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...

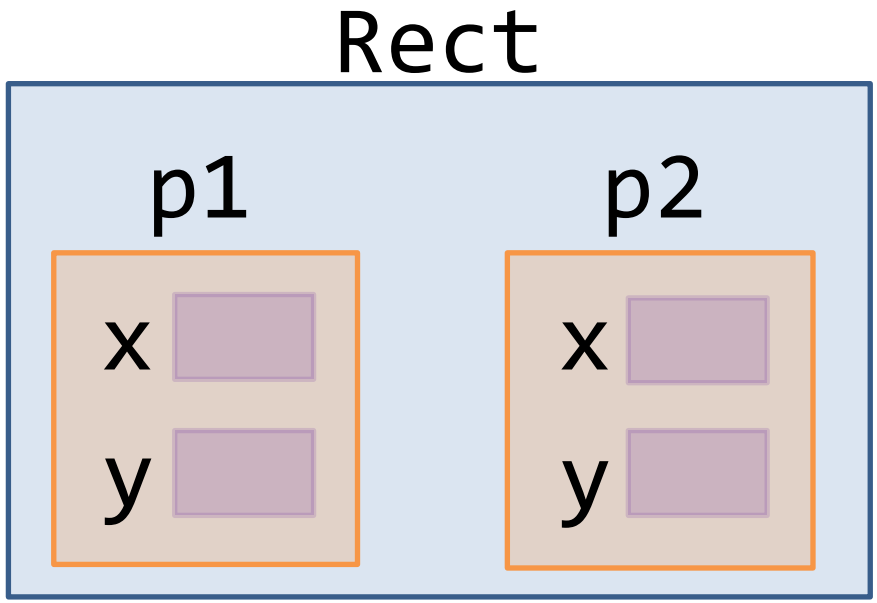
用法1
struct包struct

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;
};
```

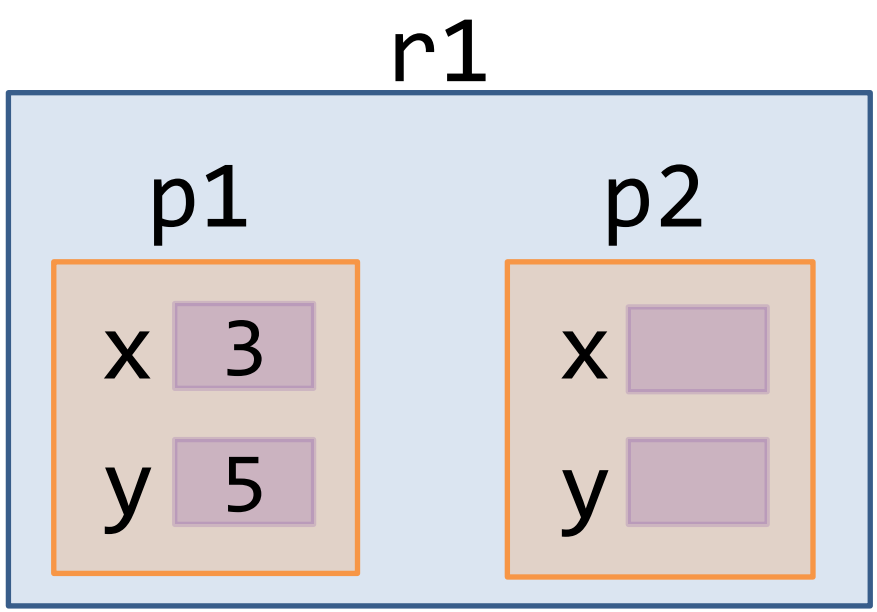
```
struct Rect{
    Point p1;
    Point p2;
};
```



```
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

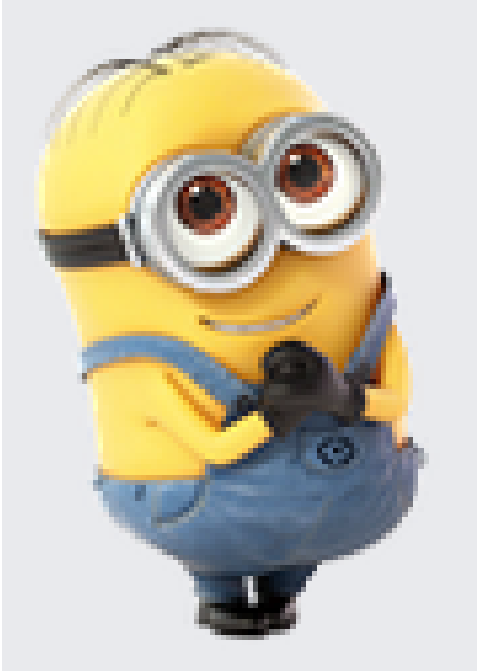
  ex: int *ptr;

