Looking Ahead in February

- Week 4 homework is due next Wednesday.
- Lab will be held next Wednesday (February 13).
- No homework will be given next Wednesday.
- Test 1 is Tuesday, February 19, in class. It will cover through the end of Chapter 6. It will be closed-book, BUT photocopies of the “Details” sections from the end of each chapter will be handed out to each student.
- Lab will be held on Wednesday, February 20th.
- A two-week long assignment will be given on Wednesday, February 20th.

Hints on Approaching the Week 4 Homework

- What struct is needed?
- How are the structs stored?
- How should the main program / main flow of the program be organized?
- What functions are needed and what should they do?
- Write the function prototypes
- Write the main program
- Write the functions — one per input option
- Compile incrementally:
  - Write empty skeletons of functions
  - Compile after main program is written
  - Compile after each function is written
• Test incrementally:
  – Write a function to output the contents of the data structure (the set of courses), just like we did in lab.
  – Use small amounts of input (smaller than the example I gave) and test each function to make sure it behaves properly. Check by comparing the printed contents of the data structure to what you think should be there.
  – Test as many possibilities as you can for each function before moving on to the next function.

If you seek help from me or from a TA, we will want to make sure that you have followed the above steps or something similar. If you haven’t, we will make you work through them.

**Review from Monday’s Class**
Koenig & Moo: Chapter 5.1-5.5
• Iterators and iterator operations. We will review:
  – Erase
  – Difference between p and *p
• Lists as a different sequential container class
• Differences between lists and vectors
• Prime numbers example

**Today’s Class — Programming Examples**
Koenig & Moo: Sections 5.6-5.9, 6.1 and 6.4
• More on strings: reading lines and character testing
• Finding lines that are palindromes
• Generic functions
• Splitting up a string
String and Character Manipulation

- Reading a line at a time — getline. Here's the prototype:

```
istream& getline( istream&, string& );
```

Returning the `istream` reference seems a bit strange, but it is common practice.

- The `string` class has a `substr` member function that extracts a substring starting at a given location. For example:

```
string s = "Hello world";
string t = s.substr(6,5);
cout << t << endl;  // Outputs: world
```

- The header file `<cctype>` provides prototypes for character functions from the C library (hence the 'c' in front of 'ctype'). Here are some examples:

  - `isspace(c)`
  - `isalpha(c)`
  - `isdigit(c)`
  - `isupper(c)`
  - `tolower(c)`

Writing a Program Find Palindromes

- A palindrome is a string that reads the same forward and backward.

- We will write a program to read lines of input and determine if the alphabetic letters on the line form a palindrome.

- This will illustrate several of the functions described above.

- A skeleton of this program is provided. We will write the rest in class.
Generic Functions — Overview

- Library functions that are not associated with any particular container or type stored in the container. This is a powerful idea!
  - We have already looked at `sort` extensively
- Usually, generic functions work through iterators or iterators in combination with type-specific functions.
- We will look at an example where a function called `find_if` is used in breaking up an input line into strings of contiguous non-whitespace characters.

Breaking Up a Line Into Strings

Program is attached (`break_up.cpp`).

- Two different versions of major function to break up a string.
- Both return a vector of strings
- The first version uses indexing on strings and the `substr` member function.
  - We will look at the structure of the loops.
- The second version uses iterators and the generic `find_if` function:
  - The prototype for this function is in `<algorithm>`.
  - Aside: iterators are defined on strings
  - constant iterators `beg` and `end` are used to delimit the bounds of the substring of interest. The `find_if` function is used to fill them in.
  - `find_if` takes the start and end of the search interval and a boolean function
    * It returns an iterator indicating the first location (at `beg` or beyond) where the boolean function returns true when applied to what’s stored at the iterator.
    * It returns the end of the search interval if the boolean function returns false on each entry in the interval.
    * The iterator denoting the end of the search interval is not tested.
- The substring to be stored in the vector is constructed just from the iterators!

More On Generic Functions

- Many functions exist. Here are a few:
  - `accumulate( beg_itr, end_itr, init_value);`
  - `equal( beg_itr, end_itr, beg_itr2 )`
  - `find( beg_itr, end_itr, value)`
  - `copy( beg_itr, end_itr, itr`

- The text discusses the notion of an *iterator adaptor*, which is needed for the copy function. We will not use this idea often, at least not yet.

- More details on how to use these functions and on what functions exist are in the text. Even more details are in resources linked from the web page.

- In general, it is often easy to guess both what a function does and how to use it!

- You can use some of these in your week 4 homework, but you do not need to.