**LAB 05**

**Recursion**

Nguyễn Minh Đạt – ITDSIU22166

Problem 1: Use the following function puzzle(..) to answer problems 1 - 3.

1. Base case: With the base case, base > limit then the profram will return -1 because

A black background with white text

AI-generated content may be incorrect.

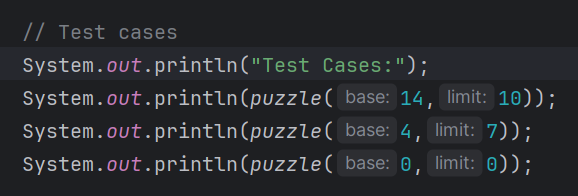
1. Recursive case: When the base = limit, the puzzle will be 1, when the base < limit, the program will return base \* puzzle(base + 1, limit), which mimics a factorial process over a flexible range.

A screenshot of a computer

AI-generated content may be incorrect.

1. Following task:

A screen shot of a test

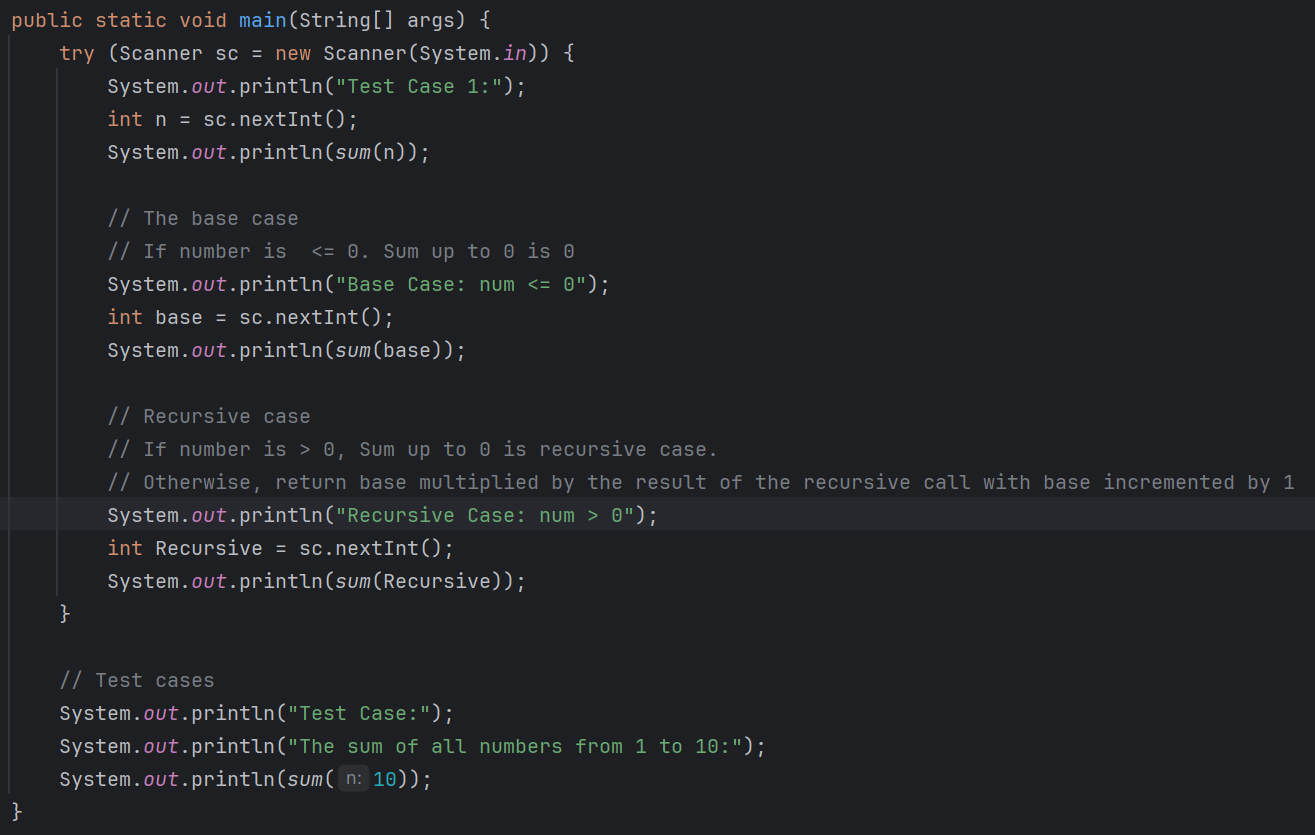
AI-generated content may be incorrect. 

a. System.out.print(puzzle(14,10));

b. System.out.print(puzzle(4,7));

c. System.out.print(puzzle(0,0));

Problem 3: Write a recursive function that computes the sum of all numbers from 1 to n, where n is given as a parameter.



