鏈結串列
Link list

Chun-Jung Lin

國立勤益科技大學資訊工程系
Department of computer science and information engineering

OUTLINE

- 鏈結串列(Linked List)
- 環狀鏈結串列(Circular Linked List)
- 雙向鏈結串列(Double Linked List)

鏈結串列(Linked List)

- 鏈結串列(Linked List)是使用Pointer(指標)串接資料，使用鏈結串列的好處是找到指定位置後，可以有效率地插入或刪除元素，陣列不適合在中間位置插入或刪除元素，因為需要花較多時間搬移元素，適合在兩端插入或刪除元素。
- 鏈結串列不能隨機讀取指定位置的元素，只能從前往後一個一個走到指定的位置才能讀取，而陣列可以使用索引值(隨機)讀取陣列中指定位置的元素。

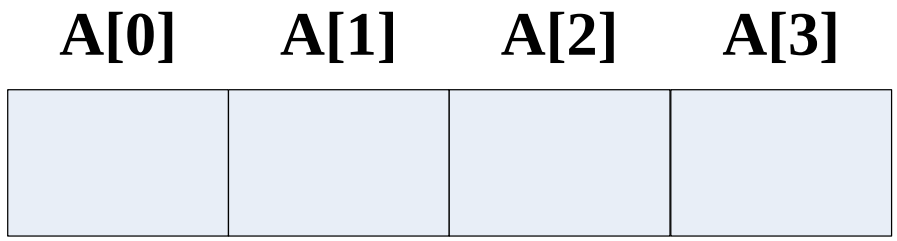
鏈結串列

- 鏈結串列(linked list)是由許多節點所組成的，在加入和刪除功能上比陣列容易許多。

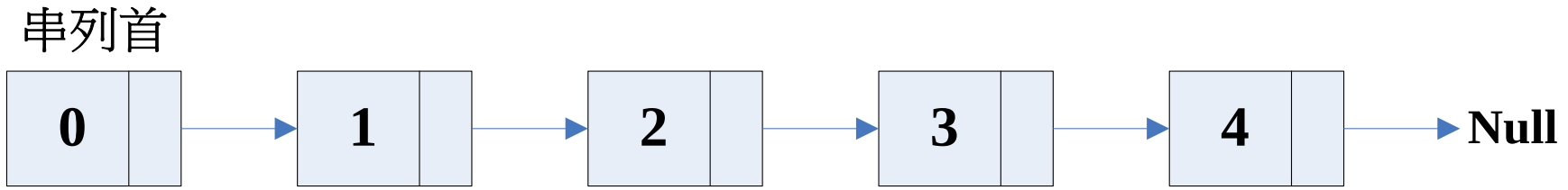
	靜態資料結構(如：陣列)	動態資料結構(如：串列)
1	比較節省記憶體空間	比較浪費記憶體空間，因為必須要多出一個指標
2	加入、刪除及合併時，必須做大量資料的移動	加入、刪除及合併時，只須要改變指標即可
3	可以直接存取	不可以直接存取
4	可進行二分法搜尋	不可以進行二分法搜尋

串列 List

- **循序串列(Sequential List)**：如陣列它是以連續的記憶體位置呈現。



- **鏈結串列(Linked List)**：利用指標來串接所有的節點，並且最後一個節點指標指向Null



Discussion

- 能從中間穿插資料的鏈結串列，在日常生活中可以有什麼應用？

鏈結串列的生活應用

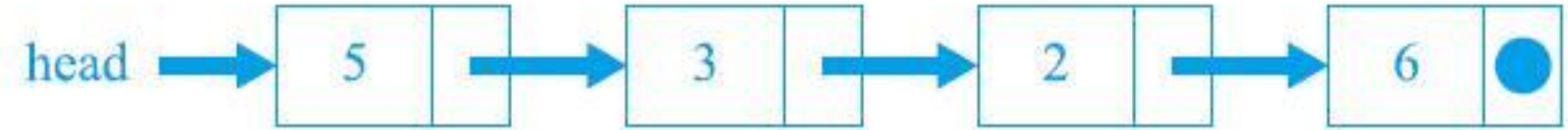
- 網頁的超連結
- KTV點歌
  - 可以插播
- 電腦硬碟重整

鏈結串列 (1/2)

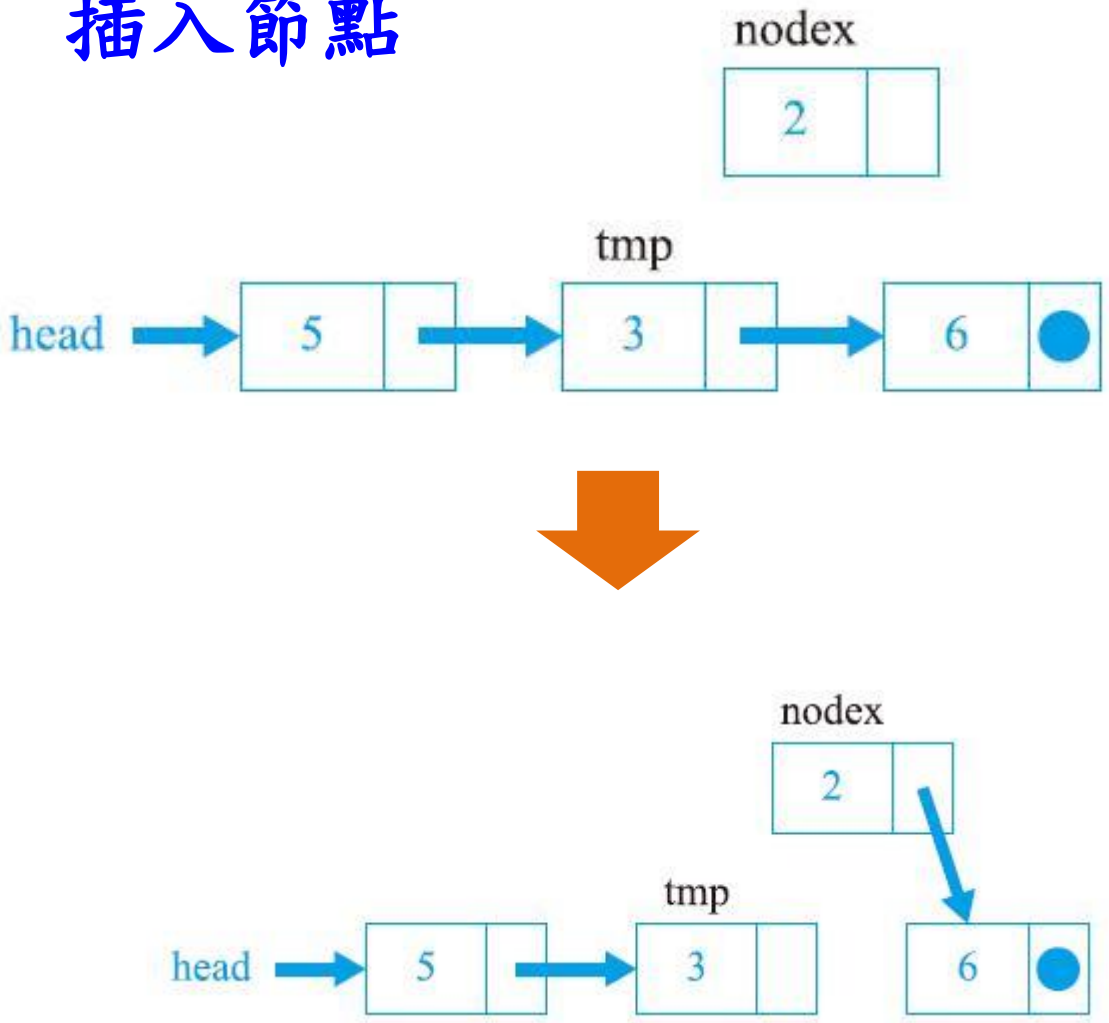
- 鏈結串列(Linked List)是使用Pointer(指標)串接資料。
- 使用鏈結串列的好處是找到指定位置後，可以有效率地插入或刪除元素，陣列不適合在中間位置插入或刪除元素，因為需要花較多時間搬移元素，適合在兩端插入或刪除元素。
- 鏈結串列不能隨機讀取指定位置的元素，只能從前往後一個一個走到指定的位置才能讀取，而陣列可以使用索引值(隨機)讀取陣列中指定位置的元素。

鏈結串列 (2/2)

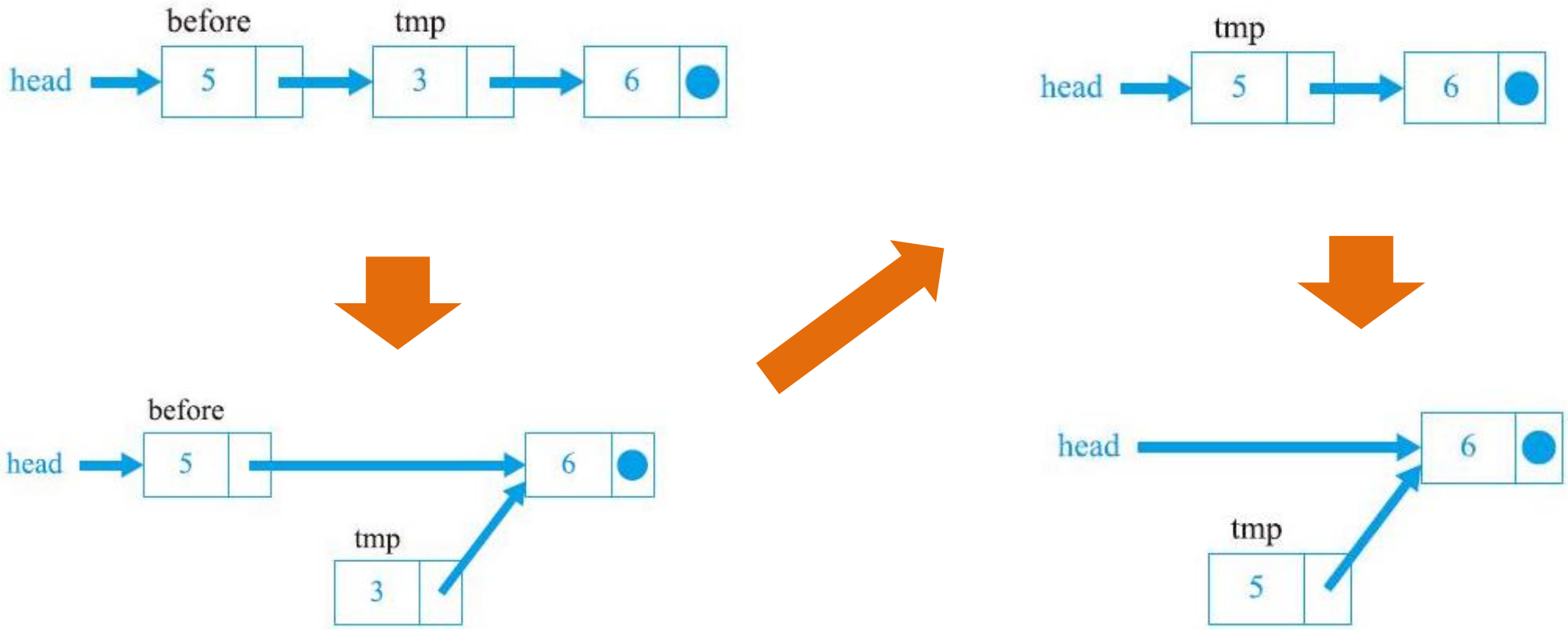
- 指標head指向鏈結串列的第一個元素5，元素5的指標指向下一個元素3，元素3的指標指向下一個元素2，元素2的指標指向下一個元素6，元素6的指標沒有下一個元素，使用●表示None表示鏈結串列到達終點。
- 鏈結串列需要一個指標，指向下一個元素。
- 若下一個元素是空的時候設定為None，None就是空指標，鏈結串列走訪時遇到None，就不會再走訪下去。



插入節點

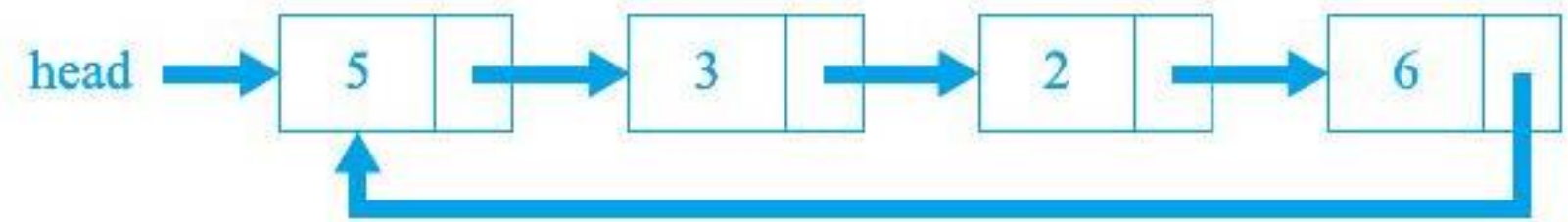


刪除節點

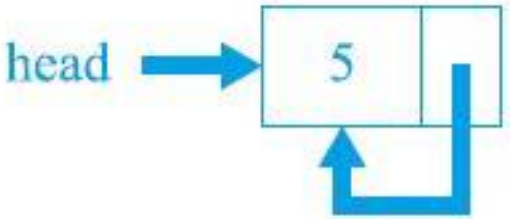

before
tmp
head
5
3
6
tmp
head
5
6
before
head
5
6
tmp
3
head
6
tmp
5

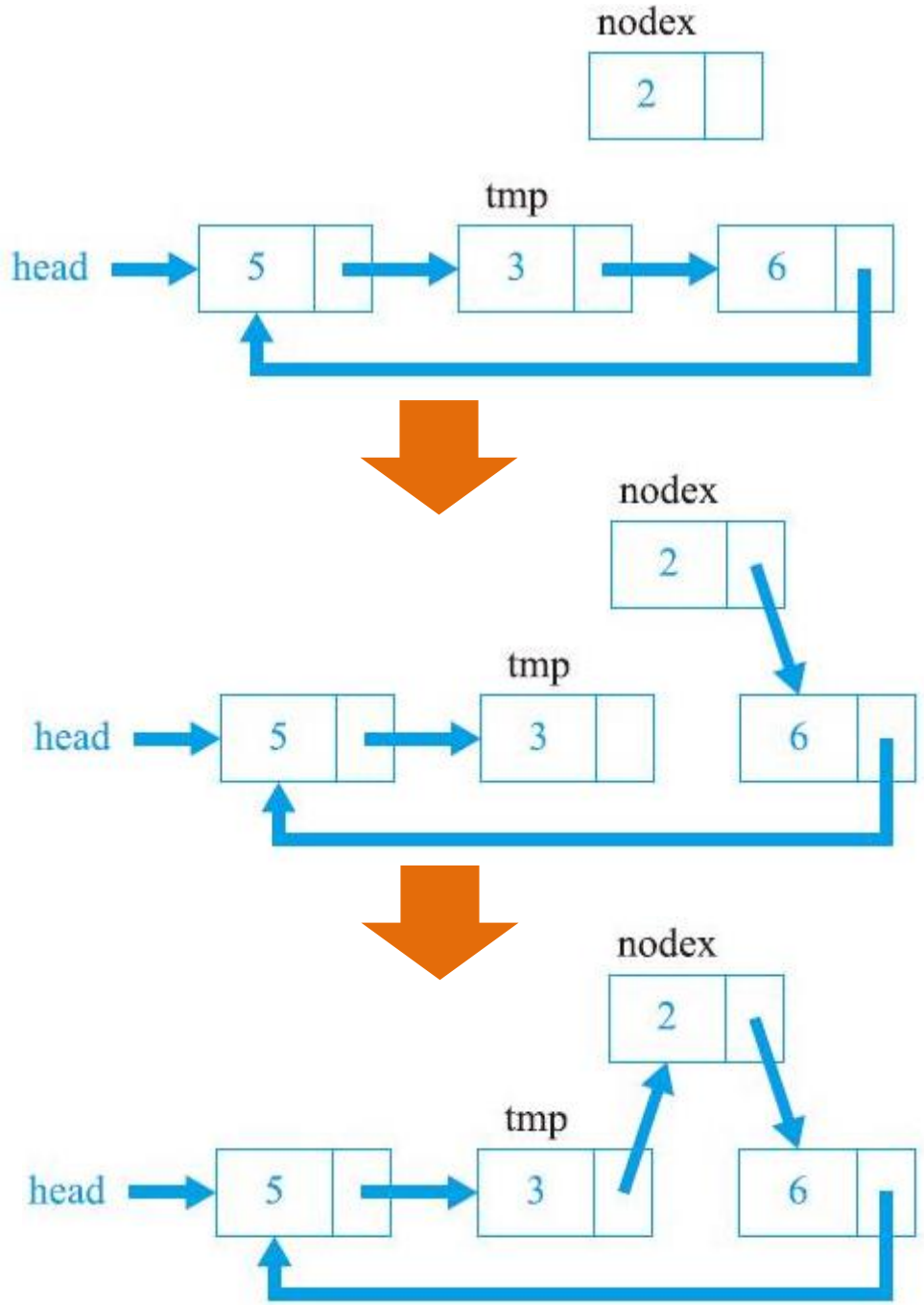
環狀鏈結串列

- 環狀鏈結串列(Circular Linked List)是使用Pointer(指標)串接資料，且最後一個元素可以連結到第一個元素。
- 使用環狀鏈結串列在找到指定位置後，可以在短時間插入或刪除元素。
- 環狀鏈結串列不能隨機讀取指定位置的元素，只能從前往後一個一個走到指定的位置才能讀取。

