第三章

鏈結串列

Linked List

本章內容

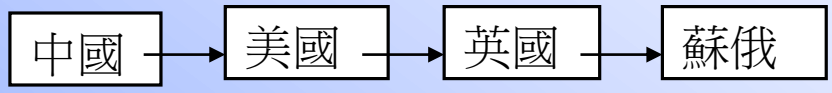
在射擊遊戲中，**子彈**朝不同的角度發射，有些一路前進，有些則擊中目標或碰到邊界而銷毀。
遊戲程式設計者**如何管理這些子彈**？

答案是使用**「鏈結串列」(linked list)**。遊戲程式設計者將子彈等隨時會產生也隨時會消滅的**動態物件放在鏈結串列中**

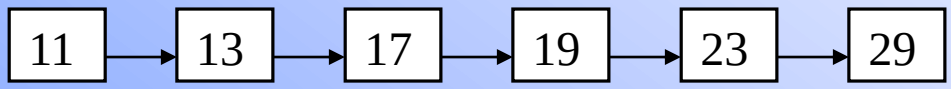
3-1 串列(list)

串列 是許多項目有順序的排列 (集合 (set) 裡的元素則沒有前後之分)

二次大戰同盟國國名串列



10到30之間的質數由小到大排列



常用於串列上的幾個運算

1. 插入新節點 (Insert) － 在原有的串列中加入一個新的元素。
2. 刪除某節點 (Delete) － 將某一個元素從原本的串列中移除。
3. 找尋某節點 (Find) － 依照需求將所指定的（或第幾個）元素值讀出來

串列的實作方式

「循序配置串列」(sequential allocation list)

「鏈結配置串列」(linked allocation list)

A = { 中國, 美國, 英國, 蘇俄 }

1. 「集合」可以執行"插入新元素到第一個位置"的運算嗎？為什麼？

2. 「集合」可以執行串列三個運算的哪些項目？

3-2 循序配置串列：用陣列直接儲存串列

利用陣列循序儲存串列的方式，具有以下的優缺點：

優點：只要標明要查詢第幾個元素，就能很快地根據元素註標讀寫該元素，因為陣列的隨機存取 (random access) 效率相當高，時間複雜度為 O(1)。

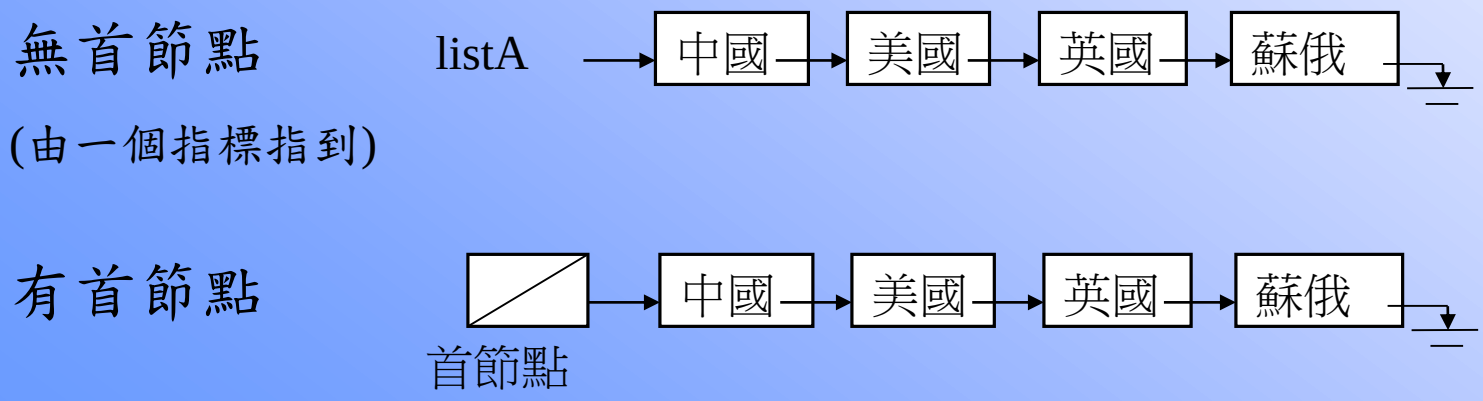
缺點：插入元素 (insert) 和刪除元素 (delete) 兩個動作所需時間的複雜度為 O(*n*)。這是因為陣列所有元素在記憶體中的實體位置 (physical location) 是連續的，因此不論是執行元素的插入或刪除，都必須進行資料的搬移，以維持正確的順序(參考2.2節)。

3-3 鏈結配置列：串列加上鏈結

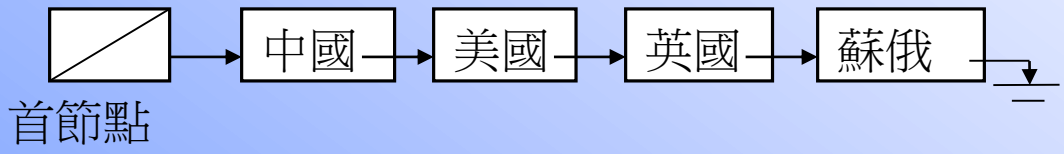
「鏈結串列」(linked list) 可以用來解決陣列循序配置的缺點：

1. 鏈結串列的元素之間不必實體連續（不必依元素順序佔用記憶體中的連續位址），只要有邏輯上 (logical) 的順序存在即可

2. 「鏈結」(link) 就是用來維持這順序的工具，它可以告訴我們「下一個元素放在哪裡」。



鏈結串列中，我們可以直接存取第 3 個節點資料而不經過前 2 個節點嗎？



3-4 用結構體陣列實作鏈結串列

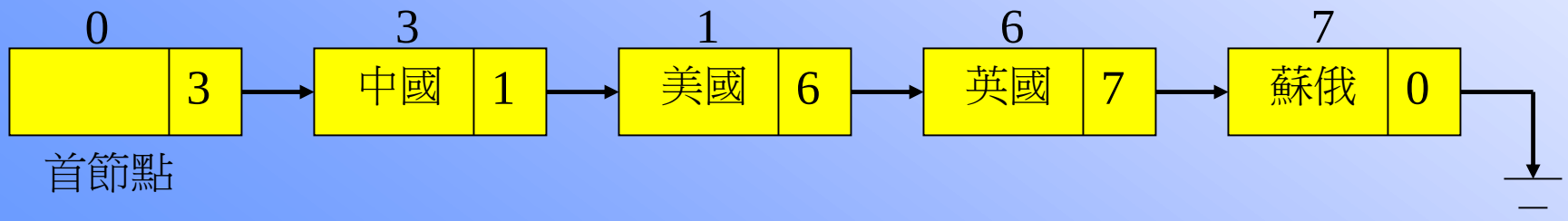
	data	next
Table[0]		3
[1]	美　國	6
[2]		-1
[3]	中　國	1
[4]		-1
[5]		-1
[6]	英　國	7
[7]	蘇　俄	0
[8]		-1

從這裡開始

data 欄位存放資料本身

next 欄位存放下一個資料的位置

節散散佈在陣列中，實體位置沒有連續，但根據鏈結可以得到以下次序



資料會這麼"散亂"，是假設經過了一系列的插入及刪除等動態運算，用來說明"實體上雖然散亂、邏輯上仍然有序"的概念。

插入新節點

要在 ‘英國’ 節點之後插入一個新節點 ‘法國’

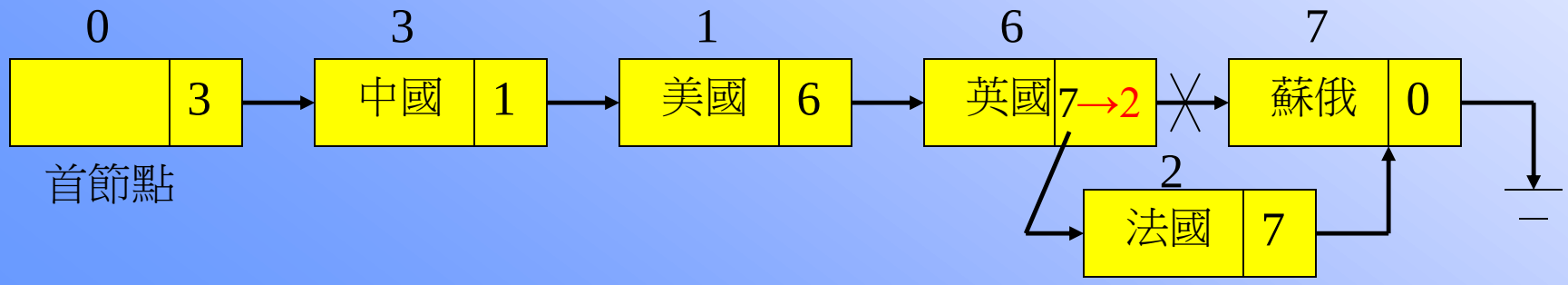
	data	next
Table[0]		3
[1]	美　國	6
[2]	① 法　國	③ -1→7
[3]	中　國	1
[4]		-1
[5]		-1
[6]	② 英　國	④ 7 →2
[7]	蘇　俄	0
[8]		-1

