



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Lab Assignment

Student Name: Rishu Raj

UID:22BCS15617

Branch: BE-CSE

Section/Group: IOT_631/B

Semester: 6th

DOP:04/04/2025

Subject Name: PBLJ

Subject Code: 22CSH-359

Aim: Problem 1.

Consider a function `public String matchFound(String input 1, String input 2)`, where `input1` will contain only a single word with only 1 character replaces by an underscore `'_'` `input2` will contain a series of words separated by colons and no space character in between `input2` will not contain any other special character other than underscore and alphabetic characters.

The methods should return output in a String type variable `"output1"` which contains all the words from `input2` separated by colon which matches with `input` All words in `output1` should be in uppercase.

Objective:

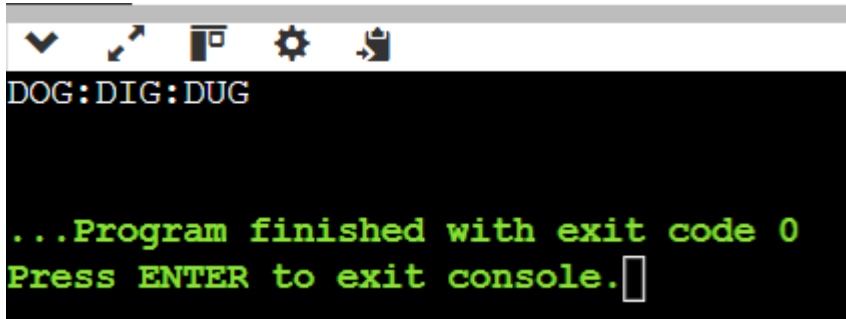
- Pattern Matching: Identify words from `input2` that match the structure of `input1`, where `_` acts as a wildcard.
- String Processing: Extract, compare, and transform matched words to uppercase while maintaining a colon-separated format.
- Efficiency & Robustness: Ensure optimized performance with minimal memory usage and handle edge cases like no matches or empty inputs.

Code:

```
public class MatchFinder {
    public static String matchFound(String input1, String input2) {
        StringBuilder output1 = new StringBuilder();
        String[] words = input2.split(":");
        for (String word : words) {
            if (matchesPattern(input1, word)) {
                if (output1.length() > 0) {
                    output1.append(":");
                }
                output1.append(word.toUpperCase());
            }
        }
        return output1.toString();
    }
    private static boolean matchesPattern(String pattern, String word) {
```

```
    if (pattern.length() != word.length()) {  
        return false;  
    }  
    // Iterate through the characters of both strings  
    for (int i = 0; i < pattern.length(); i++) {  
        if (pattern.charAt(i) != '_' && pattern.charAt(i) != word.charAt(i)) {  
            return false;  
        }  
    }  
    return true;  
}  
public static void main(String[] args) {  
    String input1 = "d_g";  
    String input2 = "dog:dig:dug:log:fog";  
    String result = matchFound(input1, input2);  
    System.out.println(result); // Output: DOG:DIG:DUG  
}  
}
```

Output:



```
DOG:DIG:DUG  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Aim: Problem 4:

String t is generated by random shuffling string s and then add one more letter at a random position.
Return the letter that was added to t. Hint: Input: s = "abcd", t = "abcde" Output: "e"

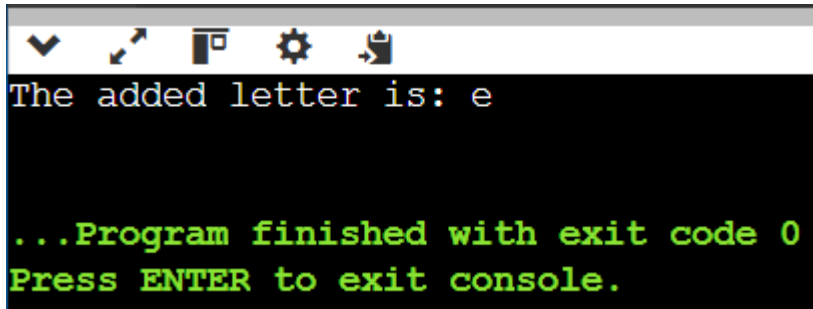
Objective:

- Character Difference Identification: Determine the extra letter added to t by comparing it with s.
- Optimized String Processing: Efficiently handle character comparisons without excessive loops or memory usage.
- Robustness & Accuracy: Ensure correctness even with shuffled input and different character positions.

Code:

```
public class FindAddedLetter {  
    public static char findAddedLetter(String s, String t) {  
        int charSumS = 0, charSumT = 0;  
  
        for (char c : s.toCharArray()) {  
            charSumS += c;  
        }  
        for (char c : t.toCharArray()) {  
            charSumT += c;  
        }  
        return (char) (charSumT - charSumS);  
    }  
    public static void main(String[] args) {  
        String s = "abcd";  
        String t = "abcde";  
        System.out.println("The added letter is: " + findAddedLetter(s, t)); // Output: e  
    }  
}
```

Output:



```
The added letter is: e  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Aim: Java's BigDecimal class can handle arbitrary-precision signed decimal numbers. Let's test your knowledge of them!

Given an array, , of real number strings, sort them in descending order — but wait, there's more! Each number must be printed in the exact same format as it was read from stdin, meaning that is printed as , and is printed as . If two numbers represent numerically equivalent values (e.g.,), then they must be listed in the same order as they were received as input).

You must rearrange array 's elements according to the instructions above.

Input Format

The first line consists of a single integer, , denoting the number of integer strings. Each line of the subsequent lines contains a real number denoting the value of .

Constraints

- Each has at most digits.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Sample Input

-100

50

56.6

90

0.12

.12

02.34

Sample Output

90

56.6

50

02.34

0.12

.12 0

-100

Objective:

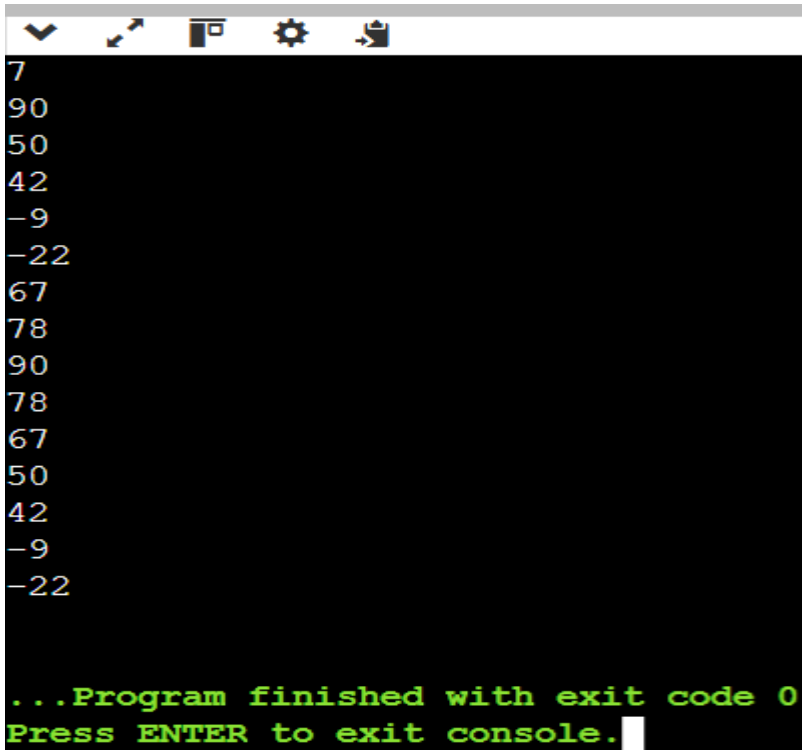
- Arbitrary Precision Handling: Use Java's BigDecimal to sort large and precise decimal numbers accurately.
- Maintain Original Formatting: Preserve the input format while sorting, ensuring no unintended modifications.
- Stable Sorting Algorithm: Maintain the order of numerically equivalent values as they appear in the input.

Code:

```
import java.math.BigDecimal;
import java.util.*;
public class BigDecimalSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = Integer.parseInt(scanner.nextLine());
        String[] numbers = new String[n];
        for (int i = 0; i < n; i++) {
            numbers[i] = scanner.nextLine();
        }
        // Custom Comparator for sorting
        Arrays.sort(numbers, (a, b) -> new BigDecimal(b).compareTo(new BigDecimal(a)));
    }
}
```

```
// Print sorted numbers in original format
for (String num : numbers) {
    System.out.println(num);
}
scanner.close();
}}
```

Output:



```
7
90
50
42
-9
-22
67
78
90
78
67
50
42
-9
-22

...Program finished with exit code 0
Press ENTER to exit console.
```

Aim: Problem 6:

A string containing only parentheses is balanced if the following is true: 1. if it is an empty string

2. if A and B are correct, AB is correct,

3. if A is correct, (A) and {A} and [A] are also correct.

