INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Class Test - 7 **MA517M-Basic Programming Laboratory** 29 September 2025

Name **Roll No.: MA25M**

Recursive Maze Solver

A robot is placed in a **2D grid maze** of size $N \times M$. Some cells are blocked (value 1), some are open (value 0). The robot can move only **right** or **down**.

Your task as a group is to:

1. Recursively **count all possible paths** from the top-left cell $(0, 0)$ to the bottom-right cell $(N-1, M-1)$.
2. Recursively **print one valid path**.
3. Represent the maze using a **C++ structure** for each cell:

```
struct Cell {
    int x, y;      // Coordinates
    bool blocked;  // true if the cell is blocked
};
```

4. Display the maze with the path marked.

Input Example

```
Grid (0 = open, 1 = blocked):
0 0 0
0 1 0
0 0 0
```

Expected Output Example

```
Total paths: 2
Path example: (0,0) -> (0,1) -> (0,2) -> (1,2) -> (2,2)
```

Hints

- Base Case: Reached destination $(N-1, M-1) \Rightarrow 1$ path
- Blocked cell $\Rightarrow 0$ paths

- Recursive formula for counting paths:

$$\text{paths}(i, j) = \text{paths}(i + 1, j) + \text{paths}(i, j + 1)$$

- For printing a path, maintain a vector of cells visited and backtrack recursively.
- Use a `struct Cell` to store coordinates and blocked status.

Group Task Distribution (Suggested)

1. **Student 1:** Implement recursive counting of paths.
2. **Student 2:** Implement recursive path printing.
3. **Student 3:** Input maze creation, validation, and structure initialization.
4. **Student 4:** Display maze with marked path, handle output formatting.

Notes

- Do **not use loops** for path counting/printing; recursion must be used.
- You may use loops only for input or displaying the maze.

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Class Test - 7 **MA517M-Basic Programming Laboratory** 29 September 2025

Name **Roll No.: MA25M**

Recursive Treasure Hunt

A treasure hunter is in a **2D grid** of size $N \times M$. Each cell contains some gold (non-negative integer). Some cells may be blocked (cannot step on). The hunter can move **right**, **down**, or **diagonally (right-down)**.

Your group task:

1. Recursively **find the maximum gold** collectible from the top-left cell $(0, 0)$ to the bottom-right cell $(N - 1, M - 1)$.
2. Recursively **print one path** that collects this maximum gold.
3. Represent the grid using a **C++ structure** for each cell:

```
struct Cell {
    int x, y;     // Coordinates
    int gold;     // Gold in this cell (0 if blocked)
    bool blocked; // true if cell cannot be entered
};
```

4. Display the grid with the path marked.

Input Example

```
Grid (0 = blocked, numbers = gold):
2 3 0
0 4 5
1 0 2
```

Expected Output Example

```
Maximum gold: 12
Path example: (0,0) -> (0,1) -> (1,2) -> (2,2)
```

Hints

- Base Case: Reached destination $(N-1, M-1) \Rightarrow$ return gold in this cell
- Blocked cell $\Rightarrow$ return 0
- Recursive formula for maximum gold:

$$gold = cell.gold + \max(collectGold(i+1, j), collectGold(i, j+1), collectGold(i+1, j+1))$$

- For printing the path, maintain a vector of `Cell` and backtrack recursively.

Group Task Distribution (Suggested)

1. **Student 1:** Recursive function to compute maximum gold.
2. **Student 2:** Recursive function to print one path with maximum gold.
3. **Student 3:** Input grid creation and validation using `Cell` structures.
4. **Student 4:** Display grid, print collected gold, and format output.

Notes

- Do **not use loops** for gold collection/path finding; recursion must be used.
- Loops may be used only for input or displaying the grid.

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Class Test - 7 **MA517M-Basic Programming Laboratory** 29 September 2025

Name **Roll No.: MA25M**

Island Counting problem

You are given a **2D grid map** of size $N \times M$ consisting of 0s (water) and 1s (land). An **island** is a group of connected 1s (land) connected **horizontally, vertically, or diagonally**.

Your group task:

1. Recursively **count the total number of islands** in the grid.
2. Recursively **mark all the cells of one island** for visualization.
3. Represent the grid using a **C++ structure** for each cell:

```
struct Cell {
    int x, y;     // Coordinates
    bool isLand;  // true if cell is land
    bool visited; // true if visited during recursion
};
```

4. Display the grid with one island highlighted (optional).

Input Example

```
Grid (0 = water, 1 = land):
1 1 0
0 1 0
1 0 1
```

Expected Output Example

```
Total islands: 3
Cells of first island: (0,0), (0,1), (1,1)
```

Hints

- Base Case: Cell is out of bounds, water, or already visited → return
- Recursive case: For a land cell, mark it visited, then recursively explore all **8 neighbors** (horizontal, vertical, diagonal)
- Use a vector of `Cell` to record the coordinates of one island

Group Task Distribution (Suggested)

1. **Student 1:** Implement recursive function to count total islands.
2. **Student 2:** Implement recursive function to record and print cells of one island.
3. **Student 3:** Input grid creation and structure initialization.
4. **Student 4:** Display grid, format output, and optionally visualize islands.

