09-STL 容器 - vector

9-1 vector

1. 為什麼需要 vector？

1.1 傳統陣列的限制

在 C++ 中，我們熟悉的傳統陣列有一個根本的問題：**大小必須在編譯時決定，且無法改變。**

```
int scores[100];   // 固定 100 格，多了浪費，少了不夠用
```

傳統陣列的痛點：

- 宣告時必須給定大小（或使用 `new`，但要自己管理記憶體）
- 不知道目前有幾個元素（需要額外的變數追蹤）
- 插入、刪除元素需要手動搬移資料
- 忘記 `delete[]` 就記憶體洩漏(memory leak)

```
// 傳統做法：又長又容易出錯
int* scores = new int[100];  // 動態跟作業系統要一塊記憶體
int count = 0;

scores[count++] = 90;  // 注意這裡的 count++，count 值先被評估，再遞增 1
scores[count++] = 85;

cout << "We have " << count << " items in scores[]" << endl;

for(int i=0; i<count; i++)
{
    cout << "scores[" << i << "] = " << scores[i] << endl;
}

// 用完還要記得把記憶體還作業系統
delete[] scores;
```

1.2 vector 是什麼？

`vector` 是 C++ **標準模板庫（STL）**中的一種**動態陣列容器**。

它的核心優勢：

- 大小可以**自動擴張**，不需要手動管理記憶體
- 提供豐富的**成員函數**，操作方便
- 支援與陣列相同的**下標存取** `[]`
- 離開作用域後**自動釋放記憶體**

1.3 圖解：記憶體動態擴張

```
初始狀態（capacity = 1）：
┌───┐
│ 3 │
└───┘
  size=1, capacity=1

push_back(7)，空間不足 → 自動擴張為 capacity=2：
┌───┬───┐
│ 3 │ 7 │
└───┴───┘
  size=2, capacity=2

push_back(5)，空間不足 → 自動擴張為 capacity=4：
┌───┬───┬───┬───┐
│ 3 │ 7 │ 5 │   │
└───┴───┴───┴───┘
```

```
  size=3, capacity=4（預留空間）

push_back(1)，空間足夠 → 直接寫入：
┌───┬───┬───┬───┐
│ 3 │ 7 │ 5 │ 1 │
└───┴───┴───┴───┘
  size=4, capacity=4
```

重點：size 是目前元素數量，capacity 是已分配的記憶體空間。vector 通常以倍增方式擴張，以攤平擴張的成本。

9-2 vector 的基礎操作

vector 基礎操作

1. 引入標頭檔與宣告

使用 `vector` 前，需要引入標頭檔：

```
#include <iostream>
#include <vector>

using namespace std;
```

2. 宣告與初始化

```
// 方式一：宣告空的 vector
// capacity:0, size: 0
vector<int> v1;

// 方式二：指定初始大小（元素值為 0）
// capacity:5, size: 5
vector<int> v2(5);          // [0, 0, 0, 0, 0]

// 方式三：指定大小與預設值
// capacity:5, size: 5
vector<int> v3(5, 100);     // [100, 100, 100, 100, 100]

// 方式四：用初始化列表
// capacity:5, size: 5
vector<int> v4 = {3, 1, 4, 1, 5};  // [3, 1, 4, 1, 5]

// 方式五：複製另一個 vector
// capacity:5, size: 5
vector<int> v5(v4);         // [3, 1, 4, 1, 5]

// 也可以存其他型別
vector<double> scores;
vector<string> names;
vector<bool>   flags;
```

3. 查詢大小與清空

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

cout << v.size();    // 5：目前元素個數
cout << v.empty();   // 0 (false) ：是否為空

v.clear();           // 清空所有元素
cout << v.size();    // 0
cout << v.empty();   // 1 (true)
```

4. 新增與移除元素

在末尾新增元素

vector 的成員函數 `push_back(val)`，可以將一個值新增到目前 vector 的尾端。

```
vector<int> v;
v.push_back(10);   // [10]
v.push_back(20);   // [10, 20]
v.push_back(30);   // [10, 20, 30]
```

移除末尾元素

vector 的成員函數 `pop_back()`，可以將 vector 尾端的那個元素移除。

```
v.pop_back();      // [10, 20]
```

⚠ **注意**：對空的 vector 呼叫 pop_back() 是未定義行為（UB），請先確認 !v.empty()。

5. 存取元素

下標運算子 `[]`

```
vector<int> v = {10, 20, 30};
cout << v[0];   // 輸出 10
v[1] = 99;      // 修改第二個元素
```

帶邊界檢查的存取 `at(i)`

