NEXT
CAREER DEVELOPMENT ACADEMY

EMBEDDED SYSTEM ENGINEER CAREER

Table of Content

C Programming Essentials

Days:	1 Day
Duration:	4 Hours
Language:	English

Job Description

The C programming language was developed in 1972 by Dennis Ritchie at Bell Laboratories; one of its first uses was in the rewriting of the UNIX operating system.
So, strictly speaking, C is a system programming language. However, it is also useful for application programming. On completion of this course, the students will know how to write useful programs using the language.

Job Profile Outcome

- Demonstrate ability to develop and write useful C code.
- Successfully manage to use C statements, operators, structures, functions, pointers, and data type
- Use C Preprocessors when required
- Write and read from files

Job Outline

INTRODUCTION

- The Origins of the C Language
- Differences between the various versions of C
- C is a medium level language
- Key words
- Structure
- Comments
- Libraries
- Terms

DATA TYPES

- Identifiers
- Declaring a variable
- Formal parameters
- Access modifiers
- Storage class types
- Variable initializations
- Constants
- Backslash character constants
- Assignment statements

OPERATORS

- Precedence of operators

FUNCTIONS

- Passing an array to a function
- Passing parameters to main
- Returning from a function
- Function prototypes
- The importance of prototyping

PROGRAM CONTROL STATEMENTS

- Selection statements
- Iteration statements
- Jump statements
- More about ?
- Continue

ARRAYS

- Single-dimension arrays
- Passing single-dimension arrays to functions
- Strings
- Two-dimensional arrays 33
- Passing two-dimensional arrays to functions
- Arrays of strings
- Array initialization

THE C PREPROCESSOR

- define 35
- #error
- #include
- #if, #else, #elif, #endif

- #ifdef, #ifndef
- #undef
- #line
- #pragma
- Predefined macro names

INPUT AND OUTPUT

- Input
- Output
- Direct console I/O

POINTERS

- Pointer variables
- Pointer arithmetic
- Pointer comparisons
- Dynamic memory allocation
- Arrays and pointers
- Problems with pointers

STRUCTURES

- Declaration of structures
- Passing structures elements to functions
- Passing entire structures to functions
- Structure pointers
- Arrays and structures within structures
- Bit fields
- Unions
- Enumerations
- typedef

FILE I/O

- Buffered streams
- Random access using streams
- Handles
- Advanced file I/O
- Predefined I/O streams

LINKING, LIBRARIES AND HEADER FILES

- The linker
- Declaration versus definition
- Relocatable versus absolute code
- Library files versus object files
- Header files

STRINGS

- Strtok()
- Converting numbers to And from strings
- Text handling

DYNAMIC MEMORY ALLOCATION

UNIX and Linux Essentials

Days:	1 Day
Duration:	4 Hours
Language:	English

Job Description

This UNIX and Linux Essentials course is designed for users and administrators who are new to the Oracle Solaris 11 and Oracle Linux operating systems. It will help you develop the basic UNIX skills needed to interact comfortably and confidently with the operating system.

Job Profile Outcome

- View and modify file and directory permissions
- Describe the UNIX operating system
- Work with files and directories
- Use the vi editor to create and modify files
- Use commands within the default shell
- Manage processes
- Use advanced shell features in shell scripts
- Archive files and perform remote file transfer

Job Outline

Introduction to UNIX

- Overview of the UNIX Operating System
- Executing Commands from the Command Line

Working with Files and Directories

- Determining Where You Are in the Directory Structure
- Viewing File Content
- Copying Files and Directories
- Creating and Removing Files and Directories
- Searching Files and Directories

Using the vi Editor

- Introducing the vi Editor
- Modifying Files with the vi Editor

Using Commands within the Default Shell

- Using Shell Expansion
- Using Variables in the Bash Shell
- Displaying the Command History
- Redirecting Commands
- Working with User Initialization Files

Using Basic File Permissions

- Viewing File and Directory Permissions
- Changing the Permissions
- Modifying Default Permissions

