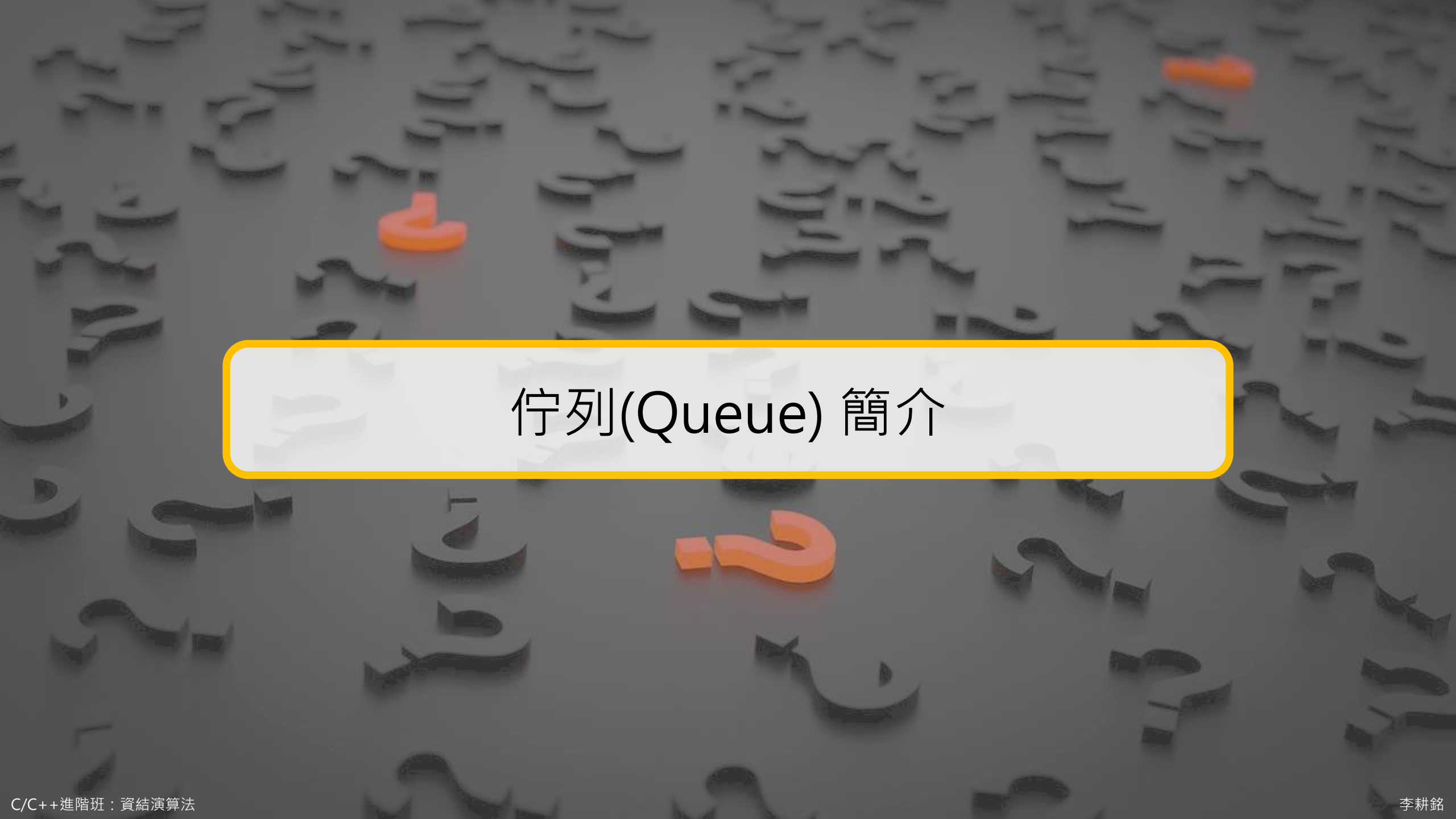
C/C++ 進階班
資料結構

佇列
(Queue)

李耕銘

課程大綱

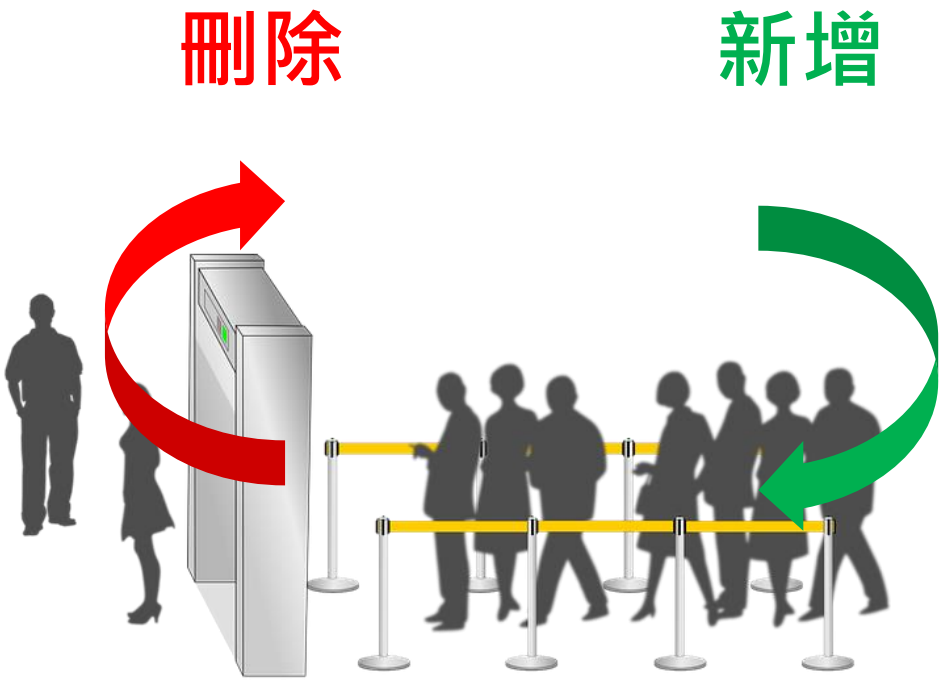
- 佇列(Queue)簡介
- 佇列(Queue)實作
- Queue @ C++ STL
- Deque
- Priority Queue



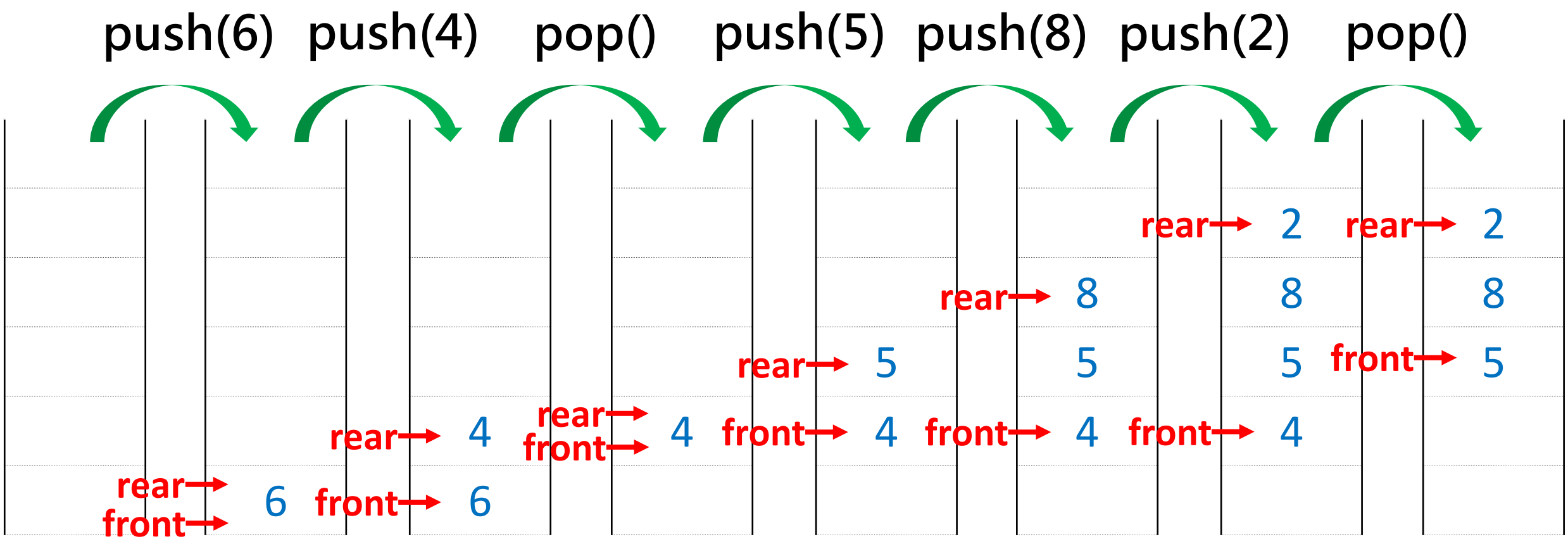
佇列(Queue) 簡介

佇列(Queue)

