How do you iterate backwards through an STL list?

I'm writing some cross-platform code between Windows and Mac.

If list::end() "returns an iterator that addresses the location succeeding the last element in a list" and can be checked when traversing a list forward, what is the best way to traverse backwards?

This code workson the Mac but not on Windows (can't decrement beyond first element):

```
list<DVFGfxObj*>::iterator iter = m_Objs.end();
for (iter--; iter!=m_Objs.end(); iter--)// By accident discovered that the iterator
is circular ?
{
}
```

this works on Windows:

```
list<DVFGfxObj*>::iterator iter = m_Objs.end();
    do{
        iter--;
    } while (*iter != *m_Objs.begin());
```

Is there another way to traverse backward that could be implemented in a for loop?

c++ list stl iterator traversal

edited Oct 9 '08 at 20:13
KTC
7,777 5 23 36

asked Oct 9 '08 at 19:57
AlanKley
1,751 6 25 39

1 It would only be an accident of implementation that your first example (circular iterator, comparing against end()) would work. – Justsalt Oct 9 '08 at 20:23

5 Answers

Use reverse_iterator instead of iterator. Use rbegin() & rend() instead of begin() & end().

Another possibility, if you like using the BOOST_FOREACH macro is to use the BOOST_REVERSE_FOREACH macro introduced in Boost 1.36.0.

edited Oct 9 '08 at 20:20

answered Oct 9 '08 at 19:58
Ferruccio
70.7k 31 179 269

The documentation for iterator and reverse_iterator are almost the same. an iterator is bidirectional so what's the diff? – AlanKley Oct 9 '08 at 20:00

2 The difference is that you still do "++Iter" to increment the iterator, vs. "--Iter". Or am i wrong? – steffenj Oct 9 '08 at 20:03

No, you're right which is a little odd to increment to go backwards but also makes sense. Though the reverse_iterator seems unnecessary given that iterator is bidirectional. The docs for reverse_iterator says it acts on a reversed list; surely it doesn't reverse the list internaly first. – AlanKley Oct 9 '08 at 20:08

@AlanKey: If you know you're dealing with a list, you might just want to decrement the normal iterator. Reverse iterators come into their own when you're writing generic code - it doesn't need to do anything special to go through a collection backwards - it just needs to be given reverse iterators – Michael Burr Oct 9 '08 at 21:56

2 Not all "forward" iterators are "bidirectional". It depends on the collection class. – Jesse Chisholm Oct 17 '13 at 3:05

The best/easiest way to reverse iterate a list is (as already stated) to use reverse iterators rbegin/rend.

However, I did want to mention that reverse iterators are implemented storing the "current" iterator position off-by-one (at least on the GNU implementation of the standard library).

This is done to simplify the implementation, in order for the range in reverse to have the same semantics as a range forward [begin, end) and [rbegin, rend)

What this means is that dereferencing an iterator involves creating a new temporary, and then decrementing it, *each and every time*:

```
  reference
  operator*() const
  {
_Iterator __tmp = current;
return *--__tmp;
  }
```

Thus, *dereferencing a reverse_iterator is slower than an normal iterator.*

However, You can instead use the regular bidirectional iterators to simulate reverse iteration yourself, avoiding this overhead:

```
for ( iterator current = end() ; current != begin() ; /* Do nothing */ )
{
    --current; // Unfortunately, you now need this here
    /* Do work */
    cout << *current << endl;
}
```

Testing showed this solution to be ~5 times faster *for each dereference* used in the body of the loop.

Note: Testing was not done with the code above, as that std::cout would have been the bottleneck.

Also Note: the 'wall clock time' difference was ~5 seconds with a std::list size of 10 million elements. So, realistically, unless the size of your data is that large, just stick to rbegin() rend()!

edited Oct 22 '08 at 3:01

answered Oct 21 '08 at 20:33

mmocny
5,803 6 28 43

Looking at this again, Probably you want to just initialize with current = --end(); and leave the increment step inside the for loop. This would also guard against empty array which my version above doesn't. I'll leave the original posted for now since I haven't tested. – mmocny Oct 22 '13 at 20:45

I don't think that this would work unless you also change your loop condition. Otherwise you'll be missing the first item (cause the loop wont execute if `current == begin()`) – Griddo Jul 21 '14 at 11:53

First line of the for loop does a decrement, so yes it would, I think. Just write a quick test to try it! Anyway, I no longer think this is the nicest way to use this idiom, and some recent tests I ran no longer show a marked speed improvement to reverse iterators under full optimization. However, its worth still noting whats happening under the hood and testing accordingly! – mmocny Jul 21 '14 at 13:15

1 I intended to comment on your comment, not the answer, but SO removed the @-part. The code in your answer works perfectly fine, however, I agree it might not be the best out there ;) Edit: quick test – Griddo Jul 22 '14 at 5:56

You probably want the reverse iterators. From memory:

```
list<DVFGfxObj*>::reverse_iterator iter = m_Objs.rbegin();
for( ; iter != m_Objs.rend(); ++iter)
{
}
```

edited Oct 9 '08 at 20:07

answered Oct 9 '08 at 19:59

Anthony Cramp
2,633 4 16 25

1 Thanks that sounds good. But also seems a waste to create a special reverse_iterator when iterator is suppose to be bi-directional – AlanKley Oct 9 '08 at 20:02

it should be "...>::reverse_iterator iter = ..." – steffenj Oct 9 '08 at 20:04

@AlanKley I think the for loop you've put in your question is fine. The reason why I think it works is because

the .end() member function returns a sentinel value that is assigned as the value of the next pointer from the last element and also the prev pointer on the first element. – Anthony Cramp Oct 9 '08 at 20:30

Oops, reread your question ... doesn't work on Windows. I tested the code on a Mac as well. – Anthony Cramp Oct 9 '08 at 20:31

This should work:

```
list<DVFGfxObj*>::reverse_iterator iter = m_Objs.rbegin();
for (; iter!= m_Objs.rend(); iter++)
{
}
```

answered Oct 9 '08 at 20:05

steffenj
4,529 10 28 40

As already mentioned by Ferruccio, use reverse_iterator:

```
for (std::list<int>::reverse_iterator i = s.rbegin(); i != s.rend(); ++i)
```

answered Oct 9 '08 at 20:05

ejgottl
2,393 11 16