```
cout << v.at(0);   // 輸出 10
cout << v.at(5);   // 超出範圍 → 丟出 std::out_of_range 例外
```

方式	速度	邊界檢查	建議使用時機
`v[i]`	較快	□ 沒有	確定索引合法時
`v.at(i)`	稍慢	□ 有	需要安全保障時

6. 範例：動態收集成績

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> scores;
    int n, score;

    cout << "請輸入學生人數：";
    cin >> n;

    for (int i = 0; i < n; i++)
    {
        cout << "輸入第 " << i + 1 << " 位學生成績：";
        cin >> score;
        scores.push_back(score);   // 動態新增
    }

    // 計算總分
    int total = 0;
    for (int i = 0; i < scores.size(); i++) {
        total += scores[i];
    }

    cout << "平均分數：" << (double)total / scores.size() << endl;
    return 0;
}
```

9-3 走訪 vector

走訪 vector

1. 方法一：下標 for 迴圈

最直覺的方式，適合需要使用索引時：

```
vector<int> v = {10, 20, 30, 40, 50};

for (int i = 0; i < v.size(); i++) {
    cout << v[i] << " ";
}
// 輸出：10 20 30 40 50
```

> ⚠ v.size() 回傳型別為 size_t（無號整數），若用 int i = v.size() - 1 倒著走時要小心，空 vector 的 0 - 1 會變成一個很大的數。

2. 方法二：Range-based for（範圍式 for）

C++11 起支援，最簡潔：

```
// 唯讀走訪
for (int x : v) {
    cout << x << " ";
}

// 修改元素（加上 &）
for (int& x : v) {
    x *= 2;   // 每個元素乘以 2
}
```

3. 方法三：迭代器（Iterator）

迭代器的概念類似**指標**，這也是為什麼熟悉指標的你能很快上手。

圖解：迭代器與指標的對比

```
傳統陣列指標：
  int arr[] = {10, 20, 30};
  int* ptr = arr;       // ptr 指向第一個元素

  ┌────┬────┬────┐
  │ 10 │ 20 │ 30 │
  └────┴────┴────┘
    ↑
   ptr

  ptr++;                // 指向下一個
  *(ptr) = 99;          // 解參考後修改

vector 迭代器：
  vector<int> v = {10, 20, 30};
  auto it = v.begin();  // it 指向第一個元素

  ┌────┬────┬────┬───────┐
  │ 10 │ 20 │ 30 │(end)  │
  └────┴────┴────┴───────┘
    ↑                ↑
  begin()          end()（指向最後一個之後）

  it++;             // 指向下一個（同 ptr++）
```

```
    *it = 99;          // 解參考後修改（同 *ptr）
```

迭代器的基本用法

```
vector<int> v = {10, 20, 30, 40, 50};

// 傳統的寫法，宣告 iterator 的型別有點複雜
// vector<int>::iterator it = v.begin();

// 自 C++ 11 起可以用 auto 自動推導型別
auto it = v.begin();    // 指向第一個元素

while (it != v.end()) {
    cout << *it << " ";  // 解參考取值
    it++;                // 移動到下一個
}
// 輸出：10 20 30 40 50
```

更常見的寫法（for 迴圈）：

```
for (auto it = v.begin(); it != v.end(); it++) {
    cout << *it << " ";
}
```

三種走訪方式比較

方式	優點	缺點
下標 `v[i]`	直覺，可使用索引	需要注意型別
range-based for	最簡潔	無法直接取得索引
迭代器	與 STL 演算法相容	語法略繁瑣

✅ 實務建議：平常用 range-based for，需要插入/刪除時用迭代器，需要索引時用下標。

9-4 常用成員函數

其他常用成員函數

1. front() 與 back()

```
vector<int> v = {10, 20, 30, 40, 50};

cout << v.front();  // 10：第一個元素
cout << v.back();   // 50：最後一個元素

v.front() = 99;     // 可以修改
v.back()  = 1;
// v 現在是 {99, 20, 30, 40, 1}
```

2. insert()：在指定位置插入

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

// insert(位置迭代器, 值)
auto it = v.begin() + 2;   // 指向索引 2（值為 3）
v.insert(it, 99);

// v 現在是 {1, 2, 99, 3, 4, 5}
```

圖解：

```
插入前：
┌───┬───┬───┬───┬───┐
│ 1 │ 2 │ 3 │ 4 │ 5 │
└───┴───┴───┴───┴───┘
          ↑
        it（索引 2）

