

4-3 環形佇列

由於佇列有一個問題，就是前端 (Front) 尚有空位時，再加入元素，卻發現此佇列已滿。此時的解決方法就是使用環形佇列 (Circular Queue)。

○ 定義

是指一種環形結構的佇列。

○ 作法

將一維陣列的第 0 個元素，存放到環形佇列第 0 個位置中，一維陣列的第 1 個元素，存放到環形佇列第 1 個位置中，以此類推，同時 $Q[0]$ 為 $Q[N-1]$ 的下一個元素。

○ 圖解說明

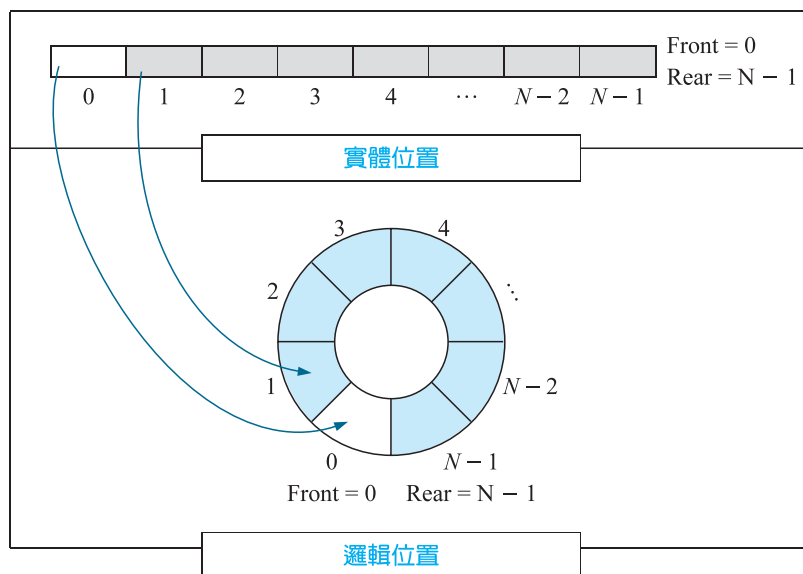
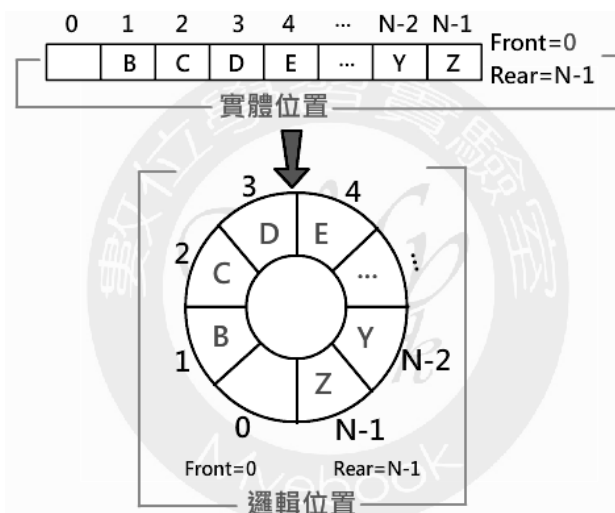


圖 4-4 環形佇列

○ 缺點

佇列滿了，浪費一個空格沒有使用，若再加入一個項目時，Rear 等於 0，也就是 Front 等於 Rear，如此無法分辨此時佇列是空的還是滿的。因此，環形佇列必須浪費一個空格。當 Front 等於 Rear 時，環形佇列為空的。當 $(Rear+1) \bmod N$ 等於 Front 時，環形佇列為已滿。



單元評量

1. 使用一般佇列存放資料時，當前端 (Front) 尚有空位時，再加入元素，卻發現此佇列已滿，請問此時使用下列那一個方法較佳？
(A) 優先佇列 (B) 環形佇列 (C) 雙向佇列 (D) 以上皆非
2. 在環形佇列中，它是利用一種 $Q[0: N-1]$ 的一維陣列來存放資料，請問佇列 $Q[0]$ 是那一個的下一個元素？
(A) $Q[1]$ (B) $Q[2]$ (C) $Q[N-1]$ (D) $Q[N-2]$
3. 使用環形佇列存放資料時，往往會浪費多少空間？
(A) 1 (B) 2 (C) 3 (D) 3 個以上

4-3.1

環形佇列的基本運作功能

其實「環形佇列」基本運作功能必須撰寫以下三個函數，才能算是一個完整環形結構的功能。

1. Create(CQueue) 函數。
2. Add 函數。
3. Delete 函數。

一、Create (CQueue) 函數

是指用來建立一個空的環形佇列。

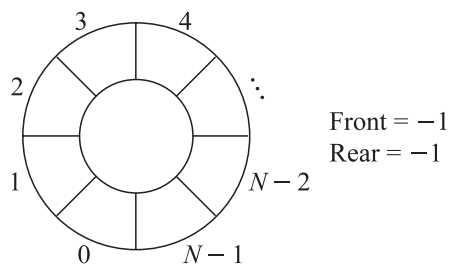
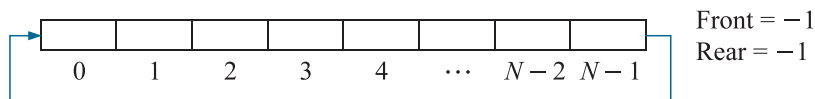
(一) 演算法

```
#define N 10           // 定義環形佇列大小
char CQueue[N];       // 以陣列 CQueue 當作佇列
int Rear=-1;          // 設定後端的索引值之初值為 -1
int Front=-1;         // 設定前端的索引值之初值為 -1
```

(二) 陣列的記憶體配置 (空的 Circular Queue)

圖解說明

假設我們定義 $N = 10$ (也就是編號 $0 \sim 9$) 共 10 個空間，並且 Front 與 Rear 的初始值都為 -1 ，其中，Front 指標是用來記錄目前環形佇列前端的索引值，並且 Rear 指標是用來記錄目前環形佇列後端的索引值，而 Front 與 Rear 指標初始值設為 -1 表示環形佇列為空。



參看隨書光碟。

二、Add 函數

是指加入新項目到環形佇列中。

(一) 演算法

將資料放入環形佇列

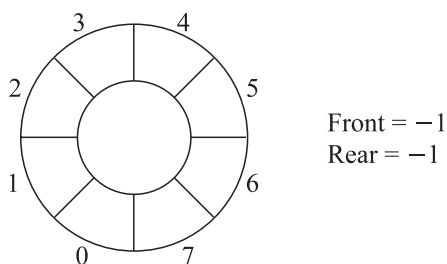
```

01 Procedure Add(item,CQueue)
02   Begin
03     Rear=(Rear+1) Mod N    //N 代表環形佇列大小
04     if (Rear=Front)
05       Circular Queue Is Full; //Circular Queue 已滿
06     else
07       {
08         CQueue[Rear]=item;
09       }
10     End
11   End Procedure
12

