**Assignment 1**

Following things to be added in each question:

-Program

-Flow chart

-Explanation

-Output

-Time and Space complexity

Submission Date: 26/09/2024

1. Armstrong Number

Problem: Write a Java program to check if a given number is an Armstrong number.

Test Cases:

Input: 153

Output: true

Input: 123

Output: false

public class QE1 {

    public static boolean isArmstrong(int number) {

        int originalNumber = number;

        int sum = 0;

        int numDigits = String.valueOf(number).length();

        while (number > 0) {

            int digit = number % 10;

            sum += Math.pow(digit, numDigits);

            number /= 10;

        }

        return sum == originalNumber;

    }

    public static void main(String[] args) {

        int testCase1 = 153;

        int testCase2 = 123;

        System.out.println(testCase1 + " is Armstrong: " + isArmstrong(testCase1));

        System.out.println(testCase2 + " is Armstrong: " + isArmstrong(testCase2));

    }

}

2. Prime Number

Problem: Write a Java program to check if a given number is prime.

Test Cases:

Input: 29

Output: true

Input: 15

Output: false

public class QE2 {

    public static boolean isPrime(int number) {

        if (number <= 1) {

            return false;

        }

        for (int i = 2; i <= Math.sqrt(number); i++) {

            if (number % i == 0) {

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        int testCase1 = 29;

        int testCase2 = 15;

        System.out.println(testCase1 + " is Prime: " + isPrime(testCase1)); // Output: true

        System.out.println(testCase2 + " is Prime: " + isPrime(testCase2)); // Output: false

    }

}

3. Factorial

Problem: Write a Java program to compute the factorial of a given number.

Test Cases:

Input: 5

Output: 120

Input: 0

Output: 1

public class QE3 {

    public static int factorial(int number) {

        int result = 1;

        for (int i = 2; i <= number; i++) {

            result \*= i;

        }

        return result;

    }

    public static void main(String[] args) {

        int testCase1 = 5;

        int testCase2 = 0;

        System.out.println("Factorial of " + testCase1 + ": " + factorial(testCase1)); // Output: 120

        System.out.println("Factorial of " + testCase2 + ": " + factorial(testCase2)); // Output: 1

    }

}

4. Fibonacci Series

Problem: Write a Java program to print the first n numbers in the Fibonacci series.

Test Cases:

Input: n = 5

Output: [0, 1, 1, 2, 3]

Input: n = 8

Output: [0, 1, 1, 2, 3, 5, 8, 13]

import java.util.Arrays;

public class QE4 {

    public static int[] fibonacci(int n) {

        int[] fibSeries = new int[n];

        if (n > 0) {

            fibSeries[0] = 0;

        }

        if (n > 1) {

            fibSeries[1] = 1;

        }

        for (int i = 2; i < n; i++) {

            fibSeries[i] = fibSeries[i - 1] + fibSeries[i - 2];

        }

        return fibSeries;

    }

    public static void main(String[] args) {

        int testCase1 = 5;

        int testCase2 = 8;

        System.out.println("First " + testCase1 + " Fibonacci numbers: " + Arrays.toString(fibonacci(testCase1))); // Output: [0, 1, 1, 2, 3]

        System.out.println("First " + testCase2 + " Fibonacci numbers: " + Arrays.toString(fibonacci(testCase2))); // Output: [0, 1, 1, 2, 3, 5, 8, 13]

    }

}

5. Find GCD

Problem: Write a Java program to find the Greatest Common Divisor (GCD) of two numbers.

Test Cases:

Input: a = 54, b = 24

Output: 6

Input: a = 17, b = 13

Output: 1

public class QE5 {

    public static int findGCD(int a, int b) {

        while (b != 0) {

            int temp = b;

            b = a % b;

            a = temp;

        }

        return a;

    }

    public static void main(String[] args) {

        int a1 = 54, b1 = 24;

        int a2 = 17, b2 = 13;

        System.out.println("GCD of " + a1 + " and " + b1 + ": " + findGCD(a1, b1)); // Output: 6

        System.out.println("GCD of " + a2 + " and " + b2 + ": " + findGCD(a2, b2)); // Output: 1

    }

}

6. Find Square Root

Problem: Write a Java program to find the square root of a given number (using integer approximation).

Test Cases:

Input: x = 16

Output: 4

Input: x = 27

Output: 5

public class QE6 {

    public static int findSquareRoot(int x) {

        if (x == 0 || x == 1) {

            return x;

        }

        int start = 1, end = x, result = 0;

        while (start <= end) {

            int mid = (start + end) / 2;

            if (mid \* mid == x) {

                return mid;

            }

            if (mid \* mid < x) {

                start = mid + 1;

                result = mid;

            } else {

                end = mid - 1;

            }

        }

        return result;

    }

    public static void main(String[] args) {

        int testCase1 = 16;

        int testCase2 = 27;

        System.out.println("Square root of " + testCase1 + ": " + findSquareRoot(testCase1)); // Output: 4

        System.out.println("Square root of " + testCase2 + ": " + findSquareRoot(testCase2)); // Output: 5

    }

}

7. Find Repeated Characters in a String

Problem: Write a Java program to find all repeated characters in a string.

Test Cases:

Input: "programming"

Output: ['r', 'g', 'm']

Input: "hello"

Output: ['l']

import java.util.HashMap;

import java.util.ArrayList;

import java.util.List;

public class QE7 {

    public static List<Character> findRepeatedCharacters(String str) {

        HashMap<Character, Integer> charCount = new HashMap<>();

        List<Character> repeatedChars = new ArrayList<>();

        for (char ch : str.toCharArray()) {

            charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);

        }

        for (char ch : charCount.keySet()) {

            if (charCount.get(ch) > 1) {

                repeatedChars.add(ch);

            }

        }

        return repeatedChars;

    }

    public static void main(String[] args) {

        String testCase1 = "programming";

        String testCase2 = "hello";

        System.out.println("Repeated characters in \"" + testCase1 + "\": " + findRepeatedCharacters(testCase1)); // Output: [r, g, m]

        System.out.println("Repeated characters in \"" + testCase2 + "\": " + findRepeatedCharacters(testCase2)); // Output: [l]

    }

}

8. First Non-Repeated Character

Problem: Write a Java program to find the first non-repeated character in a string.

Test Cases:

Input: "stress"

Output: 't'

Input: "aabbcc"

Output: null

9. Integer Palindrome

Problem: Write a Java program to check if a given integer is a palindrome.

Test Cases:

Input: 121

Output: true

Input: -121

Output: false

10. Leap Year

Problem: Write a Java program to check if a given year is a leap year.

Test Cases:

Input: 2020

Output: true

Input: 1900

Output: false