- 佇列(Queue)
  - 插入、刪除在異側
  - first-in-first-out(FIFO)
- 常見的操作
  1. push：新增一筆資料
  2. pop：刪除一筆資料
  3. front：回傳前端的資料。
  4. rear：回傳末端的資料
  5. empty：確認 queue 裡是否有資料
  6. size：回傳 queue 的資料個數



佇列(Queue)


push(6)
push(4)
pop()
push(5)
push(8)
push(2)
pop()
rear
front
6
rear
front
4
6
rear
front
4
rear
front
5
4
front
4
rear
front
8
5
4
front
rear
2
8
5
4
front
rear
2
8
front
5

Practice

給定 queue = {1, 2, 3}，方向為右進左出，經過以下操作後，該 queue 的最後內容為何？

1. push(4)
2. pop()
3. push(5)
4. push(6)
5. push(7)
6. pop()
7. pop()

Practice

給定 queue = {1, 2, 3}，方向為右進左出，經過以下操作後，該 queue 的最後內容為何？

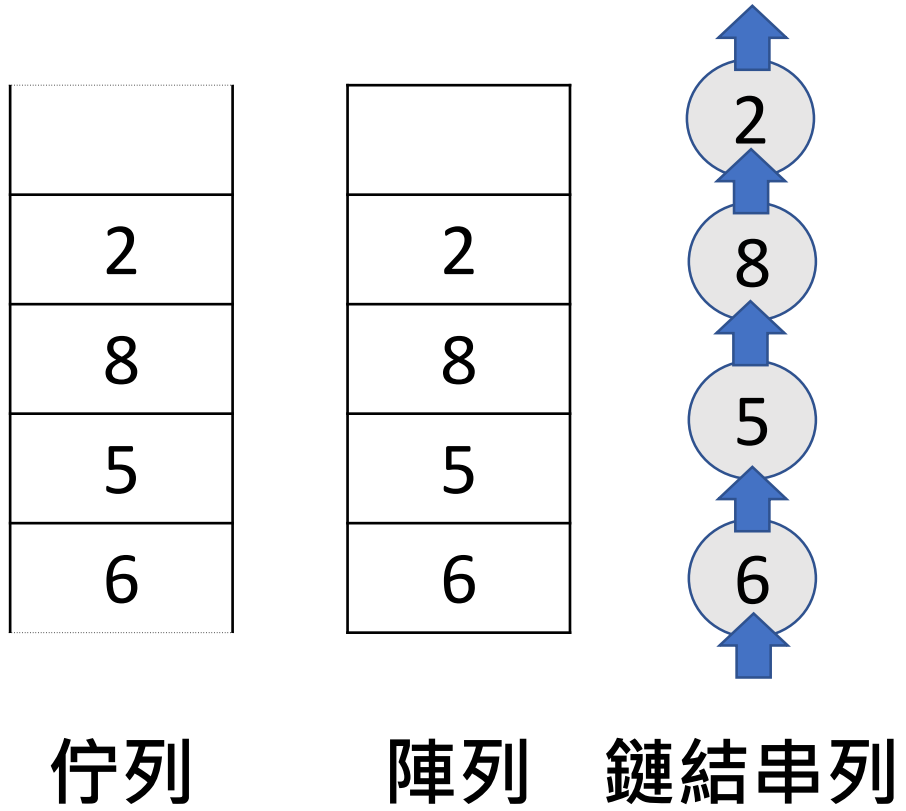
1. push(4)：{1, 2, 3, 4}
2. pop()：{2, 3, 4}
3. push(5)：{2, 3, 4, 5}
4. push(6)：{2, 3, 4, 5 ,6}
5. push(7)：{2, 3, 4, 5, 6, 7}
6. pop()：{3, 4, 5, 6, 7}
7. pop()：{4, 5, 6, 7}

佇列(Queue)的用途

- 依序處理先前的資訊
  - 常用來做資料的緩衝區
    1. 記憶體(標準輸出、檔案寫入)
    2. 引表機輸出
    3. CPU的工作排程
- 迷宮探索、搜尋
  - Breadth-First Search
- 無法得知 queue 裡有哪些資料
  - 只能以 pop() 一個個把資料拿出來

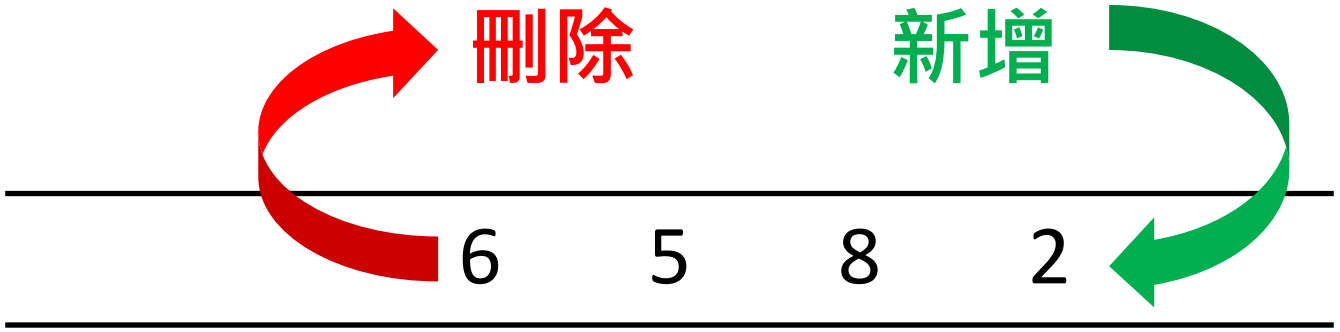
佇列(Queue)

- 線性佇列
  - 可以以陣列或鏈結串列來實作
  - 佇列在陣列中容易遇到容量問題



佇列(Circular Queue)

