Advanced 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.

This course continues from where the Introduction to C Programming course ended. Students explore a number of advanced topics, delve deeper into good C program structure and design and work with the standard library.

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:

- Work with function pointers to solve some standard C problems (eg. dispatch tables and callbacks)
- Work with varadic functions
- Work with multi-dimensional arrays
- Work with Unicode Strings and internationalization
- Work with complex structures and embedded structures
- Write conditional code and macros for the pre-processor
- Debug and stepping through code
- Code review C code and identify dangerous or inefficient code constructs
- Create a reusable C library
- Use the stdef, stdlib, time, and other standard C library modules
- Using assertions for code correctness
- Benchmarking and profiling C code
- Multi-threaded C applications
- Analyzing legacy C code
- C code design best practices
- C code smells and refactoring C code

Topics

- Function pointers and varadic functions
- Multi-dimensional arrays and complex structures
- Strings and Unicode
- C99 and C11 Data Types
- Times, dates, and localization
- Multi-threaded C
- Macros
- Debugging, benchmarking, and profiling C code
- C program and library design

Audience

This course is for programmers who already have completed the Introduction to C programming or its equivalent.

Prerequisites

Students are assumed to understand the concepts from the Introduction to C programming course and to be able to apply them to writing C code.

Duration

Two days

Advanced C Programming

Course Outline

I. **Function pointers and varadic functions**
   A. Declaring and using function pointers
   B. Arrays of function pointers
   C. Callbacks and passing function pointers as arguments to functions
   D. Using dispatch tables and pointers to function pointers
   E. Writing and using varadic functions

II. **Multi-dimensional arrays and complex structures**
   A. Defining and using multi-dimensional arrays
   B. Ragged arrays
   C. C99 variable length arrays
   D. Combing structs and multi-dimensional arrays
   E. Anonymous unions and structures
   F. Best practices, limitations and problems with multi-dimensional arrays

III. **Strings and Unicode**
   A. Unicode, ANSII, UTF-8 and UTF-16
   B. C char and wide char data types
   C. The <ctype.h> and <string.h> standard libraries
   D. The <wchar.h> and <wctype.h> libraries for wide strings
   E. The <ucode.h> unicode standard library
   F. Deep dive into the <stdio.h> standard library

IV. **C99 and C11 Data Types**
   A. Exact width, minimum width, fastest width and maximum width integer types
   B. The <inttypea.h> and <stdint.h> standard libraries
   C. The complex data type and the <complex.h> standard library
   D. Boolean types using the <stdbool.h> standard library
   E. Floating point data using the <float.h> and <fenv.h> standard libraries
   F. Using the <math.h> standard library
   G. Constants and pre-processor defined values

V. **Times, dates and localization**
   A. Using the <time.h> standard library
   B. Working with time and date structures
   C. Operations on dates and times
   D. Localization functions with the <local.h> standard library

VI. **Multi-threaded C**
   A. Writing a multi-treaded application
   B. Threads, mutexes, condition variable and shared resources
   C. Using the <threads.h> and <stdatomic.h> standard libraries
   D. Best practices and potential pitfalls with multi-threaded applications.

VII. **Macros**
   A. Object and function like macros
   B. Macro arguments and varadic macros
   C. Macro directives, defining and undefining macros
   D. C99 type generic macros
   E. Using the <tgmath.h> standard library
   F. Common problems with macros

VIII. **Debugging, benchmarking and profiling C code**
   A. Using a debugger and stepping through C code
   B. Profiling and performance tools – instrumentation framework
   C. Using gcc gprof
   D. Other tool suites: gperftools, valgrind, etc
   E. Benchmarking C code – survey of tools

IX. **C program and library design**
   A. Structured code design principles
   B. Coupling and cohesion between program components
   C. Clean, robust and readable functions
   D. Analyzing legacy C code
   E. C code smells
   F. Refactoring and redesigning C code