Review from last Thursday’s Class
Koenig & Moo: Chapter 3

• `vector` container class,
• `sort` function,
• C++ and standard library:
  – `double`,
  – reading an arbitrarily long sequence of values,
  – i/o manipulation

Today’s Class
Koenig & Moo: Chapter 4

• There’s a lot of material in this chapter. The most important topics to focus on are:
  – Using functions to improve clarity and promote code re-use
  – Separating a program across multiple files.
  – Structs for organizing data.
• We will not cover exceptions and you will not be responsible for them.
• Pay attention to the use of output stream formatting and use it in your programs. You will not be tested on it.

Functions
Purpose:
• Break program up into coherent, logical, testable units.
• Make program easier to read.
• Develop code that can be re-used in other programs

We will rewrite the program `average_median_mode.cpp` to use functions and use this to review aspects of parameter passing.
Program Structure

- Function prototypes are at the top of the file
- Function calls are made from the main program
- Different, more general names of functions and parameters are chosen.
- Both the `compute_median` and `compute_mode` functions sort the values because they may be called independently.
- No output is done from the functions.
- Asserts are used instead of exceptions; they are much simpler.

Parameter Passing — A Review

- Parameters and argument lists
- A reference is an alias, not a new variable
- Three different types of parameter passing:
  - Pass by value
  - Pass by reference
  - Pass by constant reference
- Which should you use when?

Splitting Programs Across Multiple Files

- Purpose: readability, testability, code reuse — a repeat of the reasons for using functions..
- Header (.h) files contain declarations:
  - function prototypes
  - struct declarations (see below), and
  - class declarations (later in the semester).
  These all tell the compiler what functions and structs look like without giving the details of how they are implemented.
- Source (.cpp) file contain definitions:
Mostly function implementations

- Header files are included in multiple source files, and NOT compiled on their own.

- Source files are compiled separately, and the resulting object files (.o files) are joined together at the linking stage.

- An example with the average, median, and mode program is attached to the notes and will be reviewed in class.

- In lab we will see how to handle this in visual studio.

struct

- Method of grouping / organizing data

- Example: for a student you might keep his/her name, id number, test scores, homework scores, and averages.

- Here is how it will appear as a struct:

```cpp
struct StudentRec {
    string first_name;
    string last_name;
    string id_number;
    vector<int> hw_scores;
    double hw_avg;
    vector<int> test_scores;
    double test_avg;
    double final_avg;
}; // semi-colon is required!
```

- Here's an example of manipulating the struct:

```cpp
StudentRec rec;
rec.id_number = "012340123";
rec.first_name = "John";
rec.last_name = "Doe";
rec.test_scores.push_back( 98 );
rec.test_scores.push_back( 92 );
StudentRec y;
```
y.first_name = rec.first_name;
y.last_name = "Appleseed";
cout << "First two test scores: "
   << rec.test_scores[0] << " " << rec.test_scores[1]
   << endl;

• Note the use of the . notation
  – rec is a variable of type StudentRec, which is a struct
  – id_number, first_name, test_scores, etc., are data members declared in the struct StudentRec
  – There is one instance of each data member for each StudentRec instance.
  – Data members are accessed from the variable name using the "." notation.

Student Grading Program
The use of structs and separate program is combined in a program called student_avg. Here are several of its properties:

• Uses a separate header and source file for StudentRec declarations and definitions.

• Reuses the statistics code, but only a small part of it.

• StudentRec variables are passed as parameters just like ordinary variables.

• We pass a comparison function called less_names to the sort function to alphabetically sort student records:
  – It uses the operator < which works on strings by returning true if the first string is lexicographically less than the second.
  – It combines comparisons on first and last names to properly sort.
  – The constant reference passing of the parameters is required!

• My version of the algorithm header does not include the function max so I defined it here for unsigned ints.

• Output formatting is done using the length of the strings. (Note that the output formatting in the program is not perfect.)
Topics We Did Not Cover

We will briefly touch on the following. See the text for details if you are interested:

• exceptions

• i/o error states

• output formatting