

C/C++ 進階班 資料結構

向量 (Vector)

李耕銘

向量 (Vector) 簡介

- 本章目標

1. 了解 `vector`

2. 實作 `vector`

3. 使用 `vector`

課程大綱

- 向量 (Vector) 簡介
- 實作向量
 1. 建構式與動態記憶體配置
 2. 新增與刪除資料
 3. reserve 與 resize
 4. 迭代器(iterator)
- Vector @ C++ STL

向量 (Vector) 簡介

向量 (Vector) 簡介

- 陣列能夠滿足我們的要求嗎？
- 陣列的限制：
 - 知道大小且固定大小
 1. `int a[5];`
 - 不知道大小或大小不固定
 1. `int n;`
 2. `cin >> n;`
 3. `int *p1 = new int[n];`
 4. `int *p2 = (int*) malloc(sizeof(int)*n);`

陣列有固定的大小，無法輕易刪除資料

向量 (Vector) 簡介

想要一個可以隨時改變長度的陣列

➤ 向量 (vector) !

向量：

➤ 本身是個類別模板

➤ 可以像陣列一般操作

➤ 可以根據需要自動擴展大小

向量 (vector) ~ 可以自動變大的陣列

建構式與動態記憶體配置

向量 (Vector) 簡介

因為長度不固定，需要使用**指標**操作記憶體空間

向量下的成員：

- 資料成員
 1. 指標，透過類別模板決定資料型態
 2. 長度
- 函式成員
 1. 建構式，以 malloc 建立陣列
 2. 解構式，釋放指標指到的空間
 3. 新增資料(push_back)
 4. 運算子重載[]

向量 (Vector) 簡介

加大指標所指的空間並不容易！

1. 以 malloc 或 calloc 挖出新的空間
2. 把舊空間的資料移到新空間中
3. 釋放舊空間的資料
4. 把舊指標指到新空間
5. 長度+1



向量 (Vector) 簡介

加大指標所指的空間並不容易！

1. 以 **malloc** 或 **calloc** 挖出新的空間
2. 把舊空間的資料移到新空間中
3. 釋放舊空間的資料
4. 把舊指標指到新空間
5. 長度+1

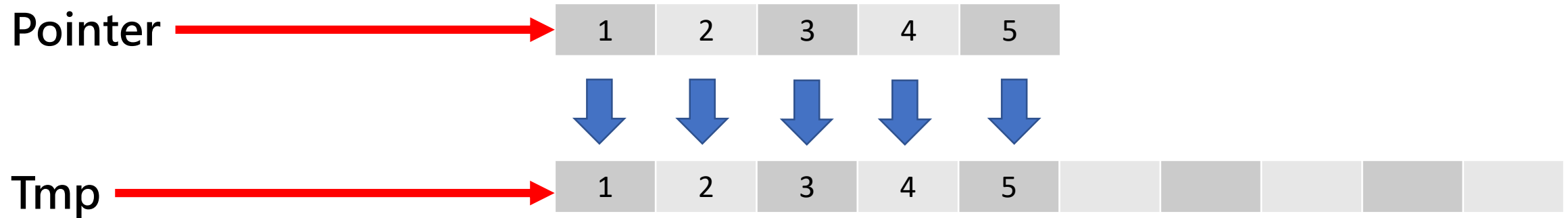
Pointer → 

Tmp → 

向量 (Vector) 簡介

加大指標所指的空間並不容易！

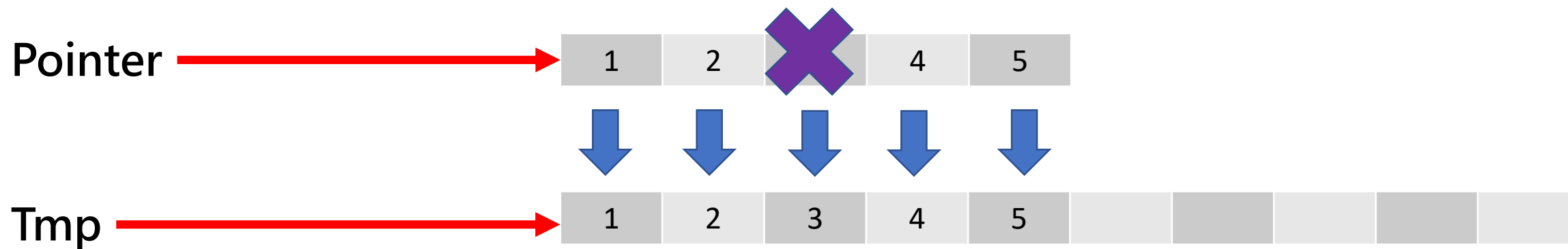
1. 以 malloc 或 calloc 挖出新的空間
2. 把舊空間的資料移到新空間中
3. 釋放舊空間的資料
4. 把舊指標指到新空間
5. 長度+1



向量 (Vector) 簡介

加大指標所指的空間並不容易！

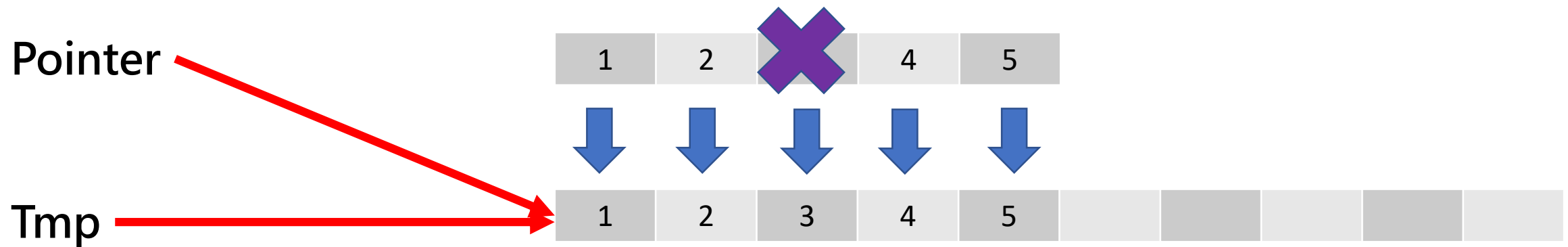
1. 以 malloc 或 calloc 挖出新的空間
2. 把舊空間的資料移到新空間中
3. 釋放舊空間的資料
4. 把舊指標指到新空間
5. 長度+1



向量 (Vector) 簡介

加大指標所指的空間並不容易！

1. 以 malloc 或 calloc 挖出新的空間
2. 把舊空間的資料移到新空間中
3. 釋放舊空間的資料
4. 把舊指標指到新空間
5. 長度+1



向量 (Vector) 簡介

malloc

配置空間大小後不歸零

```
Pointer = (資料型態 *) malloc(sizeof(資料型態) * 個數);
```

calloc

配置空間大小後歸零

```
Pointer = (資料型態 *) calloc(個數, sizeof(資料型態));
```

realloc

重新配置空間大小

```
Pointer2 = (資料型態 *) realloc(Pointer1, sizeof(資料型態) * 個數);
```

可能經過

1. 挖空間
2. 移動資料
3. 刪除舊空間

向量 (Vector) 簡介

```
#include <iostream>
#include <iomanip>
#include <stdlib.h>

using namespace std;

int main() {
    int *p = (int*) malloc(1);
    int *last = p;
    for(int i=2;i<50000;i++){
        p = (int*) realloc(p,i);
        if(last!=p)
            cout << setw(5) << i << "-th address:" << p << endl;
        last = p;
    }
    return 0;
}
```

換空間後原有空間不能夠使用！

```
2-th address:0x700d78
3-th address:0x700ca8
4-th address:0x700da8
5-th address:0x700d68
6-th address:0x700da8
7-th address:0x700e48
8-th address:0x700cb8
9-th address:0x707a28
57-th address:0x7012e0
217-th address:0x7014a0
761-th address:0x70abd0
5137-th address:0x70bfe8
12281-th address:0xbb0048
20377-th address:0xbb4fe8
20473-th address:0xbb9fe8
24569-th address:0xbb0048
```

Example Code

Mission

建立向量(Vector)的類別模板，並且包括：

1. 指標
2. 長度
3. 建構式
4. 解構式
5. 回傳第一個元素 (Front)
6. 回傳最後一個元素 (Back)

※皆採用大蛇式命名法

➤ How_Do_You_Turn_This_On

Practice

Mission

續寫向量(Vector)的類別模板，並且包括：

1. 取出特定索引值的資料 (At)
2. 運算子重載[]
3. 回傳目前陣列長度 (Size)
4. 判斷是否為空 (Empty)

