

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

```
/* Armstrong number*/
import java.util.Scanner;

class ArmstrongNumber{
    public static void main(String args[]){

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter a number to check if it is armstrong number");
        int num = sc.nextInt();
        int originalnum = num;

        double sum =0;

        while(num>0){
            int rem = num % 10;
            sum = sum + Math.pow(rem, 3);
            num = num / 10;
        }
        if(sum == originalnum)
            System.out.println(true);
        else
            System.out.println(false);
    }
}
```

```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>javac ArmstrongNumber.java
```

```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java ArmstrongNumber
```

```
Enter a number to check if it is armstrong number
```

```
153
```

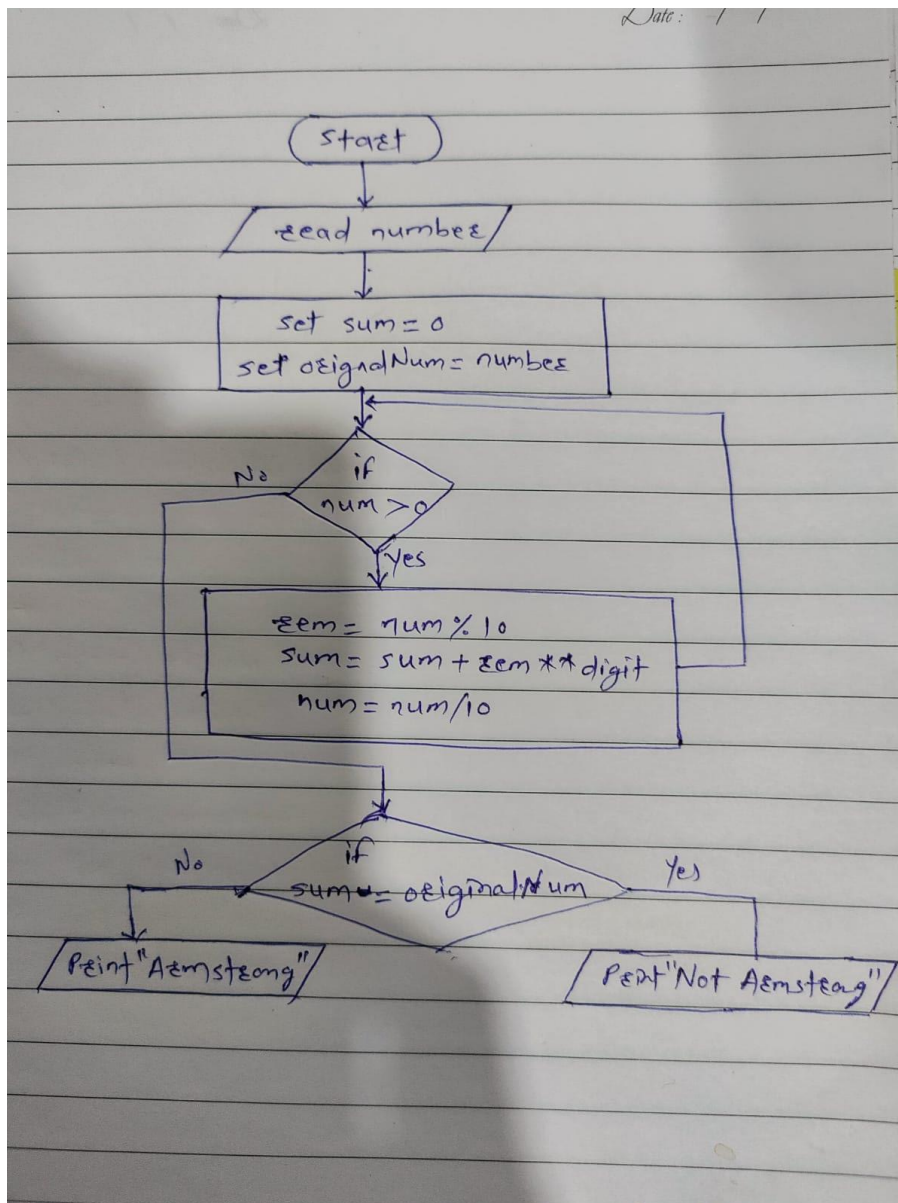
```
true
```

```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java ArmstrongNumber
```

```
Enter a number to check if it is armstrong number
```

```
123
```

```
false
```



Explanation :

The given program is to check if the entered number is Armstrong number or not. Armstrong number is the number in which the sum of each digit with power of the total number of digits is the same as the given number.

