Section 1. C++ fundamentals

Section materials curated by Neel Kishnani, drawing upon materials from previous quarters.

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.

Starter code

1) Returning and Printing

Topics: Function call and return, return types

Below is a series of four **`printLyrics_v#`** functions, each of which has a blank 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.

Is it appropriate for each of these functions to be named **`printLyrics`**? Why or why not?

```
_____ printLyrics_v1() {
    cout << "Havana ooh na na" << endl;
}
_____ printLyrics_v2() {
    return "Havana ooh na na";
}
_____ printLyrics_v3() {
    return "H";
}
_____ printLyrics_v4() {
    return 'H';
```

```
}
```

Solution

---

2) 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.

Solution

---

3) Testing and Debugging

Topics: Testing, loops, types, function call and return

Consider the following piece of code:

```
/* Watch out! This code contains many bugs! */
bool hasDoubledCharacter(string text) {
    for (int i = 0; i < text.size(); i++) {
        string current = text[i];
        string previous = text[i - 1];
        return current == previous;
    }
}
```

This code attempts to check whether a string contains at least two consecutive copies of the same character. Unfortunately, it has some errors in it.

- Identify and fix all the errors in this code.

We can write test cases to check our work and ensure that the code indeed works as expected. Imagine you're given the following provided test:

```
PROVIDED_TEST("Detects doubled characters") {
    EXPECT(hasDoubledCharacter("aa"));
    EXPECT(hasDoubledCharacter("bb"));
}
```

This test checks some cases, but leaves others unchecked. As a result, even if these tests pass, it might still be the case that the function is incorrect.

- Identify three types of strings not tested by the above test case. For each of those types of strings, write a `STUDENT_TEST` that covers that type of string.

Solution

---

4) 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 below.

Your task is to write a function

```
int peopleInPyramidOfHeight(int n);
```

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.

Solution

---

5) Random Shuffling

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

```
string randomShuffle(string input)
```

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

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

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!

Solution

---

6) 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
}
```

Solution

---

7) 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:

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

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

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

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

```
int syllablesIn(string word);
```

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:

```
#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
}
```

Note: we are not providing a starter file for this one; we encourage you to think about how you want to decompose it and discuss with your section leader and section-mates!