



佇列與堆疊

Chun-Jung Lin



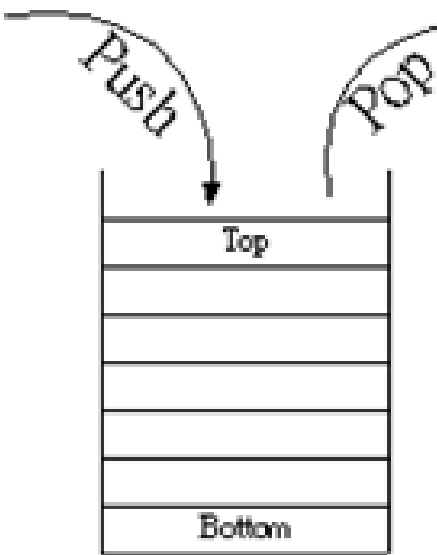
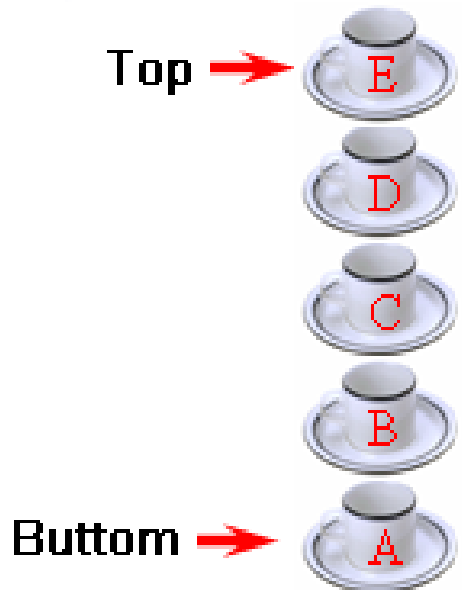
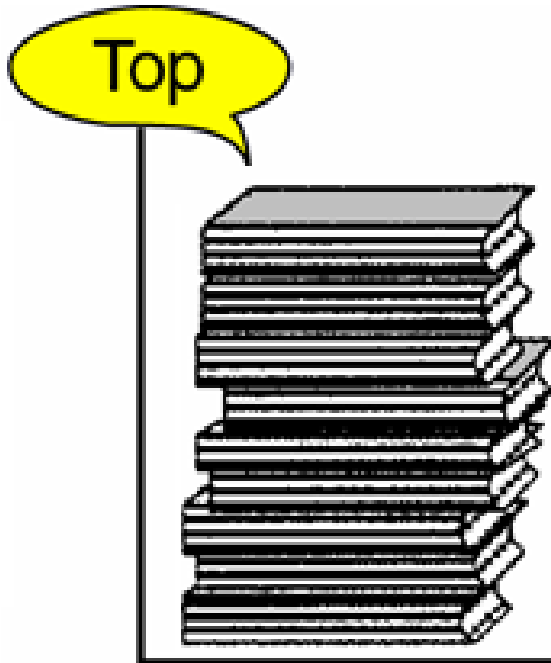
國立勤益科技大學資訊工程系

Department of computer science and information engineering



堆疊和佇列基本觀念 (1/3)

- **堆疊(Stack)**是一有序串列(order list)，其加入(insert)和刪除(delete)動作都在**同一端**，此端通常稱之為頂端(top)。
- 加入一元素於堆疊，此動作稱為推入(push)，與之相反的是從堆疊中刪除一元素；此動作稱為彈出(pop)。
- 由於堆疊具有先進去的元素最後才會被搬出來的特性，所以又稱堆疊是一種後進先出(**Last In First Out, LIFO**)串列。
- 日常生活中堆疊的例子
 - 堆盤子、書本裝箱，都是一層一層的堆上去，如果想要取出箱子中某一本書，也只能從最上面開始取出。

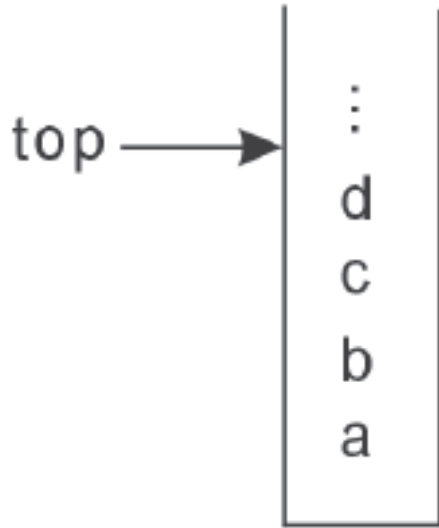
堆疊的基本運作	實例 1：自助餐廳的盤疊	實例 2：書本裝箱
		



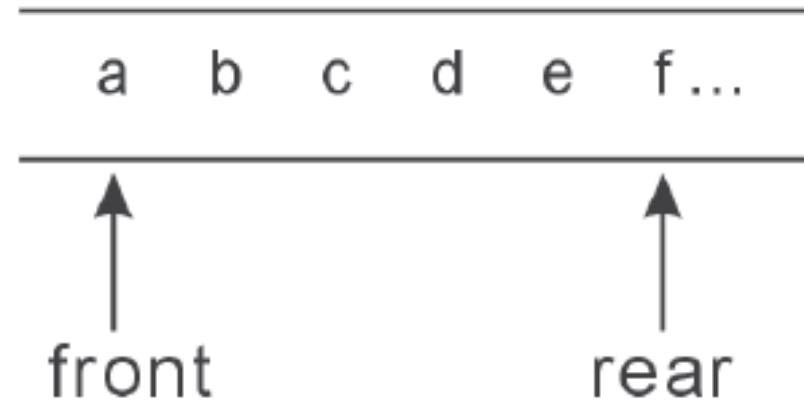
堆疊和佇列基本觀念 (2/3)

- **佇列(Queue)**是屬於線性串列，加入和刪除不在同一端，刪除的那一端為前端(front)，而加入的一端稱為後端(rear)。
- 由於佇列具有先進先出(**First In First Out, FIFO**)的特性，因此也稱佇列為先進先出串列，假若佇列兩端皆可做加入或刪除的動作，則稱之為雙佇列(double-ended Queue, deque)。
- 日常生活中佇列的例子
 - 排隊上公車、排隊買電影票等，皆是先到達的先處理、後到的後處理。

堆疊和佇列基本觀念 (3/3)



(a) 堆疊



(b) 佇列



佇列

- 佇列(Queue)是先進來的元素先出去(First In First Out, FIFO)的資料結構，通常用於讓程式具有排隊功能，依序執行工作。
 - 例如：印表機同時間有多個檔案等待列印，在印表機內會有一個佇列功能，將準備列印的檔案暫存在佇列等待印表機提供列印服務，先送到印表機的檔案先印出來。



環狀佇列

- 佇列目前隨著資料的新增與刪除後，已經儲存過的空間就無法使用，所以產生環狀佇列(Circular Queue)的概念。
- 當環狀佇列到達儲存空間的最後一個位置後，若前方元素已經取出，就可以將資料儲存到第一個位置，繼續往後儲存直到滿了為止。



堆疊(Stack)

- 堆疊(Stack)是後進來的元素先出去(Last In First Out, LIFO)的資料結構，函式的遞迴呼叫使用系統堆疊進行處理，因為遞迴的過程中，最後呼叫的函式要優先處理，系統會實作堆疊程式自動處理遞迴呼叫，不須自行撰寫堆疊。
 - 例如：程式的括弧配對檢查，右大括號配對最接近未使用的左大括號，將左大括號加進堆疊中，一遇到右大括號就從堆疊中取出一個左大括號配對。