Eg.- 153 , the total number of digits in the given number are 3. So we will take the addition of each digit with power 3 as

$$1^3 + 5^3 + 3^3 = 153$$

While iterating the each digit from number using modulus num % 10

Then raise the digit with the number of digits and adds the result to the sum

Finally check if the original number matches the resulted sum. If it does it is the Armstrong number.

Time Complexity :

The time complexity of this program is $O(\log n)$

The space complexity of this program is $O(1)$.

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

```
/* Prime Number*/
```

```
import java.util.Scanner;
```

```
class PrimeNumber{
```

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

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

```
        System.out.println("Enter a number to check if it is armstrong number");
```

```
        int num = sc.nextInt();
```

```
        boolean isPrime = true;
```

```
        for(int i=2; i<=num/2; i++){
```

```
            if(num % i ==0){
```

```
                isPrime = false; // if divisible by any number it is not a prime number
                break;
```

```
            }
```

```
        }
```

```
        if(isPrime)
```

```
            System.out.println(true);
```

```
        else
```

```
            System.out.println(false);
```

```
    }
```

```
}
```

E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>javac PrimeNumber.java

E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java PrimeNumber

Enter a number to check if it is armstrong number

29

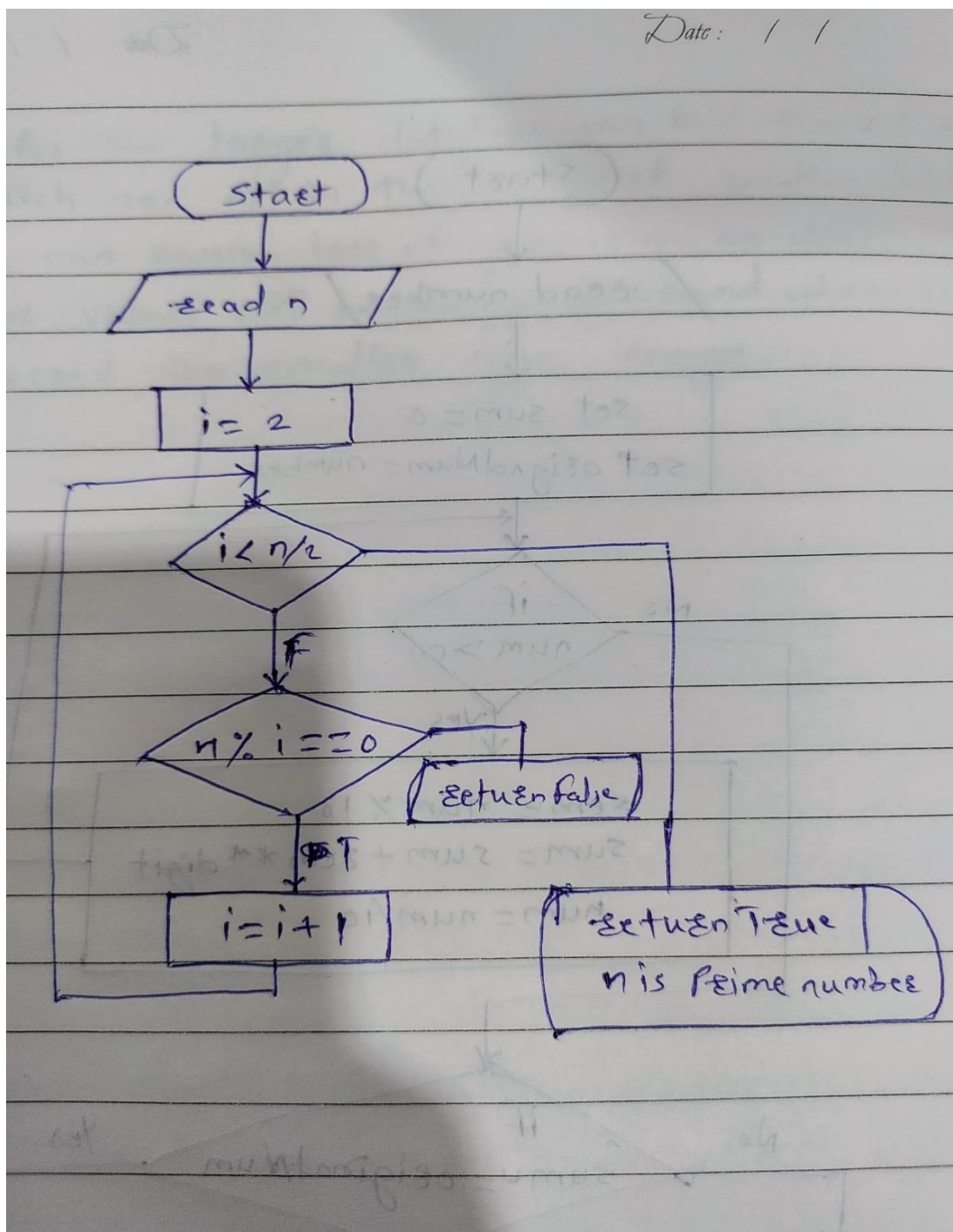
true

E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java PrimeNumber

Enter a number to check if it is armstrong number

15

false



Explanation :

Prime number is the number which has no divisor other than 1 and itself.

Check if the number is divisible by any of the number from 2 to num/2. So if the number get divisible by any of these in iteration, it is not the prime number and it will result false

If no divisor found the number is the Prime number and it will return true.

Time & Space Complexity :

The time complexity for this program is $O(n)$

The space complexity of this program is $O(1)$

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

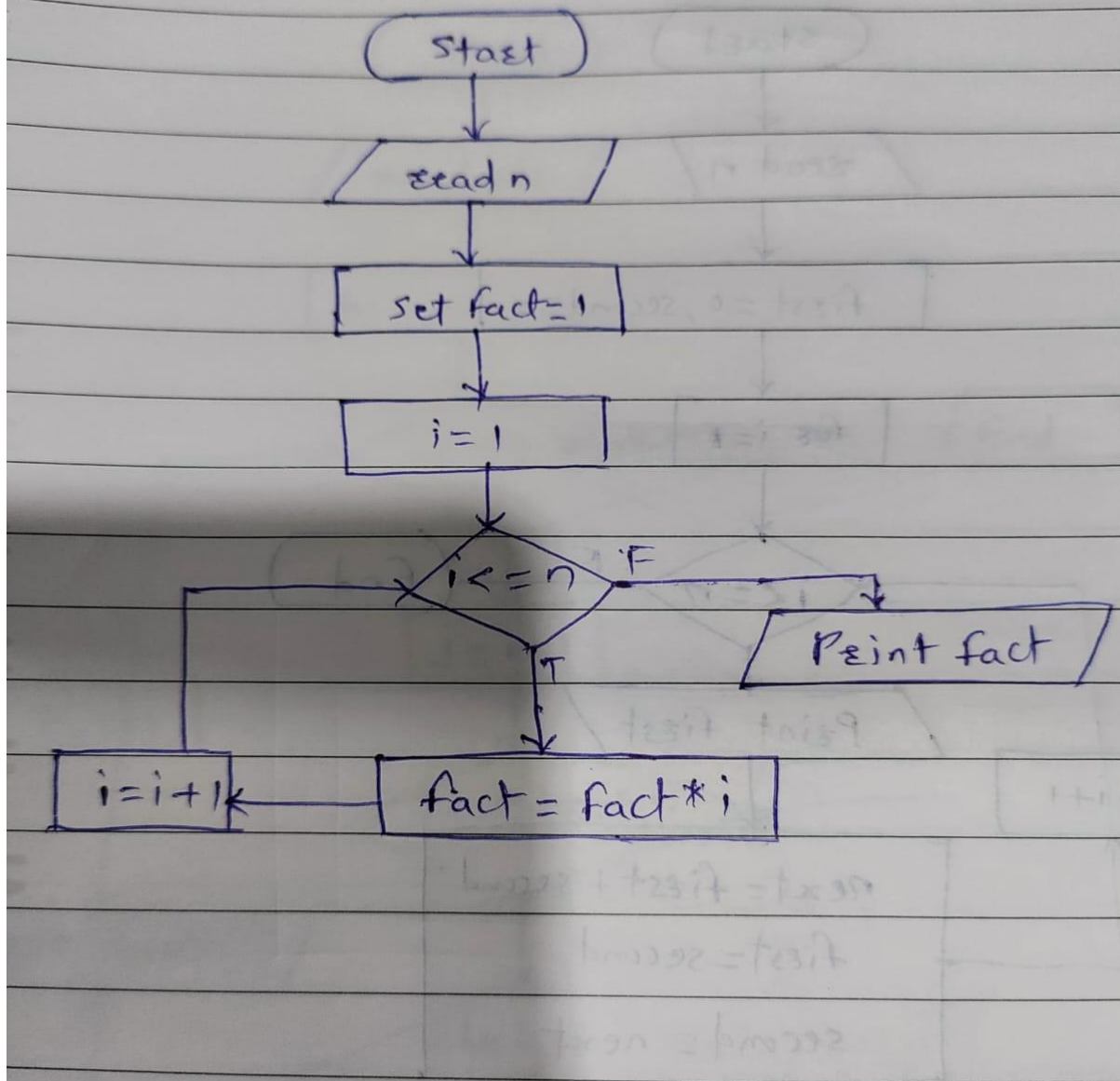
```
/* Factorial*/
import java.util.Scanner;

class FactorialOfNumber {
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        int num = sc.nextInt();
        int fact = 1;

        for(int i=1; i<=num; i++){
            fact = fact * i;
        }
        System.out.println(fact);
    }
}
```

```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>javac FactorialOfNumber.java
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java FactorialOfNumber
5
120
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java FactorialOfNumber
0
1
```

