### ****1. C Programming – Basics****

#### ****1.1 Language Fundamentals****

* Structure of a C program
* Data types and variables
* Constants and literals
* Operators and expressions
* Tokens, keywords, identifiers
* Data types, modifiers
* Input/Output (scanf, printf)
* Type casting

#### 1.2 Control Flow

* if, else, switch
* Loops: for, while, do-while
* break, continue, goto

#### 1.3 Functions and Scope

* Function declaration and definition
* Parameter passing (by value, by reference)
* Recursion
* Scope and lifetime of variables (local, global, static, extern)

#### 1.4 Arrays and Strings

* 1D and 2D Arrays
* String manipulation and functions
* Character arrays vs char\*
* String library functions

#### 1.5 Pointers

* Pointer declaration and dereferencing
* Pointer arithmetic
* Pointers to arrays and strings
* Double pointers
* Pointer to function
* Pointers and const
* Void pointers

### ****2. C Programming – Intermediate to Advanced****

#### 2.1 Structures, Unions & Bit-fields

* Structures and nested structures
* Unions and memory sharing
* Bit-fields and memory optimization
* Typedefs
* Nested structures

#### 2.2 Memory Management

* Static vs dynamic allocation
* Heap and stack
* Double/triple pointer handling

#### 2.3 Dynamic Memory Management

* malloc(), calloc(), realloc(), free()
* Memory leaks and tools to detect
* Dangling pointers and wild pointers

#### 2.4 Advanced Pointers

* Pointers to functions
* Callback functions
* Array of pointers to functions

#### 2.5 Preprocessor Directives & Macros

* #define, #include, #ifdef, #ifndef
* Macros vs inline functions
* File inclusion guards
* Conditional compilation
* Macros with arguments
* Multi-line macros

#### 2.6 Compilation Process

* Preprocessing, compiling, linking
* Object files and binary generation
* .h vs .c separation

#### 2.4 File I/O

* File handling with fopen(), fread(), fwrite(), fclose()
* Text vs binary mode

#### 2.5 Advanced Concepts

* const, volatile, restrict qualifiers
* Type qualifiers - static, extern, auto, register
* Memory alignment and padding
* Function pointers and callbacks
* Variable argument functions (va\_list)
* setjmp() and longjmp()

#### 2.6 Error Handling

* errno, perror, strerror
* Return codes vs exceptions

### ****3. C for Firmware Development****

#### 3.1 Embedded System Concepts

* Memory-mapped I/O
* GPIO and peripheral control
* Bit masking & shifting

#### 3.2 Embedded C Essentials

* Bitwise operations
* GPIO manipulation
* Register-level programming

#### 3.2 Memory and Optimization

* Stack vs Heap
* Static vs dynamic memory
* Code size vs execution speed

#### 3.3 Coding for Hardware

* Register-level coding
* Peripheral initialization
* Startup code and ISR vector table

#### 3.4 Interfacing & Protocols

* UART, I2C, SPI communication
* Interrupt handling in C
* Polling vs Interrupt-driven code

#### 3.5 Compiler & Build Process

* Toolchains (gcc, arm-gcc)
* Startup files and linker scripts
* Optimization flags
* Cross-compiling

#### 3.6 Coding for Hardware

* Register-level coding
* Peripheral initialization
* Startup code and ISR vector table

#### 3.7 Optimization Techniques

* Code size reduction
* Compiler optimization flags
* Inlining functions

#### 3.8 Embedded Build System

* Cross-compiling using gcc
* Linker scripts (.ld)
* Startup assembly file

#### 3.9 Bare-metal Programming

* Writing code without OS
* Interrupt vector tables
* Board support package (BSP)

### ****4. Device Driver Development (Linux Focus)****

#### 4.1 Kernel Programming Basics

* User space vs kernel space
* Kernel modules (insmod, rmmod, modinfo)
* Makefile for modules
* Linux kernel build system
* Kernel logging (printk)
* Module loading/unloading

#### 4.2 Character Device Drivers

* file\_operations structure
* open(), read(), write(), close()
* ioctl() usage
* Registering device numbers
* cdev, file\_operations
* Device node creation

#### 4.3 Memory and Interrupts

* copy\_to\_user, copy\_from\_user
* IRQ handling in kernel
* Debouncing and shared IRQs

#### 4.4 Interrupts and Synchronization

* Atomic operations
* request\_irq() and ISR
* Spinlocks, mutexes, semaphores
* Tasklets and workqueues
* Wait queues and completion APIs

#### 4.5 Memory Management in Kernel

* kmalloc(), kzalloc()
* get\_user() and put\_user()

#### 4.6 Platform and Misc Drivers

* Device Tree basics
* Writing platform drivers
* Sysfs and procfs interfaces

#### 4.7 Advanced Topics

* DMA programming
* Platform drivers and OF tree
* PCI/USB device drivers (Intro)

### ****5. C in Linux Environment****

#### 5.1 System Programming

* File descriptors and low-level I/O
* open(), read(), write(), close()
* File permission handling
* Processes and fork()
* Signals and handlers

#### 5.2 Processes

* fork(), exec(), wait(), exit()
* Zombie and orphan processes

#### 5.3 Signals

* Signal handling using signal()/sigaction()
* kill(), alarm(), pause()

#### 5.4 Inter-Process Communication (IPC)

* Pipes, FIFOs
* Shared memory
* Semaphores and Message queues
* Sockets

#### 5.5 Sockets

* TCP/UDP socket programming
* Server-client architecture

#### 5.6 Multithreading

* POSIX threads (pthread\_create, pthread\_join)
* Mutexes, condition variables
* Thread synchronization (mutex, cond vars)
* Race conditions and deadlocks

#### 5.7 Makefile and Project Structure

* Creating and managing Makefiles
* Static vs shared libraries
* Debugging with gdb, valgrind

### ****6. C with RTOS (Real-Time Operating Systems)****

#### 6.1 RTOS Basics

* Task creation and management
* Task priorities and scheduling
* Preemptive vs cooperative multitasking
* Context switching

#### 6.2 Scheduling and Timing

* Priority scheduling
* Round-robin scheduling
* Delays and timers

#### 6.3 Synchronization and Communication

* Semaphores (binary, counting)
* Mutexes and deadlock prevention
* Message queues and mailboxes
* Queues and mailboxes
* Event flags

#### 6.4 ISR and RTOS

* ISR to task signaling
* Deferred ISR using semaphores or queues

#### 6.5 RTOS APIs (Using FreeRTOS/RT-Thread/Zephyr/CMSIS-RTOS)

* xTaskCreate, vTaskDelay
* xSemaphoreGive, xSemaphoreTake
* ISR interaction with tasks
* Tick rate configuration
* Task control functions
* Queue APIs
* Hook functions
* Idle and tick hooks

#### 6.4 RTOS with Embedded C

* RTOS on ARM Cortex (e.g., STM32)
* Interrupts with RTOS
* Timer management
* Porting RTOS to a new platform