- 佇列在陣列中容易遇到容量的問題
  - ➢ 陣列長度 N，最多只能 push N次

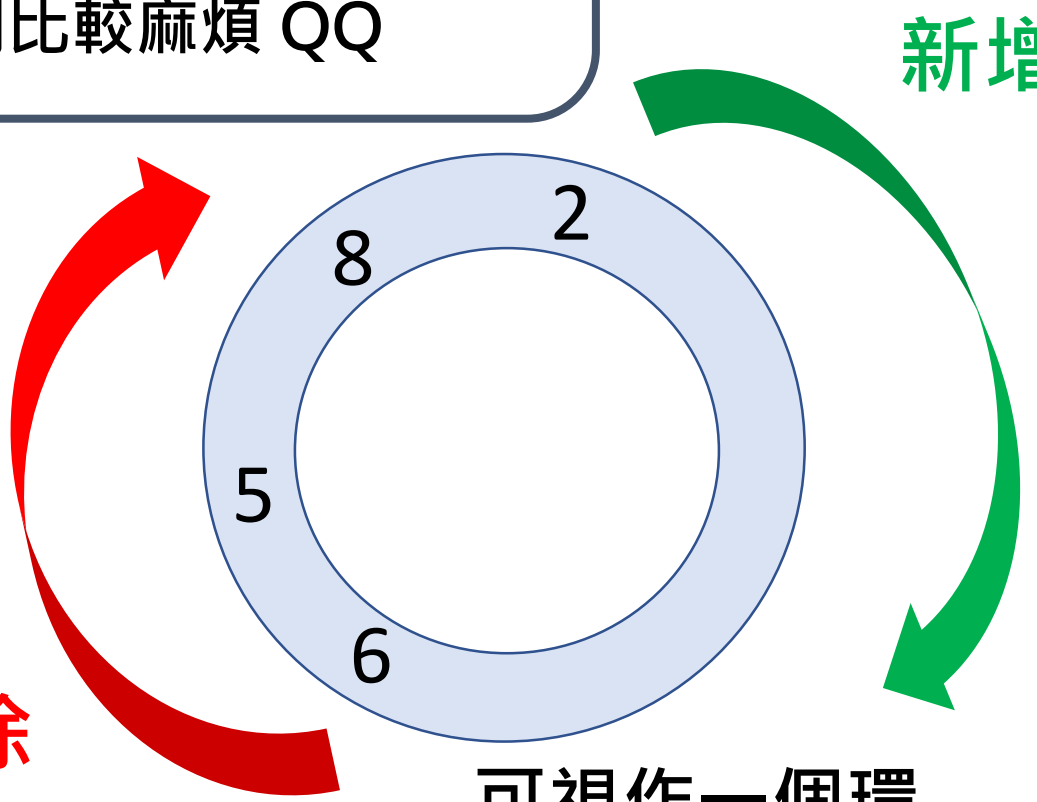


一下就滿了 QQ

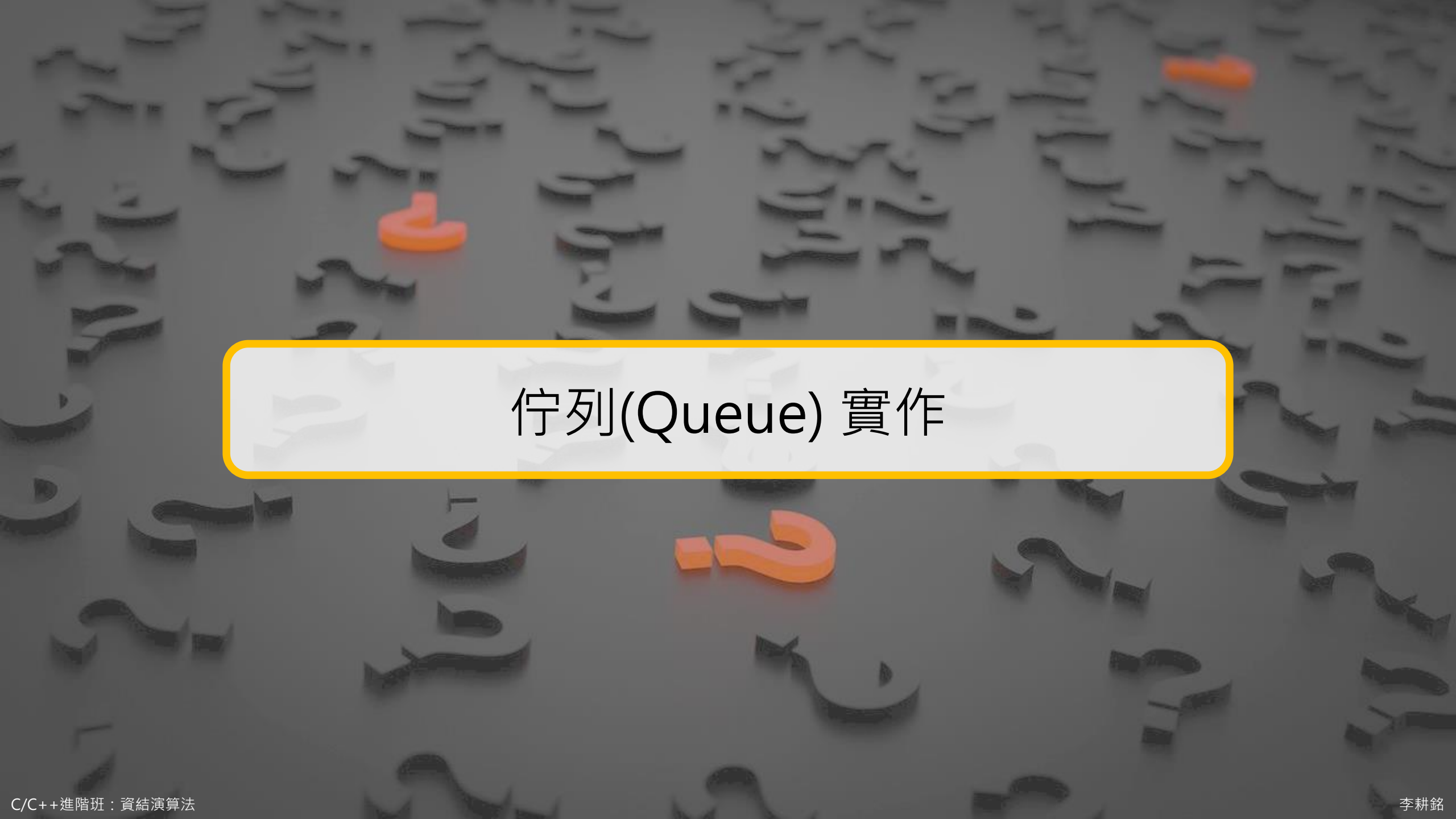
環狀佇列(Circular Queue)

- 佇列在陣列中容易遇到容量的問題
  - 解決方式：環狀佇列(Circular Queue)
  - 但環狀佇列擴充空間比較麻煩 QQ

塞滿了就回頭塞



可視作一個環



佇列(Queue) 實作

佇列實作

- 以鏈結串列實作佇列的類別
  - (回憶)宣告一個包含資料與指標的 Node

```
template <typename T>
struct Node{
    T Data;
    Node* Next;
};
```

佇列實作

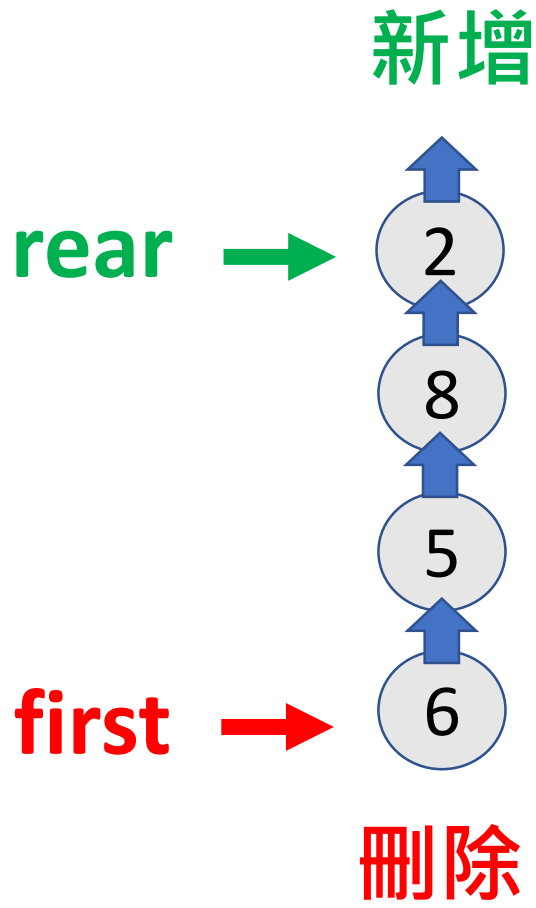
- 以鏈結串列實作佇列的類別
  1. 在佇列的類別中宣告 First 與 Rear 指標
  2. First, Rear 初始化為空指標 0
  3. 分別完成下列函式
     a) Front
     b) Back
     c) Empty
     d) Size
     e) Push
     f) Pop
     g) Print_Queue (正常不會有)

```
template <typename T>
class Queue{
   private:
      Node<T>* First;
      Node<T>* Rear;
   public:
      Queue();
      T Front();
      T Back();
      bool Empty();
      int Size();
      void Push(T);
      void Pop();
      void Print_Queue();
};
```

佇列實作

- Front→回傳刪除端的資料
  1. 確認 First 不為空指標
  2. 回傳 First 指到的資料

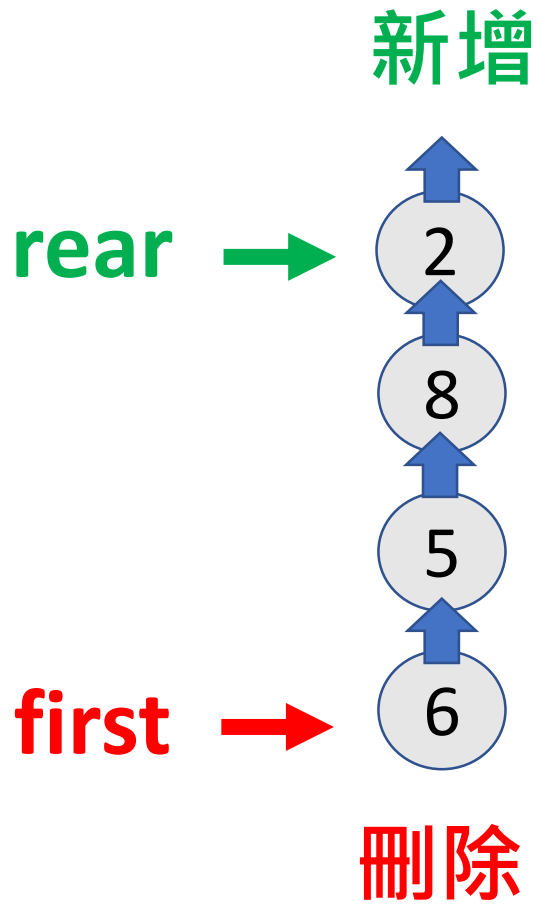
- Rear→回傳新增端的資料
  1. 確認 Rear 不為空指標
  2. 回傳 Rear 指到的資料



