1) Jake has a string array arr of size n and two special characters, target1 and target2. He wants to count the total number of occurrences of target1 in all strings in arr, but only if target1 is immediately followed by target2 in the string.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static int buggyCountTargetCharacterPairs(List<String> arr, char target1, char target2) {

        // Fix the code here

        int count = 0;

        for (String s : arr) {

            for (int i = 0; i < s.length() - 1; i++) {

                if (s.charAt(i) == target1 && s.charAt(i + 1) == target2) {

                    count++;

                }

            }

        }

        return count;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n = Integer.parseInt(scan.nextLine().trim());

        List<String> arr = new ArrayList<>(n);

        for(int j=0; j<n; j++) {

            arr.add(scan.nextLine());

        }

        char target1 = scan.nextLine().charAt(0);

        char target2 = scan.nextLine().charAt(0);

        int result = buggyCountTargetCharacterPairs(arr, target1, target2);

        System.out.println(result);

    }

}

2) Joe has two integers a and b. He wants to find the remainder when a is divided by b and do the following:

1. If the remainder is an even number greater than 2, then output the sum of a, b, and the remainder.
2. If the remainder is an odd number, then output the product of a and the remainder.
3. If the remainder is 0, 1, or 2, then output the sum of a and the remainder.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

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

        // Fix the code here

        int remainder = a % b;

        if (remainder > 2 && remainder % 2 == 0) {

            return a + b + remainder;

        } else if (remainder % 2 == 1) {

            return a \* remainder;

        } else if (remainder >= 0 && remainder <= 2) {

            return a + remainder;

        } else {

            System.out.println("Unexpected case");

            return -1;  // Return some error code

        }

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int a = Integer.parseInt(scan.nextLine().trim());

        int b = Integer.parseInt(scan.nextLine().trim());

        int result = buggyRemainderOperation(a, b);

        System.out.println(result);

    }

}

3) John has an array arr of size n. He wants to find the sum of squares of all even numbers in arr that are also divisible by 4.

John has written a function buggySumOfEvenSquaresDivisibleByFour that accepts n and arr as a list of integers in its arguments. However, there are some bugs in the function logic.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static int buggySumOfEvenSquaresDivisibleByFour(List<Integer> arr) {

        // Fix the code here

        int result = 0;

        for (int num : arr) {

            if (num % 2 == 0 && num % 4 == 0) {

                result += num \* num;

            }

        }

        return result;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n = Integer.parseInt(scan.nextLine().trim());

        List<Integer> arr = new ArrayList<>(n);

        for(int j=0; j<n; j++) {

            arr.add(Integer.parseInt(scan.nextLine().trim()));

        }

        int result = buggySumOfEvenSquaresDivisibleByFour(arr);

        System.out.println(result);

    }

}

4) Bob has a string S of size N. He wants to create a transformed string by replicating the first vowel character in S across the entire length of S. If no vowel is present, he wants to replicate the first character of S.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static String ReplicateVowel(int N, String S) {

        //Fix the code here

        Set<Character> vowels = new HashSet<>(Arrays.asList('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'));

        for(int i = 0; i < N; i++) {

            char c = S.charAt(i);

            if(vowels.contains(c)) {

                char[] chars = new char[N];

                Arrays.fill(chars, c);

                return new String(chars);

            }

        }

        char[] chars = new char[N];

        Arrays.fill(chars, S.charAt(0));

        return new String(chars);

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int N = Integer.parseInt(scan.nextLine().trim());

        String S = scan.nextLine();

        String result = ReplicateVowel(N, S);

        System.out.println(result);

    }

}

5) Bob has a string S. He wants to replace all vowels in S with the number 3. However, if a vowel is surrounded by consonants on both sides, it should be replaced with the number 5 instead.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static String BuggyReplaceVowels(String s) {

