UNIT – 3

Dr.Kavitha N
Associate Professor
Dept. of IT

UNIT- III

Topics to be covered :

Functions: Built-in, User defined functions, categories of function, passing parameter techniques, scope of names and array of functions.

Recursion: TheNatureofRecursion,TracingaRecursiveFunction,RecursiveMathematical Functions, Recursive Functions with Array and String Parameters.

Pointers: Introduction, pointer arithmetic, pointer with arrays and strings.

Dr.Kavitha N
Associate Professor
Dept. of IT

FUNCTIONS

A function is a self-contained, named block of code that performs a specific, well-defined task and can be called (invoked) from any part of the program multiple times.

Why Functions? - The Real-World Analogy

- Think of functions like a recipe in a cookbook:
- The recipe has a name (function name)
- It lists ingredients needed (parameters)
- It describes step-by-step instructions (function body)
- It produces a dish (return value)
- You can use the same recipe multiple times without rewriting it

Key Characteristics

- Modularity: Break large problems into smaller, manageable tasks
- Reusability: Write once, use multiple times throughout code
- Abstraction: Hide implementation details behind simple interface
- Testability: Isolated units are easier to test individually
- Maintainability: Changes localized to specific functions

The Mandatory Function: main()

main() is the entry point of every C program - the first function that executes when the program runs.

- Every C program must have exactly one main() function
- Program execution starts at the first line of main()
- When main() ends (return 0;), the program terminates
- All other functions are called directly or indirectly from main()

Dr.Kavitha N
Associate Professor
Dept. of IT

Function Components: Declaration • Definition • Call

Function Declaration

Function Declaration (Prototype): A function declaration (also called a prototype) is a statement that tells the compiler about a function's existence, including its name, return type, and the number and types of parameters it accepts, without providing the actual implementation.

Syntax: returnType functionName(parametersList);

Syntax Breakdown:

return_type function_name(parameter_type1, parameter_type2, ...);

↓ ↓ ↓

What it returns — What to call it — What inputs it needs

Examples with Different Return Types

Integer Return

int findMax(int a, int b);

Void Return (No Value)

void displayMenu(void);

Character Return

char getGrade(float marks);

Double Return

double computeInterest(double principal, int years);

Dr.Kavitha N
Associate Professor
Dept. of IT

Function Definition

Function Definition (Implementation) : A function definition is the complete specification of a function that includes both the function header and the function body containing the actual executable statements that perform the function's task.

Syntax :

```
returnType functionName(parametersList)
 {
 Actual code...
 }
```

Syntax Breakdown :

```
return_type function_name(parameter_type parameter_name) {  ← Header
    // Local variable declarations                          ← Body begins
    statement1;
    statement2;
    ...
    return value;  // if return_type is not void            ← Body ends
}
```

Example :

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

- int → return type (the function returns an integer)
- add → function name
- (int a, int b) → parameters
- { ... } → function body containing statements

Dr.Kavitha N
Associate Professor
Dept. of IT

Function Call

Function Call (Invocation) : A function call is a statement that transfers program control from the calling function to the called function, executes the called function's code, and returns control along with any return value back to the point where the call was made.

Syntax :

function_name(parameters);

Syntax Breakdown :

function_name(argument1, argument2, ...);

↓ ↓

Name must match def — Actual values (match type & order)

Example :

```
#include <stdio.h>

void greet() {
    printf("Hello, welcome to C
programming!\n");
}

int main() {
    greet();   // Function call
    return 0;
}
```


Dr.Kavitha N
Associate Professor
Dept. of IT

Purpose of Functions

Modularity

Modularity is decomposing a program into independent, reusable units or modules, with each module handling a specific task or functionality.

Reusability

Write once, call many times. Functions allow code to be defined once and used repeatedly throughout a program or even across different projects.

Abstraction

Hide implementation complexity behind simple interfaces. Abstraction lets users call functions without needing to understand their internal details.

