ProTech protechtraining.com *"Charting the Course ... ... to Your Success!"*

Introduction to C Programming

Course Summary

Description

The original C programming language was designed by Denis Richie at Bell Labs in 1972 as part of the development of the UNIX operation system. Since then, it and has become the workhorse of operating systems, embedded systems applications, real time applications and is ubiquitous in most IT infrastructures. C has a massive installed code base on almost every operation system and hardware platform from super computers to micro-devices.

The course has twin objectives. The first is to teach students the C programming language from both a code development and code maintenance perspective since many students will be faced the challenges of updating and modifying legacy C code. Students are taught how to design and develop a C application but also how to read and understand existing C code.

The second objective is to teach students the "C programming style." C programmers have developed very characteristic ways writing code which leverages the strengths of the language while avoiding the pitfalls that naive C programmers fall into. For example, C code is written "close to metal" which means that C programs can exert fine grained control over hardware resources, a feature which allows for very tight and fast executables, but also allows programmers to create havoc when the code runs. As Richie said, "If we prevent programmers from doing dangerous things, we also prevent them from doing brilliant things." The C programming style is a way of writing C code that has evolved to write safe, fast, and tight code.

Since its original release, C has undergone several revisions, the most notable being the ISO standard C99 in 1999 and the current standard C11 in 2011. The material in the course highlights the differences in the language in the various standards since students are likely to be working with not just the current standard but legacy code that may have been written to a different standard.

Over half of the class time will be spent doing hands on exercises or labs. The gcc compiler is used for the class in either a Linux or Windows environment with Eclipse as a visual IDE.

Objectives

After taking this course, students will be able to:

- Identify and describe the significant differences in the C programming standards (C90, C99, C11)
- Identify the main program components and structure of a C application
- Write, compile and link C applications in using standard gcc tools and the make utility
- Work with different built in data types, variables and pointers
- Work with the C operators and control structures including logical and bitwise operators
- Work with allocation heap memory, structures and arrays
- Write and use functions effectively and describe how the stack works for variables and functions
- Work with raw and formatted I/O including file I/O
- Work with strings and characters, including Unicode
- Work with the standard C libraries
- Work with macros and pre-processor directives
- Design a C application describing the best practices used in function and module design
- Describe C coding "style" and program best practices for each programming topic
- Debug a C application
- Analyze an existing C application
- Describe common C programming mistakes and how to avoid them

Introduction to C Programming

Course Summary (cont'd)

Topics

- Background to C
- Working in the C environment
- Basic C Program Structure
- Data Types
- Operators
- Using the Heap
- Structs, Unions, and Arrays
- Functions and Macros
- I/O
- C Best Practices

Audience

This course is for programmers who want to learn C programming.

Prerequisites

This is not an introduction to programming course so it is assumed that students have some basic programming experience and understand the fundamental concepts of functions, variable, data types that are common across programming languages. Students are also assumed to be comfortable working at a command line interface.

Duration

Three days

ProTech protechtraining.com *"Charting the Course ...*

... to Your Success!"

Introduction to C Programming

Course Outline

I. Background to C
- A. Brief history of C
- B. C standards: K&R, ANSI C (C89), C90, C99, C11
- C. Current status of C
- D. C and C-style languages: C++, Java, PERL, etc.
- E. C compatibility with emerging languages, eg. Go and Rust

II. Working in the C environment
- A. Compiling and linking using gcc
- B. Setting up a C development project
- C. Using the C libraries and header files
- D. Introduction to the pre-processor
- E. Using utilities like lint and make

III. Basic C Program Structure
- A. The main() function
- B. Variables: local and static
- C. The extern keyword
- D. Variables: storage and scope
- E. Functions: declarations and definitions
- F. C program formatting
- G. Comments
- H. Statements and expressions

IV. Data Types
- A. Numeric Data Types: integer, floating point and complex (C11)
- B. Casting data and type conversions
- C. Character and string data
- D. Boolean data
- E. Address operators and pointers
- F. Using integers as bit masks
- G. Constants and pre-processor defined values

V. Operators
- A. Conditional operators for flow control
- B. Looping constructs with break and continue
- C. Arithmetic and Boolean operators
- D. Bitwise operators
- E. Increment and decrement and related operators
- F. The switch statement

VI. Using the Heap
- A. Using malloc(), free() and related memory allocation operations.
- B. Understanding memory leaks.
- C. Pointers to allocated memory and pointers to pointers
- D. Casting pointers

VII. Structs, Unions and Arrays
- A. Defining new types with structs
- B. Allocating structs
- C. Unions as overlays
- D. Creating and using unions
- E. Defining, creating and using arrays
- F. Arrays as pointers
- G. Arrays of arrays and arrays of structures
- H. Strings as arrays of characters

VIII. Functions and Macros
- A. How functions work, return values and parameters.
- B. Pass by pointer versus pass by value
- C. Function pointers
- D. Passing structs and arrays as parameters and return values

IX. I/O
- A. Stdin, stdout, sterr
- B. Formatting input and output with the printf() family of functions
- C. Opening and closing files
- D. Serial file input and output
- E. Random access file input and output

X. C Best Practices
- A. “First solve the problem, then write the code.”
- B. Modular function design: high cohesion and low coupling
- C. Write unit tests before coding
- D. Header files are interfaces – good API design
- E. Readable C code
- F. Assertions are your friend
- G. Common C errors