CH4、Linked List

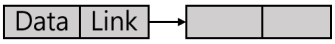
鏈結串列

目錄：

Linked List(鏈結串列)

Def：為節點(Node)所構成之有限集合，其中 Node 包含：

1. Data Field：Save Data
2. Link Field：Pointer to next Node

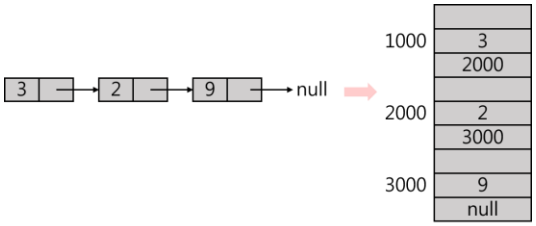


例 1：3, 2, 9

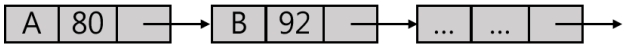
In Array：

3	2	9

In Linked List：



例 2：

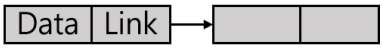


Note：Data, Link Field 視需求可有多個欄位

比較表

	Linked List	Array
Linked List 優點	不連續配置=>有彈性、易擴充	連續=>彈性差
	Node 的 Data Type 可不同	各元素 Data Type 需相同
	Insert, Delet 容易	Insert, Delet 不容易
	串列合併、共享容易	串列合併、共享不容易
Array 優點	Pointer 佔空間	無 Pointer 不佔空間
	僅能 Sequential Access	Random Access 與 Sequential Access 皆可支持

Node Def：

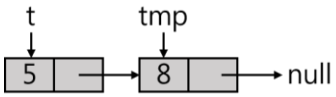


In C++(考試常用)	In C

class Node
{
public:
int data;
Node *link;
}
	struct Node
	{
	int data;
	Node *link;
	}

操作&配置方法

例：

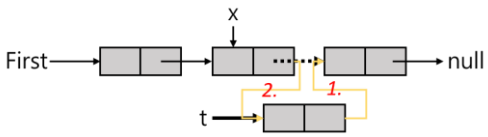



```
Node *t=new Node();
Node *tmp=new Node();
t->data = 5
t->link = tmp;
tmp->link = null;
```

Linked List 的實作

1. Insert：

   例：將一 Node(t)，插入在 Node x 之後，問演算法？



程式 1：

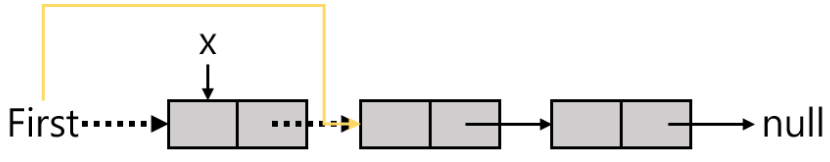
```
void insert(Node *x, Node *t)
{
        t->link = x->link;          //1.
        x->link = t;                //2.
}
```

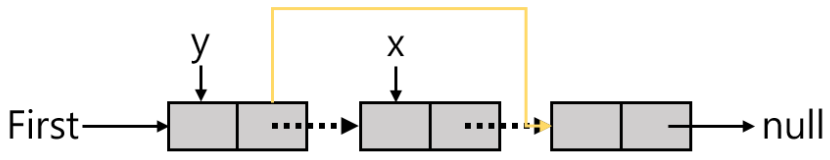
程式 2：

```
void insert(Node *x, Node *t)
{
        Node *t = new Node();
        t->data=item;
        t->link = x->link;
        x->link = t;
}
```

2. Delete：

   例：在 First 串列中，Delete x 此一 Node(令 y 為 x 的前一 Node)



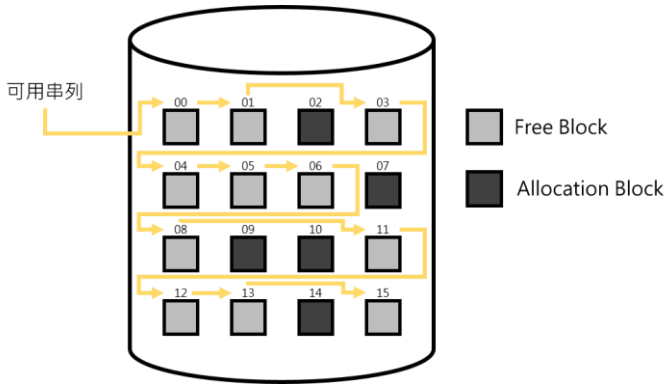


程式：

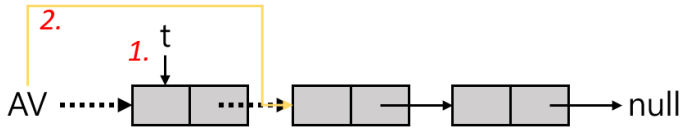
```
void delete(Node *x, Node y)
{
        if(y==null)
                First=x->link;
        else
                y->list=x->link;
        delete x;               //free(x);
}
```