後序運算

- 數學運算式為中序運算式，例如：「 $3+2*5-9$ 」是中序運算式，因為運算子(+, -, *, /)介於數字的中間。
- 若轉成後序運算式，則因為先乘除後加減，所以後序運算式為「 $3\ 2\ 5\ *\ +\ 9\ -$ 」，運算子移到數字的後面，將中序運算式轉成後序運算式，就可以使用堆疊進行運算。
- 運算原理如下：
 - 如果遇到數字就加入堆疊，遇到運算子(+, -, *, /)就從堆疊中取出兩個數字進行運算，結果回存堆疊，運算到後序運算式全部執行完成後，會剩下一個數字在堆疊內就是答案。



堆疊的加入與刪除 (1/6)

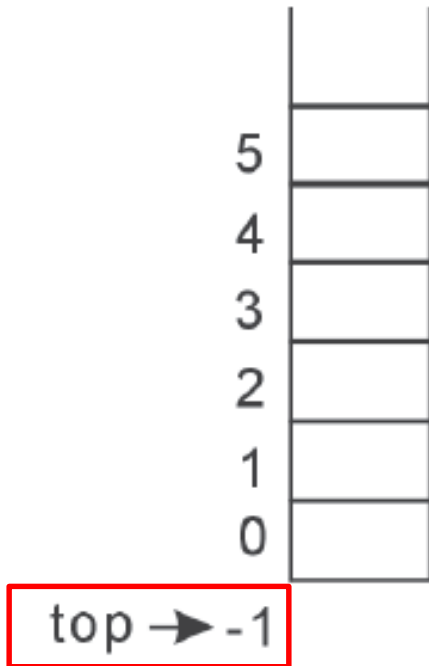
- 設定一陣列來表示堆疊，此堆疊是有最大容量的，如
 - `int a[10];`
- 表示此陣列的最大容量是10個元素。除了有一陣列外，也指定了一個變數`top`做為追蹤目前堆疊的位置。
- `top`的初始值為-1。



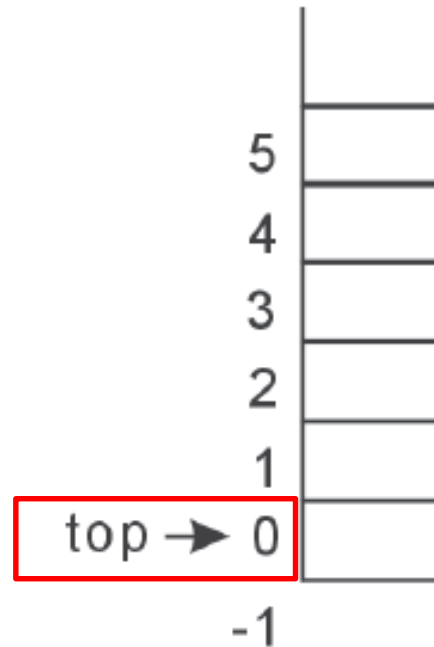
堆疊的加入與刪除 (2/6)

- 加入一元素(**push** an item)到堆疊，主要考慮會不會因為加入此一元素而溢位(**overflow**)，亦即加入時要考慮不可超出堆疊的最大容量。
- 若沒有超出，則**先將top加入**，**再將元素填入a[top]中**。

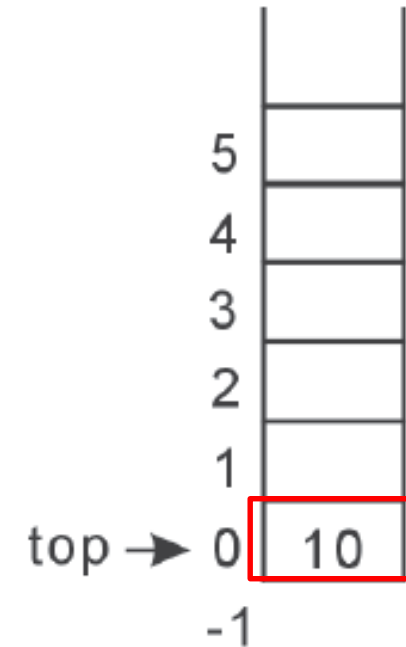
堆疊的加入與刪除 (3/6)



① top 的起始值



② 將 top 加 1



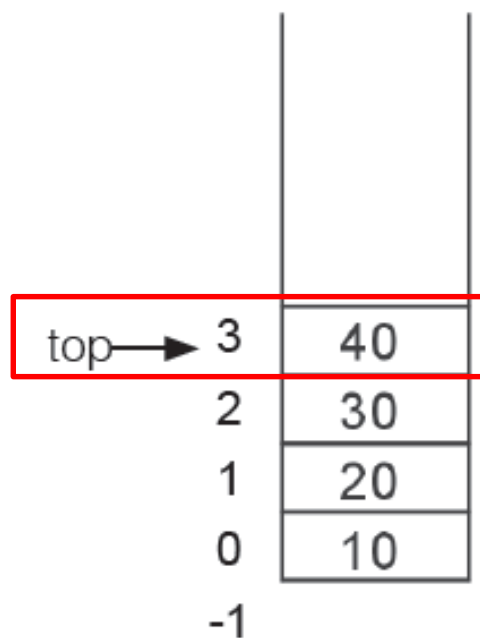
③ 再將元素 (假設 10) 填入



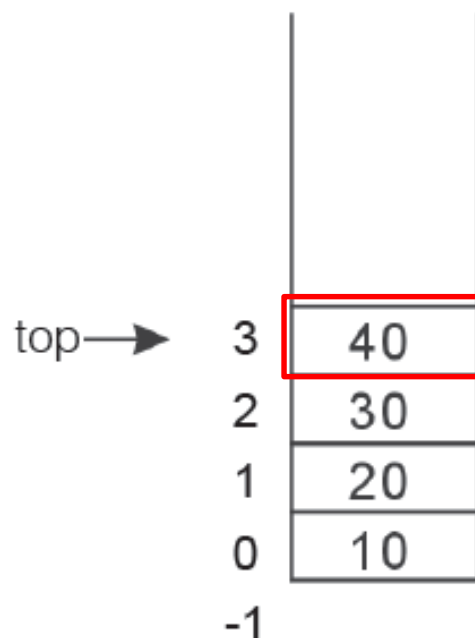
堆疊的加入與刪除 (4/6)

- 從堆疊**刪除**一元素，等於從堆疊彈出(**pop**)一元素，此時我們必須注意堆疊是否為空的，亦即top的值為-1。
- 若不是空的，表示堆疊有元素存在，此時，**先將a[top]的元素移除，之後將top減1**。

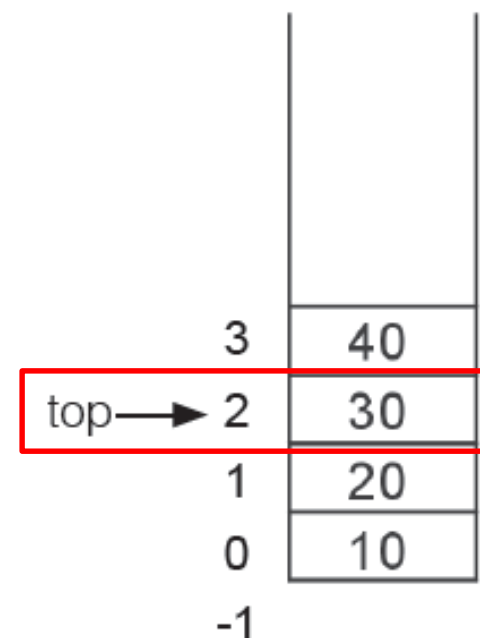
堆疊的加入與刪除 (5/6)



