

# Data Structures: Linked Lists

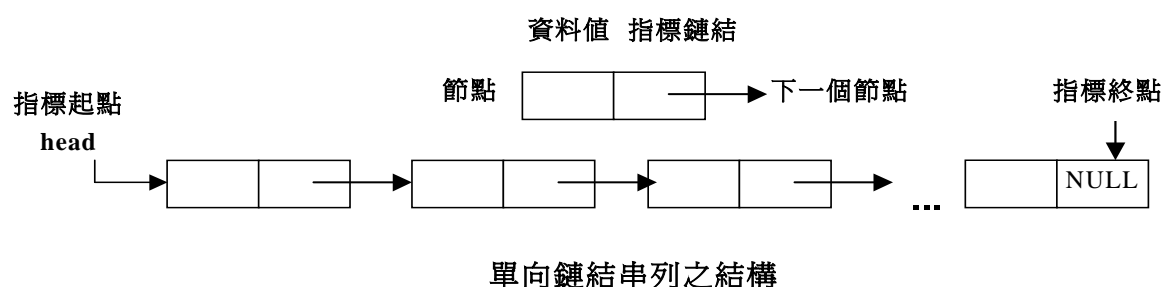
## 一. 何謂鏈結串列(Linked List)?

- 有次序排列之資料稱為串列(List)，如一年四季，數字 0~9。
- 鏈結串列各元素在記憶體之位置是不連續、隨機(Random)的。它是由動態記憶體分配節點(Node)串接而成。(相形之下，陣列為一個循序(Sequential)之記憶體結構)。

陣列製作串列		鏈結製作串列	
優點	缺點	優點	缺點
1. 易製作，宣告即可 2. 亦存取資料，利用所以對應	1. 刪除、插入及易動資料會造成資料移動頻繁，減少系統效率 2. 宣告記憶體空間、造成不必要之浪費	補足陣列串列之缺點	缺乏陣列串列之優點

## 二. 單向鏈結串列之資料型態

- 單向鏈結串列之結構如下圖所示

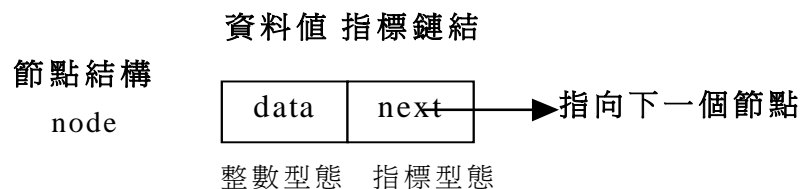


其中 head：指向串列前端之指標，tail：指向串列尾端之指標

- 鏈結串列透過儲存元素在記憶體之位址為指標(Pointer)或鏈結(Link)取得下一個節點。故

節點 = 資料 + 指標鏈結

- 定義節點結構： 假設有一節點結構如下圖所示



節點結構

則其節點結構可定義如下：

```
typedef struct node {          /* 以結構體表示節點*/  
  
    int    data;              /* data 儲存節點資料項目*/  
  
    struct node *next; } NODE; /* next 儲存下一個節點位址 */  
  
/* NODE 表新定義之節點結構資料型態 */
```

一個指標變數 head 當作鏈結串列之起始指標，其宣告如下

```
NODE *head; /* head 為一個指標，指向鏈結串列之起始節點*/
```

### 三. 單向鏈結串列之基本運算

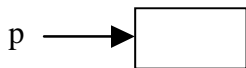
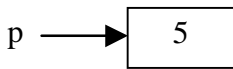
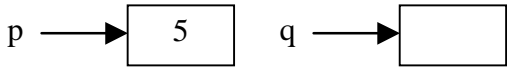
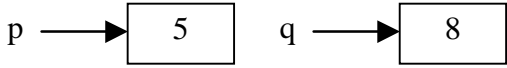

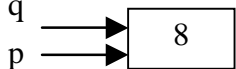
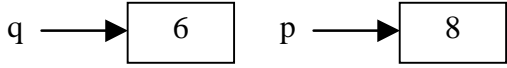
#### (1.) 產生新節點

```

NODE *getnode (void) /* 此函數產生一個新節點 */
{
    NODE *p;
    p = (NODE *) malloc(sizeof(NODE));
    /* malloc 會動態地配置大小為 sizeof 的記憶體*/
    /* sizeof 會傳回一個型態為 NODE 之值*/
    if ( p == NULL)
    {
        printf ("記憶體不足");
        exit(1);
    }
    return(p);
}

