RECURSIVE FUNCTION(HEAD)
==========

```
#include <stdio.h>

void display(int n);

int main()
{
    int n=1;
    display(n);
    return 0;
}
void display(int a){
    if(a<=10){          //base condition
      printf("%d->",a);
      a=a+1;
      display(a);       //reursive call
    }
}
```

TAIL RECURSION
==========

```
#include <stdio.h>

void display(int n);

int main()
{
    int n=1;
    display(n);
    return 0;
}
void display(int a){
    if(a<=10){//base condition
      a=a+1;
      display(a);
      printf("%d->",a);

    }
}
```

SUMMATION USING RECURSIVE FUNCTION
==========

```
#include <stdio.h>

int sumNatural(int);  // Correct the return type to int

int main()
{
    int n = 10;
    int sum = sumNatural(n);  // Capture the return value
    printf("Summation value = %d \n", sum);
    return 0;
```

```
}

int sumNatural(int n)  // Change return type to int
{
    if (n == 0)
    {
        return 0;
    }
    return n + sumNatural(n - 1);  // Simplified return
}
```

USING POINTERS
==========

```
#include <stdio.h>

void sumNatural(int, int*);

int main()
{
    int n = 10;
    int sum = 0;
    sumNatural(n, &sum);
    printf("Summation value = %d\n", sum);
    return 0;
}

void sumNatural(int n, int *sum)
{
    if (n == 0)
    {
        return;
    }
    *sum += n;  // Add the current value of n to the sum
    sumNatural(n - 1, sum);  // Recursive call
}
```

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

WITHOUT POINTERS
==========

```
#include <stdio.h>

// Function to calculate factorial recursively
int factorial(int n) {
    // Base case: factorial of 0 or 1 is 1
    if (n == 0 || n == 1) {
        return 1;
    }
    // Recursive case: n * factorial of (n - 1)
    return n * factorial(n - 1);
```

```
}

int main() {
    int num;

    // Input a non-negative integer
    printf("Enter a non-negative integer: ");
    scanf("%d", &num);

    // Ensure the number is non-negative
    if (num < 0) {
        printf("Please enter a non-negative integer.\n");
    } else {
        // Call the factorial function and display the result
        printf("Factorial of %d is %d\n", num, factorial(num));
    }

    return 0;
}
```

WITH POINTERS

==========

```
#include <stdio.h>

// Function to calculate factorial recursively using a pointer
int factorial(int n) {
    // Base case: factorial of 0 or 1 is 1
    if (n == 0 || n == 1) {
        return 1;
    }
    // Recursive case: n * factorial of (n - 1)
    return n * factorial(n - 1);
}

// Function to calculate factorial recursively with pointer parameter
void factorial_ptr(int n, int *result) {
    // Base case: factorial of 0 or 1 is 1
    if (n == 0 || n == 1) {
        *result = 1;
        return;
    }
    // Recursive case: n * factorial of (n - 1)
    int temp_result;
    factorial_ptr(n - 1, &temp_result);  // Recursive call
    *result = n * temp_result;
}

int main() {
    int num;
    int result;

    // Input a non-negative integer
    printf("Enter a non-negative integer: ");
    scanf("%d", &num);
```

```
    // Ensure the number is non-negative
    if (num < 0) {
        printf("Please enter a non-negative integer.\n");
    } else {
        // Call the factorial function with pointers
        factorial_ptr(num, &result);
        printf("Factorial of %d is %d\n", num, result);
    }

    return 0;
}
```

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

WITHOUT *

==========

```
#include <stdio.h>

// Function to calculate the nth term of Fibonacci series
int fibonacci(int n) {
    // Base cases
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    // Recursive case
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n;

    // Get the nth term from the user
    printf("Enter the value of n: ");
    scanf("%d", &n);

    // Call the recursive function to get the nth Fibonacci number
    printf("Fibonacci term at position %d is: %d\n", n, fibonacci(n));

    return 0;
}
```

