Section Handout #1

Sections will meet once a week to give you a more intimate environment to discuss course material, work through problems, and raise any questions you have. Each week we will hand out a set of section exercises, which are the problems proposed for section that week. While we will not be collecting or grading these problems, you will receive greater benefit from section if you've looked over the problems in advance and tried to sketch solutions. Your section leader won't necessarily cover every exercise in depth, so be sure to speak up if there are particular problems you'd like to focus on. Solutions to all problems will be given in section so you can work through any remaining exercises on your own. Many of our section exercises have been taken from old exams, so they are also an excellent source of study questions when exam time rolls around.

Problem 1: Removing all occurrences of characters

Write a function `CensorString` that takes two strings as input and returns the first string with all of the characters that are present in the second removed.

> "Stanford University" with "nt" removed becomes "Saford Uiversiy"
> "Llamas like to laugh" with "la" removed becomes "Lms ike to ugh"
> and so on . . .

Note that the function is case sensitive. This function could be written two ways. One way is to return a completely new string, and the other is to modify the original string. For practice write both of these functions. First write a function that returns a completely new string with the following prototype:

```
string CensorString1(string text, string remove);
```

and then write a function that modifies the original string with the following prototype:

```
void CensorString2(string & text, string remove);
```

Problem 2: Files and Structs

When we grade your exams, we're going to keep track of some statistics like the min, max and average scores. Define a struct containing these statistics. Then, write a function that takes a filename, reads the scores from it (one per line where 0 <= score <= 100), and returns the struct you defined. For efficiency's sake, your function should make only a single pass over the file.

Problem 3: Vectors
Write a function `CountLetters` that takes a filename and prints the number of times each letter of the alphabet appears in that file. Because there are 26 numbers to be printed, `CountLetters` needs to create a Vector. For example, if the file is:

Abcd K..
ijk;;
cab-Bage
fad

`CountLetters` should print the following:

a: 4
b: 3
c: 2
d: 2
...
z: 0

When you really print this out, it should be 26 elements long, but we couldn't easily display that on the page. Note that there may be upper case letters, lower case letters, and non-letter characters in the file. All letters should be counted regardless of case (so "Aa" is two a's), and non-alphabetic characters should be ignored. You should use the following prototype.

```
void CountLetters(string filename);
```

Problem 4: Memory Diagram
Trace through the following bit of code and draw a diagram showing the contents of memory at the indicated point. Be sure to differentiate between the stack and the heap.

```
struct heroT {
  string name;
  string weakness;
  int powerLevel;
};

struct villianT {
  string name;
  string evilPlan;
  int attackLevel;
};

void Battle(heroT aang, villianT & zuko)
{
  int pos = 1;
  int level = aang.powerLevel;
  string name = zuko.name;
```

```
  while(level > 20)
  {
    zuko.evilPlan[pos--] -= (level / 10);
    level /= 2;
  }

  zuko.attackLevel -= level;

  pos = name.find(aang.weakness);
  while(pos != string::npos)
  {
     aang.powerLevel /= 2;
     name.replace(pos, 2, "");
     pos = name.find(aang.weakness, pos);
  }

  if(aang.powerLevel > zuko.attackLevel)
  {
    zuko.name = "Loser";
  }
  else
  {
    aang.name = "Big Baby";
  }

  /* DRAW THE STATE OF MEMORY HERE */

  return;
}

int main()
{
  heroT julie;
  villianT tom;

  julie.name = "Super Lecturer";
  julie.weakness = "Gr";
  julie.powerLevel = 60;

  tom.name = "Grumpy Grad Student";
  tom.evilPlan = "Frowning";
  tom.attackLevel = 30;

  Battle(julie, tom);

  return 0;
}
```