Section Solutions #1

Problem 1: Removing all occurrences of a character

If we want to remove the occurrences of the letter one at a time, returning a completely new string at the end, we can write the following function:

```
/* Function: CensorString
 * Usage: s = CensorString(input, 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 double for loop to iterate through the string testing each
 * character to see if it matches any of the letters to remove, and
 * building the resultant string character by character.
 */

string CensorString1(string text, string remove)
{
      string result = "";

      for (int i = 0; i < text.length(); i++)
      {
            bool found = false;
            for (int k = 0; k < remove.length(); k++)
            {
                  if (text[i] == remove[k])
                  {
                        found = true;
                        break;
                  }
            }

            if(!found)
            {
                  result += text[i];
            }
      }

      return result;
}
```

We can also do the same thing by using the `.find` and `.substr` methods from the `string` class:

```
string CensorString1(string text, string remove)
{
      int pos;
      string result = text;
```

```
	for(int i = 0; i < remove.length(); i++)
	{

		while (true)
		{
			pos = result.find(remove[i]);
			if (pos == string::npos) // No more of this char
							present
			{
				break;
			}
			else
			{
				// We want stringUntilCh + stringAfterCh
				result = result.substr(0, pos) +
					result.substr(pos + 1);
			}
		}
	}

	return result;
}
```

To write it so that we modify the original string rather than returning a new string, we could do the following:

```
void CensorString2 (string &text, string remove)
{
	for(int i = 0; i < remove.length(); i++)
	{
		int pos = 0;

		while ((pos = text.find(remove[i], pos)) != string::npos)
		{
			text.replace(pos, 1, ""); // replace char with empty
							string
		}
	}
}
```

Problem 2: Files and Structs

```
struct statsT {
      int low;
      int high;
      double average;
};

/* CalculateStatistics()
 * Usage: stats = CalculateStatistics(filename)
 * --------------------------------------------
 * This function keeps track of the running low/high value
 * as it reads the file, as well as a total and a count to compute
 * the average when we're done
 */
statsT CalculateStatistics(string filename) {
      statsT stats;
      // Since we know scores are between 0 and 100, we can set low and
      // high to beyond their range.  This way, the first update is
      // just like the rest.
      // Otherwise, we'd need a sentinel and a little more logic
      stats.low = 101;
      stats.high = -1;

      int total = 0;
      int count = 0;

      // Open a new filestream and make sure it worked
      ifstream in;
      in.open(filename.c_str());
      if (in.fail()) Error("Couldn't read '" + filename + "'");

      while(true) {
            int num;
            in >> num;
            // Check that we read successfully
            if (in.fail()) break;
            // Update or data if we need to
            if (num < stats.low) stats.low = num;
            if (num > stats.high) stats.high = num;
            total += num;
            count++;
      }

      // Don't forget to watch for integer division!
      stats.average = double(total)/count;
      // And make sure to close your files
      in.close();
      return stats;
}
```

Problem 3: Vectors

```
const int AlphabetSize = 26;


void CountLetters(string filename)
{
	// Open a new filestream and make sure it worked
	ifstream in;
	in.open(filename.c_str());
	if (in.fail()) Error("Couldn't read '" + filename + "'");

	Vector<int> result;

	for (int i = 0; i < AlphabetSize; i++)
	{
		result.add(0); // must initialize contents
	}

	string line;
	while(true)
	{
		getline(in, line);
		// Check that we got a line
		if (in.fail()) break;

		line = ConvertToLowerCase(line);
		for (int j = 0; j < line.length(); j++)
		{
			int index = line[j] - 'a';
			if(index >= 0 && index < AlphabetSize) {
				int prevTotal = result[index];
				result[index] = prevTotal + 1;
			}
		}
	}

	for(int k = 0; k < AlphabetSize; k++)
	{
		char currLetter = 'a' + k;
		cout << currLetter << ": " << result[k] << endl;
	}
}
```

Problem 4: Memory Diagram