Bài 3.10: Giới thiệu lớp list

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

Khái niệm

- list là một container tuần tự cho phép chèn xóa phần tử ở vị trí bất kì trong nó với thời gian hằng số và cho phép duyệt theo cả hai chiều.
- list được triển khai như danh sách liên kết đôi. Các phần tử trong danh sách có thể được lưu trữ tại các vùng nhớ không liên tiếp nhau trong bộ nhớ.
- Lớp template list nằm trong namespace std. Để sử dụng list ta include thư viện <list> ở đầu file chương trình.
- Mỗi phần tử trong list có một liên kết trỏ đến phần tử kế tiếp và một liên kết trỏ đến phần tử liền trước của nó.

Đặc điểm

- Tương tự như danh sách liên kết đơn nhưng khác biệt nằm ở chỗ list có thể duyệt theo cả hai chiều do mỗi phần tử chứa 2 liên kết.
- So với các container tuần tự khác(array, vector, dequeue), 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 hỗ trợ truy cập trực tiếp qua chỉ số vào phần tử trong list như vector hay array.

Các hàm thông dụng và mô tả

Tên hàm	Mô tả
list();	Khởi tạo một list rỗng.
list(size_type n, const Allocator& alloc = Allocator());	Khởi tạo list với n phần tử mới ở kiểu T được chèn vào. Không sao chép các phần tử.
Template<class InputIt> list (InputIt first, InputIt last, const Allocator& alloc = Allocator());	Tạo một 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).
list (const list& other);	Tạo list và copy các phần tử trong other vào theo đúng trật tự các phần tử xuất hiện trong other.
list(list&&other);	Move constructor dùng để tạo một danh sách với nội dung là các phần tử của một danh sách khác.
list (const list& other, const Allocator& alloc);	Tạo container với các phần tử của other. Sử dụng alloc làm allocator.
list(initializer_list<T> init, const Allocator& alloc = Allocator());	Tạo một list với danh sách các phần tử cho trước.
operator=	Phép gán(copy hoặc move) dùng để thay thế nội dung của list bằng nội dung mới bên phải dấu =.
reference front(); const_reference front() const;	Trả về tham chiếu đến phần tử đầu tiên trong container. Gọi hàm này trên container rỗng sẽ cho kết quả không xác định.

Các hàm thông dụng và mô tả

reference back(); const_reference back() const;	Trả về tham chiếu đến phần tử cuối cùng trong container. Gọi hàm này trên container rỗng sẽ gây ra hành vi không xác định.

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 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 list. Nếu 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 list. Đây là iterator chỉ dùng để xác định sự kết thúc của list. Không phân giải tham chiếu với giá trị trả về bởi hàm này.

reverse_iterator rbegin() noexcept;
const reverse_iterator rbegin() const noexcept;
const reverse_iterator crbegin() const noexcept;
	Trả về một iterator ngược trỏ đến phần tử đầu của danh sách đảo ngược thứ tự phần tử. Đó là vị trí phần tử cuối trong danh sách hiện thời. Nếu danh sách rỗng, kết quả tương tự rend().

Các hàm thông dụng và mô tả

reverse_iterator rend() noexcept;
const reverse_iterator rend() const noexcept;
const reverse_iterator crend() const noexcept;
	Trả về một iterator đảo ngược trỏ tới phần tử cuối của danh sách đảo ngược thứ tự phần tử. Đó là vị trí trước phần tử đầu của danh sách hiện thời. Chức năng để làm mốc xét xem việc duyệt các phần tử trong danh sách đã kết thúc chưa.

iterator insert(const_iterator pos, const T& val);(1) iterator insert(const_iterator pos, const T&& val);(2) iterator insert(const_iterator pos, size_type n, const T& val);(3) template<class InputIt> iterator insert(const_iterator pos, InputIt first, InputIt last);(4) iterator insert(const_iterator pos, initializer_list<T> list);(5)	Chèn các phần tử vào trước vị trí pos. (1), (2) chèn 1 giá trị val vào trước vị trí pos. (3) Chèn n bản sao của val trước vị trí pos. (4) Chèn các phần tử trong nửa khoảng [first, last) vào trước vị trí pos. (5) Chèn các phần tử từ danh sách khởi tạo vào trước vị trí pos.
size_type size() const noexcept;	Trả về số lượng phần tử hiện có trong container.
void push_front(const T& value); void push_front(T&& value);	Thêm phần tử mới vào đầu container.
void push_back(const T& value);(1) void push_back(T&& value);(2)	Thêm một phần tử mới vào cuối danh sách hiện thời. (1) Tạo bản sao của value và sau đó chèn vào container. (2) value được di chuyển vào container thành phần tử mới.

