Day 15 programs

1.Factorial Calculation: Write a recursive function to calculate the factorial of a given non-negative integer n.

Without using pointers

```
#include <stdio.h>

int factorial(int n);

int main() {
    int n = 5;
    printf("Factorial of %d = %d\n", n, factorial(n));
    return 0;
}

int factorial(int n) {
    if (n == 0 || n == 1) {  // Base condition
        return 1;
    }
    return n * factorial(n - 1);  // Recursive call
}
```

Using Pointers

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

```
int factorial(int* n);


int main() {
    int n = 5;
    printf("Factorial of %d = %d\n", n, factorial(&n));
    return 0;
}


int factorial(int* n) {
    if (*n == 0 || *n == 1) {  // Base condition
        return 1;
    }
    int current = *n;  // Store the current value of n
    (*n)--;
    return current * factorial(n);  // Recursive call
}
```

2.Fibonacci Series: Create a recursive function to find the nth term of the Fibonacci series.

Without using pointers

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


int fibonacci(int n);


int main() {
    int n = 4;
    printf("Fibonacci term %d = %d\n", n, fibonacci(n));
    return 0;
}

int fibonacci(int n) {
    if (n == 0) {  // Base case 1
        return 0;
    } else if (n == 1) {  // Base case 2
        return 1;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);  // Recursive case
}
```

Using Pointers

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

int fibonacci(int *n);

int main() {
    int n = 4;
    printf("Fibonacci term %d = %d\n", n, fibonacci(&n));
    return 0;
}

int fibonacci(int *n) {
    if (*n == 0) {  // Base case 1
        return 0;
    } else if (*n == 1) {  // Base case 2
```

```
        return 1;
    }
    int current=*n;
    int first=current-1;
    int second=current-2;
    return fibonacci(&first) + fibonacci(&second);  // Recursive case
}
```

3.Sum of Digits: Implement a recursive function to calculate the sum of the digits of a given positive integer.

Without using pointers

```
#include <stdio.h>

int sumOfDigits(int n);

int main() {
    int n = 1098;
    printf("Sum of digits of %d = %d\n", n, sumOfDigits(n));
    return 0;
}

int sumOfDigits(int n) {
    if (n == 0) {  // Base case
        return 0;
    }
    return (n % 10) + sumOfDigits(n / 10);  // Recursive case
}
```

Using Pointers

```
#include <stdio.h>


int sumOfDigits(int* n);


int main() {
    int n = 1098;  // Example input
    printf("Sum of digits of %d = %d\n", n, sumOfDigits(&n));
    return 0;
}


int sumOfDigits(int* n) {
    if (*n == 0) {  // Base case
        return 0;
    }
    int digit = *n % 10;  // Extract the last digit
    *n = *n / 10;         // Remove the last digit
    return digit + sumOfDigits(n);  // Recursive case
}
```

4.Reverse a String: Write a recursive function to reverse a string.

Without using pointers

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

```
void reverseString(char str[], int start, int end);

int main() {
    char str[] = "Hello, World!";  // Input string
    int length = strlen(str);

    reverseString(str, 0, length - 1);  // Call the reverse function
    printf("Reversed string: %s\n", str);  // Output the reversed string

    return 0;
}

void reverseString(char str[], int start, int end) {
    if (start >= end) {  // Base case: If pointers have met or crossed
        return;
    }

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

    // Recursive call with reduced range
    reverseString(str, start + 1, end - 1);
}
```

Using Pointers

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

void reverseString(char* str, int start, int end);

int main() {
    char str[] = "Hello, World!";  // Input string
    int length = strlen(str);

    reverseString(str, 0, length - 1);  // Call the reverse function
    printf("Reversed string: %s\n", str);  // Output the reversed string

    return 0;
}

void reverseString(char* str, int start, int end) {
    if (start >= end) {  // Base case: If pointers have met or crossed
        return;
    }

    // Swap characters at start and end using pointer arithmetic
    char temp = *(str + start);
    *(str + start) = *(str + end);
    *(str + end) = temp;

    // Recursive call with reduced range
    reverseString(str, start + 1, end - 1);
}
```

5.Power Calculation: Develop a recursive function to calculate the power of a number x raised to n.

Without using pointers