- 使用

```
int x = 10;
ptr = &x;
int *ptr2 = &x;
//*ptr = ??? *ptr2 = ???
```

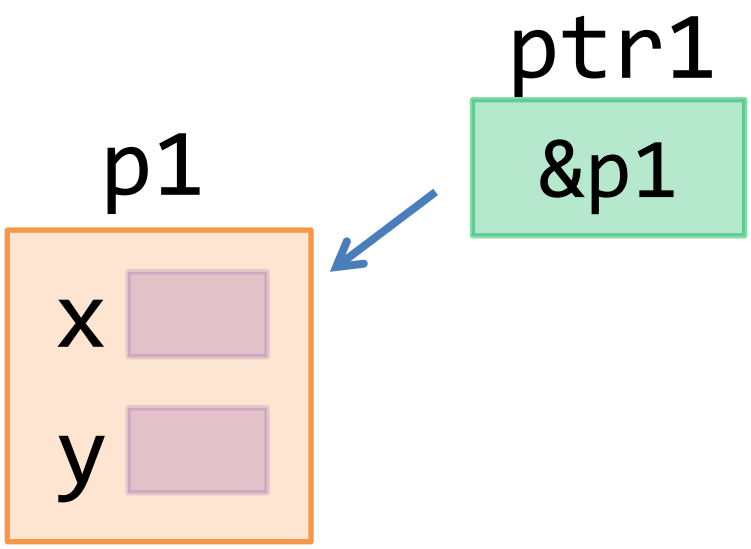
int是個type, 我的struct也是個type,
所以...

我的struct也可以有指標嗎??

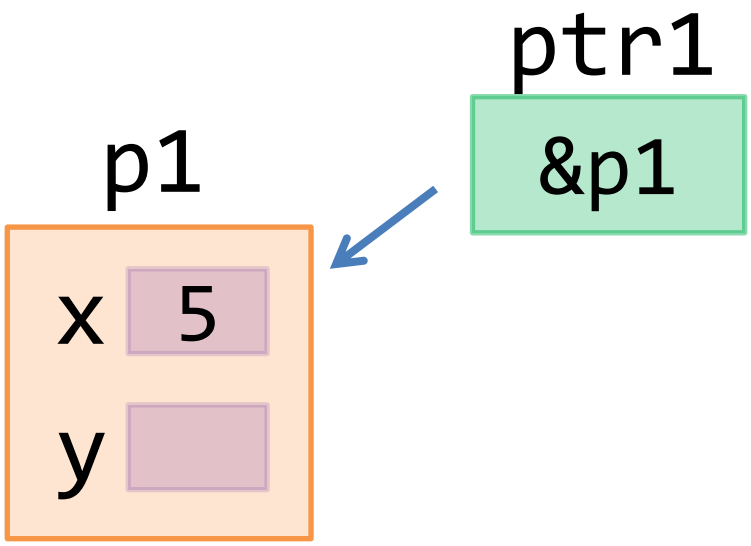


```
struct Point{
    int x;
    int y;
};
```

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

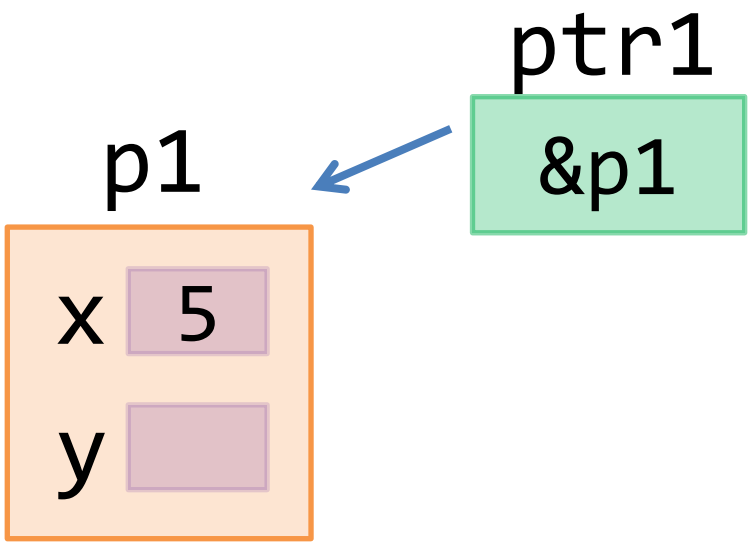


```
struct Point{
    int x;
    int y;
};
```




```
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;
};
```

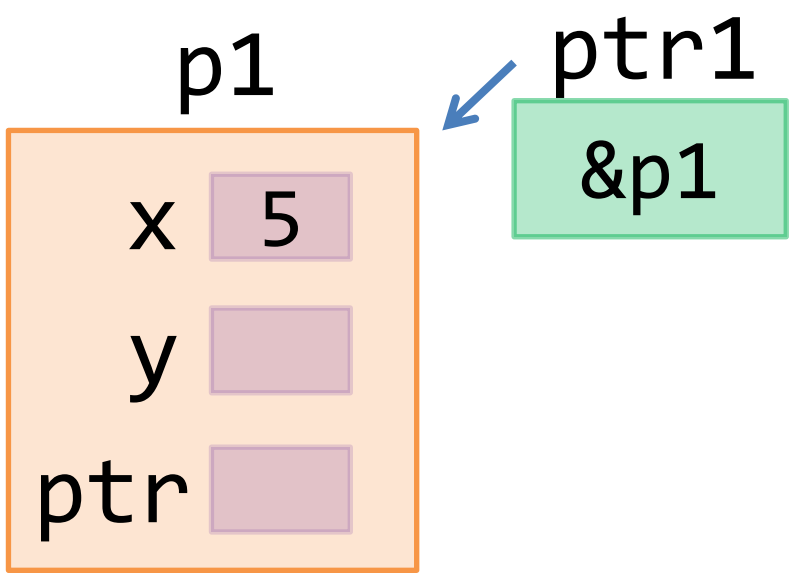



```
Point p1;
Point *ptr1 = &p1;
//*ptr1和p1代表的一樣;
p1.x = 5;
(*ptr1).x = 5;
ptr1->x = 5;
```

用法3
struct包指標

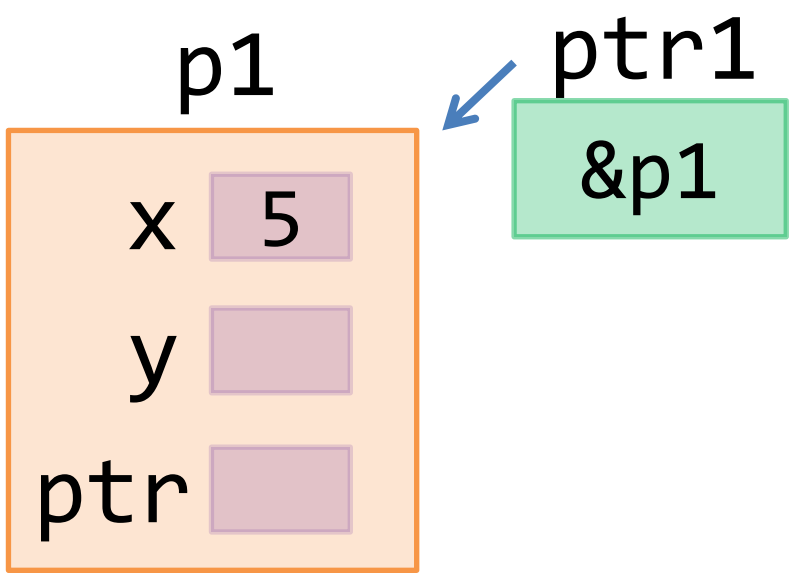
```
struct Point{
    int x;
    int y;
    int *ptr;
};
```

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



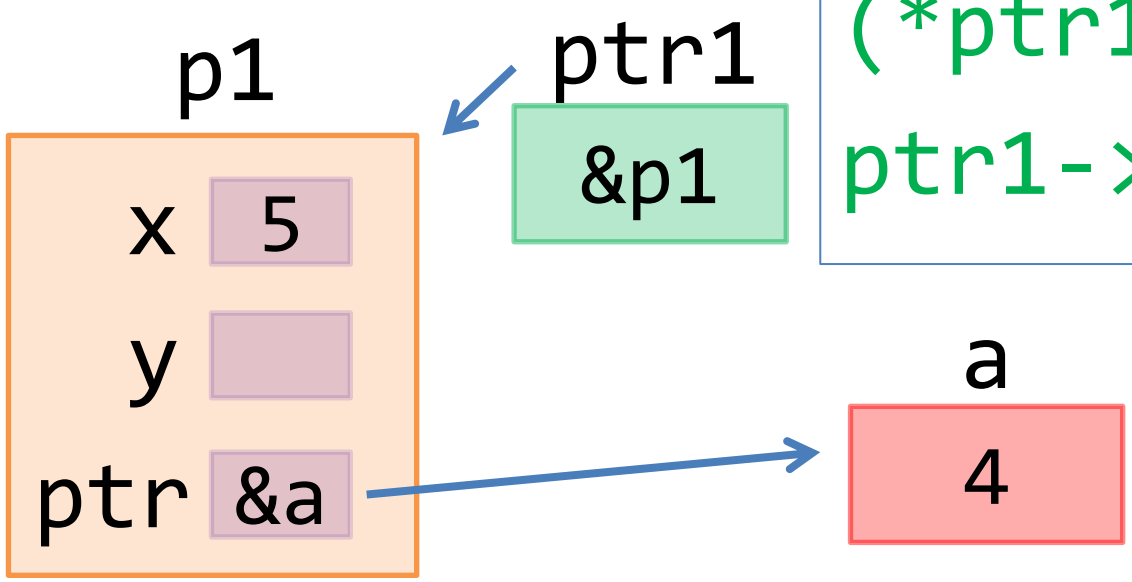
```
struct Point{
    int x;
    int y;
    int *ptr;
};
```

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



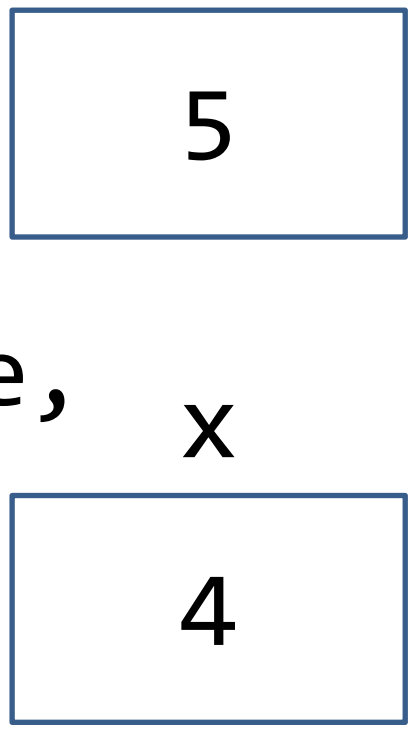
```
struct Point{
    int x;
    int y;
    int *ptr;
};
```

```
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;
```



當程式執行到變數的scope,
這些變數就會自動
被配置記憶體空間。

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!!!

所以,
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];
```

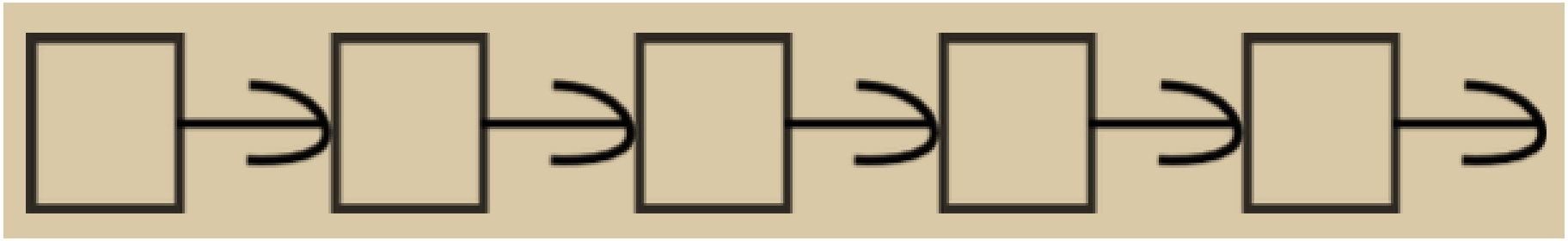
```
1.int *ptr = new int;
  delete ptr;
2.int *ptr = new int[100];
  delete [] ptr;
3.Point *ptr = new Point;
  delete ptr;
4.Point *ptr = new Point[100];
  delete [] ptr;
```

```
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

linked list是什麼??

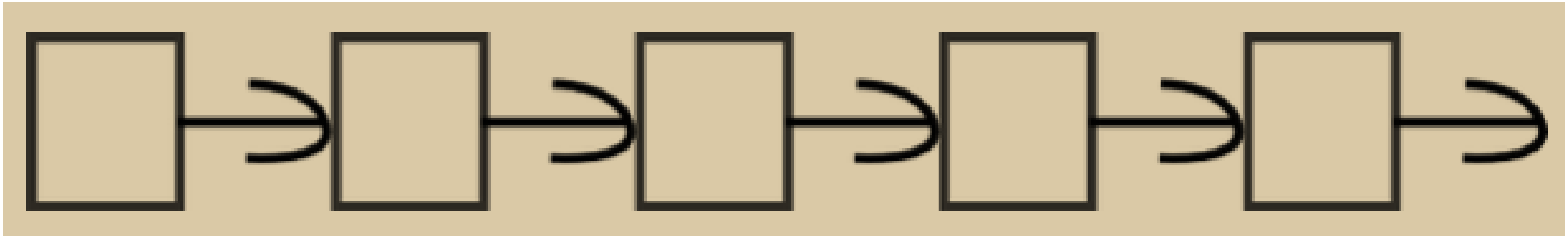


就是鍊結串列!

每一格都有資料,並且連接起來

跟陣列有甚麼不同?

- 可以放不同型態的資料
- 有幾格用幾格,不浪費空間
- 可以隨時插入或移除
- 有方向性



怎麼放不同型態的資料?

ans: struct

怎麼隨時插入或移除方格?

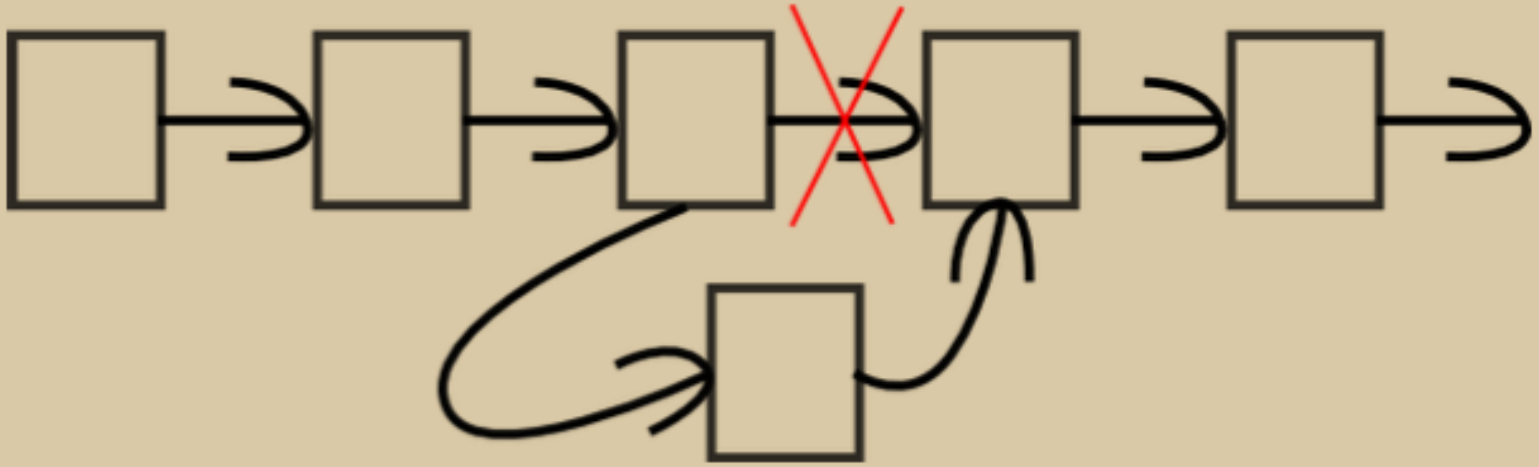
ans:想成鎖鏈!

每一格都用鏈子串起來,

要插入或移除時,

只要先斷開再接回去就好了!

插入



删除

接著,想想怎麼轉成code...

首先,需要什麼??

- 放資料的方格(struct)
- 接到下一格的鎖鏈

- 接到下一格的鎖鏈??

  有個東西...

  1.可以指著別人

  2.知道別人的位址

  3.知道位址後就可以拿裡面的東西

指標!!

把方格串在一起的,就是指標

先宣告一個struct叫node

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

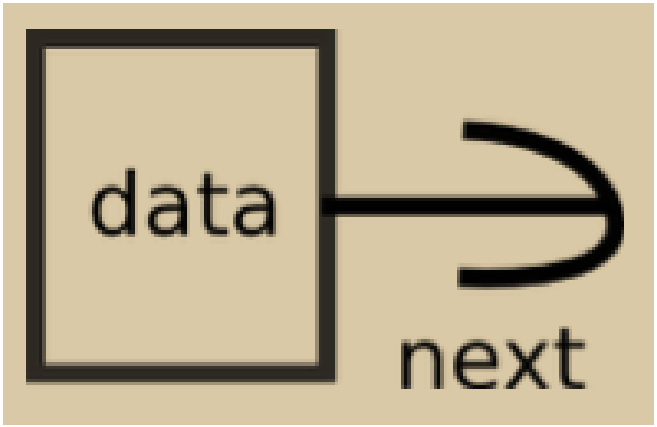
先宣告一個struct叫node

```
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：



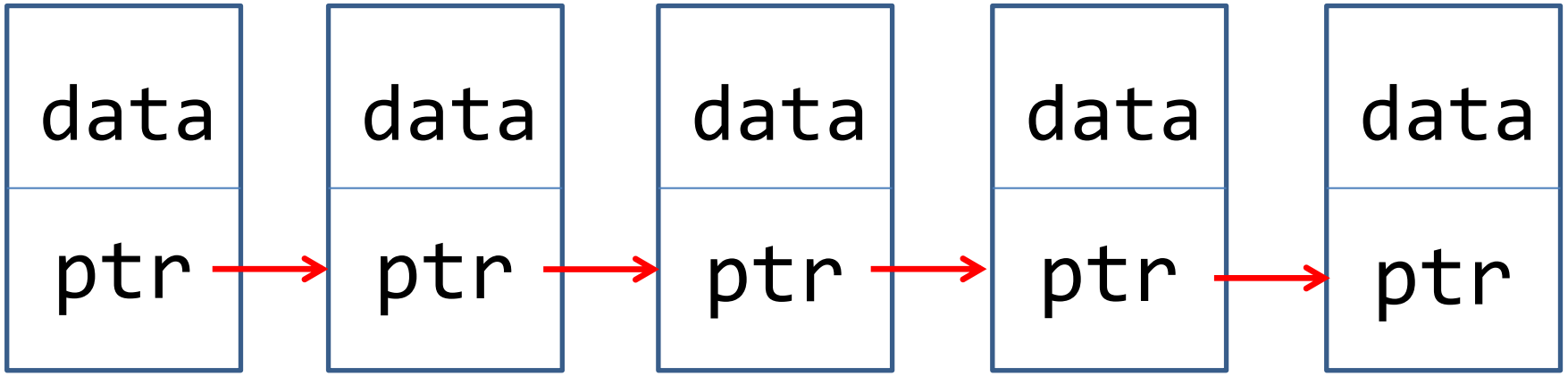
有了一個node,就可以用出很多node,
但如何把node串在一起,
形成一條 linked list呢?

先來想想一條 linked list 需要哪些重要的東西...

- 開頭:有了開頭,就有辦法沿著 next, 走完整條 linked list
- 結尾:要有個結尾才有走完的一天

於是我們用開頭來代表一個 linked list, 而 next == NULL 的 node 則代表 linked list 的結尾。

```
//假設開頭叫作 head
node *ptr = head;
while(ptr != NULL) {
    //對目前的 node 做處理
    ptr = ptr->next; //往前走一步
}
```


data
ptr
data
ptr
data
ptr
data
ptr
data
ptr