

# Making the Switch to C and C++

## an accelerated course for experienced programmers

C, with its object-oriented superset C++, is becoming established as the most widely used programming language for developing both system software and applications for most operating platforms. The successful programmer of the 1990's needs a command of C++ in his or her professional repertoire.

C began as a *small* language. With the addition, however, of object-oriented facilities and standard libraries of container and interface modules, C++ has grown to become perhaps the *largest* of all languages, one that presents difficulties for even the most experienced programmer. This course emphasizes techniques and principles of good practice that minimize those difficulties and quickly get participants started developing *good* C++ programs.

**PREREQUISITES:** (1). Experience in using at least one procedural programming language<sup>1</sup>.  
(2). Familiarity with *structured coding* concepts and *modular program* organization.

**LENGTH:** 7 half-day sessions

**CLASS SIZE:** 6 - 18

### GENERAL DESCRIPTION

We start with a quick survey of enough elements of C to let participants develop and run straightforward programs that require little or no manipulation of pointers. Noting C's richness of operators but poverty of data types, we contrast its original limited goals with its expanded role today.

We then discuss in depth *data type* and *data representation*, laying a foundation for *class* hierarchies, the *object-oriented paradigm*, and relevant C++ facilities, including single- and multiple *inheritance*, *encapsulation*, and *overloading*. Within the limitations of C++, our examples emphasize using OOP to model *application* domains rather than just the container data structures and user-interface objects emphasized by other courses and textbooks.

After explaining the central role of *pointers* in C and developing confidence with everyday situations requiring pointer manipulation, we examine handling of arrays, strings, data structures, and function values. We show how to exploit C++ class hierarchies and dynamic *polymorphism* to simplify program structures and avoid or localize error-prone or implementation-dependent techniques.

For audiences interested in business applications we show classes for dealing with money, strings, dates, and other essential elementary data. For audiences interested in numerical computation we show classes for

dealing with complex numbers, electrical quantities, numeric vectors, and matrices. In either case, we point out that after going to some initial trouble, we've achieved major advantages over purely procedural languages in program maintainability and reliability.

We proceed to consider composite data classes with inheritance, drawing a clear distinction between gen-spec (*is-a*) hierarchy and whole-part (*has-a*) hierarchy.

We explore function templates and polymorphism as two ways of supporting generic functions. We then examine class templates and representative container classes, introducing the key features of the Standard Template Library.

We don't try to persuade participants that C++ is a perfect language. Throughout the course we confront its shortcomings and critically examine how a skilled programmer can overcome or circumvent them.

We conclude by drawing on the repertoire of language facilities and principles of *software engineering* to develop a small but non-trivial program. As an important by-product, we stress exploiting opportunities to draw from and contribute to a library of re-usable modules and classes.

Participants use a computer for exercises outside class time. Completed assignments are examined by the instructor and returned to the student with comments.

---

<sup>1</sup> When most students in a class share a common language background, we tailor early examples and presentations to that language so as to give them a familiar point of reference and highlight relative strengths and weaknesses. Such custom course versions are available for those with Basic, Pascal, PL/I, and (with 1 extra half-day session) COBOL backgrounds.