插入後（後面的元素全部後移）：
┌───┬───┬────┬───┬───┬───┐
│ 1 │ 2 │ 99 │ 3 │ 4 │ 5 │
└───┴───┴────┴───┴───┴───┘
```

⚠ insert() 後，原本的迭代器可能**失效**，請重新取得。

3. erase()：移除指定位置的元素

```
vector<int> v = {1, 2, 99, 3, 4, 5};

// 移除單一元素
v.erase(v.begin() + 2);   // 移除索引 2（值 99）
// v 現在是 {1, 2, 3, 4, 5}

// 移除一個範圍 [begin+1, begin+3)（左閉右開）
v.erase(v.begin() + 1, v.begin() + 3);
// 移除索引 1 和 2（值 2 和 3）
// v 現在是 {1, 4, 5}
```

4. resize() 與 reserve()

這兩個函數常被搞混，用下圖區分：

resize(n)：改變 size（元素個數）

```
vector<int> v = {1, 2, 3};  size=3, capacity=3
```

```
v.resize(5);
→ {1, 2, 3, 0, 0}           size=5, capacity=5（新元素填 0）

v.resize(2);
→ {1, 2}                    size=2, capacity=5（縮小 size，不影響 capacity）
```

reserve(n)：改變 capacity（預留記憶體）

```
vector<int> v = {1, 2, 3};  size=3, capacity=3

v.reserve(10);
→ {1, 2, 3}                 size=3, capacity=10（只預留空間，不新增元素）
```

何時用 reserve？

當你事先知道大概會新增多少元素時，使用 `reserve` 可以避免多次重新分配記憶體，提升效能：

```
vector<int> v;
v.reserve(1000);          // 預先分配 1000 個空間

for (int i = 0; i < 1000; i++) {
    v.push_back(i);       // 不會觸發記憶體重新分配
}
```

5. capacity()：查詢已分配空間

```
vector<int> v;
cout << v.size()     << endl;   // 0
cout << v.capacity() << endl;   // 0（或平台相關的初始值）

v.push_back(1);
cout << v.capacity() << endl;   // 1（或更大，依平台）

v.push_back(2);
cout << v.capacity() << endl;   // 2（擴張）

v.push_back(3);
cout << v.capacity() << endl;   // 4（擴張為倍數）
```

6. 二維 vector

`vector` 可以嵌套，用來表示矩陣或表格：

```
// 宣告 3x4 的二維 vector，初始值全為 0
vector<vector<int>> matrix(3, vector<int>(4, 0));
```

圖解：

```
matrix：
        [0]  [1]  [2]  [3]
[0] ──► ┌────┬────┬────┬────┐
        │ 0  │ 0  │ 0  │ 0  │
        └────┴────┴────┴────┘
[1] ──► ┌────┬────┬────┬────┐
        │ 0  │ 0  │ 0  │ 0  │
        └────┴────┴────┴────┘
[2] ──► ┌────┬────┬────┬────┐
        │ 0  │ 0  │ 0  │ 0  │
        └────┴────┴────┴────┘
```

```
// 存取與修改
matrix[1][2] = 99;

// 走訪二維 vector
```

```
for (int i = 0; i < matrix.size(); i++) {
    for (int j = 0; j < matrix[i].size(); j++) {
        cout << matrix[i][j] << " ";
    }
    cout << endl;
}

// 使用 range-based for
for (auto& row : matrix) {
    for (int val : row) {
        cout << val << " ";
    }
    cout << endl;
}
```

注意：二維 vector 的每一列長度可以不同（鋸齒狀陣列），這與傳統二維陣列不同。

9-5 實作練習

練習一：成績管理系統（基礎操作）

題目敘述

請你撰寫一個程式，功能如下：

1. 讀入 N 筆學生成績（整數，0~100）
2. 輸出所有成績
3. 輸出最高分、最低分、平均分數（取到小數點後兩位）
4. 刪除最後一筆成績後，再輸出一次所有成績

範例輸入

```
5
90 75 88 62 95
```

範例輸出

```
成績：90 75 88 62 95
最高分：95
最低分：62
平均：82.00
刪除最後一筆後：90 75 88 62
```

提示

- 使用 `push_back` 加入一筆成績
- 善用 `size()`、`pop_back()`

▼ **參考解答（請先自己嘗試！）**

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

int main() {
    int n;
    cin >> n;

    vector<int> scores;
    for (int i = 0; i < n; i++) {
        int s;
        cin >> s;
        scores.push_back(s);
    }

    // 輸出所有成績
    cout << "成績：";
    for (int s : scores) cout << s << " ";
    cout << endl;

    // 找最大最小
    int maxVal = scores[0], minVal = scores[0];
    int total = 0;
    for (int s : scores) {
        if (s > maxVal) maxVal = s;
        if (s < minVal) minVal = s;
        total += s;
    }

    cout << "最高分：" << maxVal << endl;
```

```
    cout << "最低分：" << minVal << endl;
    cout << fixed << setprecision(2);
    cout << "平均：" << (double)total / scores.size() << endl;

    scores.pop_back();

    cout << "刪除最後一筆後：";
    for (int s : scores) cout << s << " ";
    cout << endl;

    return 0;
}
```

