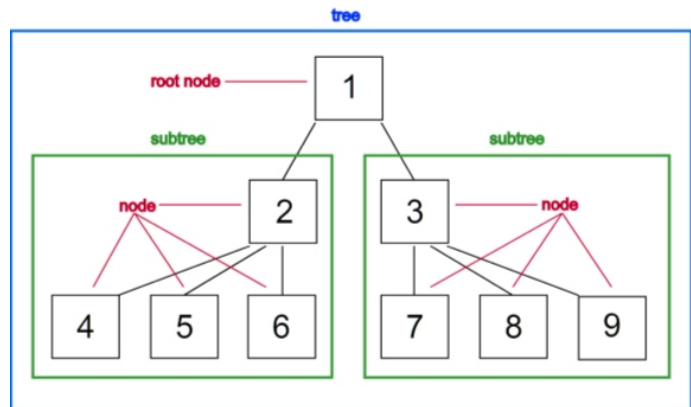


C & C++



Corporate Trainer's Profile

Corporate Trainers are having the experience of 4 to 12 years in development , working with TOP CMM level 5 companies (Project Leader /Project Manager) qualified from NIT/IIT/IIM and work exp in USA and UK.



Capability Maturity Model level Project Standard*** :-

The Capability Maturity Model (CMM) is a method for evaluating the maturity of organizations on a scale of 1 to 5.

Get the Opportunities to work on Client Projects Of US/UK, which follow the all standard of CMM level 5 Company.

Projects



Embedded C Programming with Real Time Linux (i86) A 5-day Course

Synopsis

This C course is extensive and contains many advanced concepts. The range of modules covers a full introduction to C, real-time and embedded systems concepts through to the design and implementation of real time embedded or standalone systems based on real-time operating systems and their device drivers. Real time Linux (RTLinux) is used as an example of such a system. The modules include an introduction to the development of Linux device drivers.

Predominantly intended to be taught to development teams at customer sites it is not expected that any one course would cover the full range of modules in a typical five day period. For teams without experience of C and high-end real time operating systems it would typically require eight to ten days of intensive training to give full coverage to the topics included here. The course covers all of the important features of the C language as well as a good grounding in the principles and practices of real-time systems development including the POSIX threads (pthreads) specification.

The design of the modules is intended to provide an excellent working knowledge of the C language and its application to serious real time or embedded systems. Those wanting in-depth training specifically on RTLinux or Linux kernel internals should contact us to discuss their requirements; this set of modules is geared more towards providing the groundwork for approaching those domains rather than as in-depth training on a specific approach.

The course contains essential information for anyone developing embedded systems such as microcontrollers, real-time control systems, mobile device, PDAs and similar applications. This C course is based on many years experience of teaching C, extensive industrial programming experience and also participation in the ANSI X3J11 and BSI standards bodies that produced the standard for C. We focus on the needs of day-to-day users of the language with the emphasis being on practical use and delivery of reliable software.

Suitable for

Programmers and engineers who already have some understanding of programming and who now wish to gain a solid understanding of the use of C for embedded and real-time systems software development.

Prerequisites

Programming experience

A good understanding of real-time programming issues

Delivery

This is instructor led C training. Each section of the material covered by the tutor is followed by hands-on practical exercises for which worked examples of the solutions are typically provided.

This embedded C course is available only as an in-house course. For details of this course and bespoke C courses that we can provide/

Contents

- An Introduction to C
- Overview
- The C Programming Language
- The C Standards
- The Importance of Standards
- A Simple C Program
- Comments
- Preprocessor Directives
- Functions
- main
- stdio
- Libraries
- printf
- Strings
- Return Values
- Statements
- Review
- Introduction to Embedded C Programming
- Overview
- Embedded C and Standard C
- Simple Embedded C Program
- Practical Compiling the Program
- Differences Between the Standard and Embedded Versions
- Benefits of Using C for Embedded Systems
- C in the Embedded Environment
- C in the Real Time Linux Environment
- Learning Standard C
- Variable Types and Constants
- Overview
- Variables
- Basic Variable Types
- Other Variable Types
- Sizes of Data Types
- Declaring Variables
- Declaring Multiple Variables
- Variable Names
- Initialising Variables at Declaration
- Character Constants
- Named Constants
- Boolean Type Truth Values
- Casting Type Conversion
- Decimal, Octal, and Hexadecimal Notations
- Operators
- Overview
- Operators

