Problem 1: Identifying the White Box in a Black & White Matrix

1. **Initialization: Begin by setting variables to track the top-left corner, height, and width of the white box.**
2. **Finding the Top-Left Corner: Iterate over the matrix to find the first occurrence of a 'w' character. This position will be the top-left corner of the white box.**
3. **Determining Width and Height: Continue iterating to determine the width (how many 'w' characters are there horizontally from the top-left corner) and height (how many 'w' characters are there vertically from the top-left corner).**
4. **Return Values: Finally, return an object or a structure containing the top-left corner coordinates, the width, and the height of the white box.**

Sol:

```
  const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

class WhiteBox {
  constructor(top, left, height, width) {
    this.top = top;
    this.left = left;
    this.height = height;
    this.width = width;
  }
}

function findWhiteBox(matrix) {
  const numRows = matrix.length;
  const numCols = matrix[0].length;

  let top = numRows;
  let bottom = 0;
  let left = numCols;
  let right = 0;

  // Iterate through the matrix to find the boundaries of the white box
  for (let i = 0; i < numRows; i++) {
    for (let j = 0; j < numCols; j++) {
        if (matrix[i][j] === 'w') {
            top = Math.min(top, i);
            bottom = Math.max(bottom, i);
            left = Math.min(left, j);
            right = Math.max(right, j);
        }
    }
  }

  // Calculate the dimensions of the white box
  const height = bottom - top + 1;
  const width = right - left + 1;

  // Return the WhiteBox object
  return new WhiteBox(top, left, height, width);
}

// Function to prompt the user and read the input matrix
function getInputMatrix() {
  return new Promise((resolve, reject) => {
    const matrix = [];
    console.log('Enter the pixel matrix (b for black, w for white):');
    rl.prompt();
    rl.on('line', (line) => {
        const row = line.trim().split(' ');
        matrix.push(row);
        if (matrix.length === 12) {
            rl.close();
            resolve(matrix);
        }
    });
  });
}

async function main() {
  const matrix = await getInputMatrix();
  // Find the white box in the matrix
  const whiteBox = findWhiteBox(matrix);
  console.log('White box found:', whiteBox);
}

main();
```

Problem 2. the given array [9,33,0,7,2,82,77], WAP to divide each number of the array by the next number. Divide the last number by first number of array. Provide proper exceptional handling for 0

Array Division and Exception Handling

1. **Iteration: Go through the array elements one by one.**
2. **Division Logic: For each element, divide it by the next element in the array. For the last element, divide it by the first element.**
3. **Exception Handling: Implement a check for division by zero and handle it appropriately, possibly by skipping the operation or setting a specific value (like None or an error message).**

Sol:

```
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

function arrayDivision(arr) {
  const result = [];

  for (let i = 0; i < arr.length; i++) {
    let dividend = arr[i];
    let divisor = arr[(i + 1) % arr.length]; // Index of the next element or 0 if it's the last element

    try {
      if (divisor === 0) {
        throw new Error('Division by zero encountered');
      }

      result.push(dividend / divisor);
    } catch (error) {
      console.error(`Error: ${error.message}. Skipping division.`);
      result.push(null); // Or any other value you want to use to indicate an error
    }
  }

  return result;
}

// Prompt the user to enter array elements separated by spaces
rl.question("Enter array elements separated by spaces (e.g., 1 2 3): ", (input) => {
  // Convert the input string to an array of numbers
  const arr = input.split(" ").map(Number);

  // Perform array division
  const divisionResult = arrayDivision(arr);
  console.log("Division Result:", divisionResult);

  rl.close();
});
```

3.Sum of Numbers Divisible by 3 in a String

1. **Extract Numbers**: Use a method to find all numbers in the given string.
2. **Check Divisibility and Sum**: Iterate through these numbers. If a number is divisible by 3, add it to a sum variable and keep track of it as the last divisible number found.
3. **Result**: After iteration, return the total sum and the last number that was divisible by 3

```
function sumDivisibleBy3(string) {
    let totalSum = 0;
    let lastDivisible = null;

    const numbers = string.match(/\d+/g);

    if (numbers) {
      Check divisibility and sum
        for (const numStr of numbers) {
            const num = parseInt(numStr);
            if (num % 3 === 0) {
                totalSum += num;
                lastDivisible = num;
            }
        }
    }
    return [totalSum, lastDivisible];
}


const userInput = prompt("Enter a string containing numbers:");

const [sumResult, lastResult] = sumDivisibleBy3(userInput);
console.log("Sum:", sumResult);
console.log("Last Divisible:", lastResult);
```

4.There are 100 man making a circle each man is wearing a T-shit with a number 1 to 100 in series. Person with Number 1 on his/her T-Shirt got a gun now 1 kill 2 and give that gun to 3 and then 3 kill 4 and give that gun to 5.. then so on 99 killed 100 and give that gun again to 1.WAP to find which man is left with a gun on his hand at the end ??

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

function josephus(n, k) {
    if (n === 1) {
        return 1;
    } else {
        return (josephus(n - 1, k) + k - 1) % n + 1;
    }
}

// Prompt the user to enter the total number of people and the counting interval (k)
rl.question("Enter the total number of people: ", (n) => {
    rl.question("Enter the counting interval (k): ", (k) => {
        // Convert input strings to integers
        n = parseInt(n);
        k = parseInt(k);

        // Find the position of the last survivor
        const lastSurvivorPosition = josephus(n, k);
        console.log("The last survivor's position is:", lastSurvivorPosition);

        rl.close();
    });
});
```

5.Database Schema for Hotels and Menus

1. **Identify Entities and Relationships**: Recognize that there are three main entities: Hotels, Menus, and Food Items. Understand the relationships: a hotel can have multiple menus, a menu can have multiple food items, and food items can be on multiple menus.
2. **Designing Tables**: Create separate tables for Hotels, Menus, and Food Items.
3. **Managing Many-to-Many Relationship**: Since a food item can belong to multiple menus, create a junction table (like MenuFoodItems) to handle this many-to-many relationship.
4. **Key Considerations**: Define primary keys for each table and use foreign keys to establish relationships between tables.

1.Hotels Table:

Columns:

- hotel_id (Primary Key)
- hotel_name
- location
- other hotel details

2.Menus Table:

Columns:

- menu_id (Primary Key)
- hotel_id (Foreign Key referencing hotel_id in the Hotels table)
- menu_name
- description
- other menu details

3.Food Items Table:

Columns:

- food_item_id (Primary Key)
- food_item_name
- description
- price
- other food item details

4.MenuFoodItems Table (Junction Table for Many-to-Many Relationship between Menus and Food Items):

Columns:

- menu_id (Foreign Key referencing menu_id in the Menus table)

- food_item_id (Foreign Key referencing food_item_id in the Food Items table)
- quantity (optional, if quantity of each food item in a menu is needed)