

RECURSION

TYPE OF PROCESS (METHOD) // FUNCTION

1. ITERATIVE METHOD

input, cal., print
looping (REPEAT)

(while , do-while , for) -----> if - else

(direct method)

$$4! = 1 * 2 * 3 * 4$$

2. RECURSIVE METHOD

(itself)

(indirect method)

$$4! = 4 * 3!$$

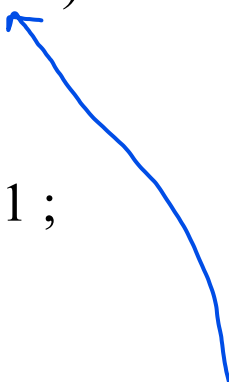
fact(n) = 1 if n == 0 // END PROCESS

n * fact(n-1) if n > 0 // REPEAT

$$4! = 4 * 3!$$

// WAP FOR FACTORIAL USING RECURSION

```
#include<stdio.h>
int fact( int n );
int main()
{
    int n , t;
    printf(" ENTER NO \n ");
    scanf("%d", &n);
    t = fact(n);
    printf(" FACT = %d \n " , t);
}
int fact( int n ) // RECURSIVE FUNCTION
{
    int f;
    if( n == 0 )
    {
        f = 1 ;
    }
    else
    {
        f = n * fact( n-1 );
    }
    return(f);
}
```



TRACE- METHOD - 1 :-

n = 3

t = fact(3) = 6

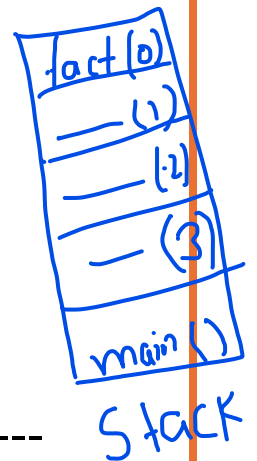
fact(3) = 3 * fact(2) = 3 * 2 = 6

fact(2) = 2 * fact(1) = 2 * 1 = 2

fact(1) = 1 * fact(0) = 1 * 1 = 1

fact(0) = 1

LIFO



TRACE :- METHOD - 2

n = 3

t = fact(3)

t = 3 * fact(2)

= 3 * 2 * fact(1)

= 3 * 2 * 1 * fact(0)

= 3 * 2 * 1 * 1

= 6

SUM OF N NOS (1 + 2 + 3 + + n)

sum(n) = 0 if n = 0

n + sum(n-1) if n > 0

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

```
int sum( int n);
```

```
int main()  
{
```

```
int n , t;
```

```
printf("enter no\n");  
scanf(" %d", &n);  
t = sum(n);
```

```
printf(" sum = %d\n",t);
```

```
}
```

```
int sum ( int n )  
{  
    int f;  
  
    if( n == 0 )  
    {  
        f = 0 ;  
    }  
    else  
    {  
        f = n + sum(n-1);  
    }  
    return(f);  
}
```

e.g. n = 3

$$\begin{aligned} t &= \text{sum}(3) \\ &= 3 + \text{sum}(2) \\ &= 3 + 2 + \text{sum}(1) \\ &= 3 + 2 + 1 + \text{sum}(0) \\ &= 3 + 2 + 1 + 0 \\ &= 6 \end{aligned}$$

power(x , n) = 1 if n = 0

x * power(x, n-1) if n > 0

e.g. power (2, 4) = 2 * power(2, 3) ;

power (x, n) = x * power(x, n-1) ;

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

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

```
int main()
```

```
{   int x , n , t;
```

```
    printf("enter x amd n\n");
```

```
    scanf ("%d%d",&x,&n);
```

```
    t = power(x,n);
```

```
    printf("ans = %d\n",t);
```

```
}
```

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

e.g. $x = 2, n = 2$

$t = \text{power}(2,2)$

$= 2 * \text{power}(2,1)$

$= 2 * 2 * \text{power}(2,0)$

$= 2 * 2 * 1$

$= 4$

TYPES OF RECURSION

1. DIRECT RECURSION
2. INDIRECT RECURSION
3. TREE RECURSION

2. INDIRECT RECURSION

```
int fun1()
{
    fun2();
    ----
}

int fun2()
{
    ----
    fun1();
    ----
}
```

The diagram shows two functions, `fun1()` and `fun2()`, each calling the other. A blue arrow originates from the `fun2();` line within the `fun1()` function block and points to the `int fun2()` function definition. A second blue arrow originates from the `fun1();` line within the `fun2()` function block and points back to the `int fun1()` function definition, forming a cycle that represents indirect recursion.

e.g. of tree recursion

FIBONACCI SERIES

0 1 1 2 3 5 8 13 21

0th 1st 2nd 3rd

FIND FIBONACCI NO.

N = 3 (index) ans = 2

- fib(n) = $\begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1 \end{cases}$

```
// wap for find Fibonacci no.
```

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

```
int fib(int n);
```

```
int main()
```

```
{
```

```
    int n,t;
```

```
    printf("enter no\n");
```

```
    scanf("%d", &n);
```

```
    t = fib(n);
```

```
    printf("fib. no. =%d\n",t);
```

```
}
```

```
// tree recursion
```

```
int fib( int n )
```

```
{
```

```
    int f;
```

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

```
    {  
        f = n;  
    }  
else  
    {  
        f=fib(n-1)+fib(n-2);  
    }  
return(f);  
}
```

e.g. $n = 3$

$$t = \text{fib} (3)$$
$$= \text{fib}(2) + \text{fib}(1)$$
$$= \text{fib}(1) + \text{fib}(0) + \text{fib}(1)$$
$$= 1 + \text{fib}(0) + \text{fib}(1)$$
$$= 1 + 0 + \text{fib}(1)$$
$$= 1 + 1$$
$$= 2$$

// WAP TO FIND FIBONACCI SERIES

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

```
int fib(int n);
```

```
int main()
```

```
{
```

```
    int n , i, t;
```

```
    printf("enter n\n");
```

```
    scanf("%d", &n);
```

```
    for( i = 0 ; i <=n ; i++)
```

```
    {
```

```
        t = fib(i);
```

```
        printf("%4d",t);
```

```
    }
```

```
}
```

```
int fib( int n )
{
    int f;

    if( n == 0 || n == 1)
    {
        f = n;
    }
else
    {
        f=fib(n-1)+fib(n-2);
    }
return(f);
}
```