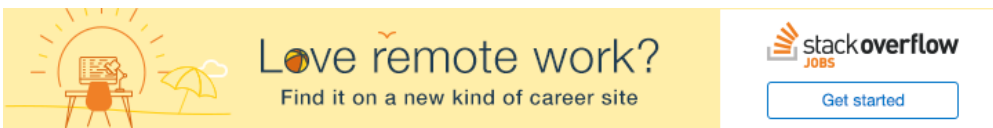


Join the Stack Overflow Community

Stack Overflow is a community of 7.6 million programmers, just like you, helping each other.
Join them; it only takes a minute:

[Sign up](#)

Element at index in a std::set?



I've stumbled upon this problem: I can't seem to select the item at the index' position in a normal std::set. Is this a bug in STD?

Below a simple example:

```
#include <iostream>
#include <set>

int main()
{
    std::set<int> my_set;
    my_set.insert(0x4A);
    my_set.insert(0x4F);
    my_set.insert(0x4B);
    my_set.insert(0x45);

    for (std::set<int>::iterator it=my_set.begin(); it!=my_set.end(); ++it)
        std::cout << ' ' << char(*it); // ups the ordering

    //int x = my_set[0]; // this causes a crash!
}
```

Anything I can do to fix the issue?

c++ set std

asked Dec 9 '13 at 18:07

 **hauron**
2,309 1 17 36

4 my_set[0] shouldn't compile. – [chris](#) Dec 9 '13 at 18:08

2 You are asking the wrong question, because you are using the wrong container. Each standard container has been designed with a certain number of uses in mind, and in turn does not allow others (directly). So, first off, you need to identify what operations you need and then [pick the right container](#) – [Matthieu M.](#) Dec 9 '13 at 18:29

Possible duplicate of [Get element from arbitrary index in set](#) – [Ciro Santilli](#) 刘骁波死 六四事件 法轮功 Jan 15 at 17:42

It was a joke question initially, but the answers proved to be pretty useful. If taken seriously, it is indeed a duplicate. – [hauron](#) Jan 16 at 11:06

5 Answers

It doesn't cause a crash, it just doesn't compile. set doesn't have access by index.

You can get the nth element like this:

```
std::set<int>::iterator it = my_set.begin();
std::advance(it, n);
int x = *it;
```

Assuming `my_set.size() > n`, of course. You should be aware that this operation takes time approximately proportional to `n`. In C++11 there's a nicer way of writing it:

```
int x = *std::next(my_set.begin(), n);
```

Again, you have to know that `n` is in bounds first.

edited Dec 9 '13 at 18:19

answered Dec 9 '13 at 18:10



Steve Jessop


216k 22 330 580

Of course, if OP expected `my_set[0]` to return `0x4A`, this still won't do what's wanted. – Useless Dec 9 '13 at 18:15

3 @Useless: true. But if they expect it to return the first value from the loop earlier in their code then they're good. In general, if the questioner doesn't know what a `set` is they're going to see a series of surprising results until they finally give up and RTFM :-p – Steve Jessop Dec 9 '13 at 18:16


Pardon me for this, but the question was supposed to be a joke (the values are ASCII hex codes for "JOKE", thought `set` is unordered, hence iterating won't yield the same result). However, I did not know about `std::advance` or `std::next`, so thanks for sharing that! – hauron Mar 25 '14 at 15:39

1 @hauron I'm not too concerned with how you conceived this question. But I believe myself and 13 other people think you should accept this answer. – Jonathan Mee Mar 16 '16 at 13:28



Love remote work?

Find it on a new kind of career site



Get started

A usual implementation of `std::set` is to use [binary search trees](#), notably [self-balancing binary search trees](#) such as [red-black trees](#)

They don't give you constant time access to the `n`-th element. However, you seems to want the first. So try in [C++11](#):

```
auto it = my_set.begin();
int first=0;
if (it != my_set.end()) first = *it;
```

answered Dec 9 '13 at 18:10



Basile Starynkevitch

142k 9 124 260

Thank you for the answer to this - sorry for that - joke question. However, from C++11 your code only seems to take the new meaning of 'auto'. – hauron Mar 25 '14 at 15:41

This is not a bug in the STD. There is no random access in a `std::set`. If you need random access by index, you can use `std::vector`

edited Dec 10 '13 at 19:22

answered Dec 9 '13 at 18:11



José D.

1,625 2 17 28

1 Well, there is. By default, it uses `std::less` to order things. – chris Dec 9 '13 at 18:13

You are right, I was thinking of `std::unordered_set`. Edited – José D. Dec 9 '13 at 18:14

1 The order is not implementation-dependent. As Chris says, it uses `std::less` unless told otherwise, which in the case of `int` is just a fancy way of saying `<`. – Steve Jessop Dec 9 '13 at 18:21

Trollkemada, thank you for the answer, though, this question was kind of a troll question (see comments to the question or Steve's answer). – hauron Mar 25 '14 at 15:39

Sometimes there's a good reason for needing a set you can index into. I had to implement this functionality recently to support a legacy API which has functions to return the number of items, and the item at an index, so that the caller can enumerate the items.

My way of solving the problem is to use `std::vector`, and use `std::equal_range` to find and insert or delete items in the set. For example, inserting a new item into the set looks like this:

```
std::vector<std::string> my_set;

...

std::string new_item("test");

auto range = std::equal_range(my_set.begin(), my_set.end(), new_item);
if (range.first == range.second)
    my_set.insert(range.first, new_item);
```

Deleting is very similar: use `equal_range` to find the item, and if `range.first` is *not* equal to `range.second`, delete that range.

answered Aug 27 '15 at 9:53



Graham Asher

812 7 17

```
std::set<int> my_set;
my_set.insert(0x4A);
my_set.insert(0x4F);
my_set.insert(0x4B);
my_set.insert(0x45);

int arr[my_set.size()];

set<int>::iterator it = my_set.begin();
for (int i = 0; i < my_set.size(); i++) {
    arr[i] = *it;
    it++;
}
cout << arr[0];
```

Edit: Edited code. You can't access set using index but the above method would provide an "index" i if you want to copy the elements from set into an array, provided you have created an array of sufficient size before hand.

edited Apr 19 at 16:11

answered Apr 8 at 15:22



katta

99 1 1 8

How does this answer the question? You're only iterating over the set in a slightly different manner. – [cpburnz](#) Apr 8 at 18:40

@cpburnz Non of the other other answers iterated a set with an index. There is an use for iterating a set using index as shown in my example. – [katta](#) Apr 19 at 16:16