Bài 2.12: Lớp vector

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

Định nghĩa và đặc điểm

- Vector là một container tuần tự đại diện cho mảng có thể co dãn kích thước.
- Lớp template vector nằm trong namespace std. Để sử dụng vector ta include thư viện <vector> ở đầu file chương trình.
- Giống như mảng, vector sử dụng các vùng nhớ được cấp phát liên tiếp nhau trong bộ nhớ để lưu trữ các phần tử của nó.
- Có thể truy cập trực tiếp vào bất kì phần tử nào của vector thông qua chỉ số phần tử như truy cập mảng.
- Khác với mảng, kích thước của vector có thể tự động thay đổi để phù hợp với ngữ cảnh sử dụng: tăng kích thước để lưu trữ thêm phần tử và giảm kích thước khi các phần tử bị xóa bớt khỏi vector.
- Để có được khả năng tự động thay đổi kích thước, vector sử dụng mảng cấp phát động để lưu trữ các phần tử của nó.
- Khi xảy ra việc cấp phát lại bộ nhớ, tất cả các phần tử trong mảng cũ sẽ được di chuyển sang mảng mới.

Đặc điểm

- Điều này sẽ tốn thời gian và phức tạp nên vector sẽ cấp phát thừa bộ nhớ để dành chỗ cho các phần tử kế tiếp sẽ được chèn vào. Do vậy khả năng lưu trữ của vector thường >= kích thước vector.
- Thư viện C++ có thể triển khai các chiến lược cấp phát bộ nhớ khác nhau cho vector để cân bằng giữa việc cấp phát lại và lượng bộ nhớ được sử dụng.
- Trong hầu hết các trường hợp, việc cấp phát lại bộ nhớ chỉ nên xảy ra theo hàm logarit của kích thước vector.
- Vector là lựa chọn hợp lý cho các ứng dụng yêu cầu truy cập nhanh vào các phần tử; hỗ trợ thêm và xóa phần tử ở cuối; cho phép co dãn kích thước container.
- Việc chèn và xóa phần tử tại các vị trí khác vị trí cuối vector không hiệu quả so với list và forward_list.

Các hàm thông dụng và mô tả

Tên hàm	Mô tả
vector()	Khởi tạo một vector rỗng.
vector(size_type n) vector(size_type n, const value_type& x)	Khởi tạo vector với n phần tử. Mỗi phần tử có giá trị x nếu tham số thứ 2 được chỉ rõ.
vector(InputIterator first, InputIterator last)	Tạo một vector 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).
vector(const vector& x)	Tạo vector và copy các phần tử trong x vào theo đúng trật tự các phần tử xuất hiện trong x.
operator=	Gán nội dung mới cho container, thay thế nội dung hiện tại của container vế trái = và tự sửa đổi kích thước mới tương ứng.
iterator begin() noexcept; const_iterator begin() const noexcept;	Trả về một iterator trỏ đến phần tử đầu của vector. Nếu vector rỗng, không nên phân giải tham chiếu giá trị trả về từ hàm này.
iterator end() noexcept; const_iterator end() const noexcept;	Trả về một iterator tham chiếu đến vị trí sau phần tử cuối của vector. Đây là iterator chỉ dùng để xác định sự kết thúc của vector, thường dùng kèm với vector::begin() và không nên phân giải tham chiếu. Nếu vector rỗng, hàm trả về giá trị tương tự vector::begin().

Các hàm thông dụng và mô tả

reverse_iterator rbegin() noexcept; const_reverse_iterator rbegin() const noexcept;	Trả về một iterator ngược, nó trỏ đến phần tử cuối trong vector. Iterator này duyệt theo chiều ngược từ cuối vector đến đầu vector. Khi ++ nó di chuyển đến phần tử liền trước của vector.
reverse_iterator rend() noexcept; const_reverse_iterator rend() const noexcept;	Trả về một iterator ngược trỏ đến vị trí trước phần tử đầu tiên của vector. Nửa khoảng [vector::rbegin(), vector::rend()) chứa tất cả các phần tử của vector.

const_iterator cbegin() const noexcept;
const_iterator cend() const noexcept;
const_iterator crbegin() const noexcept;
const_iterator crend() const noexcept;
	Trả về const_iterator tương ứng với từng hàm đã xét ở trên. Trong đó const_iterator là một iterator trỏ đến nội dung chỉ đọc, không được phép và không thể thay đổi. Được thêm vào từ phiên bản C++ 11.

size_type size() const noexcept;	Trả về số phần tử hiện thời của vector.
size_type max_size() const noexcept;	Trả về kích thước tối đa có thể đạt được của vector 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 vector về n phần tử. Hàm này thay đổi các phần tử của vector bằng cách thêm hoặc xóa các phần tử vào/từ vector.
size_type capacity() const noexcept;	Trả về kích thước vùng nhớ đã cấp phát hiện thời cho vector tính theo số lượng phần tử có thể chứa.
bool empty() const noexcept;	Kiểm tra xem vector hiện có rỗng hay không. Return true nếu vector hiện không chứa phần tử nào.

