Bài 8.11: Tìm hiểu lớp unordered_map

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

Tổng quan

- unordered_map là một container liên kết không có thứ tự để chứa dữ liệu dạng cặp key-value với các key là duy nhất.
- Các phần tử trong container này không được sắp xếp theo bất kì một trật tự nào. Chúng được lưu vào các vùng chứa tùy theo giá trị băm của nó nhằm tăng tốc quá trình truy cập phần tử cụ thể qua key của nó.
- Xét về tốc độ truy cập phần tử đơn, unordered_map nhanh hơn map.
- Container này hỗ trợ truy cập trực tiếp vào value của một key được chỉ định qua toán tử [].
- Để sử dụng map, ta include thư viện <unordered_map> vào đầu file chương trình.

Một số hàm thông dụng

Các hàm thông dụng và mô tả
unordered_map(); *explicit unordered_map(size_type bucket_count, const Hash& hash = Hash(), const key_equal& equal = key_equal(), const Allocator& alloc = Allocator()); (Từ C++11)* *explicit unordered_map(const Allocator& alloc); (Từ C++11)* *unordered_map(size_type bucket_count, const Allocator& alloc); (Từ C++14)* *unordered_map(size_type bucket_count, const Hash& hash = Hash(), const Allocator& alloc); (Từ C++14)* Hàm tạo một container rỗng. Thiết lập max_load_factor() bằng 1.0.
template<class InputIt> *unordered_map(InputIt first, InputIt last, size_type bucket_count = default, const Hash& hash = Hash(), const key_equal& equal = key_equal(), const Allocator& alloc = Allocator()); (Từ C++11)* *template<class InputIt>* *unordered_map(InputIt first, InputIt last, size_type bucket_count, const Allocator& alloc); (Từ C++14)* *template<class InputIt>* *unordered_map(InputIt first, InputIt last, size_type bucket_count, const Hash& hash = Hash, const Allocator& alloc); (Từ C++14)* Hàm tạo một container với nội dung cho trong khoảng [first, last). Gán max_load_factor bằng 1.0. Nếu có nhiều key tương đương nhau nó sẽ không xác định phần tử nào sẽ được thêm vào container.
unordered_map(const unordered_map& other); *unordered_map(const unordered_map& other, const Allocator& alloc);* Hàm tạo copy. Tạo container với nội dung của other. Sao chép cả load factor, predicate và hàm băm.

Một số hàm thông dụng

unordered_map(initializer_list<value_type> init, size_type bucket_count = default, const Hash& hash = Hash(), const key_equal& equal = key_equal(), const Allocator& alloc = Allocator()); (Từ C++11) *unordered_map(initializer_list<value_type> init, size_type bucket_count, const Allocator& alloc); (Từ C++14)* *unordered_map(initializer_list<value_type> init, size_type bucket_count, const Hash& hash, const Allocator& alloc); (Từ C++14)* Hàm tạo dùng để tạo một container với nội dung cho trong danh sách khởi tạo init.
unordered_map& operator=(const unordered_map& other); *unordered_map& operator=(unordered_map&& other); (C++11 – C++17)* *unordered_map& operator=(unordered_map&& other) noexcept; (Từ C++17)* *unordered_map& operator=(initializer_list<value_type> init); (Từ C++17)* Phép gán. Thay thế nội dung cũ của container bằng nội dung bên phải dấu =.
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 std::out_of_range.
T& operator[](const Key& key); (Từ C++11) *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 unordered_map.

Một số hàm thông dụng

iterator begin() noexcept; (Từ 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 unordered_map. Nếu unordered_map rỗng trả về giá trị tương đương end().
iterator end() noexcept; (Từ 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 unordered_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.
bool empty() const noexcept; (C++11-C++20) *[[nodiscard]] bool empty() const noexcept; (Từ C++20)* Kiểm tra xem container có rỗng hay không. Container rỗng khi begin() == end().
size_type size() const noexcept; (Từ C++11) Trả về số lượng phần tử hiện có trong container. Giá trị này tương đương std::distance(begin(), end()).
size_type max_size() const noexcept; (Từ C++11) 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() noexcept; Xóa toàn bộ các phần tử trong container.
pair<iterator, bool> insert(const value_type& value); (Từ C++11) *template<class P> pair<iterator, bool> insert(P&& value); (Từ C++11)* *iterator insert(const_iterator hint, const value_type& value); (Từ C++11)*