Notes

- Do **not use loops** for island counting; recursion must be used.
- Loops may be used only for input or displaying the grid.

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Class Test - 7 **MA517M-Basic Programming Laboratory** 29 September 2025
Name **Roll No.: MA25M**

Josephus Problem

Scenario:

A cloud computing system schedules n tasks in a circular queue. Due to limited resources, every k^{th} task is terminated to free up CPU time. The system administrator wants to know which task will survive the elimination process to ensure critical operations continue uninterrupted.

Evolution to Programming Problem:

1. Each task is assigned a unique ID from 1 to n. 2. Starting from the first task, every k^{th} task is removed in order until only one remains. 3. The safe task ID must be computed efficiently.

Recursive Relation:

Let $f(n, k)$ be the safe task position among n tasks with step k:

$$f(1, k) = 0, \quad f(n, k) = (f(n-1, k) + k) \bmod n$$

Add 1 at the end to shift from 0-based to 1-based indexing.

Programming Tasks:

1. Implement a recursive function to compute the surviving task ID.
2. (Optional) Use a `struct` or pointer-based array to store the elimination order for analysis.
3. Write a `main()` function to take inputs n and k, and output the surviving task.

Example: Input: $n = 7$ tasks, $k = 3$ Output: Task 4 survives

Fibonacci Problem

Scenario:

A wildlife researcher is monitoring the population of a species in a reserve. The population grows according to Fibonacci-like patterns: each new generation is the sum of the previous two generations. The researcher is particularly interested in even-numbered populations because only these can be divided evenly among sub-regions for conservation studies.

Evolution to Programming Problem:

1. Let $F(n)$ be the population at generation n:

$$F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2)$$

2. Track each generation's population along with its parity (even/odd). 3. Compute the sum of all even populations up to generation N.

Structure Definition:

```
struct FibInfo {
    int value;   // population
    bool isEven; // true if population is even
};
```

Programming Tasks:

1. Implement a recursive function to generate Fibonacci numbers up to N.
2. Store each generation's population and parity in a pointer-based dynamic array of `FibInfo`.
3. Calculate and output the sum of even populations.
4. Display each generation's population along with its parity.

Example: Input: $N = 10$ generations Output:

```
Fib(0) = 0 -> Even
Fib(1) = 1 -> Odd
Fib(2) = 1 -> Odd
Fib(3) = 2 -> Even
Fib(4) = 3 -> Odd
Fib(5) = 5 -> Odd
Fib(6) = 8 -> Even
Fib(7) = 13 -> Odd
Fib(8) = 21 -> Odd
Fib(9) = 34 -> Even
Fib(10) = 55 -> Odd
Sum of even populations = 44
```

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Class Test - 7 **MA517M-Basic Programming Laboratory** 29 September 2025

Name **Roll No.: MA25M**

Recursive Digit Sum

A financial system handles extremely large transaction IDs. To quickly verify data integrity, the system computes a checksum by summing the digits of each transaction ID. For very large numbers, this operation must be implemented recursively to handle numbers of arbitrary size efficiently.

Evolution to Programming Problem:

1. Given a number n, compute the sum of its digits recursively. 2. Store the number and its digit sum in a structure for reporting or logging purposes.

Recursive Formula:

$$F(n) = \begin{cases} n & n < 10 \\ F(\lfloor n/10 \rfloor) + (n \bmod 10) & n \geq 10 \end{cases}$$

Structure Definition:

```
struct DigitSumInfo {
    long long number;
    int sum;
};
```

Programming Tasks:

1. Implement a recursive function to compute the digit sum of a number.
2. Use a pointer-based structure to store the number and its digit sum.
3. Display the number along with its computed digit sum.

Example: Input: $n = 987654$ Output: `Number: 987654, Digit Sum: 39`

Collatz Sequence

In a research simulation of population dynamics, the population count evolves according to a rule-based system. For even populations, it halves (resource scarcity), and for odd populations, it increases following a nonlinear growth rule. Researchers want to study how many steps it takes for a population to reach stability (1). This is modeled by the Collatz sequence.

Evolution to Programming Problem:

1. Implement the Collatz sequence recursively:

$$n \to \begin{cases} n/2 & \text{if } n = 2k \\ 3n+1 & \text{if } n = 2k+1 \end{cases}, \quad k \in \mathbb{N}$$

2. Count the length of the sequence until $n = 1$. 3. Store the starting number and the length of the sequence in a structure for reporting.

Structure Definition:

```
struct CollatzInfo {
    long long number;
    int length;
};
```

Programming Tasks:

1. Implement a recursive function to compute the length of the Collatz sequence for a given number.
2. Store each number and its sequence length in a pointer-based structure.
3. Display the number along with the length of its Collatz sequence.