① 找到一個空節點(next值為-1)，填入資料 ‘法國’

② 找到英國，或者條件已知

③ 將英國的 next (7) 複製到法國的next

④ 將法國的位置 (2) 填入英國的next



刪除節點

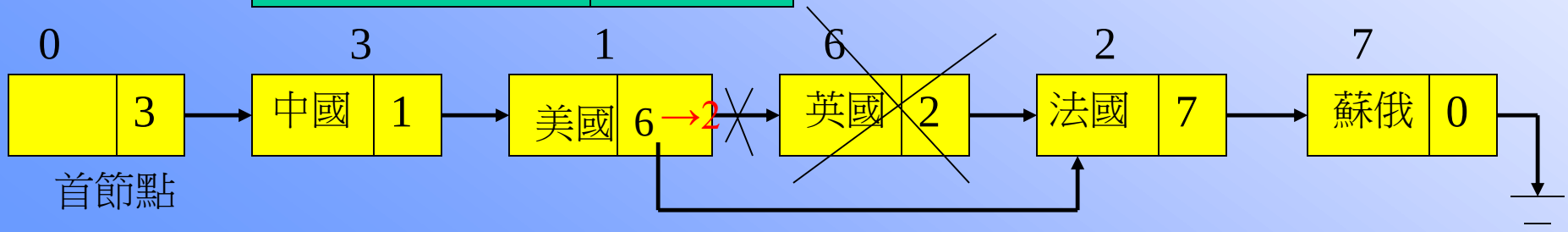
要刪除 ‘英國’ 節點

	data	next
Table[0]		3
[1]	美 國	6 →2
[2]	法 國	7
[3]	中 國	1
[4]		-1
[5]		-1
[6]	~~英 國~~	2 → -1
[7]	蘇 俄	0
[8]		-1

① 沿著串列找到英國 (6)，以及英國的前一個節點 ‘美國’ (1)

② 將英國的 next (2) 複製到美國的 next

③ 將英國節點還原為空節點

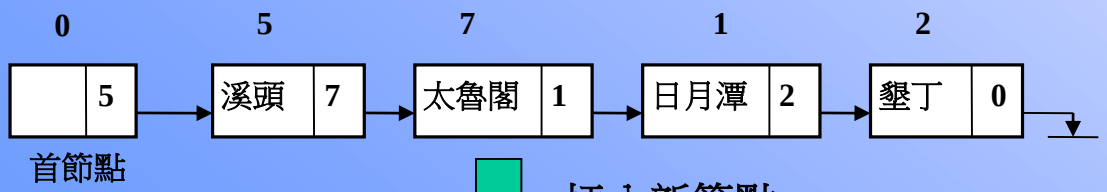


要將「阿里山」加在「太魯閣」之後

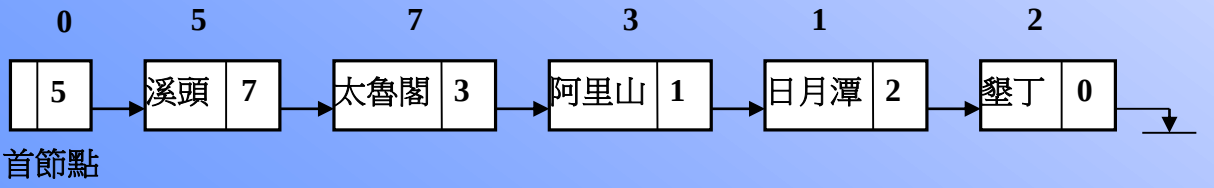
	data	next
table[0]		5
table[1]	日 月 潭	2
table[2]	墾　丁	0
table[3]		-1
table[4]		-1
table[5]	溪　頭	7
table[6]		-1
table[7]	太 魯 閣	1
table[8]		-1

	data	next
table[0]		5
table[1]	日 月 潭	2
table[2]	墾　丁	0
table[3]	阿 里 山	-1→1
table[4]		-1
table[5]	溪　頭	7
table[6]		-1
table[7]	太 魯 閣	1→3
table[8]		-1

鏈結串列圖示



插入新節點

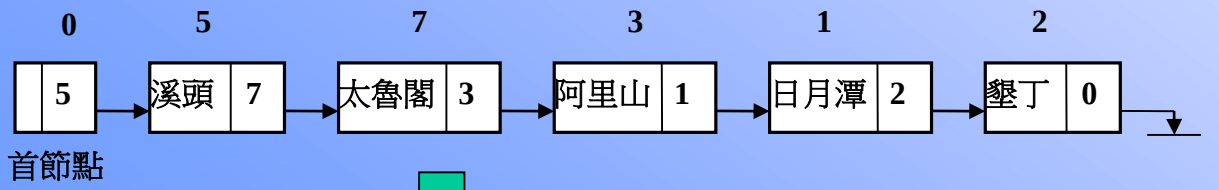


刪除「太魯閣」

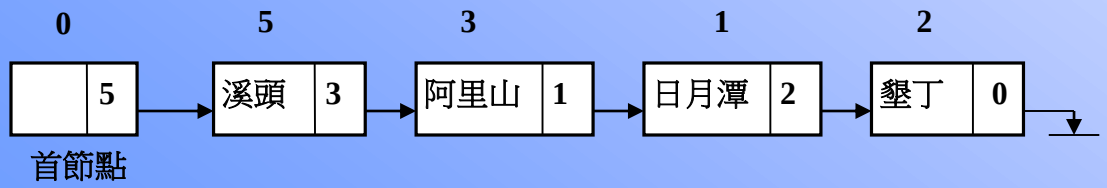
	data	next
table[0]		5
table[1]	日 月 潭	2
table[2]	墾　丁	0
table[3]	阿 里 山	-1→1
table[4]		-1
table[5]	溪　頭	7
table[6]		-1
table[7]	太 魯 閣	1→3
table[8]		-1

	data	next
table[0]		5
table[1]	日 月 潭	2
table[2]	墾　丁	0
table[3]	阿 里 山	1
table[4]		-1
table[5]	溪　頭	7→3
table[6]		-1
table[7]	~~太 魯 閣~~	3→-1
table[8]		-1

