Chinh Phục C++ Iterator

Hành trình của "Người Soát Vé"

Slide Learning C++

Ngày 14 tháng 1 năm 2026

1. Bức tranh toàn cảnh: Iterator là gì?

Phép ẩn dụ: Đoàn tàu hỏa

Hãy tưởng tượng `vector` trong C++ là một đoàn tàu chở đầy đồ chơi. Để kiểm tra từng món đồ, ta cần một **Người soát vé**.

- Người soát vé không phải là đoàn tàu.
- Người soát vé không phải là món đồ chơi.
- Người soát vé biết cách: **đứng tại một toa**, **xem bên trong**, và **bước tiếp**.

Định nghĩa

Iterator chính là "Người soát vé"giúp bạn duyệt qua danh sách mà không cần nhớ số thứ tự (index).

2. Giải mã Siêu năng lực (4 Lệnh cơ bản)

Để làm việc, "Người soát vé"cần 4 lệnh cơ bản:

1. `begin()`: Vạch xuất phát. Nhảy dù xuống **toa đầu tiên**.
2. `end()`: Biển báo "Vực thẳm".

Lưu ý quan trọng

`end()` **KHÔNG PHẢI** là toa cuối cùng. Nó là khoảng không **ngay sau** toa cuối cùng.

3. `*` (Dereference): Đôi mắt thần. "Mở cửa toa"để xem dữ liệu bên trong (ví dụ: `*it`).
4. `++` (Cộng cộng): Bước chân. Đi sang toa kế tiếp.

3. Code mẫu: Iterator truyền thống

Dưới đây là cách "Người soát vé"đi bộ từ đầu đến cuối tàu.

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main() {
  // 1. Tao doan tau (vector)
  vector<string> doan_tau = {"Pizza", "Ga ran", "Tra sua"};

  // 2. Tao nguoi soat ve (iterator)
  vector<string>::iterator nguoi_soat_ve;

  // 3. Bat dau hanh trinh
  for (nguoi_soat_ve = doan_tau.begin();
       nguoi_soat_ve != doan_tau.end();
       nguoi_soat_ve++) {

    // Dung mat than (*) de xem mon an
    cout << "Mon an: " << *nguoi_soat_ve << "\n";
  }
  return 0;
}
```

Thử thách tư duy

Câu hỏi kiểm tra

Nếu "Người soát vé"(Iterator) đang đứng ở vị trí `end()`, chú ấy có thể dùng "mắt thần"(*) để lấy dữ liệu được không? Tại sao?

Thử thách tư duy

Câu hỏi kiểm tra

Nếu "Người soát vé"(Iterator) đang đứng ở vị trí `end()`, chú ấy có thể dùng "mắt thần"(*) để lấy dữ liệu được không? Tại sao?

Câu trả lời

KHÔNG! Vì `end()` là vực thẳm (vạch vôi đỏ) sau toa cuối cùng. Ở đó không có toa tàu nào cả. Nếu cố tình dùng *, chương trình sẽ bị lỗi (crash).

4. Từ khóa auto và Băng chuyền tự động

Thay vì viết dài dòng `vector<string>::iterator`, ta dùng `auto`.

- `auto`: Chiếc kính thông minh, tự nhận diện kiểu dữ liệu.
- **Range-based for loop**: Biến đoàn tàu thành "Băng chuyền".

So sánh hình ảnh

- **Cách cũ:** Người soát vé đi bộ từng toa (`begin`, `++`, `*`, `end`).
- **Cách mới:** Bạn đứng yên, băng chuyền tự đẩy món đồ (`item`) đến trước mặt.

Code mẫu: Số tự động (Range-based for loop)

Cú pháp sạch và đẹp hơn rất nhiều:

```cpp
int main() {
  vector<string> doan_tau = {"Pizza", "Ga ran", "Tra sua"};

