Chapter 4 鏈結串列

- 4.1 單向鏈結串列
- 4.2 環狀串列
- 4.3 雙向鏈結串列
- 4.4 鏈結串列的應用

4.1 單向鏈結串列

- 為何使用鏈結串列(linked list)？
  - 為了避免以陣列方式來存放資料時，在插入(insert)或刪除(delete)某一節點所遇到的困難
  - 節省配置的記憶體空間
- 鏈結串列 vs. 陣列
  - 在加入和刪除時利用指標(pointer)，因此比陣列來得簡單
  - 鏈結串列在搜尋上所花費的時間會比陣列來得久

4.1 單向鏈結串列

- 假設鏈結串列中每個節點有姓名(name)、分數(score)及指向下一個節點的指標(next)，若將節點結構定義為Node 型態，則宣告的方式如下：

```
class Node  {
   public int data;     // 分數
   public Node next;    // 指向下一個節點的指標
}
```

4.1 單向鏈結串列

- 這是一個很典型的單向鏈結串列(Single linked list)，如串列 A = {98, 76}，其圖形如下：

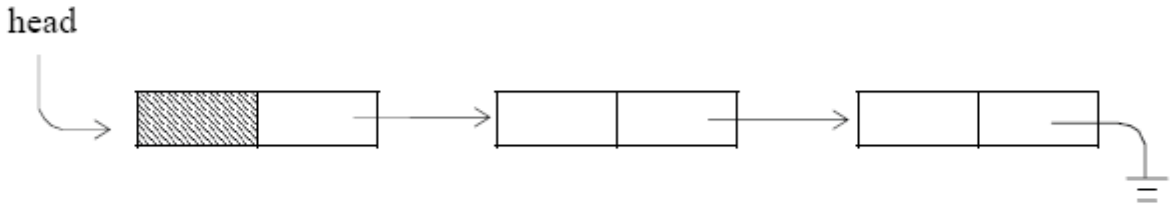
- 假設鏈結串列的第一個節點(亦即head 所指向的節點)的score 欄位不放任何資料。讓我們來看看鏈結串列的加入與刪除的動作，而這些動作可能作用於前端或尾端或某一特定的節點。

4.1 單向鏈結串列

4.1.1 加入動作

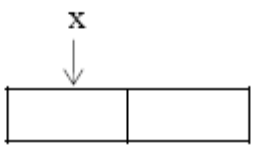
1. 加入一節點於串列的前端

假設有一串列如下：



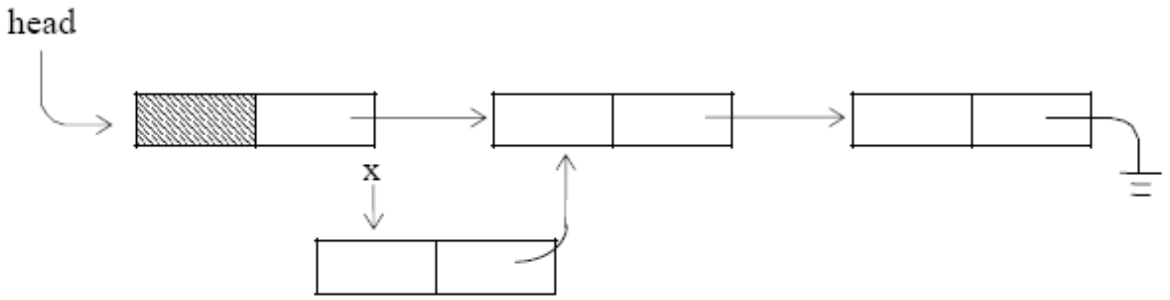
有一節點 x 將加入於串列的前端，執行的步驟和示意圖如下：

(1) x = new Node ();

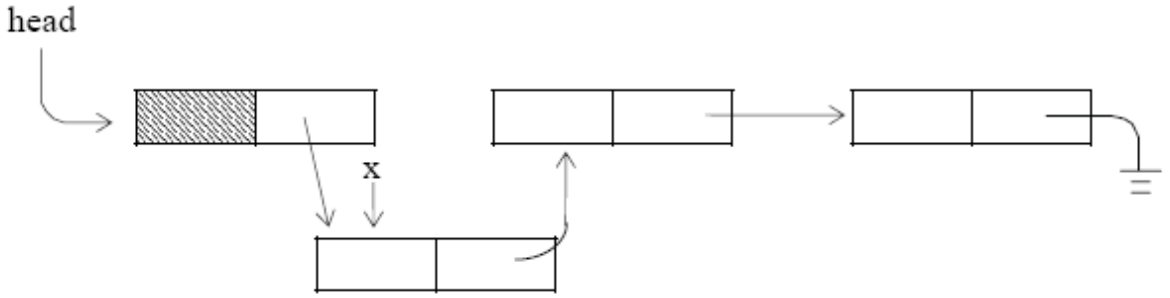


4.1 單向鏈結串列

(2) x.next = head.next;



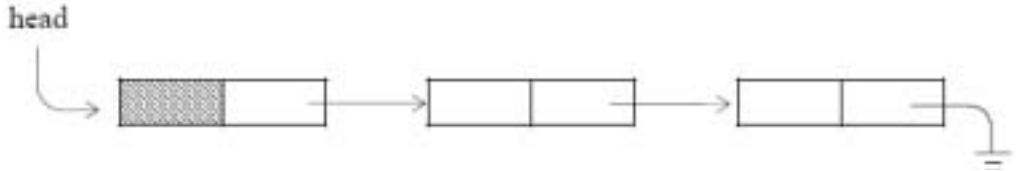
(3) head.next = x



4.1 單向鏈結串列

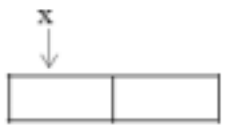
2. 加入一節點於串列的尾端

假設有一串列如下：

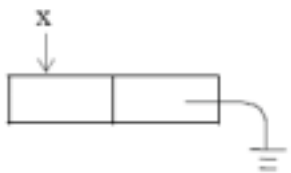


有一節點 x 將加入於串列的尾端，執行的步驟和示意圖如下：

(1) x = new Node ();



(2) x.next = null;



4.1 單向鏈結串列

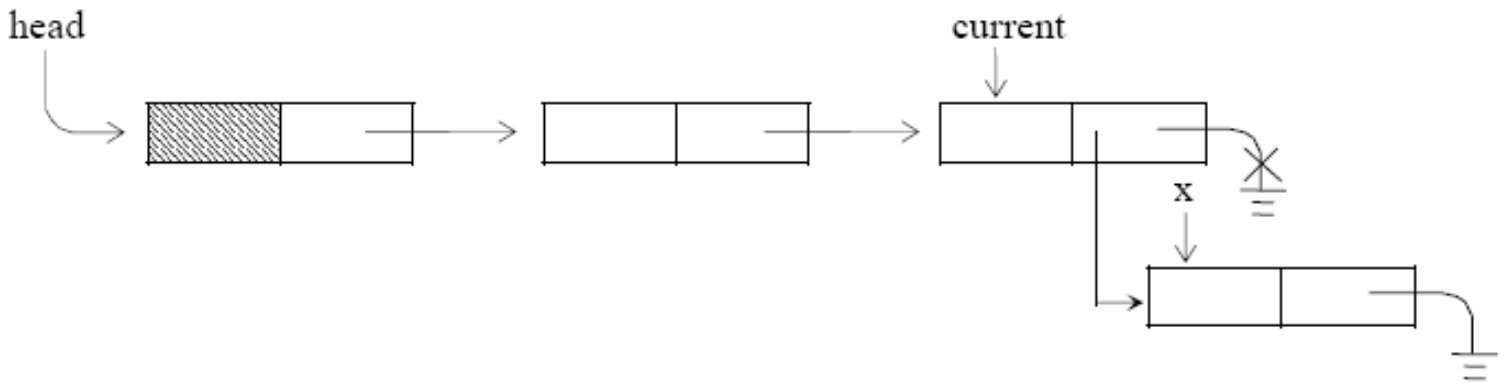
(3) current = head.next;

while (current.next != null)

current = current.next;

current.next = x;

上述的迴圈敘述主要是在追蹤串列的尾端，整個敘述的示意圖如下：



4.1 單向鏈結串列

3. 加入於某一特定節點之後：

假設有一串列是依資料的大小所建立的，其片段程式如下：

Java 片段程式：依據 data 由大至小建立之

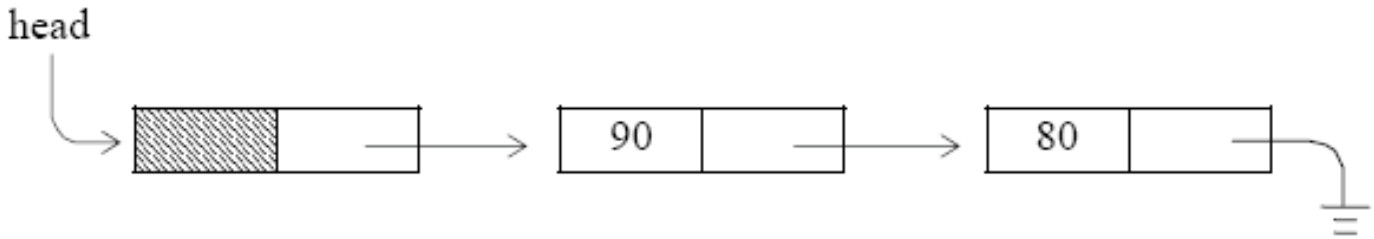
```
 //依分數的高低加入
public static void insert_f()
{
     ptr = new Node();
     ptr.next=null;

     System.out.print(" 請輸入一整數 ");
     ptr.data = keyboard.nextInt();
     System.out.println("");
```

4.1 單向鏈結串列

```
    prev = head;
    current = head.next;
    while ((current != null) && (current.data >= ptr.data)) {
        prev = current;
        current = current.next;
    }
    ptr.next = current;
    prev.next = ptr;
}
```

4.1 單向鏈結串列

假設有一串列如下，



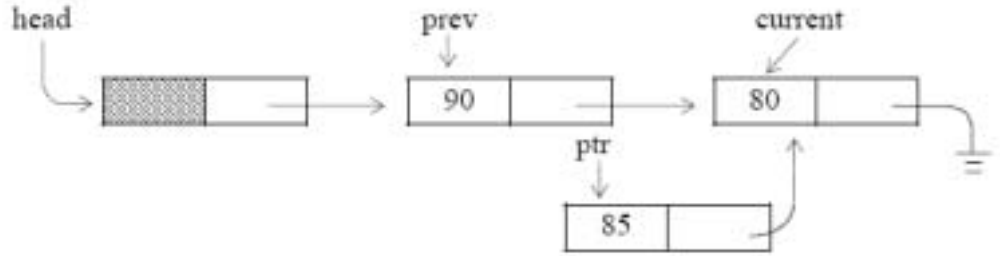
現欲要加入 85，其過程如下，首先利用 while 迴圈敘述找到適當的加入位

```
while ((current != null) && (current.data >= ptr.data)) {
      prev = current;
      current = current.next;
}
```