```

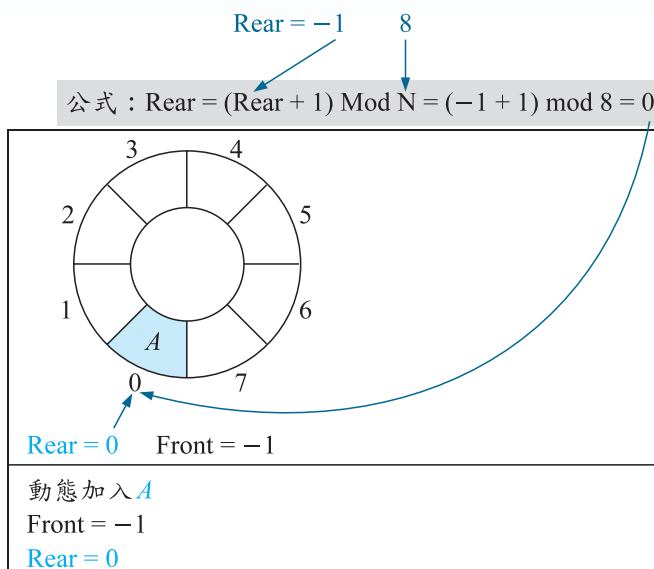
(二) 實例

假設現在有 3 個資料項，分別為 A, B, C ，接下來，請依序將這三個資料項加入環形佇列中，並且呈現以上佇列運作之後的 $Front$ 與 $Rear$ 之值。

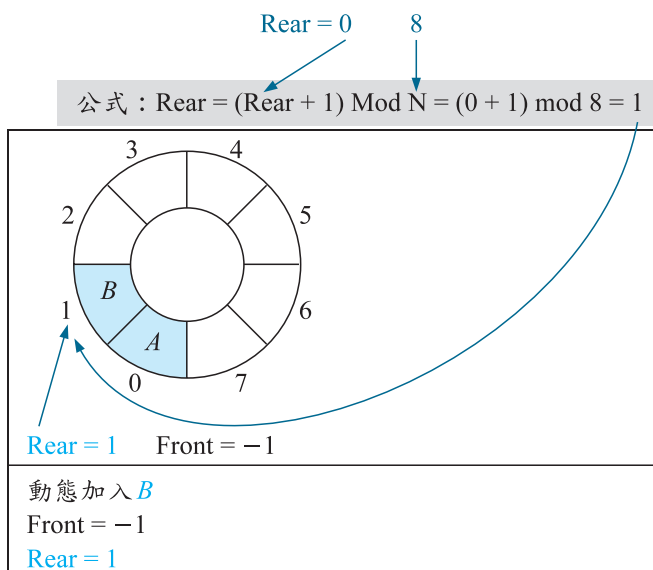


解答

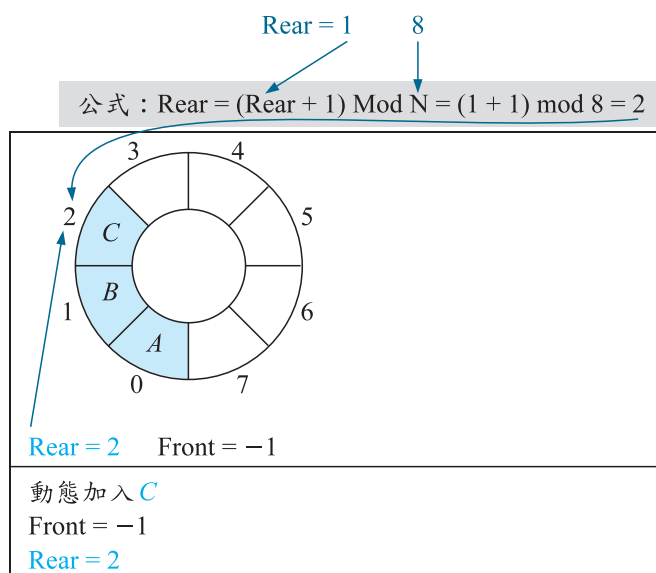
- **步驟一：**當我們要加入第一個資料項時，演算法會先計算欲加入資料的所在位置 $Rear$ 指標，此時， $Rear$ 指標從 -1 指向 0 (也就是 $Rear$ 加 1 之後，除以 8 ，再取餘數)，並且 $Rear$ 指標不等於 $Front$ 指標，所以，會將資料項 A 加入 $Rear$ 指標所在的佇列中。



- **步驟二：**當我們要加入第二個資料項時，演算法會再計算欲加入資料的所在位置 Rear 指標，此時，Rear 指標從 0 指向 1 (也就是 Rear 加 1 之後，除以 8，再取餘數)，並且 Rear 指標不等於 Front 指標，所以，會將資料項 B 加入 Rear 指標所在的佇列中。



- **步驟三：**當我們要加入第三個資料項時，演算法會再計算欲加入資料的所在位置 Rear 指標，此時，Rear 指標從 1 指向 2 (也就是 Rear 加 1 之後，除以 8，再取餘數)，並且 Rear 指標不等於 Front 指標，所以，會將資料項 C 加入 Rear 指標所在的佇列中。



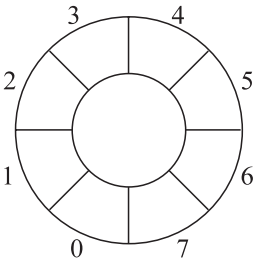
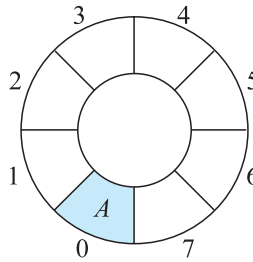
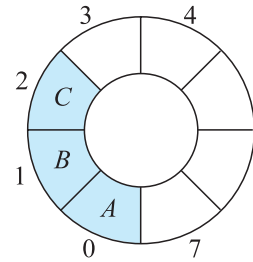
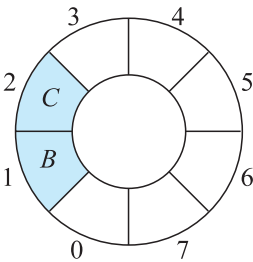
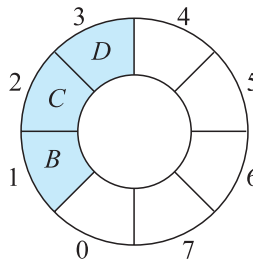
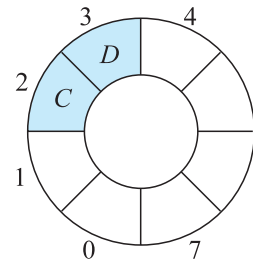
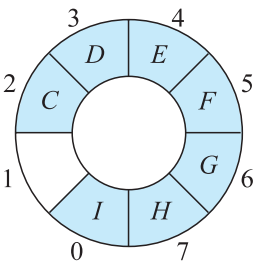
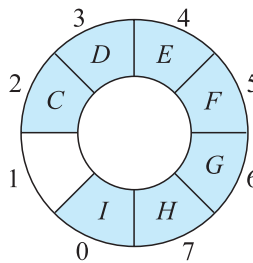
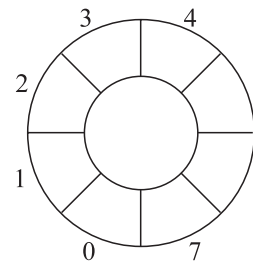
隨堂練習

假設有一個空的環形佇列 CQueue，實施下列動作：

Add	A						
Add	B,C						
Delete							
Add	D						
Delete							
Add	E,F,G,H,I						
Add	J						
Delete	Delete	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

		
空佇列 Front = -1 Rear = -1	加入 A Front = -1 Rear = 0	加入 B, C Front = -1 Rear = 2
		
刪除 Front = 0 Rear = 2	加入 D Front = 0 Rear = 3	刪除 Front = 1 Rear = 3
		
加入 E, F, G, H, I Front = 1 Rear = 0 佇列已滿	加入 J Front = 1 Rear = 1 失敗	刪除 7 次 Front = 0 Rear = 0 佇列為空



參看隨書光碟。

三、Delete 函數

是指從環形佇列中取出資料。

(一) 演算法

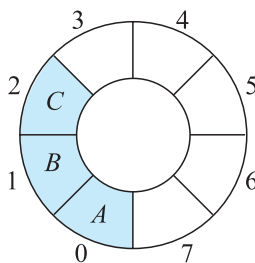
從環形佇列刪除資料

```

01 Procedure Delete(item, CQueue)
02   Begin
03     if (Front=Rear)
04       Circular Queue Is Empty; //Queue 為空
05     else
06       {
07         Front=(Front+1) Mod N
08         item=CQueue[Front];
09       }
10   End
11 End Procedure
    
```

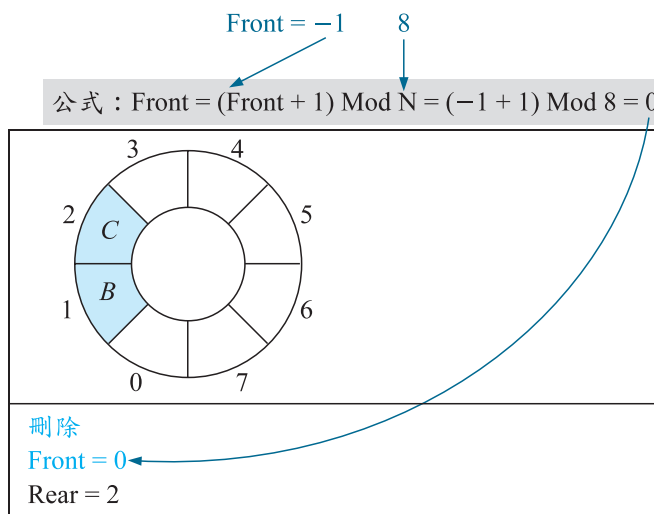
(二) 實例

假設目前在環形佇列中已經有 3 個資料項，分別為 A, B, C ，此時，欲取出 A 資料項，請問環形佇列運作之後的 Front 與 Rear 之值為何？



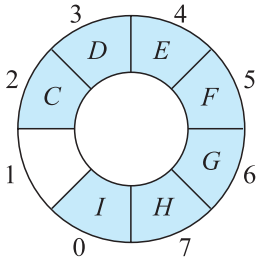
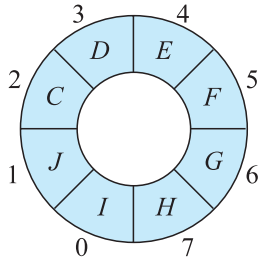
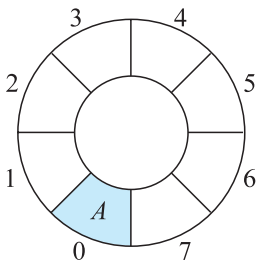
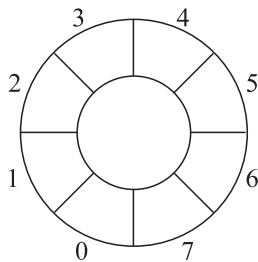
解答

當我們想從環形佇列中取出資料項時，演算法會先判斷 Front 指標是否等於 Rear 指標，如果不是，則必須先計算欲取出資料的所在位置，此時，Front 值等於 0 (也就是 Front 加 1 之後，除以 8，再取餘數)，然後，再從環形佇列中取出 Front 指標所在位置的資料。



四、分析

1. 若硬要使用 N 個空間，則 $Rear = Front$ 條件式成立時，將無法區分出佇列是否真的 Full 或 Empty，如下圖所示。
2. 判斷佇列是否已滿或為空，則使用相同的條件式 ($Rear = Front$)。
3. 加入或刪除動作的時間複雜度均為 $O(1)$ 。

	
<p>僅剩下一個空間 Front = 1 Rear = 0</p>	<p>再加入一項資料 J 時， Rear = 1 使得 Front = Rear → 環形佇列已滿</p>
	
<p>僅剩下一個空間 Front = 1 Rear = 0</p>	<p>再刪除一項資料 A 時， Front = 0 使得 Front = Rear → 環形佇列已空</p>

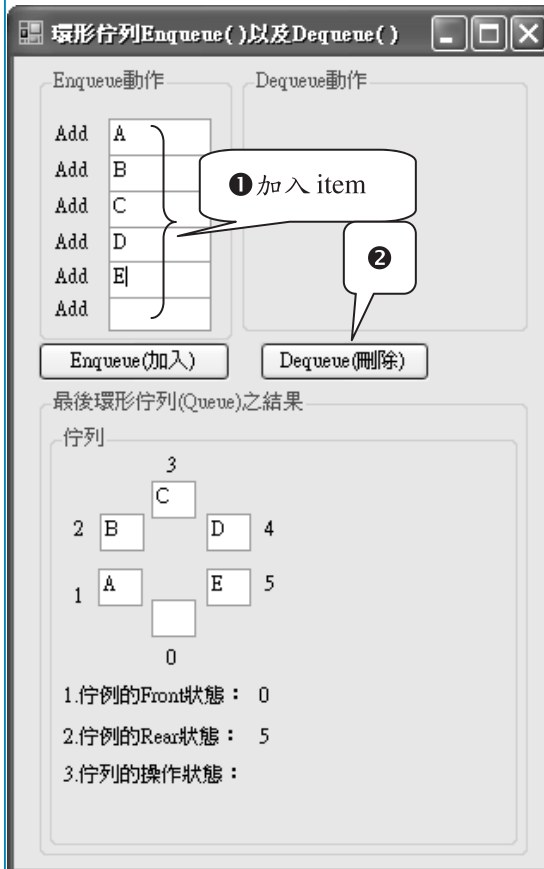


參看隨書光碟。

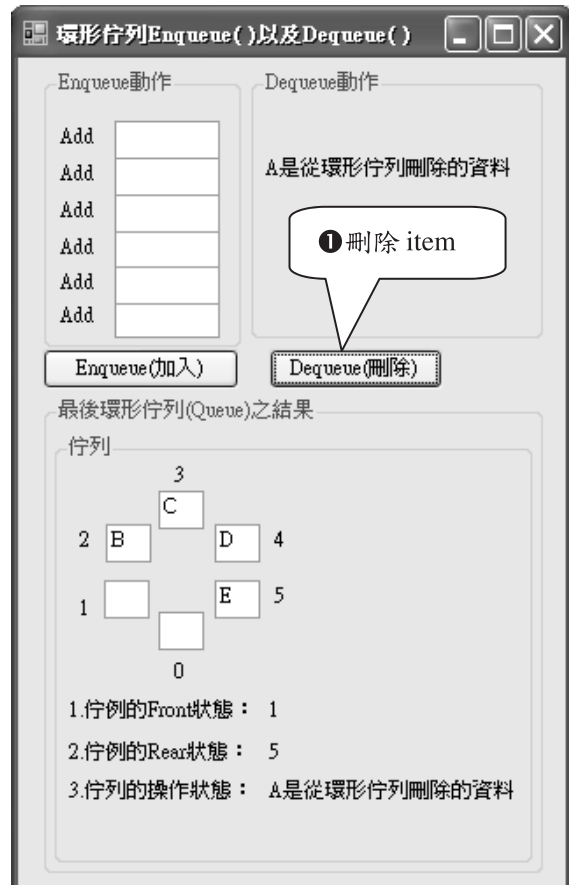
○ 老師上課動態展示

檔案請參隨書光碟。

1. 加入 5 個資料項



2. 刪除 A 資料項



單元評量

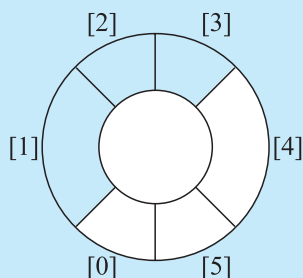
- 當採用環形佇列其佇列元素個數為 Size 時，判斷佇列為「滿」的條件是：

(A) $\text{Rear} \bmod \text{Size} = \text{Front}$ (B) $(\text{Rear}+1) \bmod \text{Size} = \text{Front}$

(C) $\text{Front} = \text{Rear}$ (D) $(\text{Front} + 1) \bmod \text{Size} = \text{Rear}$
- 當採用環形佇列其佇列元素個數為 Size 時，判斷佇列為「空」的條件是：

(A) $\text{Rear} \bmod \text{Size} = \text{Front}$ (B) $(\text{Rear} + 1) \bmod \text{Size} = \text{Front}$

(C) $\text{Front} = \text{Rear}$ (D) $(\text{Front} + 1) \bmod \text{Size} = \text{Rear}$
- 用陣列來實作環形佇列時，當佇列某一時間點資料，如下圖所示時，其中位置 [1][2][3] 代表有資料，Front 與 Rear 分別為何：



- (A) Front = 3, Rear = 3 (B) Front=3, Rear = 1
(C) Front = 0, Rear = 3 (D) Front = 3, Rear=0

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，其改良作法就是另外再增加一個變數 (Tag) 來判斷，以記錄每一次的動作。

優點

最多可以利用到 N 個空間。

缺點

Add 與 Delete 動作均須額外多加一條條件判斷式。

一、Add (item, CQueue)

將資料加入環形佇列。

(一) 演算法

```

Procedure Add(item,CQueue)
Begin
  Rear=(Rear+1) Mod n
  if (Rear=Front) And (Tag=1)
    CircularQueue Is Full; // 佇列已滿
  else
  {
    CQueue[Rear]=item;
    if(Rear=Front) then Tag=1;
  }

```

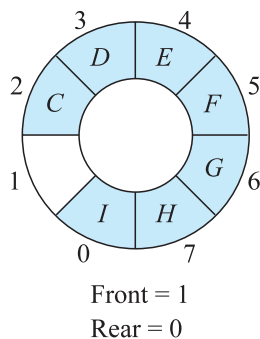
```

    }
End
End Procedure

```

(二) 實例

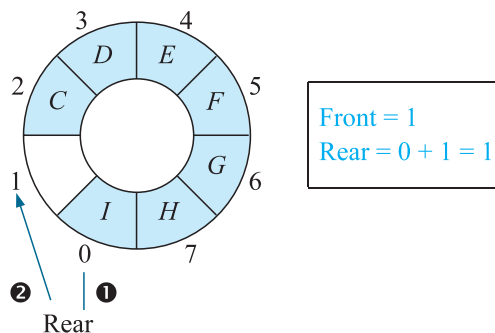
假設我們現在有一個環形佇列，共 8 個空間 (也就是編號 0~7)，並且已經存入 7 個資料項，如圖所示。接下來，我們想再加入一個資料項 *J*，請呈現運作之後的 Front 與 Rear 之值及環形佇列的狀態。



解答

當我們要加入資料項時，演算法會先計算欲加入資料的所在位置 Rear 指標，此時，Rear 指標從 0 指向 1，我們發現 Rear 指標等於 Front 指標，但是，Tag 為 0，表示環形佇列還沒有滿。

公式： $\text{Rear} = (\text{Rear} + 1) \text{ Mod } N$ // N 代表環形佇列大小



if (Rear = Front) And (Tag = 1) → 佇列已滿

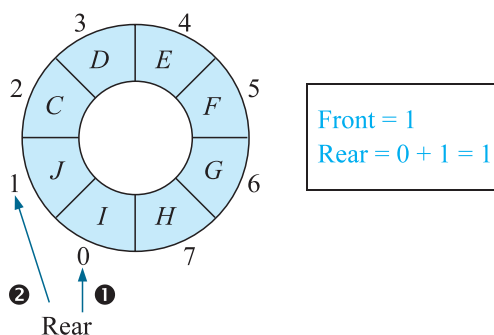
✓

✗

(Tag = 0) → 佇列未滿

所以，我們還可以再將 J 元素加入環形佇列中。此時，演算法會再判斷 $Rear$ 指標是否等於 $Front$ 指標，並且 Tag 是否等於 1，如果不是，就可以再將資料項加入佇列中，然後，再設定 Tag 等於 1。因此，可清楚得知增加一個 Tag 指標之後，就可以改善環形佇列浪費一個空間的問題。

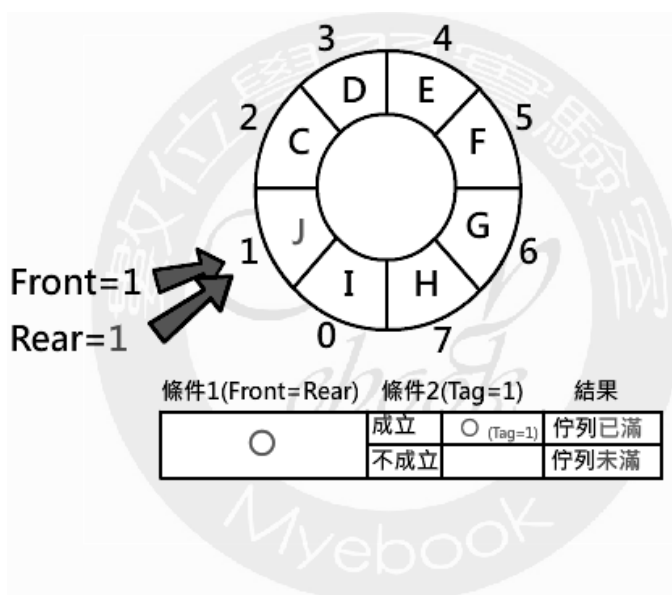
公式： $Rear = (Rear + 1) \text{ Mod } N$ // N 代表環形佇列大小



if ($Rear = Front$) And ($Tag = 0$) → 佇列未滿



($Tag = 1$) → 佇列已滿



二、Delete (item,CQueue)

從環形佇列刪除資料。

(一) 演算法

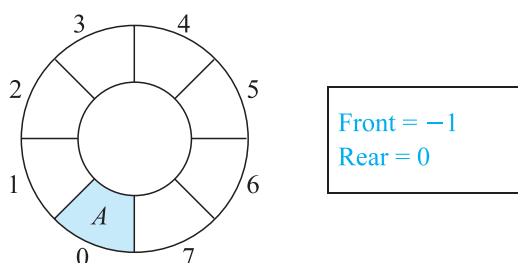
```

Procedure Delete(item, CQueue)
Begin
  if (Front=Rear) And (Tag=0)
    Circular Queue Is Empty; // 佇列為空
  else
  {
    Front=(Front+1) Mod n
    item=CQueue[Front];
    if(Rear=Front) then Tag=0;
  }
End
End Procedure

```

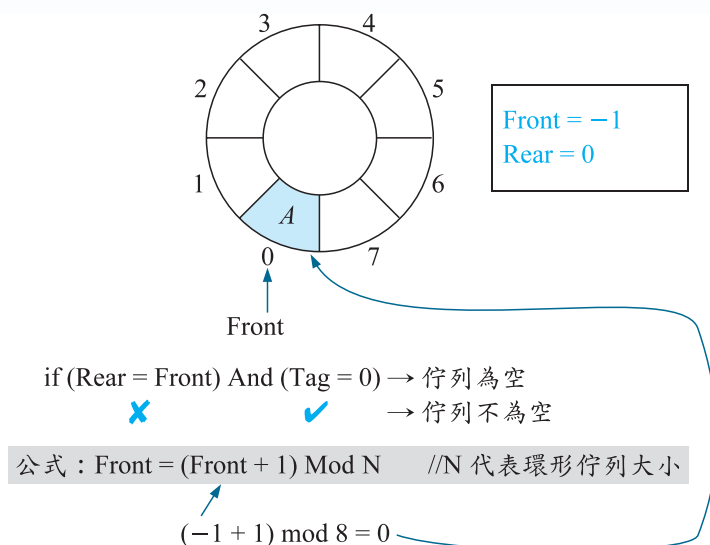
(二) 實例

假設我們現在有一個環形佇列，共 8 個空間 (也就是編號 0~7)，並且只存放 1 個資料項 A ，如圖所示。接下來，我們想取出此資料項 A ，請呈現運作之後的 Front 與 Rear 之值及環形佇列的狀態。



