

程式設計與實習(二)

BY 孫茂勛


EMAIL:JOHN85051232@GMAIL.COM

Pointer

今天有一種變數可以儲存某個記憶體位置，叫做指標變數

Ex : `int *b = &a;` //宣告**b**是個指標變數，指向(儲存)**a**變數的記憶體位置

記憶體	0x000001	0x000005
值	10	0x000001



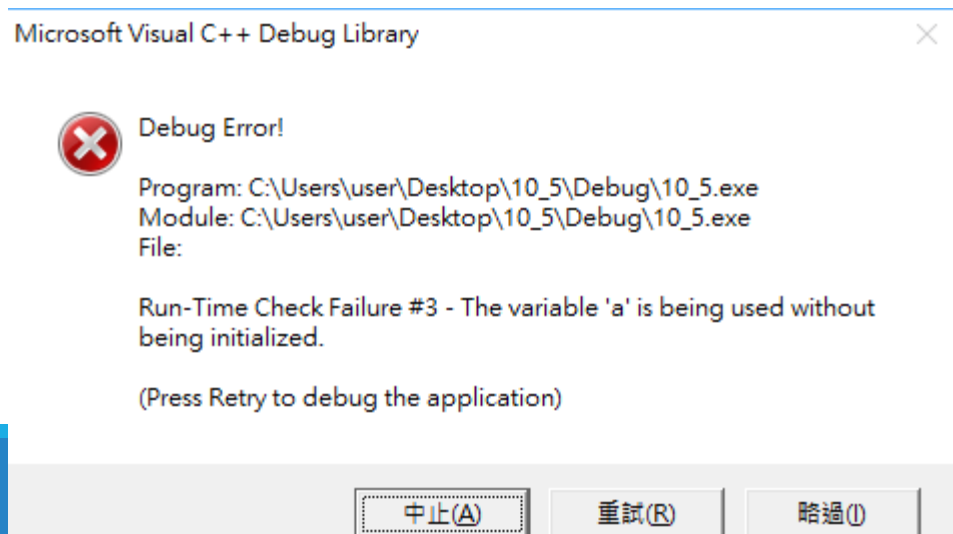
變數**b**有自己的記憶體位址，但儲存的值是**a**的記憶體位址。
知道**a**變數的記憶體位址，我們就可以透過**b**去修改**a**。

Pointer

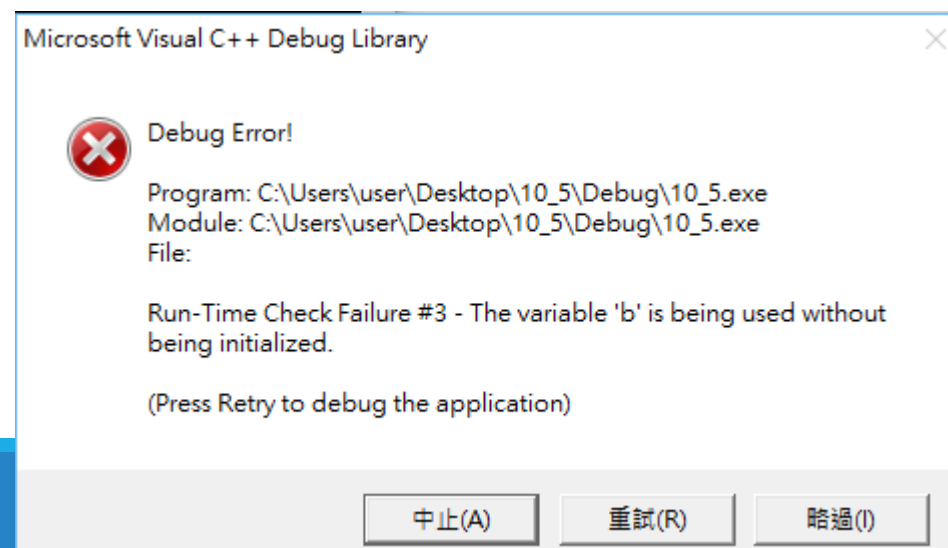
```
int num = 10;  
int *ptr = NULL; //宣告一個指標，不指向任何東西  
  
ptr = &num; //ptr指向num的記憶體位址  
  
printf("num的值 : %d\n", num);  
printf("num的記憶體位址 : %p\n", &num);  
  
printf("ptr指向位址的值 : %d\n", *ptr);  
printf("ptr指向位址的記憶體位址 : %p\n", ptr);  
printf("ptr的記憶體位址 : %p\n", &ptr);
```

