# C++ map : Erase element by key or Iterator or Range

Varun February 11, 2017 erase, std::map, STL 1 Comment

In this article we will discuss the different ways to delete a key-value pair from map.

std::map provides 3 overloaded version of erase() to remove elements from map i.e.

- Erase by key
- Erase by Iterator
- Erase a range

Lets discuss them one by one,

## Erase Element from Map by Key

std::map provides a erase function that accepts the key and removes the elements (Key- Value pairs) whose key matches the passed key k.

```
1 size_type erase (const key_type& k);
```

It returns the number of elements deleted, but as there can be unique keys only in std::map. Therefore it will return 1 if element is deleted else it will return 0 if given key is not found in map.

Vector List Deque
Set Map MultiMap
STL Algorithms

1.) What's std::vector and why to use it?
2.) Different ways to initialize a vector
3.) How does std::vector works internally
4.) User Defined Objects & std::vector
5.) How to use vector efficiently in C++?
6.) std::vector and Iterator Invalidation
7.) Remove repeated elements from a vector
8.) Fill a vector with random numbers
9.) Hidden cost of std::vector
10.) Adding elements in Vector

Suppose we have a map of word and int i.e.

```
// Map of string & int i.e. words as key & there
// occurrence count as values
std::map<std::string, int> wordMap = { { "is", 6 }, { "the", 5 }
        { "hat", 9 }, { "at", 6 }, { "of", 2 }, { "hello", 1 } }
```

Now, lets delete the element with key “is” i.e.

```
// Removes the element from map with given key.
int result = wordMap.erase("is");
```

Checkout Complete example as follows,

```
#include <iostream>
#include <map>
#include <string>

int main() {

    // Map of string & int i.e. words as key & there
    // occurrence count as values
    std::map<std::string, int> wordMap = { { "is", 6 }, { "the",
            { "hat", 9 }, { "at", 6 }, { "of", 2 }, { "hello", 1

    std::cout << "Map Entries Before Deletion" << std::endl;
    // Print the map elements
    for (auto elem : wordMap)
        std::cout << elem.first << " :: " << elem.second << std:


    // Removes the element from map with given key.
    int result = wordMap.erase("is");

    // Check if element is actually deleted from map
    if(result == 1)
        std::cout<<"Element with key 'is' deleted"<<std::endl;
    else
        std::cout<<"Element with key 'is' Not Found"<<std::endl;


    std::cout << "Map Entries After Deletion" << std::endl;
    // Print the map elements
    for (auto elem : wordMap)
        std::cout << elem.first << " :: " << elem.second << std:

    return 0;
}
```

**Output:**

```
Map Entries Before Deletion
at :: 6
hat :: 9
hello :: 1
is :: 6
of :: 2
the :: 5
Element with key 'is' deleted
```

```
Map Entries After Deletion
at :: 6
hat :: 9
hello :: 1
of :: 2
the :: 5
```

# Erase Element from Map by Iterator

std::map provides a erase function that accepts the Iterator and removes the element pointed by the iterator.

```
iterator erase (const_iterator position);
```

It returns the iterator of the next element.

Suppose we have a map of word and int i.e.

```
// Map of string & int i.e. words as key & there
// occurrence count as values
std::map<std::string, int> wordMap = { { "is", 6 }, { "the", 5 }
        { "hat", 9 }, { "at", 6 }, { "of", 2 }, { "hello", 1 } }
```

Now, lets delete the element with key “of” i.e.

Let’s first find the iterator pointing to it i.e.

```
// Get the iterator of element with key 'of'
std::map<std::string, int>::iterator it = wordMap.find("of");
```

Then check if iterator is valid or not. If its valid then only remove the element through it

```
if(it != wordMap.end())
{
    // Remove the element pointed by iterator
    wordMap.erase(it);
}
```

Checkout Complete example as follows,

```
#include <iostream>
#include <map>
#include <string>

int main() {

    // Map of string & int i.e. words as key & there
    // occurrence count as values
    std::map<std::string, int> wordMap = { { "is", 6 }, { "the",
```

```
            { "hat", 9 }, { "at", 6 }, { "of", 2 }, { "hello", 1

    std::cout << "Map Entries Before Deletion" << std::endl;
    // Print the map elements
    for (auto elem : wordMap)
        std::cout << elem.first << " :: " << elem.second << std:


    // Get the iterator of element with key 'of'
    std::map<std::string, int>::iterator it = wordMap.find("of")

    // Check if iterator is valid.
    if(it != wordMap.end())
    {
        // Remove the element pointed by iterator
        wordMap.erase(it);

        std::cout<<"Element Removed"<<std::endl;
    }
    else
        std::cout<<"Key Not Found"<<std::endl;


    std::cout << "Map Entries After Deletion" << std::endl;
    // Print the map elements
    for (auto elem : wordMap)
        std::cout << elem.first << " :: " << elem.second << std:


    return 0;
}
```

