

struct 2

先來複習一下

struct 是我們自己定義的型態
可以包含不同種類(型態)的東西(變數)

先來複習一下

- 宣告

```
struct my_type{  
    int x;  
    char y;  
};
```

- 使用

```
my_type t1, t2;  
t1.x = 5;  
std::cin>>t2.y;  
t2 = t1;  
my_type arr[10];
```

其實還有...

用法1

struct包struct

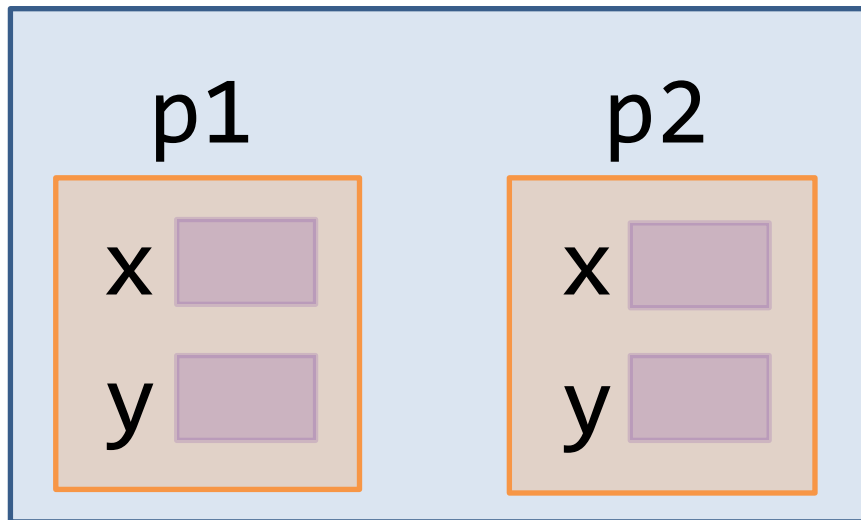
既然**struct**定義出來的是一種「型態」
那我把這個型態用在其他**struct**裡...
應該也可以吧?!

```
struct Point{  
    int x;  
    int y;  
};
```

```
struct Point{  
    int x;  
    int y;  
};
```

```
struct Rect{  
    Point p1;  
    Point p2;  
};
```

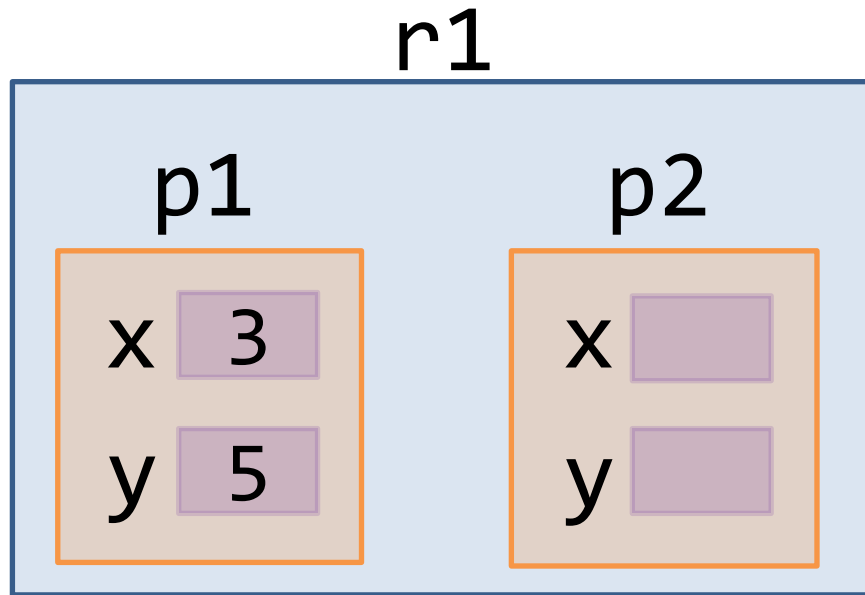
Rect




```
struct Point{  
    int x;  
    int y;  
};
```

```
struct Rect{  
    Point p1;  
    Point p2;  
};
```

```
Rect r1, r2;  
r1.p1.x = 3;  
r1.p1.y = 5;  
r2 = r1;
```