新增與刪除資料

新增與刪除資料

1. 尾端新增

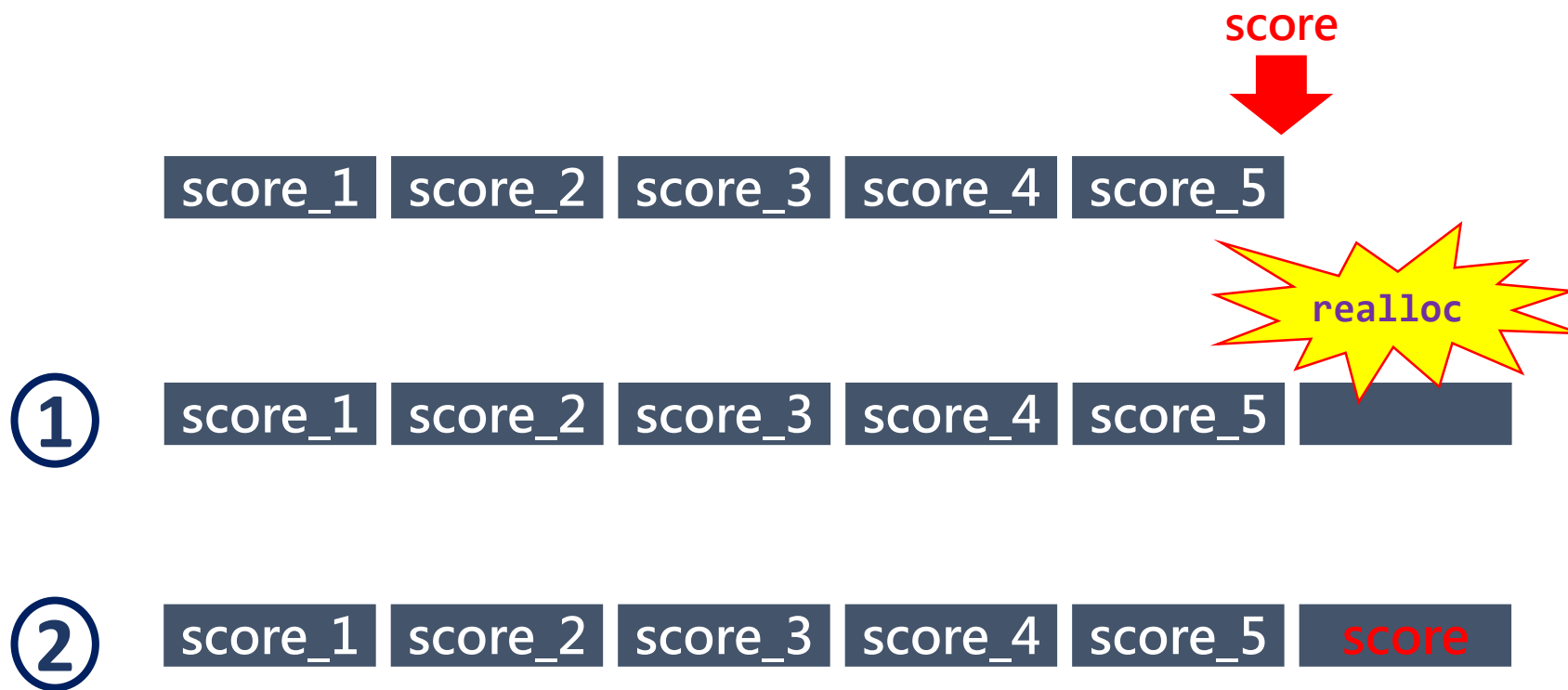
2. 尾端刪除

3. 插入資料

4. 刪除特定索引值

5. 刪除區間中的資料

6. 清空所有資料



新增與刪除資料

1. 尾端新增
2. 尾端刪除
3. 插入資料
4. 刪除特定索引值
5. 刪除區間中的資料
6. 清空所有資料

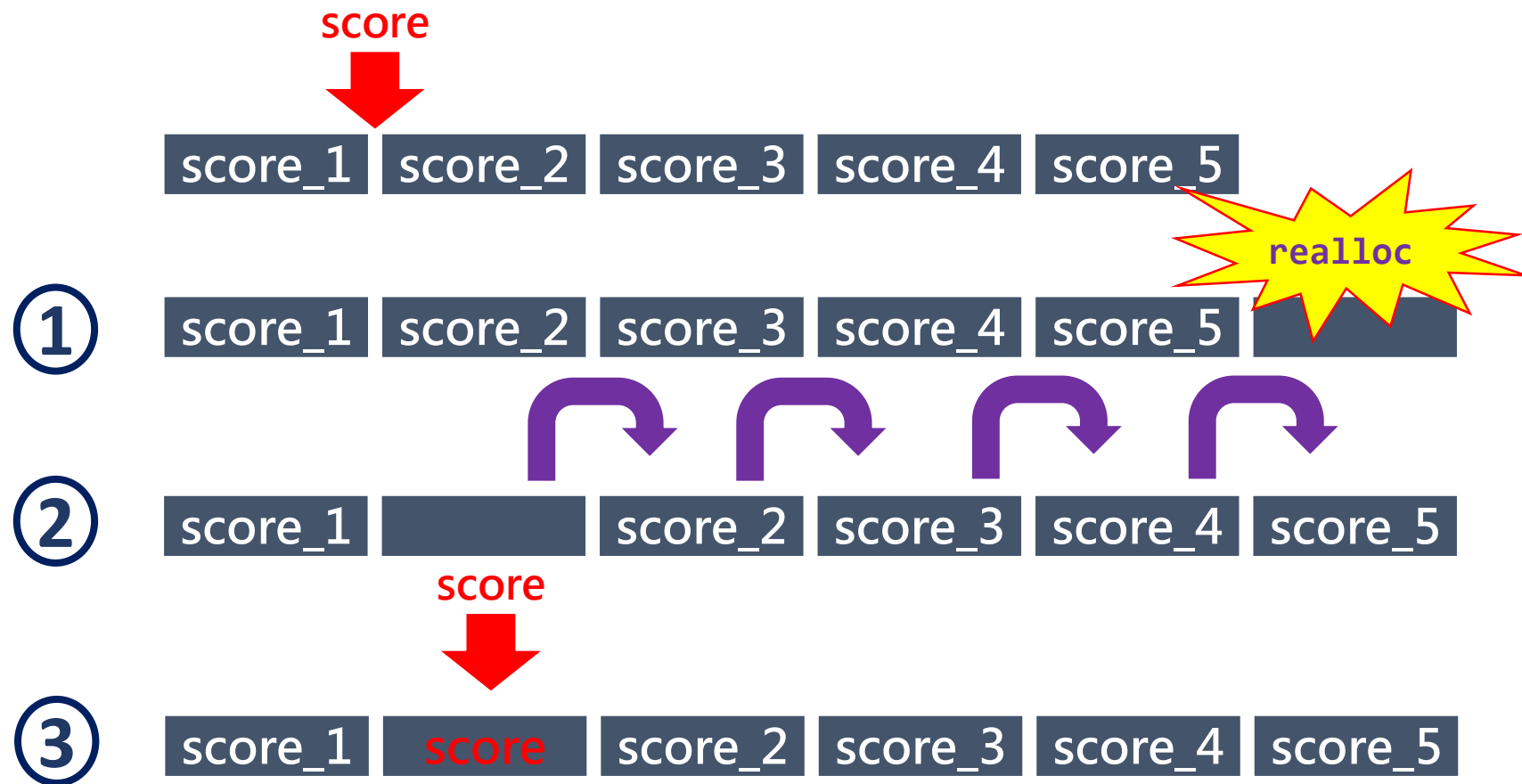
①

score_1 score_2 score_3 score_4 ~~score_5~~

score_1 score_2 score_3 score_4 

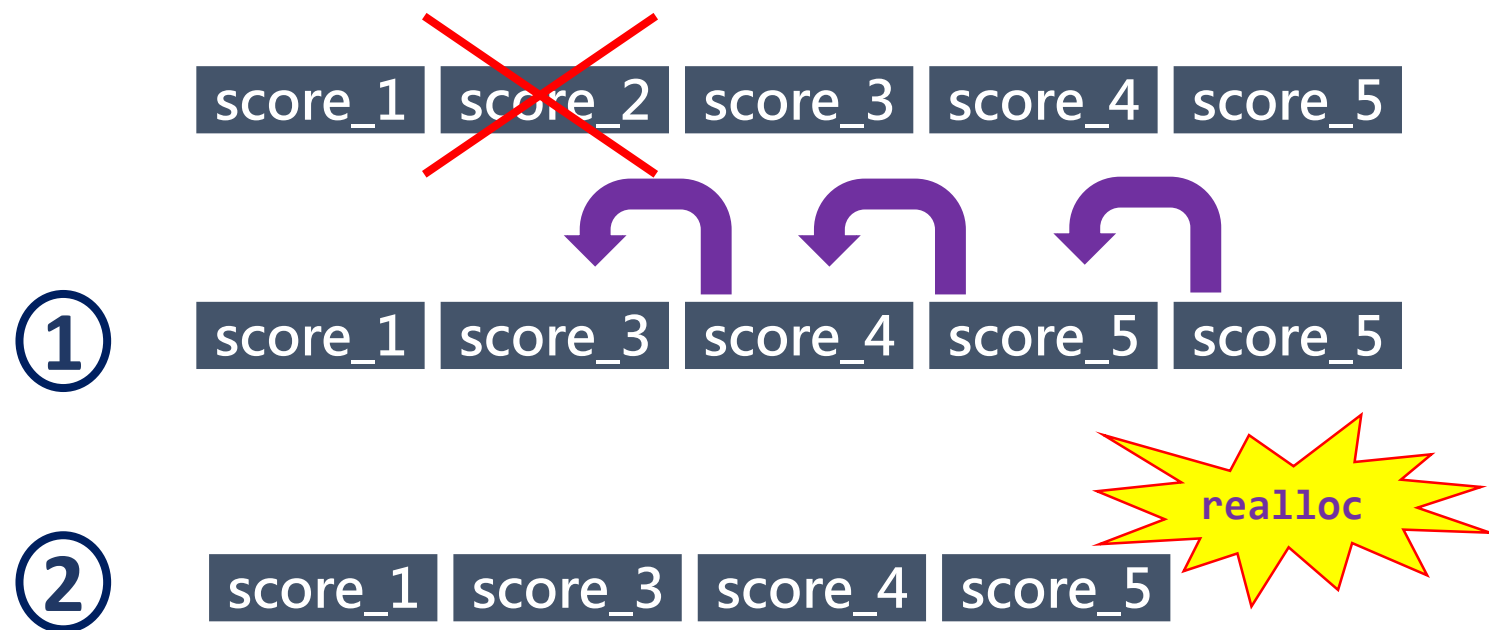
新增與刪除資料

1. 尾端新增
2. 尾端刪除
3. 插入資料
4. 刪除特定索引值
5. 刪除區間中的資料
6. 清空所有資料



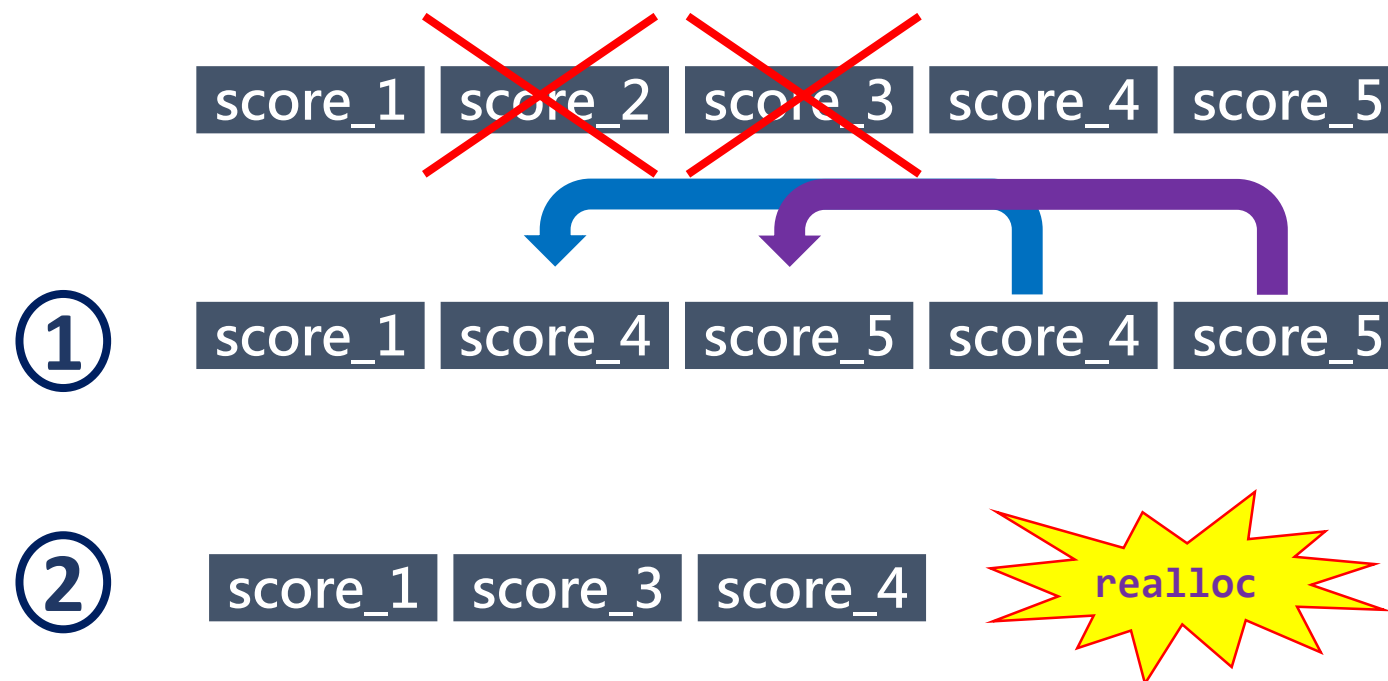
新增與刪除資料

1. 尾端新增
2. 尾端刪除
3. 插入資料
4. 刪除特定索引值
5. 刪除區間中的資料
6. 清空所有資料



新增與刪除資料

1. 尾端新增
2. 尾端刪除
3. 插入資料
4. 刪除特定索引值
5. 刪除區間中的資料
6. 清空所有資料



新增與刪除資料

1. 尾端新增
2. 尾端刪除
3. 插入資料
4. 刪除特定索引值
5. 刪除區間中的資料
6. 清空所有資料



①



Example Code

Mission

續寫向量(Vector)的類別模板，並且新增：

1. 從尾端新增資料 (Push_Back)
2. 從尾端刪除資料 (Pop_Back)

Practice

Mission

續寫向量(Vector)的類別模板，並且新增：

1. 在特定索引值插入資料 (Insert)
2. 刪除特定索引值的資料 (Erase)
3. 刪除特定區間的資料 (Erase)
4. 清空所有資料 (Clear)

