

INTUITIVE EMBEDDED TECH

Interactive Tech Education

www.intuitiveembedded.com

Call: 9900721303 / 9892616426 | Email: info@intuitiveembedded.com

Complete Course in C, C++ and DS (Data-Structure) with Linux

Total Duration for Complete Course: -

70 Hours (C) + 20 Hours (DS) + 40 Hours (C++) + 20 Hours (Debugging): => Approx. 150 Hours

NOTE: Practice will be done on Linux (Ubuntu)

Why 'C': -

C is a middle level language. C is the language of compilers, interpreters, editors, operating systems and embedded programming. When you learn to program in C you almost have to gain an understanding of how programs execute. You know what things like register, stack, heap and memory mapped IO mean. Because C has pointers, bitwise operators, and the keywords: extern, volatile, static, and register—meaning you'll understand more about writing efficient code than you can glean from any higher-level language. The only thing that will teach you these things any better is a low-level language, hence it is said C is very close to low level language. Hence "C" is must for anyone who aspires to become an IT professional.

C Topics

Duration: (70 Hours)

1. Getting Started

1. Overview of Programming Languages
 1. What is C LANGUAGE?
 2. Why C Language
 3. Types of Programming Languages.
 1. Machine Language
 2. Assembly Language
 3. High Level Language
2. Development Environment in Windows and Linux
 1. Editor
 2. Compiler
 3. Interpreter
 4. Linker
 5. Loader
 6. Debugger
 7. Profiler
 8. Libraries
3. Getting Familiar with Linux
 1. Understanding the Linux Environment
 2. Basic Linux Commands
 3. Getting familiar with Vim Editor
 4. Basic Vim Editor Command
 5. Understanding 'Make' Command
 6. Writing Make Files.

2. The C Language

1. C Program Compilation and Execution Process
2. Tokens of C Program
3. C Instructions
4. Constants, Variables, Identifiers and Keywords
5. Primitive Data Types
6. Different types of operators and their precedence.
7. Expressions and Sub-Expressions
8. Variable declaration, definition and Initialization
9. Statements
 1. If, if-else
 2. for loop
 3. While, do-while
 4. Switch, break, continue, goto

3. Functions

1. What Is function? Invoking Functions
2. Passing Arguments to Functions
3. Call by Value & call by Reference
4. Is C call by Value? Or Is C call by Reference?
5. Recursion and Recursive function call (Basic)
6. Inter-function communication patterns
7. Rules for Evaluation of Function Arguments
8. Static Vs Dynamic Runtime Environment
9. Function Call and Runtime Stack

4. Introduction to Pointers

1. What is Pointers?
2. Pointer – Memory Allocation
3. Pointer – Declaration & Initialization
4. Pointer – Dereferencing
5. Efficient way of Printing Pointer
6. Rules for Pointers

5. Array (One-dimensional and Multi-dimensional)

1. Why Arrays? Array – Representation
2. Array – Memory Allocation
3. Array – Declaration & Initialization
4. Array as Function Argument
5. Rules for Array Argument Passing
6. Searching algorithm
7. Sorting algorithm
8. Two Dimensional Arrays
9. Multi-dimensional Array Argument Passing

6. Strings

1. Declaring and using Strings
2. Different types of operations on Strings
3. Command Line parameters
4. String library
5. String Handling Functions
6. String Conversion Functions

7. User Defined Types (UDT)

1. Structures
2. Unions
3. Enumerations
4. Declaring and defining UDTs and using typedef
5. Applications of UDTs

8. Advanced Topics on Pointers

1. Pointers & Function
 1. Pointer as Function parameter
 2. Function returning Pointer
 3. Pointer to functions
 4. Function Pointers
 1. Declaration and Usage of Function Pointers
 2. typedef'ing a Function Pointer
 3. Function Pointers as Function Parameters
 4. Practical Example of Function Pointers
2. Pointers & Arrays
 1. Pointers with One-dimensional array
 2. Pointers with N-dimensional array
 3. Character Arrays
 1. Character Arrays using Pointers
 2. Array of Character Pointers
 3. Memory Diagram – Array of Char Pointers
 4. Character Array Vs String
 5. Passing / Assessing/ Returning Char Array from Function
 4. Strings
 1. Declaring and using Strings
 2. Different types of operations on Strings
 3. Command Line parameters
 4. String library
 5. String Handling Functions
 6. String Conversion Functions
 7. Efficient usage of -
 1. sscanf ()
 2. sprintf ()
 3. snprintf
 4. strncat etc..
 8. Character Array Vs String / Character Array as String
 5. Various Array operations -
 1. Find the number of elements in an array
 2. Find the size of an array
 3. Inserting Element in an array
 4. Removing Element from an array
 5. Rotating Element of an array
 6. Bit-wise operations with array
 7. itoa and atoi and strtol (for int, for hex, for char etc..)
 8. Adding/removing a set of strings stored in an array (for e.g. comma, quote, etc...)
 10. Buffer manipulation functions (Library function)
 - memset() / memcpy()/ memmove()/ memicmp()/ memchr()
 6. Arrays as Pointers – a[i] == i[a]?
 7. Pitfalls with arrays and pointers
 8. Pointer to an Integer Array
 9. Array of Integer Pointer