鏈結串列圖示



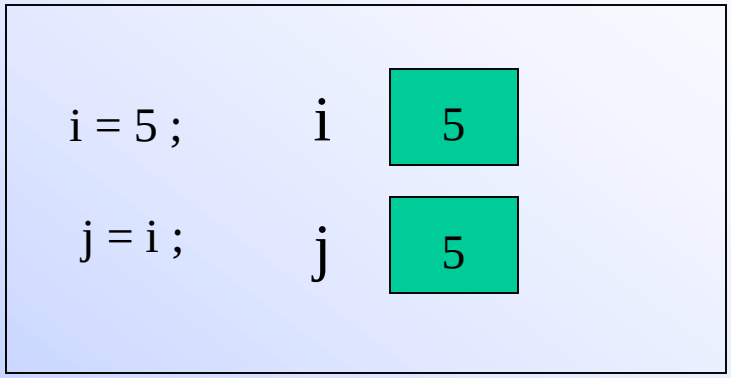
刪除節點



3-5 指標與結構體

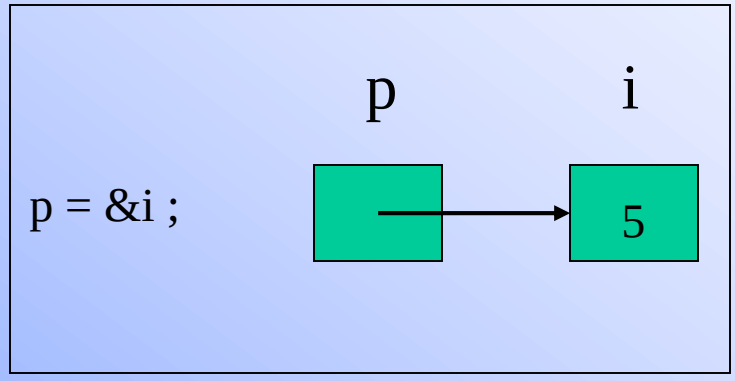
◎變數是存放資料的地方

```
int   i, j;
```



◎指標是存放位址的地方

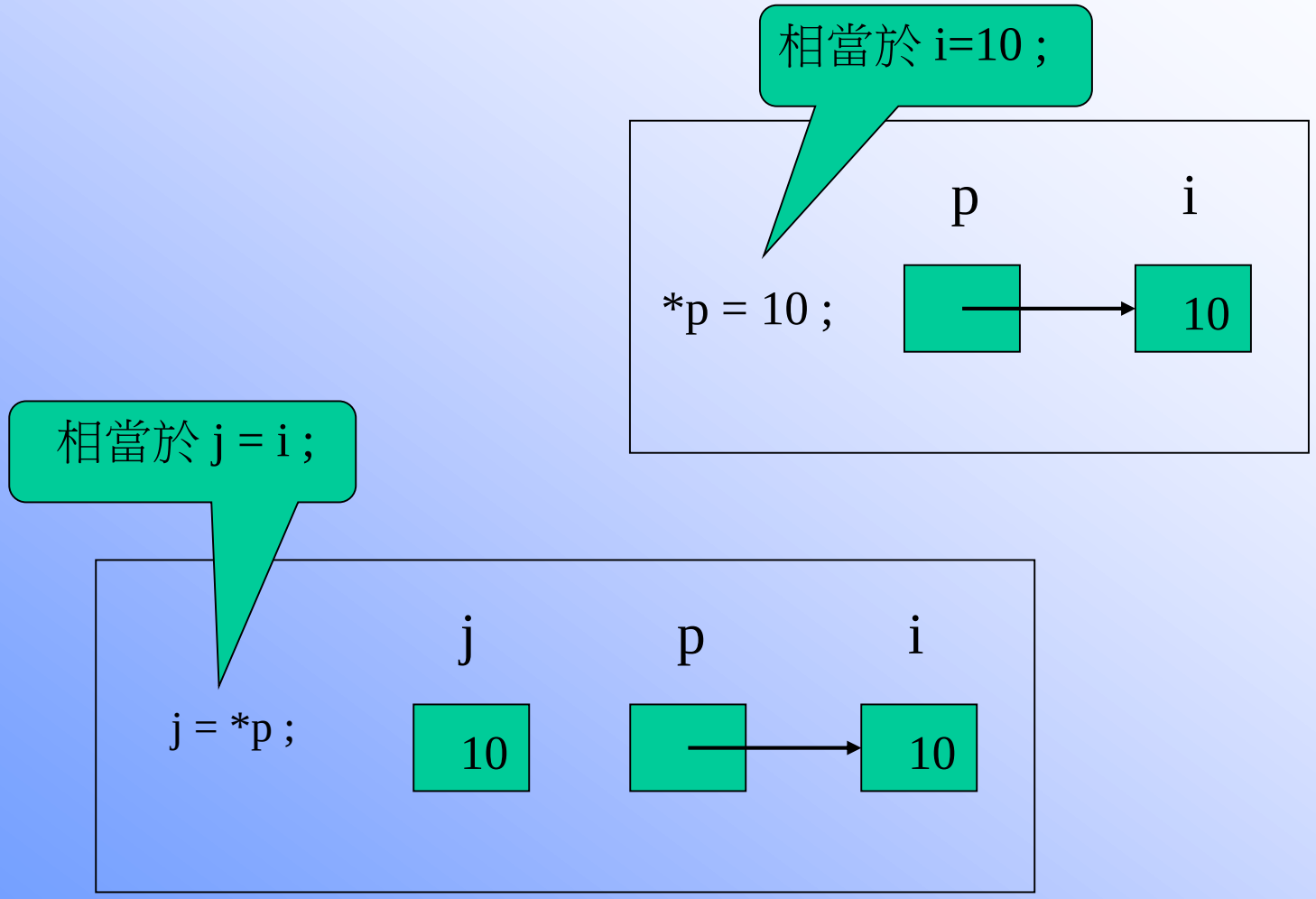
```
int   *p;
```



同義詞

p 指向 i ⟷ p 裡面存放 i 的位址

◎ *p 是 “p 所指變數” 的同義詞

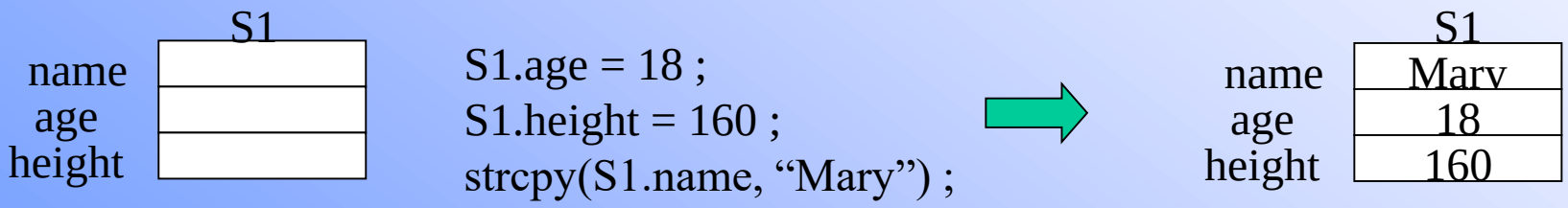


結構體 (structure)

➢結構體是將不同型別的元素集合起來，形成一種新的資料型別

```
1.    struct  student {
2.              char      name[8]; //姓名欄位
3.              int       age;     //年齡欄位
4.              int       height;  //身高欄位
5.    }         ;
```

此時就會產生一種叫做*student*的新結構體，這種結構體含有三個欄位，其中姓名欄位是一個字元陣列、年齡欄位是整數、身高欄位也是整數。



➢ 宣告 typedef int INTEGER ;

就產生一個「新」的型別叫INTEGER，可以根據這個型別來宣告變數：

INTEGER i, j ;

它的效果就好像宣告 int i, j; 一樣。

3-6 動態配置節點實作鏈結串列

動態配置節點方式實作鏈結串列，常會執行的串列運算按照順序為：

1. 宣告節點結構
2. 動態配置節點
3. 指標在串列中移動
4. 插入新節點
5. 刪除節點
6. 建立串列

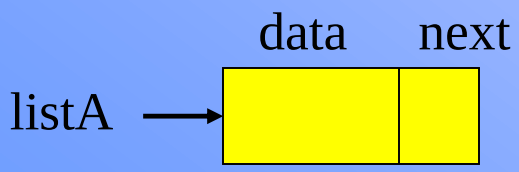
1. 宣告節點結構

```
typedef    struct      listnode
        {      int data;                    /*資料欄位*/
               struct listnode *next;   /*鏈結欄位*/
        }NODE;
NODE *listA;
```

注意：鏈結欄位
改為指標

2. 動態配置節點

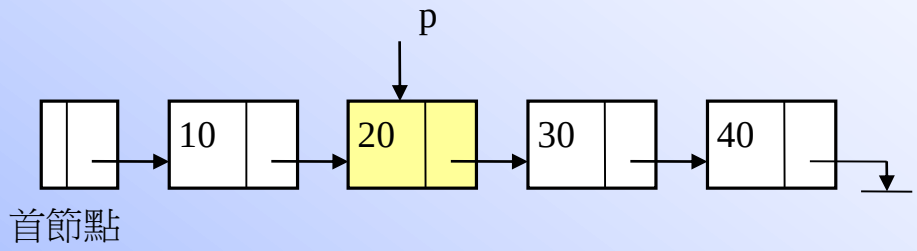
```
listA = (NODE *) malloc( sizeof(NODE)) ;
在C++語言中則可用 new 運算子
listA = new NODE ;
```



20 填入資料欄位，NULL 填入鏈結欄位：

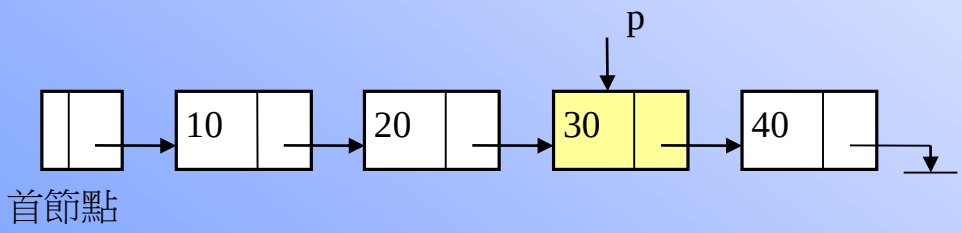
```
listA->data = 20 ;
listA->next = NULL ;
```

3. 指標在串列中移動



執行敘述

p = p->next ;



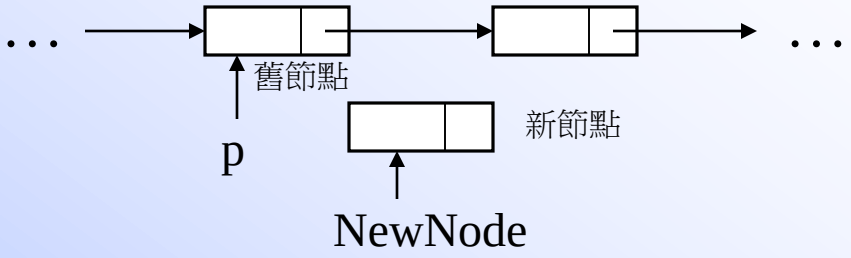
指標 p 指向下一個節點

走訪整個鏈結串列

```
void  ListTraverse(ListNode *head)
{       ListNode *p = head;
        p = p->next ;
        while ( p != NULL )
        {           printf( “\n%d”, p->data ); //印出節點資料
                    p = p->next ; //指標p沿著鏈結而指向下一個節點
        }
}
```

