**Subject: Algorithm and Data Structure**

**Assignment 1**

**Solve the assignment with following thing to be added in each question.**

**-Program**

**-Flow chart**

**-Explanation**

**-Output**

**-Time and Space complexity**

**Program code:**

**import java.util.Scanner;**

**public class Armstrong {**

**public static void main(String[] args) {**

**Scanner sc = new Scanner(System.in);**

**System.out.print("Enter a number: ");**

**int number = sc.nextInt();**

**sc.close();**

**boolean isArmstrong = checkArmstrong(number);**

**System.out.println("Is Armstrong Number: " + isArmstrong);**

**}**

**public static boolean checkArmstrong(int number) {**

**int temp = number;**

**int digits = 0, sum = 0;**

**while (temp != 0) {**

**digits++;**

**temp /= 10;**

**}**

**temp = number;**

**while (temp != 0) {**

**int remainder = temp % 10;**

**sum += Math.pow(remainder, digits);**

**temp /= 10;**

**}**

**return sum == number;**

**}**

**}**

**Output:**

**C:\Users\rushi\ADS>javac Armstrong.java**

**C:\Users\rushi\ADS>java Armstrong**

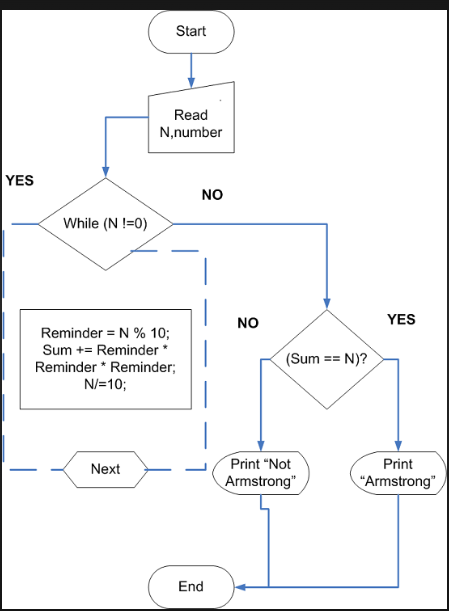
**Enter a number: 153**

**Is Armstrong Number: true**

**C:\Users\rushi\ADS>java Armstrong**

**Enter a number: 123**

**Is Armstrong Number: false**

**Flowchart:Flowchart would look something like this**

**Explanation:**

1. **Input: The program starts by taking an integer input from the user.**
2. **Digit Calculation: We calculate the number of digits in the input number by dividing the number iteratively by 10.**
3. **Digit Power Sum: Then, for each digit in the number, we compute the digit raised to the power of the number of digits and sum them up.**
4. **Comparison: Finally, we compare this sum with the original number. If they are equal, the number is an Armstrong number; otherwise, it is not.**

**Time and Space Complexity:**

* **Time Complexity: O(d), where d is the number of digits in the input number. This is because we process each digit of the number once to compute the sum of powered digits.**
* **Space Complexity: O(1), since we are only using a constant amount of extra space, regardless of the size of the input number.**

**2. Prime Number**

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

**Program:**

**import java.util.Scanner;**

**public class PrimeNumber {**

**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) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter a number: ");**

