**C-DAC Mumbai Date 25/09/2024**

**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

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

Program code:

import java.util.Scanner;

public class Armstrong {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a Number:");

int num = sc.nextInt();

int n1 = num;

int result = 0, rem, n = 0;

while (num != 0) {

num /= 10;

n++;

}

num = n1;

while (num != 0) {

rem = num % 10;

result += Math.pow(rem, n);

num /= 10;

}

if (n1 == result) {

System.out.println(n1 + " is an Armstrong number.");

} else {

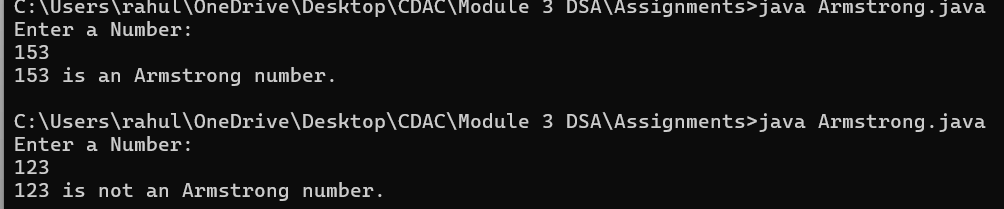
System.out.println(n1 + " is not an Armstrong number.");

}

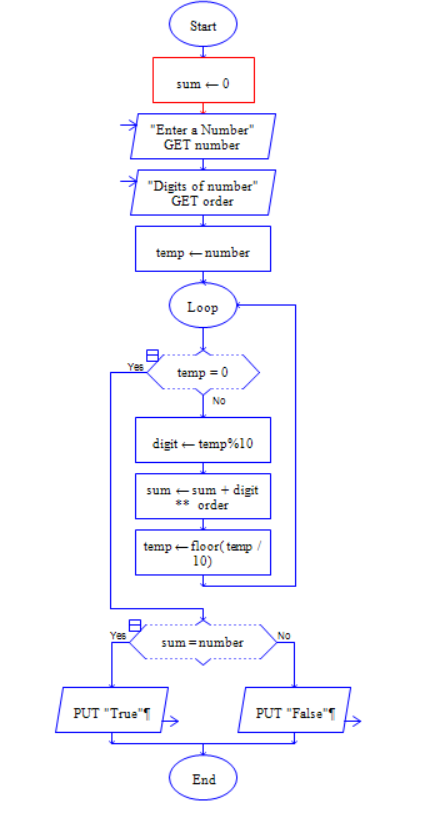
}

}

Output:



Flow Chart:



**Code Explanation:**

* The program prompts use to enter a number and stores it in num.
* It first counts the number of digits in the number (n) by repeatedly dividing num by 10.
* Then, it calculates the sum of each digit raised to the power of n using Math.pow().
* Finally, we checks if the sum is equal to the original number (n1). If true, it declares it as an Armstrong number; otherwise, it isn't.

**Time complexity: Olog(n)**

**Space complexity:** O(1)

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

Program code:

import java.util.Scanner;

public class PrimeNumber{

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

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

int number = sc.nextInt();

boolean isPrime = true;

if(number <= 1){

isPrime = false;

}

else{

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

if(number % i == 0){

isPrime = false;

break;

}

}

}

if(isPrime){

System.out.println( number +" is a prime number");

}

else{

System.out.println(number + " is not a prime number");

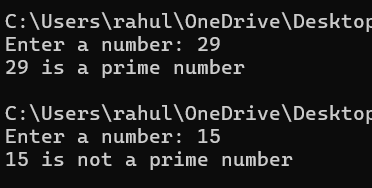
}

sc.close();

