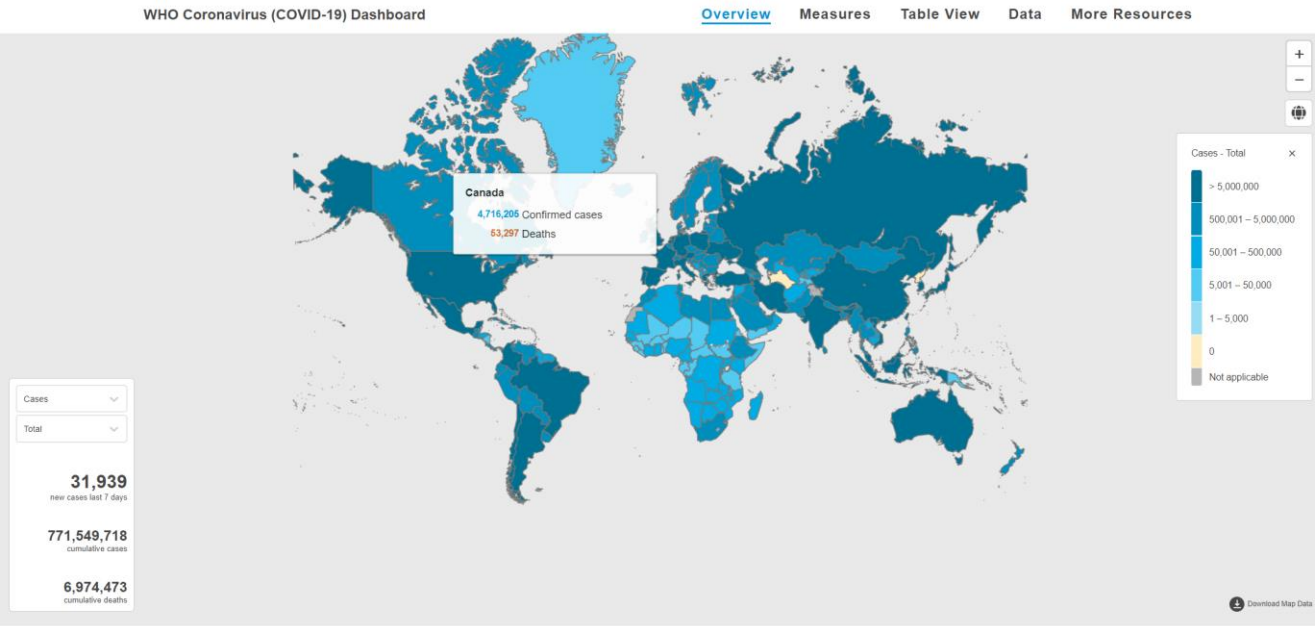
CPSC 2430 Data Structures

Winter Quarter 2024

Assignment 4 – Covid Data Tracker (A Hash Table, actually…)
Due: 10:00 pm, Feb 22, 2024

In Assignment 4, we will implement our own version of a Covid Data Tracker (actually, it is just a hash table…). This assignment will help you practice your STL (vector) and file operations in C++, hash table operations, and collision resolving.

The World Health Organization (WHO) is a special agency affiliated with the United Nations responsible for international public health. WHO Covid-19 Dashboard provides you a visualization of Covid cases and deaths worldwide (see below).



What we want to achieve for this assignment is to transfer this map to a hash table.

Design and implement a C++ class named CovidDB, which supports:

1. bool add(DataEntry* entry)
   add() function should insert one data entry into the hash table. Returns true if record is added, false if record is rejected (date < than current date)
3. DataEntry* get(string country)
   get() retrieves a data entry with the specific country name(string country). It returns nullptr if no such entry exists
3. void remove(string country)

This function removes the data entry with the specific country name (string country).

4. void displayTable()
   Displays the contents of your hash table.

A data entry contains the following information:

```
Class DataEntry{
    private:
        string date;
        string country;
        int c_cases;
        int c_deaths;
    public:
        // constructor, optional getters and setters
        …
 }
```

For example, a data entry

```
04/27/2022, Afghanistan, 178769, 7683
```

represents: as of 04/27/2022, the ***cumulative cases and deaths*** in Afghanistan are 178769 and 7683, respectively.

Your hash table size is ***m=17***. Your hash table must be implemented with STL vector, and use ***separate chaining*** as the technique to solve collisions in the hash table. Your hash table must hold pointers to DataEntry objects.

Your hash function is defined as:

```
int hash(string country) {
    int sum = 0;
    //pseudo code
    for each character c at position i in string country{
        // m is the hash table size, c is the ASCII code for char.
        sum = ∑((i+1) * c) % m;
    }
    return sum;
}
```

For example, hash(Japan) is calculated as:

```
hash(Japan) = (1*74 + 2*97 + 3*112 + 4*97 + 5*110) % 17
(ASCII : J: 74 a: 97 p:112 a: 97 n:110)
```

Recommended steps to complete assignment 4:

1. You should implement your hash table first. This will allow you to write a test function and enter fake data entries to test the add(), get(), and remove() functions to ensure your hash function and handling of collisions via separate chaining are working correctly.

