How to erase element from std::vector<> by index?

I have a std::vector, and I want to delete the n'th element. How do I do that?

```
std::vector<int> vec;

vec.push_back(6);
vec.push_back(-17);
vec.push_back(12);

vec.erase(???);
```

Please help!

c++ stl vector erase

edited Jan 7 '14 at 22:49
rstackhouse
874 10 19

asked May 17 '09 at 17:59
dau_man
1,184 2 8 5

3 Consider using a std::deque which provides inserting and deleting at both ends. – Dario May 17 '09 at 18:20

8 Answers

To delete a single element, you could do:

```
std::vector<int> vec;

vec.push_back(6);
vec.push_back(-17);
vec.push_back(12);

// Deletes the second element (vec[1])
vec.erase(vec.begin() + 1);
```

Or, to delete more then one element at once:

```
// Deletes the 2nd through 3rd elements (vec[1], vec[2])
vec.erase(vec.begin() + 1, vec.begin() + 3);
```

edited Sep 27 '14 at 15:43
ilent2
2,923 2 10 22

answered May 17 '09 at 18:01
mmmmmmmm
8,205 2 21 51

49 `vec.erase(vec.begin() + 1, vec.begin() + 2);` will only erase one element, since the second iterator is exclusive. – fredoverflow Jan 28 '12 at 17:13

13 Note: @FredOverflow's comment applies only when the 2nd arg to `erase` was a **2**, but it has since been changed to a **3** – bobobobo Mar 14 '13 at 23:09

8 Note also binary `operator+` is **not** necessarily defined for iterators on other container types, like `list<T>::iterator` (you cannot do `list.begin() + 2` on an `std::list`, you have to use `std::advance` for that) – bobobobo Mar 14 '13 at 23:35

are you stating that the "+1" is the first element myVector[0] or the actual position myVector[1] – Karl Morrison Sep 19 '14 at 7:38

it+1 is element with id 1, e.g container[1]. first element is +0. See the comment below... – Nick Apr 12 at 6:22

The erase method on std::vector is overloaded, so its probably clearer to call:

```
vec.erase(vec.begin() + index);
```

when you only want to erase a single element.

answered May 17 '09 at 18:08

CodeBuddy
1,881 1 10 23

This is not good if you have only one element ... – alap Jan 20 '14 at 10:10

@Laszlo-AndrasZsurzsa Why? – manuell Feb 4 '14 at 17:50

2 But that problem appears no matter how many elements you have. – Zyx 2000 Sep 1 '14 at 9:32

8 if there's only one element, index is 0, and so you get `vec.begin()` which is valid. – Anne Quinn Jan 27 '15 at 18:28

3 I wish someone would have mentioned that `vec.erase(0)` does not work, but `vec.erase(vec.begin()+0)` (or without +0) does. Otherwise I get no matching function call, which is why I came here – ItsmeJulian Feb 15 at 20:19

|

```
template <typename T>
void remove(std::vector<T>& vec, size_t pos)
{
    std::vector<T>::iterator it = vec.begin();
    std::advance(it, pos);
    vec.erase(it);
}
```

answered Mar 10 '11 at 20:47

Max
402 4 8

2 Max, what makes that function better than: `template <typename T> void remove(std::vector<T>& vec, size_t pos) { vec.erase(vec.begin + pos); }` I'm not saying either is better, merely asking out of personal interest and to return the best result this question could get. – user1664047 Sep 11 '12 at 20:50

8 @JoeyvG: Since a `vector<T>::iterator` is a random-access iterator, your version is fine and maybe a bit clearer. But the version that Max posted should work just fine if you change the container to another one that doesn't support random-access iterators – Kevin Ballard Sep 11 '12 at 21:28

5 RIP, user1664047. – Limited Atonement Feb 23 '15 at 21:55

I've always found the `begin() + n` thing a little odd.