**Output:**

```
Map Entries Before Deletion
at :: 6
hat :: 9
hello :: 1
is :: 6
of :: 2
the :: 5
Element Removed
Map Entries After Deletion
at :: 6
hat :: 9
hello :: 1
is :: 6
the :: 5
```

# Erase Element from Map by Iterator Range

std::map provides a erase function that accepts the Iterator start and end and removes all the elements in the given range i.e. from start to end -1 i.e.

```
<span class="kw4">void</span> erase<span class="br0">(</span> it
```

Suppose we have a map of word and int i.e.

```C++
// occurrence count as values
std::map<std::string, int> wordMap = { { "is", 6 }, { "the", 5 }
        { "hat", 9 }, { "at", 6 }, { "of", 2 }, { "hello", 1 } }
```

Now, lets delete the elements in range of two iterator it1 & it2 i.e.

```
// Remove the element pointed by iterator
 wordMap.erase(it1, it2);
```

It will delete the elements from it1 to it2 -1.

Checkout Complete example as follows,

```
#include <iostream>
#include <map>
#include <string>

int main() {

    // Map of string & int i.e. words as key & there
    // occurrence count as values
    std::map<std::string, int> wordMap = { { "is", 6 }, { "the",
            { "hat", 9 }, { "at", 6 }, { "of", 2 }, { "hello", 1


    std::cout << "Map Entries Before Deletion" << std::endl;
    // Print the map elements
    for (auto elem : wordMap)
        std::cout << elem.first << " :: " << elem.second << std:

    // Create an iterator pointing to begin of map
    std::map<std::string, int>::iterator it1 = wordMap.begin();

    // Create an iterator pointing to begin of map
    std::map<std::string, int>::iterator it2 = wordMap.begin();
    // Increment Iterator
    it2++;
    // Increment Iterator
    it2++;
    // Itr2 is now pointing to 3rd element


    // Check if iterator is valid.
    if (it1 != wordMap.end() && it2 != wordMap.end())
    {
        // Remove the element pointed by iterator
        wordMap.erase(it1, it2);
        std::cout << "Elements Removed" << std::endl;
    }
    else
        std::cout << "Key Not Found" << std::endl;


    std::cout << "Map Entries After Deletion" << std::endl;
    // Print the map elements
    for (auto elem : wordMap)
        std::cout << elem.first << " :: " << elem.second << std:

```

```
46      return 0;
47  }
```

**Output:**

```
1  Map Entries Before Deletion
2  at :: 6
3  hat :: 9
4  hello :: 1
5  is :: 6
6  of :: 2
7  the :: 5
8  Elements Removed
9  Map Entries After Deletion
10 hello :: 1
11 is :: 6
12 of :: 2
13 the :: 5
```



## Related Posts:

- C++ Map: Erase by Value or Callback while iterating...
- How to Erase / Remove an element from an unordered_map
- c++11 unordered_map : erase elements while iterating...
- Different ways to Erase / Delete an element from a...
- C++ Map : Operator [] – Usage Details
- How check if a given key exists in a Map | C++
- How to remove elements from a List while Iterating
- How to erase elements from a list in c++ using iterators
- How to search by value in a Map | C++
- How to copy all Values from a Map to a Vector in C++

- Different ways to insert elements in an unordered_map
- C++ std::list Tutorial, Example and Usage Details
- Erase elements from a Set while Iterating in C++...
- How to find an element in unordered_map
- How to iterate over an unordered_map in C++11
- C++ : How to find duplicates in a vector ?
- Python : Check if all elements in a List are same or...
- How to remove Substrings from a String in C++
- Different Ways to initialize an unordered_map
- Python : How to Check if an item exists in list ? |...
- Unordered_map Usage Tutorial and Example
- C++ : How to find an element in vector and get its index ?
- C++ : Different ways to insert elements in Set
- map vs unordered_map | When to choose one over another ?
- How to Access Element by index in a Set | C++

std::map, std::map::erase

## 1 Comment Already

## Leave a Reply

Your email address will not be published. Required fields are marked * This site

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

uses Akismet to reduce spam. Learn how your comment data is processed.

