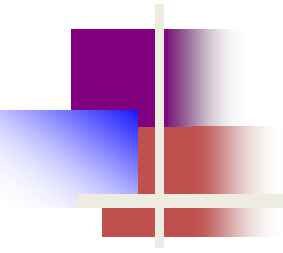


# 第五章
# 鏈結串列(Link List)

# 本章學習目標

1. 讓學生了解**動態記憶體**的**配置方法**及**釋回記憶體空間**。
2. 介紹**陣列**與**鏈結串列**的差異。
3. 介紹**鏈結串列的運作原理**。
4. 介紹**鏈結串列的應用**。

# 本章內容

5-1 串列(List)

5-2 陣列(Array)與鏈結串列(Link List)比較

5-3 動態記憶體配置(Dynamical Memory Allocation)

5-4 鏈結串列

5-5 單向鏈結串列(Singly Linked List)

5-6 鏈結堆疊與鏈結佇列

5-7 環狀鏈結串列(Circular Linked List)

5-8 雙向鏈結串列(Double Linked List)

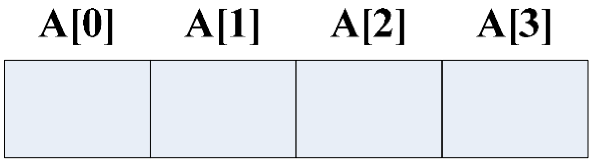
5-9 多項式串列表示法

# 5-1 串列(List)

**串列**是指**有次序**的**資料**組合而成。一般而言，串列可**分為兩種**，分別為**循序串列**(Sequential List)**與鏈結串列**(Linked List)。

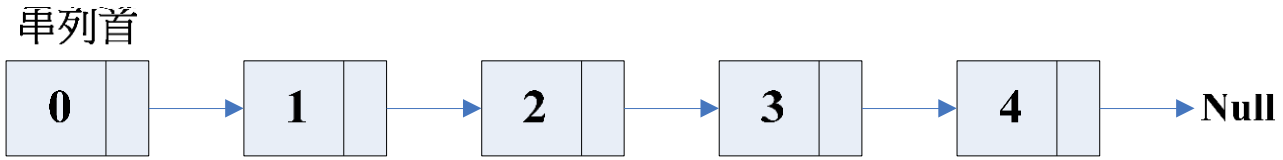
1.循序串列**(Sequential List)**：如陣列

它是以**連續的記憶體位置**呈現。




2.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



# 一、循序串列(Sequential List)

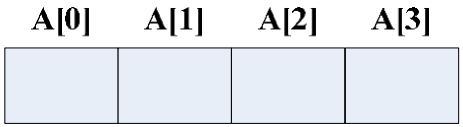
【定義】

是由**許多元素**A(0)、A(1)、A(2)、 ...、A(n-1)；n>0所組成，**元素與元素**之間有**線性的相對**關係，並且以**循序方式儲存**。

例如： **陣列**

【題目】宣告一個A **陣列**來**存放一年的四季名稱**

**步驟一**：建立空的陣列空間(**循序串列**)




**步驟二**：**依序存入**資料

【優點】

1. 資料的搜尋方便，可隨機讀取資料。
2. 設計時，資料結構簡單。

【缺點】

1. 事先需宣告固定記憶空間，彈性小。
2. 刪除或加入資料需移動大量資料。

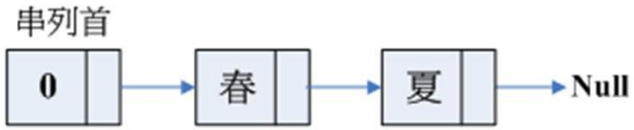
# 二、鏈結串列(Linked List)

【定義】它是以**指標**將**存放串列**的**記憶體鏈結起來**。

【題目】利用鏈結串列來**存放一年的四季名稱**

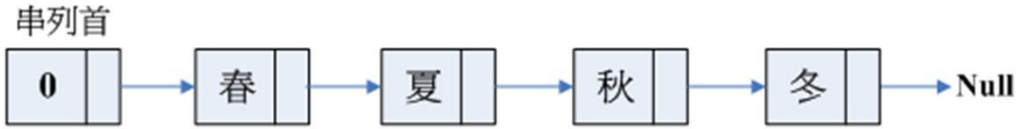
**步驟一**：建立**串列首**節點和**第一個節點**、**內容**及指向Null


**步驟二**：建立**第二個節點**、**內容**及指向Null




**步驟三**：建立**第三個節點**、**內容**及指向Null


**步驟四**：建立**第四個節點**、**內容**及指向Null



【優點】

1. 記憶點配置較有彈性
2. 串列分裂、合併容易

【缺點】

1. 搜尋某個元素時，可能會較為耗時
2. 可靠度低(指標(point)斷裂時，資料會遺失(lost))

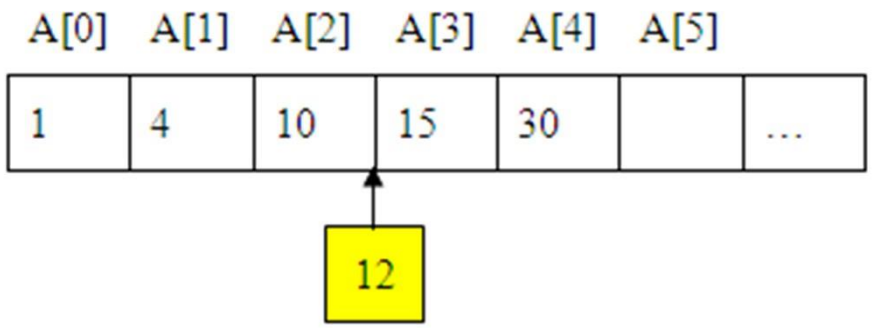
# 5-2 陣列與鏈結串列比較

基本上，**串列**依照**次序**來分為**有序串列**與**無序串列**。

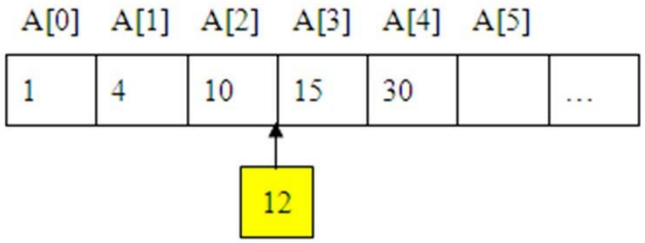
**一、有序串列**：資料儲存在**連續記憶空間**，例如「**陣列**」。

但**無法**任意**增/刪**空間。

【題目】假設目前有一**陣例**，其內容如下：此時欲**插入數字**12到10**與**15**之間**，請呈現**搬移過程**。

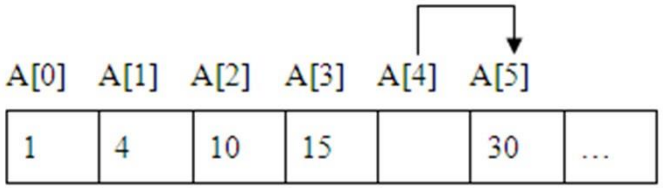


【題目】假設目前有一**陣例**，其內容如下：此時欲**插入數字**12到10**與**15**之間**，請呈現**搬移過程**。

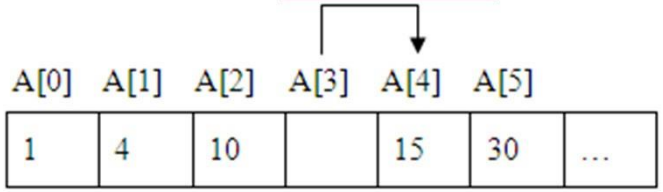



【解答】

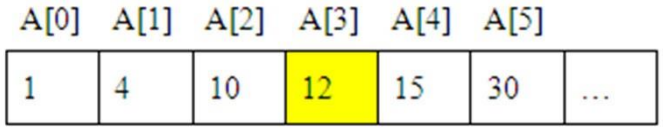
**步驟一**：將**數字**30往**右移一格**




**步驟二**：將**數字**15往**右移一格**

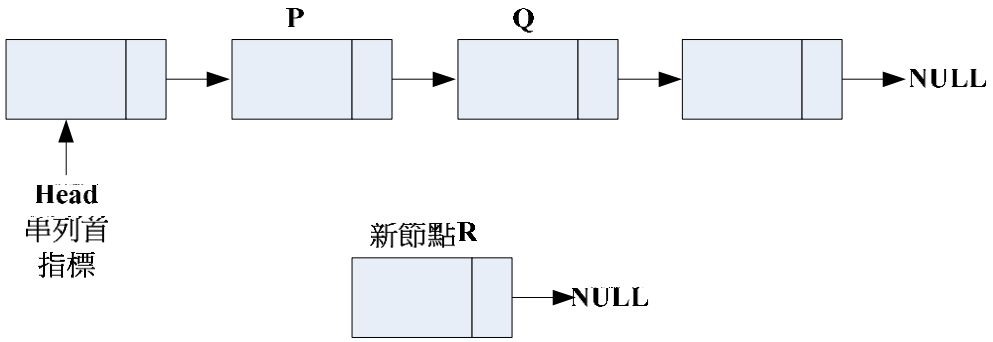



**步驟三**：將**數字**12**插入**到A[3]**空格**中



## 二、無序串列：資料乃是儲存在非連續性的記憶空間，它是透過指標可以彈性的在串列上增減元素。

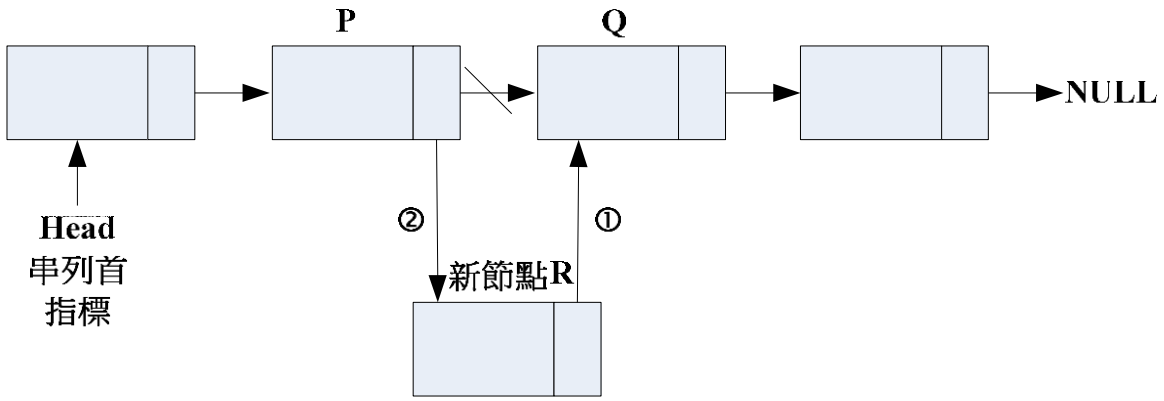
【題目】在鏈結串列中加入新節點




【解答】

步驟一：先把新節點R指向Q節點

步驟二：再將P節點接到節點R，並刪除P節點指向Q節點的指標



# 5-3 動態記憶體配置

**動態記憶體配置**(Dynamical Memory Allocation)**主要是利用鏈結串列的方法來解決一些無法事先預測處理資料多寡的問題。**

**【定義】**

**動態記憶體配置**是在等到執行階段，才向作業系統要求配置所需的記憶體空間。

**靜態記憶體配置**則是編譯階段時就配置記憶體空間。

# 【C語言作法】

## 1. 動態記憶體配置

**指標變數** = (**資料型態***) malloc(sizeof(**資料型態**));

**【舉例】**

```
int *pt;                                  //宣告一個指標變數pt
pt = (int *) malloc(sizeof(int));         //指標變數pt 則是指向一個整數的記憶體
```

