Agenda / Learning Objectives:

1. Run the following command and extract lab08.tar in your venus account (note the dot):

   cp ~ctse/cs211/lab07.tar . ; tar xvf lab07.tar

2. How do we debug with print statements?
3. Read the tips and pitfalls mentioned in chapter 6 (see page 3)
4. Work on the following two programming exercises from chapter 6 of the textbook.

Chapter 6 q1:

Write a grading program for a class with the following grading policies:

a. There are two quizzes, each graded on the basis of 10 points.
b. There is one midterm exam and one final exam, each graded on the basis of 100 points.
c. The final exam counts for 50% of the grade, the midterm counts for 25%, and the two quizzes together count for a total of 25%. (Do not forget to normalize the quiz scores. They should be converted to a percentage before they are averaged in.)

Any grade of 90 or more is an A, any grade of 80 or more (but less than 90) is a B, any grade of 70 or more (but less than 80) is a C, any grade of 60 or more (but less than 70) is a D, and any grade below 60 is an F. The program will read in the student's scores and output the student's record, which consists of two quiz and two exam scores as well as the student's average numeric score for the entire course and final letter grade. **Define and use a structure for the student record**.

Chapter 6 q8:

Define a **class** named **Money** that stores a monetary amount. The class should have two private integer variables, one to store the number of dollars and another to store the number of cents. Add accessor and mutator functions to read and set both member variables. Add another function that returns the monetary amount as a double. Write a program that tests all of your functions with at least two different **Money** objects.

From teacher's note for Absolute C++:

1) In earlier chapters, we saw the array, the first of the C++ tools for creating data structures. The other tools are the `struct` and the `class`. We saw that the **array** provides a **homogeneous, random access data structure**. This chapter introduces tools to create **heterogenous data structures, namely the struct and class**. While the struct and class are identical except for default access, the text takes the didactic approach of first ignoring struct function members. This chapter deals with the struct as C treats it. The text then develops the simpler ideas involved in declaration and access for the struct. Then the machinery of class creation, protection mechanisms and some ideas of encapsulation of functions with data are treated. Constructors are left to Chapter 7.

2) **Encapsulation**. The notion of encapsulation means, "to collect together, as if to place in a capsule", from which we may infer the purpose, "to hide the details". The text says a data type has a set of values to be manipulated, and sets of operations to manipulate the values, but the details are available to the user or client. A data type becomes an abstract data type if the client does not have access to the implementation details.

3) **Thinking Objects.** When programming with classes, data rather than algorithms takes center stage. The difference is in the point of view compared to students that have never programmed with objects before.

4) **Pitfalls**

Omitting the semicolon at the end of a struct or class definition. The text points out that a structure definition is required to have a semicolon following the closing curly brace. The reason is that it is possible to define a variable of the struct type by putting the identifier between the closed curly brace, }, and the semicolon.

```
struct A
{
  int a;
  int b;
} c;
```

The variable c is of struct type A. With some compilers, this pitfall generates particularly uninformative error messages. It will be quite helpful for the student to write several examples in which the closing semicolon in a structure definition is deliberately omitted.