C++ Container Iterator Invalidation

April 5, 2011 by Louis Feng 7 Comments

In a recent coding review a coworker pointed out in my code that modifying a STL list might invalidate the iterators pointing to the elements in the list. That's a good observation. I know STL map will not invalidate iterators when elements are inserted or erased from the container (except the elements that's modified). But for STL list, I can't be so certain myself. Considering how a list is usually implemented (CS 101) with pointers, it shouldn't be a problem to maintain the iterator ++ operation properly even after insert or erase. But my experience of using different compilers suggest that only God knows how different compilers implement their STL. It could have gone either way! Plus, what about other types of containers? STL vector, deque, etc.?

To settle this once and for all, I looked up the Standard for Programming Language C++. Chapter 23. The information below are extracted from the Standard (working draft 2010 – 3225). **Red colored** operations are ones might invalidate iterators.

General Container Requirements

- Unless otherwise specified (either explicitly or by defining a function in terms of other functions), invoking a container member function or passing a container as an argument to a library function shall not invalidate iterators to, or change the values of, objects within that container.

deque

- insert(), push_front(), push_back()
  - An insertion in the middle of the deque invalidates all the iterators and references to elements of the deque. An insertion at either then end of the deque invalidates all the iterators to the deque, but has no effect on the validity of references to the elements of the deque.
- erase()
  - An erase operation that erases the last element of a deque invalidates only the past-the-enditerator and all iterators and references to the erased elements. An erase operation that erases the firstelement of a deque but not the last element invalidates only the erased elements. An erase operationthat erases neither the first element nor the last element of a deque invalidates the past-the-end iteratorand all iterators and references to all the elements of the deque.

list

- insert(), push_front(), push_back()
  - Does not affect the validity of iterators and references.
- erase(), pop_front(), pop_back(), clear()
  - Invalidates only the iterators and references to the erased elements.

vector

Archives

- July 2015
- April 2015
- February 2012
- July 2011
- June 2011
- April 2011
- March 2011

Tags

book C++ clang cocoa container diff GDB git iOS ipad iphone iterator laptop list llvm map memory leak mobile nib software engineering stl Surface 3 Surface Pro 3 UITableViewController UIViewController vector vectorization view controller windows workstation

- insert(), push_back()
  - Causes reallocation if the new size is greater than the old capacity. If no reallocation happens, all the iterators and references before the insertion point remain valid. (It doesn't say in the standard, but if reallocation happens, all iterators and references are probably invalid).
- erase()
  - Invalidates iterators and references at or after the point of erase.

Associative Containers: set, multiset, map and multimap

- The insert members shall not affect the validity of iterators and references to the container, and the erase members shall invalidate only iterators and references to the erased elements.

There you have it.

Filed Under: Blog, Code Tagged With: C++, container, iterator, list, map, stl, vector

About Louis Feng
I have been a computer graphics enthusiast and researcher for many years. My interests has broadened to include mobile, high performance computing, machine learning, and computer vision.

Comments

Roger says:
June 13, 2011 at 12:05 am

Mr Feng,

I have read your post above about iterators used for maps and how they do not give problems when erasing elements of the map. I am struggling with the code below – I get a seg fault when I re-enter the for loop after erasing an element of the map. Can you spot anything else in my code that could be causing this to occur, or could it be that the iterator is confused after erasing the member?

Kind regards.

```
// function to remove unwanted elements from a map
void FindPath::keep_these(map dist_orig, set& Movieset){ // dist_other is the dist of orig FindPath object
map::iterator itm;
for (itm = dist.begin(); itm != dist.end(); itm++){
if (dist_orig.find((*itm).first) == dist_orig.end() ||
Movieset.find((*itm).first) != Movieset.end())
dist.erase(itm);
}
}
```

REPLY

Louis Feng says:
June 13, 2011 at 1:28 am

I'm not sure where dist comes from in your function. It's not declared. If you

erase itm, then that iterator is invalidated as I pointed out in the post and you can't increment it in the for statement. You should try this form:

```
TestMap::iterator iter = testMap.begin();
while(iter != testMap.end()) {
    if (shouldErase()) {
        testMap.erase(iter++);
    }
    else {
        ++iter;
    }
}
```

The key here is the post increment inside erase(iter++). It will properly increment the iterator to the next element but ensure the current one is erased.

REPLY

olin says:

July 27, 2012 at 1:48 am

General item erasing for STL collections is "iter = collection.erase(iter)". Post-incrementing works only for particular types of collections.

REPLY

John says:

September 26, 2012 at 1:49 am

With GCC's C++ library, std::map::erase (oddly) doesn't return an iterator, like .e.g std:.vector does. The erase(iter++) technique mentioned is a good work-around.

REPLY

nick says:

January 11, 2015 at 10:11 am

Actually, std::map does invalidate the iterator to the erased element:

http://stackoverflow.com/questions/16178898/weird-seg-fault-when-erasing-from-a-map?lq=1

REPLY

Louis Feng says:

April 24, 2015 at 1:34 pm

That's correct, as I mentioned in the last line of the post.

REPLY

Trackbacks

Iterator Invalidation | Mens supra materia says:

July 25, 2014 at 1:32 am

[...] (*) Detailed by Louis Feng here. [...]

REPLY

Speak Your Mind

Name *

Email *

Website

CAPTCHA Code *

POST COMMENT

RETURN TO TOP OF PAGE