        // Fix the code here

        StringBuilder sb = new StringBuilder();

        int len = s.length();

        for (int i = 0; i < len; i++) {

            char ch = s.charAt(i);

            if ("aeiouAEIOU".indexOf(ch) != -1) {

                if (i > 0 && i < len - 1 && "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ".indexOf(s.charAt(i - 1)) != -1

                        && "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ".indexOf(s.charAt(i + 1)) != -1) {

                    sb.append('5');

                } else {

                    sb.append('3');

                }

            } else {

                sb.append(ch);

            }

        }

        return sb.toString();

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String s = scan.nextLine();

        String result = BuggyReplaceVowels(s);

        System.out.println(result);

    }

}

6) John has an array arr of size n. He wants to find the sum of squares of all odd numbers in arr that are also perfect squares.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static int buggySumOfOddPerfectSquareSquares(int n, List<Integer> arr) {

        // Fix the code here

        int sum = 0;

        for (int i = 0; i < n; i++) {

            int root = (int) Math.sqrt(arr.get(i));

            if (arr.get(i) % 2 != 0 && root \* root == arr.get(i)) {

                sum += arr.get(i) \* arr.get(i);

            }

        }

        return sum;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n = Integer.parseInt(scan.nextLine().trim());

        List<Integer> arr = new ArrayList<>(n);

        for(int j=0; j<n; j++) {

            arr.add(Integer.parseInt(scan.nextLine().trim()));

        }

        int result = buggySumOfOddPerfectSquareSquares(n, arr);

        System.out.println(result);

    }

}

7) You are given a partially implemented code to check if a given string is a specific type of palindrome known as a **"mirrored string"**. If the given string is a **mirrored string**, return 1; else return 0.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static int buggyIsMirroredString(String str) {

        // Fix the code here

        String mirrored\_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        str = str.toUpperCase();

        return (str.equals(new StringBuilder(str).reverse().toString()) &&

                str.chars().allMatch(ch -> mirrored\_chars.indexOf(ch) != -1)) ? 1 : 0;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String str = scan.nextLine();

        int result = buggyIsMirroredString(str);

        System.out.println(result);

    }

}

8) John has an array arr of size n. He wants to find the sum of squares of all even numbers in arr that are at odd-indexed positions. It is given that arr is 0-indexed.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static int buggySumOfEvenSquaresAtOddIndices(int n, List<Integer> arr) {

        // Fix the code here

        int sum = 0;

        for(int i = 1; i < n; i += 2) {

            if(arr.get(i) % 2 == 0) {

                sum += Math.pow(arr.get(i), 2);

            }

        }

        return sum;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n = Integer.parseInt(scan.nextLine().trim());

        List<Integer> arr = new ArrayList<>(n);

        for(int j=0; j<n; j++) {

            arr.add(Integer.parseInt(scan.nextLine().trim()));

        }

        int result = buggySumOfEvenSquaresAtOddIndices(n, arr);

        System.out.println(result);

    }

}

9) You are given a partially implemented code to determine if a given string contains all unique characters, excluding whitespace. The code reads a string and calls the function buggyHasUniqueNonWhitespaceCharacters to check if the string, after removing all whitespace, has all unique characters.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static boolean buggyHasUniqueNonWhitespaceCharacters(String str) {

        // Fix the code here

        str = str.replaceAll(" ", "");

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

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

            if (charCount.containsKey(c)) {

                return false;

            }

            charCount.put(c, 1);

        }

        return true;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String str = scan.nextLine();

        boolean result = buggyHasUniqueNonWhitespaceCharacters(str);

        System.out.println(result? "1": "0");

    }

}

10) You are given a partially implemented code to determine if a given string is a valid URL with a specific domain suffix. The code reads a string and calls the function buggyIsValidURLWithSpecificSuffix to check if the string is a valid URL and ends with the domain suffix ".org".