**int number = scanner.nextInt();**

**if (isPrime(number)) {**

**System.out.println(number + " True");**

**} else {**

**System.out.println(number + " False");**

**}**

**scanner.close();**

**}**

**}**

**Output:**

**C:\Users\rushi\ADS>javac PrimeNumber.java**

**C:\Users\rushi\ADS>java PrimeNumber**

**Enter a number: 29**

**29 True**

**C:\Users\rushi\ADS>java PrimeNumber**

**Enter a number: 15**

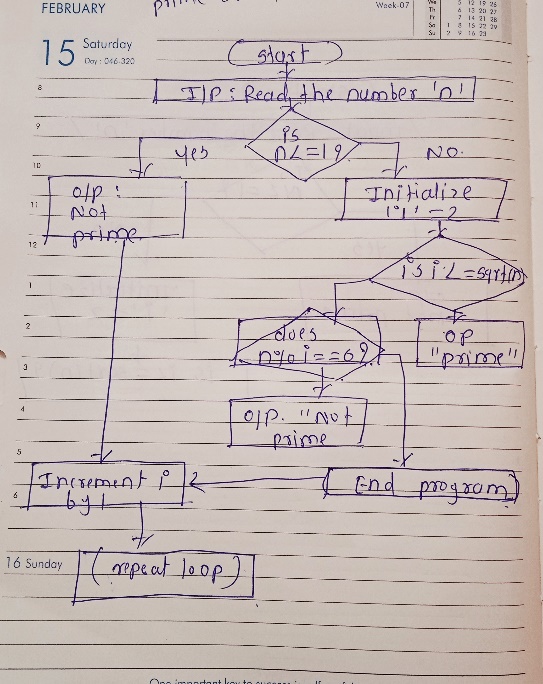
**15 False**

**Explanation:**

1. **Input: The user inputs an integer.**
2. **Prime Check:**
   * **Numbers less than or equal to 1 are automatically not prime.**
   * **For numbers greater than 1, we check divisibility by integers starting from 2 up to the square root of the number. If the number is divisible by any of these integers, it is not a prime number.**
   * **If no divisors are found, the number is prime.**
3. **Output: The program prints whether the number is prime or not based on the above logic.**

**Time Complexity: O(n)**

**Space Complexity:O(1)**

****

**3. Factorial**

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

**Program code:**

**import java.util.Scanner;**

**public class Factorial {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter a number to compute its factorial: ");**

**int number = scanner.nextInt();**

**scanner.close();**

**long factorial = calculateFactorial(number);**

**System.out.println("Factorial of " + number + " is: " + factorial);**

**}**

**public static long calculateFactorial(int num) {**

**long fact = 1;**

**for (int i = 1; i <= num; i++) {**

**fact \*= i;**

**}**

**return fact;**

**}**

**}**

**Output:**

**C:\Users\rushi\ADS>java Factorial**

**Enter a number to compute its factorial: 5**

**Factorial of 5 is: 120**

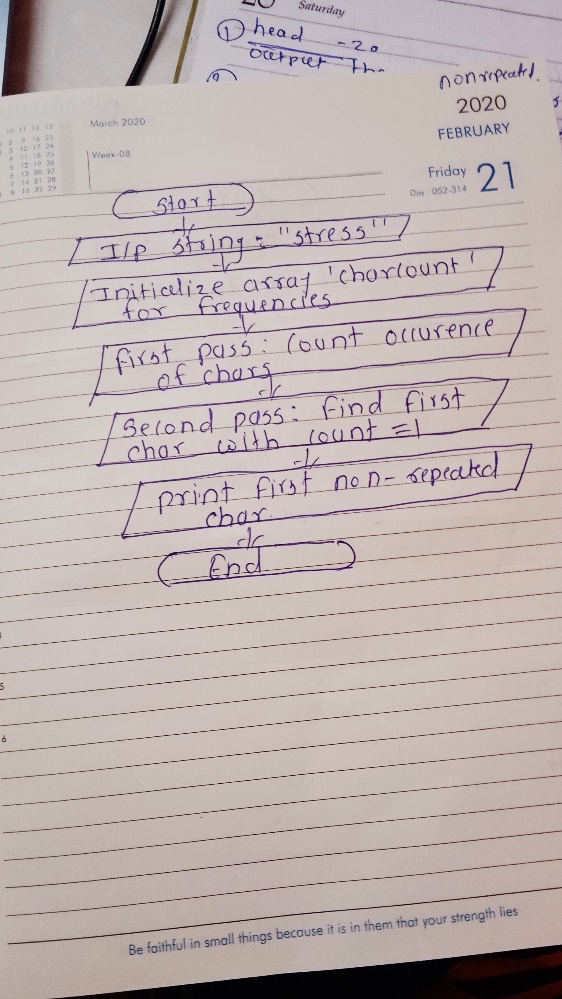
**C:\Users\rushi\ADS>java Factorial**

**Enter a number to compute its factorial: 0**

**Factorial of 0 is: 1**

**Explanation:**

1. **Input: The program takes an integer number from the user.**
2. **Factorial Calculation: It uses a method calculateFactorial(int num) that computes the factorial of num using a loop. The loop multiplies each number from 1 to num to get the factorial.**
3. **Output: After calculating the factorial, the result is printed.**
4. ** Time Complexity: O(n)**
5. ** Space Complexity: O(1}**

****

**4. Fibonacci Series**

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

**Program code:**

**import java.util.Scanner;**

**public class FibonacciSeries {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the number of terms for the Fibonacci series: ");**

**int n = scanner.nextInt();**

**scanner.close();**

**System.out.print("Fibonacci Series: ");**

**printFibonacci(n);**

**}**

**public static void printFibonacci(int n) {**

**if (n <= 0) {**

**System.out.println("Input should be a positive integer.");**

**return;**

**}**

**int firstTerm = 0, secondTerm = 1;**

**if (n == 1) {**

**System.out.println("[" + firstTerm + "]");**

**return;**

**}**

**System.out.print("[" + firstTerm + ", " + secondTerm);**

**for (int i = 3; i <= n; i++) {**

**int nextTerm = firstTerm + secondTerm;**

**System.out.print(", " + nextTerm);**

**firstTerm = secondTerm;**

**secondTerm = nextTerm;**

**}**

**System.out.println("]");**

**}**

**}**

**Output :**

**C:\Users\rushi\ADS>javac FibonacciSeries.java**

**C:\Users\rushi\ADS>java FibonacciSeries**

**Enter the number of terms for the Fibonacci series: 5**

**Fibonacci Series: [0, 1, 1, 2, 3]**

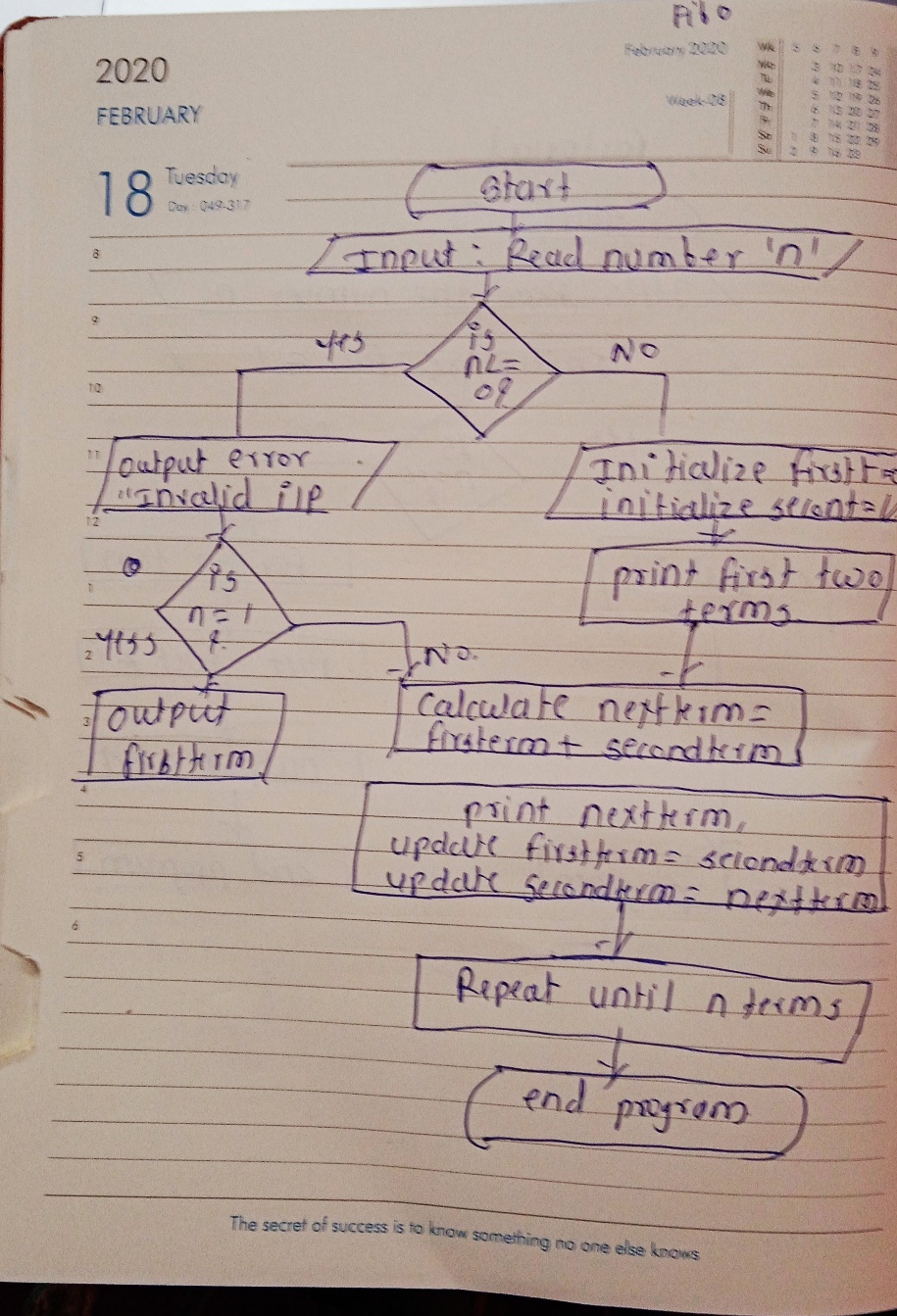
**C:\Users\rushi\ADS>java FibonacciSeries**

