Submission

Sets and Maps

Introduction

In this lab we will be looking into how to store data read from text files into sets and maps, both of which can be thought of as using binary search trees in their implementations. We will be utilizing classes from the STL (Standard Template Library) such as fstream, set and map. The goal of this lab will be to become familiar with use cases of set and map, as well as their operations to provide a framework in which to understand the underlying data structures that make them tick.

Data

Today we'll be working with a real-world dataset. This dataset has 3,958 rows of data, each with 14 columns in the following format:

```
FIPS,Admin2,Province_State,Country_Region,Last_Update,Lat,Long_,Confirmed,Deaths,Recovered,Active,Combined_Key,Incidence_Rate,Case-F
```

To handle the commas, we can use an overloaded version of `std::getline()`. If you reference the manual page (http://www.cplusplus.com/reference/string/string/getline/) you'll see that there is a version of the function that accepts a delimiter. By passing an argument here, we specify how we want the string to be split up. Every call to `std::getline()` will read the data up to the next symbol set as the delimiter. You can place this in a loop to parse a delimited string. This would turn the code above into:

C++

```
#include <iostream>
#include <fstream>
#include <sstream>

int main() {
    std::string line;
    std::string entry;
    std::ifstream table;                // 1. Create instance
    table.open("file.csv");                 // 2. Open the file
    if(table.fail()){                        //    Check open
```

```
        cerr << "Can't open file\n";
        return 1;
    }
    while(std::getline(table, line)){           // 3. get a line of data
```

Sets

The reason for being called a "set" comes directly from set theory, a branch of mathematics, in which a set is a collection of distinct objects. In other words, each element that could be in the set is either in the set or not in the set, there is no "amount" associated with a given element. There are a myriad of operations that you can perform on a set. Look at the [set reference](http://en.cppreference.com/w/cpp/container/set) page to familiarize yourself with the basic operations.

C++

```
#include <set>
#include <string>

int main() {
    std::set<std::string> animals; // creating a set of strings
    myset.insert("cat");
    myset.insert("dog");
    myset.insert("horse");
    // ...
}
```

Maps

Maps are similar to sets, in that each contains a number of unique elements that are in order. The **key** difference between sets and maps in general is that sets store elements all by themselves, whereas maps store **key-value pairs**. A key-value pair is a pair of two elements where the first element, the key, is used to index the map, and the second element, the value, is what is stored/returned. Similar to the set reference page, there exists a [map reference](https://en.cppreference.com/w/cpp/container/map) page for you to explore as well.

Here is an example of maps in action:

C++

```
#include <iostream>
```

```cpp
#include <map>

int main() {
    std::map<std::string, int> mymap;

    mymap["dog"] = 7;
    mymap.emplace("cat", 4); // This is the same as mymap["cat"] = 4
    mymap.insert({{"fish", 11}}); // This is the same as mymap["fish"] =
    mymap["cat"]++;

    std::cout << mymap["dog"] << std::endl; // Prints 7
    std::cout << mymap["cat"] << std::endl; // Prints 5
    std::cout << mymap["fish"] << std::endl; // Prints 11
```

Note that when you use `emplace` or `insert` with a key that is *already in the map*, the value will **not** be replaced.

Your Task

You will be working with a database compiled and maintained by John Hopkins University containing information on cases of COVID-19 (The full database can be found here https://github.com/CSSEGISandData/COVID-19). The goal of this task is to read through the provided text file and construct some data structures that will allow efficient lookup operations on them. **Note:** This lab uses an older version of the dataset! While you may browse the version provided on the GitHub repo to examine the dataset in more detail, be sure to follow the formatting explanation given in this lab document.

Set

This is fairly straightforward, insert each country into a set.

Nested Map

Your nested map structure will look like this: {Country, {State, Cases}}, with the structures being read as {KEY, VALUE}.

So the KEY to the outer map is the Country name, and its VALUE is another Map.

The KEY to the inner map is the State, and its VALUE is the number of **confirmed** COVID-19 cases.

The Dataset

The dataset is comma-separated; the third column of data is the state/province where the COVID-19 cases are occurring. The fourth column of data is the country and the current confirmed cases of COVID-19 is located in the eighth column. Every row has a country name and the same number of commas, but not every row has a state/province.

Note: If a line of of the text file ****does not have a state/province, then use the country as the key****.

Code

You are given a sample of the database of confirmed COVID-19 cases, and some example commands. Your task is to write a program to the following specifications:

- Accept the arguments `<dataset> <command_list>`
- Read the `dataset` then create a `set` and a `map` to the specifications given above
- Read the command list file that handles the following commands:
    - `set <country>`
        - should print whether the country exists in the set (1 or 0)
    - `map <country>`
        - should print the # of confirmed cases in that country (-1 if the country is not in the map!)
    - `map <country>;<state/province>`
        - should print the # of confirmed cases in that state/province (-1 if the country or state/province is not in the map!)

Running your program with `confirmed_cases_sample.txt` and `commands_sample.txt` should yield the following output:

```
1
0
1
-1
26555
21827
7176
```

Notes

- You don't need all of the data in the dataset! Only store exactly what you need as you read through the file.
- Construct a simple map to make sure you understand it before jumping into the nested map.
- The `stringstream` library's `peek()` and `get()` functions can help you handle edge cases in the dataset. e.g. South Korea's entry.
- You only have a small subset of the full dataset: 20/3958 rows.
    - Extrapolate from the data samples you have to determine if there may be other edge cases to account for.