系統可用 Space 的管理

OS 會用一條 AV List 的串列，將可用的 Node 連接(Storage Pool)



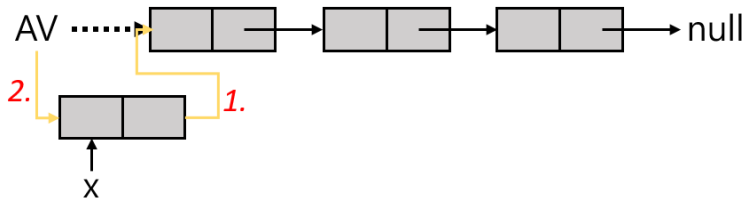
1. new(t)：要求一 Node(t)



程式：

```
Node *new(Node *AV)
{
        if(AV==null)
                "No Free Node";
        else
        {
                Node *t=AV;       //1.
                AV=AV->link;      //2.
                return t;
        }
}
```

2. ret(t)：歸還一 Node(t)



程式：

```
void ret(Node *AV, Node *t)
{
        t->link=AV;         //1.
        AV=t;               //2.
}
```

利用 Linked List 製作 Stack

[資結]

```
class Node
{
        publi:
        int data;
        Node *link;
};
Node *top=null;
```

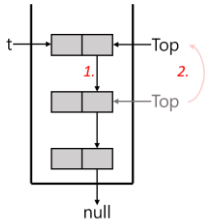
程式：

1. create

程式：

```
void create()
{
        top=null;
}
```

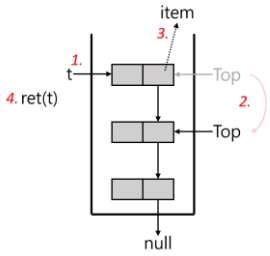
2. push



程式：

```
void push(Node *t)
{
        t->link=top;        //1.
        top=t;              //2.
}
```

3. pop



程式：

```
bool pop(int &item)
{
        if(top==null)
                return false;//Empty
        else
        {
                Node *t=top;        //1.
                top=top->link;      //2.
                item=t->data;       //3.
                ret(t);             //4.
                return true;
        }
}
```

利用 Linked List 製作 Queue

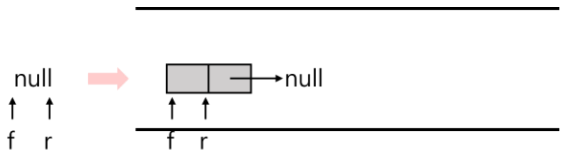
[法一]Single Linked List 製作 Queue

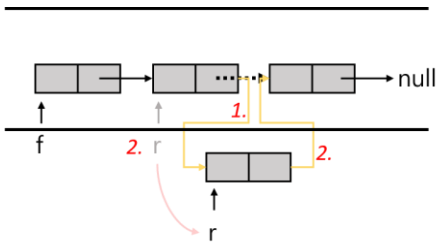
[資結]

```
class Node
{
        publi:
        int data;
        Node *link;
};
Node *front, *rear;
```

程式：

1. add

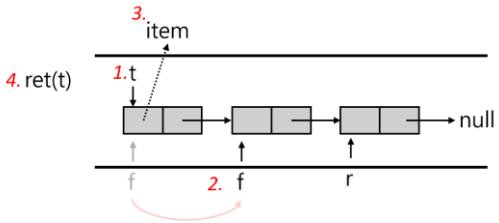




程式：

```
void add(Node *t)
{
        if(rear==null)
                front=rear=t;
        else
        {
                rear->link=t;       //1.
                rear=t;             //2.
        }
}
```

2. delete

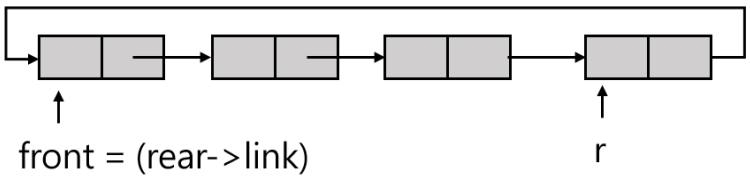


程式：

```
bool delete(int &item)
{
        if(front==null)
                return false;//Empty
        else
        {
                Node *t=front;          //1.
                front=front->link;      //2.
                item=t->data;           //3.
                ret(t);                 //4.
                if(front==null)         //有寫會比較詳細
                        rear=null;
                return true;
        }
}
```