reserve 與 resize

向量 (Vector) 簡介

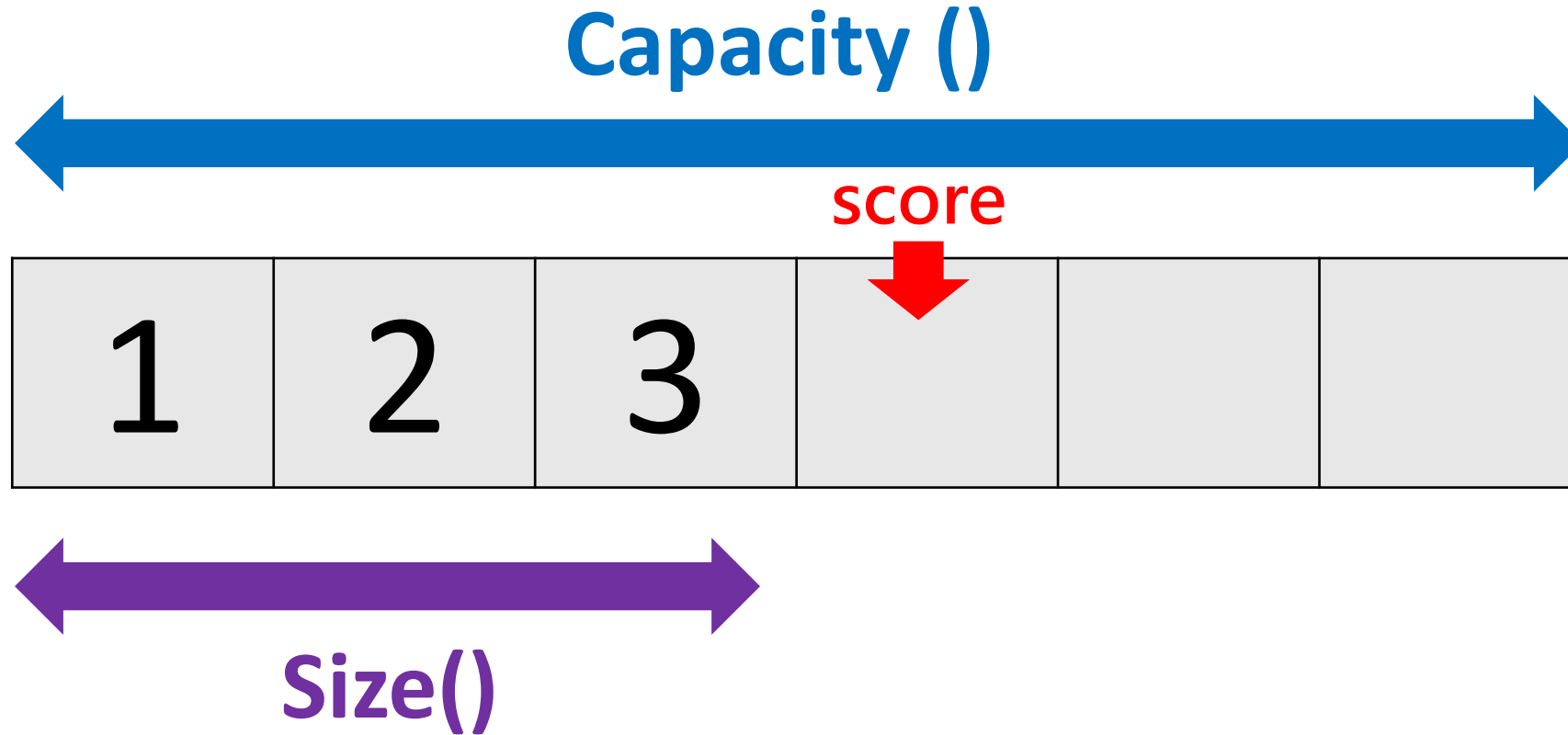
加大指標所指的空間並不容易！

1. 以 malloc 或 calloc 挖出新的空間
2. 把舊空間的資料移到新空間中
3. 釋放舊空間的資料
4. 把舊指標指到新空間
5. 長度+1

Push_Back 會是一件很花資源&時間的事情

如何改進？

reserve 與 resize



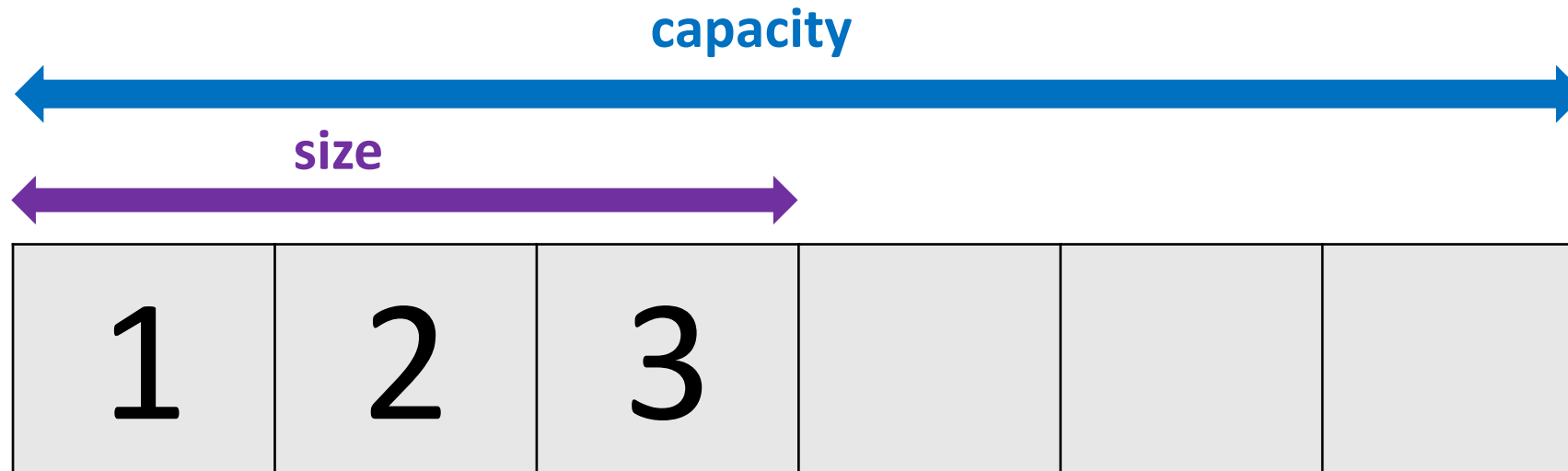
預先開好多餘的空間以供未來新增用

reserve 與 resize

reserve() v.s. resize()

reserve() 改變的是 capacity

resize() 改變的是 size



Example Code

Mission

在類別模板中新增以下私有屬性：

1. 容量 (Capacity)

並新增以下函式：

1. 擴展陣列容量 (Reserve)
2. 修改陣列長度 (Resize)

Practice

Mission

修改類別模板中的 Push_Back 函式：

- 當長度 < Capacity
 1. 直接往後塞資料
 2. 長度++
- 當長度 == Capacity
 1. 呼叫 Reserve 函式擴展陣列為原有的兩倍大小
 2. 往後塞資料
 3. 長度++

試比較兩者間的效能差異

Practice

Mission

修改類別模板中函式：

1. 尾端刪除 (Pop_Back)

✓ 無須 realloc

2. 插入資料 (Insert)

✓ 檢查 Capacity 是否足夠

A. 足夠：直接插入

B. 不夠：呼叫 Reserve 函式擴展陣列為原有的兩倍大小

3. 刪除特定索引值 (Erase)

✓ 無須 realloc，直接覆蓋

4. 刪除區間中的資料 (Erase)

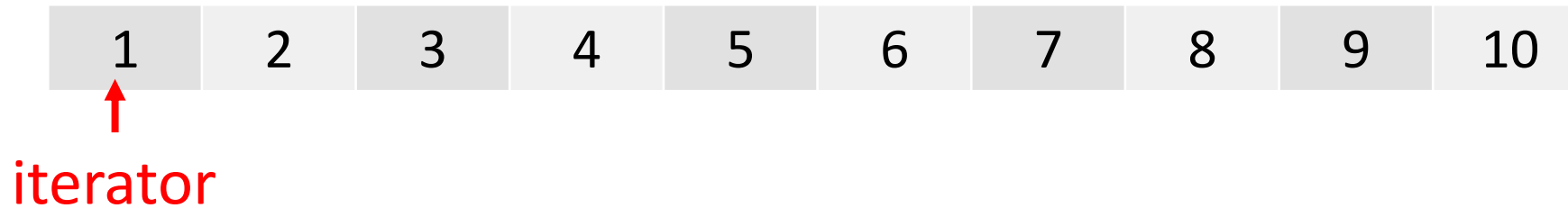
✓ 無須 realloc，直接覆蓋

迭代器(iterator)

Iterator(迭代器)

迭代器 (iterator) 指向容器 (container) 內的元素

讓使用者可以操作或存取資料



Q：為什麼不直接用索引值取值就好

A：不是所有的容器都能用索引值
(不是所有容器內的資料都連續)

Iterator(迭代器)

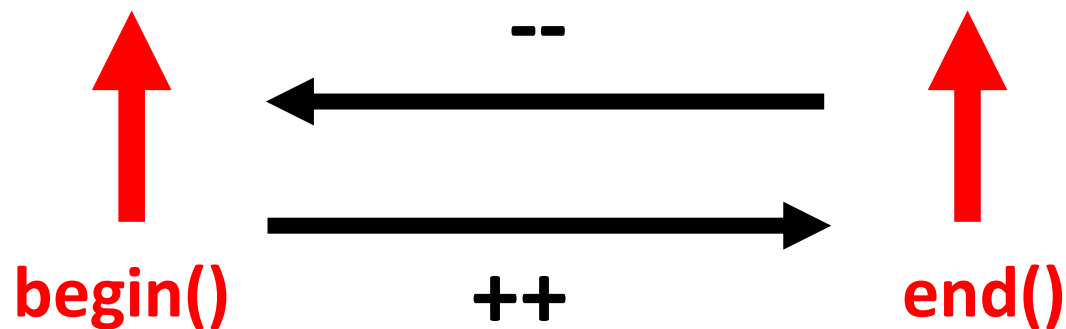
- 讓使用者可以逐個存取容器(Container) 中元素的工具
- 大部分 STL 裡的 container 都有Iterator
 - 可利用 container::iterator 宣告
- Container 都有 begin()、end() 函式
 - 回傳容器的第一個元素的記憶體位址
 - 回傳最後一個元素後一個位置的記憶體位址

迭代器(iterator)

Capacity ()

Size()

原則：
前閉後開



迭代器(iterator)

Capacity ()

Size()

原則：
前閉後開



↑
rend()

