OBJECT-ORIENTED PROGRAMMING: C++ OVERVIEW

TUTORIAL PRESENTATION

John Minor Ross
Indiana University Kokomo
PO Box 9003, Kokomo, IN 46904-9003
317-455-9213 (area code 765 as of Feb. 1997)
jmross@indiana.edu

This tutorial presumes the attendees will have at least a basic understanding of the C language. Based on that presumption, only C features that are impacted by the transition to object-oriented programming in C++ are reviewed. Anyone not familiar with OOP terms and trends may wish to attend the earlier tutorial covering those topics.

C++ has many changes from C. C++ may be used as a 'better' C without implementing any OO approaches. Because some of these enhancements pervade the typical C++ program, they will be introduced first. Following this, sample code will help attendees compare and contrast traditional structured programming in C with OOP in C++. The basics of encapsulation, inheritance and polymorphism will be shown.

- Non-OOP changes from C. Before learning to use C++ for OOP, several enhancements to C are worth discussing. For example, the printf() and scanf() functions are seldom used in favor of the cout and cin objects. References (synonyms) often replace the complexity of using pointers. Function name overloading (multiple functions with identical names) and operator overloading (teaching old operators new tricks) will also be covered.
- Encapsulation using C++. User-defined, abstract data types in C++ are established using the class keyword. Classes group data and methods that may access the data together. Instead of calling a function in C, OO programmers may say they are passing a message to a method.
- Inheritance using C++. New classes may inherit the attributes and behaviors of an existing class (note: classes inherit; objects do not). Inheritance allows the sharing of tested code rather than reinventing the wheel.
- Polymorphism using C++. An employee base class may have a 'pay' method set up as 'virtual.' This allows classes that inherit from the employee class, say hourly worker and commissioned worker, to have different routines also named pay that calculate pay differently. When the program is run, however, using a base class employee pointer to invoke the pay method of a worker results in the selection of the correct pay routine for the type of employee referenced by the pointer.

A final segment of the presentation will provide heuristics about transitioning into Windows OOP using C++. This likely next step presents the programmer with another steep, yet scaleable, learning curve.