Recursive Function

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

Direct:

```
#include<stdio.h>

int fact(int n);

int main(){

    int n=5;

    int factorial=fact(n);

    printf("Factorial=%d\n",factorial);

    return 0;

}

int fact(int n){

    if(n==0 || n==1){

        return 1;

    }

    return n*fact(n-1);

}
```

Using Pointer

```
#include<stdio.h>

int fact(int *n);

int main(){

    int n=5;

    int factorial=fact(&n);

    printf("Factorial=%d\n",factorial);
```

```
    return 0;
}
int fact(int *n){
    if(*n==0 || *n==1){
        return 1;
    }
    int current =*n;
    (*n)--;
    return current*fact(n);
}
```

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

Direct

```
#include<stdio.h>

int fibonacci(int n);

int main() {
    int n = 10;
    int result = fibonacci(n);
    printf("The %dth term in the Fibonacci series is: %d\n", n, result);
    return 0;
}

int fibonacci(int n) {
```

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

Using Pointers:

```
#include<stdio.h>

int fibonacci(int *n);

int main() {
    int n = 10;
    int result = fibonacci(&n);
    printf("The %dth term in the Fibonacci series is: %d\n", n, result);
    return 0;
}

int fibonacci(int *n) {
    if (*n <= 0) {
        return 0;
    }
    if (*n == 1) {
```

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

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

```
#include<stdio.h>

int sumOfDigits(int n);

int main() {
    int n = 12345;
    int result = sumOfDigits(n);
    printf("The sum of digits of %d is: %d\n", n, result);
    return 0;
}

int sumOfDigits(int n) {
    if (n == 0) {
        return 0;
    }
    return (n % 10) + sumOfDigits(n / 10);  // Add the last digit and recurse with the remaining number
```

```
}
```

Using Pointer:

```
#include<stdio.h>


int sumOfDigits(int *n);


int main() {
    int n = 12345;
    int result = sumOfDigits(&n);
    printf("The sum of digits of %d is: %d\n", n, result);
    return 0;
}


int sumOfDigits(int *n) {
    if (*n == 0) {
        return 0;
    }
    int digit=*n/10;
    return (*n % 10) + sumOfDigits(&digit);  // Add the last digit and recurse with the remaining number
}
```

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

Direct:

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

```
void reverse(char string[],int start,int end);
int main(){
    char string[10]={"Athira"};
    int length=strlen(string);
    reverse(string,0,length-1);
    printf("Reversed String=%s",string);
    return 0;
}

void reverse(char string[],int start,int end){
    if(start>=end){
        return;
    }
    char temp=string[start];
    string[start]=string[end];
    string[end]= temp;

    reverse(string,start+1,end-1);
}
```

Using Pointers:

```
#include<stdio.h>
#include<string.h>
void reverse(char *string,int start,int end);
int main(){
    char string[10]={"Athira"};
```

```
    int length=strlen(string);
    reverse(string,0,length-1);
    printf("Reversed String=%s",string);
    return 0;
}


void reverse(char *string,int start,int end){
    if(start>=end){
        return;
    }
    char temp=string[start];
    string[start]=string[end];
    string[end]= temp;


    reverse(string,start+1,end-1);
}
```

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

Direct:

```
#include<stdio.h>
int power(int x,int y);
int main(){
    int x=2,n=5;
    int result= power(x,n);
    printf("%d power %d =%d",x,n,result);
```

```
    return 0;
}
int power(int x,int y){
    if(y==0){
        return 1;
    }
    return x*power(x,y-1);

}
```

Pointer:

```
#include<stdio.h>
int power(int *x,int *y);
int main(){
    int x=2,n=6;
    int result= power(&x,&n);
    printf("%d power %d =%d",x,n,result);
    return 0;
}
int power(int *x,int *y){
    if(*y==0){
        return 1;
    }
    int next=*y-1;
    return (*x)*power(x,&next);
```

