CHAPTER

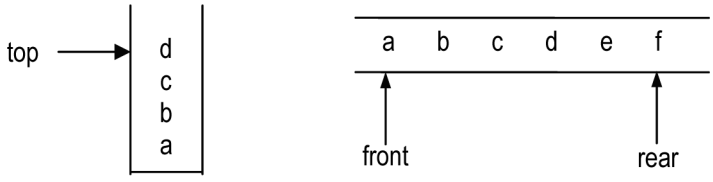
3

堆疊與佇列

3.1 堆疊和佇列基本觀念

堆疊是一有序串列(order list)，或稱線性串列(linear 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)。加入一資料於堆疊，此動作稱為加入(push)，與之相反的是從堆疊中刪除一資料；此動作稱為彈出(pop)。由於堆疊具有先被推入的資料，最後才會被彈出的特性，所以我們稱它為先進後出(First In Last Out, FILO)或後進先出(Last In First Out, LIFO)串列。

佇列(queue)亦是屬於線性串列，與堆疊不同的是加入和刪除不在同一端，刪除的那一端稱為前端(front)，而加入的那一端稱為後端(rear)。由於佇列具有先進先出(First In First out , FIFO)的特性，因此稱佇列為先進先出或後進後出串列。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)。堆疊、佇列的表示法，如圖 3-1 之(a)、(b)所示。



(a)堆疊　　(b)佇列

圖 3-1　堆疊與佇列

其中(a)為一堆疊，它有如一容器，且有最大的容量限制，每次加入的資料，都會往上堆，好比疊書本一般。top 指向堆疊的最上端，加入時 top 會加 1，而刪除時 top 會減 1。

而(b)為一佇列，有如一排隊的隊伍，其中 front 所指的是隊伍的前端，而 rear 所指的是隊伍的後端。這好比您排隊上車，新來的人會排在隊伍的後端。上車的順序是從隊伍的前端開始，這就是佇列的特性。

練習題

請你舉一些有關堆疊和佇列的例子。

3.2 堆疊的加入與刪除

在堆疊的運作上，加入時必需注意是否會超出堆疊的最大容量，而在刪除時必需判斷堆疊是否還有資料。一般的作法是，利用一變數 top 來輔助之。當 push 一個資料時，將 top 加 1; 反之，pop 一個資料，將 top 減 1。我們可以利用串列來表示堆疊，如 st[MAX]，表示堆疊 st 的最大容量為 MAX。top 的初值設為 –1。

3.2.1 堆疊的加入

堆疊的加入應注意堆疊是否已滿，若未滿，則將輸入的資料 push 到堆疊的上方。其 Python 的片段程式如下：

Python 片段程式》 堆疊的加入函數

```
def  push_f():
    if top >= MAX-1:
        print("\n   堆疊是滿的!")
    else:
        top += 1
        st[top] = input("\n  請輸入一筆資料(字串的格式)： ")
    print()
```

》 程式解說

st[ ]用來表示堆疊串列，MAX 為堆疊所能容納的最大資料個數，top 為目前堆疊最上面資料的註標。當 top >= MAX - 1 時，表示堆疊已滿。注意! 堆疊是從 st[0]開始，st[MAX-1]結尾，所以條件式為 top >= MAX - 1，而非 top >= MAX。若堆疊還有空間，則將 top 加 1，並要求使用者輸入資料，直接存入堆疊 st[top]中。

3.2.2 堆疊的刪除

從堆疊的刪除資料時，應注意堆疊是否為空的。其片段程式如下：

Python 片段程式》 堆疊的刪除函數

```
def pop_f():
     if top < 0:
         print("\n     堆疊是空的!")
     else:
         print("\n     %s 已被刪除!" % st[top])
         top -= 1
     print()
```

》程式解說

當 top < 0，表示堆疊是空的。因為 st[0]是堆疊最底下的資料，而非 st[1]，所以條件式不是 top <= 0，而是 top < 0；若堆疊中還有 item，則輸出 st[top]，並將 top 減 1。