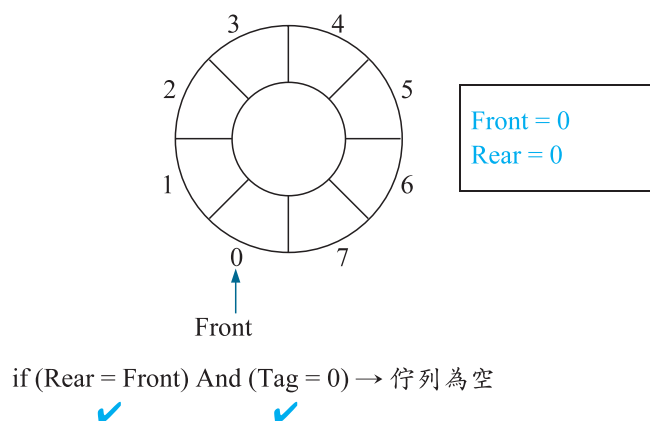
解答

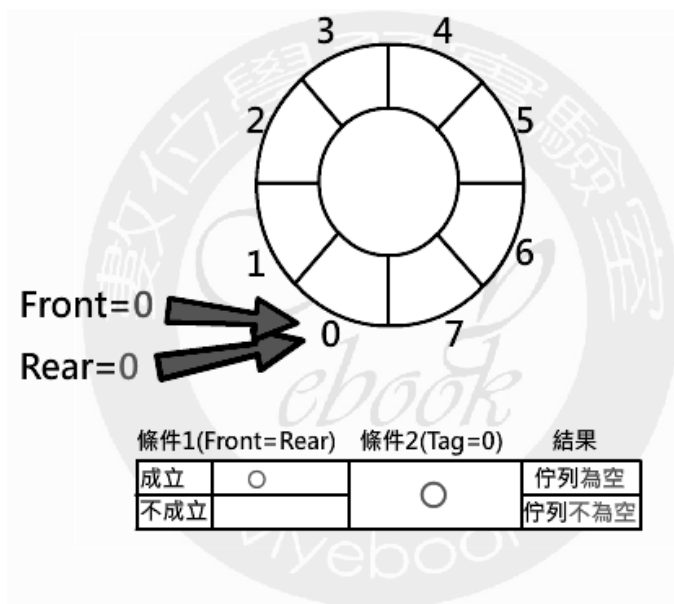
當我們要取出資料項時，演算法會先判斷 Rear 指標是否等於 Front 指標，並且 Tag 是否等於 0，因此，我們發現 Front = -1 而 Rear = 0，所以，佇列不為空。因此，先計算欲取出資料的所在 Front 指標，然後，再取出資料項 A 。



取出資料項 *A* 之後，最後，再判斷 Rear 指標是否等於 Front 指標，如果是的話，則設 Tag 為 0。因此，Rear 指標等於 Front 指標，並且設 Tag 為 0，所以，代表此佇列為空。

公式：Front = (Front + 1) Mod N //N 代表環形佇列大小





請利用 C 語言撰寫一個環形佇列的程式。



環形佇列 Enqueue() 以及 Dequeue()

程式檔名：ch4-3.2

```

01 #define NUM 8           // 定義環形佇列大小
02 void Enqueue(int);      // 宣告 Enqueue 副程式
03 int Dequeue(void);      // 宣告 Dequeue 副程式
04 void PrintQueue(void);  // 宣告列印目前環形佇列的內容之副程式
05 typedef struct queue {
06     int CQueue[NUM];
07     int Rear;
08     int Front;
09 } Queue;
10 Queue qu;
11 int main(void) // 主程式
12 {
13     int op=0,item;
14     qu.Rear = 0;
  
```

```

15  qu.Front = 0;
16  printf("===== 程式描述 =====\n");
17  printf("= 程式名稱：ch4-3.2.c                =\n");
18  printf("= 程式目的：環形佇列進行 Enqueue 以及 Dequeue    =\n");
19  printf("===== \n");
20  while(1)
21  {
22      printf("===== \n");
23      printf("= 1.Enqueue(加入)                =\n");
24      printf("= 2.Dequeue(刪除)                =\n");
25      printf("= 3. 顯示目前環形佇列的內容            =\n");
26      printf("= 4. 結束                            =\n");
27      printf("===== \n");
28      printf(" 請輸入你的操作：");
29      scanf("%d",&op);
30      switch(op)
31      {
32          case 1: printf(" 請輸入你要加入的資料：");
33                  scanf("%d",&item);
34                  Enqueue(item);    // 呼叫 Enqueue 副程式
35                  break;
36          case 2: item = Dequeue( ); // 呼叫 Dequeue 副程式
37                  if(item != -1)
38                      printf("%d 是從環形佇列刪除的資料 \n",item);
39                  break;
40          case 3: PrintQueue( );    // 呼叫列印環形佇列的內容之副程式
41                  break;
42          case 4: printf("\n");      // 結束
43                  return (0);
44      }
45  }
46  system("pause");    // 使程式暫停在執行畫面讓我們看到結果
47  return 0;
48  }
49  void Enqueue(int item) // 加入 item 到佇列中的副程式
50  {
51      qu.Rear = (qu.Rear+1)%NUM;
52      if((qu.Rear)%NUM == qu.Front) {
53          printf(" 環形佇列是滿的 !\n");
54          system("pause");

```

```
55     exit(1);
56 } else
57     qu.CQueue[qu.Rear] = item;
58 }
59 int Dequeue(void) // 從佇列中刪除 item 的副程式
60 {
61     if(qu.Front == qu.Rear) {
62         printf(" 環形佇列是空的 !\n");
63         system("pause");
64         exit(1);
65     } else {
66         qu.Front = (qu.Front+1)%NUM;
67         return qu.CQueue[qu.Front];
68     }
69 }
70 void PrintQueue(void) // 顯示目前環形佇列的內容之副程式
71 {
72     int i;
73     printf(" 目前環形佇列的內容 : ");
74     for(i=qu.Rear;i!=qu.Front;i=(i+NUM-1)%NUM)
75         printf("%d ",qu.CQueue[i]);
76     printf("\n");
77 }
```

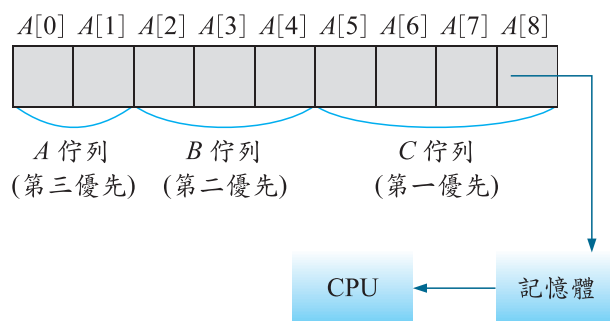
單元評量

- 1 在儲存佇列元素時，以環狀陣列來取代線性陣列，最主要可以得到下列那一項優點：
(A) 較節省儲存空間 (B) 易於修正佇列的前端索引
(C) 避免大量資料搬移 (D) 可儲存較多資料項
- 2 為了讓環形佇列可以使用 N 個空間，增加一個變數 (Tag)，因此，當佇列已滿，Tag 一般會設定為多少？
(A) Tag = -1 (B) Tag = 0 (C) Tag = 1 (D) Tag = 2

4-4 進階佇列

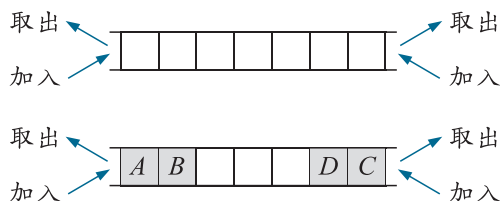
在前面單元中，我們已經學會「一般佇列」與「環狀佇列」，因此，在本單元中，我們再來介紹比較特殊的佇列。例如：優先佇列及雙向佇列。

