Course: Advanced Programming In C **Semester:**1st term 2026/2027	**Lecturers:** Dr Dina Abdelhafiz

Tutorial 3: Recursion in C Programming

What is Recursion?

Recursion is a programming technique where a function calls itself to solve a problem. It typically involves breaking down a problem into smaller, more manageable sub-problems until reaching a base case, which is a condition that stops the recursion.

Key Concepts of Recursion:

1. **Base Case**: This is the condition under which the recursion terminates. It prevents infinite loops and allows the function to return a result.
2. **Recursive Case**: This is the part of the function where the function calls itself with a modified argument, moving toward the base case.
3. **Stack Memory**: Each recursive call consumes stack memory. If the recursion is too deep, it can lead to a stack overflow.

Objective

The goal of this lab assignment is to understand and implement recursion in C by calculating the Fibonacci sequence. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1.

The sequence looks like this:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The Fibonacci sequence can be defined recursively as follows:

- Base Cases:
  - $F(0) = 0$
  - $F(1) = 1$
- Recursive Case:
  - For $n > 1$, $F(n) = F(n-1) + F(n-2)$

```c
#include <stdio.h>

// Function prototype
int fibonacci(int n);

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

    if (n < 0) {
        printf("Please enter a non-negative integer.\n");
    } else {
        printf("Fibonacci of %d is: %d\n", n, fibonacci(n));
    }

    return 0;
}
```

```c
// Recursive function to calculate Fibonacci number
int fibonacci(int n) {
    if (n == 0) {
//TO DO: return the base case

    } else if (n == 1) {
//TO DO: return the other base case

    } else {

//TO DO: return the recursive case


    }
}
```

A table summarizing the advantages and disadvantages of recursion

Advantages of Recursion	Disadvantages of Recursion
Easy to Understand	Extra Time and Space
Recursive solutions can be clearer and easier to grasp.	Recursive calls use more time and memory.
Less Code	Risk of Stack Overflow
Requires fewer lines of code, making it simpler.	Too many recursive calls can crash the program.
Good for Certain Problems	Can Be Slow
Works well for problems with tree-like structures.	Some recursive methods can take a long time.
Helpful for Exploring Options	Harder to Debug
Great for solving puzzles and exploring different choices.	Finding mistakes in recursive code can be tricky.
Good for Math Problems	Limited Optimization
Can easily solve math problems, like finding factorials or Fibonacci numbers.	Not all programming languages improve performance for deep recursion.

Happy Coding!