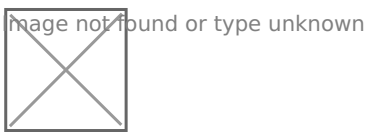


# Mảng trong C++

## Khái niệm

Mảng là một dãy các phần tử có cùng kiểu được đặt liên tiếp trong bộ nhớ và có thể truy xuất đến từng phần tử bằng cách thêm một chỉ số vào sau tên của mảng.

Điều này có nghĩa là, ví dụ, chúng ta có thể lưu 5 giá trị kiểu `int` mà không cần phải khai báo 5 biến khác nhau. Ví dụ, một mảng chứa 5 giá trị nguyên kiểu `int` có tên là *billy* có thể được biểu diễn như sau:



trong đó mỗi một ô trống biểu diễn một phần tử của mảng, trong trường hợp này là các giá trị nguyên kiểu `int`. Chúng được đánh số từ 0 đến 4 vì phần tử đầu tiên của mảng luôn là 0 bất kể độ dài của nó là bao nhiêu.

Như bất kì biến nào khác, một mảng phải được khai báo trước khi có thể sử dụng. Một khai báo điển hình cho một mảng trong C++ như sau:

```
type name [elements];
```

trong đó `type` là một kiểu dữ liệu hợp lệ (`int`, `float`...), `name` là một tên biến hợp lệ và trường `elements` chỉ định mảng đó sẽ chứa bao nhiêu phần tử

Vì vậy, để khai báo *billy* như đã trình bày ở trên chúng ta chỉ cần một dòng đơn giản như sau:

```
int billy [5];
```

**Chú ý:** Trường `elements` bên trong cặp ngoặc `[]` phải là một giá trị hằng khi khai báo một mảng, vì mảng là một khối nhớ tĩnh có kích cỡ xác định và trình biên dịch phải có khả năng xác định xem cần bao nhiêu bộ nhớ để cấp phát cho mảng trước khi các lệnh có thể được thực hiện.

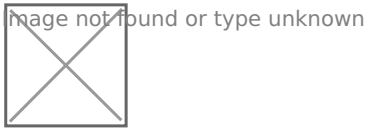
## Khởi tạo một mảng

Khi khai báo một mảng với tầm hoạt động địa phương (trong một hàm), theo mặc định nó sẽ không được khởi tạo, vì vậy nội dung của nó là không xác định cho đến khi chúng ta lưu các giá trị lên đó.

Nếu chúng ta khai báo một mảng toàn cục (bên ngoài tất cả các hàm) nó sẽ được khởi tạo và tất cả các phần tử được đặt bằng 0. Vì vậy nếu chúng ta khai báo mảng toàn cục:

```
int billy [5];
```

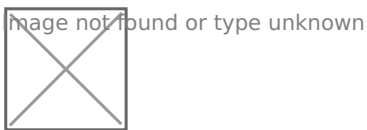
mọi phần tử của billy sẽ được khởi tạo là 0:



Nhưng thêm vào đó, khi chúng ta khai báo một mảng, chúng ta có thể gán các giá trị khởi tạo cho từng phần tử của nó. Ví dụ:

```
int billy [5] = { 16, 2, 77, 40, 12071 };
```

lệnh trên sẽ khai báo một mảng như sau:



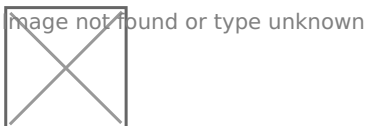
Số phần tử trong mảng mà chúng ta khởi tạo với cặp ngoặc nhọn { } phải bằng số phần tử của mảng đã được khai báo với cặp ngoặc vuông [ ]. Bởi vì điều này có thể được coi là một sự lặp lại không cần thiết nên C++ cho phép để trống giữa cặp ngoặc vuông, kích thước của mảng được xác định bằng số giá trị giữa cặp ngoặc nhọn.

## Truy xuất đến các phần tử của mảng

Ở bất kì điểm nào của chương trình trong tầm hoạt động của mảng, chúng ta có thể truy xuất từng phần tử của mảng để đọc hay chỉnh sửa như là đối với một biến bình thường. Cấu trúc của nó như sau:

```
name[ index]
```

Như ở trong ví dụ trước ta có mảng billy gồm 5 phần tử có kiểu int, chúng ta có thể truy xuất đến từng phần tử của mảng như sau:



Ví dụ, để lưu giá trị 75 vào phần tử thứ ba của billy ta viết như sau:

```
billy[ 2] = 75;
```

và, ví dụ, để gán giá trị của phần tử thứ 3 của billy cho biến a, chúng ta viết:

```
a = billy[2];
```

Vì vậy, xét về mọi phương diện, biểu thức `billy[2]` giống như bất kì một biến kiểu `int`.

