C Programming

This C Programming Course covers the C Programming language comprehensively. C is popular with amateur and commercial programmers alike. Learn to use this powerful and popular language, and get an idea of the kinds of environments in which you will most likely find yourself in when in a C coding job.

Prerequisites

You should not be a complete beginner for this course. If you cannot pass this test, you must do Intro To Programming first.

Course Material

Provided

Course Contents

DAY 1:

Overview and background of the C programming language

- Design Features
- Advantages
- Shortcomings
- What Computers Do
- High-level Computer Languages and Compilers
- Language Standards
- The First ANSI/ISO C Standard
- The C99 Standard
- The C11 Standard
- Commentary
- Programming Mechanics
- Object Code Files, Executable Files, and Libraries
- Unix System
- The GNU Compiler Collection and the LLVM Project

- Linux Systems
- Command-Line Compilers for the PC
- Integrated Development Environments (Windows)
- The Windows/Linux Option
- C on the Macintosh

Introducing C

- A Simple Example of C
- The Example Explained
- Pass 1: Quick Synopsis
- Pass 2: Program Details
- The Structure of a Simple Program
- Tips on Making Your Programs Readable
- Taking Another Step in Using C
- Documentation
- Multiple Declarations
- Multiplication
- Printing Multiple Values
- While You’re at It—Multiple Functions
- Introducing Debugging
- Syntax Errors
- Semantic Errors
- Program State
- Keywords and Reserved Identifiers

Data

- A Sample Program
- What’s New in This Program?
- Data Variables and Constants
- Data: Data-Type Keywords
- Integer Versus Floating-Point Types
- The Integer
- The Floating-Point Number
- Basic C Data Types
- The int Type
- Other Integer Types

- Using Characters: Type char
- The _Bool Type
- Portable Types: stdint.h and inttypes.h
- Types float, double, and long double
- Complex and Imaginary Types
- Beyond the Basic Types
- Type Sizes
- Using Data Types
- Arguments and Pitfalls
- One More Example: Escape Sequences
- What Happens When the Program Runs
- Output

Character Strings and Formatted Input/Output

- Program
- Character Strings: An Introduction
- Type char Arrays and the Null Character
- Using Strings
- The strlen() Function
- Constants and the C Preprocessor
- The const Modifier
- Manifest Constants on the Job
- Exploring and Exploiting printf() and scanf()
- The printf() Function
- Using printf()
- Conversion Specification Modifiers for printf()
- What Does a Conversion Specification Convert?
- Using scanf()
- The * Modifier with printf() and scanf()
- Usage Tips for printf()

DAY 2

Operators, Expressions, and Statements

- Introducing Loops
- Fundamental Operators

- Assignment Operator: =
- Addition Operator: +
- Subtraction Operator: –
- Sign Operators: – and +
- Multiplication Operator: *
- Division Operator: /
- Operator Precedence
- Precedence and the Order of Evaluation
- Some Additional Operators
- The sizeof Operator and the size_t Type
- Modulus Operator: %
- Increment and Decrement Operators: ++ and –
- Decrementing: –
- Precedence
- Expressions and Statements
- Compound Statements (Blocks)
- Type Conversions
- The Cast Operator
- Function with Arguments
- A Sample Program

C Control Statements: Looping

- Revisiting the while Loop
- Program Comments
- C-Style Reading Loop
- The while Statement
- Terminating a while Loop
- When a Loop Terminates
- while: An Entry-Condition Loop
- Syntax Points
- Which Is Bigger: Using Relational Operators and Expression
- What Is Truth?
- What Else Is True?
- Troubles with Truth
- The New _Bool Type

- Precedence of Relational Operators
- Indefinite Loops and Counting Loops
- The for Loop
- Using for for Flexibility
- More Assignment Operators: +=, -=, *=, /=, %=
- The Comma Operator
- Zeno Meets the for Loop
- An Exit-Condition Loop: do while
- Which Loop?
- Nested Loops
- Program Discussion
- A Nested Variation
- Introducing Arrays
- Using a for Loop with an Array
- A Loop Example Using a Function Return Value
- Program Discussion
- Using Functions with Return Values

C Control Statements: Branching and Jumps

- The if Statement
- Adding else to the if Statement
- Another Example: Introducing getchar() and putchar()
- The ctype.h Family of Character Functions
- Multiple Choice else if
- Pairing else with if
- More Nested ifs
- Let’s Get Logical
- Alternate Spellings: The iso646.h Header File
- Precedence
- Order of Evaluation
- Ranges
- A Word-Count Program
- The Conditional Operator: ?:
- Loop Aids: continue and break
- The continue Statement
- The break Statement