}

}

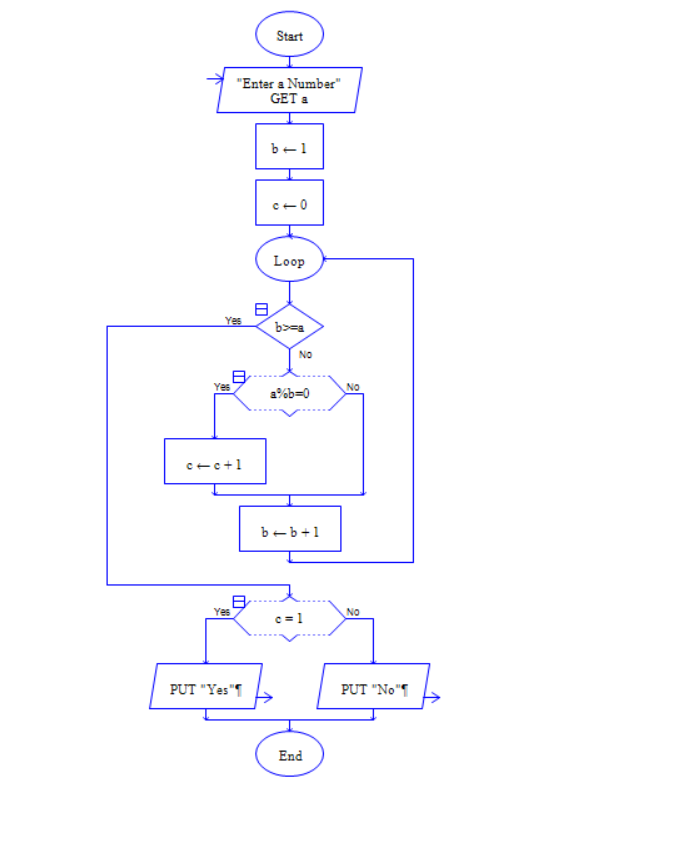
Output:



**Code Explanation:**

* we write a program for to check a given number is prime or not
* The function takes an integer n as input.
* It checks if the number is less than or equal to 1, marking it as not prime.
* If the number is greater than 1, it loops from 2 to number-1, checking if number is divisible by any value in this range.
* If any divisor is found, the number is marked as not prime; otherwise, it is prime.

Flow chart:



**Time Complexity:**

* **Worst case:** O(N), where N is the input number. The for loop runs up to N-2 times, checking for divisors.
* **Best case:** O(1),

**Space complexity:** O(1)

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

Program:

class Factorial{

static int fact(int n) {

if (n <= 1) {

return 1;

} else {

return n \* fact(n - 1);

}

}

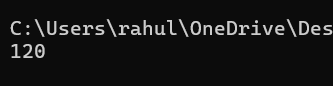
public static void main(String args[]) {

System.out.println(fact(5));

}

}

Output:



Flow chart:

**Code Explanation:**

* We write a program for factorials using recursion method
* The fact() method is a recursive function that calculates the factorial of a given number n.
* The base case checks if n is less than or equal to 1, returning 1 as factorial of 0 and 1 is 1.
* Otherwise, the method recursively calls itself with n-1, multiplying n with the factorial of n-1 until it reaches the base case.
* In the main() method, the program calls fact(5) and prints the result (5! = 120).

**Time complexity:** O(n)

**Space complexity:** O(n)

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]

Programing code:

import java.util.Scanner;

public class FibonacciSeries{

public static void main(String[] arge){

Scanner sc = new Scanner(System.in);

System.out.println("Enter number for fibseq: ");

int n = sc.nextInt();

int a = 0;

int b = 1;

System.out.println(" fibonacci series: ");

for(int i = 1; i<=n;i++){

System.out.println(a);

int c = a+b;

a = b;

b = c;

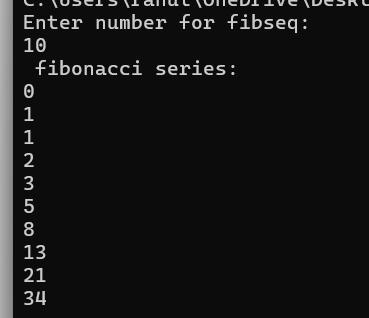
}

sc.close();

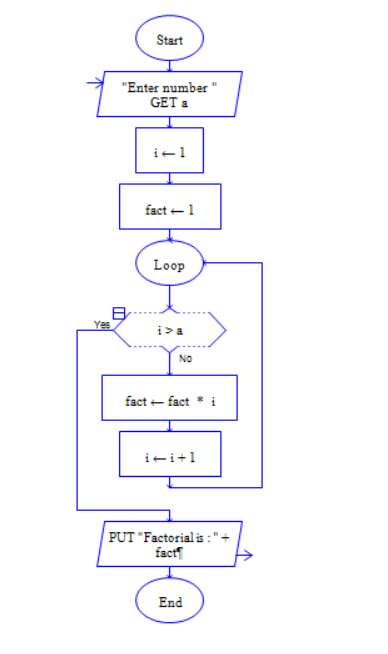
}

}

Output:



Flowchart:



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

Program code:

import java.util.Scanner;

public class GCD {

// GCD using subtraction

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

while (a != b) {

if (a > b) {

a = a - b;

} else {

b = b - a;

}

}

return a;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int a = sc.nextInt();

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

int b = sc.nextInt();

int gcd = findGCD(a, b);

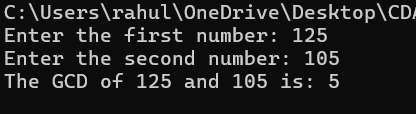
System.out.println("The GCD of " + a + " and " + b + " is: " + gcd);

sc.close();

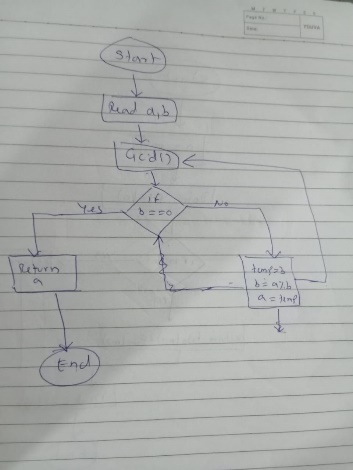
}

}

Output:



Flow chart:



**Explaination:**

1: Initialize Numbers

2: Check Base Case

3: Calculate Remainder

4: Recursive Replacement

5: return GCD

**Time Complexity: O(log(n))**

**Space Complexity: O(log(n))**

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

Program code:

import java.util.\*;

public class SquareRoot {

public static void main (String args[]) {

Scanner sc = new Scanner (System.in);

System.out.println("Enter the number whose square root you want: ");

double x = sc.nextDouble();

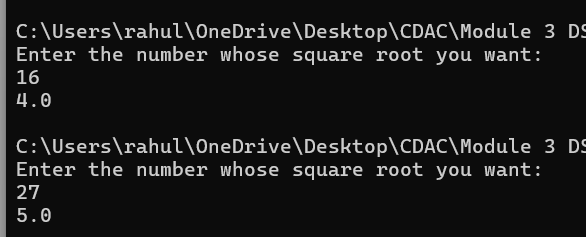
double ans = (int) Math.sqrt(x); //narrowing conversion - data type

System.out.println(ans);

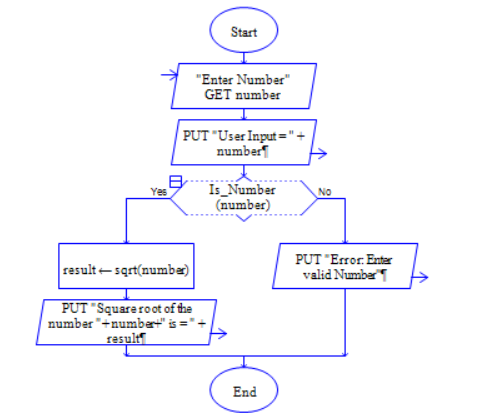
}

}

Output:



Flow chart:



**Code Explanation:**

1: Input Number

2: Calculate Square Root

3: Data Type Conversion (Narrowing)

4: Store Result

5: Display Result

**Time Complexity: O(1)**

**Space Complexity: O(1)**

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']

Program code:

import java.util.Scanner;

public class RepeatedCharacters{

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string: ");

String input = scanner.nextLine();

int[] charCount = new int[26];

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

charCount[input.charAt(i) - 'a']++;

}

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

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

if (charCount[i] > 1) {

System.out.print((char) ('a' + i) + ", ");

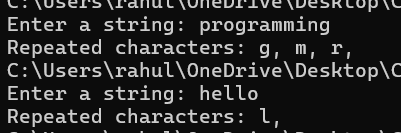
}

}

}

}

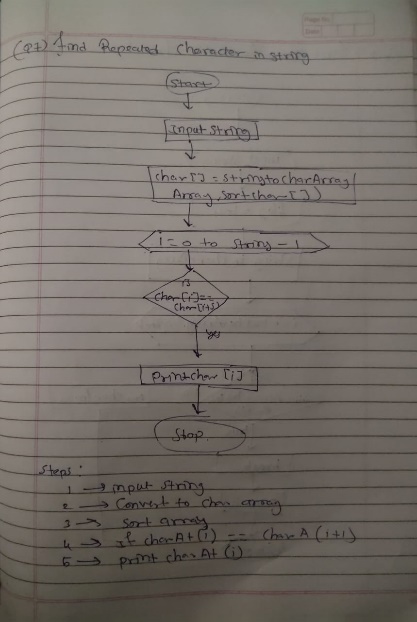
Output:



**Time Complexity: O(n)**

**Space Complexity: O(1)**

Flow chart:



**Code Explanation:**

1. First we take string as input.
2. It initializes an integer array charCount[26] to store the count of each character from 'a' to 'z'.
3. The for loop iterates over the string, converting each character to its corresponding index in the charCount array and incrementing the count for that character.
4. It then checks which characters have been repeated (count > 1) and prints them.