(1) 堆疊的初始狀況 top 值為 3



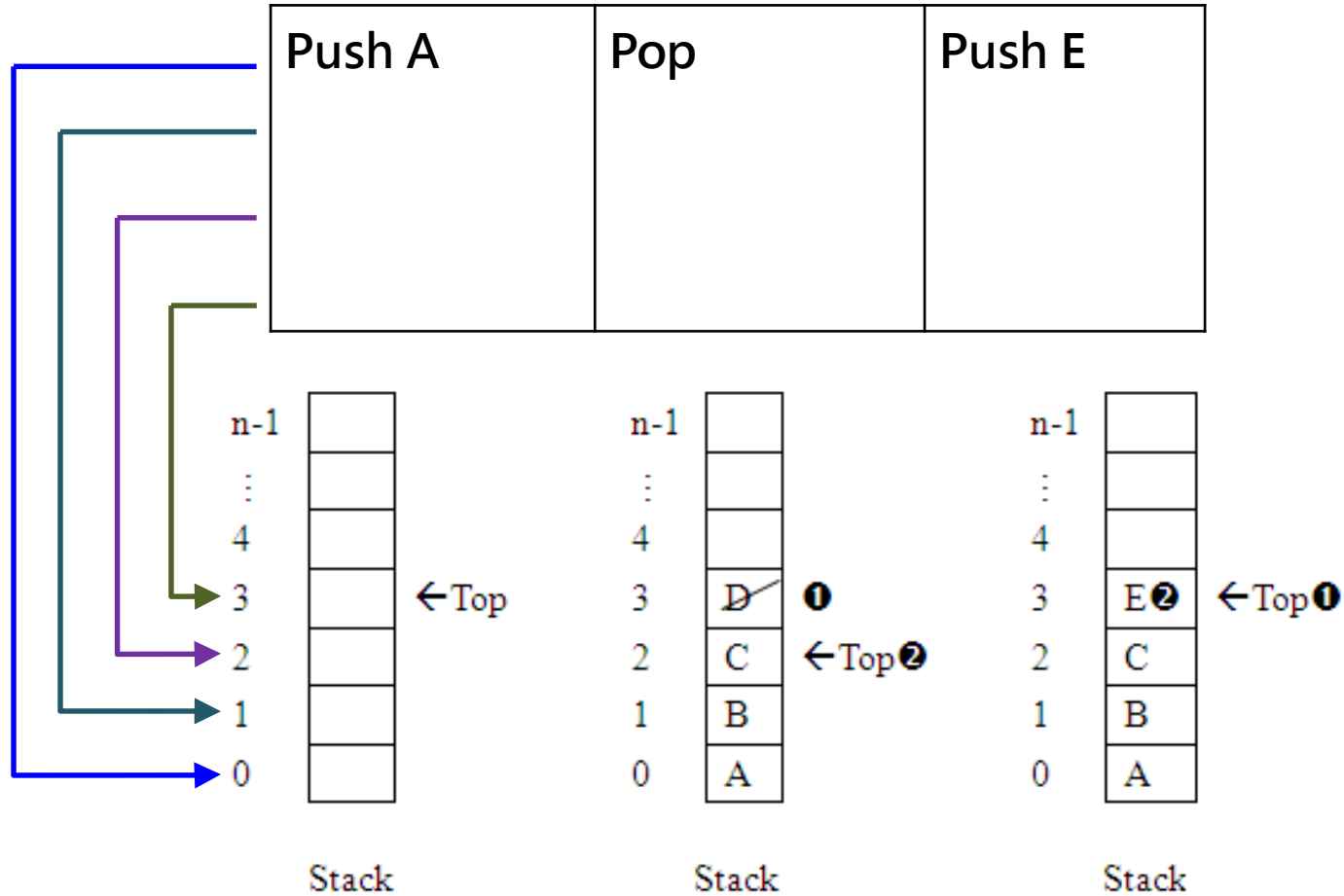
(2) 將 a[3]; 即 40 刪除



(3) 將 top 減 1

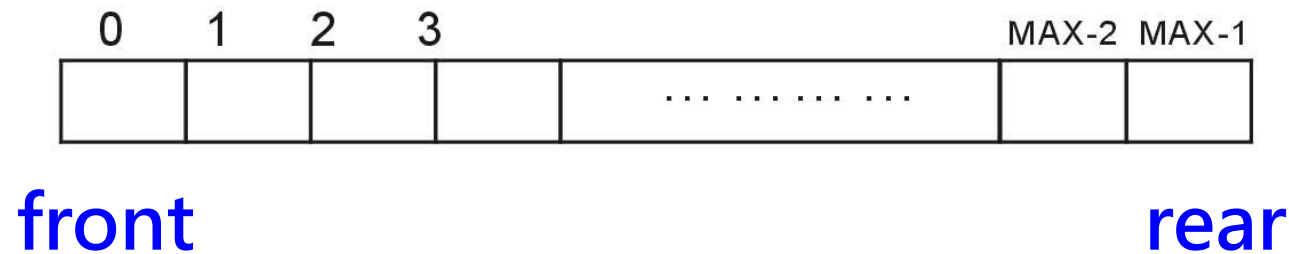


堆疊的加入與刪除 (6/6)



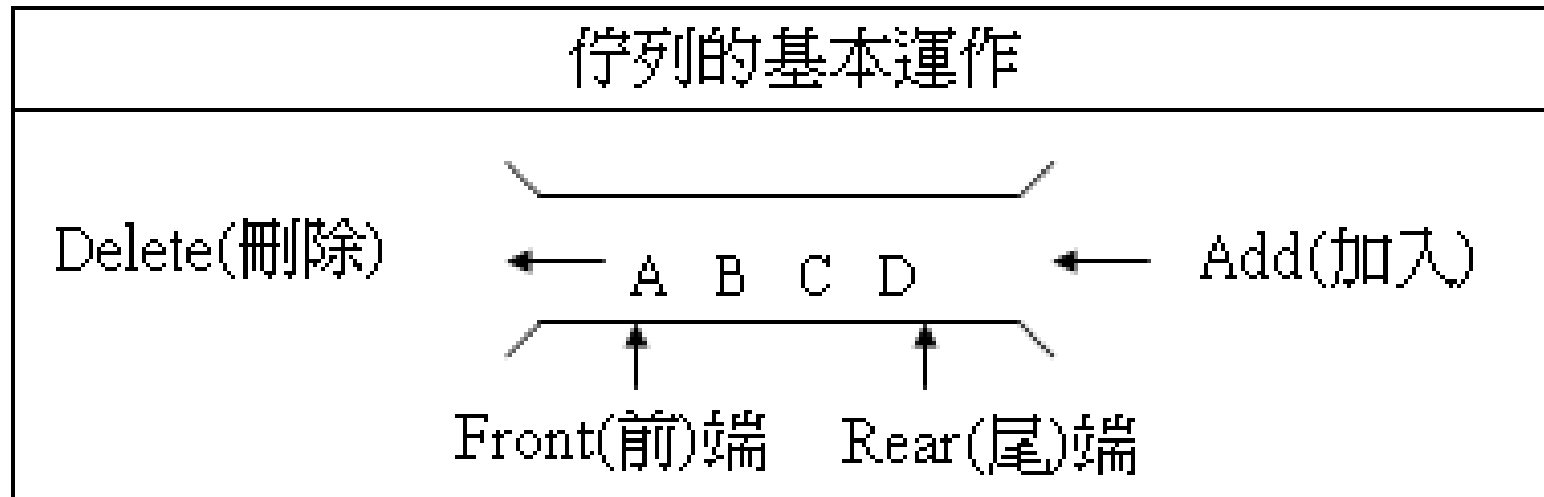
佇列的加入與刪除 (1/8)

- 佇列的操作行為是**先進先出**。
- 假想在一陣列中有二端分別為front和rear端，每次**加入**都加在**rear**端，而**刪除**(即將被服務)的在**front**端。
- 一開始，佇列的front=-1，rear=-1，當加入一元素到佇列時，主要判斷rear是否會超過其陣列的最大容量。





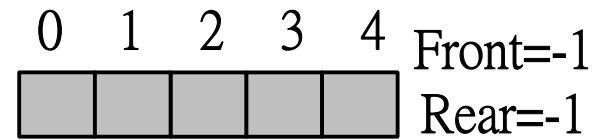
佇列的加入與刪除 (2/8)





佇列的加入與刪除 (3/8)

- 當rear為MAX-1時，表示陣列已到達最大容量了，此時不能再加任何元素進來；反之，則陣列未達最大容量，因此可以加入任何元素。



A , N=1

AB , N=2

ABC , N=3

ABCD , N=4

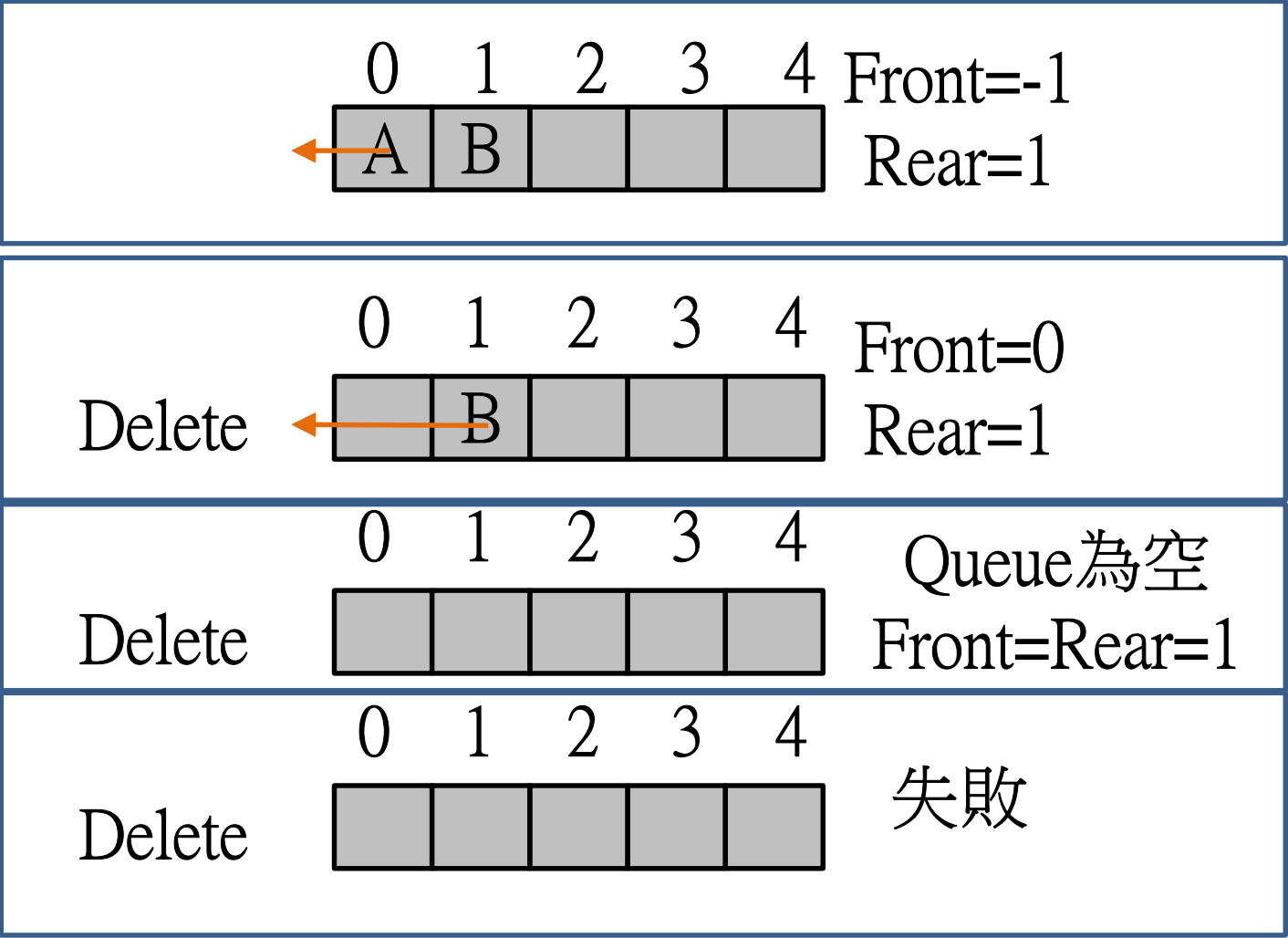
ABCDE , N=5

ABCDE , N=5



佇列的加入與刪除 (4/8)

- 刪除佇列的元素則需考慮佇列是空的情況，因為佇列為空時，表示沒有元素在佇列中。
- 當 $\text{front} \geq \text{rear}$ 時，則表示佇列是空的。





佇列的加入與刪除 (5/8)

```
■ void Queue::enqueue_f(void)
■ {
■     if (rear >= MAX-1)
■         cout << "佇列已滿 \n");
■     else {
■         rear++;
■         cout << "請輸入一個物件到佇列: ";
■         cin >> a[rear];
■     }
■ }
```



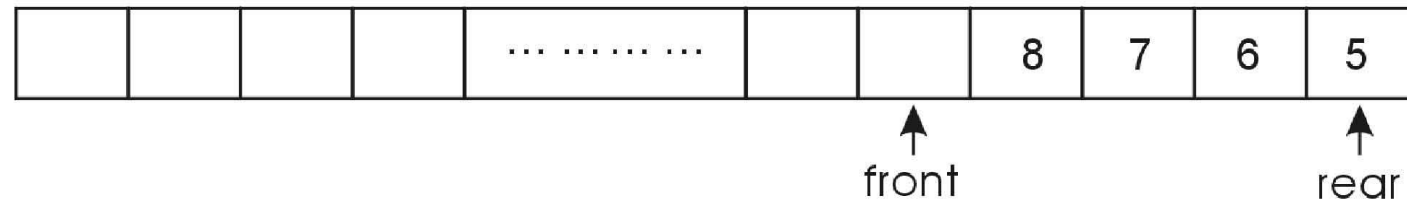
佇列的加入與刪除 (6/8)

```
■ void Queue::dequeue_f(void)
■ {
■     if (front == rear)
■         cout << "佇列是空的 \n";
■     else {
■         front++;
■         cout << "從佇列刪除 " << a[front] " \n";
■     }
■ }
```



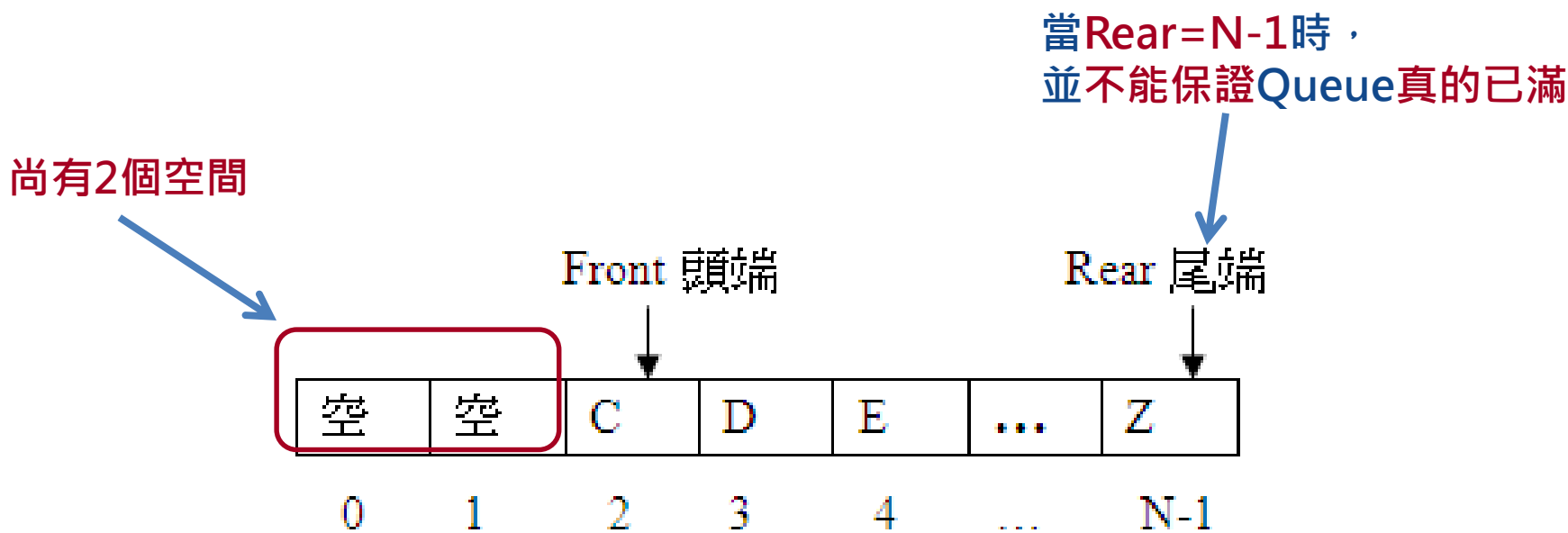
佇列的加入與刪除 (7/8)

- 上述的佇列是線性佇列(linear queue)，表示的方式為Q(0:MAX-1)。
- 此線性佇列不太合理的現象就是當rear到達MAX-1時，即無法加入，因為上述的加入片段程式會印出佇列是滿的訊息，因此它沒考慮佇列的前面是否還有空的位子。





佇列的加入與刪除 (8/8)





Discussion

■ 您覺得Queue不能使用前面的空位子合理嗎？為什麼？



Exercise

- 使用 Queue 表示門診叫號流程。
 - 門診最多可以有5個人掛號看診。
 - 醫師每次叫目前排在最前面的病人看診，若掛號人數已滿，則下回請早。
 - 請隨機執行掛號與看診，例如：一開始有3人掛號，醫生看診2人後，又有2人掛號，一直到掛滿號和醫生看完診後結束。
 - 註：不考慮掛號未到診與取消掛號等情況。



環狀佇列 (1/9)

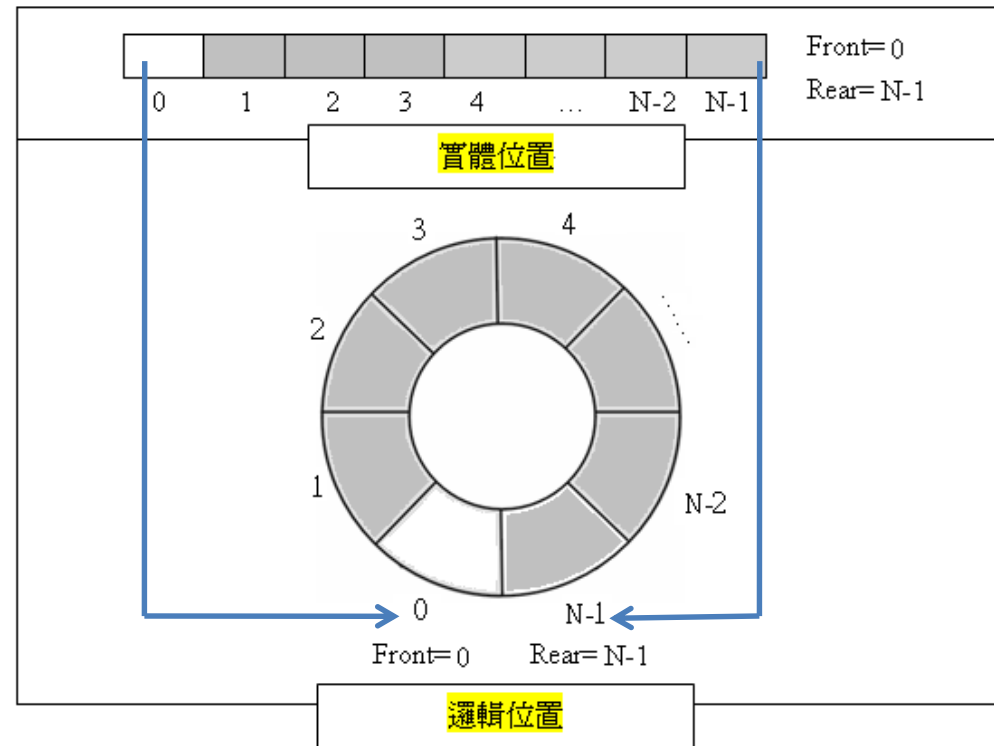
■ 定義

- 環狀佇列(**circular queue**)就是一種環形結構的佇列，它是利用一種 $Q[0:N-1]$ 的一維陣列，同時 $Q[0]$ 為 $Q[N-1]$ 的下一個元素。
- 最多使用 $(N-1)$ 個空間。
- 指標**Front**指向佇列前端元素的**前一個位置**。
- 指標**Rear**指向佇列尾端的元素。

■ 缺點

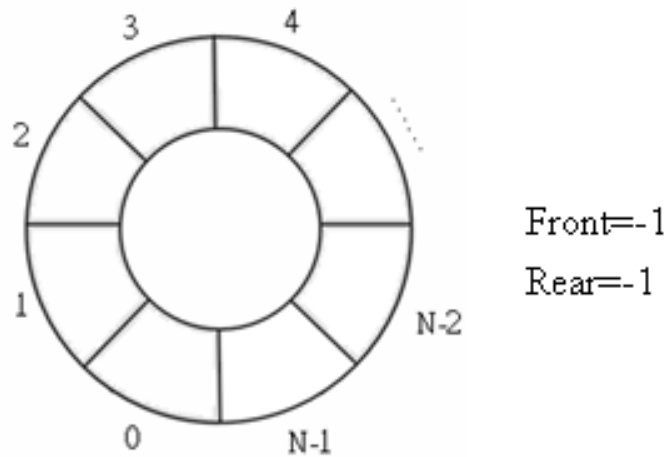
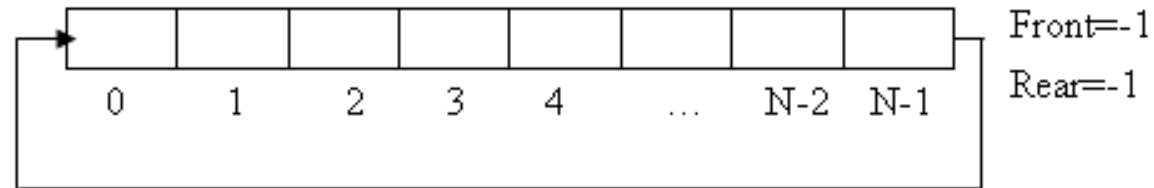
- 佇列滿了，**浪費一個空格**沒有使用。