佇列實作

- Empty→確認 Queue 內是否有資料
  1. 確認 First 跟 Rear 是否為空指標

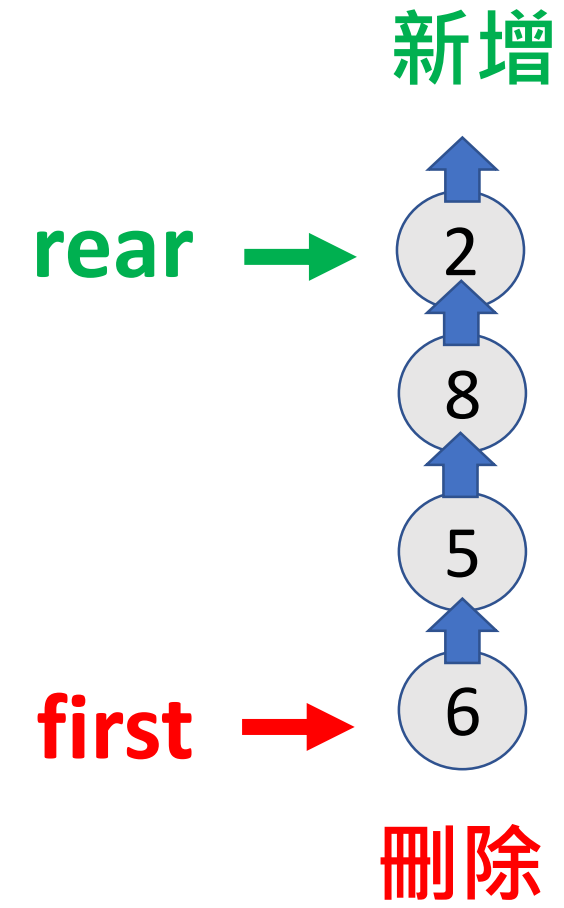
- Size→查詢 Queue 的長度
  1. 確認 First 跟 Rear 皆不為空指標
  2. 從 First 走到 Rear 需要經過幾個 node



佇列實作

- Push→新增一筆資料
  1. 宣告一個指向空指標的新 Node
  2. 讓 Rear 指到的 Node 指向該 Node
  3. 讓 Rear 也指向該 Node

- Pop→刪除一筆資料
  1. 讓 First 指向下個 Node
  2. 若下個 Node 為空指標，則令 Rear 為空指標



Example Code

Mission

初始化一個佇列，並完成其中的：

1. 建構式
2. Print_Queue()

```
template <typename T>
class Queue{
   private:
      Node<T>* First;
      Node<T>* Rear;
   public:
      Queue();
      T Front();
      T Back();
      bool Empty();
      int Size();
      void Push(T);
      void Pop();
      void Print_Queue();
};
```

Practice

Mission

完成以下六個函式：

1. Empty()
2. Size()
3. Front()
4. Back
5. Push()
6. Pop()

```
template <typename T>
class Queue{
   private:
      Node<T>* First;
      Node<T>* Rear;
   public:
      Queue();
      T Front();
      T Back();
      bool Empty();
      int Size();
      void Push(T);
      void Pop();
      void Print_Queue();
};
```

Practice

Mission

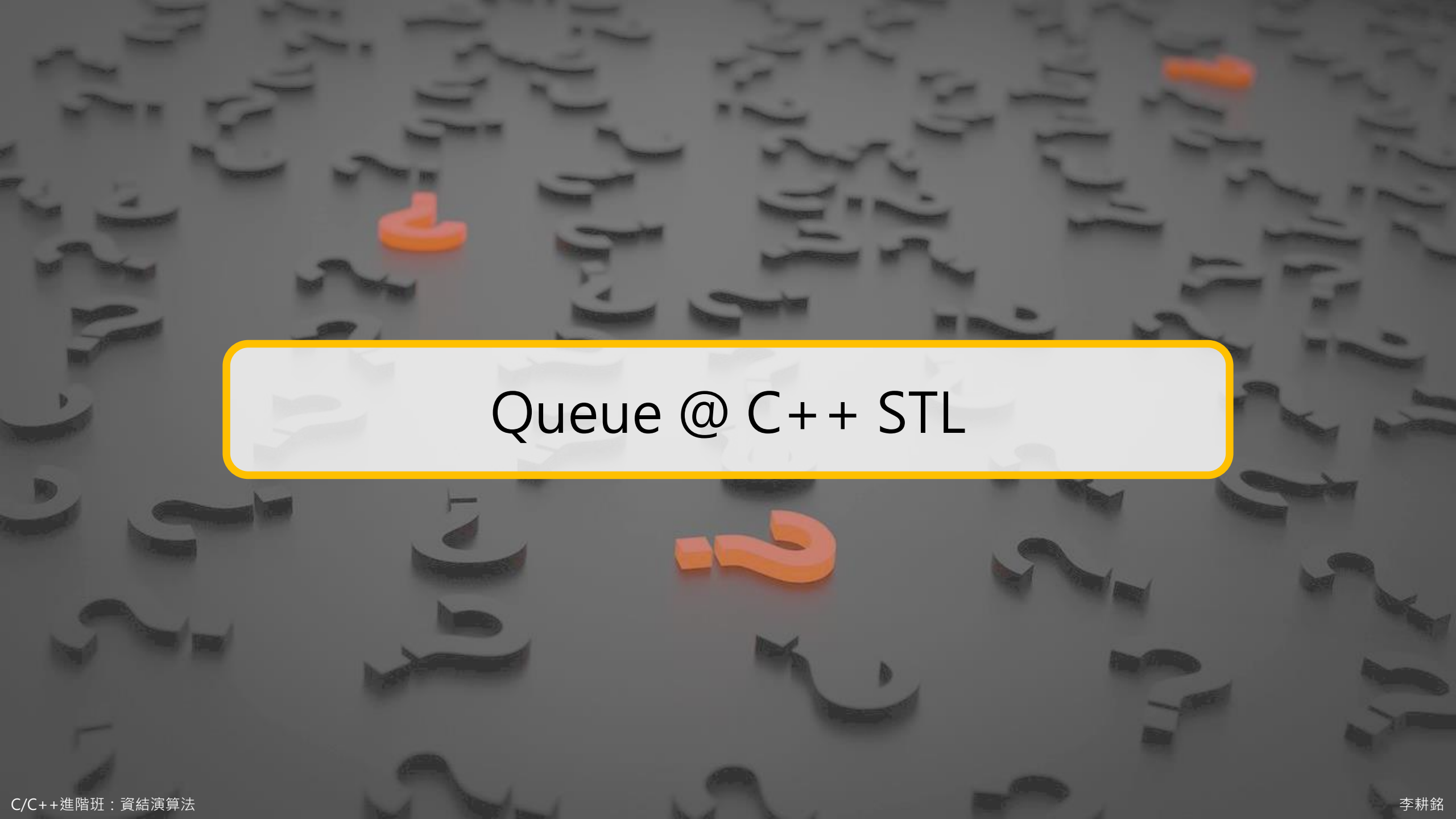
給一疊卡牌，輸入 n 時代表牌有 n 張，編號分別從 1 到 n， 每次操作會依照下面順序：

1. 將目前最上面的卡牌丟掉
2. 再把一張卡牌從最上面放到最下面

重複以上操作，直到剩下最後一張牌，請你輸出這最後一張牌的編號。

```
Please enter N:
5
Queue: 1 2 3 4 5
Now, discard card #1
Now, put the card #2 to the bottom.
Queue: 3 4 5 2
Now, discard card #3
Now, put the card #4 to the bottom.
Queue: 5 2 4
Now, discard card #5
Now, put the card #2 to the bottom.
Queue: 4 2
Now, discard card #4
Now, put the card #2 to the bottom.
Queue: 2
The last, remaining card is:2
```