Performing Basic Process Control

- System Processes Overview
- Managing Processes

Using Advanced Shell Features in Shell Scripts

- Using Advanced Shell Features
- Creating Shell Scripts

Archiving Files and Performing Remote Transfer

- Archiving and Retrieving Files
- Compressing, Viewing, and Uncompressing Files
- Performing Remote Connections and File Transfers

Embedded Software Engineering

Days:	2 Days
Duration:	8 Hours
Language:	English

Job Description

This Course gives you the techniques and technologies in software engineering to optimally design and implement your embedded system. Written by experts with a solutions focus, this encyclopedic reference gives you an indispensable aid to tackling the day-to-day problems when using software engineering methods to develop your embedded systems.

Job Profile Outcome

- The principles of good architecture for an embedded system
- Design practices to help make your embedded project successful
- Details on principles that are often a part of embedded systems, including digital signal processing, safety-critical principles, and development processes
- Techniques for setting up a performance engineering strategy for your embedded system software
- How to develop user interfaces for embedded systems
- Strategies for testing and deploying your embedded system, and ensuring quality development processes
- Practical techniques for optimizing embedded software for performance, memory, and power
- Advanced guidelines for developing multicore software for embedded systems
- How to develop embedded software for networking, storage, and automotive segments
- How to manage the embedded development process

Job Outline

Software Engineering of Embedded and Real-Time Systems

- Software engineering
- Embedded systems
- Real-time systems
- Challenges in real-time system design
- Distributed and multi-processor architectures
- Software for embedded systems
- Hardware abstraction layers (HAL) for embedded systems

Embedded Systems Hardware/Software Co-Development

- Today's embedded systems — an example
- HW/SW prototyping Users
- HW/SW prototyping options
- Prototyping decision criteria
- Choosing the right prototype
- Industry design chain
- The need to change the design flow
- Different types of virtual prototypes

- A brief history of virtual prototypes
- The limits of proprietary offerings
- What makes virtual prototypes fast
- Standardization: the era of SystemC TLM-2.0
- Architecture virtual prototypes
- Software virtual prototypes

Software Modeling for Embedded Systems

- When and why should you model your embedded system?
- Modeling
- What is a modeling language?
- Examples of modeling languages
- The V diagram promise
- So, why would you warn to model your embedded system?
- When should you model your embedded system?
- Operational complexity
- Cost of defect versus when detected
- Large development learns require modeling
- Modeling is often the only choice
- So — modeling is great, but aren't all models wrong?
- You have your prototype — now what?
- Conclusion
- Next steps — try it!
- Software Design Architecture and Patterns for Embedded Systems
- Overview of architecture and design
- Three levels of design
- What are design patterns?
- Software architecture categories and views

Embedded Systems: Introduction to ARM Microcontrollers

Days:	2 Day
Duration:	8 Hours
Language:	English

Job Description

This course teach the fundamentals of embedded systems as applied to the ARM® Cortex™-M family of microcontrollers, it designs of software in assembly language and C, elementary data structures, programming input/output including interrupts, analog to digital conversion, digital to analog conversion.

Job Profile Outcome

- Introduction to Computers and Electronics
- Introduction to Embedded Systems
- Introduction to the ARM© Cortex™-M Processor
- Introduction to Input/Output
- Modular Programming
- Pointers and Data Structures
- Variables, Numbers, and Parameter Passing
- Analog I/O Interfacing
- Communication Systems

Job Outline

Introduction to Computers and Electronics

- Review of Electronics
- Binary Information Implemented with MPS transistors
- Digital Logic
- Digital Information stored in Memory
- Numbers
- Character information
- Computer Architecture
- Flowcharts and Structured Programming
- Concurrent and Parallel Programming
- Exercises

Introduction to Embedded Systems

- Embedded Systems
- Applications Involving Embedded Systems
- Product Life Cycle
- Successive Refinement
- Quality Design
- Debugging Theory
- Switch and LED Interfaces
- Introduction to C
- Exercises

