# Bài 8.8: Tìm hiểu lớp set

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

# Tổng quan

- set là một container liên kết chứa tập hợp các đối tượng duy nhất và đã được sắp đặt có thứ tự của một kiểu T nào đó.
- Việc sắp xếp các key được thực hiện dựa vào hàm compare: std::less<Key>.
- Giá trị của các phần tử ở trong set là không thể sửa đổi nhưng ta có thể thêm mới, xóa bỏ chúng.
- Thao tác tìm kiếm, xóa, chèn thêm phần tử mới có độ phức tạp logarit.
- Về tốc độ, set chậm hơn unordered_set khi truy cập 1 phần tử đơn dựa vào key của nó.
- Set thường được triển khai như cây nhị phân tìm kiếm.
- Các tính chất của set: tính liên kết, tính có trật tự, tính tập hợp, tính duy nhất và tự động quản lý cấp phát bộ nhớ.

# Các hàm thông dụng

| |
|---|
| ***set();***<br>***explicit set(const Compare& comp, const Allocator& alloc = Allocator());***<br>***explicit set(const Allocator& alloc);***<br>Hàm tạo mặc định. Dùng để tạo một container rỗng. |
| ***set(InputIt first, InputIt last, const Compare& comp = Compare(), const Allocator& alloc = Allocator());***<br>***set(InputIt first, InputIt last, const Allocator& alloc); (Từ C++14)***<br>Hàm tạo container với nội dung cho trong một khoảng [first, last). Nếu trong khoảng đã cho có nhiều phần tử có giá trị tương đương, nó sẽ không xác định cụ thể phần tử nào được chọn để chèn vào set. |
| ***set(const set& other);***<br>***set(const set& other, const Allocator& alloc);***<br>Hàm tạo copy, tạo container với các bản sao của nội dung cho trong other. |
| ***set(initializer_list<value_type> init, const Compare& comp = Compare(), const Allocator& alloc = Allocator());***<br>***set((initializer_list<value_type> init, const Allocator& alloc); (Từ C++14)***<br>Tạo một container mới với nội dung cho trước trong danh sách khởi tạo init. |
| ***iterator begin(); (Tới C++11)***<br>***iterator begin() noexcept; (Từ C++11)***<br>***const_iterator begin() const; (Tới C++11)***<br>***const_iterator begin() const noexcept; (Từ C++11)***<br>***const_iterator cbegin() const noexcept; (Từ C++11)***<br>Trả về iterator trỏ tới phần tử đầu tiên trong set. Nếu set rỗng thì kết quả trả về tương đương end(). |

# Các hàm thông dụng

| |
|---|
| ***iterator end(); (Tới C++11)***<br>***iterator end() noexcept; (Từ C++11)***<br>***const_iterator end() const; (Tới C++11)***<br>***const_iterator end() const noexcept; (Từ C++11)***<br>***const_iterator cend() const noexcept; (Từ C++11)***<br>Trả về một iterator trỏ tới vị trí sau phần tử cuối của set. Phần tử này chỉ sử dụng để làm giá trị đối sánh. Việc cố truy cập tới nó sẽ gây ra hành vi không xác định. |
| ***reverse_iterator rbegin(); (Tới C++11)***<br>***reverse_iterator rbegin() noexcept; (Từ C++11)***<br>***const_reverse_iterator rbegin() const; (Tới C++11)***<br>***const_reverse_iterator rbegin() const noexcept; (Từ C++11)***<br>***const_reverse_iterator crbegin() const noexcept; (Từ C++11)***<br>Trả về một iterator đảo ngược trỏ tới phần tử đầu của set đảo ngược. Nó tương đương phần tử cuối trong set thông thường. Nếu set rỗng, kết quả trả về tương đương rend(). |
| ***reverse_iterator rend(); (Tới C++11)***<br>***reverse_iterator rend() noexcept; (Từ C++11)***<br>***const_reverse_iterator rend() const; (Tới C++11)***<br>***const_reverse_iterator rend() const noexcept; (Từ C++11)***<br>***const_reverse_iterator rend() const noexcept; (Từ C++11)***<br>Trả về một iterator đảo trỏ tới phần tử sau phần tử cuối của set đảo ngược. Phần tử này tương ứng với phần tử trước phần tử đầu tiên trong set thông thường. Phần tử này cũng chỉ được sử dụng với mục đích làm mốc đánh dấu sự kết thúc của iterator đảo. Nếu cố truy cập vào nó có thể gây hành vi không xác định. |
| ***bool empty() const; (Tới C++11)***<br>***bool empty() const noexcept; (Từ C++11-C++20)*** |

