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

```
Sol: #include <stdio.h>
int fact(int n);
int main()
{
    int num, f;
    printf("Enter the number: ");
    scanf("%d", &num);
    f = fact(num);
    printf("Factorial of %d = %d\n", num, f);
    return 0;
}
int fact(int n)
{
    if(n)
        return n * fact(n - 1);
    else
        return 1;
}
```

With pointers:

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

```
int fact(int *n);
int main()
{
    int num, f;
    printf("Enter the number: ");
    scanf("%d", &num);
    f = fact(&num);
    printf("Factorial = %d\n", f);
    return 0;
}
int fact(int *n)
{
    if (*n > 0)
    {
        int f1 = *n;
        (*n)--;
        return f1 * fact(n);
    }
    else
        return 1;
}
```

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

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

int fibonacci(int n);

int main() {
    int n;
    printf("Enter the number: ");
    scanf("%d", &n);

    if (n < 0) {
        printf("Fibonacci is not defined for negative integers.\n");
    } else {
        int result = fibonacci(n);
        printf("Fibonacci number %d is %d\n", n, result);
    }

    return 0;
}


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

//with pointers
#include <stdio.h>
int fibonacci(int *n);

int main() {
    int num;
    printf("Enter the number: ");
    scanf("%d", &num);
    if (num < 0) {
        printf("Fibonacci is not defined for negative integers.\n");
    } else {
        int result = fibonacci(&num);  // Pass the address of num to the recursive function
        printf("Fibonacci number %d is %d\n", num, result);
    }

    return 0;
}

int fibonacci(int *n) {
    if (*n == 0) {
        return 0;
    } else if (*n == 1) {
        return 1;
    } else {
        int n_minus_1 = *n - 1;  // Calculate n-1
        int n_minus_2 = *n - 2;  // Calculate n-2
        return fibonacci(&n_minus_1) + fibonacci(&n_minus_2);
    }
}
```

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

Sol:

```
#include <stdio.h>

int sumOfDigits(int n);

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num < 0) {
        printf(" enter a positive integer.\n");
    } else {
        int result = sumOfDigits(num);
        printf("Sum of digits of %d is %d\n", num, result);
    }

    return 0;
}

int sumOfDigits(int n) {
    if (n == 0) {
        return 0;
    } else {
        return (n % 10) + sumOfDigits(n / 10);  // Adding the last digit to the sum of remaining digits
    }
}*/

// with pointers
#include <stdio.h>

int sumOfDigits(int *n);

int main() {
    int num, result;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num < 0) {
        printf("enter a positive integer.\n");
    } else {
        result = sumOfDigits(&num);
        printf("Sum of digits = %d\n", result);
    }

    return 0;
}
int sumOfDigits(int *n) {
    if (*n == 0) {
        return 0;
    } else {
        int lastDigit = *n % 10;
        *n = *n / 10;
        return lastDigit + sumOfDigits(n);
    }
}
```

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

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

#include <string.h>

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

```
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);  // Read the string
    int len = strlen(str);
    reverseString(str, 0, len - 1);  // Call recursive function to reverse the string
 printf("Reversed string: %s\n", str);
  return 0;
}
void reverseString(char *str, int start, int end) {
    if (start >= end) {
        return;
    }
    // Swap characters at start and end
    char temp = str[start];
    str[start] = str[end];
    str[end] = temp;
  // Recursive call to reverse the inner substring
    reverseString(str, start + 1, end - 1);
}*/
```

```
//with pointers
#include <stdio.h>
#include <string.h>  // Include the string.h header for strlen()
// Function to reverse a string using recursion and pointers
void reverseString(char *str, char *start, char *end);
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    reverseString(str, str, str + strlen(str) - 1);
  printf("Reversed string: %s\n", str);
  return 0;
}
void reverseString(char *str, char *start, char *end) {
    if (start >= end) {
        return;
    }
 // Swap characters at start and end
    char temp = *start;
    *start = *end;
    *end = temp;
    // Recursive call with the next start and previous end pointers
    reverseString(str, start + 1, end - 1);
}
```

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

