Advanced C++

Yahoo! Inc.
Sunnyvale, CA
January 2012

Instructor: Leor Zolman ("Lee-Ore")
leor@bdsoft.com
Course by Stephen C. Dewhurst

Please grab one book, one USB stick

Introductions

- Me
  - Born/raised in L.A., now reside in North Reading, MA
  - 1st Computer: an 8080-based IMSAI (kit) in '75
  - First used C/Unix in '78, wrote "BDS C" for CP/M in '79
  - On staff at *The C/C++ Users' Journal* '88-'92
  - Began full-time training in '92
  - Wrote *STLFilt* in '01 to help "decrypt" C++ diagnostics
- You
  - (By show of hands…) How much C++ programming experience do you have?
  - What do you most want to learn about C++ ?

Overview

"Standard" Modules

(Printed)

0. Intro and "Review Quiz" on C++ and OOP
1. "Recent" Additions to the C++ Language and Libraries
2. Template Mechanics
3. The Standard Template Library
   a. Introduction to the STL
   b. STL Algorithms
   c. STL Containers
   d. STL Function Objects
4. Exception Handling / Coding for Exception Safety
5. Inheritance and Object-Oriented Design (Meyers)

Overview

Optional and Supplementary Modules

(PDF Only)

6. Advanced Memory Management
7. Copying, Conversions and Temporaries
8. Pointers, References and Addresses

Questions and Answers

- Please think of this training environment as "informal"
- Questions/discussions are welcome at any time
- Feel free to *interrupt* me if you need any clarification
- There are ***no*** "dumb" questions!
  - C++ is a complex, nuanced language
  - If you haven't had a lot of formal training, there are a lot of things you've probably never heard or seen
  - If you see or hear one of them this week without an accompanying explanation, please ask me for one!

Labs

- Hands-on time is valuable; most units have lab exercises at the end
- There are more labs in each lab session than most people can realistically finish
  - That's to help keep the really advanced folks from getting too bored
  - I'll suggest which ones offer the best "bang for the buck"
- Solutions are provided electronically in the `\AdvCPP` file tree (which you'll be copying to your HD)

Hours, Breaks, etc.

- Official class hours are 9am – 5pm (?)
- We'll try to have a short (around ten minute) break *approximately* every hour
- Lab sessions include some break time
- During a break, lunch or lab session, or at the end of the day, the time we're to resume will be posted somewhere prominent. Please be back in the training room at that time
- Lab times may be extended

Development Tools and Class Files: Load from USB and Test

- Automatically installed by the `setup-core.bat` file:
  - Course files (`AdvCpp` hierarchy, `\PDF`)
  - Boost libraries
  - Perl interpreter (for STLFilt capability)
  - Integration via batch files (Windows) or shell scripts (Unix)
- If necessary or desired: gcc C++ command-line compiler via `setup-gcc.bat`

Taking Stock: A Review "Quiz"

- As a fast-paced review of some basic C++ and Object-Oriented concepts…
- I'll put up slides containing 18 questions, each pertaining to either a syntactic or idiomatic aspect of the language
- For each, jot down your best short answer. Note: Some questions are "open ended"
- After seeing all the questions, we'll go back over them all in detail
- Let's get started!

Questions #1-4

1) How many distinct contexts are there in C++ for the use of the `static` keyword?

2) What does the term *static storage* mean? (In this case, the word *static* is *not* being used as a language keyword)

3) What are the C++ "storage classes" ?

4) What's the difference between *internal linkage* and *external linkage?*

Questions #5-7

5) In the class hierarchy below, is `~D` (class `D`'s destructor) virtual?
6) Does `D::f(double)` override `B::f(double)` ?
   If not, what *does* it do?
7) For the calls marked #1-#4: Legal? If so, what gets called?

```
class B
   {
public:
    B();
    virtual ~B();
    int f(double x);
    virtual void g(int i);
    virtual void g();
 };

int main() {
    B *bp = new D;
    bp->f(2.23);        // Call #1
    bp->g(10);          // Call #2
    bp->f();            // Call #3

    D *dp = new D;
    dp->g();            // Call #4
};
```

```
class D : public B
{
 public:
   D();
   int f(double x);
   int f();
   void g(int i);
};
```

Questions #8-10

8) Is the code below C++ Standard-conformant?

