Lesson:

Map One Shot 2

Pre-Requisites

- Hashing
- Hashing Functions
- Map in C++ STL

List of Concepts Involved

- Maps in C++ STL
- MCQs
- Problem
- Unordered Map
- Multimap
- Unordered Multimap
- MCQs

Topic: Map in C++ STL

Header File Required

To create a map in C++ we require the map header file. We can include it in the following way:

```
#include<map>
```

Also, we can use `bits/stdc++.h` as it includes every standard library. But it in itself is a non-standard header file of the GNU C++ library. It doesn't include only `map` but also other libraries thus it increases the compilation time. We can include that in the following way:

```
#include<bits/stdc++.h>
```

Declaring a Map in C++

A map in C++ can be created by the following syntax:

In ascending order (by default):

```
map<data_type1, data_type2>map_name;
```

In descending order:

```
map<data_type1, data_type2, greater<data_type1>>map_name;
```

Here `data_type1` represents the data type of the key and `data_type2` represents the data type of value. They can be same or different.

For example:

```
map<int, int>m1;
map<int, string>m2;
map<string, string>m3;
```

Initializing a Map

You can also create and initialize a map in the following way:

```
unordered_map<key_datatype, value_datatype>name = {{key1, value1},
                                                   {key2, value2},
                                                   {key3, value3}};
```

For example:

```
map<int, int>m4 = {{1,11}, {2,22}, {3,33}};
map<int, string>m5 = {{1, "Physics"},
                      {2, "College"},
                       {3, "Wallah"}};
map<string, string>m6 = {{"A)", "College"}, {"B)", "Wallah"}};
```

Inserting into a Map

Let us create a map and then insert key-value pairs into it.

```
map<char, string>m;
m['A'] = "Apple";
m['B'] = "Ball";
m['C'] = "College Wallah";
```

Printing a Map

We can print the pairs of the unordered_map using a for each loop.

Say we have to print the unordered_map we created above.

```
for(auto i : m)
   cout<<i.first<<" - "<<i.second<<endl;
```

Here, `i.first` prints the key and `i.second` prints the value of the pair in the map.

The **output** will be:

```
A - Apple
B - Ball
C - College Wallah
```

Member Functions

Say there is an unordered_map<int, int>m;

We can perform the following functions on it:

Modifiers

Functions	Explanation	Example	Time Complexity
insert()	Inserts element	`m.insert({1, 2});`	If a single element is inserted, then O(logN). If n elements are inserted then n*O(N+n), where N stands for the number of elements in the map (size)
insert_or_assign()	Works the same as insert() but if the key is already present then it assigns the new value to the key.	`m.insert_or_assign(1,11);`	O(LogN), where N is the size of the container, map. If the key is already present, then amortized constant time complexity.

erase()	Erases elements by
	i) Iterator
	ii) Key
	iii) Range
		`i) m.erase(m.find(1));`
		`ii) m.erase(2);`
		`iii)m.erase(m.begin(),m.end());`
			i) Amortized Constant
			ii) O(logN) (N: size of map)
			iii) Linear in distance between the range

swap()	Swaps 2 maps with the same type, size may differ.	`m1.swap(m2);` `swap(m1, m2);`	Constant
clear()	Removes all the elements	`m.clear();`	O(N), linear is size of map

Capacity

Functions	Explanation	Example	Time Complexity
empty()	Returns 1 if the unordered_map is empty else 0	`m.empty();`	Constant, O(1)
size()	Returns the size	`m.size();`	Constant, O(1)
max_size()	Returns the maximum size (a really large number)	`m.max_size();`	Constant, O(1)

Operations

Functions	Explanation	Example	Time Complexity
find()	Returns the iterator to the element	`m.find();`	O(logN)
count()	Returns the no. of occurrences of the key passed and since the map has only unique keys so it returns 1 if present else 0	`m.count();`	O(logN)
upper_bound()	Returns the iterator to the next greater element	`m.upper_bound(4)`	O(logN)
lower_bound()	Returns the iterator to the element (if present) otherwise the next greater element	`m.lower_bound(4)`	O(logN), where N stands for the size of map
equal_range()	Returns the bounds of a range that includes all the elements in the container which have a key equivalent to k.	`m.equal_range(k)`	O(logN)

Element access

Functions	Explanation	Example	Time Complexity
Operator [ ]	Accesses the value by key The disadvantage of the [ ] operator is, it creates an entry if not present with the default value, which increases memory usage and can lead to bugs if the default value can be one of the actual values.	`map[3] = 4;`	O(logN)
at()	Accesses the value by key	`m.at(2);`	O(logN)

Iterators

Functions	Explanation	Example	Time Complexity
begin()	Returns an iterator to the first element in the map	`m.begin();`	Constant
end()	Returns an iterator to the theoretical element after the last element	`m.end();`	Constant
rbegin()	Returns a reverse iterator pointing to the last element of the map	`m.rbegin();`	Constant
rend()	Returns a reverse iterator pointing to the theoretical element just before the first element in the map	`m.rend();`	Constant

Note:

```
map<string, int> m = {{"physics", 1}};
if(m["wallah]) {
cout<<"wallah is present in map"<<endl;
}

cout<<m.size()<<endl; // it should return 1 as we only inserted "physics" as a key but it will return 2 as "wallah" is also inserted by mistake during the lookup.
```

Topic: MCQ Questions

1. What will be the output of the following code:

```
map<int,int>m;
m.insert({1, 10});
m.insert({1, 20});
m.insert({3, 30});
m[4] = 40;

for(auto i : m)
    cout<<i.first<<" - "<<i.second<<"\n";
```

a) 1 - 10
3 - 30
4 - 40

b). 1 - 10
1 - 20
3 - 30
4 - 40

c). 1 - 20
3 - 30
4 - 40

d). 1 - 20
3 - 30

Answer: a

Explanation: The map stores only unique keys. So, a single key 1 cannot have two values 10 and 20. Once a pair of {1,10} is created, it can only be modified by m[1] = 20. But we if we try to insert a new key-value pair with the key already present inside the map then nothing will happen.

Also we can modify and insert elements in a map by [ ] operator. Hence, {4,40} pair is added to the map, making a) as the correct option.

2) What will be the output of the following code:

```
map<int,char>m = {{1,'a'}, {2,'b'}, {3,'c'}, {4,'d'}};
m.erase(m.find(3), m.end());

for(auto i : m)
    cout<<i.first<<" - "<<i.second<<"\n";
```

a) 1 - a
2 - b
4 - d

b) 1 - a
2 - b

c) 1 - a
2 - b
3 - c
4 - d

d) Compilation Error

Answer: b

Explanation: It is possible to remove elements from a map within a range. Here, will return an iterator pointing at key 3 and will be pointing at the theoretical element after the last element. So, all the pairs from key till end will be removed from the map. Hence, pairs with only 1, 2 keys will be left. So, b) is the correct option.

Problem: Sum of Repetitive Elements

You are given an integer n, representing the number of elements. Then, you will be given n elements. You have to return the sum of repetitive elements i.e. the elements that appear more than one time.

Example 1:

Input: n = 7

Elements = [1, 1, 2, 1, 3, 3, 3]

Output: 4

Explanation: The repetitive elements are 1, 3 and the sum is 4.

Example 2:

Input: n = 6

Elements = [1, 1, 1, 1, 1, 2]

Output: 1

Explanation: Here, 1 is the only element that is repeating. So, the sum is 1.

Example 3:

Input:

n = 5

Elements = [1, 2, 3, 4, 5]

Output: 0

Explanation: No element is repeating here, so the sum is 0.

Solution Code: https://pastebin.com/y2knju5m

Code Explanation: Here, we are creating a map in which key stores the element of the array and value stores the frequency of each element. Now, if the frequency that is the value of any key is more than 1, then it means that it is a repeating element. So, we add that key to the sum which was initially initialized to 0. When we have traversed the entire map, we have added all the repetitive keys to our variable sum and hence we return it.

Time complexity: O(N), since we are traversing the list of elements once and adding them to the map container. Also, we are traversing the map once to check for the count of each element whether it is repetitive or not, hence linear time complexity.