- Expressions
- Assignment Operator
- Chaining Assignments
- Arithmetic Operators
- Arithmetic Operators: Division and Remainder
- Example of Remainder
- Increment and Decrement Operators
- Postfix Increment and Decrement
- Relational Operators
- Logical Negation
- Logical Operators
- Short-Circuiting
- Compound Assignment Operators
- Flow Control
- Overview
- The if Statement
- Assignment and Equality
- Nested if Statements
- Multi-Way if Statements
- The for Loop
- for Loop Syntax
- The while Loop
- The do-while Loop
- Infinite Loops
- break Statement
- continue Statement
- Conditional Operator
- switch Statement
- switch Caveats
- Shortened Statements
- if Statement Trap
- goto Statement
- Functions
- Overview
- The Rôle of Functions
- Function Libraries
- Functions Syntax
- Return Values
- Function Arguments
- Local Variables
- Scope and Lifetime
- Functions Calling Functions
- Call-by-Value
- Definition and Declaration
- Definition After Use
- Prototypes and Old-Style Declarations

- Variadic Functions
- Arrays
 - Overview
 - Array Basics
 - Array Example
 - Looping Over an Array
 - Array Sizes
 - Copying Arrays
 - Initialising Arrays
- Strings
 - Array Caveats
 - Evaluating Expressions
 - Truth Values
 - Precedence
 - Associativity
 - Operand Evaluation
 - Casting - Type Conversion
 - Operator Precedence Table
 - Operator Precedence Table Continued
- Bit Manipulation
 - Objectives
 - Introduction
 - Bit Twiddling Operators
 - Bit Shifting
 - Portable Bit Manipulation
 - Print Bit Representations
 - Example Bit Manipulation
 - Summary
- Pointers
 - Overview
 - What is a Pointer?
 - Creating a Pointer
 - Pointer Types
 - Uninitialised Pointers
 - Null Pointers
 - Pointers as Function Arguments
 - Pointers and Arrays
 - Pointer Arithmetic
 - Using sizeof
 - Array/Pointer Equivalence
 - Passing Arrays to Functions
 - Pointers to Constant Data
 - Passing Pointers-to-Const
 - Converting Pointers-to-Const
 - Other Data Types: Structures, Unions, and Enums
 - Overview
 - Defining Structures

- Using Structures
- Initialising Structure Variables
- Problems with Initialising Structures
- Initialising Arrays of Structures
- Pointers to Structures
- Passing Structures to Functions
- Passing Structure Pointers
- Structures as Data Types
- Enumerations
- Unions
- Discriminating a Union
- The Preprocessor and Multiple-File Programs
- Overview
- Standard Headers
- The Preprocessor
- The #define Directive
- Large Programs
- Deciding How to Split a Program
- Sharing Declarations
- Using Header Files
- Writing a Header File
- Conditional Compilation
- Module-Private Functions
- Module-Private Variables
- Function-Private Variables
- Public Module Variables
- When to Use Global Variables
- String Handling
- Passing Strings to Functions
- String Handling
- Standard String Functions
- Character Classes
- Available Character Classes
- Letter Case
- Converting a String to a Number
- Converting a Number to a String
- String Input Real-Time - C Techniques for Dealing with Time
- What is Real-Time?
- Characteristics of Real-Time Applications
- Synchronising I/O with CPU Via Polling
- Synchronising I/O with CPU Via Interrupts
- Interrupts Capabilities of the 16f877 Pic
- Handling Interrupts with C2C
- Interrupts in Real Time Linux
- Generating Delays Via Software
- C Delay Loops
- Generating Delays Via Hardware
- Generating Delays in RTOS