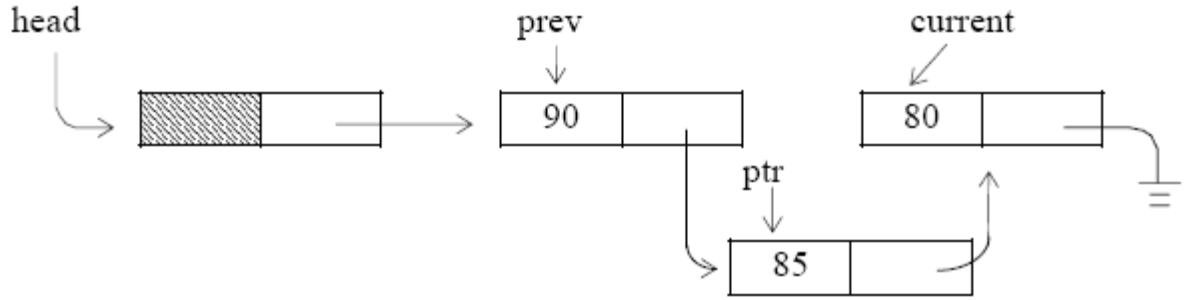
4.1 單向鏈結串列

得知 ptr 所指向的節點(85)應加在 prev 所指向節點的後面，接下來執行的步驟和示意圖如下：

(1) ptr.next = current;



(2) prev.next = ptr;　//經由此敘述，就可將 85 加入於串列中



程式練習

- 修改上述insert_f()程式，使之可以執行以下命令，並將最後之串列印出。

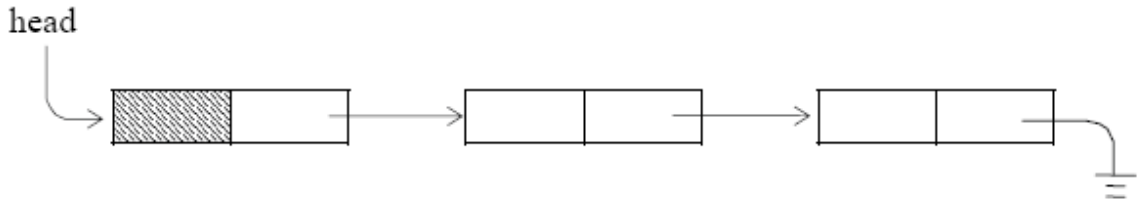
```
insert_f(90);
insert_f(80);
insert_f(85);
```

4.1 單向鏈結串列

4.1.2 刪除動作

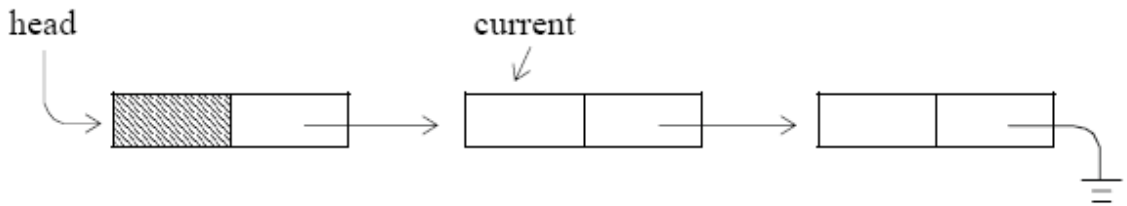
1. 刪除串列前端的節點：

假設有一串列如下，



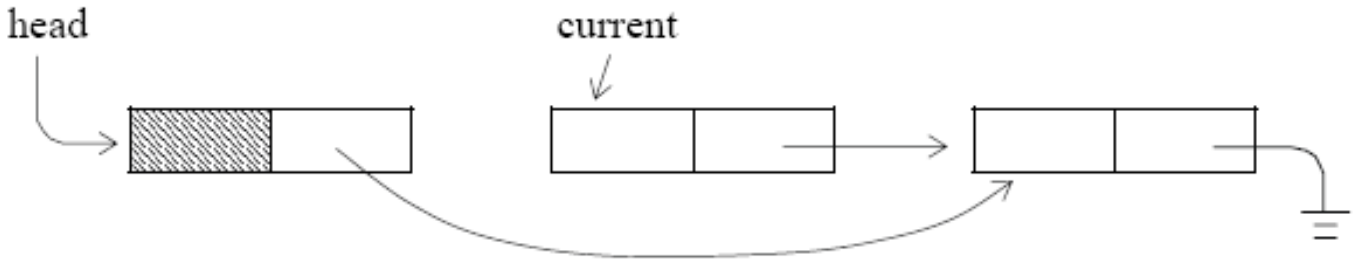
今欲刪除串列的前端節點，執行的步驟和示意圖如下：

(1) current = head.next;　//current 指向 head 的下一節點

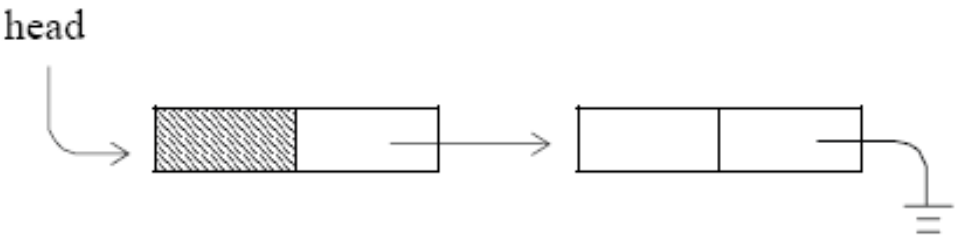


4.1 單向鏈結串列

(2) head.next = current.next; //head 的 next 指向 current 的下一節點



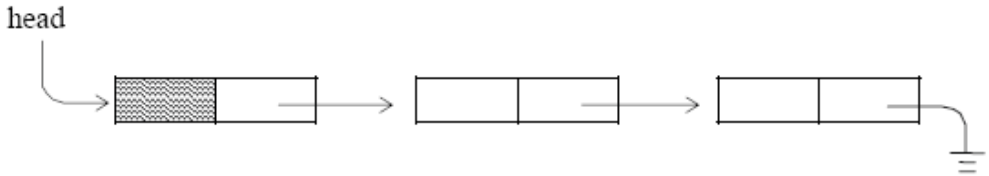
(3) current = null; //將 current 節點回收



4.1 單向鏈結串列

2. 刪除串列的尾端節點：

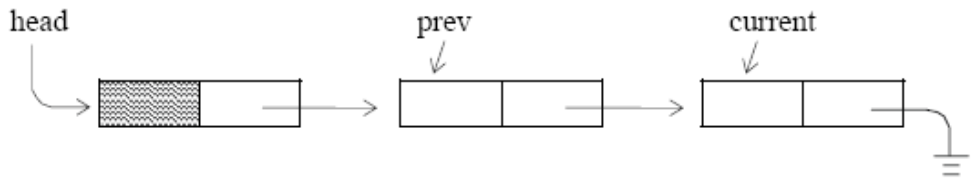
假設有一串列如下：



若欲刪除串列的尾端節點，則須先追蹤串列的尾端節點，其執行的步驟與示意圖如下：

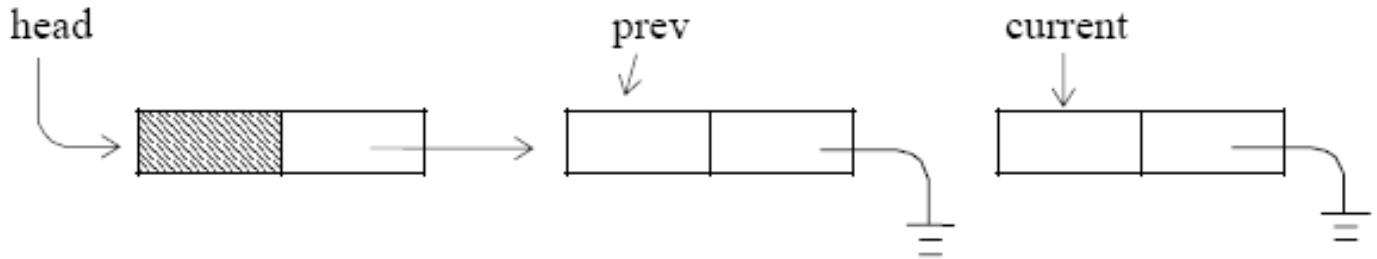
(1) current = head.next;

```
while(current.next != null ) {    // 找出尾端節點
        prev = current;
        current = current.next;
}
```

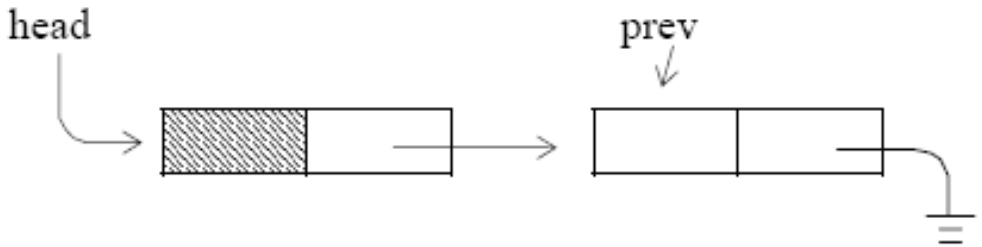


4.1 單向鏈結串列

(2) prev.next = null; //將 prev 的 next 設為 null



(3) current = null; //將 current 節點回收



4.1 單向鏈結串列

3. 刪除某一特定的節點：

刪除單向鏈結串列的某一特定節點之片段程式如下：

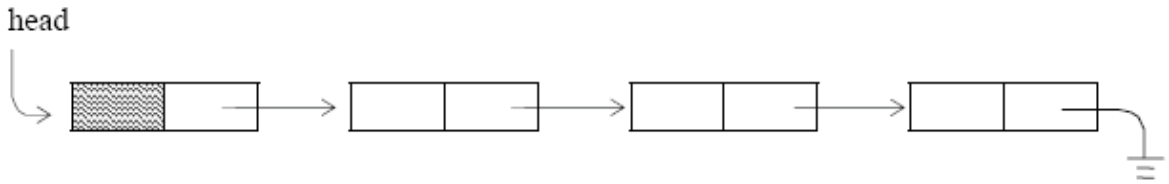
Java 片段程式：刪除單向鏈結串列的某一特定節點

```
public static void delete_f()
{
     if (head.next == null)
        System.out.print(" 串列是空的\n");
     else {
        System.out.print(" 欲刪除的資料: ");
        del_node = keyboard.nextInt();
        prev = head;
        current = head.next;
        while ((current != null) && (!(del_node.equals(current.data)))) {
           prev = current;
           current = current.next;
        }
        if (current != null){
           prev.next = current.next;
           current = null;
           System.out.printf("%d  has been deleted\n\n", del_node);
        }
        else
           System.out.printf(" 資料不存在\n\n", del_node);
     }
}
```

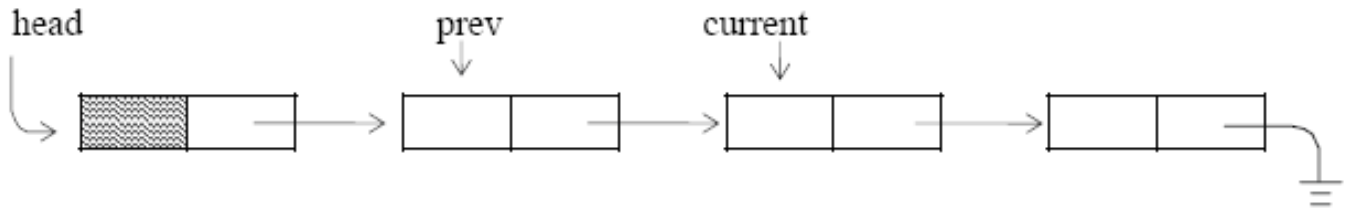
4.1 單向鏈結串列

程式解說

隨機刪除某一節點，首先判斷鏈結串列是不是空的，若不是空的串列，則利用 prev 和 current 指標加以完成之，其中 current 指向即將被刪除節點，而 prev 指向即將被刪除節點的前一節點。如有一串列如下：

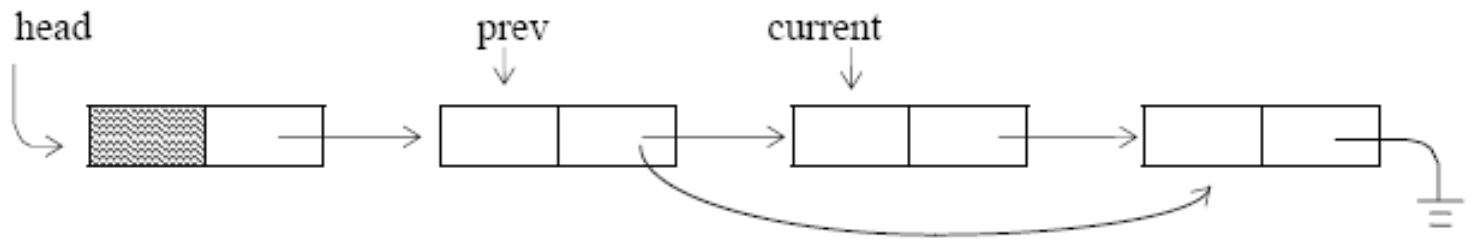


當 del_node 與 current.data 相等時，current 和 prev 分別指向適當的節點，如下圖所示：

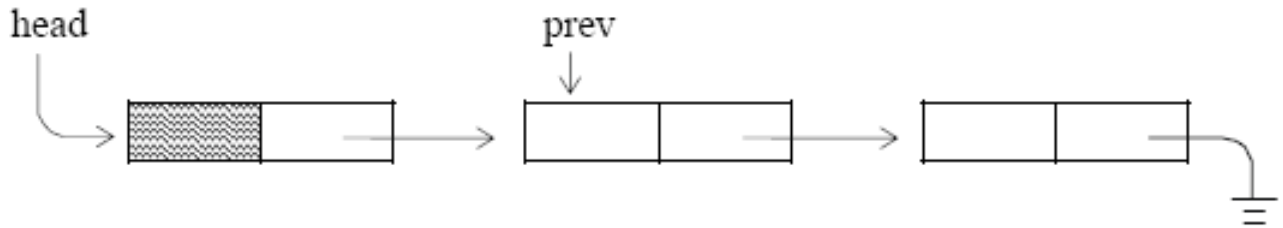


4.1 單向鏈結串列

(1) prev.next = current.next;　　//prev 的 next 指向 current 的下一節點



(2) current = null;　//將 current 節點回收



程式練習

- 修改上述delete_f()程式，使之可以執行以下命令，並將最後之串列印出。

```
insert_f(90);
insert_f(80);
insert_f(85);
insert_f(70);
delete_f(85);
delete_f(80);
```

4.1 單向鏈結串列

4.1.3 將兩串列相連接

串列的相連(concatenate)，顧名思義就是將某一串列加在另一串列的尾端，其片段程式如下：

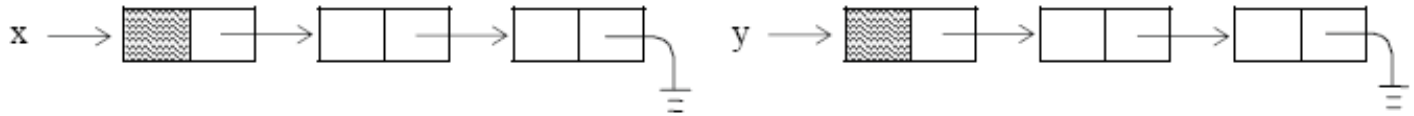
Java 片段程式：鏈結串列的相連

```
public static void concatenate( )
{
   if(x == null)
      z = y;
   else if(y == null)
      z = x;
   else{
      z = x;
      xtail = x.next;
      while(xtail.next != null)
         xtail = xtail.next;
      xtail.next = y.next;
      y = null;
   }
}
```

4.1 單向鏈結串列

程式解說

假設已有兩個鏈結串列如下所示：



此程式乃將 x 與 y 串列合併為 z 串列，其執行的步驟與示意圖如下：

(1) 當 x 串列是空的時候，直接將 y 串列指定給 z 串列。

```
if (x == null)
    z = y;
```

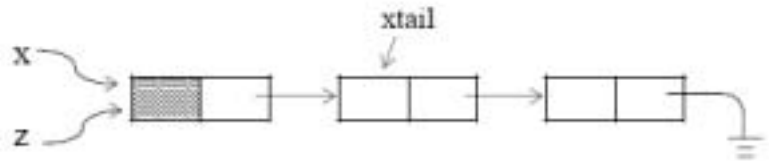
(2) 當 y 串列是空的時候，直接將 x 串列指定給 z 串列。

```
if (y == null)
    z = x;
```

4.1 單向鏈結串列

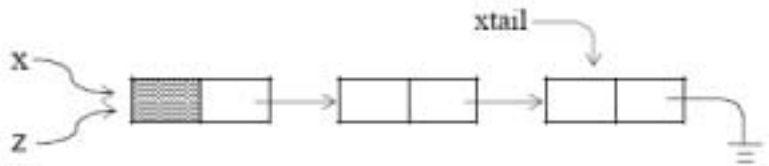
(3) 當 x 和 y 串列都不是空的

```
z = x;
xtail = x.next;
```



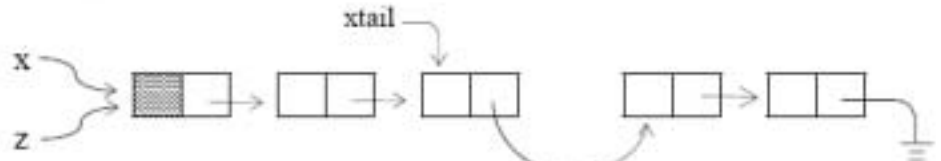
(4) while(xtail.next != null)　　//尋找最後一個節點

```
xtail = xtail.next;
```



(5) 由於 y 指向的節點不含資料，所以將 y.next 指定給 xtail.next

```
xtail.next = y.next;
y = null;
```



程式練習

- 利用先前練習程式產生兩個串列x與y, 並將x與y連接成z。

4.1 單向鏈結串列

4.1.4 將一串列反轉

串列的反轉(invert)，顧名思義就是將串列的前端變為尾端，尾端變為前端，其片段程式如下：

Java 片段程式：鏈結串列的反轉

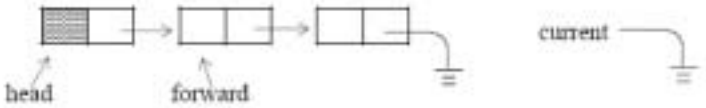
```
public static void invert( )
{
   forward = head.next;
   current = null;
   while(forward != null) {
      prev = current;
      current = forward;
      forward = forward.next;
      current.next = prev;
   }
   tail = head;
   head = current;
}
```

4.1 單向鏈結串列

程式解說

此程式使用了三個 Student 物件，分別為 prev、current 與 forward，用來鎖定前、中、後三個節點，以便做反轉的工作。執行的步驟與示意圖如下：

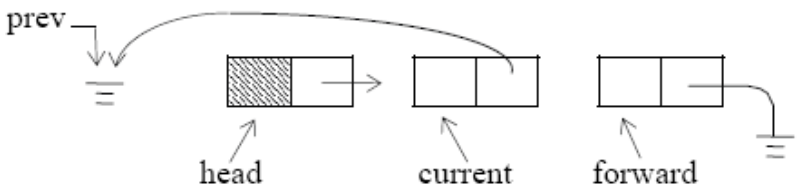
(1) forward = head.next;
current = null;



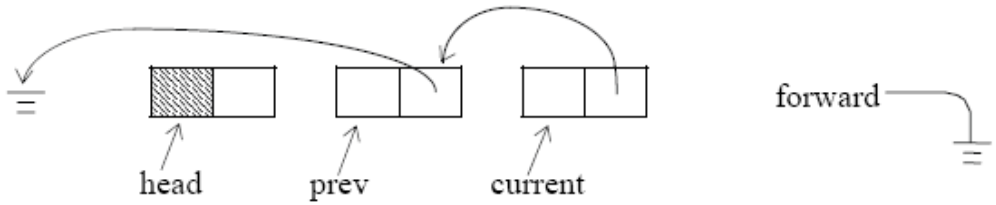

```
(2) while(forward != null) {
        prev = current;
        current = forward;
        forward = forward.next;
        current.next = prev;
    }
```

4.1 單向鏈結串列

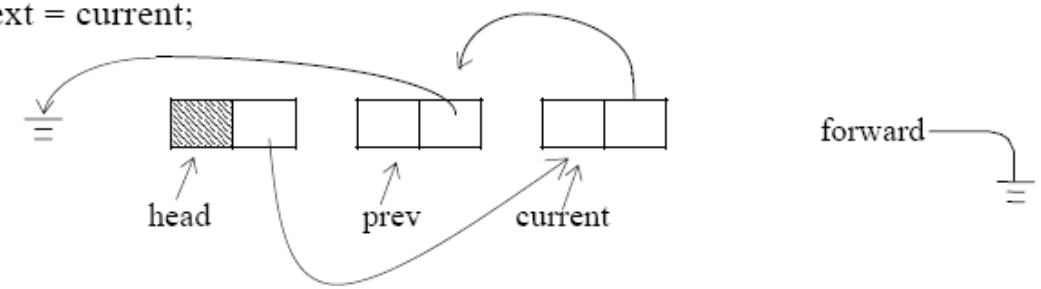
執行完第一次的迴圈後，其示意圖如下：



由於此時 forward 不等於 null，while 迴圈會繼續執行，直到 forward 等於 null。



(3) head.next = current;



4.1 單向鏈結串列

4.1.5 計算串列的長度

計算串列的長度(length)，就是計算串列中有多少個節點。其片段程式如下：

Java 片段程式：計算串列的長度

```
public static int length( )
{
    int leng = 0;
    p = head.next;
    while(p != null)
    {
       leng++;
        p = p.next;
    }
    return leng;
}
```

程式解說

計算串列長度十分簡單，唯一要注意的是，while 迴圈的條件判斷式為 p != null，而不是 p.next != null。這二個判斷式的差異是很大的。

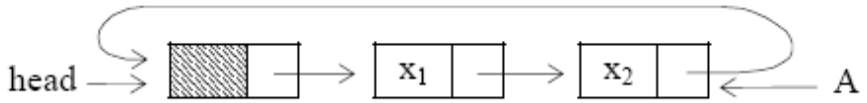
練習題目

- 假設串列有5個節點以上，執行以下程式後，current指標指向何處?

  current = head.next.next.next;

4.2 環狀鏈結串列

若將單向鏈結串列的最後一個節點的 next 指標，指向第一個節點時，則稱此串列為環狀串列(circular list)，如下圖所示：



環狀串列可以從任一節點來追蹤所有節點，同樣我們也假設環狀串列第一個節點不放資料。

4.2 環狀鏈結串列

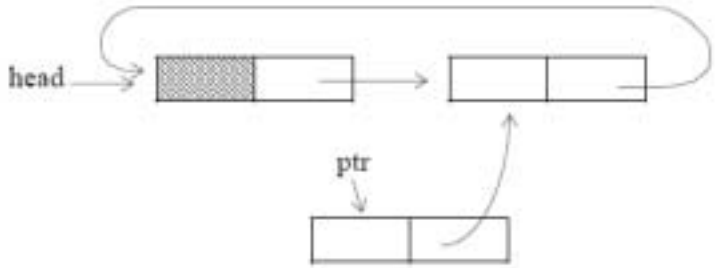
4.2.1 加入動作

1. 加入一節點於環狀串列的前端

今假設有一環狀串列如下：

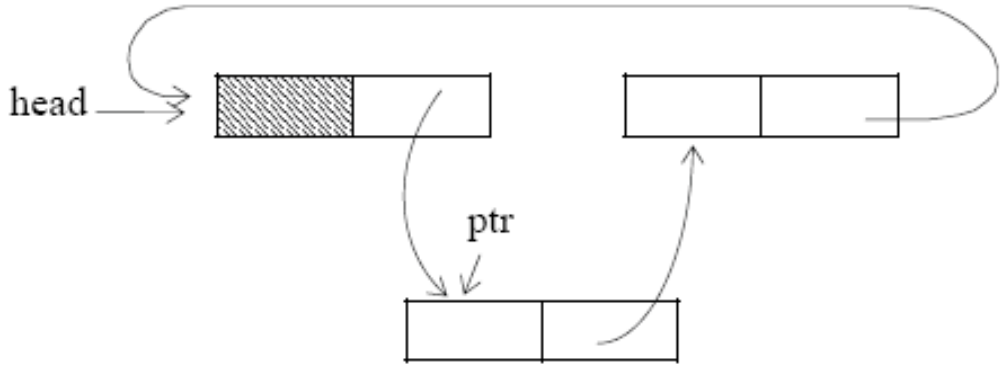
現將 ptr 節點加入於環狀串列的前端，其執行的步驟與示意圖如下：

(1) ptr.next = head.next;



4.2 環狀鏈結串列

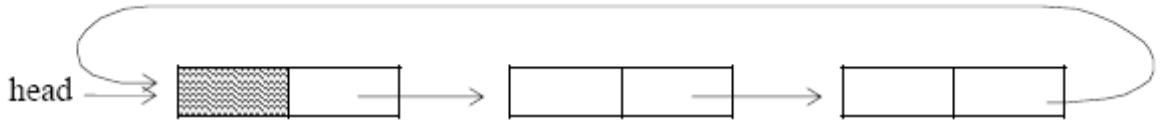
(2) head.next = ptr;



4.2 環狀鏈結串列

2. 加入一節點於環狀串列的尾端

假設有一環狀串列如下：

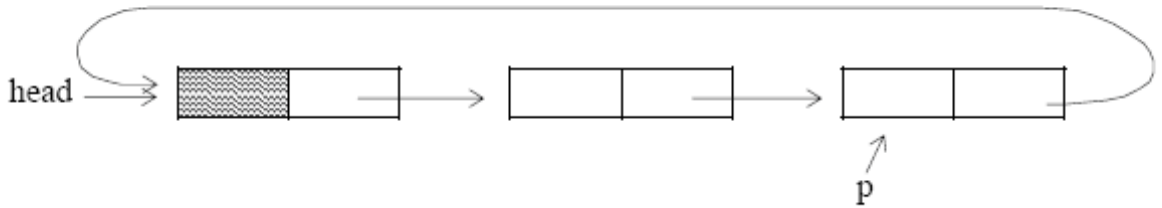


現將 ptr 節點加入於環狀串列的尾端，其執行的步驟與示意圖如下：

(1) 首先要尋找環狀串列的尾端

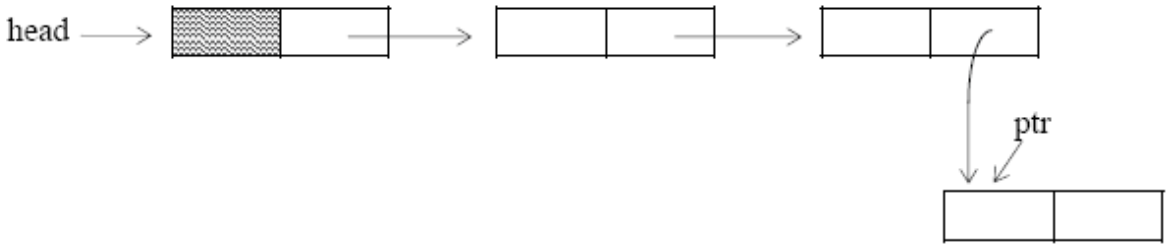
```
p = head.next;
while (p.next != head)
    p = p.next;
```

與單向鏈結串列不同的是，此處是比較 p.next 是否等於 head。

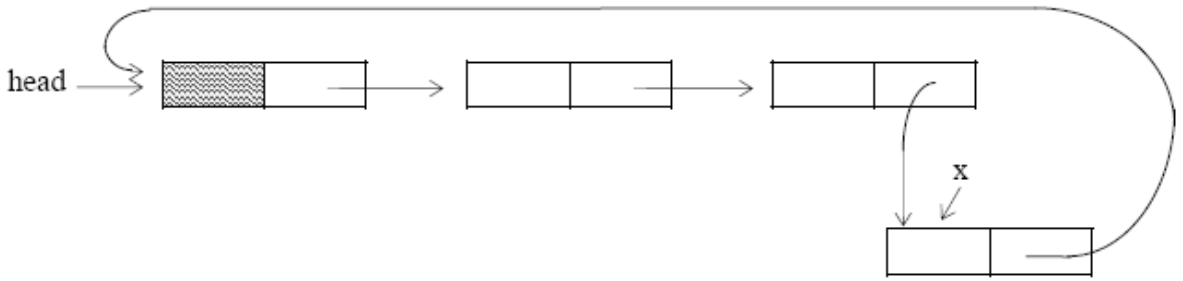


4.2 環狀鏈結串列

(2) p.next = ptr;



(3) ptr.next = head;



4.2 環狀鏈結串列

3. 加入於某一特定節點之後

假設環狀串列是依資料的大小所建立的，其片段程式如下：

Java 片段程式：依資料的大小，由大至小加入於環狀串列

```
System.out.print("欲刪除的資料!");
del_node = keyboard.nextInt();
prev = head;
current = head.next;
while ((current != head) && (current.data >= ptr.data)) {
     prev = current;
     current = current.next;
}
ptr.next = current;
prev.next = ptr;
```

程式解說

此片段程式與加入一節點於單向鏈結串列的某一特定節點相似，在此不再贅述。其差異為迴圈的判斷式，如下所示：

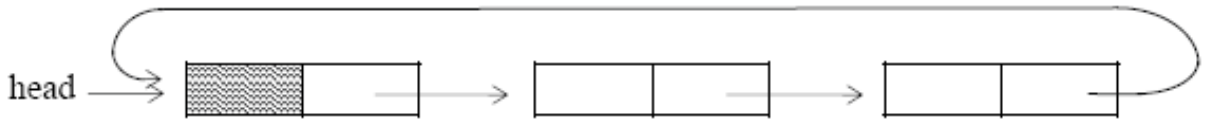
```
while ((current != head) && (current.data >= ptr.data))
```

4.2 環狀鏈結串列

4.2.2 刪除的動作

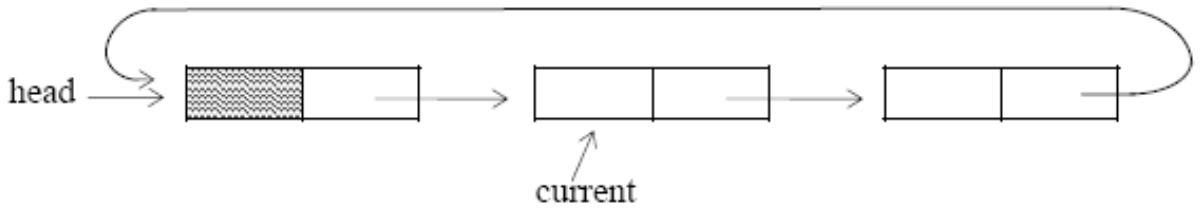
1. 刪除環狀串列的前端

若有一環狀串列如下：



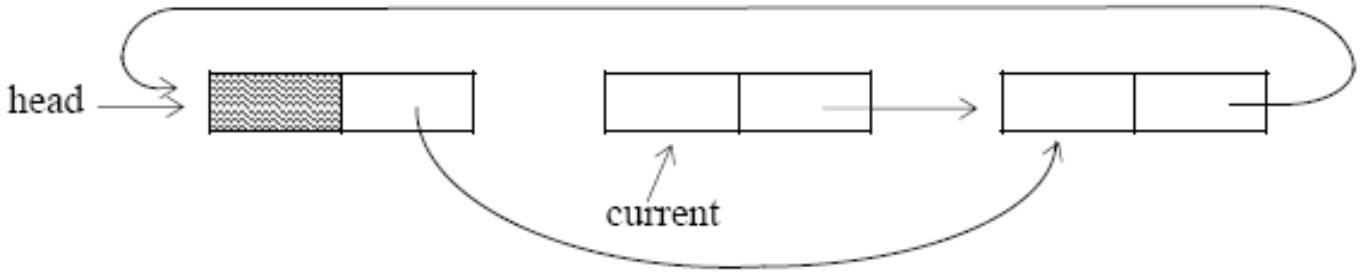
刪除前端節點的執行步驟與示意圖如下。

(1) current = head.next;

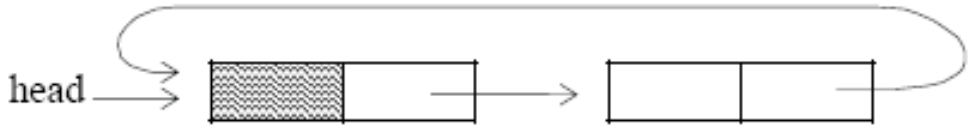


4.2 環狀鏈結串列

(2) head.next = current.next;



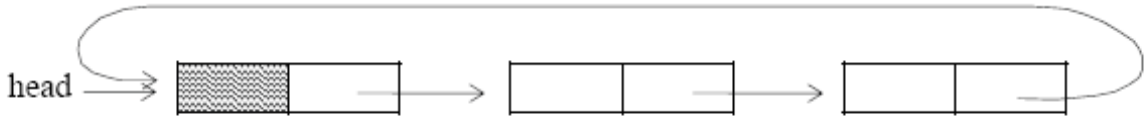
(3) current = null;



4.2 環狀鏈結串列

2. 刪除環狀串列的尾端

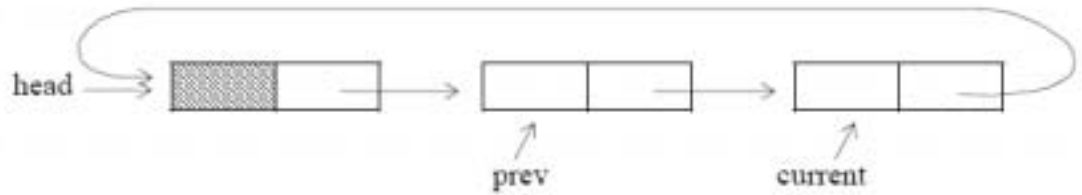
有一環狀串列如下：



其刪除尾端節點的執行步驟與示意圖如下。

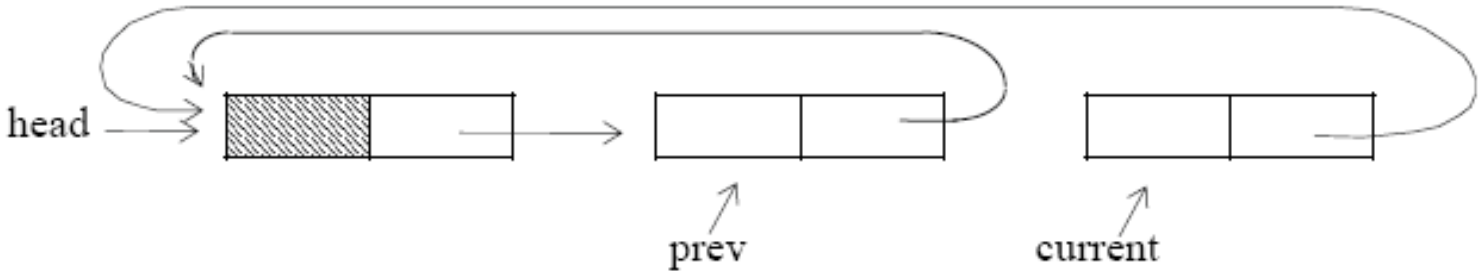
(1) 首先找出環狀串列的尾端

```
current = head.next;
while (current.next != head) {
        prev = current;
        current = current.next;
}
```



4.2 環狀鏈結串列

(2) prev.next = current.next;



(3) current = null;

4.2 環狀鏈結串列

3. 刪除環狀串列的某一特定節點

刪除環狀串列的某一特定節點之片段程式如下：

Java 片段程式：刪除環狀鏈結串列的某一特定節點

```
System.out.print("欲刪除的資料!");
del_node = keyboard.nextInt();
prev = head;
current = head.next;
while ((current != head) && (!(del_node.equals(current.data)))) {
     prev = current;
     current = current.next;
}
if (current != head){
  prev.next = current.next;
  current = null;
```

4.2 環狀鏈結串列

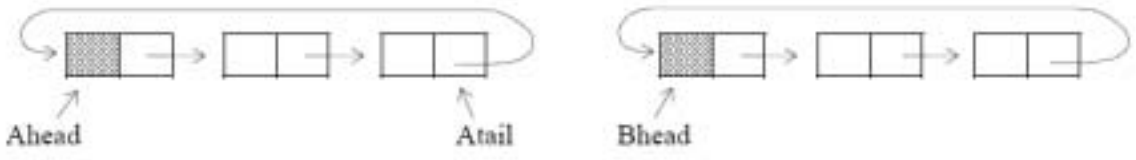
4.2.3 兩個環狀串列之相連

串列的相連就是將一串列加在另一串列的尾端，假設今有二個環狀串列如下：

以下是串列相連的執行步驟與示意圖：

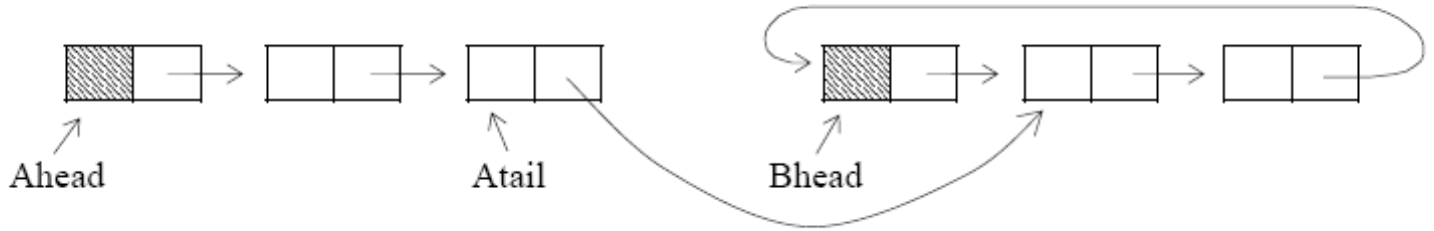
1. 先追蹤第一個環狀串列的尾端

```
Atail = Ahead.next;
while (Atail.next != Ahead)
        Atail = Atail.next;
```



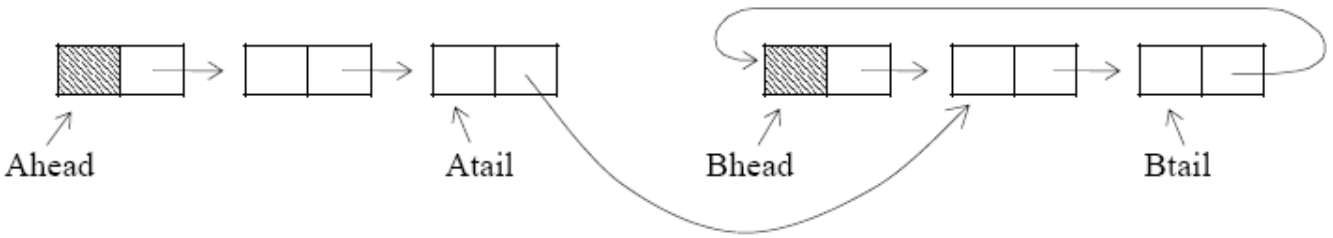
4.2 環狀鏈結串列

2. Atail.next = Bhead.next;



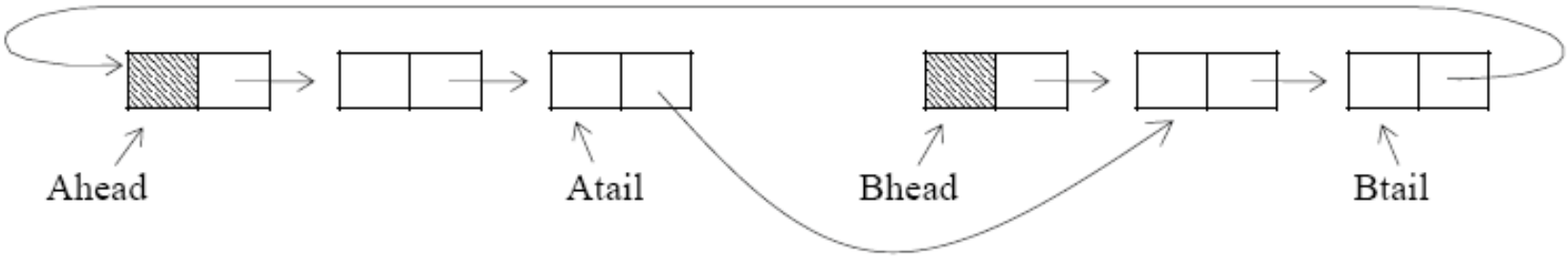
3. 追蹤第二個環狀串列的尾端

```
Btail = Bhead.next;
while (Btail.next != Bhead)
        Btail = Btail.next;
```

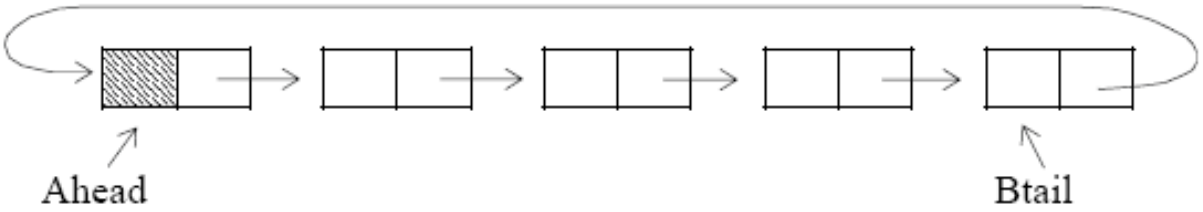


4.2 環狀鏈結串列

4. Btail.next = Ahead;



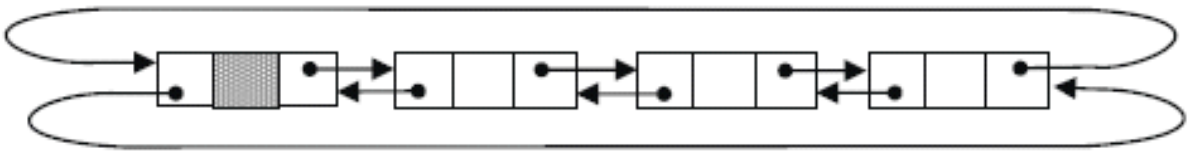
5. Bhead = null;



4.3 雙向鏈結串列

- 雙向鏈結串列(doubly linked list) 乃是每個節點皆具有三個欄位，一為左鏈結(LLINK)，二為資料(DATA)，三為右鏈結(RLINK)，其資料結構如下：

- 其中LLINK 指向前一個節點，而RLINK 指向後一個節點。通常在雙向鏈結串列加上一個串列首，此串列首的資料欄不存放資料。如下圖所示：



4.3 雙向鏈結串列

雙向鏈結串列具有下列兩點特性：

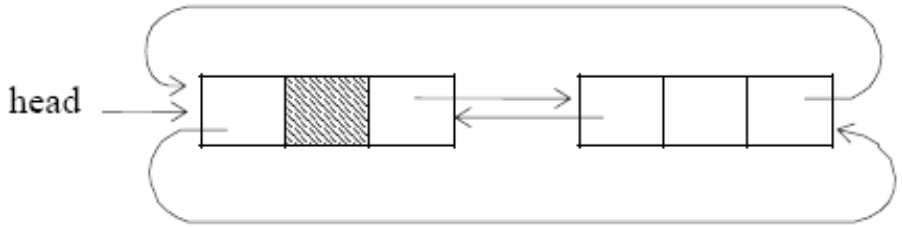
1. 假設ptr 是任何節點的指標，則
   ptr = ptr.llink.rlink = ptr.rlink.llink;
2. 若此雙向鏈結串列是空串列，則只有一個串列首。

4.3 雙向鏈結串列

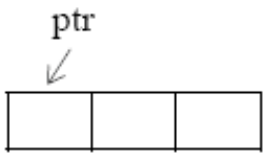
4.3.1 加入動作

1. 加入一節點於雙向鏈結串列的前端

假設有一雙向鏈結串列如下：



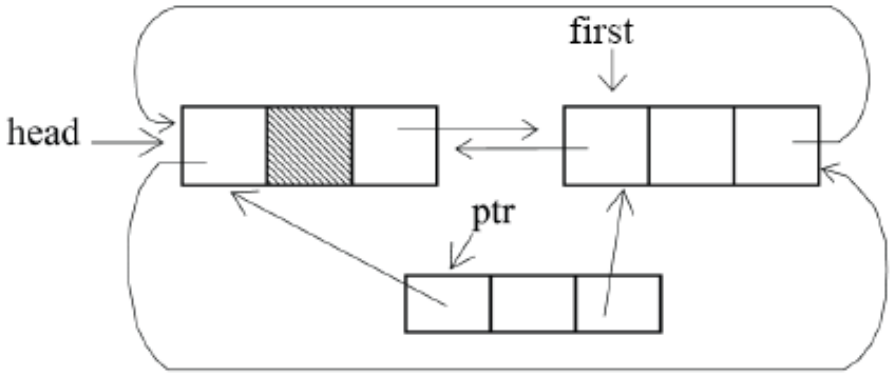
今欲將 ptr 的節點加入於雙向鏈結串列的前端，其執行步驟與示意圖如下:



4.3 雙向鏈結串列

(1) 經由下列敘述即可完成

```
first = head.rlink;
ptr.rlink = head.rlink;
ptr.llink = head;
```

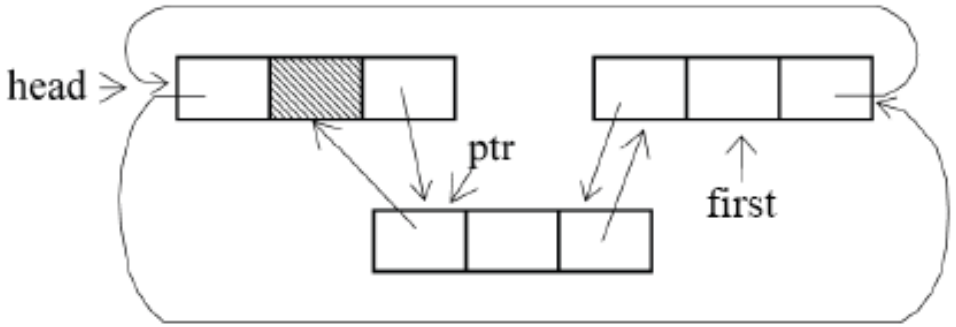


此時 ptr 的 rlink 和 llink 就可指向適當的節點。

4.3 雙向鏈結串列

(2) 之後，將 ptr 指定給 head 的 rlink 及 first 的 llink。

```
head.rlink = ptr;
first.llink = ptr;
```



就可完成加入的動作。

4.3 雙向鏈結串列

2. 加入一節點於雙向鏈結串列的尾端

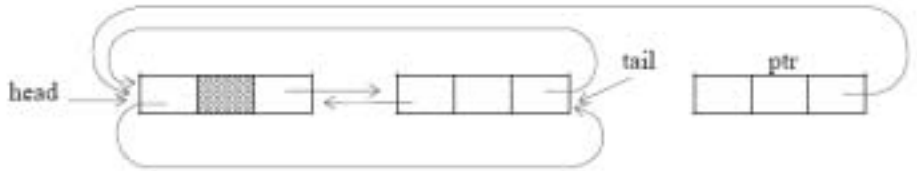
假設有一串列如下：

(1) 首先利用

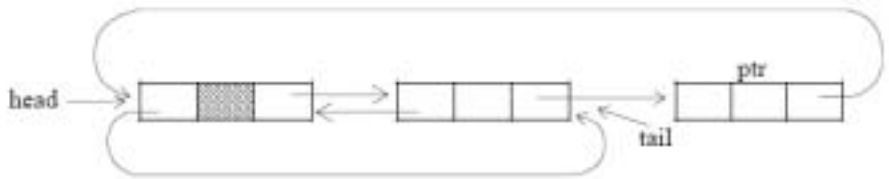
tail = head.llink;

找到串列的尾端。

(2) ptr.rlink = tail.rlink;

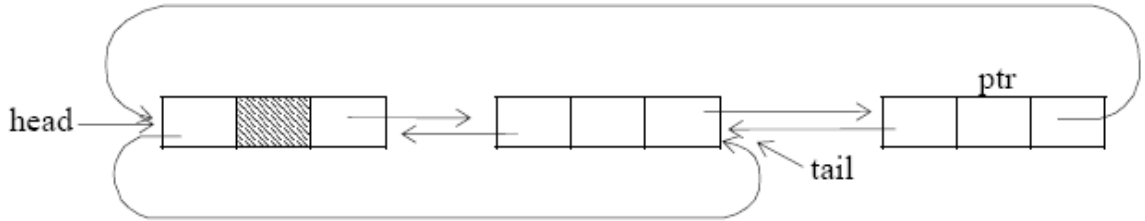


(3) tail.rlink = ptr;

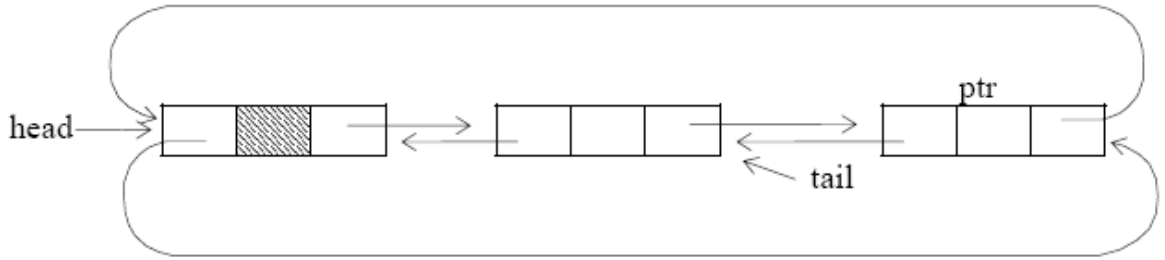


4.3 雙向鏈結串列

(4) ptr.llink = tail;



(5) head.llink = ptr;



4.3 雙向鏈結串列

3. 加入一節點於串列某一特定節點之後

假設雙向鏈結串列是依資料大小所建立的，其片段程式如下：

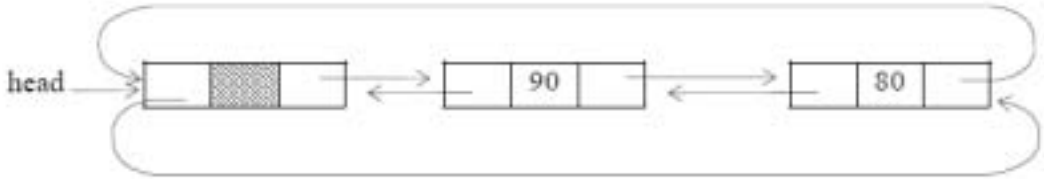
Java 片段程式：依 data 由大至小所建立的雙向鏈結串列

```
prev = head;
current = head.rlink;
while((current != head) && (current.data >= ptr.data)) {
  prev = current;
  current = current.rlink;
}
ptr.rlink = current;
ptr.llink = prev;
prev.rlink = ptr;
current.llink = ptr;
```

4.3 雙向鏈結串列

程式解說

假設有一雙向鏈結串列如下：

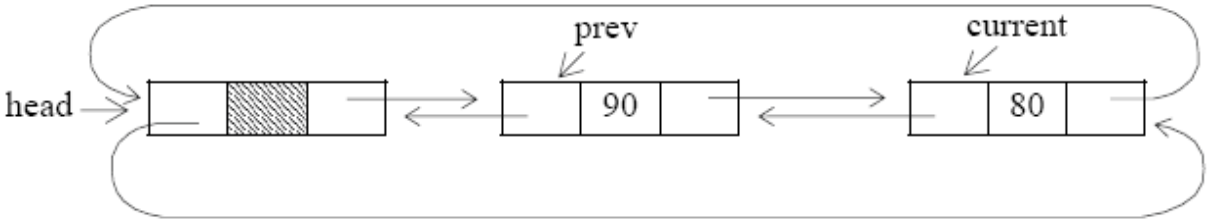


今欲將 ptr 所指向的節點(鍵值為 85)加入於雙向鏈結串列，以下是其執行步驟與示意圖。

1. 首先，利用迴圈敘述找到欲插入節點的位置

```
prev = head;
current = head.rlink;
while((current != head) && (current.data >= ptr.data)) {
        prev = current;
        current = current.rlink;
}
```

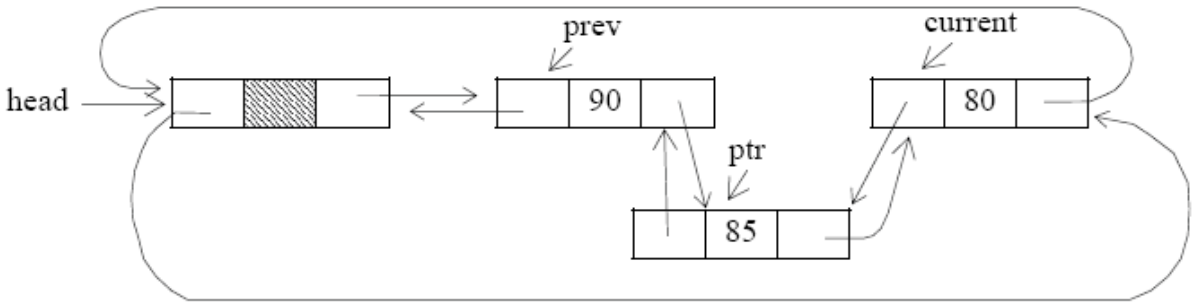
4.3 雙向鏈結串列



2. 之後，經由下列敘述就可達成加入的動作，

```
ptr.rlink = current;
ptr.llink = prev;
prev.rlink = ptr;
current.llink = ptr;
```

最後的圖形如下所示：



程式練習

- 修改上述insert_f()程式，使之可以執行以下命令，並將最後之串列印出。

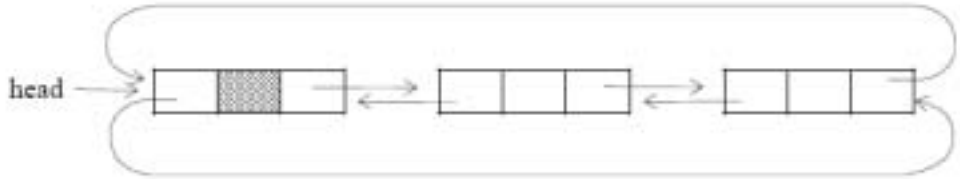
```
insert_f(90);
insert_f(80);
insert_f(85);
```

4.3 雙向鏈結串列

4.3.2 刪除的動作

1. 刪除雙向鏈結串列的前端節點：

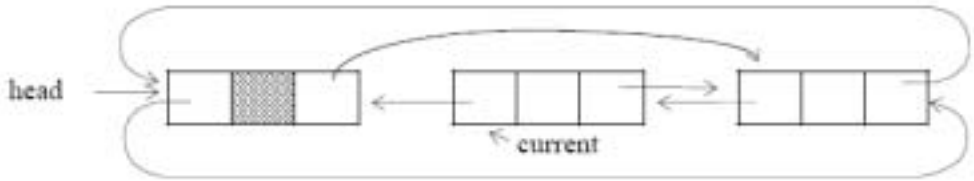
   此處的前端節點乃指 head.rlink 所指向的節點，因為 head 指向的節點沒有存放資料。



