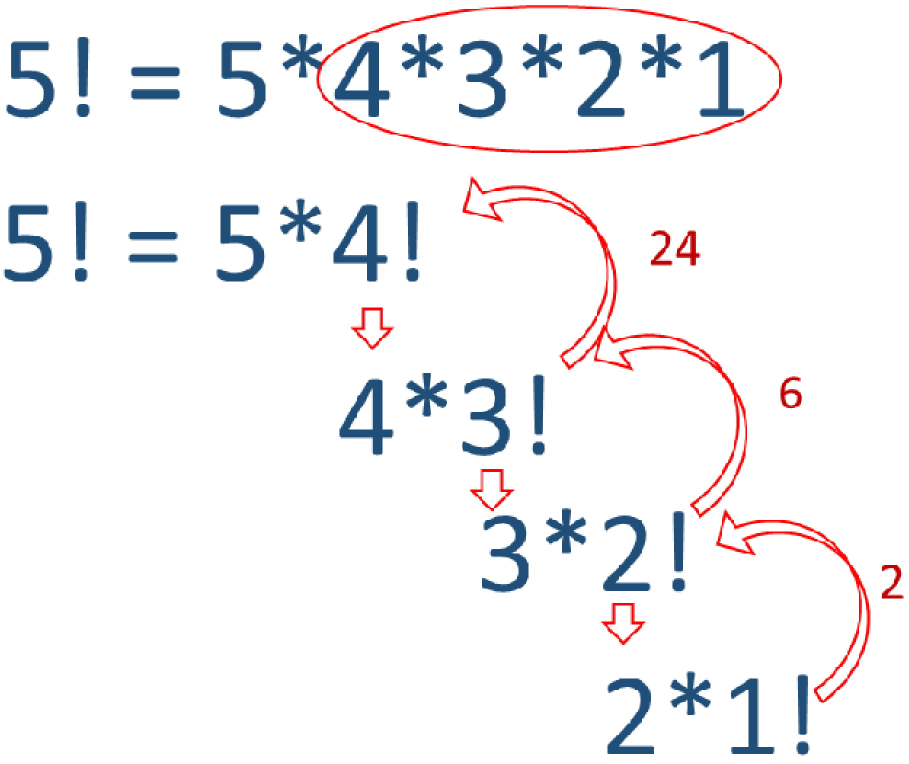
Recursion

Recursion is a programming technique where a function calls itself to solve a problem. The idea is to divide a complex problem into smaller, similar sub-problems until the base case is reached, which can be solved directly. Once the base case is resolved, the function "unwinds," combining the solutions of the sub-problems to solve the original problem.

Key Components of Recursion

1. **Base Case**: The condition where the function stops calling itself. Without a base case, recursion can lead to infinite calls and a stack overflow error.
2. **Recursive Case**: The part of the function where it calls itself with modified parameters to break the problem into smaller sub-problems.

Recursion tree for factorial of 5



```
#include <iostream>
using namespace std;


// Function to calculate factorial using recursion
long long factorial(int n) {
    if (n <= 1) return 1; // Base case
    return n * factorial(n - 1); // Recursive call
}


int main() {
    int n;
    cout << "Enter a number to calculate its factorial: ";
    cin >> n;


    if (n < 0) {
        cout << "Factorial is not defined for negative numbers." << endl;
    } else {
        cout << "Factorial of " << n << " is: " << factorial(n) << endl;
    }

    return 0;
}
```

Java

```
import java.util.Scanner;

public class Factorial {

    // Function to calculate factorial using recursion
    public static long factorial(int n) {
        if (n <= 1) return 1; // Base case
        return n * factorial(n - 1); // Recursive call
    }


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

        System.out.print("Enter a number to calculate its factorial: ");
        int n = scanner.nextInt();

        if (n < 0) {
            System.out.println("Factorial is not defined for negative numbers.");
        } else {
            System.out.println("Factorial of " + n + " is: " + factorial(n));
        }

        scanner.close();
    }
}
```

Direct and indirect recursion

Direct Recursion Example

In direct recursion, a function calls itself directly.

```
#include <iostream>
using namespace std;

void directRecursion(int n) {
    if (n == 0) return; // Base case
    cout << n << " ";
    directRecursion(n - 1); // Directly calling itself
}

int main() {
    cout << "Direct Recursion: ";
    directRecursion(5); // Example input
    return 0;
}
```

