Bài 5.8: Thư viện deque

- ✓ Tổng quan
- ✓ Các hàm thông dụng và mô tả
- ✓ Ví dụ minh họa

Tổng quan

- deque là viết tắt của double ended queue. Đây là một container tuần tự với kích thước có thể thay đổi và hỗ trợ các thao tác thêm, xóa ở cả hai đầu.
- Lớp template deque nằm trong namespace std. Để sử dụng deque ta include thư viện <deque> ở đầu file chương trình.
- Mỗi thư viện có thể triển khai container này theo cách khác nhau, hầu hết sử dụng mảng cấp phát động với cơ chế xử lý tự động mở rộng hoặc thu hẹp kích thước của container khi cần.
- Nhưng dù triển khai theo cách nào thì deque cũng hỗ trợ truy cập phần tử thông qua chỉ số của phần tử.

Đặc điểm

- Việc mở rộng của container này tốn ít chi phí hơn so với vector vì không phải cấp phát lại và sao chép phần tử vào vùng nhớ vừa cấp phát.
- Mặt khác container này tiêu tốn nhiều bộ nhớ hơn so với vector. Khi hoạt động nó sẽ cấp phát một mảng với tối đa bộ nhớ tức 4096byte trong thư viện 64 bit stdc++.
- Cung cấp chức năng tương tự vector nhưng cách thức hoạt động lại tương đối khác nhau.
- Chi phí chèn, xóa phần tử khác vị trí đầu cuối container kém hiệu quả hơn list và forward_list.
- Nhìn chung deque phức tạp hơn vector nhưng hiệu quả hơn trong một số điều kiện cụ thể ví dụ như lượng dữ liệu cần lưu trữ rất lớn.

Các hàm thông dụng và mô tả

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

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

const_iterator end() const noexcept; (Từ C++11) const_iterator cend() const noexcept; (Từ C++11)	container và không có giá trị sử dụng khi phân giải địa chỉ.
reverse_iterator rbegin(); (Tới C++11) reverse_iterator rbegin() noexcept; (Từ C++11) const_reverse_iterator rbegin() const; (Tới C++11) const_ reverse_iterator rbegin() const noexcept; (Từ C++11) const_ reverse_iterator crbegin() const noexcept; (Từ C++11)	Trả về một iterator đảo trỏ 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 rend().
reverse_iterator rend(); (Tới C++11) reverse_iterator rend() noexcept; (Từ C++11) const_reverse_iterator rend() const; (Tới C++11) const_ reverse_iterator rend() const noexcept; (Từ C++11) const_ reverse_iterator crend() const noexcept; (Từ C++11)	Trả về một iterator đảo ngược trỏ 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.
bool empty() const; (Tới C++11) bool empty() const noexcept; (Từ C++11-C++20) [[nodiscard]]bool empty() const noexcept; (Từ C++20)	Kiểm tra xem container có rỗng không. Trả về true nếu rỗng và false trong trường hợp ngược lại.
size_type size() const; (Tới C++11) size_type size() const noexcept; (Từ C++11)	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 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)	Thêm một tập các phần tử vào trước vị trí pos.
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

iterator erase(iterator first, iterator last); (Tới C++ 11) iterator erase(const_iterator first, const_iterator last); (Từ C++ 11)	Xóa các phần tử trong nửa khoảng [first, last) khỏi container.
void push_front(const T& value); void push_front(T&& value); (Từ C++ 11)	Chèn thêm phần tử mới có giá trị value vào đầu container.
void pop_front();	Xóa phần tử đầu container. Nếu không có phần tử nào trong container, hành vi trở thành không xác định.
void push_back(const T& value); void push_back(T&& value); (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.
void pop_back();	Xóa phần tử cuối khỏi container. Gọi hàm này trên container rỗng sẽ gây ra hành vi không xác định.
void resize(size_type count); (Từ C++ 11) void resize(size_type count, T value = T()); (Tới C++ 11) void resize(size_type count, const value_type& value); (Từ C++ 11)	Co dãn 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