21125052 – Pham Vo Quynh Nhu

std:: unordered_set

Member functions

1. std::unordered_set::bucket

size_type bucket (const key_type& k) const;

Locate element's bucket

- Returns the bucket number where the element with value k is located.
- A bucket is a slot in the container's internal hash table to which elements are assigned based on their hash value. Buckets are numbered from 0 to (bucket_count-1).
- Individual elements in a bucket can be accessed by means of the range iterators returned by unordered_set::begin and unordered_set::end.

Parameters:

k: Value whose bucket is to be located.

Example:

```
#include <iostream>
#include <string>
#include <unordered_set>
int main ()
{
  std::unordered_set<std::string> myset = {"water","sand","ice","foam"};
  for (const std::string& x: myset) {
    std::cout << x << " is in bucket #" << myset.bucket(x) << std::endl;
  }
  return 0;
}
```

Output:

```
ice is in bucket #0
foam is in bucket #2
sand is in bucket #2
water is in bucket #4
```

2. std::unordered_set::bucket_count

size_type bucket_count() const noexcept;

Return number of buckets

- Returns the number of buckets in the unordered_set container.
- A bucket is a slot in the container's internal hash table to which elements are assigned based on their hash value.
- The number of buckets influences directly the load factor of the container's hash table (and thus the probability of collision). The container automatically increases

the number of buckets to keep the load factor below a specific threshold (its max_load_factor), causing a rehash each time the number of buckets needs to be increased.

Parameters:

None

Example:

```
#include <iostream>
#include <string>
#include <unordered_set>
int main ()
{
  std::unordered_set<std::string> myset =
  {"Mercury","Venus","Earth","Mars","Jupiter","Saturn","Uranus","Neptune"};
  unsigned n = myset.bucket_count();
  std::cout << "myset has " << n << " buckets.\n";
  for (unsigned i=0; i<n; ++i) {
    std::cout << "bucket #" << i << " contains:";
    for (auto it = myset.begin(i); it!=myset.end(i); ++it)
      std::cout << " " << *it;
    std::cout << "\n";
  }
  return 0;
}
```

Output:

```
myset has 11 buckets.
bucket #0 contains:
bucket #1 contains: Venus
bucket #2 contains: Jupiter
bucket #3 contains:
bucket #4 contains: Neptune Mercury
bucket #5 contains:
bucket #6 contains: Earth
bucket #7 contains: Uranus Saturn
bucket #8 contains: Mars
bucket #9 contains:
bucket #10 contains:
```

3. std::unordered_set::bucket_size

size_type bucket_size (size_type n) const;

Return bucket size

- Returns the number of elements in bucket n.
- A bucket is a slot in the container's internal hash table to which elements are assigned based on their hash value.

- The number of elements in a bucket influences the time it takes to access a particular element in the bucket. The container automatically increases the number of buckets to keep the load factor (which is the average bucket size) below its max_load_factor.

Parameters:

n: Bucket number. This shall be lower than bucket_count.

Example:

```
#include <iostream>
#include <string>
#include <unordered_set>
int main ()
{
  std::unordered_set<std::string> myset =
  { "red", "green", "blue", "yellow", "purple", "pink" };
  unsigned nbuckets = myset.bucket_count();
  std::cout << "myset has " << nbuckets << " buckets:\n";
  for (unsigned i=0; i<nbuckets; ++i) {
    std::cout << "bucket #" << i << " has " << myset.bucket_size(i) << "
elements.\n";
  }
  return 0;
}
```

Output:

```
myset has 7 buckets:
bucket #0 has 1 elements.
bucket #1 has 1 elements.
bucket #2 has 2 elements.
bucket #3 has 0 elements.
bucket #4 has 1 elements.
bucket #5 has 1 elements.
bucket #6 has 0 elements.
```

4. std::unordered_set::equal_range

pair<iterator,iterator>   equal_range (const key_type& k);

pair<const_iterator,const_iterator>   equal_range (const key_type& k) const;

Get range of elements with a specific key

- Returns the bounds of a range that includes all the elements that compare equal to k. In unordered_set containers, where keys are unique, the range will include one element at most.
- If k does not match any element in the container, the range returned has end as both its lower and upper range bounds.
- All iterators in an unordered_set have const access to the elements (even those whose type is not prefixed with const_): Elements can be inserted or removed, but not modified while in the container.

