



第四章

佇列(Queue)

本章學習目標

1. 讓學生了解**日常生活**有許多例子都是**佇列的應用**。
2. 讓學生了解**佇列的運作原理**。

本章內容

4-1 佇列

4-2 以陣列來製作佇列

4-3 環形佇列(circular queue)

4-4 進階佇列

4-5 佇列在電腦資料處理的應用

4-1 佇列

佇列(Queue)是一種先進先出 (First In First Out, FIFO) 的有序串列，它與堆疊處理資料方式是不大一樣的，亦即資料處理是在不同邊進行，也就是資料由一端加入，由另一端刪除。

因此日常生活中，也有一些佇列的例子，如排隊上公車、排隊買電影票等，皆是先到達的先處理、後到的後處理。

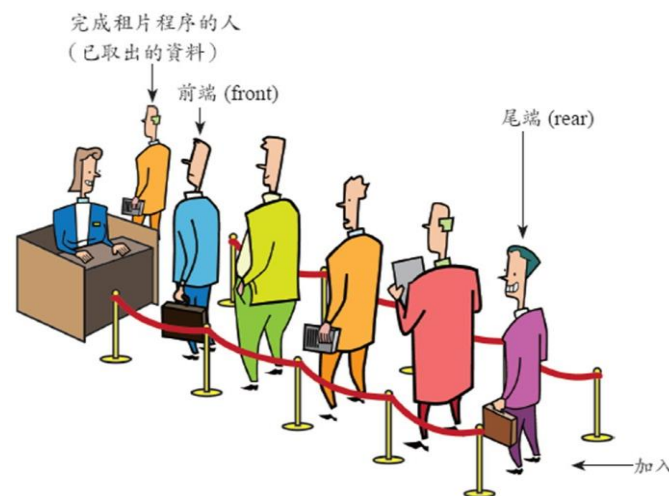
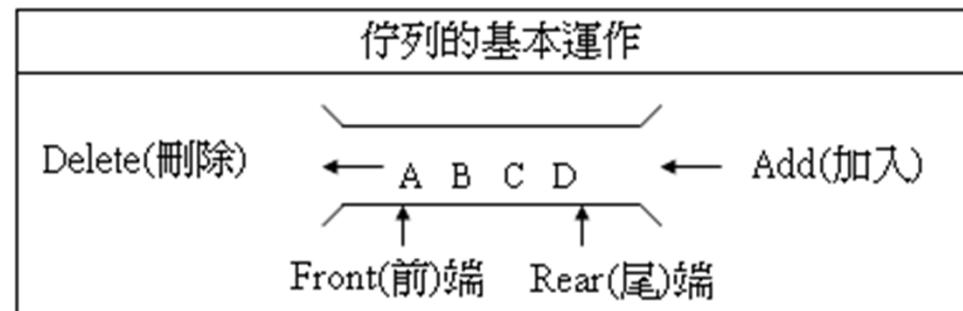


圖4-1 佇列的例子

【定義】

1. 一群**相同性質元素**的組合，即**有序串列**(Ordered List)。
2. 具有**先進先出** (First In First Out, FIFO) 的性質。
3. **加入元素**的動作發生在**Rear(尾)端**。
4. **刪除元素**的動作發生在**Front(前)端**。
5. Add(**加入**)/Delete(**刪除**)的動作皆發生在**不同兩端**。



【佇列常用專有名詞】

1. Create(Queue) : 建立一個空的佇列(Queue)。

2. Add(item, Queue) : 又稱Enqueue

指由佇列的後端 (Rear) 加入一個新項目。

3. Delete(item, Queue) : 又稱Dequeue

指從佇列的前端 (Front) 刪除一個項目。

佇列的兩個動作

4. IsFull(Queue)

判斷佇列是否已滿，若已滿為真(True)，否則為假(False)。

5. IsEmpty(Queue)

判斷佇列是否為空，若空為真(True)，否則為假(False)。

【佇列的兩個動作】

1. Add(item, Queue) : 又稱Enqueue

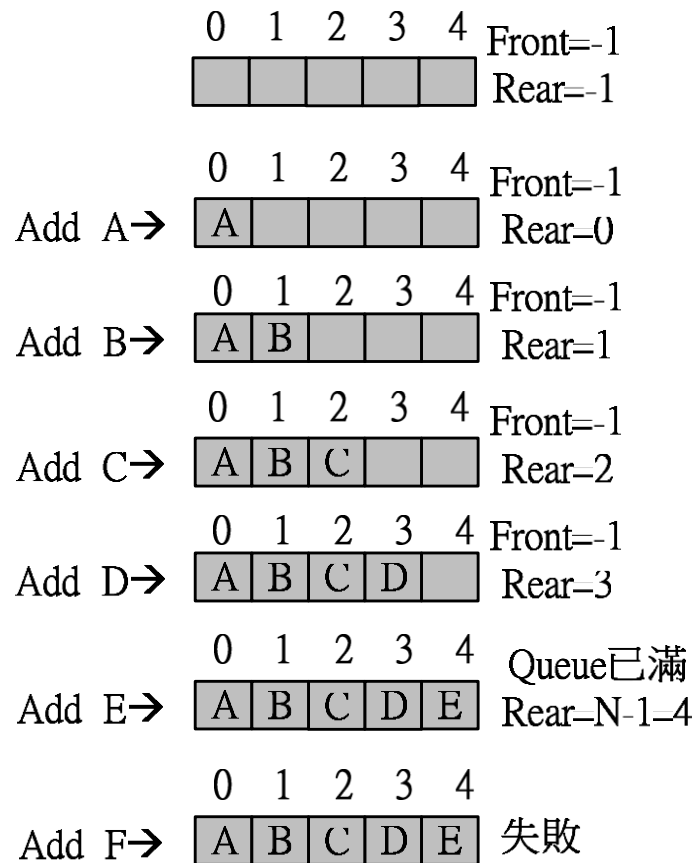
指由佇列的後端 (Rear) 加入一個新項目。

【演算法】

```
Procedure Add(item, Queue)
Begin
  if (Rear=N-1)           //如果Rear指標指到佇列的尾端，則
    Queue Is Full;        //代表佇列已滿
  else                    //如果不是，則
  {
    Rear=Rear+1;          //Rear指標加1
    Queue[Rear]=item;      //再將資料加入到Rear指標所在的佇列中
  }
End
End Procedure
```

【佇列的兩個動作】 (續)

【運作過程】



【佇列的兩個動作】 (續)

2. Delete(item, Queue) : 又稱Dequeue

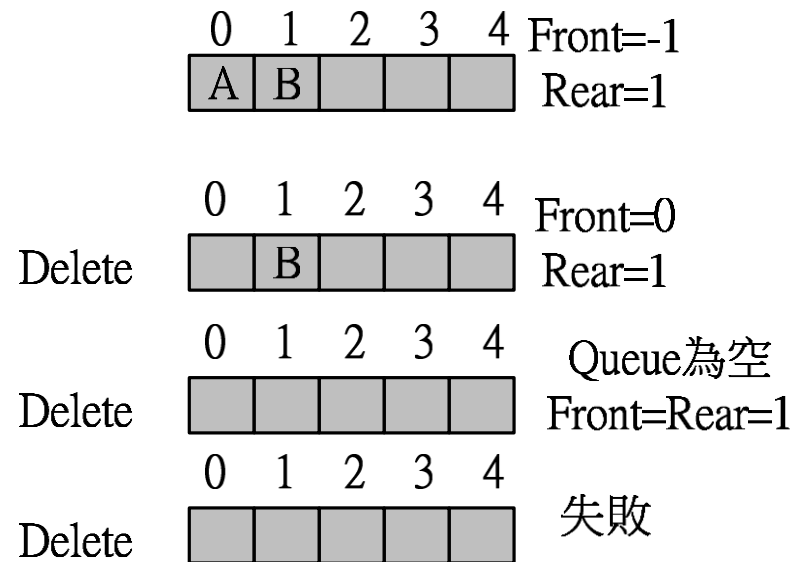
指從佇列的前端 (Front) 刪除一個項目。

【演算法】

```
Procedure Delete(item, Queue)
Begin
  if (Front=Rear)           //如果前端指標Front等於尾端指標Rear時，則
    Queue Is Empty;        //代表佇列為空
  else                      //否則
  {
    Front=Front+1;          //Front指標加1
    item=Queue[Front];      //再將Front指標所在佇列內容刪除
  }
End
End Procedure
```

【佇列的兩個動作】 (續)

【運作過程】



【舉例】假設有一個空的Queue，實施下列的動作：

Add A,B,C

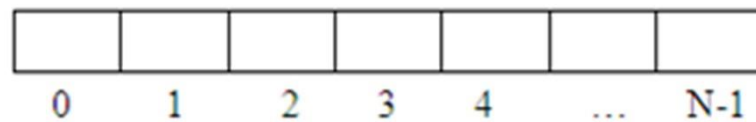
Delete A,B

Add D

請呈現以上佇列運作的過程。

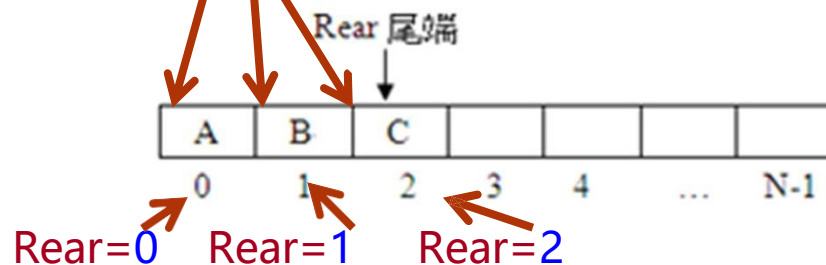
【解答】

①空的Queue



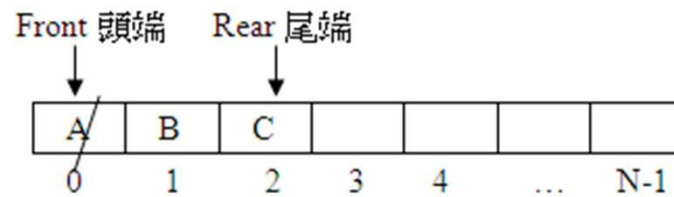
Front=-1
Rear=-1

② Add A到Queue
Add B到Queue
Add C到Queue



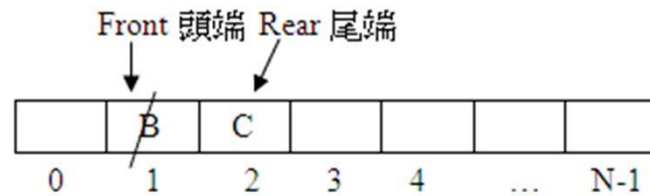
Front=-1
Rear=2

③ Delete 從Queue



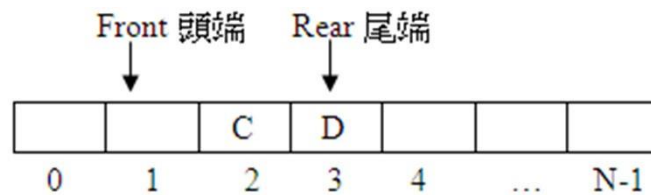
Front=0
Rear=2

④ Delete 從Queue



Front=1
Rear=2

⑤ Add D到Queue



Front=1
Rear=3

【佇列的應用】

佇列經常應用在電腦中，一般作業系統內均有一個工作佇列(job queue)，若系統不考慮工作的優先權時，則工作將依進入系統的先後順序來處理，亦即先到的工作先處理。例如：我們使用的印表機的工作也是採用「佇列」的特性來處理。

1. 作業系統的工作排程。

2. 計算機的模擬程式。

3. 磁碟管理的輸出入

(input/output) 排序。

4. I/O 的緩衝區(Buffer)上。

5. 優先佇列(Priority Queue)

6. 圖形的廣度追蹤(BFS)

4-2 以陣列來製作佇列

製作佇列最常用的方法就是利用一維陣列來完成。

其製作的過程如下：

1. 建立佇列結構：Create(Queue) // 指建立一個空的Queue

【演算法】

Procedure Create(Queue)

Begin

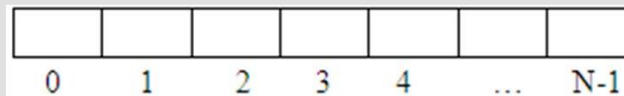
{

```
#define N 10           //定義佇列大小
char Queue[N];        //以陣列Queue當作佇列
int Front=-1;          //設定佇列頭端的索引值之初值為-1
int Rear=-1;          //設定佇列尾端的索引值之初值為-1
```

}

End

End Procedure



空的佇列(Queue)

Front=-1

Rear=-1

2.將資料加入佇列(Add動作)：Add(item, Queue)

將資料(item)加入到Queue中；成為Rear端元素。

如果佇列已滿，則無法進行。

【演算法】

01	Procedure Add(item,Queue)	
02	Begin	
03	if (Rear=N-1)	//如果Rear指標指到佇列最尾端
04	Queue Is Full;	//代表佇列已滿
05	Else	//如果不是，則
06	{	
07	Rear=Rear+1;	//Rear指標位址加1
08	Queue[Rear]=item	//再將資料加入到Rear指標所在的佇列中
09	}	
10	End	
11	End Procedure	

【實例】

假設有一個空的Queue，實施下列的動作：

Add A到Queue

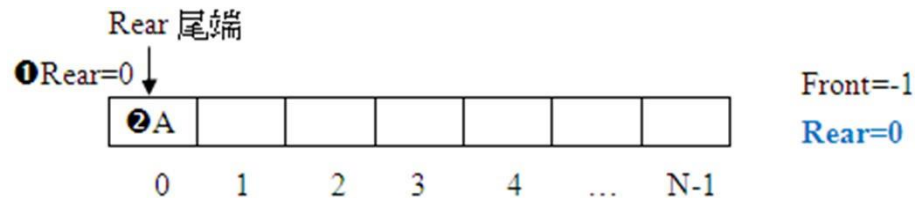
Add B到Queue

Add C到Queue

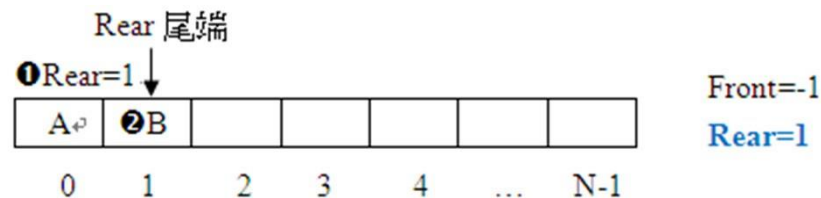
請呈現以上佇列運作之後的Front與Rear之值。

【解答】

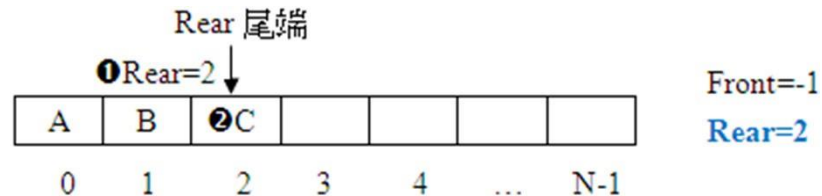
步驟1：先將Rear指標位址加1，此時Rear指向0，再將A加入到佇列中。



步驟2：先將Rear指標位址加1，此時Rear指向1，再將B加入到佇列中。



步驟3：先將Rear指標位址加1，此時Rear指向2，再將C加入到佇列中。



3.刪除資料>Delete動作)：Delete(item,Queue)

指刪除Queue的Front頭端元素，如果Queue是空，則無法進行。

【演算法】

01	Procedure Delete(item, Queue)
02	Begin
03	if (Front=Rear) //如果Front等於Rear時，則
04	Queue Is Empty //代表佇列為空
05	Else //否則
06	{
07	Front=Front+1; //Front指標位址加1
08	item=Queue[Front]; //再將Front指標所在佇列內容刪除
09	}
10	End
11	End Procedure

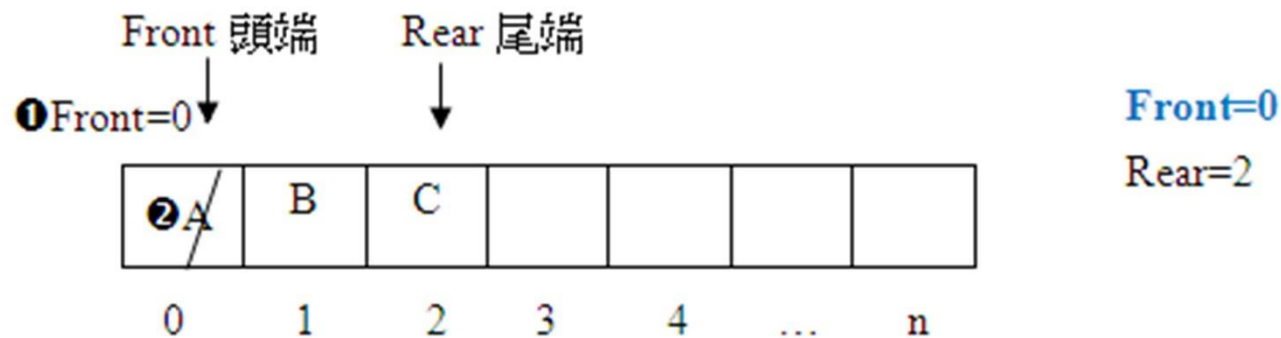
【實例】

假設有一個Queue，已經有A、B、C三個資料項，如果再實施下列的動作：

Delete A從Queue

請呈現以上佇列運作之後的Front與Rear之值。

【解答】



說明： 先將Front指標位址加1，此時Front指向0
再將Front指標所在佇列內容刪除

4.判斷佇列是否已滿：IsFull(Queue)

判斷Rear是否等於N-1，若是則傳回True，否則傳回False。

【基本演算法】

01	Procedure IsFull(Queue)
02	Begin
03	if (Rear=N-1) //如果Rear指標指到佇列最尾端，則
04	return True; //傳回True
05	else //如果不是，則
06	return False; //傳回False
07	End
08	End Procedure

【缺點】在佇列中，當Rear=N-1時，並不能保證Queue真的已滿。

【分析原因】

尚有**2**個空間



當**Rear=N-1**時，
並不能保證**Queue**真的已滿

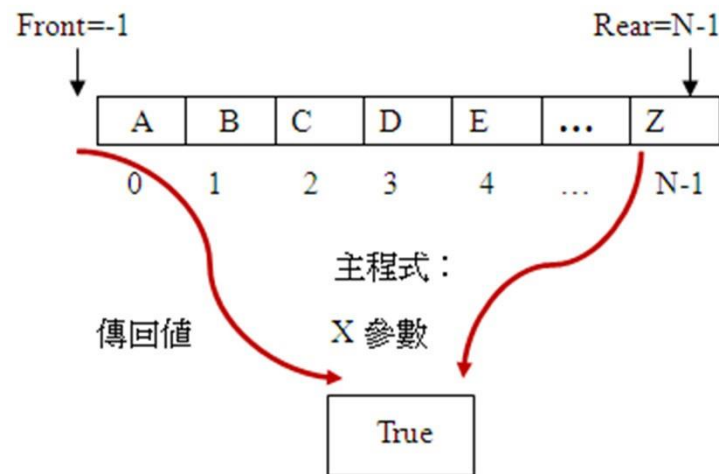
在上圖中，判斷佇列是否為滿的副程式IsFull(Queue)，必須要**額外**增加一個判斷條件為：`if (Front=-1)`，否則像上圖中，尚有**2**個空間，而結果卻為滿。

【改良後演算法】

```
01 Procedure IsFull(Queue)
02   Begin
03     if (Rear=N-1) And (Front=-1) //如果Rear=N-1及Front=-1，則
04       return True;                //傳回True
05     else                            //如果不是，則
06       return False;               //傳回False
07   End
08 End Procedure
```

【實例】

假設目前在佇列中已經有 $N-1$ 個資料項，並且Front指向 -1 ，欲判斷佇列是否已滿，請將它會傳回什麼值？



說明：

- ① 先判斷Rear指標是否為 $N-1$ ，並且Front指向 -1 ，如果是，則
- ② 傳回True，否則傳回False。

5.判斷佇列是否是空的：IsEmpty(Queue)

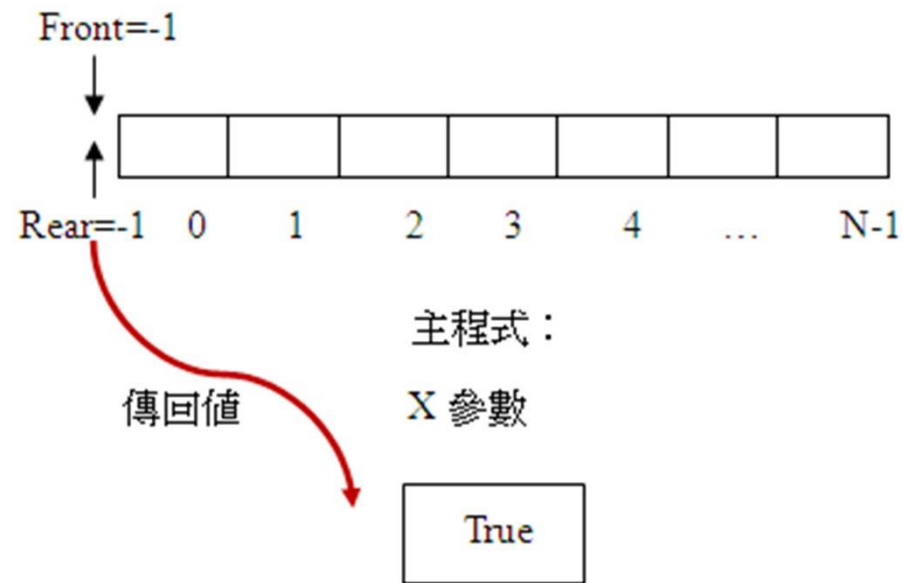
判斷Front是否等於Rear，若是則傳回True，否則傳回False。

【演算法】

01	Procedure IsEmpty (Queue)
02	Begin
03	if (Front=Rear) //如果Front等於Rear時，則
04	return True; //傳回True
05	else //如果不是，則
06	return False; //傳回False
07	End
08	End Procedure

【實例】

假設目前在佇列中沒有任何資料，欲判斷佇列是否空，請將它會傳回什麼值？

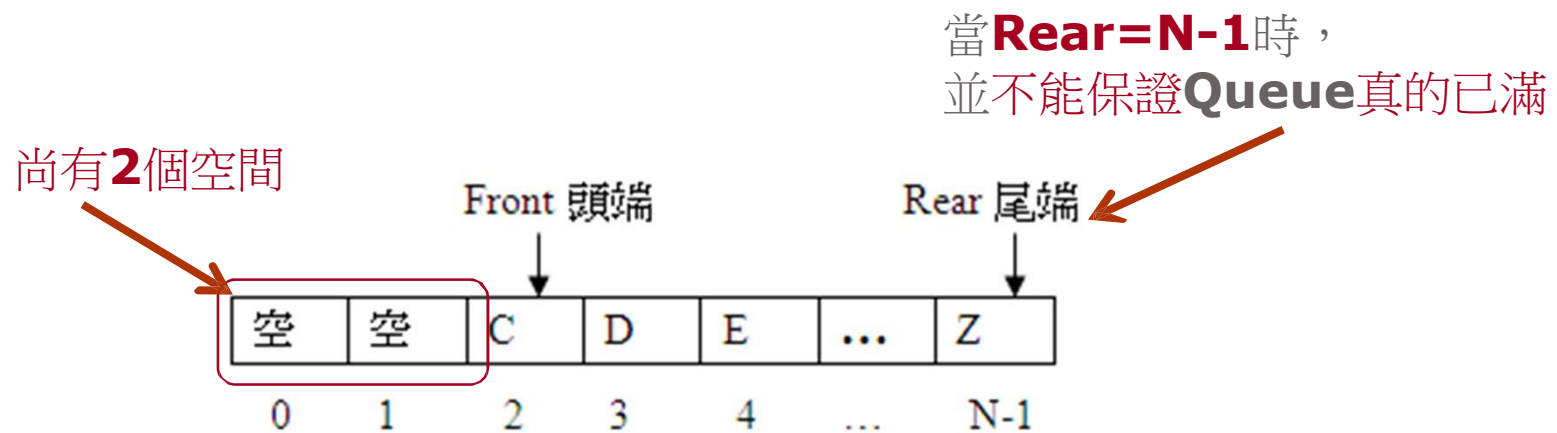


說明：

- ① 判斷Rear 是否等於Front，如果是，則
- ② 傳回True，否則傳回False。

4-3 環形佇列(circular queue)

由於佇列有一個問題，就是前端(Front)尚有空位時，卻再加入元素時，發現此佇列已滿。



解決方法：使用環形佇列(circular queue)。

【定義】

1. 環狀佇列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。

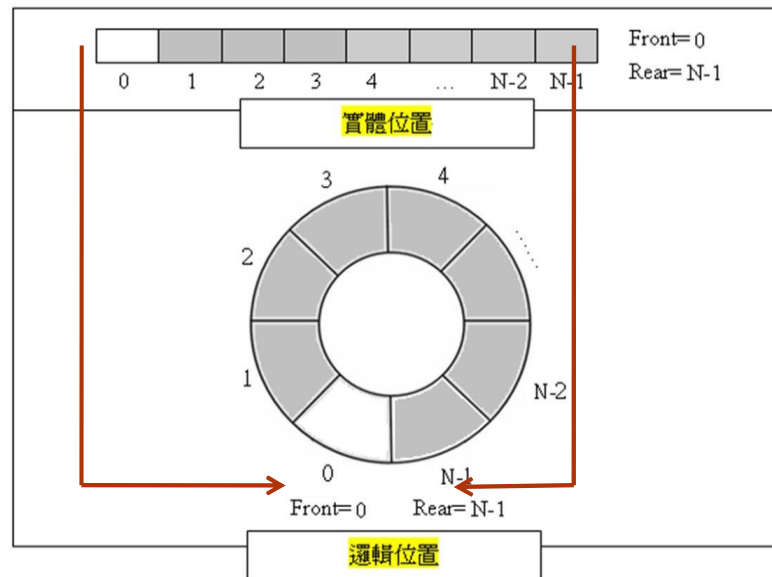
2. 最多使用 $(N-1)$ 個空間。

3. 指標 $Front$ 永遠以逆時鐘方向指向佇列前端元素的前一個位置。

4. 指標 $Rear$ 則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！



4-3.1 環形佇列的基本運作功能

1.產生環形佇列結構：Create(CQueue) //指建立一個空的環形佇列

【演算法】

Procedure Create(CQueue)

Begin

{

#define N 10 //定義環形佇列大小

char CQueue[N]; //以陣列CQueue當作佇列

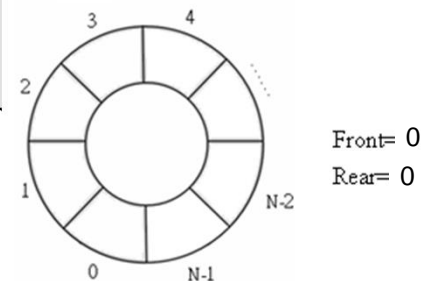
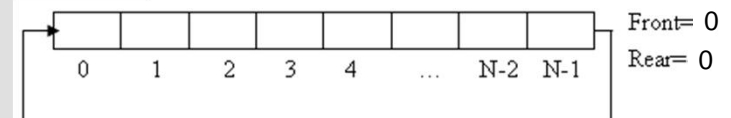
int Rear=0; //設定尾端的索引值之初值為0

int Front=0; //設定頭端的索引值之初值為0

}

End

End Procedure



2.將資料放入環形佇列：Add(item, CQueue)

【演算法】

01	Procedure Add(item,CQueue)	
02	Begin	//N代表環形佇列大小
03	Rear=(Rear+1) Mod N	//先計算欲加入資料的所在位置
04	if (Rear=Front)	//如果Rear=Front，則
05	Circular Queue Is Full;	//代表環形佇列已滿
06	Else	//如果不是，則
07	CQueue[Rear]=item;	//將資料加入到Rear指標所在的佇列中
08	End	
09	End Procedure	

【題目】 假設有一個空的環形佇列CQueue，實施下列的動作：

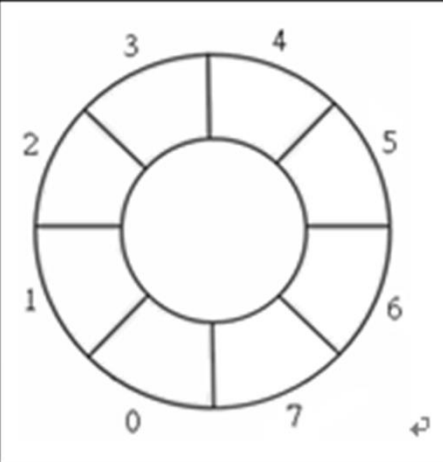
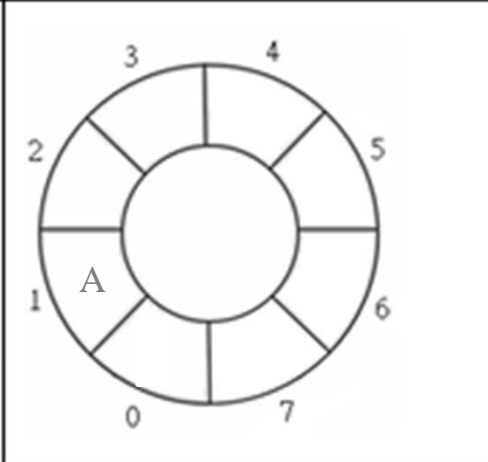
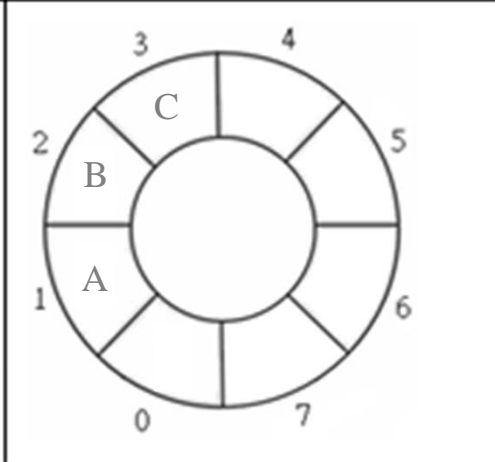
Add A

Add B,C

請呈現以上佇列運作之後的Front與Rear之值。

【解答】

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

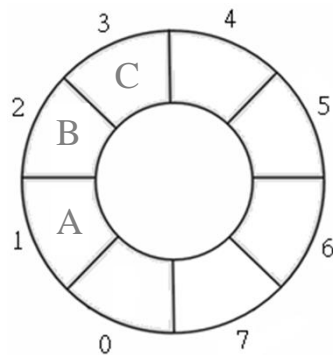
		
空佇列 Front= 0 Rear= 0	動態加入 A Front= 0 Rear= 1	動態加入 B,C Front= 0 Rear= 3

3.從環形佇列刪除資料：Delete(item,Cqueue)

【演算法】

01	Procedure Delete(item, CQueue)	
02	Begin	
03	if (Front=Rear)	//如果Front等於Rear時，則
04	Circular Queue Is Empty;	//代表佇列為空
05	Else	//否則
06	{	
07	Front=(Front+1) Mod N	//先計算欲刪除資料的所在位置
08	item=CQueue[Front];	//再刪除資料
09	}	
10	End	
11	End Procedure	

