Bài 3.11: Giới thiệu lớp forward_list

- ✓ Khái niệm và đặc điểm
- ✓ Các hàm thông dụng và mô tả
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Khái niệm

- forward_list là một container tuần tự cho phép chèn xóa phần tử ở vị trí bất kì trong nó với thời gian hằng số.
- Danh sách chuyển tiếp được triển khai như danh sách liên kết đơn. Các phần tử trong danh sách có thể được lưu trữ tại các vùng nhớ không liên tiếp nhau trong bộ nhớ.
- Lớp template forward_list nằm trong namespace std. Để sử dụng forward_list ta include thư viện <forward_list> ở đầu file chương trình.
- Thứ tự các phần tử được lưu giữ bằng liên kết trỏ tới phần tử kế tiếp trong danh sách.

Đặc điểm

- Chỉ tồn tại liên kết theo một chiều tiến từ phần tử trước đến phần tử sau và không có điều ngược lại. Do đó chỉ duyệt được danh sách khi biết phần tử đầu của nó.
- Hiệu quả sử dụng bộ nhớ và tốc độ tốt hơn list.
- So với các container tuần tự khác(array, vector, dequeue), forward_list cho hiệu quả tốt hơn khi chèn, xóa phần tử ở bất kì vị trí nào trong container.
- Tiêu tốn thêm vùng nhớ để lưu trữ địa chỉ của phần tử kế tiếp.
- Không có hàm giám sát kích thước, có thể dùng hàm distance trong thư viện <iterator> để tính khoảng cách hay số lượng phần tử giữa vị trí begin và end.

Các hàm thông dụng và mô tả

Tên hàm	Mô tả
forward_list();	Khởi tạo một forward_list rỗng.
forward_list(size_type n); forward_list (size_type n, const value_type& x);	Khởi tạo forward_list với n phần tử. Mỗi phần tử có giá trị x nếu tham số thứ 2 được chỉ rõ.
forward_list (InputIterator first, InputIterator last);	Tạo một forward_list có số phần tử bằng với số phần tử từ first đến trước last. Các phần tử được gán theo đúng trật tự trong khoảng [first, last).
forward_list (const forward_list& x);	Tạo forward_list và copy các phần tử trong x vào theo đúng trật tự các phần tử xuất hiện trong x.
forward_list(initializer_list<T> init, const Allocator& alloc = Allocator());	Tạo một forward_list với danh sách các phần tử cho trước.
operator=	Phép gán(copy hoặc move) dùng để thay thế nội dung của forward_list bằng nội dung mới bên phải dấu =.

Các hàm thông dụng và mô tả

iterator before_begin() noexcept;
const_iterator before_begin() const noexcept;
const_iterator cbefore_begin() const noexcept;
	Trả về iterator trỏ đến vị trí trước phần tử đầu tiên của forward_list. Việc cố tình truy xuất giá trị tại vị trí này sẽ gây ra hành vi không xác định. Thường sử dụng kết hợp các hàm: insert_after(), emplace_after(), erase_after(), splice_after() và toán tử ++. Việc áp dụng ++ với iterator trả về bởi before_begin() cho kết quả tương tự kết quả trả về từ hàm begin()/cbegin().

iterator begin() noexcept;
const_iterator begin() const noexcept;
const_iterator cbegin() const noexcept;
	Trả về một iterator trỏ đến phần tử đầu của forward_list. Nếu forward_list rỗng, không nên phân giải tham chiếu giá trị trả về từ hàm này.

iterator end() noexcept;
const_iterator end() const noexcept;
const_iterator cend() const noexcept;
	Trả về một iterator tham chiếu đến vị trí sau phần tử cuối của forward_list. Đây là iterator chỉ dùng để xác định sự kết thúc của forward_list. Nếu container rỗng, hàm trả về giá trị tương tự begin()/cbegin().

Các hàm thông dụng và mô tả

iterator insert_after(const_iterator pos, const T& val);(1) iterator insert_after(const_iterator pos, const T&& val);(2) iterator insert_after(const_iterator pos, size_type n, const T& val);(3) iterator insert_after(const_iterator pos, InputIt first, InputIt last);(4) iterator insert_after(const_iterator pos, initializer_list<T> list);(5)	Chèn các phần tử vào sau vị trí pos. (1), (2) chèn 1 giá trị. (3) Chèn n bản sao của val sau vị trí pos. (4) Chèn các phần tử trong nửa khoảng [first, last) vào sau vị trí pos. (5) Chèn các phần tử từ danh sách khởi tạo vào sau vị trí pos.
iterator erase_after(const_iterator pos); iterator erase_after(const_iterator first, const_iterator last);	Xóa bỏ 1 hoặc các phần tử từ first đến last khỏi container. Trả về iterator trỏ đến phần tử sau vị trí xóa hoặc end() nếu sau vị trí xóa không tồn tại phần tử nào.
void push_front(const T& value); void push_front(T&& value);	Thêm phần tử mới vào đầu container.

