Session 07 C

Thursday, 30 October 2025 11:31 AM

Recursion

🌀 Recursion in C

Definition:
Recursion in C is a process where a function **calls itself** directly or indirectly to solve a problem.
Each recursive call reduces the problem into smaller subproblems until a **base condition** is reached, which stops further calls.

🔷 Syntax Example

```
#include <stdio.h>
void display(int n) {
   if (n == 0)   // Base condition
      return;
   printf("%d ", n);
   display(n - 1);   // Recursive call
}
int main() {
   display(5);
   return 0;
}
```

Output:

5 4 3 2 1

⚙️ How It Works

- Each recursive call is stored in the **function call stack**.
- When the base condition is met, the stack unwinds and functions return in reverse order.

🎯 Uses of Recursion

Recursion is commonly used in:

1. **Mathematical problems**
   (e.g., factorial, Fibonacci series, power of a number)
2. **Data structures**
   (e.g., tree traversal, graph traversal)
3. **Divide and conquer algorithms**
   (e.g., merge sort, quick sort, binary search)
4. **Backtracking algorithms**
   (e.g., maze solving, N-Queens problem)
5. **File system traversal**

✅ Advantages of Recursion

1. **Simpler Code:**
   Reduces complex problems into smaller, readable parts.
2. **Ease of Implementation:**
   Some problems are easier to express recursively (like tree traversal).
3. **Avoids Loops:**
   Eliminates the need for explicit loops in certain situations.
4. **Modularity:**
   Each recursive function call handles one part of the problem.

❌ Disadvantages of Recursion

1. **High Memory Usage:**
   Each function call adds a new frame to the **stack**, which can cause **stack overflow**.
2. **Slower Execution:**
   Due to repeated function calls and stack management.
3. **Difficult to Debug:**
   Tracing recursive calls can be confusing.
4. **Not Always Efficient:**
   Iterative solutions are often faster and use less memory.

🧮 Example – Factorial Using Recursion

```
#include <stdio.h>
int factorial(int n) {
   if (n == 0 || n == 1)
      return 1;  // Base condition
   else
      return n * factorial(n - 1);  // Recursive call
}
int main() {
   int num = 5;
   printf("Factorial of %d = %d", num, factorial(num));
   return 0;
}
```

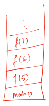
Output:

Factorial of 5 = 120

```
main()
{
   f(5);
}
```

```
void f(int n)
{
   printf("%d", n);
   n++;
   f(n);
}
```

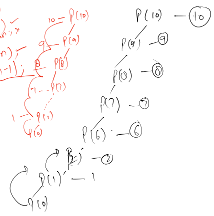
5 6 7 8 9 10




```
#include <stdio.h>
int main()
{
   Print(10);
}
```

```
void Print(int n)
{
   if (n<1)
      return;
   printf("%d", n);
   Print(n-1);
}
```

Print(n-1);
printf("%d", n);



```
#include <stdio.h>
void printDsc(int n)
{
    if (n < 1)
        return;
    printf("%d\n", n);
    printDsc(n - 1);
}
int main()
{
    printAsc(5);
    printDsc(5);
    return 0;
}
```

```
void printAsc(int n)
{
    if (n < 1)
        return;
    printAsc(n - 1);
    printf("%d\n", n);
}
```

5, 15

```
for(i=5; i<=15; i++)
{
   printf("%d", i);
}
```

```
#include <stdio.h>
void PrintRange(int start, int end)
{
   if (start > end)
      return;
   printf("%d", start);
   PrintRange(start+1, end);
}
```

(4, 10)

4 + Sum(5, 10); → 45
└ 5 + Sum(6, 10) → 40
└ 6 + Sum(7, 10) → 34
└ 7 + Sum(8, 10) → 27
└ 8 + Sum(9, 10) → 19
└ 9 + Sum(10, 10) → 10
└ 10 + Sum(11, 10) → 0

```
int Sum(int start, int end)
{
   if (start > end)
      return 0;
   return start + Sum(start+1, end);
}
```

Sum(5, 10)
5 + Sum(6, 10) → 40
6 + Sum(7, 10) → 34
7 + Sum(8, 10) → 27
8 + Sum(9, 10) → 19
9 + Sum(10, 10) → 10
10 + Sum(11, 10) → 0

```
int factorial(int n)
{
   if(n==1)
      return 1;
   return n * factorial(n-1);
}
```

factorial(5)
5 * factorial(4)
4 * fact(3)
3 * fact(2)
2 * fact(1)
1 * fact(0)
1