How useful is the hint passed to the std::unordered_… collections?

devblogs.microsoft.com/oldnewthing/20241028-00

October 28, 2024

A little while ago, we learned about the value of the `hint` to the C++ ordered associative containers `std::map`, `std::multimap`, `std::set`, and `std::multiset`. But what about the unordered associative containers `std::unordered_map`, `std::unordered_multimap`, `std::unordered_set`, and `std::unordered_multiset`?

The answer varies by implementation.

For Microsoft's STL, the hint is used by `_Find_Hint`:[1] If the hint's key is equivalent to the inserted item's key, then we keep the hint. Otherwise, we ignore it. For containers that do not permit duplicate keys, a matching hint provides the exact node to update, avoiding the need to do a search for it. The code can just update the value or do nothing, depending on the method being implemented: `insert` does nothing, whereas `insert_or_assign` updates the value. For containers that allow duplicate keys, a matching hint is used to insert the new element without performing a search.

The clang libcxx implementations outright ignores the hint for containers that do not permit duplicate keys. For containers that permit duplicate keys, a hint whose key is equivalent to the item being inserted is used to insert the new element without performing a search.

The gcc libstdc++ implementation is mostly the same as the clang implementation: It ignores the hint for containers that do not permit duplicate keys, and it uses the hint for containers that permit duplicate keys provided the hint's key matches that of the inserted item. As a special case, for containers that permit duplicate keys that are "small" and have "slow" hash functions, it uses the hint as the starting point for a linear search through the hash table contents looking for a matching key. The idea is that if the hash function is "slow", then it's faster to just compare keys instead of going to effort of hashing them.

Wait, what do “small” and “slow” mean? The gcc libstdc++ implementation considers a container to be “small” if it has at most 20 elements, and it considers all hash functions by default to be “fast”, except for `long double`.

To summarize:

	Must keys be unique?	
	Yes	**No**
msvc/STL	Hint used if matches	Hint used if matches
clang/libcxx	Hint ignored	Hint used if matches
gcc/libstdc++ (large or fast)	Hint ignored	Hint used if matches
gcc/libstdc++ (small and slow)	Hint ignored	Hint used

[1] In the STL code, you should know that the `_Traitsobj::operator()` returns `true` if the keys are *unequal*.