# 演算法與程式解題實務

Mong-Jen Kao (高孟駿)

Monday 18:30 – 21:20

# C++ STL 系列 – I

使用標準函式庫的 sort 函式

# 前言

- 本系列的目標為介紹 C++ 標準範本函式庫 (Standard Template Library, STL) 裡，重要的泛用資料容器 (Data Container) 與演算法的使用方法.
  - 今天我們將概略介紹 Pair 和 Vector 容器以及 sort 演算法.

# 「泛用」排序演算法的要素

- 顧名思義, 泛用排序演算法可以用來排序「任何型態」的資料
  - 例如： 型態為 int 的陣列、型態為 double 的陣列、
  - n 個平面上的點、 n 個身份證字號、 n 個平面上的向量, etc.
- 使用時, 它必須要知道以下的資訊 (我們必須 直接 / 間接 告訴它)
  1. 指向陣列開頭的指標 (陣列的位置)
  2. 陣列的長度 (陣列有幾個儲存格)
  3. 陣列每個儲存格的大小 (每個儲存格有幾個 byte)
  4. 如何比較 (compare) 兩個儲存格的順序 (任兩個儲存格間的排序順序)

# 使用 sort 函式

- 使用時, 需引入 **<algorithm>**
- 函式的 prototype:

```
std::sort( Iterator first, Iterator last );

std::sort( Iterator first, Iterator last, Compare comp );
```

- 參數說明
  - first 和 last 為指向陣列開頭與結尾的 Iterator

    目前可將它們當成指向(一維)陣列開頭與結尾的指標
  - comp 為比較任意兩筆儲存格資料的 compare function 或 compare object

# 思考

- 作為「泛用」排序演算法,
  sort 如何從它的 prototype 定義中, 取得它排序過程中所需的資訊？

- 函式的 prototype:

```
std::sort( Iterator first, Iterator last );
std::sort( Iterator first, Iterator last, Compare comp );
```

- 參數說明
  - first 和 last 為指向(一維)陣列開頭與結尾的指標
  - comp 為比較任意兩筆儲存格資料的 compare function 或 compare object

# 餵給 sort 函式的 compare function

- 函式的 prototype:

```
bool cmp( type_t &a, type_t &b);
```

- 功能要求:

  傳入兩個指向儲存格的 reference a 與 b, 比較兩個儲存格的順序

  - 若 a 必定在 b 前面, 那麼就回傳 true
  - 若 a 與 b 順序相同、或者在 b 之後,
    那麼就回傳 false

以由小到大排序 int 整數來說,

其實 compare function 問的是 < :

「a 是否 < b」

# 範例: 排序 int 序列

```
int  A[1000], n;
```

- 如上, 假設給定長度為 n 的 int 序列, 儲存在陣列 A 裡.

```
// 先實作 sort 需要的 compare function
bool cmp( int a, int b )
{
    return  a < b;
}
```

**問**： 有 / 無 & 運算子的差別在哪裡？ (重要)

# 範例: 排序 int 序列

```
int  A[1000], n;
```

- 如上, 假設給定長度為 n 的 int 序列, 儲存在陣列 A 裡.

```
// 呼叫 sort 排序, 傳入各個所需的參數
sort( A, A+n, cmp );
```

# 餵給 sort 函式的 compare function

- 注意:
  - compare function 需要依實際儲存格的內容、以及排序規則來實作,

    不同類型的儲存格, 或是不同的排序規則,

    會有不同的 compare function

# 範例: 排序二維平面上的整數點

- 要求: 依照 x 座標由小到大排序, 若 x 座標相同, 則 y 座標小的在前.

```
array<int,2>  P[1000]; // [0]為x座標, [1]為y座標
int n;
```

- 那麼, sort 所需的 compare function 如下:

```
int cmp_2D_pts( array<int,2> &A, array<int,2> &B )
{
      return  A[0] == B[0] ? A[1] < B[1] : A[0] < B[0];
}
```

# 範例: 排序二維平面上的整數點

- 要求: 依照 x 座標由小到大排序, 若 x 座標相同, 則 y 座標小的在前.

```
array<int,2>  P[1000]; // [0]為x座標, [1]為y座標
int n;
```

- 呼叫 sort 進行排序

```
sort( P, P+n, cmp_2D_pts );
```

# 本週程式題導覽

# Problem - A

- 實作 custom compare function,
  使用 C++ 的 sort 演算法排序讀入的資料.
  - 宣告並使用 `struct` 儲存讀入的資料

# Problem - B

- 計算最大的重疊區間數.
  - 想法：

    從左至右, 依序看過所有的區間(Interval), 並算出答案.

  - 如何適當地儲存/排序各個區問？

# Problem - C

- 使用經典的 Graham-Scan 演算法,
  計算給定的平面點集 (2-D point set) 的 Convex Hull (最小凸多邊形).