Introduction to the ARM© Cortex™-M Processor

- Cortex™-M Architecture
- The Software Development Process
- ARM Cortex-M Assembly Language
- Simplified Machine Language Execution .
- CISC versus RISC
- Details Not Covered in this Book
- Exercises

Introduction to Input/Output

- Texas Instruments Microcontroller I/O pins
- Basic Concepts of Input and Output Ports
- Phase-Lock-Loop
- SysTick Timer
- Standard I/O Driver and the printf Function
- Debugging monitor using an LED
- Performance Debugging
- Exercises
- Lab Assignments

Modular Programming

- C Keywords and Punctuation
- Modular Design using Abstraction
- Making Decisions
- Assembly Macros
- Recursion
- Writing Quality Software
- How Assemblers Work
- Functional debugging
- Exercises
- Lab Assignments

Pointers and Data Structures

- Indexed Addressing and Pointers
- Arrays
- Strings
- Structures
- Finite State Machines with Linked Structures
- Dynamically Allocated Data Structures
- Matrices and Graphics
- Tables
- Functional Debugging
- Exercises
- Lab Assignments

Variables, Numbers, and Parameter Passing

- Local versus global
- I/O Synchronization
- Interrupt Concepts
- Interthread Communication and Synchronization
- NVIC on the ARM Cortex-M Processor
- Edge-triggered Interrupts
- SysTick Periodic Interrupts
- Timer Periodic Interrupts
- Hardware debugging tools
- Profiling
- Exercises
- Lab Assignments

Analog I/O Interfacing

- Approximating continuous signals in the digital domain
- Digital to Analog Conversion
- Music Generation
- Analog to Digital Conversion
- Real-time data acquisition
- Exercises
- Lab Assignments

Communication Systems

- Introduction
- Reentrant Programming and Critical Sections
- Producer-Consumer using a FIFO Queue
- Serial port interface using interrupt synchronization.
- Distributed Systems
- Exercises
- Lab Assignments

C Programming for Embedded Microcontrollers

Days:	2 Day
Duration:	8 Hours
Language:	English

Job Description

The C Programming Language Technology is constantly changing. New microcontrollers become available every year and old ones become redundant. The one thing that has stayed the same is the C programming language used to program these microcontrollers.

Job Profile Outcome

- Your First C Program
- C Basics
- Comparative Operators and Decisions
- The while Loop
- Functions
- Number Systems
- Memory and Microcontrollers
- Your First Embedded C Program
- Embedded I/O & Memory Maps
- The DBGU Serial Port
- Previous C Topics Revisited
- Arrays and Strings
- Bit Manipulation and Logical Operators
- More Hardware Programming
- Wrapping Up

Job Outline

Your First C Program

- About the Programming Tools
- Downloading the DJGPP Compiler
- Installing the DJGPP Compiler
- Setting Up the DJGPP Compiler
- Installing the Compiler from the Elektor Website
- How C Programs are Created
- Start Programming
- Compiling the Program
- About Your First C Program
- Analysing the Program

C Basics

- Setting up Programmer's Notepad
- How the sticky.c Program Works
- Input, Output and Variables
- Variable Types
- Field Width Specifiers
- Compiling and Linking
- Errors and Warnings
- Link Errors

Comparative Operators and Decisions

- Comparative Operators
- Decisions

The while Loop

- The while Loop
- Using if Inside the while Loop
- The Guess My Number Game
- Back to the Temperature Controller Example
- Commenting Programs
- Programming Style

Functions

- Your Second Function
- Passing Data to a Function
- Passing More Than One Value to a Function
- Passing a Variable to a Function
- Getting a Value Back from a Function
- Passing Values to a Function and Receiving a Value Back
- Flashing LED Simulation Program
- Pre-processor Directives
- Functions Calling Functions
- Using Multiple Source Files

- Header Files
- The make Program and Make File
- How Functions Relate to Linking and Library Files

Number Systems

- Binary Basics
- The Need for Binary Numbers
- Numbering Systems
- Working with Hexadecimal Numbers in C
- The ASCII Alphanumeric Code

