thispointer.com

Set vs Map : How to choose a right associative container ?

Varun May 6, 2015 C++, std::map, std::set, STL 3 Comments

In this article we will discuss how set and map are different and what factors we need to keep in mind while choosing right associative container.

Let's start from set,

Set :

- Set is an associative container which we need to store unique elements.
- It always keeps the elements in sorted order.
- Internally it maintains a balanced binary search tree of elements. Therefore when we search an element inside the set then it takes only log(n) complexity to search it.

Important Point about set – Once added then cannot change i.e.

Each element added inside the set is const i.e. you cannot modify that element, because if you could, then it would hamper set's internal data structure i.e. it will lose its internal balanced binary search tree property and results in undefined behavior.

Let's understand by example,

Create a set to store unique names of Departments and then try to change one,

```
#include <iostream>
#include <map>
#include <set>
#include <iterator>
#include <algorithm>
#include <string>

void example1()
{
    std::set<std::string> setOfDepartments;

    setOfDepartments.insert("First");
    setOfDepartments.insert("Second");
    setOfDepartments.insert("Third");
    std::for_each(setOfDepartments.begin(), setOfDepartments.end
                                            std::cout<<(elem)<<"
                                            });

    // Now Try to change the element

     std::set<std::string>::iterator it = setOfDepartments.find(
     if(it != setOfDepartments.end())
     {
         std::cout << std::endl<< *it;
         //*it = "Fourth"; // NOT ALLOWED
     }

}
int main()
{
    Example1();
    return 0;
}
```

In above example you cannot change the element through iterator because here our complete element which we have added in SET is a key and you cannot modify any key inside the SET.
But what if want to associate some Value with this key and want to change that associated value at run time. Then we need something other than SET and that's it MAP.

Map:

- Map is a associative container that is used to store key-value pair of elements with unique keys.
- It always keeps the inserted pairs in sorted order based on the key.

- Internally it maintains a balanced binary search tree to store keys. Therefore when searching key inside the map takes only log(n) complexity.
- We cannot modify the key of any inserted pair in map.
- We can modify the value associated with a key in any inserted pair in map.

Let's understand by example,

Create a map of Departments and No of Employee count. Then search a Department and change its associated no of employee count.

```
#include <iostream>
#include <map>
#include <set>
#include <iterator>
#include <algorithm>
#include <string>

void example2()
{
    // Map of Department and Employee count in that Department
    std::map<std::string, int > mapOfDepEmpCount;

    mapOfDepEmpCount.insert(std::make_pair("First", 0));
    mapOfDepEmpCount.insert(std::make_pair("Second", 0));
    mapOfDepEmpCount.insert(std::make_pair("Third", 0));

    std::map<std::string, int >::iterator it = mapOfDepEmpCount.
     if(it != mapOfDepEmpCount.end())
     {
         std::cout << std::endl << "Department = "<< it->first <
         // You can change the value associated with the key
         it->second = 10;
         //it->first = "sss"; // You cannot chage the key
     }

     it = mapOfDepEmpCount.find("Second");
     if(it != mapOfDepEmpCount.end())
     {
         std::cout << std::endl << "Department = "<< it->first <

     }

}

int main()
{
    example2();
    return 0;
}
```

When to choose SET and when MAP?

So, if you want to maintain a data structure of unique keys only without any associated value that plan to modify in future then use set. If you want to modify any element in set then erase it and then insert the new one.
Whereas, use map if you want to maintain a data structure of unique keys and some associated value with each key that you want to change in future.

Click Here to Subscribe for more Articles / Tutorials like this.

Related Posts:

- map vs unordered_map | When to choose one over another ?
- Unordered_map Usage Tutorial and Example
- Different ways to insert elements in an unordered_map
- C++ : Different ways to insert elements in Set
- multimap Example and Tutorial in C++
- What is std::deque and how deque works internally.
- How to Access Element by index in a Set | C++
- C++ Map Insert Example
- How to Iterate over a map in C++
- How to Insert elements in an unordered_set in C++11
- How to Sort a Map by Value in C++
- C++ : How to check if a Set contains an element |...
- How to search by value in a Map | C++
- C++ : How to find duplicates in a vector ?
- C++ : How to insert element in vector at specific...
- How to Erase / Remove an element from an unordered_map
- C++11 'auto' Tutorial and Examples
- std::unordered_set Basic Usage & Example
- How to find an element in unordered_map

- std::generate Tutorial and example
- Designing Callbacks in C++ – Part 2: Function...
- How to iterate a map in reverse order – C++
- How check if a given key exists in a Map | C++
- C++ : How to get element by index in List
- Overloading new and delete operators at Global and...

map, set, set vs map, std::map, std::set, std::set vs std::map

3 Comments Already

Bilal Tahir Khan - May 18th, 2015 at 11:13 pm

Hi,
Superb post Thanks for sharing with us keep up it
Have a nice week Ahead.

Reply

Vishal Kukreja - September 10th, 2015 at 8:49 pm

Hi,
Actually Map is based on RED-BLACK tree structure which is more defined version of Binary search Tree.

Reply

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.

« Iterating over a range of User Defined objects and calling member function using std::for_each

How to be Self Motivated as a Software Developer »

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

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

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

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.

Python : Dictionary

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

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

Basic Usage Detail and Example

Initializing an unordered_map

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<>

Pointer vs Refrence

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

2.) deque vs vector : What to choose ?

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

Java – HashSet

What is Hashing and Hash Table?

Create and add elements in a HashSet

Iterate over a HashSet

Search for an element in HashSet

Merge two HashSets

Initializing HashSet from an Array

Convert a HashSet into an Array

Merge an Array in a HashSet

Java Interview Questions

Method Overriding Tutorial

Overriding with Different Return Type

Calling Base class's overridden method

Preventing Method Overriding

Need of preventing Method Overriding

Tutorial

C++11 Rvalue References

lvalue vs rvalue

Is rvalue immutable in C++?

What is rvalue reference in C++11

Move Contsructor

Polymorphism

Virtual Functions

vTable and vPointer

2.) std::map and Comparator

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

4.) Set vs Map

5.) How to Iterate over a map in C++

6.) Map Insert Example

7.) Iterate a map in reverse order

8.) Check if a key exists in a Map

9.) Search by value in a Map

10.) Erase by Key | Iterators

11.) C++ Map : Operator []

12.) Erase by Value or callback

13.) copy all Values from a Map to vector

STL Multimap

MultiMap Example and Tutorial

multimap::equals_range – Tutorial

STL Algorithms

std::sort Tutorial & Example

std::unique Tutorial & Example

1.) Using std::find & std::find_if with User Defined Classes

2.) Iterating over a range of User Defined objects and calling member function using std::for_each

Terms and Conditions

Terms and Conditions Policy