用法2

struct+指標

指標

- 宣告

資料型態 *指標變數名稱

ex: `int *ptr;`

- 使用

```
int x = 10;
```

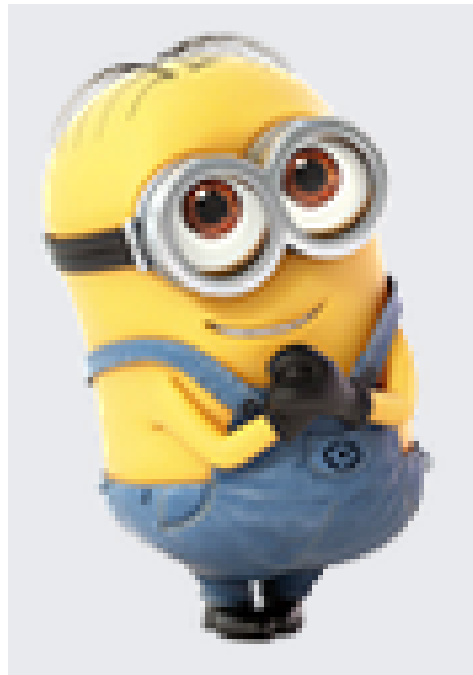
```
ptr = &x;
```

```
int *ptr2 = &x;
```

```
//*ptr = ??? *ptr2 = ???
```

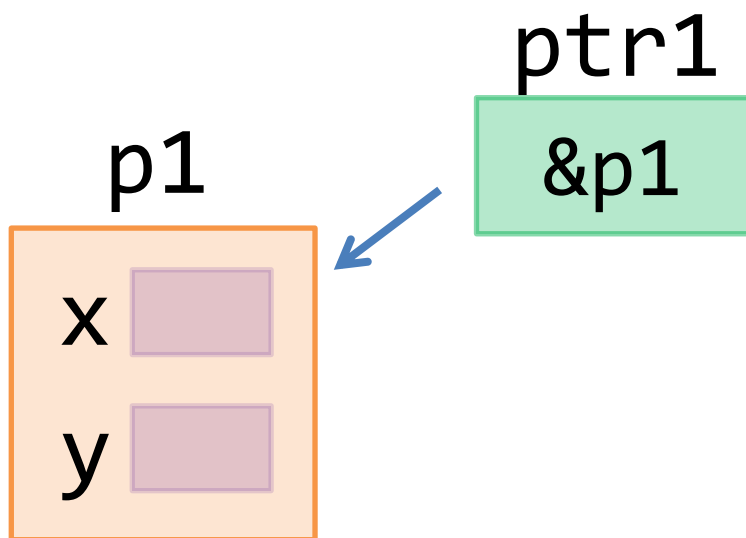
`int`是個type， 我的`struct`也是個type，
所以...

我的`struct`也可以有指標嗎??



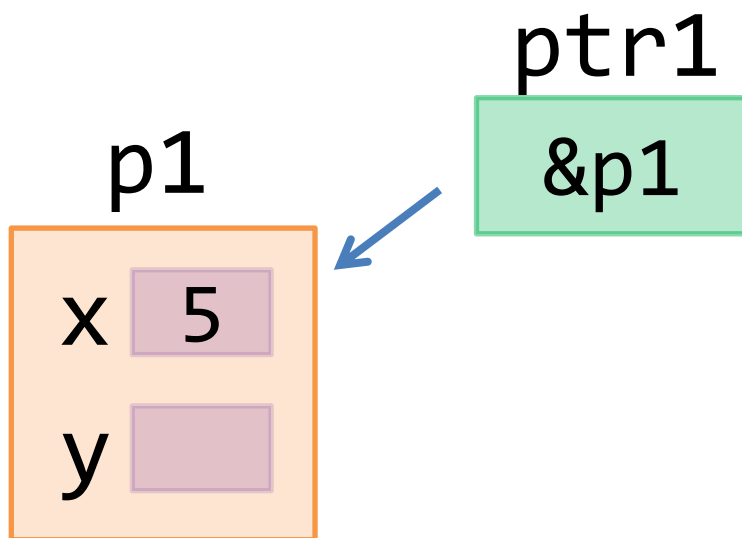
```
struct Point{  
    int x;  
    int y;  
};
```

```
Point p1;  
Point *ptr1 = &p1;  
// *ptr1和p1代表的一樣;
```

