INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Project - 1 **MA517M-Basic Programming Laboratory** Last Date: 09 November 2025
Name **Roll No.: MA25M106**

Implementation of Sets and their Operations using C++ and Classes

1. Design a C++ class `Set` to represent a mathematical set of real numbers. The set should:
   - Contains unique elements (no duplicates),
   - Allow dynamic resizing using arrays or STL `vector`,
   - Support both finite and empty sets.

   Example:
   $$A = \{1.5,\ 3.2,\ 7.8\}, \quad B = \{3.2,\ 4.9,\ 9.5\}$$

2. Implement the following constructors and basic member functions:
   - Default constructor (creates an empty set),
   - Parameterized constructor (creates a set from an array or vector),
   - Copy constructor and assignment operator,
   - A function `insert(double x)` that inserts an element if not already present,
   - A function `remove(double x)` that removes an element if present,
   - A function `contains(double x)` that checks if $x \in$ Set,
   - A function `size()` that returns the number of elements,
   - A function `print()` to display the set.

3. Use **operator overloading** to define the following set operations:
   $$\begin{aligned} A + B &\implies A \cup B \quad \text{(Union)} \\ A * B &\implies A \cap B \quad \text{(Intersection)} \\ A - B &\implies A \setminus B \quad \text{(Difference)} \end{aligned}$$

   Example:
   $$A = \{1, 2, 3, 4\}, \quad B = \{3, 4, 5\} \implies A + B = \{1, 2, 3, 4, 5\}, \quad A * B = \{3, 4\}, \quad A - B = \{1, 2\}$$

4. Implement comparison operator overloads:
   $$\begin{aligned} A == B &\iff \text{if all elements are identical,} \\ A < B &\iff A \subset B, \\ A > B &\iff A \supset B. \end{aligned}$$

5. Implement a function to compute the **power set** $\mathcal{P}(A)$ of a given set A and display all subsets.

$$A = \{a, b, c\} \implies \mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}\}$$

6. Add additional utility functions:
   - `isDisjoint(const Set& B)` — checks if $A \cap B = \emptyset$,
   - `isSubset(const Set& B)` — checks if $A \subseteq B$,
   - `cardinality()` — returns the number of elements in the set,
   - `complement(Set U)` — returns the complement of A with respect to universal set U.

7. Develop a main program to:
   - Create three sets A, B, and U (universal set),
   - Perform all operations $A + B$, $A * B$, $A - B$, and compare A and B,
   - Display whether A and B are disjoint,
   - Display $\mathcal{P}(A)$,
   - Compute the complement of A w.r.t. U.

8. (Optional Extension:) Implement the **symmetric difference** operation using operator overloading:

$$A \oplus B = (A - B) \cup (B - A)$$

   and overload it using the operator `^`:

$$A \,\hat{}\, B$$

Project - 2: Conway's Game of Life Using C++ Classes

Problem Statement

Design and implement **Conway's Game of Life** using **C++ classes**. This is a zero-player game where the evolution of a 2D grid is determined by its initial state and simple rules. Each cell in the grid can be *alive* or *dead*, and the next generation of the grid is calculated based on the number of alive neighbors. The project should utilize object-oriented programming concepts such as classes, objects, encapsulation, and methods for handling game logic and grid updates.

Project Requirements

1. Create a `Cell` class to represent a single cell in the grid.
   (a) Data member: current state (alive or dead)
   (b) Methods: setState(), getState()

2. Create a `Grid` class to represent the game board (matrix).
   (a) Initialize the grid with random alive and dead cells.
   (b) Include a method to display the current state of the grid.

(c) Include a method to count alive neighbors for each cell.

(d) Include a method to update the grid according to the following rules:

- Any live cell with fewer than two live neighbors dies (underpopulation).
- Any live cell with two or three live neighbors lives on.
- Any live cell with more than three live neighbors dies (overpopulation).
- Any dead cell with exactly three live neighbors becomes alive (reproduction).

3. Create a `Game` class to manage simulation.

   (a) Display options for the user: `Start`, `Next Generation`, or `Quit`.

   (b) Allow the user to advance the simulation one generation at a time.

   (c) Allow the user to quit and confirm before exiting.

   (d) Optionally, allow the user to specify the number of generations to simulate automatically.

4. Ensure proper encapsulation of cell and grid operations within the respective classes.

Suggested Class Structure

1. `Cell` Class:
   - Data member: `bool isAlive`
   - Methods: setState(bool state), getState()
2. `Grid` Class:
   - Data member: 2D array of `Cell` objects
   - Methods: initializeGrid(), displayGrid(), countAliveNeighbors(int row, int col), updateGrid()
3. `Game` Class:
   - Data member: `Grid` object, user choice, generation counter
   - Methods: startGame(), nextGeneration(), confirmQuit()

Reference

For more details about Conway's Game of Life, visit: `https://en.wikipedia.org/wiki/Conway's_Game_of_Life`