1. Problem: Matrix Multiplication Problem Statement: Write a C program to multiply two matrices entered by the user. The program should: Read the dimensions (rows and columns) of two matrices, ensuring the number of columns in the first matrix equals the number of rows in the second matrix. Read the elements of both matrices from the user. Perform matrix multiplication. Print the resulting matrix.

Code:
```c
#include <stdio.h>

int main() {
    int m1, n1, m2, n2;

    // Read dimensions of the first matrix
    printf("Enter rows and columns for the first matrix: ");
    scanf("%d %d", &m1, &n1);

    // Read dimensions of the second matrix
    printf("Enter rows and columns for the second matrix: ");
    scanf("%d %d", &m2, &n2);

    // Check if the matrices can be multiplied
    if (n1 != m2) {
        printf("Matrix multiplication is not possible. Number of columns of the first matrix must equal
the number of rows of the second matrix.\n");
        return 1;
    }

    int matrix1[m1][n1], matrix2[m2][n2], result[m1][n2];

    // Input elements of the first matrix
    printf("Enter elements of the first matrix:\n");
    for (int i = 0; i < m1; i++) {
        for (int j = 0; j < n1; j++) {
            scanf("%d", &matrix1[i][j]);
        }
```

```
}

// Input elements of the second matrix
printf("Enter elements of the second matrix:\n");
for (int i = 0; i < m2; i++) {
    for (int j = 0; j < n2; j++) {
        scanf("%d", &matrix2[i][j]);
    }
}

// Initialize the result matrix with zeros
for (int i = 0; i < m1; i++) {
    for (int j = 0; j < n2; j++) {
        result[i][j] = 0;
    }
}

// Perform matrix multiplication
for (int i = 0; i < m1; i++) {
    for (int j = 0; j < n2; j++) {
        for (int k = 0; k < n1; k++) {
            result[i][j] += matrix1[i][k] * matrix2[k][j];
        }
    }
}

// Print the result matrix
printf("Resultant matrix:\n");
for (int i = 0; i < m1; i++) {
    for (int j = 0; j < n2; j++) {
        printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

2. Problem: Check if a Number is Armstrong

Problem Statement:
Write a C program to check if a given number is an Armstrong number. A number is called an Armstrong number if the sum of its digits raised to the power of the number of digits equals the number itself.

For example:

- **153 is an Armstrong number because 13+53+33=1531^3 + 5^3 + 3^3 = 15313+53+33=153.**
- **9474 is an Armstrong number because 94+44+74+44=94749^4 + 4^4 + 7^4 + 4^4 = 947494+44+74+44=9474.**

Code:

```
#include <stdio.h>
#include <math.h>

// Function to calculate the number of digits in a number
int countDigits(int num) {
    int count = 0;
    while (num != 0) {
        num /= 10;
        count++;
    }
    return count;
}

// Function to check if a number is an Armstrong number
int isArmstrong(int num) {
    int originalNum = num;
    int sum = 0;
    int numDigits = countDigits(num);

    while (num != 0) {
        int digit = num % 10; // Extract the last digit
        sum += pow(digit, numDigits); // Add digit^numDigits to sum
        num /= 10; // Remove the last digit
    }

    return sum == originalNum; // Check if sum equals the original number
}

int main() {
    int number;

    // Read an integer from the user
    printf("Enter a number: ");
    scanf("%d", &number);

    // Check if the number is an Armstrong number
    if (isArmstrong(number)) {
        printf("%d is an Armstrong number.\n", number);
    } else {
        printf("%d is not an Armstrong number.\n", number);
    }

    return 0;
}
```

3. Problem: Reverse a String Using Recursion

Problem Statement:
Write a C program to reverse a string using recursion. The program should define a function called reverseString() that takes a string and its length as parameters. The function should:

1. **Use recursion to reverse the string in place.**
2. **Stop the recursion once it has processed half of the string.**

In the main() function:

1. **Read a string from the user.**
2. **Call reverseString() to reverse the string.**
3. **Print the reversed string.**

Code: #include <stdio.h>

```
#include <string.h>

// Recursive function to reverse a string in place
void reverseString(char str[], int start, int end) {
  if (start >= end) {
    // Base case: stop when start index meets or exceeds end index
    return;
  }

  // Swap characters at start and end positions
  char temp = str[start];
  str[start] = str[end];
  str[end] = temp;

  // Recursive call to reverse the rest of the string
  reverseString(str, start + 1, end - 1);
}

int main() {
  char str[100]; // Declare a string with a maximum size of 100 characters

  // Read a string from the user
  printf("Enter a string: ");
  fgets(str, sizeof(str), stdin);
```

```
    // Remove the newline character if present
    str[strcspn(str, "\n")] = '\0';

    int length = strlen(str); // Get the length of the string

    // Call the recursive function to reverse the string
    reverseString(str, 0, length - 1);

    // Print the reversed string
    printf("Reversed string: %s\n", str);

    return 0;
}
```

4. Problem: Count Function Calls Problem Statement: Write a C program that defines a function called countCalls(). The function should: Use a static variable to count how many times the function has been called. Print the number of times the function has been called each time it is invoked. In the main() function: in c language Call countCalls() multiple times and verify the count is updated correctly.

Code: #include <stdio.h>

```
// Function to count the number of times it has been called
void countCalls() {
    static int callCount = 0; // Static variable to retain value across calls
    callCount++; // Increment the call count
    printf("countCalls has been called %d times.\n", callCount);
}

int main() {
    // Call countCalls multiple times to verify the count
    countCalls();
    countCalls();
    countCalls();
    countCalls();


    return 0;
}
```

5. Problem: Find the Longest Palindromic Substring

Problem Statement:
Write a C program to find the longest palindromic substring in a given string. A palindrome is a sequence of characters that reads the same forward and backward.

Requirements:

1. Read a string from the user.
2. Use a function to determine if a substring is a palindrome.
3. Iterate through all possible substrings to find the longest one that is a palindrome.
4. Print the longest palindromic substring and its length.

```
#include <stdio.h>
#include <string.h>


// Function to check if a substring is a palindrome
int isPalindrome(char str[], int start, int end) {
    while (start < end) {
        if (str[start] != str[end]) {
            return 0; // Not a palindrome
        }
        start++;
        end--;
    }
    return 1; // Is a palindrome
}


// Function to find the longest palindromic substring
void findLongestPalindrome(char str[]) {
    int n = strlen(str);
    int maxLength = 0;
    int start = 0;

    // Iterate through all possible substrings
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            // Check if the current substring is a palindrome
            if (isPalindrome(str, i, j)) {
                int length = j - i + 1;
                if (length > maxLength) {
                    maxLength = length;
                    start = i;
                }
            }
        }
    }

    // Print the longest palindromic substring
    printf("Longest palindromic substring: ");
    for (int i = start; i < start + maxLength; i++) {
        printf("%c", str[i]);
    }
    printf("\nLength: %d\n", maxLength);
}

int main() {
    char str[100];

    // Read a string from the user
    printf("Enter a string: ");
    scanf("%s", str);
    // Find and print the longest palindromic substring
    findLongestPalindrome(str);

    return 0;
}
```