Arrays

What is an array?

- Compound data type or data structure
- Collection of elements
- All elements are of the same type
- Each element can be accessed directly

Array的特點

- 固定大小
- 所有元素都是同一個type
- 連續地儲存在memory裡
- 個別的元素可以用其位置或索引存取
- 第一個元素在位置**0**
- 最後一個元素在 **index size-1**
- 不會檢查是否超出界線

宣告Array

```
Element_Type array_name [constant number of elements] {init list};
```

例如:

```
int test_scores [5] {100,95,99,97,98};
int high_score [10] {3,5} //前兩個元素分別是3,5，剩下的都是0
int another_array [] {1,2,3,4,5} //大小會自動計算
```

Array是怎麼運作的?

- Array的名字代表的是第一個元素(index 0)的位置
- [index]代表的是對於array開頭的偏移量
- C++就是單純做計算來找到正確的位置

Multi-dimensional arrays

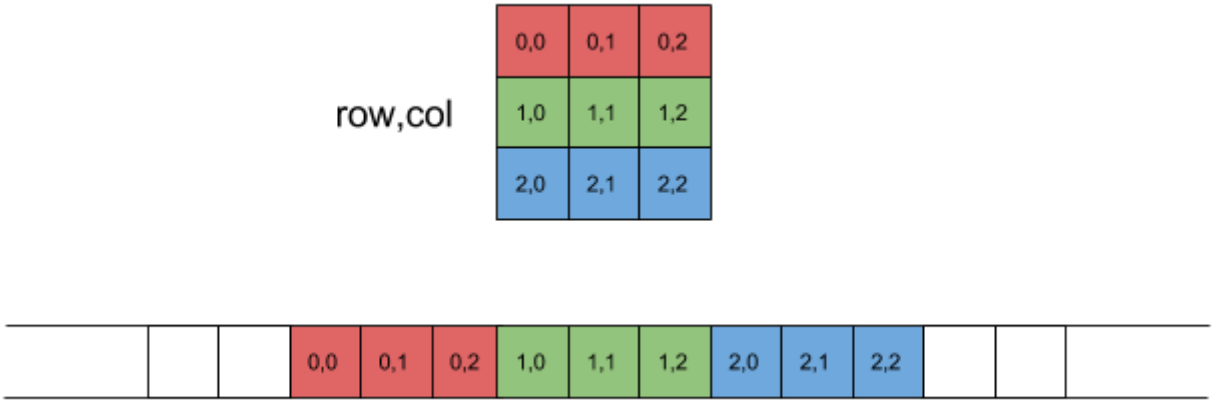
宣告:

```
Element_Type array_name [dim1_size] [dim2_size]
```

