Bài 2.2: Mảng một chiều

- ✓ Định nghĩa, cú pháp
- ✓ Khởi tạo, duyệt mảng
- ✓ Mảng staic
- ✓ Cấp phát động mảng 1 chiều
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Định nghĩa

- Mảng là tập hợp của một nhóm các vùng nhớ được cấp phát liên tiếp nhau trong bộ nhớ. Tất cả các vùng nhớ này có cùng kiểu và được đặt dưới một cái tên chung là tên mảng.
- Mảng là loại tập hợp có kích thước cố định kể từ sau khi nó được tạo ra tới khi bị hủy đi.
- Để truy cập đến một phần tử mảng, ta sử dụng tên mảng và chỉ số phần tử mảng.
- Trong đó chỉ số của phần tử mảng luôn bắt đầu từ 0 kết thúc ở n-1. Với n là số phần tử tối đa mà mảng có thể chứa.
- Ví dụ về mảng:

16	12	-91	7	80	1004	20	43	67	-88
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]	arr[7]	arr[8]	arr[9]

Cú pháp tổng quát

Cú pháp tổng quát: **type arrayName [numOfElements];**

- Type: là kiểu của mảng. Có thể là bất kì kiểu hợp lệ nào trong C++ như int, float, bool…
- **arrayName**: là tên của tập hợp mà mảng đại diện, thường sử dụng danh từ số nhiều. **[]**
- Cặp móc vuông là dấu hiệu nhận biết biến kiểu mảng.
- **numOfElements**: số phần tử có thể chứa của mảng, luôn là số nguyên dương.
- Kết thúc khai báo biến kiểu mảng bằng dấu chấm phẩy **;**
- Khuyến nghị chỉ khai báo một biến trên một dòng cho dễ đọc và tránh nhầm lẫn.
- Ví dụ:

```
int numbers[10]; // mảng chứa các số nguyên
float grades[15]; // mảng chứa các đầu điểm
char name[20];    // mảng chứa các kí tự cấu thành tên
```

Truy xuất phần tử mảng

- Khi truy xuất phần tử mảng, cần đảm bảo chỉ số mảng nằm trong đoạn [0, n-1] để tránh các lỗi không đáng có.
- Ví dụ sau truy cập phần tử mảng ở một số vị trí.

```
numbers[0] = 120; // gán giá trị cho phần tử tại vị trí 0
numbers[2] = -9;  // gán giá trị cho phần tử tại vị trí 2
cout << numbers[0] << endl; // in ra giá trị phần tử tại vị trí 0
```

Khởi tạo giá trị cho các phần tử mảng

- Mảng có thể được khởi tạo ngay khi khai báo.
- Trong đó các phần tử khởi tạo sẽ được liệt kê trong cặp ngoặc {} phân tách nhau bằng dấu phẩy.
- Kết thúc quá trình khởi tạo luôn có dấu chấm phẩy;
- Khi ta không cung cấp đủ số phần tử đã chỉ định trong móc vuông, tất cả các phần tử còn thiếu sẽ điền giá trị mặc định của kiểu của mảng.
- Giá trị mặc định của kiểu:
  - Các kiểu số là 0.
  - Kiểu bool là false.
  - Các kiểu tham chiếu là NULL.

```
int myArr[] = { 1, 2, 3, 4, 5 }; // khoi tao mang 5 phan tu
int myNumbers[10] = { 0 }; // khoi tao tat ca cac phan tu trong
mang bang 0
```

Khởi tạo giá trị cho các phần tử mảng

- Lưu ý không gán giá trị khác kiểu cho phần tử mảng để tránh lỗi và các kết quả không mong muốn.
- Ví dụ:

```
int numbers[10];
numbers[0] = 1; // ok
numbers[1] = "Hello"; // error!
numbers[2] = 'H'; // ok but invalid
numbers[3] = true; // ok but invalid
```

Duyệt mảng

- Thường sử dụng kết hợp vòng lặp for và mảng một chiều.
- Vòng lặp này có biến kiểm soát lặp nên ta dễ dàng quản lý chỉ số phần tử mảng dựa vào biến kiểm soát lặp.
- Kiểu của biến kiểm soát lặp nên để là unsigned nhằm hạn chế trường hợp ta cung cấp chỉ số mảng âm cho chương trình
- Nếu đã biết trước hoặc muốn giới hạn số phần tử tối đa, ta nên định nghĩa một hằng số thể hiện giá trị này.

```
const unsigned int SIZE = 5;
int numbers[SIZE] = {1,2,3,4,5};
cout << left << setw(10) << "Vi tri"
      << setw(10) << "Gia tri" << endl;
for (size_t i = 0; i < SIZE; i++)
{
     cout << left << setw(10) << i
          << setw(10) << numbers[i] << endl;
}
```

Mảng static

Khi nói về mảng static, chúng ta đề cập chủ yếu là mảng cục bộ. Một số đặc điểm của nó như sau:

- Giữ nguyên trạng thái của lần thay đổi ngay trước đó kể cả khi chương trình đã thoát khỏi khối, hàm chứa nó.
- Giá trị của các phần tử trong mảng static tự động được khởi tạo bằng giá trị mặc định của kiểu mà mảng đó đại diện nếu bạn không khởi tạo các giá trị tường minh cho nó.
- Những điều trên không xảy ra với mảng cục bộ thông thường.

