Eric Roberts
CS106B

Handout #17A
January 23, 2013

Solutions to Section Handout #2

Problem 1. Using grids

```
/*
 * Function: fixCounts
 * Usage: fixCounts(mines, counts);
 * --------------------------------
 * Uses the first grid to indicate where mines are located and
 * creates a second grid showing the count of mines in the
 * neighborhood of each square.
 */

void fixCounts(Grid<bool> & mines, Grid<int> & counts) {
   int nRows = mines.numRows();
   int nCols = mines.numCols();
   counts.resize(nRows, nCols);
   for (int i = 0; i < nRows; i++) {
      for (int j = 0; j < nCols; j++) {
         counts[i][j] = countNearbyMines(mines, i, j);
      }
   }
}

/*
 * Function: countNearbyMines
 * Usage: int nMines = countNearbyMines(mines, row, col);
 * ------------------------------------------------------
 * Counts the number of mines in the immediate neighborhood.  In this
 * code, the neighborhood includes the current square, but that
 * distinction would never come up in the actual game.
 */

int countNearbyMines(Grid<bool> & mines, int row, int col) {
   int nMines = 0;
   for (int i = -1; i <= 1; i++) {
      for (int j = -1; j <= 1; j++) {
         if (mines.inBounds(row + i, col + j)) {
            if (mines[row + i][col + j]) nMines++;
         }
      }
   }
   return nMines;
}
```

Problem 2. Using queues

```
/*
 * Function: reverseQueue
 * Usage: reverseQueue(queue);
 * ---------------------------
 * Reverses the order of the elements in a queue.  This
 * implementation does so by using a stack to hold the
 * values as they are being reversed.
 */

void reverseQueue(Queue<string> & queue) {
   Stack<string> stack;
   while (!queue.isEmpty()) {
      stack.push(queue.dequeue());
   }
   while (!stack.isEmpty()) {
      queue.enqueue(stack.pop());
   }
}
```

Problem 3. Using maps

```
/*
 * File: MorseCode.cpp
 * -------------------
 * This program translates to and from Morse Code using maps to
 * assist in the translation.
 */

#include <iostream>
#include <string>
#include "console.h"
#include "error.h"
#include "map.h"
#include "strlib.h"
using namespace std;

/* Function protoypes */

string translateLettersToMorse(string line);
string translateMorseToLetters(string line);
Map<string,string> invertMap(const Map<string,string> & map);
Map<string,string> createMorseCodeMap();

/*
 * Constant maps: LETTERS_TO_MORSE, MORSE_TO_LETTERS
 * -------------------------------------------------
 * These variables contain maps that convert in each direction between
 * uppercase letters and their Morse Code equivalent.  Because these
 * variables are initialized once and retain their values throughout
 * the lifetime of the program, they are best treated as constants
 * that are shared among the different functions instead of as variables
 * that are passed as parameters.
 */

const Map<string,string> LETTERS_TO_MORSE = createMorseCodeMap();
const Map<string,string> MORSE_TO_LETTERS = invertMap(LETTERS_TO_MORSE);

/* Main program */

int main() {
   cout << "Morse code translator" << endl;
   while (true) {
      string line;
      cout << "> ";
      getline(cin, line);
      if (line == "") break;
      line = toUpperCase(line);
      if (line[0] == '.' || line[0] == '-') {
         cout << translateMorseToLetters(line) << endl;
      } else {
         cout << translateLettersToMorse(line) << endl;
      }
   }
   return 0;
}
```