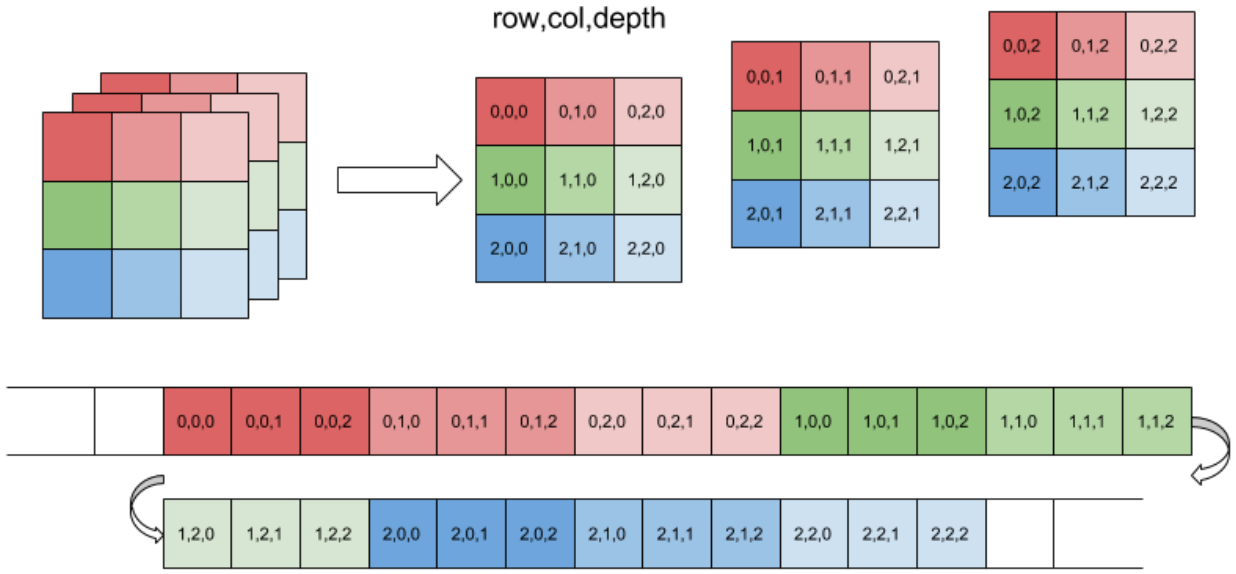
範例:

```
int movie_rating [3][4];
```

- 2D row-major



- 2D column-major



Vectors

`vector` 是 C++ 標準函式庫 STL（Standard Template Library）中提供的一種**序列容器**，可以視為一種**動態陣列**（Dynamic Array），其大小可自動調整。
宣告:

```
vector<type> name;
```

範例:

```
vector<int> v(5); // 建立一個有 5 個元素的 vector，預設值為 0
vector<int> v(5, 42); // 建立 5 個元素，每個初始值都是 42

vector<int> v1 = {1, 2, 3};
vector<int> v2(v1); // 複製 v1 的內容給 v2

vector<int> v1 = {10, 20, 30, 40};
vector<int> v2(v1.begin() + 1, v1.end() - 1); // 複製範圍內元素
```

🟢 `vector` 的優勢

1. 📏 動態大小調整

不像傳統陣列需要指定固定大小，`vector` 會自動根據需求調整容量。

```
std::vector<int> v;
v.push_back(1);
```

```
v.push_back(2); // 自動擴展空間，不需手動處理記憶體
```

2. 🧠 自動記憶體管理

- 底層會自動配置、釋放記憶體，避免記憶體洩漏。
- 無需手動使用 `new` 或 `delete`。

3. 📚 支援 STL 演算法

`vector` 能直接配合 `std::sort`, `std::find`, `std::for_each` 等 STL 函式使用。

```
std::sort(v.begin(), v.end());
```

4. 🚀 高效的尾端插入操作（push_back）

尾端插入時間複雜度平均為 **O(1)**（攤銷分析下）。

5. 🔄 隨機存取支援（Random Access）

支援像陣列一樣用 `v[i]` 隨機存取元素，時間複雜度為 **O(1)**。

6. 🧪 豐富的成員函式

如 `size()`, `capacity()`, `empty()`, `clear()`, `insert()`, `erase()` ，操作彈性大。

7. 🧱 容易與 C-style 陣列整合

```
std::vector<int> v = {1, 2, 3};
int* ptr = v.data(); // 取得底層指標
```

存取Vector elements

```
vector_name [element_index]
```

```
vector_name.at(element_index)
```

Vector的特點->可以增長

```
vector_name.push_back(element)//在vector結尾增加element
```

Vector的宣告

🔷 1. 基本宣告方式

```
#include <vector>  // 引入標頭檔
std::vector<int> v;  // 宣告一個空的 vector，元素型態為 int
```

🔷 2. 指定初始大小

```
std::vector<int> v(5);  // 建立一個有 5 個元素的 vector，預設值為 0
```

📌 效果：`v = {0, 0, 0, 0, 0}`

🔷 3. 指定大小與初始值

```
std::vector<int> v(5, 42);  // 建立 5 個元素，每個初始值都是 42
```

📌 效果：`v = {42, 42, 42, 42, 42}`

🔷 4. 使用初始化列表（C++11 起）

```
std::vector<int> v = {1, 2, 3, 4, 5};
```

📌 效果：直接初始化內容，比較直覺。

🔷 5. 複製其他 vector

```
std::vector<int> v1 = {1, 2, 3}; std::vector<int> v2(v1);  // 複製 v1 的內容給 v2
```

🔷 6. 用迭代器範圍初始化

```
std::vector<int> v1 = {10, 20, 30, 40}; std::vector<int> v2(v1.begin() + 1, v1.end() - 1);  // 複製範圍內元素
```

📌 效果：`v2 = {20, 30}`

🔷 7. 宣告二維 vector（常用在矩陣）

```
std::vector<std::vector<int>> matrix(3, std::vector<int>(4, 0));
```

📌 效果：建立一個 3x4 的二維矩陣，全部初始化為 0。

ASCII 示意圖：
[
[0, 0, 0, 0],
[0, 0, 0, 0],
[0, 0, 0, 0]
]

push.back()本身是傳值

所以2-dim vector

```
vector <int> vector1;
vector <vector <int>> vector_2d push_back(vector1);//是複製一份vector1
```