  // Dich nghia: "Voi moi 'mon an' nam trong 'doan tau'..."
  for (auto mon_an : doan_tau) {

    // O day 'mon_an' da la du lieu that roi!
    // Khong can dung dau * nua.
    cout << "Mon an tren bang chuyen: " << mon_an << "\n";
  }
  return 0;
}
```

5. So sánh nhanh

Đặc điểm	Cách cũ (Iterator)	Cách mới (Auto + Range)
Hình ảnh	Người đi bộ từng toa	Băng chuyền tự động
Độ dài	Rất dài, dễ sai	Ngắn gọn, súc tích
Quản lý	Phải lo `begin`, `++`	Máy tính lo hết
Kiểm soát	Biết rõ vị trí (địa chỉ)	Chỉ biết giá trị món đồ

Vấn đề nhỏ

Khi dùng băng chuyền, bạn biết đó là "Pizza", nhưng bạn **không biết** nó nằm ở toa số mấy. Nếu muốn kiểm soát vị trí, ta phải kết hợp `auto` với vòng lặp truyền thống.

6. Kết hợp: auto + Vòng lặp truyền thống

Dùng auto làm "thẻ tên tắc kè hoa"thay cho kiểu dữ liệu dài dòng, nhưng vẫn giữ cơ chế điều khiển thủ công.

```
1  vector<string> doan_tau = {"Pizza", "Ga ran"};
2
3  // 'auto' tu hieu la iterator
4  for (auto it = doan_tau.begin(); it != doan_tau.end(); it++) {
5
6    // Van can dung (*) vi it la con tro/dia chi
7    cout << "Mon an: " << *it << "\n";
8  }
```

Lợi ích

Giúp bạn thực hiện các thao tác nâng cao: Đi bước đôi (`it += 2`), đi lùi, hoặc sửa đổi dữ liệu tại chỗ.

7. Quyền năng thay đổi thực tại

Vì Iterator nắm giữ địa chỉ thật, khi bạn dùng $*$ để mở cửa, bạn có thể **THAY ĐỔI** món đồ bên trong.

```
for (auto it = doan_tau.begin(); it != doan_tau.end(); it++) {

    // Neu thay Ga ran
    if (*it == "Ga ran") {
        // PHEP THUAT: Bien hinh!
        *it = "Com tam";
    }
}
// Ket qua: Pizza, Com tam, Tra sua...
```

Phép ẩn dụ

Khác với xem TV (chỉ nhìn), Iterator cho phép bạn bước vào bếp và đổi cái bánh Pizza thành Bánh Mì.

8. Cái bẫy `const auto`

Muốn tạo iterator chỉ đọc (Read-only), nhiều bạn dùng `const auto it`. Đây là sai lầm!

Cái bẫy: Chân bị xích

`const auto it = ...` nghĩa là: "Tạo ra một iterator và **đóng băng** nó".

- **Hậu quả:** Bạn không thể thực hiện `it++` (bước đi).
- **Vòng lặp:** Báo lỗi ngay lập tức.

8. Cái bẫy `const auto`

Muốn tạo iterator chỉ đọc (Read-only), nhiều bạn dùng `const auto it`. Đây là sai lầm!

Cái bẫy: Chân bị xích

`const auto it = ...` nghĩa là: "Tạo ra một iterator và **đóng băng** nó".

- **Hậu quả:** Bạn không thể thực hiện `it++` (bước đi).
- **Vòng lặp:** Báo lỗi ngay lập tức.

Giải pháp: `cbegin()`

Muốn tạo "Khách tham quan"(đi được nhưng không sửa được), hãy dùng `cbegin()` và `cend()`. Khi đó `auto` sẽ trở thành `const_iterator`.

Code mẫu: cbegin() (An toàn tuyệt đối)

```cpp
vector<string> doan_tau = {"Pizza", "Ga ran"};

// Dung cbegin (Const Begin)
for (auto it = doan_tau.cbegin(); it != doan_tau.cend(); it++) {