Java

```
public class DirectRecursion {
    static void directRecursion(int n) {
        if (n == 0) return; // Base case
        System.out.print(n + " ");
        directRecursion(n - 1); // Directly calling itself
    }

    public static void main(String[] args) {
        System.out.print("Direct Recursion: ");
        directRecursion(5); // Example input
    }
}
```

Indirect Recursion Example

In indirect recursion, a function calls another function, which then calls the first function.

```
#include <iostream>
using namespace std;

void functionA(int n);
void functionB(int n);

void functionA(int n) {
    if (n <= 0) return; // Base case
    cout << "A: " << n << " ";
    functionB(n - 1); // Calls functionB
}

void functionB(int n) {
    if (n <= 0) return; // Base case
    cout << "B: " << n << " ";
    functionA(n - 1); // Calls functionA
}

int main() {
    cout << "Indirect Recursion: ";
    functionA(5); // Example input
    return 0;
}
```

Java

```
public class IndirectRecursion {
    static void functionA(int n) {
        if (n <= 0) return; // Base case
        System.out.print("A: " + n + " ");
        functionB(n - 1); // Calls functionB
    }

    static void functionB(int n) {
        if (n <= 0) return; // Base case
        System.out.print("B: " + n + " ");
        functionA(n - 1); // Calls functionA
    }
    public static void main(String[] args) {
        System.out.print("Indirect Recursion: ");
        functionA(5); // Example input
    }
}
```

Tail recursion

In this approach, the recursive call is the **last operation** performed in the function.

```
#include <iostream>
using namespace std;

void displayTail(int start, int n) {
    if (start > n) return; // Base case
    cout << start << " ";  // Print before recursive call
    displayTail(start + 1, n); // Recursive call is the last operation
}

int main() {
    int n = 5;
    cout << "Displaying 1 to " << n << " using Tail Recursion: ";
    displayTail(1, n);
    return 0;
}
```

Java

```
public class DisplayTailRecursion {

    static void displayTail(int start, int n) {
        if (start > n) return; // Base case
        System.out.print(start + " "); // Print before recursive call
        displayTail(start + 1, n); // Recursive call is the last operation
    }

    public static void main(String[] args) {
        int n = 5;
        System.out.print("Displaying 1 to " + n + " using Tail Recursion: ");
        displayTail(1, n);
    }
}
```

Non Tail recursion

In this approach, the recursive call is **not** the last operation. There is a pending cout (in C++) or System.out.println (in Java) statement after the recursive call.

```
#include <iostream>
using namespace std;

void displayNonTail(int n) {
    if (n == 0) return; // Base case
    displayNonTail(n - 1); // Recursive call first
    cout << n << " ";    // Print after recursive call
}
```

```
int main() {
    int n = 5;
    cout << "Displaying 1 to " << n << " using Non-Tail Recursion: ";
    displayNonTail(n);
    return 0;
}
```

Java

```
public class DisplayNonTailRecursion {

    static void displayNonTail(int n) {
        if (n == 0) return; // Base case
        displayNonTail(n - 1); // Recursive call first
        System.out.print(n + " "); // Print after recursive call
    }

    public static void main(String[] args) {
        int n = 5;
        System.out.print("Displaying 1 to " + n + " using Non-Tail Recursion: ");
        displayNonTail(n);
    }
}
```

Tail recursion for factorial

```
#include <iostream>
using namespace std;

// Tail-recursive function for factorial
void tailRecursion(int n, int result) {
    if (n == 0) {
        cout << "Result: " << result << endl;
        return;
    }
    tailRecursion(n - 1, n * result); // Recursive call is the last operation
}

int main() {
    int n = 5;
    cout << "Tail Recursion: Calculating factorial of " << n << endl;
    tailRecursion(n, 1); // Start with the result as 1
    return 0;
}
```

Java

```
public class TailRecursion {

    // Tail-recursive function for factorial
    static void tailRecursion(int n, int result) {
        if (n == 0) {
            System.out.println("Result: " + result);
            return;
        }
        tailRecursion(n - 1, n * result); // Recursive call is the last operation
    }
    public static void main(String[] args) {
        int n = 5;
        System.out.println("Tail Recursion: Calculating factorial of " + n);
        tailRecursion(n, 1); // Start with the result as 1
    }
}
```

