

Bài 5.1: Tìm hiểu deque

- ✓ Khái niệm
- ✓ Đặc điểm
- ✓ Các hàm thông dụng và mô tả
- ✓ Ví dụ minh họa

Khái niệm

- Deque là viết tắt của double-ended queue. Đây là một container tuần tự với kích thước có thể co dãn để tương thích ngữ cảnh sử dụng.
- Deque cho phép chèn thêm, xóa phần tử ở cả hai đầu thay vì chỉ 1 đầu như queue truyền thống.
- Các thư viện thường triển khai deque sử dụng mảng động, cho phép truy cập trực tiếp và ngẫu nhiên vào phần tử.
- Bộ nhớ nội bộ trong deque được quản lý động, có thể mở rộng hoặc thu hẹp khi cần thiết.

Đặc điểm

- Deque cung cấp các tính năng tương tự như vector trong đó bổ sung hiệu quả cho thao tác chèn và xóa ở đầu thay vì chỉ ở cuối.
- Deque không đảm bảo rằng các phần tử được lưu trữ tại những vùng nhớ liền kề nhau.
- Deque có cấu trúc nội bộ phức tạp hơn vector. Nó cho phép sử dụng các vùng nhớ khác nhau để lưu trữ các phần tử.
- Với các ứng dụng yêu cầu chèn, xóa phần tử ở vị trí khác đầu, cuối tập hợp ta nên sử dụng list, forward_list để đạt hiệu quả cao.
- Đặc trưng của deque: tính tuần tự, cấp phát động, tự động nhận diện sự biến đổi của số lượng phần tử và cấp phát cho phù hợp.

Các hàm thông dụng và mô tả

Tên hàm	Mô tả
<code>deque();</code>	Hàm tạo mặc định tạo một deque rỗng.
<code>explicit deque(const Allocator& alloc);</code>	Tạo một deque rỗng với allocator cho trước.
<code>explicit deque(size_type count, const T& value = T(), const Allocator& alloc = Allocator());</code> (Tới C++11) <code>explicit deque(size_type count, const T& value, const Allocator& alloc = Allocator());</code> (Từ C++11)	Tạo một container với count bản sao của các phần tử có giá trị value.
<code>explicit deque(size_type count);</code> (C++11 - C++14) <code>explicit deque(size_type count, const Allocator& alloc = Allocator());</code> (Từ C++14)	Tạo một container với count giá trị mặc định đã chèn vào.
<code>template<class InputIt> deque(InputIt first, InputIt last, const Allocator& alloc = Allocator());</code>	Tạo một container với các phần tử cho trong nửa khoảng [first, last).
<code>deque(const deque& other);</code>	Copy constructor, tạo container với bản sao các phần tử của một deque khác.
<code>deque(const deque& other, const Allocator& alloc);</code> (Từ C++11)	Copy constructor tạo container với nội dung copy từ other. Sử dụng alloc làm allocator.
<code>deque(const deque&& other);</code> (Từ C++11)	Move constructor, tạo container với nội dung của other.
<code>Deque(std::initializer_list<T> init, const Allocator& alloc = Allocator());</code> (Từ C++11)	
<code>deque(const deque&& other, const Allocator& alloc);</code> (Từ C++11)	

Các hàm thông dụng và mô tả

reference at(size_type pos); const_reference at(size_type pos) const;	Trả về một tham chiếu đến phần tử tại vị trí xác định, có kiểm tra biên. Nếu pos không nằm trong biên của container, văng ngoại lệ std::out_of_range.
reference operator[](size_type pos); const_reference operator[](size_type pos) const;	Trả về một tham chiếu tới phần tử tại vị trí pos. Không có kiểm tra biên nào được thực hiện.
reference front(); const_reference front() const;	Trả về một tham chiếu tới phần tử đầu tiên trong container. Nếu container rỗng, hành động này là hành vi không xác định.
reference back(); const_reference back() const;	Trả về một tham chiếu tới phần tử cuối cùng trong container. Nếu container rỗng, hành động này là hành vi không xác định.
iterator begin(); (Tới C++11) iterator begin() noexcept; (Từ C++11) 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 một iterator trỏ tới phần tử đầu tiên trong deque. Nếu deque rỗng, giá trị trả về tương đương với hàm end().
iterator end(); (Tới C++11) iterator end() noexcept; (Từ C++11) const_iterator end() const; (Tới C++11)	Trả về một iterator trỏ tới phần tử sau phần tử cuối của deque. Phần tử này là một giá trị nhận diện sự kết thúc của