- **Clarity: Each function implements a single responsibility, making code more understandable**
- **Collaboration: Team members can work on different functions simultaneously**
- **Testing: Individual functions can be tested in isolation with unit tests**
- **Debugging: Errors can be isolated to specific functions, easier to locate and fix**
- **Maintainability: Functions can be modified independently without affecting other parts**
- **Reduced code duplication - Define logic in one place**
- **Easier maintenance - Change implementation in a single location**
- **Clear interfaces - Function signatures provide documentation**
- **Better testing - Isolated units are easier to test**
- **Readability - Function names describe their purpose**

Dr.Kavitha N
Associate Professor
Dept. of IT

Standard Library Functions

Standard Library Functions in C are predefined functions that are already written, compiled, and provided as part of the C Standard Library to perform commonly needed operations like input/output, string handling, mathematical computations, memory management, etc.

Standard Library Functions are also known as:

- Built-in Functions
- Predefined Functions
- Library Functions
- System-defined Functions

Header File	Function Name	Purpose
<stdio.h>	printf(), scanf()	Input and output operations
<math.h>	sqrt(), pow()	Mathematical calculations
<string.h>	strlen(), strcpy()	String manipulation
<stdlib.h>	malloc(), rand(), exit()	Utility and memory functions

Example :

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

int main() {
    printf("Hello, World!\n");     // from <stdio.h>
    printf("Square root of 16 = %.2f\n", sqrt(16));  // from <math.h>
    return 0;
}
```

Output:

Hello, World!
Square root of 16 = 4.00

Dr.Kavitha N
Associate Professor
Dept. of IT

User-Defined Functions

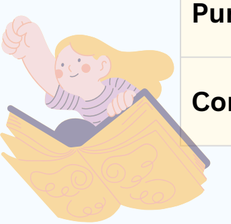
User-Defined Functions are functions created by the programmer to perform specific tasks according to program requirements. These functions contain user-written logic and can be reused throughout the program.

User-Defined Functions are also known as:

- Programmer-defined Functions
- Custom Functions
- Developer-defined Functions

Difference between Standard Library Functions and User Defined Functions

Feature	Standard Library Function	User-Defined Function
Definition	Predefined by C compiler	Created by the programmer
Header File	Declared in .h files	Declared by user in program
Example	printf(), sqrt()	add(), display()
Purpose	Perform general tasks	Perform specific tasks
Control	Not modifiable	Fully controlled by programmer



Example:

```
#include <stdio.h>

// User-defined function
int add(int a, int b) {
    int sum = a + b;
    return sum;
}
int main() {
    int result = add(10, 20);   // Function call
    printf("Sum = %d\n", result);
    return 0;
}
```

Output:

Sum = 30

Dr.Kavitha N
Associate Professor
Dept. of IT

Categories of Functions

Based on the data flow between the calling function and the called function, C functions are classified into four categories:

1. Function without Parameters and without Return value
2. Function with Parameters and without Return value
3. Function without Parameters and with Return value
4. Function with Parameters and with Return value

Purpose

- To understand how data is passed and received between functions.
- To make programs modular, clear, and reusable.

Dr.Kavitha N
Associate Professor
Dept. of IT

Function without Parameters and without Return Value

- The function does not take any input from the calling function.
- The function does not return any value back.
- It performs a specific task independently.

Syntax:

```
void function_name() {
    // statements
}
```

Example:

```
#include <stdio.h>
void welcomeMessage() {   // no parameters, no return
    printf("Welcome to the Coding World!\n");
}
int main() {
    welcomeMessage();
    return 0;
}
```

Output:

```
Welcome to the Coding World!
```


Dr.Kavitha N
Associate Professor
Dept. of IT

Function with Parameters and without Return Value

- Function receives data (arguments) from the calling function.
- Performs the required operation using the data.
- Does not return any result back — output is shown inside the function itself.

Syntax:

```
void function_name(data_type var1, data_type var2)
{
    // process the values
}
```

Example:

```
#include <stdio.h>
void printStars(int n) {    // parameters, no return
    for(int i = 0; i < n; i++)
        printf("*");
    printf("\n");
}
int main() {
    printStars(5);
    return 0;
}
```

Output:

```
*****
```


Dr.Kavitha N
Associate Professor
Dept. of IT

Function without Parameters and with Return Value

- No input values are given to the function.
- Function performs task internally and returns a result to the caller.

Syntax:

```
data_type function_name() {
    // computation
    return value;
}
```

Example:

```
#include <stdio.h>
int getTemperature() {    // no parameters, returns int
    int temp = 28;
    return temp;
}
int main() {
    printf("Today's temperature is %d°C\n",
getTemperature());
    return 0;
}
```

Output:

Today's temperature is 28°C

Dr.Kavitha N
Associate Professor
Dept. of IT

Function with Parameters and with Return Value

- The most versatile and commonly used function type.
- Function receives inputs (parameters) and returns a result.

Syntax:

```
data_type function_name(data_type var1, data_type var2)
{
    // statements
    return result;
}
```

Example:

```
#include <stdio.h>
int power(int base, int exp) {   // parameters + return
    int result = 1;
    for (int i = 0; i < exp; i++)
        result *= base;
    return result;
}
int main() {
    printf("2 raised to 5 = %d\n", power(2, 5));
    return 0;
}
```

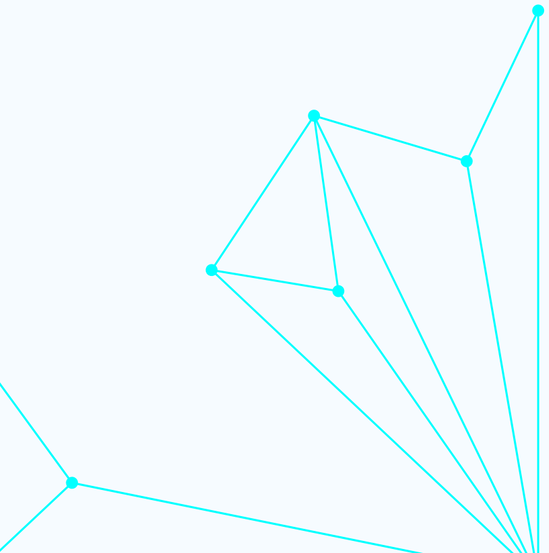
Output:

```
2 raised to 5 = 32
```


Dr.Kavitha N
Associate Professor
Dept. of IT

Parameter Passing Techniques

- Parameters are used to transfer data between the calling function and the called function.
- The data sent to a function is called Actual Parameters (arguments).
- The variables declared in the function definition are called Formal Parameters.
- When a function is called, actual parameters are mapped to formal parameters according to the chosen parameter passing technique.



Techniques in C :

C supports two primary parameter passing techniques**:**

1. Call by Value
2. Call by Reference

Dr.Kavitha N
Associate Professor
Dept. of IT

Call by Value

- Call by Value is a parameter passing technique in which the copy of the actual parameter's value is passed to the function.
- The function works on this copied value, and any changes made inside the function do not affect the original variable in the calling function.

Call by Reference

- Call by Reference is a parameter passing technique in which the address of the actual parameter is passed to the function.
- The function uses this address to directly access and modify the original variable's value in memory.

Example:

```
#include <stdio.h>
void val(int x) {    // Call by Value
    x = x + 5;
}
void ref(int *y) {   // Call by Reference
    *y = *y + 5;
}
int main() {
    int a = 10, b = 10;
    val(a);
    ref(&b);
    printf("After Call by Value: a = %d\n", a);
    printf("After Call by Reference: b = %d\n", b);
    return 0;
}
```

Output:

```
After Call by Value: a = 10
After Call by Reference: b = 15
```


Dr.Kavitha N
Associate Professor
Dept. of IT

Scope of Names in C

- The scope of a name in C refers to the region of a program where a variable, function, or identifier is accessible and valid for use.
- In other words, it defines where a variable can be used or modified within a program.

Type of Scope	Definition
1. Block Scope (Local Scope)	Variables declared inside a block ({ }) are accessible only within that block. They are created when the block is entered and destroyed when it ends.
2. Function Scope	Labels used with goto statements have function scope — they can be accessed anywhere within the same function.
3. File Scope (Global Scope)	Variables declared outside all functions are global and accessible from any function within the same file.
4. Function Prototype Scope	Variables declared inside a function's parameter list exist only within that prototype — they are temporary and local to the declaration.

Array of Functions in C

- In C, you cannot create an array of functions directly, but you can create an array of pointers to functions.
- This allows calling different functions dynamically using array indexing — often used in menus, callbacks, or command systems.

Example:

```
#include <stdio.h>

void add() { printf("Addition\n"); }
void sub() { printf("Subtraction\n"); }

int main() {
    void (*func[2])() = {add, sub};  // Array of function pointers
    func[0]();   // Calls add()
    func[1]();   // Calls sub()
    return 0;
}
```

Output:

```
Addition
Subtraction
```


Dr.Kavitha N
Associate Professor
Dept. of IT

Recursion

- Recursion is a programming technique where a function calls itself directly or indirectly to solve a problem.
- It continues until a specified base condition is met, which stops further recursive calls.

Syntax:

```
return_type function_name(parameters) {
    if (base_condition)
        return value;
    else
        function_name(arguments);   // recursive call
}
```

Characteristics / Nature of Recursion:

- Self-calling nature: The function invokes itself.
- Base condition: Prevents infinite recursion (must be defined).
- Stack usage: Each call is stored in the function call stack until it returns.
- Reduced problem size: Each call simplifies the problem.
- Termination: Stops when the base condition is true.

Example:

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

Output:

Factorial = 120

Dr.Kavitha N
Associate Professor
Dept. of IT

Types of Recursive Functions

Type of Recursion	Description	Example
1. Direct Recursion	A function calls itself directly.	factorial(n) calls factorial(n-1)
2. Indirect Recursion	A function calls another function, which in turn calls the first function.	funA() calls funB(), and funB() calls funA()
3. Tail Recursion	The recursive call is the last statement in the function; no operations are performed after it returns.	Used in factorial or sum calculations
4. Non-Tail Recursion	The recursive call is followed by additional operations after returning.	return n * factorial(n-1);
5. Nested Recursion	The function's recursive call uses another recursive function call as its argument.	fun(fun(n - 1));

Dr.Kavitha N
Associate Professor
Dept. of IT

Recursive Mathematical Functions

A recursive mathematical function is a function that solves mathematical problems by calling itself repeatedly until a base condition is reached.

Example:

```
#include <stdio.h>
int fibonacci(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
    for (int i = 0; i < 6; i++)
        printf("%d ", fibonacci(i));
    return 0;
}
```

Output:

0 1 1 2 3 5

Dr.Kavitha N
Associate Professor
Dept. of IT

Recursive Functions with Array Parameters

A recursive function with an array parameter is a function that processes array elements recursively, by operating on one element per call and moving towards the end (base case).

Output:

Sum = 20

Example:

```
#include <stdio.h>

int sumArray(int arr[], int n) {
    if (n == 0)
        return 0;                    // Base case
    else
        return arr[n - 1] + sumArray(arr, n - 1);
}

int main() {
    int a[] = {2, 4, 6, 8};
    int n = 4;
    printf("Sum = %d", sumArray(a, n));
    return 0;
}
```


Dr.Kavitha N
Associate Professor
Dept. of IT

Recursive Functions with String Parameters

- A recursive function with string parameters works by processing one character of the string at a time, and recursively calling itself for the remaining substring.
- Used for operations like string reversal, palindrome check, etc.

Example:

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

void reverse(char str[], int i) {
    if (i < strlen(str)) {
        reverse(str, i + 1);
        printf("%c", str[i]);
    }
}

int main() {
    char str[] = "C Language";
    reverse(str, 0);
    return 0;
}
```

Output:

egaugnaL C

Dr.Kavitha N
Associate Professor
Dept. of IT

Pointers

- A pointer is a variable that stores the memory address of another variable.
- Instead of holding a value directly, it holds the location where that value is stored in memory.