# Các hàm thông dụng

| |
|---|
| ***[[nodiscard]] bool empty() const noexcept; (Từ C++20)***<br>Hàm kiểm tra xem container có rỗng không. Nếu container rỗng thì begin() == end(). |
| ***size_type size() const; (Tới C++11)***<br>***size_type size() const noexcept; (Từ C++11)***<br>Trả về số phần tử hiện có trong container. |
| ***size_type max_size() const; (Tới C++11)***<br>***size_type max_size() const noexcept; (Từ C++11)***<br>Trả về số lượng phần tử tối đa mà container có thể chứa tùy theo hệ điều hành và thư viện đang sử dụng. Đây là giá trị theo lý thuyết có thể đạt được tuy nhiên nó phụ thuộc vào trạng thái RAM khả dụng của thiết bị. |
| ***void clear(); (Tới C++11)***<br>***void clear() noexcept; (Tới C++11)***<br>Xóa tất cả các phần tử có trong container. Sau khi gọi hàm này, size của container sẽ về 0. |
| ***std::pair<iterator, bool> insert(const value_type& value);***<br>***std::pair<iterator, bool> insert(value_type& value); (Từ C++11)***<br>***iterator insert(iterator hint, const value_type& value); (Tới C++11)***<br>***iterator insert(const_iterator hint, const value_type& value); (Từ C++11)***<br>***iterator insert(const_iterator hint, value_type&& value); (Từ C++11)***<br>***void insert(InputIt first, InputIt last);***<br>***void insert(initializer_list<value_type> iList); (Từ C++11)***<br>Hàm này dùng để chèn thêm 1 hoặc một số phần tử vào container nếu set chưa tồn tại phần tử nào tương đương với key truyền vào. Hint là một iterator dùng để gợi ý vị trí bắt đầu việc tìm kiếm chỗ để chèn (các) phần tử mới vào. |

# Các hàm thông dụng

| |
|---|
| ***std::pair<iterator, bool> emplace(Args&&... args); (Từ C++11)***<br>Chèn thêm phần tử mới vào set nếu phần tử đó chưa tồn tại trong set. Việc sử dụng hàm này sẽ trực tiếp tạo đối tượng mới, tránh được việc sao chép hoặc di chuyển không cần thiết. |
| ***void erase(iterator pos); (Tới C++11)***<br>***iterator erase(const_iterator pos); (Từ C++11)***<br>***iterator erase(iterator pos); (Từ C++17)***<br>***void erase(iterator first, iterator last); (Tới C++11)***<br>***iterator erase(const_iterator first, const_iterator last); (Từ C++11)***<br>***size_type erase(const key_type& key);***<br>Xóa một phần tử cụ thể khỏi container. Có thể xóa phần tử tại vị trí pos, trong khoảng [first, last) hoặc xóa phần tử có giá trị tương đương với key truyền vào. |
| ***void swap(set& other); (Tới C++17)***<br>***void swap(set& other) noexcept; (Từ C++17)***<br>Tráo đổi nội dung của container hiện thời với container other. |
| ***node_type extract(const_iterator position); (Từ C++17).(1)***<br>***node_type extract(const key_type& x); (Từ C++17).(2)***<br>(1) Hủy liên kết node chứa phần tử trỏ tới bởi position và trả về node chứa phần tử đó.<br>(2) Nếu container chứa một phần tử trùng với x, hủy liên kết node chứa phần tử đó khỏi container và trả về một node chứa nó. Ngược lại, trả về một node rỗng. |
| ***void merge(set<Key, C2, Allocator>& source); (Từ C++17)***<br>***void merge(set<Key, C2, Allocator>&& source); (Từ C++17)***<br>***void merge(multiset<Key, C2, Allocator>& source); (Từ C++17)***<br>***void merge(multiset<Key, C2, Allocator>&& source); (Từ C++17)*** |

# Các hàm thông dụng

| |
|---|
| Thử trích xuất từng phần tử trong source và chèn nó vào set hiện thời sử dụng đối tượng so sánh của *this. Nếu có một phần tử trong *this có key tương đương với key của phần tử trong source, phần tử đó sẽ không được trích xuất từ source. Chỉ xảy ra việc thay đổi con trỏ bên trong các set tham gia vào hành động này. |
| ***size_type count(const Key& key) const;***<br>***template<class K> size_type count(const K& x) const; (Từ C++14)***<br>Trả về số lượng phần tử có key tương đương với tham số đầu vào. Tức là có thể là 1 hoặc 0 do set không cho phép các phần tử trùng lặp. |
| ***iterator find(const Key& key);***<br>***const_iterator find(const Key& key) const;***<br>**iterator find(const K& x); (Từ C++14)**<br>**const_iterator find(const K& x) const; (Từ C++14)**<br>Tìm phần tử có key trùng với key trong tham số hoặc tương đương với giá trị x truyền vào. |
| ***bool contains(const Key& key) const; (Từ C++20)***<br>***template<class K> bool contains(const K& x) const; (Từ C++20)***<br>Kiểm tra xem có tồn tại phần tử có key tương đương với tham số trong container không. |

# Nội dung tiếp theo

## Tìm hiểu lớp map