SRM SYSTEMS®
way to success...

C Programming 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
- String Constants

Operators and Expressions

- Arithmetic, Logical, and Bit Operators
- Precedence and Associativity

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
- Symbolic Debuggers

Flow Control Constructs

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

- Assignment and Casting
- The Conditional Operator

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

Advanced Structures and Unions

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

Strings and Character Manipulation

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

Structured Programming

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

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

C Runtime Library Standard Functions

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

Accessing Command Line Arguments and Environment Symbols

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

Advanced Programming Consideration

- Writing Portable Code
- Use of Macros
- ANSI C Limits
- Feature Test Macros

- Client/Server Design
- Performance Considerations

C++ Programming Course Overview

Moving from C to C++

- New Compiler Directives
- Stream Console I/O
- Explicit Operators
- Standard Libraries
- Data Control Capabilities

Handling Data

- New Declaration Features
- Initialization and Assignment
- Enumerated Types
- The `bool` Type
- Constant Storage
- Pointers to Constant Storage
- Constant Pointers
- References
- Constant Reference Arguments
- Volatile Data
- Global Data

Functions

- Function Prototypes and Type Checking
- Default Function Data Types
- Function Overloading
- Problems with Function Overloading
- Name Resolution
- Promotions and Conversions
- Call by Value
- Reference Declarations
- Call-by-Reference and Reference Types
- References in Function Return
- Constant Argument Types
- Conversion of Parameters Using Default Initializers
- Providing Default Arguments
- Inline Functions

Creating and Using Objects

- Creating Automatic Objects
- Creating Dynamic Objects
- Calling Object Methods
- Constructors
- Initializing Member consts
- Initializer List Syntax
- Allocating Resources in Constructor
- Destructors
- Block and Function Scope
- File and Global Scope
- Class Scope
- Scope Resolution Operator ::
- Using Objects as Arguments
- Objects as Function Return Values
- Constant Methods
- Containment Relationships

Dynamic Memory Management

- Advantages of Dynamic Memory Allocation
- Static, Automatic, and Heap Memory
- Free Store Allocation with new and delete
- Handling Memory Allocation Errors

Controlling Object Creation

- Object Copying and Copy Constructor
- Automatic Copy Constructor
- Conversion Constructor

Inheritance

- Inheritance and Reuse
- Composition vs. Inheritance
- Inheritance: Centralized Code
- Inheritance: Maintenance and Revision
  - Public, Private and Protected Members
  - Redefining Behavior in Derived Classes
  - Designing Extensible Software Systems
- Syntax for Public Inheritance
- Use of Common Pointers
- Constructors and Initialization
- Inherited Copy Constructors
- Destructors and Inheritance
- Public, Protected, Private Inheritance

Streaming I/O

- Streams and the `iostream` Library
- Built-in Stream Objects
- Stream Manipulators
- Stream Methods
- Input/Output Operators
- Character Input
- String Streams
- Formatted I/O
- File Stream I/O
- Overloading Stream Operators
- Persistent Objects

Introduction to Object Concepts

- The Object Programming Paradigm
- Object-Orientated Programming Definitions
- Information Hiding and Encapsulation
- Separating Interface and Implementation
- Classes and Instances of Objects
- Overloaded Objects and Polymorphism

Templates

- Purpose of Template Classes
- Constants in Templates
- Templates and Inheritance
- Container Classes
- Use of Libraries

Strings in C++

- Character Strings
- The String Class
- Operators on Strings

Exceptions

- Types of Exceptions
- Trapping and Handling Exceptions
- Triggering Exceptions

- Member Functions of the String Class
- Handling Memory Allocation Errors

C++ Program Structure

- Organizing C++ Source Files
- Integrating C and C++ Projects
- Using C in C++

Reliability Considerations in C++ Projects

- Function Prototypes
- Strong Type Checking
- Constant Types
- C++ Access Control Techniques

Polymorphism in C++

- Definition of Polymorphism
- Calling Overridden Methods
- Upcasting
- Accessing Overridden Methods
- Virtual Methods and Dynamic Binding
- Virtual Destructors
- Abstract Base Classes and Pure Virtual Methods

Multiple Inheritance

- Derivation from Multiple Base Classes
- Base Class Ambiguities
- Virtual Inheritance
  - Virtual Base Classes
  - Virtual Base Class Information

Declaring and Defining Classes

- Components of a Class
- Class Structure
- Class Declaration Syntax
- Member Data
- Built-in Operations
- Constructors and Initialization
- Initialization vs. Assignment
- Class Type Members
- Member Functions and Member Accessibility
- Inline Member Functions
- Friend Functions
- Static Members
- Modifying Access with a Friend Class

Operator Overloading

- Advantages and Pitfalls of Overloading
- Member Operator Syntax and Examples
- Class Assignment Operators
- Class Equality Operators
- Non-Member Operator Overloading
- Member and Non-Member Operator Functions
- Operator Precedence
- The this Pointer
- Overloading the Assignment Operator
- Overloading Caveats

The Standard Template Library

- STL Containers
- Parameters Used in Container Classes
- The Vector Class

- STL Algorithms
- Use of Libraries