RECURSION

>>Head recursion.

```
#include <stdio.h>

void display(int );

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); // recursive call
   }
}
```

>>Tale recursion

```
#include <stdio.h>

void display(int );

int main()
{
   int n =1;

   display(n);

   return 0;
}

void display(int a){

   if(a <= 10){    // Base condition
       a+=1;
       display(a); // recursive call
       printf("%d : ",a);

   }
}
```

>>>   Summation Of first N natural numbers

```
#include <stdio.h>

int sumNatural(int );

int main()
{
    int n = 10;


    int sum = sumNatural(n);
    printf (" sum value = %d \n",sum);

    return 0;
}

int sumNatural(int n){

    if(n == 0){
        return 0;
    }
    int summation = n + sumNatural(n-1);
    return summation;
}
```

---------→using pointers

```c
#include <stdio.h>

int sumNatural(int *n);

int main()
{
    int n = 10;

    int sum = sumNatural(&n);
    printf("Sum value = %d \n", sum);

    return 0;
}

int sumNatural(int *n)
{
    if (*n == 0)
    {
        return 0;
    }

    int temp = *n;
    (*n)--;
    int summation = temp + sumNatural(n);
    return summation;
}
```

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

```
#include<stdio.h>

int factorial(int n) {
    if (n == 0) {           // Base condition
        return 1;
    }

    return n * factorial(n - 1);
}

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


    if (n < 0) {
        printf("Enter a positivr integer only\n");
    } else {
        printf("Factorial of %d is %d\n", n, factorial(n));
    }

    return 0;
}
```

------------→ Using pointers

```
#include<stdio.h>

int factorial(int n) {
   if (n == 0) {           // Base condition
      return 1;
   }

   return n * factorial(n - 1);
}

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



   if (n < 0) {
      printf("Enter a positivr integer only\n");
   } else {
      printf("Factorial of %d is %d\n", n, factorial(n));
   }

   return 0;
}
```

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

```
#include <stdio.h>



int fibonacci(int n) {


    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {


        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}


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




    if (n < 0) {
        printf("Fibonacci series is not defined for negative numbers.\n");
    } else {
        printf("Fibonacci number at position %d is %d\n", n, fibonacci(n));
    }
    return 0;
}
```

---------------→Using pointers

```
#include <stdio.h>



int fibonacci(int *n) {
    // Base cases
    if (*n == 0) {
        return 0;
    } else if (*n == 1) {
        return 1;
    } else {
        int temp1 = *n - 1;
        int temp2 = *n - 2;

        return fibonacci(&temp1) + fibonacci(&temp2);
    }
}

int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    if (n < 0) {
        printf("Fibonacci series is not defined for negative numbers.\n");
    } else {
        int original = n;
        printf("Fibonacci number at position %d is %d\n", original, fibonacci(&n));
    }

    return 0;
}
```

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

```
#include <stdio.h>

//function to calculate the sum of the digits

int sumOfDigits(int n) {

    if (n == 0) {       // Base condition
        return 0;
    }

    return (n % 10) + sumOfDigits(n / 10); //recursive call
}
```

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

    if (n < 0) {
        printf("Sum of digits is not defined for negative numbers.\n");
    } else {
        printf("Sum of digits of %d is %d\n", n, sumOfDigits(n));
    }

    return 0;
}
```

------------------→ Using Pointers

```
#include <stdio.h>


int sumOfDigits(int *n) {

    if (*n == 0) {    // Base conditions
        return 0;
    }

    int lastDigit = *n % 10;
    *n = *n / 10;
    return lastDigit + sumOfDigits(n);//recursive call
}


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



    if (n < 0) {
        printf("Sum of digits is not defined for negative numbers.\n");
    } else {
        int original = n;
        printf("Sum of digits of %d is %d\n", original, sumOfDigits(&n));
    }

    return 0;
}
```

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

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


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

    if (start >= end) { // Base condition
        return;
    }

    char temp = str[start];
    str[start] = str[end];
    str[end] = temp;

    reverseString(str, start + 1, end - 1); // Recursivecall
}

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

    int length = strlen(str);
    reverseString(str, 0, length - 1);

    printf("Reversed string: %s\n", str);
    return 0;
}
```

