



Embedded C Programming, Linux, and Vxworks.

Synopsis

This 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. Vxworks is used as an example of such 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 two month period. For teams without experience of C and high-end real time operating systems it would typically require three months 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 IPC specification.

The design of the modules is intended to provide an excellent working knowledge of the embedded C language and its application to serious real time systems. Those wanting in-depth training other system applications and kernel internals should contact us to discuss their requirements; this set of modules is geared more towards providing the groundwork for approaching those domains.

The course contains essential information for anyone developing embedded systems such as microcontrollers, real-time control systems, mobile device, PDAs and similar applications. This embedded 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

- A degree (B.E., B.Tech, MCA, M.Tech) in Electronics/Electrical, Computer Science or Information Technology.
- Programming experience
- Working capability on any one operating systems (Windows, Linux).

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.

Our embedded C courses now have a monthly public schedule for Pennsylvania (USA) and Noida (India). Please see www.miracleindia.com for details.

For details of the in-house and bespoke C courses that we can provide, please email

info@miracleindia.com

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

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 in Real Time O/S
- Generating Delays Via Software
- C Delay Loops
- Generating Delays Via Hardware
- Generating Delays in RTOS
- Introducing Multi-Tasking
- Real-Time Operating Systems
- RTOS Overview
- Scheduling Algorithms

Tasks, Threads,

- Basics
- Why Tasks?
- Multitasking Example
- Processes and Threads Analogy
- Synchronisation, Scheduling and Races
- Thread Basics
- 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

- 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

Arrays

- Overview
- Array Basics
- Array Example
- Looping Over an Array
- Array Sizes
- Copying Arrays
- Initializing Arrays
- Strings
- Array Caveats

Evaluating Expressions

- Truth Values
- Precedence

- Details of malloc and free
- Doubly Linked Lists

Developing on Real Time Operating Systems

- Real Time O/S Environment
- Typical RTOS Features
- Core RTOS Facilities
- VxWorks
- Architecture of Vxworks Application
- Vxworks Module Example
- Compiling and Running Modules
- Tiers of Services
- Creating a Vxworks Thread
- Simple Parallel Port Manipulation
- Requesting IO Ports
- Measuring Time
- Elapsed Time in Vxworks

Inter-process Communication in Real Time Operating Systems

- Basics of IPC
- FIFOs
- Shared Memory in Vxworks
- 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

Vxworks Device Drivers

- Vxworks Device Drivers
- Driver Types
- Device Number
- Driver Initialization
- 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

- 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

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

Understanding Embedded Systems

- Introduction to Embedded Systems
- What is Unique about the design goals for Embedded Software?
- Developing an Embedded Application
- Microprocessor versus Micro controller.

Processor Architecture and Memory

Organization

- Processors and Memory Organization.
- Von Neumann and Harvard Architecture.
- RISC vs. CISC
- Processor Selection of an Embedded System.
- ARM Architecture.

Linux Internals

- Device Driver and Interrupt Service Mechanism.
- Linux Internals Kernel Programming
- Inter Process Communication

- 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

NOTE: The trainee has to sign the Non disclosure Agreement(NDA) before joining any industrial project.

Features:

- The study material and references will be provided to the trainees by Miracle Embedded Systems.
- This reference material is developed by our corporate trainers and software engineers from top notch industries.
- There will regular attendance of the student. One has to show at least 75% attendance then only he will be eligible for certification.
- The grades will be assigned on the basis of regular test results.
- There will be recognitions and rewards for well performing candidates.

Duration

3.0 Months

NOTE:- All training will be given by Corporate Trainers only .

All trainer from IIT (M.Tech.) background having four to seven years experience in respected field ,working with CMM Level 5 companies

Corporate Client: -L&T InfoTech , Motorola, Samsung Electronic, ST Microelectronic and many more

The major aims of the Program are to:

1. Provide a strong foundation in the emerging disciplines of RTOS, Embedded Systems and its applications for professionals in the software development industry.
2. Will improve the required skill set of the system software professional.
3. Program will provide perspective in RTOS, Embedding Systems and its applications in up coming field like Telecom, Wireless communications, Network Management, Automation and Process Controls.
4. Incorporates the required skills and experience on Embedded Systems and RTOS in the professionals for the exponentially growing industrial demands on this line.