Các hàm thông dụng và mô tả

<code>const_iterator end() const noexcept; (Từ C++11)</code> <code>const_iterator cend() const noexcept; (Từ C++11)</code>	container và không có giá trị sử dụng khi phân giải địa chỉ.
<code>reverse_iterator rbegin(); (Tới C++11)</code> <code>reverse_iterator rbegin() noexcept; (Từ C++11)</code> <code>const_reverse_iterator rbegin() const; (Tới C++11)</code> <code>const_reverse_iterator rbegin() const noexcept; (Từ C++11)</code> <code>const_reverse_iterator crbegin() const noexcept; (Từ C++11)</code>	Trả về một iterator đảo trở tới phần tử đầu của deque đảo. Nó tương đương phần tử cuối của deque không đảo ngược. Nếu deque rỗng, giá trị trả về từ hàm này tương đương <code>rend()</code> .
<code>reverse_iterator rend(); (Tới C++11)</code> <code>reverse_iterator rend() noexcept; (Từ C++11)</code> <code>const_reverse_iterator rend() const; (Tới C++11)</code> <code>const_reverse_iterator rend() const noexcept; (Từ C++11)</code> <code>const_reverse_iterator crend() const noexcept; (Từ C++11)</code>	Trả về một iterator đảo ngược trở tới phần tử sau phần tử cuối của deque đảo ngược. Nó tương đương phần tử trước phần tử đầu tiên của deque không đảo ngược. Giá trị này chỉ có tác dụng làm mốc đánh dấu sự kết thúc của deque. Nó không có giá trị sử dụng khác.
<code>bool empty() const; (Tới C++11)</code> <code>bool empty() const noexcept; (Từ C++11-C++20)</code> <code>[[nodiscard]] bool empty() const noexcept; (Từ C++20)</code>	Kiểm tra xem container có rỗng không. Trả về <code>true</code> nếu rỗng và <code>false</code> trong trường hợp ngược lại.
<code>size_type size() const; (Tới C++11)</code> <code>size_type size() const noexcept; (Từ C++11)</code>	Trả về số phần tử hiện có trong container.

Các hàm thông dụng và mô tả

size_type max_size() const; (Tới C++11) size_type max_size() const noexcept; (Từ C++11)	Trả về số phần tử tối đa có thể chứa của container tùy theo hệ điều hành và thư viện đang sử dụng.
void shrink_to_fit(); (Từ C++11)	Yêu cầu loại bỏ phần vùng nhớ không được sử dụng.
void clear(); (Tới C++11) void clear() noexcept; (Từ C++11)	Xóa tất cả các phần tử có trong container.
iterator insert(iterator pos, const T& value); (Tới C++11) iterator insert(const_iterator pos, const T& value); (Từ C++11) iterator insert(const_iterator pos, T&& value); (Từ C++11)	Chèn phần tử có giá trị value vào trước vị trí pos.
void insert(iterator pos, size_type count, const T& value); (Tới C++ 11) iterator insert(const_iterator pos, size_type count, const T& value); (Từ C++ 11)	Chèn count bản sao của value trước vị trí pos.
template<class InputIt> void insert(iterator pos, InputIt first, InputIt last); (Tới C++ 11) template<class InputIt> void insert(const_iterator pos, InputIt first, InputIt last); (Từ C++ 11)	Thêm các phần tử trong nửa khoảng [first, last) vào trước vị trí pos.
iterator insert(const_iterator pos, std::initializer_list<T> ilist); (Từ C++ 11)	
iterator erase(iterator pos); (Tới C++ 11) iterator erase(const_iterator pos); (Từ C++ 11)	Xóa phần tử cụ thể tại vị trí pos khỏi container.

Các hàm thông dụng và mô tả

<code>iterator erase(iterator first, iterator last);</code> (Tới C++ 11) <code>iterator erase(const_iterator first, const_iterator last);</code> (Từ C++ 11)	Xóa các phần tử trong nửa khoảng [first, last) khỏi container.
<code>void push_front(const T& value);</code> <code>void push_front(T&& value);</code> (Từ C++ 11)	Chèn thêm phần tử mới có giá trị value vào đầu container.
<code>void pop_front();</code>	Xóa phần tử đầu container. Nếu không có phần tử nào trong container, hành vi trở thành không xác định.
<code>void push_back(const T& value);</code> <code>void push_back(T&& value);</code> (Từ C++ 11)	Thêm phần tử vào cuối container hiện tại. Với lựa chọn đầu ta copy value vào. Với lựa chọn 2, value được move vào phần tử mới.
<code>void pop_back();</code>	Xóa phần tử cuối khỏi container. Gọi hàm này trên container rỗng sẽ gây ra hành vi không xác định.
<code>void resize(size_type count);</code> (Từ C++ 11) <code>void resize(size_type count, T value = T());</code> (Tới C++ 11) <code>void resize(size_type count, const value_type& value);</code> (Từ C++ 11)	Có sẵn kích thước của container về count phần tử. Nếu size hiện tại lớn hơn count, số phần tử sẽ giảm xuống. Nếu size hiện tại nhỏ hơn count, các phần tử mới với giá trị mặc định hoặc value sẽ được thêm vào container.

Nội dung tiếp theo

Tìm hiểu về cấu trúc dữ liệu cây