## 2. 釋放記憶體運算子

free(**指標變數**);

**【舉例】**

```
free(pt);                                 //把前面pt 所配置的記憶體釋放出來
```

# 【C++作法】

## 1. 動態記憶體配置

資料型態 *指標名稱 =new 資料型態;

或

資料型態 *指標名稱 =new 資料型態[陣列長度];

【舉例】

```
int *pt=new int;          //動態地配置記憶體空間，宣告一個指標變數pt指向該整數的記憶體
```

## 2. 釋放記憶體運算子

delete 指標名稱;

【舉例】

```
delete pt;                //前面pt所配置的記憶體釋放出來
```

# 5-3.1 靜態與動態資料結構

在**靜態資料結構**中，由於資料所佔用的**空間大小**及**資料數目**要**事先宣告**，因此，**在執行時**，**空間的大小**及**資料數目**是**不會改變**的(因為利用**陣列結構**)。

在**動態資料結構**中，資料所佔用之**空間大小**及**資料數目**不必事先宣告，**在執行時**視實際需要而**動態的增加**或**減少**(因為利用**串列結構**)。

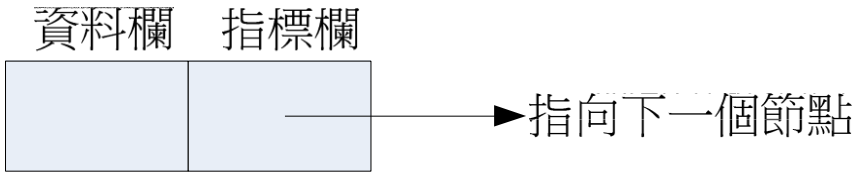
表5-1 靜態與動態資料結構比較表

| | 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 |

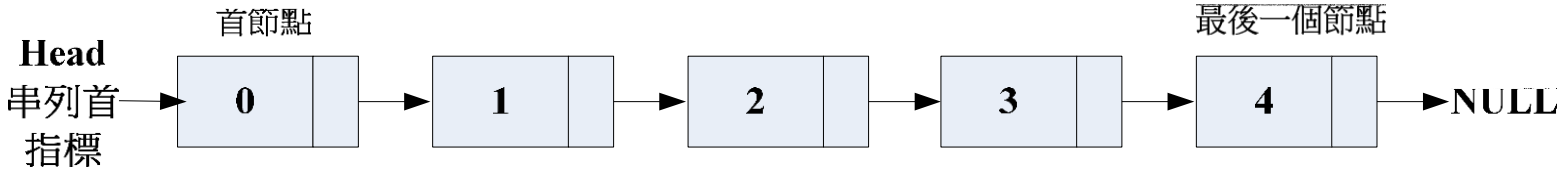
# 5-3.2 動態記憶體在資料結構中的應用

由於使用「陣列」實作堆疊，會受到陣列大小必須事先宣告的限制，因此，我們可以使用鏈結（Link）的方式來實作堆疊，它是以「動態記憶體」宣告的方式來新增每一個元素。

鏈結（Link）是由節點組成，每一個節點儲存資料之外，還儲存指向下一個節點的位置，如下圖所示：




如果想要將兩個或兩個以上的節點串連在一起的情況如下所示：



# 【串列的加入與刪除動作】

一般而言，串列的動作會有兩種不同的方法：

**一、加入新節點**

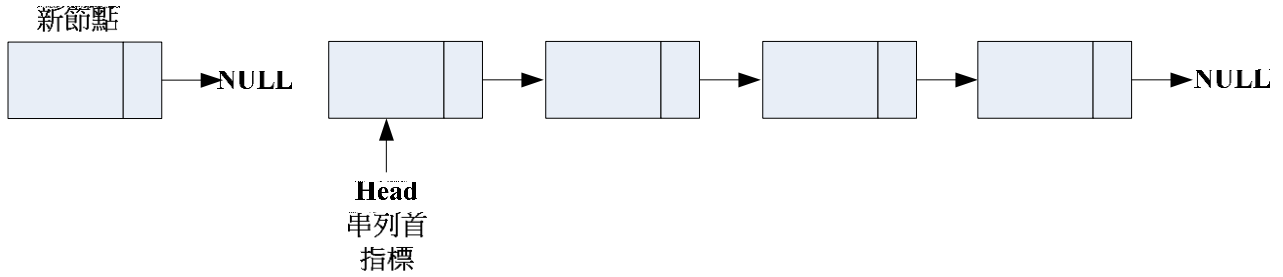
只需把**新節點**的**指標指向串列首**，再把**串列首**移到**新節點**上即可。

**二、刪除節點**

只要把**串列首指標指向第二個節點**即可。

【題目1】假設目前已經有一個鏈結串列，現在欲再加入一個新節點，請問如何進行？

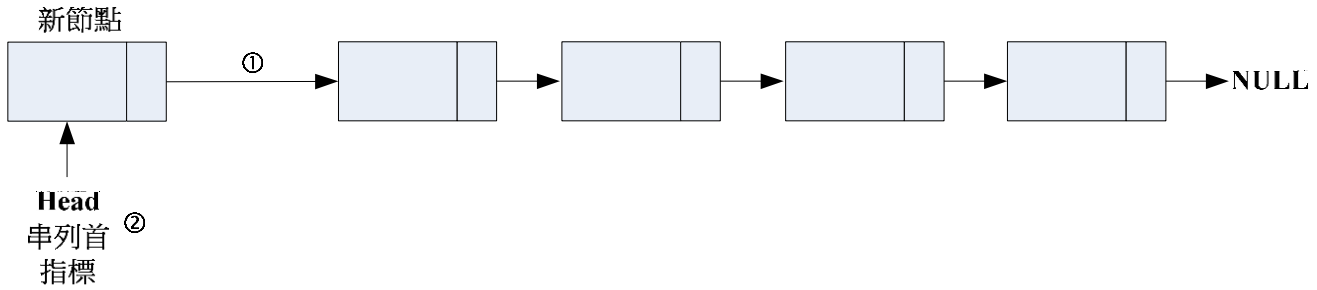
**原來串列：**




【解答】

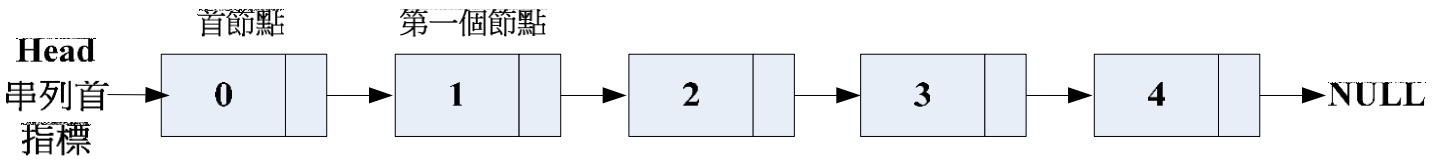
只需把新節點的指標指向串列首，再把串列首移到新節點上即可。

**新增後的串列：**



【題目2】假設目前已經有一個鏈結串列，現在欲**刪除第一個節點**，請問如何進行？

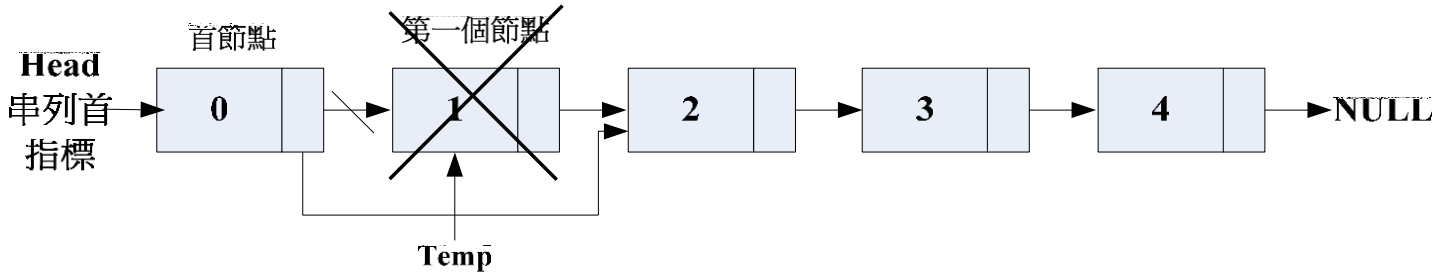
**原來串列：**




【解答】

只要把**串列首指標指向第二個節點**即可。

**刪除後的串列：**



# 5-4 鏈結串列

【定義】

由一個或一個以上動態記憶體分配的「節點」所組成，每一個節點是由兩個欄位所組成，分別存放「資料」及「指標」，此指標稱為鏈結(Link)。

【特性】

1. 各節點不一定要佔用連續的記憶體空間。
2. 各節點之資料型態不一定要相同。
3. 插入/刪除節點方便。

【分類】

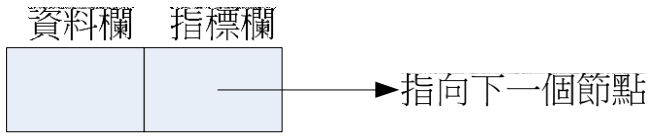
1. 單向鏈結串列(Single Linked List)
2. 雙向鏈結串列(Double Linked List)

# 一、單向鏈結串列(Single Linked List)

**單向鏈結串列是串列中最常用的一種，它像火車一般，**

**所有節點串成一列。**

**【節點結構】**




其中：

1. 資料欄：用來儲存此節點的資料
2. 指標欄：用來指向下一個節點的位置

# 二、雙向鏈結串列(Double Linked List)

【定義】

是指由一個「**資料欄**」與**兩個**「**指標欄**」所組成。其中，「**資料欄**」用來儲存此節點的資料，而**兩個**「**指標欄**」的其中一個是用來**指向前一個的節點位置**，另一個則用來**指向下一個節點位置**。

