C Programming

Duration: 5 days *(Face-to-Face & Remote-Live)*, or 35 Hours *(On-Demand)*

Price: $2495 *(Face-to-Face & Remote-Live)*, or $1495 *(On-Demand)*

Discounts: We offer multiple discount options. Click here for more information.

Delivery Options: Attend face-to-face in the classroom or remote-live attendance.

Students Will Learn

- Components of a C program
- Using the C preprocessor
- Using standard runtime libraries
- Using `make` to build programs
- Working with debugger utilities
- Using data types, storage classes and scope
- Using `typedef` to make code more readable and portable
- Using operators and expressions
- Working with conditional and looping constructs
- Initializing a pointer
- Accessing the value addressed by a pointer
- Returning the value of a function
- Declaring argument data types
- ANSI function prototype syntax
- Declaring and initializing arrays and multidimensional arrays
- Using Strings and character manipulation
- Declaring and instancing a structure
- Defining a union
- Accessing command line arguments and environment variables
- C runtime library standard I/O functions

Course Description

This hands on C programming course provides a comprehensive introduction to the ANSI C language, emphasizing portability and structured design. Students are introduced to all major language elements including fundamental data types, flow control, and standard function libraries. Thorough treatment is given to the topics of string and character manipulation, dynamic memory allocation, standard I/O, macro definition, and the C runtime library. The course explains the use of aggregate structures, unions, and pointers early on so the students can practice extensively in the hands on labs. Structured programming constructs and varargs functions are also covered. Emphasis is given to the processing of command line arguments and environment variables so students will be able to write flexible, user-friendly programs. The course also includes coverage of portability tips drawn from experienced programmers working in production environments.

Comprehensive hands on exercises are integrated throughout to reinforce learning and develop real competency.

Course Prerequisites

Understanding of fundamental programming concepts.

Course Overview

Overview of C

- Operating System Independence
- Design Goals and Capabilities
- Flavors of C

Fundamental Data Types, Storage Classes, and Scope

- Fundamental Data Types and Qualifiers
- Constants and Strings
- Storage Classes
- Scope and Block Structure
- Scope and Data Hiding
- Data Initialization

Macros

- Functions vs. Inlining
- Purpose of Macros
- Use of Macros
  - Making Code More Readable
  - Auto Adjustment of Compile Time Values
  - Conditional Compilation
  - Making Code Portable
  - Simplifying Complex Access Calculations
- Advanced Micro Design Tips
- Using Macros to Help Write Portable Programs
- When to Use a Macro instead of a Function
- Using Macros for Debugging

Basic Formatted I/O

- Standard I/O Library
- Character Set Encoding
- Standard Input and Output
- Character I/O Functions
- Formatted I/O Functions

Compiler Directives and the C Preprocessor

- Compile-Time Directives
- Use of `typedef`
- C Preprocessor Syntax

Pointers and Dynamic Allocation

- Advantages of Pointers
- User of Pointers
- Pointer and Address Arithmetic
- Dynamic Storage Allocation
- `sizeof` Operator
- Double Indirection

Arrays

- Purpose of Arrays
- Declaring an Array
- Initializing an Array
- Addressing Elements
- Stepping Through an Array
- Variable Size Arrays
- Arrays of Pointers
- Arrays of Strings
- Passing an Array to a Function
- Dynamic Memory Allocation
- Multidimensional Arrays

Program Debugging

- Problem Analysis
- Instrumenting with `printif`
- Instrumenting with `ctrace`
- The Purpose of Debuggers
- How Not to Use Debuggers

String Constants

Symbolic Debuggers

Operators and Expressions

- Arithmetic, Logical, and Bit Operators
- Precedence and Associativity
- Assignment and Casting
- The Conditional Operator

Flow Control Constructs

- Conditional Constructs: `if`, `switch`
- Looping Constructs: `while`, `do`, `for`
- Programming Style

Functions (Subroutines)

- Purpose of Functions
- Functions vs. Inlining
- Automatic Variables
- The Argument Stack
- Passing By Value
- Passing By Reference
- Declaring External Functions
- Function Prototyping
- ANSI Prototyping
- The `_NO_PROTO` Compiler Symbol
- Varargs Functions
- Passing a Function as an Argument
- Designing Functions for Reusability
- Calling a Function from Another Language
- Returning a Dynamically Allocated Value Using Double Indirection
- Casting the Return Value of a Function
- Recursion and Reentrancy

Structures

- Purpose of Structures
- Defining and Declaring Structures
- Accessing Members
- Pointers to Structures
- Dynamic Memory Allocation
- Passing a Structure to a Function
  - As a Pointer
  - Passing the Actual Structure

Advanced Structures and Unions

- Nested Structures
- Arrays of Structures
- Bit Fields
- Unions
- Linked Lists

C Runtime Library Standard Functions

- Character I/O
- Unformatted File I/O
- Formatted File I/O
- Math Functions
- Miscellaneous Functions

Strings and Character Manipulation

- Strings as Character Arrays
- String Library Functions
- Reading and Writing Strings

Accessing Command Line Arguments and Environment Symbols

- `argc` and `argv`
- Parsing Command Line Options
- Accessing the Environment Array

Structured Programming

- Structuring Code for Quality, Reliability, Maintainability
- Designing for Modularity and Reusability

Advanced Programming Consideration

- Writing Portable Code
- Use of Macros
- ANSI C Limits
- Feature Test Macros
- Client/Server Design
- Performance Considerations

Hands On Technology Transfer
The Best Way to Transfer Technology Skills

1 Village Square, Suite 8
14 Fletcher Street
Chelmsford, MA 01824