[法二]Circular Linked List 製作 Queue

概念：



優點：只有 Rear 一個指標、不需 Front

[資結]

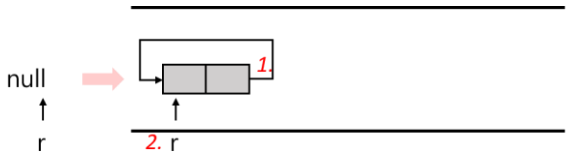
```
class Node
{
        publi:
        int data;
        Node *link;
};
Node *rear;          //初始值為 null
```

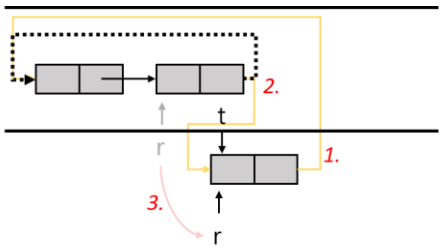
程式：

1. create

```
void create()
{
        rear=null;
}
```

2. add

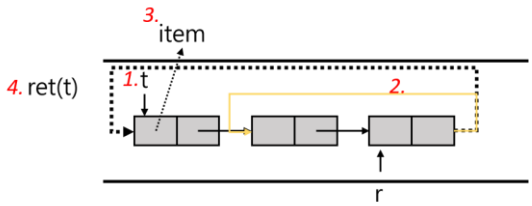




程式：

```
void add(Node *t)
{
        if(rear==null)
        {
                t->link=t;          //1.
                rear=t;             //2.
        }
        else
        {
                t->link=rear->link;     //1.
                rear->link=t;           //2.
                rear=t;                 //3.
        }
}
```

3. delete



程式：

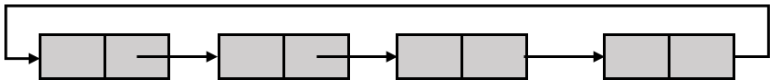
```
bool delete(int &item)
{
        if(rear==null)
                return false;//Empty
        else
        {
                Node *t=rear->link;                 //1.rear->link 即是 front
                rear->link=(rear->link)->link;      //2.
                item=t->data;                       //3.
                if(rear==t)
                        rear=null;
                ret(t);                             //4.
                return true;
        }
}
```

Linked List 種類

一、Single(單向)
二、Circular(環狀)
三、Double(雙向)

二、Circular Linked List

Def：指其最後一個 Node 會指向第一個 Node 謂之

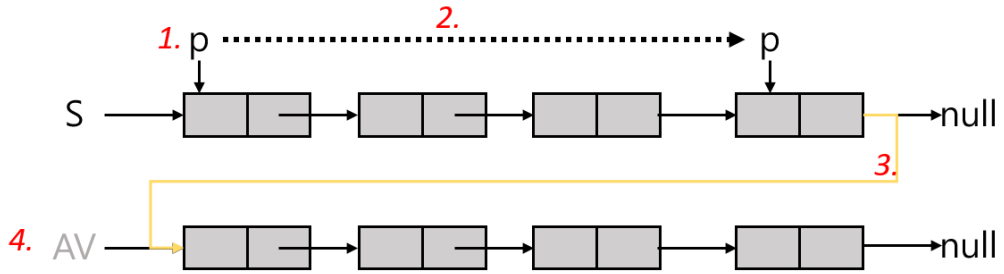


Note：Circular 和 Single 的差別

1. 任一 Node 皆可拜訪所有節點一次
2. 串列回收容易：O(1)

串列回收：

1. Single Linked List

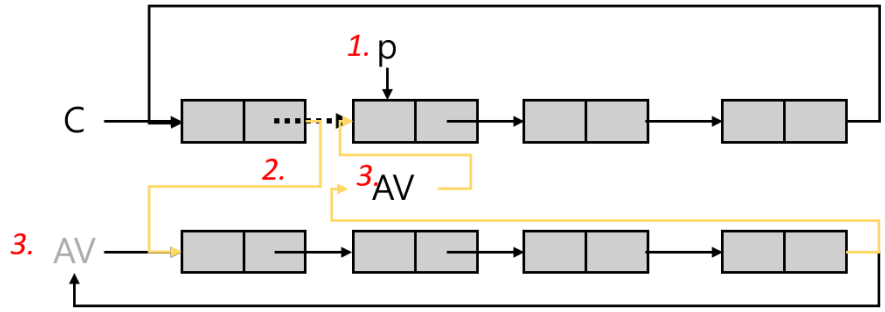


程式：

```
void ret(Node *s)
{
        Node *p=S;                              //1.
        while(p->link!=null)                    //2.
        {
                p=p->link;  //p 往下一個 Node
        }
        p->link=AV;                             //3.
        AV=S;                                   //4.
}
```