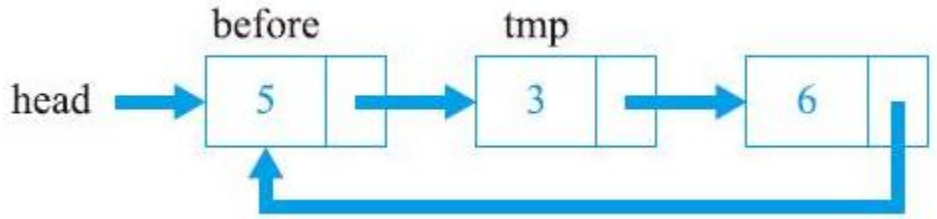


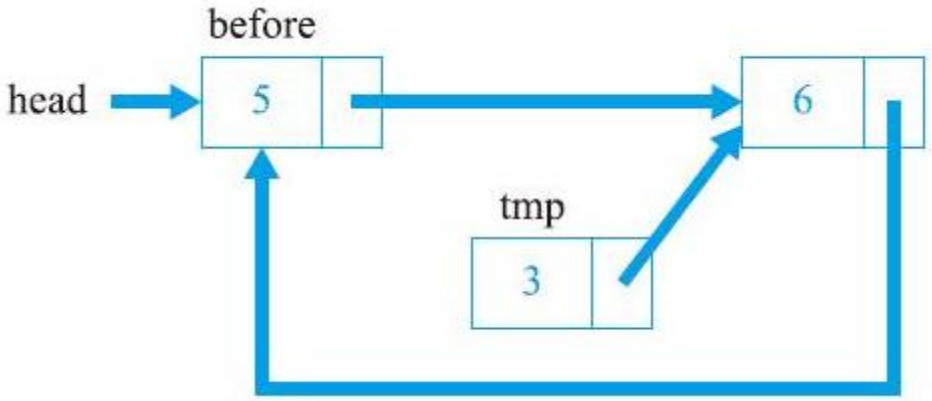
插入元素

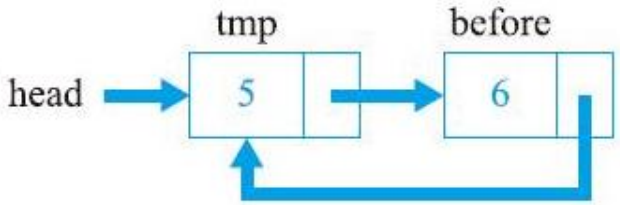

head
5

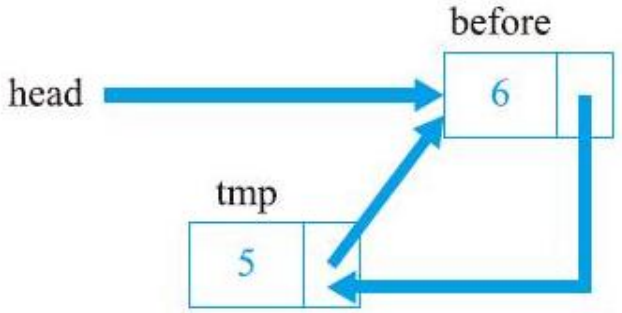

nodex
2
tmp
head
5
3
6
nodex
2
tmp
head
5
3
6
nodex
2
tmp
head
5
3
6

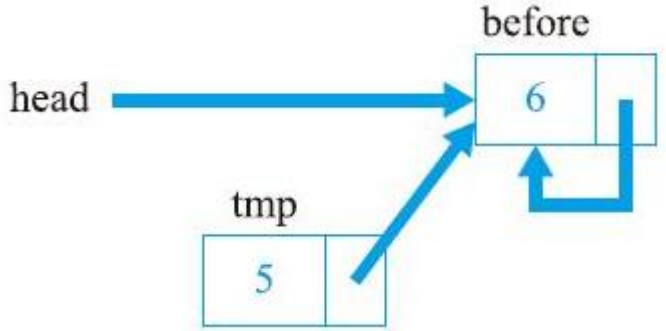
刪除元素

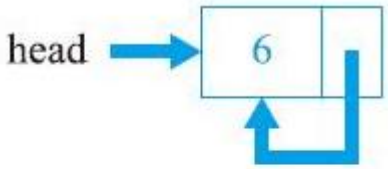

before
tmp
head
5
3
6

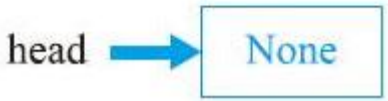

before
head
5
6
tmp
3


tmp
before
head
5
6


before
head
6
tmp
5

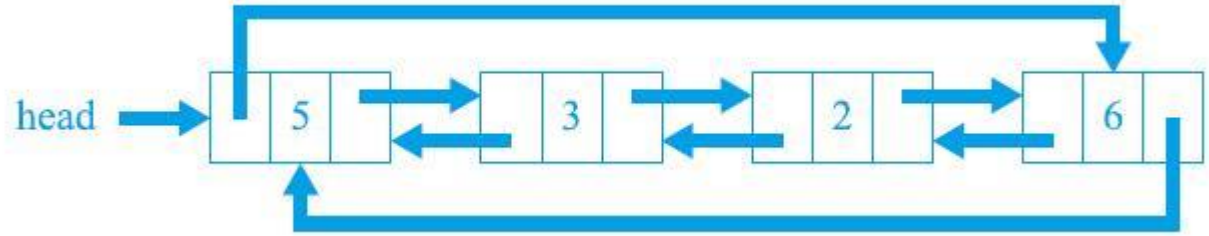

before
head
6
tmp
5


head
6

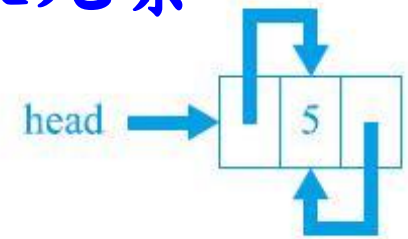

head
None

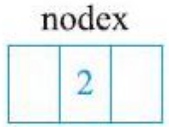
雙向鏈結串列

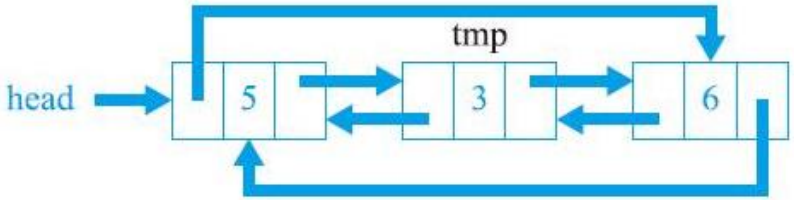
- 雙向鏈結串列(Double Linked List)是使用Pointer(指標)串接資料。
- 使用雙向鏈結串列的好處是找到指定位置後，可以很有效率地插入或刪除元素，且很容易找到節點的前一個元素與下一個元素。
- 雙向鏈結串列不能隨機讀取指定位置的元素，只能從前往後或從後往前一個一個走到指定的位置才能讀取。

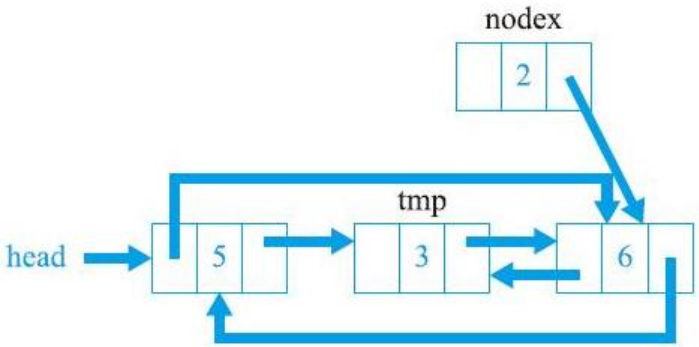


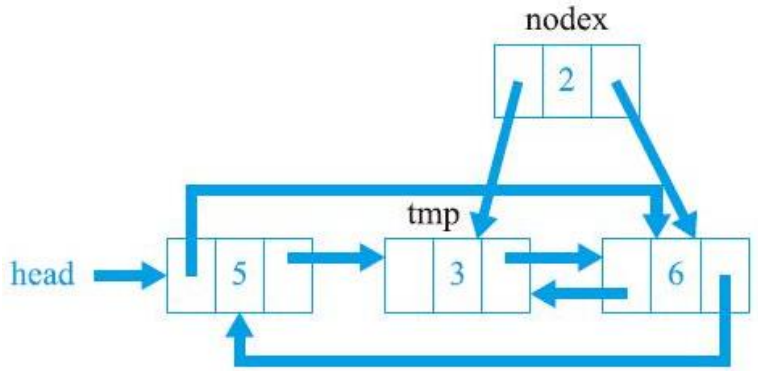
插入元素

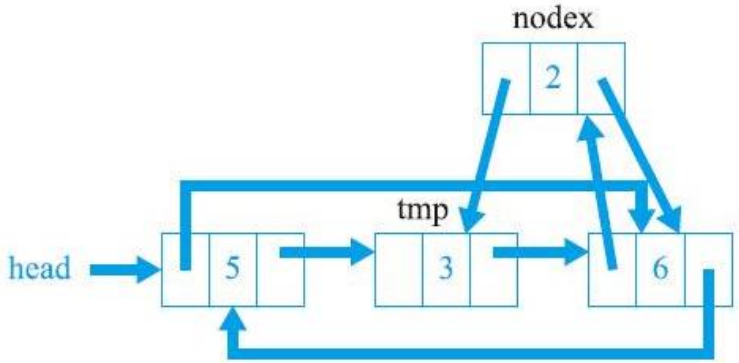

head
5

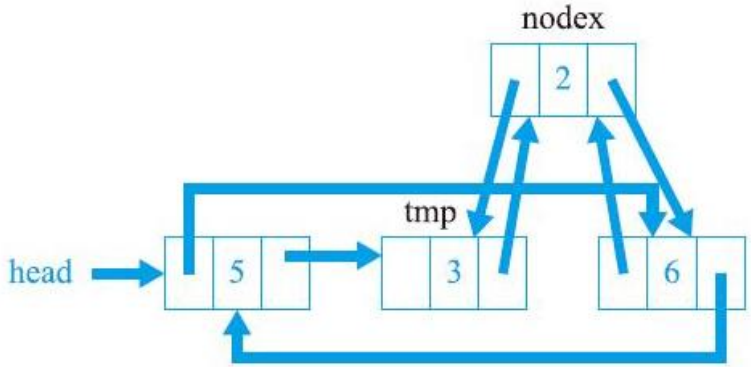

nodex
2


tmp
head
5
3
6

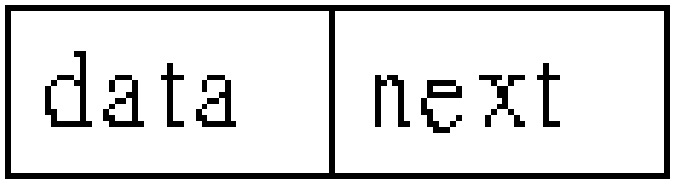

nodex
2
tmp
head
5
3
6

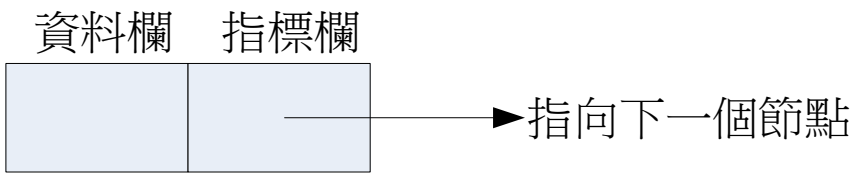

nodex
2
tmp
head
5
3
6

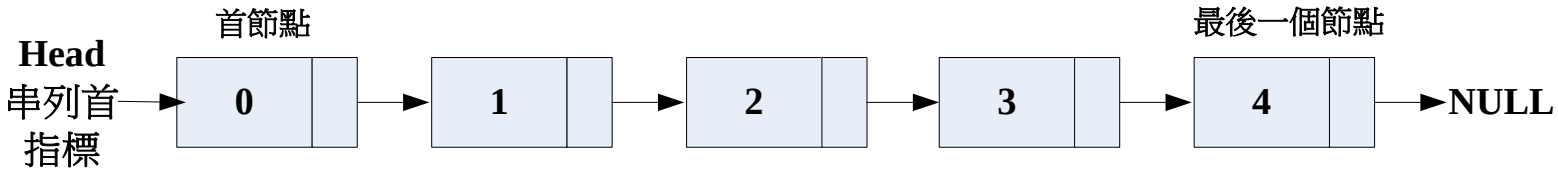

nodex
2
tmp
head
5
3
6


nodex
2
tmp
head
5
3
6

單向鏈結串列 (1/3)

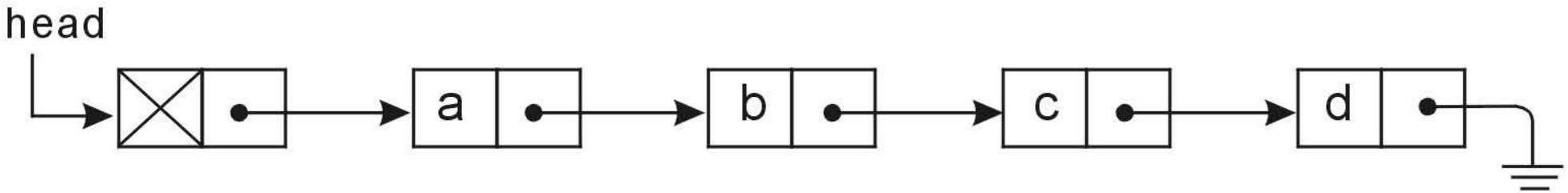




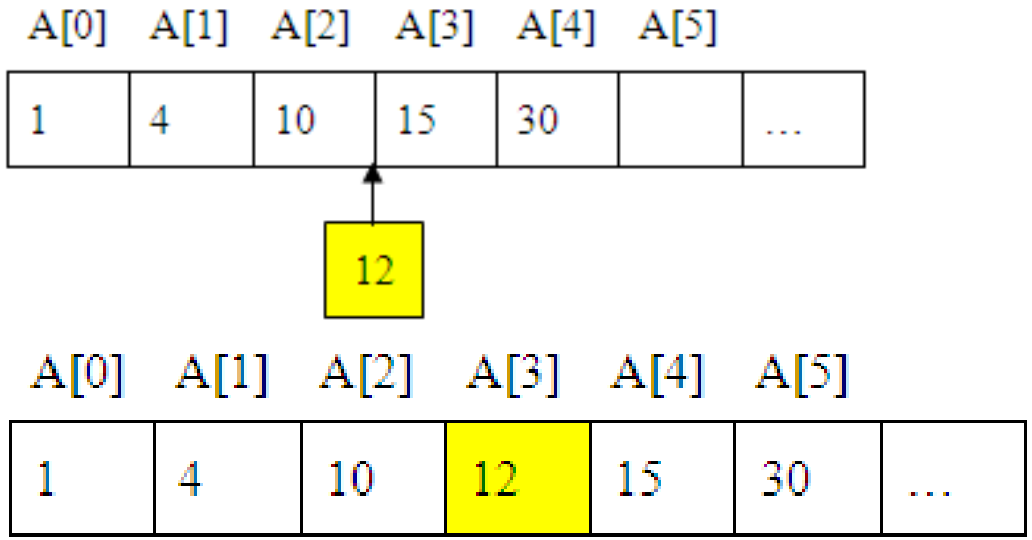


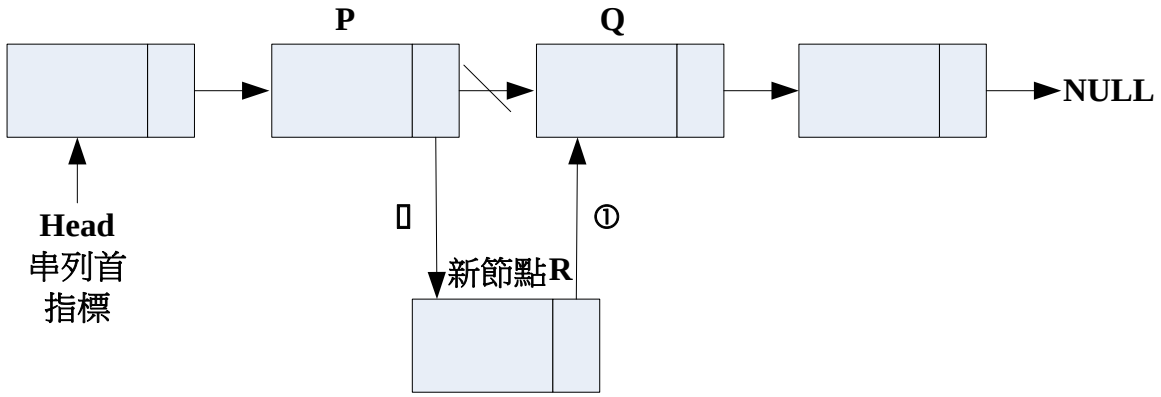
單向鏈結串列 (2/3)

- 如串列 A={a, b, c, d}，其資料結構如下：



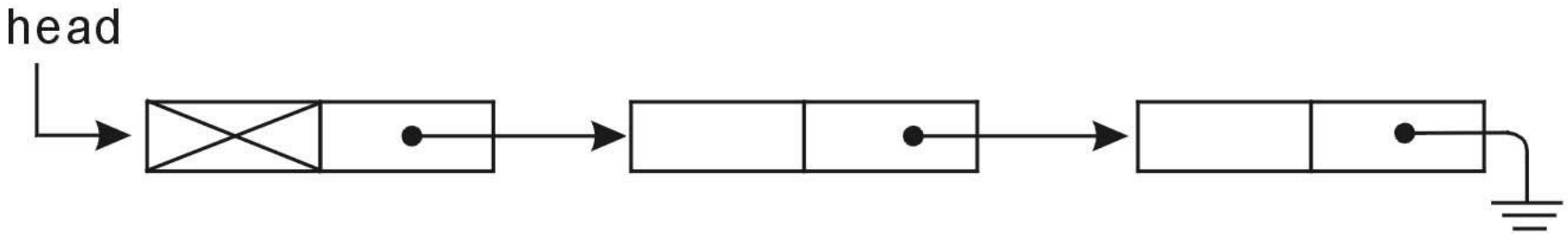
單向鏈結串列 (3/3)


A[0] A[1] A[2] A[3] A[4] A[5]
1 4 10 15 30 …
12
A[0] A[1] A[2] A[3] A[4] A[5]
1 4 10 12 15 30 …


P
Q
NULL
Head
串列首
指標
新節點R
①

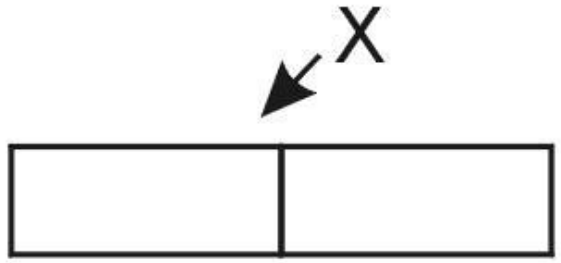
單向鏈結串列-加入前端 (1/4)

- 加入於串列的前端
  - 假設有一串列如下：



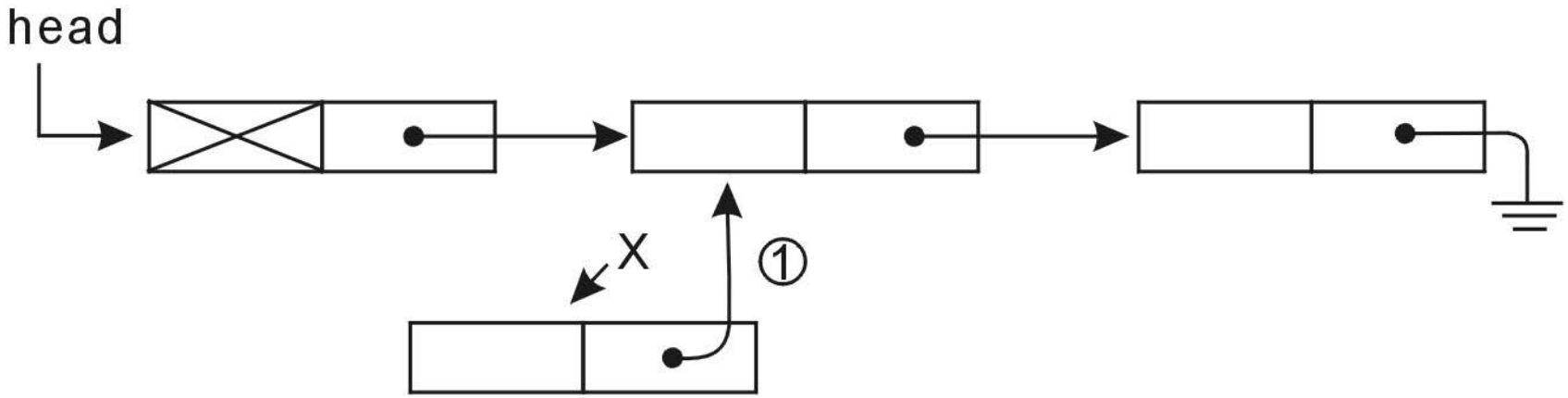
單向鏈結串列-加入前端 (2/4)

- 有一節點x將加入於串列的前端，其步驟如下：
  - x=(struct node*) malloc(sizeof(struct node));



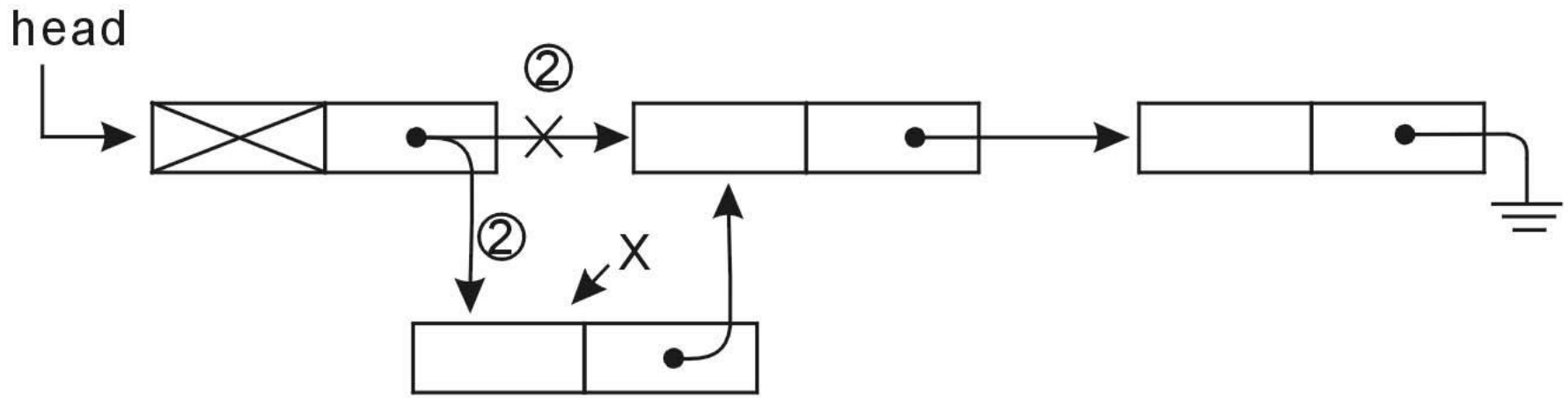
單向鏈結串列-加入前端 (3/4)

- (x->next=head->next; /* ① */



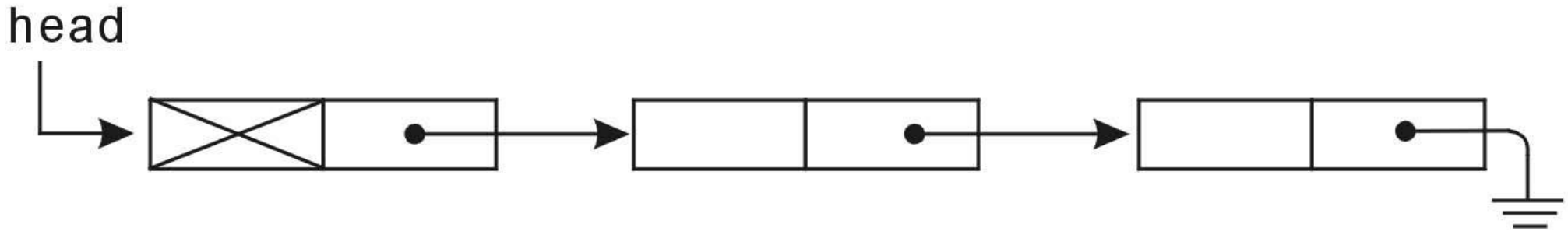
單向鏈結串列-加入前端 (4/4)

- **head->next=x;　　/* ② */**



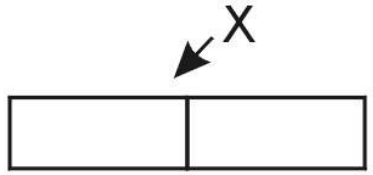
單向鏈結串列-加入尾端 (1/4)

- 有一節點x將加入於串列的尾端，其步驟如下：

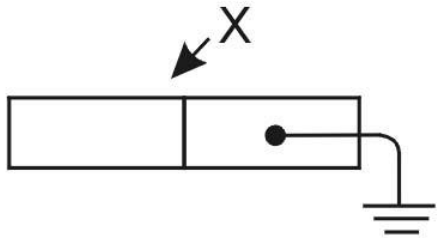


單向鏈結串列-加入尾端 (2/4)

- x=(struct node *)malloc(sizeof(struct node));

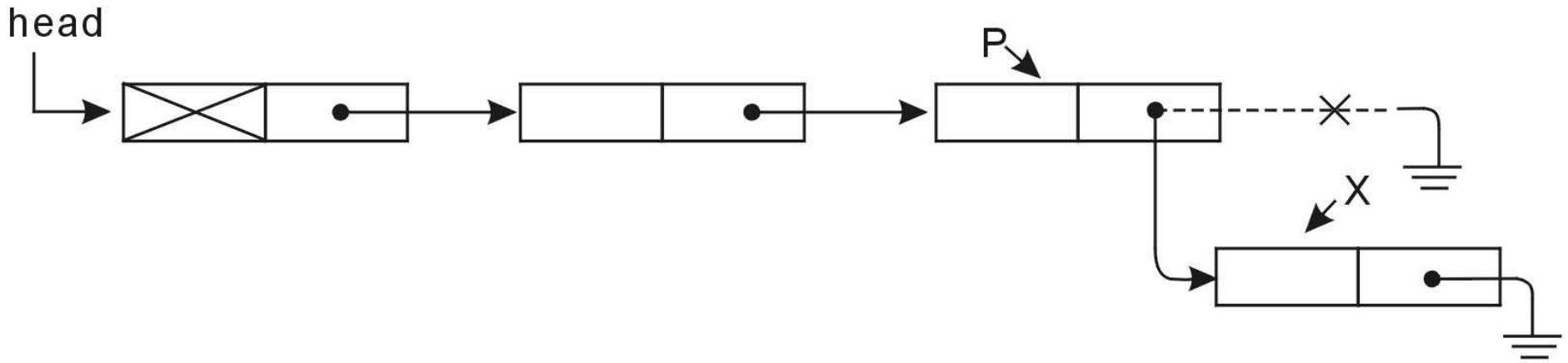


- x->next=NULL;

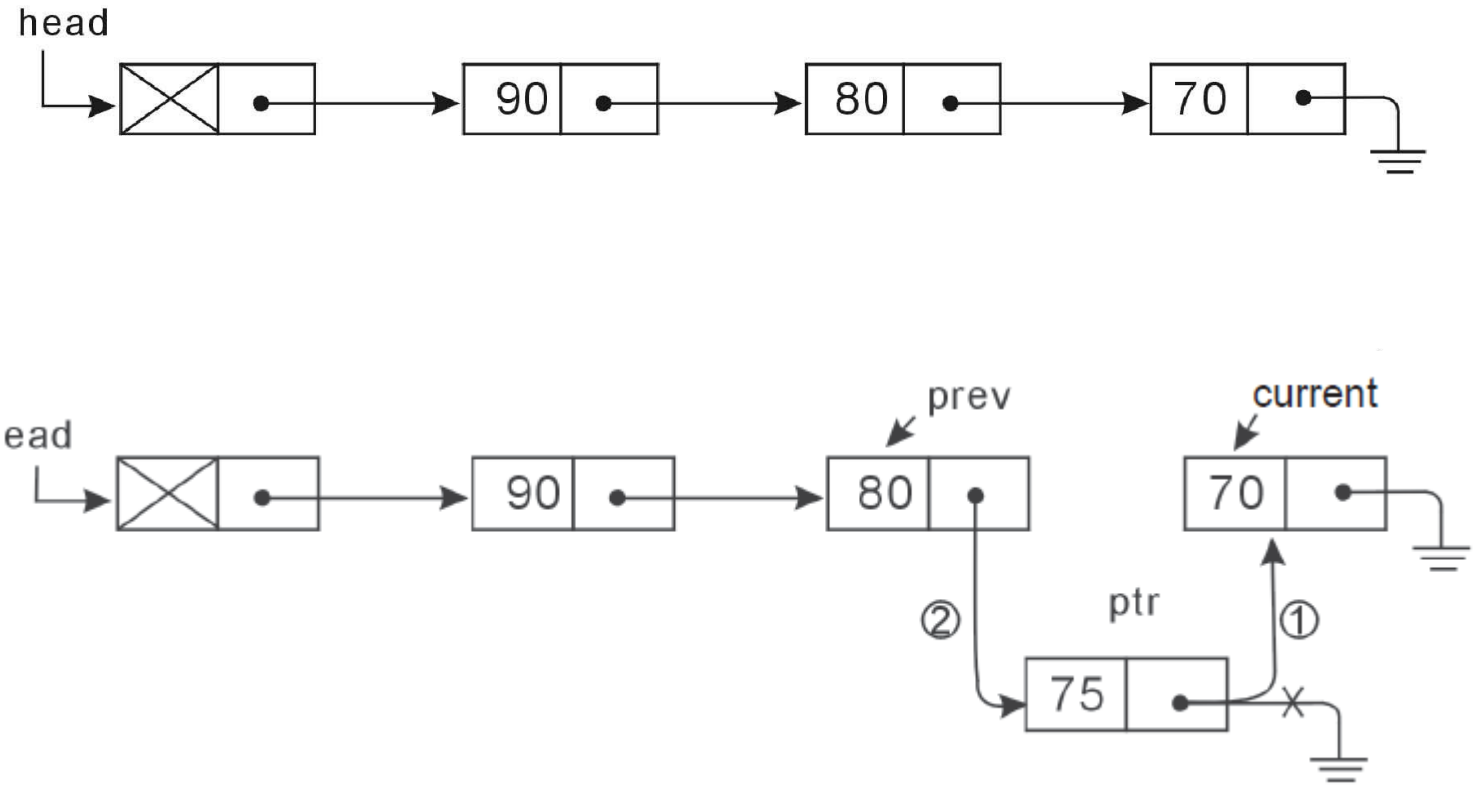


單向鏈結串列-加入尾端 (3/4)

- **p->next=x;**

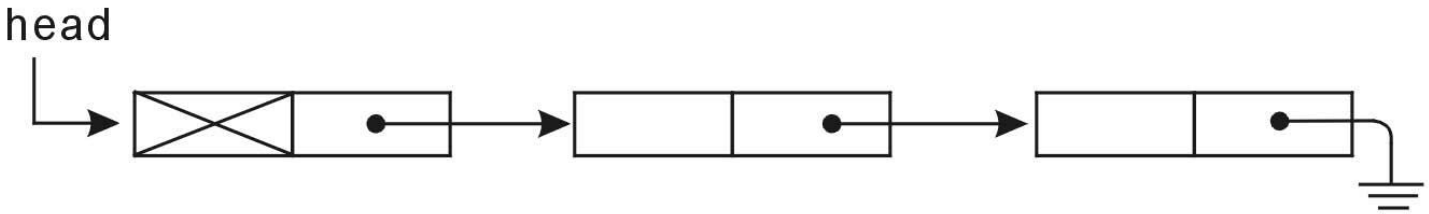


單向鏈結串列-加入尾端 (4/4)



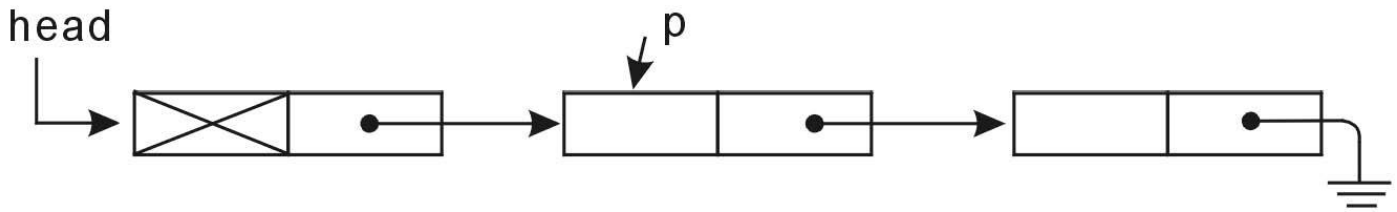
單向鏈結串列-刪除 (1/8)

- 刪除動作

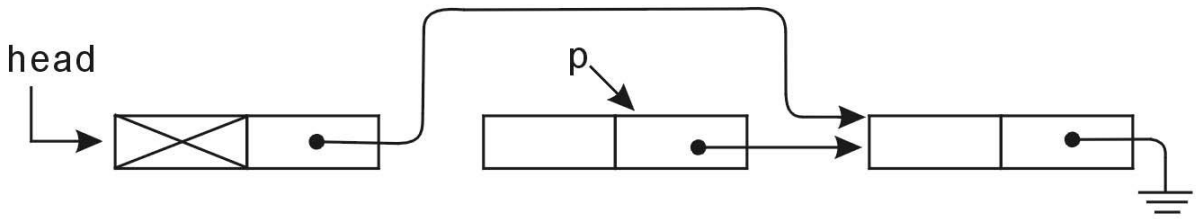


單向鏈結串列-刪除 (2/8)