1. 優先佇列



處理順序： C 序列 $\rightarrow B$ 佇列 $\rightarrow A$ 佇列

2. 雙向佇列



單元評量

- 基本上，我們所探討的佇列，除了一般佇列之外，還有那些佇列呢？
(A) 環狀佇列 (B) 優先佇列 (C) 雙向佇列 (D) 以上皆是
- 下列那一種佇列具有兩端都可做加入與取出資料的動作？
(A) 環狀佇列 (B) 優先佇列 (C) 雙向佇列 (D) 以上皆是

4-4.1

優先佇列

在優先佇列 (Priority Queue) 中，其元素的加入及刪除不須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博愛座等等。

○ 定義

1. 「優先佇列」是一種不必遵守「先進先出」佇列的有序串列。
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。

○ 缺點

在經常需要大量資料異動時，無法預先知道需要保留多少後端空間給插入的工作使用。

實例

假設現在有三個佇列，分別是 A 佇列、 B 佇列及 C 佇列，而這三個佇列的優先權分別為 2, 3, 4 (其中數值愈大，代表優先權愈高)，接下來，請利用「一般佇列」與「優先佇列」來比較這三個佇列的差異。

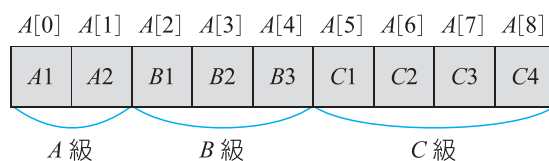
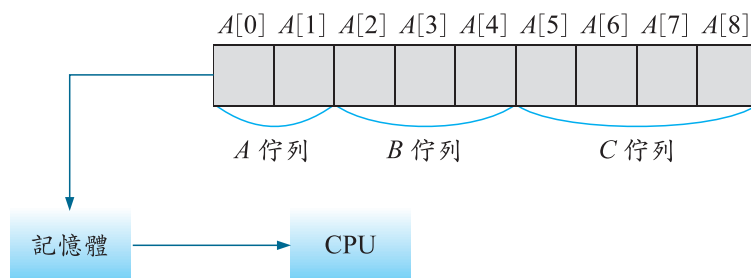


圖 4-5 優先佇列結構

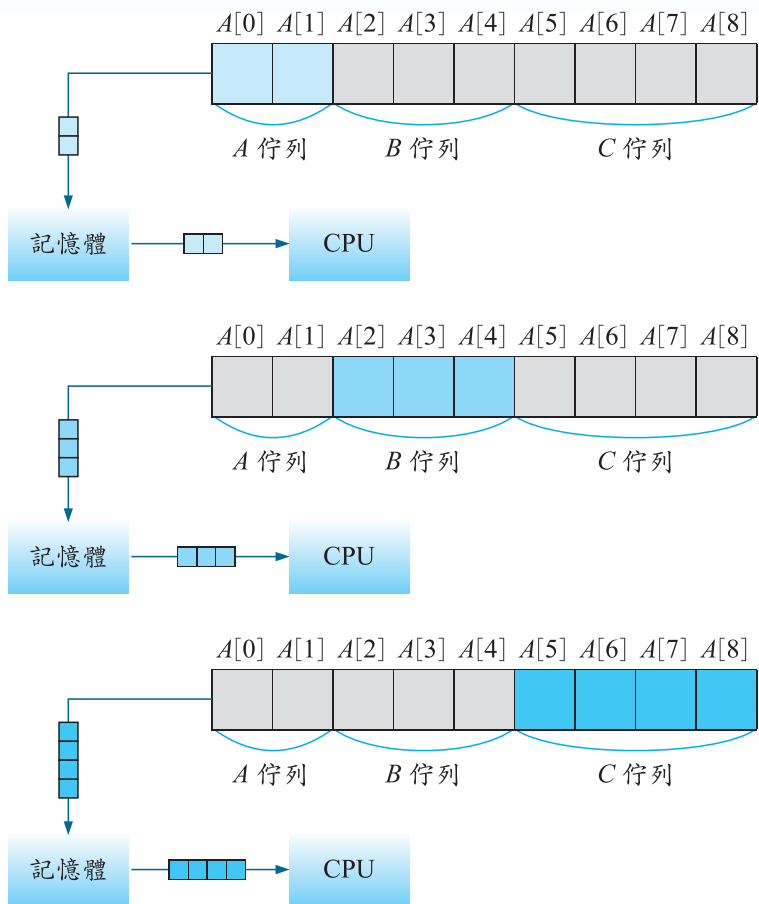
解答

首先，我們先來看上面的「一般佇列」，由於每一個佇列的優先權都相同，因此， A 佇列先到，所以，先處理，而 B 佇列排在 A 佇列的後面，所以，等 A 佇列處理完成之後，才能處理 B 佇列，並且 C 佇列是最晚到的佇列，所以，最後處理。如上圖所示。

(一) 一般佇列 (優先權相同，先到先處理)

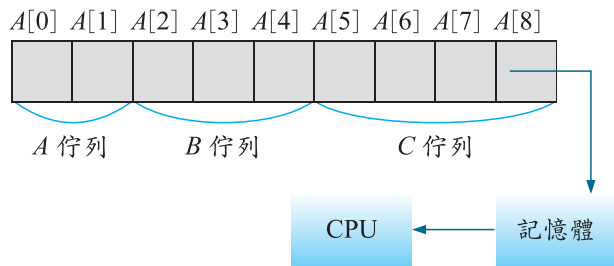


處理順序： A 序列 → B 佇列 → C 佇列

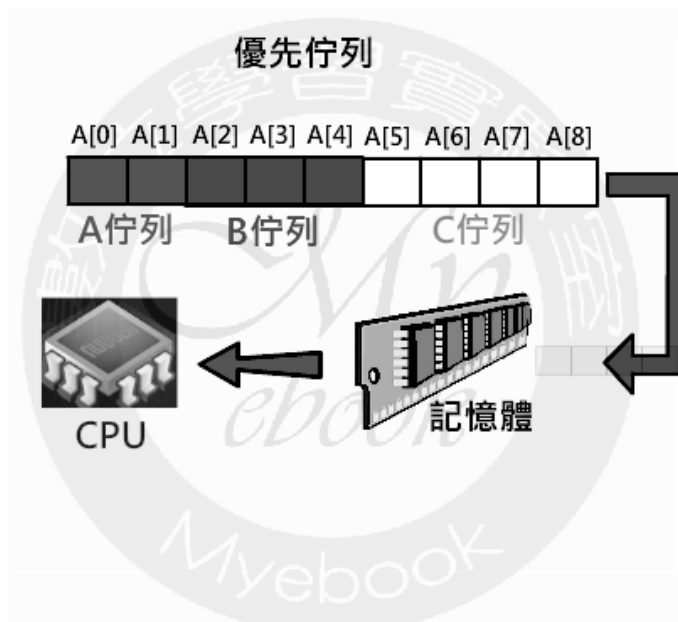


(二) 優先佇列

由於「優先佇列」不須遵守先進先出的特性，因此，雖然 C 佇列是最晚到的佇列，但優先權最高，所以，最早被處理。而 B 佇列是第二早到的佇列，並且它的優先權排在 C 佇列之後，所以，第二個被處理。 A 佇列的優先權最低，雖然是最早到的佇列，但是，最後才處理。如下圖所示。