⇨ O(n)

2. Circular Linked List



程式：

```
void ret(Node *c)
{
        Node *p=c->link;        //1.
        c->link=AV;             //2.
        AV=p;                   //3.
}
```

⇨ O(1)

比較表

Double Linked List	Circular Linked List
任一 Node 可知上、下一個 Node	無法得知上一個 Node
Delete 不需告之前一 Node	Delete 需告知前一 Node
可靠性佳	可靠性差(Link Broken 則 Data Lost)
Insert 需更動 4 個指標	Insert 需更動 2 個指標
Delete 需更動 2 個指標	Delete 需更動 1 個指標
任一 Node 可拜訪所有 Node	必需從頭拜訪起
較耗空間	較省空間

三、Double Linked List

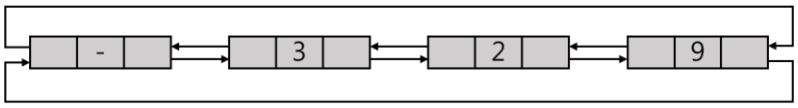
Node Structure：

Llink	Data	Rlink

[資結]

```
class Node
{
        public:
                Node *Llink;
                int data;
                Node *Rlink;
};
```

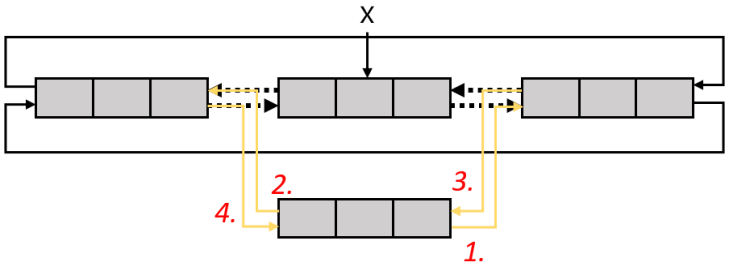
例：3, 2, 9



head(串列首)

⇨ 可多加一 Node(head)，形成 2 個 Circular Linked List

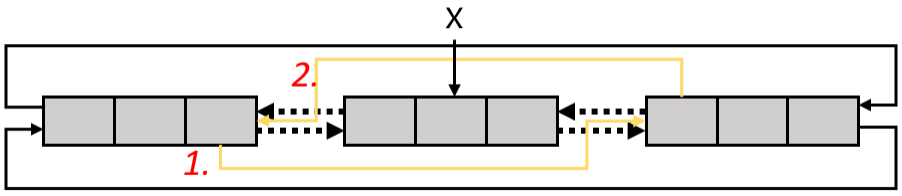
1. insert：insert a Node(t)到 x 之後



程式：

```
void insert(Node *x, Node *t)
{
        t->Rlink=x->Rlink;      //1.
        t->Llink=x;             //2.
        (x->Rlink)->Llink=t;    //3.
        x->Rlink=t;             //4.
}
```

2. delete：delete a Node(x)



程式：

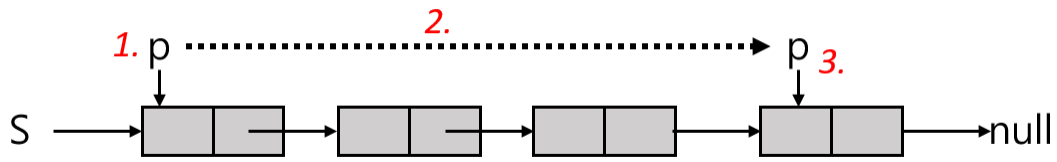
```
void delete(Node *x)
{
        (x->Llink)->Rlink=(x->Rlink);       //1.
        (x->Rlink)->Llink=(x->Llink);       //2.
}
```

Linked List 應用

一、串列回收
二、長度計算
三、[補充]Single→模擬 Double
四、串列合併
五、反轉串列

二、Linked List 長度計算

1. Sinlge Linked List

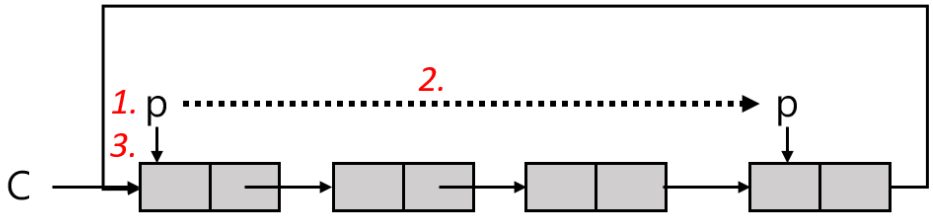


程式：

```
int SLength(Node *s)
{
        int count=0;
        if(s!=null)
        {
                Node *p=s;          //1.
                while(p!=null)      //2.
                {
                        count++;
                        p=p->link'
                }
        }
        return count;               //3.
}
```

⇨ O(n)

2. Circular Linked List



程式：

```
int CLength(Node *c)
{
        int count=0;
        if(c!=null)
        {
                Node *p=c;          //1.
                do                  //2.
                {
                        count++;
                        p=p->link'
                }while(p!=c);
        }
        return count;               //3.
}
```

⇨ O(n)

三、Single Linked List 製作出 Double Linked List 之效果

提示：用”⊕”(Exclusive-Or, xor)

作法：Input：A⊕B =>相同為 0；相異為 1

Truth Table(真值表)

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

常見公式：A⊕0=A, A⊕A=0, A⊕1=A', A⊕A'=1

作法：

在各 Node 中多一個欄位記錄”L⊕R”之值

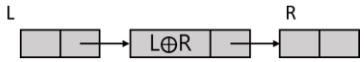
Node：

L 為此 Node 之前一 Node 位址

R 為此 Node 之後一 Node 位址

Null 之位址為 0

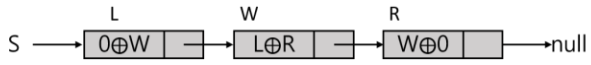
概念：



(L⊕R) ⊕R=L

(L⊕R) ⊕L=R

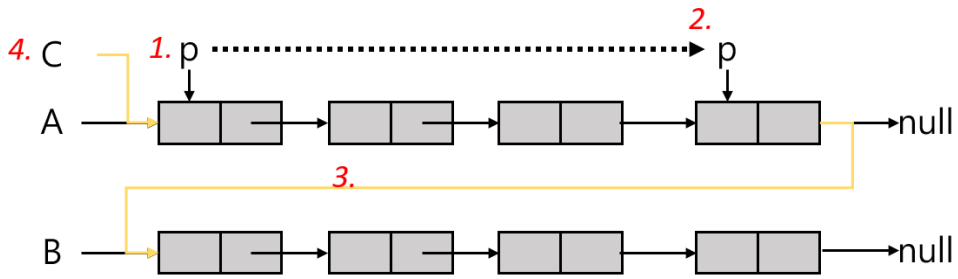
例：



R 的上一個：w⊕0⊕0 = w
W 的上一個：L⊕R⊕R=L

四、串列合併

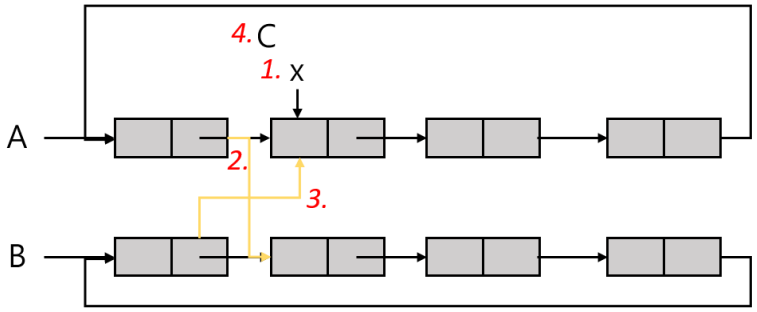
1. Single Linked List



程式：

```
void con(Node *A, Node *B, Node *C)
{
        if(A==null && B==null)
                C=null;
        else if(A!=null && B==null)
                C=A;
        else if(A==null && B!=null)
                C=B
        else
        {
                Node *p=A;              //1.
                while(p->link!=null)    //2.
                {
                        p=p->link;
                }
                p->link=B;              //3.
                C=A;                    //4.
        }
}
```

2. Circular Linked List



程式：

```
void con(Node *A, Node *B, Node *C)
{
        if(A==null && B==null)
                C=null;
        else if(A!=null && B==null)
                C=A;
        else if(A==null && B!=null)
                C=B
        else
        {
                Node *x=A->link;        //1.
                A->link=B->link;        //2.
                B->link=x;              //3.
                C=x;                    //4.
        }
}
```

⇨ O(1)