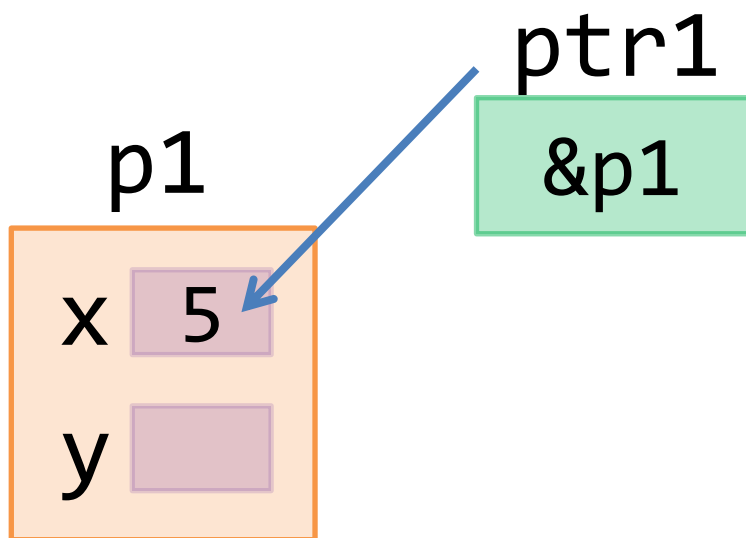


```
struct Point{  
    int x;  
    int y;  
};
```

```
Point p1;  
Point *ptr1 = &p1;  
// *ptr1和p1代表的一樣;  
p1.x = 5;  
(*ptr1).x = 5;
```



```
struct Point{  
    int x;  
    int y;  
};
```



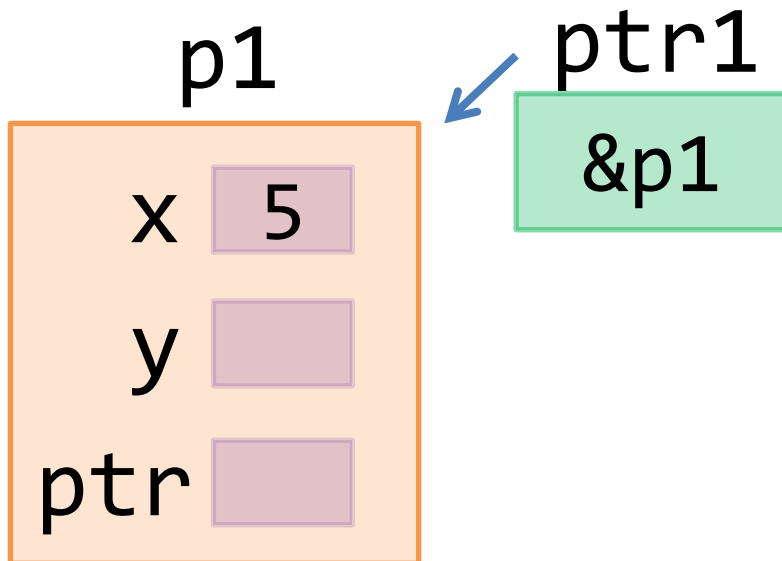
```
Point p1;  
Point *ptr1 = &p1;  
// *ptr1和p1代表的一樣;  
p1.x = 5;  
(*ptr1).x = 5;  
ptr1->x = 5;
```

