Object-Oriented Principles and Practice / C++

Alice E. Fischer

August 29, 2016

Course Information

All information about this course is posted on the course website:

http://eliza.newhaven.edu/cpp/

- Syllabus
- The main textbook, *Exploring C++* (my compiled notes)
- The lecture schedule and weekly lecture notes
- Code samples
- Programming assignments

Course Requirements: A term project built by combining about ten weekly programming projects, with documentation (∼50%), midterm (∼20%), and final (∼30%).

Course mechanics

- I expect you to learn and follow the design and coding standards covered in the lecture and the book.
- Turn in your zipped-up assignments by email to my home address.
- It is your job to prove that the code works. Therefore: include adequate test output with every submission.
- If you need help with an assignment, zip it and send it to me with HELP in the subject line. I cannot give you any realistic help unless I can see the entire code base, as it was, when the problem occurred.
- Late assignments will be accepted. However, lateness is self-punishing. If you cannot do the work on time, you will not understand the lectures and you will do badly on the exams.

Your Programming Environment

Please download and use either XCode (Macintosh) or Eclipse (Windows and Linux). Eclipse users need to get the CLang compiler or the Gnu C++ compiler (g++) not the stripped-down mingw compiler. Widows people will need to use cygwin.

Everything you turn in must be written entirely in standard C++ 11 standard, and I must be able to compile it on my machine without modification. DO NOT include any Windows pathnames or any Microsoft-only header files! (conio.h, windows.h).

The Visual Studio IDE puts inappropriate things into your code and does not give good error comments. Students who use it inevitably have more trouble than others.

CodeBlocks does not supply the minimum level of support and diagnostics that you need. Don't waste your time on it.

Reference

For online reference, use one of these sites.
www.cplusplus.com
http://en.cppreference.com/w/

If you want a printed textbook, I recommend that you buy the book Herbert Schildt, *C++: The Complete Reference*. It serves as a reference manual for C++ syntax. as well as a basic text.

Do not use online sites such as Stack Overflow or Geeks for Geeks. They contain notoriously bad advice and bad code that does not meet the standards of this course.

Kinds of Programming

Three views of programming

People program for different reasons.

Programming can be ...

1. One-time software: code that solvea a computational problem.
2. Personal software that you can reuse.
3. Building a product to be used by others, over a period of several years.

Problem solving

Desired properties of single-use programs for solving problems:

- Correct outputs from correct inputs
- Succinct expression of algorithm
- Simple development cycle

Beginning programming courses focus on solving small problems.

Script programming is quickly written and, often, quickly forgotten.

Software Product Construction

Desired properties of software constructed for widespread use:

- Correct outputs from correct inputs
- Robust in face of bad inputs; reliable
- Economical in resource usage (time and space)
- Understandability and verifiability of code
- Security
- Ease of repurposing
- Ease of deployment
- Maintainability

Programming in the large

This course will focus on constructing **large-scale** software.

- Interface design and quality is important
- Thousands of lines of code
- Potentially written by many programmers
- And maintained over the years by different programmers
- Over a large span of time
- Deployed on a large number of computers
- With different architectures and operating systems
- Interacting with foreign code and devices

Course Goal: Recognize Quality C++ Programming

Code should . . .

1. Meet the specifications.
2. Be correct. I should have reason to trust the correctness.
3. Be debuggable.
4. Be portable.

Quality C++ programs are written according to strict OO design standards.

Course goal: Professional Programming

- Write quality code.
- Learn about classes, objects, type hierarchies, templates, exceptions, the template library, and their implementations.
- Learn how C++ objects are implemented and managed, and how to use move semantics.
- Learn what object-oriented programming is – and isn't.
- Learn the basic principles and patterns of OO design.
- Learn how C++differs in syntax and semantics from standard ISO C on the one hand and from other languages with support for OO-programming such as Python, Ruby, and Java.

Course goals, continued.

- Follow instructions, and question them if you think they are wrong.
- Learn how to get a big job done one module at a time.
- Use a reference manual.
- Learn clean OO design paradigms and practices.
- Write code that is concise, readable, and non-repetitive.
- Test, analyze, debug, and present your work in a professional manner.
- Become proficient at modern C++ programming.

Tools to Achieve the Goals

All of these tools are part of the course:

1. Privacy: isolation of each class from the others.
2. Type clarity: use the correct type for everything. No more using 1 for a truth value.
3. A good compiler that identifies type errors and gives clear error comments.
4. A knowledge of how to debug effectively without a debugger.
5. Carefully designed test suites and well documented test results.
6. The C++ 11 software toolset, including exceptions, derivation, algorithms, templates, move semantics and initialization options.
7. Design patterns.