Memory and Microcontrollers

- Memory Basics
- A Look at a Memory Chip
- How Microprocessors Access Memory and Peripherals
- Pointers
- More on C Data Types
- Choosing a Microcontroller and Embedded System

Your First Embedded C Program

- How Embedded Programming Differs from PC Programming
- The Embedded C Programming Tools
- The YAGARTO Toolchain
- Writing Your First Embedded Program

Embedded I/O & Memory Maps

- Loading a Program into SRAM
- Writing to More than One LED
- Reading the Switches and Writing to the LEDs (I/O)
- The AT91SAM7S Memory Map
- A Closer Look at the PIO Controller

The DBGU Serial Port

- Hardware Requirements for PC to uC Serial Communications
- Programming the Serial Port
- Using the sprintf () Function
- Receiving Data on the Serial Port

Previous C Topics Revisited

- Serial Port Driver
- Format Specifiers
- Escape Sequences
- Loops
- Nested Loops and Decisions
- Decision Making with the s witch Statement
- The Conditional Operator
- Functions and Pointers
- Variables and Scope
- Static Variables
- Floating Point Data Types
- Casts

Arrays and Strings

- Arrays
- Strings
- Arrays and Addresses
- Strings as Pointers
- A Look at the DBGUTxMsg () Function
- Multidimensional Arrays

Bit Manipulation and Logical Operators

- Bit Manipulation with Bitwise Operators
- Logical Operators
- Operator Precedence

More Hardware Programming

- The AT91SAM7S Timer Counter
- The Analogue to Digital Converter (ADC)
- Using the Timer and Interrupt
- The Watchdog Timer

Wrapping Up

- Structures
- Unions
- Enumerated Type
- The typedef Declarator
- Storage Class Specifiers
- Type Qualifiers
- The goto Statement
- A List of All C Keywords
- More Preprocessor Directives
- Debugging
- Some Final Example Programs

Embedded Systems: Real-Time Interfacing

Days:	2 Day
Duration:	8 Hours
Language:	English

Job Description

This course teaches the fundamentals of embedded systems as applied to the ARM Cortex-M family of microcontrollers.

Job Profile Outcome

- Introduction to Embedded Systems
- ARM Cortex M Processor
- Software Design
- Hardware-Software Synchronization
- Interrupt Synchronization
- Time Interfacing
- Serial Interfacing
- Analog Interfacing
- System-Level Design
- Data Acquisition Systems
- Introduction to Communication Systems

Job Outline

- Introduction to Embedded Systems
- ARM Cortex M Processor
- Software Design
- Hardware-Software Synchronization
- Interrupt Synchronization
- Time Interfacing
- Serial Interfacing
- Analog Interfacing
- System-Level Design
- Data Acquisition Systems
- Introduction to Communication Systems

Embedded Systems: Real-Time Operating Systems RTOS

Days:	2 Day
Duration:	8 Hours
Language:	English

Job Description

Embedded systems are a ubiquitous component of our everyday lives. We interact with hundreds or tiny computers every day that are embedded into our houses, our cars, our toys, and our work. As our world has become more complex, so have the capabilities of Uic microcontrollers embedded into our devices. The ARM* Cortex™-M family represents the new class of microcontrollers much more powerful than the devices available ten years ago. The purpose of this book is to present the design methodology to train young engineers to understand the basic building blocks that comprise devices like a cell phone, an MP3 player, a pacemaker, antilock brakes, and an engine controller.