【題目】 假設目前有一個環形佇列CQueue，其內容如下：



Front= 0
Rear=3

現在實施下列的動作：

Delete A

Add D

Delete B

Add E,F,G,H,I

Add J

Delete Delete Delete Delete Delete Delete

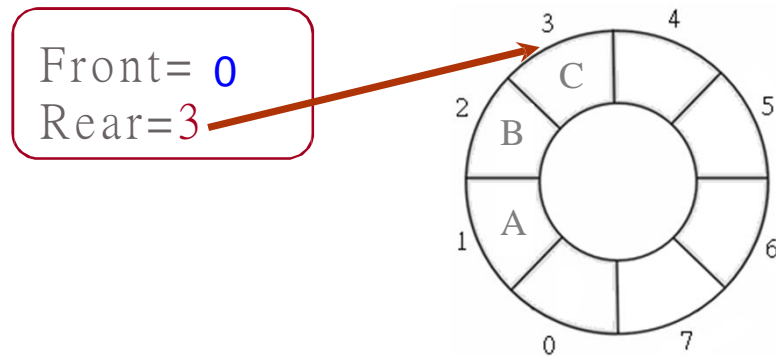
請呈現以上佇列運作之後的Front與Rear之值。

【解答】 在下一頁

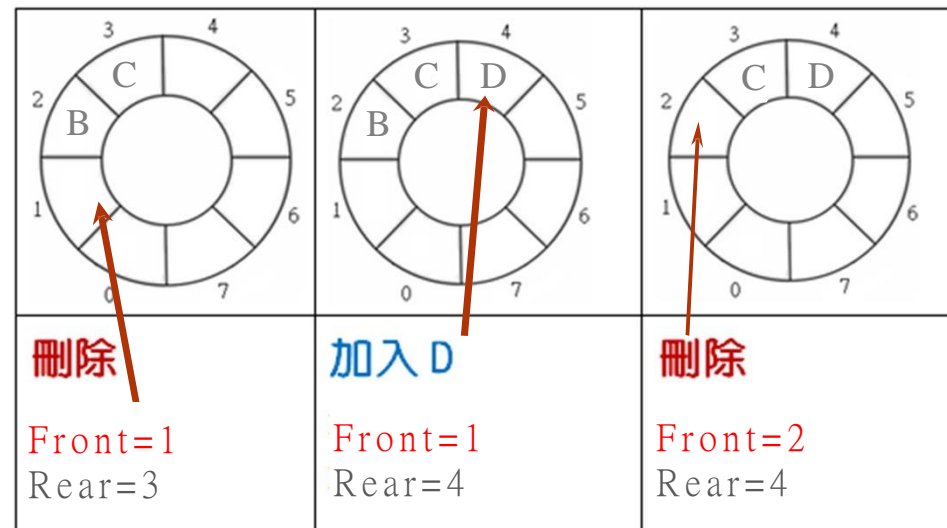
題目：運作前

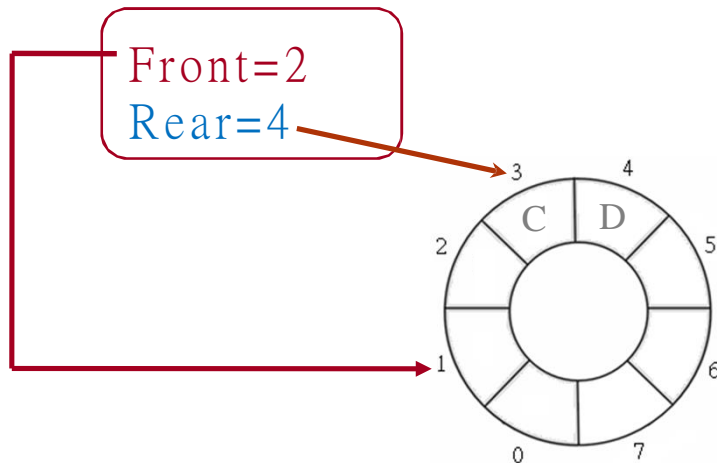
加入公式： $\text{Rear} = (\text{Rear} + 1) \text{ Mod } N$

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



運作過程：





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

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

01	Rear=(Rear+1) Mod N	//先計算欲加入資料的所在位置
02	if (Rear=Front)	//如果Rear=Front, 則
03	Circular Queue Is Full;	//代表環形佇列已滿
04	
05	

01	if (Front=Rear)	//如果Front等於Rear時, 則
02	Circular Queue Is Empty;	//代表佇列為空
03	
04	

運作過程：

加入 E, F, G, H, I	加入 J	刪除 7 次
Front= 2 Rear= 1	Front= 2 Rear= 2	Front= 2 Rear= 2
佇列已滿	失敗	佇列為空

4-3.2 環形佇列的改良

由於傳統的環形佇列會浪費一個空間，因此，其改良作法為另外再增加一個變數(Tag)來判斷，以記錄每一次的動作。

【作法】當每次加入資料時，將Tag設為1，而刪除資料時設為0。

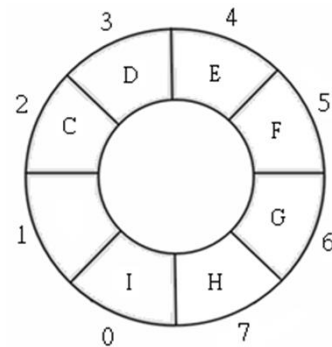
因此，在檢查Front是否等於Rear時，先檢查Tag的狀態。

1.將資料加入環形佇列：Add(item, CQueue)

【演算法】

```
Procedure Add(item,CQueue)
Begin
  if (Rear=Front) And (Tag=1)           //如果Rear=Front, 並且Tag=1時, 則
    CircularQueue Is Full;               //表示佇列已滿
  Else                                   //如果不是, 則
  {
    Rear=(Rear+1) Mod n                  //先計算欲加入資料的所在位置
    CQueue[Rear]=item;                   //將資料加入到Rear指標所在的佇列中
    if(Rear=Front) then Tag=1;           //如果Rear=Front, 則設Tag為1
  }
End
End Procedure
```

【題目】 假設目前有一個環形佇列CQueue，其內容如下：



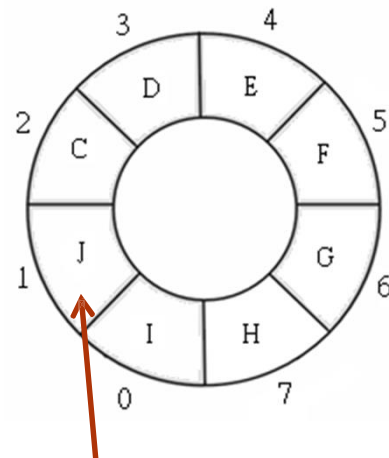
僅剩下一個空間

Front=1

Rear=0

此時欲再加入一個元素(J)，請呈現運作之後的Front與Rear之值及環形佇列的狀態。

【解答】



Front=1

Rear=1

再加入一項資料J時：

Rear=1

使得Front=Rear，並且Tag=1

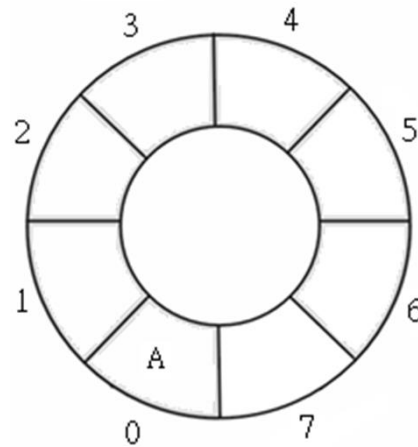
代表環形佇列已滿

2. 從環形佇列刪除資料：Delete(item, CQueue)

【演算法】

```
Procedure Delete(item, CQueue)
Begin
  if (Front=Rear) And (Tag=0)           //如果Rear=Front, 並且Tag=0時, 則
    Circular Queue Is Empty;           //表示佇列已空
  Else                                   //如果不是, 則
  {
    Front=(Front+1) Mod n               //先計算欲刪除資料的所在位置
    item=CQueue[Front];                 //再刪除資料
    if(Rear=Front) then Tag=0;          //如果Rear=Front, 則設Tag為0
  }
End
End Procedure
```

