How to find if an item is present in a std::vector?

All I want to do is to check whether an element exists in the vector or not, so I can deal with each case.

```
if ( item_present )
   do_this();
else
   do_that();
```

c++ vector std

edited Oct 8 '16 at 4:09
User that is not a user
128 10

asked Feb 20 '09 at 21:58
Joan Venge
71.9k 145 356 586

searching in a vector is very slow since you have to look at every single element of the vector so consider using a map if you're doing a lot of lookups – naumcho Feb 20 '09 at 22:31

4 @naumcho: If the vector is sorted there's always binary search, as posted below. This makes it as fast as a map and if you're only storing values (not key/value maps) then it's going to use a lot less memory. – Adam Hawes Feb 21 '09 at 1:01

1 maps are certainly not the best choice, but using set might be useful. If you need O(1) lookup time, hash_set is the way to go. – Philipp Oct 8 '10 at 8:58

A superb answer present on a duplicate question: stackoverflow.com/a/3451045/472647 – CodeMouse92 Jun 18 '15 at 3:11

17 Answers

You can use `std::find` from `<algorithm>` :

```
std::find(vector.begin(), vector.end(), item) != vector.end()
```

This returns a bool (`true` if present, `false` otherwise). With your example:

```
#include <algorithm>

if ( std::find(vector.begin(), vector.end(), item) != vector.end() )
   do_this();
else
   do that();
```

edited Oct 7 '15 at 13:46
Ziezi
3,664 3 15 31

answered Feb 20 '09 at 22:00
MSN
39.6k 5 53 85

174 I don't see how count() could be faster than find(), since find() stops as soon as one element is found, while count() always has to scan the whole sequence. – Éric Malenfant Feb 21 '09 at 3:29

89 Don't forget to `#include <algorithm>` or else you might get very strange errors like 'can't find matching function in namespace std' – RustyX Mar 2 '12 at 15:46

46 Has it not bothered anyone that despite the STL being "object-oriented", `.find()` is still *not* a member function of `std::vector`, as you'd expect it should be? I wonder if this is somehow a consequence of templating. – bobobobo Dec 7 '12 at 2:33

46 @bobobobo: OOP has nothing to do with members vs. non-members. And there is a widespread school of thought that if something does not have to be a member, or when it does not give any advantage when implemented as a member, than it should not be a member; `std::vector<>::find()` would not give any advantage, nor is it needed, therefore, no, it should not be a member. See also en.wikipedia.org/wiki/Coupling_%28computer_programming%29 – Sebastian Mach Feb 4 '13 at 13:54

27 This is so much worse than Python's `item in vector` .... – Claudiu Apr 3 '14 at 19:35

As others have said, use the STL `find` or `find_if` functions. But if you are searching in very large vectors and this impacts performance, you may want to sort your vector and then use the `binary_search`, `lower_bound`, or `upper_bound` algorithms.

edited Oct 20 '12 at 19:27 user283145 — answered Feb 20 '09 at 22:26 Brian Neal **20.8k** 4 40 56

2 Good answer! Find is always o(n). lower_bound is o(log(n)) if used with random-access iterators. – Stephen Edmonds Jul 8 '09 at 19:54

12 Sorting is O(nlogn) though, so it's worth only if you're doing more than O(logn) searches. – liori Jun 15 '14 at 0:48

4 @liori True it depends on your usage patterns. If you only need to sort it once, then repeatedly do many searches it can save you. – Brian Neal Jun 17 '14 at 16:24

Use find from the algorithm header of stl.I've illustrated its use with int type. You can use any type you like as long as you can compare for equality (overload == if you need to for your custom class).

```
#include <algorithm>
#include <vector>

using namespace std;
int main()
{
    typedef vector<int> IntContainer;
    typedef IntContainer::iterator IntIterator;

    IntContainer vw;

    //...

    // find 5
    IntIterator i = find(vw.begin(), vw.end(), 5);

    if (i != vw.end()) {
        // found it
    } else {
        // doesn't exist
    }

    return 0;
}
```

edited Apr 29 at 14:37 isanae **2,115** 1 5 26 — answered Feb 20 '09 at 22:06 m-sharp **8,195** 1 16 24

2 Depending on the OP's needs, find_if() could also be appropriate. It allows to search using an arbitrary predicate instead of equality. – Éric Malenfant Feb 20 '09 at 22:12

Oops, saw your comment too late. The answer I gave also mentions find_if. – Frank Feb 20 '09 at 22:19

If your vector is not ordered, use the approach MSN suggested:

```
if(std::find(vector.begin(), vector.end(), item)!=vector.end()){
      // Found the item
}
```