WITH *

========

```
#include <stdio.h>

// Function to calculate the nth Fibonacci number using pointers
int fibonacci(int n, int *result) {
    // Base cases
    if (n == 0) {
        *result = 0;
        return 0;
    }
    else if (n == 1) {
```

```
        *result = 1;
        return 1;
    }
    // Recursive case
    else {
        int prev1, prev2;
        fibonacci(n - 1, &prev1);  // Get fibonacci(n-1)
        fibonacci(n - 2, &prev2);  // Get fibonacci(n-2)
        *result = prev1 + prev2;
        return *result;
    }
}

int main() {
    int n;
    int result;

    // Get the nth term from the user
    printf("Enter the value of n: ");
    scanf("%d", &n);

    // Call the recursive function to get the nth Fibonacci number using pointers
    fibonacci(n, &result);

    // Print the result
    printf("Fibonacci term at position %d is: %d\n", n, result);

    return 0;
}
```

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

WITHOUT *

==========

```
#include <stdio.h>

// Recursive function to calculate sum of digits
int sumOfDigits(int n) {
    // Base case: if n is a single digit, return the digit
    if (n == 0) {
        return 0;
    }

    // Recursive step: sum the last digit and recurse for the rest of the digits
    return (n % 10) + sumOfDigits(n / 10);
}

int main() {
    int number;

    // Input a positive integer
    printf("Enter a positive integer: ");
    scanf("%d", &number);
```

```
    // Ensure the number is positive
    if (number <= 0) {
        printf("Please enter a positive integer.\n");
        return 1;
    }

    // Calculate and print the sum of digits
    int result = sumOfDigits(number);
    printf("The sum of digits of %d is: %d\n", number, result);

    return 0;
}
```

WITH *

========

```
#include <stdio.h>

// Recursive function to calculate sum of digits using pointers
int sumOfDigits(int *n) {
    // Base case: if the number is a single digit (less than 10)
    if (*n < 10) {
        return *n;
    }

    // Recursive case: sum the last digit and recursively process the rest
    return (*n - (*n / 10) * 10) + sumOfDigits(&(*n / 10));
}

int main() {
    int number;

    // Input a positive integer
    printf("Enter a positive integer: ");
    scanf("%d", &number);

    // Ensure the number is positive
    if (number <= 0) {
        printf("Please enter a positive integer.\n");
        return 1;
    }

    // Calculate and print the sum of digits
    int result = sumOfDigits(&number);
    printf("The sum of digits of %d is: %d\n", number, result);

    return 0;
}
```

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

WITHOUT *

==========

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

```
// Function to reverse a string using recursion
void reverseString(char str[], int start, int end) {
    // Base case: If start index is greater than or equal to the end index, return
    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 substring excluding the first and last characters
    reverseString(str, start + 1, end - 1);
}

int main() {
    char str[] = "Hello, World!";

    int length = 0;
    while (str[length] != '\0') {
        length++;
    }

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

    printf("Reversed string: %s\n", str);

    return 0;
}
```

WITH *

==========

```
#include <stdio.h>

// Function to reverse a string using recursion and pointers
void reverseString(char *start, char *end) {
    // Base case: If start pointer is greater than or equal to end pointer, return
    if (start >= end) {
        return;
    }

    // Swap characters at start and end
    char temp = *start;
    *start = *end;
    *end = temp;

    // Recursive call to reverse the substring excluding the first and last characters
    reverseString(start + 1, end - 1);
}

int main() {
```

```
    char str[] = "Hello, World!";

    // Find the end of the string (the last character before the null terminator)
    char *end = str;
    while (*end != '\0') {
        end++;
    }
    end--; // Move back one character to point to the last character in the string

    // Call the recursive function to reverse the string
    reverseString(str, end);

    printf("Reverse string: %s\n", str);

    return 0;
}
```

5.Power Calculation: Develop a recursive function to calculate the power of a number x raised to n
Greatest Common Divisor (GCD): Create a recursive function to find the GCD of two given integers using the Euclidean algorithm.

WITHOUT *

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

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

int main() {
    int x, n;

    printf("Enter base (x): ");
    scanf("%d", &x);

    printf("Enter exponent (n): ");
    scanf("%d", &n);

    int result = power(x, n);
    printf("%d raised to the power of %d is %d\n", x, n, result);

    return 0;
}
```

GCD CALCULATION

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

// Recursive function to find GCD using Euclidean algorithm
int gcd(int a, int b) {
```