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:

import java.util.Scanner;

public class NonRepeatedCharacter{

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

System.out.print("Enter String: ");

String str = sc.nextLine();

char[] arr = str.toCharArray();

for(int i=0; i<arr.length; i++)

{

for(int j=i+1; j<arr.length; j++)

{

if(arr[i] != arr[j])

{

System.out.println(arr[j]);

System.exit(0);

}

else

{

System.out.println("null");

System.exit(0);

}

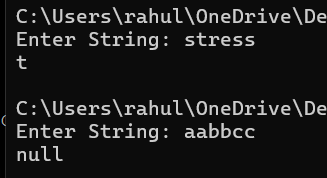
}

}

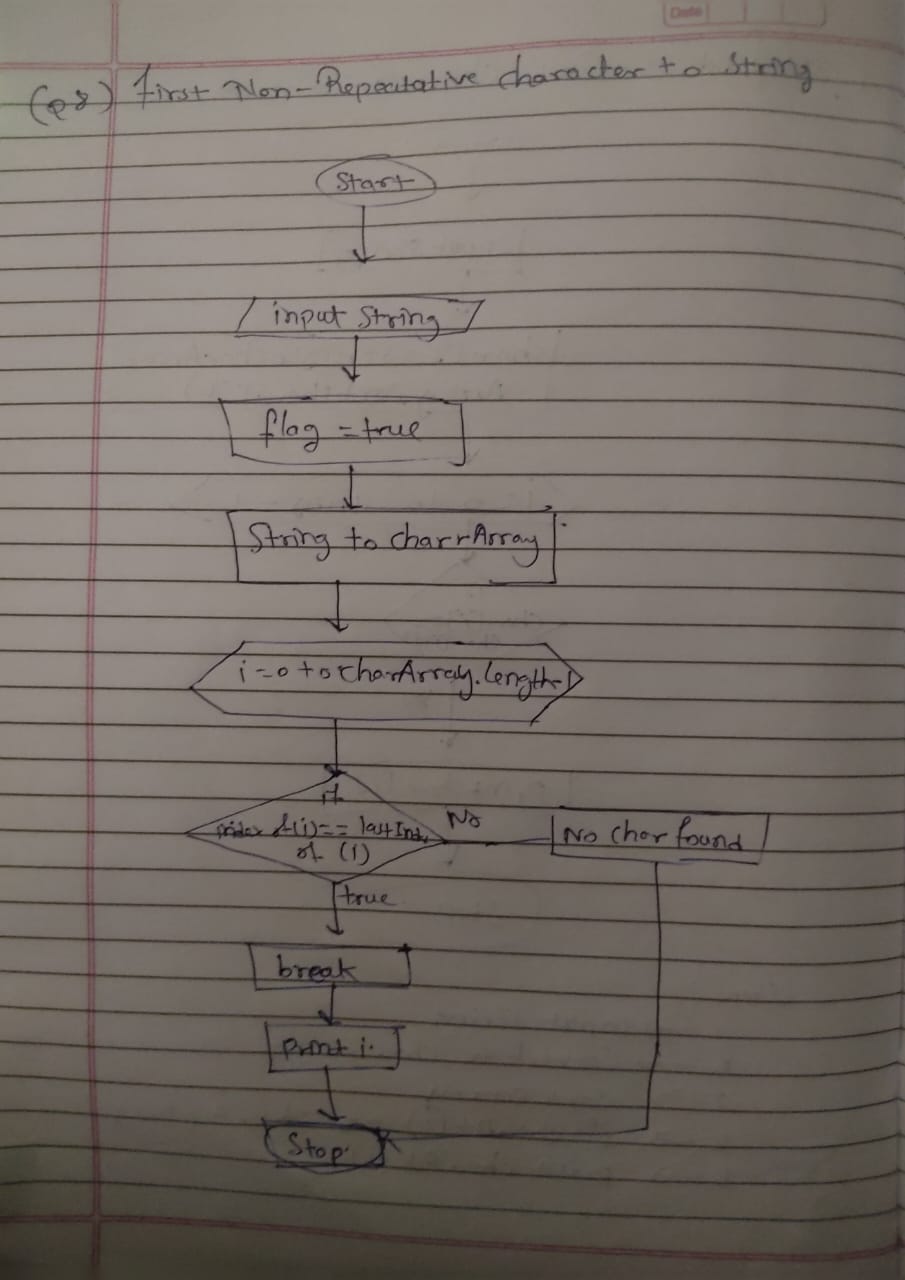
}

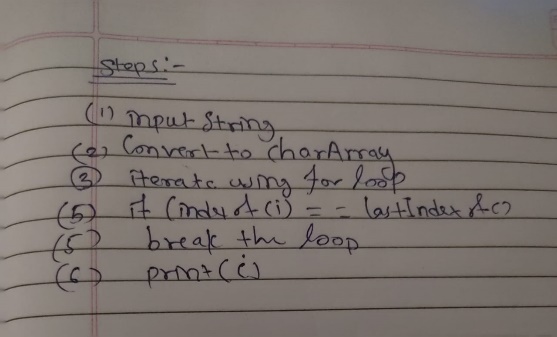
}

Output:



Flow chart:





**Code Explanation:**

1. First we set string and converts the string to a character array (arr).
2. It uses two nested loops to compare each character with the subsequent ones in the array.
3. If the character at index i is not equal to the character at index j (first non-repeated), it prints the character and exits.
4. If the characters are equal (repeated), it prints "null" and exits the program immediately.

**Time complexity:** O(n²)

**Space complexity:** 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

Input: -121

Output: false

Program code:

import java.util.Scanner;

public class IntegerPalindrome {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a word: ");

String input = sc.nextLine();

boolean isPalindrome = true;

int left = 0;

int right = input.length() - 1;