不管什麼變數初始化很重要

```
int main()
{
    int a;
    printf("%d",a);//a是什麼?
    return 0;
}
```



```
int main()
{
    int *b;
    printf("%d",*b);//b是什麼?
    return 0;
}
```



New、Delete運算子

首先先宣告一個變數跟陣列來複習一下

`Int a = 0;`

adress	0x0
value	0

`Int a[3] = {0};`

adress	0x0	0x4	0x8
value	0	0	0

電腦要給多少的記憶體空間都是程式執行前就知道的

Q：想在程式執行中根據變數大小產生陣列？

New、Delete運算子

試試看

```
Int x = 3;  
Int arr[x] = {0};
```

C/C++無法用變數來宣告陣列，那如果我想要每次執行程式的時候陣列的大小都不一樣？

New、Delete運算子

New：為變數動態產生一塊記憶體空間

Delete：將該變數的記憶體空間歸還(刪除)

```
int x = 3;
```

```
int *arr = new int[x]; // 不能用arr[]，必須要用指標的方式
```

之前學過的變數宣告都是在**程式執行前**電腦就把**記憶體配置好了**，所以無法使用變數來宣告。

new可以讓電腦**動態配置記憶體**給變數，所以大小可以不固定。

New、Delete運算子

之前的陣列宣告



變數空間再程式執行前完成，無法用x配置記憶體空間。

使用動態宣告



程式執行後才用x配置記憶體給變數。

New、Delete運算子

```
int *ptr = new int;//產生一個int的變數
```

```
printf("ptr的值為:%d\n",*ptr);  
printf("ptr指向的記憶體位址:%p\n",ptr);  
printf("ptr的記憶體位址:%p\n",&ptr);  
delete ptr;
```

delete完後再printf一次變數試試看

New、Delete運算子

```
int x = 10;  
int *arr = new int[x];//用指標方式產生一個int的陣列  
for(int i = 0; i < x ; ++i)  
{  
    arr[i] = i;//配置空間不會主動做初始化，要自己做  
    printf("%d ",arr[i]);  
}  
delete arr;//將這個指標的空間歸還給電腦
```

delete完後再printf一次陣列試試看

Pointer

```
int main()
{
    int *b;
    printf("%d",*b);
    //因為*b還沒儲存任何記憶體位址，無法將該位址的資料取出
    return 0;
}
```

如何解決？

1. `b = &a;`//b指標參考其他變數的記憶體位址

2. `int *b = new int;`//讓電腦生出一個整數的記憶體位址給b

Pointer

比較一下兩個的差別

```
int main()
{
    int a = 10;
    int *b = &a;
    *b = 300;
    printf("%d",*b);
    return 0;
}
```

```
int main()
{
    int *b = new int;
    *b = 300;
    printf("%d",*b);
    return 0;
}
```

