Bài 8.9: Tìm hiểu lớp map

- ✓ Tổng quan
- ✓ Các phương thức và mô tả
- ✓ Ví dụ minh họa

Tổng quan

- Map là một container liên kết có thứ tự để chứa dữ liệu dạng cặp key-value với các key là duy nhất.
- Các key được sắp xếp theo trật tự sử dụng hàm compare.
- Các thao tác tìm kiếm, chèn, xóa có độ phức tạp logarit.
- Map thường được triển khai dưới dạng cây đỏ đen.
- Để sử dụng map, ta include thư viện <map> vào đầu file code.

Các hàm thông dụng

Các hàm thông dụng và mô tả
map(); *explicit map(const Compare& comp, const Allocator& alloc = Allocator());* *explicit map(const Allocator& alloc); (Từ C++11)* Hàm tạo một container rỗng.
template<class InputIt> *map(InputIt first, InputIt last, const Compare& comp = Compare(),* *const Allocator& alloc = Allocator());* *template<class InputIt>* *map(InputIt first, InputIt last, const Allocator& alloc); (Từ C++14)* Hàm tạo một container với nội dung cho trong khoảng [first, last). Nếu có nhiều key tương đương nhau nó sẽ không xác định phần tử nào sẽ được thêm vào container.
map(const map& other); *map(const map& other, const Allocator& alloc);* Hàm tạo copy. Tạo container với nội dung của other.
map(initializer_list<value_type> init, const Compare& comp = Compare(), const Allocator& alloc = Allocator()); *map(initializer_list<value_type> init, const Allocator& alloc);* Hàm tạo dùng để tạo một container với nội dung cho trong danh sách khởi tạo init. Nếu nhiều phần tử có cùng key nó sẽ không xác định phần tử nào sẽ được chèn vào container.

Các hàm thông dụng

map& operator=(const map& other); *map& operator=(map&& other); (C++11 – C++17)* *map& operator=(map&& other) noexcept; (Từ C++17)* *map& operator=(initializer_list<value_type> init);* Phép gán. Thay thế nội dung cũ của container bằng nội dung bên phải dấu =.
allocator_type get_allocator() const; (Tới C++11) *allocator_type get_allocator() const noexcept; (Từ C++11)* Trả về allocator liên kết với container.
T& at(const Key& key); *const T& at(const Key& key) const;* Trả về một tham chiếu tới giá trị liên kết với key trong tham số. Nếu không có phần tử nào thỏa mãn, văng ngoại lệ out_of_range.
T& operator[](const Key& key); *T& operator[](Key&& key); (Từ C++11)* Trả về tham chiếu tới giá trị value liên kết với key trong tham số. Thực hiện chèn key ới vào nếu key này chưa tồn tại trong map.
iterator begin(); (Tới C++11) *iterator begin() noexcept; (Từ C++11)*

Các hàm thông dụng

const_ iterator begin() const; (Tới C++11) *const_ iterator begin() const noexcept; (Từ C++11)* *const_ iterator cbegin() const noexcept; (Từ C++11)* Trả về iterator trỏ tới phần tử đầu tiên trong map. Nếu map rỗng trả về giá trị tương đương end().
iterator end(); (Tới C++11) *iterator end() noexcept; (Từ C++11)* *const_ iterator end() const; (Tới C++11)* *const_ iterator end() const noexcept; (Từ C++11)* *const_ iterator cend() const noexcept; (Từ C++11)* Trả về một iterator trỏ tới phần tử sau phần tử cuối của map. Phần tử này chỉ đóng vai trò cột mốc đánh dấu sự kết thúc của container, không có giá trị sử dụng khi phân giải tham chiếu. Nếu cố gắng truy cập vào giá trị đó sẽ gây hành vi không xác định.
iterator rbegin(); (Tới C++11) *iterator rbegin() noexcept; (Từ C++11)* *const_ iterator rbegin() const; (Tới C++11)* *const_ iterator rbegin() const noexcept; (Từ C++11)* *const_ iterator crbegin() const noexcept; (Từ C++11)* Trả về một iterator đảo ngược trỏ tới phần tử đầu tiên trong map đảo ngược. Phần tử đó tương đương phần tử cuối trong map thường. Nếu map rỗng ta nhận được giá trị tương đương rend().
iterator rend(); (Tới C++11) *iterator rend() noexcept; (Từ C++11)* *const_ iterator rend() const; (Tới C++11)* *const_ iterator rend() const noexcept; (Từ C++11)* *const_ iterator crend() const noexcept; (Từ C++11)* Trả về iterator đảo ngược trỏ tới phần tử sau phần tử cuối của map đảo ngược. Phần tử đó tương đương phần tử trước phần tử đầu tiên của map thông thường. Nếu cố truy cập vào giá trị của vị trí đó sẽ gây hành vi không xác định.