Ref: https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=1876



Queue @ C++ STL

STL 中的 stack 與 queue

- C++
  - stack 是 stack
  - queue 是 queue
  - priority_queue 是 priority queue
  - deques 是 double-ends queue
    - ✓ 插入、搜尋、刪除：$O(1)$
    - ✓ 但只能在特定位置！
- Python
  - list 治百病！

STL 中的 stack

- stack 與 queue 的使用
  - 引用函式庫

```
#include <stack>
#include <queue>
```

  - 宣告

```
stack<datatype> stack_name;
queue<datatype> queue_name;
```

STL 中的
stack 與 queue 沒有 iterator

stack 與 queue 的操作

- 新增一筆資料

  stack.push(value);
- 刪除一筆資料

  stack.pop();
- 回傳一筆資料

  stack.top();
- 判斷 stack 是否為空

  stack.empty();
- 回傳 stack 長度

  stack.size();

- 新增一筆資料

  queue.push(value);
- 刪除一筆資料

  queue.pop();
- 回傳一筆資料

  刪除端：queue.front();
  新增端： queue.back();
- 判斷 queue 是否為空

  queue.empty();
- 回傳 stack 長度

  queue.size();

queue 的操作

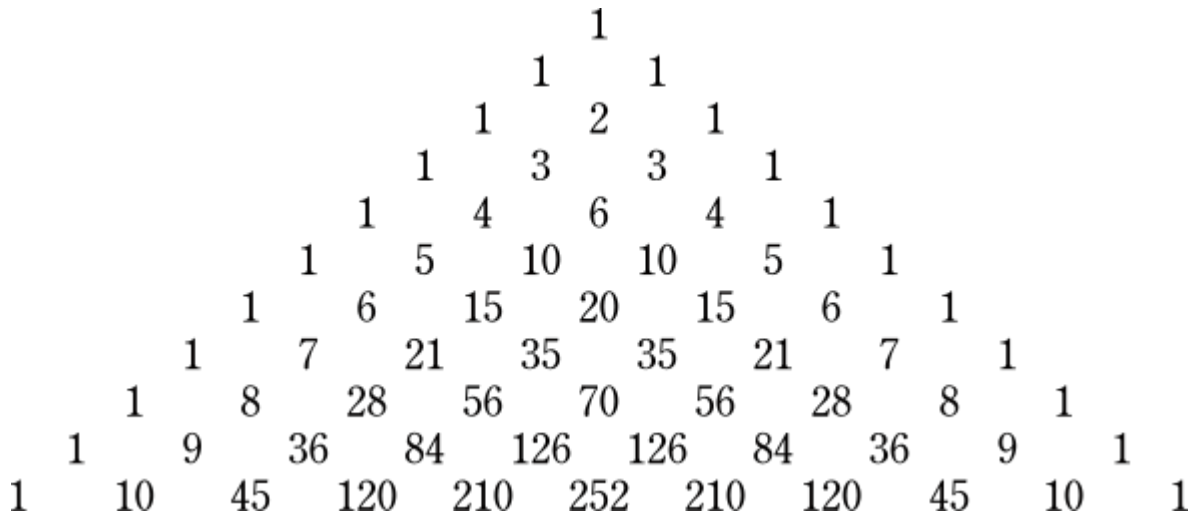
```
#include <iostream>
#include <queue>
using namespace std;
int main()
{
    queue<int> data;
    for(int i=0;i<10;i++)
        data.push(i);
    // 0 1 2 3 4 5 6 7 8 9
    cout << data.front() << endl;
    // 0
    cout << data.back() << endl;
    // 9
    data.pop();
    data.pop();
    // 2 3 4 5 6 7 8 9
    cout << data.front() << endl;
    // 2
    cout << data.back() << endl;
    // 9
    return 0;
}
```

Example Code

Mission

巴斯卡三角是二項式定理中的圖形，以下圖為例，每列數列中的第一個與最後一個皆為一，其餘都是上列數列中前後兩個的和。

讓使用者輸入一整數 N，輸出 N 層巴斯卡三角形。



Example Code

Mission

今天舉辦一個舞會，但舞會中的男生、女生人數不相同，為了公平起見人數多的性別必須輪流與異性跳舞，請用大寫 A、B、C ...表男生，小寫 a、b、c 表女生，並且輸出 N 輪跳舞的配對情形。

- 輸入：跳舞輪數、男生數目、女生數目
- 輸出：每輪的舞伴配對狀況

Example Code

Example：四男三女，共跳舞六輪

男：ABCD

女：abc

```
Please enter rounds of party, number of boy and girls:
6 4 3
There are 6 rounds, 4 boys, and 3 girls.
It's party time!
Rounds #1: A<-->a
Rounds #2: B<-->b
Rounds #3: C<-->c
Rounds #4: D<-->a
Rounds #5: A<-->b
Rounds #6: B<-->c
```

Practice

Mission

LeetCode #232 Implement Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

Implement the MyQueue class:

- *void push(int x) Pushes element x to the back of the queue.*
- *int pop() Removes the element from the front of the queue and returns it.*
- *int peek() Returns the element at the front of the queue.*
- *boolean empty() Returns true if the queue is empty, false otherwise.*

Ref: https://leetcode.com/problems/implement-queue-using-stacks/

Practice

Mission

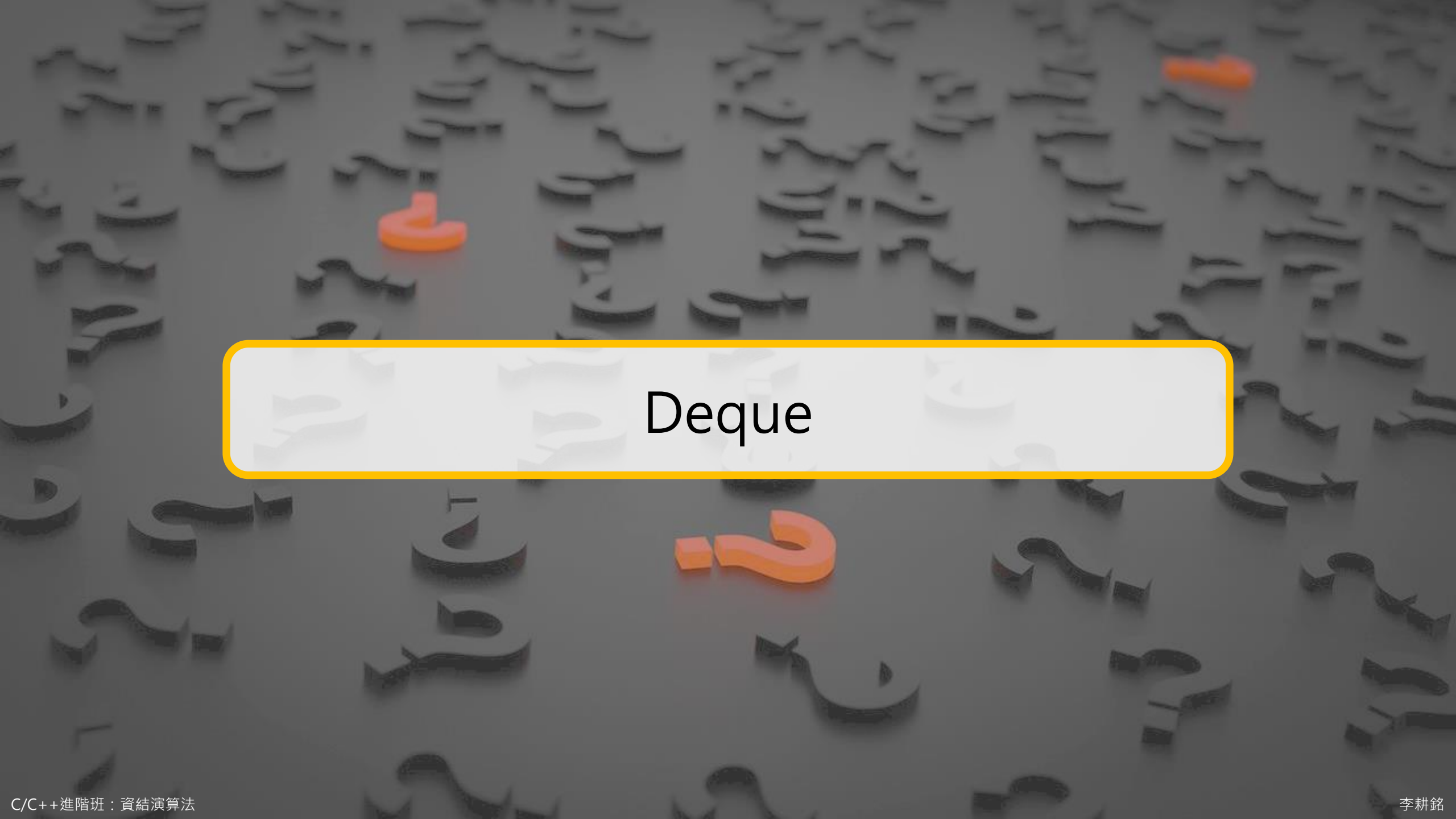
LeetCode #1700. Number of Students Unable to Eat Lunch

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack.