New、Delete運算子

動態宣告在C/C++有不同的function：

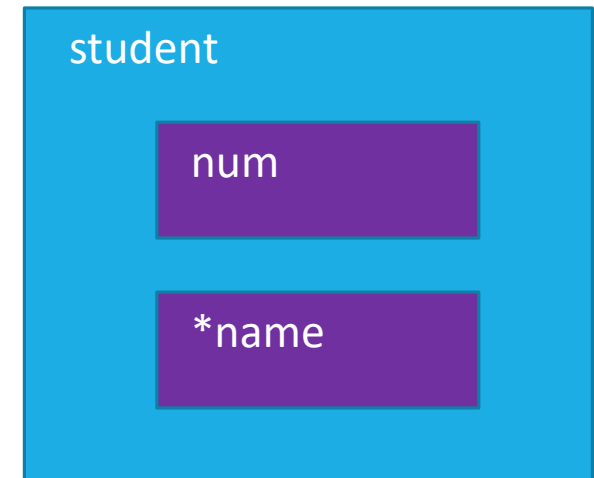
C：malloc、free

C++：new、delete

當struct遇上指標

複習一下struct是什麼：可以把東西包在一起的概念
Ex：學生的資料有座號、姓名.....

```
struct student
{
    int num;
    char *name; //如果用char name[10]有什麼差別?
};
```



如何得到struct內的資料？

student.num
student.name

當struct遇上指標

```
...  
int main()  
{  
    student s;  
    s.num = 1;  
    s.name = "Abel";  
    printf("%d %s\n",s.num,s.name);  
}
```


當struct遇上指標

那struct的變數能不能是指標？當然可以

```
int main()
{
    student *p1 = new student;//動態配置記憶體
    system("pause");
    return 0;
}
```

功能和一般變數用.是一樣的效果，
只是指標取得內部成員變數的方式不太一樣

如何得到struct指標變數內的變數？

student->num
student->name

當struct遇上指標

試試看

```
int main()
{
    student *p1 = new student;
    printf("%s %d\n",p1->name,p1->num);
    system("pause");
    return 0;
}
```

當struct遇上指標

剛剛的Code會亂碼，記得只要是變數都要乖乖給初始值

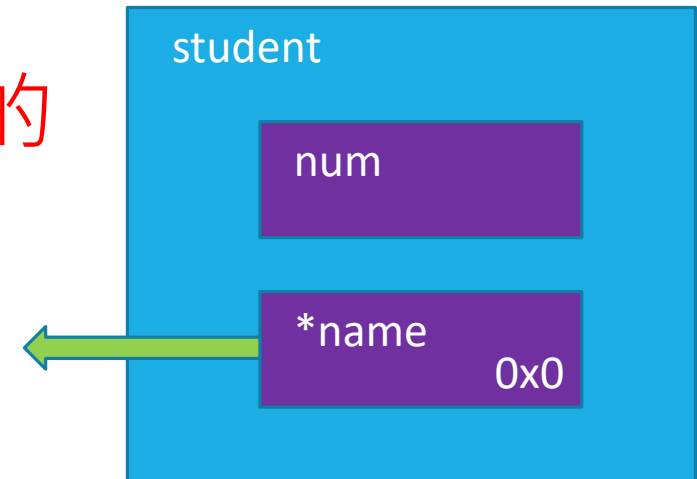
```
int main()
{
    student *p1 = new student;
    p1->name = "john";
    p1->num = 123;
    printf("%s %d\n",p1->name,p1->num);
    system("pause");
    return 0;
}
```

當struct遇上指標

struct變數內部的指標變數代表什麼？

我們說過指標是紀錄一個變數記憶體的位置

`char *name;`
一個紀錄字串(字元陣列)開頭位址的指標變數

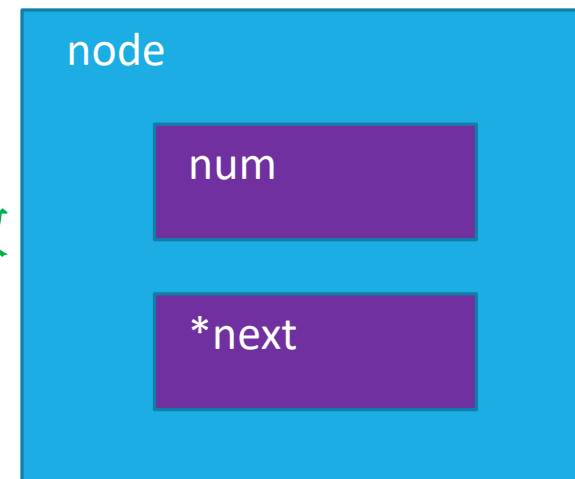


adress	0x0	0x1	0x2	0x3
value	'J'	'O'	'H'	'N'

