Bài: Lớp dựng sẵn Array trong C++11

Xem bài học trên website để ủng hộ Kteam: Lớp dựng sẵn Array trong C++11

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 về VÒNG LẶP FOR EACH TRONG C++11 (For each loops), nó được sử dụng để lặp qua các phần tử trên 1 mảng (hoặc các cấu trúc danh sách khác như vectors, linked lists, trees, và maps)

Hôm nay, mình sẽ giới thiệu cho các bạn về **Lớp std::array trong C++11**.

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 TRONG C++ (For statements)
- MẢNG 1 CHIỀU (Arrays)

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Tổng quan về lớp std::array trong C++11

Tổng quan về lớp std::array trong C++11

Trong bài học MẢNG 1 CHIỀU (Arrays), bạn đã biết cách sử dụng mảng 1 chiều trong C++. Mảng 1 chiều là mảng tĩnh, và nó có 1 số khuyết điểm: mảng trở thành con trỏ và mất thông tin chiều dài khi truyền vào hàm, không có nhiều hàm hỗ trợ sẵn.

Để giải quyết những vấn đề về quản lý và sử dụng mảng tĩnh, thư viện chuẩn C++ đã cung cấp lớp **std::array** được khai báo trong thư viện **<array>** thuộc **namespace std**.

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

C++:

```
#include <array>
using namespace std;
```

Khai báo và khởi tạo mảng kiểu std::array

Khai báo mảng kiểu std::array

Khi khai báo 1 biến kiểu **std::array**, bạn cần xác định kiểu dữ liệu và số phần tử của mảng:

C++:

```
array <int, 5> arr; // mảng arr có 5 phần tử kiểu int
array <string, 10> arr2; // mảng arr2 có 10 phần tử kiểu string
```

Tương tự như mảng 1 chiều tĩnh, số phần tử của **std::array** phải được xác định cụ thể (hằng số) khi khai báo.

> **Chú ý:** Khi chưa khởi tạo, các phần tử của mảng sẽ mang giá trị rác.

Khởi tạo giá trị cho mảng kiểu std::array

Có thể khởi tạo cho mảng kiểu **std::array** như bên dưới:

C++:

```
array <int, 5> arr = { 2, 5, 8, 3, 1 };
array <int, 5> arr2{ 2, 5, 8, 3, 1 }; // khởi tạo đồng nhất (C++11)

array <int, 5> arr3;
arr3 = { 2, 5, 8, 3, 1 }; // khởi tạo đầy đủ các phần tử
arr3 = { 2, 5, 8 }; // khởi tạo 3 phần tử đầu, còn lại là 0
```

Không khởi tạo nhiều giá trị hơn số lượng phần tử đã khai báo:

C++:

```
array <int, 3> arr = { 2, 5, 8, 3, 1 }; // lỗi biên dịch
```

Không giống như khai báo mảng 1 chiều tĩnh, bạn không thể bỏ qua số lượng phần tử mảng khi khai báo:

C++:

```
array <int, > arr = { 2, 5, 8, 3, 1 }; // lỗi biên dịch
```

Truy cập phần tử trong mảng kiểu std::array

Để truy cập phần tử trong mảng kiểu **std::arra**y, bạn sử dụng toán tử **[]** tương tự như mảng 1 chiều tĩnh:

C++:

```
array <int, 5> arr = { 2, 5, 8, 3, 1 };
arr[1] = 3; // gán 3 cho phần tử thứ 2
cout << arr[4]; // truy xuất giá trị phần tử thứ 5
```

Tương tự như trên mảng 1 chiều tĩnh, toán tử **[]** không thực hiện kiểm tra phạm vi của mảng. Truy cập 1 phần tử với chỉ số không hợp lệ sẽ cho kết quả không như mong muốn (có thể gây chết chương trình).

Lớp **std::array** cung cấp hàm **at()** để truy cập vào phần tử mảng, nó bao gồm việc kiểm tra phạm vi của mảng:

C++:

```
array <int, 5> arr = { 2, 5, 8, 3, 1 };
arr.at(1) = 3; // gán 3 cho phần tử thứ 2
cout << arr.at(6); // ngoài phạm vi mảng, ném ra 1 lỗi và kết thúc chương trình
```

Nếu bạn có chắc chắn rằng chỉ số nằm trong phạm vi mảng mà không cần kiểm tra trong thời gian chạy, bạn có thể sử dụng toán tử **[]**. Nếu không, sử dụng hàm **at()**, hoặc thêm kiểm tra của riêng bạn trước khi truy cập phần tử mảng.