You are given two integer arrays students and sandwiches where sandwiches[i] is the type of the ith sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the jth student in the initial queue (j = 0 is the front of the queue). Return the number of students that are unable to eat.

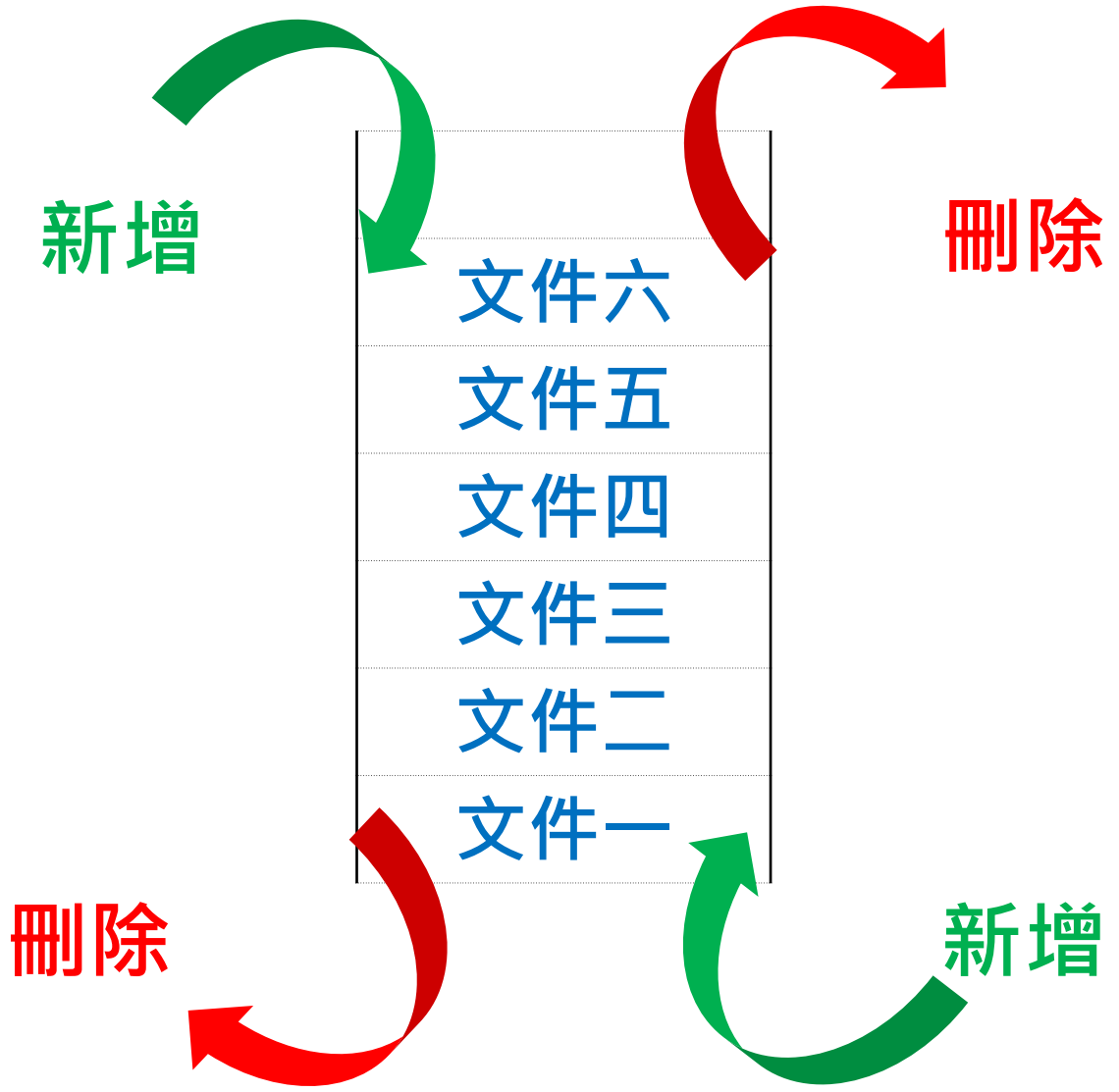
Ref: *https://leetcode.com/problems/number-of-students-unable-to-eat-lunch/*



Deque

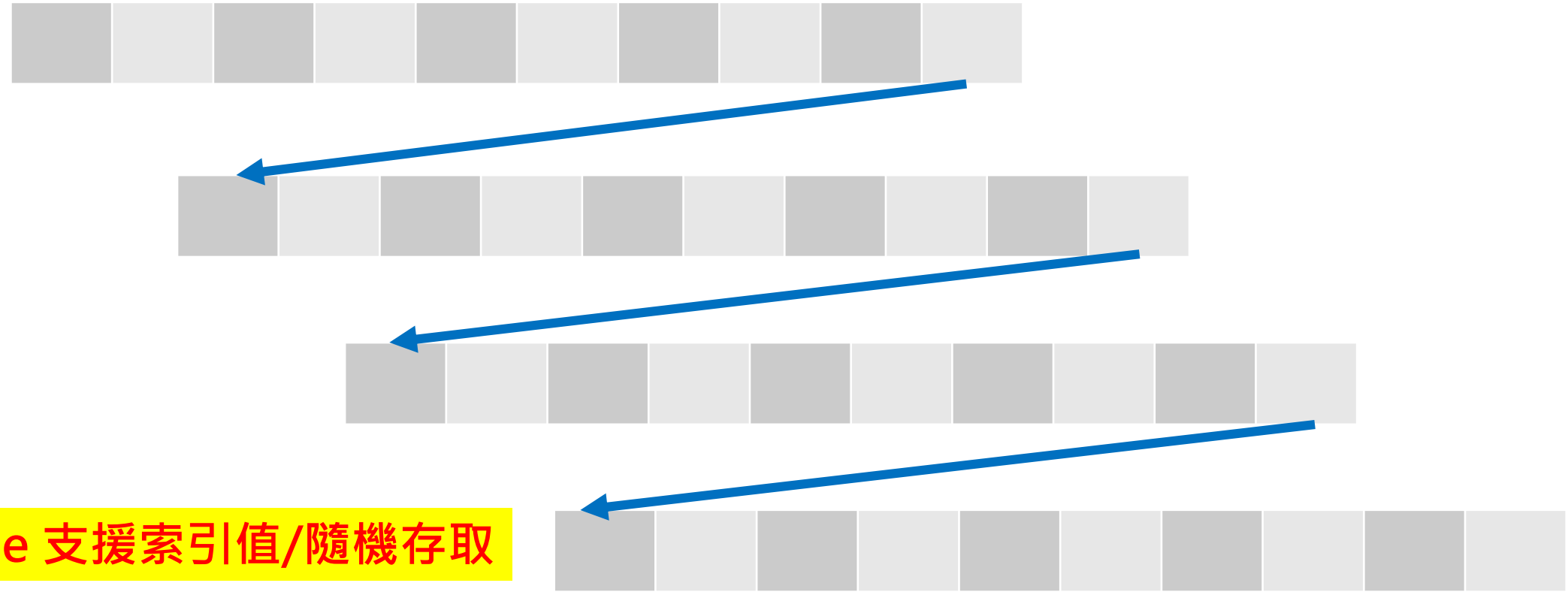
Deque

- Deque
  - double-ends queue
  - 兩端都允許新增或刪除
  - 支援索引值存取與迭代器
  - #include <deque>
  - 利用間接索引完成
    - ✓ map 到許多記憶體空間
    - ✓ 零散的連續記憶體