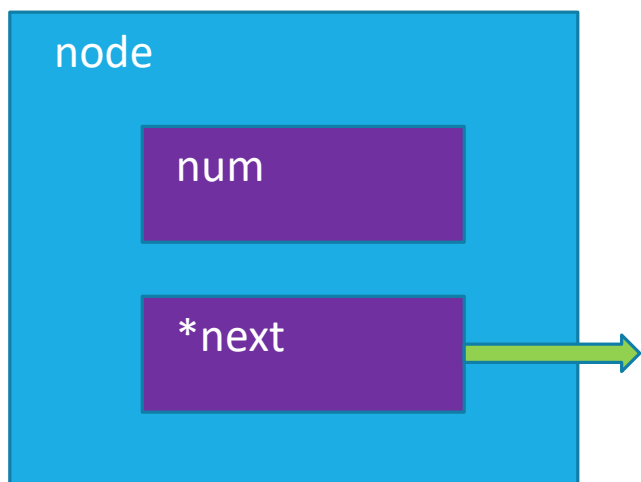
```
printf("%c",*name);//j  
printf("%p",name);//0x0
```

當struct遇上指標

```
struct node
{
    int num;
    node *next; //宣告一個node結構的指標變數
};
int main()
{
    node *n1 = new node; //宣告一個node結構的指標變數
    ...
}
```



當struct遇上指標



`node* next;`
可以用來紀錄node結構變數的記憶體位址

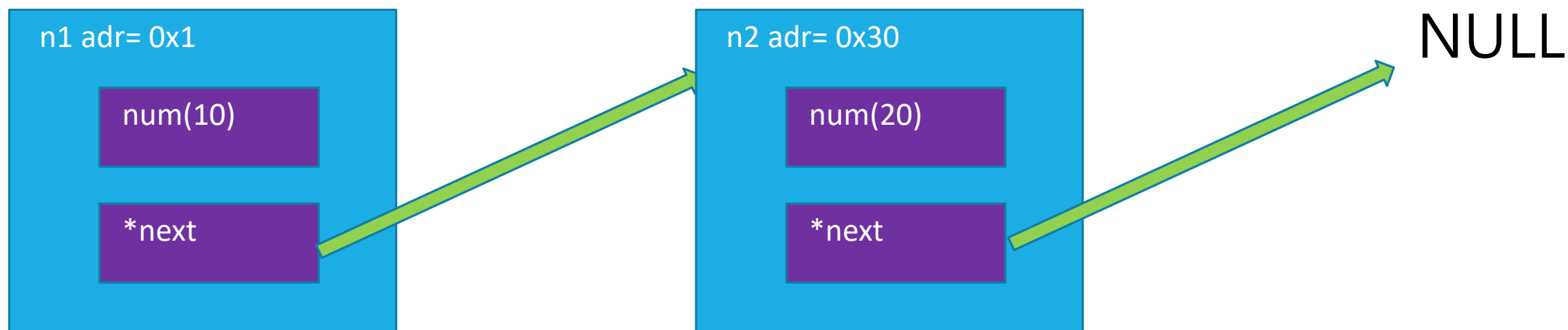
當struct遇上指標

```
int main()
{
    //宣告node結構的指標變數
    node *n1 = new node;
    node *n2 = new node;
    n1->num = 10;
    n2->num = 20;
    n1->next = n2;
    n2->next = NULL;
    printf("n1的值:%d n1->next儲存的記憶體位置:%p\n", n1->num, n1->next);
    printf("n2的值:%d n2的記憶體位置:%p\n", n2->num, n2);

    system("pause");
    return 0;
}
```

當struct遇上指標

What Happen?

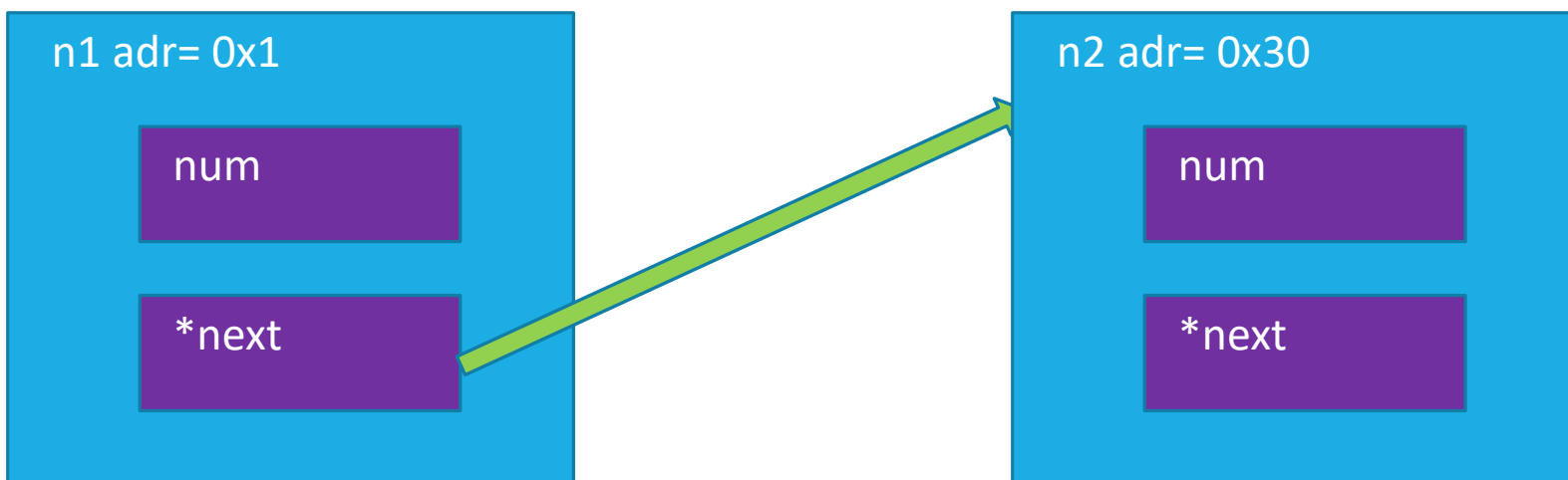


當struct遇上指標

```
printf("n2的值:%d\n", n1->next->num);
```



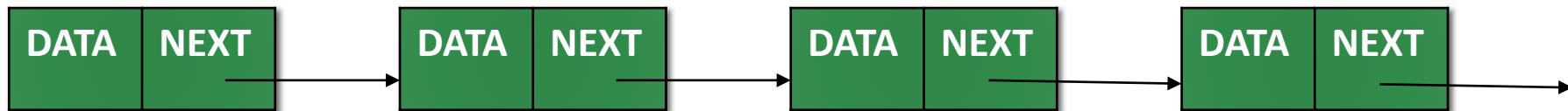
n2的記憶體位置



Linked list

鏈結串列(Linked list)：

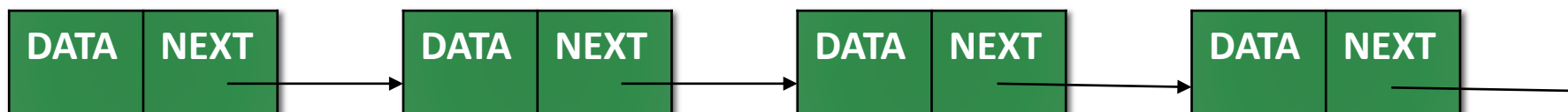
- 資料結構的一種
- **struct + pointer**
- 每一個節點(node)都包含指向下一個節點的指標