- 10. Various experiments
- 3. Pointers & Structure
 - 1. Accessing Structure Elements using pointer
 - 2. Array of Structure
 - 3. Pointer to Structure
- 4. Void pointer
- 5. Constant Pointer and pointer to constant
- 6. Pointer Aliasing
- 7. Pointer Arithmetic
- 8. Double pointer
- 9. Pitfalls with pointers
- 10. type-casting
- 11. C Pointers Complexity Chart
 - int **p
 - int (*p)()
 - int (*p[]) int *p()
 - int *(*p[])()
 - int *(*p)[]

9. Advanced Topics on Primitive Datatypes

- 1. Data-type Alignments
- 2. Compiler Memory Allocation for Data-types
- 3. Range of Signed/Unsigned Data-types
- 4. Programming Model & Memory Sizes
- 5. Why Sizeof Int and Long is 4 or 8?
- 6. Use of long long in 32-bit Architecture
- 7. Practical Example of long long
- 8. IA-32, IA-64, ILP-32, LP64, x86-64
- 9. Accessing, printing every bits and bytes of Char and Int

10. Advanced Topics on UDT

- 1. Compiler Memory Allocation for Structures
- 2. Compiler Memory Allocation for Unions
- 3. Union – Data Corruption
- 4. Bit-Fields
 - 1. Bit-Fields in Structure
 - 2. Practical Usage and examples of Bit-fields
 - 3. Bit-fields Overflow
 - 4. Printing every bytes of an Integer
 - 5. Practical examples of Bit-field Usage
- 5. Structure Padding & Pitfalls
- 6. Passing and Returning Structure / Union form Function
- 7. Operations on structures
- 8. #pragma pack () definition and its usage

11. Dynamic Memory Allocations (DMA)

- 1. What is Dynamic Memory Allocations? Usage
- 2. malloc, calloc, realloc, free
- 3. malloc Vs calloc
- 4. Heap Memory
- 5. Stack Memory – Pitfalls
- 6. Dangling Pointers
- 7. DMA – Errors
- 8. Best Practices for malloc() & free()

9. DMA – Unspecified Behaviour

12. Pre-processing and Header Files

1. Header files
 1. Why Header files?
 2. Examples of pre-built and custom built header files
 3. Multiple Inclusion of a Header File?
2. Different types of pre-processor directives
 1. Preprocessor – #include statements
 2. Preprocessor – #define statements
 3. Preprocessor – Conditional Compilation
 4. Conditional Directives for Debugging
3. Macros
 1. Preprocessor – Nested Macros
 2. Preprocessor – Multiline Macros
 3. Preprocessor – Stringizer
 4. Preprocessor – Token Concatenation
 5. Preprocessor – Useful Directives
 6. Where Macros are Heavily Used
 7. Practical Examples of Macros
 8. Macros Pitfalls
 9. Macros Vs Enums
4. Inline Function
 1. Inline Functions
 2. Macros Vs Inline
 3. Inline Recursive Functions

13. Other Advance Topics

1. Computing Basic
 1. Binary & Octal Systems
 2. Decimal & Hexadecimal Systems
2. Signed Representations in Memory
 1. Binary Shifts – Right & Left
 2. Sign Bits and Bit-Shift Operations
 3. Right Shift – Logical Vs Arithmetic Shift
 4. Bit-Shift Overflow
 5. ASCII Representations
 6. Endian-ness – Little Vs Big
 7. Endian-ness – Portability Issues
3. Operators
 1. Bitwise Operations
 2. Logical Operators – Short Circuit
 3. Bitwise Vs Logical Operations
 4. sizeof() operator
 5. Pitfalls/Issues with sizeof () usage
 6. Pointer Increment & Scaling
 7. Operator Precedence & Operator Associativity
 8. Examples of Precedence & Associativity
 9. Ternary Operator & Ternary Operator Associativity Rule
4. Data-type Conversion Rules
 1. Float to Int to Float Conversions
 2. Variadic functions & default promotion rules
 3. Printf, Scanf idiosyncrasy
 4. Pointer Format Specifiers

5. Signed Vs Unsigned – Pitfalls
5. Evaluation of $i = ++i + ++i$
6. Concept of Sequence Points with Examples
7. Memory Organization
 1. Code Segment
 2. Data Segment
 3. Heap Segment
 4. Stack Segment
 5. Stack Frames and stack corruption
 6. Calling Sequence
 7. View of Runtime Stack with Example
 8. Access to Local Variable in Stack
8. Storage Classes
 1. Storage Class Specifiers
 2. Scope of a Variable
 3. Register, Auto, Static, Extern
 4. Why Register Class and Practical Examples
 5. Automatic Variables and Stack
 6. Static Variables and Functions
 7. True meaning of Extern
 8. How to Use extern across Multiple Files with Examples
 9. Best Practices for Extern Usage
 10. Local/Block/Global Scope
 11. Nesting of Scope
 12. Lifetime of a Variable
 13. Linkage of a Variable
9. Qualifiers
 1. Const Qualifier
 1. What is Const?
 2. Practical Examples of Const Qualifier
 3. Usage of Constant in library functions (libc)
 2. Volatile Qualifier
 1. What is Volatile?
 2. Practical Examples of Volatile Qualifier
 3. Const Volatile Together?
 4. Register Vs Volatile Performance?
 5. Practical Examples of Const Volatile
 3. Restrict Qualifier
 1. What is Restrict Qualifier?
 2. Restrict Keyword and Compiler Optimization
 3. Examples of Restrict Qualifier
10. Different stages of c compilation
11. Command Line Argument
12. Environment Variables in C Programs
13. Big-O Notation (Code/Space/Time Complexity) (Basic)
14. C99 Features
 1. Variable length argument
 2. variable-length arrays
 3. Flexible array members
 4. Inline functions
 5. Boolean data type
 6. Restrict qualification