If your vector is ordered, use binary_search method Brian Neal suggested:

```
if(binary_search(vector.begin(), vector.end(), item)){
     // Found the item
}
```

binary search yields O(log n) worst-case performance, which is way more efficient than the first approach. In order to use binary search, you may use qsort to sort the vector first to guarantee it is ordered.

edited Jul 9 '16 at 12:07 answered Nov 23 '12 at 8:58

hfossli **15.4k** 7 85 105 Think Recursively **1,053** 11 18

3 Don't you mean `std::sort` ? `qsort` is very inefficient on vectors.... see: stackoverflow.com/questions/12308243/… – Jason R. Mick Aug 16 '13 at 1:57

Binary search will perform better for larger containers, but for small containers a simple linear search is likely to be as fast, or faster. – BillT Mar 31 at 14:07

I use something like this...

```
#include <algorithm>


template <typename T>
const bool Contains( std::vector<T>& Vec, const T& Element )
{
    if (std::find(Vec.begin(), Vec.end(), Element) != Vec.end())
        return true;

    return false;
}

if (Contains(vector,item))
   blah
else
   blah
```

...as that way it's actually clear and readable. (Obviously you can reuse the template in multiple places).

edited Jan 9 '14 at 23:11 answered Sep 4 '13 at 16:34

Community♦ **1** 1 Andy Krouwel **444** 4 9

and you can make it work for lists or vectors by using 2 typenames – Erik Aronesty Mar 3 '15 at 15:36

@ErikAronesty you can get away with 1 template argument if you use `value_type` from the container for the element type. I've added an answer like this. – Martin Broadhurst Feb 11 '16 at 21:40

Bear in mind that, if you're going to be doing a lot of lookups, there are STL containers that are better for that. I don't know what your application is, but associative containers like std::map may be worth considering.

std::vector is the container of choice unless you have a reason for another, and lookups by value can be such a reason.

answered Feb 20 '09 at 22:42

David Thornley **46.5k** 8 75 135

Even with lookups by value the vector can be a good choice, as long as it is sorted and you use binary_search, lower_bound or upper_bound. If the contents of the container changes between lookups, vector is not very good because of the need to sort again. – Renze de Waal Feb 20 '09 at 22:49

Use the STL find function.

Keep in mind that there is also a find_if function, which you can use if your search is more complex, i.e. if you're not just looking for an element, but, for example, want see if there is an element that fulfills a certain condition, for example, a string that starts with "abc". (`find_if` would give you an iterator that points to the first such element).

answered Feb 20 '09 at 22:18

Frank **24.3k** 68 187 284

You can try this code:

```
#include <algorithm>
#include <vector>

// You can use class, struct or primitive data type for Item
struct Item {
    //Some fields
};
typedef std::vector<Item> ItemVector;
typedef ItemVector::iterator ItemIterator;
//...
ItemVector vtItem;
//... (init data for vtItem)
```

```
Item itemToFind;
//...

ItemIterator itemItr;
itemItr = std::find(vtItem.begin(), vtItem.end(), itemToFind);
if (itemItr != vtItem.end()) {
    // Item found
    // doThis()
}
else {
    // Item not found
    // doThat()
}
```

answered Apr 28 '12 at 15:29
TrungTN
1,584 1 8 4

In C++11 you can use `any_of`. For example if it is a `vector<string> v;` then:

```
if (any_of(v.begin(), v.end(), bind2nd(equal_to<string>(), item)))
   do_this();
else
   do_that();
```

edited Feb 9 '16 at 0:18
answered Aug 11 '15 at 4:15
Deqing
5,507 5 43 71

Here's a function that will work for any Container:

```
template <class Container>
const bool Contains(Container& container, const typename Container::value_type&
element)
{
    return std::find(container.begin(), container.end(), element) !=
container.end();
}
```

Note that you can get away with 1 template parameter because you can extract the `value_type` from the Container. You need the `typename` because `Container::value_type` is a dependent name.

answered Feb 11 '16 at 21:38
Martin Broadhurst
6,342 1 16 26

Note that this is sometimes a bit too broad - it'd work for std::set for example, but give terrible performance compared to the find() member function. I've found it best to add a specialisation for containers with a faster search (set/map, unordered_*) – Andy Krouwel Nov 2 '16 at 12:01