**Notes:**

A valid URL should:

1. Start with a valid scheme (e.g., "http", "https", "ftp").
2. Followed by "://".
3. Then, a domain part, which may contain alphanumeric characters (both uppercase and lowercase), hyphens (-), and periods (.). The domain part must start and end with an alphanumeric character.
4. End with the domain suffix ".org".
5. Optionally, a path part after the domain, which may contain alphanumeric characters (both uppercase and lowercase), hyphens (-), slashes (/), and periods (.)

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static boolean buggyIsValidURLWithSpecificSuffix(String url) {

        // Fix the code here

        String[] schemes = {"http://", "https://", "ftp://"};

        String[] parts = url.split("://");

        if (parts.length != 2 && !Arrays.asList(schemes).contains(parts[0])) {

            return false;

        }

        parts = parts[1].split("/");

        String domain = parts[0];

        if (!domain.endsWith(".org") || !Character.isLetterOrDigit(domain.charAt(0)) || !Character.isLetterOrDigit(domain.charAt(domain.length() - 5))) {

            return false;

        }

        for (char c : domain.toCharArray()) {

            if (!Character.isLetterOrDigit(c) && c != '-' && c != '.') {

                return false;

            }

        }

        for (String part : Arrays.copyOfRange(parts, 1, parts.length)) {

            for (char c : part.toCharArray()) {

                if (!Character.isLetterOrDigit(c) && c != '-' && c != '/' && c != '.') {

                    return false;

                }

            }

        }

        return true;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String url = scan.nextLine();

        boolean result = buggyIsValidURLWithSpecificSuffix(url);

        System.out.println(result? "1": "0");

    }

}

11) You are given a partially implemented code to determine if a given string is a valid email address. The code reads a string and calls the function buggyIsValidEmail to check if the string is a valid email address.

**Notes:**

A valid email address for this task should:

1. Consist of a local part followed by an "@" symbol, and a domain part.
2. The local part may contain alphanumeric characters (both uppercase and lowercase), periods (.), and plus signs (+), but it should not start or end with a period or plus sign.
3. The domain part may contain alphanumeric characters (both uppercase and lowercase) and hyphens (-), but it must start and end with an alphanumeric character.
4. The domain part should also contain at least one period (.) symbol, indicating a domain suffix like .com, .org, etc.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static boolean buggyIsValidEmail(String emailStr) {

        int atPos = emailStr.indexOf('@');

        if (atPos == -1) {

            return false;

        }

        String local = emailStr.substring(0, atPos);

        String domain = emailStr.substring(atPos + 1);

        if (local.charAt(0) == '.' || local.charAt(0) == '+' || local.charAt(local.length() - 1) == '.' || local.charAt(local.length() - 1) == '+' || local.contains("..")) {

            return false;

        }

        if (domain.charAt(0) == '-' || domain.charAt(0) == '.' || domain.charAt(domain.length() - 1) == '-' || domain.charAt(domain.length() - 1) == '.' || domain.contains("..")) {

            return false;

        }

        if (!domain.contains(".")) {

            return false;

        }

        return true;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String emailStr = scan.nextLine();

        boolean result = buggyIsValidEmail(emailStr);

        System.out.println(result? "1": "0");

    }

}

12) Jill has a string str. She wants to find the number of words in str that both start and end with a specified character c.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static int buggyCountWordsStartingAndEndingWithCharacter(String str, char c) {

        // Fix the code here

        int count = 0;

        String[] words = str.split(" ");

        for(String word : words) {

            if(word.startsWith(Character.toString(c)) && word.endsWith(Character.toString(c))) {

                count++;

            }

        }

        return count;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String str = scan.nextLine();

        char c = scan.nextLine().charAt(0);

        int result = buggyCountWordsStartingAndEndingWithCharacter(str, c);

        System.out.println(result);

    }

}

13) Jake has a string array arr of size n, an integer value length, and a character targetChar. He wants to count the total number of strings in arr whose length is equal to the given integer value and that also start with the targetChar.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static int buggyCountStringsOfTargetLengthAndChar(int n, List<String> arr, int length, char targetChar) {

