

Section Solution

Solution 1: String Explosions

```
Vector<string> Explode(string str, char delim)
{
    Vector<string> explosion;
    string cluster;
    str += delim; // last cluster gets appended w/ minimal special casing

    for (int i = 0; i < str.size(); i++) {
        if (str[i] == delim) {
            explosion.add(cluster);
            cluster.clear(); // cluster = "" works, too
        } else {
            cluster += str[i];
        }
    }

    return explosion;
}
```

Solution 2: URL Parameter Map

```
Map<string> extractQueryMap(string url)
{
    Map<string> parameters;
    int index = url.find('?');
    if (index == string::npos) return parameters;

    string query = url.substr(index + 1);
    index = query.find('#');
    if (index != string::npos) {
        query = query.substr(0, index); // chop off hash
    }

    Vector<string> pairs = Explode(query, '&');
    for (int i = 0; i < pairs.size(); i++) {
        Vector<string> components = Explode(pairs[i], '=');
        string key = components[0];
        string value = "true";
        if (components.size() > 1 && !components[1].empty()) {
            value = components[1];
        }
        parameters[key] = value;
    }

    return parameters;
}
```

Solution 3: Happy Numbers

```

int ComputeNext(int num)
{
    int sumOfSquares = 0;
    while (num > 0) {
        int digit = num % 10;
        sumOfSquares += digit * digit;
        num /= 10;
    }
    return sumOfSquares;
}

bool IsHappy(int num)
{
    Set<int> previouslySeen;
    while (num > 1 && !previouslySeen.contains(num)) {
        previouslySeen.add(num);
        num = ComputeNext(num);
    }
    return num == 1;
}

```

Solution 4: FilterPossibilities

```

int CountLetters(string str, char letter)
{
    int count = 0;
    for (int i = 0; i < str.size(); i++) {
        if (str[i] == letter) {
            count++;
        }
    }
    return count;
}

void FilterPossibilities(Set<string>& words, Map<int>& lettersMap)
{
    Set<string> wordsToDelete;
    foreach (string word in words) {
        foreach (string key in lettersMap) {
            char letter = key[0];
            if (CountLetters(word, letter) > lettersMap[key]) {
                wordsToDelete.add(word);
                break; // word's already been marked for removal... move on
            }
        }
    }
    words.subtract(wordsToDelete);
}

```

Another solution might build up a letter frequency map of all of the words in `words`, and that make sure that all counts in this frequency map are less than the corresponding counts in **lettersMap**.