有關堆疊的加入和刪除之程式實作，請參閱3.5節。

練習題

若將上述堆疊的加入和刪除函數中 top 的初值設為 0，試問上述的加入和刪除之片段程式應如何修改。

3.3 佇列的加入與刪除

佇列有兩端，分別是 front 和 rear 端。佇列從 rear 端加入資料，而從 front 端刪除。加入時要注意是否會超出最大的容量。由於我們設定 rear 變數的初值為 –1，所以要先將 rear 加 1 之後，再加入資料。front 變數的初值設定為 0，因此，若不是空佇列，則刪除的動作是先刪除資料，之後再將 front 加 1。

3.3.1 佇列的加入

佇列的加入是作用在 rear 端，其片段程式如下：

Python 片段程式》 佇列的加入函數

```
def enqueue_f():
    if rear >= MAX-1:
        print("\n  此佇列已滿的!")
    else:
        rear += 1
```

```
        q[rear] = input("\n  請輸入一筆資料(字串格式): ")
        print()
```

》程式解說

以 q[ ] 串列表示一佇列，MAX 為佇列所能容納的最大資料個數，rear 為佇列最後一個資料項目的註標。由於佇列是從 q[0]開始，q[MAX-1]結尾，所以當 rear 大於等於 MAX -1 時，表示佇列已滿(注意!不是 rear 大於等於 MAX)；若佇列中還有空間，則執行 rear++，並要求使用者輸入資料，將它存放於 q[rear]。

3.3.2 佇列的刪除

佇列的刪除是作用在 front 端，其片段程式如下：

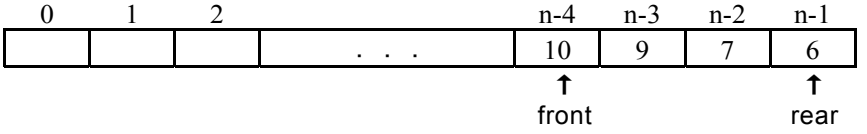
Python 片段程式》 佇列的刪除函數

```
def dequeue_f():
    if front > rear:
        print("\n       此佇列是空的!")
    else:
        print("\n     %s 已被刪除!" % q[front])
        front += 1
    print()
```

》程式解說

以 q[ ] 串列表示一佇列，front 是佇列第一個資料項目的註標，而 rear 是最後一個資料項目的註標。當 front > rear 時，表示佇列是空的，此時無法做刪除工作；若佇列不是空的，則輸出 q[front]後，再執行 front++。

若佇列的表示方式是 Q(0: n-1) 時，常常會發生佇列前端還有空位，但要加入資料時，卻產生佇列已滿，因為 rear 已大於等於 n-1，如下圖所示：



此時若要加入 8，依照上述的片段程式，卻產生額滿的現象，為了解決此一問題，佇列常常以環狀佇列(circular queue)來表示。圖 3-2 為一環狀佇列 CQ(0 : n-1)。