Deques 與記憶體

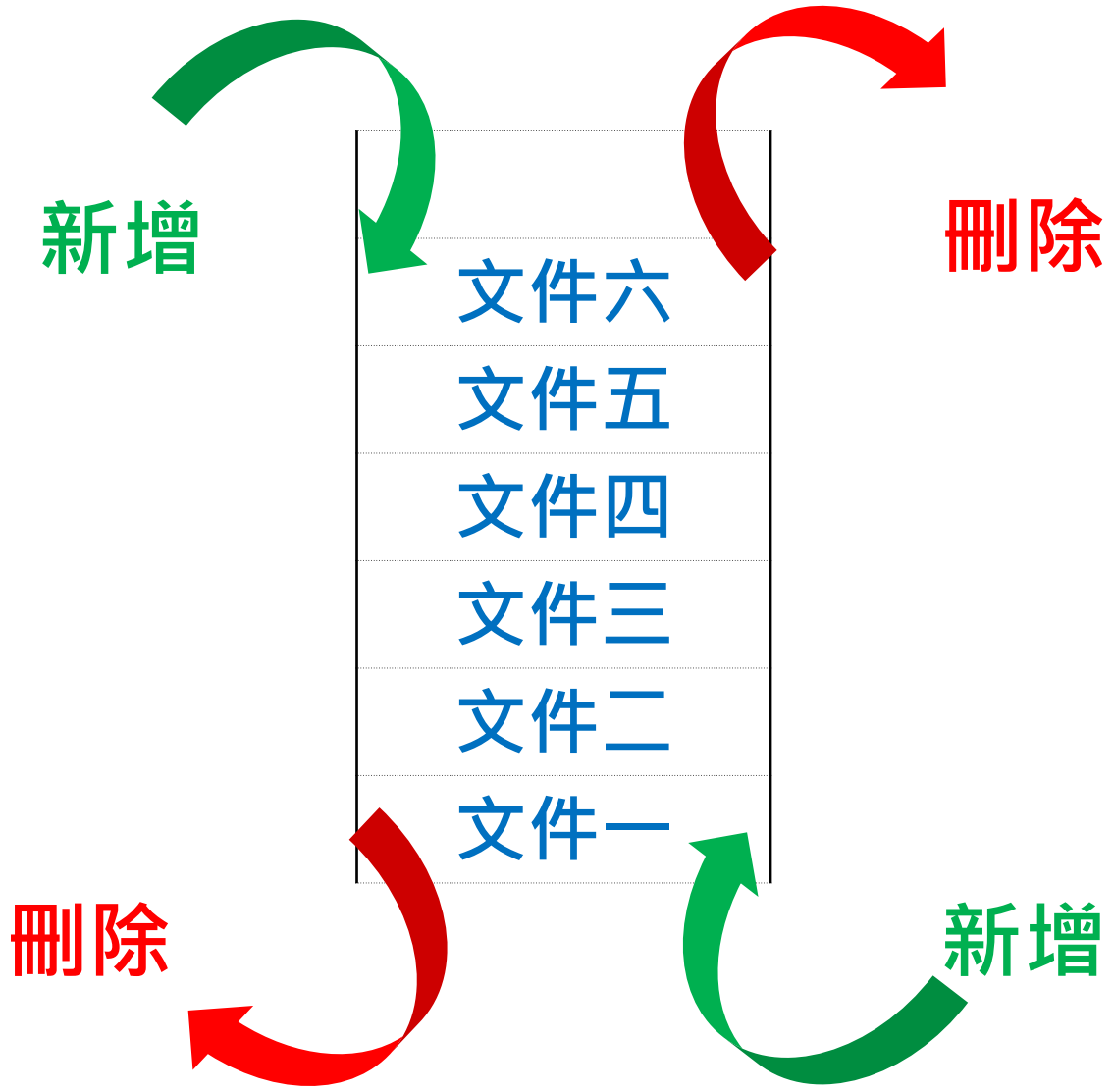
連續的記憶體組成不連續的塊



Deque 支援索引值/隨機存取

Deque

- Deque 常用語法
  - push_back：尾端新增
  - push_front ： 頭端新增
  - pop_back：刪除尾端元素
  - pop_front：刪除頭端元素
  - insert：插入特定元素於特定位置
  - erase：移除某筆資料
  - clear：清空整個 deque



Deques 與 Vector

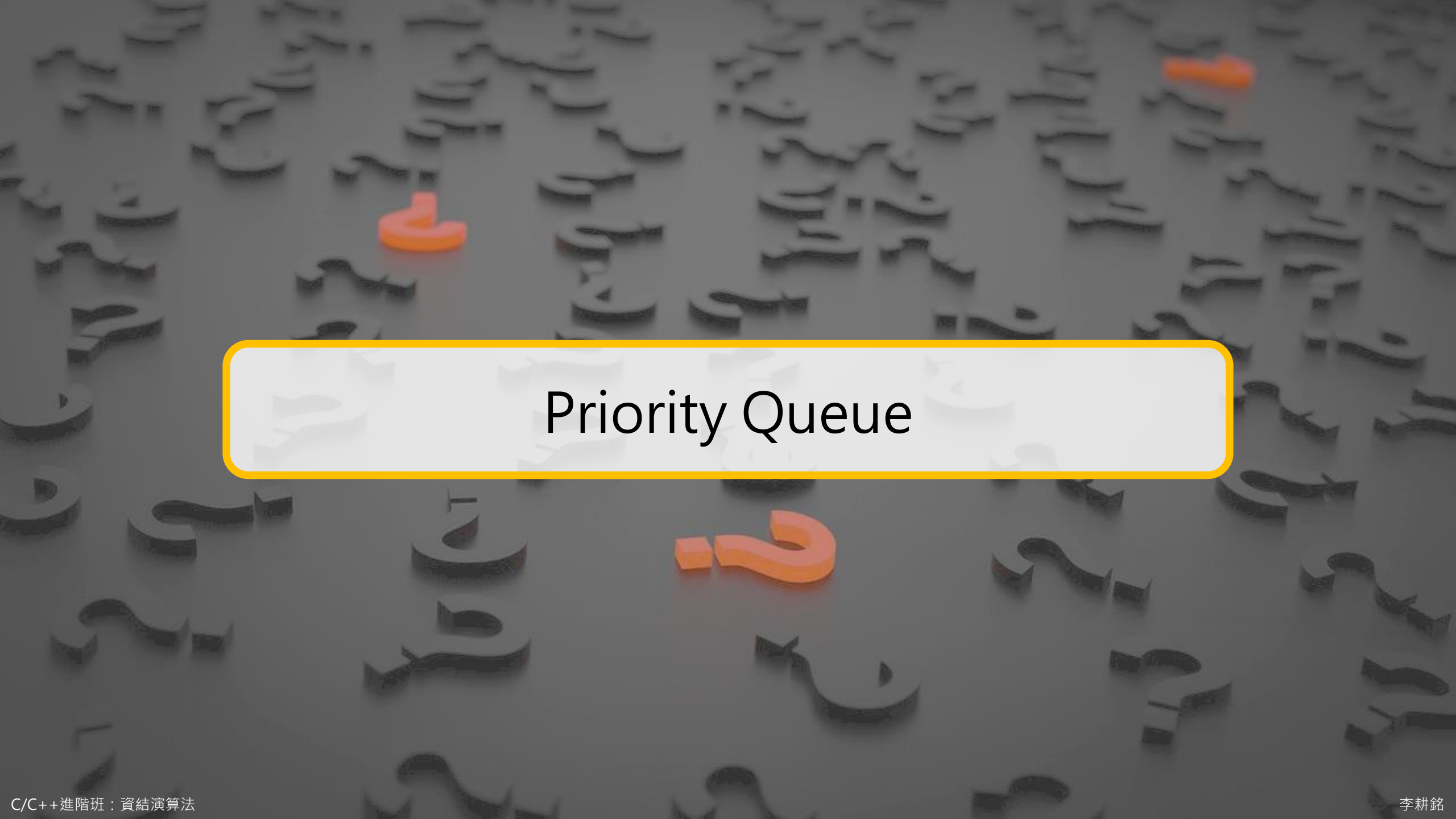
	deque	vector
頭端新增/刪除	O(1)	O(N)
尾端新增/刪除	O(1)	O(1)
中段新增/刪除	O(N)	O(N)
索引值/迭代器存取	支援	支援
記憶體	連續的記憶體組成不連續的塊	連續
大小	彈性	固定
適用情形	兩端新增/刪除 不須取得中段的資料	隨機/索引值存取

