C++PROG 711 - C++ Programming: Introduction

Instructor: David Nielsen
nielsen@uw.edu

Course Times

Each week of the course there will be a synchronous online learning session. These sessions are Tuesday evenings from 7:00pm to 8:30 Pacific Time. Topics will vary week to week based on the predefined material and student requests. See Course Resources on the course Canvas site for details.

Course Preview:

- 10 lessons with corresponding reading assignments
- Weekly online learning sessions
- Homework assignments
- Course project

Course Overview

Description

C++ Programming: Introduction is the first course of a three-course sequence designed to teach the C++ language and to guide you in understanding and applying object-oriented design principles. Each course builds upon what you have learned in the previous one by introducing new concepts and developing existing ones to a greater depth. This hands-on class introduces object-oriented design as a powerful new approach to programming. You are encouraged to develop and practice good software design methodologies using the C++ language.

The course begins by looking at how we would like to use the language. As we do that, we will examine the process of moving from the problem statement to a well-thought-out C++ object-centered design. We will then move deeper into the development process as we study models, modeling, and several different tools that can help us manage software complexity and develop high-quality software applications. We will follow by looking at the C++ extensions and additions to the C language. We will examine the class construct and objects and messages, followed by the ideas of classes and inheritance—two of the strengths of C++ and object-oriented design. This course concludes by extending the notion of reusability with the topic of polymorphism.

The last third of the course includes a programming project. The project is designed to be a scaled down simulation of a real-world project. It includes a student-written specification, design description, and test plan, and concludes with the implementation and testing.

Note: Although C++ 711 is part of the C++ Certificate Program, you do not need to pursue the C++ certificate to take this course.

Course Goals

By the end of the class, you will have a working knowledge of C++ fundamentals and will be able to design, write, and debug a robust multiple-file program of moderate size.

Technology Requirements

You will need to have a C++11 compiler. You need to know how edit, compile, and run a program on your chosen C++ compiler.
Recommended compilers available for Windows and most other popular operating systems include

- Visual Studio 2013 or newer.
- GCC. This is a common C++ compiler for Linux users. Information is at: http://gcc.gnu.org/ A windows version is provided with Code::Blocks.
- Xcode. A C/C++ development environment for Mac OS. http://developer.apple.com/technologies/tools/whats-new.html

Required Materials

- Lippman, Stanley. C++ Primer: 5th ed., Addison Wesley, 2013. ISBN: 978-0-321-71411-4.
- Meyers, Scott. Effective C++: 55 Specific Ways to Improve Your Programs and Designs: 3rd ed., Addison Wesley, 2005. ISBN: 0321334876.

Lippman's book is our main class textbook.
Meyer's book describes (in a witty style) how to best use C++.

Recommended Materials

The following books are good as C++ language desk references. Eckel's book provides a good introduction to object oriented programming in C++. Note that Eckel's book has not been updated for C++11.

- Fowler, Martin, UML Distilled, 3rd ed. Prentice Hall, 2003. ISBN: 0321193687
- Meyers, Scott. Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14, O'Reilly ISBN: 978-1491903995
- Josuttis, Nicolai. The C++ Standard Library: A Tutorial and Reference (2nd Edition).Addison Wesley, 2012. ISBN: 0321623215.
- Stroustrup, Bjarne. The C++ Programming Language: 4th ed., Addison Wesley, 2013. ISBN: 0-321-56384-0.
- Eckel, Bruce, Thinking in C++ (2nd Edition)

About the Lessons

The course is divided into 10 lessons. Each lesson includes:

- A list of relevant chapters from the text and references
- Commentary on the lesson topic that includes examples of the concepts
- A set of practice exercises at the end that will help you review and reinforce what you have learned

See the Course Schedule on the Canvas course site for details about each lesson

Evaluation/Grading

Most lessons conclude with a code or written assignment. You will receive a grade of either satisfactory completion (SC) or unsatisfactory completion (UC) for the course based on your performance on the assignments.