練習二：移除特定元素（insert / erase）

題目敘述

給定一個整數序列，請你：

1. 讀入 N 個整數
2. 讀入一個目標值 K
3. 移除序列中**所有等於 K 的元素**
4. 在序列**開頭**插入一個數字 0
5. 輸出最終序列

範例輸入

```
8
1 3 5 3 2 3 7 9
3
```

範例輸出

```
0 1 5 2 7 9
```

提示

- 移除元素後，迭代器的位置會改變，請注意 `erase()` 的回傳值（它會回傳被刪除元素的下一個位置）
- 在開頭插入用 `v.insert(v.begin(), 0)`

常見陷阱

```
// ❌ 錯誤寫法：erase 後 it 已失效，不可 it++
for (auto it = v.begin(); it != v.end(); it++) {
    if (*it == K) v.erase(it);
}

// ✅ 正確寫法：erase 回傳下一個有效迭代器
for (auto it = v.begin(); it != v.end(); ) {
    if (*it == K)
        it = v.erase(it);  // 不 it++
    else
        it++;
}
```

▼ 參考解答（請先自己嘗試！）

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

int main() {
    int n;
```

```
    cin >> n;

    vector<int> v;
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        v.push_back(x);
    }

    int K;
    cin >> K;

    // 移除所有等於 K 的元素
    for (auto it = v.begin(); it != v.end(); ) {
        if (*it == K)
            it = v.erase(it);
        else
            it++;
    }

    // 在開頭插入 0
    v.insert(v.begin(), 0);

    // 輸出
    for (int x : v) cout << x << " ";
    cout << endl;

    return 0;
}
```

練習三：矩陣加法（二維 vector）

題目敘述

給定兩個 N×M 的矩陣 A 和 B，請計算 C = A + B，並輸出結果矩陣 C。

範例輸入

```
2 3
1 2 3
4 5 6
7 8 9
1 0 1
```

（第一行為 N M，接下來 N 行為矩陣 A，再 N 行為矩陣 B）

範例輸出

```
8 10 12
5 5 7
```

提示

- 使用 `vector<vector<int>>` 來儲存矩陣
- 初始化：`vector<vector<int>> C(n, vector<int>(m, 0));`

▼ **參考解答（請先自己嘗試！）**

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

int main() {
    int n, m;
```

```
    cin >> n >> m;

    vector<vector<int>> A(n, vector<int>(m));
    vector<vector<int>> B(n, vector<int>(m));
    vector<vector<int>> C(n, vector<int>(m, 0));

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> A[i][j];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> B[i][j];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            C[i][j] = A[i][j] + B[i][j];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout << C[i][j];
            if (j < m - 1) cout << " ";
        }
        cout << endl;
    }

    return 0;
}
```

重點總覽

常用成員函數速查表

函數	功能	時間複雜度
`push_back(val)`	在末尾新增元素	O(1) 均攤
`pop_back()`	移除末尾元素	O(1)
`v[i]`	存取索引 i 的元素（無檢查）	O(1)
`v.at(i)`	存取索引 i 的元素（有檢查）	O(1)
`size()`	回傳元素個數	O(1)
`empty()`	是否為空	O(1)
`clear()`	清空所有元素	O(n)
`front()`	第一個元素	O(1)
`back()`	最後一個元素	O(1)
`insert(it, val)`	在迭代器位置前插入	O(n)
`erase(it)`	移除迭代器位置的元素	O(n)
`resize(n)`	調整元素個數	O(n)
`reserve(n)`	預留記憶體空間	O(n)
`capacity()`	已分配的空間大小	O(1)
`begin()`	指向第一個元素的迭代器	O(1)
`end()`	指向最後一個之後的迭代器	O(1)

三大常見陷阱

1. **`erase` 後迭代器失效**：請使用 `it = v.erase(it)` 取得新的有效迭代器。
2. **`pop_back` 空 `vector`**：呼叫前先確認 `!v.empty()`。
3. **`size()` 型別為 `size_t`**：比較或相減時注意無號整數的行為。