

Lộ Trình Khám Phá: Làm Chủ C++ List

Từ Kệ Sách đến Đoàn Tàu Hỏa

Người Đồng Hành Learning How to Learn

Ngày 15 tháng 1 năm 2026

Lộ Trình Khám Phá

Chào bạn! Hôm nay chúng ta sẽ học cách làm "trưởng tàu" với `std::list`.

Lộ Trình: Làm Chủ C++ List

- **Chương 1: Cuộc Đối Đầu Giữa Đoàn Tàu và Kệ Sách**
Hiểu bản chất `std::list` vs `std::vector`.
- **Chương 2: Người Dẫn Đường Tự Động**
Làm quen với iterator và `auto`.
- **Chương 3: Hành Trình Kiểm Tra Từng Toa Tàu**
Duyệt danh sách bằng `while` và `for`.
- **Chương 4: Những Câu Thần Chú Của Trưởng Tàu**
Các thuật toán: `sort`, `merge`, `reverse`,...

Chương 1: Cuộc Đối Đầu Giữa Đoàn Tàu và Kệ Sách

Hãy tưởng tượng bộ nhớ máy tính là một sân chơi.

1. Vector: Chiếc Kệ Sách Gỗ Cố Định

- **Đặc điểm:** Các ô sách dính liền thành một khối.
- **Điểm mạnh:** Truy cập ngẫu nhiên siêu nhanh (Lấy ngay ô số 5).
- **Điểm yếu:** Chèn vào giữa rất mệt (Phải đẩy toàn bộ sách phía sau lùi lại).

2. List: Đoàn Tàu Hỏa Nối Đầu Nhau

Trong C++, nó là *Doubly Linked List*.

- **Toa tàu (Node):** Chứa dữ liệu.
- **Móc nối (Pointer):** Các toa nối với nhau bằng móc xích.

Tại sao "Đoàn Tàu"(List) thắng "Kệ Sách"(Vector)?

Sự linh hoạt

Khi chèn một toa mới vào giữa:

- ❶ Tháo móc xích giữa hai toa cũ.
- ❷ Móc toa mới vào giữa.
- ❸ Xong! Không cần di chuyển các toa khác.

Cái giá phải trả

- **Vector:** Nhanh khi tìm, chậm khi sửa đổi.
- **List:** Chậm khi tìm (phải đi bộ từ đầu tàu), siêu nhanh khi sửa đổi.

Bài toán Playlist Nhạc

Bạn viết ứng dụng nghe nhạc. Người dùng hay kéo bài hát từ cuối lên đầu, hoặc chèn bài mới vào giữa. Bạn chọn **Vector** hay **List**?

Bài toán Playlist Nhạc

Bạn viết ứng dụng nghe nhạc. Người dùng hay kéo bài hát từ cuối lên đầu, hoặc chèn bài mới vào giữa. Bạn chọn **Vector** hay **List**?

Đáp án: `std::list` (Đoàn tàu).

Giải thích: Việc đổi thứ tự bài hát chỉ đơn giản là tháo móc toa này gắn vào toa kia. Không cần bê vác nặng nhọc như vector.

Chương 2: Người Dẫn Đường Tự Động

Với List, bạn không thể gọi "Toa số 5". Bạn cần một **Iterator** (Người Soát Vé).

- Bắt đầu tại `begin()`.
- Bước sang toa kế tiếp bằng `++`.
- Soi đèn lấy dữ liệu bằng `*iterator`.
- Dừng lại tại `end()` (khoảng không sau toa cuối).

Sức mạnh của auto

Thay vì viết tên kiểu dài dòng, hãy dùng thẻ tên vụn vặt.

```
1 // Cách cũ: Dài dòng
2 std::list<int>::iterator nguoi_soat_ve = my_list.begin();
3
4 // Cách mới: Ngắn gọn, sạch đẹp
5 auto nguoi_soat_ve = my_list.begin();
```

Ví dụ Minh Họa: Iterator

```
1 #include <iostream>
2 #include <list>
3 using namespace std;
4
5 int main() {
6     // 1. Tao doan tau (List)
7     list<int> doan_tau = {10, 20, 30};
8
9     // 2. Thue nguoi soat ve (Iterator) dung auto
10    // Ong ay dang dung o toa dau tien (so 10)
11    auto nguoi_soat_ve = doan_tau.begin();
12
13    // 3. Soi den pin xem trong toa co gi (* de lay gia tri)
14    // In ra: Toa dau tien cho: 10
15    cout << "Toa dau tien cho: " << *nguoi_soat_ve << endl;
16
17    return 0;
18 }
```

Câu hỏi

Người Soát Vé đang đứng yên ở đầu tàu. Để ông ấy đi từ toa này sang toa kế tiếp, chúng ta dùng phép toán nào?

Câu hỏi

Người Soát Vé đang đứng yên ở đầu tàu. Để ông ấy đi từ toa này sang toa kế tiếp, chúng ta dùng phép toán nào?

Đáp án: Phép toán ++ (Increment).

Bạn ra lệnh `it++`, Người Soát Vé sẽ bước sang toa kế tiếp ngay lập tức.

Chương 3: Hành Trình Kiểm Tra Từng Toa Tàu

Cách 1: Thủ Công (While) - Hiểu rõ từng bước chân.

```
1 list<int> diem_so = {8, 9, 10};
2 auto it = diem_so.begin(); // Dung o dau tau
3
4 while (it != diem_so.end()) { // Chung nao chua rot khoi tau
5     cout << *it << " ";      // 1. Soi den pin
6     it++;                     // 2. Buoc sang toa ke tiep
7 }
```

Cách 2: Tự Động (Range-based for loop) - Băng chuyền siêu tốc.

```
1 // Voi moi 'x' nam TRONG 'diem_so'
2 for (auto x : diem_so) {
3     cout << x << " ";
4 }
```

Nguy Hiểm: Xóa Toa Tàu (Erase)

CẢNH BÁO CRASH

Nếu bạn xóa toa tàu hiện tại (`erase(it)`) nhưng vòng lặp `for` vẫn tự động `it++`, chương trình sẽ sập vì iterator bị treo lơ lửng.