++



--



↑
rbegin()

r = reverse

Example Code

Mission

在 **Vector** 類別裡新增 **Iterator** 類別，並：

1. 重載運算子

- ✓ ++
- ✓ --
- ✓ =
- ✓ ==
- ✓ !=
- ✓ *

2. 把之前函式中的索引值都改成迭代器

Example Code

Mission

在 Vector 類別裡新增 Begin()、End()：

1. Begin()：回傳第一個元素的指標
2. End()：回傳最後一個元素**下一筆資料**的指標
3. 比較運算子

試著操作 Vector 類別並利用 Iterator 輸出 Vector 內的資料

Practice

Mission

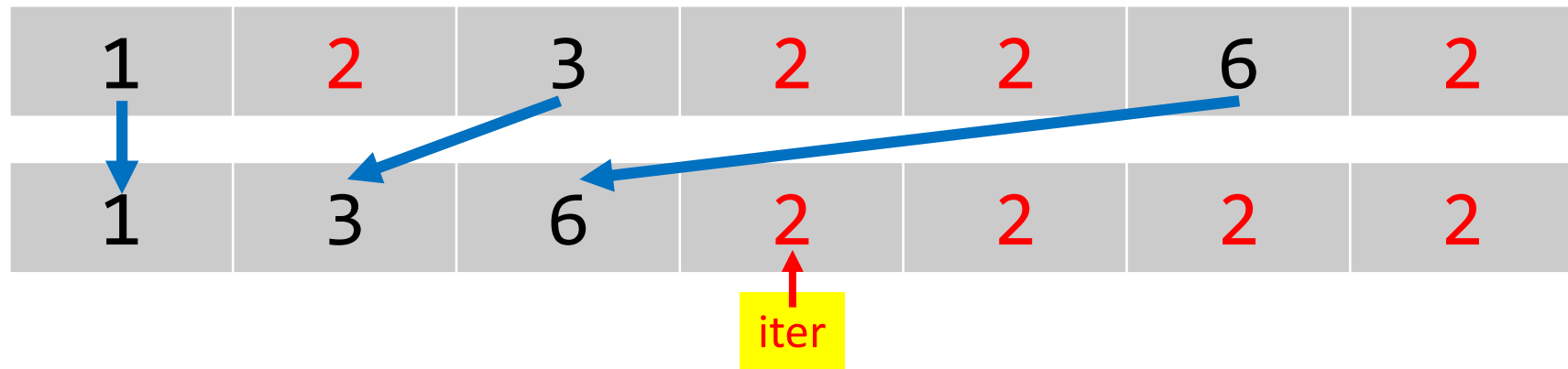
新增外部函式

1. 搜尋 (Find)

- ✓ 回傳搜尋到的特定資料迭代器
- ✓ 若未搜尋到，則回傳 End()

2. 刪除特定資料 (Remove)

- ✓ 把特定區間中的特定資料移到最後面
- ✓ 並保持其餘資料的次序
- ✓ 回傳第一個特定資料的位置



Vector @ C++ STL

Vector @ C++ STL

- 資料放在記憶體中的連續位置
- 支援隨機存取、索引值存取
- 數列尾端新增刪除很快： $O(1)$
- 數列中間新增刪除費時： $O(n)$
- 以 template 撰寫
 - 可儲存任意類型的變數
 - 包含自定義的資料型態

Vector @ C++ STL

標頭檔

```
#include <vector>
```

宣告

這就是 Template !


```
vector<資料型別> 變數名稱;
```

```
vector<int> v_int;  
vector<string> v_string;  
vector<int * > v_int_pointer;  
vector<float> v_float;
```

Vector @ C++ STL

初始化

1. 利用 `push_back`
2. 指定長度與初始值
3. 利用 `fill` 給值
4. 仿照陣列給定初始值
5. 從另一陣列或向量初始化



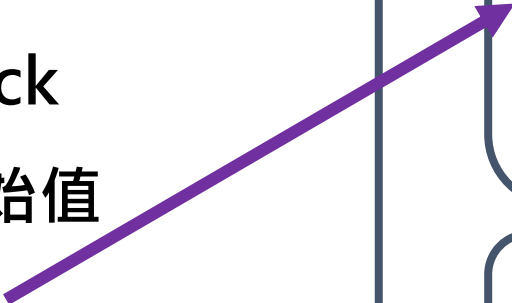
```
vector<int> vec;  
vec.push_back(1);  
vec.push_back(2);  
vec.push_back(3);  
// 1 2 3
```

```
vector<int> vec(3);  
// 0 0 0  
vector<int> vec(3,100);  
// 100 100 100
```

Vector @ C++ STL

初始化


1. 利用 `push_back`
2. 指定長度與初始值
3. 利用 `fill` 給值
4. 仿照陣列給定初始值
5. 從另一陣列或向量初始化



```
vector<int> vec(3);  
// 0 0 0  
fill(vec.begin(), vec.end(), 5);  
// 5 5 5
```



```
vector<int> vec{4, 5, 6};  
// 4 5 6
```



```
int arr[] = {1, 2, 3};  
vector<int> vec1(arr, arr+3);  
// 1 2 3  
vector<int> vec2(vec1.begin(), vec1.end());  
// 1 2 3
```

Vector @ C++ STL

語法	功能	語法	功能
<code>vector[i]</code>	存取索引值 <code>i</code> 的元素	<code>vector.insert()</code>	插入元素至特定位置
<code>vector.at(i)</code>	存取索引值 <code>i</code> 的元素	<code>vector.erase()</code>	刪除特定元素
<code>vector.front()</code>	第一個元素的值	<code>vector.clear()</code>	清空所有元素
<code>vector.back()</code>	最後一個元素的值	<code>vector.empty()</code>	判斷是否為空
<code>vector.push_back()</code>	尾端新增一個元素	<code>vector.capacity()</code>	回傳 capacity
<code>vector.pop_back()</code>	尾端刪除一個元素	<code>vector.begin()</code>	第一個元素的迭代器
<code>vector.reserve()</code>	擴展陣列容量	<code>vector.end()</code>	最後一個元素後一個的迭代器
<code>vector.resize()</code>	擴展陣列長度		

Vector @ C++ STL

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

int main() {
    vector<int> v{1, 2, 3, 4, 5, 6};
    v.push_back(7); // v = 1 2 3 4 5 6 7
    for(int i=0;i<v.size();i++)
        cout << v[i] << " ";
    cout << endl;

    v[1] = 100; // v = 1 100 3 4 5 6 7
    v.at(2)= 0; // v = 1 100 0 4 5 6 7
    v.pop_back(); // v = 1 100 0 4 5 6
    v.insert(v.begin()+2, 10); // v = 1 100 10 0 4 5 6
    v.erase(v.begin()); // v = 100 10 0 4 5 6
    v.erase(v.begin()+2,v.begin()+4); // v = 100 10 5 6
    for(vector<int>::iterator iter=v.begin();iter!=v.end();iter++)
        cout << *iter << " ";
    cout << endl;

    return 0;
}
```


Vector @ C++ STL

插入資料

1. `vector.insert(iterator, value);`
2. `vector.insert(iterator, size, value);`
3. `vector.insert(iterator, iterator1, iterator2);`

