

Bài 52: CƠ BẢN VỀ LỚP VECTOR

Xem bài học trên website để ủng hộ Kteam: [Cơ bản về lớp Vector](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

Dẫn nhập

Ở bài học trước, mình đã chia sẻ cho các bạn khái niệm [CON TRỎ TRỎ ĐẾN CON TRỎ \(Pointers to pointers\)](#) trong C++.

Lưu ý : Bạn nên tránh sử dụng con trỏ trỏ đến con trỏ trừ khi không có giải pháp nào khác.

Hôm nay, mình sẽ giới thiệu cho các bạn một cách để sử dụng mảng động mà không cần thao tác quá phức tạp bằng con trỏ, đó là **Cơ bản về `std::vector`**.

Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về:

- [TỪ KHÓA AUTO TRONG C++11 \(The auto keyword\)](#)
- [VÒNG LẶP FOR EACH TRONG C++11 \(For each loops\)](#)
- [MẢNG MỘT CHIỀU \(Arrays\)](#)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Tổng quan về lớp `std::vector`
- Cơ chế ngăn chặn rò rỉ bộ nhớ của `std::vector`
- Một số thao tác với mảng kiểu `std::vector`

Tổng quan về lớp std::vector

Trong bài học [LỚP DUNG SẴN ARRAY](#), chúng ta đã biết lớp **std::array** dùng để giải quyết những vấn đề về quản lý và sử dụng mảng tĩnh.

Tương tự, C++ cung cấp lớp **std::vector** hỗ trợ việc thao tác với mảng động an toàn và dễ dàng hơn. Với lớp **std::vector**, bạn **có thể tạo các mảng động** mà **không cần phải cấp phát và thu hồi vùng nhớ** bằng cách sử dụng toán tử **new** và **delete**.

Để sử dụng lớp std::vector, bạn cần khai báo thư viện và namespace:

```
#include <vector>
using namespace std;
// không cần cung cấp độ dài mảng tại thời điểm biên dịch
vector<int> array;
vector<int> array2 = { 9, 7, 5, 3, 1 };
vector<int> array3{ 9, 7, 5, 3, 1 };
```

Giống như **std::array**, việc truy cập các phần tử mảng có thể được thực hiện thông qua toán tử **[]** (không kiểm tra phạm vi mảng) hoặc hàm **at()** (có kiểm tra phạm vi mảng):

```
array[1] = 5; // không kiểm tra phạm vi mảng
array.at(1) = 5; // có kiểm tra phạm vi mảng
```

Cơ chế ngăn chặn rò rỉ bộ nhớ của std::vector

Khi một biến vector **ra khỏi phạm vi** được định nghĩa, nó sẽ **tự động giải phóng vùng nhớ** mà nó nắm giữ. Điều này không chỉ tiện dụng (vì bạn không phải tự làm điều đó), nó còn giúp ngăn ngừa rò rỉ bộ nhớ.

Xét hàm dưới đây:

```
void doSomething(bool earlyExit)
{
    int *array = new int[3]{ 1, 3, 2 };

    if (earlyExit) // thoát khỏi hàm
        return;

    delete[] array; // trường hợp hàm thoát sớm, array sẽ không bị xóa
}
```

Tuy nhiên, nếu biến array là kiểu vector, bộ nhớ sẽ được giải phóng ngay khi thoát khỏi hàm. Điều này làm cho **std::vector** ngăn chặn được việc rò rỉ bộ nhớ.