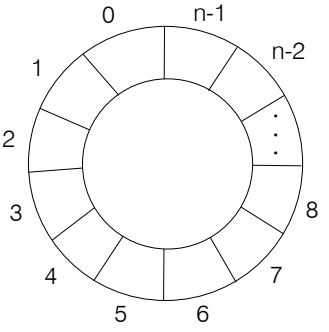


圖 3-2 環狀的佇列

3.3.3 環狀佇列的加入

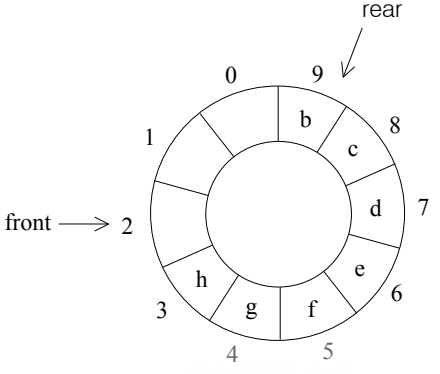
環狀佇列開始的時候，將 front 與 rear 之初值設為 MAX–1。

Python 片段程式》 環狀佇列的加入函數

```
# 加入函數
def enqueue_f():
     rear = (rear + 1) % MAX
     if front == rear:
         if rear == 0:
             rear = MAX - 1
         else:
             rear = rear - 1
         print("\n\n 此佇列已滿!")
     else:
         cq[rear] = input("請輸入一筆資料(字串格式): ")
```

》程式解說

以 cq[ ] 串列表示一環狀佇列，其中敘述 rear = (rear + 1) % MAX 主要的用意是讓新加入的資料可以利用空白的空間。若有一環狀佇列經過一些加入和刪除的動作後之圖形如下：



此時若加入一資料，則 rear 會指向 CQ[0]的位置，而不會產生額滿的現象。此片段程式是利用 if (front == rear)來判斷環狀佇列是否已滿。

3.3.4 環狀佇列的刪除

Python 片段程式》 環狀佇列的刪除函數

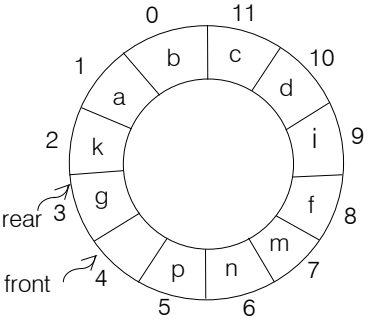
```
# 刪除函數
def dequeue_f():
     if front == rear:
         print("\n 此佇列是空的!")
     else:
         front = (front + 1) % MAX
         print("\n\n%s 已被刪除!" % cq[front])
```

》程式解說

以 cq[ ]串列表示一環狀佇列，MAX 為 cq 可容納的最大資料個數，front 為佇列前端，rear 為後端。若 rear == front 時，則印出此佇列是空的! 的訊息，表示環狀佇列中無資料。否則利用 front = (front + 1) % MAX; 敘述來取得刪除的資料項目。

必須要注意的是，環狀佇列的加入必需先找一位置，然後再做判斷，而刪除則是先做判斷，然後再找位置。還有一點要注意的是，在環狀佇列中會留一個空的位置，此乃為了辨別是否已額滿或空而設的。

1. front 在 cq[4]，而 rear 在 cq[2]。
2. 加一資料 g，此時 (2+1) % 12 = 3 ，因此 rear 指向 cq[3]的地方，如下圖所示：



3. 若再加一資料時，rear 指向 cq[4]；此時 rear == front， 因此輸出 Queue is full!，但是從圖得知 cq[4]是空的。若繼續使用此空間，則在下一次要刪除資料時，會產生問題，根據刪除的片段程式，當 front == rear 時，會顯示 Queue is empty!，這與事實不符。

若非用此空間不可，則必需另加一條件來輔助之，此處是利用 tag 變數是否等於 0 或 1 來輔助。環狀佇列開始時，front 和 rear 都設為 MAX-1，而且 tag 設為 0。請看以下的片段程式：

Python 片段程式》 環狀佇列的加入函數－使用 TAG 變數

```
def enqueue_f():
    if front == rear and tag == 1:
        print("\n\n    此佇列已滿!!!")
    else:
        rear = (rear + 1) % MAX
        cq[rear] = input("\n    請輸入一筆資料(字串格式): ")

        if front == rear:
            tag = 1
    print()
```

》 程式解說

當 front == rear 且 tag == 1 的情況下，表示佇列已滿，無法新增 item；若不是，則以(rear + 1) % MAX 取得新的 rear 值(當原來的 cq[rear]為佇列中最後一個資料時，則(rear + 1) % MAX 會使 rear 值為 0，將資料置於 cq[0])。新增後，若 front 與 rear 相等，則表示佇列已滿，並將 tag 設定為 1。接著來看刪除的片段程式。

Python 片段程式》 環狀佇列的刪除函數－使用 TAG 變數

```
def dequeue_f():
     if front == rear and tag == 0:
         print("\n    此佇列是空的!")
     else:
         front = (front + 1) % MAX
         print("\n\n    %s 已被刪除!!" % cq[front])
         if front == rear:
             tag = 0
         print()
```

》 程式解說

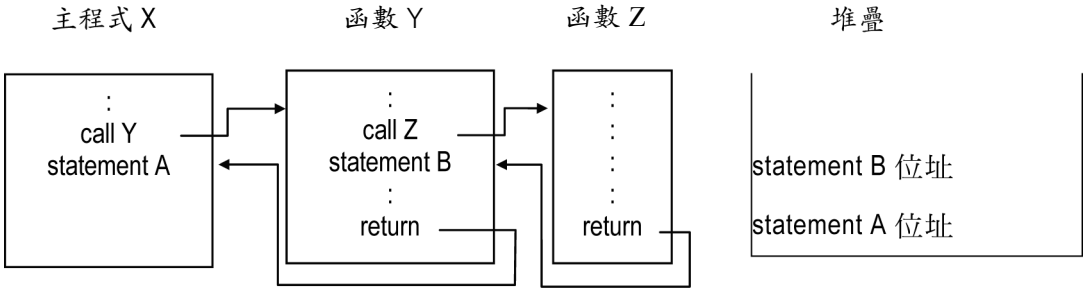
當 front == rear 且 tag == 0 的情況下，表示佇列為空的，無法做刪除工作；若佇列中還有資料，則以(front + 1) % MAX 取得 front 的新值(意義與佇列加入時之 rear 相同)，輸出 cq[front]。此時若 front 與 rear 相等，則將 tag 設定為 0。

此一演算法與不加 TAG 變數的演算法，旨在說明時間和空間之間的取捨(trade off)。為了能充分使用空間，我們加入了 TAG 是否為 1 或 0 的判斷，使得時間花得比較多，但空間能百分之百的使用; 而不加 TAG 變數的判斷，則執行時間較快，但空間無法百分之百的使用。

有關環狀佇列的加入和刪除之程式實作，請參閱 3.5 節。

3.4 堆疊的應用

由於堆疊具有先進後出的特性，因此凡是具有此性質的問題，皆可使用堆疊來解決，例如函數的呼叫。假設有一主程式 X 呼叫函數 Y，此時將 statement A 的位址加入(push)堆疊，在函數 Y 呼叫函數 Z，將 statement B 的位址加入堆疊。當函數 Z 執行完畢後，從堆疊彈出(pop)返回函數 Y 的 statement B 位址，而當函數 Y 執行完畢後，再從堆疊彈出返回主程式 X 的位址，如下圖所示。



3.4.1 中序表示式轉為後序表示式

堆疊除了可應用於上述的函數呼叫外，還可應用於如何將算術運算式由中序表示式(infix expression)轉換為後序表示式(postfix expression)。一般的算術運算式皆是中序表示式，亦即運算子(operator)置於運算元(operand)的中間(假若只有一個運算元，則運算子置於運算元的前面)。而後序表示式則是將運算子置於其對應運算元後面。我們所熟悉的數學運算式 A * B // C，就是中序表示式，而此運算式的後序表示式為 AB * C //。

為什麼需要由中序表示式變為後序表示式呢？因為運算子有優先順序與結合性，以及有括號先處理的問題，為了方便處理，通常會將中序表示式，先轉為後序表示式。

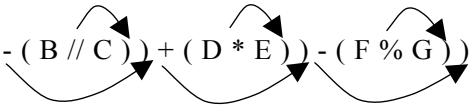
如何將中序表示式轉為後序表示式，可依下列三步驟進行即可：

1. 將中序表示式加入適當的括號，此時須考慮運算子的運算優先順序。
2. 將所有的運算子移到它所對應右括號的右邊。
3. 將所有的括號去掉。

如將 A * B // C 化為後序表示式：

1. ((A * B) // C)
2. ((A * B) // C) => ((AB) * C) //
3. AB * C //

再看一例，將 A - B // C + D * E - F % G 轉為後序表示式

1. (((A - (B // C)) + (D * E)) - (F % G))
2. 
3. ABC // - DE * + FG % -

算術運算式由中序表示式轉為後序表示式，通常是利用堆疊來完成的。首先要了解算術運算子的 in-stack 與 in-coming 的優先順序。

符號	in-stack priority	in-coming priority
)	－	－
＋(正), －(負), ！	3	4
＊ , //, %	2	2
＋(加), －(減)	1	1
(	0	4

開始時堆疊是空的，我們將運算式中的運算子和運算元看成是 token。當 token 是運算元，則直接輸出; 反之，若 token 是運算子而且此 token 的 in-coming priority(ICP) 小於或等於堆疊上端的運算子之的 in-stack priority(ISP)，則輸出堆疊中運算子，直到 ICP > ISP，再將此 token 加入於堆疊。我們以 A + B * C 中序表示式來說明如何將此中序表示式轉為後序表示式，其過程如下：

token	stack	output	說明
none		none	
A		A	由於 A 是運算元，故直接輸出。
+		A	
B		AB	B 是運算元，故直接輸出。
*		AB	由於 * 的 in-coming priority 大於 + 的 in-stack priority
C		ABC	C 是運算元，故直接輸出。
none		ABC*	pop 堆疊頂端的資料 *
none		ABC*+	再 pop 堆疊頂端的資料+

再來看一範例，若有一中序表示式為A * (B + C) * D

token	stack	output	說明
none		none	
A		A	由於 A 是運算元，故直接輸出。
*	*	A	
(	(*	A	由於(的 in-coming priority 大於 * 的 in-stack priority
B	(*	AB	B 是運算元，故直接輸出。
+	+ (*		+ 的 in-coming priority 大於 (的 in-stack priority
C	+ (*	ABC	C 是運算元，故直接輸出。
)	*	ABC+	) 的 in-coming priority 小於+ 的 in-stack priority，故輸出 +，之後再去掉 (
*	*	ABC+*	此處輸出的 * ，是在堆疊裏的 *
D	*	ABC+*D	D 是運算元，故直接輸出。
none		ABC+*D*	輸出堆疊中的 *

有關將運算式由中序表示式轉為後序表示式之程式實作，請參閱 3.5。

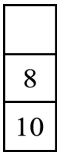
3.4.2 如何計算後序表示式

當我們將中序表示式轉換為後序表示式後，就可以很容易將此運算式的值計算出來，其步驟如下：

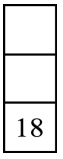
1. 將此後序表示式以一字串表示之。
2. 每次取一個 token，若此 token 為一運算元，則將它 push 到堆疊。若此 token 為一運算子，則自堆疊 pop 出二個運算元，並做適當的運算。若此 token 為 '\0' ，則跳到步驟 4。
3. 將步驟 2 的結果，push 到堆疊，之後再回到步驟 2。

4. 彈出堆疊的資料，此資料即為此後序表示式計算的結果。我們以下例說明之，如有一中序表示式 10+8–6*5，已轉為後序表示式 10 8 + 6 5 * –，接著利用上述的規則執行。

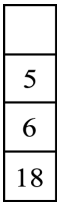
(1) 因為 10 為一運算元，故將它 push 到堆疊。同理 8 也是，故堆疊有 2 個資料分別為 10 和 8



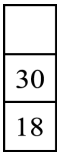
(2) 之後的 token 為 +，故 pop 堆疊的 8 和 10 做加法運算，結果為 18，將 18 push 到堆疊



(3) 接下來，將 6 和 5 push 到堆疊



(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



(5) 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) 對於 + 和 *，此順序並不影響，但對 – 和 // 就非常重要。

(6) 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。

練習題

1. 將下列中序表示式轉換為後序表示式

(1) a > b && c > d && e < f

(2) (a + b) * c // d + e – 8

2. 有一中序表示式如下：

5//3 * (1–4) + 3 – 8

請先將它轉換為後序表示式，再求出其結果為何。

3.5 程式實作

(一) 堆疊的運作

Python 程式語言實作》使用堆疊新增、刪除與顯示資料

```
01  # 使用堆疊處理資料——新增、刪除、輸出
02  # File Name： Stack.py
03  # Version 3.0, March 13th, 2017
04
05  import sys
06
07  MAX = 10
08  st = [''] * MAX
09  top = -1
10
11  def push_f(): # 新增函數
12      global MAX
13      global st
14      global top
15
16      if top >= MAX - 1: # 當堆疊已滿，則顯示錯誤
17          print('\n   堆疊是滿的！')
18      else:
19          top += 1
20          st[top] = input('\n 請輸入一筆資料（字串的格式）：')
21      print()
22
23  def pop_f(): # 刪除函數
24      global st
25      global top
26
27      if top < 0: # 當堆疊沒有資料存在，則顯示錯誤
28          print('\n   堆疊是空的！')
29      else:
30          print('\n   %s 已被刪除！' % st[top])
31          top -= 1
32      print()
33
34  def list_f(): # 輸出函數
35      global st
36      global top
37
38      count = 0
```

```
39
40      if top < 0:
41          print('\n    堆疊是空的！')
42      else:
43          print('\n\n 堆疊有下列的資料：')
44          print('--------------------')
45          i = top
46          while i >= 0:
47              print(' ', end = '')
48              print(st[i])
49              count += 1
50              i -= 1
51          print('--------------------')
52          print(' 堆疊共有%d 筆資料。\n' % count)
53      print()
54
55  def main(): # 主函數
56      option = 0
57
58      while True:
59          print('***** 堆疊的選單 *****')
60          print('      1. Insert      ')
61          print('      2. Delete      ')
62          print('      3. List        ')
63          print('      4. Exit        ')
64          print('**********************')
65
66          try:
67              option = eval(input('    請選擇您要執行的項目：'))
68          except ValueError:
69              print('Not a correct number.')
70              print('Try again\n')
71
72          if option == 1:
73              push_f() # 新增函數
74          elif option == 2:
75              pop_f() # 刪除函數
76          elif option == 3:
77              list_f() # 輸出函數
78          else:
79              sys.exit(0)
80
81  main()
```

輸出結果

```
***** 堆疊的選單 *****
      1. Insert
      2. Delete
      3. List
      4. Exit
*********************
   請選擇您要執行的項目：1

 請輸入一筆資料（字串的格式）：iPhone

***** 堆疊的選單 *****
      1. Insert
      2. Delete
      3. List
      4. Exit
*********************
   請選擇您要執行的項目：1

 請輸入一筆資料（字串的格式）：iPod

***** 堆疊的選單 *****
      1. Insert
      2. Delete
      3. List
      4. Exit
*********************
   請選擇您要執行的項目：3


 堆疊有下列的資料：
-------------------
 iPod
 iPhone
-------------------
 堆疊共有 2 筆資料。


***** 堆疊的選單 *****
      1. Insert
      2. Delete
      3. List
      4. Exit
*********************
   請選擇您要執行的項目：1

 請輸入一筆資料（字串的格式）：iMac

***** 堆疊的選單 *****
      1. Insert
```

```
      2. Delete
      3. List
      4. Exit
*********************
   請選擇您要執行的項目：3


 堆疊有下列的資料：
-------------------
 iMac
 iPod
 iPhone
-------------------
 堆疊共有 3 筆資料。


***** 堆疊的選單 *****
      1. Insert
      2. Delete
      3. List
      4. Exit
*********************
   請選擇您要執行的項目：2

   iMac 已被刪除！

***** 堆疊的選單 *****
      1. Insert
      2. Delete
      3. List
      4. Exit
*********************
   請選擇您要執行的項目：3


 堆疊有下列的資料：
-------------------
 iPod
 iPhone
-------------------
 堆疊共有 2 筆資料。


***** 堆疊的選單 *****
      1. Insert
      2. Delete
      3. List
      4. Exit
*********************
   請選擇您要執行的項目：4
```

```
 +：加            -：減
 (：左括號        )：右括號
*****************************
請輸入一中序運算式：(a+b)*c/d+e*f
Postfix expression: ab+c*d/ef*+
```