```
vector<int> vec{1, 2, 3};  
vec.insert(vec.begin(), 0);  
// 0 1 2 3  
vec.insert(vec.begin() + 2, 4);  
// 0 1 4 2 3  
vec.insert(vec.begin() + 2, {50, 100});  
// 0 1 50 100 4 2 3  
vec.insert(vec.begin() + 3, 2, 5);  
// 0 1 50 5 5 100 4 2 3  
vec.insert(vec.begin() + 1, vec.begin() + 1, vec.begin() + 3);  
// 0 1 50 1 50 5 5 100 4 2 3
```

Vector @ C++ STL

刪除資料

1. `vector.erase(iterator);`
2. `vector.erase(iterator1, iterator2);`

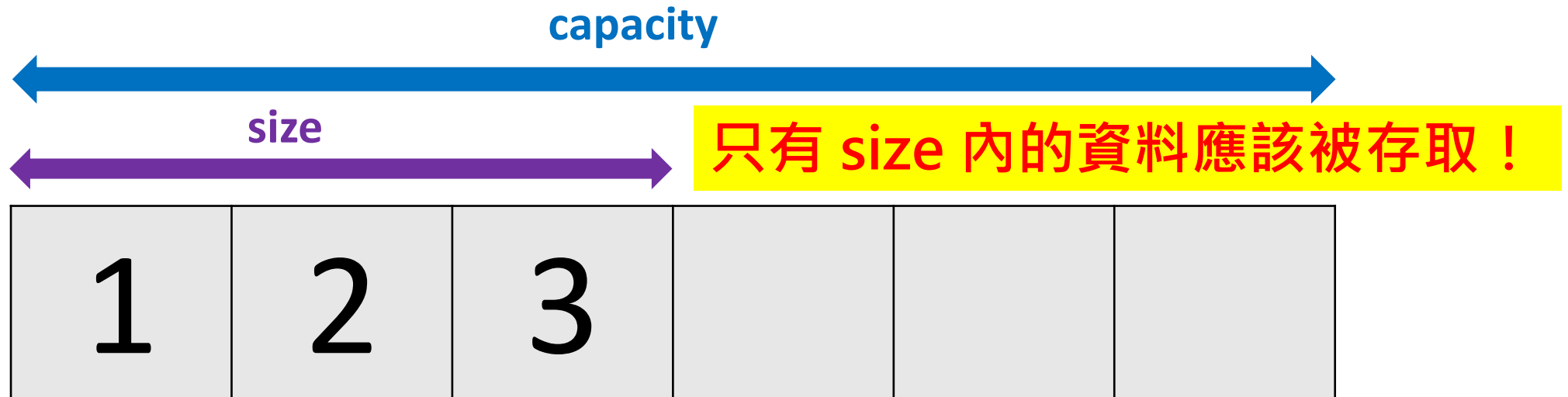
```
vector<int> vec{1, 2, 3, 4, 5, 6, 7};  
vec.erase(vec.begin() + 2);  
// 1 2 3 4 5 6 7  
vec.erase(vec.begin() + 1, vec.begin() + 4);  
// 1 2 4 5 6 7
```

Vector @ C++ STL

reserve() v.s. resize()

reserve() 改變的是 capacity

resize() 改變的是 size



Vector @ C++ STL

empty() v.s. size() == 0

優先使用 empty()

Why ?

size() : $O(1)$? $O(n)$?

Vector @ C++ STL

迭代器(iterator)的宣告

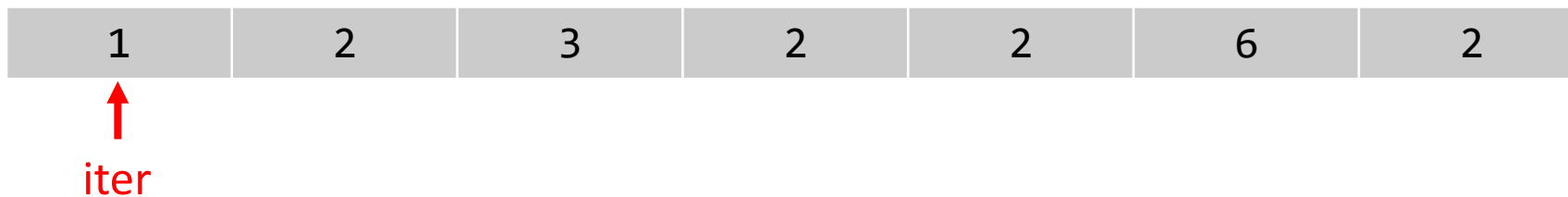
`vector< 資料型別 >::iterator` 迭代器名稱;

```
vector<int>::iterator it;  
for(it=begin ; it!=end ; it++){  
    cout << *it << endl;  
}
```

Vector @ C++ STL

iterator 的陷阱

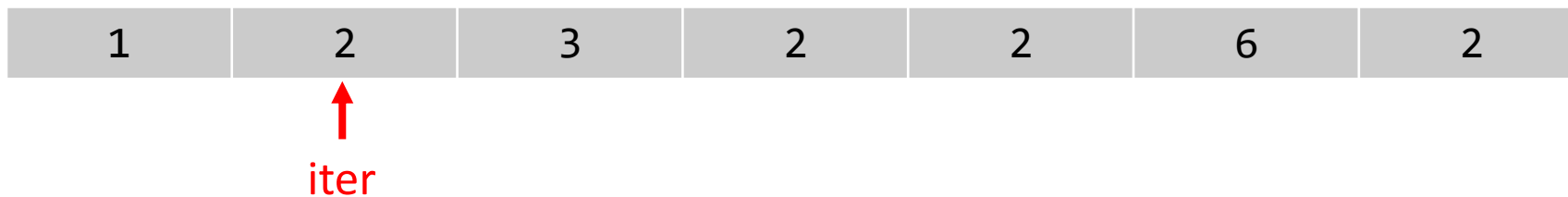
```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



Vector @ C++ STL

iterator 的陷阱

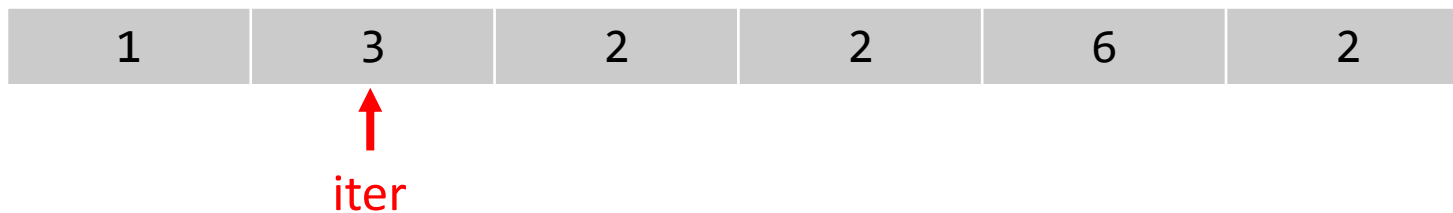
```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



Vector @ C++ STL

iterator 的陷阱

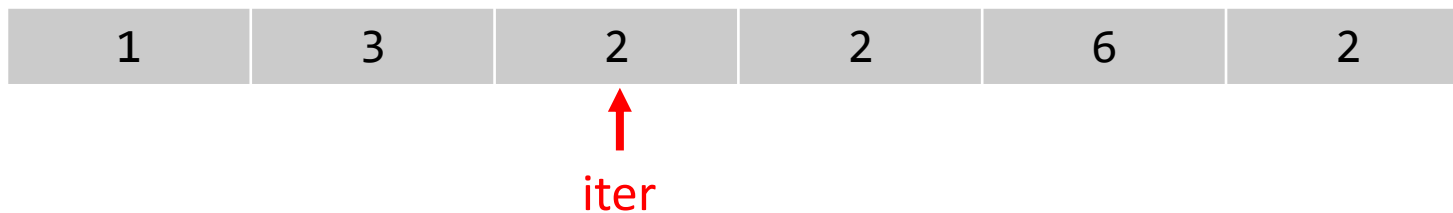
```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



Vector @ C++ STL

iterator 的陷阱

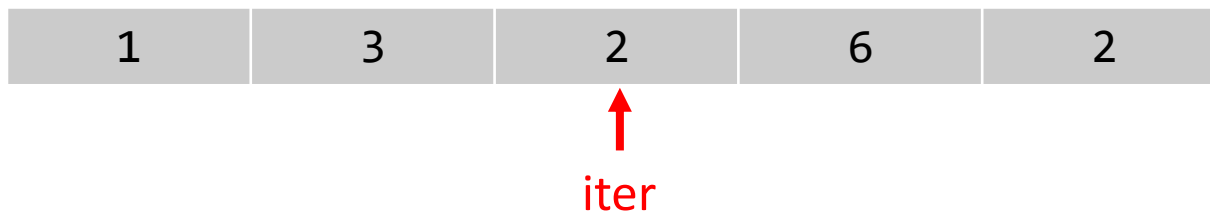
```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



Vector @ C++ STL

iterator 的陷阱

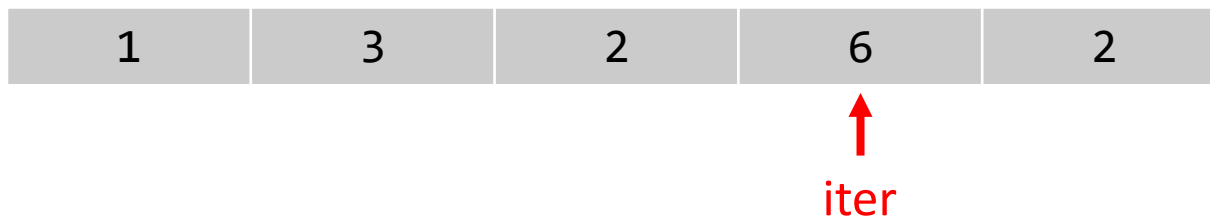
```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



Vector @ C++ STL

iterator 的陷阱

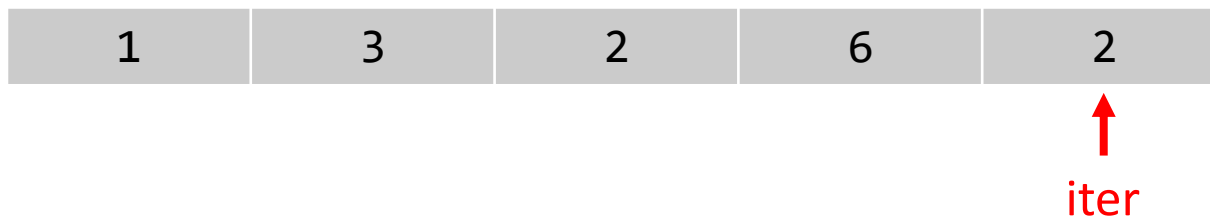
```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



Vector @ C++ STL

iterator 的陷阱