Một số thao tác với mảng kiểu **std::vector**

Xem kích thước của mảng kiểu **std::vector**

Để xem kích thước mảng gồm bao nhiêu phần tử, sử dụng hàm **size()**:

```
vector<int> arr = { 2, 5, 8, 3, 1 };
cout << arr.size() << endl;
```

Output: 5

Thay đổi kích thước mảng kiểu **std::vector**

Thay đổi kích thước một mảng được cấp phát động rất phức tạp. Tuy nhiên, điều này rất đơn giản đối với **std::vector** thông qua hàm **resize()**:

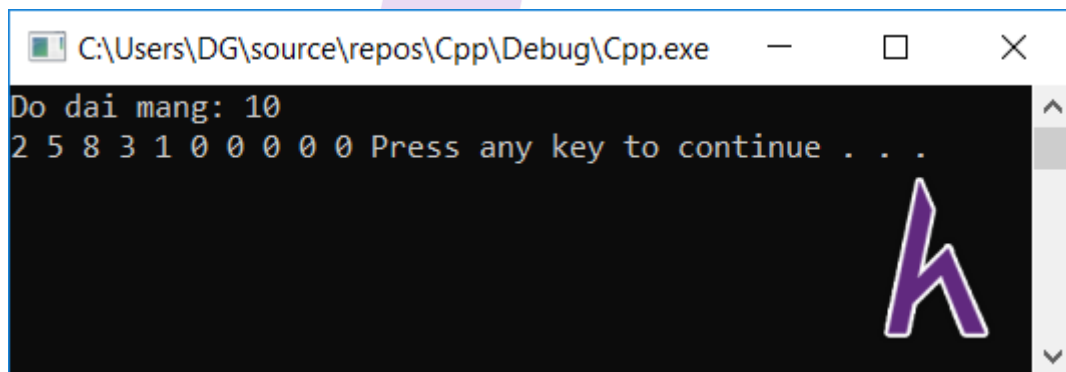
```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector<int> array{ 2, 5, 8, 3, 1 }; // mảng 5 phần tử
    array.resize(10); // thay đổi độ dài mảng thành 10 phần tử

    cout << "Do dai mang: " << array.size() << '\n';

    for (auto const &item : array)
        cout << item << ' ';

    system("pause");
    return 0;
}
```



Chú ý: Khi thay đổi kích thước mảng **std::vector**, các giá trị phần tử hiện có được giữ nguyên. Các phần tử mới được khởi tạo bằng giá trị mặc định của kiểu dữ liệu mảng.

Các vector có thể được thay đổi kích thước để nhỏ hơn:

```
vector<int> array{ 2, 5, 8, 3, 1 }; // mảng 5 phần tử
array.resize(3); // 2 5 8, phần tử thứ 4 trở đi sẽ bị cắt bỏ
```

Dung lượng và kích thước của std::vector

Không giống như mảng thông thường hoặc mảng dựng sẵn **std::array** (chỉ chứa kích thước mảng), **std::vector** chứa hai thuộc tính riêng biệt: **kích thước (size)** và **dung lượng (capacity)**.

- **Kích thước (size)** trả về số lượng phần tử đang được sử dụng trong mảng.
- **Dung lượng (capacity)** trả về số lượng phần tử được cấp phát cho vector trong bộ nhớ.

Ví dụ:

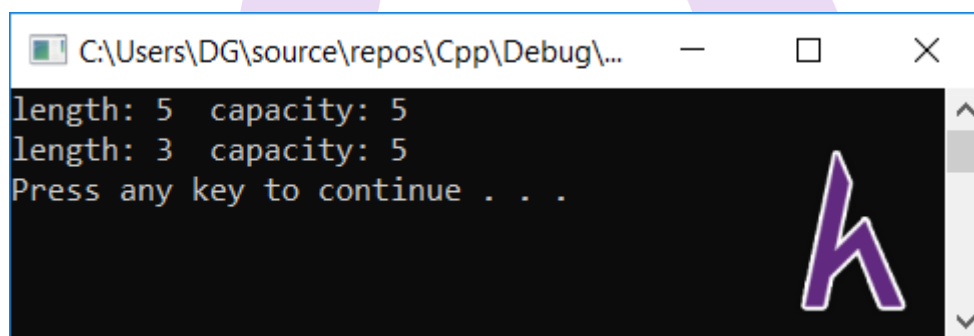
```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector<int> array;
    array = { 0, 1, 2, 3, 4 }; // length = 5, cap = 5
    cout << "length: " << array.size() << " capacity: " << array.capacity() << '\n';

    array = { 9, 8, 7 }; // length = 3, cap = 5
    cout << "length: " << array.size() << " capacity: " << array.capacity() << '\n';

    system("pause");
    return 0;
}
```

Output:



Thay đổi kích thước một mảng rất tốn kém về mặt tính toán, vì vậy **dung lượng (capacity)** sinh ra để giảm thiểu việc cấp phát lại vùng nhớ khi thay đổi kích thước mảng.