Parameters:

k: Value to be compared.

5. std::unordered_set::find

```
  iterator find ( const key_type& k );
const_iterator find ( const key_type& k ) const;
```

Get iterator to element

- Searches the container for an element with k as value and returns an iterator to it if found, otherwise it returns an iterator to unordered_set::end (the element past the end of the container).
- Another member function, unordered_set::count, can be used to just check whether a particular element exists.
- All iterators in an unordered_set have const access to the elements (even those whose type is not prefixed with const_): Elements can be inserted or removed, but not modified while in the container.

Parameters:

k: Key to be searched for.

Example:

```
#include <iostream>
#include <string>
#include <unordered_set>
int main ()
{
  std::unordered_set<std::string> myset = { "red","green","blue" };
  std::string input;
  std::cout << "color? ";
  getline (std::cin,input);
  std::unordered_set<std::string>::const_iterator got = myset.find (input);
  if ( got == myset.end() )
    std::cout << "not found in myset";
  else
    std::cout << *got << " is in myset";
  std::cout << std::endl;
  return 0;
}
```

Output:

```
color? blue
blue is in myset
```

6. std::unordered_set::hash_function

```
hasher hash_function() const;
```

Get hash function

- Returns the hash function object used by the unordered_set container.

- The hash function is a unary function that takes an object of type key_type as argument and returns a unique value of type size_t based on it. It is adopted by the container on construction (see unordered_set's constructor for more info). By default, it is the default hashing function for the corresponding key type: hash<key_type>.

Parameters:

None.

Example:

```
#include <iostream>
#include <string>
#include <unordered_set>
typedef std::unordered_set<std::string> stringset;
int main ()
{
  stringset myset;
  stringset::hasher fn = myset.hash_function();
  std::cout << "that: " << fn ("that") << std::endl;
  std::cout << "than: " << fn ("than") << std::endl;
  return 0;
}
```

Output:

```
that: 1046150783
than: 929786026
```

7. std::unordered_set::rehash

void rehash (size_type n);

Set number of buckets

- Sets the number of buckets in the container to n or more.
- If n is greater than the current number of buckets in the container (bucket_count), a rehash is forced. The new bucket count can either be equal or greater than n.
- If n is lower than the current number of buckets in the container (bucket_count), the function may have no effect on the bucket count and may not force a rehash.
- A rehash is the reconstruction of the hash table: All the elements in the container are rearranged according to their hash value into the new set of buckets. This may alter the order of iteration of elements within the container.
- Rehashes are automatically performed by the container whenever its load factor is going to surpass its max_load_factor in an operation.
- Notice that this function expects the number of buckets as argument. A similar function exists, unordered_set::reserve, that expects the number of elements in the container as argument.

Parameters:

n: The minimum number of buckets for the container hash table.

8. std::unordered_set::size

size_type size() const noexcept;

Return container size

- Returns the number of elements in the unordered_set container.

Parameters:

None.

Example:

```
#include <iostream>
#include <string>
#include <unordered_set>

int main ()
{
  std::unordered_set<std::string> myset;
  std::cout << "0. size: " << myset.size() << std::endl;
  myset = {"milk","potatoes","eggs"};
  std::cout << "1. size: " << myset.size() << std::endl;
  myset.insert ("pineapple");
  std::cout << "2. size: " << myset.size() << std::endl;
  myset.erase ("milk");
  std::cout << "3. size: " << myset.size() << std::endl;
  return 0;
}
```

Output:

```
0. size: 0
1. size: 3
2. size: 4
3. size: 3
```