15. Int, char validation functions (Library function)
 - isalpha() / isdigit() / isalnum() / isspace() / islower() / isupper() / isxdigit() / iscntrl() / isprint() / ispunct() / isgraph() / tolower() / toupper() etc..
16. Typecasting Functions
 - atof() / atoi() / atol() / itoa() / ltoa() etc..
17. Other Important Miscellaneous function (Library function)
 - getenv() / setenv() / putenv() / perror() / rand() / delay() etc..

14. Recursion

1. Power of Recursion
2. Recursion v/s Iteration
3. Recursion Example
4. Tower of Hanoi
5. Sorting Algorithms using Recursion

15. File Handling

1. Files & Streams
2. Streams Buffers
3. IO Buffers – Line Vs Full Vs No-Buffers
4. Setting & Flushing Buffers
5. File Access
6. File Access Modes
7. Sequential Vs Random Access
8. Concept of File Offsets
9. File Operation Errors
10. End-of-File Condition?
11. Return Values and Error Values
12. Character Based File I/O
13. Line Based File I/O
14. Formatted File I/O
15. Block File I/O
16. Dangerous – gets () Vs fgets ()
17. File Random Access Methods
18. Library functions

16. Modular programming

1. Difference between Procedural and Modular Programming
2. Defining and using modules in C
3. Separate Compilation and role of header files
4. Member access across files

C++ Topics

Duration: (40 Hours)

17. C++

1. An overview of C++
2. C vs C++
2. Flexible variable declarations
3. Reference variables
4. cin, cout, cerr, clog

18. Object-Based Programming

1. Class and Object
2. Static / Non-Static Function and Static / Non-Static Variable
3. Constructor, Destructor
4. Copy constructor
5. This Pointer
6. New and Delete Function
7. Const Object, Const Function
8. Static and instance members
9. Function Overloading and Operator Overloading
10. Object Composition
 1. Linking
 2. Embedding
11. Heap Manager
12. Mutable
13. Namespace, using, Namespace aliasing

19. Object-Oriented Programming

1. Introduction to Inheritance
2. Multiple Inheritance
3. Liskov's Principle of Substitution
4. Virtual function
5. Polymorphism
 1. Compile Time Polymorphism
 2. Runtime Polymorphism
6. Abstract Class and methods
7. Issues related to Multiple Inheritance
8. Interface
9. RTTI (Run Time Type Identifiers)
10. Introduction to Templates
12. Exception Handling

Data-Structures Topics

Duration: (20 Hours)

20. Data-Structures

1. Stack
2. Queue
3. Circular Queue
4. Ring Buffer
5. Dynamic Data-Structures
 1. Simple Linked List
 2. Doubly Linked List
 3. Implementing Stack using Dynamic Memory Allocation (i.e. from Linked List)
 4. Implementing Queue using Dynamic Memory Allocation (i.e. from Linked List)
6. Binary Tree
7. Various Sorting and Searching Techniques

Debugging Technique in C

Duration: (10 Hours)

21. Linux and C Debugging Technique

1. Libraries
 1. Static Vs Dynamic Library
 2. Static Library Generation
 3. Dynamic Library Generation
 4. Linking with Libraries
2. Make, clean and Makefile
3. File Formats
 1. ELF Format
 2. Symbol Tables
 3. Anatomy of a Process
 4. Process Map
 5. Memory Layout
 6. Code Segment
 7. Data Segment
 8. Stack Segment
 9. Heap Segment
4. Debugging Tools
 1. GCC, GDB and Breakpoints
 2. Valgrind
 3. Gprof, MemProf
 4. strace, Pmap / Pstack
5. Analysis
 1. Object Dump
 2. Stack Frames
 3. Mapping Assembly to C
 4. Managing Heap
 5. Debugging Running Process
 6. Core dump analysis
 7. Symbols & Optimizations
 8. Default Optimization Levels
6. Debugging
 1. Common Failures
 2. Failure Notifications
 3. Common Faults
 4. Segmentation Violation
 5. Stack Overflow
 6. Heap Overflow
 7. Arithmetic Overflow
 8. Illegal code execution
 9. Stack Issue
 10. Endian-ness issue
 11. C library assertion
 12. Data Type mismatch

THANK YOU