Course Programming Standards

Five Commandments for this Course

Read and think about Chapter 1 of Exploring C++

1. Use C++ input and output, not C I/O for all assigned work.
2. Don't use global variables. If you think you need one, your class-design is probably defective. Ask for help.
3. Test every line of code you write. It is your job to prove to me that your entire program works. If I get a program without a test plan and output, I will assume that it does not compile. A program with a partial or inadequate test plan will be assumed to be buggy.
4. Clean up after yourself. Close files. Delete dynamic memory.
5. Put your name and the file name in comments at the top of every file.

Basic Design Principles and Standards

1. Conform to the specifications.
2. Keep it simple.
3. Make your code readale.
4. Identify the expert for each task.
5. Maintain data privacy. Use delegation.
6. Maximize data isolation: use const wherever meaningful.
7. Know what object has custody of dynamic memory areas at all times.
8. Avoid hardware- or OS-dependent code.

Some of these topics are elaborated in the following slides.

Conform to the Specification

A product is worthless if it does not do what the client ordered.

1. In this course, PLEASE read the instructions. Read all of the instructions before you begin to write code. Take the time to understand what you are supposed to do.
2. Code design takes time but it makes code production much faster and easier. Take the time to understand the class structure you are trying to implement.
3. In this course, the assignments are designed to give you practice using a wide range of tools. Use the tools and techniques that are required. You might find something different on the internet, but that is irrelevant. (You are wiser not to search for solutions "out there".)
4. If you think my specification is wrong, ask. Don't just ignore it.

Keep it Simple

CAN is not the same as SHOULD.

1. A function is too long if it does not fit on your screen all at once.
2. A function is not simple if its logic is nested to level 4 or more.
3. A function is not simple if I cannot comprehend its purpose and action in 30 seconds.
4. Very long and very short identifiers are not simple.
5. A name is not simple if you cannot pronounce it.
6. A class name that is the same as a variable name is not simple. Avoid it.

Make your code readable

1. Minimize the use of underscores and variable names i, l, and O. Remember – my eyes cannot see fine detail. Use letters I can see (j, k, n).
2. Put a space after every comma or semicolon. Donotwritecodethatlookslikethis.
3. Use blank lines to divide code into paragraphs. DO NOT USE THEM EVERYWHERE.
4. Indent your code properly. Xcode and Eclipse will do this for you.
5. Restrict code and comments to 80 columns. Avoid names that are so long that you cannot do this.
6. Put a divider-comment before the first line of every function.
7. PLEASE put the opening { on the same line as the `if` or `for` or function name.

Identify and Rely on the Expert

1. The expert on type A objects is class A.
2. All functions that work on type A objects belong in this class. item All changes to type A objects should be done by functions in class A.
3. A class definition defines ONE object of that class. Functions that operate on ONE object, or binary operators that operate on two belong in the class.
4. Function that operate on many of these objects belong in a different class.

Maintain Data Privacy

1. All member variables and class variables should be private or protected.
2. Read-only access can be supplied by "getters".
3. "Setters" breach privacy. Do not use them in this class.
4. The class constructor(s) should store data in a new class object. Don't call the constructor until you know what data should be used.
5.

Can is not the same as should!

Chapter 1 of Exploring C++

- C++ is a very powerful language, which, if used badly can produce projects that are badly designed, badly constructed, and impossible to debug or maintain.
- Your goal is to learn to use the language well, and with good style.
- Please read *and follow* the style guidelines in Section 1.2
- Download the two tools files from the Code Examples page of the website.
- Then read Section 1.3, about the tools library, and use this information to customize your own copy of the tools.

Using C++

The topics below relate to the DataPack example program attached to the Code Examples page of the website under Week 1. The files are `main.cpp`, `pack.hpp`, `pack.cpp`.

Program Structure

Chapter 4.1 of Exploring C++

- A simple 1-class C++ program has three files:
  1. One file contains the main function, some unit-test functions, and very little else.
  2. One file is the header for the class. It contains the class declaration, list of data members, definitions of inline functions, and prototypes of the functions that are too long to be inline.
  3. The third file contains the definitions of the functions that were prototyped in the header file.
- In today's example, these files are `main.cpp`, `pack.hpp`, and `pack.cpp`.

The Main Program

- The `main()` function is not part of a class. (Unlike Java.)
- Its prototype is one of these:

```
int main( void )
int main( int argc, char* argv[] )
```

- Include the header file for your major class in this file.
- The main function should call `banner()` and `bye()`
- Between these calls, it must create an object of the primary application class (say, Cls).
- In a completed application, `main()` will then call `Cls.run()`.
- However, when building an application, `main()` is used to call a series of **unit test** functions.

