## Map class

The map class uses templating so that any data type can be used. The data type requires the < overload to sort the data. Most fundamental types already have the < overload already, while custom classes will require the developer to overload it. The currently tested classes that work are; int, float, string, struct and class.

Data is stored dynamically with an array pointer to a struct for each key value pair. The map class also stores, the current array size, the amount of currently stored data and whether the data types are trivially copiable. The data is stored in struct to help with maintenance and readability.

The default size of the array is 10 and once that limit has been reached the array doubles in size, this then expands the array exponentially, so that as the data size increases there are less resizes which on large arrays can take a while, and small arrays could potentially require more resizes than if it expanded linearly (dependant on the size of linear expansion) e.g. 1000 data, doubling resize 10, 20, 40, 80, 160, 320, 640, 1280. Only 7 resizes, while a linear at a buffer of 10 would take 100 resizes, even a buffer of 100 would take 10 resizes, and this buffer means something that is only 10 big would require a 100-size map.

A iterator was added to the map, giving the ability to interate over the whole map, useful functions where given to help with; Begin, End and overloads; ++, --, \*, ->, ==, !=.

A copy constructor and a = overload has been implemented so that map classes can be copied easily increase you want a copy of the data, so you can alter one but keep an original as well.

Insert added data to the array and returns a boolean on success. First of it check whether the data is in the array if so, it returns false, then checks with the array is big enough if its too small the array will increase in size. The function will then move the data and insert it in order, this makes it easier to the iterator and searching.

Erase removes data from the function and returns a Boolean on success. First of it checks if the key is in the array, if it is not then it returns false. If it exists the data is copied down the array, and the current data value is decreased. This means the is a duplicate value at the end, but the current data value stops it before any duplicate data.

For developers that know the size of the array they need, SetSize has been implemented, there is also a ForceSetSize, which will bypass checks for the array size, but this can cause data loss, the array is ordered, and resizing should take the new array size from the start, but this is untested and in not its intentional use. If you want to resize the data firstly use the Erase to remove the data that is not needed, now use SetSize to make the map the correct size.

Clear is a function that delete the array and sets everything back to default values, so array size is 10 and current = 0.

Find is the function that returns a value given a key. The function searches the array for the value, and returns a pointer to the value, if it can not be found it returned a nullptr.

FindInArray is a private function that is used to find if a key value pair is stored in the array and what position it is in. This is used but several functions including Insert, Erase and Find. The function uses binary search to go through the array to see if the value exists and return a Boolean if it is found and the position it ended on in case of an insert. A binary search has a time complexity of O(log n) while a linear search is O(n) so on average a binary search is faster.

## Recommendations

The current implementation is designed for readability and maintenance, this design may cause an increase in data size required and small performance loss.

If data size is a concern, the struct can be removed and use two separated arrays for key and value. This is due to struct may make a container for the data bigger that total value of the variables inside it.

If performance is a concern then all the data values can be stored as individual pointers which then could be stored in an array, this give the ability to swap pointer rather than copy data, this will make only insert and erase faster, all other functions will be the same performance.

My map class is set up in a similar way to SLT. SLT’s version does contain more functions for ease of use and constant version for safety, but my version can do the majority if not all of what STL::map can do. My map uses dynamic memory in to form of pointer to struct array while the SLT uses red-black tree.

## Further development

If this is to be further developed, reverse iterator should be implemented for developers so that is it more visual that a developer is iterating backwards, while this can be done with an iterator it may not be clear for maintenance.

Constant variant of the iterator could be implemented to help developers program defensively so that data that should not be changed can’t be changed.