Các hàm thông dụng và mô tả

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.
void pop_back();	Xóa bỏ phần tử cuối trong container. Gọi hàm này trên container rỗng sẽ gây ra hành vi không xác định.
size_type max_size() const noexcept;	Trả về kích thước tối đa có thể đạt được của 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 list về n phần tử. Hàm này thay đổi các phần tử của list bằng cách thêm hoặc xóa các phần tử vào/từ list. Val là giá trị được dùng để thêm vào cuối container nếu n > size hiện thời của container.

void merge(list& other);
void merge(list&& other);
void merge(list& other, Compare comp);
void merge(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 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 list hiện có rỗng hay không. Return true nếu list hiện không chứa phần tử nào.
void splice (const_iterator pos, list& other);(1)	Chuyển các phần tử từ một danh sách khác vào danh sách hiện thời. (1), (2) chuyển tất cả các phần tử của

Các hàm thông dụng và mô tả

void splice (const_iterator pos, list&& other);(2) void splice (const_iterator pos, list& other, const_iterator it);(3) void splice (const_iterator pos, list& other, const_iterator first, const_iterator last);(4) void splice (const_iterator pos, list&& other, const_iterator first, const_iterator last);(4)	other vào danh sách hiện tại. Chúng được chèn trước vị trí được chỉ ra bởi pos. Sau khi thực hiện, other rỗng. (3) Chuyển phần tử tại vị trí it đang trỏ tới trong other vào trước vị trí pos của danh sách hiện thời. Nếu pos == it hoặc pos == ++it thì hàm không thực hiện gì cả. (4) Chuyển các phần tử trong nửa khoảng [first, last) từ other vào trước 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);(to C++20) size_type remove(const T& value);(C++20) template<class UnaryPredicate> void remove_if(UnaryPredicate p);(to 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 có giá trị 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.
void unique();(to C++20)(1) size_type unique();(C++20)(1) template<class BinaryPredicate> void unique(BinaryPredicate p);(to C++20)(2) template<class BinaryPredicate> size_type unique(BinaryPredicate p);(C++20)(2)	Xóa tất cả các phần tử trùng lặp liên tiếp khỏi danh sách. Chỉ những phần tử xuất hiện đầu tiên của nhóm được giữ lại. Phiên bản (1) sử dụng == để so sánh. Phiên bản(2) dùng predicate p. Nếu muốn loại bỏ hết các phần tử trùng nhau liên tiếp ta cần sắp xếp trước khi gọi hàm này.

Các hàm thông dụng và mô tả

void sort();(1) template<class Compare> void sort(Compare comp);(2)	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 kiểu bool làm điều kiện so sánh các phần tử.
void assign(size_type n, const T& val);(1) void assign(InputIterator first, InputIterator last);(2) void assign(initializer_list<T> il);(3)	Gán nội dung mới cho list, (1)thay thế nội dung cũ bởi n phần tử có giá trị val hoặc (2)nhân bản các phần tử trong nửa khoảng [first, last) của iterator truyền vào hoặc thay thế nội dung của container hiện tại bằng các phần tử từ danh sách khởi tạo truyền vào.
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 list.
void swap(list& x);(until C++17) void swap(list& x) noexcept();(C++17)	Tráo đổi nội dung của list với một list x cùng kiểu. Sau khi tráo đổi, các phần tử của x sẽ thành các phần tử của list hiện thời và ngược lại.
void clear() noexcept;	Xóa tất cả các phần tử hiện có trong list. Sau khi gọi hàm này size của list sẽ bằng 0.
Các hàm nạp chồng toán tử !=, <, <=, >, >=(đã bị gỡ bỏ từ C++20), toán tử ==.	Dùng để so sánh các phần tử trong 2 container.
operator <=> (từ C++20)	Dùng để so sánh nội dung của 2 container.

Ví dụ

- Viết chương trình sử dụng list:
  - Thêm mới phần tử vào đầu list.
  - Thêm mới phần tử vào cuối list.
  - Sắp xếp list theo thứ tự giảm dần.
  - Xóa tất cả các phần tử có giá trị x khỏi list.

Nội dung tiếp theo

Giới thiệu lớp forward_list