```
Sol: #include <stdio.h>
int power(int x, int n);
int main() {
    int x, n;
    printf("Enter base number: ");
    scanf("%d", &x);
    printf("Enter exponent: ");
    scanf("%d", &n);
     int result = power(x, n);
    printf("%d raised to the power %d is: %d\n", x, n, result);
 return 0;
}
int power(int x, int n) {
    if (n == 0) {
        return 1;
    } else {
        return x * power(x, n - 1);  // Recursive case: x^n = x * x^(n-1)
    }
}*/
#include <stdio.h>
// Recursive function to calculate power using pointers
int power(int *x, int *n);
int main() {
    int x, n;
    printf("Enter base number: ");
    scanf("%d", &x);
    printf("Enter exponent: ");
    scanf("%d", &n);
     int result = power(&x, &n);  // Passing the address of x and n
    printf("%d raised to the power %d is: %d\n", x, n, result);
    return 0;
}
// Recursive function to calculate power using pointers
int power(int *x, int *n) {
    if (*n == 0) {
        return 1;
    } else {
        int temp = *n - 1;  // Decrement the exponent
        return *x * power(x, &temp);  // Recursive case: x^n = x * x^(n-1)
    }
}
```

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

```
Sol: #include <stdio.h>
int gcd(int x, int y);

int main() {
    int x, y;
    printf("Enter two numbers: ");
    scanf("%d %d", &x, &y);

    int result = gcd(x, y);
    printf("The GCD of %d and %d is: %d\n", x, y, result);

    return 0;
}
int gcd(int x, int y) {
    if (y == 0) {
        return x;
    } else {
        return gcd(y, x % y);
    }
}*/

//with pointers
#include <stdio.h>
int gcd(int *x, int *y);

int main() {
    int x, y;
    printf("Enter two numbers: ");
    scanf("%d %d", &x, &y);

    int result = gcd(&x, &y);
    printf("The GCD of %d and %d is: %d\n", x, y, result);

    return 0;
}
int gcd(int *x, int *y) {
    if (*y == 0) {
        return *x;

    } else {

        int remainder = *x % *y;  // Calculate the remainder

        return gcd(y, &remainder);  // Recursive case: GCD(y, remainder)

    }

}
```

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

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

int countOccurrences(char *str, char target);



int main() {

    char str[100], target;

    printf("Enter a string: ");

    scanf("%s", str);

    printf("Enter a character to count: ");

    scanf(" %c", &target);



    int result = countOccurrences(str, target);

    printf("The character '%c' appears %d times in the string.\n", target, result);



    return 0;
```

```
}
int countOccurrences(char *str, char target) {
    if (*str == 0) {
        return 0;
    } else {
        if (*str == target) {
            return 1 + countOccurrences(str + 1, target);
        } else {
            return countOccurrences(str + 1, target);
        }
    }
}*/


//with pointers
#include <stdio.h>
int countOccurrences(char *str, char target);


int main() {
    char str[100], target;
    printf("Enter a string: ");
    scanf("%s", str);
    printf("Enter a character to count: ");
    scanf(" %c", &target);

    int result = countOccurrences(str, target);
    printf("The character '%c' appears %d times in the string.\n", target, result);

    return 0;
}
int countOccurrences(char *str, char target) {
    if (*str == '\0') {  // Base case: end of string
        return 0;
    } else {
        if (*str == target) {
            return 1 + countOccurrences(str + 1, target);
        } else {
            return countOccurrences(str + 1, target);
        }
    }
}
```

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

Sol:

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

```
#include <string.h>

// Function to check if a string is a palindrome (without pointers)
int isPalindrome(char str[], int start, int end) {
    // Base case: If start index is greater than or equal to end index, it's a palindrome
    if (start >= end) {
        return 1;
    }

    // Compare the characters at start and end
    if (str[start] != str[end]) {
        return 0; // Not a palindrome
    }

    // Recur with the next characters
    return isPalindrome(str, start + 1, end - 1);
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);


    int length = strlen(str);
    if (isPalindrome(str, 0, length - 1)) {
        printf("The string is a palindrome.\n");
    } else {
        printf("The string is not a palindrome.\n");
    }


    return 0;
}
```

With Pointers:

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


// Function to check if a string is a palindrome using pointers
int isPalindromeWithPointers(char *start, char *end) {
    // Base case: If start pointer is greater than or equal to end pointer, it's a palindrome
    if (start >= end) {
        return 1;
    }
    // Compare the characters at start and end
    if (*start != *end) {
        return 0; // Not a palindrome
    }

    // Recur with the next characters
    return isPalindromeWithPointers(start + 1, end - 1);
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);

    char *start = str;
    char *end = str + strlen(str) - 1;

    if (isPalindromeWithPointers(start, end)) {
        printf("The string is a palindrome.\n");
    } else {
        printf("The string is not a palindrome.\n");
    }
    return 0;
}
```

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

Sol: #include <stdio.h>

```

// Recursive function to calculate the length of a string (without pointers)
int stringLength(char str[], int index) {
    // Base case: If the current character is the null terminator, return 0
    if (str[index] == '\0') {
        return 0;
    }