Example Flash Led Using Timer with Interrupt
Introducing Multi-Tasking
Real-Time Operating Systems
RTLinux Kernel
Scheduling Algorithms
Tasks, Threads, POSIX Pthreads
Basics
Why Tasks?
Multitasking Example
Our Terminology
Processes and Threads Analogy
Synchronisation, Scheduling and Races
Thread Basics
Creating Posix Pthreads
Creating Threads Example
Example of Multiple Threads
Shared Data Problems
Race Conditions
Dealing with Critical Sections
Example of Unprotected Critical Section
Protecting Critical Sections
Drawbacks to Mutexes
Using Condition Variables
Operations on Condition Variables
Semaphores
Spinlocks
Debugging Multithreaded Programs
Thread-Safe Functions
Deadlocks
Priority Inversion
Self-Referential and Dynamic Data Structures
Objectives
Common Data Structures
Circular Buffers
Race Conditions
Singly Linked List
Data Allocation
Using malloc and free
Heap allocation example
Details of malloc and free
Supporting Functions
Doubly Linked Lists
Developing on Real Time Linux
Real Time Linux Environment
Typical RTOS Features
Core RTOS Facilities
RTLinux

- Architecture of RTLinux Application
- Linux Module Example
- Compiling and Running Modules
- Tiers of Services
- Creating a RTLinux Thread
- Simple Parallel Port Manipulation
- Requesting IO Ports
- Measuring Time
- Elapsed Time in RTLinux
- Interprocess Communication in Real Time Linux
- Basics of IPC
- FIFOs
- Shared Memory in RTLinux
- Allocating Shared Memory
- User-Level Shared Memory
- Shared Memory Thread
- Shared Memory Module
- IPC Practical
- Servo Control Example
- Servo Controlled Via IPC
- Software Interrupts
- Software Interrupt API
- Real (Hard) Interrupts
- Hard Interrupt Example
- Linux Device Drivers
- Linux Device Drivers
- Driver Types
- Device Number
- Driver Initialisation
- Module Load/Unload
- Providing Module Parameters
- Open and Release Functions
- Read and Write
- Write
- Reading from Device
- Read Code
- Tasklet and Bottom Half Code
- Interrupt and Tasklet Code
- Further Pointers
- Pointers to Pointers
- Using Pointers to Pointers to Structures
- Pointers to Constant Values
- Constant Pointers
- Increment Operators and Pointers
- Pointers to Functions
- Initialising Function Pointer Values
- Calling Functions Through Pointers
- Call-Back Functions

- Arrays of Pointers to Functions
- Function Pointers for State Machines
- State Machine Example: a Toaster
- Building Software with Make
- Introduction
- Different Versions of Make
- A Simple Project
- Software Building Concepts
- Targets and Prerequisites in Make
- Building an Object File with Make
- Pattern Rule Variables
- Building an Executable with Make
- Rebuilding When Prerequisites Change
- Default Rules
- Substitution References
- Implicit Rules
- Configuring Implicit Rules
- Adding More Rules
- Advanced Features
- Advanced Use of the Preprocessor
- Parametric Macros
- Problems with Macros
- Macros and Scope
- Macros and Precedence
- Testing Assertions
- Stringification
- Assertions with Side Effects
- Token Pasting
- When to Use Parametric Macros
- Efficient C Programming
- Basics
- Principal Causes of Inefficiency
- Architecture Issues
- Efficient Algorithms
- Coding Tactics and Compilers
- Difficulties with Code Optimisation
- Common Tricks - Subexpressions
- Using register Variables
- Loop Unrolling
- Space-Time Tradeoffs
- Floating-Point Arithmetic
- Compiler Optimisation
- Avoid Pass-by-Value of Large Objects
- MISRA C Designing Safer C Programs
- In This Module Will Will Cover:
- MISRA C for Safety-Related Programming
- MISRA C - Rules

- Environment Rules
- Character Sets and Comments
- Identifiers and Types
- Constants
- Declarations and Definitions
- Initialisation
- Operators
- Expressions
- Control Flow
- Functions
- Pre-Processor Directives
- Pointers and Arrays
- Structures and Unions
- Standard Libraries
- MISRA C Tools
- References for MISRA C and Safer