Syntax:

```
data_type *pointer_name;
```

Example:

```
int a = 10;
int *p = &a;
// p stores the address of variable a
printf("%d", *p);
// prints 10 (value at address stored in p)
```

Advantages of Pointers in C

- Enable dynamic memory allocation using malloc() and free().
- Allow call by reference, modifying actual arguments directly.
- Simplify array, string, and structure handling efficiently.
- Useful in creating linked lists, trees, and graphs.
- Provide faster access to memory and improve program efficiency.

Disadvantages of Pointers in C

- Complex syntax makes programs harder to read and debug.
- Improper use can cause memory leaks and crashes.
- Dangling pointers may lead to undefined behavior.
- Security risks if invalid memory locations are accessed.
- Hard to trace errors, making maintenance difficult.

Dr.Kavitha N
Associate Professor
Dept. of IT

Types of Pointers

Type	Description	Example
1. Null Pointer	Points to nothing (address = 0).	int *p = NULL;
2. Wild Pointer	Uninitialized pointer; may point anywhere.	int *p;
3. Void Pointer (Generic Pointer)	Can hold address of any data type.	void *ptr;
4. Dangling Pointer	Points to a memory location that is freed or deleted.	After free(ptr);
5. Constant Pointer	Address stored in pointer cannot change.	int *const p = &a;
6. Pointer to Constant	Value pointed to cannot change.	const int *p = &a;
7. Function Pointer	Points to a function instead of a variable.	int (*fptr)(int, int);
8. Double Pointer	Stores address of another pointer.	int **pp;

Dr.Kavitha N
Associate Professor
Dept. of IT

Pointer Arithmetic Operations

Pointer arithmetic refers to performing mathematical operations (like addition, subtraction, comparison) on pointer variables.

Example:

```
#include <stdio.h>
int main() {
    int arr[3] = {10, 20, 30};
    int *p = arr;

    printf("%d ", *p);   // 10
    p++;
    printf("%d ", *p);   // 20
    p += 1;
    printf("%d", *p);    // 30
    return 0;
}
```

Valid Operations:

1. Increment (++) - Moves pointer to next memory location.
   Ex: p++
2. Decrement (--) - Moves pointer to previous memory location.
   Ex: p--
3. Addition (p + n) - Moves pointer ahead by n elements.
4. Subtraction (p - n) - Moves pointer backward by n elements.
5. Pointer Difference (p2 - p1) - Gives number of elements between two pointers.

Output:

10 20 30

Dr.Kavitha N
Associate Professor
Dept. of IT

Pointers with Arrays

A pointer with an array means using a pointer to access or manipulate the elements of an array, since the array name itself acts as a pointer to its first element.

Example:

```
#include <stdio.h>
int main() {
    int arr[3] = {5, 10, 15};
    int *p = arr;

    for (int i = 0; i < 3; i++)
        printf("%d ", *(p + i));

    return 0;
}//Output:5 10 15
```

Pointers with Strings

A pointer with a string is a pointer that points to the first character of a string (character array) and can be used to traverse or modify it using pointer operations.

Example:

```
#include <stdio.h>
int main() {
    char str[] = "C Language";
    char *p = str;

    while (*p != '\0') {
        printf("%c", *p);
        p++;
    }
    return 0;
}//Output: C Language
```


Dr.Kavitha N
Associate Professor
Dept. of IT