2. Once your hast table implantation is fully working, then you can add the functionality to your program to open and read through the data file, *WHO-COVID-Data.csv*. Use the data entries there to build your ***initial hash table***.
   Please pay attention that this file shows you the new cases and deaths on a certain day. For example, line 326 in the table shown below indicates that on 11/23/2020, the new cases and new deaths in Afghanistan are 203 and 12, respectively.

326	11/23/2020	Afghanistan	203	12
327	11/24/2020	Afghanistan	282	8
328	11/25/2020	Afghanistan	290	17
329	11/26/2020	Afghanistan	212	13
330	11/27/2020	Afghanistan	226	12
331	11/28/2020	Afghanistan	123	3

   As you read and process each line of WHO-COVID-Data.csv, you need to calculate the ***cumulative cases and deaths*** by the current line date and insert/update each country entry in your hash table. For example, in *WHO-COVID-Data.csv*, line 1 has header information (you should skip this line). Line 2 – line 1036 shows daily COVID new cases and deaths in Afghanistan from 1/3/2020 to 11/02/2022. After processing all Afghanistan rows, the only entry in your hash table should be:

   `11/2/2022, Afghanistan, 203167, 7823`

   instead of all the daily data entries (203167, 7823 are cumulative cases and deaths, respectively). In other words, each country *only has one data entry* in your hash table.

3. After the creation of your initial hash table, you should revise your add() function to reflect any update made later to the hash table via the UI. Suppose after creating your initial hash table successfully using the *WHO-COVID-Data.csv*, the data entry for Afghanistan is

   `11/2/2022, Afghanistan, 203167, 7823`

   Now we want to insert another data entry where
   a. The country already exists in the hash table
   b. The date is ***later than*** the existing data entry's date.

   You should update your hash table accordingly. For example, if we add a data entry

   `11/03/2022, Afghanistan, 10, 3`

   The hash table should be updated with:

   `11/03/2022, Afghanistan, 203177, 7826`

   If the date is earlier than the date in the current hash table, your add() function must ignore the new entry and return false.

4. Provide a user interface to allow users to operate on your hash table ***continuously***. For example,

```
// Covid Tracker
// Please choose the operation you want:
// 1. Create the initial hash table
// 2. Add a new data entry
// 3. Get a data entry
// 4. Remove a data entry
// 5. Display hash table
```

`// 0. Quit the system`

When you choose 1, the hash table is created based on *WHO-COVID-Data.csv*. Operation 2, 3, 4, and 5 are based on the hash table you create in Operation 1.

Assignment 4 Submission

You need to submit your **code files** and a **Makefile** in a tar named:

- assignment4.tar

Use the tar command in CS1 server to create a tar that includes all of your files (code files, data files, and Makefile), for example:

```
tar -cvf assignment4.tar CovidDB.cpp CovidDB.h main.cpp Makefile WHO-COVID-Data.csv
```

Note: **the above is just an example.** You must split your code into multiple files, for example: CovidDB.cpp, CovidDB.h, main.cpp for better organization. Then make sure to include ALL your code, header files, Makefile and .csv in the tar (by adapting the example above to fit your files).

Your Makefile (a text file, without extension) should have all instructions needed by the make utility to build your code files into an executable. You must include the usual compilation flags: `-Wall -Werror -pedantic -std=c++11`.
Test that your Makefile works correctly by running "make" on the CS1 server.

Before submission, you should ensure your program has been compiled and tested (extensively).

You can submit your program multiple times before the deadline. The last submission will be used for grading. To submit your lab, execute the following script (assuming your files are on cs1.seattleu.edu):

```
/home/fac/mjilani/submit/24wq2430/assignment4_submit
```

The submission script will (1) look for assignment4.tar in the current directory and copy it over to the grading area, (2) untar the files, (3) run make to build your executable. If any of the steps fails, you will receive an error and your submission will fail. Make sure to test the submission days in advance of the deadline so you can seek help if needed. Your assignment receives zero if your code cannot be compiled and executed.

Assignment 4 Grading Breakdown

Breakdown	Points	Note
initial hash table	30	The initial hash table based on the csv file is correct
add	30	Add function could successfully: 1. Add a new data entry if the country in the data entry is missing from the existing hash table (10pts) 2. Update existing data entry if the country already exists in the hash table (10pts) 3. Ignore attempts to add a data entry with a date greater than what is already summarized in the hash table (10pts)
search	10	Return the data entry based on the country's name
remove	10	Remove the data entry based on the country's name
display	10	Display the contents of your hash table
user interface	10	Provide a user interface to interact with your hash table
coding	0	Deductions possible for lack of comments, lack of functional decomposition, use of global variables, etc.