CS106B
Summer 2012

Handout #08S
July 2nd, 2012

Section Solutions 1

Problem 1. Big Brother is Watching

There are several strategies for implementing the character-removal problem. The implementations shown below go through the **text** string and then check to see whether the character in that position appears in the **remove** string. Another possible (but generally less efficient) approach would be to make several passes over the **text** string, moving one character from the **remove** string on each pass.

```
/*
 * Function: censorString1
 * Usage: s = censorString1(text, remove);
 * ---------------------------------------
 * This function takes two strings and returns the first string with
 * all the occurrences of letters in the second string removed.
 * It uses a for loop to iterate through the original string and
 * the find method to check whether that character is in the remove
 * string.  This version builds a new string character by character.
 */

string censorString1(string text, string remove) {
   string result = "";
   for (int i = 0; i < text.length(); i++) {
      if (remove.find(text[i]) == string::npos) {
         result += text[i];
      }
   }
   return result;
}

/*
 * Function: censorString2
 * Usage: censorString2(text, remove);
 * -----------------------------------
 * This function takes two strings and updates the first string
 * by removing all occurrences of letters in the second string.
 * Note that the implementation must decrement i after removing
 * the character to ensure that the following character is checked.
 */

void censorString2(string & text, string remove) {
   for (int i = 0; i < text.length(); i++) {
      if (remove.find(text[i]) != string::npos) {
         text.replace(i, 1, "");
         i--;
      }
   }
}
```

Problem 2. How Did We Do?

```
/*
 * Function: readStats
 * Usage: readStats(filename, min, max, mean);
 * -------------------------------------------
 * Reads a data file whose name is given in filename and computes the
 * minimum score, the maximum score, and the average score, storing
 * these values in the reference parameter variables min, max, and mean.
 */

void readStats(string filename, int & min, int & max, double & mean) {
   ifstream in;
   in.open(filename.c_str());
   if (in.fail()) error("Couldn't read " + filename);
   double total = 0;
   int count = 0;
   while (true) {
      int score;
      in >> score;
      if (in.fail()) break;
      if (score < 0 || score > 100) error("Score out of range");
      if (count == 0 || score < min) min = score;
      if (count == 0 || score > max) max = score;
      total += score;
      count++;
   }
   mean = (double) total / count;
   in.close();
}
```

Problem 3. Stacking Cannonballs

```
/*
 * Function: cannonball
 * Usage: n = cannonball(height);
 * ------------------------------
 * This function computes the number of cannonballs in a stack
 * that has been arranged to form a pyramid with one cannonball
 * at the top sitting on top of a square composed of four
 * cannonballs sitting on top of a square composed of nine
 * cannonballs, and so forth. The function cannonball computes
 * the total number based on the height of the stack.
 */

int cannonball(int height) {
   if (height == 0) {
      return 0;
   } else {
      return height * height + cannonball(height - 1);
   }
}
```