Chú ý rằng phần tử thứ ba của `billy` là `billy[2]`, vì mảng bắt đầu từ chỉ số 0. Vì vậy, phần tử cuối cùng sẽ là `billy[4]`. Vì vậy nếu chúng ta viết `billy[5]`, chúng ta sẽ truy xuất đến phần tử thứ 6 của mảng và vượt quá giới hạn của mảng.

Trong C++, việc vượt quá giới hạn chỉ số của mảng là hoàn toàn hợp lệ, tuy nhiên nó có thể gây ra những vấn đề thực sự khó phát hiện bởi vì chúng không tạo ra những lỗi trong quá trình dịch nhưng chúng có thể tạo ra những kết quả không mong muốn trong quá trình thực hiện. Nguyên nhân của việc này sẽ được nói đến kĩ hơn khi chúng ta bắt đầu sử dụng con trỏ.

Cần phải nhấn mạnh rằng chúng ta sử dụng cặp ngoặc vuông cho hai tác vụ: đầu tiên là đặt kích thước cho mảng khi khai báo chúng và thứ hai, để chỉ định chỉ số cho một phần tử cụ thể của mảng khi xem xét đến nó.

```
int billy[5]; // khai báo một mảng mới.  
billy[2] = 75; // truy xuất đến một phần tử của mảng.
```

Một vài thao tác hợp lệ khác với mảng:

```
billy[0] = a;  
billy[a] = 75;  
b = billy[a+2];  
billy[billy[a]] = billy[2] + 5;
```

Ví dụ chương trình:

```
// ví dụ với mảng  
#include <iostream.h>  
  
int billy [] = {16, 2, 77, 40, 12071};  
int n, result=0;  
  
int main ()  
{  
    for ( n=0 ; n<5 ; n++ )  
    {  
        result += billy[n];  
    }  
    cout << result;
```

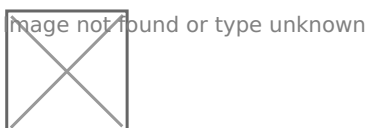
```
return 0;
}
```

Kết quả:

12206

## Mảng nhiều chiều

Mảng nhiều chiều có thể được coi như mảng của mảng, ví dụ, một mảng hai chiều có thể được tưởng tượng như là một bảng hai chiều gồm các phần tử có kiểu dữ liệu cụ thể và giống nhau.

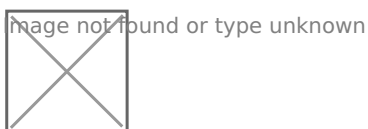


`jimmy` biểu diễn một mảng hai chiều kích thước 3x5 có kiểu `int`. Cách khai báo mảng này như sau:

```
int jimmy [3][5];
```

và, ví dụ, cách để truy xuất đến phần tử thứ hai theo chiều dọc và thứ tư theo chiều ngang trong một biểu thức như sau:

```
jimmy[1][3]
```



Mảng nhiều chiều không bị giới hạn bởi hai chỉ số (hai chiều), Chúng có thể chứa bao nhiêu chỉ số tùy thích mặc dù ít khi cần phải dùng đến mảng lớn hơn 3 chiều. Hãy thử xem xét lượng bộ nhớ mà một mảng có nhiều chỉ số cần đến. Ví dụ:

```
char century [100][365][24][60][60];
```

gán một giá trị char cho mỗi giây trong một thế kỉ, phải cần đến hơn 3 tỷ giá trị chars! Chúng ta sẽ phải cần khoảng 3GB RAM để khai báo nó.