如下圖所示：

【雙向鏈結串列之節點結構】


其中：

1. Data欄：用來**儲存**此**節點的資料**
2. LLink欄：用來指向**前一個**節點的位置
3. RLink欄：用來指向**下一個**節點的位置

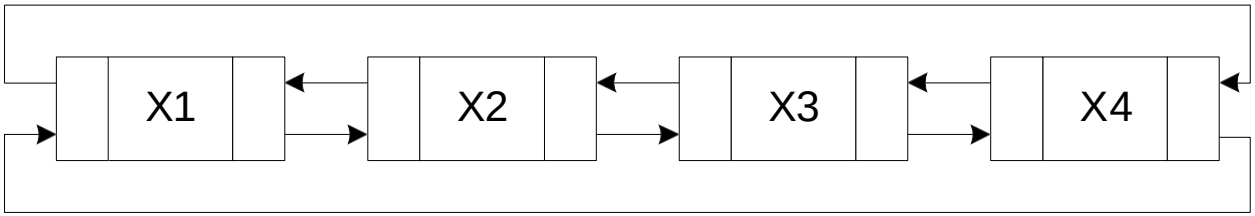


表5-3單向鏈結串列與雙向鏈結串列之比較

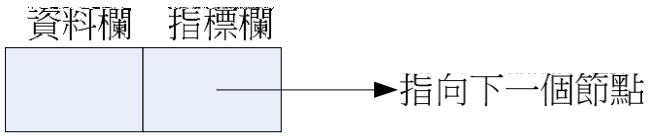
| | 單向鏈結串列 | 雙向鏈結串列 |
|---|---|---|
| 節點結構 | 一個資料欄(Data)和一個指標欄(Link)。 | 一個資料欄(Data)和兩個指標欄(LLink和RLink)。 |
| 優點 | 較節省空間 | 1. 容易找出某節點的前後節點。<br>2. 當一方向之鏈結斷落時可以用另一方向之鏈結來修復之。 |
| 缺點 | 當鏈結斷落時無法修護，將造成資料之遺失。 | 1. 較浪費空間。<br>2. 加入一個新節點或刪除一個節點所須改變之鏈結較多，因此較費時。 |

# 5-5 單向鏈結串列

【定義】

是指由「**資料欄**」與「**指標欄**」所組成。其中，「**資料欄**」用來儲存此節點的資料，「**指標欄**」用來指向下一個節點的位置。

【單向鏈結串列之節點結構】

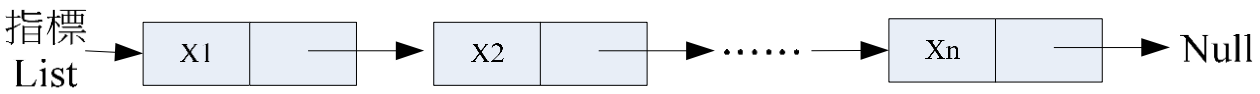



其中：

1.**資料欄**：用來儲存此節點的資料

2.**指標欄**：用來指向下一個節點的位置

例如：有n個單向鏈結串列，如下圖所示：



# 5-5.1 單向鏈結串列的建立

【定義】利用動態記憶體來產生鏈結串列的新節點。

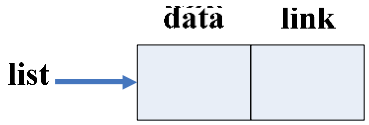
【題目】先建立一個新節點，再存放資料10到資料欄，

並且讓新節點的指標欄位指向空節點。

```
01  list=NewNode();      //先建立一個新節點
02  list->data=10;       //再存放資料10到資料欄
03  list->link=NULL;     //新節點的指標欄位指向空節點
```

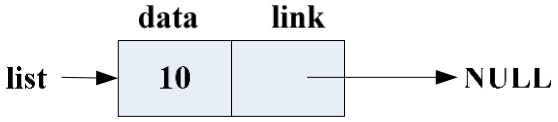
【解答】

步驟一：先建立一個新節點




步驟二：再存放資料10到資料欄


步驟三：新節點的指標欄位指向空節點



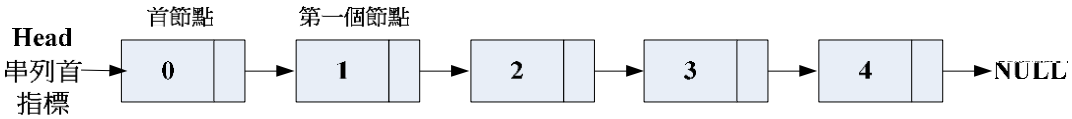
# 5-5.2 單向鏈結串列中節點的刪除

基本上，討論鏈結串列內的**節點刪除**時，**依據所刪除節點的位置會有三種不同的情形：**

1. 刪除**串列**的**第一個節點**
2. 刪除**串列內**的**中間節點**
3. 刪除**串列後**的**最後一個節點**

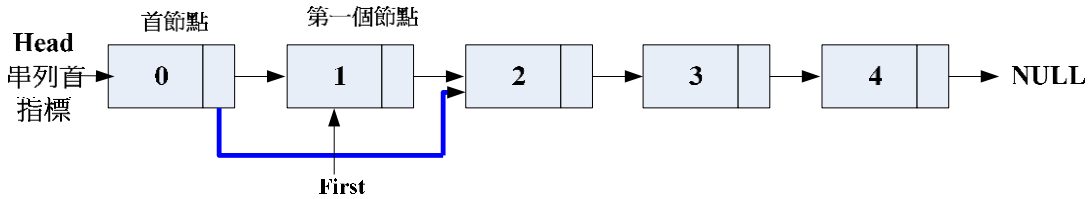
1.刪除串列的第一個節點：只要把串列首指標指向第二個節點即可。

(1)原來串列：

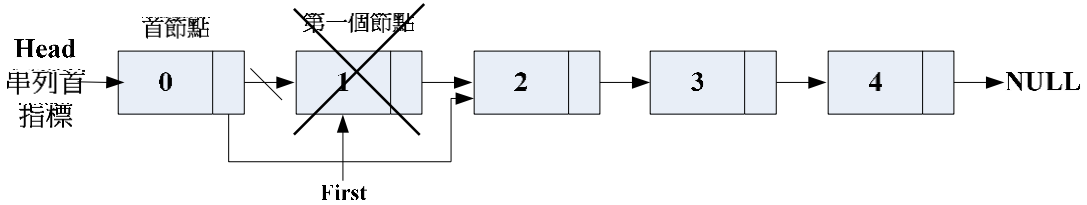



(2)刪除後的串列：

步驟一：串列首指標指向第二個節點

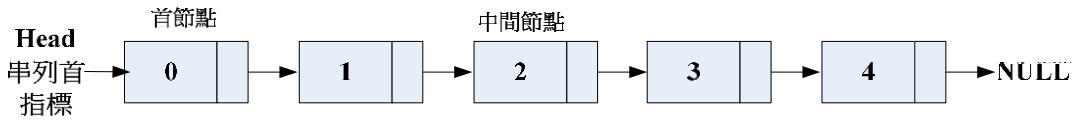



步驟二：將First節點釋放出記憶體空間



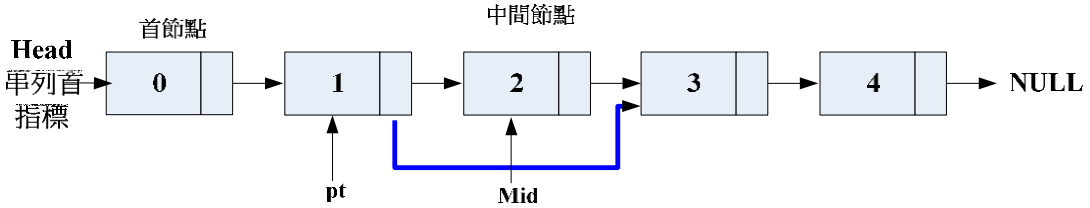
2.**刪除串列內的中間節點**：只要**刪除節點的前一個節點的指標**，
指向**欲刪除節點的下一個節點**即可。

(1)原來串列：

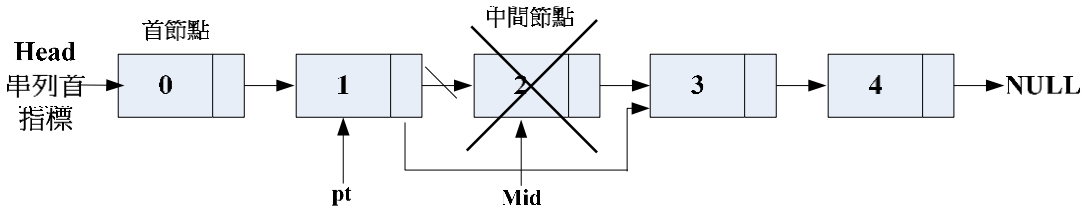



(2)**刪除後**的串列：

**步驟一**：將欲刪除的**中間節點(Mid)**的指標指定給pt節點的指標

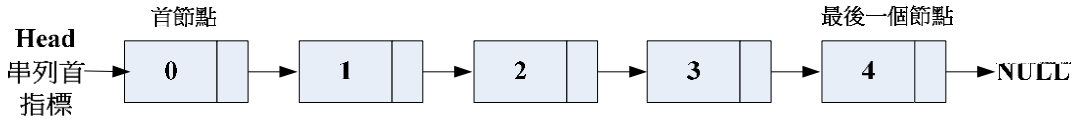



**步驟二**：再將**Mid節點釋放出**記憶體空間



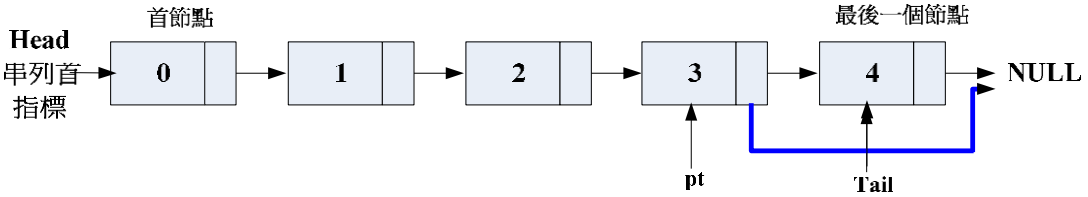
3.**刪除串列的最後一個節點**：只要指向**最後一個節點的指標**，直接**指向**NULL即可。