執行的步驟與示意圖如下：

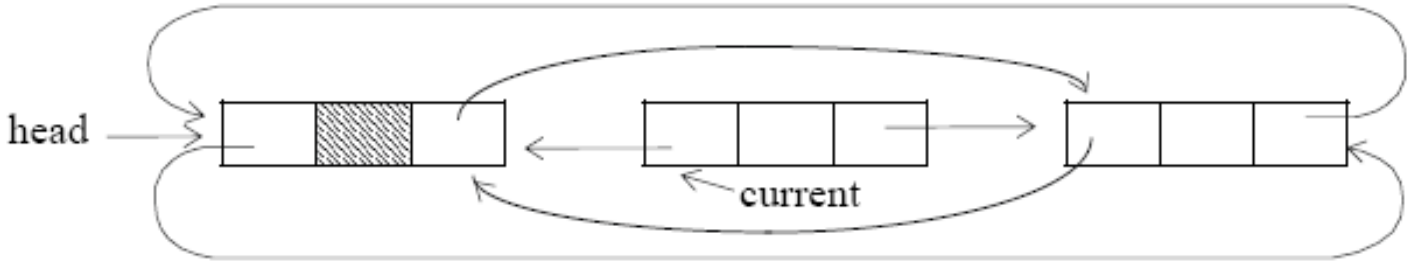
(1) current = head.rlink;

(2) head.rlink = current.rlink;

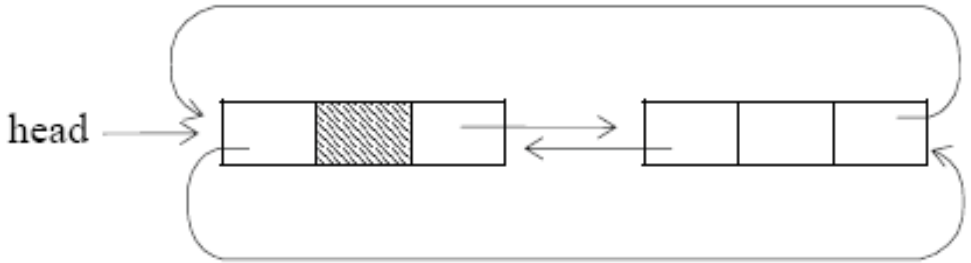


4.3 雙向鏈結串列

(3) current.rlink.llink = current.llink;



(4) current = null;



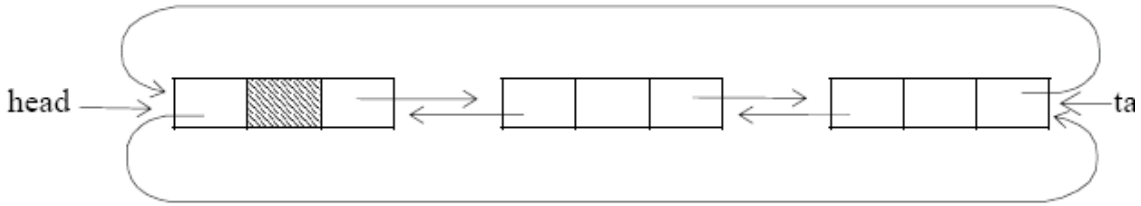
4.3 雙向鏈結串列

2. 刪除雙向鏈結串列的尾端節點：

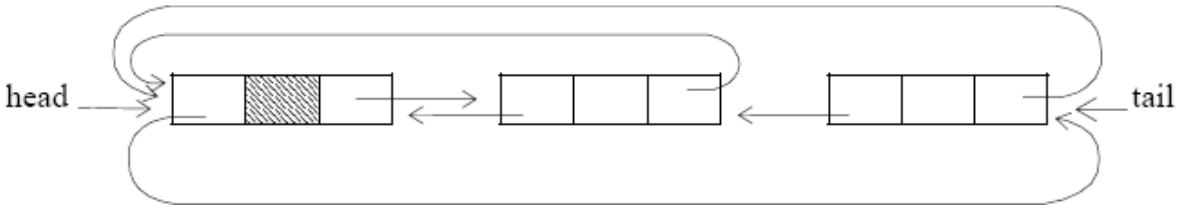
刪除雙向鏈結串列尾端節點的執行步驟與示意圖如下：

(1) 首先經由下一敘述，將 tail 指向串列的尾端。

```
tail = head.llink;
```

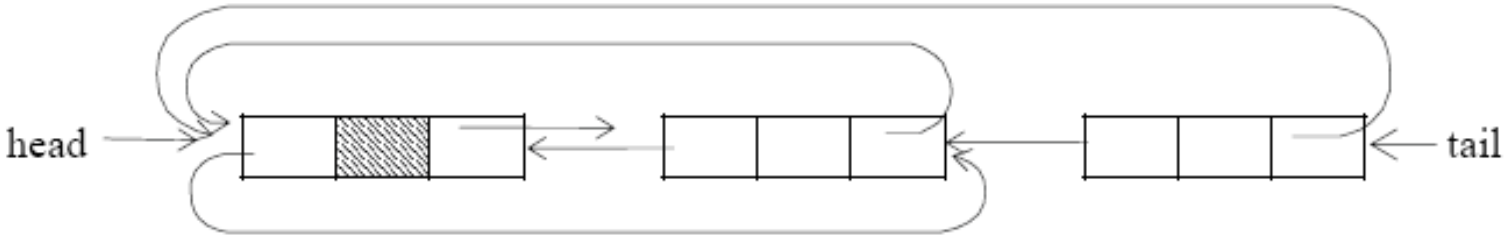


(2) tail.llink.rlink = tail.rlink;

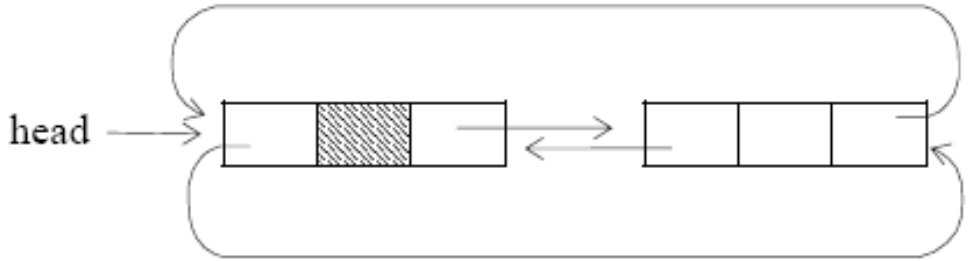


4.3 雙向鏈結串列

(3) head.llink = tail.llink;



(4) tail = null;



4.3 雙向鏈結串列

3. 刪除雙向鏈結串列的某一特定節點

刪除某一特定節點的片段程式如下：

Java 片段程式：隨機刪除雙向鏈結串列的某一節點

```
System.out.print("欲刪除的資料!");
del_node = keyboard.nextInt();
prev = head;
current = head.rlink;
while((current.rlink != head) && (!del_node.equals(current.data))) {
   prev = current;
   current = current.rlink;
}
prev.rlink = current.rlink;
current.rlink.llink = prev;
current = null;
System.out.println("The" + del_node + " record(s) deleted !!\n");
if(current == head)
    System.out.println("The" + del_node + " not found !!\n");
```

4.3 雙向鏈結串列

程式解說

此片段程式的重點如下：

1. 使用下一迴圈敘述來搜尋資料

   while ((current.rlink != head) && (!del_node.equals(current.data)))

   當找到符合的資料後，利用以下敘述

```
prev.rlink = current.rlink;
current.rlink.llink = prev;
current = null;
```

   即可刪除該筆資料。

2. 使用以下的選擇敘述，判斷此資料是否存在。

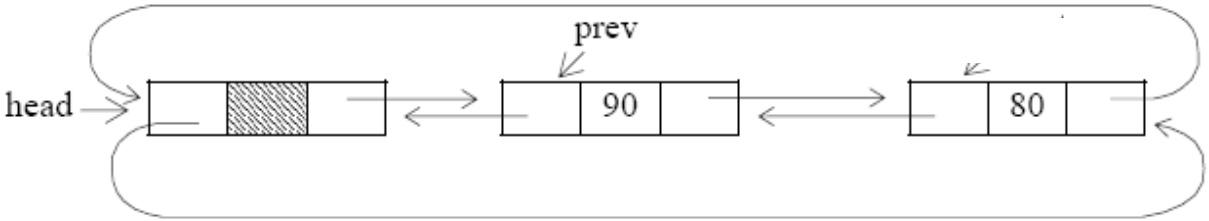
```
if(current == head)
    System.out.println("The" + del_node + " not found !!\n");
```

程式練習

- 修改上述delete_f()程式，使之可以執行以下命令，並將最後之串列印出。

```
insert_f(90);
insert_f(80);
insert_f(85);
insert_f(70);
delete_f(85);
delete_f(80);
```

練習題目



- 請寫出將一新節點x加在prev後面之程式片段。

```
x.rlink = prev.rlink;
prev.rlink.llink = x;
x.llink = prev;
prev.rlink = x;
```

練習題目

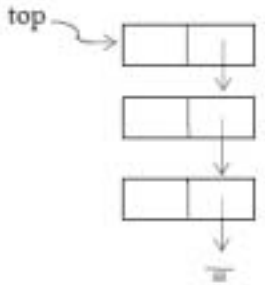
- 假設有一雙向鏈結串列，請寫出將中間節點x刪除之程式片段。

```
x.llink.rlink = x.rlink;
x.rlink.llink = x.llink;
```

4.4 鏈結串列的應用

4.4.1 以鏈結串列表示堆疊

1. 加入一個節點於堆疊中：由於堆疊的運作都在同一端，因此可將它視為將節點加入於串列的前端。假設第一個節點有存放資料，如下圖所示:



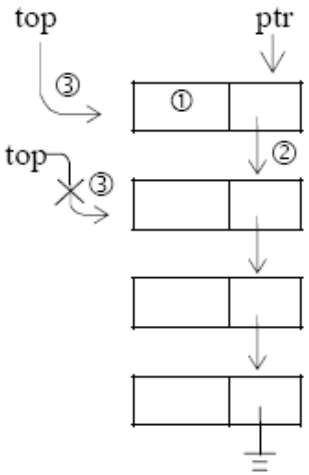
Java 片段程式：堆疊的加入

```
public void push_stack( )
{
    ptr = new Node();
    ptr.data = java_score;
    ptr.next = top;
    top = ptr;
}
```

4.4 鏈結串列的應用

程式解說

1. 程式中 java_score 為新增的資料。堆疊的加入好比將資料加入於鏈結串列的前端。也就是將 ptr 加入於 top 之前。



說明：

① ptr.data = java_score;

② ptr.next = top;

③ top = ptr;

—X→ 表示鏈結斷掉

4.4 鏈結串列的應用

2. 從堆疊刪除一節點：好比刪除鏈結串列的前端節點。

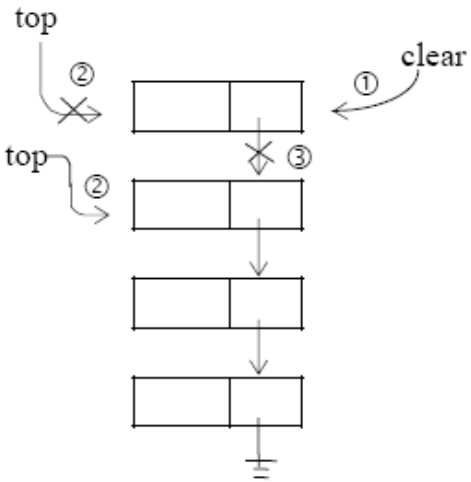
Java 片段程式：堆疊的刪除

```
public void pop_stack( )
{
  Node clear;
  if(top == null){
     System.out.print("堆疊是空的");
     return;
  }
  clear = top;
  Delete_data = top.data;
  top = top.next;
  clear = null;
}
```