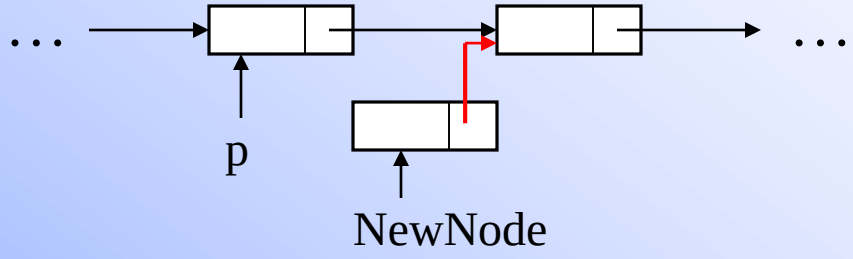
插入新節點

要將指標 NewNode 所指到的新節點，插入指標 p 所指舊節點之後



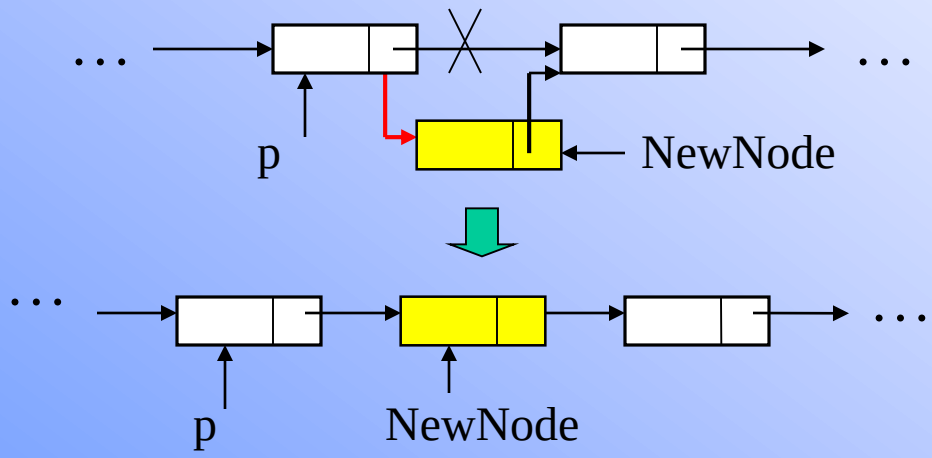
*第一步：掛上**新節點**的鏈結*

```
NewNode->next = p->next ;
```



*第二步：改變**舊節點**的鏈結*

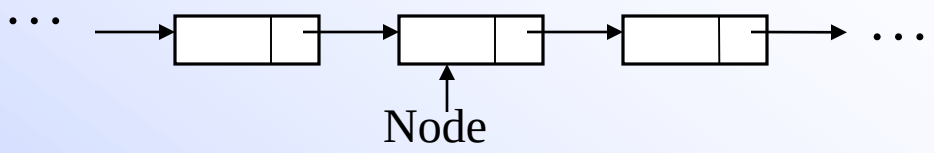
```
p->next = NewNode ;
```



```
1.      int  InsertAfter(ListNode *p, int value)
2.      {    ListNode  *NewNode;
3.           NewNode = (ListNode*) malloc(sizeof(ListNode));  //動態配置節點
             // C++:  NewNode = new ListNode;
4.           if (NewNode == NULL)
5.                return FALSE;              //配置失敗
6.           NewNode->data = value;          //將新節點的值填入
7.           NewNode->next = p->next;        //掛上新節點的鏈結
8.           p->next = NewNode;              //改變原有節點的鏈結
9.           return TRUE;                    //傳回成功訊息
10.     }
```

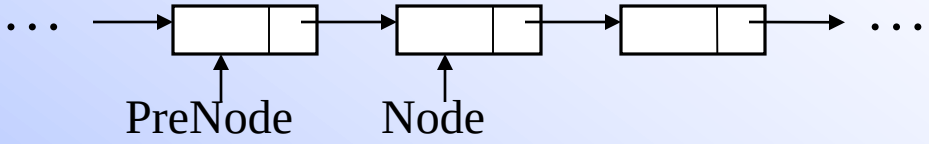
刪除節點

要刪除指標 Node 所指到的節點



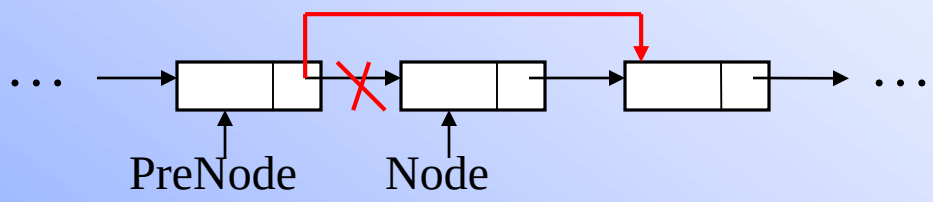
第一步：找到前一個節點

```
PreNode = GetPreNode(head,Node) ;
```



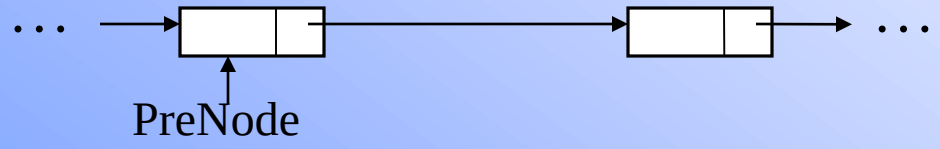
第二步：前一節點的鏈結，
使它越過舊節點

```
PreNode->next = Node->next ;
```