**Enter the number of terms for the Fibonacci series: 8**

**Fibonacci Series: [0, 1, 1, 2, 3, 5, 8, 13]**

**Explanation:**

1. **Input: The program prompts the user to enter the number n (how many terms they want in the Fibonacci series).**
2. **Fibonacci Series Calculation:**
   * **If n == 1, the program simply outputs [0].**
   * **If n > 1, it initializes the first two terms (0 and 1) and prints them.**
   * **It then computes each subsequent term by adding the previous two terms until it reaches the desired n terms.**
3. **Output: The program prints the Fibonacci sequence.**
4. ** Time Complexity: O(n)**
5. ** Space Complexity: O(1)**

****

**5. Find GCD**

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

**Program code:**

**import java.util.Scanner;**

**public class GCD {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the first number (a): ");**

**int a = scanner.nextInt();**

**System.out.print("Enter the second number (b): ");**

**int b = scanner.nextInt();**

**scanner.close();**

**int gcd = findGCD(a, b);**

**System.out.println("GCD of " + a + " and " + b + " is: " + gcd);**

**}**

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

**if (b == 0) {**

**return a;**

**}**

**return findGCD(b, a % b);**

**}**

**}**

**Output:**

**C:\Users\rushi\ADS>javac GCD.java**

**C:\Users\rushi\ADS>java GCD**

**Enter the first number (a): 54**

**Enter the second number (b): 24**

**GCD of 54 and 24 is: 6**

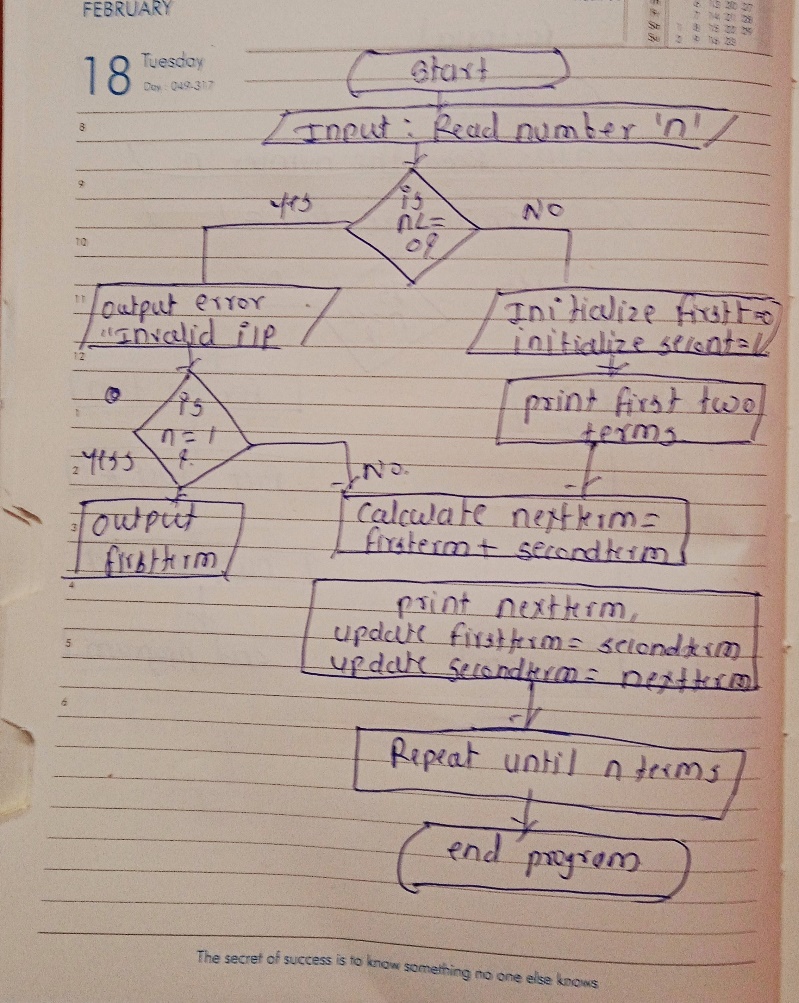
**C:\Users\rushi\ADS>java GCD**

**Enter the first number (a): 17**

**Enter the second number (b): 13**

**GCD of 17 and 13 is: 1**

**Explanation:**

1. **Input: The program prompts the user to enter two integers, a and b.**
