P0000R1
Add more std::hash specializations

New Proposal, 2018-12-04

Authors:

Alexander Zaitsev (Solarwinds) zamazan4ik@tut.by

Anthony Polukhin (Yandex Taxi) antoshkka@gmail.com

Project:

ISO/IEC JTC1/SC22/WG21 14882: Programming Language — C++

Audience:

LEWGI, LEWG, LWG

Source:

https://github.com/ZaMaZaN4iK/ConfsANDProps/blob/master/Proposals/complex_hash.bs

---

Abstract

In Standard library we already have std::hash specializations for some classes like std::string. Unfortunately, we have no specializations for a lot of other classes from Standard Library like std::vector, std::array, etc. - but we have possbility to calculate hash from these containers. People who need hash calucations for such containers must use Boost.Hash functions or write std::hash specialization manually. This proposal adds std::hash specializations for different containers from Standard Library. Addresses an issue LWG #1025.

Table of Contents

§ 1. Design decisions

- Proposed hash specializations give the same hash value for different containers with the same content. This may be useful for users to compare different containers in a fast maner. This also allows users to have heterogeneous searches in unordered maps by keays of different type.
- We do not enable `hash` for unordered_set, unordered_map, unordered_multiset, unordered_multimap because of the hashing collisions and buckets count. Position of the elment depends on those two factors, which leads to different hashes for containers with the same content.
- We do not enable hash for `stack` and `queue` adapters for now.

§ 2. Proposed wording

Add a new Section "19.4.6, Hash support [pair.hash]", with following content:

```
template<typename A, typename B>
  struct hash<pair<A, B>>;
```

Enabled if specializations `hash<remove_const_t<A>>` and `hash<remove_const_t<B>>` are both enabled, and disabled otherwise.

Let `PAIR` denote a `pair` type, `x` denote a value of type `PAIR`. For enabled specialization `hash<PAIR>` the following holds: `hash<PAIR>{}(x) == hash<decltype(tuple{x})>{}(x)`.

Add a new Section "19.5.3.11, Hash support [tuple.hash]", with following content:

```
template<typename... T>
  struct hash<tuple<T...>>;
```

Enabled if specialization `hash<remove_const_t<U>>` is enabled for every template argument `U` in the parameter pack, and disabled otherwise.

Add a new Section "21.3.7.7, Hash support [array.hash]", with following content:

```
template<typename T, std::size_t N>
  struct hash<array<T, N>>;
```

Enabled if specialization `hash<remove_const_t<T>>` is enabled, and disabled otherwise.

Add a new Section "21.3.8.6, Hash support [deque.hash]", with following content:

```
template<typename T, typename Allocator>
  struct hash<deque<T, Allocator>>;
```

Enabled if specialization `hash<remove_const_t<T>>` is enabled, and disabled otherwise.

Add a new Section "21.3.9.8, Hash support [forward_list.hash]", with following content:

```
template<typename T, typename Allocator>
  struct hash<forward_list<T, Allocator>>;
```

Enabled if specialization `hash<remove_const_t<T>>` is enabled, and disabled otherwise.

Add a new Section "21.3.10.7, Hash support [list.hash]", with following content:

```
template<typename T, typename Allocator>
  struct hash<list<T, Allocator>>;
```

Enabled if specialization `hash<remove_const_t<T>>` is enabled, and disabled otherwise.

Add a new Section "21.3.11.7, Hash support [vector.hash]", with following content:

```
template<typename T, typename Allocator>
  struct hash<vector<T, Allocator>>;
```

Enabled if specialization `hash<remove_const_t<T>>` is enabled, and disabled otherwise.

Add a new Section "21.4.4.6, Hash support [map.hash]", with following content:

```
template<typename Key, typename T, typename Compare, typename Allocator>
  struct hash<map<Key, T, Compare, Allocator>>;
```

Enabled if specialization if specializations `hash<remove_const_t<Key>>` and `hash<remove_const_t<T>>` are both enabled, and disabled otherwise.

Add a new Section "21.4.5.5, Hash support [multimap.hash]", with following content:

```
template<typename Key, typename T, typename Compare, typename Allocator>
  struct hash<multimap<Key, T, Compare, Allocator>>;
```

Enabled if specialization if specializations `hash<remove_const_t<Key>>` and `hash<remove_const_t<T>>` are both enabled, and disabled otherwise.

Add a new Section "21.4.6.4, Hash support [set.hash]", with following content:

```
template<typename Key, typename Compare, typename Allocator>
  struct hash<set<Key, Compare, Allocator>>;
```

Enabled if specialization if specializations `hash<remove_const_t<Key>>` is enabled, and disabled otherwise.

Add a new Section "21.4.7.4, Hash support [multiset.hash]", with following content:

```
template<typename Key, typename Compare, typename Allocator>
  struct hash<multiset<Key, Compare, Allocator>>;
```

Enabled if specialization if specializations `hash<remove_const_t<Key>>` is enabled, and disabled otherwise.

Add a new Section "25.7.2.9, Hash support [valarray.hash]", with following content:

```
template<typename T>
  struct hash<valarray<T>>;
```

Enabled if specialization `hash<remove_const_t<T>>` is enabled, and disabled otherwise.

Remove a paragraph from Section "20.3.5, Hash support [basic.string.hash]", with following content:

```
template<> struct hash<string>;
template<> struct hash<u8string>;
template<> struct hash<u16string>;
template<> struct hash<u32string>;
template<> struct hash<wstring>;
template<> struct hash<pmr::string>;
template<> struct hash<pmr::u8string>;
template<> struct hash<pmr::u16string>;
template<> struct hash<pmr::u32string>;
template<> struct hash<pmr::wstring>;
```

Add a new paragraph to Section "20.3.5, Hash support [basic.string.hash]", with following content:

```
template<typename charT, typename Allocator>
  struct hash<basic_string<charT, char_traits<charT>, Allocator>>;
```

Enabled if specialization `hash<remove_const_t<charT>>` is enabled, and disabled otherwise.

Add a new paragraph to "21.2.1, General container requirements [container.requirements.general]", with following content:

Let `X` denote a container type, `x` denote a value of type `X`, `TUPLE` denote a `tuple` value of all the values of `x` from `begin()` to `end()` where `get<i>(TUPLE)` equals `*advance(x.begin(), i)` for any i in range `[0; x.size())`. For enabled specialization `hash<X>` the following holds: `hash<X>{}(x) == hash<decltype(TUPLE)>{}(TUPLE)`.

§ 3. Possible implementation

Some possible implementations can be found in Boost.Hash library.

§ 4. References

Boost.Hash