I prefer this, which has the added advantage of being shorter to type.

```
vec.erase(&vec[index]);
```

Or, similarly, if you require bounds checking on the erase

```
vec.erase(&vec.at(index));
```

edited Mar 9 '14 at 20:36

answered Dec 6 '13 at 23:53

Roddy
40.6k 28 124 221

1 Do you know of any reason netbeans would give an error on `v.erase(&v.at(17));` ? But no error for `v.erase(v.begin() + 17)` ? – Instinct Apr 4 '14 at 22:53

15 This solution only works if the `std::vector` uses `value_type *` as its `iterator` type, or allows implicit conversion from one to the another. So you can't count on this working with all standard library implementation (e.g., neither libc++ nor libstdc++ allow this code to compile). – Ken Wayne VanderLinde Apr 19 '14 at 17:12

1 @KenWayneVanderLinde Eek! So much the "standardness" of the standard library. And why on earth does my answer has 7 upvotes and no downvotes... – Roddy Apr 19 '14 at 20:38

2 This doesn't work on Microsoft's STL implementation as T* cannot be cast to the iterator type. – Cthutu Mar 24 '15 at 20:41

this answer won't work on any major implementation of c++ and the answer have almost 20 upvoters%) – RiaD Oct 7 '15 at 18:32

|

To delete an element use the following way:

```
1  // declaring and assigning array1
2  std:vector<int> array1 {0,2,3,4};
3
4  // erasing the value in the array
5  array1.erase(array1.begin()+n);
```

for **more broad overview** you can visit:- http://www.cplusplus.com/reference/vector/vector/erase/

answered Jun 18 at 22:14

Nikhil Balyan
64 11

In Actual Erase function works for two profiles -

```
- Removing a Single Element

    iterator erase (iterator position);

- Removing a range of elements

    iterator erase (iterator first, iterator last);
```

Now since std::vec.begin() marks the start of container so if we want to delete ith element in our vector we can use

```
vec.erase(vec.begin() + index);
```

If you look closely vec.begin() is just a pointer to the starting position of our vector and adding value of i to it increments the pointer to i position, So instead we can access the pointer to ith element by

```
&vec[i]
```

so we can write

```
vec.erase(&vec[i]); //to delete the ith element
```

answered May 20 at 21:07

Varun Garg
73 6

Erase method will be used in two ways.

```
1. erasing single element,

   vector.erase( vector.begin() + 3 );        // deleting 3rd element

2. erasing range of elements,

   vector.erase( vector.begin() + 3, vector.begin() + 5 );        // deleting from
3rd element to 5th element
```

answered May 4 at 11:06

Eswaran Pandi
21 3

If you work with large vectors (size > 100000) and want to delete lots of elements, I would recommend to do something like this:

```
int main(int argc, char** argv) {

 vector <int> vec;
 vector <int> vec2;

 for (int i = 0; i < 20000000; i++){
     vec.push_back(i);}

 for (int i = 0; i < vec.size(); i++)
 {
     if(vec.at(i) %3 != 0)
         vec2.push_back(i);
 }
 vec=vec2;
 cout << vec.size() << endl;
}
```

The code takes every number in vec, that can't be divided by 3 and copies it to vec2. Afterwards it copies vec2 in vec. It is pretty fast. To proceed 20.000.000 elements this algorithm only takes 0.8 sec!

I did the same thing with the erase-method and it takes lots and lots of time:

```
Erase-Version (10k elements)  : 0.04 sec
Erase-Version (100k elements) : 0.6  sec
Erase-Version (1000k elements): 56   sec
Erase-Version (10000k elements): ...still calculating (>30min)
```

answered May 12 at 14:28

Fabian
29

3 how does this answer the question? – Regis Portalez May 12 at 14:34

This does not provide an answer to the question. Once you have sufficient reputation you will be able to comment on any post; instead, provide answers that don't require clarification from the asker. - From Review – cpburnz May 13 at 1:28

1 Interesting, but not relevant to the question! – Roddy Jun 9 at 20:30

Won't a in-place algorithm faster? – user202729 Sep 21 at 4:17