Examples of some correctly balanced strings are: "{}()", "{()}", "({})"

Examples of some unbalanced strings are: "{}(", "{()}", "[{", "}" etc.

Given a string, determine if it is balanced or not.

Input Format There will be multiple lines in the input file, each having a single nonempty string. You should read input till end-of-file.

Output Format For each case, print 'true' if the string is balanced, 'false' otherwise. Sample Input {}()

{()}) {} ([]

Sample Output true true false true

Objective:

- Stack-Based Validation: Use a stack to ensure that every opening bracket has a corresponding and correctly ordered closing bracket.
- Efficient Parsing: Process each input string in $O(n)$ time complexity, where n is the string length.
- EOF Handling & Robustness: Continuously read multiple input lines and check balance until EOF.

Code:

```
import java.util.*;

public class BalancedParentheses {

    public static boolean isBalanced(String s) {

        Stack<Character> stack = new Stack<>();

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

            if (c == '(' || c == '{' || c == '[') {
                stack.push(c);
            } else if (c == ')' || c == '}' || c == ']') {

                if (stack.isEmpty()) {
                    return false;
                }
                char top = stack.pop();

                if ((c == ')' && top != '(') || (c == '}' && top != '{') ||
                    (c == ']' && top != '[')) {
                    return false; }

            } }
        return stack.isEmpty();
    }

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

        while (scanner.hasNext()) {

            String input = scanner.next();
```

```
System.out.println(isBalanced(input));  
    }  
    scanner.close(); } }
```

Output:

```
() {} {}  
true  
{ () }  
true  
{ () }  
false  
[{ () }]  
true
```

Aim: Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]

Constraints:

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- nums is a non-decreasing array.
- $-10^9 \leq \text{target} \leq 10^9$

Objective:

- Binary Search Optimization: Use binary search to achieve $O(\log n)$ time complexity instead of linear search.
- Find Both First & Last Occurrences: Locate the leftmost (starting) and rightmost (ending) positions of the target efficiently.

- Handle Edge Cases: Return [-1, -1] when the target is not found and correctly handle arrays with duplicates.

Code:

```
import java.util.*; public class  
  
FindTargetRange {  
  
public static int[] searchRange(int[] nums, int  
target) {  
  
int first = findFirst(nums, target);  
int last = findLast(nums, target);  
  
return new int[] {first, last};  
  
}  
private static int findFirst(int[] nums, int target) {  
int left = 0, right = nums.length - 1, result = -1;  
while (left <= right) {  
int mid = left + (right - left) / 2;  
if (nums[mid] >= target) {  
right = mid - 1;        }  
else {  
left = mid + 1;  
}  
if (nums[mid] == target) {  
result = mid;  
} }  
return result;  
}  
private static int findLast(int[] nums, int target) {  
int left = 0, right = nums.length - 1, result = -1;  
while (left <= right) {
```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
int mid = left + (right - left) / 2;

if (nums[mid] <= target) {

left = mid + 1;      }

else {

right = mid - 1; }

if (nums[mid] == target) {

result = mid;

} }

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of elements:");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the elements in sorted order:");

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

nums[i] = scanner.nextInt();

}

System.out.println("Enter the target value:");

int target = scanner.nextInt();

System.out.println(Arrays.toString(searchRange(nums, target)));    scanner.close();

}
```

Output:

```
Enter the number of elements:
8
Enter the elements in sorted order:
2 4 6 7 8 9 9 9
Enter the target value:
7
[3, 3]

...Program finished with exit code 0
Press ENTER to exit console.
```

Learning Outcomes:

- Learned how to implement binary search for $O(\log n)$ efficiency and custom sorting using BigDecimal.
- Understood how to use stacks for balanced parentheses checking, ensuring correct order and matching.
- Practiced handling and manipulating strings while preserving formats, such as matching words with wildcards.
- Explored arbitrary-precision arithmetic using BigDecimal and efficient character sum methods for unique identification.
- Improved skills in handling edge cases, stable sorting, and parsing multiple inputs dynamically in Java.