環狀佇列 (2/9)



環狀佇列 (3/9)

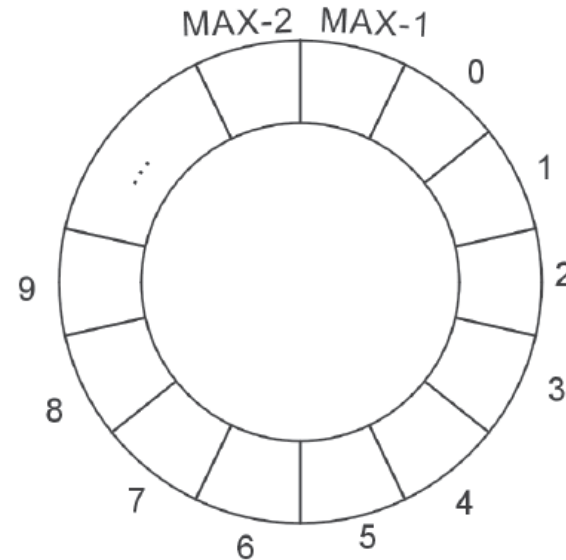
■ 產生環狀佇列結構：Create(CQueue)



環狀佇列 (4/9)

■ 環狀佇列的初始值為 $\text{front}=\text{rear}=\text{MAX}-1$ ，當有元素欲加入時，利用下一敘述

■ $\text{rear}=(\text{rear}+1) \% \text{MAX};$





環狀佇列 (5/9)

- 求出rear，主要的用意在於能夠使rear回到前面，看看是否還有空的位置可放。
- 如當rear為MAX-1時， $((MAX-1)+1) \% MAX$ ，其餘數為0，此時便可進入環狀佇列的前端了。

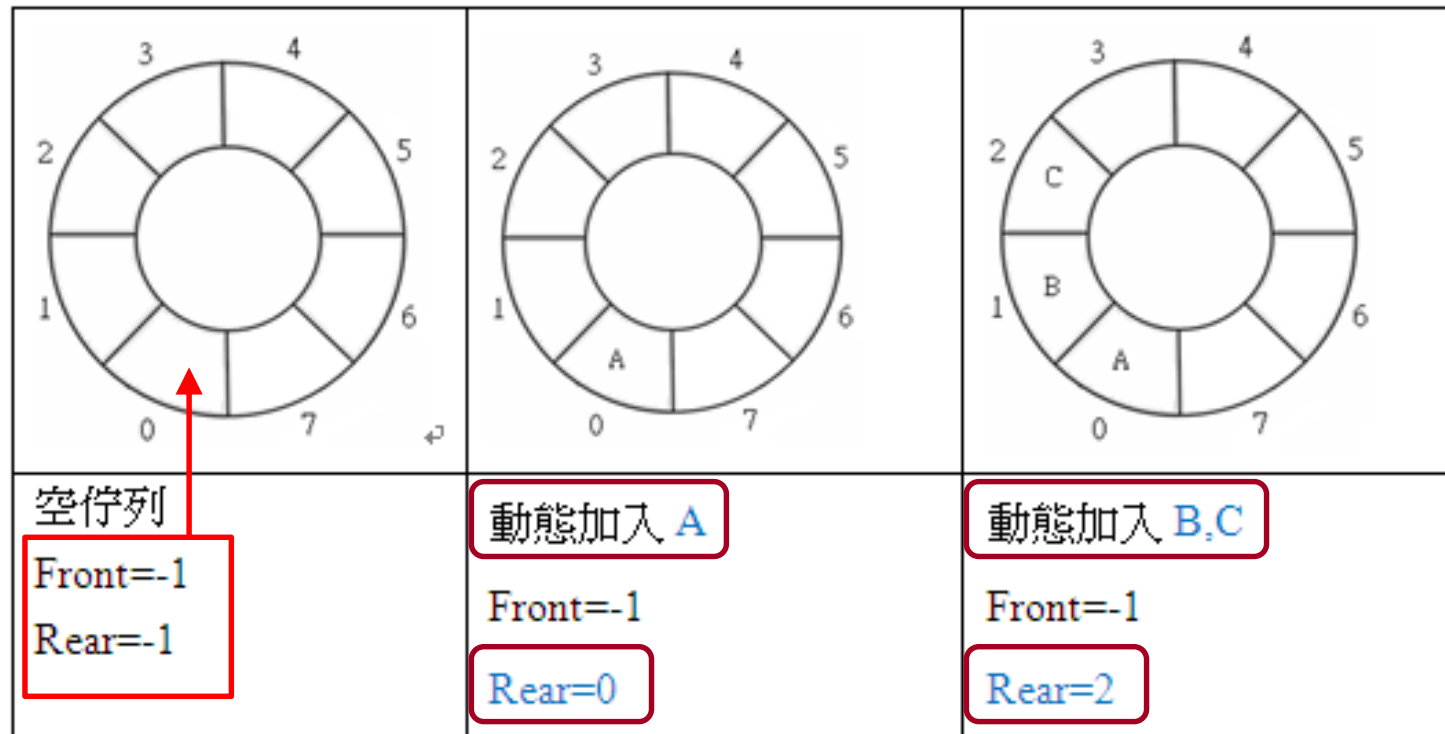


環狀佇列 (6/9)

- 當 rear 為 MAX-2，front 為 MAX-1，此時若加入一元素，會發生“滿”的訊息，主要的用意在於能確保刪除時是正確的。

環狀佇列 (7/9)

■ 將資料放入環狀佇列：Add(item, CQueue)



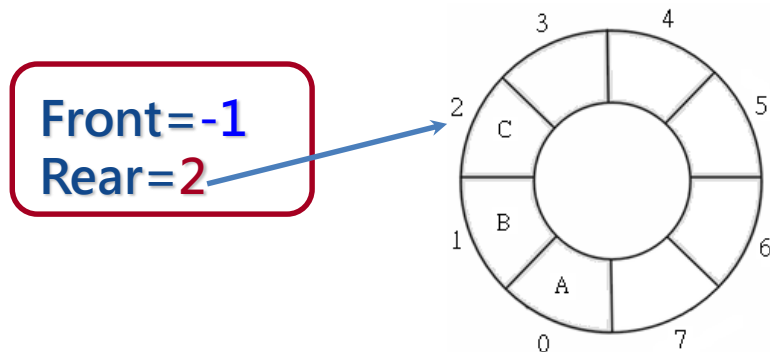
環狀佇列 (8/9)

■ 從環狀佇列刪除資料：Delete(item,Cqueue)

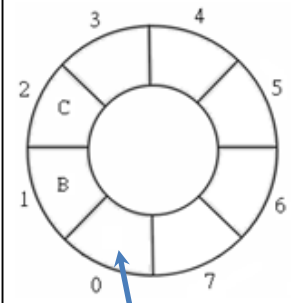
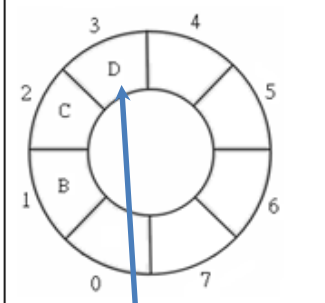
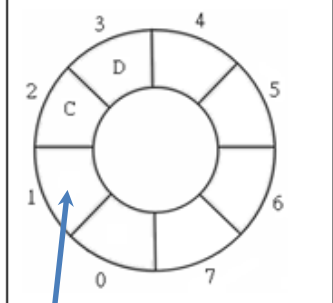
運作前