--→ Using pointers

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

// Recursive function to reverse a string
void reverseString(char *str, int start, int end) {

    if (start >= end) {   // Base casee
        return;
    }

    char temp = str[start];
    str[start] = str[end];
    str[end] = temp;

    reverseString(str, start + 1, end - 1);  //recursive call
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s",str);
    int length = strlen(str);
    reverseString(str, 0, length - 1);

    printf("Reversed 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**

```
#include <stdio.h>


double power(double x, int n) {

    if (n == 0) {    // Base case
        return 1;
    }

    return x * power(x, n - 1);  // Recursive call
}

int main() {
    double x;
    int n;
    printf("Enter the base (x): ");
    scanf("%lf", &x);
    printf("Enter the exponent (n): ");
    scanf("%d", &n);

    printf("%.2lf raised to the power of %d is %.2lf\n", x, n, power(x, n));

    return 0;
}
```

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

```
#include<stdio.h>

int gcd(int a, int b){

    if(b == 0){  // Base condition
        return a;
    }
    return gcd(b, a % b); // Recursive call
}

int main(){

    int a, b;
    printf("Enter two integers :");
    scanf("%d %d",&a,&b);
    printf("The GCD of %d %d is %d\n",a,a,gcd(a, b));
    return 0;
}
```

---→ uSing pointers

```
#include<stdio.h>

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

    if(*b == 0){  // Base condition
        return *a;
    }
   int tempa = *a;
   int tempb = *b;
   int mod = tempa % tempb;
   return gcd(&tempb , &mod);
}

int main(){

    int a, b;
    printf("Enter two integers :");
    scanf("%d %d",&a,&b);

    printf("The GCD of %d %d is %d\n",a,a,gcd(&a, &b));
    return 0;
}
```

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>


// Recursive function to count occurrences
int countOccurrences(char *str, char ch) {
    // Base condition
    if (*str == '\0') {
        return 0;
    }


    if (*str == ch) {
        return 1 + countOccurrences(str + 1, ch);
    }


    return countOccurrences(str + 1, ch);
}


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


    printf("Enter a string: ");
    scanf("%99[^\n]", str);
    // Input the character
    printf("Enter the character to count: ");
    scanf(" %c", &ch);

    // 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;
}
```

--→ using pointers

```
#include <stdio.h>

// Recursive function to count occurrences
int countOccurrences(char *str, char ch) {
    // Base conditions
    if (*str == '\0') {
        return 0;
    }

    if (*str == ch) {
        return 1 + countOccurrences(str + 1, ch);
    }
```

```c
    return countOccurrences(str + 1, ch);
}

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

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

    // Input the character using pointers
    printf("Enter the character to count: ");
    scanf(" %c", &ch);

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

    return 0;
}
```

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

```
#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;
}
```

➔ Using pointers

```
#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;
}
```

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

```
#include <stdio.h>

// Recursive function to calculate the length of a given string
int stringLength(char str[], int index) {
   // Base condition
   if (str[index] == '\0') {
      return 0;
   }
   // Recursive case: move to the next character in the string and add 1 to the result
   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;
}
```

-→using pointers

```
#include <stdio.h>


// Recursive function to calculate the length of string
int stringLength(char *str) {
   // Base conditions
   if (*str == '\0') {
      return 0;
   }
   // Recursive call
   return 1 + stringLength(str + 1);
}


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

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

   return 0;
}
```

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

```
#include <stdio.h>

// Recursive function to check if a number is prime
int isPrime(int n, int i) {
    // Base conditions
    if (n <= 1) {
        return 0;
    }

    if (i * i > n) {
        return 1;
    }
    // If n is divisible by i, it is not prime
    if (n % i == 0) {
        return 0;
    }
    // Recursive call
    return isPrime(n, i + 1);
}

int main() {
    int n;
    printf("Enter an integer: ");
    scanf("%d", &n);
    if (isPrime(n, 2)) {
        printf("%d is a prime number.\n", n);
    } else {
        printf("%d is not a prime number.\n", n);
    }

    return 0;
}
```

➔ **Using pointers**

```
#include <stdio.h>

// Recursive function to check if a number is prime
int isPrime(int *n, int i) {
    // Base conditions
    if (*n <= 1) {     //if *n is less than or equal to 1, it is not prime
        return 0;
    }
    //  if i*i is greater than *n, *n is prime
    if (i * i > *n) {
        return 1;
    }
    // If *n is divisible by i, it is not prime
    if (*n % i == 0) {
        return 0;
    }
    // Recursive call
    return isPrime(n, i + 1);
}