- (1)p=head->next;

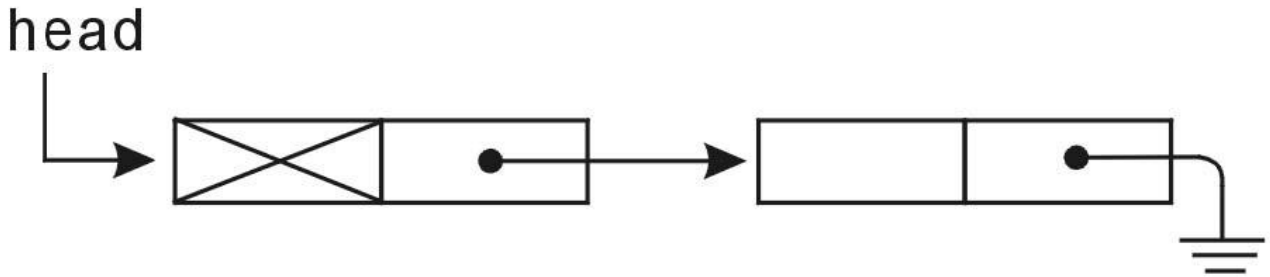


- (2)head->next=p->next;



單向鏈結串列-刪除 (3/8)

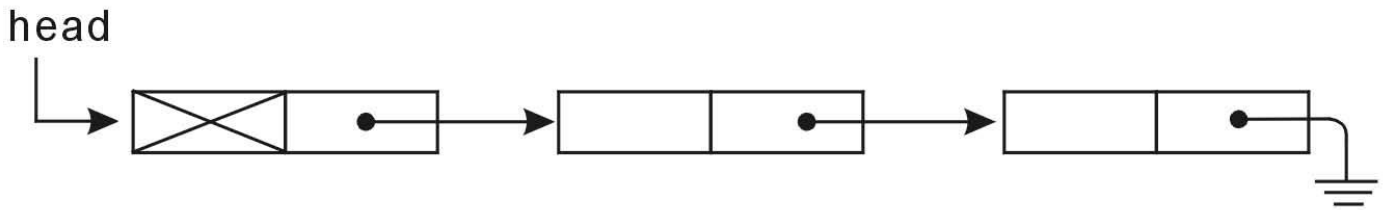
- (3)free(p);



經由free(p)便可將p節點回收。

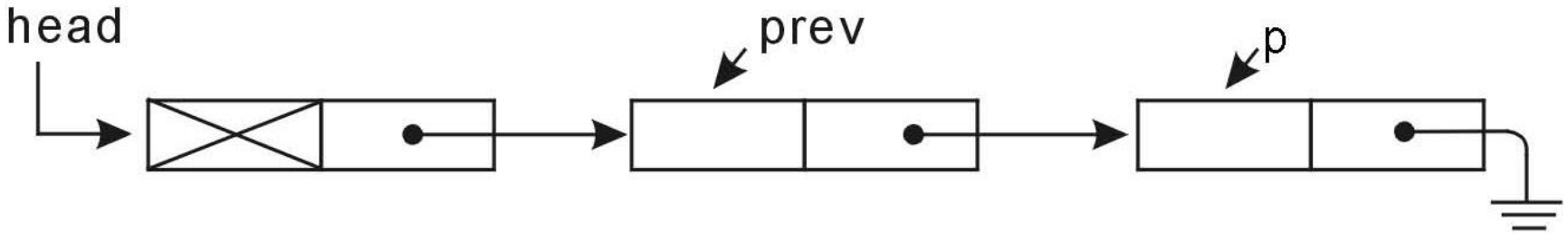
單向鏈結串列-刪除 (4/8)

- 刪除串列的最後節點：
  假設有一串列如下：



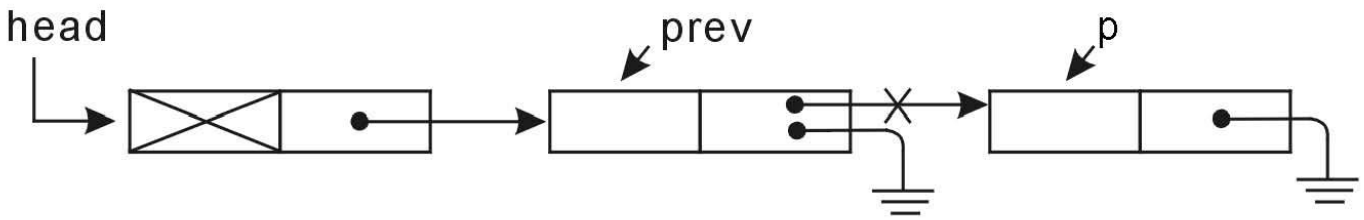
單向鏈結串列-刪除 (5/8)

- 必須先追蹤尾端及尾端的前一個節點在哪，步驟如下：
  - (1)p=head->next;

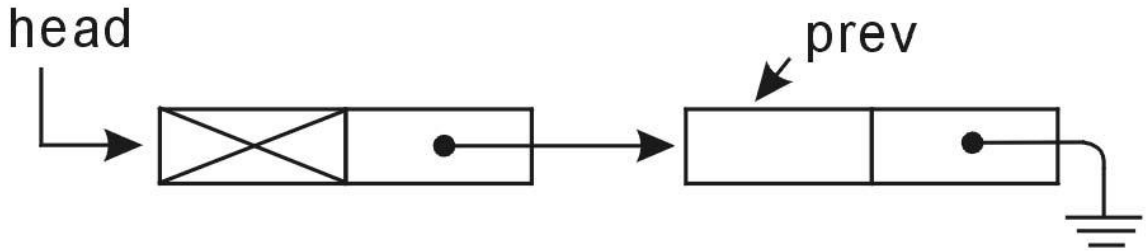


單向鏈結串列-刪除 (6/8)

- (2)prev->next=NULL;

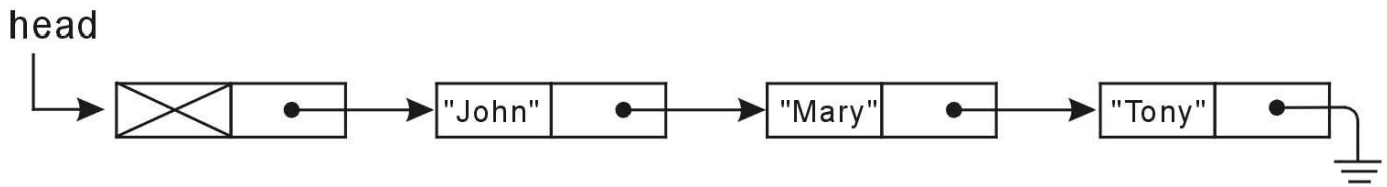


- (3)free(p);



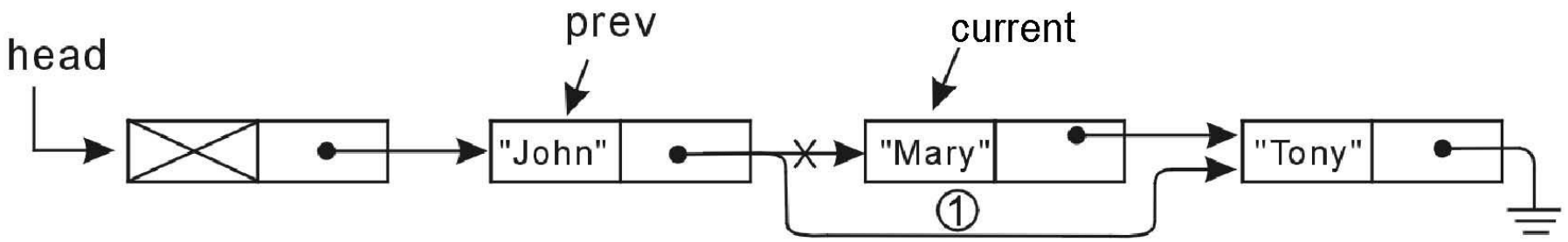
單向鏈結串列-刪除 (7/8)

- 刪除某一特定的節點：
  刪除某一特定的節點也必須利用二個指標current和prev，分別指到即將被刪除節點(current)及前一節點 (prev)，因此prev永遠跟著current。
- 假設有一單向鏈結串列如下：



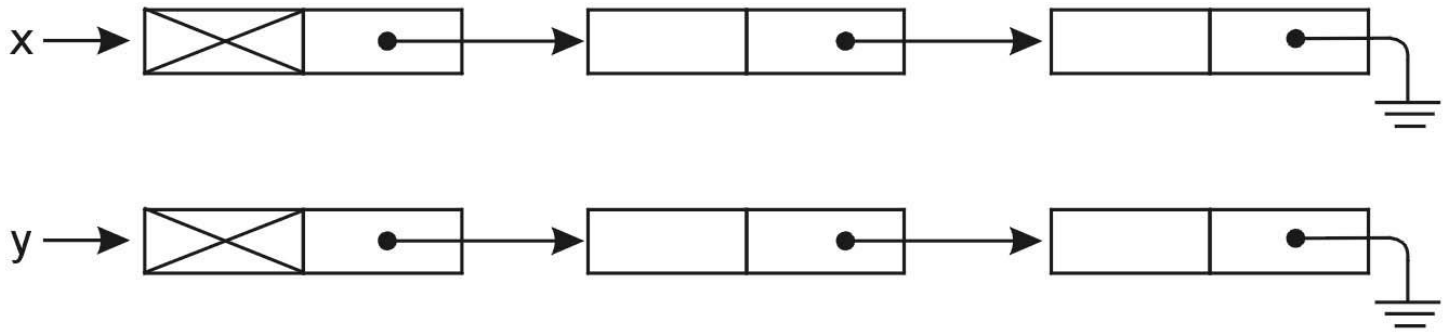
單向鏈結串列-刪除 (8/8)

- 欲刪除 “Mary”，因此將del_data變數指定為 “Mary”。



單向鏈結串列 (1/3)

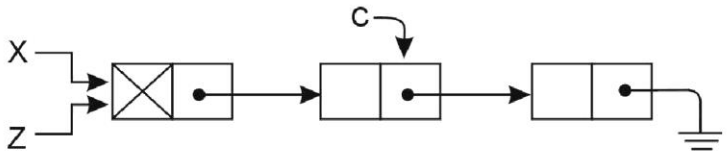
- 將兩個單向鏈結串列相互連接
  - 假設有二個串列如下：



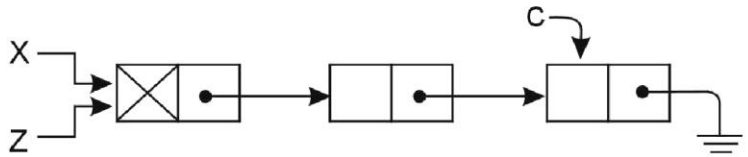
單向鏈結串列 (2/3)

- 將x與y串列合併為z串列，其步驟如下：
  - **if(x->next==NULL)**
    z=y;
    表示當 x 串列是空的時候，直接將 y 串列指定給 z 串列。
  - **if(y->next ==NULL)**
    z=x;
    表示當 y 串列是空的時候，直接將 x 串列指定給 z 串列。
  - **z=x;**
    c=x->next;

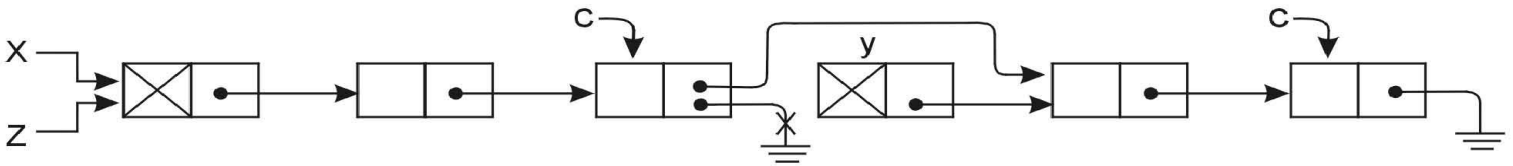
單向鏈結串列 (3/3)




```
while(c->next != NULL)
        c=c->next;
```




```
c->next=y->next;
```



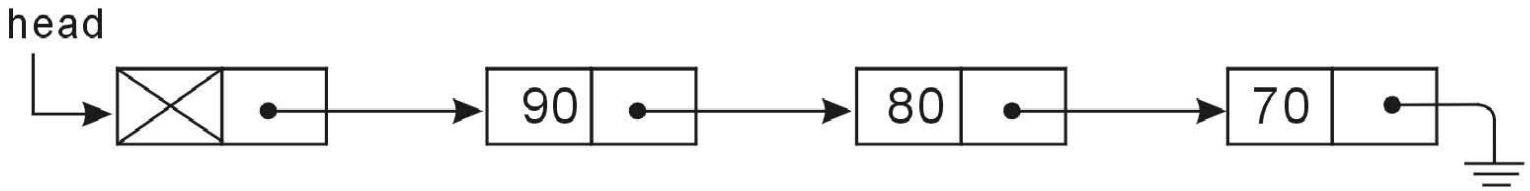

```
free(y);
```

單向鏈結串列-串列反轉 (1/6)

- 串列反轉
  - 顧名思義，串列的反轉(invert)乃將原先的串列首變為串列尾，同理，串列尾變為串列首。
  - 若有一串列乃是由小到大排列，此時若想由大到小排列，只要將串列反轉即可。

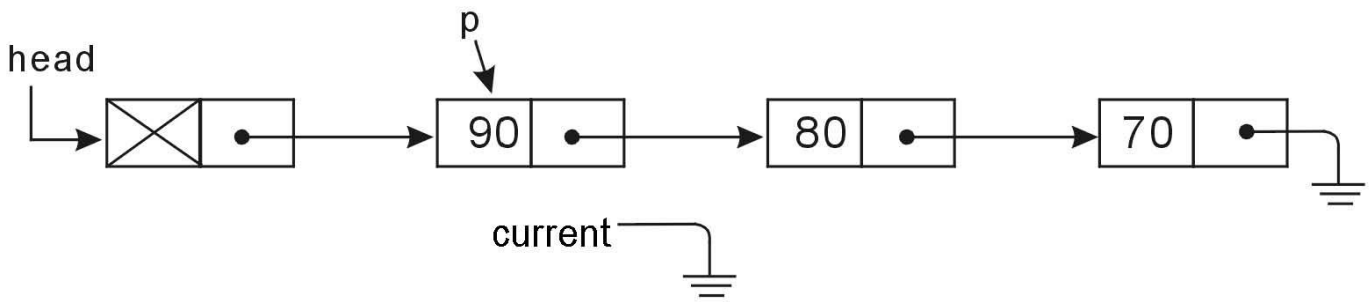
單向鏈結串列-串列反轉 (2/6)

- 假設有一串列如下：



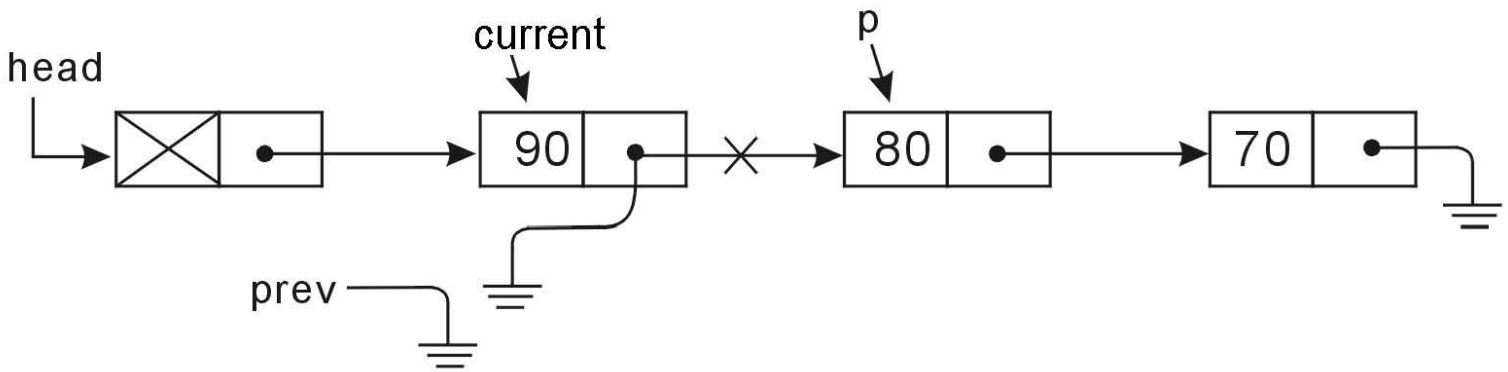
單向鏈結串列-串列反轉 (3/6)

- 經由下面幾個步驟完成反轉：
  - (1)p=head->next;
  - (2)current=NULL;



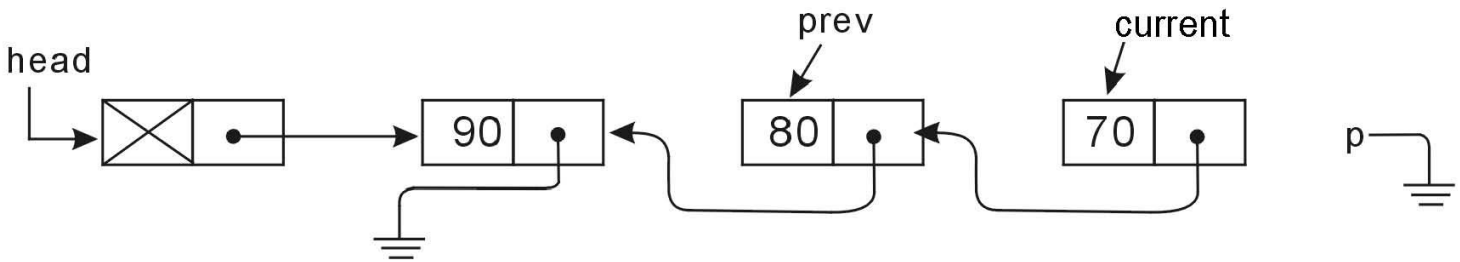
單向鏈結串列-串列反轉 (4/6)

- 此迴圈的前三個敘述p指標，current指標和prev指標有先後順序。經過一次的動作後，串列如下：



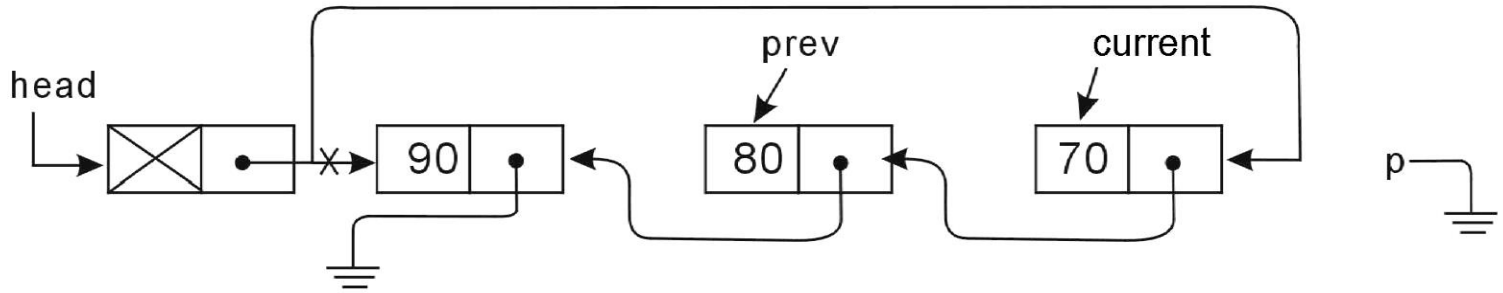
單向鏈結串列-串列反轉 (5/6)

- 依此進行到p == NULL後，迴圈停止



單向鏈結串列-串列反轉 (6/6)

- 最後，利用
  head->next=current;



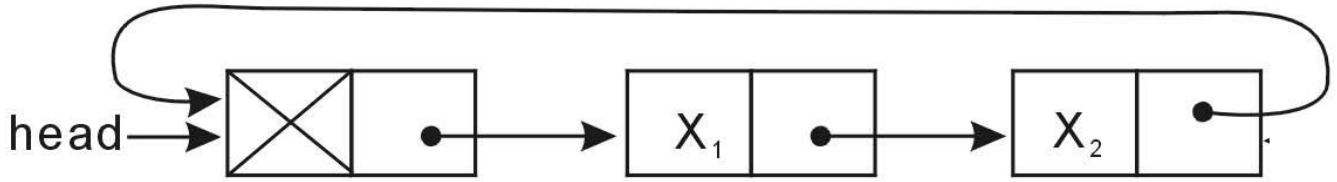
- 完成串列的反轉重點在於需要三個指標才能達成任務。

環狀鏈結串列(Circular Linked List)

- 環狀鏈結串列(Circular Linked List)是使用Pointer(指標)串接資料，且最後一個元素可以連結到第一個元素。
- 使用環狀鏈結串列的好處是找到指定位置後，可以在很短時間內插入或刪除元素；陣列不適合在中間位置插入或刪除元素，因為需要花較多時間搬移元素，適合在兩端插入或刪除元素。
- 環狀鏈結串列不能隨機讀取指定位置的元素，只能從前往後一個一個走到指定的位置才能讀取；而陣列可以使用索引值(隨機)讀取陣列中指定位置的元素。

環狀鏈結串列(Circular Linked List)

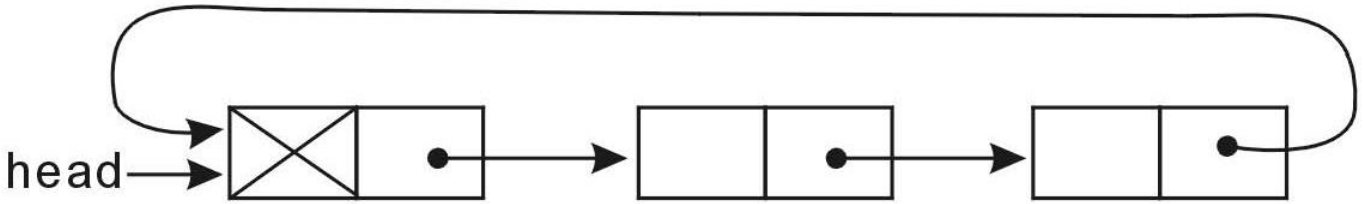
- 將鏈結串列最後一個節點的指標指向head節點時，此串列稱為環狀串列(circular list)，如下圖所示：



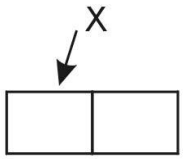
- 環狀串列可以從任一節點來追蹤所有節點。

環狀串列-加入 (1/3)

- 加入動作
- 有一環狀串列如下：



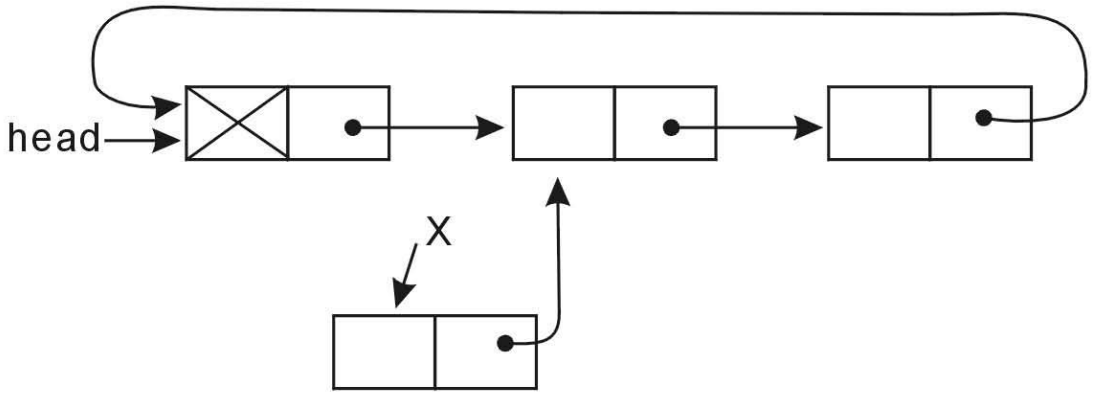
- 利用malloc函數配置了一個節點



環狀串列-加入 (2/3)

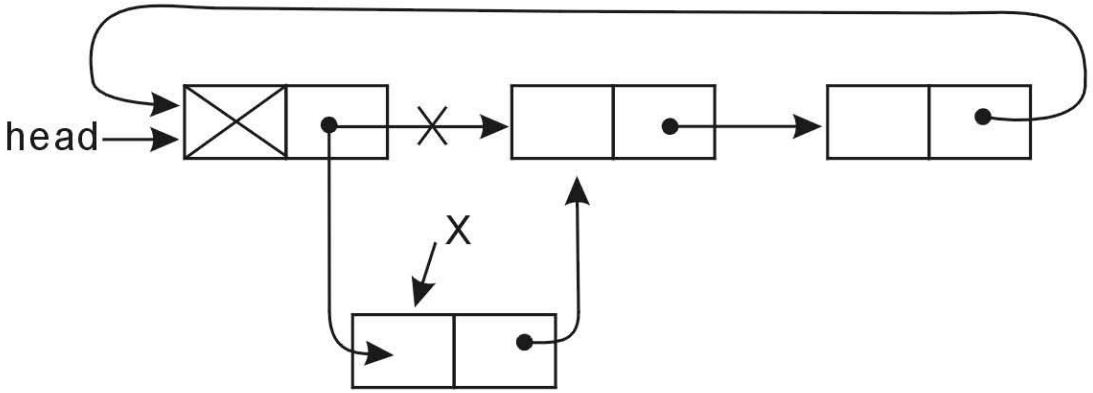
- 利用下列步驟將x指向的節點加入到環狀串列的前端。

(1) x->next=head->next;



環狀串列-加入 (3/3)

(2) head->next=x;



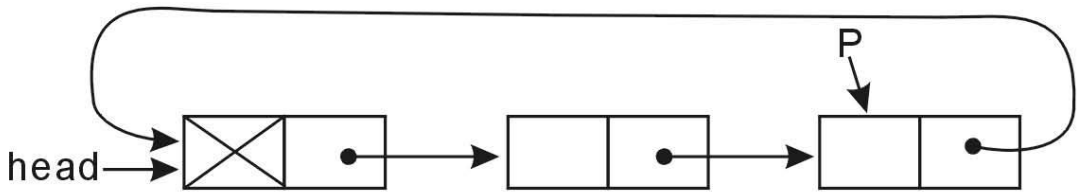
- 上述(1)(2)步驟亦適用於環狀串列開始時是空的狀況。

環狀串列-尾端加入 (1/2)

- 加入一節點於環狀串列的尾端

(1) 先找到尾端在那裏

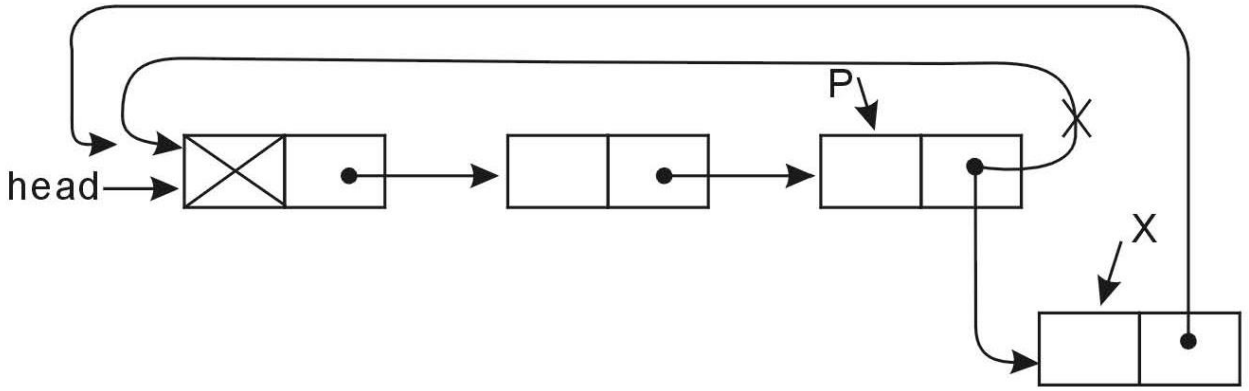
```
p=head->next;
while(p->next != head)
    p=p->next;
```



環狀串列-尾端加入 (2/2)

(2) p->next=x;

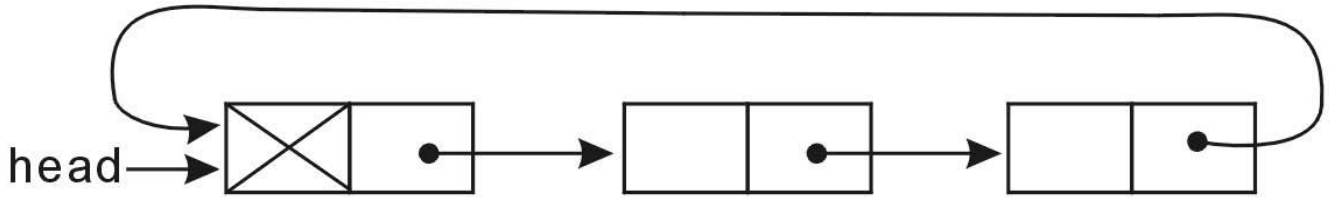
x->next=head;



- 在環狀串列的某一特定節點後加入一節點，此與單向鏈結串列相似。

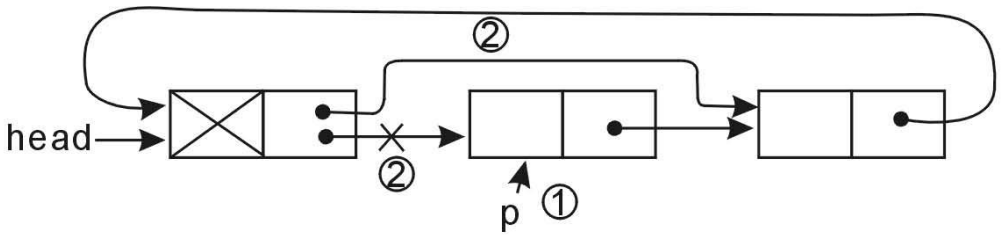
環狀串列-刪除前端 (1/2)

- 刪除的動作
- 刪除環狀串列的前端
  有一環狀的串列如下：

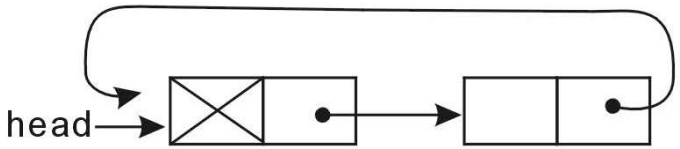


環狀串列-刪除前端 (2/2)

```
(1)  p=head->next;              /* ① */
     head->next=p->next;        /* ② */
```



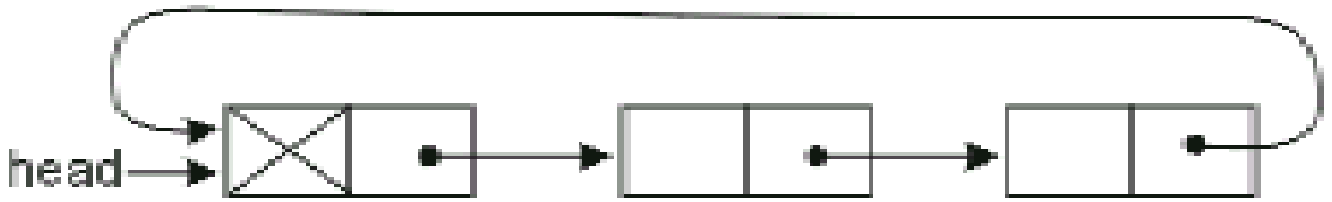

```
(2)  free(p);
```



回收p所指向的節點，此時環狀串列剩下二個節點。

環狀串列-刪除尾端 (1/3)

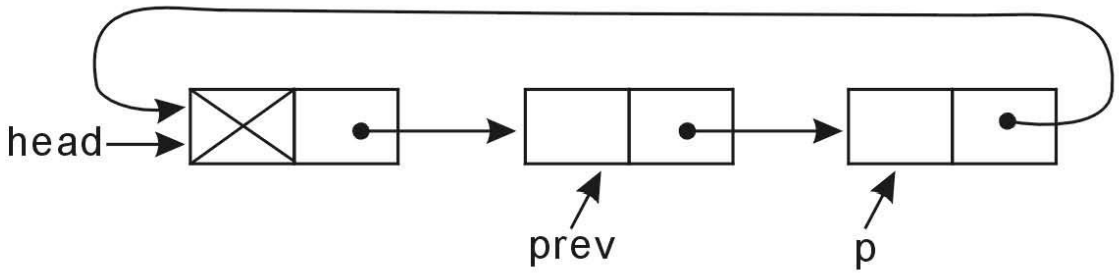
- 刪除環狀串列的尾端
  有一環狀串列如下：



環狀串列-刪除尾端 (2/3)

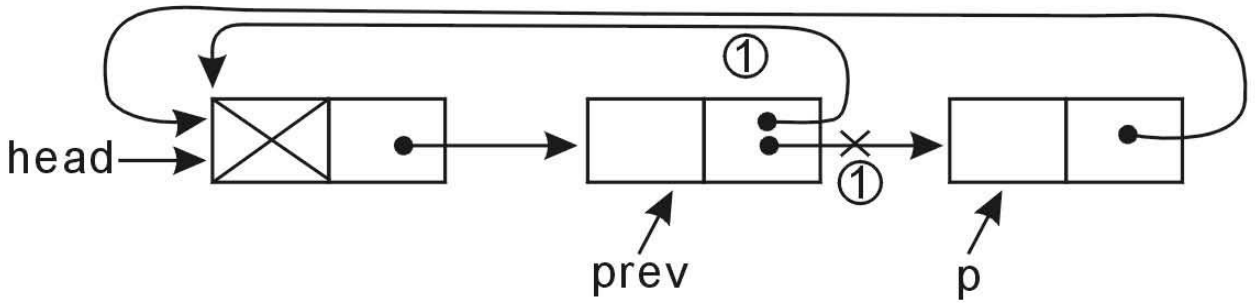
(1) 利用下列片段程式找到串列的尾端及尾端的前一節點

```
p=head->next;
while(p->next != head) {
        prev=p;
        p=p->next;
}
```

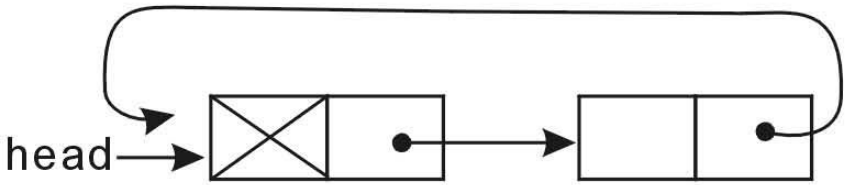


環狀串列-刪除尾端 (3/3)

(2) prev->next=p->next; /* ① */

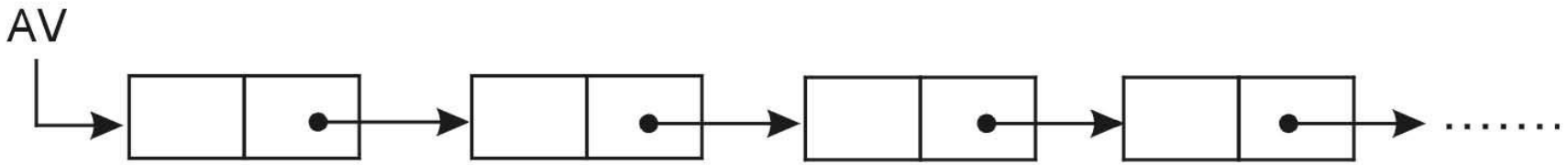


(3) free(p);



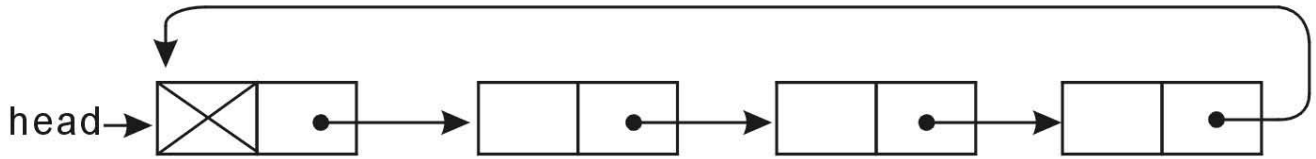
環狀串列-回收 (1/3)

- 回收整個環狀串列表示此串列皆不再需要了，因此將它歸還給系統，假設系統有一串列如下：



環狀串列-回收 (2/3)

- 而不再需要的環狀串列為



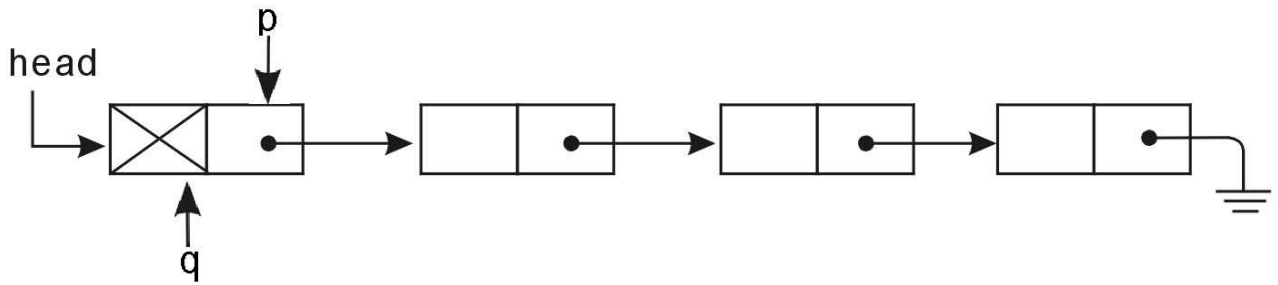

```
p=head->next;          /* ① */
head->next=AV;         /* ② */
AV=p;                  /* ③ */
```

- 只要下面三個步驟就可達成回收整個環狀串列。

環狀串列-回收 (3/3)

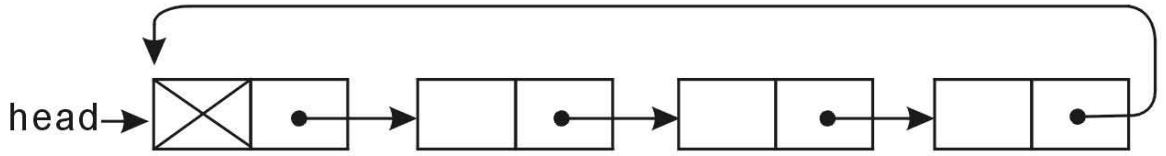
- 若要回收整個單向鏈結串列，必須一個一個回收，因此其Big-O為O(n)，表示與串列的節點數成正比，而回收整個環狀串列其Big-O為O(1)，表示它是一常數，不受節點數的影響。

```
p=head;
while(p != NULL){
    q=p;
    p=p->next;
    free(q);
}
```



環狀串列-計算長度

- 計算環狀串列的長度，與計算單向鏈結串列的長度類似。




```
p=head->next;
while (p != head) {
        count++
        p=p->next;
}
```

雙向鏈結串列(Double Linked List)

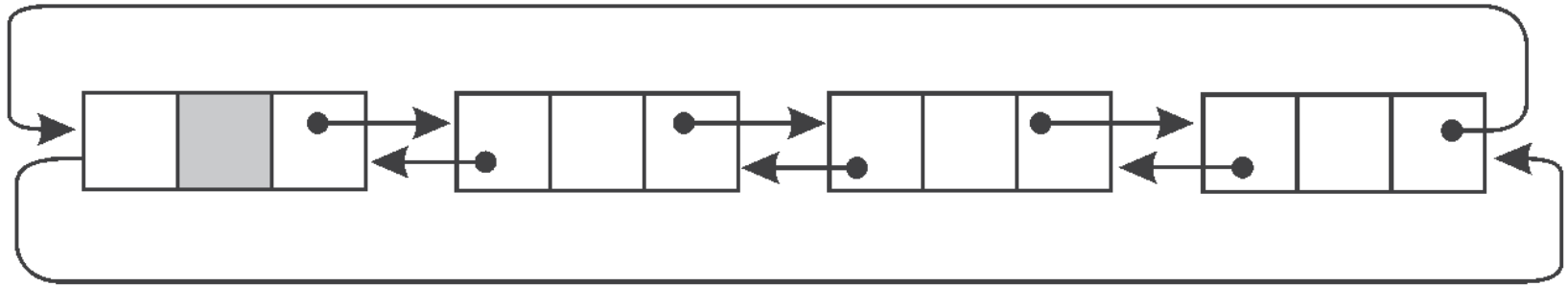
- 雙向鏈結串列(Double Linked List)是使用Pointer(指標)串接資料。
- 使用雙向鏈結串列的好處是找到指定位置後，可以很有效率地插入或刪除元素，且很容易找到節點的前一個元素與下一個元素。
- 陣列不適合在中間位置插入或刪除元素，因為需要花較多時間搬移元素，適合在兩端插入或刪除元素。

雙向鏈結串列 (1/3)

- 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）
- 資料結構如下：

雙向鏈結串列 (2/3)

- 其中LLINK指向前一節點，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。

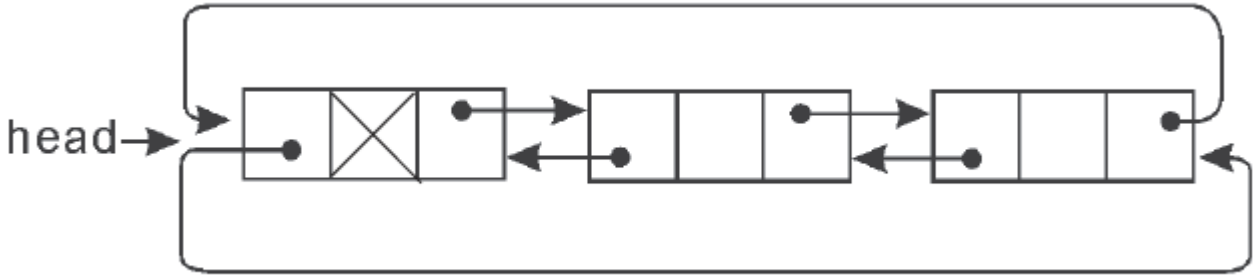


雙向鏈結串列 (3/3)

- 雙向鏈結串列具有下列兩點特性：
  - 假設ptr是指向任何節點的指標，則
    ptr == ptr->llink->rlink ==ptr->rlink->llink;
  - 若此雙向鏈結串列是空的串列，則只有一個串列首。

雙向鏈結串列-加入動作 (1/3)

- 加入於雙向鏈結串列的前端
  假設一開始雙向鏈結串列如下：

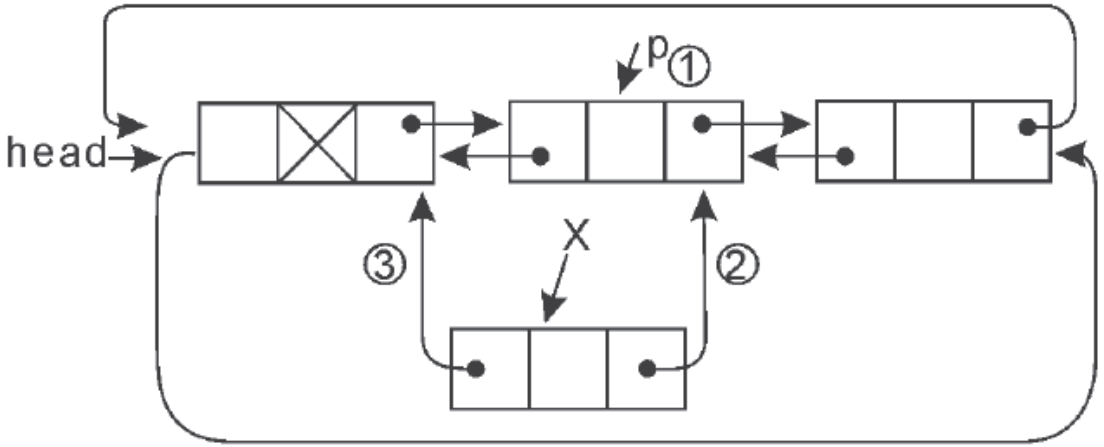


雙向鏈結串列-加入動作 (2/3)

- 經由下列步驟就可完成將已配置的x節點加入前端
- (1) p=head->rlink; /* ① */

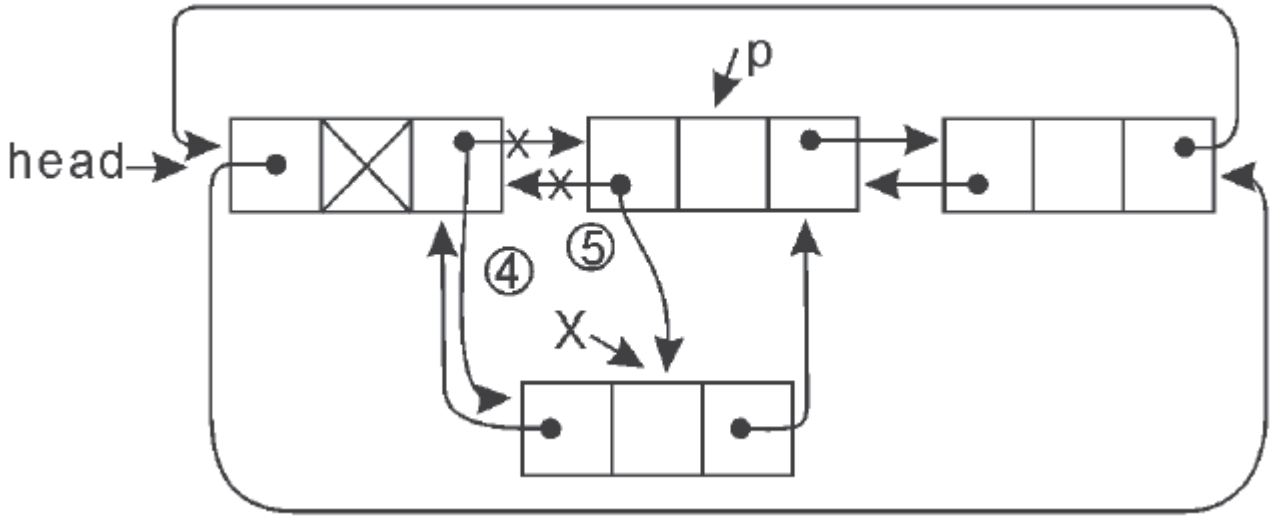
  x->rlink=p; /* ② */

  x->llink=head; /* ③ */



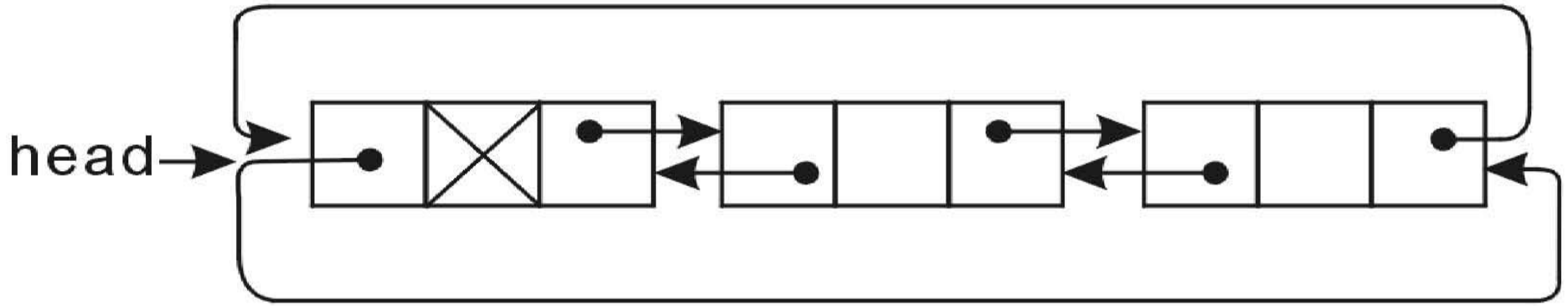
雙向鏈結串列-加入動作 (3/3)

- (2) head->rlink=x; /* ④ */
- p->llink=x; /* ⑤ */



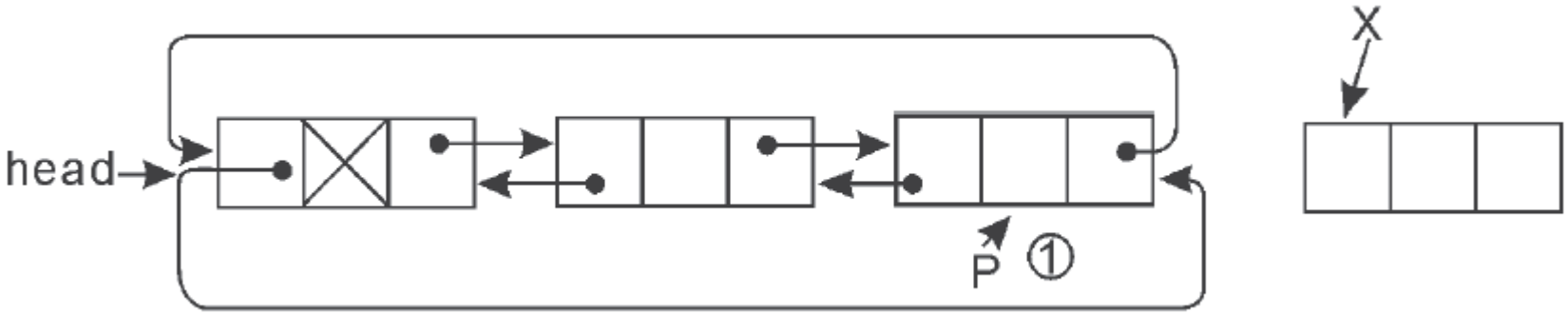
雙向鏈結串列-尾端加入 (1/4)

- 假設有一雙向鏈結串列如下：



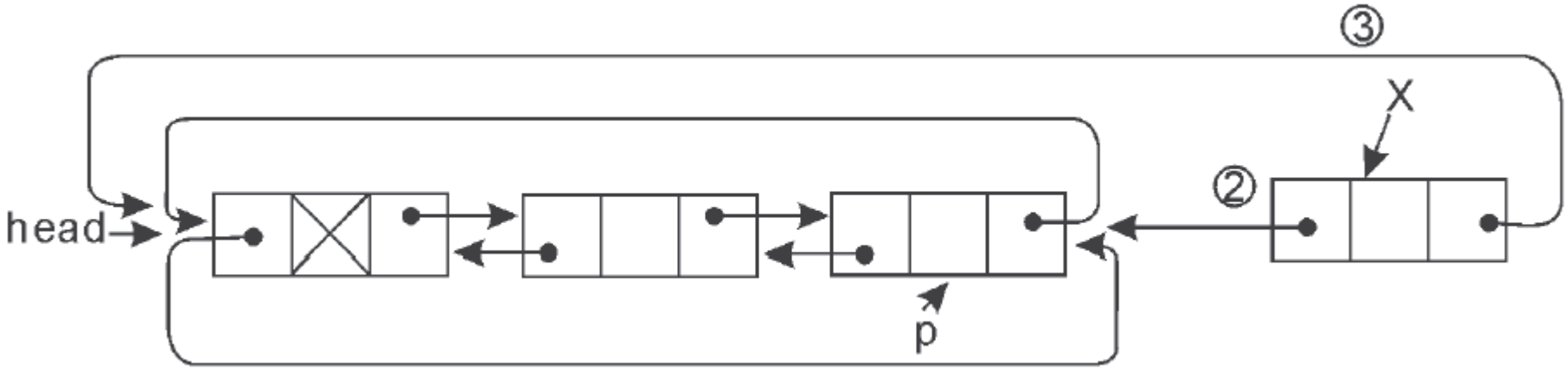
雙向鏈結串列-尾端加入 (2/4)

(1) p=head->llink; /* ① */



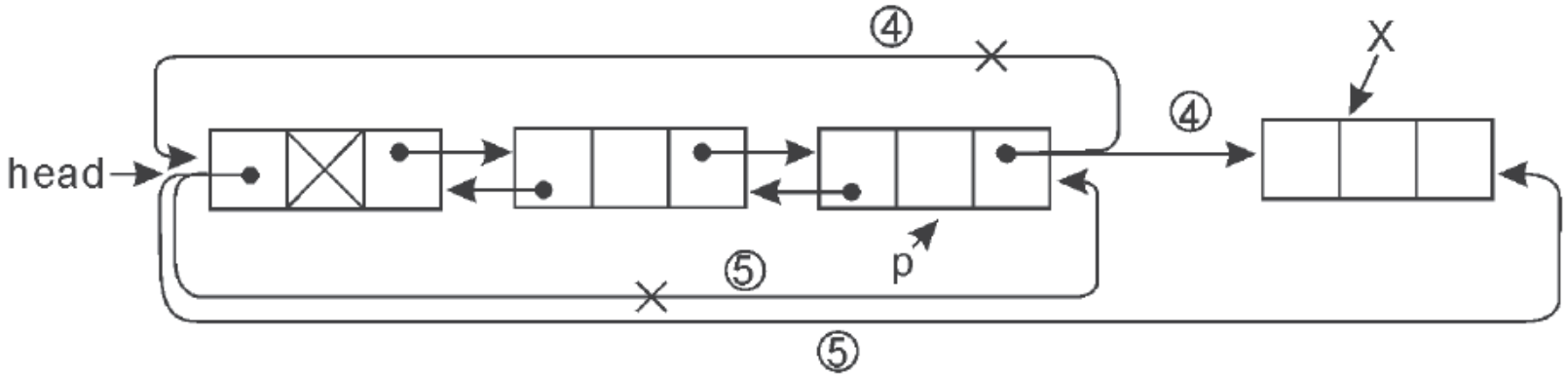
雙向鏈結串列-尾端加入 (3/4)

- (2) x->llink=p;　/* ② */
- 　　x->rlink=head;　/* ③ */
  - 此步驟乃先將x 的左、右link 欄位指向某一節點，情形如下：



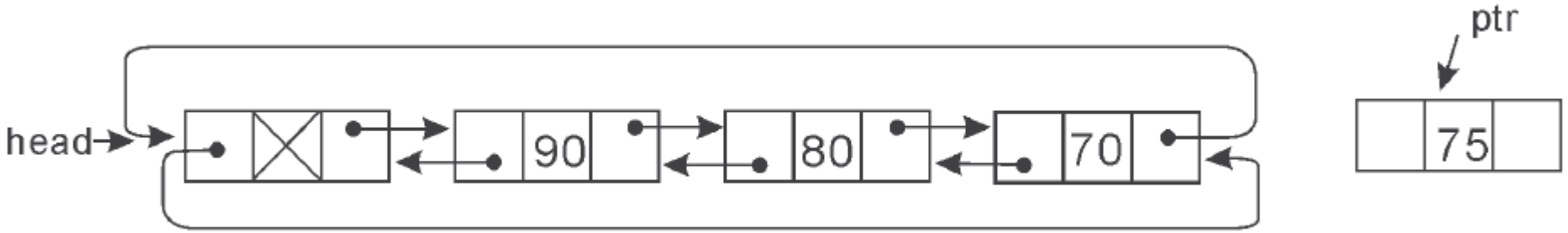
雙向鏈結串列-尾端加入 (4/4)

- (3) p->rlink=x; /* ④ */
- head->llink=x; /* ⑤ */
  - 此步驟乃將調整p 的rlink 和head 的llink

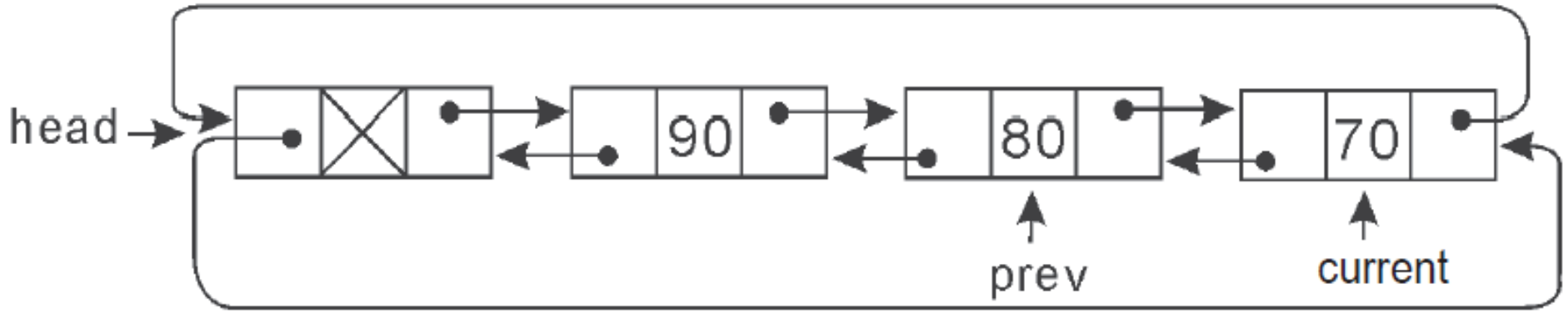


雙向鏈結串列-加入特定節點 (1/3)

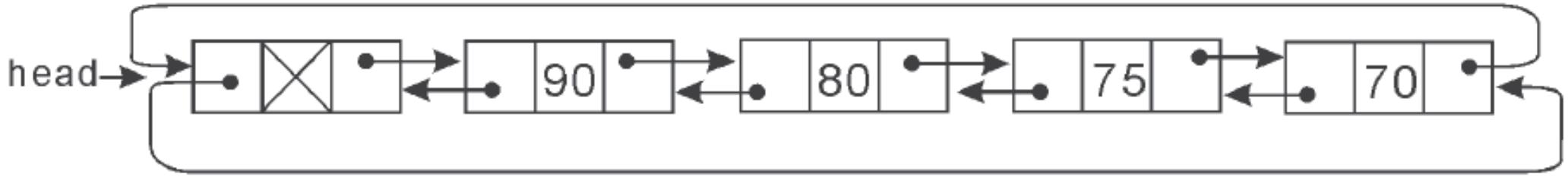
- 加入在某一特定節點的後面，理論上和單向鏈結串列相似。有一雙向鏈結串列如下（由大至小排列）



雙向鏈結串列-加入特定節點 (2/3)

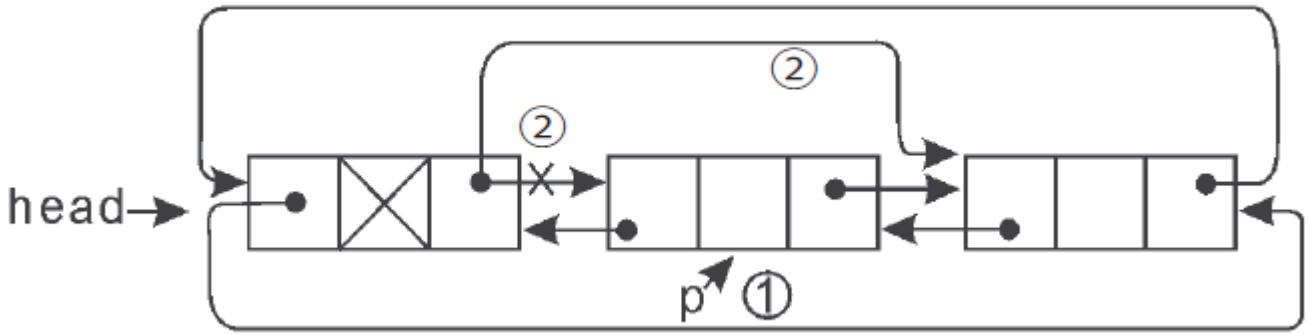


雙向鏈結串列-加入特定節點 (3/3)



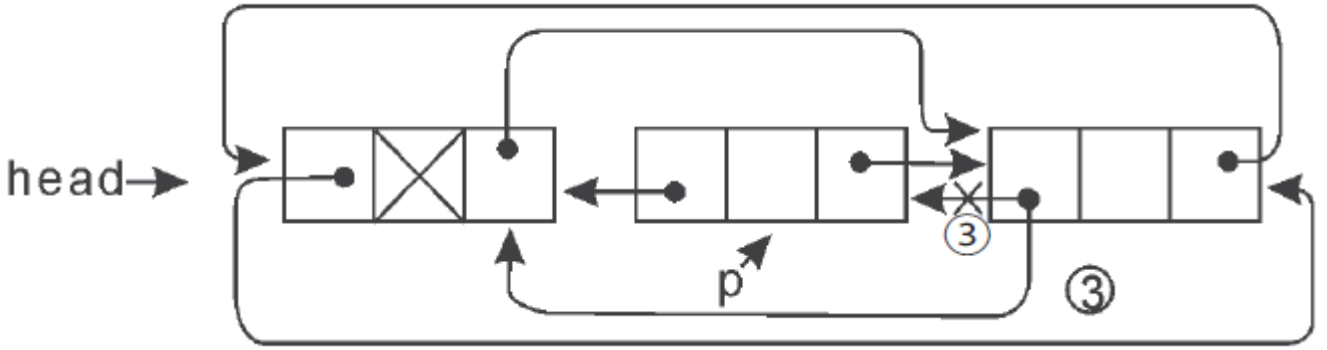
雙向鏈結串列-刪除前端 (1/2)

- 刪除雙向鏈結串列的前端，步驟如下：
  - (1) p=head->rlink; /* ① */
  - (2) head->rlink=p->rlink; /* ② */

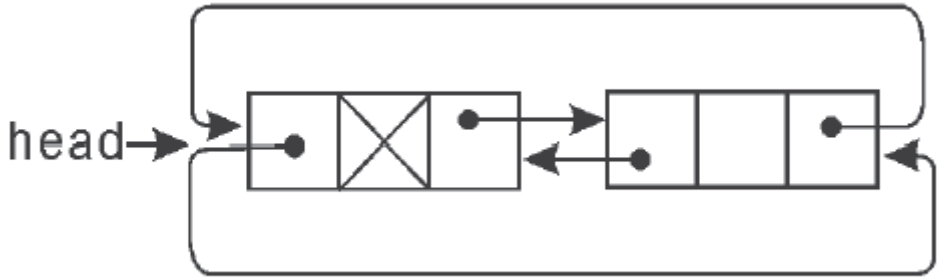


雙向鏈結串列-刪除前端 (2/2)

- (3) p->rlink->llink=p->llink; /* ③ */

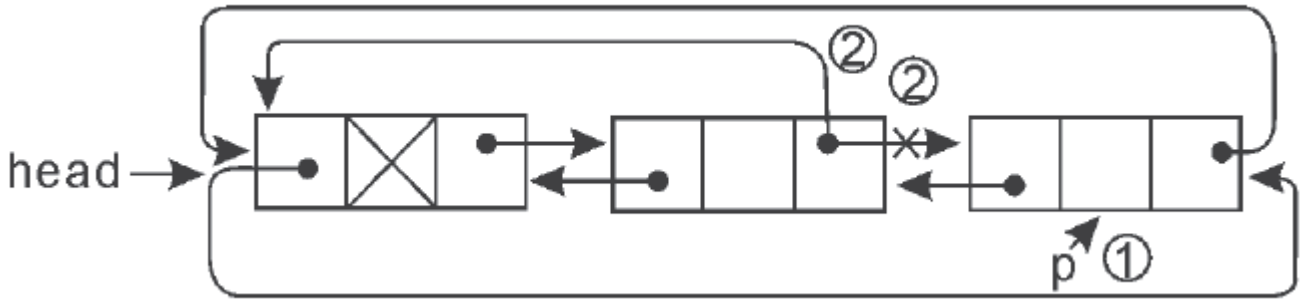


- (4) free(p);



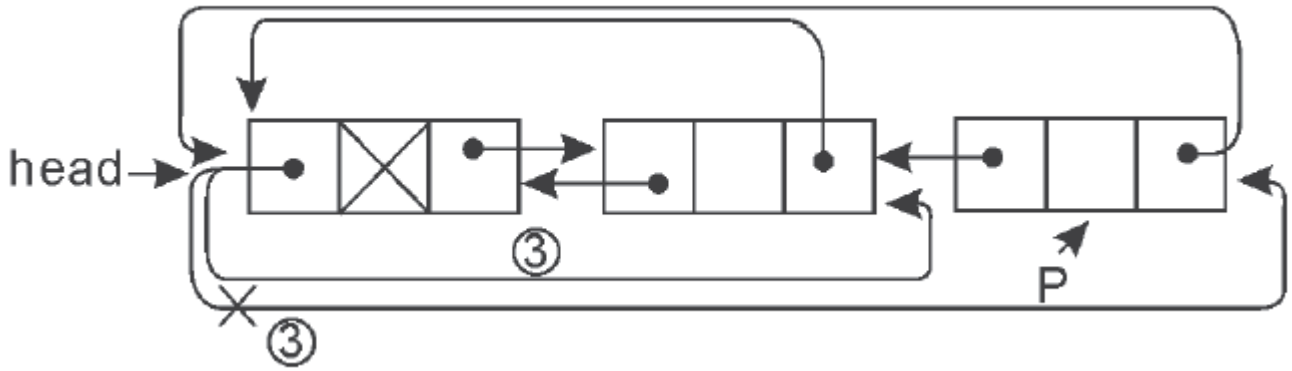
雙向鏈結串列-刪除尾端 (1/2)

- 追蹤到尾端的節點
  - (1) p=head->llink; /* ① */
  - (2) p->llink->rlink=p->rlink; /* ② */

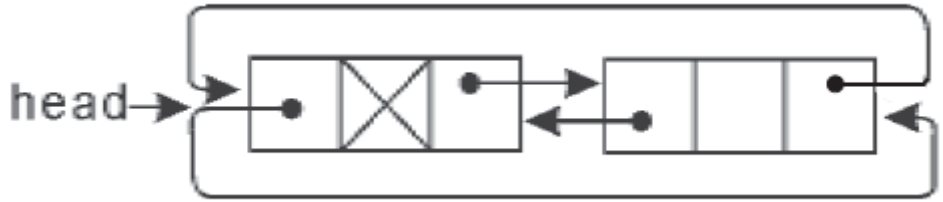


雙向鏈結串列-刪除尾端 (2/2)

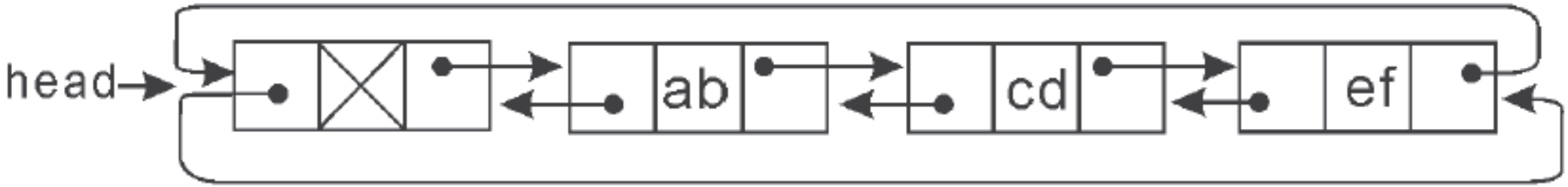
- (3) head->llink=p->llink; /* ③ */



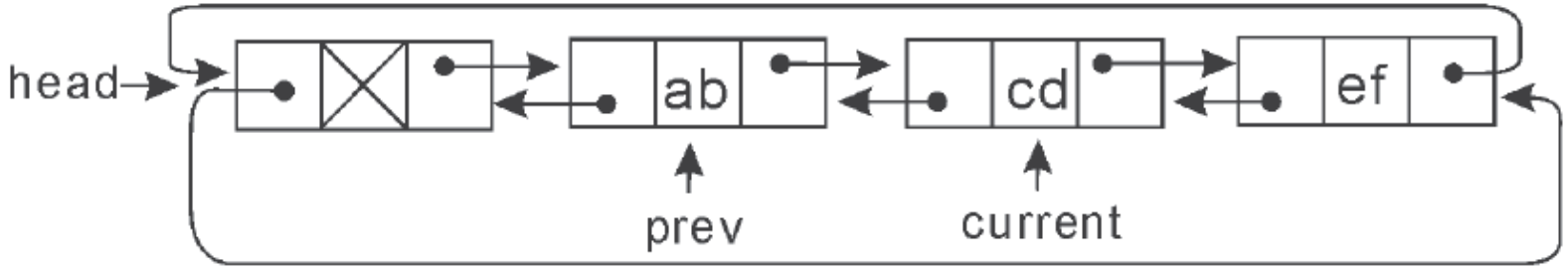
- (4) free(p)