Unit Tests.

The file that contains the `main()` function should also contain a series of unit test functions.

- There should be one unit-test for each week's assignment (each class or pair of classes).
- A unit test should call every public function in its corresponding class, one or more times.
- Choose a sequence of calls that causes every line of code in the class to be executed, including the private functions, if any.
- If you put `main()` at the bottom of the file, you don't need to write prototypes for all the unit tests.

The Header File: Figure 4.1

- The `#pragma once` compiler directive is used to make sure this header file is not included twice in any compile module.
- This does NOT prevent circular `#includes`, which are dealt with another way.
- At the top, `#include "tools.hpp"` and all other programmer-defined header files that will be needed by this class.
- Keep in mind that the tools header includes all the standard header files that you are likely to need. You may write explicit `#includes` for these files, but they just add non-functional lines of code.

The Class Declaration: in the .hpp file

- The first line of the class declaration gives the class name followed by an open-brace. Note the ; after the matching close-brace. (Like a C struct, unlike a Java class.)
- Inside the class definition are the `private` and `public` parts. You usually only need to write these keywords once per class (unlike Java).
- I like to see the data members at the top, functions below.
- Initialize the data members here, if the initial value is known at compile time.
- The public functions provide an interface to the world and control all access to the data members.
- Minimally, define a constructor and a `print()` function for every class. Also define a destructor if any part of the class allocates dynamic memory.

Inline Functions: in the .hpp file

Short functions, defined in the header file are compiled inline.

- Every time an inline function is called, the compiler removes the call and inserts the entire body of the function in its place, with the parameter references replaced by argument values. (Like expanding a macro).
- Unlike macro expansion, this is done with full parameter type-checking and type conversion, if needed.
- There is no stack frame, no jump-to-subroutine, no return.
- This is very time-efficient and is also space-efficient if the function body is short.
- Functions that contain control statements are not expanded inline.

The Output Operator: in the .hpp file

One design goal for C++ was to be able to use a class object in the same ways that predefined objects can be used.

- So we want to read and write class objects using >> and <<.
- We are able to define new methods for global operators, and we normally do define a method for << for each new class.
- The general form of this definition is:

```
inline ostream&
operator<<( ostream& out, Class& obj) {
    return obj.print( out );
}
```

- Place this definition inside the .hpp file after the `};` that terminates the class.

Output: in the .hpp and cpp files

Refer to main.cpp and the definition of `print()` in pack.cpp.

- C++ knows how to format all the built-in types.
- To print a numeric or string value, just send it to an output stream:
  `cout << k <<" " << ary <<"\n";`
- Multiple fields can be printed with one line of code.
- Remember to output spaces after each data item.
- Remember to put a newline on the end.
- You can also use `endl` to end a line.
  `cout << k <<" " << ary <<endl;`
- Formatting is possible in C++, but involves much detail. We will cover it later.

Constructors: in the .cpp file

- A constructor has the same name as the class.
- Put it in the header file if it one line. Otherwise, the prototype goes in the header file and the function definition goes in the .cpp file.
- Its purpose is to initialize objects of that class when they are created by some other class.
- Many constructors don't construct anything – they just execute a bunch of assignments.
- Normally, a constructor will have parameters. Its code will copy those parameter values into the corresponding class members.
- Class members that do not correspond to parameters are initialized by some computation based on a parameter.

The Destructor: in the .hpp or .cpp file

- The name of a destructor is a tilde followed by the class name.
- Put it in the header file if it is short. Otherwise, the prototype goes in the header file and the function definition goes in the .cpp file.
- Its job is to free any dynamic memory, attached to the dying object, that was allocated by the constructor or any other class function.
- If there IS no dynamic memory in the object, you don't need a destructor.
- However, good style says you should write simple applications the same way as complex apps. So write a default destructor: `~Person() = default;`

The .cpp file Implements the Class Functions

Non-inline functions are written in the Classname.cpp file.

- At the top, `#include` ONLY the header file for the class.
- The exception to this rule is complex situations in which you need to break a circular `#include`.
- Please start each function definition with a dashed comment line. Do this out of respect for me, and because you realize I have a lot of trouble seeing things.
- Each function definition must include the name of the class, between the return type and the function name:

```
//---------------------------------
void Person::print(){...}
```

Print Functions

- To debug a class, you have to be able to see the data in its objects.
- For this purpose, you will define a function named `print()` for every class you write.
- The `print()` function will format and print all of the data members of the class.
- Use this pattern for your print functions:

```
ostream& print( ostream& out ){
    out <<mem1 <<" \t" <<mem2 <<endl;
    return out;
}
```