Space complexity: O(N), we are creating a map and storing the count of each element in it. Since extra space is used, hence linear space complexity.

Output:

```
Enter the number of elements: 7
Enter the elements of the array: 1 1 2 1 3 3 3
Sum of Repetitive elements: 4
```

```
Enter the number of elements: 6
Enter the elements of the array: 1 1 1 1 1 2
Sum of Repetitive elements: 1
```

```
Enter the number of elements: 5
Enter the elements of the array: 1 2 3 4 5
Sum of Repetitive elements: 0
```

Topic: Unordered Map in C++

We have studied what a map is. We also know that map stores elements in either ascending or descending order. But if we look at the unordered map, as the name suggests it is unordered i.e. data will be stored in the form of key-value pair only, as in a map except for the fact that it will be in random order and not ascending or descending order.

Definition

unordered_map is an associated container in C++ STL that stores key-value pairs. Internally unordered_map is implemented using Hash Table. There isn't a specific order of storing keys in it, they are in random order only.

Header File Required

To create an unordered_map in C++ we require the `unordered_map` header file. We can include it in the following way:

```
#include<unordered_map>
```

Also, we can use `bits/stdc++.h` as it includes every standard library. But it in itself is a non-standard header file of the GNU C++ library. It doesn't include only `unordered_map` but also other libraries thus it increases the compilation time. We can include that in the following way:

```
#include<bits/stdc++.h>
```

Declaring an unordered_multimap in C++

An unordered_map in C++ can be created by the following syntax:

```
unordered_map<data_type1, data_type2>name;
```

Here data_type1 represents the data type of the key and data_type2 represents the data type of value. They can be the same or different.

For example:

```
unordered_multimap<int, int>m1;
unordered_multimap<int, string>m2;
unordered_multimap<string, string>m3;
```

Initializing a Map

You can also create and initialize a map in the following way:

```
unordered_map<key_datatype, value_datatype>name = {{key1, value1},
                                                   {key2, value2},
                                                   {key3, value3}};
```

For example:

```
unordered_map<int, int>m4 = {{1,11}, {2,22}, {3,33}};
unordered_map<int, string>m5 = {{1, "Physics"},
                                {2, "College"},
                                {3, "Wallah"}};
unordered_map<string, string>m6 = {{"A)", "College"},
                                   {"B)", "Wallah"}};
```

Inserting into an unordered_map

Let us create a map and then insert key-value pairs into it.

```
    unordered_map<char, string>m;
    m['A'] = "Apple";
    m['B'] = "Ball";
    m['C'] = "College Wallah";
```

Printing an unordered_map

We can print the pairs of the unordered_map using a for each loop.

Say we have to print the unordered_map we created above.

```
    for(auto i : m)
        cout<<i.first<<" - "<<i.second<<endl;
```

Here, `i.first` prints the key and `i.second` prints the value of the pair in the map.

The **output** will be:

```
C College Wallah
B Ball
A Apple
```

Notice, how no order is there, it is random.

Member Functions

Say there is an unordered_map<int, int>m;

We can perform the following functions on it:

Modifiers

Functions	Explanation	Example	Time Complexity
insert()	Inserts element	`m.insert({1, 2});`	If a single element is inserted: Average - O(1) Worst - O(N) Multiple elements inserted: Average: O(n) Worst - O(n*(N+1)) Where n elements are inserted and N stands for the number of elements in the unordered_map (size)
insert_or_assign()	Works the same as insert() but if the key is already present then it assigns the new value to the key.	`m.insert_or_assign(1,11);`	Average: Constant, O(1) Worst: Linear in container size, i.e. O(N)

erase()	Erases elements by
	
	i) Iterator
	ii) Key
	iii) Range
		`i) m.erase(m.find(1));`
		`ii) m.erase(2);`
		`iii)m.erase(m.begin(),m.end());`
			Average: Linear in the number of elements removed i.e. O(n), n is no. of elements removed
			