```
#include <stdio.h>


int power(int x, int n);


int main() {
    int x = 2, n = 3;  // Example: 2^3
    printf("Result: %d\n", power(x, n));
    return 0;
}

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

Using Pointers

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

```
int power(int* x, int n);


int main() {
    int x = 2, n = 3;  // Example: 2^3
    printf("Result: %d\n", power(&x, n));
    return 0;
}


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

6.Count Occurrences of a Character: Develop a recursive function to count the number of times a specific character appears in a string.

Without using pointers

```
#include <stdio.h>


int countOccurrences(char str[], char ch, int index);


int main() {
    char str[] = "hello";
    char ch = 'l';
    int count = countOccurrences(str, ch, 0);

    printf("The character '%c' appears %d times in \"%s\".\n", ch, count, str);
    return 0;
}

int countOccurrences(char str[], char ch, int index) {
    if (str[index] == '\0') {
        return 0;
    }
    // Check if the current character matches and add 1 if true
    int count = (str[index] == ch) ? 1 : 0;
    return count + countOccurrences(str, ch, index + 1);  // Recursive call for the next character
}
```

Using Pointers

```
#include <stdio.h>

int countOccurrences(char* str, char ch);

int main() {
    char str[] = "hello";
    char ch = 'l';
    int count = countOccurrences(str, ch);

    printf("The character '%c' appears %d times in \"%s\".\n", ch, count, str);
    return 0;
}


int countOccurrences(char* str, char ch) {
    if (*str == '\0') {
        return 0;
    }
    // Check if the current character matches and add 1 if true
    int count = (*str == ch) ? 1 : 0;
    return count + countOccurrences(str + 1, ch);  // Recursive call to the next character
}
```

7.Palindrome Check: Create a recursive function to check if a given string is a palindrome.

Without using pointers

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


bool isPalindrome(char str[], int start, int end) {
    if (start >= end) {
        return true;
    }
    if (str[start] != str[end]) {
        return false;
    }
    return isPalindrome(str, start + 1, end - 1);
}


int main() {
    char str[] = "radar";
    if (isPalindrome(str, 0, 4)) {
        printf("\"%s\" is a palindrome.\n", str);
    } else {
        printf("\"%s\" is not a palindrome.\n", str);
    }
    return 0;
}
```

Using Pointers

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

bool isPalindrome(char* str, int start, int end) {
    if (start >= end) {
        return true;
    }
    if (*(str + start) != *(str + end)) {
        return false;
    }
    return isPalindrome(str, start + 1, end - 1);
}

int main() {
    char str[] = "radar";
    if (isPalindrome(str, 0, 4)) {
        printf("\"%s\" is a palindrome.\n", str);
    } else {
        printf("\"%s\" is not a palindrome.\n", str);
    }
    return 0;
}
```

8.String Length: Write a recursive function to calculate the length of a given string without using any library functions.

Without using pointers

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

int stringLength(char str[], int index);

int main() {
    char str[] = "Hello World!";
    printf("The length of the string is: %d\n", stringLength(str, 0));
    return 0;
}

int stringLength(char str[], int index) {
    if (str[index] == '\0') {
        return 0;
    }
    return 1 + stringLength(str, index + 1);
}
```

Using Pointers

```
#include <stdio.h>


int stringLength(char* str);


int main() {
   char str[] = "Hello World!";
   printf("The length of the string is: %d\n", stringLength(str));
   return 0;
}


int stringLength(char* str) {
   if (*str == '\0') {
      return 0;
   }
   return 1 + stringLength(str + 1);
}
```

9.Check for Prime Number: Implement a recursive function to check if a given number is a prime number.

Without using pointers

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


bool isPrime(int n);


int main() {
    int n = 13;
    if (isPrime(n)) {
        printf("%d is a prime number.\n", n);
    } else {
        printf("%d is not a prime number.\n", n);
    }
    return 0;
}

bool isPrime(int n) {
    if (n <= 1) {
        return false;
    }
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
```

Using Pointers

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

bool isPrime(int* n);
```

```
int main() {
    int n = 13;
    if (isPrime(&n)) {
        printf("%d is a prime number.\n", n);
    } else {
        printf("%d is not a prime number.\n", n);
    }
    return 0;
}

bool isPrime(int* n) {
    if (*n <= 1) {
        return false;
    }
    for (int i = 2; i * i <= *n; i++) {
        if (*n % i == 0) {
            return false;
        }
    }
    return true;
}
```

10.Print Numbers in Reverse: Create a recursive function to print the numbers from n down to 1 in reverse order.