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

int main() {
    int a, b;

    printf("Enter first integer (a): ");
    scanf("%d", &a);

    printf("Enter second integer (b): ");
    scanf("%d", &b);

    int result = gcd(a, b);
    printf("The GCD of %d and %d is %d\n", a, b, result);

    return 0;
}
```

WITH *(POWER CALUCULATION)

==========

```
#include <stdio.h>

// Recursive function to calculate power using pointers
int power(int x, int *n) {
    if (*n == 0) {
        return 1;  // Base case: x^0 = 1
    } else {
        (*n)--;  // Decrement exponent using pointer
        return x * power(x, n);  // Recursive case: x^n = x * x^(n-1)
    }
}

int main() {
    int x, n;

    printf("Enter base (x): ");
    scanf("%d", &x);

    printf("Enter exponent (n): ");
    scanf("%d", &n);

    int result = power(x, &n);  // Passing pointer to n
    printf("%d raised to the power of %d is %d\n", x, n, result);

    return 0;
}
```

GCD CALCULATION

==========

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

```
// Recursive function to find GCD using Euclidean algorithm with pointers
int gcd(int *a, int *b) {
    if (*b == 0) {
        return *a;  // Base case: gcd(a, 0) = a
    } else {
        int temp = *a % *b;
        *a = *b;  // Update a with b's value
        *b = temp; // Update b with the remainder
        return gcd(a, b);  // Recursive case
    }
}

int main() {
    int a, b;

    printf("Enter first integer (a): ");
    scanf("%d", &a);

    printf("Enter second integer (b): ");
    scanf("%d", &b);

    int result = gcd(&a, &b);  // Passing pointers to a and b
    printf("The GCD of %d and %d is %d\n", a, b, result);

    return 0;
}
```

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

WITHOUT *

==========

```
#include <stdio.h>

// Recursive function to count occurrences of 'ch' in the string 'str'
int countOccurrences(char *str, char ch) {
    // Base case: If the string is empty, return 0
    if (*str == '\0') {
        return 0;
    }

    // If the current character matches 'ch', return 1 + count in the rest of the string
    if (*str == ch) {
        return 1 + countOccurrences(str + 1, ch);
    }

    // Otherwise, just move to the next character in the string
    return countOccurrences(str + 1, ch);
}

int main() {
    char str[100], ch;

    // Input the string
```

```
    printf("Enter a string: ");
    scanf("%99[^\n]", str);  // Using scanf to read the full line until newline

    // Input the character
    printf("Enter the character to count: ");
    scanf(" %c", &ch);  // Notice the space before %c to skip any leading whitespace

    // Call the recursive function and print the result
    int result = countOccurrences(str, ch);
    printf("The character '%c' appears %d times in the string.\n", ch, result);

    return 0;
}
```

WITH *

========

```
#include <stdio.h>

// Recursive function to count occurrences of 'ch' in the string 'str' using pointers
int countOccurrences(char *str, char ch) {
    // Base case: If the string is empty, return 0
    if (*str == '\0') {
        return 0;
    }

    // If the current character matches 'ch', return 1 + count in the rest of the string
    if (*str == ch) {
        return 1 + countOccurrences(str + 1, ch);
    }

    // Otherwise, just move to the next character in the string
    return countOccurrences(str + 1, ch);
}

int main() {
    char str[100], ch;

    // Input the string using pointers
    printf("Enter a string: ");
    scanf("%99[^\n]", str);  // Read a full line of input into str using scanf

    // Input the character using pointers
    printf("Enter the character to count: ");
    scanf(" %c", &ch);  // Use a space before %c to handle any leading whitespace

    // Call the recursive function and print the result
    int result = countOccurrences(str, ch);
    printf("The character '%c' appears %d times in the string.\n", ch, result);

    return 0;
}
```

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

WITHOUT *
==========

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

// Recursive function to check if a string is a palindrome
int isPalindrome(char str[], int start, int end) {
    // Base case: If start index is greater than or equal to end, it's a palindrome
    if (start >= end) {
        return 1;
    }

    // Skip non-alphanumeric characters (spaces, punctuation)
    if ((str[start] < 'A' || (str[start] > 'Z' && str[start] < 'a') || str[start] > 'z') &&
        (str[start] < '0' || str[start] > '9')) {
        return isPalindrome(str, start + 1, end);
    }

    if ((str[end] < 'A' || (str[end] > 'Z' && str[end] < 'a') || str[end] > 'z') &&
        (str[end] < '0' || str[end] > '9')) {
        return isPalindrome(str, start, end - 1);
    }

    // Compare characters at start and end (case-sensitive comparison)
    if (str[start] != str[end]) {
        return 0;
    }

    // Recursive case: Check the next pair of characters
    return isPalindrome(str, start + 1, end - 1);
}

int main() {
    char str[100];

    // Use scanf to read the input string (without spaces)
    printf("Enter a string: ");
    scanf("%99[^\n]", str);  // Reads the entire line including spaces

    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 *
========

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

