Maps

Sometimes we need to store certain elements(keys) along with a certain value associated with each element. While the first idea that comes to mind would be an array , it is unideal as the array must contain at least as many elements as the largest key. (a[2048] would require an array size of at least 2048 in spite of only one element being stored). Additionally it requires the key element to be an integer value. Furthermore it suffers from the same issue of duplication that sets do. Maps function similar to sets , but unlike sets which contain only a 'key' , they contain a 'key' and a 'value'. They are sorted with respect to the value of keys.

Declaring Sets:

Similar to sets , maps have only one type of constructor: *map<T1,T2> st;*

The above constructor creates an empty map , with key of type T1 and values of type T2.

Inserting elements into a Map:

The *insert()* method of a set takes the key and value to be inserted as a parameter and adds it to the map. Insertion rearranges the previous elements in the map so that a sorted order is maintained. Similar to sets , the *insert()* method takes O(log(n)). If the key is already present, the iterator to that key is returned.

Complexity : O(log n)

Accessing/Traversing a Map:

Similar to sets, one can use iterators to access the values in a set. *st.begin()* returns an iterator to the first element. **st.begin()* will de-reference the iterator to get the first element of the set, and a similar statement holds for *st.end()*. **Note:** Also use of **(st.begin() + i)* to access the i^{th} element is also not possible. Also a map stores the elements internally as a pair , so it->first resolves the key and it->second resolves the value.

All elements of a set can be traversed by using a regular container traversal method:

```
for(map<T1,T2> :: iterator it = mp.begin(); it != mp.end(); it++)
        cout << it->first;
```

Similar to vectors, *set<T> :: iterator* can be replaced by *auto*,
for(auto it = mp.begin(); it != mp.end(); it++)

```
        cout << it->first;

for(auto it :
mp)
        cout << it->first;
```

Finding an element in a Map:

Maps also provide ability to find a key in O(log n) and thereby its corresponding value.The *find()* method takes the key as an argument and returns an iterator to the element at which the key is found. If the key is not present in the map , it returns an iterator to mp.end().

Ex: For map<char,int> mp , mp.find('b') returns an iterator to the first occurrence of the key 'b' , if it exists , or mp.end otherwise.

Additionally if only the value is needed , one may use the *at()* method to return the value associated with the key passed as parameter. This works similarly to the *at()* method in vectors.

Ex: For the map in the example above , mp.at('b') would return a reference to the value of the given key.

Complexity : O(log n)

Updating value of an element in a Map:

A common use of maps it to count the number of times a key occurs in some data. To this end , we need to update the value associated with the key. This can be accomplished with the *at()* method mentioned above. Since the *at()* method returns a reference to the value , it may also be used to set the value.

Ex: For the map in the previous example , mp.at('b')=4 would set the value associated with the key 'b' to 4.

Deleting elements from a Map:

The *erase()* method can be used to delete an element from the map. An iterator or **key** of the element to be deleted may be specified as the parameter. Ex: *mp.erase(mp.begin())* or *mp.erase(8);*

Complexity : O(1) for deletion by iterator , O(log n) for deletion by key.

More methods related to maps can be found at:
https://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl