Stacks and Queues
堆疊和佇列

林劭原老師

Templates in C++

- 為何要有templates?
- 比如上週教的排序法，將輸入n個integers由小到大排序。如果我們現在需要一個排序n個floats的程式，則我們可以將程式中儲存資料的陣列宣告由int改成float，這種做法會寫出另一個sort的程式。
- 然而這個新出爐的sort程式和原本的sort程式，除了有些int被改成float，幾乎是一模一樣，為什麼浪費力氣重複寫幾乎一模一樣的code兩次呢?
- 利用templates，我們可製作template function和template class，這些都是「模板」，「模板」讓我們不必重複寫幾乎一模一樣的code兩次。

pseudo code of selection sort(original)

- function Selection_sort (array, length) {
  for(i from 0 to length-1){
  j=i;
  for(k from i+1 to length-1)
  if (array[k] < array[j]) j=k;
  swap($a[i], a[j]$);
  }
  }

pseudo code of selection sort(using template)

- **template<class T>**
  function Selection_sort (**T* a**, length) {
  for(i from 0 to length-1){
  j=i;
  for(k from i+1 to length-1)
  if (array[k] < array[j]) j=k;
  swap($a[i], a[j]$);
  }
  }
- T* a表示：a是T*型態,i.e., a is a pointer that can point to 儲存型態T資料的空間，即a指向共n個儲存型態T資料的空間其中的a[0]

Templates in C++

- 很輕鬆地，可以sort integers或floats如下：
- float farray[100];
  int intarray[250];
  |
  SelectionSort(farray,100);//會去查farray，farray為float，則將T改為float
  SelectionSort(intarray,250);

Templates in C++

- 我們用template來使得排序integers,floats,rectangles...可以只寫一個程式，省掉重複寫幾乎一模一樣的code的力氣，而且也節省了memory。

Using Templates to represent Container Classes
將模板用在類別上

- 考慮用來儲存data objects的class，叫做**container class**, array就是a container class.我們將用Bag(袋子)這個container class來說明如何用template.

- Bag 建構子constructor，給初值用
- ~Bag 解構子destructor，歸還記憶體用
- Size 傳回Bag的元素個數
- IsEmpty 傳回Bag是不是空的(用處:當Bag是空的，就不給執行Pop)
- Push 亦即insert,不在乎儲存在哪個位置，當然可考慮儲存再the first available 位置
- Pop 亦即delete，移除後，要不要將空位補上?

Bag 的實例

- 我寫了三個版本，兩個是來自課本，一個是我自己寫的。
- Bag(object-oriented).cpp 是課本中只使用物件導向的版本
- Bag(object-oriented&template).cpp是課本中使用物件導和模板的版本
- Bag(function).cpp 是使用函式的版本
- 因為我們沒有教物件導向，模板怕大家看不懂，所以寫了第三個，之後的stack和queue也會用函式版本來寫。有興趣的同學也可以用物件導向和模板來寫。

Stack 堆疊 & Queue 佇列

- 除了array,stack和queue是常用的ordered list，他們的應用實在太多，例如：stack可以用於system stack,maze(迷宮),運算式求值，queue可以用於job scheduling.
- 這就是我們講Bag的原因之一，它們都會用到Bag中的Push,Pop,IsEmpty的概念。

Stack 堆疊

- A **stack** is an ordered list in which insertions and deletions are made at one end called the top.
- 只能從top那端Push或Pop，沒有別種選擇！

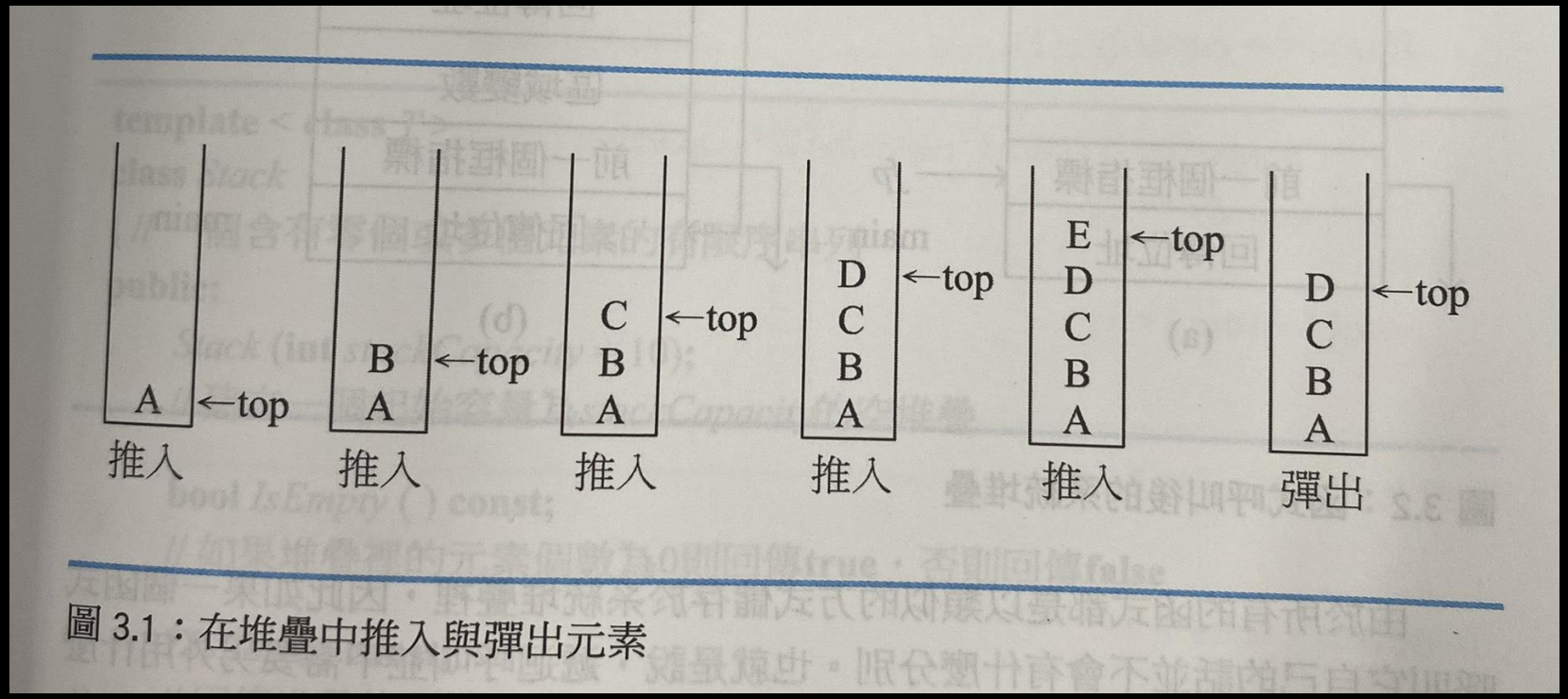

圖 3.1：在堆疊中推入與彈出元素

Stack 堆疊

- 若stack s $= (a_0, a_1, \ldots, a_{n-1})$，則 a_0 叫 the **bottom element**，a_{n-1} 叫 the **top element**.
- A stack is also known as a **Last-In-First-Out(LIFO)** list.(也可寫成FILO)
- 請參考Stack(function).cpp

Queue佇列

- A **queue** is an ordered list in which all insertions are made at one end(called **rear**) and all delections are made at different end(called front).
- 只能從**rear**那端insert東西，只能從**front**那端delete東西，沒有別種選擇！

Queue

必須有 front, 因為 the front element 未必一直是 queue[0], 見下面

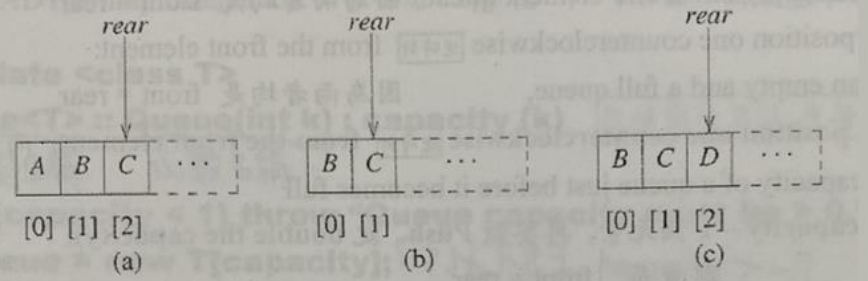

Figure 3.5: Queues represented with front element in *queue* [0]

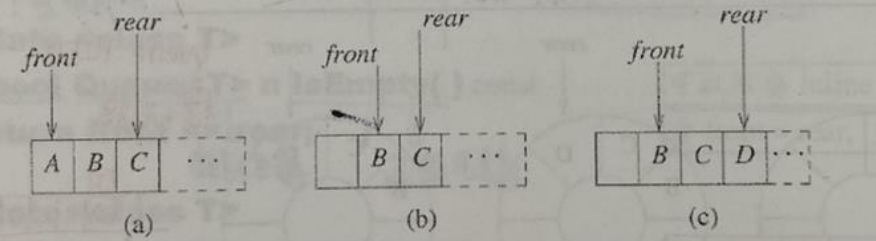

Figure 3.6: Queues represented with front element in *queue* [*front*]

由於 queue 會慢慢向右移, 因此, 當 rear = capacity – 1 時, queue 可能並未填滿, 它前端可能尚有空位, 此時可以考慮

(A) **Shift** the entire queue left. (非常花時間) 或 (B)將 queue 看成是 circular. (比較 efficient)

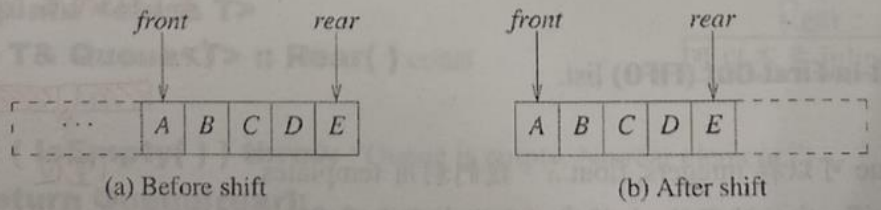

Figure 3.7: Shifting queue elements to the left

Queue佇列

- **Circular queue** 會是一個比較好的解決方法。
- When an array is viewed as a circle, each array position has a next and a previous position.
- The position next to position capacity-1 is 0,and the position that precedes 0 is capacity-1.
- queue[capacity-1]的下一個位置是queue[0]
- queue[0] 的前一個位置是queue[capacity-1]

Queue佇列

- 概念上是circular queue，實作上仍然是儲存在queue[0], queue[1],…queue[capacity-1]
- 下面的code讓circular得以實現.
- if(rear == capacity-1) rear = 0;
  else rear++;

Queue佇列

- 課本有討論front應該放在the front element或position one counterclockwise(逆時針) from the front element.
- 我們假設queue不會滿，所以讓front是position one counterclockwise(逆時針) from the front element.

- Queue又叫做**First-In-First-Out(FIFO)** list.

Qu

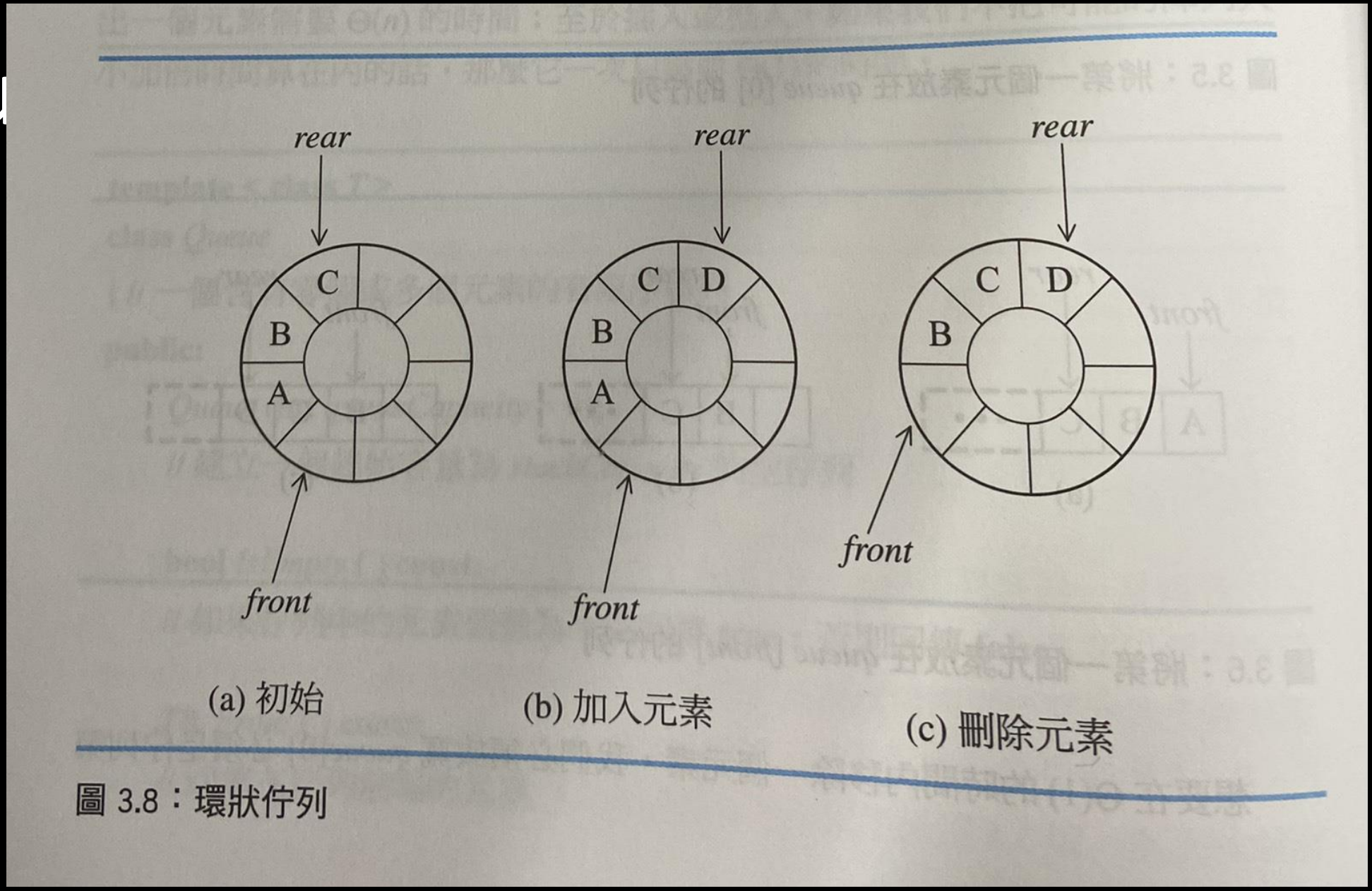
rear
C
B
A
front
(a) 初始
rear
C
D
B
A
front
(b) 加入元素
rear
C
D
B
front
(c) 刪除元素

圖 3.8：環狀佇列

Queue佇列

- 請參考 Queue(function).cpp

A Mazing Problem 迷宮問題

- 用 a 2-dim array maze[m][p] 表迷宮，0表可走，1表不可走，且可以走斜的。入口在 maze[1][1]，出口在 maze[m][p]。程式的目的是要找出一條從入口到出口的路徑。

entrance

0	1	0	0	0	1	1	0	0	0	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	0	0	1	1	1
0	1	1	0	0	0	0	1	1	1	1	0	0	1	1
1	1	0	1	1	1	1	0	1	1	0	1	1	0	0
1	1	0	1	0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	0	1	0	0	1	0	1
0	0	1	1	0	1	1	1	0	1	0	0	1	0	1
0	1	1	1	1	0	0	1	1	1	1	1	1	1	1
0	0	1	1	0	1	1	0	1	1	1	1	1	0	1
1	1	0	0	0	1	1	0	1	1	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0	1	1	1	1	0
0	1	0	0	1	1	1	1	1	0	1	1	1	1	0

exit

A Mazing Problem 迷宮問題

- 程式技巧1：設置圍牆
- 將迷宮外圍包圍一圈1，迷宮變成maze[m+2][p+2]
- 若目前在maze[i][j]，則下一位置有 8 種可能

NW [i-1][j-1]	N [i-1][j]	NE [i-1][j+1]
W [i][j-1]	X [i][j]	E [i][j+1]
SW [i+1][j-1]	S [i+1][j]	SE [i-1][j+1]

- 事實上，有些位置的下一位置並沒有 8 可能！
- 為避免檢查這些border conditions，將迷宮外面包圍一圈1，表示不可走。

A Mazing Problem 迷宮問題

- 程式技巧2：事先將8種可能的走法存好
- struct offsets{
  int a,b;
  }
  //struct 結構可以一次宣告不同資料型態的變數，此時offsets為資料型態
  enum directions{N,NE,E,SE,S,SW,W,NW};
  //相當於寫 const int N=0,NE=1,…,NW=7;
  offsets move[8];
- 以目前在[i][j]，下一位置向SW走到[g][h]來看，程式碼:
- g = i + move[SW].a;
  h = j + move[SW].b;

q	move[q].a	move[q].b
N	-1	0
NE	-1	1
E	0	1
SE	1	1
S	1	0
SW	1	-1
S	0	-1
NW	-1	-1

A Mazing Problem 迷宮問題

- 程式技巧3：用 stack 記錄過程
- 由於有 8 個走法可選，而我們也不知道哪一種走法能到出口，因此用stack記錄過程
- stack中儲存(x,y,dir)
  (x,y)為位置，dir為應嘗試的direction(注意:不是記錄之前的direction)
- struct Items{
  int x,y,dir;
  }

A Mazing Problem 迷宮問題

- 程式技巧4：為避免重複走(回頭)，利用array mark[m+2][p+2]

- 程式技巧5：下一步的走法依順時針方向考慮，N開始,i.e.,dir=0,1,2,...,7,才不至於漏掉solution。

作業2

- 你有兩個作業可以選擇。
- (1)排行程問題
- (2)迷宮問題
- (1)較容易，(2)較困難，兩個都完成的可獲得額外加分。
- 其中作業2(2)有部分程式碼已經完成(請參考Maze.cpp)，可以直接把缺的地方補上，也可以自己重新寫

作業2(1):排行程問題

- 即將退休的林老師每天都有忙不完的工作，因此想在不同情境底下安排每天需要完成的事情。
- (i)情境一：他認為臨時進來的事情應該比較緊急，因此有新的工作時，會優先完成較新的工作，較新的工作完成後才完成較舊的工作，已知他每天早上7點開始工作，且工作輸入有照開始時間排序。(50分)
- 輸入：
  工作項目數量n
  工作1的名稱 工作1的開始時間 工作1的持續時間
  工作2的名稱 工作2的開始時間 工作2的持續時間...
  工作3的名稱 工作n的開始時間 工作n的持續時間
- 輸出：每個時段進行的工作，如果該時段沒有工作，則顯示break

作業2(1):排行程問題

- 即將退休的林老師每天都有忙不完的工作，因此想在不同情境底下安排每天需要完成的事情。
- (ii)情境二：他認為應該尊重優先被要求的工作，因此有新工作時，會優先完成正在進行的工作，才完成較新的工作，如果沒有正在進行的工作，則馬上處理。已知他每天早上7點開始工作 ，且工作輸入有照開始時間排序。(50分)
- 輸入：
  工作項目數量n
  工作1的名稱 工作1的開始時間 工作1的持續時間
  工作2的名稱 工作2的開始時間 工作2的持續時間...
  工作3的名稱 工作n的開始時間 工作n的持續時間
- 輸出：每個時段進行的工作，如果該時段沒有工作，則顯示break

作業2(1):排行程問題

- 例如：情境一
- 輸入：
  3
  breakfast 0700 30
  read 0800 90
  drinkwater 0900 5
- 輸出：
  0700-0730 breakfast
  0730-0800 break
  0800-0900 read
  0900-0905 drinkwater
  0905-0935 read
  0935-2400 break

- 例如：情境二
- 輸入：
  3
  breakfast 0700 30
  read 0800 90
  drinkwater 0900 5
- 輸出：
  0700-0730 breakfast
  0730-0800 break
  0800-0930 read
  0930-0935 drinkwater
  0935-2400 break

作業2(1):排行程問題

- 加分題(20分)：工作輸入沒有依照開始時間排序

作業2(2):迷宮問題(100分)

- 輸入：迷宮

entrance

0	1	0	0	1	0	1	0	1	0	0	0
0	0	0	1	0	1	0	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	1	1	1	0	0	1
1	1	1	1	0	1	1	1	0	1	1	0
1	1	0	0	1	0	1	1	0	1	1	0
1	0	1	1	0	1	0	1	1	0	0	1
0	1	1	0	1	0	1	0	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1
1	0	1	0	1	0	1	0	1	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0

exit

- 輸出：一條從入口到出口的路徑

作業2(2):迷宮問題(加分題20分)

- 輸入：使用者自訂大小與自訂障礙的迷宮
- 輸出：一條從入口到出口的路徑

作業2注意事項

- 1. 因為這次作業比較困難，而且鄰近段考，所以繳交期限為5/19(三) 12:00 以前
- 2. 請將檔案上傳至 共用雲端硬碟\109多元選修_基礎資料結構與演算法\0. 個人作業\作業2_ stack&queue
- 3. 檔名格式為 hw2_班級座號
- 4. 請上傳原始檔(.cpp檔)，不要只上傳執行檔
- 5. 同學之間可以互相討論，也可以來問我，但不要複製貼上，抄襲者以 0 分計