Without using pointers

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

```c
void printReverse(int n);

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

void printReverse(int n) {
    if (n == 0) {  // Base case
        return;
    }
    printf("%d ", n);  // Print the current number
    printReverse(n - 1);  // Recursive call with n-1
}
```

Using Pointers

```c
---------------
#include <stdio.h>

void printReverse(int* n);

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

```
void printReverse(int* n) {
    if (*n == 0) {  // Base case
        return;
    }
    printf("%d ", *n);  // Print the current number using pointer dereferencing
    (*n)--;  // Decrease n using pointer
    printReverse(n);  // Recursive call with updated n
}
```

11.Array Sum: Write a recursive function to find the sum of all elements in an array of integers.

Without using pointers

```
#include <stdio.h>

int arraySum(int arr[], int n);

int main() {
    int arr[] = {1, 2, 3, 4, 5};  // Example array
    int n = 5;  // Size of the array
    int sum = arraySum(arr, n);
    printf("The sum of the array elements is: %d\n", sum);
    return 0;
}

int arraySum(int arr[], int n) {
    if (n == 0) {
        return 0;  // Base case: No elements left
    }
    return arr[n - 1] + arraySum(arr, n - 1);  // Recursive call with reduced array size
}
```

Using Pointers

```
#include <stdio.h>

int arraySum(int* arr, int n);

int main() {
    int arr[] = {1, 2, 3, 4, 5};  // Example array
    int n = 5;  // Size of the array
    int sum = arraySum(arr, n);
    printf("The sum of the array elements is: %d\n", sum);
    return 0;
}

int arraySum(int* arr, int n) {
    if (n == 0) {
        return 0;  // Base case: No elements left
    }
    return *(arr + n - 1) + arraySum(arr, n - 1);  // Recursive call with pointer arithmetic
}
```

12.Permutations of a String: Develop a recursive function to generate all possible permutations of a given string.

Without using pointers

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

void permute(char str[], int l, int r);

int main() {
    char str[] = "AB";  // Simplified example string
    int n = strlen(str);
    permute(str, 0, n - 1);
    return 0;
}

void permute(char str[], int l, int r) {
    if (l == r) {
        printf("%s\n", str);  // Print permutation
    } else {
        for (int i = l; i <= r; i++) {
            // Swap and recurse
            char temp = str[l];
            str[l] = str[i];
            str[i] = temp;

            permute(str, l + 1, r);
            // Swap back
            temp = str[l];
            str[l] = str[i];
            str[i] = temp;
        }
    }
}
```

Using Pointers

```
#include <stdio.h>

void permute(char* str, int l, int r);

int main() {
    char str[] = "AB";  // Simplified example string
    permute(str, 0, 1);  // Length of "AB" is 2
    return 0;
}

void permute(char* str, int l, int r) {
    if (l == r) {
        printf("%s\n", str);  // Print permutation
    } else {
        for (int i = l; i <= r; i++) {
            // Swap using pointers and recurse
            char temp = *(str + l);
            *(str + l) = *(str + i);
            *(str + i) = temp;

            permute(str, l + 1, r);

            // Swap back
            temp = *(str + l);
            *(str + l) = *(str + i);
            *(str + i) = temp;
        }
    }
}
```

13.Greatest Common Divisor (GCD): Create a recursive function to find the GCD of two given integers using the Euclidean algorithm.

Without using pointers

```
#include <stdio.h>

int gcd(int a, int b);

int main() {
    int a = 56, b = 98;  // Example: GCD of 56 and 98
    printf("GCD: %d\n", gcd(a, b));
    return 0;
}

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

Using Pointers

```
#include <stdio.h>

int gcd(int* a, int* b);

int main() {
    int a = 56, b = 98;  // Example: GCD of 56 and 98
    printf("GCD: %d\n", gcd(&a, &b));
    return 0;
}

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