    // Recursive case: Increment length by 1 and move to the next character
    return 1 + stringLength(str, index + 1);
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int length = stringLength(str, 0);
    printf("The length of the string is: %d\n", length);

    return 0;
}
```

With pointers:

```
#include <stdio.h>

// Recursive function to calculate the length of a string using pointers
int stringLengthWithPointers(char *str) {
    // Base case: If the current character is the null terminator, return 0
    if (*str == '\0') {
        return 0;
    }

    // Recursive case: Increment length by 1 and move to the next character
    return 1 + stringLengthWithPointers(str + 1);
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);

    int length = stringLengthWithPointers(str);
    printf("The length of the string is: %d\n", length);

    return 0;
}
```

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

Sol: #include <stdio.h>

```

// Recursive function to check if a number is prime (without pointers)
int isPrime(int num, int i) {
    // Base cases:
    // If num is less than or equal to 1, it's not a prime number
    if (num <= 1) {
        return 0;
    }
    // If i is greater than or equal to num/2, we have checked all possible divisors
    if (i == num / 2 + 1) {
        return 1;
    }
    // If num is divisible by i, it's not a prime number
    if (num % i == 0) {
        return 0;
    }
    // Recur with next divisor
    return isPrime(num, i + 1);
}

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (isPrime(num, 2)) {
        printf("%d is a prime number.\n", num);
    } else {
        printf("%d is not a prime number.\n", num);
    }
    return 0;
}
```

With pointer

```
#include <stdio.h>

// Recursive function to check if a number is prime using pointers
int isPrimeWithPointers(int num, int i) {
    // Base cases:
    // If num is less than or equal to 1, it's not a prime number
    if (num <= 1) {
        return 0;
    }
    // If i is greater than or equal to num/2, we have checked all possible divisors
    if (i >= num / 2) {
        return 1;
    }
    // If num is divisible by i, it's not a prime number
    if (num % i == 0) {
        return 0;
    }
    // Recur with next divisor
    return isPrimeWithPointers(num, i + 1);
}


int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);


    if (isPrimeWithPointers(num, 2)) {
        printf("%d is a prime number.\n", num);
    } else {
        printf("%d is not a prime number.\n", num);
    }


    return 0;
}
```

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

Sol: #include <stdio.h>

```


// Recursive function to print numbers from n down to 1 (without pointers)
void printReverse(int n) {
    // Base case: If n is less than 1, stop the recursion
    if (n < 1) {
        return;
    }

    // Print the current number
    printf("%d ", n);

    // Recursively call the function for the next number
    printReverse(n - 1);
}

int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);

    printf("Numbers in reverse order: ");
    printReverse(n);
    printf("\n");
    return 0;
}
```

With pointers

```
#include <stdio.h>

// Recursive function to print numbers from n down to 1 using pointers
void printReverseWithPointers(int *n) {
    // Base case: If n is less than 1, stop the recursion
    if (*n < 1) {
        return;
    }

    // Print the current number
    printf("%d ", *n);

    // Recursively call the function for the next number
    (*n)--;
    printReverseWithPointers(n);
}

int main() {
    int n;

    printf("Enter a number: ");

    scanf("%d", &n);



    printf("Numbers in reverse order: ");

    printReverseWithPointers(&n);

    printf("\n");



    return 0;

}
```

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

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



// Recursive function to calculate the sum of elements in an array (without pointers)

int arraySum(int arr[], int size) {

    // Base case: If the size is 0, return 0

    if (size == 0) {

        return 0;

    }
    // Recursive case: Add the first element and recurse for the rest
    return arr[0] + arraySum(arr + 1, size - 1);  // arr + 1 moves to the next element
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Sum of array elements: %d\n", arraySum(arr, size));

    return 0;
}
```

With pointers

```
#include <stdio.h>

// Recursive function to calculate the sum of elements in an array (with pointers)
int arraySumWithPointers(int *arr, int size) {
    // Base case: If size is 0, return 0
    if (size == 0) {
        return 0;
    }
    // Recursive case: Add the current element (pointed by arr) and recurse for the next element
    return *arr + arraySumWithPointers(arr + 1, size - 1);  // arr + 1 moves the pointer to the next element
}


int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);


    printf("Sum of array elements: %d\n", arraySumWithPointers(arr, size));


    return 0;
}
```

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

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


// Function to swap characters at positions i and j
void swap(char *x, char *y) {
    char temp = *x;
    *x = *y;
    *y = temp;
}