```
#include <string.h>

// Recursive function to check if a string is a palindrome using pointers
int isPalindrome(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;
    }

    // Skip non-alphanumeric characters (spaces, punctuation)
    if ((*start < 'A' || (*start > 'Z' && *start < 'a') || *start > 'z') &&
        (*start < '0' || *start > '9')) {
        return isPalindrome(start + 1, end);
    }

    if ((*end < 'A' || (*end > 'Z' && *end < 'a') || *end > 'z') &&
        (*end < '0' || *end > '9')) {
        return isPalindrome(start, end - 1);
    }

    // Compare characters at start and end (case-sensitive comparison)
    if (*start != *end) {
        return 0;
    }

    // Recursive case: Check the next pair of characters
    return isPalindrome(start + 1, end - 1);
}

int main() {
    char str[100];

    // Use scanf to read the input string (including spaces)
    printf("Enter a string: ");
    scanf("%99[^\n]", str);  // Reads the entire line including spaces

    // Pointer to the start and end of the string
    char *start = str;
    char *end = str + strlen(str) - 1;

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

    return 0;
}
```

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

WITHOUT *

==========

```
#include <stdio.h>

// Recursive function to calculate the length of a string
int stringLength(char str[]) {
    // Base case: if the current character is the null terminator, return 0
    if (str[0] == '\0') {
        return 0;
    } else {
        // Recursive case: 1 + length of the remaining string
        return 1 + stringLength(str + 1);
    }
}

int main() {
    char str[100];

    // Input the string character by character
    printf("Enter a string: ");
    int i = 0;
    char ch;
    while ((ch = getchar()) != '\n' && ch != EOF) {
        str[i++] = ch;
    }
    str[i] = '\0';  // Null-terminate the string

    // Calculate and print the length of the string
    int length = stringLength(str);
    printf("The length of the string is: %d\n", length);

    return 0;
}
```

WITH *

========

```
#include <stdio.h>

// Recursive function to calculate the length of a string using pointers
int stringLength(char *str) {
    // Base case: if the current character is the null terminator, return 0
    if (*str == '\0') {
        return 0;
    } else {
        // Recursive case: 1 + length of the remaining string
        return 1 + stringLength(str + 1);
    }
}

int main() {
    char str[100];

    // Input the string character by character using pointers
    printf("Enter a string: ");
    int i = 0;
    char ch;
```

```
    while ((ch = getchar()) != '\n' && ch != EOF) {
        str[i++] = ch;
    }
    str[i] = '\0';  // Null-terminate the string

    // Calculate and print the length of the string
    int length = stringLength(str);
    printf("The length of the string is: %d\n", length);

    return 0;
}
```

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

WITHOUT *
==========

```
#include <stdio.h>

int isPrime(int num, int divisor) {
    // Base case: If divisor is 1, it means the number is prime
    if (divisor == 1) {
        return 1;
    }

    // If num is divisible by any number other than 1 and itself, it's not prime
    if (num % divisor == 0) {
        return 0;
    }

    // Recursively check for smaller divisors
    return isPrime(num, divisor - 1);
}

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

    // Edge case: If the number is less than 2, it's not prime
    if (num < 2) {
        printf("%d is not a prime number.\n", num);
    } else {
        // Start the recursion with divisor set to num-1
        if (isPrime(num, num - 1)) {
            printf("%d is a prime number.\n", num);
        } else {
            printf("%d is not a prime number.\n", num);
        }
    }

    return 0;
}
```

WITH *

==========

```
#include <stdio.h>

int isPrime(int num, int *divisor) {
    // Base case: If divisor is 1, the number is prime
    if (*divisor == 1) {
        return 1;
    }

    // If num is divisible by the divisor, it’s not prime
    if (num % *divisor == 0) {
        return 0;
    }

    // Recursively check with the next smaller divisor
    (*divisor)--; // Decrement divisor by 1
    return isPrime(num, divisor);
}

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

    // Edge case: If the number is less than 2, it’s not prime
    if (num < 2) {
        printf("%d is not a prime number.\n", num);
    } else {
        int divisor = num - 1;

