selectWord()`, `shuffleWord()`, `displayWord()`
2. **Game Class**:
 - Data member: `Word` object, user choice, attempt counter
 - Methods: `playGame()`, `showSolution()`, `processGuess(string guess)`, `confirmQuit()`