Kỹ thuật "Nhảy tàu" (Safe Erase): Hàm `erase()` trả về địa chỉ toa kế tiếp. Hãy bám lấy nó!

```
1 list<int> doan_tau = {1, 5, 9, 5, 10};
2
3 // Phan 'buoc di' trong for de TRONG
4 for (auto it = doan_tau.begin(); it != doan_tau.end(); /* TRONG */
5     ) {
6     if (*it == 5) {
7         // Ky thuât "Nhay tau":
8         // Xoa toa hien tai va nhay sang toa ke tiep
9         it = doan_tau.erase(it);
10    }
11    else {
12        // Neu khong xoa, moi di bo
13        it++;
14    }
15 }
```



Câu Hỏi Kiểm Tra "Trưởng Tàu"

Tình huống

Danh sách: {2, 4, 6}. Bạn xóa số **6** (toa cuối) bằng lệnh `it = erase(it)`. Lúc này ông Soát Vé sẽ đứng ở đâu?

Câu Hỏi Kiểm Tra "Trưởng Tàu"

Tình huống

Danh sách: {2, 4, 6}. Bạn xóa số **6** (toa cuối) bằng lệnh `it = erase(it)`. Lúc này ông Soát Vé sẽ đứng ở đâu?

Đáp án: Ông ấy đứng ở `end()`.

Hàm `erase` trả về `end()` khi xóa phần tử cuối cùng. Vòng lặp kiểm tra thấy `it == end()` nên dừng lại an toàn.

Chương 4: Những Câu Thần Chú Của Trườg Tàu

Cái Bẫy Sắp Xếp (Sorting Trap)

`std::sort(list.begin(), list.end())` → **LỖI!** Lý do: List không hỗ trợ truy cập ngẫu nhiên.

Giải pháp: Dùng hàm thành viên

```
my_list.sort();
```

Bộ Ba Phép Thuật:

- **Sắp xếp:** `my_list.sort()`; (Bé → Lớn).
- **Đảo ngược:** `my_list.reverse()`; (Đầu ↔ Đuôi).
- **Lọc trùng:** `my_list.unique()`; (Chỉ lọc 2 toa **đứng cạnh nhau**).

Ví dụ Tổng Hợp Sức Mạnh

```
1 #include <iostream>
2 #include <list>
3 using namespace std;
4
5 int main() {
6     list<int> tau = {4, 2, 2, 5, 1, 5};
7
8     // 1. Sắp xếp trước
9     tau.sort();
10    // Tau thành: {1, 2, 2, 4, 5, 5}
11
12    // 2. Loại bỏ toa trùng nhau (khi đã xếp ke nhau)
13    tau.unique();
14    // Tau thành: {1, 2, 4, 5}
15
16    // 3. Đảo ngược lại
17    tau.reverse();
18    // Tau thành: {5, 4, 2, 1}
19
20    return 0;
21 }
```

Câu Hỏi Tốt Nghiệp (Final Boss)

Thử thách Unique

Danh sách: `list<int> my_list = {1, 2, 1, 2};` Chạy ngay: `my_list.unique();` (KHÔNG sort trước). Kết quả là gì?

- A. {1, 2}
- B. {1, 2, 1, 2}

Câu Hỏi Tốt Nghiệp (Final Boss)

Thử thách Unique

Danh sách: `list<int> my_list = {1, 2, 1, 2};` Chạy ngay: `my_list.unique();` (KHÔNG sort trước). Kết quả là gì?

- A. {1, 2}
- B. {1, 2, 1, 2}

Đáp án: B. {1, 2, 1, 2}

Lý do: `unique()` bị "cận thị", chỉ lọc được các phần tử đứng sát cạnh nhau. **Quy tắc vàng:** Luôn sort trước khi `unique`.

Nâng Cao: Sắp Xếp Giảm Dần

Muốn xếp từ Lớn về Bé, ta dùng `sort(greater...)`.

Lưu ý về cú pháp

Viết `greater<auto>()` → **SAI Cú pháp**.

Giải pháp: Transparent Operator (C++14)

Dùng `greater<>()` (Bỏ trống ngoặc nhọn). Đây là chiếc găng tay "Free Size".

```
1 #include <functional> // Bat buoc
2
3 list<int> diem = {5, 1, 9, 3};
4
5 // Cach cu (Phai chi dinh kieu):
6 // diem.sort(greater<int>());
7
8 // CACH MOI (Khuyen dung, ngan gon):
9 diem.sort(greater<>());
10
11 // Ket qua: 9, 5, 3, 1
```



Tổng Kết Hành Trình

Chúc mừng bạn đã hoàn thành khóa học cấp tốc về `std::list`!

Tóm tắt kho báu

- ❶ **Bản chất:** Đoàn tàu móc xích. Thêm/Xóa nhanh, Tìm kiếm chậm.
- ❷ **Duyệt:** Dùng `for(auto x : list)` hoặc Iterator.
- ❸ **Thao tác:** Nhớ kỹ thuật "Nhảy tàu" `it = erase(it)` để xóa an toàn.
- ❹ **Thuật toán:** Dùng đồ "chính chủ": `sort()`, `reverse()`, `unique()`.

Bạn đã sẵn sàng cho bài tập thực hành chưa?