Modern cpp 30 lectures — complex container

Friday, December 18, 2020 9:33 AM

less

表示大小关系，在标准库中定义：

```
template <class T>
struct less
  : binary_function<T, T, bool> {
  bool operator()(const T& x,
                  const T& y) const
  {
    return x < y;
  }
};
```

less是一个函数对象，并且是个二元函数，执行对任意类型值的比较，返回布尔类型。
作为函数对象，它定义了函数调用运算符 operator()，并且缺省行为是对指定类型的对象进行 < 的比较操作。
在需要大小比较的场合，eg. 容器元素的排序默认使用less.

hash

把一个某种类型的值转换成一个无符号整数哈希值，类型为 size_t，

```
template <class T> struct hash;

template < >
struct hash<int>
  : public unary_function<int, size_t>{
  size_t operator() (int v) const
    noexcept
  {
    return static_cast<size_t>(v);
  }
};
```

Eg.

```
#include <algorithm>   // std::sort
#include <functional>  // std::less/greater/hash
#include <iostream>    // std::cout/endl
#include <string>      // std::string
#include <vector>      // std::vector
#include "output_container.h"

using namespace std;

int main()
{
  vector<int> v{13, 6, 4, 11, 29};
  cout << v << endl;

  sort(v.begin(), v.end());
  cout << v << endl;
                          // 从大到小排序
  sort(v.begin(), v.end(), greater<int>());
  cout << v << endl;

  cout << hex;

  auto hp = hash<int*>();
  cout << "hash(nullptr) = " << hp(nullptr) << endl;
  cout << "hash(v.data()) = " << hp(v.data()) << endl;
  cout << "v.data() = " << static_cast<void*>(v.data()) << endl;
```

vector_name.data() returns a pointer to the first element in the array which is used internally by the vector.

```
  auto hs = hash<string>();
  cout << "hash(\"hello\") = " << hs(string("hello")) << endl;
  cout << "hash(\"hellp\") = " << hs(string("help")) << endl;
}
```

Output:

```
{13, 6, 4, 11, 29}
{4, 6, 11, 13, 29}
{29, 13, 11, 6, 4}
hash(nullptr) = a8···c5
hash(v.data()) = 7a···d2
v.data() = 00···E0
hash("hello") = a4···0b
hash("hellp") = a4···22
```

priority_queue

也是一个容器适配器，用到了比较函数，和stack相似，支持push, pop, top等。
在使用缺省的less作为其Compare模板参数时，最大的数值会出现在容器的顶部。如果需要最小的数值出现在容器顶部，则可以传递greater作为其Compare模板参数。

```
#include <functional>  // std::greater
#include <iostream>    // std::cout/endl
#include <memory>      // std::pair
#include <queue>       // std::priority_queue
#include <vector>      // std::vector
#include "output_container.h"

using namespace std;

int main()
{
  priority_queue<
    pair<int, int>,
    vector<pair<int, int>>,
    greater<pair<int, int>>>
    q;

  q.push({1, 1});
  q.push({2, 2});
  q.push({0, 3});
  q.push({9, 4});

  while(!q.empty()) {
    cout << q.top() << endl;
    q.pop();
  }
}
```

Output:

```
(0, 3)
(1, 1)
(2, 2)
(9, 4)
```

关联容器

有set, map, multiset, multimap.

```
#include <functional>
#include <map>
#include <set>
#include <string>

using namespace std;

set<int> s{1, 1, 1, 2, 3, 4};   ← s: {1, 2, 3, 4}

multiset<int, greater<int>> ms{1, 1, 1, 2, 3, 4};   ← ms: {4, 3, 2, 1, 1, 1}

map<string, int> mp{
  {"one", 1},
  {"two", 2},
  {"three", 3},
  {"four", 4}
};
// mp: {"four" ⇒ 4, "one" ⇒ 1, "three" ⇒ 3, "two" ⇒ 2}

mp.insert({"four", 4});   ← mp: {"four" ⇒ 4, "one" ⇒ 1, "three" ⇒ 3, "two" ⇒ 2}

mp.find("four") == mp.end();   ← false

mp.find("five") == mp.end();   ← true

mp["five"] = 5;   ← mp: {"five" ⇒ 5, "four" ⇒ 4, "one" ⇒ 1, "three" ⇒ 3, "two" ⇒ 2}

multimap<string, int> mmp{   ← mmp: {"four" ⇒ 4, "one" ⇒ 1, "three" ⇒ 3, "two" ⇒ 2}
  {"one", 1},
  {"two", 2},
  {"three", 3},
  {"four", 4}
};

mmp.insert({"four", -4});   ← mmp: {"four" ⇒ 4, "four" ⇒ -4, "one" ⇒ 1, "three" ⇒ 3, "two" ⇒ 2}
```

