Embedded C

Mentor: Indhira N G

Short Description:

Welcome to the Embedded C course! In this course, you'll dive into the fascinating world of embedded systems programming using the C language. Embedded systems are everywhere around us, from smartphones to medical devices and cars. This course will equip you with the essential skills and knowledge needed to design, develop, and troubleshoot software for these embedded systems.

Description:

Throughout the course, you'll learn the fundamentals of C programming with a specific focus on its application in the embedded domain. We'll cover topics like data types, control flow, functions, memory management, and input/output operations.

Additionally, you'll explore how to interface with hardware components and peripherals, write efficient and optimized code, and debug embedded systems using various tools and techniques.

By the end of this course, you'll have a solid foundation in embedded C programming and the ability to build robust and reliable software for a wide range of embedded systems. Whether you're interested in pursuing a career in embedded systems development or simply want to explore this exciting field, this course will provide you with the necessary skills to excel.

Get ready to unlock the potential of embedded systems through the power of C programming!"

Total Duration : 15 Hrs | Modules : 10

Related Tags : Embedded Systems , C Programming , Microcontrollers , Real-Time Systems , Embedded Software , Low-Level Programming , Peripherals.

Modules and Description:

Module - 1 | Duration: 1 Hr

1. Introduction to Embedded C
2. IDE Installation

Key Description:

Introduction to Embedded C: This module provides an overview of embedded systems and introduces the C programming language.
IDE Installation: In this module, you will learn how to set up and configure an Integrated Development Environment (IDE) for embedded C programming.

Module - 2 | Duration: 2 Hrs

1. Data types and Variables
2. Address variables and Storage classes
3. Functions & Type Casting

Key Description :

Data types and variables : This module explores the different data types available in embedded C, such as integers, floating-point numbers, characters and teaches how to declare and manipulate variables to store and manipulate data effectively.
Address variables and storage classes : In this module, you'll learn about the concept of memory addresses in embedded C, how to access and manipulate them and understand storage classes like static, and extern, which determine the lifetime and scope of variables.
Functions and type-casting : Discover the power of functions in embedded C, including their role in code modularity and reusability, and explore type-casting techniques for converting data between different types to ensure proper data manipulation and compatibility.

Module - 3 | Duration :

1. Microcontroller and Hello world
2. Build process and analysing embedded C code

Key Description :

Microcontroller and Hello world: This module covers the classic 'Hello World' program as an introduction to embedded C development, and guides you through the build process, including compiling, linking, and generating the binary code that can be loaded onto the target microcontroller.

Build process and analyzing embedded C code : Discover techniques for analyzing and understanding embedded C code.

Module - 4 | Duration :

1. Pointers

Key Description:

Pointers: Pointers in embedded C allow you to directly access and manipulate memory locations, enabling efficient memory management, data sharing, and interaction with hardware peripherals.

Module - 5 | Duration:

1. Operators
2. Decision making loops
3. Bitwise operators

Key Description:

Operators: Operators in embedded C are symbols or keywords used to perform various mathematical, logical, and relational operations on data, enabling complex computations and data manipulation within embedded systems.

Decision making loops : Decision-making loops in embedded C, such as if-else and switch statements, provide the ability to make choices and execute different blocks of code based on specific conditions, enhancing program flexibility and responsiveness.

Bitwise operators: Bitwise operators in embedded C allow manipulation and analysis of individual bits within variables, enabling efficient handling of flags, bit-level operations, and optimizing memory usage in embedded systems.

Module - 6 | Duration:

1. Embedded C coding exercise for Blinking LED
2. Bitwise Shift Operators

3. Looping
4. Type Qualifier 'Const'

Key Description:

Embedded C coding exercise for Blinking LED: This coding exercise involves writing an embedded C program to control a microcontroller's GPIO pins, implementing a blinking LED functionality, providing hands-on experience in configuring hardware.

Bitwise Shift Operators: Bitwise shift operators in embedded C, such as left shift (<<) and right shift (>>), enable efficient manipulation of data at the bit level, allowing shifting of bits to the left or right positions, useful for tasks like multiplying or dividing by powers of 2.

Looping : Looping structures, like for, while, and do-while loops, in embedded C provide a way to repeat a block of code multiple times, allowing efficient iteration over data, implementing timed delays, and performing repetitive tasks in embedded systems.

Type Qualifier 'Const': The type qualifier 'const' in embedded C allows the declaration of variables that are read-only or constant, ensuring that their values cannot be modified once initialized, providing safety, optimization, and improved code readability.

Module - 7 | Duration:

1. Pin-read and Optimization
2. 'volatile' type Qualifier

Key Description:

Pin-read and Optimization: In embedded C, pin-read refers to reading the state of a specific GPIO pin on a microcontroller, allowing the system to detect input signals or monitor the status of external devices. Optimization techniques can be applied to minimize latency and maximize the efficiency of pin-read operations for improved system responsiveness.

'volatile' type Qualifier : The 'volatile' type qualifier in embedded C is used to indicate that a variable can be modified by external factors outside the control of the program, such as hardware interrupts or concurrent tasks. It ensures that the

variable is always read from and written to memory, preventing the compiler from optimizing it in ways that could lead to unexpected behavior in the presence of such external modifications.

Module - 8 | Duration:

1. Structures and Bit field
2. Unions

Key Description:

Structures and Bit field: Structures in embedded C allow grouping related variables together, while bit fields enable efficient packing and manipulation of individual bits within a structure, useful for handling hardware registers or compact data storage.

Unions: Unions in embedded C provide a way to store different types of data in the same memory space, allowing efficient memory utilization and type conversion, often used in embedded systems for data sharing or protocol handling.

Module - 9 | Duration:

1. Usage of Bit filed in embedded code

Key Description:

Usage of Bit filed in embedded code: The usage of bit fields in embedded code allows efficient utilization of memory, compact representation of data structures, and manipulation of individual bits, commonly employed in scenarios such as register configuration or protocol parsing.

Module - 10 | Duration:

1. Keypad Interfacing
2. Arrays & Strings
3. Pre-processor Directives in C

Key Description:

Keypad Interfacing: Keypad interfacing in embedded C involves connecting and communicating with a keypad matrix to detect user input, enabling the implementation of user interfaces and user interaction in embedded systems.

Arrays & Strings: Arrays in embedded C provide a way to store and manipulate multiple elements of the same data type, while strings are arrays of characters. They are extensively used in embedded programming for data storage, manipulation, and communication tasks.

Pre-processor Directives in C : Pre-processor directives in embedded C, such as #define, #ifdef, or #include, enable conditional compilation, macro definition, and header file inclusion, enhancing code modularity, flexibility, and reuse.