Các hàm thông dụng

bool empty() const; *bool empty() const noexcept;* *[[nodiscard]] bool empty() const noexcept;* Kiểm tra xem container có rỗng hay không. Container rỗng khi begin() == end().
size_type size() const; *size_type size() const noexcept;* Trả về số lượng phần tử hiện có trong container. Giá trị này tương đương std::distance(begin(), end()).
size_type max_size() const; *size_type max_size() const noexcept;* Trả về số phần tử tối đa có thể có của container tùy theo hệ điều hành và thư viện mà container được triển khai.
void clear(); *void clear() noexcept;* Xóa toàn bộ các phần tử trong container.
pair<iterator, bool> insert(const value_type& value); *iterator insert(iterator hint, const value_type& value); (Tới C++11)* *iterator insert(const_iterator hint, const value_type& value); (Từ C++11)* *template<class InputIt> void insert(InputIt first, InputIt last);* *void insert(initializer_list<value_type>) iList;*

Các hàm thông dụng

Chèn thêm phần tử vào container nếu container chưa có phần tử nào trùng key với phần tử cần chèn.
template<class M> pair<iterator, bool> insert_or_assign(const Key& k, M&& obj); (Từ C++17) *template<class M> pair<iterator, bool> insert_or_assign(Key& k, M&& obj); (Từ C++17)* Nếu một key nào đó đã tồn tại trong map trùng với k, giá trị của key đó sẽ được cập nhật bởi obj. Nếu key đó chưa tồn tại thì cặp key-value mới sẽ được thêm vào map.
void erase(iterator pos); (Tới C++11) *iterator erase(iterator pos); (Từ C++11)* *iterator erase(const_iterator pos); (Từ C++11)* *void erase(iterator first, iterator last); (Tới C++11)* *iterator erase(const_iterator first, const_iterator last); (Từ C++11)* *size_type erase(const Key& key);* Xóa phần tử cụ thể khỏi container.
void swap(map& other); (Tới C++17) *void swap(map& other) noexcept; (Từ C++17)* Tráo đổi các phần tử của container hiện tại với container other.
node_type extract(const_iterator position); (1) *node_type extract(const Key& k); (2)* *template<class K> node_type extract(K&& x); (Từ C++23)* (1)Hủy liên kết tới node chứa phần tử trỏ tới bởi iterator position và trả về node handle sở hữu nó. (2)Nếu container có phần tử với key tương đương k, hủy liên kết bode chứa phần tử đó khỏi container và trả về node handle sở hữu nó. Ngược lại trả về một node handler rỗng.

Các hàm thông dụng

template<class C2> void merge(map<Key, T, C2, Allocator>& source); (Từ C++17) *template<class C2> void merge(map<Key, T, C2, Allocator>&& source); (Từ C++17)* *template<class C2> void merge(multimap<Key, T, C2, Allocator>& source); (Từ C++17)* *template<class C2> void merge(multimap<Key, T, C2, Allocator>&& source); (Từ C++17)* Trích xuất từng phần tử trong source chèn vào map hiện tại sử dụng đối tượng so sánh của *this. Nếu có phần tử trong *this mà key trung với key phần tử trong source thì phần tử trong source đó không được trích xuất ra.
size_type count(const Key& key) const; *template<class K> size_type count(const K& key) const;* Trả về số phần tử có key tương đương với tham số truyền vào. Kết quả có thể là 0 hoặc 1.
bool contains(const Key& key) const; (Từ C++20) *template<class K> bool contains(const K& key) const; (Từ C++20)* Kiểm tra xem có tồn tại phần tử với key cho trước trong container không.
iterator lower_bound(const Key& key); *const_iterator lower_bound(const Key& key) const;* *template<class K> iterator lower_bound(const K& key);* *template<class K> const_iterator lower_bound(const K& key) const;* Trả về một iterator trỏ tới phần tử đầu không nhỏ hơn key.
iterator upper_bound(const Key& key); *const_iterator upper_bound(const Key& key) const;* *template<class K> iterator upper_bound(const K& key);* *template<class K> const_iterator upper_bound(const K& key) const;* Trả về một iterator trỏ tới phần tử đầu tiên lớn hơn key.
key_compare key_comp() const; Trả về đối tượng hàm sử dụng để so sánh các key.
map::value_compare value_comp() const;

Nội dung tiếp theo

Tìm hiểu lớp unordered_set