- Multiple Choice: switch and break
- Using the switch Statement
- Reading Only the First Character of a Line
- Multiple Labels
- switch and if else
- The goto Statement
- Avoiding goto

Character Input/Output and Input Validation

- Single-Character I/O: getchar() and putchar()
- Buffers
- Terminating Keyboard Input
- Files, Streams, and Keyboard Input
- The End of File
- Redirection and Files
- Unix, Linux, and Windows Command Prompt Redirection
- Creating a Friendlier User Interface
- Working with Buffered Input
- Mixing Numeric and Character Input
- Input Validation
- Analyzing the Program
- The Input Stream and Numbers
- Menu Browsing
- Tasks
- Toward a Smoother Execution
- Mixing Character and Numeric Input

DAY 3

Functions

- Reviewing Functions
- Creating and Using a Simple Function
- Analyzing the Program
- Function Arguments
- Defining a Function with an Argument: Formal Parameters
- Prototyping a Function with Arguments

- Calling a Function with an Argument: Actual Arguments
- The Black-Box Viewpoint
- Returning a Value from a Function with return
- Function Types
- ANSI C Function Prototyping
- The Problem
- The ANSI C Solution
- No Arguments and Unspecified Arguments
- Hooray for Prototypes
- Recursion
- Recursion Revealed
- Recursion Fundamentals
- Tail Recursion
- Recursion and Reversal
- Recursion Pros and Cons
- Compiling Programs with Two or More Source Code Files
- Unix
- Linux
- DOS Command-Line Compilers
- Windows and Apple IDE Compilers
- Using Header Files
- Finding Addresses: The & Operator
- Altering Variables in the Calling Function
- Pointers: A First Look
- The Indirection Operator: *
- Declaring Pointers
- Using Pointers to Communicate Between Functions

Arrays and Pointers

- Arrays
- Initialization
- Designated Initializers (C99)
- Assigning Array Values
- Array Bounds
- Specifying an Array Size
- Multidimensional Arrays

- Initializing a Two-Dimensional Array
- More Dimensions
- Pointers and Arrays
- Functions, Arrays, and Pointers
- Using Pointer Parameters
- Comment: Pointers and Arrays
- Pointer Operations
- Protecting Array Contents
- Using const with Formal Parameters
- More About const
- Pointers and Multidimensional Arrays
- Pointers to Multidimensional Arrays
- Pointer Compatibility
- Functions and Multidimensional Arrays
- Variable-Length Arrays (VLAs)
- Compound Literals

Character Strings and String Functions

- Representing Strings and String I/O
- Defining Strings Within a Program
- Pointers and Strings
- String Input
- Creating Space
- The Unfortunate gets() Function
- The Alternatives to gets()
- The scanf() Function
- String Output
- The puts() Function
- The fputs() Function
- The printf() Function
- The Do-It-Yourself Option
- String Functions
- The strlen() Function
- The strcat() Function
- The strncat() Function
- The strcmp() Function

- The strcpy() and strncpy() Functions
- The sprintf() Function
- Other String Functions
- A String Example: Sorting Strings
- Sorting Pointers Instead of Strings
- The Selection Sort Algorithm
- The ctype.h Character Functions and Strings
- Command-Line Arguments
- Command-Line Arguments in Integrated Environments
- Command-Line Arguments with the Macintosh
- String-to-Number Conversions

DAY 4

Storage Classes, Linkage, and Memory Management

- Storage Classes
- Scope
- Linkage
- Storage Duration
- Automatic Variables
- Register Variables
- Static Variables with Block Scope
- Static Variables with External Linkage
- Static Variables with Internal Linkage
- Multiple Files
- Storage-Class Specifier Roundup
- Storage Classes and Functions
- Which Storage Class?
- A Random-Number Function and a Static Variable
- Roll ’Em
- Allocated Memory: malloc() and free()
- The Importance of free()
- The calloc() Function
- Dynamic Memory Allocation and Variable-Length Arrays
- Storage Classes and Dynamic Memory Allocation
- ANSI C Type Qualifiers

- The const Type Qualifier
- The volatile Type Qualifier
- The restrict Type Qualifier
- The _Atomic Type Qualifier (C11)
- New Places for Old Keywords

File Input/Output

- Communicating with Files
- What Is a File?
- The Text Mode and the Binary Mode
- Levels of I/O
- Standard Files
- Standard I/O
- Checking for Command-Line Arguments
- The fopen() Function
- The getc() and putc() Functions
- End-of-File
- The fclose() Function
- Pointers to the Standard Files
- A Simple-Minded File-Condensing Program
- File I/O: fprintf(), fscanf(), fgets(), and fputs()
- The fprintf() and fscanf() Functions
- The fgets() and fputs() Functions
- Adventures in Random Access: fseek() and ftell()
- How fseek() and ftell() Work
- Binary Versus Text Mode
- Portability
- The fgetpos() and fsetpos() Functions
- Behind the Scenes with Standard I/O
- Other Standard I/O Functions
- The int ungetc(int c, FILE *fp) Function
- The int fflush() Function
- The int setvbuf() Function
- Binary I/O: fread() and fwrite()
- The size_t fwrite() Function
- The size_t fread() Function

- The int feof(FILE *fp) and int ferror(FILE *fp) Functions
- An fread() and fwrite() Example
- Random Access with Binary I/O

Structures and Other Data Forms

- Sample Problem: Creating an Inventory of Books
- Setting Up the Structure Declaration
- Defining a Structure Variable
- Initializing a Structure
- Gaining Access to Structure Members
- Initializers for Structures
- Arrays of Structures
- Declaring an Array of Structures
- Identifying Members of an Array of Structures
- Program Discussion
- Nested Structures
- Pointers to Structures
- Declaring and Initializing a Structure Pointer
- Member Access by Pointer
- Telling Functions About Structures
- Passing Structure Members
- Using the Structure Address
- Passing a Structure as an Argument
- More on Structure Features
- Structures or Pointer to Structures?
- Character Arrays or Character Pointers in a Structure 627
- Structure, Pointers, and malloc()
- Compound Literals and Structures (C99)
- Flexible Array Members (C99)
- Anonymous Structures (C11)
- Functions Using an Array of Structures
- Saving the Structure Contents in a File
- A Structure-Saving Example
- Program Points

- Structures: What Next?
- Unions: A Quick Look
- Using Unions
- Anonymous Unions (C11)
- Enumerated Types
- enum Constants
- Default Values
- Assigned Values
- enum Usage
- Shared Namespaces
- typedef: A Quick Look
- Fancy Declarations
- Functions and Pointers

DAY 5

Bit Fiddling

- Binary Numbers, Bits, and Bytes
- Binary Integers
- Signed Integers
- Binary Floating Point
- Other Number Bases
- Octal
- Hexadecimal
- C’s Bitwise Operators
- Bitwise Logical Operators
- Usage: Masks
- Usage: Turning Bits On (Setting Bits)
- Usage: Turning Bits Off (Clearing Bits)
- Usage: Toggling Bits
- Usage: Checking the Value of a Bit
- Bitwise Shift Operators
- Programming Example
- Another Example
- Bit Fields
- Bit-Field Example

- Bit Fields and Bitwise Operators
- Alignment Features (C11)

The C Preprocessor and the C Library

- First Steps in Translating a Program
- Manifest Constants: #define
- Tokens
- Redefining Constants
- Using Arguments with #define
- Creating Strings from Macro Arguments: The # Operator
- Preprocessor Glue: The ## Operator
- Variadic Macros: … and __VA_ARGS__
- Macro or Function?
- File Inclusion: #include
- Header Files: An Example
- Uses for Header Files
- Other Directives
- The #undef Directive
- Being Defined—The C Preprocessor Perspective
- Conditional Compilation
- Predefined Macros
- #line and #error
- #pragma
- Generic Selection (C11)
- Inline Functions (C99)
- _Noreturn Functions (C11)
- The C Library
- Gaining Access to the C Library
- Using the Library Descriptions
- The Math Library
- A Little Trigonometry
- Type Variants
- The tgmath.h Library (C99)
- The General Utilities Library
- The exit() and atexit() Functions
- The qsort() Function

- The Assert Library
- Using assert
- _Static_assert (C11)
- memcpy() and memmove() from the string.h Library
- Variable Arguments: stdarg.h

Advanced Data Representation

- Exploring Data Representation
- Beyond the Array to the Linked List
- Using a Linked List
- Afterthoughts
- Abstract Data Types (ADTs)
- Getting Abstract
- Building an Interface
- Using the Interface
- Implementing the Interface
- Getting Queued with an ADT
- Defining the Queue Abstract Data Type
- Defining an Interface
- Implementing the Interface Data Representation
- Testing the Queue
- Simulating with a Queue
- The Linked List Versus the Array
- Binary Search Trees
- A Binary Tree ADT
- The Binary Search Tree Interface
- The Binary Tree Implementation
- Trying the Tree
- Tree Thoughts

Duration and pricing

In Price Group A

Certificate

Read about our certificates here

Bookings

You can download the course registration form on our home page or by clicking here

Brochure

You may download a pdf copy of this page by clicking on the pdf icon at the top of the page.

Questions

Please email us

Schedule

On the calendar below. If your browser doesn’t display the calendar below, please click on this link or try using Google Chrome, alternatively please enquire via our Contact Us page.