2. **GCD Calculation:**
   * **The program uses the Euclidean algorithm to compute the GCD:**
     + **If b == 0, then a is the GCD.**
     + **Otherwise, recursively compute findGCD(b, a % b) until the remainder becomes 0.**
3. **Output: The program prints the GCD of the two numbers.**
4. ** Time Complexity: O(log⁡(min⁡(a,b))**
5. 
6. ** Space Complexity: O(log⁡(min⁡(a,b)))**

**6. Find Square Root**

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

**Program code:**

**import java.util.Scanner;**

**public class SimpleSquareRoot {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter a number to find its square root: ");**

**int x = scanner.nextInt();**

**scanner.close();**

**int sqrt = findSquareRoot(x);**

**System.out.println("The integer square root of " + x + " is: " + sqrt);**

**}**

**public static int findSquareRoot(int x) {**

**int i = 1;**

**while (i \* i <= x) {**

**i++;**

**}**

**return i - 1;**

**}**

**}**

**Output code:**

**C:\Users\rushi\ADS>javac SimpleSquareRoot.java**

**C:\Users\rushi\ADS>java SimpleSquareRoot**

**Enter a number to find its square root: 16**

**The integer square root of 16 is: 4**

**C:\Users\rushi\ADS>java SimpleSquareRoot**

**Enter a number to find its square root: 27**

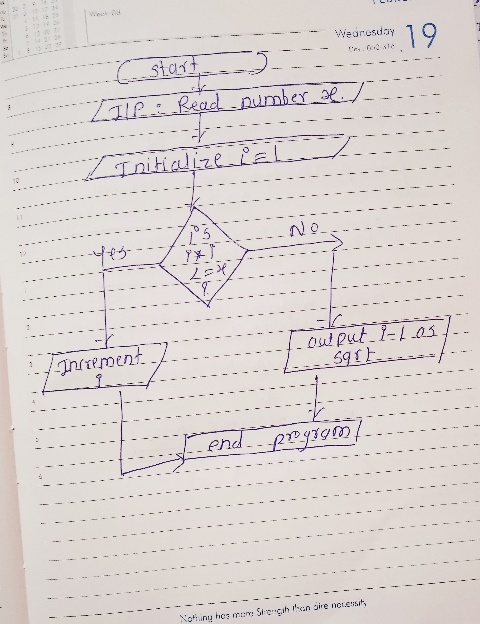
**The integer square root of 27 is: 5**

**Explanation:**

1. **Input: The program prompts the user to input an integer x.**
2. **Square Root Calculation:**
   * **The program starts with i = 1 and increments i until i \* i exceeds x.**
   * **The largest i such that i \* i <= x is the integer approximation of the square root.**
3. **Output: The program prints the largest integer square root of x.**