(1)原來串列：

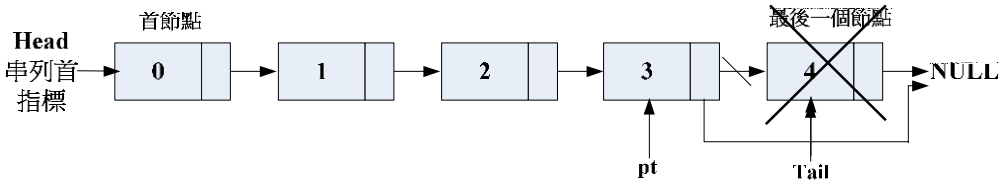



(2)**刪除後**的串列：

**步驟一**：將欲刪除的**最後一個節點的指標**指定給pt節點的指標




**步驟二**：再將Tail**節點釋放出**記憶體空間



# 5-5.3 單向鏈結串列的插入節點

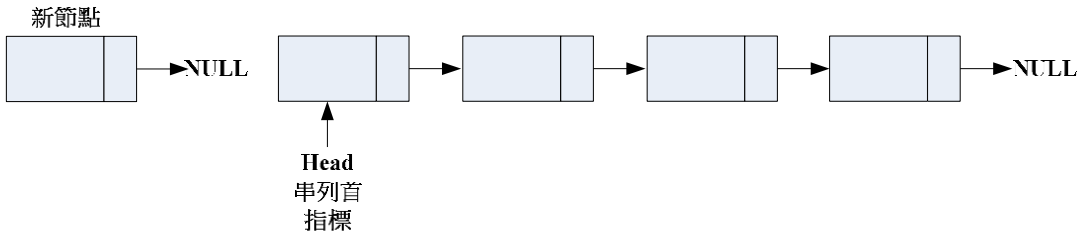
基本上，討論在鏈結串列內的插入節點時，依據所插入節點的位置會有三種不同的情形：

1. 在串列的第一個節點前插入節點
2. 在串列的最後一個節點後面插入節點
3. 在串列的中間位置插入節點

## 1.在串列的第一個節點前插入節點：

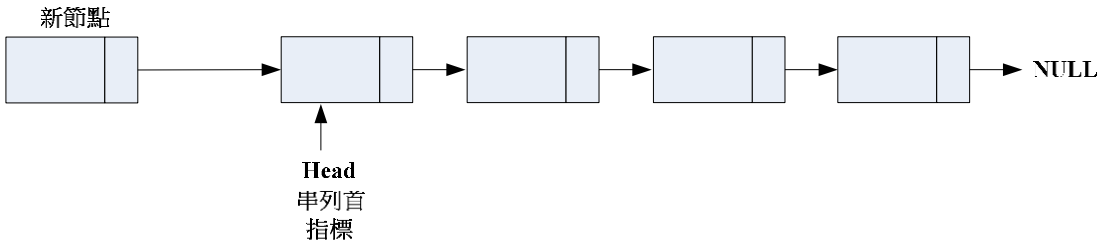
只需把新節點的指標指向串列首，再把串列首移到新節點上即可。

(1)原來串列：

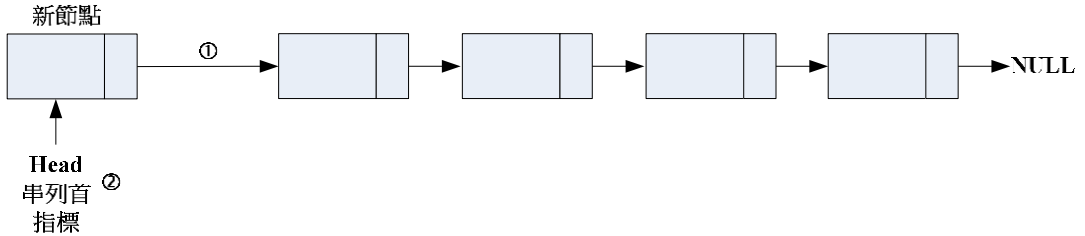



(2)加入後的串列：

**步驟一**：把新節點的指標指向串列首



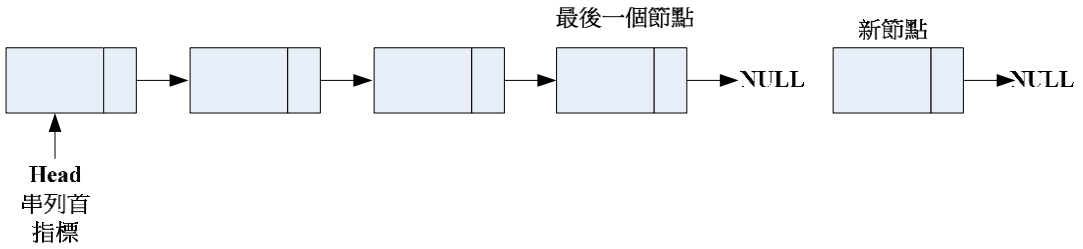

**步驟二**：再把串列首移到新節點上



## 2.在串列的最後一個節點後面插入節點：

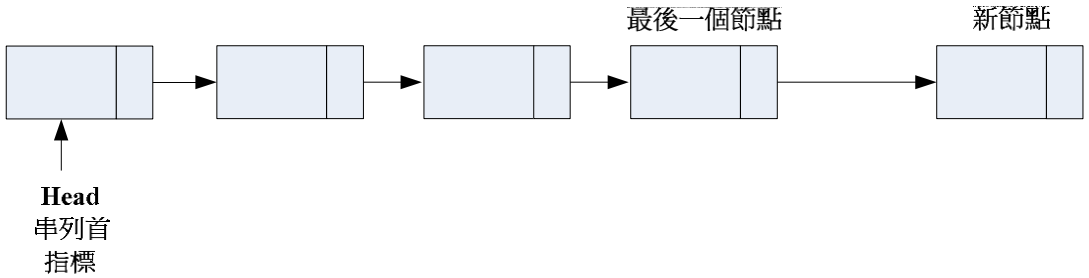
把串列的最後一個節點的指標指向新節點，新節點再指向NULL即可。

(1)原來串列：

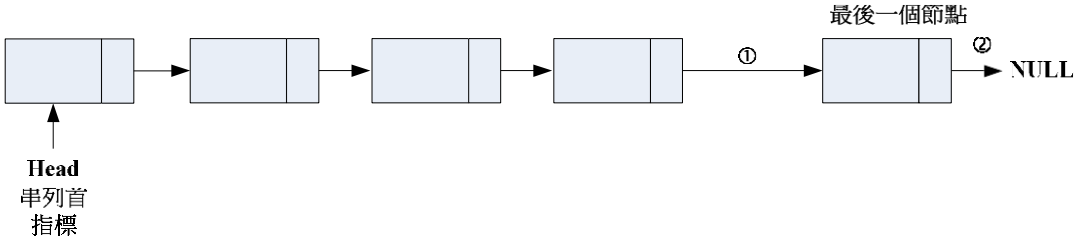



(2)加入後的串列：

步驟一：把串列的最後一個節點的指標指向新節點



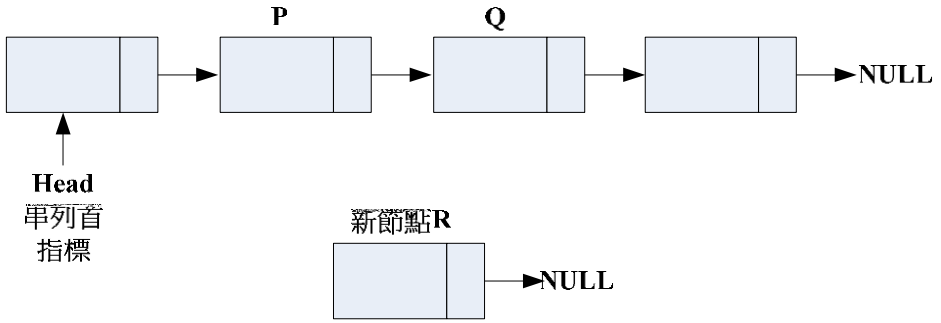

步驟二：新節點再指向NULL



## 3.在串列的中間位置插入節點：

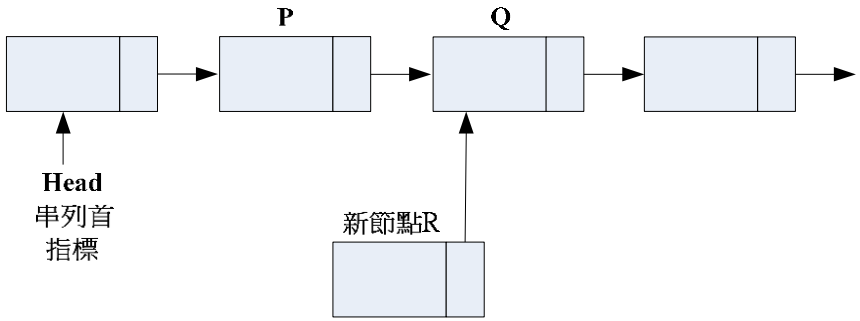
如果插入的節點是在P與Q之間，只要先把新節點R指向Q節點，再將P節點接到節點R即可。

(1)原來串列：




(2)加入後的串列：

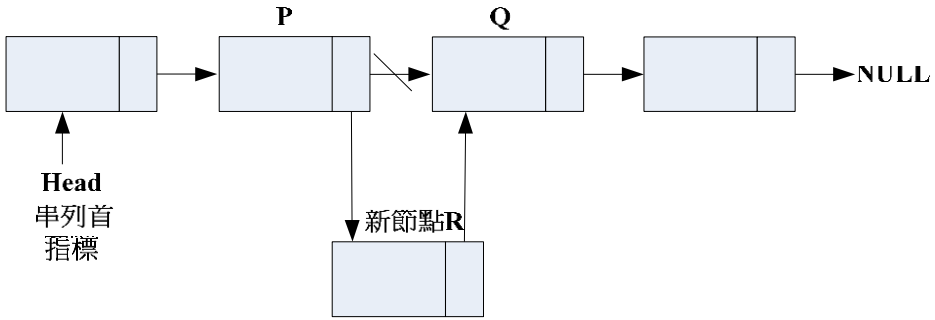
步驟一：先把新節點R指向Q節點



## 3.在串列的中間位置插入節點：(續)

(2)加入後的串列：

**步驟二：再將P節點接到節點R**



# 5-6 鏈結堆疊與鏈結佇列

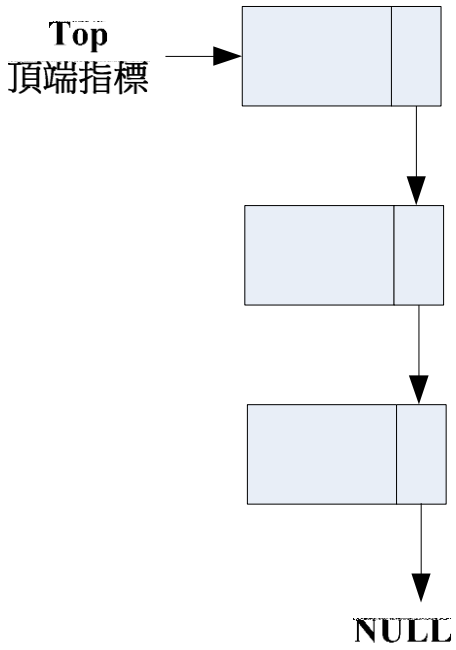
【引言】

由於我們前面介紹的**堆疊**具有**後進先出**(LIFO)與**佇列**具有**先進先出**(FIFO)等特性，並且都是使用**陣列**來實作**堆疊與佇列**。因此，在使用之前必須要**先宣告陣列的大小**。