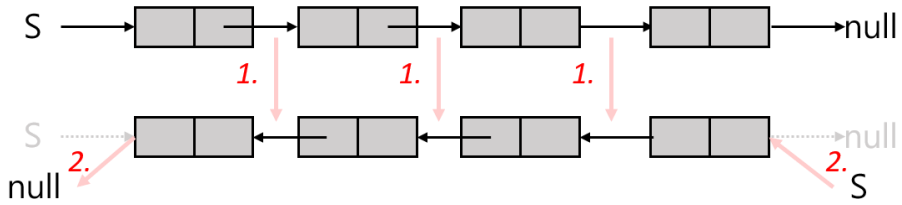
五、反轉串列

先看程式：

```
void invert(Node *s)
{
        Node *r, *q=null, *p=s;
        while(p!=null)
        {
                r=q;                    //1.
                q=p;                    //2.
                p=p->link;              //3.
                q->link=r;              //4.
        }
        s=q;
}
```

概念：

1. 將原始 Linked List 每個 Link 都反轉後，再把首尾交換



2. 至於將由誰來執行"Link 的反轉"：以 Recursive 方式，假定另外三者：r, q, p

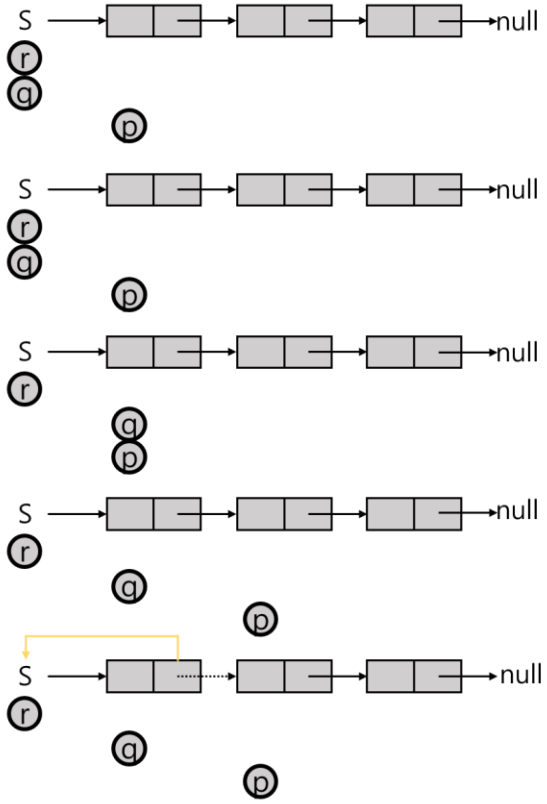
比喻：命運三女神(過去 r、現在 q、未來 p)

概念：由『未來』手拉手、帶領著『現在』與『過去』，走在鏈上(Link)、向前探索(三女神僅為旁觀執行者，並非鏈結 Link 本身的主事者)。過程中，由『現在』負責進行調整(反轉 Link)，一直到『未來』到達終點、不再有未來(null)為止

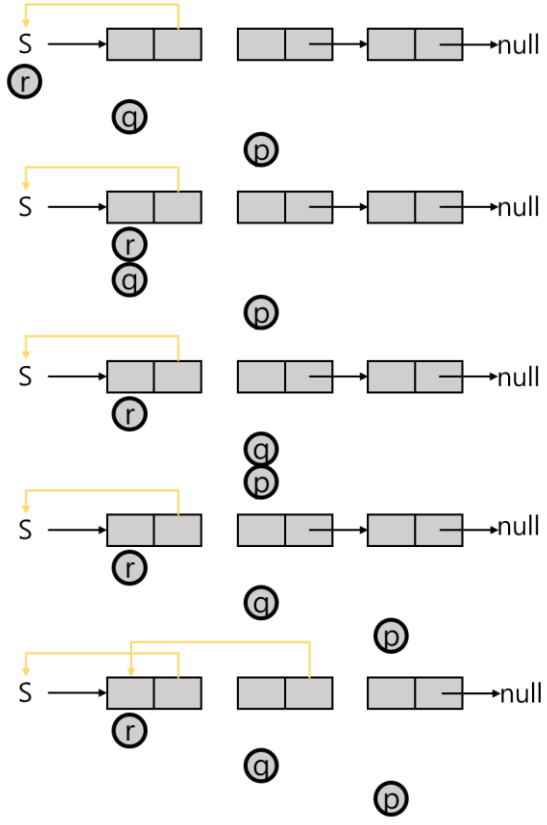
『現在』是唯一負責人(有實際做事)，而『過去』與『未來』基本上只是協助『現在』辯認，並且遞移往前(無實質做事)