If you wanna find a string in a vector:

```
    struct isEqual
{
    isEqual(const std::string& s): m_s(s)
    {}

    bool operator()(OIDV* l)
    {
        return l->oid == m_s;
    }

    std::string m_s;
};
struct OIDV
{
    string oid;
//else
};
VecOidv::iterator itFind=find_if(vecOidv.begin(),vecOidv.end(),isEqual(szTmp));
```

answered May 31 '13 at 13:55
Gank
2,669 1 31 32

You can use count too. It will return the number of items present in a vector.

```
int t=count(vec.begin(),vec.end(),item);
```

edited Apr 17 '15 at 6:18
Pang
5,703 14 48 81

answered Mar 11 '15 at 10:00
Aditya
75 7

2 `find` is faster than `count` , because it doesn't keep on counting after the first match. – Camille Goudeseune Aug 16 '15 at 20:27

```
template <typename T> bool IsInVector(T what, std::vector<T> * vec)
{
    if(std::find(vec->begin(),vec->end(),what)!=vec->end())
        return true;
    return false;
}
```

answered Jan 26 '14 at 17:55
user3157855
85 9

You can use the `find` function, found in the `std` namespace, ie `std::find` . You pass the `std::find` function the `begin` and `end` iterator from the vector you want to search, along with the element you're looking for and compare the resulting iterator to the end of the vector to see if they match or not.

```
std::find(vector.begin(), vector.end(), item) != vector.end()
```

You're also able to dereference that iterator and use it as normal, like any other iterator.

answered Jun 15 '14 at 0:36
TankorSmash
7,125 1 33 71

Another sample using C++ operators.

```
#include <vector>
#include <algorithm>
#include <stdexcept>

template<typename T>
inline static bool operator ==(const std::vector<T>& v, const T& elem)
{
  return (std::find(v.begin(), v.end(), elem) != v.end());
}

template<typename T>
inline static bool operator !=(const std::vector<T>& v, const T& elem)
{
  return (std::find(v.begin(), v.end(), elem) == v.end());
}

enum CODEC_ID {
  CODEC_ID_AAC,
  CODEC_ID_AC3,
  CODEC_ID_H262,
  CODEC_ID_H263,
  CODEC_ID_H264,
  CODEC_ID_H265,
  CODEC_ID_MAX
};

void main()
{
  CODEC_ID codec = CODEC_ID_H264;
  std::vector<CODEC_ID> codec_list;

  codec_list.reserve(CODEC_ID_MAX);
  codec_list.push_back(CODEC_ID_AAC);
  codec_list.push_back(CODEC_ID_AC3);
  codec_list.push_back(CODEC_ID_H262);
  codec_list.push_back(CODEC_ID_H263);
  codec_list.push_back(CODEC_ID_H264);
  codec_list.push_back(CODEC_ID_H265);

  if (codec_list != codec)
  {
    throw std::runtime_error("codec not found!");
  }

  if (codec_list == codec)
  {
    throw std::logic_error("codec has been found!");
  }
}
```

answered May 13 '16 at 8:57
Valdemar_Rudolfovich
1,134 8 7

2 I wouldn't recommend abusing operator overloading in such a way. – Leon Jun 8 '16 at 16:13

Leon, I agree with you, semantically it isn't correct. I use it to make unit tests more clearly. – Valdemar_Rudolfovich Jun 10 '16 at 12:34

With boost you can use `any_of_equal`:

```
#include <boost/algorithm/cxx11/any_of.hpp>

bool item_present = boost::algorithm::any_of_equal(vector, element);
```

answered Sep 27 '16 at 16:02
Mikhail
10.6k 3 27 86

The brute force approach (again presuming int as the stored type):

```
int value_to_find;
vector<int> cont;
vector<int>::const_iterator found = cont.find(value_to_find);
if (found != cont.end()) {
    do_this();
} else {
    do_that();
}
```

If you're doing many lookups in large vectors, this can be inefficient. You may want to cache your results in order to avoid doing the same search twice (assumes int as the stored type):

```
int value_to_find;
vector<int> cont;                          // main container
map<int, size_t> contPos;                  // position cache

// first see if the value is in cache
map<int, size_t>::const_iterator foundCache = contPos.find(value_to_find);
if (foundCache != contPos.end()) {
    do_this();
}
// not in cache, now do brute force search
vector<int>::const_iterator found = cont.find(value_to_find);
if (found != cont.end()) {
    // cache the value with its position
    contPos[value_to_find] = found - cont.begin();

    do_this();
} else {                                   // in neither
    do_that();
}
```

edited Feb 24 '09 at 22:54

answered Feb 21 '09 at 0:37
David M. Miller
104 5

24 `std::vector` doesn't have a find member function. – Brian Neal Apr 21 '10 at 17:54