Date: / /



Explanation :

Factorial of number is the product of all the integers from 1 to the given number.
We need to initialize a factorial to 1, then multiply it by each number from 1 to num.
It can be solved with recursion.

Time & Space Complexity :

The time complexity for this program is $O(n)$
The space complexity of this program is $O(1)$

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]

```
/* Fibonacci Series */
import java.util.Scanner;

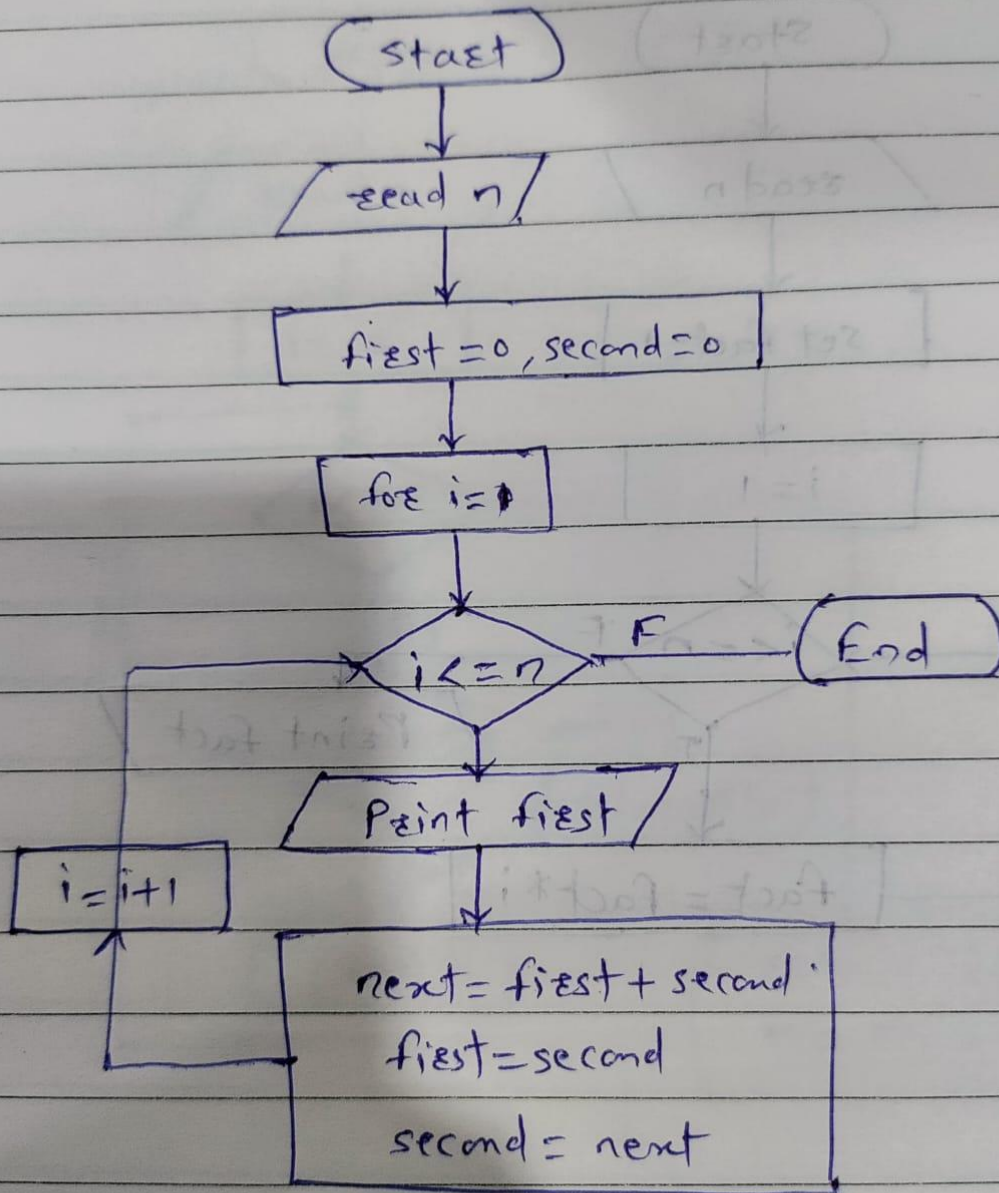
class FibonacciSeries{

    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        int first = 0;
        int second = 1;
        for(int i=1; i<=n;i++){
            System.out.print(first + ", ");
            int next = first + second;
            first = second;
            second = next;
        }
    }
}
```

```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>javac FibonacciSeries.java
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java FibonacciSeries
5
0, 1, 1, 2, 3,
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java FibonacciSeries
8
0, 1, 1, 2, 3, 5, 8, 13,
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>
```


