CSCE 221 Cover Page
Programming Assignment #6
Due **April 29** by midnight to eCampus

First Name: Asa Last Name: Hayes UIN: 525003952

User Name: AsaHayes E-mail address: asahayes@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero. According to the University Regulations, Section 42, scholastic dishonesty are including: acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion read more: Aggie Honor System Office

Type of sources			
People			
Web pages (provide URL)			
Printed material			
Other Sources			

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Your Name Asa R Hayes Date 29 April 2018

Programming Assignment #6

Due ***April 29*** to submit to eCampus

Part I
Bracket Operator

The function of the bracket operators essentially serves as an access point for our map. We've seen brackets implemented as default when working with arrays and vectors previously, and they work the same here: inside the brackets is the index in the array that you can access the data value of. In previous cases, it has been much easier to deal with, as the index has just been integers, which are easy to deal with, but in this case, the index is a word, and you can't just count up words to get to the value that you need to. Overwriting our own bracket operator allows us to access each value by the word it is associated with, aided by the iterator which helps in searching. The way I have implemented it is by doing at maximum two searches and one insertion per element. The first is to determine if the key already exists in the table, and the second is after an insertion of a new key-value pair that allows us to return a reference to that value. Because of this, each use of the bracket operator is O(n). For improvement of the running time, there are many things possible, but that I did not have time to invest into. The largest improvement would be using a binary search tree as the basis for the map, as that would immediately improve search and access times. Additionally, there is most likely a way to do an insertion without two full traversals of the map.

Part II
Different Implementations of Map

0.1 A vector of key_values:

Initially, the most obvious advantage that a vector map has over a tree map is that it is easier to implement than binary trees. The traversal operations are simpler, and the iterator barely needs any special instructions to work properly. It also works a bit more intuitively, as the map is still essentially just a regular array/vector. However, the linear way it has to search means that the time is not going to be very good even in the best cases.

0.2 A tree of key_values.

Regular binary tree maps don't have a lot of advantage overall against a vector map, as a binary tree with no balancing can approach very bad runtimes as well. Red-black and 2-4 trees are where binary tree maps become directly superior for most uses. While AVL trees have better search times, there is a larger cost for insertion, of which there usually is a lot for any given map usage. Red-black and 2-4 trees are good all around due to their balancing, and keeping a consistent structure gives possibility for more optimized design due to less need to account for bad pathing.

0.3 A Hash Table of key_values

Depending on implementation, a hash table may be the fastest of the three options, as traversal takes much less time. The largest limited would be on size, as the table would be crowded very quickly, complicating the retrieval of any element and requiring a very well designed hash algorithm to try and avoid as much collision as possible. In a way, a map acts as a hash table already, as it based on key-value pairs.

1 Real Applications for Maps

One large us of maps in the real world that I have seen is their use in encryption. We see the effects of it everyday, as compression of resources until use saves a great deal of loading time or drive space for all users. One of the more basic and widely used encryption methods is simple substitution, where a smaller data object can be used as a placeholder for larger more complex ones. As long as you have the reversal method to translate back into the original sized values from the keys, there is very little negative consequence on using a map for this purpose. This general concept of compression is used widely for files, communications (all the way back to Morse Code), and even in actual maps, which you can see any time you change height in Google Maps when the quality changes. In general, maps are used when you need to store values of any type with a more manageable or specific means of organization than just a numerical ordering.