雙向鏈結串列-刪除特定節點 (1/3)



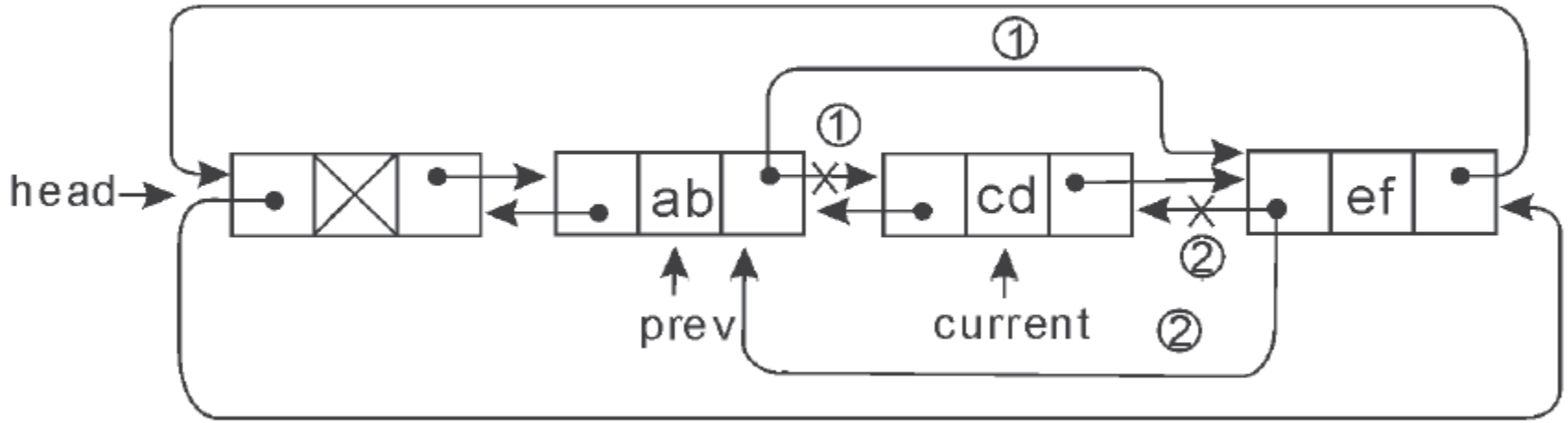
雙向鏈結串列-刪除特定節點 (2/3)




```
if (current != head) {
    prev->rlink=current->rlink;              /*   ①   */
    current->rlink->llink=prev;              /*   ②   */
}
else
    printf("the data not found")
```

雙向鏈結串列-刪除特定節點 (3/3)

上述片段程式如下圖所示：




```
free(current);
```

鏈結串列之應用

- 堆疊的加入和刪除操作都在同一端，因此，我們可以將它視為每次將節點加入與刪除的動作，是串列的前端或尾端。
- 佇列的加入和刪除是在不同端，因此我們可以想像加入的動作是在串列的尾端，而刪除的動作是在前端即可。

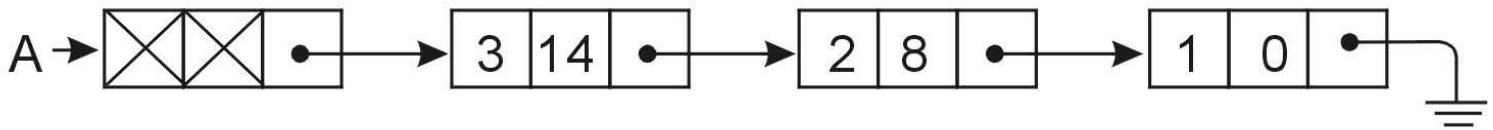
鏈結串列之應用-多項式相加 (1/7)

- 多項式相加可以利用鏈結串列來完成。多項式以鏈結串列來表示的話，其資料結構如下：

- COEF表示變數的係數，EXP表示變數的指數，而LINK為指到下一節點的指標。

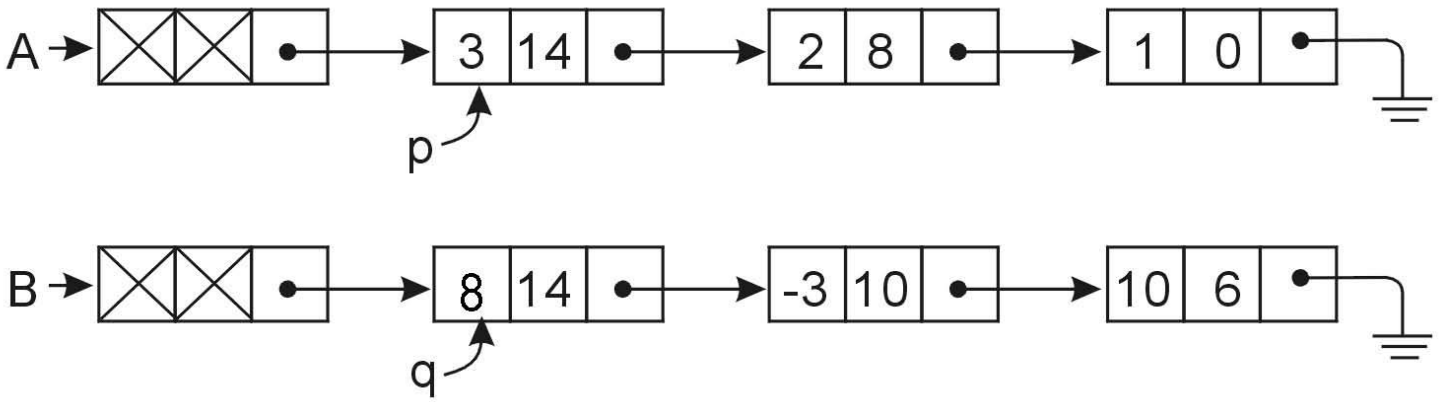
鏈結串列之應用-多項式相加 (2/7)

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



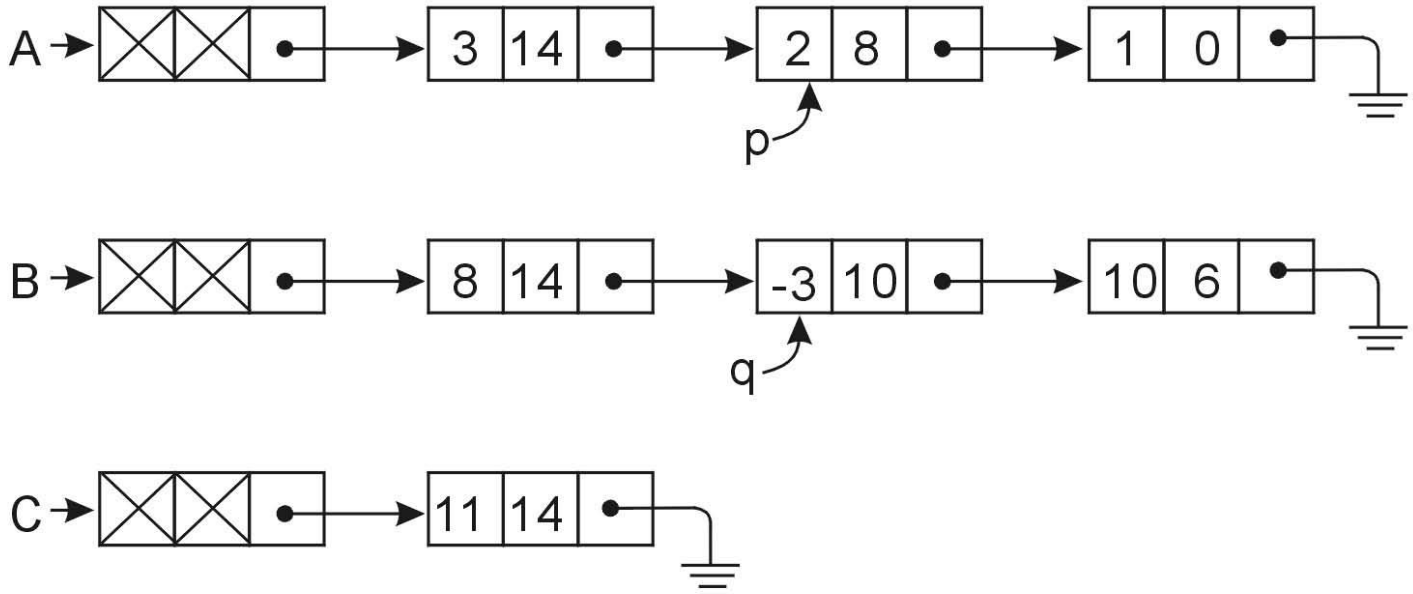
鏈結串列之應用-多項式相加 (3/7)

- 兩個多項式相加其原理如下圖所示：
- $A=3x^{14}+2x^{8}+1$; $B=8x^{14}-3x^{10}+10x^{6}$



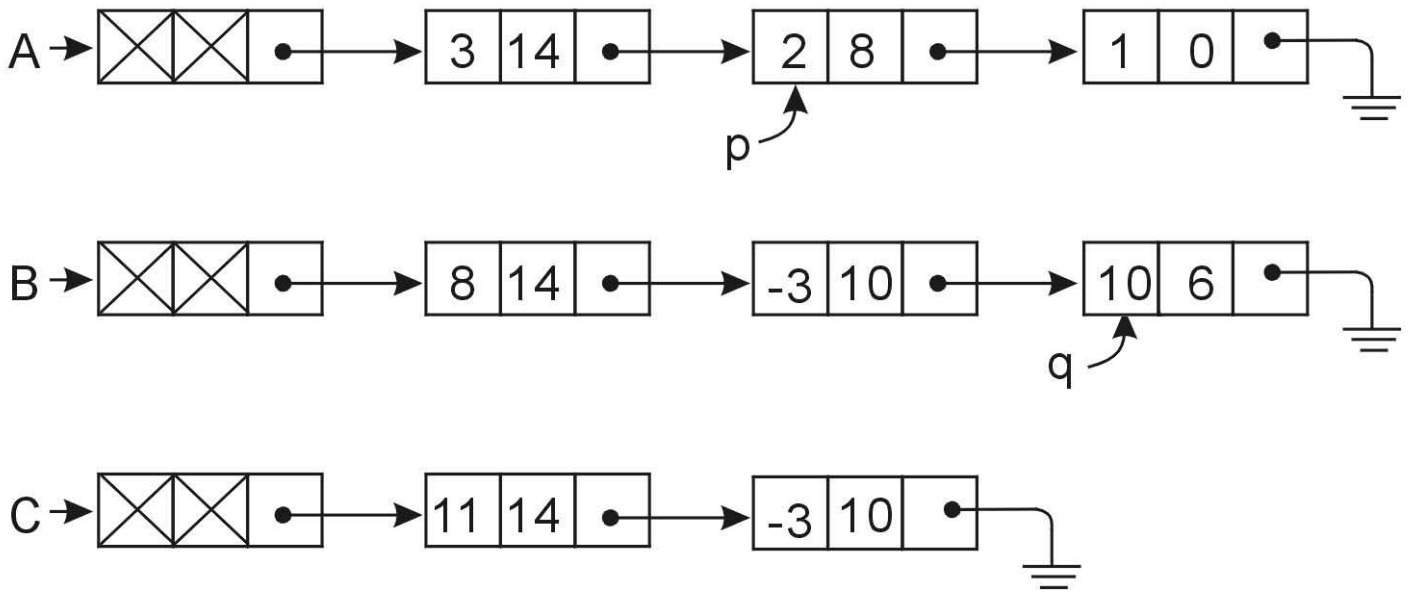
鏈結串列之應用-多項式相加 (4/7)

- 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p、q的指標指向下一個節點。



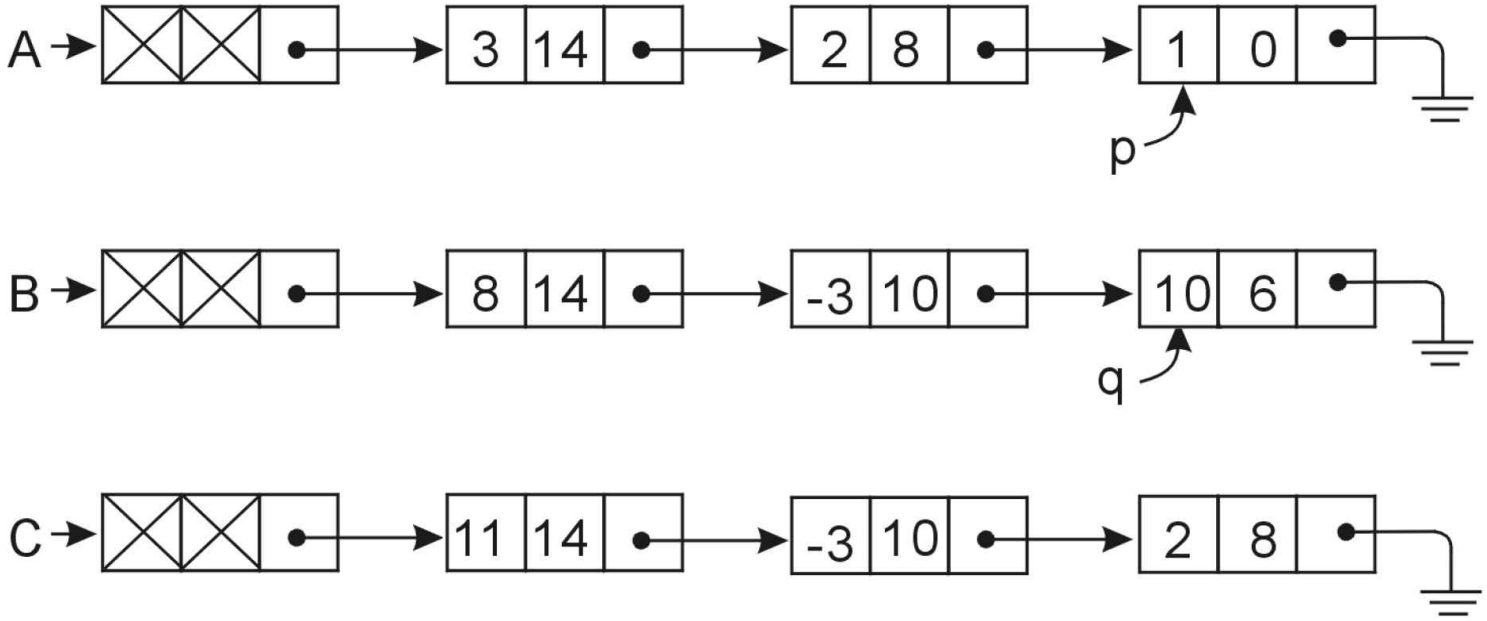
鏈結串列之應用-多項式相加 (5/7)

- 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q指標指向下一個節點。



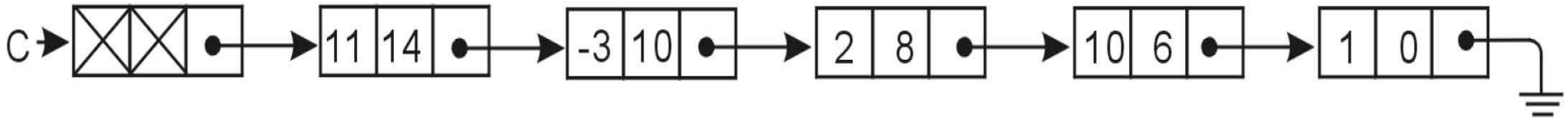
鏈結串列之應用-多項式相加 (6/7)

- 由於EXP(p)=8>EXP(q)=6，所以將A多項式的第二個節點加入C多項式，p指標指向下一個節點。



鏈結串列之應用-多項式相加 (7/7)

- 以此類推，最後C多項式為
  - $C=11x^{14}-3x^{10}+2x^{8}+10x^{6}+1$



Practice

- 設計KTV點歌系統
  - 提示功能選單
    - 1. 點歌 2.刪歌 3.插播 4.目前歌單
  - 使用者必須輸入功能選項與歌曲代碼。
  - 選擇點歌則在歌單的最後增加一首歌曲代碼。
  - 選擇刪歌則把輸入的歌曲刪除。
  - 選擇插播則把歌曲代碼排到第一首的位置。
  - 選擇目前歌單則把已經輸入的全部歌曲代碼顯示。

索引
0
1

陣列位置	0	1	2	3	4	5	6	7
歌曲編號	100	200	300	400	500	600	700	800
下一首歌	1	2	3	4	5	6	7	NULL

陣列位置	0	1	2	3	4	5	6	7
歌曲編號	100	200	300	400	500	600	700	800
下一首歌	1	2	3	5	5	6	7	NULL

陣列位置	0	1	2	3	4	5	6	7	8
歌曲編號	100	200	300	400		600	700	800	900
下一首歌	1	2	3	5		6	7	8	NULL

陣列位置	0	1	2	3	4	5	6	7
歌曲編號	100	200	300	400	900	600	700	800
下一首歌	1	2	3	5	NULL	6	7	4

索引
0
7

陣列位置	0	1	2	3	4	5	6	7
歌曲編號	100	200	300	400	900	800	900	50
下一首歌	1	2	3	4	5	6	NULL	0

索引
0
8

陣列位置	0	1	2	3	4	5	6	7	8
歌曲編號	100	200	300	400	900	800	900	50	20
下一首歌	1	2	3	4	5	6	NULL	0	7