Linked list

基本操作：

- 移動(走訪)
- 新增
- 刪除



Linked list

先建好結構

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int num;
    node *next;
};
```

```
int main( )
{
    node *n1 = new node;
    node *n2 = new node;
    node *n3 = new node;
    n1->num = 10;
    n1->next = n2;
    n2->num = 20;
    n2->next = n3;
    n3->num = 30;
    n3->next = NULL;

    system( "pause" );
    return 0;
}
```

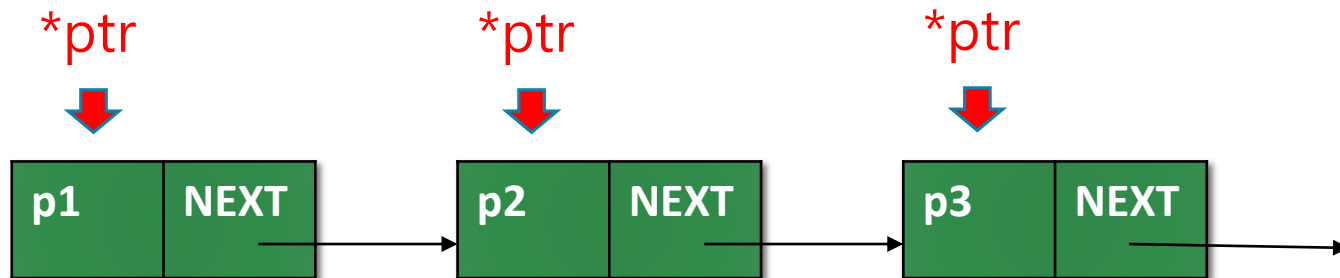
Linked list – 移動

```
printf("%d %d %d \n",p1->num,p1->next->num,p1->next->next->num);
```



Linked list – 移動

```
node *ptr = p1;  
printf("%d",ptr->num);  
  
ptr = ptr->next;  
printf("%d",ptr->num);  
  
ptr = ptr->next;  
printf("%d",ptr->num);
```



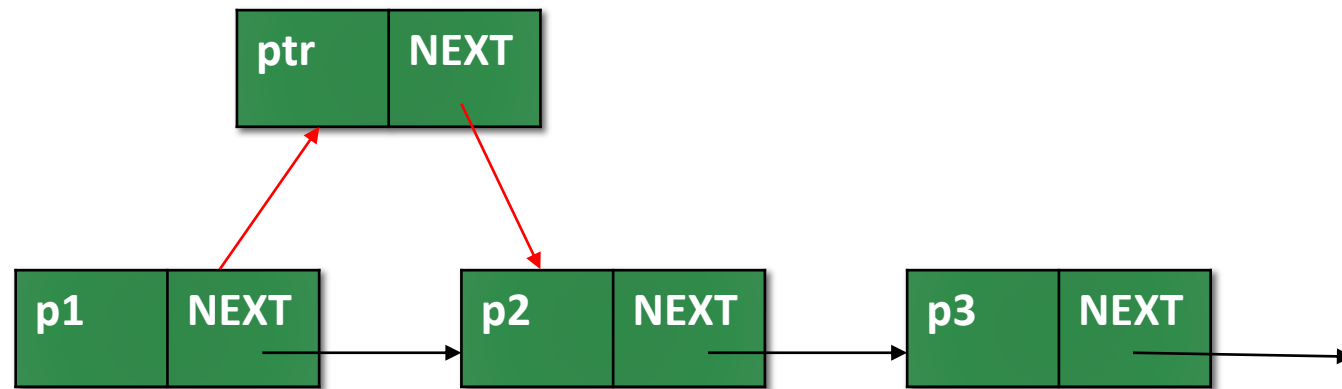
Linked list – 移動

寫成迴圈