        // Pass the pointer to divisor to the isPrime function
        if (isPrime(num, &divisor)) {
            printf("%d is a prime number.\n", num);
        } else {
            printf("%d is not a prime number.\n", num);
        }
    }

    return 0;
}
```

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

WITHOUT *
==========

```
#include <stdio.h>

// Recursive function to print numbers in reverse order
void printNumbers(int n) {
    // Base case: If n is less than 1, return
    if (n < 1) {
        return;
```

```
    }

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

    // Recursive call with n-1
    printNumbers(n - 1);
}

int main() {
    int n;

    // Get input from the user
    printf("Enter a number: ");
    scanf("%d", &n);

    // Call the recursive function
    printf("Numbers in reverse order: ");
    printNumbers(n);

    return 0;
}
```

WITH *

==========

```
#include <stdio.h>

// Recursive function to print numbers in reverse order using pointers
void printNumbers(int *n) {
    // Base case: If n is less than 1, return
    if (*n < 1) {
        return;
    }

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

    // Decrement the value at pointer and make the recursive call
    (*n)--;
    printNumbers(n);
}

int main() {
    int n;

    // Get input from the user
    printf("Enter a number: ");
    scanf("%d", &n);

    // Call the recursive function
    printf("Numbers in reverse order: ");
    printNumbers(&n);

    return 0;
```

```
}
```

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

WITHOUT *
==========

```
#include <stdio.h>

// Function to calculate the sum of elements in an array
int arraySum(int arr[], int size) {
    // Base case: if the array has no elements, return 0
    if (size == 0) {
        return 0;
    }
    // Recursive case: sum the first element and recursively sum the rest
    return arr[0] + arraySum(arr + 1, size - 1);
}

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

    int sum = arraySum(arr, size);
    printf("The sum of the array elements is: %d\n", sum);

    return 0;
}
```

WITH *
==========

```
#include <stdio.h>

// Function to calculate the sum of elements in an array using pointers
int arraySum(int *arr, int size) {
    // Base case: if the array has no elements, return 0
    if (size == 0) {
        return 0;
    }
    // Recursive case: sum the first element and recursively sum the rest
    return *arr + arraySum(arr + 1, size - 1);
}

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

    int sum = arraySum(arr, size);
    printf("The sum of the array elements is: %d\n", sum);

    return 0;
}
```

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

string.

WITHOUT *
==========

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

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

// Recursive function to generate permutations of the string
void permute(char *str, int left, int right) {
    if (left == right) {
        // If left index equals right, we print the string
        printf("%s\n", str);
    } else {
        // Generate permutations by swapping each character with the left
        for (int i = left; i <= right; i++) {
            // Swap the current character with the left
            swap((str + left), (str + i));

            // Recur to generate permutations for the next position
            permute(str, left + 1, right);

            // Backtrack: Swap back to restore the original string
            swap((str + left), (str + i));
        }
    }
}

int main() {
    char str[] = "ABC"; // Example string

    int n = strlen(str);
    printf("All permutations of the string \"%s\" are:\n", str);

    permute(str, 0, n - 1);

    return 0;
}
```

WITH *
==========

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

// Function to swap characters at the addresses pointed by x and y
void swap(char *x, char *y) {
    char temp = *x;
```

```
    *x = *y;
    *y = temp;
}

// Recursive function to generate permutations of the string using pointers
void permute(char *str, int left, int right) {
    if (left == right) {
        // If left index equals right, we print the string
        printf("%s\n", str);
    } else {
        // Generate permutations by swapping each character with the left
        for (int i = left; i <= right; i++) {
            // Swap the current character with the left using pointers
            swap(str + left, str + i);

            // Recur to generate permutations for the next position
            permute(str, left + 1, right);

            // Backtrack: Swap back to restore the original string
            swap(str + left, str + i);
        }
    }
}

int main() {
    char str[] = "ABC"; // Example string

    int n = strlen(str);
    printf("All permutations of the string \"%s\" are:\n", str);

    permute(str, 0, n - 1);

    return 0;
}
```

LINKED LIST

==========

****

==========

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

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

int main()
{
  struct Node *first=NULL;
  first =(struct Node*)malloc(sizeof(struct Node));
  first->data=10;
  first->next=NULL;
```

```
  struct Node *second =NULL;
  second=(struct Node*)malloc(sizeof(struct Node));
  second->data=20;
  second->next=NULL;

  first->next=second;

  struct Node *third =NULL;
  third=(struct Node*)malloc(sizeof(struct Node));
  third->data=30;
  third->next=NULL;

  second->next=third;

  struct Node *p=first;
  while(p!=NULL){
      printf("%d ->",p->data);
      p=p->next;
  }

    return 0;
}
```