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

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

    return 0;
}
```

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

```
#include <stdio.h>

void printReverse(int n) {
    // Base condition
    if (n < 1) {
        return;
    }

    printf("%d\n", n);
    // Recursive call
    printReverse(n - 1);
}

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

    printReverse(n);

    return 0;
}
```

---→ using pointer

```
#include <stdio.h>

void printReverse(int n) {
    // Base condition
    if (n < 1) {
        return;
    }

    printf("%d\n", n);
    // Recursive call
    printReverse(n - 1);
}
```

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

    printReverse(n);

    return 0;
}
```

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

```
#include <stdio.h>

// Recursive function to find the sum of all elements in an array
int arraySum(int arr[], int n) {
    // Base condition
    if (n == 0) {
        return 0;
    }
    // Recursive call
    return arr[n - 1] + arraySum(arr, n - 1);
}
```

```
int main() {
    int arr[100], n, sum;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    printf("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

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

    return 0;
}
```

-→ using pointers

```
#include <stdio.h>



int arraySum(int *arr, int n) {
    // Base conditions
    if (n == 0) {
        return 0;
    }
    // Recursive call
    return *arr + arraySum(arr + 1, n - 1);
}

int main() {
    int arr[100], n, sum;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    printf("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

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

    return 0;
}
```

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

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

```
// Function to swap characters at position x and y
void swap(char str[], int x, int y) {
    char temp;
    temp = str[x];
    str[x] = str[y];
    str[y] = temp;
}

// Recursive function to generate permutations
void permute(char str[], int l, int r) {
    int i;
    if (l == r) {                  // Base condition
        printf("%s\n", str);
    } else {
        for (i = l; i <= r; i++) {
            swap(str, l, i);
            permute(str, l + 1, r);// recursive call
            swap(str, l, i);
        }
    }
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%99s", str);

    int n = strlen(str);
    printf("Permutations of the string are:\n");
    permute(str, 0, n - 1);

    return 0;
}
```

- ➔ **Using pointers**

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

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

// Recursive function to generate permutations
void permute(char *str, int l, int r) {
    int i;
    if (l == r) {
        printf("%s\n", str);
    } else {
        for (i = l; i <= r; i++) {
            swap((str + l), (str + i));
            permute(str, l + 1, r);
            swap((str + l), (str + i));
        }
    }
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%99s", str);

    int n = strlen(str);
    printf("Permutations of the string are:\n");
    permute(str, 0, n - 1);

    return 0;
}
```

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

struct Node{

    int data;
    struct Node *next; //self referencing
};

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;
}
```

Using a function

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

struct Node{

    int data;
    struct Node *next; //self referencing
};
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;
    }
}
```

Recrsion also

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

struct Node{

    int data;
    struct Node *next; //self referencing
};
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){
        printf(" %d ->",p->data);
        Display(p->next);


    }
}
```

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 *p);

int Count(struct Node *p);

int Sum(struct Node *p);

int Max(struct Node *p);

int Min(struct Node *p);

int Search(struct Node *p,int n);


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

 seventh ->next = eighth;


 printf("Limked list ->");
 Display(first);
 printf("\n");

int count = Count(first);
 printf("no of elements = %d\n",count);


 int sum = Sum(first);
 printf("sum of elements = %d\n",sum);
```

```
    int max = Max(first);
    printf("maximum element = %d\n",max);


    int min = Min(first);
    printf("minimum element = %d\n",min);

    int n;
    printf("enter the element to search : ");
    scanf("%d",&n);
    if(Search(first,n)){

        printf("element %d is found",n);
    }else {
        printf("element %d is not found",n);
    }


    return 0;
}

void Display(struct Node *p){

    if(p != NULL){
        printf(" %d ->",p->data);
        Display(p->next);
    }
}

int Count(struct Node *p){

    int count = 0;
    while(p != NULL){
        count ++;
        p = p->next;
    }

    return count;
}

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

int Max(struct Node *p){
    int max = p->data;
    while(p != NULL){
```

```
    if(p->data > max){
        max = p->data;
    }
    p = p->next;
    }
    return max;
}

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


int Search(struct Node *p,int n){
    while(p != NULL){
        if(p->data == n){
            return 1;
        }
        p = p->next;
    }
    return 0;
```

```
}
```