用法3

struct包指標

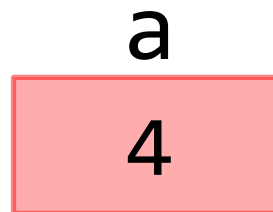
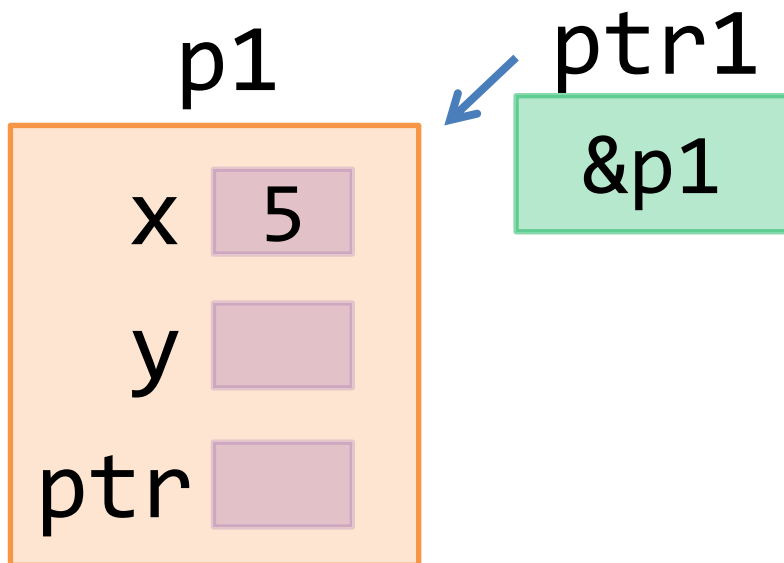

```
struct Point{  
    int x;  
    int y;  
    int *ptr;  
};
```

```
Point p1;  
Point *ptr1 = &p1;
```



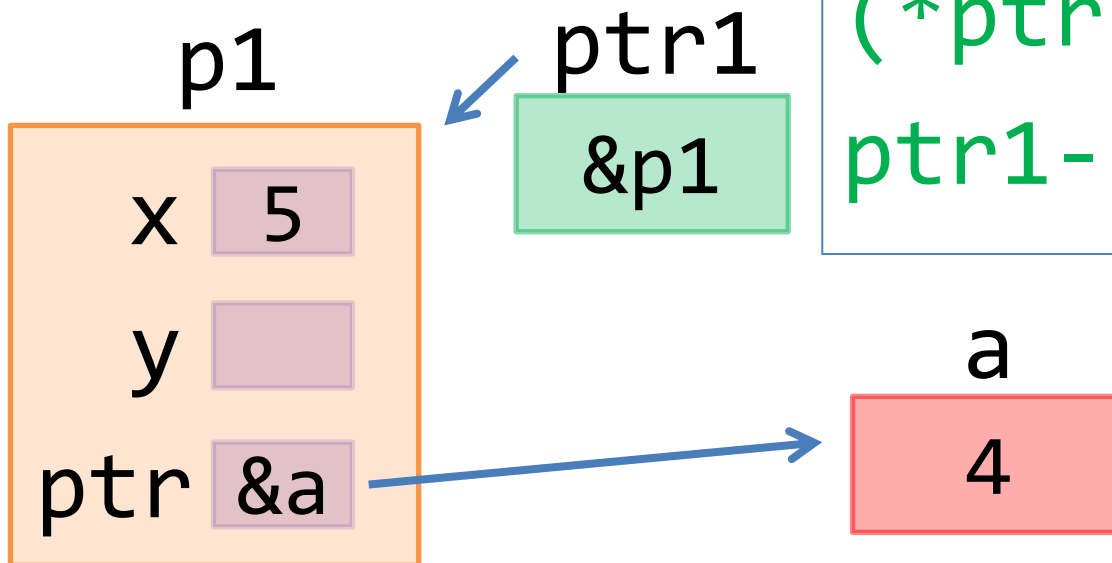
```
struct Point{  
    int x;  
    int y;  
    int *ptr;  
};
```

```
Point p1;  
Point *ptr1 = &p1;  
int a = 4;
```



```
struct Point{  
    int x;  
    int y;  
    int *ptr;  
};
```

```
Point p1;  
Point *ptr1 = &p1;  
int a = 4;  
p1.ptr = &a;  
(*ptr1).ptr = &a;  
ptr1->ptr = &a;
```



new/delete

```
int i = 5;
```

```
int x = 4;
```

i

5

當程式執行到變數的**scope**,

x

這些變數就會自動

被配置記憶體空間。

4

如果我想等要用的時候再給空間呢？

如果我用完了想要拿回空間呢？

（像旅館！）

如果我想等要用的時候再給空間呢？

ANS: **new**

如果我用完了想要拿回空間呢？

ANS: **delete**

(像旅館！)

new

先打打看...

```
std::cout<<new int<<std::endl;
```

先打打看...

```
std::cout<<new int<<std::endl;
```

我的結果：

0x5213f0

這不是記憶體位址嗎!!!