9) If so, will the output be as the comments indicate?

10) Does `vp` conceptually represent a *pointer* or an *array*?

```
#include <iostream>
int main()
{
      double vals[] = { 1.1, 2.2, 3.3, 4.4};
      std::cout << vals[1] << std::endl; // 2.2
      std::cout << 1[vals] << std::endl; // 2.2
      double *vp = vals + 1;
      std::cout << vp[0] << std::endl;   // 2.2
      std::cout << 0[vp] << std::endl;   // 2.2
}           // (Aside: anything missing here?)
```

Questions #11-13

11) What does it mean for a program to be "`const` correct" ?

12) What's the difference (if any) between the following two function declarations:

```
int f(const int x);
int f(int x);
```

13) [Guru Question!]: What *implicit* declaration is affected by taking a non-`const` member function and changing its declaration and definition so that it becomes a `const` member function?

Questions #14-16

14) From an OO perspective, what "policy" (with respect to the function in question) is being dictated by a base class to its subclasses in each of the following cases?

- A non-virtual (non-static) member function
- An impure virtual function
- A pure virtual function

15) What is meant by the concept of an *Abstract Base Class*, and how do you define one in C++?

16) What does an ABC *demand* of its derived classes if they wish to be *concrete*?

Question #17-18

17) Is the following C++ code Standard-conformant? Will it compile without error? Why or why not?

18) [Guru question!]: If it compiles, what are three different possible runtime results?

```
#include <iostream>
int main()
{
        char *str1 = "ABCDEFG";
        char *str2 = "ABCDEFG";


        str1[3] = '*';

        std::cout << str1 << std::endl;
        std::cout << str2 << std::endl;
}
```

What You Are Assumed to More-or-Less Already Know…

- The full syntax and semantics of C
  - C is an *almost*-proper subset of C++
- Abstraction – Basic mechanisms
  - Classes: defining, construction/destruction, copying, assigning, overloading, etc.
  - "`const` correctness" for pointers and references
  - `static`: what are *all* the meanings of this overloaded word?
- Inheritance and polymorphism
  - Virtual and pure virtual functions, abstract base classes
  - *Overriding* vs. *hiding* of names (variables and functions)
  - What makes a class *polymorphic*? What are the performance implications (speed and space) that implies?

If You've *Seen* These Topics Before, You'll Have a Leg Up...

(...but if not, most are, or *can* be, covered in detail during the course)

- Exception handling
- Basic class and function templates
- STL containers and algorithms
  - `std::vector`
  - The *erase-remove* idiom
  - `auto_ptr`
- Function objects (a.k.a. "functors")

"Notes" Section on Slides

- Many course slides have clarifications, examples, and obscure tidbits down in the notes area of the printed student guides
- These do not show up on screen during the lectures
- If you have any questions about any material that appears down in the notes sections, please feel free to ask about them

About the Course Author

Steve Dewhurst is the co-founder and president of Semantics Consulting, Inc. Steve is the author of numerous technical articles on C++ programming techniques and compiler design, and is the co-author of *Programming in C++* (Prentice Hall, 1989, second edition 1995) and the author of *C++ Gotchas* (Addison Wesley, 2003) and *C++ Common Knowledge* (Addison Wesley, 2005).

As a Member of Technical Staff in the UNIX Development Laboratory at AT&T Bell Laboratories, Steve worked with Bjarne Stroustrup, the designer and first implementer of C++, on the first public release of the language and cfront C++ compiler, then served as the lead designer and implementer of the first non-cfront C++ compiler. As a compiler architect at Glockenspiel, Ltd., he designed and implemented a second C++ compiler. He has been programming in C++ as long as it's been possible to program in C++.

Steve has served on the ANSI/ISO C++ standardization committee, was the C++ training series adviser for Technology Exchange Company (Addison Wesley), was a member of the editorial board of and columnist for *C++ Report*, and was co-founder and member of the editorial board of *The C++ Journal*. He has taught extensively in both university and commercial settings. Before that, he wrote C, COBOL, and Pascal compilers, was a principal on the ANSI/IEEE Pascal Standardization Committee, and a reviewer for *ACM Computing Reviews*.