To obtain a SC grade, you must maintain the equivalent of an 80% average as compared to a graded university-level course. Your instructor will provide you with graded and written feedback on each of your assignments within one week of your submission. In addition, the instructor will provide you with a cumulative summary of your progress with each homework assignment.

A completed coding assignment consists of a code listing, a copy of the output, and annotated test cases and their results. Assignments are evaluated based upon:

- Clarity and consistency of style (30%);
- The correctness of the solution (40%); and
- Demonstrated testing of the solution (30%).

In weeks 7-9, you will work on a programming project that simulates a real-world project development project using C++. You will develop it using a standard software development process. This includes a specification phase, design phase, implementation phase, and test phase.

You should demonstrate as many of the principles of object-oriented design as possible in your project. Your grade will be based on style, the thoroughness of your design, the correctness of your implementation, and the completeness of your testing.

The project will be due in three phases:

1. Proposal;
2. Specification and design; and
3. Implementation and testing.

I recommend making a schedule for yourself and trying to stick to it. This should like a compressed version of real-life software development. Good object-oriented design is an iterative process. Experiment with your design while you are in the specification phase, even to the point of thinking about the implementation.

When choosing your project, consider the following:

- The time you have to do the project;
- The relevance of the project to the work you do; and
- Whether this project works well for an object-oriented design.

How to submit homework

Each assignment has a corresponding assignment drop box. Zip the source files from your homework into a single zip file. Upload this single zip file to the assignment drop box.

- Do include your code .cpp and .h files and your unit test .cpp files
- Do not include exe, obj, or other results of compilation and linking. This will make your zip files smaller.

- If you are using Visual Studio, you may optionally turn in the .vcproj and .sln files
- It's best to place all the files in a single directory as this makes it easier to find the pertinent files
- Use the same names for class and function names as specified in the assignments. This improves grading quality as it makes it easier to find things.

How to ask a question

If you have a question on a lesson or assignment, you can post the question on the corresponding lesson forum. Using the forum has a couple of benefits. Other students often have the same question you have and thus will benefit from the discussion. Other students may have answers to your questions, and they may post more quickly than the instructor does. I've found that while students learn a great deal from textbooks and the instructor, they also learn from collaborating with their classmates. Forums provide the mechanism for collaboration for online classes.

How to ask for help on a problem

When you ask for help on a specific programming problem, please include a short program fragment illustrating the problem. If you can distill your problem to a small example, you will get a quicker reply. Quite often, during the process of distilling down your example, you will determine the cause of the problem on your own.

Once you have distilled the problem into a small example, you could post the problem and question to the appropriate lesson forum. Alternately, you could email the instructor for a private discussion of the problem.

Remember, too, that the compiler only identifies a problem at the point where it can't figure out what to do. The cause of the problem is often in code earlier in the program from the point at which the compiler gave up. In addition, the compiler may think there is no problem, so the symptoms won't appear until the program is executing. This means that including only the line where you get the problem may be of little help to the instructor. For example, here's a fragment of an original program:

```
#include <iostream>
using namespace std;
int main()
{
    char* aPointer = 0; // Source of problem here
    // Assume 100 lines of code

    cout << *aPointer;  // Symptom of problem here
    return 0;
}
```

Do not send your instructor a note like the following:

> Dear Instructor:
>
> I am getting an error on this line of code every time I run my program:
>
> cout << *aPointer;
>
> What is wrong?
>
> Thank you,
>
> C. Student

From this note, your instructor will be unable to figure out what's going on because there is no error in the line of code shown in the note.

Now, let's look at different problem and an example of the note you **should** send:

> Dear Instructor:
>
> I am having a problem whenever I return from a function called Calculate(). Here is a list of tests and the values I've used:
>
> **** < list of tests and their values>
>
> I have attached the Calculate() function and the code that calls it. I think the problem seems to occur when I use large numbers but I don't see how it's happening.
>
> Thanks,
>
> A. Student