- 本題目標
  - 理解如何使用向量外積 (`Cross-product`) 判斷兩個向量的相對方向順序 (`Orientation`)
  - 理解如何依逆時針順序排序向量
  - 理解 `Graham-Scan` 如何運作以及如何實作

# 平面向量的外積 (Cross-product of 2D Vectors)

- 令 $\vec{u}$ 和 $\vec{v}$ 為三維空間中，落在 x-y 平面上的兩個向量.

$$\vec{u} = (a_1, b_1, 0), \qquad \vec{v} = (a_2, b_2, 0)$$

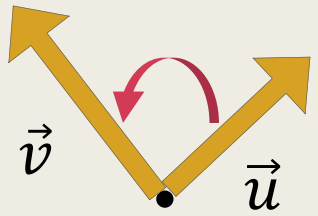
- 根據「右手定則」，
  $\vec{u} \times \vec{v}$ 的結果將會是一個垂直於 x-y 平面的向量，即

$$\vec{u} \times \vec{v} = (0, 0, k), \qquad \text{其中 } k = a_1 b_2 - a_2 b_1.$$

且 $k$ 將滿足以下的條件.

- 令 $\vec{u}$ 和 $\vec{v}$ 為落在 x-y 平面上的兩個向量.

$$\vec{u} = (a_1, b_1, 0), \qquad \vec{v} = (a_2, b_2, 0)$$




$$a_1 b_2 - a_2 b_1 > 0$$

- 根據「右手定則」, $\vec{u} \times \vec{v}$ 的結果為

$$\vec{u} \times \vec{v} = (0, 0, k), \qquad \text{其中 } k = a_1 b_2 - a_2 b_1.$$

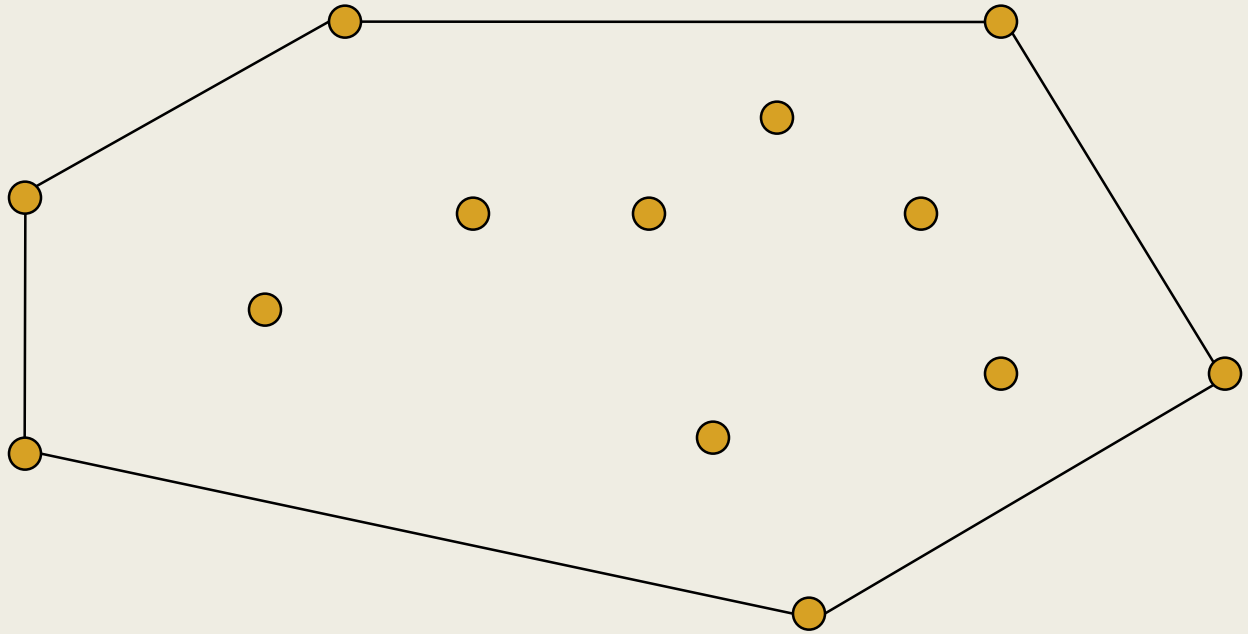
且 $k$ 滿足以下的條件.

- 若 $\vec{u}$ 轉至 $\vec{v}$ 為**逆時針**(Counter-Clockwise), 則 k > 0.
- 若 $\vec{u}$ 轉至 $\vec{v}$ 為**順時針**(Clockwise), 則 k < 0.
- 若 $\vec{u}$ **平行於** $\vec{v}$ (Parallel), 則 k = 0.