也就是說，
new會跟系統說"我要空間"，
系統就會給**new**空間，並告訴空間的位址，
這樣我們才能利用

為什麼用指標接**new**傳回的值??

- 要**利用**這個空間，
用*****就可以存取空間內的值，
並做運算

ex:

```
int i = new int; //wrong
```

```
int *ptr = new int;
```

寫法：

指標型態 *指標名稱 = new 空間型態；

```
int *ptr1 = new int;
```

既然可以給"變數"空間，
也可以...

1. 給"陣列"空間

2. 給"**struct**"空間

3. 給"**struct**的陣列"空間

1. 給"陣列"空間
2. 給"struct"空間
3. 給"struct的陣列"空間

```
1.int *ptr = new int[100];  
2.Point *ptr = new Point;  
3.Point *ptr = new Point[100];
```

1. 給"陣列"空間
2. 給"struct"空間
3. 給"struct的陣列"空間

```
1.int *ptr = new int[100];  
2.Point *ptr = new Point;  
3.Point *ptr = new Point[100];
```

P.S. 只要是跟陣列有關的, 都會回傳空間的**第一個**位址

delete

```
1.int *ptr = new int;
```

```
2.int *ptr = new int[100];
```

```
3.Point *ptr = new Point;
```

```
4.Point *ptr = new Point[100];
```

```
1.int *ptr = new int;  
   delete ptr;  
2.int *ptr = new int[100];  
   delete [] ptr;  
3.Point *ptr = new Point;  
   delete ptr;  
4.Point *ptr = new Point[100];  
   delete [] ptr;
```

```
1.int *ptr = new int;  
   delete ptr;  
2.int *ptr = new int[100];  
   delete [] ptr;  
3.Point *ptr = new Point;  
   delete ptr;  
4.Point *ptr = new Point[100];  
   delete [] ptr;
```

```
delete ptr;  
delete [] ptr;
```

linked list

先想想陣列...

- 宣告時必須知道陣列大小
- 臨時插入或刪除元素麻煩

linked list

就是為了解決這兩個主要的缺點
但, 如何解決?

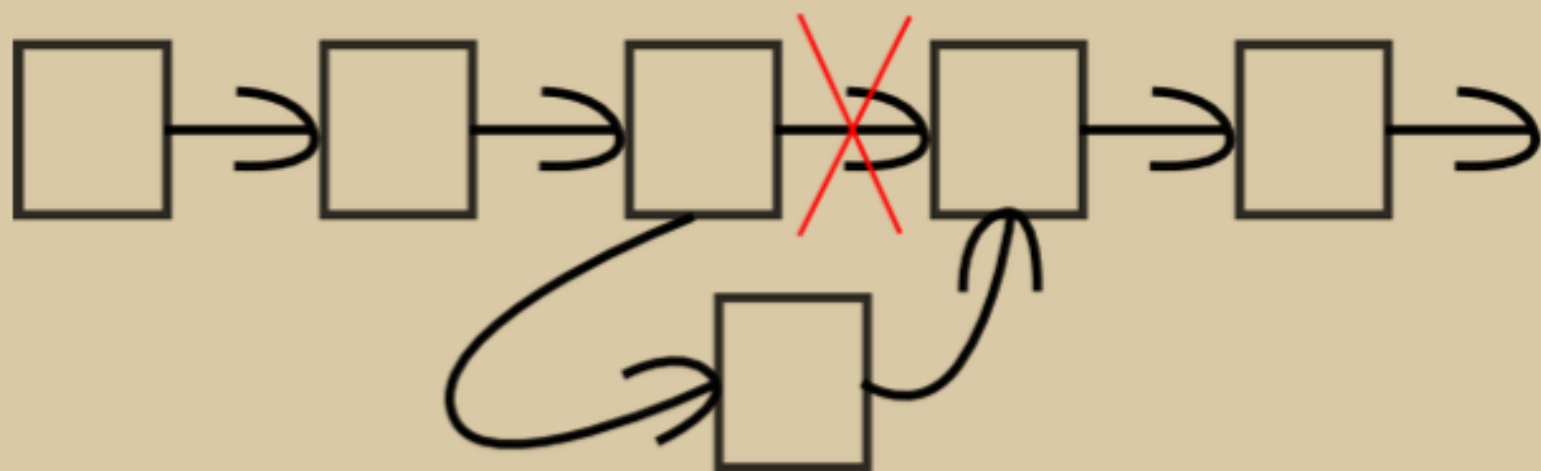
1.宣告時必須知道陣列大小
既然不知道陣列大小，
那就不要宣告陣列，
等每次要用的時候，
再宣告一格，然後接起來就好了