}

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

Direct

```
#include<stdio.h>

int gcd(int x,int y);

int main(){

    int x=25,y=55;

    int result= gcd(x,y);

    printf("GCD of %d & %d =%d",x,y,result);

    return 0;

}

int gcd(int x,int y){

    if(y==0){

        return x;

    }

   return gcd(y,x%y);



}
```

Pointer:

```
#include<stdio.h>

int gcd(int *x,int *y);

int main(){

    int x=25,y=55;

    int result= gcd(&x,&y);

    printf("GCD of %d & %d =%d",x,y,result);
```

```
    return 0;
}
int gcd(int *x,int *y){
    if(*y==0){
        return *x;
    }
    int term=*x % *y;
   return gcd(y,&term);

}
```

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

```
#include<stdio.h>
int countOccurrences(char *str, char ch);

int main() {
    char str[] = "Athira";
    char ch = 'a';
    int result = countOccurrences(str, ch);
    printf("The character '%c' appears %d times in the string.\n", ch, result);
    return 0;
}

int countOccurrences(char *str, char ch) {
```

```
    // Base case: if the string is empty, return 0
    if (*str == '\0') {
        return 0;
    }

    // Recursive case: check if the current character matches
    if (*str == ch) {
        return 1 + countOccurrences(str + 1, ch);  // Count this occurrence and recurse on the rest
    }

    // If the current character doesn't match, just recurse on the rest
    return countOccurrences(str + 1, ch);
}
```

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

Direct:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
bool isPalindrome(char *str, int start, int end);
int main() {
    char str[] = "malayalam";
    int length = strlen(str);
    if (isPalindrome(str, 0, length - 1)) {
        printf("The string \"%s\" is a palindrome.\n", str);
    } else {
```

```
        printf("The string \"%s\" is not a palindrome.\n", str);
    }

    return 0;
}
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);
}
```

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

Direct:

```
#include <stdio.h>
int stringLength(char str[]);

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

```
    printf("The length of the string \"%s\" is: %d\n", str, length);
    return 0;
}

int stringLength(char str[]) {
    if (*str == '\0') {
        return 0;
    }

    return 1 + stringLength(str + 1);
}
```

Using Pointers:

```
#include <stdio.h>
int stringLength(char *str);
int main() {
    char str[] = "Hello, World!";
    int length = stringLength(str);
    printf("The length of the string \"%s\" is: %d\n", str, length);
    return 0;
}

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

```
    return 1 + stringLength(str + 1);
}
```

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

```
#include <stdio.h>
#include <math.h>
int isPrime(int num, int divisor);
int main() {
    int num;
    printf("Enter a number to check if it's prime: ");
    scanf("%d", &num);
    if (num > 1 && isPrime(num, 2)) {
        printf("%d is a prime number.\n", num);
    } else {
        printf("%d is not a prime number.\n", num);
    }
    return 0;
}
int isPrime(int num, int divisor) {
    // Base case: If divisor exceeds the square root of num
    if (divisor > sqrt(num)) {
        return 1; // Prime
    }
    if (num % divisor == 0) {
```

```
        return 0; // Not prime
    }
    // Check next divisor
    return isPrime(num, divisor + 1);
}
```

Using pointer

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

int isPrime(int *num, int divisor);

int main() {
    int num;

    printf("Enter a number to check if it's prime: ");
    scanf("%d", &num);

    if (num > 1 && isPrime(&num, 2)) {
        printf("%d is a prime number.\n", num);
    } else {
        printf("%d is not a prime number.\n", num);
    }

    return 0;
}
```

```
int isPrime(int *num, int divisor) {
    // Base case: If divisor exceeds the square root of num
    if (divisor > sqrt(*num)) {
        return 1; // Prime
    }
    if (*num % divisor == 0) {
        return 0; // Not prime
    }
    // Check next divisor
    return isPrime(num, divisor + 1);
}
```