我們在本單元中將利用**鏈結堆疊**與**鏈結佇列**來解決此問題，所以我們就可以**不考慮堆疊或佇列被放滿**的情況。

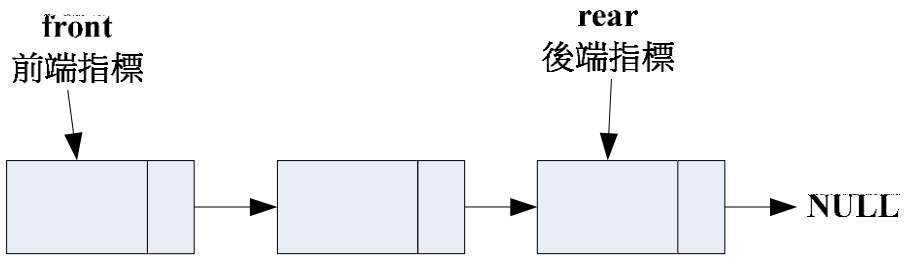
# 一、鏈結堆疊

【定義】指利用**鏈結串列**來呈現的一種**堆疊**。如下圖所示：



# 二、鏈結佇列

【定義】指利用**鏈結串列**來呈現的一種**佇列**。如下圖所示：



# 5-6.1　鏈結堆疊

基本上，鏈結堆疊有兩個基本動作：**加入**與**刪除**資料

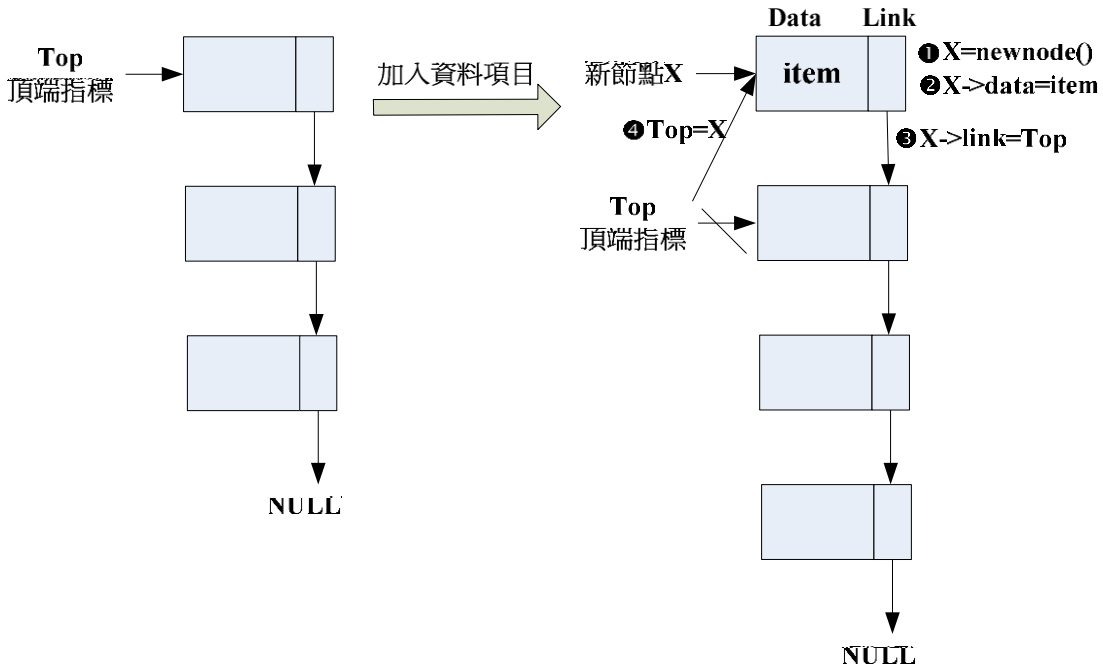
一、**加入**新資料項目到**鏈結堆疊**中

二、**刪除**鏈結堆疊的**頂端資料**

# 一、加入新資料項目到鏈結堆疊中

首先必須先**產生一個新資料項**item的新節點，假設此**節點為**X，

其**加入新資料項目**到**鏈結堆疊**中的過程如下所示：

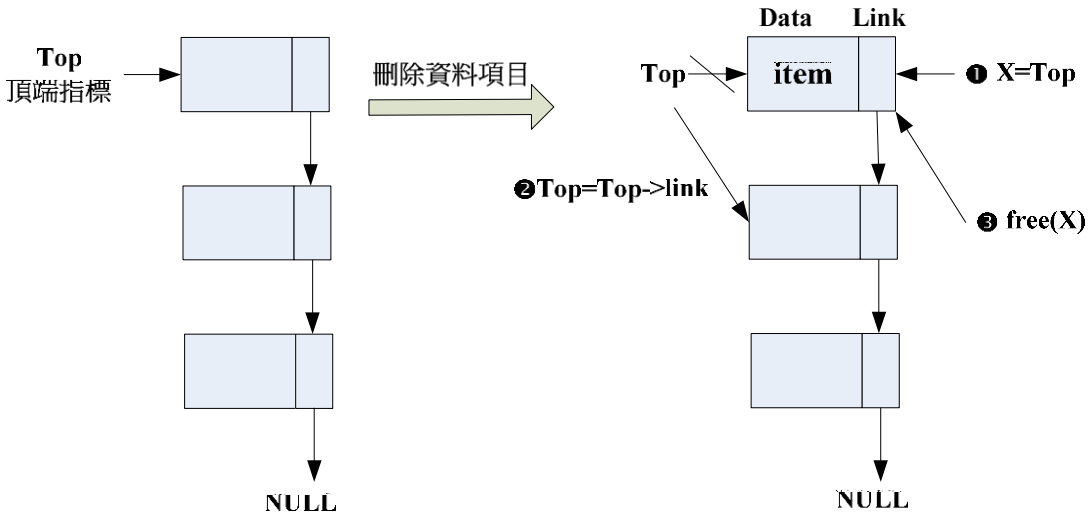


# 【演算法】

```
//加入新資料項目到鏈結堆疊中的演算法
Procedure AddStack (item, Top)   //item是加入資料，Top是目前的頂端位置
Begin
  X=newnode( );          //產生一個新節點
  X→data=item;           //加入資料到新節點中
  X → link=Top;          //再將新節點指向Top
  Top=X;                 //新節點成為Top節點
End
End Procedure
```

# 二、刪除鏈結堆疊的頂端資料

**刪除**頂端資料時，必須利用一個**指標指向頂端節點**，以便在**頂端指標** Top改為**指向第二個節點**後，還能**參照**到**原來的頂端節點**，因此，我們就可以將其**空間歸還給系統**。其過程如下所示：



# 【演算法】

```
//刪除鏈結堆疊的頂端資料之演算法
Procedure DelStack (item, Top)   //item是取出資料，Top是目前的頂端位置
Begin
  If  Top=Null  then  STACK_EMPTY;   //判斷鏈結堆疊「是否為空」，如果不為空，則
    X=Top;                  //指標指向頂端節點
    Top=Top->link;          //頂端指標Top改指向第二個節點
    free(X);                //空間歸還給系統
End
End Procedure
```