Date : / /



Explanation :

Fibonacci series is a sequence where each number is the sum of the two preceding values. It starts from 0 and 1.

The loop starts with the first and second value as 0 and 1 respectively. For the each iteration the next value is calculated as the sum of two number.

Time & Space Complexity :

The time complexity for this program is $O(n)$

The space complexity of this program is $O(1)$

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

`/* GCD */`

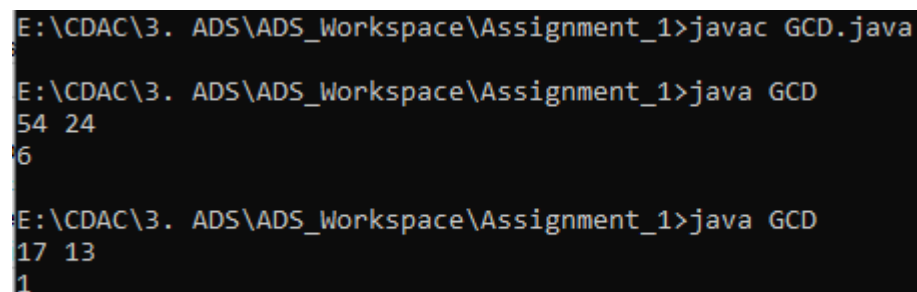
```
import java.util.Scanner;
class GCD{

    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);

        int a = sc.nextInt();
        int b = sc.nextInt();

        int gcd = 1;

        for(int i=1; i<a && i<=b; i++){
            if(a%i==0 && b%i==0)
                gcd = i;
        }
        System.out.println(gcd);
    }
}
```



```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>javac GCD.java
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java GCD
54 24
6
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java GCD
17 13
1
```

Explanation :

GCD is to find the Greatest Common Divisor

We initialize the gcd to 1. After that in each iteration we check whether a and b values are divisible by i. It becomes the current gcd value. The final value of gcd after the loop will be the greatest common divisor of a and b.

Time & Space Complexity :

The space complexity of this program is $O(1)$

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

```
/* Square root */
```

```
import java.util.Scanner;
class SquareRoot{

    public static void main(String args[]){

        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        int squareroot = n/2;
        int t;
        do{
            t = squareroot;
            squareroot = (t+(n/t)) /2;
        }while((t-squareroot) !=0);
        System.out.println(squareroot);

    }

}
```

```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>javac SquareRoot.java

E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java SquareRoot
16
4

E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java SquareRoot
27
5
```

Time & Space Complexity :

The time complexity of this program is $O(\log n)$

The space complexity of this program is $O(1)$

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']

