Section Handout 3

Based on a handout by Eric Roberts

Problem 1. Weights and Balances

```
/*
 * Function: isMeasurable
 * Usage: if (isMeasurable(target, weights) . . .
 * ----------------------------------------------
 * Determines whether it is possible to measure the specified target
 * weight using some combination of the weights stored in the vector
 * weights.  To do so, it recursively attacks the problem by considering
 * only the first weight in the array, which gives rise to the following
 * possibilities:
 *
 * 1. The first weight is unused.  In this case, it is possible
 *    to measure the target weight only if it is possible to do
 *    so using the remaining weights.
 *
 * 2. The first weight goes on the opposite side of the balance
 *    from the sample.  In this case, the target weight is
 *    effectively decreased by first, which means it can be
 *    measured only if it is possible to measure target - first
 *    ounces using the other weights.
 *
 * 3. The first weight goes on the same side of the balance
 *    from the sample.  In this case, the target weight is
 *    effectively increased by first, which means it can be
 *    measured only if it is possible to measure target + first
 *    ounces using the other weights.
 *
 * The simple case occurs when there are no weights at all, in
 * which case the target weight is measurable only if it is 0.
 */

bool isMeasurable(int target, Vector<int> & weights) {
   if (weights.isEmpty()) {
      return target == 0;
   } else {
      int first = weights[0];
      Vector<int> rest = weights;
      rest.removeAt(0);
      return isMeasurable(target, rest)
          || isMeasurable(target - first, rest)
          || isMeasurable(target + first, rest);
   }
}
```

Problem 2. Filling a Region

```
/*
 * Function: fillRegion
 * Usage: fillRegion(grid, row, col);
 * ----------------------------------
 * This function paints black pixels everywhere inside the
 * region at the specified row and column.
 */

void fillRegion(Grid<bool> & pixels, int row, int col) {
   if (pixels.inBounds(row, col) && !pixels[row][col]) {
      pixels[row][col] = true;
      fillRegion(pixels, row + 1, col);
      fillRegion(pixels, row - 1, col);
      fillRegion(pixels, row, col + 1);
      fillRegion(pixels, row, col - 1);
   }
}
```

Problem 3. Generating Multiword Anagrams

```
/*
 * Function: findAnagram
 * Usage: bool found = findAnagram(letters, english, words);
 * ---------------------------------------------------------
 * Finds a multiword anagram for the specified set of letters.
 * using only English words from the dictionary in english in
 * which each word must be at least MIN_WORD characters long.
 * If the program finds any anagrams, it stores the list of words
 * in the vector words and returns true.  If no anagrams exist,
 * the function returns false.
 */

bool findAnagram(string letters, Lexicon & english, Vector<string> & words) {
   return findAnagramWithFixedPrefix("", letters, english, words);
}

/*
 * Function: findAnagram
 * Usage: bool found = findAnagram(prefix, letters, english, words);
 * -----------------------------------------------------------------
 * Finds a multiword anagram for the specified set of letters, where
 * the current word must begin with the specified prefix.
 */

bool findAnagramWithFixedPrefix(string prefix, string rest,
                                Lexicon & english,
                                Vector<string> & words) {
   if (!english.containsPrefix(prefix)) return false;
   if (english.contains(prefix) && prefix.length() >= MIN_WORD) {
      if (rest == "" || findAnagram(rest, english, words)) {
         words.add(prefix);
         return true;
      }
   }
   for (int i = 0; i < rest.length(); i++) {
      string otherLetters = rest.substr(0, i) + rest.substr(i + 1);
      if (findAnagramWithFixedPrefix(prefix + rest[i], otherLetters,
                                     english, words)) return true;
   }
   return false;
}
```