Section Handout 1

_____

Each week, you'll meet for about an hour in a small discussion section. Your section leader will help review the material, explore some topics in more depth, and generally answer questions as appropriate. These section problems are designed to give you some extra practice with the course material. You're not expected to complete these problems before attending section and we won't be grading them for credit. Instead, think of them as a resource you can use to practice with the material as you see fit. You're not expected to cover all these problems in section, so feel free to look over the solutions to the problems you didn't complete.

Problem One: Returning and Printing

Topics: Function call and return, return types

Below is a series of four functions, each of which has a blank space where the return type should be. For each function, determine

- what the return type of the function should be,

- what value, if any, is returned, and

- what output, if any, will be produced if that function is called.

Then, determine whether there's anything unusual about any of these functions.

```
_____ printLyrics_v1() {                        _____ printLyrics_v2() {
    cout << "Havana ooh na na" << endl;             return "Havana ooh na na";
}                                               }


_____ printLyrics_v3() {                        _____ printLyrics_v4() {
    return "H";                                     return 'H';
}                                               }
```

Problem Two: Recursion Tracing

Topics: Recursion, strings, recursion tracing

In lecture, we wrote the following recursive function to reverse a string:

```
string reverseOf(string s) {
    if (s == "") {
        return "";
    } else {
        return reverseOf(s.substr(1)) + s[0];
    }
}
```

Trace through the execution of `reverseOf("stop")` along the lines of what we did in Wednesday's lecture, showing stack frames for each call that's made and how the final value gets computed.

Problem Three: Human Pyramids

Topics: Recursion

A **human pyramid** is a triangular stack of a bunch of people where each person (except the person at the top) supports their weight on the two people below them. A sample human pyramid is shown to the right. Your task is to write a function

<div align="center">

`int peopleInPyramidOfHeight(int n)`

</div>

that takes as input the height of the human pyramid (the number of layers; the pyramid to the right has height three) and returns the number of people in that pyramid. Your function should be completely recursive and should not involve any loops of any sort. As a hint, think about what happens if you take the bottom layer off of a human pyramid.

Once you've written your solution, trace through the execution of `peopleInPyramidOfHeight(3)` similarly to how we traced through `factorial(5)` in class, showing each function call and how values get returned.

As a note, there's a closed-form solution to this problem (you can directly compute how many people are in the pyramid just from the height through a simple formula). It's described in the solutions.

Problem Four: Random Shuffling

Topics: Recursion, strings, random numbers

How might the computer shuffle a deck of cards? This problem is a bit more complex than it might seem, and while it's easy to come up with algorithms that randomize the order of the cards, only a few algorithms will do so in a way that ends up generating a uniformly-random reordering of the cards.

One simple algorithm for shuffling a deck of cards is based on the following idea:

- Choose a random card from the deck and remove it.
- Shuffle the rest of the deck.
- Place the randomly-chosen card on top of the deck.

Assuming that we choose the card that we put on top uniformly at random from the deck, this ends up producing a random shuffle of the deck.

Write a function

<div align="center">

`string randomShuffle(string input)`

</div>

that accepts as input a string, then returns a random permutation of the elements of the string using the above algorithm. Your algorithm should be recursive and not use any loops (`for`, `while`, etc.).

The header file `"random.h"` includes a function

<div align="center">

`int randomInteger(int low, int high);`

</div>

that takes as input a pair of integers `low` and `high`, then returns an integer greater than or equal to `low` and less than or equal to `high`. Feel free to use that here.

Interesting note: this shuffling algorithm is a variant of the *Fisher-Yates Shuffle*. For more information on why it works correctly, take CS109!

Problem Five: Computing Statistics

Topics: Structures, file reading, loops

Imagine you have a file containing a list of real numbers, one per line (perhaps they're exam scores, or blood pressure numbers, etc.) Your task is to write a function

```
Statistics documentStatisticsFor(istream& input);
```

that takes as input a reference to an input stream pointed at the contents of a file, then returns a `Statistics` object (described below) containing statistics about that document. You can assume that the file is properly formatted and contains at least one number.

The `Statistics` type used here is a `struct` that's defined as follows:

```
struct Statistics {
    double min;     // Smallest value in the file
    double max;     // Largest value in the file
    double average; // Average value in the file
};
```

As a reminder, you can read a single value from a file by writing

```
double val;
input >> val;
```

and read all the values left in a file by writing

```
for (double val; input >> val; ) {
    // Do something with val
}
```

Problem Six: Haiku Detection

Topics: TokenScanner, procedural decomposition

A **haiku** is a three-line poem where the first line has five syllables, the second has seven syllables, and the final line has five syllables. For example, this poem is a haiku:

<div align="center">

An observation:
Haikus are concise, but they
Don't always say much.

</div>

This poem is not a haiku, because the last line has six syllables.

<div align="center">

One two three four five
Six seven eight nine ten, then
eleven, twelve, thirteen!

</div>

Your job is to write a program that reads three lines of text from the user, then checks whether those lines form a haiku. You can assume that you have access to a function

<div align="center">

`int syllablesIn(string word);`

</div>

which returns the total number of syllables in a word. You can assume that the input text consists solely of words, spaces, and punctuation marks (e.g. commas and exclamation points). You may want to use the TokenScanner type here. You can use TokenScanner to break a string apart into individual units as follows:

```cpp
#include "tokenscanner.h" // At the top of your program

TokenScanner scanner(str);
scanner.ignoreWhitespace();

while (scanner.hasMoreTokens()) {
    string token = scanner.nextToken();

    // ... Do something with token ...
}
```