Tampereen yliopisto

Invalidation in containers

COMP.CS.300 Data structures and algorithms 1

Matti Rintala (matti.rintala@tuni.fi)

Invalidation

- Invalidated iterator no longer refers to a (correct) position after insert/remove
- *Do not use* an invalidated iterator (assigning a new position is ok)
- If used, result is *undefined* (crash/messed up data/???)

Invalidation

```
vector<int> v={ 1,2,3,4};
auto i = v.begin();
auto j = i+1; // next after i
v.erase(i);
*j = 3; // !!! j invalidated!
```

Invalidation and choosing a container

- Different containers have different rules for invalidation
- Another selection criteria in addition to performance (often a compromise)
- vector and deque: rules complicated
- unordered_map/set: safe for erasing, insertion invalidates
- map/set and (forward_)list almost safe

Tampereen yliopisto

Invalidation and choosing a container

Container	After insertion	After erase	Note
vector	Invalidated!	-	*Capacity* changed
	Ok	Ok *	*Before* insertion position
	Invalidated!	Invalidated!	*After* insertion position
deque	Invalidated!	Ok *	Insert/erase of *1./last*
	Invalidated!	Invalidated!	Insert/erase of *rest*
(forward_)list	Ok	Ok *	
(multi)map/set	Ok	Ok *	
unordered_(multi) map/set	Invalidated!	-	*Rehash* occurred
	Ok	Ok *	

How to notice invalidation

- Careful planning!
- Some compilers have STL-debug features

  Gcc: -D_GLIBCXX_DEBUG
  -D_GLIBCXX_DEBUG_PEDANTIC
- Program crashes: debugger tells where?
- Program gets messed up: debugger/printouts

Invalidation, pointers and indices

- Any indicator of position may get invalidated!
- Pointer to element: element gets moved in memory
- Index to element: Insertion or removal before element
- cppreference.com has a more comprihensive table