Job Profile Outcome

- Review of Computer Architecture
- Design of I/O Interfaces
- Software Design
- Real-Time Operating Systems
- Digital Signal Processing
- High-Speed Interfacing
- File system management
- Interfacing Robotic Components
- High-Speed Networks
- Robotic Systems

Job Outline

Review of Computer Architecture

- Embedded Systems
- Computer Architecture
- Cortex™-M Processor Architecture
- Stellaris® LM3S8962 I/O pins
- ARM® Cortex™-M Assembly Language
- ARM® Cortex™ Microcontroller Software Interface Standard

Design of I/O Interfaces

- Flowcharts
- Digital Logic and Open Collector
- Parallel I/O ports
- Phase-Lock-Loop
- NVIC on the ARM® Cortex™-M Processor
- SysTick Timer
- Edge-triggered Interfacing
- Configuring Digital Output Pins
- UART Interface
- Synchronous Transmission and Receiving using the SSI
- DAC Operation and Performance Measures
- Analog to Digital Converters
- Data Acquisition Systems

Software Design

- The Design Process
- The Design Process
- Threads
- First In First Out Queue
- Interthread Communication and Synchronization

- Critical Sections
- Critical Sections

Real-Time Operating Systems

- Fundamentals
- Round-Robin Scheduler
- Semaphores
- Thread Synchronization and Communication
- Monitors
- Fixed Scheduling
- OS Considerations for I/O Devices

Digital Signal Processing

- Basic Principles
- Audio Input/Output
- Audio Input/Output
- Using the Z-Transform to Derive Filter Response
- IIR Filter Design Using the Pole-Zero Plot
- Discrete Fourier Transform
- FIR Filter Design
- Direct-Form Implementations

High-Speed Interfacing

- High-Speed Interfacing
- High-Speed I/O Applications
- General Approaches to High-Speed Interfaces
- Fundamental Approach to DMA
- Programming Flash EEPROM
- Secure digital card interface
- Camera Interface

File system management

- Introduction
- File System Allocation
- Simple File System
- File Allocation Table
- Internal Fragmentation
- External Fragmentation

Interfacing Robotic Components

- Input Capture or Input Edge Time Mode
- Output Compare or Periodic Timer
- Pulse Width Modulation
- Binary Actuators
- Sensors
- Odometry

High-Speed Networks

- Fundamentals
- Controller Area Network (CAN)
- Embedded Internet
- Exercises
- Lab Assignments

Robotic Systems

- Introduction to Digital Control Systems
- Simple Closed-Loop Control Systems
- PID Controllers
- Fuzzy Logic Control

Linux for Embedded and Real-time Applications

Days:	3 Day
Duration:	12 Hours
Language:	English

Job Description

From the viewpoint of programming, embedded systems show a number of significant differences from conventional "desktop" applications. For example, most desktop applications deal with a fairly predictable set of I/O devices—a disk, graphic display, a keyboard, mouse, sound card, perhaps a network interface.

Job Profile Outcome

- The Embedded and Real-time Space
- Introducing Linux
- The Host Development Environment
- BlueCat Linux
- Debugging Embedded Software
- Kernel Modules and Device Drivers
- Embedded Networking
- Introduction to Real-time Programming
- The RTAI Environment
- Posix Threads

Job Outline

The Embedded and Real-time Space

- What Is Embedded?
- What Is Real-time?
- How and Why Does Linux Fit in?
- Resources

Introducing Linux

- Features
- Protected Mode Architecture
- The Linux Process Model
- The Linux Filesystem
- The "root" User
- The /usr hierarchy
- The Shell
- Resources

The Host Development Environment

- Cross-Development Tools—the GNU Tool Chain
- Configuring and Building the Kernel

BlueCat Linux

- The "Less Is More" Philosophy
- Installing BlueCat Linux
- X86 Target for Blue Cat Linux
- Configuring the Workstation
- First Test Program
- Directories
- Configuration Files
- Makefile
- Target Files

Debugging Embedded Software

- The Target Setup
- GDB
- Debugging a Sample Program
- The Host as a Debug Environment
- Adding Programmable Setpoint and Limit

Kernel Modules and Device Drivers

- Kernel Modules
- What's a Device Driver Anyway?

