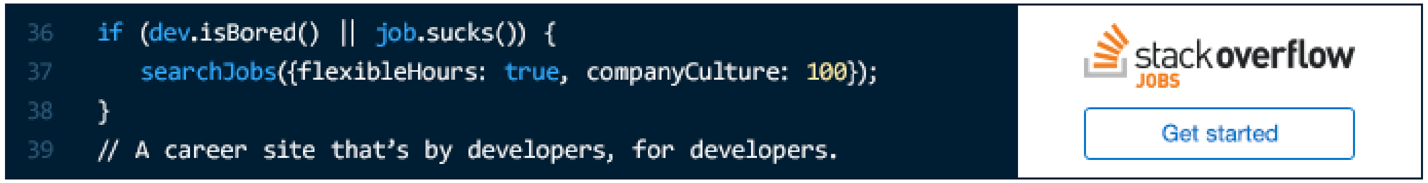
Iterating backward



Suppose I have a `vector<int> myvec` and I want to loop through all of the elements in reverse. I can think of a few ways of doing this:

```
for (vector<int>::iterator it = myvec.end() - 1; it >= myvec.begin(); --it)
{
    // do stuff here
}

for (vector<int>::reverse_iterator rit = myvec.rbegin(); rit != myvec.rend();
++rit)
{
    // do stuff here
}
```

```
for (int i = myvec.size() - 1; i >= 0; --i)
{
    // do stuff here
}
```

So my question is when should I use each? Is there a difference? I know that the first one is dangerous because if I pass in an empty vector, then `myvec.end() - 1` is undefined, but are there any other hazards or inefficiencies with this?

c++ loops iterator

asked Mar 30 '10 at 21:33
MBennett
431 2 10

6 Answers

The `reverse_iterator` version shows intent and works across all containers, regardless of their contents.

The first has the deficiency you describe. It also uses `>=`, which won't work for non-random-access iterators.

The third has the problem that `i` is an `int`. It won't be able to hold as much as `size()` could potentially return. Making it unsigned works (`vector<int>::size_type`), but then we have the same problem as solution one. (`0U - 1` -> `Funky terminating checks` -> `:|`)

edited Mar 30 '10 at 21:42

answered Mar 30 '10 at 21:36
GManNickG
250k 32 367 482

1 "It also uses >=, which won't work for non-random-access iterators.". That's OK, neither does `end() -1` ;-) – Steve Jessop Mar 30 '10 at 21:43

+1. When a library provides functionality explicitly designed to perform a task, you usually should use it when trying to perform the same task. – Brian Mar 30 '10 at 21:44

@Steve: Heh, true. I was thinking `end()--` which works for bidirectional iterators. – GManNickG Mar 30

'10 at 21:46

Generally none of the above. Instead, you should usually sit back and relax for a few seconds, figure out which *algorithm* you want to apply, and forget about writing a loop yourself at all. Chances are that you'll use `reverse_iterator` with it, but depending on what you're trying to accomplish that won't always be the case (e.g., see `std::copy_backwards`).

answered Mar 30 '10 at 21:42

Jerry Coffin
336k 33 383 791

Personally, I'd go with the second one.

As you indicate the first one requires you to wrap the loop in an `if (!myvec.empty())` to avoid undefined behaviour.

For the last one, you should probably be using a `vector<int>::size_type` or `size_t` , in which case the `>= 0` is wrong, you would need to do `!= (size_t)-1` or similar.

The `reverse_iterator` version is, therefore, cleaner.

answered Mar 30 '10 at 21:37

Charles Bailey
407k 60 496 569

As to the first version, you will also inevitably end up decrementing the `begin()` iterator at the end of a loop (undefined behavior).

The `reverse_iterator` was made for this.

The third might work somewhat better if you used the somewhat more controversial form:

```
for (size_t i = vec.size(); i --> 0; )
```

This could be an idiom if people would stop resisting. It uses a suitable counter type (unsigned), and contains mnemonics for easy memorizing and recognizing.

answered Mar 30 '10 at 21:39

UncleBens
31k 5 38 81

1 Hey, the goes-to operator! – GManNickG Mar 30 '10 at 21:43

+1 for the warning against *decrementing the begin() iterator* – Alexandre Jasmin Mar 30 '10 at 21:58

Always use the second. The first you ruled out yourself, and the third doesn't work for lists and such.

answered Mar 30 '10 at 21:36

Björn Pollex
50.5k 16 133 219

There's a fourth option (not necessarily a good option, but it exists). You can use bidirectional/random access iterators in a fashion that mimics how reverse iterators are implemented to avoid the problem with `myvec.end()-1` on an empty iterator:

```
for (vector<int>::iterator it = myvec.end(); it != myvec.begin(); --it)
{
    // convert the loop controlling iterator to something that points
    //  to the item we're really referring to

    vector<int>::iterator true_it = it;
    --true_it;
```

```
        // do stuff here
        //  but always dereference `true_it` instead of `it`
        //  this is essentially similar to the way a reverse_iterator
        //  generally works

        int& x = *true_it;
    }
```

or even:

```
    for (vector<int>::iterator it = myvec.end(); it != myvec.begin();)
    {
        // decrement `it` before the loop executes rather than after
        //  it's a bit non-idiomatic, but works
        --it;

        int& x = *it;

        // do stuff...
    }
```

Like I said, this is not necessarily a good option (I think Jerry Coffin's answer is the approach you should look to first), but I think it's of interest since it shows how reverse iterators work behind the scenes - and it avoids having to convert a reverse_iterator to a iterator for those times when you might want to use the iterator with something that won't accept a `reverse_iterator` (converting `reverse_iterator` s to `iterator` s always seems to make my head hurt, so I'll often avoid `reverse_iterators` to avoid headaches). For example, if you want to call insert() for the location a reverse iterator is referring to:

```
// if `it` is a reverse iterator, a call to insert might have to look something
like:

myvec.insert( --(it.base()), 42 );  // assume that you know the current vector
capacity
                                    //  will avoid a reallocation, so the loop's
                                    //  iterators won't be invalidated

// if `it` is a normal iterator...

myvec.insert( it, 42 );
```

answered Mar 30 '10 at 22:38

Michael Burr
251k 33 377 596