

Factorial

✓ C Code for Factorial using Recursion

```
#include <stdio.h>

// Recursive function to calculate factorial
int factorial(int n) {
    if(n == 0 || n == 1) {
        return 1; // Base case: 0! = 1! = 1
    } else {
        return n * factorial(n - 1); // Recursive call
    }
}

int main() {
    int num = 5;

    int result = factorial(num);
    printf("Factorial of %d is %d\n", num, result);
    return 0;
}
```

Explanation

- **Base Case:**
 - Jab $n == 0$ ya $n == 1$, factorial 1 hota hai
 - Isliye hum yeh base case likhte hain:
 $\text{if}(n == 0 || n == 1) \text{return } 1;$
 - **Recursive Case:**
 - Har number ka factorial:
 $n * \text{factorial}(n - 1)$
 - Jaise:
 - $5! = 5 * 4!$
 - $= 5 * 4 * 3!$
 - $= 5 * 4 * 3 * 2!$
 - $= 5 * 4 * 3 * 2 * 1!$
 - $= 5 * 4 * 3 * 2 * 1 = 120$
-

Recursion Tree (5!)

factorial(5)



5 * factorial(4)



4 * factorial(3)



3 * factorial(2)



2 * factorial(1)



1 (base case)

Then it returns back up:

$2 * 1 = 2$

$3 * 2 = 6$

$4 * 6 = 24$

$5 * 24 = 120$

Output:

Factorial of 5 is 120

Fibonacci

✅ C Code for Fibonacci (Recursive)

```
#include <stdio.h>

// Recursive function to calculate nth Fibonacci number
int fibonacci(int n) {
    if(n == 0) return 0;    // Base case 1
    if(n == 1) return 1;    // Base case 2
    return fibonacci(n - 1) + fibonacci(n - 2); // Recursive step
}

int main() {
    int n = 6;

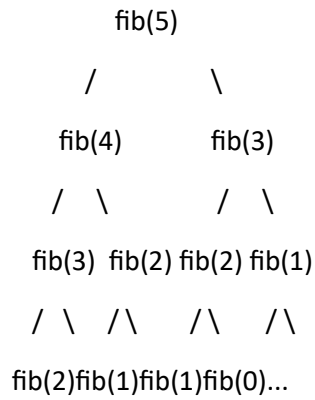
    printf("Fibonacci series up to %d terms:\n", n);
    for(int i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }

    return 0;
}
```

Explanation

- **Fibonacci Series:** 0, 1, 1, 2, 3, 5, 8, 13, ...
 - Har number = previous two numbers ka sum
$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$
 - **Base cases:**
 - $\text{fib}(0) = 0$
 - $\text{fib}(1) = 1$
 - **Recursive case:**
 - Jaise:
 - $\text{fib}(4) = \text{fib}(3) + \text{fib}(2)$
 - $= (\text{fib}(2) + \text{fib}(1)) + (\text{fib}(1) + \text{fib}(0))$
 - $= \dots$
-

Recursion Tree for fibonacci(5)



Jaise jaise tree badhta hai, **bohot saare calls repeat hote hain**, jaise fib(2), fib(1) multiple times.

Output for n = 6

Fibonacci series up to 6 terms:

0 1 1 2 3 5

Note:

- Recursive Fibonacci is **slow for large n** due to repeated work
-

Ackermann Function

What is Ackermann Function?

The **Ackermann function** $A(m, n)$ is defined as:

$$\begin{aligned} A(m, n) = & \\ & n + 1 \quad \text{if } m = 0 \\ & A(m - 1, 1) \quad \text{if } m > 0 \text{ and } n = 0 \\ & A(m - 1, A(m, n - 1)) \quad \text{if } m > 0 \text{ and } n > 0 \end{aligned}$$

Grows very fast!

Even small inputs like $A(3, 5)$ can crash a program if not careful.

C Code for Ackermann Function

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

```
int ackermann(int m, int n) {  
    if (m == 0)  
        return n + 1;  
    else if (n == 0)  
        return ackermann(m - 1, 1);  
    else  
        return ackermann(m - 1, ackermann(m, n - 1));  
}
```

```
int main() {  
    int m = 2, n = 3;  
    printf("Ackermann(%d, %d) = %d\n", m, n, ackermann(m, n));  
    return 0;  
}
```

Explanation

- Jab $m = 0$, to answer is $n + 1$
- Jab $m > 0$ & $n = 0$, to `ackermann(m-1, 1)` call karo
- Jab **dono** $m > 0$ & $n > 0$, to:
 - Pehle `ackermann(m, n-1)` call hota hai
 - Fir uska result use karke: `ackermann(m-1, result)`

Example: `ackermann(2, 1)`

`A(2, 1)`

`= A(1, A(2, 0))`


`= A(1, A(1, 1))`

`= A(1, A(0, A(1, 0)))`

...

Yeh bahut deeply nested ho jaata hai.

Sample Outputs:

A(m, n)	Result
A(0, 1)	2
A(1, 2)	4
A(2, 2)	7
A(3, 3)	61
A(4, 1)	 Stack Overflow