Mảng nhiều chiều thực ra là một khái niệm trừu tượng vì chúng ta có thể có kết quả tương tự với mảng một chiều bằng một thao tác đơn giản giữa các chỉ số của nó:

```
int jimmy [3][5]; // tương đương với
int jimmy [15]; (3 * 5 = 15)
```

Dưới đây là hai ví dụ với cùng một kết quả như nhau, một sử dụng mảng hai chiều và một sử dụng mảng một chiều:

- *Mảng hai chiều:*

```
// multidimensional array
#include <iostream.h>

#define WIDTH 5
#define HEIGHT 3

int jimmy [ HEIGHT ][ WIDTH ];
int n,m;

int main ()
{
    for ( n=0; n<HEIGHT; n++)
        for ( m=0; m<WIDTH; m++)
        {
            jimmy[ n ][ m ]=( n+1 )*( m+1 );
        }
    return 0;
}
```

- *Mảng một chiều:*

```
// pseudo-multidimensional array
#include <iostream.h>

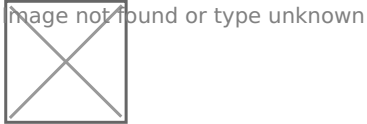
#define WIDTH 5
#define HEIGHT 3

int jimmy [ HEIGHT * WIDTH ];
int n,m;

int main ()
{
    for ( n=0; n<HEIGHT; n++)
        for ( m=0; m<WIDTH; m++)
        {
            jimmy[ n * WIDTH + m ]=( n+1 )*( m+1 );
        }
}
```

```
}  
    return 0;  
}
```

Không một chương trình nào viết gì ra màn hình nhưng cả hai đều gán giá trị vào khối nhớ có tên **jimmy** theo cách sau:



Chúng ta đã định nghĩa hằng (`#define`) để đơn giản hóa những chỉnh sửa sau này của chương trình, ví dụ, trong trường hợp chúng ta quyết định tăng kích thước của mảng với chiều cao là 4 thay vì là 3, chúng ta chỉ cần thay đổi dòng:

```
#define HEIGHT 3
```

thành:

```
#define HEIGHT 4
```

và không phải có thêm sự thay đổi nào nữa đối với chương trình.

## Dùng mảng làm tham số

Vào một lúc nào đó có thể chúng ta cần phải truyền một mảng tới một hàm như là một tham số. Trong C++, việc truyền theo tham số giá trị một khối nhớ là không hợp lệ, ngay cả khi nó được tổ chức thành một mảng. Tuy nhiên chúng ta lại được phép truyền địa chỉ của nó, việc này cũng tạo ra kết quả thực tế giống thao tác ở trên nhưng lại nhanh hơn nhiều và hiệu quả hơn.

Để có thể nhận mảng là tham số thì điều duy nhất chúng ta phải làm khi khai báo hàm là chỉ định trong phần tham số kiểu dữ liệu cơ bản của mảng, tên mảng và cặp ngoặc vuông trống. Ví dụ, hàm sau:

```
void procedure (int arg[])
```

nhận vào một tham số có kiểu "mảng của char" và có tên `arg`. Để truyền tham số cho hàm này một mảng được khai báo:

```
int myarray [ 40];
```

chỉ cần gọi hàm như sau:

```
procedure ( myarray );
```

Dưới đây là một ví dụ cụ thể:

```
// arrays as parameters
#include <iostream.h>

void printarray (int arg[], int length) {
    for (int n=0; n<length; n++)
        cout << arg[n] << " ";
    cout << "\n";
}

int main ()
{
    int firstarray[] = {5, 10, 15};
    int secondarray[] = {2, 4, 6, 8, 10};
    printarray (firstarray,3);
    printarray (secondarray,5);
    return 0;
}
```

Kết quả:

```
5 10 15
2 4 6 8 10
```

Như bạn có thể thấy, tham số đầu tiên (**int arg[]**) chấp nhận mọi mảng có kiểu cơ bản là **int**, bất kể độ dài của nó là bao nhiêu, vì vậy cần thiết phải có tham số thứ hai để báo cho hàm này biết độ dài của mảng mà chúng ta truyền cho nó.

Trong phần khai báo hàm chúng ta cũng có thể dùng tham số là các mảng nhiều chiều. Cấu trúc của mảng 3 chiều như sau:

```
base_type[][depth][depth]
```

ví dụ, một hàm với tham số là mảng nhiều chiều có thể như sau:

```
void procedure (int myarray[][3][4])
```

Chú ý rằng cặp ngoặc vuông đầu tiên để trống nhưng các cặp ngoặc sau thì không. Bạn luôn luôn phải làm vậy vì trình biên dịch C++ phải có khả năng xác định độ lớn của các chiều thêm vào của mảng.