```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



Vector @ C++ STL

iterator 的陷阱

```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



↑
iter

Vector @ C++ STL

iterator 的陷阱

```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



↑
end

↑
iter

Vector @ C++ STL

iterator 的陷阱

```
vector<int> vec{1, 2, 3, 2, 2, 6, 2};  
for(auto iter = vec.begin(); iter!=vec.end(); iter++){  
    if(*iter == 2){  
        vec.erase(iter);  
    }  
}
```



↑
end

↑
iter

Vector @ C++ STL

remove

`remove(vec.begin(), vec.end(), value)`

- ※ 刪除資料後的元素移到 vector 前，不影響 vector 長度
- ※ 需 `#include <algorithm>`
- ※ `remove` 在 `std` 下，不屬於 `vector`，因此不能改變長度

```
vector<int> vec{1, 2, 3, 4, 2, 6, 2};  
auto iter = remove(vec.begin(), vec.end(), 2);  
// 1 3 4 6 2 6 2
```

1	2	3	4	2	6	2
1	3	4	6	2	2	2



`iter`

Vector @ C++ STL

erase-remove

```
vec.erase(remove(vec.begin(), vec.end(), value), vec.end());
```

※ 刪除 vector 中的特定資料

※ remove 不改變 vector 長度，須和 erase 合作

```
vector<int> vec{1, 2, 3, 4, 2, 6, 2};  
vec.erase(remove(vec.begin(), vec.end(), 2), vec.end());  
// 1 2 3 4 2 6 2
```

Vector @ C++ STL

二維 vector 的宣告 (M X N)

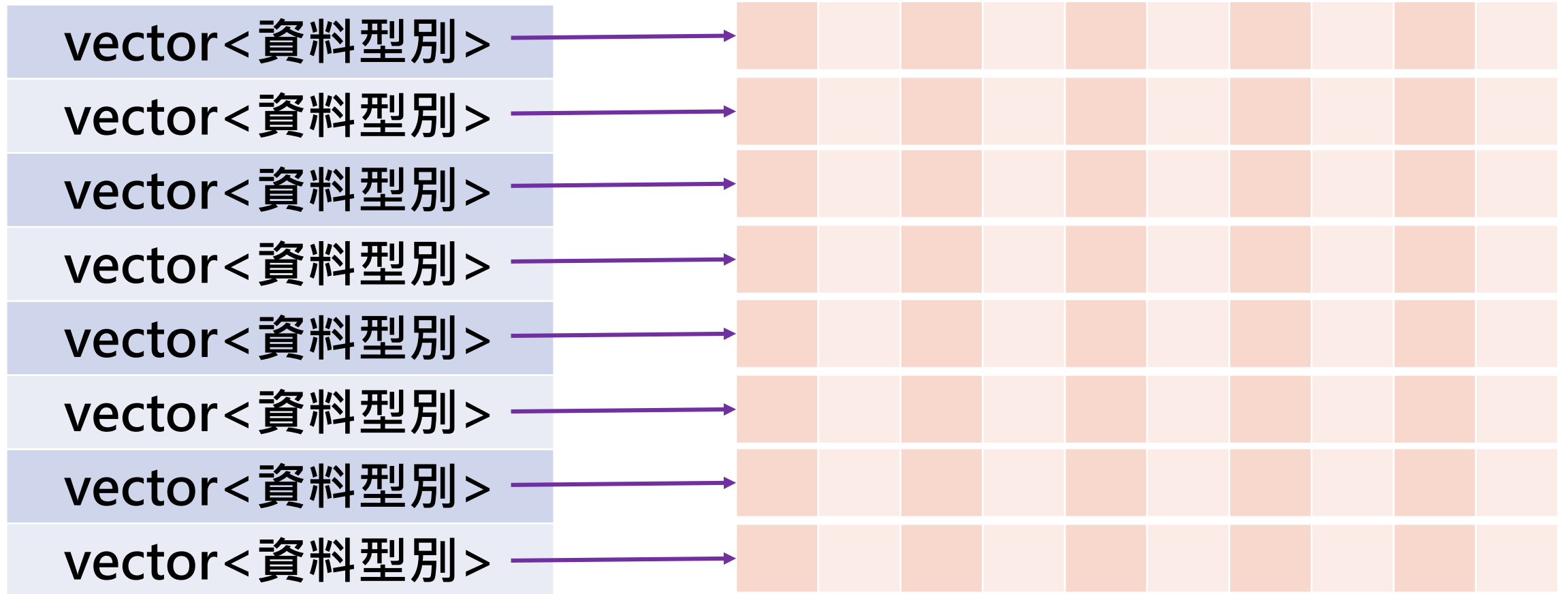
`vector< vector<資料型別> > 變數名稱(M);`

`vector< 資料型別 > 變數名稱[M];`

```
for(int i =0; i<變數名稱.size(); i++)  
{  
    變數名稱[i].resize(N);  
}
```

Vector @ C++ STL

`vector< vector<資料型別> >`



Vector @ C++ STL

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

using namespace std;

int main() {
    vector<int> vec;
    int* p,*last;
    for(int i=0;i<50000;i++){
        vec.push_back(i);
        p = &vec[0];
        if(last!=p||i==0)
            cout << setw(5) << i << "-th address:" <<p << endl;
        last = p;
    }
    return 0;
}
```

