Maps and Unordered Maps

Matt Warner

1 Maps

Maps can be used by adding the map header file to your program:

```
#include <map>
```

Maps are **associative containers** that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have the same key value.

Some basic functions associate with std::map are:

- begin() - Returns an iterator to the first element in the map.
- end() - Returns an iterator to the theoretical element that follows the last element in the map
- size() - Returns the number of elements in the map.
- max_size() - Returns the maximum number of elements that the map can hold
- empty() - Returns whether the map is empty.
- pair insert(keyvalue, mapvalue) - Adds a new element to the map.
- erase(iterator position) - Removes the elements at the position pointed by the iterator
- erase(const g) - Removes the key-value 'g' from the map.
- clear() - Removes all the elements from the map.

Examples of std::map

The following examples shows how to perform basic operations on map containers

Example 1: using .begin() and .end()

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main()
{
  map<string, int> mp;    // Creates a map of strings to integers

  // Insert some values into the map
  mp["one"] = 1;
  mp["two"] = 2;
  mp["three"] = 3;

  // Get an iterator pointing to the first element in the map
  map<string, int>::iterator it = mp.begin();

  // Iterate through the map and print the elements
  while (it != mp.end())
  {
    cout << "Key: " << it->first
         << ", Value: " << it->second endl;
    it++;
```

```
  }
  return 0;
}
```

Output:

```
Key: one, Value: 1

Key: three, Value: 3

Key: two, Value: 2
```

Note:-

Maps are implemented as a balanced binary tree, and it automatically sorts its elements based on the key.

The sorting is done in ascending order according to the key's value.

Example 2: Using size() function

```
int main(}
{
  map<string,int> map;

  map["one"] = 1;
  map["two"] = 2;
  map["three"] = 3;

  cout << "Size of map: " << map.size() << endl;

  return 0;
}
```

Output:

```
Size of map: 3
```

Example 3: inserting elements

```
  std::map<int,int> mp;

  mp.insert(pair<int,int>(1,40));  // First method for inserting elements

  mp[1] = 3;      // Second method
```

We can iterate through an unordered_map as such.

```
for (const auto& pair : mp)
;
```

1.1 Example:

Lets say that we iterated through an array of integers and created a hash map to hold their frequencys

```
for (const auto& i : array) ++mp[i];
```

Now, lets say we wanted to find the element that appears the most amount of times in the array.
We can do so by setting up two variables. One to hold the value (that is the highest count)
and one to hold the key (that is the element we are searching for)

Now, just like linear search, we can iterate through the pairs in the hash map, and find the largest value, while also storing its key in our targetelement variable.

```
int largestFrequency = 0; // starting point for comparison
int targetelement; // will end up holding our target element

for (const auto& pair : mp) {
  if (pair.second > largestFrequency)
  {
    largestFrequency = pair.second;
    targetelement = pair.first
  }
  std::cout << targetelement;
}
```

if we wanted to build on this and find the next most frequent element, we can use the `.erase(key)` method to remove it from the map and continue.

That would look something like this:

```
int i = 0; // loop counter
int k = 3; // k most frequent elements
std::vector<int> solution; // vector to hold the k most frequent elements
While (i < k) {
  int largestFrequency = 0;
  int targetElement;
  for (const auto& pair : mp) // iterate through the key,value pairs in mp
  {
    if (pair.second > largestFrequency)
    {
      largestFrequency = pair.second; // update largestFrequency
      targetElement = pair.first // update targetElement
    }
  }
  solution.push_back(targetElement); // append element to solution vector
  mp.erase(targetElement); // remove from hash map
  i++; // increment loop count

}
```