Non tail recursion for factorial

C++

```
#include <iostream>
using namespace std;

// Non-tail-recursive function for factorial
int nonTailRecursion(int n) {
    if (n == 0) return 1; // Base case
    return n * nonTailRecursion(n - 1); // Recursive call is followed by multiplication
}

int main() {
    int n = 5;
    cout << "Non-Tail Recursion: Calculating factorial of " << n << endl;
    cout << "Result: " << nonTailRecursion(n) << endl;
    return 0;
}
```

Java

```
public class NonTailRecursion {

    // Non-tail-recursive function for factorial
    static int nonTailRecursion(int n) {
        if (n == 0) return 1; // Base case
        return n * nonTailRecursion(n - 1); // Recursive call is followed by multiplication
    }

    public static void main(String[] args) {
        int n = 5;
        System.out.println("Non-Tail Recursion: Calculating factorial of " + n);
        System.out.println("Result: " + nonTailRecursion(n));
    }
}
```

Sum of 1 to n using recursion

```
#include <iostream>
using namespace std;

// Recursive function to calculate the sum from 1 to n
int sum(int n) {
    if (n == 0) return 0; // Base case
    return n + sum(n - 1); // Recursive case
}

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    if (n < 0) {
        cout << "Sum is not defined for negative numbers." << endl;
    } else {
        cout << "Sum of numbers from 1 to " << n << " is: " << sum(n) << endl;
    }

    return 0;
}
```

Java

```
import java.util.Scanner;
public class SumOfNumbers {
    // Recursive function to calculate the sum from 1 to n
    public static int sum(int n) {
        if (n == 0) return 0; // Base case
        return n + sum(n - 1); // Recursive case
    }

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

        System.out.print("Enter a number: ");
        int n = scanner.nextInt();

        if (n < 0) {
            System.out.println("Sum is not defined for negative numbers.");
        } else {
            System.out.println("Sum of numbers from 1 to " + n + " is: " + sum(n));
        }
        scanner.close();
    }
}
```

Q1. Print Numbers in Reverse (n to 1)

Print numbers from n to 1 using recursion.

```
#include <iostream>
using namespace std;

void printReverse(int n) {
    if (n == 0) return; // Base case
    cout << n << " ";   // Print current number
    printReverse(n - 1); // Recursive call
}

int main() {
    int n = 5;
    printReverse(n);
    return 0;
}
```

```
class Main {
    static void printReverse(int n) {
        if (n == 0) return; // Base case
        System.out.print(n + " "); // Print current number
        printReverse(n - 1); // Recursive call
    }

    public static void main(String[] args) {
        int n = 5;
        printReverse(n);
    }
}
```

2. Calculate Power

This is a great way to explain how recursion works with mathematical problems.

Problem:

Calculate x^n using recursion.

Code (C++):

```
#include <iostream>
using namespace std;

int power(int x, int n) {
    if (n == 0) return 1; // Base case: x^0 = 1
    return x * power(x, n - 1); // Recursive case
}

int main() {
    int x = 2, n = 3;
    cout << x << " raised to the power " << n << " is: " << power(x, n) << endl;
    return 0;
}
```

Java

```
class Main {
    static int power(int x, int n) {
        if (n == 0) return 1; // Base case: x^0 = 1
        return x * power(x, n - 1); // Recursive case
    }

    public static void main(String[] args) {
        int x = 2, n = 3;
        System.out.println(x + " raised to the power " + n + " is: " + power(x, n));
    }
}
```

3. Find Greatest Common Divisor (GCD)

This demonstrates recursion with a common real-world application (using the Euclidean algorithm).

Problem:

Find the GCD of two numbers a and b.

```
#include <iostream>
using namespace std;

int gcd(int a, int b) {
    if (b == 0) return a; // Base case
    return gcd(b, a % b); // Recursive case
}

int main() {
    int a = 56, b = 98;
    cout << "GCD of " << a << " and " << b << " is: " << gcd(a, b) << endl;
    return 0;
}
```

Java

```
class Main {

    static int gcd(int a, int b) {
        if (b == 0) return a; // Base case
        return gcd(b, a % b); // Recursive case
    }

    public static void main(String[] args) {
        int a = 56, b = 98;
        System.out.println("GCD of " + a + " and " + b + " is: " + gcd(a, b));
    }
}
```

4. Count the Number of Digits

This helps students see how recursion can simplify mathematical problems.

Problem:

Count the number of digits in an integer nnn.

Code (C++):

```
#include <iostream>
using namespace std;

int countDigits(int n) {
    if (n == 0) return 0; // Base case
    return 1 + countDigits(n / 10); // Recursive case
}

int main() {
    int n = 12345;
    cout << "Number of digits in " << n << " is: " << countDigits(n) << endl;
    return 0;
}
```

Java

```
class Main {
    static int countDigits(int n) {
        if (n == 0) return 0; // Base case
        return 1 + countDigits(n / 10); // Recursive case
    }

    public static void main(String[] args) {
        int n = 12345;
        System.out.println("Number of digits in " + n + " is: " + countDigits(n));
    }
}
```

```
public class CountDigits {

    static int countDigits(int n) {
        if (n == 0) return 0; // Base case
        return 1 + countDigits(n / 10); // Recursive case
    }

    public static void main(String[] args) {
        int n = 12345;
        System.out.println("Number of digits in " + n + " is: " + countDigits(n));
    }
}
```

Q**Palindrome Check**

Check whether a given string is a palindrome using recursion.

Key Concept:

Shows how recursion can be used for logical problems involving strings.

```
#include <iostream>
#include <string>
using namespace std;

bool isPalindrome(string str, int start, int end) {
    if (start >= end) return true; // Base case
    if (str[start] != str[end]) return false; // Mismatch found
    return isPalindrome(str, start + 1, end - 1); // Recursive call
}

int main() {
    string str = "radar";
    cout << str << " is " << (isPalindrome(str, 0, str.length() - 1) ? "a palindrome" : "not a palindrome") <<
endl;
    return 0;
}

class Main {
    static boolean isPalindrome(String str, int start, int end) {
        if (start >= end) return true; // Base case
        if (str.charAt(start) != str.charAt(end)) return false; // Mismatch found
        return isPalindrome(str, start + 1, end - 1); // Recursive call
    }

    public static void main(String[] args) {
        String str = "radar";
        System.out.println(str + " is " + (isPalindrome(str, 0, str.length() - 1) ? "a palindrome" : "not a
palindrome"));
    }
}
```

QPrint Fibonacci Numbers

This is a classic recursion example to demonstrate how recursion works but also highlight its inefficiency compared to iteration.

Problem:

Print the n-th Fibonacci number.

```
#include <iostream>
```

```
using namespace std;

int fibonacci(int n) {
   if (n <= 1) return n; // Base case
   return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case
}

int main() {
   int n = 5;
   cout << n << "-th Fibonacci number is: " << fibonacci(n) << endl;
   return 0;
}
```

```
public class Fibonacci {

   static int fibonacci(int n) {
      if (n <= 1) return n; // Base case
      return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case
   }

   public static void main(String[] args) {
      int n = 5;
      System.out.println(n + "-th Fibonacci number is: " + fibonacci(n));
   }
}
```

Array

Q **Problem Statement**

Write a program to find the maximum element in a 1D array using recursion. The program should accept an array of integers from the user and find the maximum value using a recursive function. The array size and elements should be provided by the user in the main function.

Test Cases

Test Case 1:

Input:

Array: [3, 1, 4, 1, 5, 9]

Output:

Maximum Element: 9

Test Case 2:

Input:

Array: [-10, -20, -30, -5, -15]

Output:

Maximum Element: -5

Test Case 3:

Input:

Array: [100, 200, 300, 400, 500]

Output:

Maximum Element: 500

C++ Implementation

```
#include <iostream>
#include <vector>
using namespace std;
```

```
// Recursive function to find the maximum element
int findMax(const vector<int>& arr, int n) {
    if (n == 1) {
        return arr[0];
    }
    return max(arr[n - 1], findMax(arr, n - 1));
}

int main() {
    int size;
    cin >> size;

    vector<int> arr(size);
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }

    int maxElement = findMax(arr, size);
    cout << maxElement << endl;

    return 0;
}
```

Java Implementation

```
import java.util.Scanner;

public class MaxElement {
    // Recursive function to find the maximum element
    public static int findMax(int[] arr, int n) {
        if (n == 1) {
            return arr[0];
        }
        return Math.max(arr[n - 1], findMax(arr, n - 1));
    }

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

        int size = scanner.nextInt();

        int[] arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = scanner.nextInt();
        }

        int maxElement = findMax(arr, size);
        System.out.println(maxElement);

        scanner.close();
    }
}
```

Problem Statement

Write a program to print a given string in reverse order using recursion. The program should accept a string from the user in the main function and use a recursive function to print the string in reverse order, one character at a time.

Test Cases

Test Case 1:

Input:

hello

Output:

olleh

Test Case 2:

Input:

OpenAI

Output:

IAnepO

Test Case 3:

Input:

12345

Output:

54321

C++ Implementation

```
#include <iostream>
#include <string>
using namespace std;

// Recursive function to print string in reverse order
void printReverse(const string& str, int index) {
   if (index < 0) {
      return;
   }
   cout << str[index];
   printReverse(str, index - 1);
}

int main() {
   string input;
   cin >> input;

   printReverse(input, input.size() - 1);
   cout << endl;

   return 0;
}
```

Java Implementation

```
import java.util.Scanner;

public class ReverseString {
   // Recursive function to print string in reverse order
   public static void printReverse(String str, int index) {
      if (index < 0) {
         return;
      }
```

```
        System.out.print(str.charAt(index));
        printReverse(str, index - 1);
    }

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

        String input = scanner.nextLine();

        printReverse(input, input.length() - 1);
        System.out.println();

        scanner.close();
    }
}
```

QWrite a program to reverse a given array using recursion. The program should accept an array of integers from the user in the main function, reverse it using a recursive function, and display the reversed array.

Test Cases

Test Case 1:

Input:

5

1 2 3 4 5

Output:

5 4 3 2 1

Test Case 2:

Input:

4

10 20 30 40

Output:

40 30 20 10

Test Case 3:

Input:

1

7

Output:

7

C++ Implementation

```
#include <iostream>
#include <vector>
using namespace std;

// Recursive function to reverse an array
void reverseArray(vector<int>& arr, int start, int end) {
    if (start >= end) {
        return;
    }
    swap(arr[start], arr[end]);
    reverseArray(arr, start + 1, end - 1);
}

int main() {
    int size;
    cin >> size;

    vector<int> arr(size);
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }

    reverseArray(arr, 0, size - 1);

    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Java Implementation

```
import java.util.Scanner;

public class ReverseArray {
    // Recursive function to reverse an array
    public static void reverseArray(int[] arr, int start, int end) {
        if (start >= end) {
            return;
        }
        // Swap elements
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        reverseArray(arr, start + 1, end - 1);
    }

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

        int size = scanner.nextInt();

        int[] arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = scanner.nextInt();
        }

        reverseArray(arr, 0, size - 1);

        for (int i = 0; i < size; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();

        scanner.close();
    }
}
```

Problem Statement

Write a program to calculate the sum of all elements in an array using recursion. The program should accept an array of integers from the user in the main function, compute the sum using a recursive function, and display the result.

Test Cases

Test Case 1:

Input:

Array: [1, 2, 3, 4, 5]

Output:

Sum of Elements: 15

Test Case 2:

Input:

Array: [10, 20, 30, 40]

Output:

Sum of Elements: 100

Test Case 3:

Input:

Array: [-5, 15, -10, 20]

Output:

Sum of Elements: 20

C++ Implementation

```
#include <iostream>
#include <vector>
using namespace std;

// Recursive function to calculate the sum of elements
int sumOfElements(const vector<int>& arr, int n) {
    if (n == 0) {
        return 0; // Base case: empty array
    }
    return arr[n - 1] + sumOfElements(arr, n - 1);
}

int main() {
    int size;
    //cout << "Enter the size of the array: ";
    cin >> size;

    vector<int> arr(size);
    //cout << "Enter the elements of the array: ";
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }

    int sum = sumOfElements(arr, size);
    cout << sum << endl;

    return 0;
}
```

Java Implementation

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

```
public class SumOfArray {
   // Recursive function to calculate the sum of elements
   public static int sumOfElements(int[] arr, int n) {
      if (n == 0) {
         return 0; // Base case: empty array
      }
      return arr[n - 1] + sumOfElements(arr, n - 1);
   }