# The Convex Hull Problem

- Given a set P of points in the plane,

  the **Convex Hull** of P is the smallest **convex polygon** that contains P.



# The Convex Hull Problem

- Convex Hull 可以在 O( n log n ) 的時間內建出.
  - 有許多方法可以做到這件事.
  - 以下我們將介紹 **Graham Scan** 演算法.
    此演算法的計算過程只使用到排序、以及向量的內外積。

# The Graham Scan Algorithm

1. 首先, 挑出 y-座標最小的點.

   若有多個點的 y-座標一樣小, 那麼就挑出其中 x-座標最小的.

   令這個點為 $p_1$. -- $p_1$ 必定是 convex hull 的頂點之一.

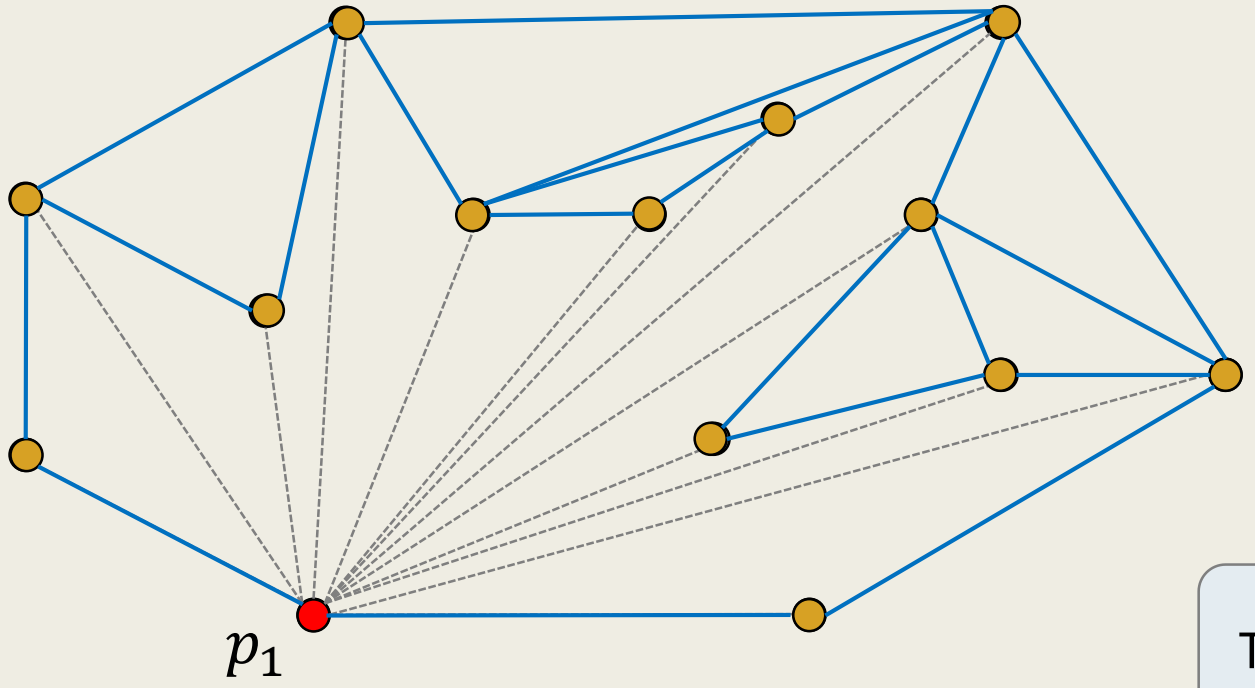
2. 考慮 $p_1$ 到其它所有點所形成的向量.

   以逆時針順序排序這些向量,

   若多個向量平行 (與 x-軸夾角相同), 則依它們的長度排序.

   - 使用向量與自己的**內積**來判斷其長度

- 依排序好的順序, 依序看過這些向量, 來決定 convex hull 的頂點.
  - 性質：**最後三個點必須形成一個 left-turn**.
  - 若不是 left-turn, 則倒數第二個點**必定不是** convex hull 的頂點.




This process takes O(n) time.

The overall complexity is O(n log n).

# The Graham Scan Algorithm

3. 令 $p_1, p_2, \ldots, p_n$ 為排序好的順序, 以及 $L := \{ p_1 \}$ 為起始的 hull vertex.

4. 依序考慮排序好的點 $i = 2,3, \ldots, n$,

   - 將 $p_i$ 加到 $L$ 的尾巴.
   - 當 $|L| \geq 3$ 且 $L$ 裡的最後三個點不形成 left-turn 時, 重覆以下步驟.
     - 刪除 $L$ 裡的倒數第二個點

- 依序輸出 $L$ 裡的點