Example: Input: $n = 6$ Collatz sequence: 6, 3, 10, 5, 16, 8, 4, 2, 1 Output: `Number: 6, Sequence Length: 9`

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Class Test - 7 **MA517M-Basic Programming Laboratory** 29 September 2025

Name **Roll No.: MA25M**

Recursive Fast Exponentiation

You are asked to design a recursive program in C++ that calculates the power of a number efficiently.

Use the following structure to represent the input:

```
struct Power {
    int base;
    int exponent;
};
```

Write a recursive function `power(int base, int exp)` that:

(a) Returns 1 if the exponent is 0.

(b) If the exponent is even, compute the result as:

$$power(base, exp) = \big(power(base, exp/2)\big)^2$$

(c) If the exponent is odd, computes the result as:

$$power(base, exp) = base \times power(base, exp - 1)$$

Example

- Input: base = 2, exponent = 10
- Output: $2^{10} = 1024$

Instructions for Students

1. Work in groups of 4.
2. Store the values of base and exponent in the given structure.
3. Implement the recursive function as described above.
4. Test your program with different inputs (small and large exponents).

Fibonacci Problem

Scenario:

A wildlife researcher is monitoring the population of a species in a reserve. The population grows according to Fibonacci-like patterns: each new generation is the sum of the previous two generations. The researcher is particularly interested in even-numbered populations because only these can be divided evenly among sub-regions for conservation studies.

Evolution to Programming Problem:

1. Let $F(n)$ be the population at generation n:

$$F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2)$$

2. Track each generation's population along with its parity (even/odd). 3. Compute the sum of all even populations up to generation N.

Structure Definition:

```
struct FibInfo {
    int value;   // population
    bool isEven; // true if population is even
};
```

Programming Tasks:

1. Implement a recursive function to generate Fibonacci numbers up to N.
2. Store each generation's population and parity in a pointer-based dynamic array of `FibInfo`.
3. Calculate and output the sum of even populations.
4. Display each generation's population along with its parity.

Example: Input: $N = 10$ generations Output:

```
Fib(0) = 0 -> Even
Fib(1) = 1 -> Odd
Fib(2) = 1 -> Odd
Fib(3) = 2 -> Even
Fib(4) = 3 -> Odd
Fib(5) = 5 -> Odd
Fib(6) = 8 -> Even
Fib(7) = 13 -> Odd
Fib(8) = 21 -> Odd
Fib(9) = 34 -> Even
Fib(10) = 55 -> Odd
Sum of even populations = 44
```

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

Class Test - 7 **MA517M-Basic Programming Laboratory** 29 September 2025

Name **Roll No.: MA25M**

Recursive Polynomial Operations

A polynomial is represented as:

$$P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

You are asked to manage polynomials using **structures**, **pointers**, and **recursion**.

1. Define a `struct` called `Polynomial` with the following fields:

```
struct Polynomial {
    int* coefficients;
    int degree;
};
```

2. Implement a **recursive function** `printPolynomial` to print the polynomial in standard form.
3. Implement a **recursive function** `evaluatePolynomial` that calculates the value of the polynomial at a given x.
4. Implement a **function using pointers** `setCoefficient` to dynamically assign the coefficient of a given power.

1. Recursive Formula for Polynomial Evaluation

Let $E(n, x)$ denote the value of the polynomial using coefficients up to degree n. The recursion is:

$$E(n,x) = \begin{cases} 0 & \text{if } n < 0 \\ a_n \cdot x^n + E(n-1,x) & \text{if } n \geq 0 \end{cases}$$

- Base case: $E(-1, x) = 0$, i.e., no terms left.
- Recursive step: Take the current term a_nx^n and add the evaluation of all lower-degree terms.

2. Recursive Formula for Polynomial Printing

Let $P(n)$ denote the polynomial string up to degree n. The recursion is:

$$P(n) = \begin{cases} \text{""} & \text{if } n < 0 \\ P(n-1) + \text{formatTerm}(a_n, n) & \text{if } n \geq 0 \end{cases}$$

Where $\text{formatTerm}(a_n, n)$ is defined as:

$$\text{formatTerm}(a_n, n) = \begin{cases} \text{""} & \text{if } a_n = 0 \\ (\text{sign}) \cdot |a_n| \cdot x^n & \text{otherwise} \end{cases}$$

Example:

- Polynomial: $3x^3 + 5x^2 - 2x + 7$
- `printPolynomial()` should output: `3x^3 + 5x^2 - 2x + 7`
- `evaluatePolynomial(2)` should return: $3 \cdot 2^3 + 5 \cdot 2^2 - 2 \cdot 2 + 7 = 39$

Hints:

- Use a `Polynomial` struct with a dynamically allocated array for coefficients.
- Recursion can be used to traverse the array for printing and evaluating.
- Use `new` to allocate the array of coefficients dynamically.