			Worst: O(N) where N stands for the size of unordered_map

swap()	Swaps two unordered_map with the same type, size may differ.	`m1.swap(m2);` `swap(m1, m2);`	Constant
clear()	Removes all the elements	`m.clear();`	O(N), linear is size of unordered_map

Capacity

Functions	Explanation	Example	Time Complexity
empty()	Returns 1 if the unordered_map is empty else 0	`m.empty();`	Constant, O(1)
size()	Returns the size	`m.size();`	Constant, O(1)
max_size()	Returns the maximum size (a really large number)	`m.max_size();`	Constant, O(1)

Operations

Functions	Explanation	Example	Time Complexity
find()	Returns the iterator to the element	`m.find();`	Average: O(1), constant Worst: O(N), N is size of container
count()	Returns the no. of occurrences of the key passed and since the unordered_map has only unique keys so it returns 1 if present else 0	`m.count();`	Average: O(n), where n is no. of elements counted Worst: O(N), N is the size of container
equal_range()	Returns the bounds of a range that includes all the elements in the container which have a key equivalent to k.	`m.equal_range(k)`	Average: Constant Worst: O(N)

Element access

Functions	Explanation	Example	Time Complexity
Operator [ ]	Accesses the value by key	`m[3] = 4;`	Average: O(1), constant Worst: O(N), N is size of container
at()	Accesses the value by key	`m.at(2);`	Average: O(1), constant Worst: O(N), N is size of container

Iterators

Functions	Explanation	Example	Time Complexity
begin()	Returns an iterator to the first element in the unordered_map	`m.begin();`	Constant
end()	Returns an iterator to the theoretical element after the last element	`m.end();`	Constant

Topic: Multimap in C++ STL

Multimap is similar to map except the fact that we can store repetitive keys in it. Not just keys, repetitive key-value pair is stored in a multimap. Also, all these pairs are stored in either ascending or descending order.

Header File Required

To create a multimap in C++ we require the `map` header file. We can include it in the following way:

```
#include<map>
```

Also, we can use `bits/stdc++.h` as it includes every standard library. But it in itself is a non-standard header file of the GNU C++ library. It doesn't include only `map` but also other libraries thus it increases the compilation time. We can include that in the following way:

```
#include<bits/stdc++.h>
```

Declaring a multimap in C++

A map in C++ can be created by the following syntax:

In ascending order (by default):

```
multimap<data_type1, data_type2>map_name;
```

In descending order:

```
multimap<data_type1, data_type2, greater<data_type1>>map_name;
```

Here `data_type1` represents the data type of the key and `data_type2` represents the data type of value. They can be same or different.

For example:

```
multimap<int, int>m1;
multimap<int, string>m2;
multimap<string, string>m3;
```

Initializing a multimap

You can also create and initialize a map in the following way:

```
multimap<key_data_type, value_data_type>map_name = {{key1, value1},
                                                    {key2, value2},
                                                    {key3, value3}};
```

For example:

```
multimap<int, int>m4 = {{1,11}, {2,22}, {3,33}, {3,33}, {2,55}};
multimap<int, string>m5 = {{1, "Physics"},
                           {2, "College"},
                           {3, "Wallah"}};
multimap<string, string>m6 = {{"A)", "College"},
                              {"B)", "Wallah"}};
```

Inserting into a multimap

Let us create a multimap and then insert key-value pairs into it.

```
multimap<char, string>m;
m.insert({'A', "Apple"});
m.insert({'B', "Ball"});
m.insert({'C', "College Wallah"});
m.insert({'C', "College"});
m.insert({'C', "Cat"});
```

Printing a multimap

We can print the pairs of the multimap using a for each loop.

Say we have to print the multimap we created above.

```
for(auto i : m)
        cout<<i.first<<" - "<<i.second<<endl;
```

Here, `i.first` prints the key and `i.second` prints the value of the pair in the map.

The **output** will be:

```
A Apple
B Ball
C College Wallah
C College
C Cat
```

Notice how repetitive keys are stored in the multimap in ascending order (by default).