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

Direct:

```
#include <stdio.h>

void printReverse(int n);

int main() {
    int n=15;

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

```
    return 0;
}

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

Using Pointers:

```
#include <stdio.h>

void printReverse(int *n);

int main() {
    int n=15;

    printf("Numbers in reverse order: ");
    printReverse(&n);
    printf("\n");

    return 0;
}
```

```
void printReverse(int *n) {
    if (*n <= 0) {
        return; // Base case: stop when n is 0 or less
    }
    printf("%d ", *n); // Print the current number
    int next=*n-1;
    printReverse(&next); // Recursive call with n - 1
}
```

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

Direct:

```
#include<stdio.h>
int sum(int array[],int n);
int main(){
    int array[]={2,4,2,3,4};
    int n=sizeof(array)/sizeof(array[0]);
    int result = sum(array,n);
    printf("Sum of elemnts=%d",result);
    return 0;
}

int sum(int array[],int n){
    if(n<0)
    return 0;
```

```
    return array[n-1]+sum(array,n-1);
}
```

Using Pointers:

```
#include<stdio.h>
int sum(int *array,int n);
int main(){
    int array[]={2,4,2,3,4};
    int n=sizeof(array)/sizeof(array[0]);
    int result = sum(array,n);
    printf("Sum of elemnts=%d",result);
    return 0;
}

int sum(int *array,int n){
    if(n<0)
    return 0;
    return array[n-1]+sum(array,n-1);
}
```

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

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

void swap(char *x, char *y) {
    char temp = *x;
```

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

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

    return 0;
}

void permute(char *str, int left, int right) {
    if (left == right) {
        printf("%s\n", str);
    } else {
        for (int i = left; i <= right; i++) {
            swap((str + left), (str + i));
            permute(str, left + 1, right);
            swap((str + left), (str + i));
        }
    }
}
```

Linked List

Q 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.

Write a program for this in linked list and explain me step by step

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
   int data;
   struct Node *next;
};
void display(struct Node *p);
int countElements(struct Node *p);
int sumElements(struct Node *p);
int maxElement(struct Node *p);
int minElement(struct Node *p);
int searchElement(struct Node *p, int key);
// 20->14>21->45->89->56->63->72
int main(){
      int choice;
   do{

   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;

printf("\nEnter a Choice\n");
printf("\n1.Display Linked List\n2.Count number of elemnts in Linked List\n");
printf("3.Sum up all the elements\n4.Find maximum Elemnt\n5.Find the minimum elemnt\n");
printf("6.Search element\n7.Exit\n");
scanf("%d",&choice);
switch (choice){
   case 1:
   display(first);
   break;

   case 2:
   int count=countElements(first);
   printf("\nNumber of Elements=%d\n",count);
   break;
```

```
case 3:
int sum = sumElements(first);
printf("Sum of Elemnts=%d\n",sum);

case 4:
int max=maxElement(first);
printf("Maximum Element=%d\n",max);

case 5:
int min=minElement(first);
printf("Minimum Element=%d\n",min);

case 6:
int key;
printf("Enter Key to search: ");
scanf("%d",&key);
int found=searchElement(first,key);
if(found){
   printf("%d found\n",key);
}
else
printf("%d not found\n",key);
break;

case 7:
printf("Exit");
break;

default:
printf("Invalid Choice\n");
```

```
        }
    }while(choice!=7);
    return 0;
}

void display(struct Node *p){
    if(p==NULL){
        return;
    }
    printf("%d,",p->data);
    display(p->next);
}
int countElements(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 maxElement(struct Node *p){
```

```
    int max=p->data;
    while(p!=NULL){
        if(p->data > max){
            max=p->data;
        }
        p=p->next;
    }
    return max;
}

int minElement(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 key){
    while(p!=NULL){
        if(p->data == key)
        return 1;
        p=p->next;
    }
    return 0;
}
```