第三步：將舊節點動態歸還

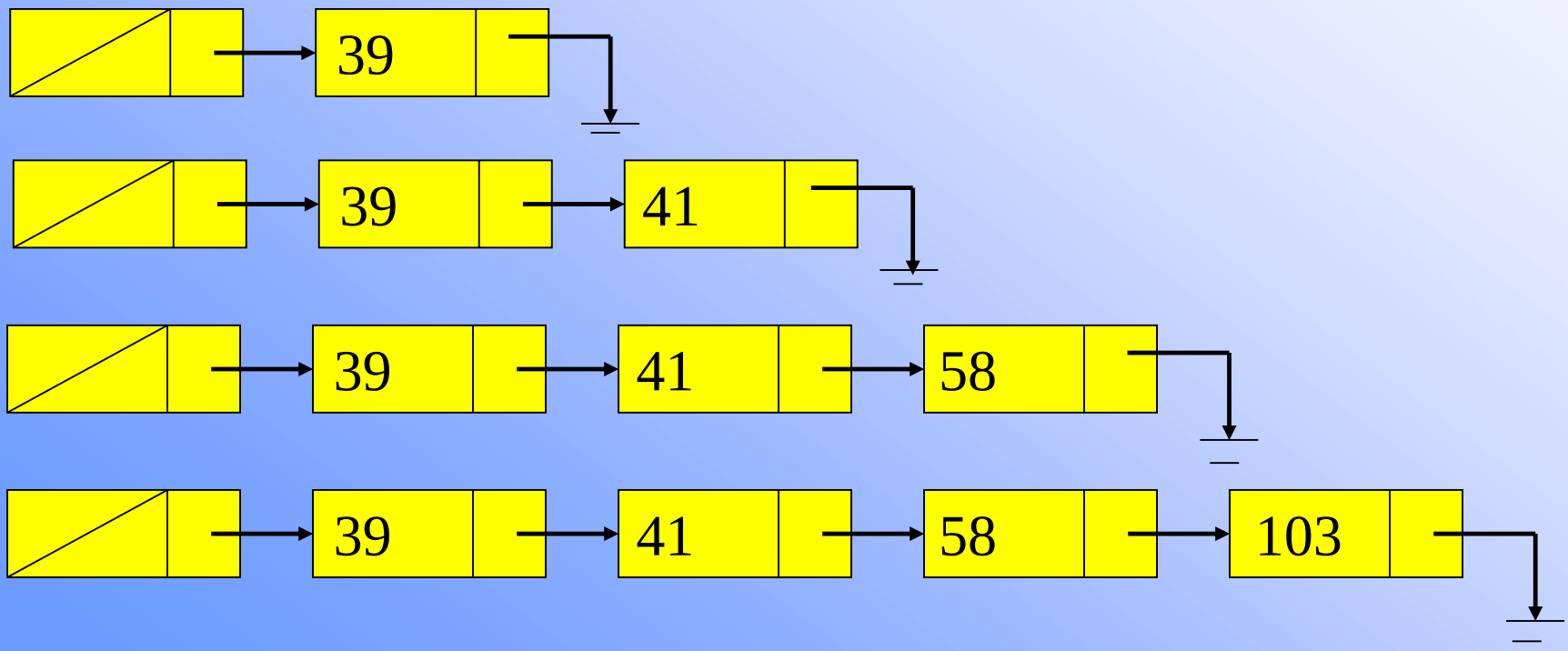
```
free( Node ) ;
```



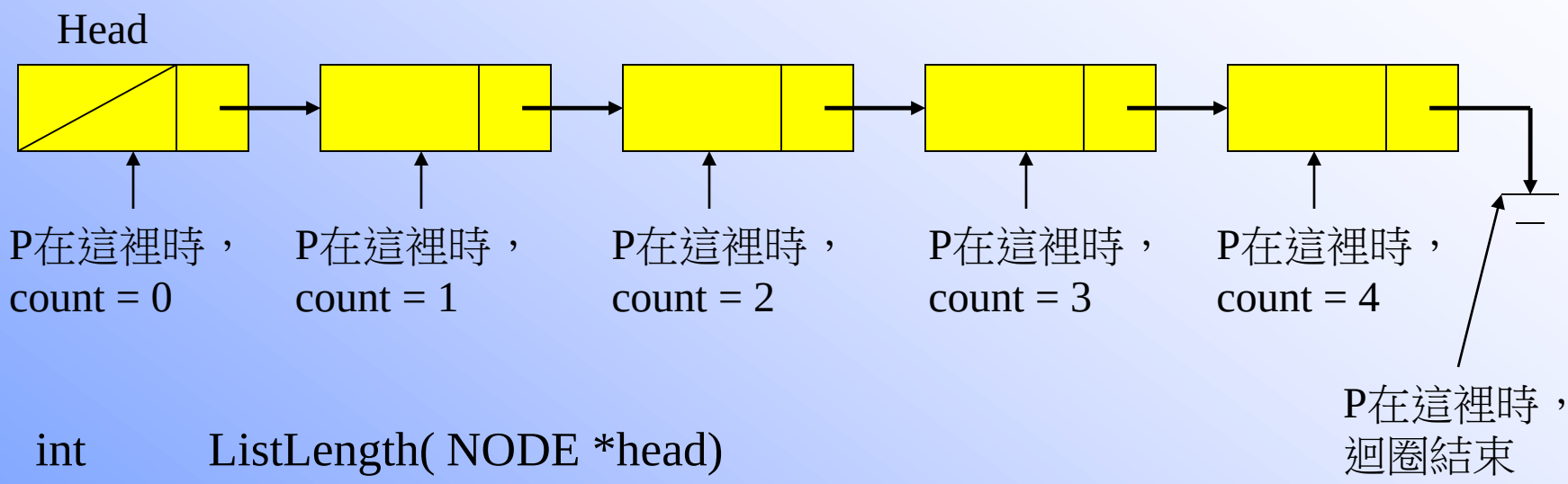
```
1.int  DeleteNode(ListNode *head, ListNode *OldNode)
2.{     ListNode *PreNode;
3.      if (head == OldNode)   //不可刪除首節點
4.            return FALSE;      //非法刪除，傳回失敗訊息
5.      PreNnode = GetPreNode(head, OldNode);
                                   //找到前一節點
6.      if ( PreNode == NULL)//OldNode節點不在串列中
7.            return FALSE;      //無法刪除，傳回失敗訊息
8.      PreNode->next = OldNode->next;
                                   //越過刪除原有節點
9.      free(OldNode);             //歸還被刪除節點
                                  // C++:  delete OldNode;
10      return TRUE;
11.}
```

6. 建立鏈結串列

- 建立鏈結串列的方法是重複地將新節點加入串列。
- 新節點加入的位置
  1. 可以是串列的前面
  2. 也可以是在串列的尾端
  3. 也可以是在串列的特定地方（例如按照 data 欄位大小排列）。

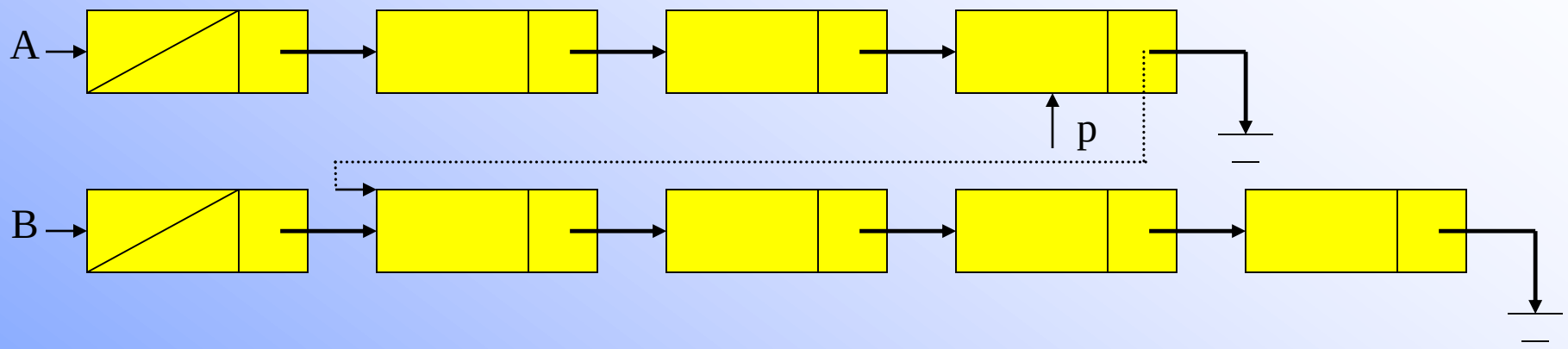


◎計算串列長度（節點數目）-- 線性鏈結串列




```
int     ListLength( NODE *head)
{       int     counter = 0 ;
        NODE *p = head ;
        while ( (p = p->next) != NULL)
                counter ++ ;
        return counter ;
}
```

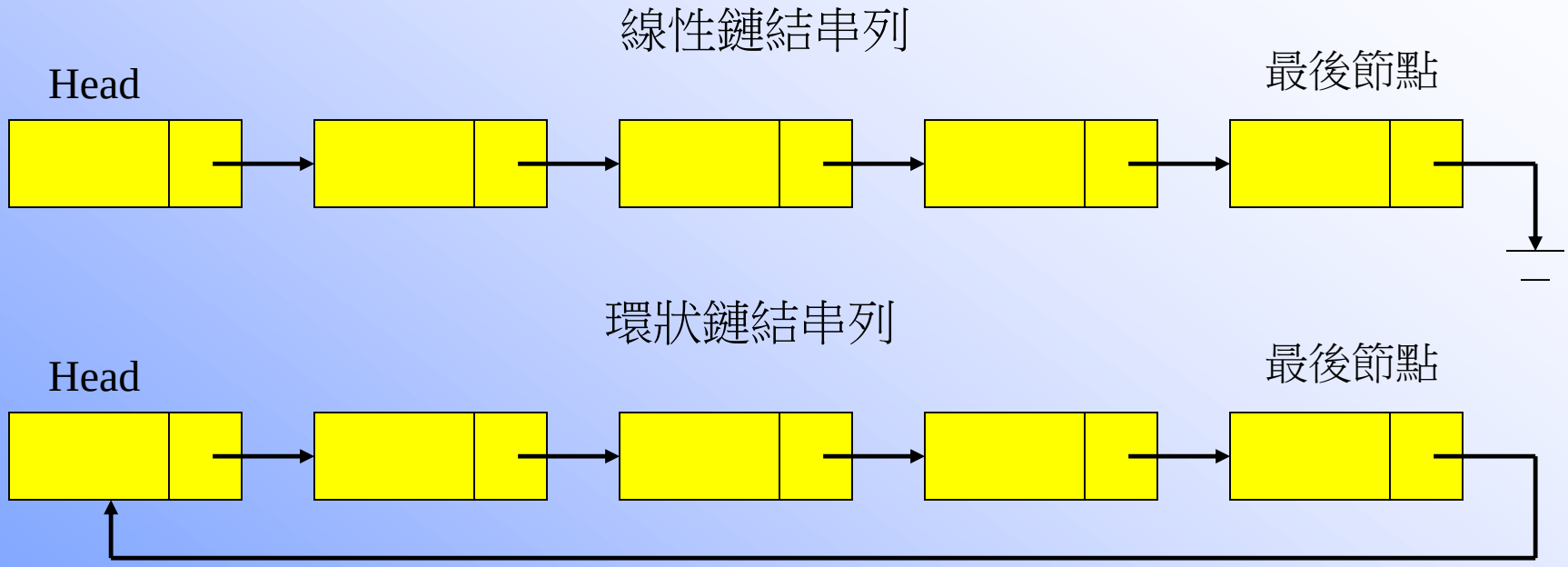
◎串接兩鏈結串列 -- 線性鏈結串列




```
void      ListConcate( NODE *listA, NODE *listB)
{         NODE *p = listA ;
          while ( p->next != NULL) // p一路往右直到最後一個節點
                    p = p->next ;
          p->next = listB->next ;      // 改變其鏈結
}
```

3.7 更多類型的鏈結串列

環狀鏈結串列



環狀鏈結串列的特點

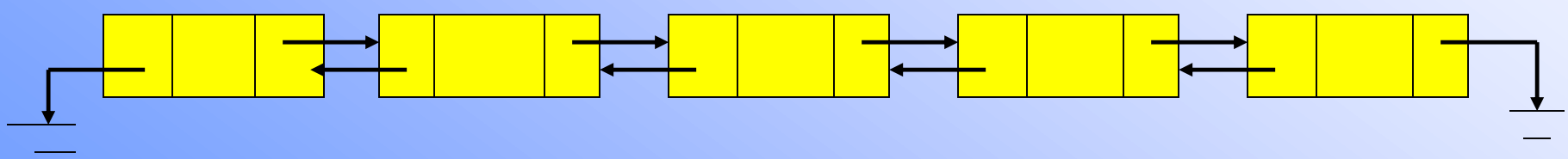
1. 最後節點的鏈結不接地，而是指向首節點

2. 從任何一個節點開始，都可以走訪所有節點。只要繞回原起點就可以停止

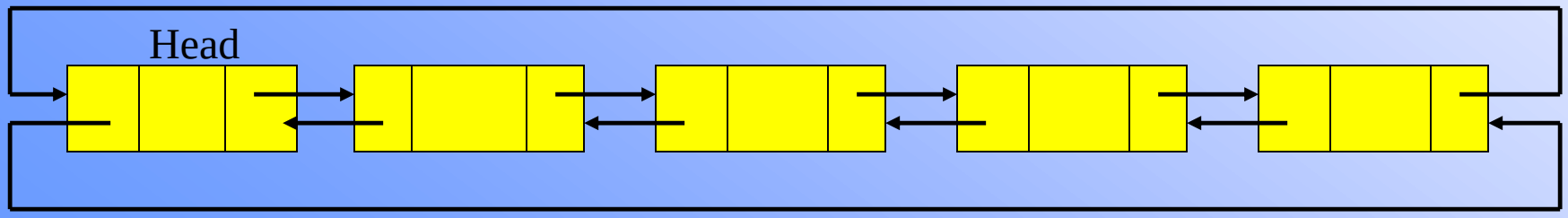
雙向鏈結串列

```
typedef   struct   dlist_node
          { struct dlist_node *left;
            int        data;
            struct dlist_node *right;
          }Dnode;
```

線性雙向鏈結串列

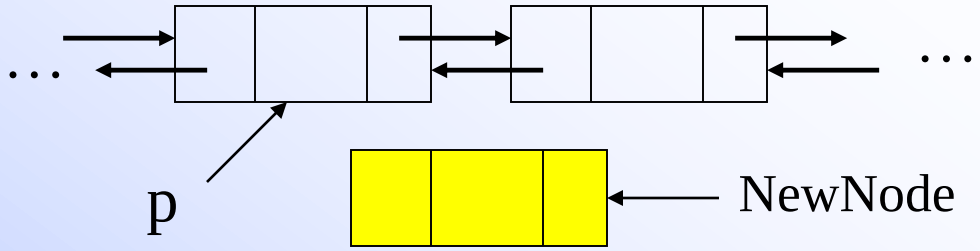


環狀雙向鏈結串列



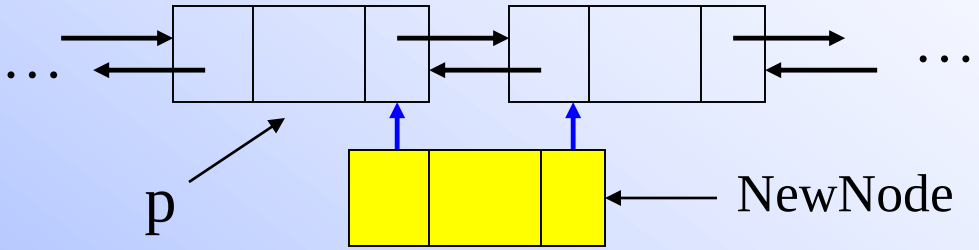
插入新節點

要將指標 NewNode 所指到的新節點，插入指標 p 所指舊節點之右



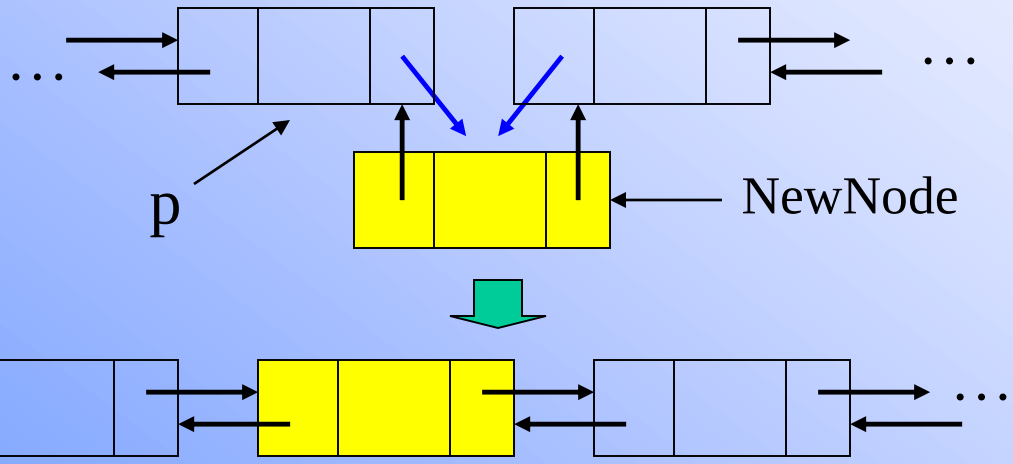
*第一步：掛上**新節點**的兩個鏈結*

```
NewNode->left = p;
NewNode->right = p->right ;
```



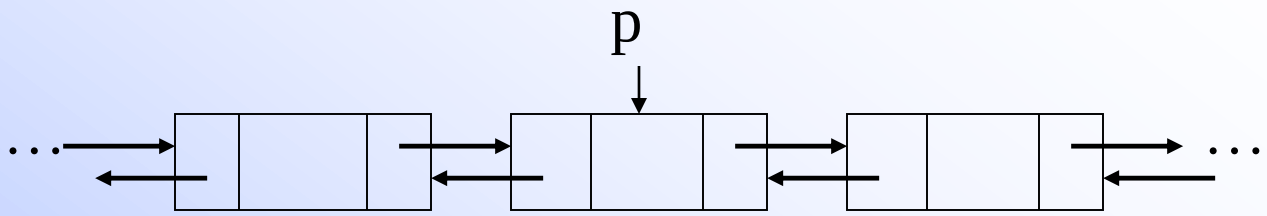
*第二步：改變**舊節點**的兩個鏈結*

```
p->right->left = NewNode;
p->right = NewNode ;
```



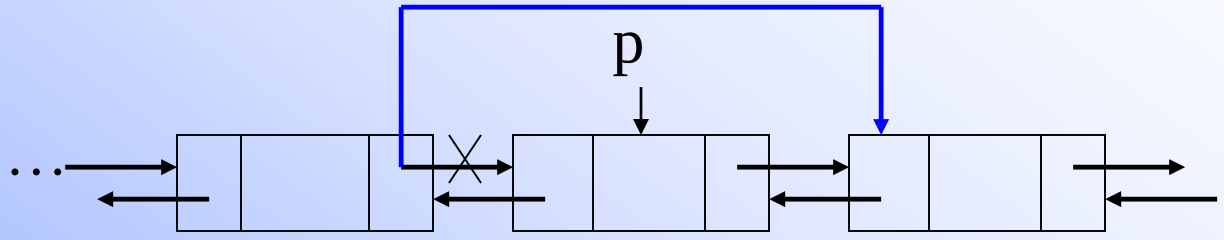
刪除節點

要刪除指標 p 所指到的節點



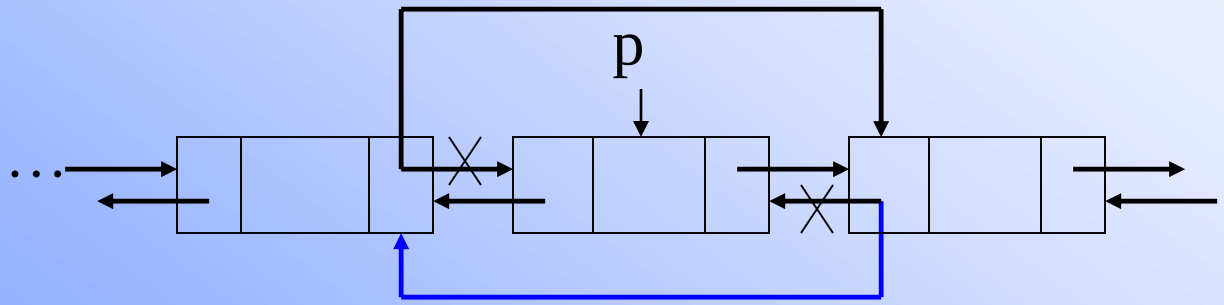
第一步：改變左邊節點的右指標使它越過舊節點

```
p->left->right = p->right ;
```