- 关联容器是一种有序的容器
- 名字带multi的允许键重复，不带的不允许键重复。
- set和multiset只能用来存放键，而map和multimap则存放一个键值对。

与序列容器相比，关联容器没有前、后的概念，及相关的成员函数，但同样提供insert, emplace等成员函数。
关联容器都有find, lower_bound, upper_bound等查找函数，结果是一个迭代器：
- find(k) 可以找到任何一个等于查找键k的元素：(!(x < k || k < x))
- lower_bound(k) 找到第一个不小于查找键k的元素：(!(x < k))
- upper_bound(k) 找到第一个大于查找键k的元素：(k < x)

```
mp.find("four")->second;   ← 4.

mp.lower_bound("four")->second;   ← 4

(--mp.upper_bound("four"))->second;   ← 4.

mmp.lower_bound("four")->second;   ← 4

(--mmp.upper_bound("four"))->second;   ← -4
```

如果要在multimap里精确查找满足某个键的区间，建议使用equal_range，可一次性取得上下界（半开半闭）：

```
#include <tuple>
multimap<string, int>::iterator
  lower, upper;
std::tie(lower, upper) =
  mmp.equal_range("four");
(lower != upper);   ← true, 检测区间非空.
lower->second;   ← 4
(--upper)->second;   ← -4.
```

无序关联容器

从C++11开始，每一个关联容器都有一个对应的无序关联容器。
- unordered_set
- unordered_map
- unordered_multiset
- unordered_multimap.

这些容器不要求提供一个排序的函数对象，而要求一个可计算哈希值的函数对象。

```
#include <complex>        // std::complex
#include <iostream>       // std::cout/endl
#include <unordered_map>  // std::unordered_map
#include <unordered_set>  // std::unordered_set
#include "output_container.h"

using namespace std;

namespace std {    在std命名空间中，添加complex number的hash特化.

template <typename T>
struct hash<complex<T>> {
  size_t
  operator()(const complex<T>& v) const
    noexcept
  {
    hash<T> h;
    return h(v.real()) + h(v.imag());
  }
};

} // namespace std

int main() {
  unordered_set<int> s{
    1, 1, 2, 3, 5, 8, 13, 21
  };
  cout << s << endl;

  unordered_map<complex<double>,
                double>
    umc{{{1.0, 1.0}, 1.4142},
        {{3.0, 4.0}, 5.0}};
  cout << umc << endl;
}
```

Output:

```
{21, 5, 8, 3, 13, 2, 1}
{(3,4) ⇒ 5, (1,1) ⇒ 1.4142}
```

无序关联容器的主要优点在于性能，
- 关联容器和priority_queue的插入和删除操作，以及关联容器的查找，复杂度都是 O(log n).
- 无序关联容器使用哈希表，可达平均 O(1).

array

```
#include <iostream>  // std::cout/endl
#include <iterator>  // std::size
int main()
{                    // C数组
  int arr[] = {1, 2, 3, 4, 5};
  std::cout << "The array length is "
            << std::size(arr)
            << std::endl;
}
```

应当避免使用C数组而使用array.

```
#include <array>     // std::array
#include <iostream>  // std::cout/endl
#include <map>       // std::map
#include "output_container.h"

typedef std::array<char, 8> mykey_t;

int main()
{
  std::map<mykey_t, int> mp;
  mykey_t mykey{"hello"};
  mp[mykey] = 5;
  std::cout << mp << std::endl;
}
```

Output:

```
{hello ⇒ 5}
```