2. 臨時插入或刪除元素麻煩

之所以麻煩,是因為陣列都連在一起,想分都分不開

不如,就讓格與格之間可以分開,
再用某種可以斷開的連接,
把格與格接在一起就好了

插入



删除



1. 等每次要用的時候，
再宣告一格，然後接起來
2. 讓格與格之間可以分開，
再用某種可以斷開的連接，
把格與格接在一起就好了

所以，

每個格子裡應該要包含什麼？

最基本的：要記錄的資料
還有!!!

接著下一格的"連接"

"連接"到底是什麼東西？

陣列雖然缺點多多，
但"連接"卻很直觀，
就是**index**值

ex: `a[5]`

但**linked list**的每一格是分散的,根本沒有**index**值,所以我們只能

找 位 址 !!!

誰可以幫我們找位址??

你知道答案嗎?

所以，

我們已經知道每一格裡要有甚麼了

1. 基本資料

2. 連接(指標)

來打code吧 ya~

每一格裡要有資料跟指標，
你有學過哪種型態是可以這樣的嗎？

用**struct**定義一個

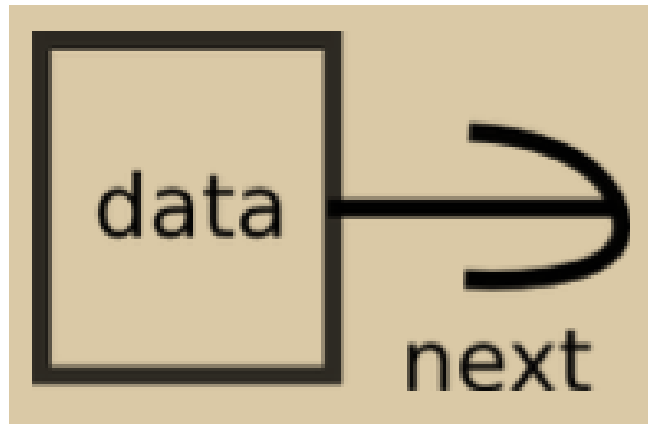
So,

```
struct node{  
    int data;  
    node *next;  
};
```

- 因為指標是要指向下一個node,
 所以next是node型態的指標

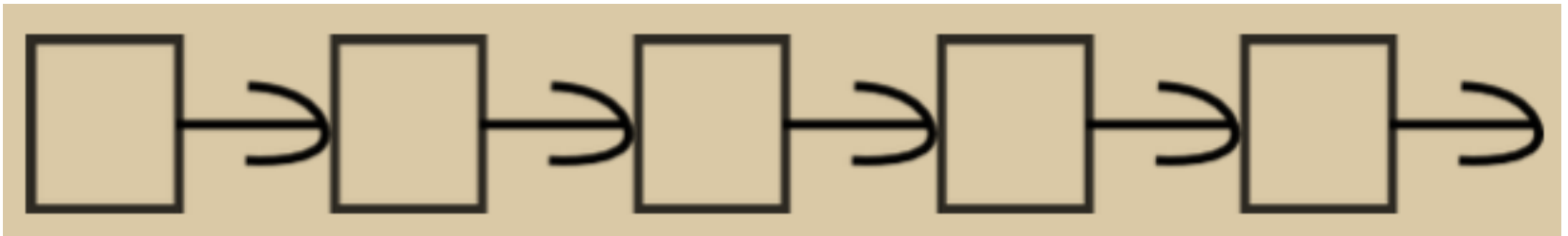
```
struct node{  
    int data;  
    node *next;  
};
```