Các hàm thông dụng và mô tả

void reserve(size_type n);	Yêu cầu vector có khả năng chứa ít nhất n phần tử. Nếu n > capacity hiện tại của vector thì vector sẽ cấp phát lại. Hàm không làm ảnh hưởng đến các phần tử và số phần tử hiện có của vector.
void shrink_to_fit();	Yêu cầu vector giảm khả năng lưu trữ của nó xuống cho vừa size hiện tại của nó. Hàm có thể làm cho vector phải cấp phát lại nhưng không ảnh hưởng đến size và các phần tử hiện có.
reference operator[] (size_type n); const_reference operator[] (size_type n) const;	Trả về tham chiếu đến phần tử tại vị trí n trong vector. Không nên gọi toán tử này với n ngoài biên của mảng.
reference at (size_type n); const_reference at (size_type n) const;	Trả về tham chiếu đến phần tử tại vị trí n trong vector. Hàm tự động kiểm tra liệu n có nằm trong biên mảng hay không. Nếu n ngoài biên mảng nó ném ngoại lệ out_of_range exception. Hàm đối nghịch với toán tử []- không kiểm tra biên mảng.
reference front (); const_reference front () const;	Trả về tham chiếu trực tiếp đến phần tử đầu tiên trong vector. Khác với vector::begin() – trả về iterator trỏ đến phần tử đầu. Gọi hàm này với vector rỗng sẽ gây hành vi không xác định.
reference back (); const_reference back () const;	Trả về tham chiếu trực tiếp đến phần tử cuối cùng trong vector. Khác với vector::end() – trả về iterator trỏ đến vị trí sau phần tử cuối. Gọi hàm này với vector rỗng sẽ gây hành vi không xác định.

Các hàm thông dụng và mô tả

value_type* data() noexcept; const_ value_type* data() const noexcept;	Trả về tham chiếu trực tiếp đến mảng dùng để lưu trữ các phần tử trong vector.

void assign(InputIterator first, InputIterator last);
void assign(size_type n, const value_type& val);
void assign(initializer_list<value_type> il);
	Gán nội dung mới cho vector, thay thế nội dung cũ và sửa đổi kích thước tương ứng của nó.

void push_back(const value_type& val); void push_back(value_type&& val);	Thêm phần tử mới vào cuối vector. Nội dung của val có thể được sao chép hoặc di chuyển vào phần tử mới. Hành động này làm tăng số phần tử của vector lên 1. Có thể khiến vector cấp phát lại nếu sau khi thêm, capacity < size.
void pop_back();	Xóa phần tử cuối vector. Giảm số phần tử trong vector đi 1. Việc này sẽ hủy hoặc xóa bỏ phần tử ở cuối.

iterator insert(const_iterator pos, const value_type& val);
iterator insert(const_iterator pos, size_type n, const value_type& val);
iterator insert(const_iterator pos, InputIterator first, Iterator last);
iterator insert(const_iterator pos, const value_type&& val);
	Chèn thêm phần tử vào trước vị trí pos. Làm tăng số lượng phần tử lên 1 lượng bằng số lượng phần tử được thêm vào. Trong đó:
	- Vị trí trong vector cần chèn: pos.
	- Giá trị để chèn vào: val.
	- Iterator xác định đoạn các phần tử để chèn: first, last.

Các hàm thông dụng và mô tả

iterator insert(const_iterator pos, initializer_list<value_type> il);	- Danh sách đối tượng được khởi tạo trước để chèn thêm vào vector: il.
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 vector. Làm giảm số phần tử hiện có của vector. Thao tác xóa này trên vector không hiệu quả so với list hay forward_list vì liên quan sự cấp phát lại.
void swap(vector& x);	Tráo đổi nội dung của vector với một vector x cùng kiểu. Sau khi tráo đổi, các phần tử của x sẽ thành các phần tử của vector hiện thời và ngược lại.
void clear() noexcept;	Xóa tất cả các phần tử hiện có trong vector. Không đảm bảo capacity của vector có sự thay đổi.
Các hàm nạp chồng toán tử quan hệ ==, <, <=, >, >=, !=	Cho phép so sánh 2 vector cùng kiểu.

Ví dụ

```
const size_t SIZE = 10;
array<int, SIZE> intArray = {1, 2, 3, 6, 5, 4, 7, 89, 9, 0};
vector<int> vectorInt(intArray.begin(), intArray.end());
vector<int> other(vectorInt.begin(), vectorInt.begin() + 5);

ostream_iterator<int> output(cout, " ");
copy(vectorInt.begin(), vectorInt.end(), output);
cout << endl;

vectorInt[1] = 100;
vectorInt.at(3) = 250;
vectorInt.insert(vectorInt.begin() + 0, 900);

cout << "Sau khi thay doi, chen phan tu vao vector: \n";
copy(vectorInt.begin(), vectorInt.end(), output);
cout << endl;

cout << "Hien thi vector theo thu tu nguoc lai: \n";
copy(vectorInt.rbegin(), vectorInt.rend(), output);
cout << endl;

other.swap(vectorInt);
cout << "Hien thi vector theo thu tu nguoc lai sau khi trao doi: \n";
copy(vectorInt.rbegin(), vectorInt.rend(), output);
cout << endl;
```

Nội dung tiếp theo

Tìm hiểu về danh sách liên kết