Lab 9 – Maps

Overview

In this assignment you are going to use the map classes from the standard template library to store different types of data. For this assignment, main() is mostly implemented for you; you just need to fill in the blanks.

Description

Part 1

For the first part, you are going to store the results of some randomly generated numbers. A map is a great data structure for storing information with identifiers, but especially when encountering data that you don't know about before the program runs. This is where maps can really shine.

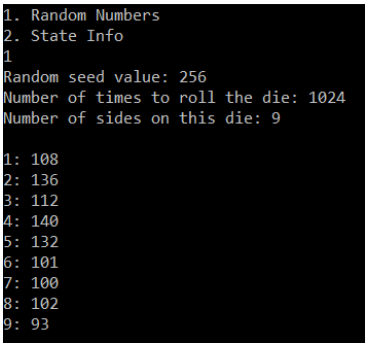
An effective method for generating random numbers is provided for you, but here's what it looks like:

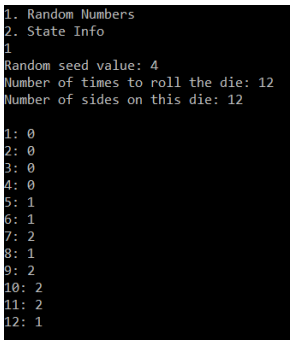
```
// An "engine" for generating random numbers
std::mt19937 random_mt;

// Using that engine, generate a random number between the given parameters
(inclusively)int Random(int min, int max)
{
   uniform_int_distribution<int> dist(min, max);
   return dist(random_mt);
}
```

You need to create a function that has the following signature:
void RollDice(int numberOfRolls, int numberOfSides);

In this function, you're going to create a `map<key,value>` object to store the results of these rolls. The number rolled is the key, and the number of times it is rolled is the value. Your output will look something like this:





With larger quantities of rolls, the distribution of each value should approach uniformity—that is, if you were to generate a random number within a certain range an infinite number of times, all numbers within that range would be generated equally. As the number of rolls shrinks, there may be a larger imbalance between some of the values, which is normal.

Since you will be recording the results that are rolled, how do you get a 0? This isn't something you would always do necessarily, but for this assignment you should "seed" the map object with a default value for all key/value pairs.

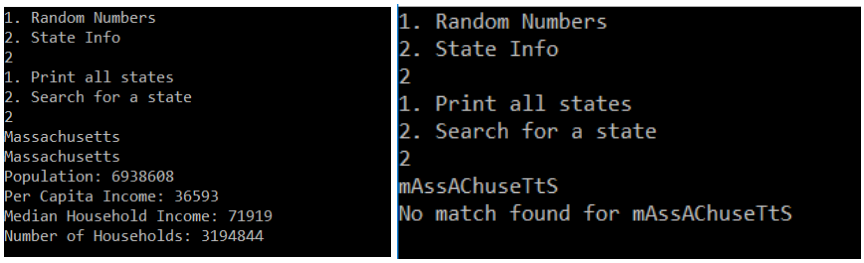
Part 2

For this part, you are going to store data in a slightly more complex format. You're going to load a file that has information about states in the US. The data will in a CSV file, and is structured like this:

State	Per capita income	Population	Median household income	Number of households
Mississippi	21036	2994079	39680	1095823
West Virginia	22714	1850326	41059	735375
Arkansas	22883	2966369	41262	1131288
Alabama	23606	4849377	42830	1841217
New Mexico	23683	2085572	44803	760916
Kentucky	23684	4413457	42958	1712094
Idaho	23938	1634464	47861	591587
Etc…				

You will need to create a class/structure to store one of these rows of data as an object, and ultimately store all of the states in a `map<string,WhateverYouCallYourClass>`.

After that, you will need to get input for one of two options: either print out all the key/values in the map object, or do a search for a particular key, and then print the key/value pair based on the search result. The search is case-sensitive (the first letter of the key is capitalized).



The key to going through STL containers is to use iterators. While some objects like vectors and strings store their data contiguously, making simple for loop iteration possible, that's not true of all containers. For that, you need an iterator. In C++ iterators commonly revolve around permutations of 2 functions: begin(), and end().

Want to start with the first element in a list? That's begin(). Want to reach the end of the list? That's just BEFORE end(). Before? Why not on end()? The end() function returns an iterator that is beyond the range of elements.

Think of it like this: if you were looping through an array of 10 elements, the valid indices would be 0-9. The "end" index would be 10, or 1 past the last element. With iterators, end() functions the same way. In both cases, you should never try to USE that "just past the end" element, but it can be helpful to check against that.