        // Fix the code here

        int count = 0;

        for (int i = 0; i < n; i++) {

            if (arr.get(i).length() == length && arr.get(i).charAt(0) == targetChar) {

                count++;

            }

        }

        return count;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n = Integer.parseInt(scan.nextLine().trim());

        List<String> arr = new ArrayList<>(n);

        for(int j=0; j<n; j++) {

            arr.add(scan.nextLine());

        }

        int length = Integer.parseInt(scan.nextLine().trim());

        char targetChar = scan.nextLine().charAt(0);

        int result = buggyCountStringsOfTargetLengthAndChar(n, arr, length, targetChar);

        System.out.println(result);

    }

}

14) Jack has two strings, str and t. He wants to find the total number of non-overlapping occurrences of string t in str.

**Notes:**

* Let str = "ababa" and t = "aba". If we only count non-overlapping occurrences, then "aba" appears only once in "ababa". This is because, after the first occurrence is found at the start of the string, the search continues from the character after the last character of the found occurrence. Thus, after finding the first "aba", the search would continue from the second "b" in "ababa", and no further occurrences of "aba" would be found. Therefore, for str = "ababa" and t = "aba", buggyCountNonOverlappingSubstringOccurrences(str, t) should return 1.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static int buggyCountNonOverlappingSubstringOccurrences(String str, String t) {

        // Fix the code here

        if (t.isEmpty()) {

            return 0;

        }

        int count = 0;

        int start = 0;

        while ((start = str.indexOf(t, start)) != -1) {

            count++;

            start += t.length();

        }

        return count;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String str = scan.nextLine();

        String t = scan.nextLine();

        int result = buggyCountNonOverlappingSubstringOccurrences(str, t);

        System.out.println(result);

    }

}

15) You are given a partially implemented code to determine if a given string is a valid credit card number. The code reads a string and calls the function buggyIsValidCreditCard to check if the string is a valid credit card number.

However, the function contains a bug that causes it to produce incorrect results.

A valid credit card number should:

1. Contain exactly 16 digits.
2. Start with 4, 5, or 6.
3. Contain only digits (0-9).
4. Pass the Luhn algorithm (a simple checksum formula used to validate a variety of identification numbers, such as credit card numbers).

The Luhn algorithm works as follows:

1. Double the value of every second digit from the rightmost digit.
2. If doubling a number results in a two-digit number, add the two digits to get a single-digit number.
3. Take the sum of all the single-digit numbers obtained in the previous steps.
4. If the total modulo 10 is equal to 0 (if the total ends in zero), the number is valid according to the Luhn formula; otherwise, it is not valid.

For example, if the input string is "4532015112830366", the output should be true, but if the input string is "1234567812345678", the output should be false because it does not pass the Luhn algorithm or start with 4, 5, or 6.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static boolean buggyIsValidCreditCard(String cardNumber) {

        // Fix the code here

        if (cardNumber.length() != 16) {

            return false;

        }

        char startDigit = cardNumber.charAt(0);

        if (startDigit != '4' && startDigit != '5' && startDigit != '6') {

            return false;

        }

        for (char c : cardNumber.toCharArray()) {

            if (!Character.isDigit(c)) {

                return false;

            }

        }

        // Luhn algorithm

        int sum = 0;

        for (int i = cardNumber.length() - 1; i >= 0; i--) {

            int digit = cardNumber.charAt(i) - '0';

            if (i % 2 == cardNumber.length() % 2) {

                digit \*= 2;

                if (digit > 9) {

                    digit -= 9;

                }

            }

            sum += digit;

        }

        return sum % 10 == 0;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String str = scan.nextLine();

        boolean result = buggyIsValidCreditCard(str);

        System.out.println(result? "1": "0");

    }

}

16) You are given a partially implemented code to determine if a given string is a valid phone number. The code reads a string and calls the function buggyIsValidPhoneNumber to check if the string is a valid phone number.