Chúng ta có thể cấp phát một **dung lượng (capacity)** ban đầu cho **std::vector** bằng hàm **reserve()**:

```
vector<int> array;  
array.reserve(5); // đặt dung lượng vector là 5
```

Hành vi ngăn xếp trên std::vector

Mặc dù **std::vector** có thể được sử dụng như một mảng động, nhưng nó cũng có thể được sử dụng như một ngăn xếp. **Std::vector** cung cấp 3 phương thức:

- **push_back()** thêm một phần tử vào cuối vector.
- **back()** trả về giá trị của phần tử cuối vector.
- **pop_back()** xóa một phần tử cuối vector.

Ví dụ:

```
#include <iostream>  
#include <vector>  
using namespace std;  
  
void printStack(const vector<int> &stack)  
{  
    for (const auto &element : stack)  
        cout << element << ' ';  
    cout << "(cap " << stack.capacity() << " length " << stack.size() << ")\n";  
}  
  
int main()  
{  
    vector<int> stack;  
  
    stack.push_back(5); // push_back() thêm một phần tử vào cuối vector  
    printStack(stack);  
  
    stack.push_back(3);  
    printStack(stack);  
  
    stack.push_back(2);  
    printStack(stack);  
}
```

```
// back() trả về giá trị của phần tử cuối
cout << "Phan tu cuoi: " << stack.back() << '\n';

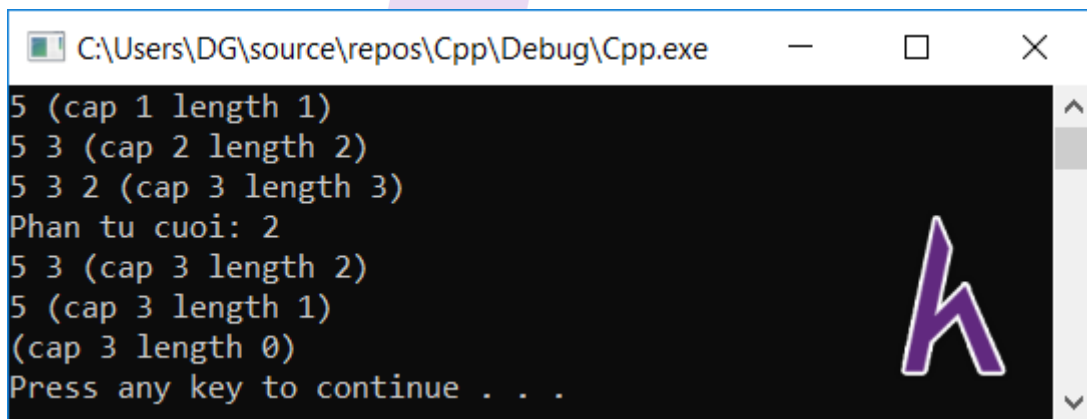
stack.pop_back(); // pop_back() xóa một phần tử cuối vector
printStack(stack);

stack.pop_back();
printStack(stack);

stack.pop_back();
printStack(stack);

system("pause");
return 0;
}
```

Output:



Kết luận

Qua bài học này, bạn đã nắm được cơ bản về Cơ bản về `std::vector` trong C++. Vì `std::vector` hỗ trợ việc thao tác với mảng động an toàn và dễ dàng hơn, nên bạn nên sử dụng `std::vector` trong hầu hết các trường hợp cần mảng động.

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn những [TRUYỀN ĐỊA CHỈ CHO HÀM](#) (Passing arguments by address) trong C++.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **"Luyện tập – Thử thách – Không ngại khó"**.