☞

```
/*
 * Function: translateLettersToMorse
 * Usage: string morse = translateLettersToMorse(line);
 * ---------------------------------------------------
 * Translates a string of letters into Morse Code characters separated
 * by spaces.  Characters that don't appear in the table are simply ignored.
 */

string translateLettersToMorse(string line) {
   string morse = "";
   for (int i = 0; i < line.length(); i++) {
      string letter = toUpperCase(line.substr(i, 1));
      if (LETTERS_TO_MORSE.containsKey(letter)) {
         if (morse != "") morse += " ";
         morse += LETTERS_TO_MORSE.get(letter);
      }
   }
   return morse;
}

/*
 * Function: translateMorseToLetters
 * Usage: string letters = translateLettersToMorse(line);
 * -----------------------------------------------------
 * Translates a string in Morse Code into English letters.
 * Because word breaks are not represented in Morse code, the
 * letters in the output will be run together.  The characters
 * of the Morse Code input must be separated by a single space.
 * Any other character in the input is simply ignored.  If there
 * is no English equivalent for the Morse Code character, this
 * function indicates that fact by inserting a question mark (?).
 *
 * Implementation note: To eliminate the special case of the last
 * character in the line, this function begins by adding a space
 * to the end of the input string.
 */

string translateMorseToLetters(string line) {
   line += " ";
   string letters = "";
   string morse = "";
   for (int i = 0; i < line.length(); i++) {
      char ch = line[i];
      if (ch == '.' || ch == '-') {
         morse += ch;
      } else if (ch == ' ') {
         if (MORSE_TO_LETTERS.containsKey(morse)) {
            letters += MORSE_TO_LETTERS.get(morse);
         } else {
            letters += '?';
         }
         morse = "";
      }
   }
   return letters;
}
```

☞

```
/*
 * Function: invertMap
 * Usage: Map<string> inverse = invertMap(map);
 * --------------------------------------------
 * Creates an inverted copy of the specified map in which the values
 * in the original become the keys of the new map and refer back to
 * their associated keys.  Thus, if "abc" is bound to "xyz" in the
 * original map, the inverted map will bind "xyz" to "abc".  If two
 * keys in the original map have the same value, this function will
 * signal an error condition.
 */

Map<string,string> invertMap(const Map<string,string> & map) {
   Map<string,string> inverse;
   foreach (string key in map) {
      string value = map.get(key);
      if (inverse.containsKey(value)) {
         error("That map cannot be inverted");
      }
      inverse[value] = key;
   }
   return inverse;
}

/*
 * Function: createMorseCodeMap
 * Usage: Map<string> map = createMorseCodeMap();
 * ----------------------------------------------
 * Returns a map in which each uppercase letter is mapped into its
 * Morse code equivalent.
 */

Map<string,string> createMorseCodeMap() {
   Map<string,string> map;
   map["A"] = ".-";       map["J"] = ".---";     map["S"] = "...";
   map["B"] = "-...";     map["K"] = "-.-";      map["T"] = "-";
   map["C"] = "-.-.";     map["L"] = ".-..";     map["U"] = "..-";
   map["D"] = "-..";      map["M"] = "--";       map["V"] = "...-";
   map["E"] = ".";        map["N"] = "-.";       map["W"] = ".--";
   map["F"] = "..-.";     map["O"] = "---";      map["X"] = "-..-";
   map["G"] = "--.";      map["P"] = ".--.";     map["Y"] = "-.--";
   map["H"] = "....";     map["Q"] = "--.-";     map["Z"] = "--..";
   map["I"] = "..";       map["R"] = ".-.";
   return map;
}
```

Problem 4. Using lexicons

```
/*
 * File: FindPalindromes.cpp
 * -------------------------
 * This program finds all English words that are palindromes.
 */

#include <iostream>
#include <string>
#include <cctype>
#include "console.h"
#include "lexicon.h"
using namespace std;

/* Function prototypes */

bool isPalindrome(string str);
string reverse(string str);

/* Main program */

int main() {
   cout << "This program finds all English palindromes." << endl;
   Lexicon english("EnglishWords.dat");
   foreach (string word in english) {
      if (isPalindrome(word)) {
         cout << word << endl;
      }
   }
   return 0;
}

/*
 * Function: isPalindrome
 * Usage: if (isPalindrome(str)) . . .
 * -----------------------------------
 * Returns true if str is a palindrome, which is a string that
 * reads the same backwards and forwards.
 */

bool isPalindrome(string str) {
   return str == reverse(str);
}

/*
 * Function: reverse
 * Usage: string rev = reverse(str);
 * ---------------------------------
 * Returns a new string consisting of the characters in str in reverse order.
 */

string reverse(string str) {
   string rev;
   for (int i = str.length() - 1; i >= 0; i--) {
      rev += str[i];
   }
   return rev;
}
```