```

#### 描述下列指令之之意義

1. <code>p= (int *) malloc (sizeof(int));</code>	
2. <code>*p = 5;</code>	
3. <code>q= (int *) malloc (sizeof(int));</code>	
4. <code>*q = 8;</code>	
5. <code>free(p)</code>	
6. <code>p=q</code>	
7. <code>q= (int *) malloc (sizeof(int));</code> 8. <code>*q=6;</code>	

## (2.) 歸還一個節點

---

```
void *freenode (NODE *p)    /* 此函數將節點還給記憶體 */
{
    free(p);
}
```

---

## (3.) 尋找一個節點

---

```
NODE search_node (NODE *p, int num )
/* 自節點 p 之後尋找一個節點其 data 項目為 search_data 之值*/
{
    NODE *q;

    q = p->next;          /* q 指向節點 p 之後第一個節點 */
    while( q!= NULL && q->data != num)
        q = q->next;      /* q 改指向下一個節點 */
    return(q);
}
```

---

## (4.) 鍵結串列的節點走訪

---

```
NODE find_node(NODE *head, int num)
{
    NODE *ptr;

    ptr = head;           /* 指向串列起始 */
    while ( ptr != NULL ) /* 走訪串列 */
    { if ( ptr->num == num ) /* 找尋 data */
        return (ptr);
        ptr = ptr->next;    } /* 指向下一節點 */
    return (ptr);
}
```

---

### (5.) 計算鏈結串列之長度

---

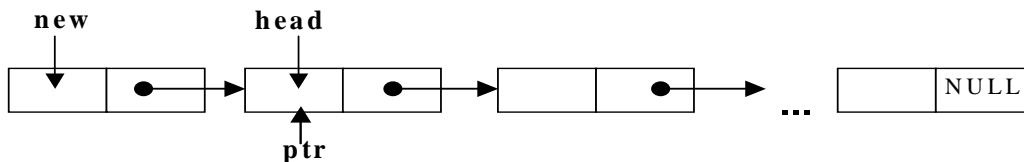
```
int length (NODE *p) /* 此函數計算節點 p 之後之長度 */
{
    int num=0;
    NODE *q = p->next;
    While (q != NULL) {
        num ++;
        q = q->next;    }
    return(num);
}
```