注意：尤於需要使用遞迴，因此關鍵在於『初始化』

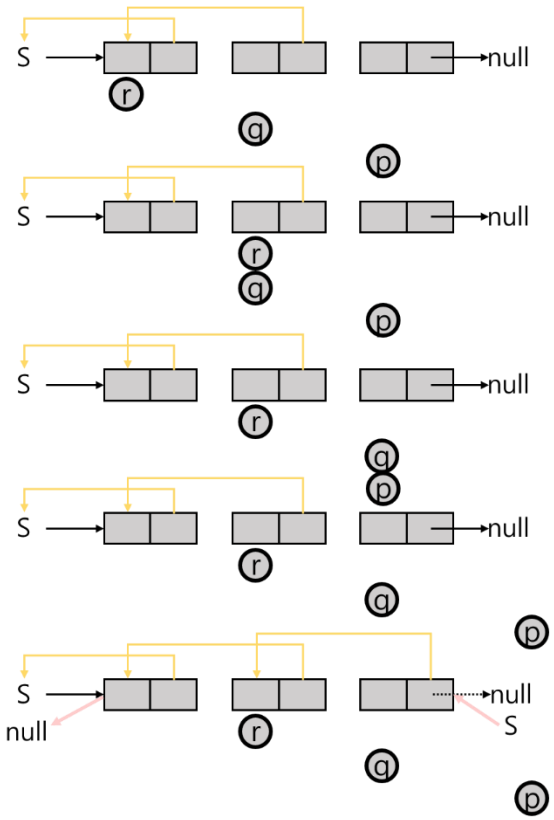
第 1 次迴圈：


S
null
r
q
p
S
null
r
q
p
S
null
r
q
p
S
null
r
q
p
S
null
r
q
p

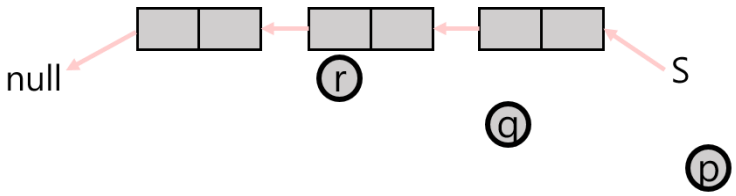
第 2 次迴圈：


S
null
r
q
p
S
null
r
q
p
S
null
r
q
p
S
null
r
q
p
S
null
r
q
p

第 3 次迴圈：


S
null
r
q
p
S
null
r
q
p
S
null
r
q
p
S
null
r
q
p
S
null
null
S
r
q
p

結果：


null
r
S
q
p

多項式表示法

一、利用 Array：CH2

二、利用 Linked List：

[法一]：用 Single/Circular/Double

[法二]：採 Generalization List

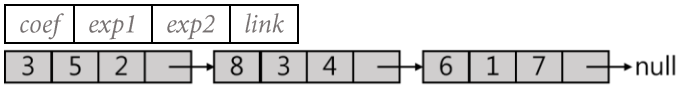
[法一]

例 1：$F(x)=3x^5+6x^3+8x+9$，以 Linked List 表示？

Node Structure：

例 2：$F(x)=3x^5y^2+8x^3y^4+6xy^7$，以 Linked List 表示？

Node Structure：



思考：是否有通用表示方式，可表達不同變數個數之多項式？ => [法二]

[法二]Generalize List(一般化/通用化串列)

Def：A 是 Generalize List，則：

$A=(a_1, a_2, \ldots, a_n)$，其中 a_i 是 A 的元素，$1 \le i \le n$

而 a_i 的型態可能為：

1. Atomic Data(原資料)
2. Sublist(子串列)，Sublist 亦為 Generalize List

相關術語：

1. a_1 是 Head
2. $\{a_2, a_3, \ldots, a_n\}$是 A 的 Tail
3. $|A|$是 A 的長度或元素個數

例：

1. A=(a, (b, c)) => |A| =2
2. B=(A, A, ()) => |B| = 3
3. C=(a, C) => |C| = 2

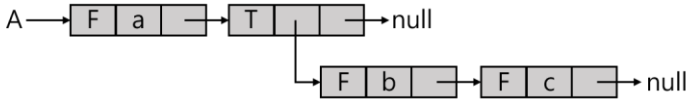
Node Structure：

Tag	變動欄位(由 Tage 決定)	Link(指向下一個 Node)

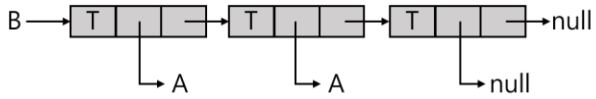
布林

1. False→Data：Save Data
2. True→Dlink：Pointer to a Sublist

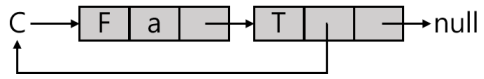
1. A=(a, b, c)