Ví dụ

void pop_front();	Xóa bỏ phần tử đầu container. Nếu không có phần tử nào trong container, hành vi này được coi là không xác định. Việc tham chiếu hoặc dùng iterator trỏ đến một phần tử đã bị xóa là không hợp lệ.
size_type max_size() const noexcept;	Trả về kích thước tối đa có thể đạt được của forward_list nhưng không đảm bảo đạt được kích thước đó.
void resize(size_type n); void resize(size_type n, const value_type& val);	Co dãn kích thước của forward_list về n phần tử. Hàm này thay đổi các phần tử của forward_list bằng cách thêm hoặc xóa các phần tử vào/từ forward_list. Val là giá trị được dùng để thêm vào container nếu n > size hiện thời của container.

void merge(forward_list& other);
void merge(forward_list&& other);
void merge(forward_list& other, Compare comp);
void merge(forward_list& other, Compare comp);
	Trộn hai danh sách đã sắp xếp lại với nhau thành 1. Các danh sách cần được sắp xếp tăng dần. Sau khi thực hiện hành động này, other sẽ bị rỗng. comp là hàm thực hiện kiểm tra khớp đối tượng trả về bool == true khi tham số thứ 1 nhỏ hơn tham số thứ 2.

bool empty() const noexcept; (C++11 - C++20) [[nodiscard]] bool empty() const noexcept; (C++20)	Kiểm tra xem forward_list hiện có rỗng hay không. Return true nếu forward_list hiện không chứa phần tử nào.

Các hàm thông dụng và mô tả

void splice_after(const_iterator pos, forward_list& other);(1) void splice_after(const_iterator pos, forward_list&& other);(2) void splice_after(const_iterator pos, forward_list& other, const_iterator it);(3) void splice_after(const_iterator pos, forward_list& other, const_iterator first, const_iterator last);(4)	Chuyển các phần tử từ một danh sách khác vào danh sách hiện thời. (1), (2) chuyển tất cả các phần tử của other vào danh sách hiện tại. Chúng được chèn sau vị trí được chỉ ra bởi pos. Sau khi thực hiện, other rỗng. (3) Chuyển phần tử sau vị trí it đang trỏ tới vào sau vị trí pos của danh sách hiện thời. Nếu pos == it hoặc pos == ++it thì hàm không thực hiện gì cả. (3) Chuyển các phần tử trong khoảng (first, last) từ other vào sau vị trí trỏ tới bởi pos của danh sách hiện tại. Hành động trở thành không xác định nếu pos nằm trong khoảng (first, last).
void remove(const T& value);(C++11-C++20) size_type remove(const T& value);(C++20) template<class UnaryPredicate> void remove_if(UnaryPredicate p);(C++11-C++20) template<class UnaryPredicate> size_type remove_if(UnaryPredicate p);(C++20);	Xóa tất cả các phần tử thỏa mãn tiêu chí cụ thể nào đó. Phiên bản thứ nhất xóa tất cả các phần tử bằng value. Phiên bản thứ hai xóa tất cả các phần tử mà predicate p trả về true.
void reverse() noexcept;	Đảo ngược thứ tự các phần tử trong container.

Các hàm thông dụng và mô tả

void unique();(C++11-C++20) size_type unique();(C++20) template<class BinaryPredicate> void unique(BinaryPredicate p);(C++11-C++20) template<class BinaryPredicate> size_type unique(BinaryPredicate p);(C++20)	Xóa tất cả các phần tử trùng lặp khỏi danh sách. Chỉ những phần tử xuất hiện đầu tiên của nhóm được giữ lại. Phiên bản đầu sử dụng == để so sánh. Phiên bản thứ hai dùng predicate p. Danh sách cần được sắp xếp trước khi gọi hàm này.
void sort(); template<class Compare> void sort(Compare comp);	Sắp xếp các phần tử trong danh sách liên kết theo thứ tự tăng dần. Phiên bản đầu sử dụng toán tử < để so sánh các phần tử. Phiên bản sau sử dụng hàm comp trả về bool.

void assign(InputIterator first, InputIterator last);
void assign(size_type n, const T & val);
void assign(initializer_list<T> il);
	Gán nội dung mới cho forward_list, thay thế nội dung cũ bởi n phần tử có giá trị val hoặc nhân bản các phần tử trong nửa khoảng [first, last) của iterator truyền vào hoặc sử dụng các phần tử từ danh sách khởi tạo truyền vào.

Các hàm thông dụng và mô tả

iterator erase(const_iterator pos); iterator erase(const_iterator first, const_iterator last);	Xóa một phần tử đơn hoặc các phần tử trong nửa khoảng [first, last) khỏi forward_list.
void swap(forward_list& x);(C++11-C++17) void swap(forward_list& x) noexcept();(C++17)	Tráo đổi nội dung của forward_list với một forward_list x cùng kiểu. Sau khi tráo đổi, các phần tử của x sẽ thành các phần tử của forward_list hiện thời và ngược lại.
void clear() noexcept;	Xóa tất cả các phần tử hiện có trong forward_list. Sau khi gọi hàm này size của forward_list sẽ bằng 0.
Các hàm nạp chồng toán tử ==, !=, <, <=, >, >=(đã bị gỡ bỏ từ C++20)	Dùng để so sánh các phần tử trong 2 container.
operator <=> (từ C++20)	Dùng để so sánh nội dung của 2 container.

Nội dung tiếp theo

Tìm hiểu về ngăn xếp-stack