Một số hàm thông dụng

iterator insert(const_iterator hint, value_type&& value); (Từ C++17) *template<class InputIt> void insert(InputIt first, InputIt last); (Từ C++11)* *void insert(initializer_list<value_type> iList); (Từ C++11)* 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 unordered_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 unordered_map.
iterator erase(iterator pos); (Từ C++11) *iterator erase(const_iterator pos); (Từ C++11)* *iterator erase(const_iterator first, const_iterator last); (Từ C++11)* *size_type erase(const Key& key); (Từ C++11)* Xóa phần tử cụ thể khỏi container.
void swap(unordered_map& other); (C++11 Tới C++17) *void swap(unordered_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.

Một số hàm thông dụng

template<class C2, class P2> void merge(unordered_map<Key, T, C2, P2, Allocator>& source); (Từ C++17) *template<class C2, class P2> void merge(multiunordered_map<Key, T, C2, P2, Allocator>& source); (Từ C++17)* Trích xuất từng phần tử trong source chèn vào unordered_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; (Từ C++11) *template<class K> size_type count(const K& key) const; (Từ C++20)* Trả về số phần tử có key tương đương với tham số truyền vào. Kết quả có thể là 0 hoặc 1.
iterator find(const Key& key); (1) *const_iterator find(const Key& key) const; (2)* *template<class K> iterator find(const K& key); (Từ C++20)* *template<class K> const_iterator find(const K& key) const; (Từ C++20)* 1,2 tìm phần tử với key tương đương tham số truyền vào. 2 hàm còn lại tìm phần tử có key so sánh tương đương với key truyền vào.
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.
pair<iterator, iterator> equal_range(const Key& key); (Từ C++11) *pair<const_iterator, const_iterator> equal_range(const Key& key) const; (Từ C++11)* *template<class K> pair<iterator, iterator> equal_range (const K& key); (Từ C++20)* *template<class K> pair<const_iterator, const_iterator> equal_range (const K& key) const; (Từ C++20)*

Một số hàm thông dụng

Trả về khoảng chứa tất cả các phần tử có key trùng với key trong container. Khoảng này được định nghĩa bởi hai iterator, iterator thứ nhất trỏ đến phần tử đầu khoảng, iterator còn lại trỏ tới phần tử sau phần tử cuối của khoảng.
size_type bucket_count() const; (Từ C++11) Trả về số lượng các vùng chứa trong container.
size_type max_bucket_count() const; (Từ C++11) Trả về số lượng các vùng chứa tối đa có thể có trong container. Giá trị này tùy thuộc hệ điều hành và thư viện đang sử dụng.
size_type bucket_size(size_type n) const; (Từ C++11) Trả về số lượng các phần tử trong vùng chứa tại vị trí n.
size_type bucket (const Key& key) const; (Từ C++11) Trả về vị trí của vùng chứa khóa key. Các phần tử với các khóa tương đương giá trị key này luôn được tìm thấy(nếu có) ở vùng chứa tại vị trí chỉ số trả về bởi hàm này.
float load_factor() const; (Từ C++11) Trả về giá trị load factor của bảng băm.
float max_load_factor() const; (Từ C++11) (1) *void max_load_factor(float ml); (Từ C++11) (2)* (1)Trả về giá trị load factor tối đa. (2)Thiết lập giá trị load factor tối đa
void rehash(size_type count); (Từ C++11) Thiết lập giá trị số lượng vùng chứa cho count và băm lại container.
void reserve(size_type count); (Từ C++11) Thiết lập số lượng vùng chứa về tối thiểu count phần tử mà không vượt quá giá trị tối đa của load factor, không băm lại bảng băm.
hasher hash_function() const; (Từ C++11) *Trả về hàm băm được sử dụng để băm các key trong container này.*

Nội dung tiếp theo

Tìm hiểu các thuật toán sắp xếp