加入公式： $\text{Rear} = (\text{Rear} + 1) \text{ Mod } N$

刪除公式： $\text{Front} = (\text{Front} + 1) \text{ Mod } N$



運作過程

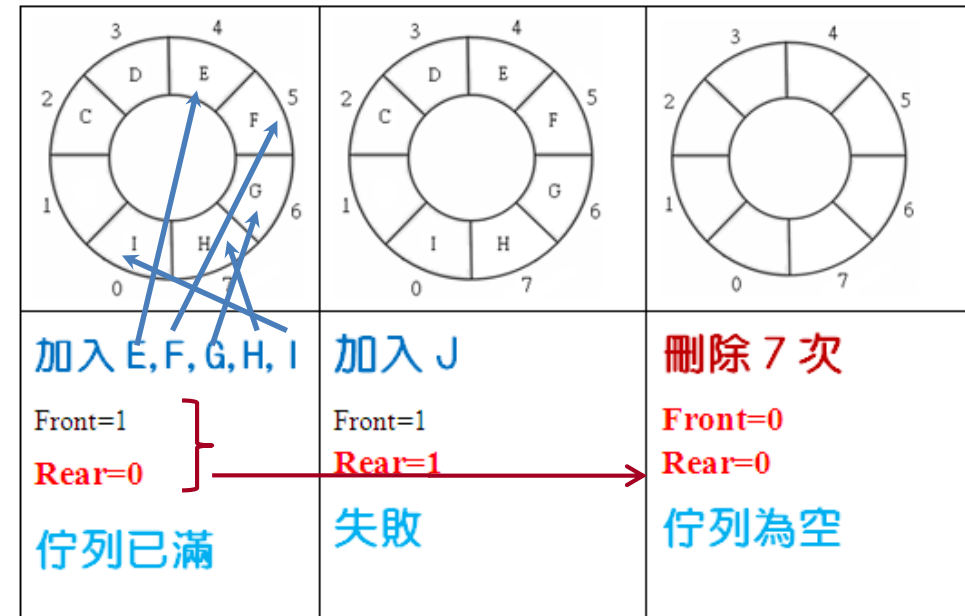
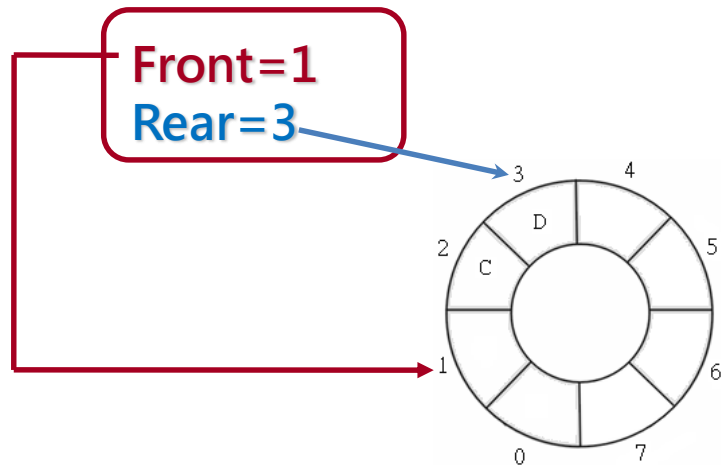
		
<p>刪除 Front=0 Rear=2</p>	<p>加入 D Front=0 Rear=3</p>	<p>刪除 Front=1 Rear=3</p>

環狀佇列 (9/9)

運作過程：

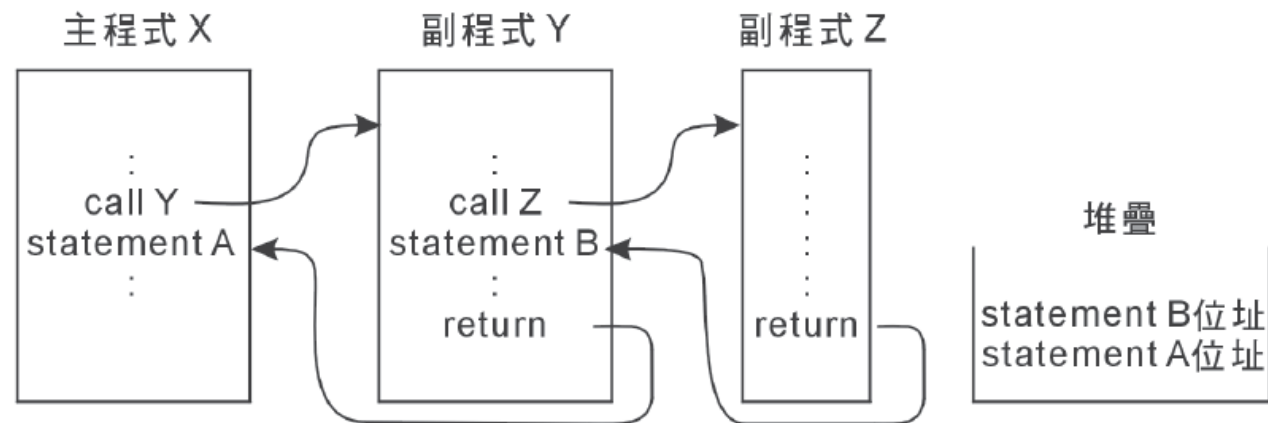
加入公式： $\text{Rear} = (\text{Rear} + 1) \text{ Mod } N$

刪除公式： $\text{Front} = (\text{Front} + 1) \text{ Mod } N$



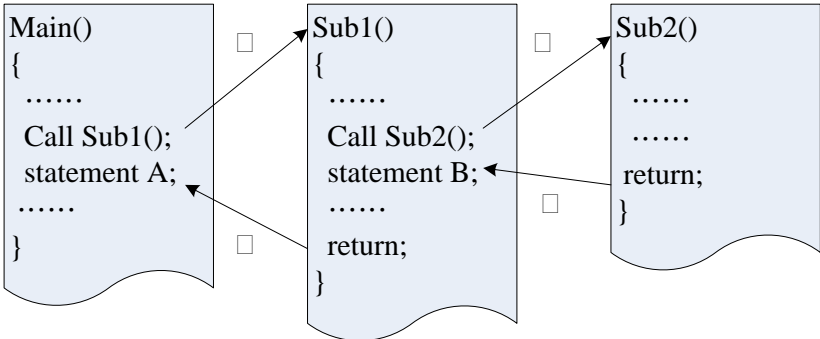
堆疊與佇列的應用 (1/13)

- 由於堆疊具有先進後出的特性，因此凡是具有後來先處理的性質，皆可使用堆疊來解決。
- 例如副程式的呼叫(subroutine calls)。

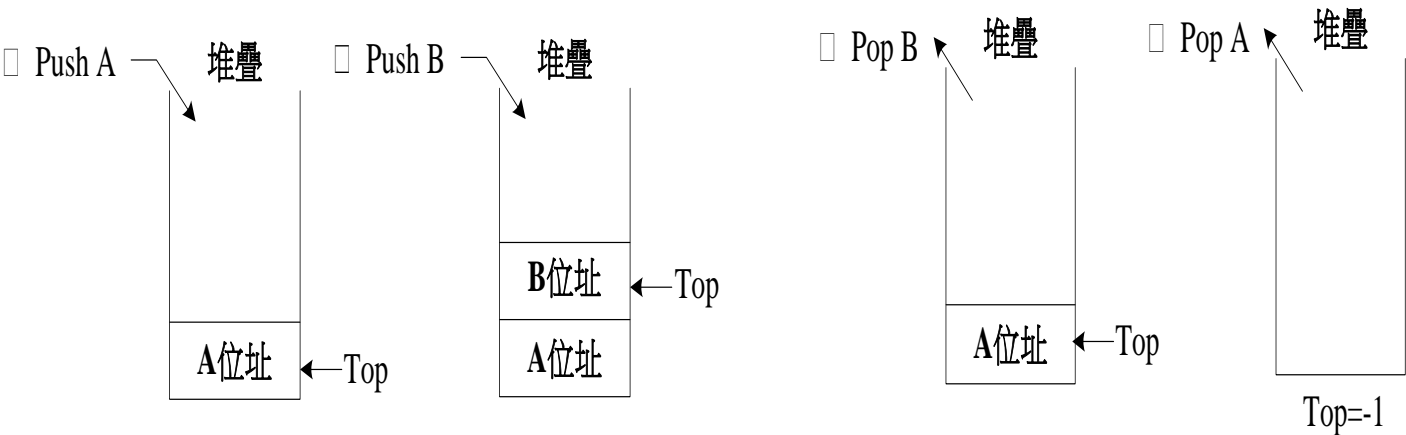


堆疊與佇列的應用 (2/13)