---

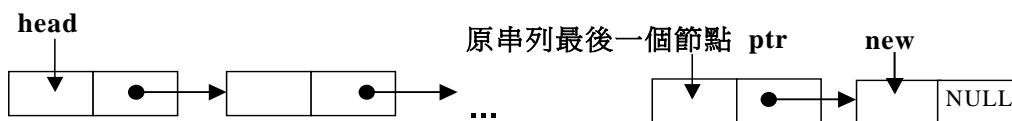
### (6.) 由鏈結串列加入一個節點

一個節點之插入有三種情況：

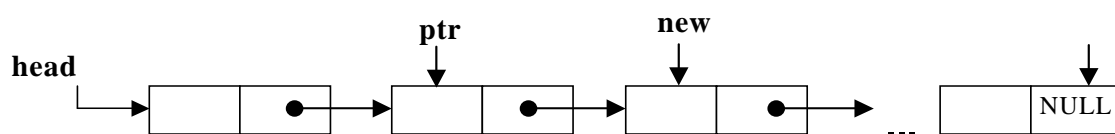
1. 節點加於第一個節點之前

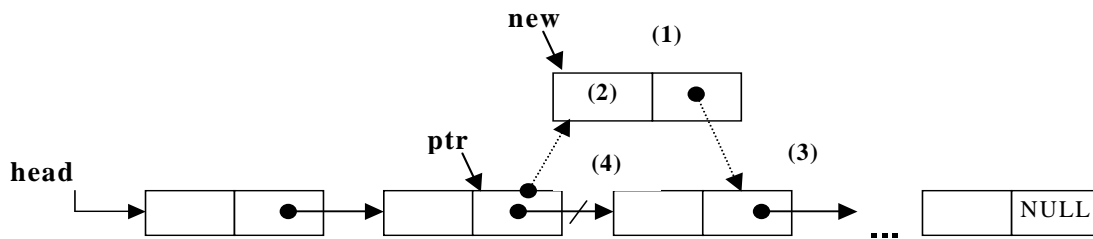


2. 節點加於最後一個節點之後



3. 加於節點中間任何一個位置





/\* 鏈結串列的節點插入\*/

---

**NODE \*insert\_node ( NODE \*head, NODE \*ptr, int value)**

```

{   NODE *new;                                /* 新節點指標變數    */

    new = getnode();                          /* (1) 建立新節點, 取得一個可用節點 */
    new->num = value;                          /* (2) 建立節點內容    */
    new->next = NULL;                         /* 設定指標初值      */

    if ( ptr == NULL )                       /* 指標 ptr 是否是 NULL */
    {
        /* 第一種情況: 插入第一個節點 */
        new->next = head;                    /* 新節點成為串列開始 */
        head = new;
    }
    else
    {
        if ( ptr->next == NULL )             /* 是否是串列結束    */
            /* 第二種情況: 插入最後一個節點 */
            ptr->next = new;                 /* 最後指向新節點    */
        else
        {
            /* 第三種情況: 插入成為中間節點 */
            new->next = ptr->next;           /* (3) 新節點指向下一節點 (3)*/
            ptr->next = new;                 /* 節點 ptr 指向新節點 (4)*/
        }
    }
    return (head);
}

```

---

## (7.) 由鏈結串列中刪除一個節點

一個節點之刪除有三種情況：

### 1. 刪除第一個節點

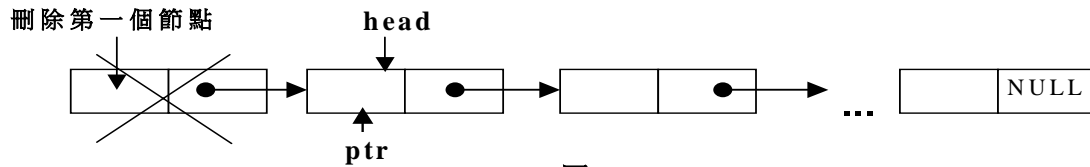
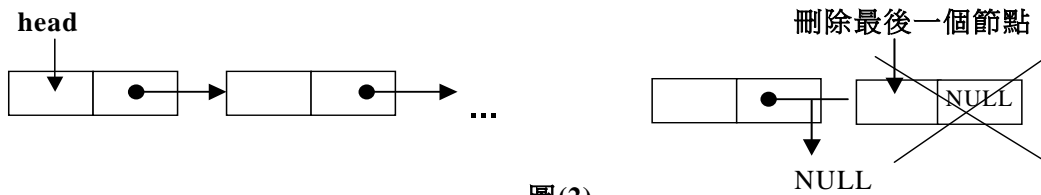


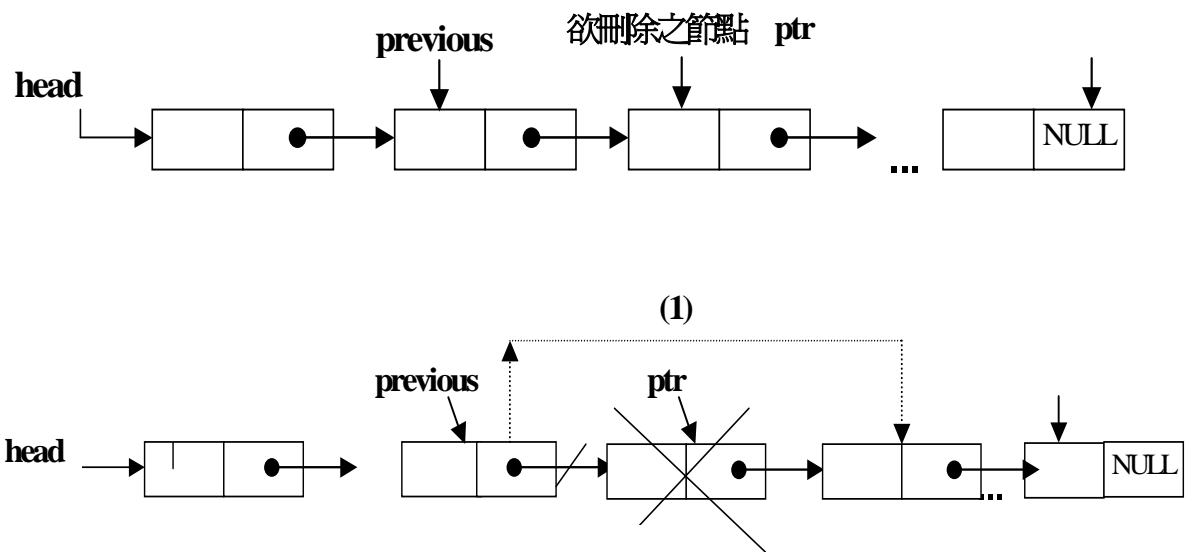
圖 (1)