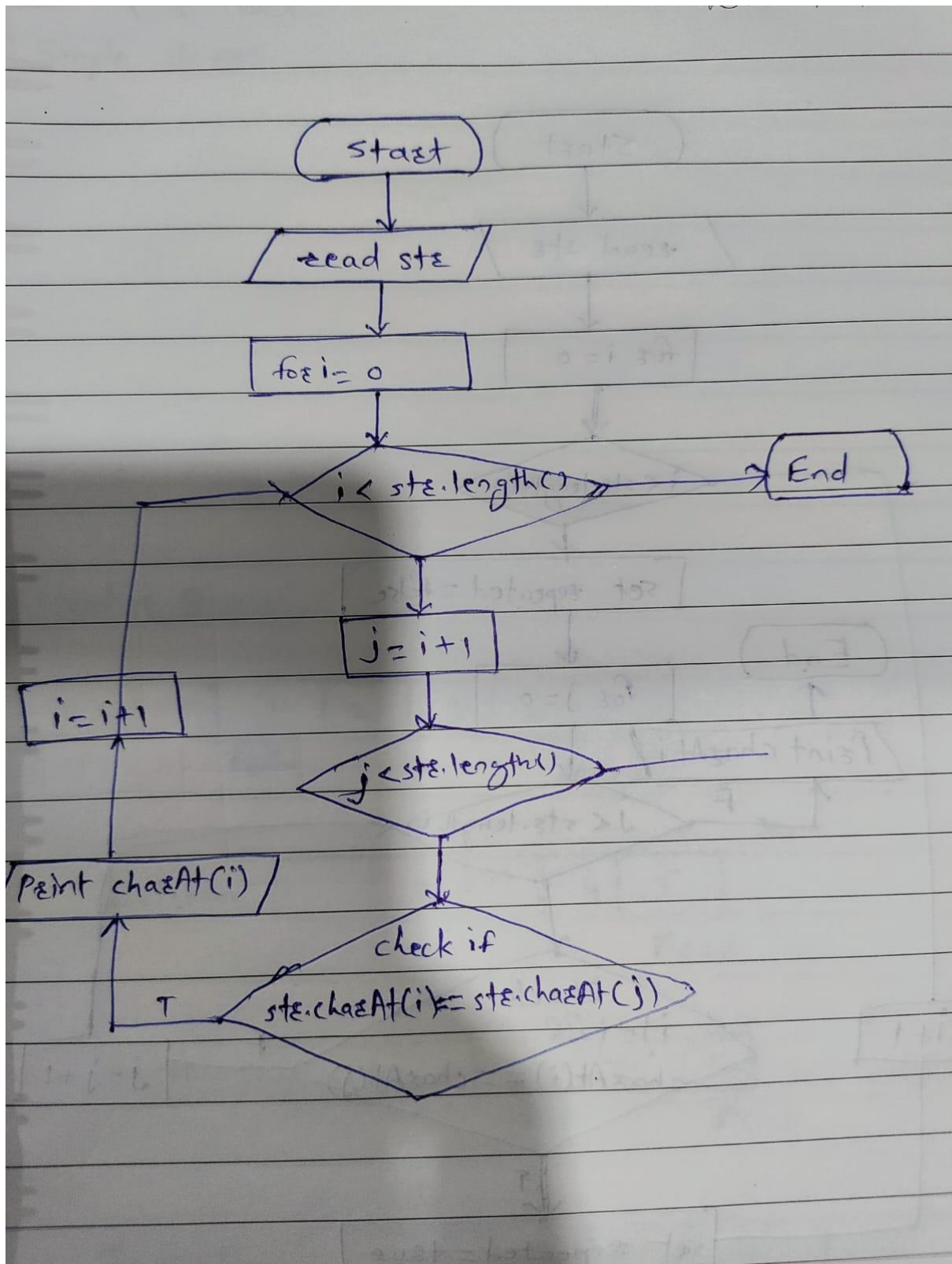
```
/*Find repeated character*/
import java.util.Scanner;
class RepeatedCharacter{

    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);

        String str = sc.nextLine();

        for(int i=0; i<str.length(); i++){
            char thisChar = str.charAt(i);
            for(int j=i+1; j<str.length(); j++){
                if(thisChar == str.charAt(j)){
                    System.out.print(thisChar + " ");
                    break;
                }
            }
        }
    }
}
```

```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>javac RepeatedCharacter.java
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java RepeatedCharacter
programming
r g m
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java RepeatedCharacter
hello
l
```



Time & Space Complexity :

The time complexity of this solution is $O(n^2)$ because of the nested loops.