在電腦中，我們可以利用
堆疊具有**後進先出**的特性
，來解決副程式的呼叫。



呼叫副程式時，必須先將**返回位址**暫時儲存到「堆疊」中。



(一)呼叫副程式

(二)return返回主程式



堆疊與佇列的應用 (3/13)

- 要解決的問題是有先進先出的性質時，則用佇列來解決。
 - 作業系統的工作安排(job scheduling)，若不考慮特權(priority)的話。
 - 銀行櫃台的服務、停車場的問題、大型計算機中心列印報表的情形，以及飛機起飛與降落等等的應用。



堆疊與佇列的應用 (4/13)

- 一般的算術運算式皆是以中序法來表示，亦即運算子(operator)置於運算元(operand)的中間。
- 而後序法表示運算子在其運算元後面，如： $A * B / C$ ，此乃中序表示式，而其後序表示式是 $AB * C /$ 。



堆疊與佇列的應用 (5/13)

- 算術運算式由中序變為後序可依下列三步驟進行：
 - 將式子中的運算單元適當的加以括號，此時須考慮運算子的運算優先順序。
 - 將所有的運算子移到其對應的**右括號**。
 - 將所有的括號去掉。



堆疊與佇列的應用 (6/13)

- 如將 $A*B/C$ 化為後序表示式，步驟如下：
 - $((A* B)/C)$
 - $((AB)*C)/$
 - $AB*C/$



堆疊與佇列的應用 (7/13)

- 再舉一例將 $A-B/C+D * E$ 化成後序表示式，步驟如下：
 - $((A-(B/C))+(D * E))$
 - $((A(BC)/)-(DE)*)+$
 - $ABC/-DE*+$



堆疊與佇列的應用 (8/13)

運算子	優先順序（數字愈大，表示優先順序愈高）
$*$, $/$, $\%$	5
$+$ （加）, $-$ （減）	4
$<$, $<=$, $>$, $>=$	3
$\&\&$	2
$ $	1



堆疊與佇列的應用 (9/13)

- 將算術運算式由中序表示改變為後序表示式，除了上述的方法，也可以利用堆疊的觀念來完成。
- 算術運算子的in-stack(在堆疊中)與in-coming(在運算式中)的優先順序。



堆疊與佇列的應用 (10/13)

符號	in-stack priority	in-coming priority
)	-	--
*, /, %	2	2
+ (加) , - (減)	1	1
(0	4



堆疊與佇列的應用 (11/13)

- 開始時堆疊是空的，假設稱運算式中的運算子和運算元是 token，當 token 是運算元，不必考慮，一律輸出
- 如果進來的 token 是運算子，而且若此運算子的 in-stack priority (ISP) 大於或等於 in-coming priority (ICP)，則輸出放在堆疊的運算子，繼續執行到 $ISP < ICP$ ，之後再將欲進來的運算子放入堆疊中。



堆疊與佇列的應用 (12/13)

■ $A+B*C$

next token	stack	output	說明
none	empty	none	
A	empty	A	由於 A 是運算元故輸出
+	<div><div></div><div>+</div></div>	A	
B	<div><div></div><div>+</div></div>	AB	B 是運算元故輸出
*	<div><div>*</div><div>+</div></div>	AB	由於 * 的 in-coming priority 大於 + 的 in-stack priority
C	<div><div>*</div><div>+</div></div>	ABC	C 是運算元
none	<div><div></div><div>+</div></div>	ABC*	pop 出堆疊頂端的元素 *
none	empty	ABC*+	再 pop 出堆疊頂端的元素 +



堆疊與佇列的應用 (13/13)

■ $A*(B+C)*D$

next token	stack	output	說明
none	empty	none	
A	empty	A	
*	<div>stack diagram: top cell empty, bottom cell contains *</div>	A	
(<div>stack diagram: top cell contains (, bottom cell contains *</div>	A	由於(的 in-coming priority 大於*的 in-stack priority
B	<div>stack diagram: top cell contains (, bottom cell contains *</div>	AB	
+	<div>stack diagram: top cell contains +, bottom cell contains (, bottom-most cell contains *</div>		的 in-coming priority 大於(的 in-stack priority
C	<div>stack diagram: top cell contains +, middle cell contains (, bottom cell contains *</div>	ABC	
)	<div>stack diagram: top cell contains +, bottom cell contains *</div>	ABC+)的 in-coming priority 小於+的 in-stack priority 故輸出+後再去掉(
*	<div>stack diagram: top cell contains +, bottom cell contains *</div>	ABC+*	此處輸出的*是在堆疊裏的*
D	<div>stack diagram: top cell contains +, bottom cell contains *</div>	ABC+*D	
none	empty	ABC+*D*	



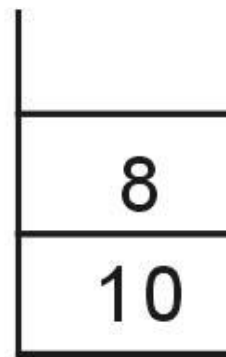
後序表示式 (1/4)

- 將中序表示式轉為後序表示式後，就可以將此式子的值計算出來，其步驟如下：
 - 將此後序表示式以一字串表示之。
 - 每次取一token，若此token為一運算元則將它push到堆疊，若此token為一運算子，則自堆疊pop出二個運算元並做適當的運算，若此token為 '\0'則goto步驟4。



後序表示式 (2/4)

- 將步驟2的結果再push到堆疊，goto步驟2。
- 彈出堆疊的資料，此資料即為此後序表示式計算的結果。
- 如有一中序表示式為 $10+8-6*5$ ，轉為後序表示式為 $10\ 8\ +\ 6\ 5\ *\ -$ ，利用上述的規則執行之，步驟如下：
 - 因為10為一運算元，故將它push到堆疊，同理8也是，故堆疊有2個資料分別為10和8。





後序表示式 (3/4)

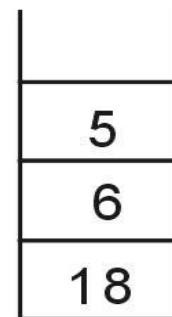
□之後的token為+，故pop出堆疊的8和10做加法運算，結果為18，再次將18push到堆疊。

10 8 + 6 5 * -



□接下來將6和5 push到堆疊。

18 6 5 * -





後序表示式 (4/4)

□ 之後的token為*，故pop 5和6做乘法運算為30，並將它push到堆疊。

18 6 5 * -

18 30 -

30
18

□ 之後的token為-，故pop 30和18，此時要注意的是18減去30，答案為-12(是下面的資料減去上面的資料)

□ 將-12 push到堆疊，由於此時已達字串結束點'\0'，故彈出堆疊的資料-12，此為計算後的結果。