Ví dụ

```
void callToStaticArr()
{
	static int arr1[SIZE];
	cout << "Values of static array before changed: " << endl;
	for (size_t i = 0; i < SIZE; i++)
	{
		cout << setw(10) << arr1[i];
	}

	cout << "\nValue of static array after changed: " << endl;
	for (size_t i = 0; i < SIZE; i++)
	{
		arr1[i] += 5;
		cout << setw(10) << arr1[i];
	}
}
```

```
First call to two functions:
Values of static array before changed:
     0     0     0     0     0
Value of static array after changed:
     5     5     5     5     5
```

```
Second call to two functions:
Values of static array before changed:
     5     5     5     5     5
Value of static array after changed:
    10    10    10    10    10
```

Cấp phát động mảng 1 chiều

- Về bản chất, tên mảng chính là một con trỏ cố định luôn trỏ tới phần tử đầu tiên trong mảng.
- Do đó ta có thể sử dụng con trỏ thay thế cho mảng trong chương trình.
- Đặc trưng của mảng là một khi mảng được tạo thì kích thước của nó không thể thay đổi.
- Do đó nếu ta sử dụng mảng có thể xảy ra các khả năng: thừa, thiếu, đủ chỗ cho các phần tử.
- Để tránh việc thừa thiếu chỗ chứa các phần tử mảng, ta sử dụng cấp phát động.
- Cú pháp tổng quát: **type* arrayName = new type[size]();**
- Nếu muốn khởi tạo giá trị cho các phần tử:
  type* arrayName = new type[size]{e1, e2, …};

Cấp phát động mảng 1 chiều

- Cú pháp tổng quát: **type* arrayName = new type[size]();**
- Nếu muốn khởi tạo giá trị cho các phần tử:
  type* arrayName = new type[size]{e1, e2, …};
- Trong đó:
  - **Type:** là kiểu dữ liệu hợp lệ trong C++. Có thể là các kiểu nguyên thủy, kiểu tham chiếu hoặc bất kì kiểu người dùng tự định nghĩa nào.
  - **arrayName:** là tên con trỏ. Cấp phát động thường liên quan đến mảng nên ta đặt tên mảng sao cho thể hiện được ý nghĩa của tập hợp.
  - Từ khóa **new** là bắt buộc để cấp phát động trong C++.
  - **Size:** là kích thước mảng. Là một số nguyên dương không quá lớn(nên để dưới 1 triệu).
  - **e1, e2:** là các phần tử khởi tạo cho mảng. Số lượng các phần tử này phải <= size.

Cấp phát động mảng 1 chiều

- Cú pháp tổng quát: **type* arrayName = new type[size]();**
- Nếu muốn khởi tạo giá trị cho các phần tử:
  type* arrayName = new type[size]{e1, e2, …};
- Trong đó:
  - Trong cú pháp trên, nếu bỏ cặp ngoặc tròn sau cùng thì các phần tử mảng sẽ không được khởi tạo.
  - Nếu để cặp () hoặc số lượng phần tử khởi tạo nhỏ hơn số lượng phần tử tối đa, các phần tử còn lại của mảng sẽ nhận giá trị mặc định của kiểu mảng.
- Để thu hồi bộ nhớ sau khi cấp phát, ta sử dụng cú pháp:
  delete [] arrayName;

Minh họa

```
// cấp phát động không khởi tạo
int* myArray = new int[20];
// cấp phát động và khởi tạo giá trị mặc định của kiểu
int* arr = new int[15]();
// cấp phát động + khởi tạo giá trị cho 1 số phần tử
int* numbers = new int[10]{ 2, 3, 5 };

// sử dụng mảng động như bình thường
for (int i = 0; i < 10; i++) {
      cout << numbers[i] << " ";
}

// thu hồi bộ nhớ cấp phát động
delete[] myArray;
delete[] arr;
delete[] numbers;
```

```
2 3 5 0 0 0 0 0 0 0
```

Sử dụng unique_ptr

- Nếu ta không muốn chủ động thu hồi bộ nhớ với delete[] mà để việc đó cho chương trình tự xử lý, ta sử dụng unique_ptr.
- Đó là một lớp template trong thư viện <memory>. Một đối tượng của unique_ptr dùng để quản lý một con trỏ trỏ tới vùng nhớ cấp phát động.
- Khi chương trình kết thúc, đối tượng của unique_ptr sẽ tự động gọi delete[] để giải phóng vùng nhớ mà nó đang quản lý.
- Cú pháp: **unique_ptr<type[]> pointerName(new type[size]());**
  - **unique_ptr** là bắt buộc.
  - **pointerName** là tên của con trỏ, có thể đặt theo style của mảng hoặc con trỏ.
  - Dấu hiệu của mảng gồm type[] đặt trong cặp <> sau unique_ptr.
  - Sử dụng **pointerName** như con trỏ hoặc mảng thông thường.

Minh họa

```cpp
#include <iostream>
#include <memory>

using namespace std;

int main() {
    // cấp phát động không khởi tạo
    unique_ptr<int[]> myArray(new int[20]);
    // cấp phát động và khởi tạo giá trị mặc định của kiểu
    unique_ptr<int[]> arr(new int[15]());
    // cấp phát động + khởi tạo giá trị cho 1 số phần tử
    unique_ptr<int[]> numbers(new int[10]{ 2, 3, 5 });

    // sử dụng mảng động như bình thường
    for (int i = 0; i < 10; i++) {
        cout << numbers[i] << " ";
    }
    // đối tượng của unique_ptr sẽ tự delete mảng cho ta
    return 0;
}
```

Nội dung tiếp theo

Mảng hai chiều