第二步：改變右邊節點的左指標使它越過舊節點

```
p->right ->left = p->left ;
```



第三步：將舊節點歸還系統

```
free( p ) ;
```

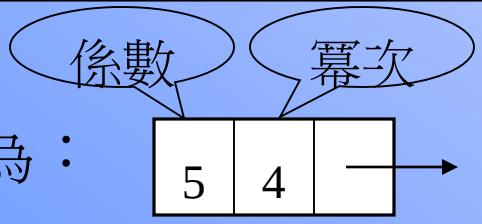
3-8 鏈結串列的應用

◎ 多項式的表示與運算

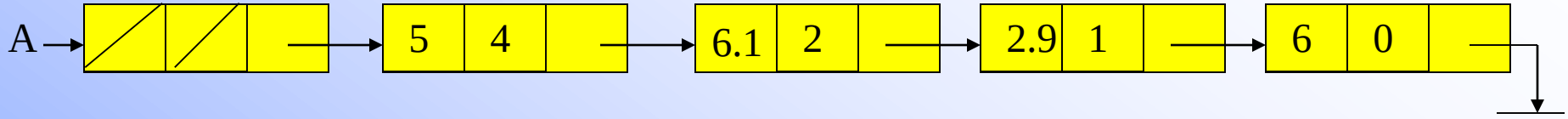
使用鏈結串列表示多項式，每個非零項使用一個節點，每個節點包含兩個資料欄位：係數欄位和冪次欄位，以及一個鏈結欄位。節點結構可以宣告為：

```
typedef   struct  plistnode
{         float  coef ;          /* 係數*/
          int    expo ;          /* 冪次*/
          struct  plistnode *next ;  /*鏈結指標*/
} Pnode;
```

$5x^4$ 可表示為：



$A(x) = 5x^4 + 6.1x^2 + 2.9x + 6$

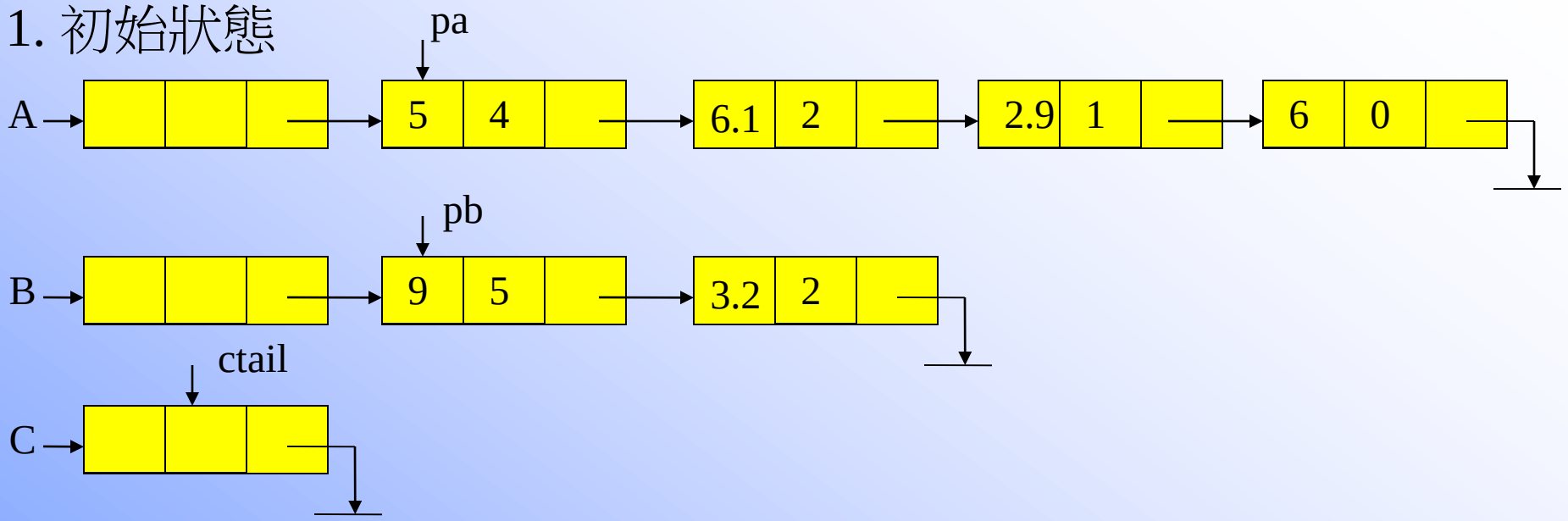


$B(x) = 9x^5 + 3.2x^2$

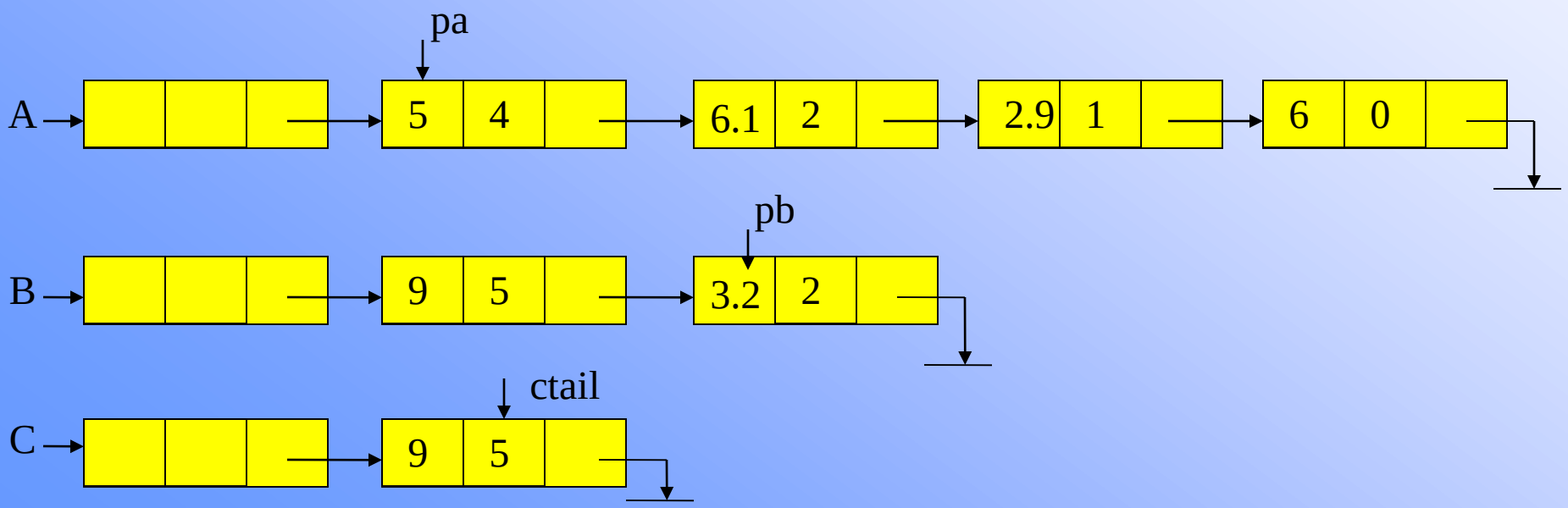
※ 多項式相加

1. 兩多項式分別從高冪次項往右掃瞄。
2. 比較兩項冪次大小，冪次大者，複製此項至C(x)，如果冪次相同，則係數相加後，同樣複製此項至C(x)。
3. 凡是已被複製的項，多項式就往右移一項。重複步驟1,2，直到兩個多項式都用完為止。