To reduce the time complexity of this program we can make use of Collection

The space complexity of this program is $O(n)$

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

```
/* First Non-repeated character*/
```

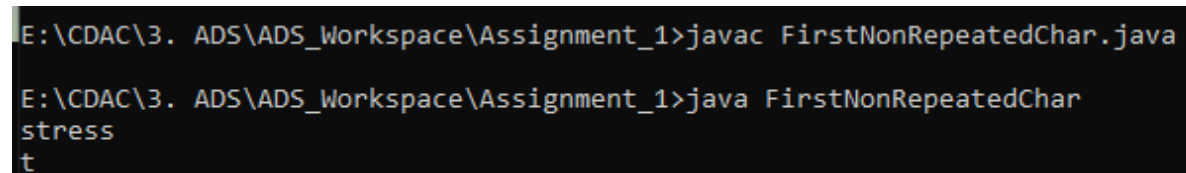
```
import java.util.Scanner;
class FirstNonRepeatedChar{

    public static void main(String args[]){

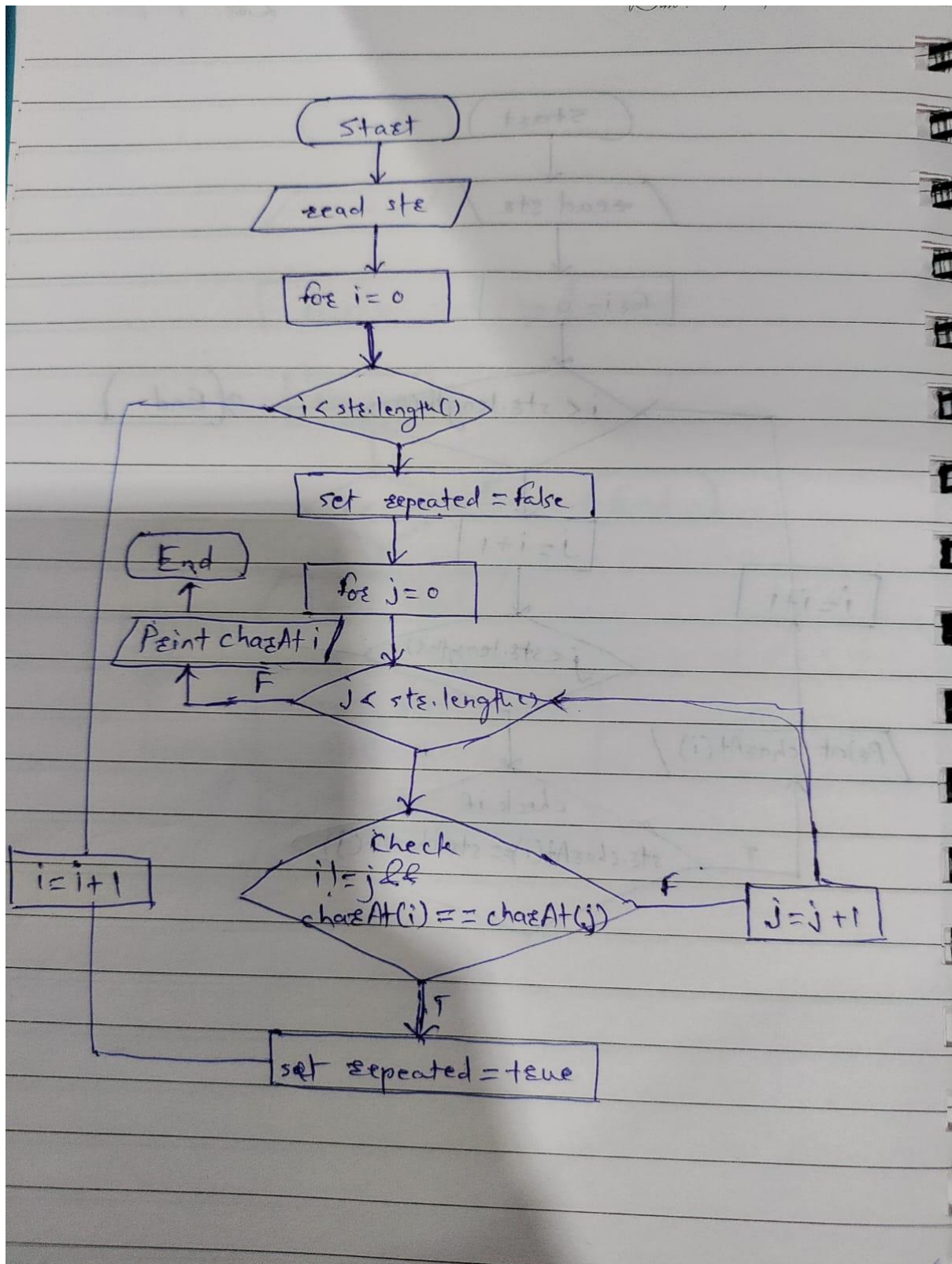
        Scanner sc = new Scanner(System.in);

        String str = sc.nextLine();

        for(int i=0; i<str.length(); i++){
            char thisChar = str.charAt(i);
            boolean repeated = false;
            for(int j=0; j<str.length(); j++){
                if(i!=j && thisChar == str.charAt(j)){
                    repeated = true;
                    break;
                }
            }
            if(!repeated){
                System.out.println(thisChar);
                break;
            }
        }
    }
}
```



```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>javac FirstNonRepeatedChar.java
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java FirstNonRepeatedChar
stress
t
```