while (left < right) {

if (input.charAt(left) != input.charAt(right)) {

isPalindrome = false;

break;

}

left++;

right--;

}

if (isPalindrome) {

System.out.println(input + " is a palindrome.");

} else {

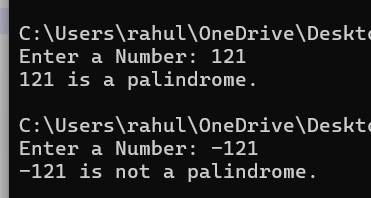
System.out.println(input + " is not a palindrome.");

}

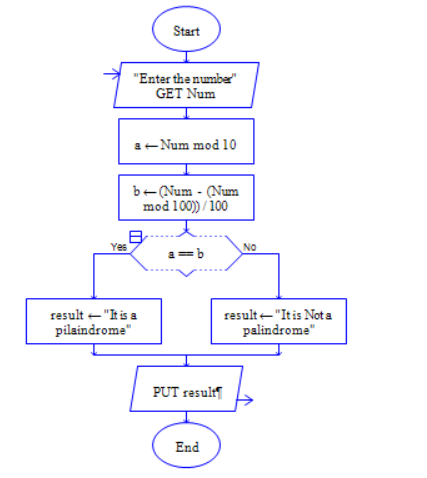
}

}

Output:



Flow chart:



**Algorithm:**

1: Initialize Variables

2: Compare Characters from Both Ends

3. Increment left and decrement right

4. Repeat steps 2-3 until left pointer meets or crosses right

5.print result.

**Time Complexity**: O(n)

**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:

import java.util.\*;

class LeapYear{

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the year you want to choose: ");

int year = sc.nextInt();

if (year % 4 == 0 && year %100 != 0 || year % 400 == 0)

{

System.out.println(true);

}

else

{

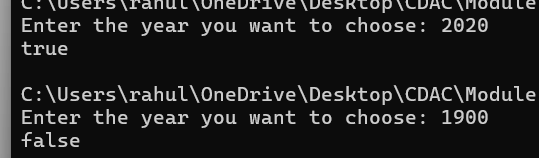
System.out.println(false);

}

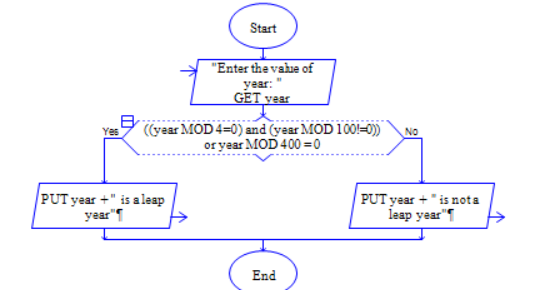
}

}

Output:



Flow chart:



**Explaination:**

1: Get User Input

2: Check Divisibility by 4

3: If year is divisible by 100, it must also be divisible by 400 to be a leap year

4:If year is divisible by 100 but not 400, it's not a leap year (goto Step 6)

5:If year passes Step 2 and Step 3 checks, it's a leap year

6:Display leap year or not

**Time Complexity**: O(n)

**Space Complexity:** O(1)