# 5-6.2　鏈結佇列

基本上，鏈結佇列有兩個基本動作：加入與刪除資料

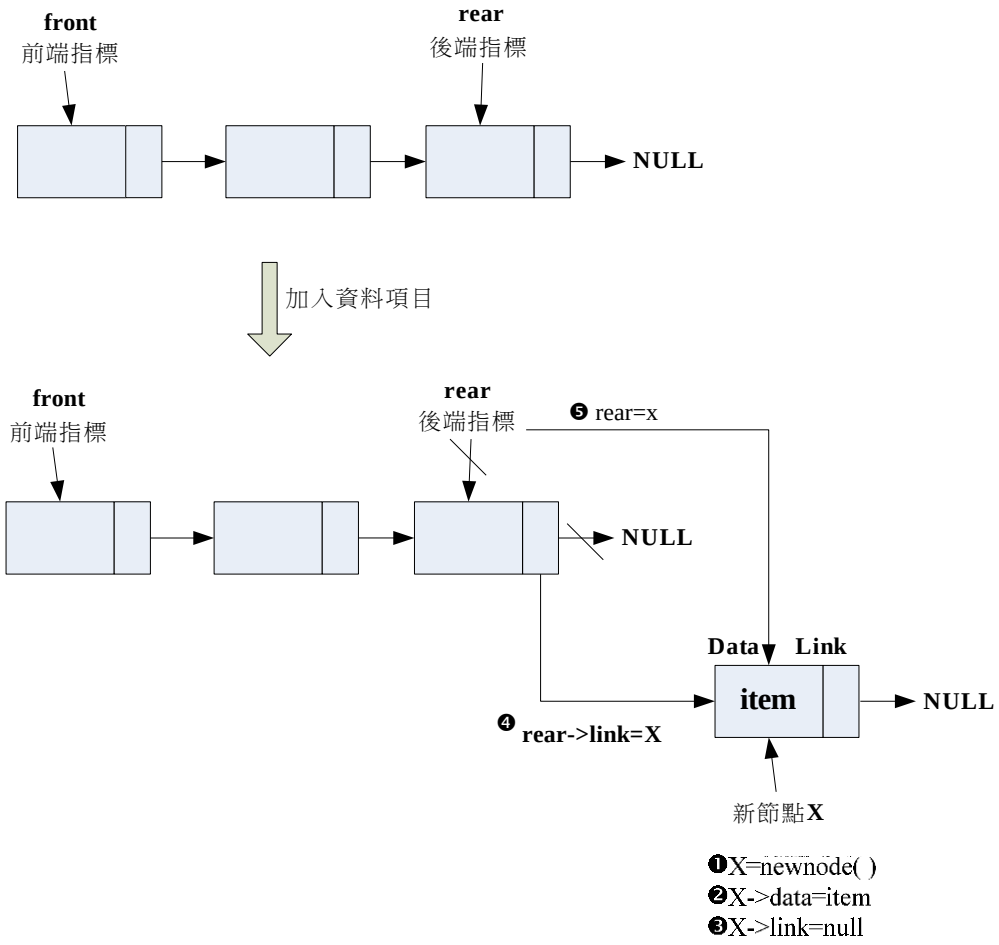
一、加入新資料項目到鏈結佇列的尾端

二、刪除鏈結佇列的頂端資料

# 一、加入新資料項目到鏈結佇列的尾端

首先必須要**產生**一個**新資料項**item**的新節點**，假設此**節點為**X，

其**加入新資料項目**到**鏈結佇列的尾端**中的過程如下所示：

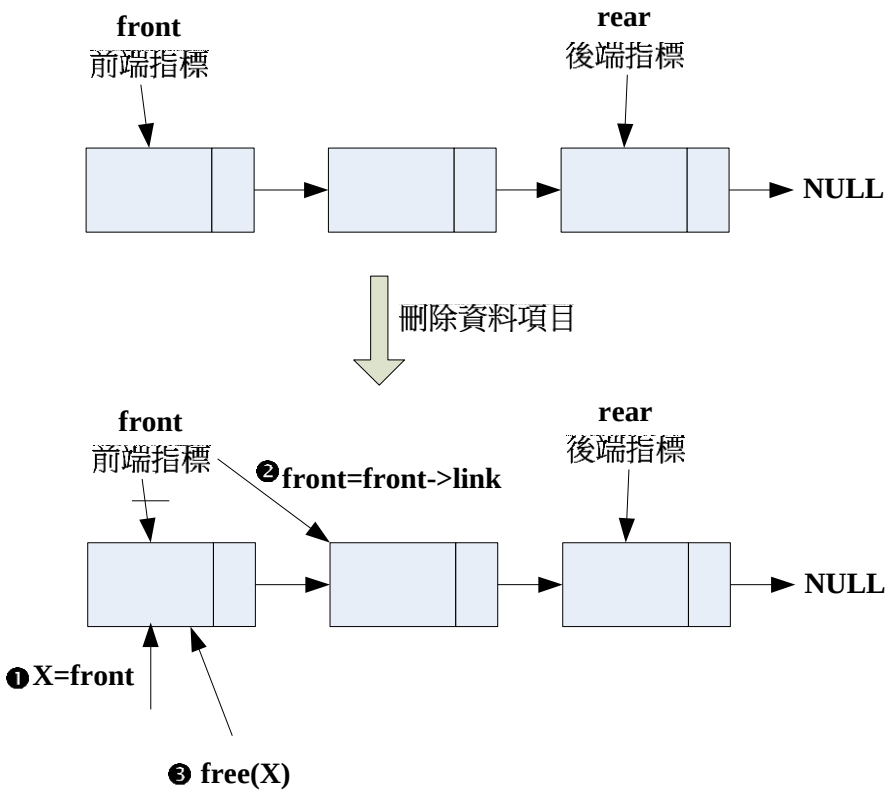


# 【演算法】

```
Procedure AddQueue (item, front, rear)
//item是加入資料，front是目前端位置，rear是目前尾端位置
Begin
  X=newnode( );          //產生一個新節點
  X->data=item;          //加入資料到新節點中
  X->link=null;          //再將新節點指向NULL
  rear->link=X;          //Rear端指向新節點
  rear=X;                //新節點成為後端節點
End
End Procedure
```

# 二、刪除鏈結佇列的頂端資料

**刪除**鏈結佇列**前端資料**，如同刪除**鏈結堆疊的頂端資料一樣**，同樣**必須先利用一個指標指向前端節點**，然後改變**前端指標**front至**下一個節點**，最後歸還**原來的前端節點空間給系統**。其刪除鏈結佇列的頂端資料的過程如下所示：



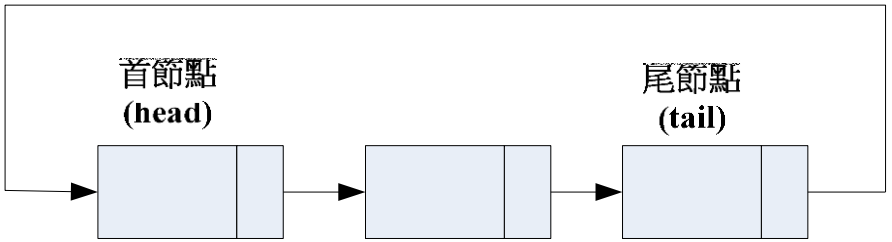
# 【演算法】

```
Procedure DelQueue (item, front, rear)
 //item是刪除資料，front是目前端位置，rear是目前尾端位置
Begin
  If  front=null then Queue_Empty ;   //判斷鏈結堆疊「是否為空」，如果不為空，則
     ❶ X = front                  //先利用一個指標指向前端節點
     ❷front=front->link           //然後改變前端指標front至下一個節點
     ❸free(X)                     //最後歸還原來的前端節點空間給系統
  End If
End
End Procedure
```

# 5-7 環狀鏈結串列(Circular Linked List)

【定義】

它是指將**鏈結串列**的**最後一個節點**的指標**指向鏈結串列**結構的**第一個節點**，我們就稱此串列為**環狀鏈結串列**。如下圖所示。




說明：**環狀鏈結串列**的建立只需將**最後1個節點的**next**指標指向第1個節點**，即可建立環狀鏈結串列。

```
tail->next = head;
```

# 5-7.1　環狀鏈結串列-加入節點

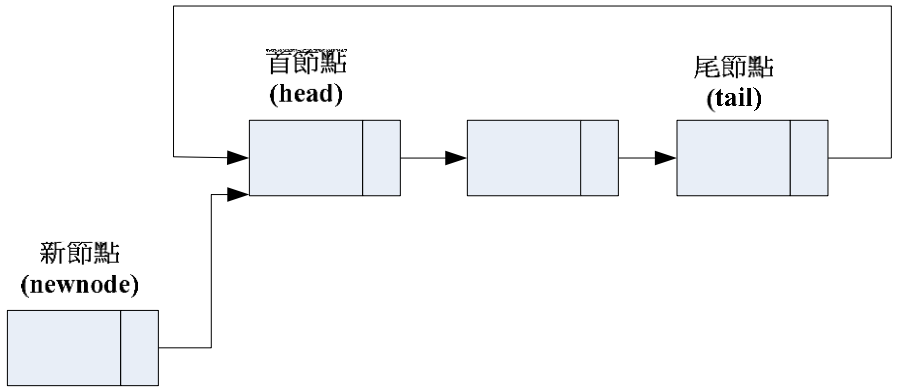
基本上，在討論環狀鏈結串列要加入節點時，會依據所加入節點的位置會有三種不同的情形：

1. 加入新資料項目到環狀鏈結串列的前端
2. 加入新資料項目到環狀鏈結串列的中間
3. 加入新資料項目到環狀鏈結串列的尾端

# 一、加入新資料項目到環狀鏈結串列的前端

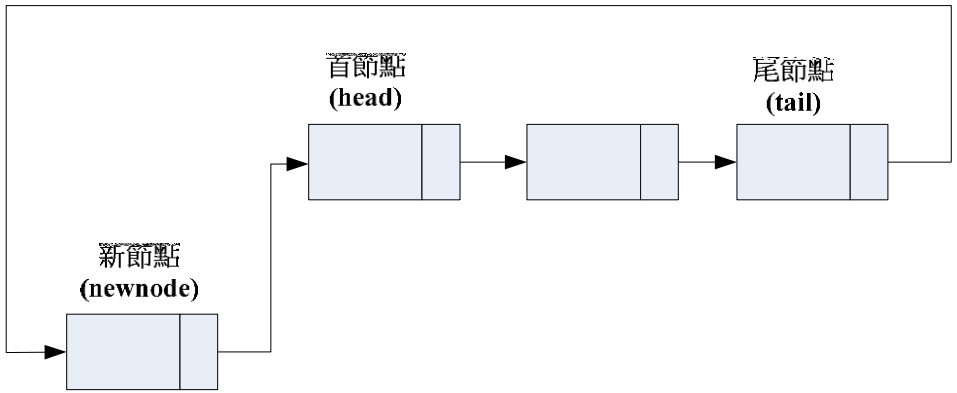
**步驟1：將新節點的next指標指向串列的第1個節點。**

newnode->next = head



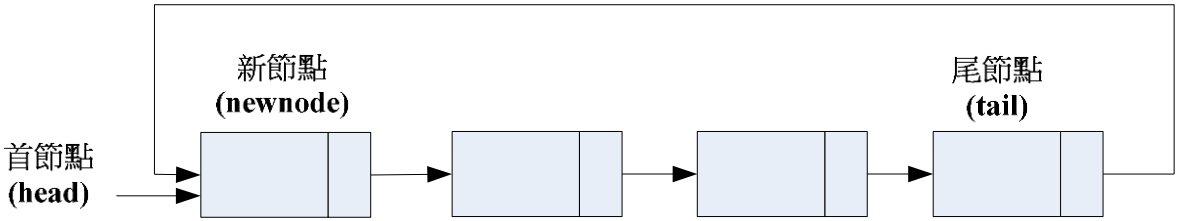
步驟2： 然後找到最後1個節點(tail)且將其指標指向新節點(newnode)。

Tail->next=newnode




步驟3： 最後再將首節點指向新節點，新節點成為串列的第1個節點。

head=newnode;



# 二、加入新資料項目到環狀鏈結串列的中間

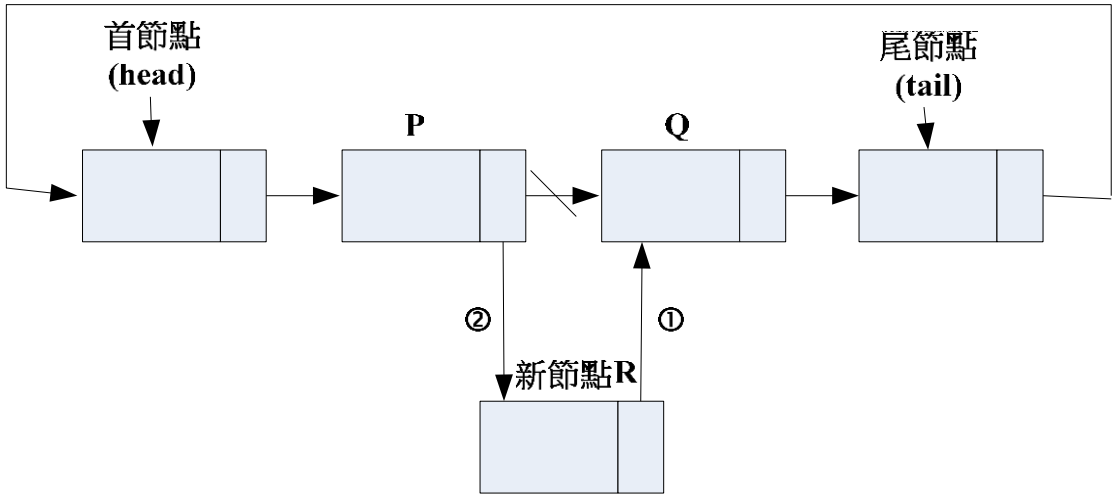
如果插入的節點是在P與Q之間，只要將新節點**R**的**next**指標指向節點**Q** ，再將節點**P**的指標指向新節點**R**即可。