處理順序： C 佇列 $\rightarrow B$ 佇列 $\rightarrow A$ 佇列



單元評量

- 請問在資料結構中那一種佇列在加入及刪除時不須遵守先進先出的特性？
(A) 雙向佇列 (B) 一般佇列 (C) 環形佇列 (D) 優先佇列
- 在優先佇列中，一般可以利用那兩種資料結構來解決？
(A) 陣列及矩陣 (B) 陣列及串列 (C) 串列與樹 (D) 陣列與圖形

4-4.2 雙向佇列

○ 定義

雙向佇列 (Double-Ended Queue: 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，不像堆疊的 LIFO 或佇列的 FIFO。亦即它是一種前後兩端都可輸入或取出資料的有序串列。如圖 4-6 所示。

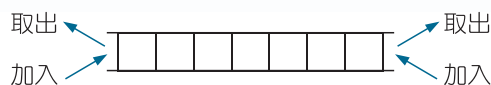


圖 4-6 雙向佇列 (Deque)

實例

假設我們循序輸入一串數字：1, 2, 3, 4, 5, 6, 7，請問是否可以輸出 5174236 呢？

解答

循序輸入一串數字：1, 2, 3, 4, 5, 6, 7，順序如下：

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

輸出：先輸出 5，再輸出 1，又輸出 7，但是，下一個輸出不是“2”就是“6”，無法輸出“4”，因此，在循序輸入 1, 2, 3, 4, 5, 6, 7 的情況下，無法得到 5174236 的輸出結果。

動書圖解

參看隨書光碟。

單元評量

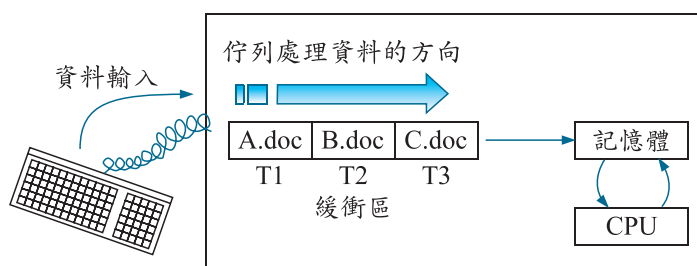
- 利用雙向佇列循序輸入 1, 2, 3, 4, 5, 6, 7，試問絕不可能得到哪種輸出？
(A) 7, 6, 1, 2, 5, 3, 4 (B) 7, 1, 2, 6, 3, 4, 5 (C) 1, 2, 7, 3, 6, 5, 4 (D) 1, 7, 4, 3, 2, 6, 5
- 利用雙向佇列循序輸入 1, 2, 3, 4, 5, 6 及 7，則下列那些結果為可能的輸出排列？
(A) 5174236 (B) 5172346 (C) 5172436 (D) 5173426

4-5 佇列在電腦資料處理的應用

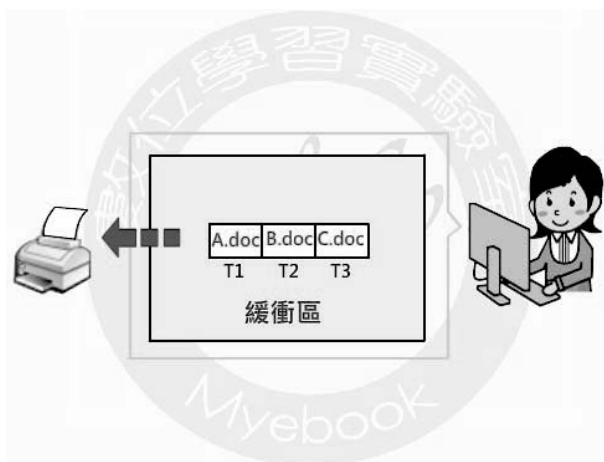
佇列在電腦科學中的應用相當廣泛，例如電腦模擬 (Computer Simulation)、作業系統 (OS)、電腦資源的分配都是利用佇列來表示。雖然佇列與堆疊都可以利用陣列來實作，但是，佇列在電腦中的應用與堆疊大不相同，因此，佇列大多運用於硬體流程的控制。並且由於佇列具有先進先出 (First In First Out) 的特性，所以，經常被運用在電

腦的作業系統，以安排電腦執行工作 (Job) 的優先順序。

在電腦的硬體結構中，最常應用於緩衝區上，舉例來說，有時候電腦在執行某一龐大的應用程式時，系統反應通常變得非常遲頓，對於使用者由鍵盤輸入的資料，常常會有「漏接」的情況，因此設計師在電腦的記憶體上，規劃出緩衝區，一旦使用者於鍵盤上輸入資料時，便會先儲存於緩衝區上，等 CPU 不忙碌時，再一個個讀至記憶體處理，因此比較不會有資料「漏接」的情況，而緩衝區的結構本身就是一種「先進先出」的佇列結構，如下圖所示。



資料來源：<http://www.chwa.com.tw/TResource/HS/book2/ch7/7-3.htm>



單元評量

- 佇列在電腦科學中的應用相當廣泛，下列何者是佇列的應用？
(A) 電腦模擬 (B) 作業系統 (C) 電腦資源的分配 (D) 以上皆是
- 假設現在有一位同學想列印三個檔案的文件資料，接下來進行一連列的動作如下：

T1 時間：按列印 A.doc

T2 時間：按列印 B.doc

T3 時間：按列印 C.doc

請問以上三個檔案在列印機佇列中的順序為何？

- (A)

A.doc	B.doc	C.doc	---- →	記憶體	---- →	CPU
-------	-------	-------	--------	-----	--------	-----
- (B)

C.doc	B.doc	A.doc	---- →	記憶體	---- →	CPU
-------	-------	-------	--------	-----	--------	-----
- (C)

B.doc	A.doc	C.doc	---- →	記憶體	---- →	CPU
-------	-------	-------	--------	-----	--------	-----
- (D)

C.doc	A.doc	B.doc	---- →	記憶體	---- →	CPU
-------	-------	-------	--------	-----	--------	-----