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

      int size = scanner.nextInt();

      int[] arr = new int[size];
      for (int i = 0; i < size; i++) {
         arr[i] = scanner.nextInt();
      }

      int sum = sumOfElements(arr, size);
      System.out.println(sum);

      scanner.close();
   }
}
```

Problem Statement

Q Write a program to check whether a given array is sorted in non-decreasing order using recursion. The program should accept an array of integers from the user in the main function, and a recursive function should determine if the array is sorted. If the array is sorted, output true; otherwise, output false.

Test Cases

Test Case 1:

Input:

```
5
1 2 3 4 5
```

Output:

```
Is Sorted: true
```

Test Case 2:

```
4
10 20 15 40
```

Output:

```
Is Sorted: false
```

Test Case 3:

```
4
5 5 5 5
```

Output:

```
Is Sorted: true
```

C++ Implementation

```
#include <iostream>
#include <vector>
using namespace std;
```

```
// Recursive function to check if array is sorted
bool isSorted(const vector<int>& arr, int index) {
    if (index == arr.size() - 1 || arr.size() == 0) {
        return true; // Base case: single element or end of array
    }
    if (arr[index] > arr[index + 1]) {
        return false;
    }
    return isSorted(arr, index + 1);
}

int main() {
    int size;
    cin >> size;

    vector<int> arr(size);
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }

    bool result = isSorted(arr, 0);
    cout << "Is Sorted: " << (result ? "true" : "false") << endl;

    return 0;
}
```

Java Implementation

```
import java.util.Scanner;

public class ArraySortedCheck {
    // Recursive function to check if array is sorted
    public static boolean isSorted(int[] arr, int index) {
        if (index == arr.length - 1 || arr.length == 0) {
            return true; // Base case: single element or end of array
        }
        if (arr[index] > arr[index + 1]) {
            return false;
        }
        return isSorted(arr, index + 1);
    }

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

        int size = scanner.nextInt();

        int[] arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = scanner.nextInt();
        }

        boolean result = isSorted(arr, 0);
        System.out.println("Is Sorted: " + (result ? "true" : "false"));
        scanner.close();
    }
}
```

Problem Statement

Write a program to search for a given element in an array using recursion. The program should accept an array of integers and a target value from the user in the main function. A recursive function should return the index of the first occurrence of the target element in the array. If the target element is not found, the function should return -1.

Test Cases

Test Case 1:

Input:

```
5
10 20 30 40 50
30
```

Output:

```
2
```

Test Case 2:

Input:

```
5
5 8 7 8 9
8
```

Output:

```
1
```

Test Case 3:

5

```
1 2 3 4 5
10
```

Output:

```
-1
```

C++ Implementation

```
#include <iostream>
#include <vector>
using namespace std;

// Recursive function to search for an element
int searchElement(const vector<int>& arr, int index, int target) {
    if (index == arr.size()) {
        return -1; // Base case: reached end of the array
    }
    if (arr[index] == target) {
        return index; // Element found
    }
    return searchElement(arr, index + 1, target);
}

int main() {
    int size, target;
    cin >> size;

    vector<int> arr(size);
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }

    cin >> target;

    int result = searchElement(arr, 0, target);
    cout << result << endl;
        return 0;
}
```

Java Implementation

```
import java.util.Scanner;

public class SearchElement {
    // Recursive function to search for an element
    public static int searchElement(int[] arr, int index, int target) {
        if (index == arr.length) {
            return -1; // Base case: reached end of the array
        }
        if (arr[index] == target) {
            return index; // Element found
        }
        return searchElement(arr, index + 1, target);
    }

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

        int size = scanner.nextInt();

        int[] arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = scanner.nextInt();
        }

        int target = scanner.nextInt();

        int result = searchElement(arr, 0, target);
        System.out.println(result);

        scanner.close();
    }
}
```

Problem Statement

Write a program to check whether a given string is a palindrome using recursion. A string is considered a palindrome if it reads the same backward as forward. The program should accept a string from the user and use a recursive function to check if the string is a palindrome.

Test Cases

Test Case 1:

Input:

String: radar

Output:

The string is a palindrome.

Test Case 2:

Input:
String: hello
Output:
The string is not a palindrome.
Test Case 3:
Input:
String: abba
Output:
The string is a palindrome.

C++ Implementation

```
#include <iostream>
#include <string>
using namespace std;