**步驟1**：將新節點R的next指標指向節點Q。

```
R->  next = P->  next;        //先把新節點R指向Q節點
```

**步驟2**：將節點P的指標指向新節點R。

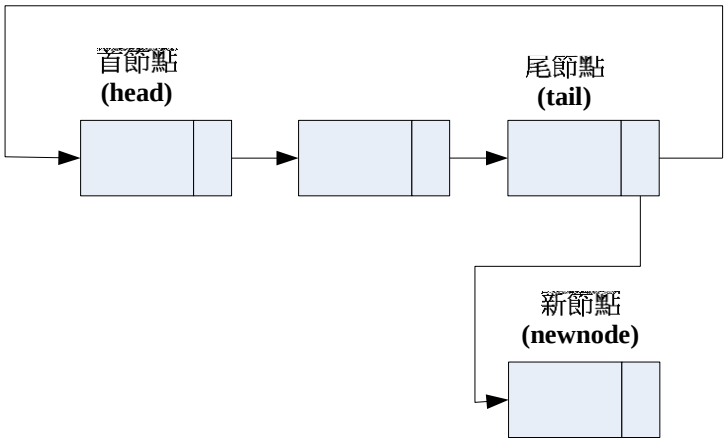
```
P->  next = R;                //再將P節點接到節點R
```



# 三、加入新資料項目到環狀鏈結串列的尾端

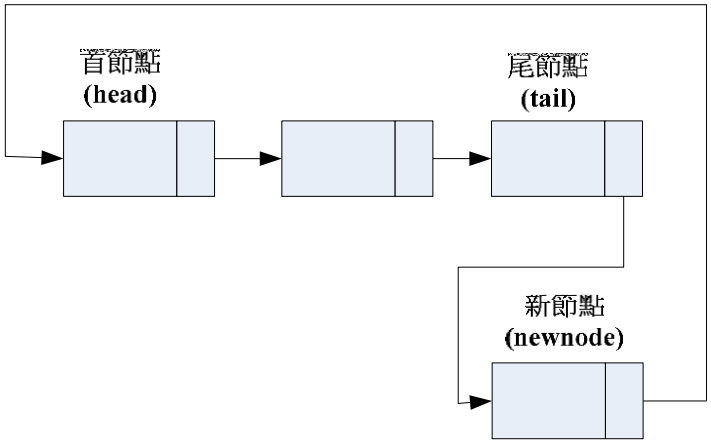
步驟1：將最後一節點(尾端)的next指標指向新節點newnode。

```
tail->next=newnode;
```



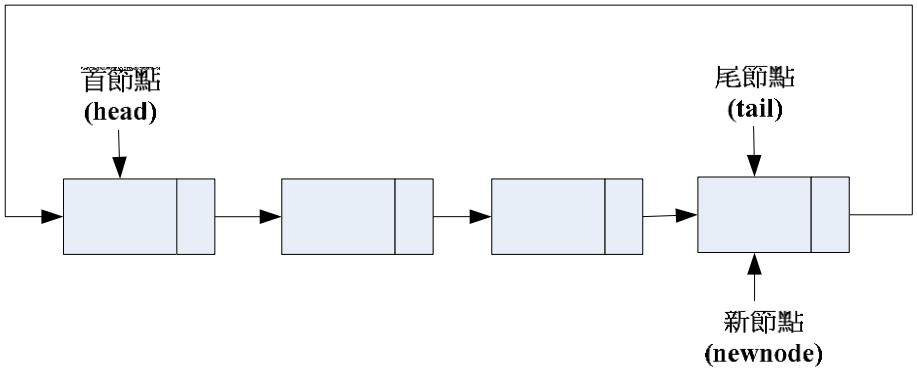
步驟2： 新節點的next指標指向首節點(head)。

```
newnode -> next=head;
```




步驟3： 再將尾節點指向新節點，新節點成為串列的最後1個節點。

```
tail=newnode;
```



# 5-7.2 環狀鏈結串列-刪除節點

基本上，在討論環狀鏈結串列要刪除節點時，會依據所刪除節點的位置會有兩種不同的情形：

一、刪除環狀串列的第一個節點

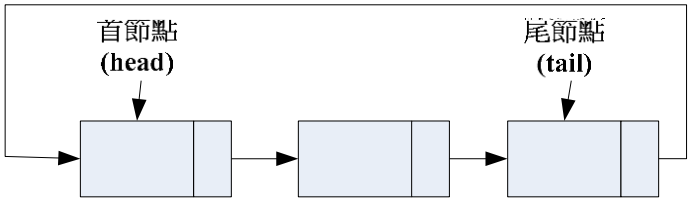
二、刪除環狀串列內的中間節點

# 一、刪除環狀串列的第一個節點

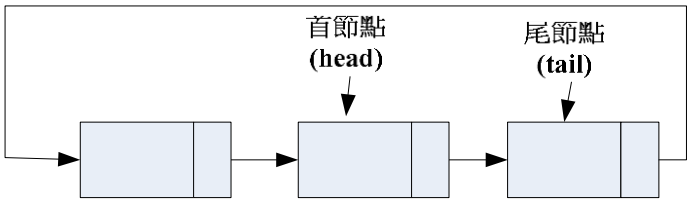
**步驟 1：將環狀串列首指標指向第二個節點。**

```
head = head->next;
```

(1)原始環狀**串列**：

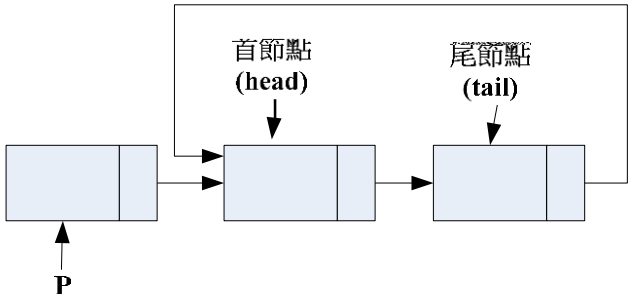



(2)**首指標指向第二個節點**：



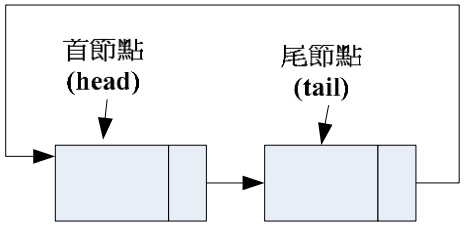
**步驟 2：將最後1個節點的next指標指向第2個節點。**

```
tail->next = head->next;
```




**步驟3：將第1個節點釋放出記憶體空間。**

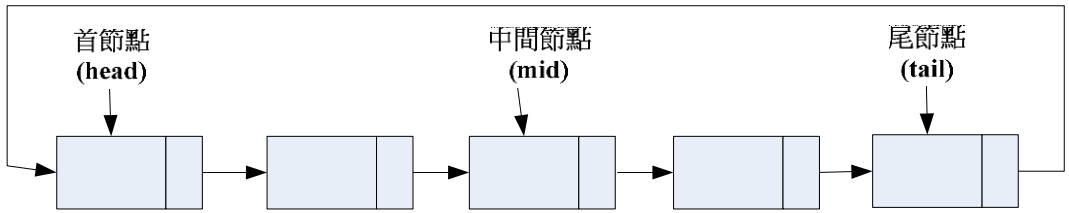
```
free(p);
```



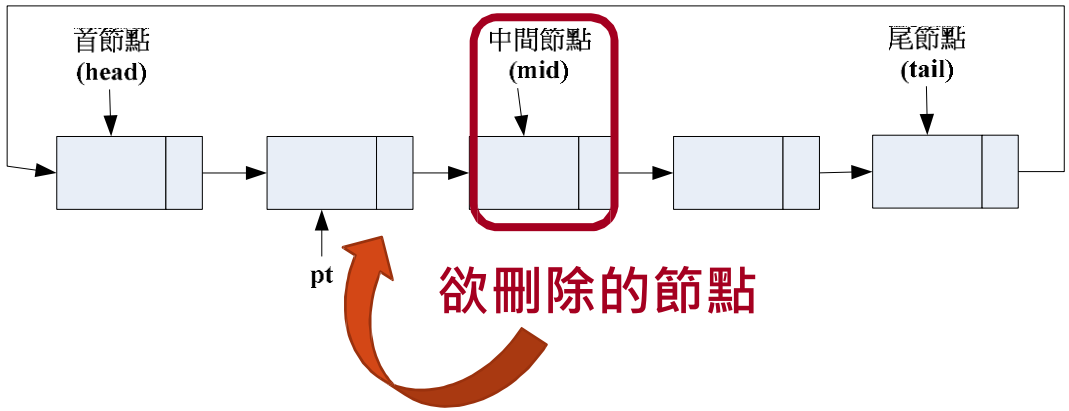
# 二、刪除環狀串列內的中間節點

只要將欲刪除節點的前一個節點的指標，指向欲刪除節點的下一個節點即可。

原始環狀串列：

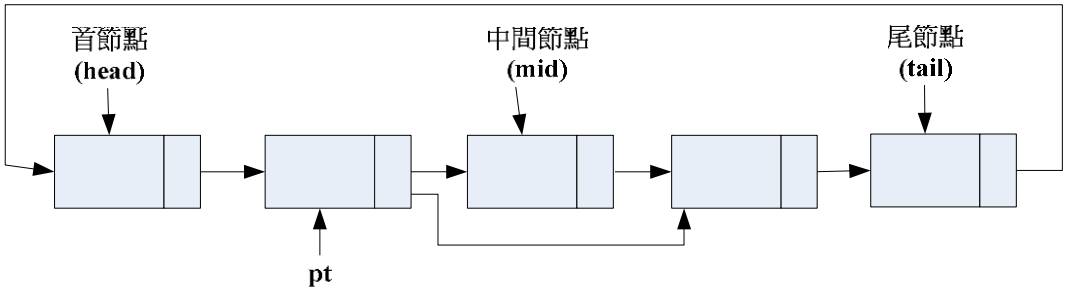



步驟1：找出欲刪除節點的前一個節點的指標(pt)：



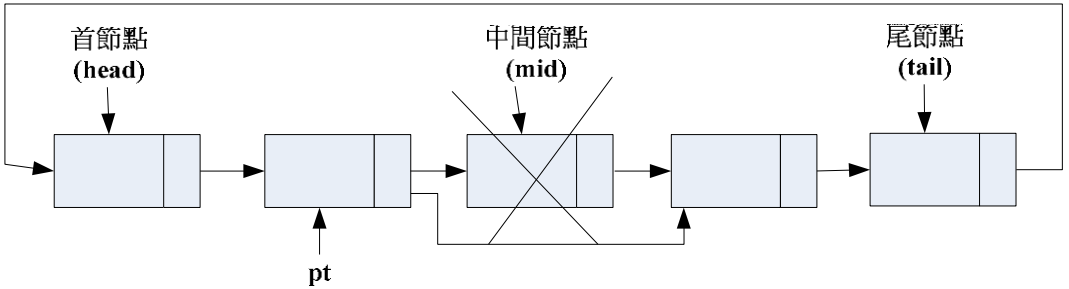
步驟2：將前一個節點(pt)的指標指向節點mid的下一個節點

pt->next = mid->next;