BY USING A FUNCTION

==========

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

struct Node {
    int data;
    struct Node *next;
};
void Display(struct Node *);
int main()
{
  struct Node *first=NULL;
  first =(struct Node*)malloc(sizeof(struct Node));
  first->data=10;
  first->next=NULL;

  struct Node *second =NULL;
  second=(struct Node*)malloc(sizeof(struct Node));
  second->data=20;
  second->next=NULL;

  first->next=second;

  struct Node *third =NULL;
  third=(struct Node*)malloc(sizeof(struct Node));
  third->data=30;
  third->next=NULL;

  second->next=third;
```

```
  Display(first);

     return 0;
}
void Display(struct Node *p){
    while(p!=NULL){
      printf("%d ->",p->data);
      p=p->next;
  }

}


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

struct Node {
    int data;
    struct Node *next;
};
void Display(struct Node *);
int main()
{
  struct Node *first=NULL;
  first =(struct Node*)malloc(sizeof(struct Node));
  first->data=10;
  first->next=NULL;

  struct Node *second =NULL;
  second=(struct Node*)malloc(sizeof(struct Node));
  second->data=20;
  second->next=NULL;

  first->next=second;

  struct Node *third =NULL;
  third=(struct Node*)malloc(sizeof(struct Node));
  third->data=30;
  third->next=NULL;

  second->next=third;

  Display(first);

     return 0;
}
void Display(struct Node *p) {
    if (p != NULL) {//if(p!=NULL){
        Display(p->next);  // Recursively call Display on the next node
         printf("%d ->", p->data);  // Print the data of the current node

    }

}
```

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

1.display 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.

CODE
==========

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int data;
    struct Node *next;
};
void Display(struct Node*);
int CountValues(struct Node*);
int sumElements( struct Node*);
int findMax( struct Node*);
int findMin( struct Node*);
int search(struct Node*,int value);

int main(){

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

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

    first->next=second;

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

    second->next=third;

    struct Node *fourth=NULL;
```

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

third->next=fourth;

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

fourth->next=fifth;

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

fifth->next=sixth;

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

sixth->next=seventh;

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

seventh->next=eight;

Display(first);
printf("\n");

int count=CountValues(first);
printf("Count=%d\n",count);
// Sum of elements
 printf("Sum of elements: %d\n", sumElements(first));

 // Maximum element
printf("Maximum element: %d\n", findMax(first));

// Minimum element
printf("Minimum element: %d\n", findMin(first));

// Search for an element (e.g., 45)
int element = 45;
if (search(first, element)) {
    printf("Element %d found in the list.\n", element);
} else {
    printf("Element %d not found in the list.\n", element);
}
```

```
    return 0;
    }


void Display(struct Node *p){
    while(p!=NULL){
        printf("%d->",p->data);
        p=p->next;
    }
}
// Function 2: Count the number of elements in the linked list
int CountValues(struct Node *p){
    int count=0;
    while(p!=NULL){
        p=p->next;
        count++;
    }
    return count;
}
// Function 3: Sum all the elements in the linked list
int sumElements(struct Node *p) {
    int sum = 0;
    while (p != NULL) {
        sum += p->data;
        p = p->next;
    }
    return sum;
}

// Function 4: Find the maximum element in the linked list

int findMax(struct Node *p) {
    if (p == NULL) {
        printf("List is empty\n");
        return -1; // Return an error value for empty list
    }
    int max = p->data;
    while (p != NULL) {
        if (p->data > max) {
            max = p->data;
        }
        p = p->next;
    }
    return max;
}

// Function 5: Find the minimum element in the linked list

int findMin(struct Node* p) {
    if (p == NULL) {
        printf("List is empty\n");
        return -1; // Return an error value for empty list
    }
    int min = p->data;
```

```
    while (p != NULL) {
        if (p->data < min) {
            min = p->data;
        }
        p = p->next;
    }
    return min;
}
// Function 6: Search for a particular element in the linked list

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