Example Code

Mission

試用 deque 完成 Practice 中的：

1. 卡牌分配
2. 括號配對

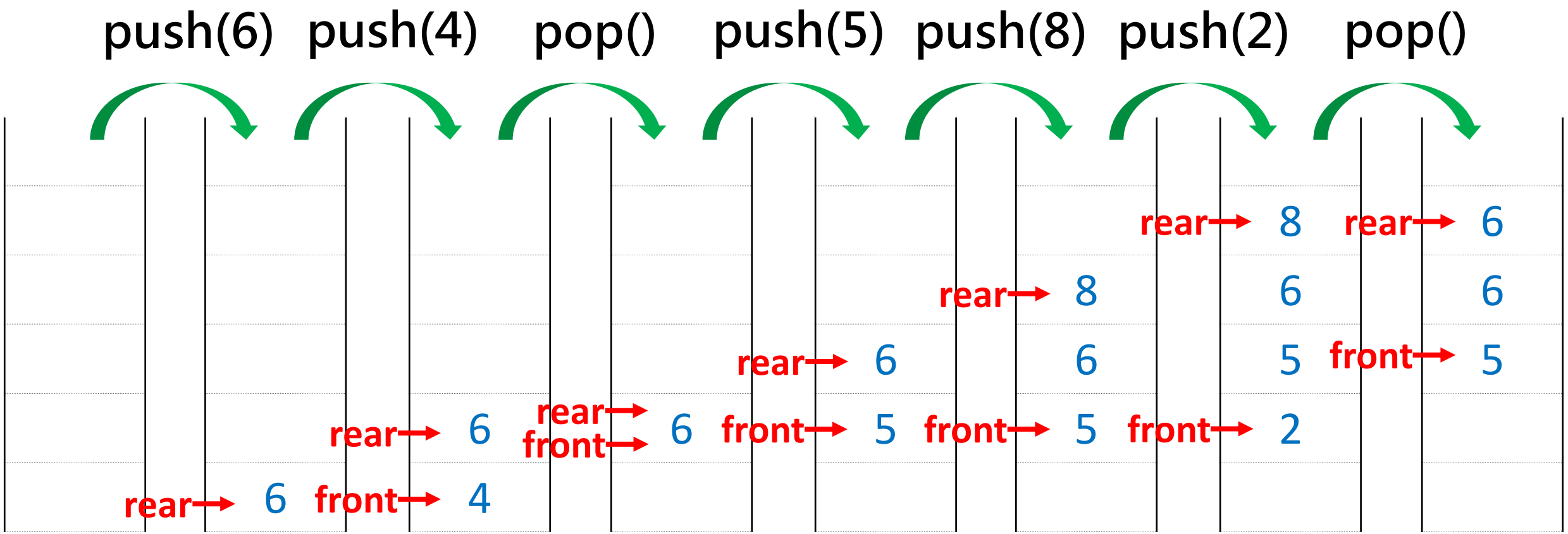


Priority Queue

Priority Queue

- Priority Queue 優先權佇列
  - ➢ Priority
    - ✓ 額外賦予資料權重
  - ➢ Queue
    - ✓ 依照優先權依序排列後再依序輸出
- 輸出次序
  - ➢ 依照資料間的權重大小

Priority Queue



每次插入就加以排序！就可以依照權重大小順序吐出資料

Priority Queue

- 基本操作
  - insert：將資料插入 queue
  - increase key：改變某資料的權重
  - extract max：取得權重最大的資料，並自 queue 中刪除
- 意義
  - 每次插入就加以排序！就可以依照權重大小順序吐出資料
- 讀取資料拿到權重最大的資料
  - Max-Priority Queue
- 讀取資料拿到權重最小的資料
  - Min-Priority Queue

Priority Queue

Priority Queue

- #include <queue>
  - ✓ priority_queue<datatype, container, compared_method>
    - ☐ datatype：要比較的資料型態
    - ☐ container：組成 queue 的容器 (vector 或 deque)
      - 預設為 vector
    - ☐ compared_method：比較方式
  - ✓ priority_queue<datatype>
- 預設權重越大越接近 top

Priority Queue

Priority Queue

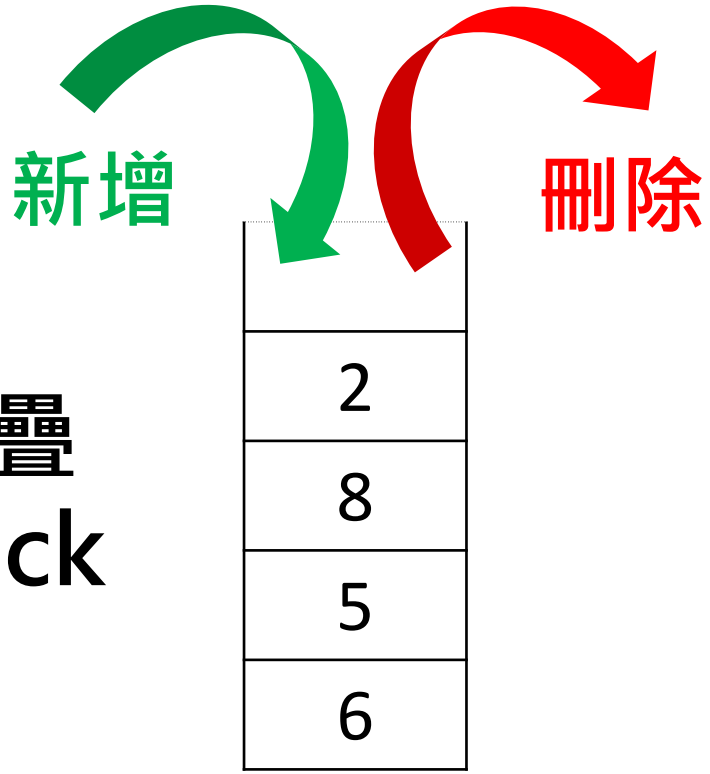
- compared_method
  - ✓ greater<datatype>：由小到大
  - ✓ lesser<datatype>：由大到小
  - ✓ 預設由大到小 (lesser)、運算子<
- 自定義函式
  - ✓ 重載運算子 <
  - ✓ 寫一個結構或類別，內含()運算子重載

Priority Queue

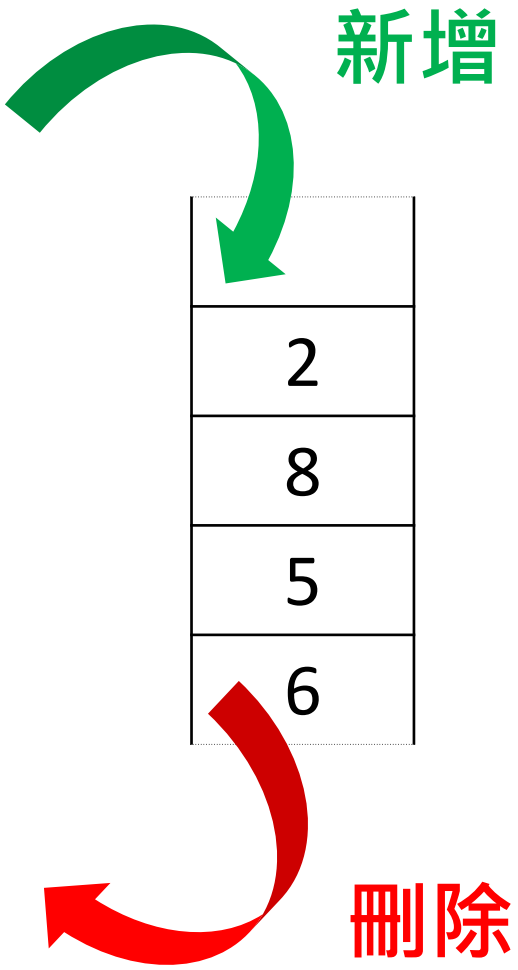
```
priority_queue<int> p_queue;
p_queue.push(8);
p_queue.push(3);
p_queue.push(5);
p_queue.push(7);
p_queue.push(2);
```

p_queue：8 7 5 3 2
p_queue.top()：8
p_queue.pop()：7 5 3 2 (return 8)

stack 與 queue 的比較



堆疊
Stack



佇列
Queue

只有插入/刪除的方向不同
stack 與 queue 的複雜度沒有差異