步驟3：將mid節點釋放出記憶體空間

free(mid);



# 5-8 雙向鏈結串列

【定義】

1. 每一個節點具有三個欄位，中間為資料欄位。

2. 左右各有一個指標欄位，分別為Llink及Rlink。

節點結構，如下圖所示：


其中Llink指向上一個節點，而Rlink指向下一個節點。

# 5-8 雙向鏈結串列(續)

雙向鏈結串列的優缺點如下說明：

**【優點】**

1. 雙向鏈結串列有**兩個指標節點**，在處理**加入或刪除**節點動作時，**速度比較快**。

2. 若雙向鏈結串列有任一端的指標連結**脫落時**，**則可以快速進行修補脫落的節點**。

**【缺點】**

1. 由於雙向鏈結串列有**兩個指標節點**，所以**比較浪費記憶體空間**。
2. 雙向鏈結串列的**加入或刪除**時，必須要有**較多的連結節點**。

# 5-8 雙向鏈結串列(續)

建立雙向鏈結串列的方法，首先就是宣告每個節點有三個欄位。

由於雙向鏈結串列的加入與刪除動作與單向鏈結串列相同，因此，不在此多加介紹。

但雙向鏈結串列也有許多優缺點，在實務的應用上，可依照情況來選擇。

# 5-9 多項式串列表示法

多項式的鏈結串列表示法主要是儲存非零項目，並且每一項的資料結構如下所示：


其中：COEF：多項式係數。

EXP：多項式指數。

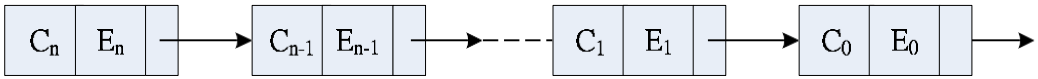
LINK：指標，指向下一個節點。

# 【串列節點宣告】

```
typedef struct poly_node
{
   int          coef;                  //宣告多項式係數
   int          exp;                   //宣告多項式指數
   struct  poly_node    *link;         //宣告指向下一個節點的指標
} POLY_NODE;
```

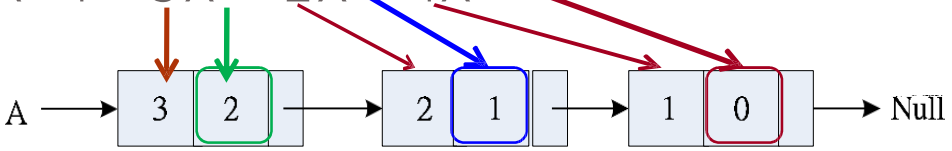
# 5-9.1 多項式串列的資料結構

假設有n個非零項，則可以表示如下圖所示：




【例如】

$A(X)=3X^2+2X+1=3X^2+2X^1+1X^0$




雖然在第三章已經介紹利用陣列來處理多項式，但是鏈結串列在處理多項式具有以下兩項優點：

1. 當多項式的內容變動(加入或刪除)時，則比較容易處理。
2. 鏈結串列可以動態的配置記憶體，因此，比較有彈性。

# 5-9.2 多項式的相加

了解**多項式串列**的**資料結構**之後，接下來，**我們來實際完成多項式的相加**。

【作法】

**逐一比較兩個多項式的項次**，**當指數相同時**，則可以**直接相加**，而當**指數較大**者，則可以**直接照抄下來**，直到兩個多項式比較完畢為止。

# 【實例】

假設有兩個多項式A與B，其表示如下所示：

$A(X) = 3X^2 + 2X + 1$

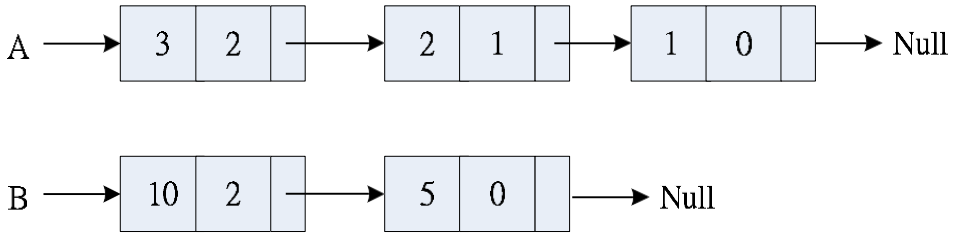
$B(X) = 10X^2 + 5$

請利用鏈結串列來進行兩個多項式的相加。

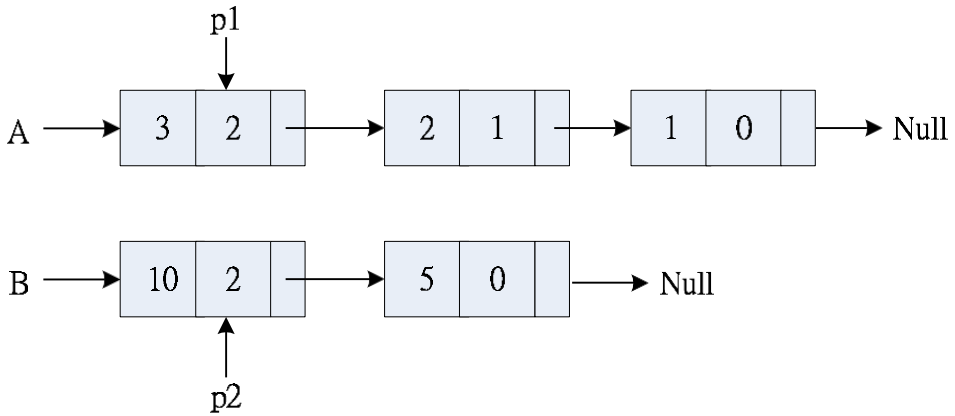
【解答】前置工作

$A(X) = 3X^2 + 2X + 1 = 3X^2 + 2X^1 + 1X^0$

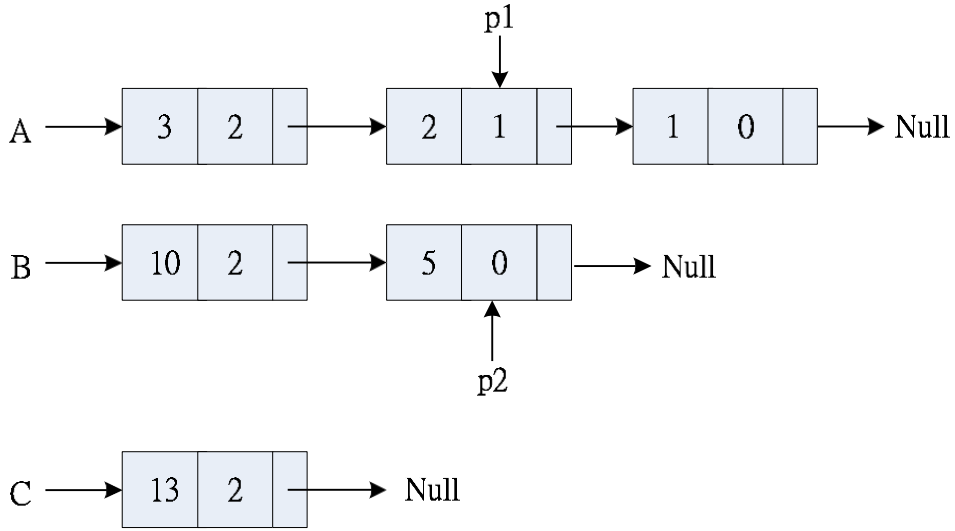
$B(X) = 10X^2 + 5 = 10X^2 + 0X^1 + 5X^0$



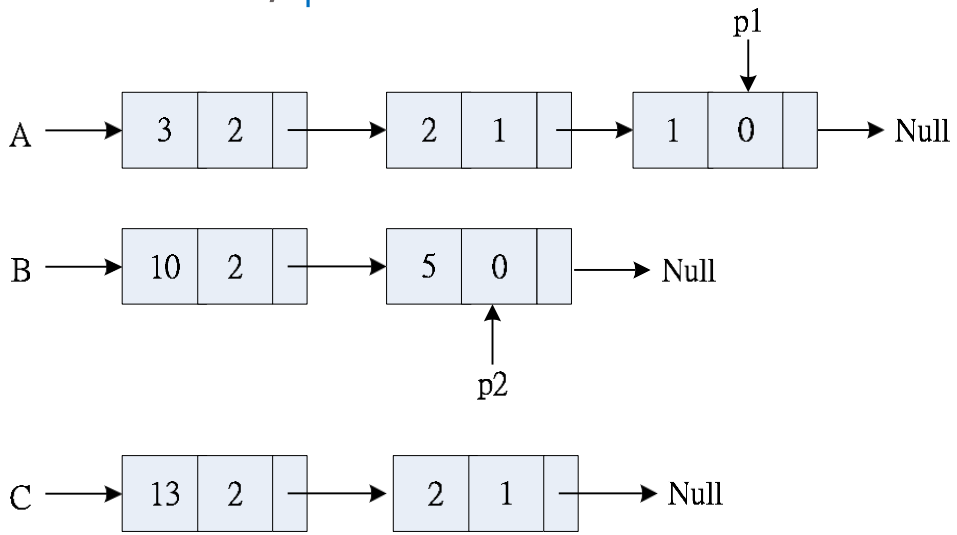
**步驟一**：將兩個多項式，分別以鏈結串列表示，並且p1與p2分別指向二串列的首節點。




**步驟二**：因為上個步驟之(Exp(p1)=Exp(p2))，因為指數相同，因此，係數可以相加，並且將相加後的結果，放到C串列中，並且p1與p2繼續往下一個指數比較。



**步驟三**：因為上個步驟之Exp(p1)>Exp(p2))，因此**複製**p1指數之項次至C串列中，p1**往下一項次前進**。




**步驟四**：因為上個步驟之Exp(p1)=Exp(p2))，因此**相加係數**放至C串列中。