```
0-th address:0x760d18
1-th address:0x760d68
2-th address:0x767a28
4-th address:0x767a40
8-th address:0x7612e0
16-th address:0x761328
32-th address:0x7614a0
64-th address:0x76abd0
128-th address:0x76add8
256-th address:0x76b1e0
512-th address:0x76b9e8
1024-th address:0x76c9f0
2048-th address:0x770048
4096-th address:0x774050
8192-th address:0x77c058
16384-th address:0x78c060
32768-th address:0x7ac068
```

Example Code

Mission

1. 宣告一個 vector，依序把使用者輸入的整數放入，直到輸入為 0，最後印出使用者輸入了幾個數字
2. 承上題，用迭代器 (iterator) 印出 vector 內的所有資料。
(Hint : `begin()~end()`)
3. 宣告一個 9x9 的二維 vector，把九九乘法表的值放入

Example Code

Mission

Try LeetCode #27. Remove Element

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` in-place. The relative order of the elements may be changed.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of the array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the final result. It does not matter what you leave beyond the first `k` elements.

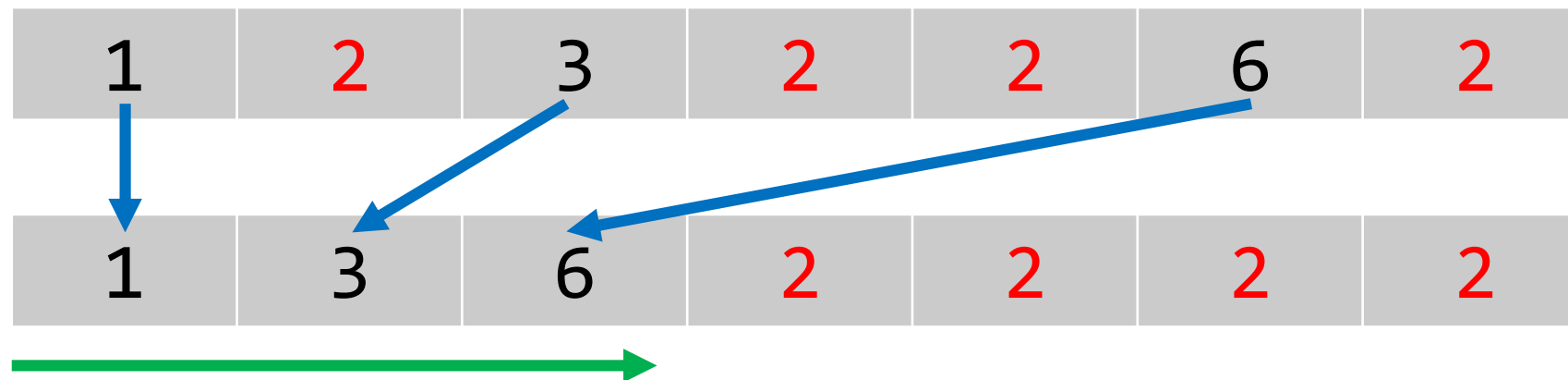
Return `k` after placing the final result in the first `k` slots of `nums`.

Do not allocate extra space for another array. You must do this by modifying the input array in-place with $O(1)$ extra memory.

Ref : <https://leetcode.com/problems/remove-element/>

Example Code

只對非目標值的元素做賦值



開一個計數器紀錄現在新增到哪

Example Code

1

2

3

2

2

6

2

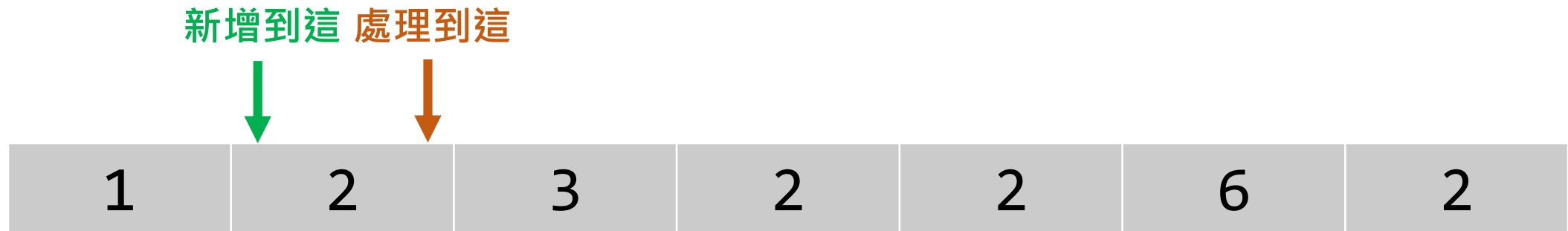
Example Code

新增到這 處理到這



swap

Example Code



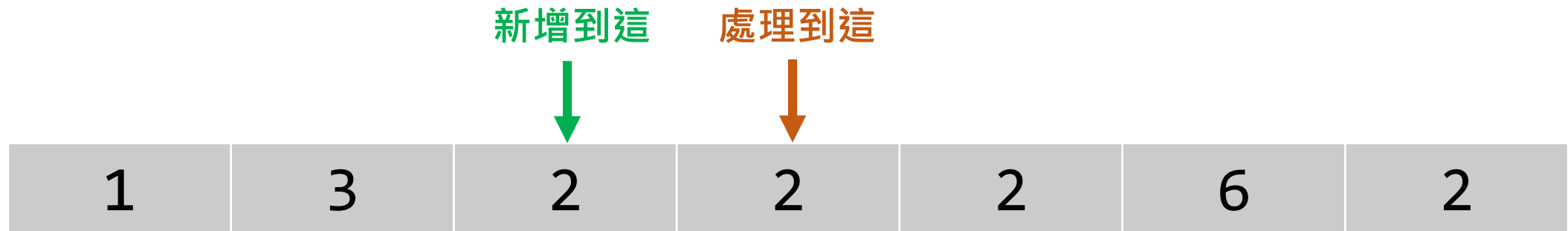
Example Code



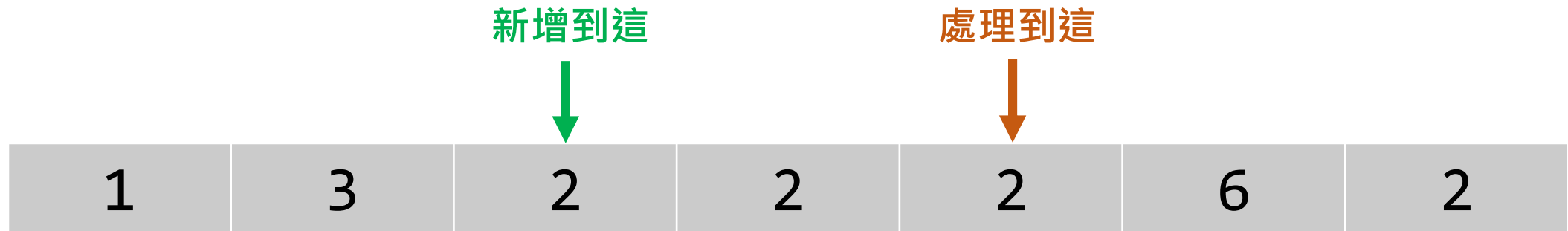
Example Code



Example Code



Example Code



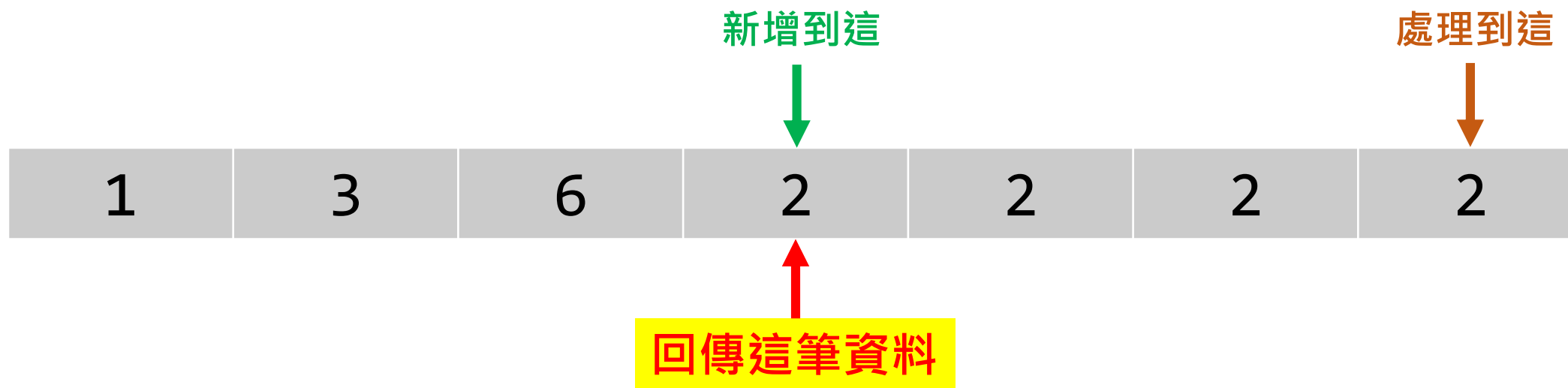
Example Code



Example Code



Example Code



Partitioning

Practice

APCS 最大和 2016/10/29 #2

給定 N 群數字，每群都恰有 M 個正整數。若從每群數字中各選擇一個數字(假設第 i 群所選出數字為 t_i)，將所選出的 N 個數字加總即可得總和 $S = t_1 + t_2 + \dots + t_N$ 。請寫程式計算 S 的最大值(最大總和)，並判斷各群所選出的數字是否可以整除 S 。

Practice

APCS 最大和 2016/10/29 #2

- 輸入格式
 1. 第一行有二個正整數 N 和 M ， $1 \leq N \leq 20$ ， $1 \leq M \leq 20$ 。
 2. 接下來的 N 行，每一行各有 M 個正整數 x_i ，代表一群整數，數字與數字間有一個空格，且 $1 \leq i \leq M$ ，以及 $1 \leq x_i \leq 256$ 。
- 輸出格式
 1. 第一行輸出最大總和 S 。
 2. 第二行按照被選擇數字所屬群的順序，輸出可以整除 S 的被選擇數字，數字與數字間以一個空格隔開，最後一個數字後無空白；若 N 個被選擇數字都不能整除 S ，就輸出-1。

Practice

APCS 最大和 2016/10/29 #2

- 範例一

- 輸入

3 2

1 5

6 4

1 1

- 正確輸出

12

6 1

(說明) 挑選的數字依序是 5,6,1，總和 $S=12$ 。而此三數中可整除 S 的是 6 與 1，6 在第二群，1 在第 3 群所以先輸出 6 再輸出 1。注意，1 雖然也出現在第一群，但她不是第一群中挑出的數字，所以順序是先 6 後 1。

Practice

APCS 最大和 2016/10/29 #2

- 範例二

- 輸入

4 3

6 3 2

2 7 9

4 7 1

9 5 3

- 正確輸出

31

-1

(說明) 挑選的數字依序是 6,9,7,9 , 總和 $S=31$ 。
而此四數中沒有可整除 S 的 , 所以第二行輸出-1。

Practice

Mission

Try LeetCode #867. Transpose Matrix

Given a 2D integer array matrix, return the transpose of matrix.

The transpose of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.



Ref : <https://leetcode.com/problems/transpose-matrix/>

Practice

Mission

Try LeetCode #54. Spiral Matrix

Given an $m \times n$ matrix, return all elements of the matrix in spiral order.

1	→	2	→	3
4	→	5		↓
↑				↓
7	←	8	←	9

Ref : <https://leetcode.com/problems/spiral-matrix/>