### 2. 刪除最後一個節點



圖(2)

### 3. 加於節點中間任何一個位置



圖(3)

**/\* 鍵結串列的節點刪除 \*/**

---

```
NODE *delete_node(NODE *head, NODE *ptr)
{

    NODE *previous;                                /* 指向前一節點 */

    if ( ptr == head )                                /* 是否是串列開始 */

        /* 第一種情況：刪除第一個節點 */
        {
            head = head->next;
            return(head);                                /* reset 起始節點指標 */
        }
    else
    {
        previous = head;
        while ( previous->next != ptr ) /* 找節點 ptr 的前節點 */
            previous = previous->next;

        if ( ptr->next == NULL )    /* 是否是串列結束 */

            /* 第二種情況：刪除最後一個節點 */

            previous->next = NULL;    /* 最後一個節點 */
        else
            /* 第三種情況：刪除中間節點 */

            previous->next = ptr->next;    /* 圖(3)之步驟(1) */
        }

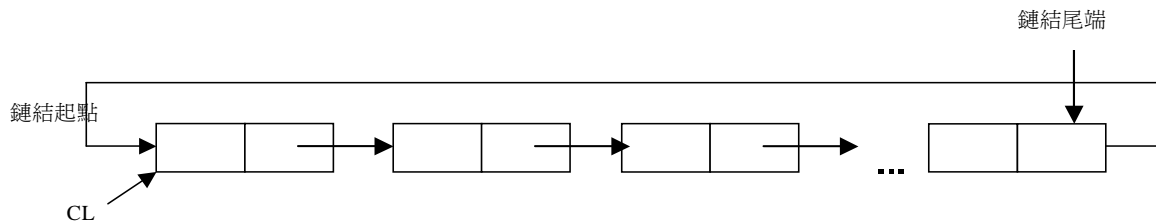
        freenode(ptr);                                /* 此函數將節點歸還給記憶體 */
        return(head);
    }
```

---



## (8.) 環狀鏈結串列之基本運算

一個鏈結串列之最後一個節點指向鏈結串列之最前端，則形成一個環狀鏈結串列



環狀鏈結串列之結構

### 計算環狀鏈結串列之長度

---

```
int  Clength (NODE *CL)  /* 此函數計算環狀鏈結串列之長度 */
{
    int num=0;
    NODE  *p;
    if (p != CL)  {
        p=CL;
        do {
            num ++;
            p = p->next;
        } while (p != CL);
    }
    return(num);
}
```

---

【習題一】 試將堆疊講義中之堆疊宣告，empty 函式， push 函式， pop()  
函式 部分改寫為以 C 語言之鏈結串列結構來製作堆疊。

【習題二】 試將佇列講義中之佇列宣告，empty 函式， enqueue 函式，  
dequeue 函式 部分改寫為以 C 語言之鏈結串列結構來製作  
一般佇列。

【習題三】 試寫一個副程式可以連結兩個鏈結串列 X、Y，形成 Z 鏈結  
串列

【習題四】 試寫一個副程式可以反轉鏈結串列

【習題五】 程式題

試使用串列結構表示下列之多項式，並計算此二個多項式之  
相加與相乘之結果

1.  $f(x) = x^4 + 2x^3 + x + 1$

2.  $g(x) = x^3 - 2x^2 - 6$