« How to search by value in a Map | C++

C++ Map : Operator [] – Usage Details »

**Python : List**

Check if an item exists in List

Check if a list contains all the elements of other list

Create a List and initialize with values

How to Iterate over a List

Insert an element at specific index in List

Sort a list of tuples by 2nd Item

Sort a list of strings

Add an element in list | append() vs extend()

Check if all elements in a List are same

Merge / Join two or more lists

Remove Duplicates from a List

Convert a list to string

Remove element from a list by value or Index

Remove multiple elements from list

**Python : Dictionary**

**C++11 – Utilities**

std::bind

auto specifier

Variadic Templates

**C++11 – Unordered Set**

1.) unordered_set Basic Example

2.) Initializing an unordered_set

3.) Inserting elements in an unordered_set

4.) Searching an element in unordered_set

5.) unordered_set – Custom Hasher & Comparator

6.) Unordered_set & User defined classes

**C++11 – UnorderedMap**

Basic Usage Detail and Example

Initializing an unordered_map



**Design Patterns**

Behavioral Design Patterns

- Observer Design Pattern
- State Design Pattern
- Strategy Design Pattern

Structural Design Patterns

- Composite Design Pattern
- Flyweight Design Pattern

Creational Design Patterns

- Factory Method Design Pattern

**Pointers**

Pointer vs Refrence

**STL – Vector**

1.) What's std::vector and why to use it?

2.) Different ways to initialize a vector

3.) How does std::vector works internally

4.) User Defined Objects & std::vector

5.) How to use vector efficiently in C++?

6.) std::vector and Iterator Invalidation

7.) Remove repeated elements from a vector

8.) Fill a vector with random numbers

9.) Hidden cost of std::vector

10.) Adding elements in Vector

**STL – Deque**

1.) What is std::deque and how deque works internally.

2.) deque vs vector : What to choose ?

Creating Dictionaries in Python

Iterating over Dictionaries in Python

Check if a key exists in Dictionary

Get list of all the keys in Dictionary

Get list of all the Values in a Dictionary

Remove multiple keys in Dictionary while Iterating

Remove a key from Dictionary

Add key/value pairs in Dictionary

Find keys by value in Dictionary

Sort a Dictionary by key or Value

Copy a dictionary | Shallow vs Deep Copy

Convert a list to dictionary

### Python Strings

Access characters in string by index in Python

Iterate over the characters in string

How to Replace characters in a string ?

### Java – Hashmap

What is Hashing and Hash Table?

Associating Multiple values with same Key

Remove elements while Iterating

Update the value of an existing key

Get all keys by a value in HashMap

### Java – HashSet

Searching in unordered_map

Insert elements in unordered_map

Erasing an element

Erase elements while iterating

std::map vs std::unordered_map

### C++11 Smart Pointers

shared_ptr<> Tutorial and Examples

shared_ptr and Custom Deletor

shared_ptr vs raw pointer

Create shared_ptr objects carefully

weak_ptr Tutorial | shared_ptr and Cyclic References

unique_ptr<> Tutorial and Examples

### C++11 Multithreading

Part 1: Three Ways to Create Threads

Part 2: Joining and Detaching Threads

Part 3: Passing Arguments to Threads

Part 4 : Sharing Data & Race Conditions

Part 5 : Fixing Race Conditions using mutex

Part 6 : Need of Event Handling

Part 7: Condition Variables

Part 8: std::future and std::promise

Part 9: std::async Tutorial & Example

Part 10: std::packaged_task<> Tutorial

Allocating 2D Array Dynamically

### Callbacks in C++

Function Pointers

Function Objects & Functors

### C++ Strings

Find and Replace all occurrences of a string

Find all occurrences of a sub string

Case Insensitive string::find

Convert First Letter of each word to Upper Case

Converting a String to Upper & Lower Case

Trim strings in C++

C++ : How to split a string using String and character as Delimiter?

startsWith() Implementation

endsWith() Implementation

Remove Sub Strings from String

### C++ Memory Manangement

Memory Leaks

new and delete operator

delete vs []delete

Out Of Memory Errors

Overload new & delete

Restrict Dynamic Deletion

Placement new operator

Delete 'this' pointer

### Polymorphism

### STL – List

1.) std::list Internals & Usage Details

2.) List vs Vector

3.) Different ways to Initialize a list

4.) Erase elements using iterators

5.) Remove elements while Iterating

6.) Remove elements based on External Criterion

7.) Get element by index in List

8.) Searching an element in std::list

9.) Different Ways to iterate over a List

10.) Sorting a List & custom Comparator

### STL – Set

1.) C++ Set basic example and Tutorial

2.) Using std::set with user defined classes

3.) std::set and external Sorting criteria | Comparator

4.) Access Element by index in Set

5.) How to insert elements in Set

6.) How to iterate over a Set

7.) Removing an element from Set

8.) Erase elements while Iterating & Generic erase_if()

### STL – Map

1.) std::map Usage Detail with examples

2.) std::map and Comparator

3.) std::map & User defined class objects as

**Java Interview Questions**



**C++11 Rvalue References**







**STL Multimap**



**STL Algorithms**



**Terms and Conditions**