// Recursive function to check if a string is a palindrome
bool isPalindrome(const string& str, int start, int end) {
    if (start >= end) {
        return true; // Base case: all characters checked
    }
    if (str[start] != str[end]) {
        return false; // Characters do not match
    }
    return isPalindrome(str, start + 1, end - 1);
}

int main() {
    string input;
    cin >> input;

    bool result = isPalindrome(input, 0, input.size() - 1);
    if (result) {
        cout << "The string is a palindrome." << endl;
    } else {
        cout << "The string is not a palindrome." << endl;
    }

    return 0;
}
```

Java Implementation

```
import java.util.Scanner;

public class PalindromeCheck {
    // Recursive function to check if a string is a palindrome
    public static boolean isPalindrome(String str, int start, int end) {
        if (start >= end) {
            return true; // Base case: all characters checked
        }
        if (str.charAt(start) != str.charAt(end)) {
            return false; // Characters do not match
        }
        return isPalindrome(str, start + 1, end - 1);
    }

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

        String input = scanner.nextLine();

        boolean result = isPalindrome(input, 0, input.length() - 1);
        if (result) {
            System.out.println("The string is a palindrome.");
        } else {
            System.out.println("The string is not a palindrome.");
        }

        scanner.close();
    }
}
```

Problem Statement

Write a program to count the number of vowels in a given string using recursion. The program should accept a string from the user and use a recursive function to count the vowels. The vowels are a, e, i, o, u (case-insensitive).

Test Cases

Test Case 1:

Input:

```
hello
```

Output:

```
2
```

Test Case 2:

Input:

```
OpenAI
```

Output:

```
3
```

Test Case 3:

Input:

```
xyz
```

Output:

```
0
```

C++ Implementation

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

// Recursive function to count vowels in a string
int countVowels(const string& str, int index) {
    if (index == str.size()) {
        return 0; // Base case: end of string
    }
    char ch = tolower(str[index]); // Convert character to lowercase
    int count = (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') ? 1 : 0;
    return count + countVowels(str, index + 1);
}

int main() {
    string input;
    cin >> input;

    int vowelCount = countVowels(input, 0);
    cout << vowelCount << endl;

    return 0;
}
```

Java Implementation

```
import java.util.Scanner;

public class VowelCount {
    // Recursive function to count vowels in a string
    public static int countVowels(String str, int index) {
        if (index == str.length()) {
            return 0; // Base case: end of string
        }
        char ch = Character.toLowerCase(str.charAt(index)); // Convert character to lowercase
        int count = (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') ? 1 : 0;
        return count + countVowels(str, index + 1);
    }

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

        String input = scanner.nextLine();

        int vowelCount = countVowels(input, 0);
        System.out.println(vowelCount);

        scanner.close();
    }
}
```

Problem Statement

Write a program to remove consecutive duplicate characters from a string using recursion. The program should accept a string from the user in the main function and use a recursive function to return a string where all consecutive duplicate characters are removed.

Test Cases

Test Case 1:

Input:

aaabbccdee

Output:

abcde

Test Case 2:

Input:

aabbaa

Output:
aba
Test Case 3:
Input:
abcd
Output:
abcd

C++ Implementation

```
#include <iostream>
#include <string>
using namespace std;

// Recursive function to remove consecutive duplicate characters
string removeDuplicates(const string& str, int index = 0) {
    if (index == str.size() - 1 || str.empty()) {
        return str.substr(index, 1); // Base case: return the last character
    }
    if (str[index] == str[index + 1]) {
        return removeDuplicates(str, index + 1); // Skip duplicate character
    }
    return str[index] + removeDuplicates(str, index + 1);
}

int main() {
    string input;
    cin >> input;

    string result = removeDuplicates(input);
    cout  << result << endl;

    return 0;
}
```

Java Implementation

```
import java.util.Scanner;

public class RemoveDuplicates {
    // Recursive function to remove consecutive duplicate characters
    public static String removeDuplicates(String str, int index) {
        if (index == str.length() - 1 || str.isEmpty()) {
            return str.substring(index); // Base case: return the last character
        }
        if (str.charAt(index) == str.charAt(index + 1)) {
            return removeDuplicates(str, index + 1); // Skip duplicate character
        }
        return str.charAt(index) + removeDuplicates(str, index + 1);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        String result = removeDuplicates(input, 0);
        System.out.println(result);
        scanner.close();
    }
}
```