2. B=(A, A, ())



3. C=(a, C)



Self-Calling 的概念

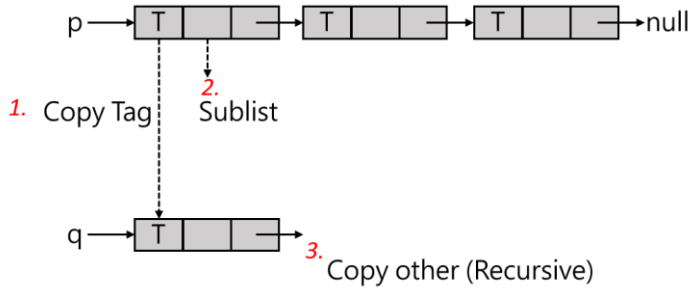
應用

一、Copy：複製一 Generalize List

二、Equal：檢查 2 條 Generalize List 是否相等

三、Depth：計算 Generalize List 的高度

一、Copy：將 p 串列複製一份給 q



程式：

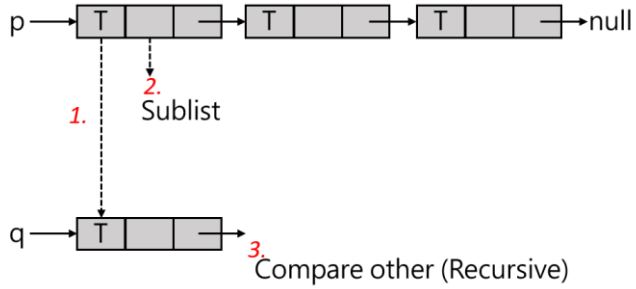
```
Node *copy(Node *p)
{
        Node *q=null;
        if(p!=null)
        {
                q=new Node();
                q->tag=p->tag;                  //1.
                if(p->tag==false)               //2.
                        q->data=p->data;
                else
                        q->dlink=copy(p->dlink);    //3.
                q->link=copy(p->link);          //3.
        }
        return q;
}
```

二、檢查 2 條 Generalize List 是否相等

說明：

1. if(s 空 and t 空) => True
2. else if(s 非空 and t 非空) => check
3. else => False

概念：



程式：

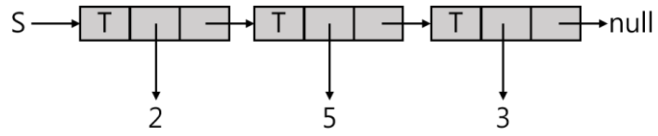
```
bool equal(Node *s, Node *t)
{
        bool x;                                         //存 step2 是否相等
        if((s==null) && (t==null))
                return true;
        else if((s!=null) && (t!=null))
        {
                if(s->tag==t->tag)                      //1.
                {
                        if(s->tag==false)               //2.第一個欄位 Tag
                        {
                                if(s->data==t->data)    //第二個欄位變動
                                        x=true;
                                else
                                        x=false;
                        }
                        else
                                x=equal=(s->dlink, t->dlink);   //3.
                        if(x==true)
                                return equal(s->link. t->link);  //3.第三個欄位 Dlink
                }
        }
        return false
}
```

三、Depth(s)

概念：

Depth(s)=

1. HMAX{depth(a_i)+1}，$a_i \in s$, a_i 是 Sublist。可或寫成：HMAX(a_1, a_2, …, a_n)+1
2. 0, if s=null



程式：

```
int depth(Node *s)
{
        Node *p=s;
        int m=0;                                    //記錄 HMAX{ai}
        if(s==null)
                return 0;
        else
        {
                while(p!=null)
                {
                        if(p->tag==true)            //成立代表有 Sublist
                        {
                                int n=depth(p->dlink);
                                if(m<n)
                                        m=n;
                        }
                }
                p=p->link;
        }
        return m+1;
}
```

Generalize List 之多項式表示法

Node Structure：

Trip	變動欄位	exp	link

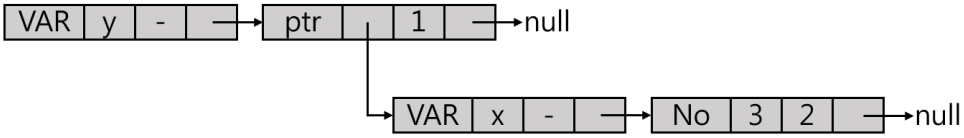
1. VAR => VAR 欄位(存變數名稱)
2. Ptr=>dlink 欄位，指向 Sublist
3. No=>coef 欄位(存係數值)

Exp：指數

Link：指向下一個 Node

例 1：$f(xy)=3x^2y$

$3x^2y = (3x^2)y$



例 2：$f(xyz)=4x^3y^2z^2+8x^2y^2z^2+9xyz^2$

$4x^3y^2z^2+8x^2y^2z^2+9xyz^2 = (4x^3y^2+8x^2y^2+9xy)z^2 = [(4x^3+8x^2)y^2+(9x)y]z^2$