Announcements

• Lab grades:
  – 1 and 2 ==> 3;
  – 3 and 4 ==> 4

  The same modification is in force for the week 3 lab.

• Test 1 is next Tuesday, in class.

• All relevant solutions will be posted on the web by mid-day on Sunday.

• The TAs will be organizing an informal help session to be held late Monday afternoon. Watch the webpage for an announcement about specifics.

• Lab WILL be held next Wednesday. It will (slowly) cover maps.

Monday’s Class

• Generic functions (Week 4, Thursday notes)

• Splitting up a string (Week 4, Thursday notes)

• Maps: associative containers

• Counting word occurrences

Today’s Class — Catch-up and Review

• Yesterday’s lab

• Using generic functions

• Alternative solutions to the word counting problem

• Overview for Test 1

• Review questions
Lab Solution

- Use of find_if in not_all_whitespace
- Combination of remove_if and erase in remove_punctuation_and_caps
- Counting words on a line is a slight rewrite of the break_v2.
- remove_extra_blanks requires repeatedly finding the beginning and ending of words.
- Centering the output requires a loop to find the longest string (line) and a loop to output the strings.

Thinking About Generic Functions

- Play with them using small examples. I can’t stress this enough.
- Write your own versions to try to figure out what they do. You can’t do this completely, because you don’t (yet) know how to make functions generic. Instead you will need to use specific types.
- Examples that we can work on in class:
  - accumulate
  - find
  - equal

Alternative Solutions to the Word Counting Program

- Vector of strings:
  - Enter each string into a vector,
  - Sort the vector,
  - Count occurrences of each word (as in the mode function)
• Vector of structs:
  – Each struct has a string and an integer counter. This is like a pair.
  – For each new word, search the vector for the word.
  – Increment the count if word is there; insert new struct if the word is not there.
  – Sort the results and output.

• Which approach do you like better?

Test 1 Overview

• Tuesday, February 19, in class.

• Coverage: through the end of Chapter 6. (Nothing on maps.)

• Closed-book, closed-notes, no crib sheets. BUT, photocopies of the “Details” sections from the end of each chapter will be handed out to each student

• Format:
  – One problem will require writing a short, but complete program.
  – There will be problems on parameter passing and iterator assignments, similar to what has been discussed in class.
  – The remaining problems will require writing code segments or short functions. These will vary in difficulty, but will be similar to lab checkpoints and the short homework problems.
Example Problems

Solutions to these problems will be posted on the web.

1. Given a vector of integers, \( v \), write a short code segment that uses iterators to add every second entry in the vector. For example, when \( v \) is

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 16 & 4 & -3 & 2 & 76 & 9 & 3 & 6
\end{array}
\]

the segment should add \( 1 + 4 + 2 + 9 + 6 \) to get 22. Store the result in a variable called \( \text{sum} \). Repeat the problem using array indices instead of iterators.

2. What is the output of the following code fragment?

```c
void foo( int x, int & y )
{
    int t=x;
    x = y;
    y = t;
    cout << x << " " << y << "\n";
}

int main()
{
    int a = 6;
    int b = 3;
    foo( a, b );
    cout << a << " " << b << "\n";
}
```

3. Given a struct:

```c
struct bar {
    int x;
    double y;
    list<string> z;
};
```

Write a function that rearranges a vector of \( \text{bar} \)'s so that they are ordered by decreasing value of \( x \), and by increasing \( y \) for \( \text{bar} \)'s with
equal values of \( x \). Also, make sure that each \( \text{bar} \) has its list \( z \) in increasing order.

4. Write a code segment that copies the contents of a string into a list of char in reverse order.

5. Write a function that takes a list of doubles and copies its values into two new lists of doubles, one containing only the negative numbers from the original list, and the other containing the non-negative numbers from the original list. For example, if the original list contains the values

\[-1.3, 5.2, 8.7, 0.0, -4.5, 7.8, -9.1, 3.5, 6.6\]

then the resulting list containing the negative numbers should contain

\[-1.3, -4.5, -9.1\]

and the resulting list containing the positive numbers should contain

\[5.2, 8.7, 0.0, 7.8, 3.5, 6.6\]

Start this problem by writing the function prototype as you think it should appear (hint: there should be three parameters) and then write the code.

6. Write a code segment that removes all occurrences of the letter ‘c’ from a string. Consider both uppercase ‘C’ and lowercase ‘c’. For example, the string

\[\text{Chocolate}\]

would become the string

\[\text{hoolate}\]

Start by not using any generic functions and then repeat the problem using \texttt{remove} and \texttt{erase}.

7. Write a function that rearranges a list of doubles so that all the negative values come before all the non-negative values AND the order of the negative values is preserved AND the order of the positive values is preserved. For example, if the list contains
Then the modified list should contain

\[-1.3, -4.5, -9.1, 5.2, 8.7, 0.0, 7.8, 3.5, 6.6\]

This is perhaps a little more challenging than some of the other problems, but still instructive.

You can also expect to see questions that depend on what you learned in writing the programs for the Week 3 and Week 4 homeworks. If you had trouble with either of these, be sure to look at the solutions!