A screenshot of a computer

AI-generated content may be incorrect.

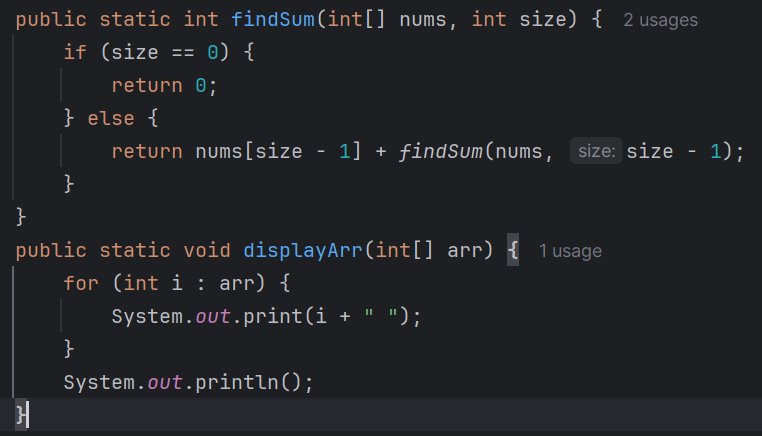
**Design:** The sum(int num) function recursively computes the sum from 1 to num, assuming num >= 1. When num <= 0, It will be the base case of program when it returns 0 as the sum of number up to 0 is 0. If num == 1, it returns 1. Otherwise, it returns num + sum(num - 1).

**Output:**

A black background with white text

AI-generated content may be incorrect.

Problem 5: Write a recursive function that computes and returns the sum of all elements in an array, where the array and its size are given as parameters.



* **Design:** The function findSum(int[] nums, int size) calculates the sum of all elements in the array using recursive logic based on array size.
* **Recursion Logic:** If the size is 0, it returns 0 (empty array case). Otherwise, it returns the last element nums[size - 1] plus the sum of the remaining array elements.
* **Observation:** This method works well for small arrays but is less efficient than an iterative approach for large datasets due to the overhead of recursive calls.
* **Output:**

A black screen with white text

AI-generated content may be incorrect.

Problem 7: Write a recursive function that takes a string as input and reverses it using recursion.

* **Design:** revStr(String str, int len) reverses a string recursively by appending the last character to the result of a recursive call on the substring.
* **Recursion Logic:** The function reduces the string length by 1 each call and concatenates the character at that position until the length becomes negative.
* **Observation:** It’s less efficient than using built-in string manipulation or iterative methods.
* **Output:**

A black screen with white text

AI-generated content may be incorrect.

Problem 8: Write a recursive function to generate all subsets of a given set.

* **Design:** The subset generator recursively builds all subsets of an array using inclusion and exclusion principles. It uses a helper function subsetRecur() and a main driver function subsets().
* **Recursion Logic:** At each index, the algorithm chooses whether to include the current element in the subset or not, then proceeds recursively through both choices.
* **Observation:** This is a classic backtracking example that demonstrates how recursion can explore all possible combinations. It scales exponentially (O(2^n)), which makes it a powerful but computationally expensive technique.
* **Output:**

A screenshot of a computer

AI-generated content may be incorrect.

Problem 11: Use recursion to generate a Von Koch snowflake

* **Design:** The KochSnowflake class uses Java Swing to create a window and draw a fractal snowflake using recursive geometry. The shape is based on subdividing triangle sides into smaller segments.
* **Recursion Logic:** The drawKochCurve() function breaks each line segment into four parts, inserting a new point that forms a peak using 60-degree rotations. It repeats recursively until the base case (level 0) is reached.
* **Observation:** This project visually demonstrates the power and complexity of recursion. As the recursion level increases, the number of line segments grows exponentially, leading to slow rendering and high CPU usage at deeper levels.
* **Output:**

A black and white snowflake

AI-generated content may be incorrect.