> **Chú ý:** Vì toán tử **[]** không kiểm tra phạm vi của mảng, nên hàm **at()** sẽ chậm hơn (nhưng an toàn hơn) so với toán tử **[]**.

Một số thao tác với mảng kiểu std::array

Xem kích thước của mảng kiểu std::array

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

C++:

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

Output: 5

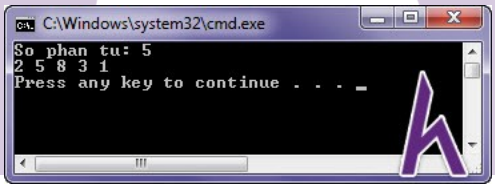
Truyền mảng kiểu std::array vào hàm

Không giống như mảng 1 chiều tĩnh, mảng kiểu **std::array** không chuyển thành con trỏ khi truyền vào hàm. Vì vậy, hàm **size()** và **for-each loops** vẫn hoạt động bên trong hàm khác:

C++:

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

#define MAX 5

void printArray(const array<int, MAX> &arr) // truyền tham chiếu (hằng) vì hiệu suất
{
	cout << "So phan tu: " << arr.size() << endl; // có thể biết số phần tử ở mọi hàm
	for (const auto &item : arr) // có thể sử dụng for-each loops ở mọi hàm
	{
		cout << item << ' ';
	}
	cout << endl;
}

int main()
{
	array <int, MAX> arr = { 2, 5, 8, 3, 1 };
	printArray(arr);

	return 0;
}
```

Output:



Quy tắc: Khi truyền mảng kiểu **std::array** vào hàm, luôn sử dụng **tham chiếu (hoặc tham chiếu hằng) vì lý do hiệu suất**.

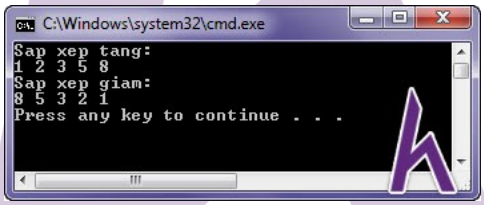
Sắp xếp mảng kiểu std::array

Mảng kiểu **std::array** có thể được sắp xếp bằng cách sử dụng hàm **std::sort()** được khai báo trong thư viện **<algorithm>**:

C++:

```
#include <iostream>
#include <array>
#include <algorithm>
using namespace std;

#define MAX 5

void printArray(const array<int, MAX> &arr)
{
	for (const auto &item : arr)
	{
		cout << item << ' ';
	}
	cout << endl;
}

int main()
{
	array <int, MAX> arr = { 2, 5, 8, 3, 1 };

	cout << "Sap xep tang: \n";
	sort(arr.begin(), arr.end()); // sắp xếp tăng
	printArray(arr);

	cout << "Sap xep giam: \n";
	sort(arr.rbegin(), arr.rend()); // sắp xếp giảm
	printArray(arr);

	return 0;
}
```

Output:



Trong chương trình trên, hàm **std::sort()** nhận vào 2 tham số là **iterator** đầu mảng (**arr.begin()**) và **iterator** cuối mảng (**arr.end()**) để sắp xếp mảng tăng.

Để sắp xếp mảng giảm, bạn sử dụng **reverse iterator** đầu mảng (**arr.rbegin()**) và **reverse iterator** cuối mảng (**arr.rend()**).

> **Lưu ý:** iterator là một đối tượng, trỏ đến 1 phần tử trong danh sách các phần tử (ví dụ như 1 mảng hoặc 1 container).

Kết luận

Qua bài học này, bạn đã biết được cách sử dụng Lớp std::array trong C++11. Từ C++11, lớp std::array thường được ưu tiên sử dụng thay thế cho mảng tĩnh, vì nó được hỗ trợ sẵn nhiều phương thức và dễ dàng quản lý hơn.

Trong bài tiếp theo, mình sẽ giới thiệu cho các bạn LỚP std::vector 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**".