Problem 1. Censorship

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.

```c++
/*
 * 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?

```c
/*
 * 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

```c
/*
 * 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);
    }
}
```
Problem 4: Xzibit Words

One possible implementation is shown here:

```c++
string mostXzibitWord(Lexicon& words) {
    /* Track the best string we've found so far and how many subwords it has. */
    string result;
    int numSubwords = 0;

    foreach (string word in words) {
        /* Store all the subwords we find. To avoid double-counting words, we'll hold this in a Lexicon. */
        Lexicon ourSubwords;

        /* Consider all possible start positions. */
        for (int start = 0; start < word.length(); start++) {
            /* Consider all possible end positions. Note that we include * the string length itself, since that way we can consider * substrings that terminate at the end of the string. */
            for (int stop = start; stop <= word.length(); stop++) {
                /* Note the C++ way of getting a substring. */
                string candidate = word.substr(start, stop - start);

                /* As an optimization, if this isn't a prefix of any legal * word, then there's no point in continuing to extend this * substring. */
                if (!words.containsPrefix(candidate))
                    break;

                /* If this is a word, then record it as a subword. */
                if (words.contains(candidate))
                    ourSubwords.add(candidate);
            }
        }

        /* Having found all subwords, see if this is better than our * best guess so far. */
        if (numSubwords < ourSubwords.size()) {
            result = word;
            numSubwords = ourSubwords.size();
        }
    }

    return result;
}
```

In case you're curious, the most Xzibit word is “foreshadowers,” with 34 subwords!
Problem 5: RNA Protein Codes

Here is one possible implementation:

```cpp
Vector<string> findProteins(string rna, Map<string, string>& codons) {
    Vector<string> result;

    /* Track at which index we are in the string. We'll be going one character
     * at a time through the string. */
    int index = 0;
    while (true) {
        /* Find the next start codon, stopping if none are left. */
        index = rna.find("AUG", index);
        if (index == string::npos) {
            return result;
        }

        /* Keep decoding codons until we hit a stop codon. */
        string protein;
        while (true) {
            /* Read the codon. */
            string codon = rna.substr(index, 3);
            index += 3;

            /* If it's a stop codon, we're done with this protein. */
            if (codons[codon] == "stop")
                break;

            /* Otherwise, add it to the result. To get the commas right, we'll
             * only add commas if the string isn't empty. */
            if (!protein.empty()) protein += ", ";
            protein += codons[codon];
        }

        /* Add this protein to the result. */
        result += protein;
    }
}
```

A process similar to this one is actually going on right now in every single cell in your body. Isn't that amazing?