** Time Complexity: he program checks each integer from 1 to √x. So, the time complexity is proportional to √x.**

** Space Complexity: O(1)**

****

**7. Find Repeated Characters in a String**

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

**Program code:**

**public class RepeatedCharacters{**

**public static void main(String[] args) {**

**String input = "programming";**

**findRepeatedCharacters(input);**

**}**

**public static void findRepeatedCharacters(String input) {**

**boolean[] visited = new boolean[256];**

**char[] repeated = new char[input.length()];**

**int index = 0;**

**for (int i = 0; i < input.length(); i++) {**

**char currentChar = input.charAt(i);**

**for (int j = i + 1; j < input.length(); j++) {**

**if (currentChar == input.charAt(j) && !visited[currentChar]) {**

**repeated[index++] = currentChar;**

**visited[currentChar] = true;**

**break;**

**}**

**}**

**}**

**System.out.print("Repeated characters: ");**

**for (int i = 0; i < index; i++) {**

**System.out.print(repeated[i] + " ");**

**}**

**}**

**}**

**Output:**

**C:\Users\rushi\ADS>**

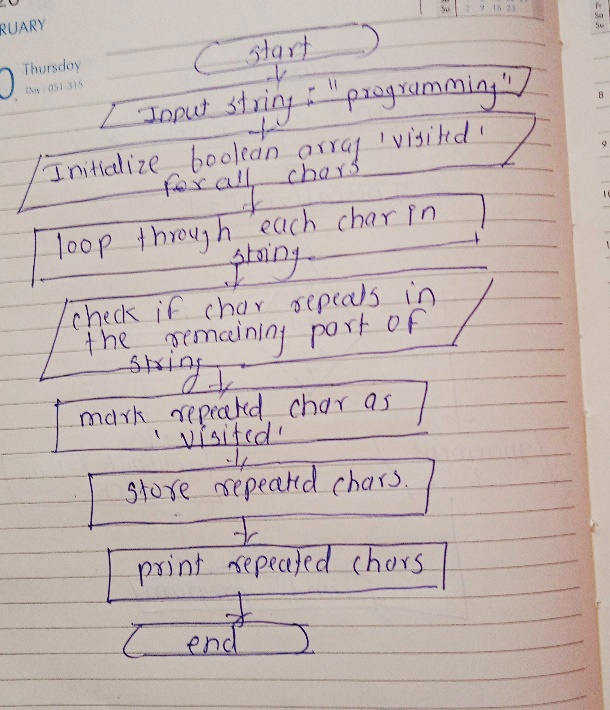
**C:\Users\rushi\ADS>javac RepeatedCharacters.java**

**C:\Users\rushi\ADS>java RepeatedCharacters**

**Repeated characters: r g m**

**Explanation:**

1. **Input: The program accepts a string as input, such as "programming".**
2. **Boolean Array: A boolean array visited[256] is used to track if a character has already been counted as repeated.**
3. **Character Comparison: The program iterates over each character in the string and compares it with every other character that comes after it. If a match is found and the character is not yet marked as "visited," it's added to the array repeated.**
4. **Output: After checking all characters, the repeated characters are printed.**

