

Re: std::vector : begin, end and insert – Using Objects instead of ints

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2007-05/msg00962.html>

- *From:* "Doug Harrison [MVP]" <dsh@xxxxxxxx>
 - *Date:* Tue, 15 May 2007 13:39:59 -0500
-

On Tue, 15 May 2007 17:29:04 +0100, Gerry Quinn <gerryq@xxxxxxxx> wrote:

Method 1 (works for both vectors and lists):

```
vector< CPoint > vec;
// put some CPoints in it

vector< CPoint >::iterater it;

for ( it = vec.begin(); it != vec.end(); it++ )
{
    CPoint & pt = *it;
    // iterators are overloaded pointers
    // they use pointer syntax
}
```

Method 2 (if you used a vector because you want something like an array, only better):

```
vector< CPoint > vec;
// put some CPoints in it

for ( int i = 0; i < vec.size(); i++ )
{
    CPoint & pt = vec[ i ];
    // or alternatively
    CPoint & something = vec.at( i );
}
```

Method 2 works on vectors and similar collection classes, but not on lists, which only have iterators. But it makes for more readable code when you're using vectors.

Get the above sorted out in your head before doing anything

Re: `std::vector` : `begin`, `end` and `insert` – Using Objects instead of ints
complicated.

Some things to note:

1. When there's a choice, prefer preincrement to postincrement, because `pre` doesn't require creation of a temporary object to return the original value. That is, use `++it` instead of `it++`. (I know, K&R taught the opposite, but this is C++, which itself would have been better named `++C`. <g>)
2. While pointers are (random access) iterators, iterators are not "overloaded pointers", and it is better to say that iterators are "an abstraction of pointers". Indeed, iterators typically are not pointers at all but class types. Even `std::vector::iterator` is a class type in VC7 and later.
3. It's arguable that the indexing approach is more readable than the iterator approach to a loop. If I were to guess, I'd say most experienced STL programmers use iterator loops, because that approach works across all containers, and because STL algorithms are designed in terms of iterator ranges, so most people end up thinking in terms of iterators by default.
4. The `vector::at` function is not really an "alternative" to `operator[]`, because the former checks its argument and throws an exception if it's out of bounds. The latter doesn't perform any error-checking, and thus there are major differences in performance and behavior between the two.
5. Instead of saying `std::list` "only has iterators", it would be more accurate to say that `std::vector` has random access iterators, while `std::list` has bidirectional iterators, which is why you can use `operator[]` on vectors.
6. If you're really nuts about efficiency, don't use `vec.end()` (or `vec.size()`) in your loop condition. Instead save its value to a (const) variable and use it instead.

—
Doug Harrison
Visual C++ MVP
.