// Recursive function to generate all permutations of a string
void permute(char str[], int l, int r) {
    // Base case: If l == r, print the permutation
    if (l == r) {
        printf("%s\n", str);
    } else {
        for (int i = l; i <= r; i++) {
            swap(&str[l], &str[i]);  // Swap the current index with the loop index
            permute(str, l + 1, r);  // Recurse with the next index
            swap(&str[l], &str[i]);  // Backtrack to the previous state
        }
    }
}

int main() {
    char str[] = "ABC";
    int n = strlen(str);
    printf("All permutations of the string: \n");
    permute(str, 0, n - 1);

    return 0;
}
```

With pointers

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

// Function to swap characters at positions i and j
void swap(char *x, char *y) {
    char temp = *x;
    *x = *y;
    *y = temp;
}

// Recursive function to generate all permutations of a string using pointers
void permuteWithPointers(char *str, int l, int r) {
    // Base case: If l == r, print the permutation
    if (l == r) {
        printf("%s\n", str);
    } else {
        for (int i = l; i <= r; i++) {
            swap(&str[l], &str[i]);  // Swap the current index with the loop index
            permuteWithPointers(str, l + 1, r);  // Recurse with the next index
            swap(&str[l], &str[i]);  // Backtrack to the previous state
        }
    }
}

int main() {
    char str[] = "ABC";
    int n = strlen(str);

    printf("All permutations of the string: \n");
    permuteWithPointers(str, 0, n - 1);

    return 0;
}
```

20->14>21->45->89->56->63->72

1.diplay the linked list

2. count the number of elements present in the link list na dprint it

3. summ up of all the lements in the linked list

4. FInd the maximum element

5, find the minmum element in the linked list

6. Search for a particullar element whether it is present in the linked list.

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

struct Node {
    int data;
    struct Node *next;
};

void Display(struct Node *);
int CountNodes(struct Node *);
int SumElements(struct Node *);
int FindMax(struct Node *);
int FindMin(struct Node *);
int SearchElement(struct Node *, int);

int main() {
    struct Node *first = NULL;

    first = (struct Node *)malloc(sizeof(struct Node));
    first->data = 20;
```

```
first->next = NULL;

struct Node *second = (struct Node *)malloc(sizeof(struct Node));
second->data = 14;
second->next = NULL;
first->next = second;

struct Node *third = (struct Node *)malloc(sizeof(struct Node));
third->data = 21;
third->next = NULL;
second->next = third;

struct Node *fourth = (struct Node *)malloc(sizeof(struct Node));
fourth->data = 45;
fourth->next = NULL;
third->next = fourth;

struct Node *fifth = (struct Node *)malloc(sizeof(struct Node));
fifth->data = 89;
fifth->next = NULL;
fourth->next = fifth;
```

```
struct Node *sixth = (struct Node *)malloc(sizeof(struct Node));
sixth->data = 56;
sixth->next = NULL;
fifth->next = sixth;

struct Node *seventh = (struct Node *)malloc(sizeof(struct Node));
seventh->data = 63;
seventh->next = NULL;
sixth->next = seventh;

struct Node *eighth = (struct Node *)malloc(sizeof(struct Node));
eighth->data = 72;
eighth->next = NULL;
seventh->next = eighth;

Display(first);

int count = CountNodes(first);
printf("Number of elements in the linked list: %d\n", count);
```

```
    int sum = SumElements(first);
    printf("Sum of all elements in the linked list: %d\n", sum);


    int max = FindMax(first);
    printf("Maximum element in the linked list: %d\n", max);


    int min = FindMin(first);
    printf("Minimum element in the linked list: %d\n", min);


    int element;
    printf("Enter the element to search: ");
    scanf("%d", &element);
    if (SearchElement(first, element)) {
        printf("Element %d is present in the linked list.\n", element);
    } else {
        printf("Element %d is not present in the linked list.\n", element);
    }


    return 0;
}
```

```
void Display(struct Node *p) {
    while (p != NULL) {
        printf("%d -> ", p->data);
        p = p->next;
    }
    printf("\n");
}

int CountNodes(struct Node *p) {
    int count = 0;
    while (p != NULL) {
        count++;
        p = p->next;
    }
    return count;
}

int SumElements(struct Node *p) {
    int sum = 0;
    while (p != NULL) {
        sum += p->data;
        p = p->next;
    }
    return sum;
}

int FindMax(struct Node *p) {
    int max = p->data;
    while (p != NULL) {
        if (p->data > max) {
            max = p->data;
        }
        p = p->next;
    }
    return max;
}

int FindMin(struct Node *p) {
    int min = p->data;
    while (p != NULL) {
        if (p->data < min) {
            min = p->data;
        }
        p = p->next;
    }
    return min;
}

int SearchElement(struct Node *p, int element) {
    while (p != NULL) {
        if (p->data == element) {
            return 1; // Element found
        }
        p = p->next;
    }
    return 0; // Element not found
}
```