Member Functions

Say we have a multimap<int, int>m;

Now we can perform the following functions on it:

Modifiers

Functions	Explanation	Example	Time Complexity
insert()	Inserts element	`m.insert({1, 2});`	If a single element is inserted, then O(logN). If n elements are inserted then n*O(N+n), where N stands for the number of elements in the multimap (size)
erase()	Erases elements by • Iterator • Key • Range	• `m.erase(m.find(1));` • `m.erase(2);` • `m.erase(m.begin(),m.end());`	• Amortized Constant • O(logN) (N: the size of multimap) • Linear in the distance between the range
swap()	Swaps 2 maps with the same type, size may differ.	`m1.swap(m2);` `swap(m1, m2);`	Constant
clear()	Removes all the elements	`m.clear();`	O(N), linear is size of multimap

Capacity

Functions	Explanation	Example	Time Complexity
empty()	Returns 1 if the multimap is empty else 0	`m.empty();`	Constant, O(1)
size()	Returns the size	`m.size();`	Constant, O(1)
max_size()	Returns the maximum size (a really large number)	`m.max_size();`	Constant, O(1)

Operations

Functions	Explanation	Example	Time Complexity
find()	Returns the iterator to the element	`m.find();`	O(logN)
count()	Returns the no. of occurrences of the key passed	`m.count();`	O(logN) + Linear in no. of matches
upper_bound()	Returns the iterator to the next greater element	`m.upper_bound(4)`	O(logN)
lower_bound()	Returns the iterator to the element (if present) otherwise the next greater element	`m.lower_bound(4)`	O(logN), where N stands for the size of map
equal_range()	Returns the bounds of a range that includes all the elements in the container which have a key equivalent to k.	`m.equal_range(k)`	O(logN)

Iterators

Functions	Explanation	Example	Time Complexity
begin()	Returns an iterator to the first element in the multimap	`m.begin();`	Constant
end()	Returns an iterator to the theoretical element after the last element	`m.end();`	Constant
rbegin()	Returns a reverse iterator pointing to the last element of the multimap	`m.rbegin();`	Constant
rend()	Returns a reverse iterator pointing to the theoretical element just before the first element in the multimap	`m.rend();`	Constant

Topic: Unordered Multimap in C++ STL

Header File Required

To create an unordered_multimap in C++ we require the `unordered_map` header file. We can include it in the following way:

```
#include<unordered_map>
```

Also, we can use `bits/stdc++.h` as it includes every standard library. But it in itself is a non-standard header file of the GNU C++ library. It doesn't include only `unordered_multimap` but also other libraries thus it increases the compilation time. We can include that in the following way:

```
#include<bits/stdc++.h>
```

Declaring an unordered_multimap in C++

A map in C++ can be created by the following syntax:

```
unordered_multimap<data_type1, data_type2>name;
```

Here `data_type1` represents the data type of the key and `data_type2` represents the data type of value. They can be same or different.

For example:

```
unordered_multimap<int, int>m1;
unordered_multimap<int, string>m2;
unordered_multimap<string, string>m3;
```

Initializing a Map

You can also create and initialize a map in the following way:

```
unordered_multimap<key_datatype,value_datatype>name={{key1,value1},
                                                     {key2, value2},
                                                     {key3, value3}};
```

For example:

```
unordered_multimap<int, int>m4 = {{1,11}, {2,22}, {3,33}};
unordered_multimap<int, string>m5 = {{1, "Physics"},
                                     {2, "College"},
                                     {3, "Wallah"},
                                     {2, "Wallah"},
                                     {1, "Wallah"},
                                     {3, "Wallah"}};
unordered_multimap<string, string>m6 = {{"A)", "College"},
                                        {"B)", "Wallah"}};
```

Inserting into an unordered_multimap

Let us create an unordered_multimap and then insert key-value pairs into it.

```
unordered_multimap<char, string>m;
m.insert({'A', "Apple"});
m.insert({'B', "Ball"});
m.insert({'C', "College Wallah"});
m.insert({'C', "College"});
m.insert({'C', "Cat"});
```

Printing a Map

We can print the pair of the unordered_multimap using a for each loop.