- Linux Device Drivers
- Internal Driver Structure
- The Hardware
- The Target Version of Thermostat
- Debugging Kernel Code
- Building Your Driver into the Kernel
- An Alternative—uCLinux

Embedded Networking

- Sockets
- A Simple Example
- A Remote Thermostat
- Embedded Web Servers

Introduction to Real-time Programming

- Polling vs. Interrupts
- Tasks
- Scheduling
- Kernel Services
- Inter-task Communication
- Problems with Solving the Resource Sharing Problem Priority Inversion
- Interrupts and Exceptions
- Critical Sections
- Linux and Real-time

The RTAI Environment

- Installing RTAI
- The RTAI Architecture
- Intertask Communication and Synchronization
- Communicating with Linux Processes
- Real-time in User Space—LXRT
- One Shot vs. Periodic Timing
- Moving to Kernel Space
- Real-time FIFOs and Shared Memory

Posix Threads

- Synchronization—Mutexes
- Communication—Condition Variables
- Pthreads in User Space
- Moving to Kernel Space
- Message Queues
- Suggestions for Further Exploration

Embedded Android

Days:	3 Day
Duration:	12 Hours
Language:	English

Job Description

Putting Android on an embedded device is a complex task involving an intricate understanding of its internals and a clever mix of modifications to the Android Open Source Project (AOSP) and the kernel on which it runs, Linux. Before we get into the details of embedding Android, however, let's start by covering some essential background that embedded developers should factor in when dealing with Android, such as Android's hardware requirements, as well as the legal framework surrounding Android and its implications within an embedded setting. First, let's look at where Android comes from and how it was developed.

Job Profile Outcome

- Introduction
- Internals Primer
- Internals Primer
- The Build System
- Hardware Primer
- Native User-Space

Job Outline

Introduction

- History
- Features and Characteristics
- Development Model
- Ecosystem
- Getting "Android"
- Legal Framework
- Hardware and Compliance Requirements
- Development Setup and Tools

Internals Primer

- App Developer's View
- Overall Architecture
- Linux Kernel
- Hardware Support
- Native User-Space
- Dalvik and Android's Java
- System Services
- Stock AOSP Packages
- System Startup

Internals Primer

- Development Host Setup
- Getting the AOSP
- Inside the AOSP
- Build Basics
- Running Android
- Using the Android Debug Bridge (ADB)
- Mastering the Emulator

The Build System

- Comparison with Other Build Systems
- Architecture
- Build Recipes
- Basic AOSP Hacks

Hardware Primer

- Typical System Architecture
- What's in a System-on-Chip (SoC)?
- Memory Layout and Mapping
- Development Setup
- Evaluation Boards

Native User-Space

- Filesystem
- Adb
- Android's Command Line
- Init
- Android Framework
- Kick-Starting the Framework
- Utilities and Commands
- Support Daemons

Embedded Systems Testing and Validation

Days:	2 Day
Duration:	8 Hours
Language:	English

Job Description

This course gives you the techniques and technologies in software engineering to optimally design and implement your embedded system. Written by experts with a solutions focus, this encyclopedic reference gives you an indispensable aid to tackling the day-to-day problems when using software engineering methods to develop your embedded systems.

Job Profile Outcome

- What is software test?
- Available techniques
- Setting the standard
- Dealing with the unusual
- Implementing a test solution environment

Job Outline

What is software test?

- Why should we test software?
- How much testing is enough?
- When should testing take place?
- Who makes the decisions?

Available techniques

- Static and dynamic analysis
- Requirements tractability
- Static analysis — adherence to a coding standard
- Essential knots & essential cyclomatic complexity — case study
- Understanding dynamic analysis
- The legacy from high-integrity systems
- Defining unit, module and integration tests
- Defining structural coverage analysis
- Achieving code coverage with unit test and system test in tandem
- Retaining the functionality through regression test
- Unit test and test-driven development
- Automatically generating test cases

Setting the standard

- The terminology of standards
- The evolution of a recognized process standard
- Freedom to choose adequate standards

Dealing with the unusual

- Working with auto-generated code
- Working with legacy code
- Tracing requirements through to object code verification (OCV)

Implementing a test solution environment

- Pragmatic considerations
- Considering the alternatives