aDDING THE MULTIPLE OF A NUMBER USING rECURSION fUNCTION

# Research

Recursion is a programming technique where a function calls itself to solve smaller instances of the same problem, allowing for elegant solutions to complex tasks. This method is particularly useful in scenarios like tree traversals, sorting algorithms, and mathematical computations such as calculating factorials or Fibonacci numbers. A recursive function typically consists of a base case, which stops the recursion, and a recursive case that reduces the problem size with each call. While recursion can simplify code and enhance readability, it may also lead to increased memory usage and potential stack overflow issues if not managed carefully. Understanding recursion is crucial for programmers as it forms the foundation for many algorithms and data structures in computer science.

# aNALYSIS

The provided C code calculates the sum of the first specified number of multiples of a given number using recursion. It consists of a main function that handles user input and output, and a recursive function that sums the multiples. The program has a time complexity of O(n)*O*(*n*) and a space complexity of O(n)*O*(*n*) due to the call stack used for recursion. While the recursive approach makes the code simple and easy to read, it can lead to performance overhead and potential stack overflow for large inputs. Overall, this code effectively demonstrates how recursion can be applied to solve summation problems, though developers should consider its limitations in terms of performance and memory usage.

# Ideate

These real-life applications illustrate how the recursive summation of multiples can be used in budgeting, inventory management, event planning, fitness tracking, and education. By adapting this code to various contexts, users can leverage its functionality to solve everyday problems efficiently.

# Build of the code

*The code is based on RECURSIVE FUNCTION*

**CODE-**

**#include <stdio.h>**

**int sumofmultiples(int number, int count) {**

**if (count == 0) {**

**return 0;**

**}**

**return (number \* count) + sumofmultiples(number, count - 1);**

**}**

**int main() {**

**int number, count;**

**printf("Enter the number to find multiples of: ");**

**scanf("%d", &number);**

**printf("Enter the number of multiples to be added: ");**

**scanf("%d", &count);**

**int result = sumofmultiples(number, count);**

**printf("The sum of the first %d multiples of %d is: %d\n", count, number, result);**

**return 0;**

**}**

# TESTing

Testing the provided C code for calculating the sum of the first specified number of multiples using recursion involves several steps to ensure its correctness and robustness. First, various test cases should be created, including typical inputs (e.g., a positive integer for the number and a small positive count) to verify that the output matches expected results, such as checking if the sum of the first 5 multiples of 3 (i.e., 3, 6, 9, 12, 15) correctly yields 45. Edge cases should also be considered, such as inputting a count of zero (which should return 0) or very large counts to assess performance and check for stack overflow errors. Additionally, negative numbers can be tested to ensure the program handles them appropriately, ideally by validating inputs before processing. By systematically testing these scenarios, we can confirm that the code functions as intended and handles various input conditions gracefully.

# Implement

To implement testing for the C code that calculates the sum of the first specified number of multiples using recursion, we can create a series of test cases. First, we should verify typical inputs, such as calculating the sum of the first 5 multiples of 3, which should yield 45. We also need to test edge cases, including a count of zero, which should return 0, and a count of one, which should return the number itself. Additionally, we can check how the program handles negative inputs by ensuring it either validates these inputs or produces a meaningful error message. Finally, we should assess performance with large counts to confirm that the program runs efficiently without causing stack overflow errors. By executing these tests and validating the outputs against expected results, we can ensure the code is robust and reliable.

**GITHUB LINK-**

Made by – SRUJAL BHONDE

PARTH TOPALE

OMKAR PATIL