Time & Space Complexity :

The time complexity is $O(n^2)$ due to the nested loops, where n is the length of the string. Space complexity used by the input string is $O(n)$.

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

```
/* Palindrome */
import java.util.Scanner;
class Palindrome{

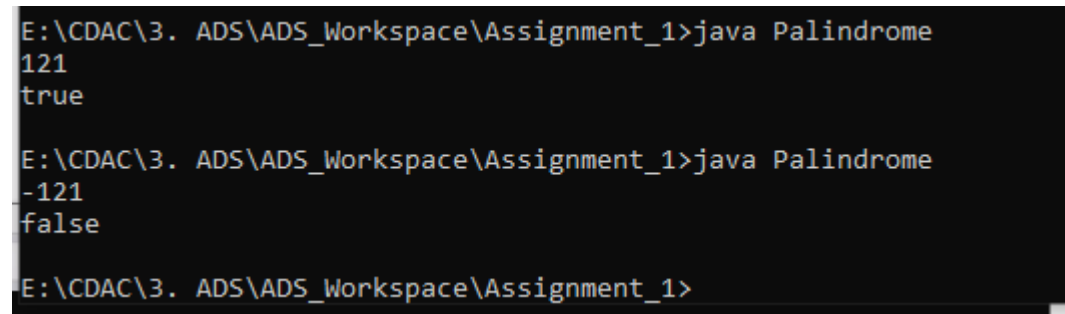
    static boolean isPalindrome(int n){
        if(n < 0)
            return false;

        int number = n;
        int reverse = 0;
        while(n != 0){
            int rem = n%10;
            reverse = reverse * 10 + rem;
            n = n/10;
        }
        return number == reverse;
    }

    public static void main(String args[]){

        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println(isPalindrome(n));

    }
}
```



```
E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java Palindrome
121
true

E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>java Palindrome
-121
false

E:\CDAC\3. ADS\ADS_Workspace\Assignment_1>
```

Explanation :

A number is considered a palindrome if it reads the same backward as forward.

We reverse the number and check if it matches with the original number. If it is same we can say that the number is palindrome.

Time & Space Complexity :

The time complexity for this program is $O(1)$

The space complexity of this program is $O(1)$

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

```
/* Leap Year */
```

```
import java.util.Scanner;
```

```
class LeapYear{
```

```
    static boolean isLeapYear(int year){
```

```
        if(year % 400 == 0 )
```

```
            return true;
```

```
        if(year % 100 == 0)
```

```
            return false;
```

```
        return year % 4 == 0;
```

```
    }
```

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

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

```
        int year = sc.nextInt();
```

```
        System.out.println(isLeapYear(year));
```

```
    }
```

```
}
```

```
E:\CDAC\3. ADS\ADS_Workspace\Queue>javac LeapYear.java  
  
E:\CDAC\3. ADS\ADS_Workspace\Queue>java LeapYear  
2020  
true  
  
E:\CDAC\3. ADS\ADS_Workspace\Queue>java LeapYear  
1900  
false
```

Explanation :

A year is a leap year if it is divisible by 400 or it is divisible by 4. But not divisible by 100.

Time & Space Complexity :

The time complexity for this program is $O(1)$

The space complexity of this program is $O(1)$