SESSION 5: FUNCTIONS & RECURSION

Understanding Functions in C:

A function is a block of code designed to perform a specific task.
It helps break large programs into smaller, reusable parts.

Real-Life Example:

Think of a washing machine — it has separate functions for washing, rinsing, and drying.
Similarly, in programming, functions divide big tasks into manageable ones.

Why Use Functions?

- Reusability (write once, use multiple times)
- Makes code organized and readable
- Easier debugging and maintenance

Function Declaration, Definition, Call:

1. Function Declaration (Prototype)

Tells the compiler about a function's name, return type, and parameters.

```
int add(int a, int b);
```

2. Function Definition

Contains the actual code (body) of the function.

```
int add(int a, int b) {
   int sum = a + b;
   return sum;
}
```

3. Function Call

Executes the function.

```
int main() {
   int result = add(5, 10);
   printf("Sum = %d", result);
   return 0;
}
```

Pass by Value vs Pass by Reference

Pass by Value:

A copy of the variable is passed to the function.

```
#include <stdio.h>
void changeValue(int x) {
   x = 10;
}

int main() {
   int a = 5;
   changeValue(a);
   printf("Value of a = %d", a);  // Output: 5
   return 0;
}
```

Pass by Reference:

The address of the variable is passed to the function (using pointers).

```
#include <stdio.h>
void changeValue(int *x) {
   *x = 10;
}

int main() {
   int a = 5;
   changeValue(&a);
   printf("Value of a = %d", a);  // Output: 10
   return 0;
}
```

Concept of Recursion:

Recursion is a process in which a function calls itself directly or indirectly.
It is used to solve problems that can be broken into smaller, similar sub-problems.

Key Points:
- Every recursive function must have a base condition to stop recursion.
- Without a base case, recursion leads to infinite calls and stack overflow.

Recursive Examples:

Example 1: Print 1 to N using Recursion

```
#include <stdio.h>
void printNumbers(int n) {
   if(n == 0)
      return;
   printNumbers(n - 1);
   printf("%d ", n);
}

int main() {
   int n;
   printf("Enter N: ");
   scanf("%d", &n);
   printNumbers(n);
   return 0;
}
```

Example 2: Fibonacci Number

```
#include <stdio.h>
int fibonacci(int n) {
   if(n == 0) return 0;
   if(n == 1) return 1;
   return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
   int n;
   printf("Enter n: ");
   scanf("%d", &n);
   printf("Fibonacci (%d) = %d", n, fibonacci(n));
   return 0;
}
```

Example 3: Reverse an Integer

```
#include <stdio.h>
int reverse(int n, int rev) {
   if(n == 0)
      return rev;
   return reverse(n / 10, rev * 10 + n % 10);
}

int main() {
   int n;
   printf("Enter number: ");
   scanf("%d", &n);
   printf("Reversed = %d", reverse(n, 0));
   return 0;
}
```

Example 4: Reverse a String

```
#include <stdio.h>
#include <string.h>
void reverseString(char str[], int index) {
   if(index < 0)
      return;
   printf("%c", str[index]);
   reverseString(str, index - 1);
}

int main() {
   char str[100];
   printf("Enter string: ");
   gets(str);
   reverseString(str, strlen(str) - 1);
   return 0;
}
```

Activity:

Task: Write a function to calculate factorial using recursion.

```
#include <stdio.h>
int factorial(int n) {
   if(n == 0 || n == 1)
      return 1;
   return n * factorial(n - 1);
}

int main() {
   int n;
   printf("Enter number: ");
   scanf("%d", &n);
   printf("Factorial = %d", factorial(n));
   return 0;
}
```

Key Points:

- Functions make programs modular and reusable.
- “Pass by value” sends a copy, “Pass by reference” modifies the original.
- Recursion simplifies complex problems like factorial, Fibonacci, or reversal tasks.
- Always include a base case in recursion to prevent infinite calls.