我們就有了一個類似這樣的東西：

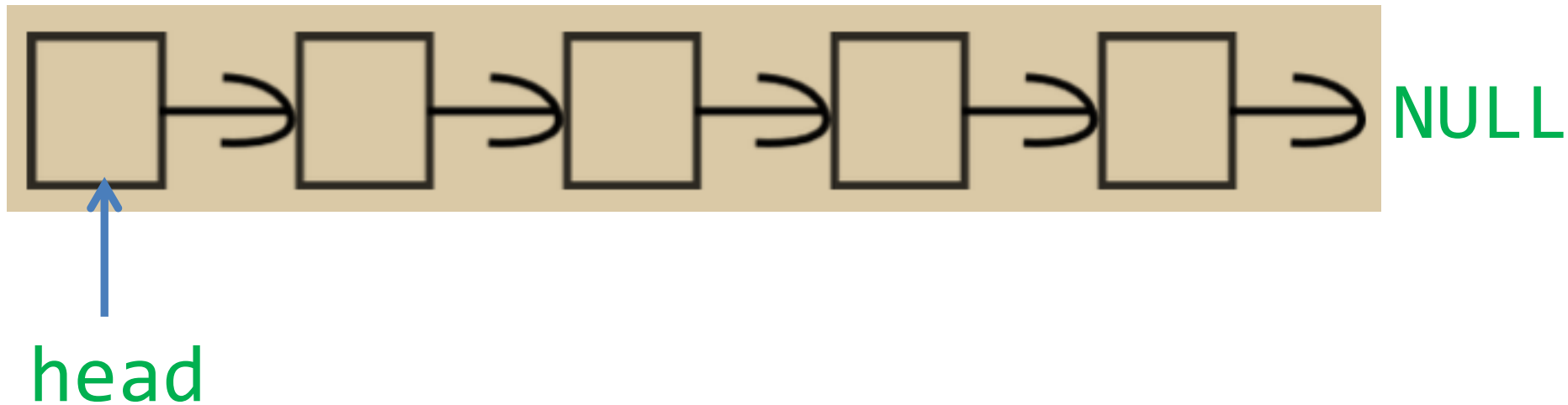


接著，
只要一直宣告node，並接起來，
就有一條linked list囉~

但是,這條**linked list**
從哪裡開始,又到哪裡結束?



所以，給個開頭(head指標)，
給個結尾(NULL)



從head開始，沿著next一直往下走，直到next==NULL為止

把建立一條完整**linked list**寫成函式

```
node *head = NULL;
```

```
node* makeNode(int d){
```

```
    //make a new node
```

```
}
```

```
void insert_front(int d){
```

```
    //insert at front
```

```
}
```

```
void remove_back(){
```

```
    //remove from back
```

```
}
```