However, the function contains a bug that causes it to produce incorrect results. Your task is to find and fix the bug in the buggyIsValidPhoneNumber function.

**Notes:**

A valid phone number should meet the following requirements:

1. Contain exactly 10 digits, length of the input can be longer if parentheses and/or dashes included.
2. Contain only digits (0-9), optional parentheses, and optional dashes.
3. If parentheses are included, they should appear around the first three digits (e.g., "(123)456-7890").
4. If dashes are included, they should appear either between the fourth and fifth digits or after the seventh digit (e.g., "123-4567890" or "123456-7890").

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static boolean buggyIsValidPhoneNumber(String s) {

        // Fix the code here

        if (s.length() == 10) {

            return s.matches("\\d+");

        } else if (s.length() == 13) {

            return s.charAt(0) == '(' && s.charAt(4) == ')' && s.charAt(8) == '-' &&

                   s.substring(1,3).matches("\\d+") &&

                   s.substring(5,7).matches("\\d+") &&

                   s.substring(9,13).matches("\\d+");

        } else if (s.length() == 11 && s.charAt(3) == '-') {

            return s.substring(0,3).matches("\\d+") &&

                   s.substring(4,11).matches("\\d+");

        } else if (s.length() == 11 && s.charAt(6) == '-') {

            return s.substring(0,6).matches("\\d+") &&

                   s.substring(7,11).matches("\\d+");

        }

        return false;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String s = scan.nextLine();

        boolean result = buggyIsValidPhoneNumber(s);

        System.out.println(result? "1": "0");

    }

}

17) John has an array arr of size n. He wants to find the sum of the factorial of all odd numbers in arr that are less than 10.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    public static int buggySumOfOddFactorialsLessThanTen(int n, List<Integer> arr) {

        // Fix the code here

        int sum = 0;

        for(int i = 0; i < n; i++){

            if(arr.get(i) < 10 && arr.get(i) % 2 != 0)

                sum += factorial(arr.get(i));

        }

        return sum;

    }

    public static int factorial(int n){

        return (n == 1 || n == 0) ? 1 : factorial(n - 1) \* n;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n = Integer.parseInt(scan.nextLine().trim());

        List<Integer> arr = new ArrayList<>(n);

        for(int j=0; j<n; j++) {

            arr.add(Integer.parseInt(scan.nextLine().trim()));

        }

        int result = buggySumOfOddFactorialsLessThanTen(n, arr);

        System.out.println(result);

    }

}

18) Sam has an array arr containing n integers. He wants to calculate the product of all prime numbers in arr that are greater than 10. Therefore, he has written a function buggyProductOfPrimesGreaterThanTen that accepts n and stores arr as a list of integers as its arguments.

import java.io.\*;

import java.util.\*;

import java.lang.Math;

public class Solution {

    static boolean isPrime(int num) {

        if (num <= 1) return false;

        if (num <= 3) return true;

        if (num % 2 == 0 || num % 3 == 0) return false;

        int i = 5;

        while (i \* i <= num) {

            if (num % i == 0 || num % (i + 2) == 0) return false;

            i += 6;

        }

        return true;

    }

    public static int buggyProductOfPrimesGreaterThanTen(int n, List<Integer> arr) {

        // Fix the code here

        int product = 1;

        for(int num : arr) {

            if (num > 10 && isPrime(num)) {

                product \*= num;

            }

        }

        return product;

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n = Integer.parseInt(scan.nextLine().trim());

        List<Integer> arr = new ArrayList<>(n);

        for(int j=0; j<n; j++) {

            arr.add(Integer.parseInt(scan.nextLine().trim()));

        }

        int result = buggyProductOfPrimesGreaterThanTen(n, arr);

        System.out.println(result);

    }

}