****

**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**

**Program code:**

**public class FirstNonRepeatedCharacter {**

**public static void main(String[] args) {**

**String input = "stress";**

**Character result = findFirstNonRepeatedCharacter(input);**

**if (result != null) {**

**System.out.println("First non-repeated character: " + result);**

**} else {**

**System.out.println("No non-repeated character found.");**

**}**

**}**

**public static Character findFirstNonRepeatedCharacter(String input) {**

**int[] charCount = new int[256];**

**for (int i = 0; i < input.length(); i++) {**

**charCount[input.charAt(i)]++;**

**}**

**for (int i = 0; i < input.length(); i++) {**

**if (charCount[input.charAt(i)] == 1) {**

**return input.charAt(i);**

**}**

**}**

**return null;**

**}**

**}**

**Output :**

**C:\Users\rushi\ADS>javac FirstNonRepeatedCharacter.java**

**C:\Users\rushi\ADS>java FirstNonRepeatedCharacter**

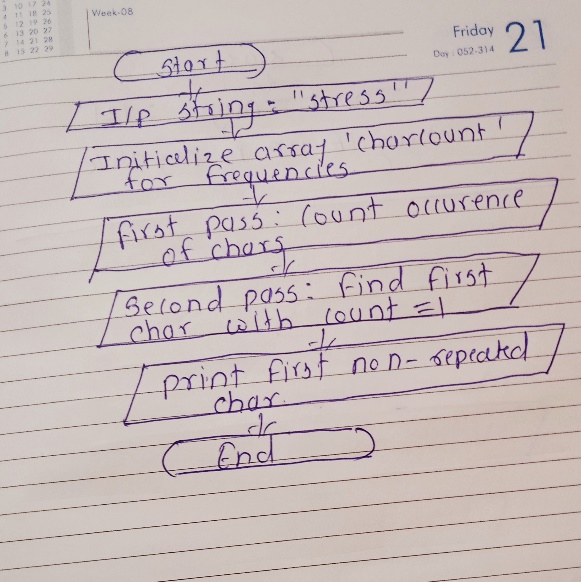
**First non-repeated character: t**

**Explanation:**

1. **Input: The program takes a string as input, such as "stress".**
2. **First Pass (Counting): An array charCount[256] is used to count the occurrences of each character in the string. This array has a fixed size of 256 to account for all ASCII characters.**
3. **Second Pass (Checking): The program iterates through the string again, checking the charCount array for each character to see if it appears only once.**
4. **Output: The first character with a count of 1 is printed as the result. If no such character is found, the program returns null.**

** Time Complexity:O(n} Space Complexity:**

**The space complexity is O(1)**

****

**9. Integer Palindrome**

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

**Test Cases:**

**Input: 121**

**Output: true**

**Program code:**

**public class IntegerPalindrome {**

**public static void main(String[] args) {**

**int input = 121; // You can change this to test other inputs.**

**boolean result = isPalindrome(input);**

**System.out.println("Is the number " + input + " a palindrome? " + result);**

**}**

**public static boolean isPalindrome(int number) {**

**if (number < 0) {**

**return false;**

**}**

**int originalNumber = number;**

**int reversedNumber = 0;**

**while (number != 0) {**

**int lastDigit = number % 10;**

**reversedNumber = reversedNumber \* 10 + lastDigit;**

**number = number / 10;**

**}**

**return originalNumber == reversedNumber;**

**}**

**}**

**Output :**

**C:\Users\rushi\ADS>javac IntegerPalindrome.java**

**C:\Users\rushi\ADS>java IntegerPalindrome**

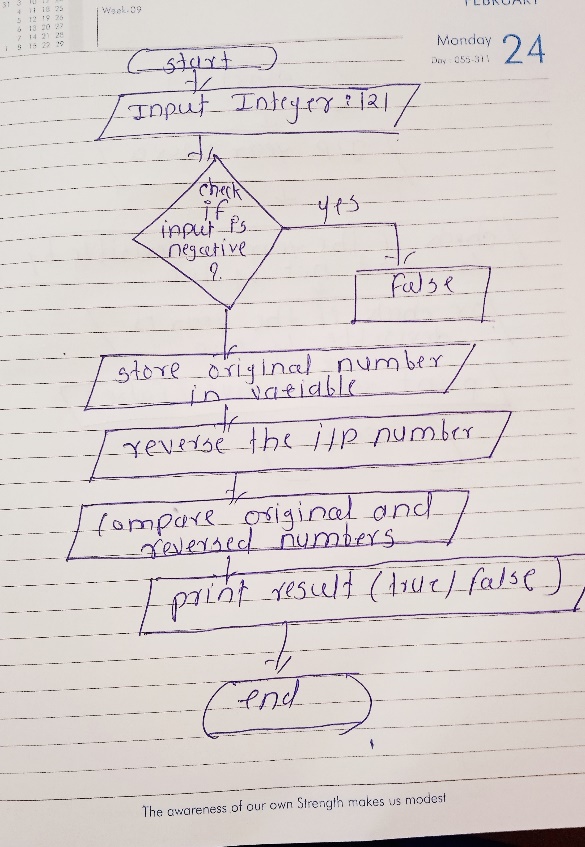
**Is the number 121 a palindrome? True**