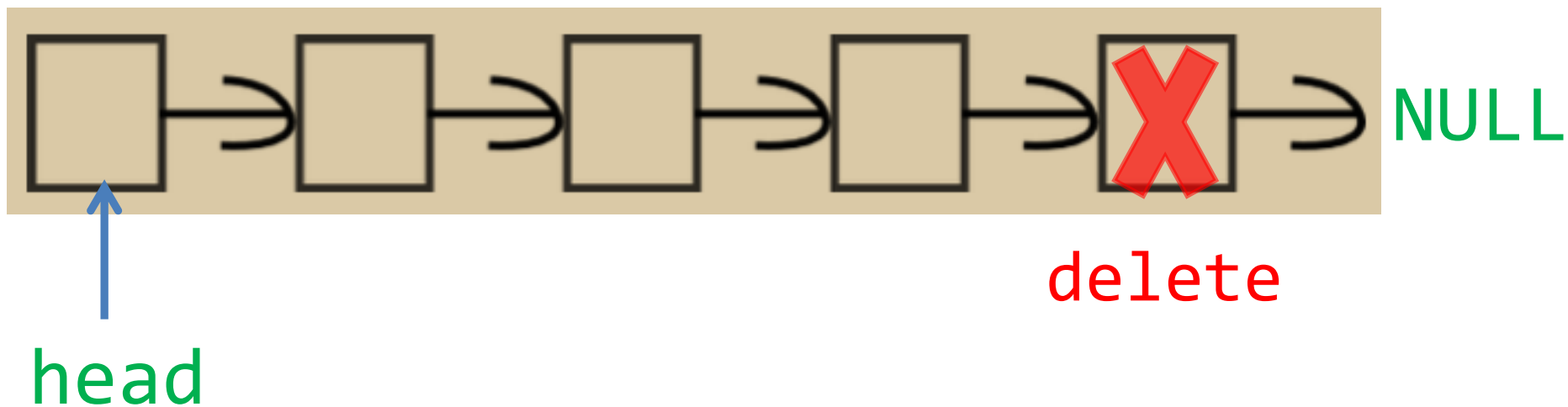
```
node* makeNode(int d){  
    node *newNode = new node;  
    newNode->data = d;  
    newNode->next = NULL;  
    return newNode;  
}
```

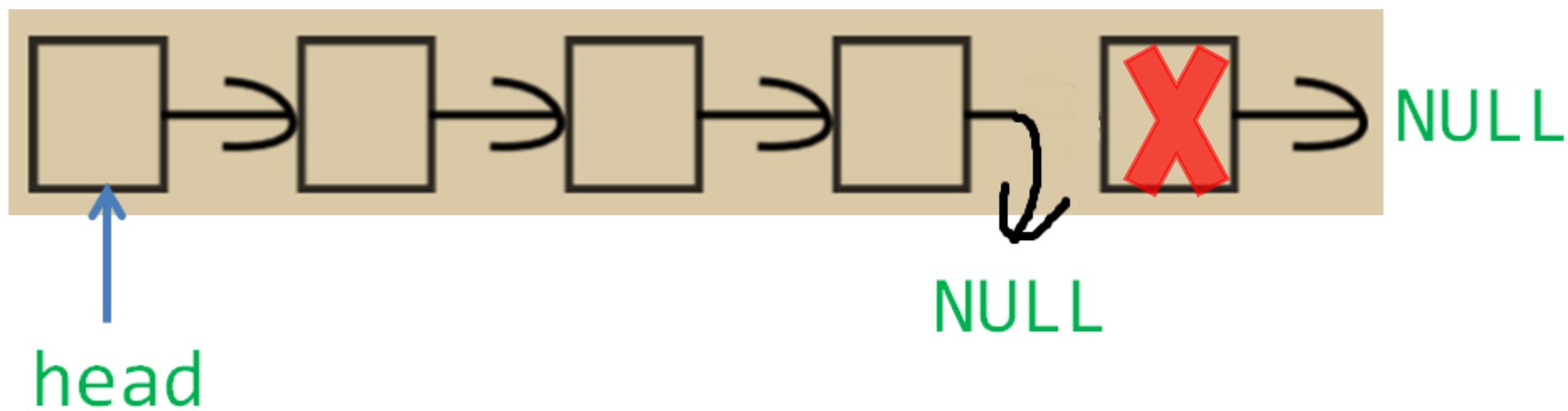
```
void insertion_front(int d){  
    node *temp = makeNode(d);  
    if (head == NULL){  
        head = temp;  
    } else {  
        temp->next = head;  
        head = temp;  
    }  
}
```

remove back

remove是甚麼意思??

- 把空間收回來,用delete!
- 收回來之後,別忘了,被移掉的前面那格要指向NULL






```
void remove_back(){
    node *cur = head;
    if (head == NULL)
        return;
    if (head->next == NULL){
        delete head;
        head = NULL;
        return;
    }
    while(cur->next != NULL){
        if(cur->next->next == NULL){
            delete cur->next;
            cur->next = NULL;
        }
        cur = cur->next;
    }
}
```