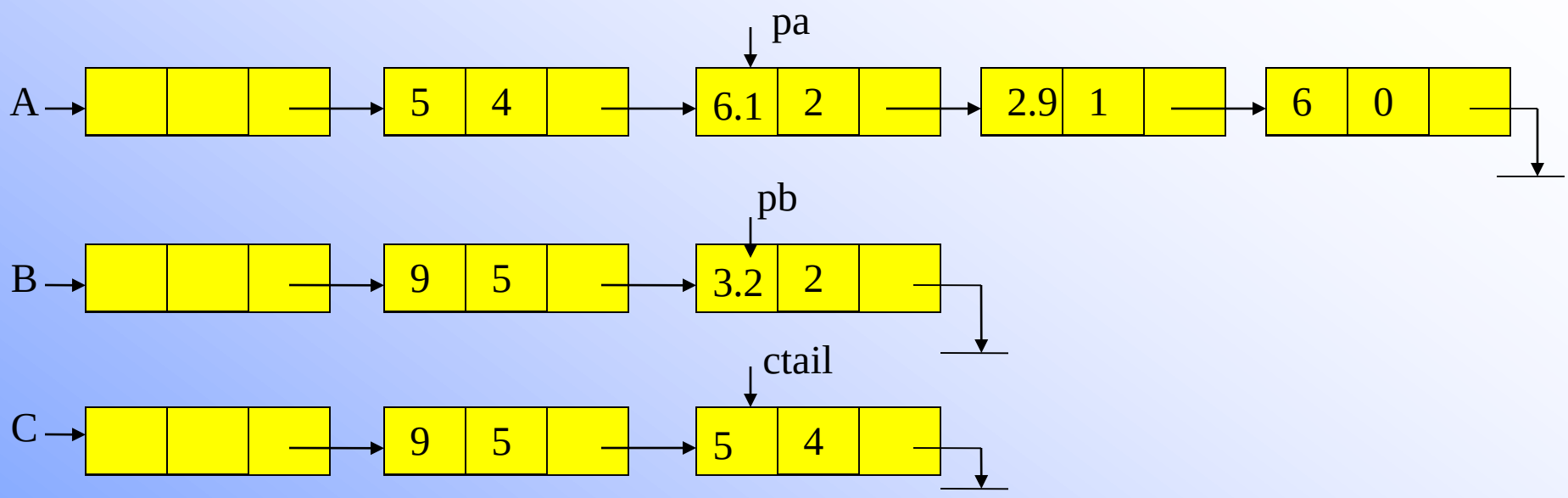
1. 初始狀態



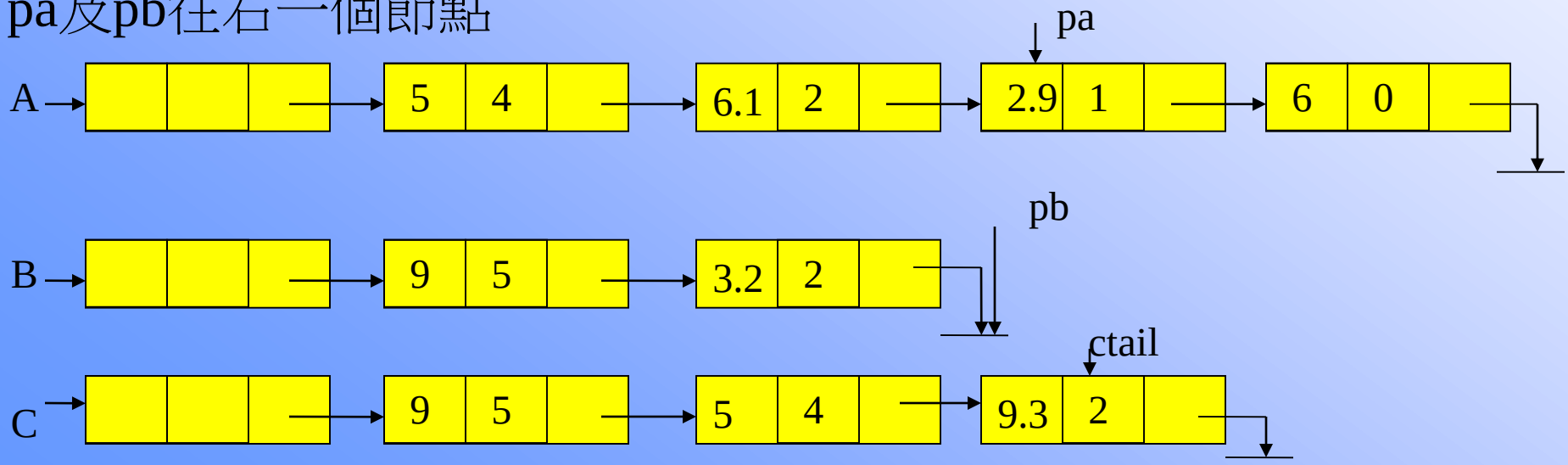
2. pb->exp(=5) 大於 pa->exp(=4) ，複製 *pb，pb往右一個節點



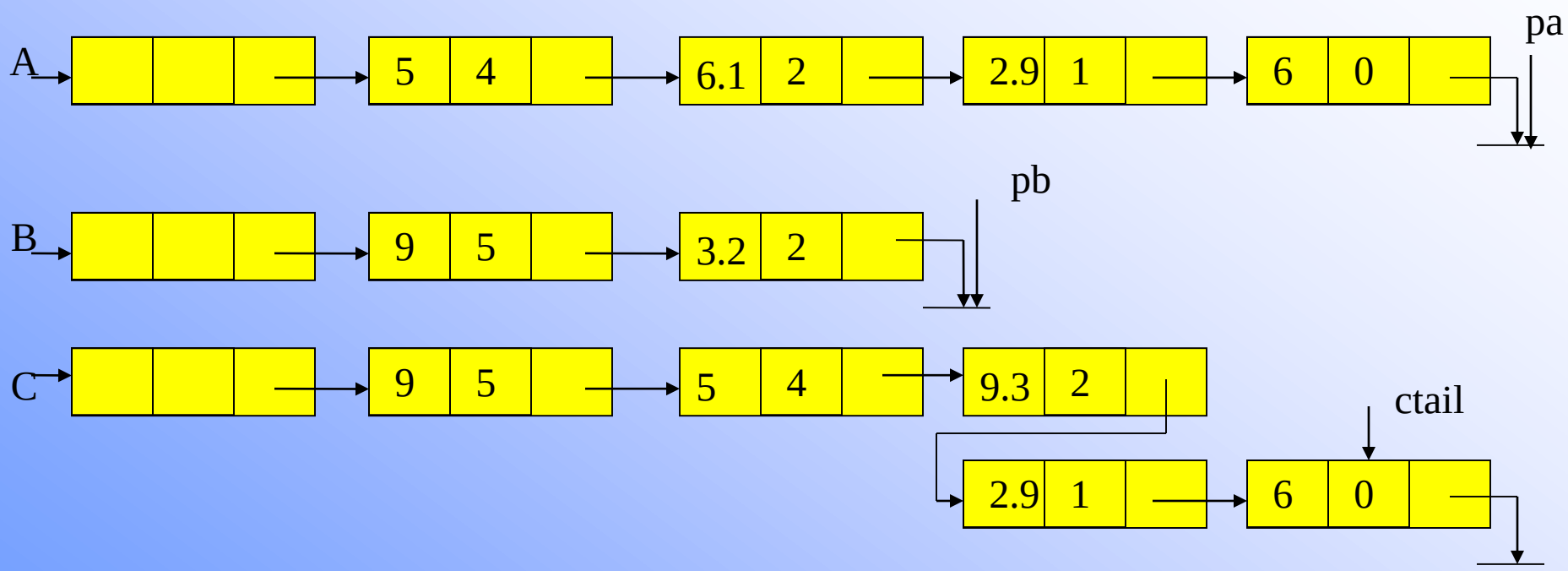
3. pa->exp(=4) 大於 pb->exp(=2) ，複製 *pa，pa往右一個節點



4. pa->exp 等於 pb->exp，新節點係數為 pa->coef 加 pb->coef，pa及pb往右一個節點



5. pb已經接地，將串列 A 剩下的節點複製到串列 C

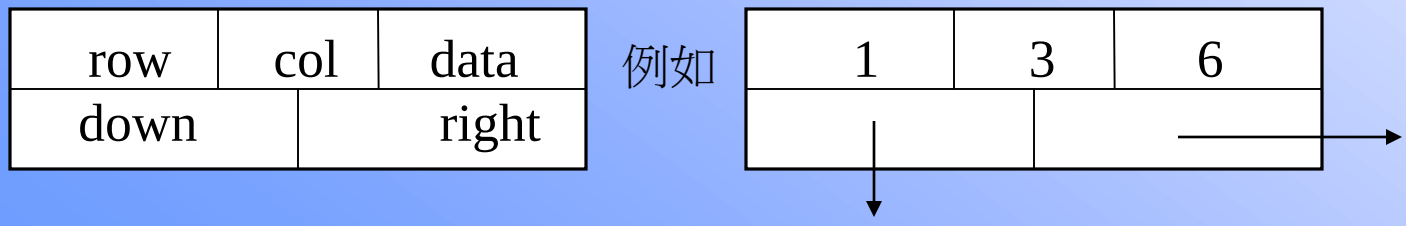


◎ 稀疏矩陣的表示

$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 2 & 0 \\ -1 & 0 & 0 & \mathbf{6} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 6 \end{pmatrix}_{5 \times 6}$$

M矩陣的 30 (5 × 6) 個元素中只有 9 個非零項，使用率只有 9/30 = 30%

◎使用節點來表示每個非零項，每個節點結構有 3 個資料欄位—
data、row、col 之外，兩個鏈結欄位— right、down。
同一列的節點藉著 right 鏈結成為一個串列，
同一行節點也藉著down鏈結成為一個串列。節點的結構可以圖示為：



$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 2 & 0 \\ -1 & 0 & 0 & \mathbf{6} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 6 \end{pmatrix}_{5 \times 6}$$