**Explanation:**

1. **Input: The program accepts an integer as input, such as 121.**
2. **Negative Check: If the integer is negative, it is immediately determined as not a palindrome and the program returns false.**
3. **Reverse the Number: The program reverses the digits of the integer by repeatedly extracting the last digit (using % 10) and building the reversed number.**
4. **Compare: After reversing the number, the program compares the original number with the reversed one.**
   * **If they are the same, the integer is a palindrome, and the program returns true.**
   * **If not, it returns false.**
5. **Output: The result is printed to indicate whether the integer is a palindrome or not.**

**Time Complexity:O(d)**

**Space Complexity:O(1)**

****

**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**

**Program code:**

**public class LeapYearChecker {**

**public static void main(String[] args) {**

**int year = 2020;**

**boolean result = isLeapYear(year);**

**System.out.println("Is the year " + year + " a leap year? " + result);**

**}**

**public static boolean isLeapYear(int year) {**

**return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);**

**}**

**}**

**Output code:**

**C:\Users\rushi\ADS>javac LeapYearChecker.java**

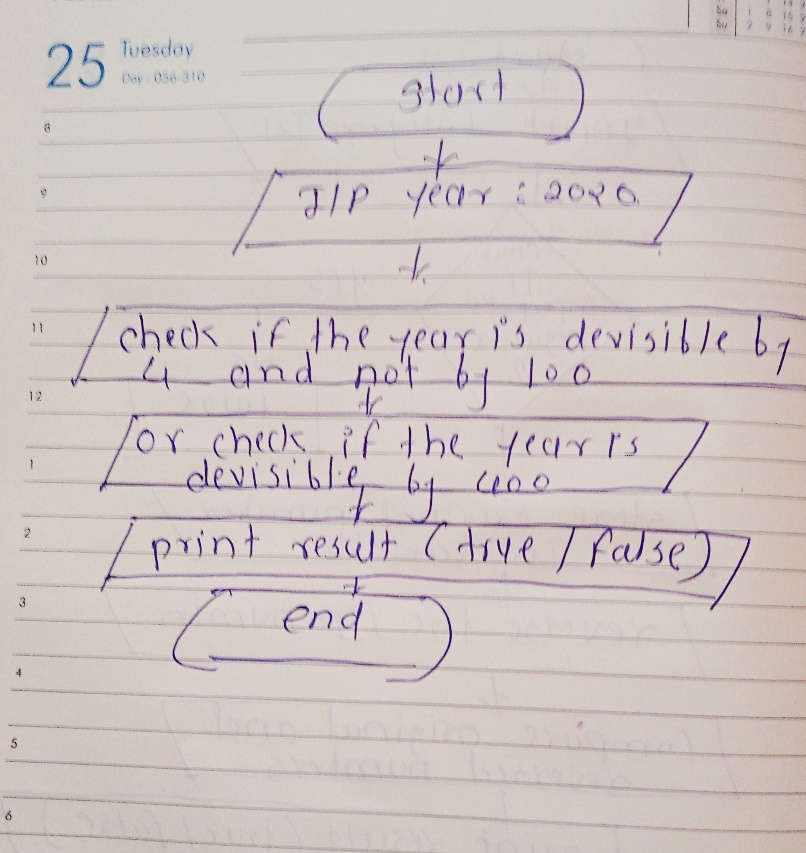
**C:\Users\rushi\ADS>java LeapYearChecker**

**Is the year 2020 a leap year? True**

**Explanation:**

1. **Input: The program takes an integer input representing a year, such as 2020.**
2. **Leap Year Check: The program uses the following logic to determine if a year is a leap year:**
   * **A year is a leap year if:**
     + **It is divisible by 4 and not divisible by 100, or**
     + **It is divisible by 400.**
3. **Output: The result is printed to indicate whether the given year is a leap year.**

**Time and Space Complexity:O(1)**

****