4.4 鏈結串列的應用

程式解說

堆疊的刪除就如同刪除單向鏈結串列於前端，在刪除前必須先以 if(top == null)來判斷堆疊是否為空，若是，則顯示堆疊內沒有資料。



說明：

① clear = top;

② top = top.next;

③ clear = null;

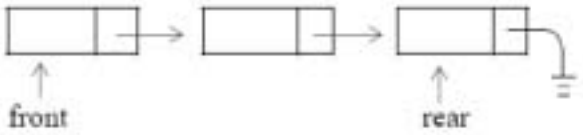
—X→ 表示鏈結斷掉

當然也可以將堆疊的加入與刪除都作用於串列的尾端，只要作用在同一端即可。

4.4 鏈結串列的應用

4.4.2 以鏈結串列表示佇列

1. 加入一節點於佇列中：好比將節點加入於鏈結串列的尾端。今有一鏈結串列如下，並假設此串列第一個節點有存放資料。



Java 片段程式：佇列的加入

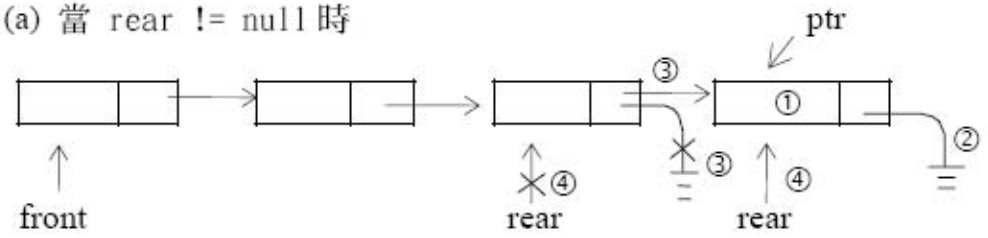
```
public void enqueue( )
{
  ptr = new Node();
  ptr.data = java_score;
  ptr.next = null;
  if(rear == null)
     front = ptr;
  else
     rear.next = ptr;
  rear = ptr;
}
```

4.4 鏈結串列的應用

程式解說

1. 先判斷 rear 是否為 null; 若是，則表示新增的資料為佇列的第一筆資料;若不是，則將 rear 的 next 指向新增節點即可。執行的步驟與示意圖如下：

(a) 當 rear != null 時



說明：

① ptr.data = java_score;

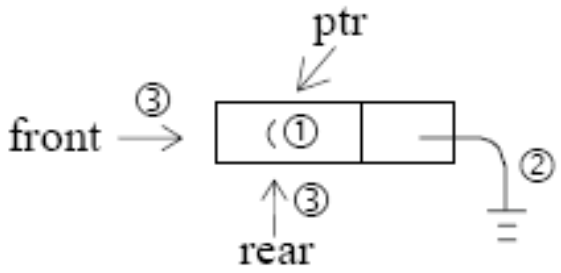
② ptr.next = null;

③ rear.next = ptr;

④ rear = ptr;

4.4 鏈結串列的應用

(b) 當 rear == null 時



說明：

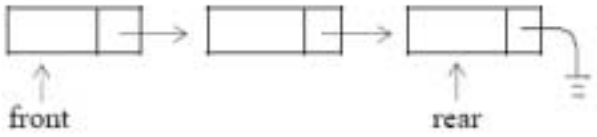
① ptr.data= java_score;

② ptr.next = null;

③ front = rear = ptr;

4.4 鏈結串列的應用

2. 刪除佇列的第一個節點：好比刪除鏈結串列的前端節點。



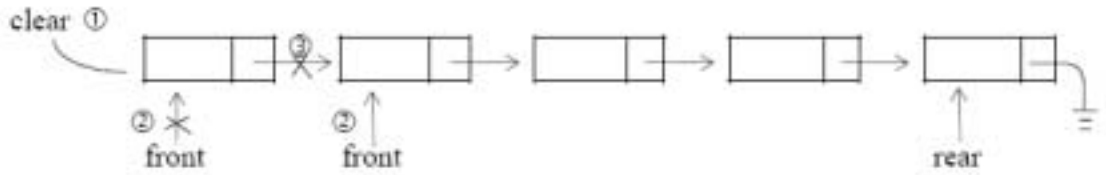
Java 片段程式：佇列的刪除

```
public void dequeue( )
{
    Node clear;
    if(front == null)
    {
       System.out.print("串列是空的");
       return;
    }
    java_score = front.data;
    clear = front;
    front = front.next;
    clear = null;
}
```

4.4 鏈結串列的應用

程式解說

若佇列的加入在串列的尾端，則刪除就是在鏈結串列的前端。當 front 等於 null 時，表示佇列內沒有資料存在。若 front 不等於 null，則比照刪除串列前端的方式來處理，如下圖所示：



說明：

① clear = front;

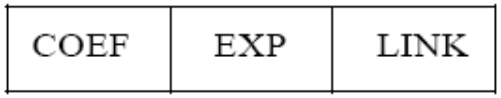
② front = front.next;

③ clear = null;

4.4 鏈結串列的應用

4.4.3 多項式相加

多項式相加可以利用鏈結串列來完成。多項式以鏈結串列的資料結構如下：



COEF 是變數的係數，EXP 為變數的指數，而 LINK 為指向下一節點的指標。

假設有一多項式　$A = 3x^{14} + 2x^{8} + 1$，以鏈結串列表示如下：

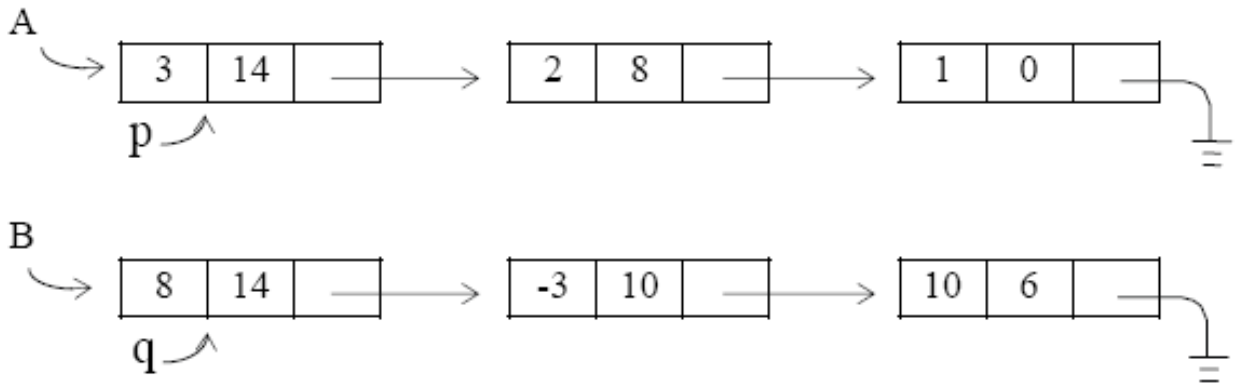
兩個多項式的相加，請看下一範例的說明。

今有二個多項式分別為

$A = 3x^{14} + 2x^{8} + 1$，　$B = 8x^{14} - 3x^{10} + 10x^{6}$

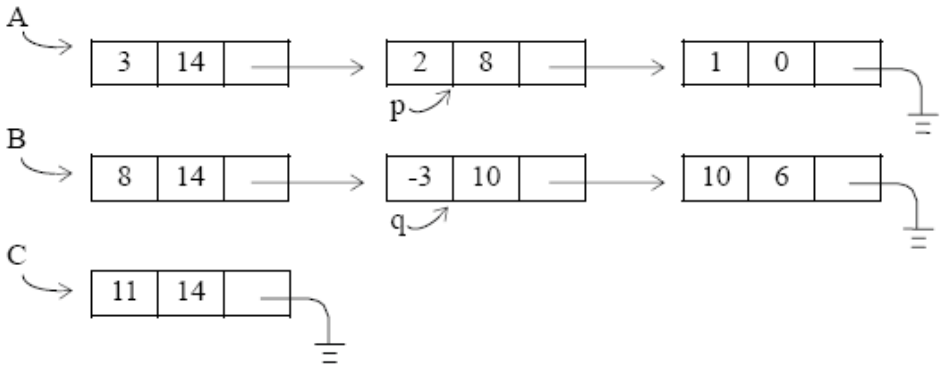
4.4 鏈結串列的應用

以多項式表示如下：

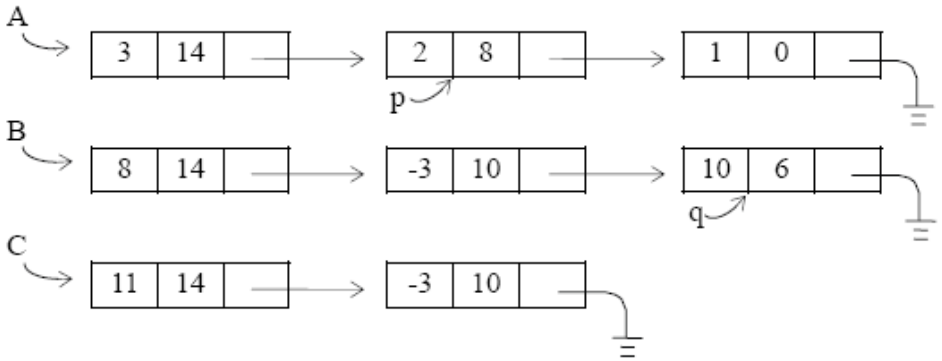


4.4 鏈結串列的應用

1. 此時 A、B 兩多項式的第一個節點 EXP 皆相同(EXP(p) = EXP(q))，所以相加後放入 C 串列，同時 A、B 的指標指向為下一個節點。

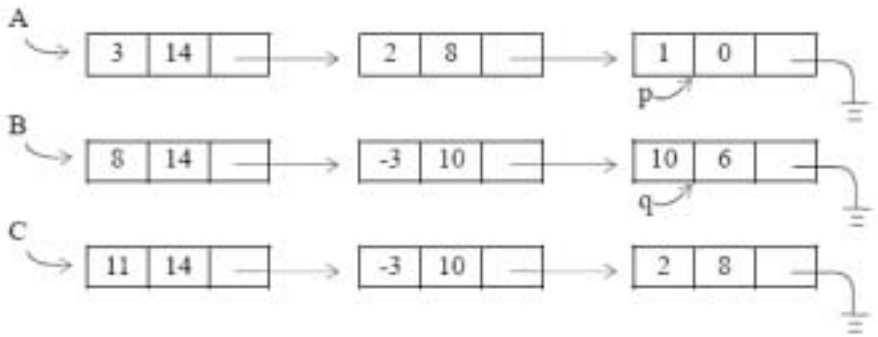


2. 由於 EXP(p) = 8 < EXP(q) = 10，因此將 B 多項式的第二個節點加入 C 多項式，並且將 q 指標指向下一個節點。



4.4 鏈結串列的應用

3. 由於 EXP(p) = 8 > EXP(q) = 6，因此將 A 多項式的第二個節點加入 C 多項式，並將 P 指標指向為下一個節點。



4. 依此類推，最後 C 的多項式為

$$C = 11x^{14} - 3x^{10} + 2x^{8} + 10x^{6} + 1$$