【題目】 假設目前有一個環形佇列CQueue，其內容如下：



僅佔用一個空間
Front=-1
Rear=0

此時欲再刪除一個元素，請呈現運作之後的Front與Rear之值及環形佇列的狀態。

【解答】

再刪除一項資料A時：

Front=0

使得Front=Rear，Tag=0

代表環形佇列已空

【優點】 最多可以利用到N個空間。

【缺點】 Add與Delete動作均須額外多了一個條件判斷式。

4-4 進階佇列

除了前面所提到的一般佇列與環狀佇列之外，在某些情況之下，也會使用到比較特殊的佇列，例如：優先佇列(priority queue)及雙向佇列(double-ended queue: deque)。

4-4.1 優先佇列(priority queue)

在優先佇列中，其元素的加入及刪除不須要遵守先進先出的特性。

這種情況在日常生活中時常遇到。例如：醫院的急診室、捷運座位提供老幼婦孺者之仁愛座等等。

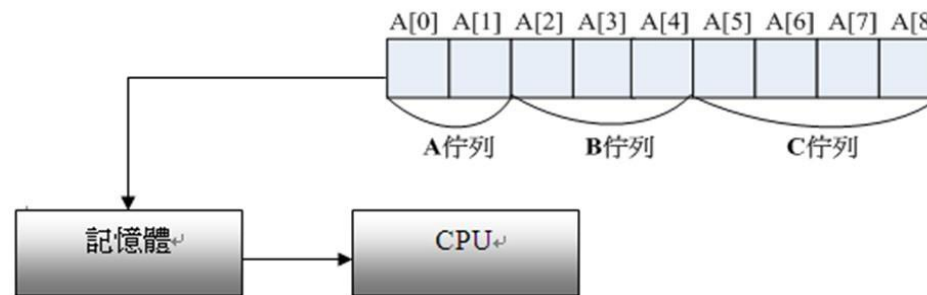
【定義】

1. 為一種不必遵守佇列特性 - FIFO(先進先出)的有序串列。
2. 每一個佇列的元素，都給予一個優先順序權，其數字最大者代表有最高優先權。
3. 優先佇列可以利用陣列結構及鏈結串列來解決。

【題目】 假設目前有**三個佇列(A,B,C)**，而這**三個佇列的優先權**分別為**2,3,4**(**數值愈大**，其**優先權愈高**)，請比較一般佇列與優先佇列的**差異**？

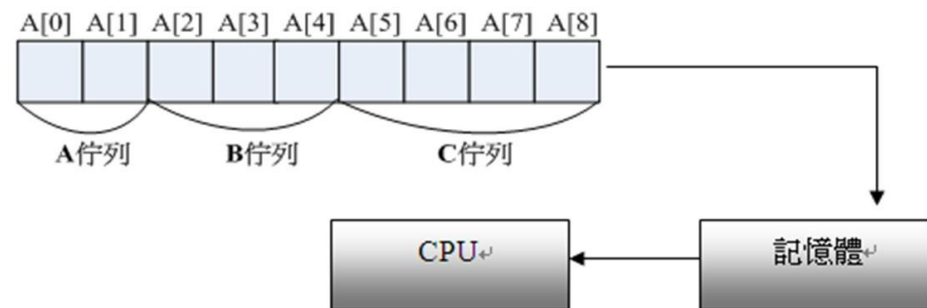
【解答】

一、一般佇列(優先權相同，先到先處理)



處理順序：A佇列 B佇列 C佇列

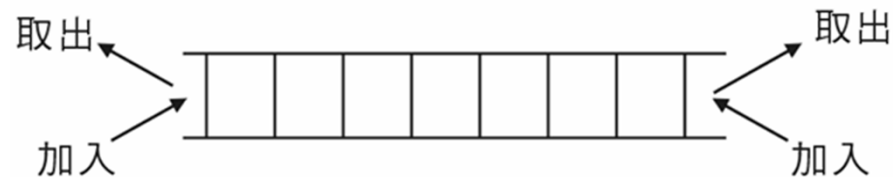
二、優先佇列



處理順序：C佇列 B佇列 A佇列

4-4.2 雙向佇列(Double-Ended Queue: Deque)

【定義】雙向佇列(Deque)是一種特殊的資料結構，它的兩端都可做加入與取出資料的動作。亦即它是一種前後兩端都可輸入或取出資料的有序串列。



【題目】 假設目前有一個10個空間的雙向佇列，此時欲輸入一串數字：

1,2,3,4,5,6,7，請問動態呈現任一種輸入方式。

【解答】

步驟一：

1									
---	--	--	--	--	--	--	--	--	--

步驟二：

1	2								
---	---	--	--	--	--	--	--	--	--

步驟三：

1	2	3							
---	---	---	--	--	--	--	--	--	--

步驟四：

1	2	3	4						
---	---	---	---	--	--	--	--	--	--

步驟五：

1	2	3	4						5
---	---	---	---	--	--	--	--	--	---

步驟六：

1	2	3	4	6					5
---	---	---	---	---	--	--	--	--	---

步驟七：

1	2	3	4	6	7	5
---	---	---	---	---	-----	-----	-----	---	---

4-5 佇列在電腦資料處理的應用

佇列在電腦科學中應用的相當廣泛，例如電腦模擬(Computer Simulation)，作業系統(OS)中電腦資源的分配都是利用佇列來表示。

【題目】 假設目前在緩衝區中有A~Z共26個佇列，當A.doc佇列在T1時間送到記憶體，並由cpu處理之後，接著B.doc，以此類推。

