Object Oriented Assignment Brief

# Functions

Insert – This would be the way to populate the map with data.

OutputData – This function is here for testing, outputting the entire map is useful to show that certain elements have been removed and that the length of the array is correct.

Clear – this function is needed because it allows the user to clear the whole map of all elements. This is useful in testing as well to show that you can still add elements after it gets cleared.

Erase – this allows for the erasure of a single element from the map. This would be useful if the user accidentally adds the wrong key or element.

Empty – this checks whether the map is empty or not. This would be useful if the user would want a function to only happen if the map had data in it.

Size – this outputs how many elements are in the map currently. This would be useful if the user wanted to have a loop run for the amount of times as there are elements

MaxSize – this outputs what the maps current max size is. This would be useful if the user wanted to, for example, completely fill the map with data using a for loop that ends the inserting of data when the map is full.

Find – this searches for a specific value given the key. This would be useful if the user has a key but needs the specific value assigned to it.

# Support for Abstract Data types

C++ gives a lot of support for having abstract data types. In my implementation, one of the most used forms of support is the typename keyword. This keyword allows the passing of any type in functions which is exactly what is needed in a map class. Combining these with templates allowed me to have functions that could take any data type as long as the data inserted after, matched the same type as the data inserted initially into the template.

# Weaknesses in my Implementation

The main weaknesses in my implementation are that there isn’t support for swapping the data that is already in the table. If a user wanted to be able to swap the data they would have to know the key to that specific data, erase it then insert the key and the data that they wanted. This instead could be done in a replace function that overrides a value given a certain key. Another thing my implementation doesn’t do is allows the user to copy the whole map into something else such as another map. This is because I haven’t overloaded the = function to allow the user to make something equal to the map. Another weakness is that I only have basic iterator functions such as getting the beginning and the end of the map. I am missing const iterators and const reverse iterators which means the user cannot have their start be a constant. The emplace function that allows elements to be constructed then inserted is also missing, this means the user will always have to have data already created to insert into the map. Finally, my map class is also missing the count function meaning that the user can’t find everything that has a specific key. This means that the user cannot find all keys that have their initially sent key in them. Another improvement my map class may need would be the ability to have multiple keys that are the same. This would be impossible to add to my map class currently because a unique id would have to be given to all keys so that having the same key is allowed and could be differentiated when inserting or erasing.

# Dynamic Array Size

The array is made dynamic in my implementation by starting off the array as being quite small then when an insert is made that makes the array go over its limit, the array is copied into a temporary array that has the max length of the array set to double the length of the array after the insert, deleted, then the main array is then remade with the new max length and the data is copied into it. This is a lot less efficient than just using a vector that is made for having a dynamic size and dynamically adding data, on top of this it is also extremely memory unsafe unlike just using vectors.

# Comparison to the STL map class

In comparison to the STL map class my map class is missing a lot of features. Examples of these are: constant iterators and constant reverse iterators that have been mentioned in the previous paragraph, some of the at function such as checking against the bounds greater than and less than, the swap function which again has been mentioned in the paragraph above, the emplace function that would allow the user to construct an element and then insert it instead of having to have the elements already created, the emplace\_hint function that does something very similar to emplace but can be input in a certain position, a count function to see which keys are included in the initial key given, the ability to return an iterator to the upper or lower bounds and finally the ability to get every element in a container equal to the key given.

Another difference is that my map uses arrays instead of vectors which are massively less efficient when making them dynamic. Because the map constantly has things inserted and deleted the initial cost in memory a vector has when it is initialised is made worth it over an array as it would take more memory to do the insertions and deletions in an array. The use of vectors would also allow the taking out of deleting of the whole array when one is cleared. This could be made to cycle through and delete everything but this takes a lot more memory where as the other method of deleting the array is quicker but also not memory safe with a risk of memory leaks. Overall it would just be better to use vectors over arrays because the whole problem of having the list be dynamic is what vectors are made for.