》程式解說

在程式中先設定一堆疊 stack_t[ ]來存放從運算式 infix_q[ ]中讀入運算子或運算元，並以 for 迴圈來控制每一個運算子或運算元的讀入動作，並於堆疊底下置入'#'表示結束，共有四種情況。

1. 輸入為)，則輸出堆疊內之運算子，直到遇到 (為止。
2. 輸入為 #，則將堆疊內還未輸出的所有運算子輸出。
3. 輸入為運算子，其優先權若小於 stack_t[top] 中的運算子，則將 stack_t[top]輸出，若優先權大於等於 stack_t[top] 存放的運算子，則將輸入之運算子放入堆疊中。
4. 輸入為運算元，則直接輸出。

其中運算子的優先權是以以下兩個串列來建立的：

infix_priority = [] 為在運算式中的優先權；
stack_priority = [] 為在堆疊中的優先權；

運算子優先權的比較是由 compare 函數來做，在代表優先權的兩個串列中，將每一個運算子在串列中所在的註標值除以 2，即為運算子的優先權，如 infix_priority[ ]中，) 為 infix_priority[1]，其優先順序為 1 // 2 等於 0；+為 infix_priority[2]，其優先順序為 2 // 2 等於 1，依此類推。所以在 compare 函數中，先找到兩運算子在串列中的註標值，再分別除以 2 來比較，即可得知優先順序孰高。

3.6 動動腦時間

1. 將下列中序(infix)運算式轉換為前序(prefix)與後序(postfix)運算式。[3.4]

   (1) A * B % C

   (2) -A + B - C + D

   (3) A // -B + C

   (4) (A + B) * D + E // (F + A * D) + C

   (5) A // (B * C) + D * E - A * C

   (6) A && B || C || ! (E > F)

(7) A // B * C + D % E - A // C * F

(8) (A * B) * (C * D) % E * (F - G) // H - I - J * K // L

(9) A * (B + C) * D

提示：前序與後序的操作二者剛反如

A + B * C → 後序為 ((A + B) * C) → AB + C *

前序為 ((A + B) * C) → * + ABC

2. 有一鐵路交換網路(switching network)如下：[3.1]

火車廂置於右邊，各節皆有編號如 1，2，3，……，n，每節車廂可以從右邊開進堆疊，然後再開到左邊，如 n = 3，若將 1，2，3 按順序開入堆疊，再駛到左邊，此時可得到 3，2，1 的順序。請問

(1) 當 n = 3 及 n = 4 時，分別有那幾種排列的方式？那幾種排序方式不可能發生？

(2) 當 n = 6 時，325641 這樣的排列是否可能發生？那 154623 的排列又是如何？

(3) 找出一公式，當有 n 個車廂時，共有幾種排列方式。

3. 在 InfixToPostfix.Python 的程式實作中，若輸入 –a*b+c，則會出現錯誤的答案，試將此程式加以修整之。並加一詢問使用者是否要繼續執行此程式的功能。