Say we have to print the unordered_multimap we created above.

```
for(auto i : m)
    cout<<i.first<<" - "<<i.second<<endl;
```

Here, `i.first` prints the key and `i.second` prints the value of the pair in the map.

The **output** will be:

```
C Cat
C College
C College Wallah
B Ball
A Apple
```

Notice repetitive keys for values. Also no specific order is maintained.

Member Functions

Modifiers

Functions	Explanation	Example	Time Complexity
insert()	Inserts element	`m.insert({1, 2});`	If a single element is inserted: Average - O(1) Worst - O(N) Multiple elements inserted: Average: O(n) Worst - O(n*(N+1)) Where n elements are inserted and N stands for the number of elements in the unordered_multimap (size)
erase()	Erases elements by • Iterator • Key • Range	• `m.erase(m.find(1));` • `m.erase(2);` • `m.erase(m.begin(),m.end());`	Average: Linear in the number of elements removed i.e. O(n), n is no. of elements removed Worst: O(N) where N stands for the size of unordered_multimap
swap()	Swaps 2 maps with the same type, size may differ.	`m1.swap(m2);` `swap(m1, m2);`	Constant
clear()	Removes all the elements	`m.clear();`	O(N), linear is size of unordered_multimap

Capacity

Functions	Explanation	Example	Time Complexity
empty()	Returns 1 if the unordered_multimap is empty else 0	`m.empty();`	Constant, O(1)
size()	Returns the size	`m.size();`	Constant, O(1)
max_size()	Returns the maximum size (a really large number)	`m.max_size();`	Constant, O(1)

Operations

Functions	Explanation	Example	Time Complexity
find()	Returns the iterator to the element	`m.find();`	Average: O(1), constant Worst: O(N), N is the size of container
count()	Returns the no. of occurrences of the key passed	`m.count();`	Average: O(n), where n is no. of elements counted Worst: O(N), N is the size of container
equal_range()	Returns the bounds of a range that includes all the elements in the container which have a key equivalent to k.	`m.equal_range(k)`	Average: Constant Worst: O(N)

Iterators

Functions	Explanation	Example	Time Complexity
begin()	Returns an iterator to the first element in the map	`m.begin();`	Constant
end()	Returns an iterator to the theoretical element after the last element	`m.end();`	Constant

MCQ Questions

1. Consider the following code:

```
multimap<int,int>m;
m.insert({1,1});
m.insert({2,4});
m.insert({3,9});
m.insert({4,16});

m[5] = 25;  // Line-1
m[4] = 20;  // Line-2
```

What will happen because of Line-1 and Line-2 of code?

a) Pair {5,25} will be added to m and {4,16} will be updated to {4,20}

b) Pair {5,25} will be added to m and a new pair {4,20} will be added instead of changing the pair {4,16}

c) Compilation Error, there is no [ ] operator in multimap

d) Pair {5,25} will be added and Line-2 will be ignored as key 4 is already present

Answer: c

Explanation: Line-1 and Line-2 will throw compilation error because there is no [ ] operator in the multimap we can only insert pairs using insert() function. Had insert function been used then {5,25} and {4,20} both pairs would have been added because it is multimap and it supports repetitive keys.

2. What will be the output of the following code?

```
multimap<int,int>m;
m.insert({1,1});
m.insert({2,4});
m.insert({3,9});
m.insert({4,16});
m.insert({4,20});

auto a = m.equal_range(4);

for(auto it = a.first; it =/= a.second; it++)
    cout<<it->first<<" => "<<it->second<<endl;
```

a) 4 => 16

b) 4 =>16

4 => 20

c) 1 => 1

2 => 4

3 => 9

4 => 16

4 => 20

d) Compilation Error

Answer: b

Explanation: The equal_range() function returns the bounds of a range that includes all the elements in the container which have a key equivalent to k, where k is passed as a parameter. If no matches are found, the range returned has a length of zero, with both iterators pointing to the first element that has a key considered to go after k.

The function returns a pair, whose member pair::first is the lower bound of the range and pair::second is the upper bound.

Upcoming Class Teasers:

- Maps One Shot 3