Faysal Aziz

C Programming Fundamentals

Syntax, memory, files and a small developer workflow

FA-0594 · 2 days (or 4 half-days)

Software Development

Write and debug small C programs using core language constructs, pointers, structures, file I/O, Git and Make.

Introduction

This practical introductory course develops C programming fundamentals through selected small examples. It moves from compiler setup and syntax to pointers, dynamic memory, structures and file operations, supported by a basic Git and Make workflow.

Participants work towards one small command-line utility or a prepared records example. Linked structures, function pointers, concurrency-independent systems concerns and larger codebases are overview topics rather than complete systems-development training. Prior programming logic is helpful, especially in this intensive format; beginners use guided scaffolding.

Learning outcomes

The course teaches participants to:

- Use basic C types, operators, control flow and functions.
- Compile, run and debug a small program with a suitable toolchain.
- Track source changes with Git and use a basic Makefile.
- Explain pointer, allocation and lifetime concepts and identify selected memory mistakes.
- Use structures and standard-library file/string operations with appropriate bounds and return-value checks.
- Combine selected skills in a small command-line program and review its limitations.

Prerequisites

- Basic programming logic is helpful but prior C experience is not required.
- A Windows, Linux or macOS development machine with a compatible C compiler, editor/debugger, Git and Make or a prepared equivalent build environment.
- For learners with no programming experience, guided examples and basic pre-class environment familiarisation are recommended.

Training outline

1. Day 1 — Language and Developer Workflow

Module 1 — Environment and Project Structure

- Where C is used in systems, embedded software and utilities, and what this introductory course covers.
- Compiler/editor setup, Git and a prepared Make-based build workflow.
- Directory structure and project layout for small C programs

Module 2 — Core C Constructs

- Variables, data types (int, char, float, double, void, etc.), constants, enums
- Operators (arithmetic, relational, logical, bitwise)
- Control flow: if/else, switch, loops (for, while, do-while), break/continue
- Functions: declaration vs definition, parameter passing (by value), return types, scope and lifetime
- A small input/output program with format, buffer and return-value checks.

Module 3 — Git and Make

- Git basics: repository initialization, commit, branch, merge, basic workflow (clone, add, commit, push/pull)
- Makefile fundamentals: targets, dependencies, rules, variables, phony targets, cleaning builds
- Track source and headers in Git while keeping generated build artefacts out of source control.
- Create a small program and Makefile; use a selected local Git workflow rather than require publishing to an external repository.

2. Day 2 — Memory, Data and a Small Utility

Module 4 — Pointers and Memory

- Storage duration and common stack/heap implementation concepts; distinguish language lifetime rules from platform-specific memory layout.
- Pointer/address/dereference/null concepts and pointer arithmetic within valid object bounds.
- malloc, calloc, realloc and free: allocation checks, ownership, lifetime, leaks and dangling pointers.
- Arrays/pointers in a selected example; pointer-to-pointer and function-pointer concepts through demonstrations.
- A small allocation and iteration exercise; inspect a prepared dynamic-array or simple structure example rather than build several complete data structures.

Module 5 — Structures, Files and Library Functions

- Defining struct, union, enum; using typedef for clarity
- Nested structures, self-referencing structures (e.g., linked list node)
- Memory layout of structures, alignment/padding concerns, best practices
- Use fictional records to practise structure fields and allocation/freeing.

File Input/Output and the C Standard Library

- Standard I/O: fopen, fclose, fread, fwrite, fprintf, fscanf, perror and error handling
- Text versus binary files and positioning; do not assume raw structure bytes are portable persistent records.
- Command-line arguments (argc, argv) and basic parsing
- Using C Standard Library: <stdlib.h>, <string.h>, <stdio.h>, <ctype.h>, <errno.h>
- String/memory functions including strcpy, strncpy, strcmp, strlen, strchr, memcpy and memset: check destination capacity and termination; strncpy does not guarantee NUL termination.
- Selected text-file processing utility with error handling and a Git/Make workflow.

Module 6 — Mini-Project and Code Review

- Project workflow: plan > code > build via Make > debug/test > commit to Git
- Complete or extend one scoped utility or prepared fictional-records example using selected pointers, structures, files and functions.
- Code review and discussion: how to write maintainable C code (naming conventions, modularization, header files, separating interface vs implementation)

- Use an IDE/GDB debugger and compatible memory-checking tools, such as Valgrind where supported.
- Safe coding guidelines in C (checking return values, validating pointers, avoiding undefined behaviour)
- Standards awareness: C23 was adopted as ISO/IEC 9899 in 2024; distinguish published standards, draft proposals and compiler support without treating a future revision as guaranteed.
- Readable modules, headers, interface/implementation separation, documentation and version-control practices.
- Next steps: how to evolve skillset into system-level development, embedded systems, performance tuning
- Review selected pointer, segmentation-fault and build issues.
- Identify further practice in systems or embedded programming; the course does not establish production codebase readiness.

© 2026 Faysal Aziz · Kuala Lumpur