What does iterator->second mean?

57

In C++, what is the type of a `std::map<>::iterator` ?

We know that an object `it` of type `std::map<A,B>::iterator` has an overloaded `operator ->` which returns a `std::pair<A,B>*` , and that the `std::pair<>` has a `first` and `second` member.

But, what do these two members correspond to, and why do we have to access the value stored in the map as `it->second` ?

12

c++ stl iterator

share edit

edited Jan 27 '14 at 21:08
Nicholas Wilson
5,324 1 12 53

asked Mar 16 '13 at 15:54
Noich
2,191 2 34 70

6 A `std::map` stores a *key* and a *value*. `map::iterator.second` refers to the *value*. – Alok Save Mar 16 '13 at 15:56

18 ironically this question which was closed as "too localized" is the top result on Google for `iterator second` which is a very reasonable query related to the standard library. – Gabriel Southern Jun 21 '13 at 15:51

4 This question helped me, I had the exact curiosity – user1720205 Sep 17 '13 at 5:58

13 This question should not be closed as localized. I had the question, googled, found this, got answer. – Pieter Müller Nov 18 '13 at 11:03

1 Wow, remarkable. I was looking for this answer on Google also! Won't help any future visitors my hat. I live in Australia, 10 to 1 that this was posted in the U.S. or the U.K.! – Chris Sherlock Jan 26 '14 at 11:49

add a comment

2 Answers

active oldest votes

77

I'm sure you know that a `std::vector<X>` stores a whole bunch of `X` objects, right? But if you have a `std::map<X, Y>` , what it actually stores is a whole bunch of `std::pair<const X, Y>` s. That's exactly what a map is - it pairs together the keys and the associated values.

When you iterate over a `std::map` , you're iterating over all of these `std::pair` s. When you dereference one of these iterators, you get a `std::pair` containing the key and its associated value.

```
std::map<std::string, int> m = /* fill it */;
auto it = m.begin();
```

Here, if you now do `*it` , you will get the the `std::pair` for the first element in the map.

Now the type `std::pair` gives you access to its elements through two members: `first` and `second` . So if you have a `std::pair<X, Y>` called `p` , `p.first` is an `X` object and `p.second` is a `Y` object.

So now you know that dereferencing a `std::map` iterator gives you a `std::pair` , you can then access its elements with `first` and `second` . For example, `(*it).first` will give you the key and `(*it).second` will give you the value. These are equivalent to `it->first` and `it->second` .

share edit

answered Mar 16 '13 at 16:04

Joseph Mansfield
67.5k 7 104 195

1 Why don't they just use [0] and [1] (for "first" and "second") like everything else in programming? – user1052335 Apr 28 at 23:20

3 @AdamCross Because `operator[]` has to return a specific type but `first` and `second` can have different types. On the other hand, `std::tuple` has a special helper function `std::get` for accessing its elements by index. – Joseph Mansfield Apr 28 at 23:29

Thank you for answering. That makes sense now. – user1052335 Apr 29 at 2:12

add a comment