  // 1. Doc: OK
  cout << *it << "\n";

  // 2. Sua: LOI!
  // *it = "Bun dau";  <-- May tinh se bao loi ngay
}
```

9. Phép thuật: Chèn toa (`insert`)

Quy tắc Cần Cẩu Khổng Lồ

Lệnh `insert(it, "Món mới")` sẽ:

1. Đẩy lùi toa hiện tại và các toa sau.
2. Thả toa mới vào **PHÍA TRƯỚC** vị trí iterator đang đứng.

Cảnh báo động đất (Iterator Invalidation)

Sau khi chèn, đường ray bị xê dịch. Iterator cũ (`it`) có thể bị hỏng. **Lời khuyên:** Cập nhật lại iterator nếu muốn dùng tiếp.

Check-point: Thứ tự chèn

Câu hỏi

Đoàn tàu: `{"A", "B", "C"}`.
Iterator `it` đang trỏ vào **"A"**.
Gọi lệnh: `vector.insert(it, "Z");`
Thứ tự mới là gì?

Check-point: Thứ tự chèn

Câu hỏi

Đoàn tàu: `{"A", "B", "C"}`.
Iterator `it` đang trỏ vào **"A"**.
Gọi lệnh: `vector.insert(it, "Z");`
Thứ tự mới là gì?

Đáp án

{"Z", "A", "B", "C"}
(Chèn vào **trước** A).

10. Hủy diệt (erase) và Cú nhảy lò xo

Lệnh `erase(it)` xóa phần tử tại vị trí `it`.

Quy tắc sinh tồn

Sau khi xóa toa tàu, iterator hiện tại sẽ rơi xuống vực. `erase` sẽ trả về vị trí của **người kế tiếp** (cơ chế lò xo). Phải luôn hứng lấy nó: `it = v.erase(it);`

```
1     auto it = v.begin() + 1; // Dang o B
2     // Xoa B, it tu dong nhay sang C
3     it = v.erase(it);
```

11. Cú nhảy cóc tai hại (Logic Trap)

Khi xóa trong vòng lặp, cẩn thận với `it++` ở tiêu đề vòng lặp.

Kịch bản lỗi

1. `erase` tự đẩy `it` sang phần tử kế tiếp.
2. Vòng lặp `for` lại thực hiện `it++` thêm lần nữa.
3. **Kết quả:** Bạn nhảy qua đầu một phần tử mà không kiểm tra (Double Jump).

Code sai:

```
1 // SAI: Vua erase day, vua it++ day -> Nhay coc
2 for (auto it = v.begin(); it != v.end(); it++) {
3     if (*it % 2 == 0) it = v.erase(it);
4 }
```

Giải pháp: Điều khiển thủ công

Đưa `it++` vào trong thân vòng lặp để kiểm soát.

```
    // CHUAN: Bo it++ o tieu de
    for (auto it = v.begin(); it != v.end(); /* Trong */ ) {

        if (*it % 2 == 0) {
            // Truong hop XOA: erase tu day it sang ke tiep
            it = v.erase(it);
        } else {
            // Truong hop KHONG XOA: Tu buoc di
            it++;
        }
    }
```

Tổng kết hành trình

Chúng ta đã tốt nghiệp khóa học "Người soát vé"!

1. **Iterator**: Con trỏ thông minh duyệt mảng.
2. `begin/end`: Điểm đầu và Vực thẳm.
3. `auto`: Thẻ tên tàng hình cho code gọn.
4. `cbegin`: Chế độ "Khách tham quan"(Chỉ đọc).
5. `insert`: Chèn phía trước (Cẩn thận động đất).
6. `erase`: Xóa và hứng lấy vị trí mới (Tránh nhảy cóc).

Bước tiếp theo

Bạn đã sẵn sàng để khám phá các thuật toán sắp xếp và tìm kiếm với Iterator chưa?