```
node *ptr = p1;  
while(ptr != NULL)  
{  
    printf("%d \n",ptr->num);  
    ptr = ptr->next;  
}
```

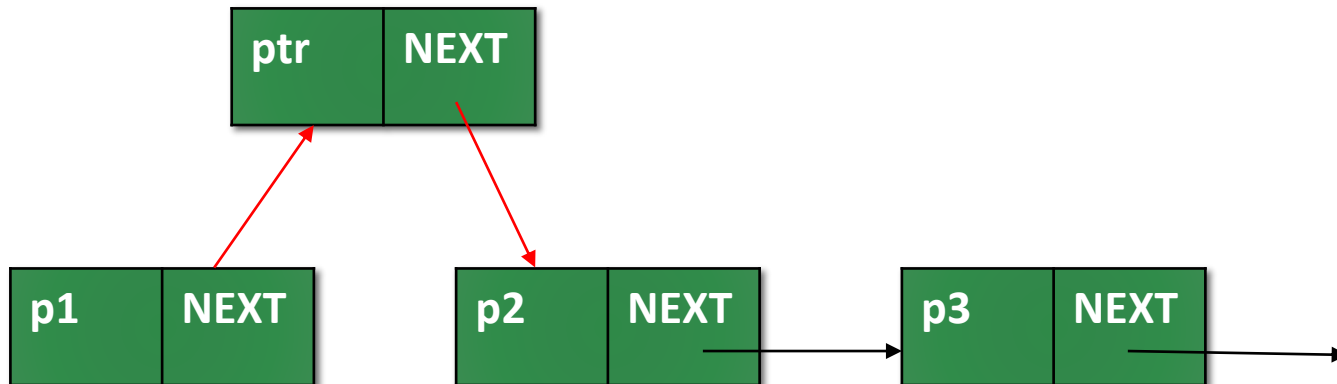


Linked list – 新增

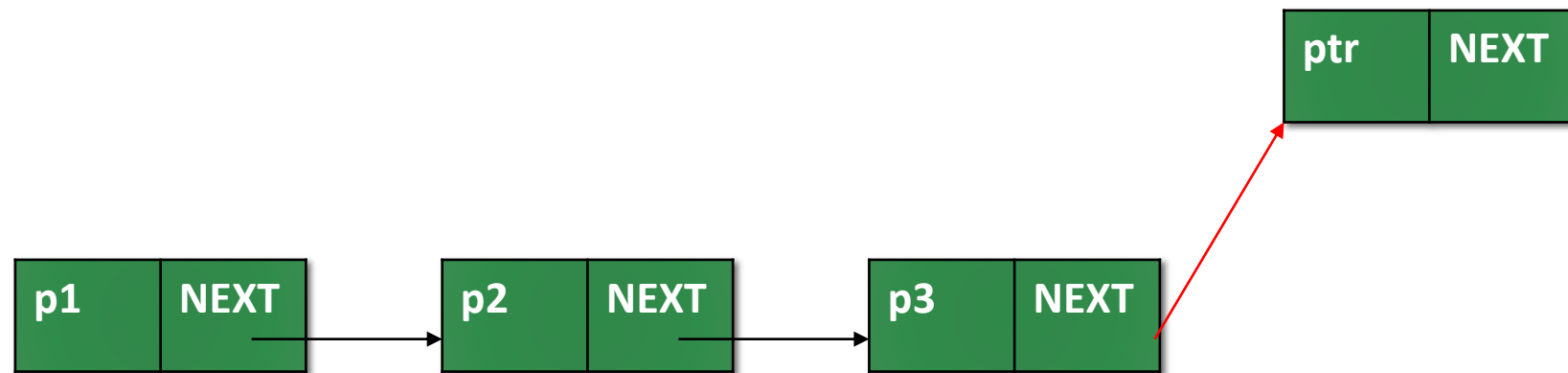


Linked list – 新增

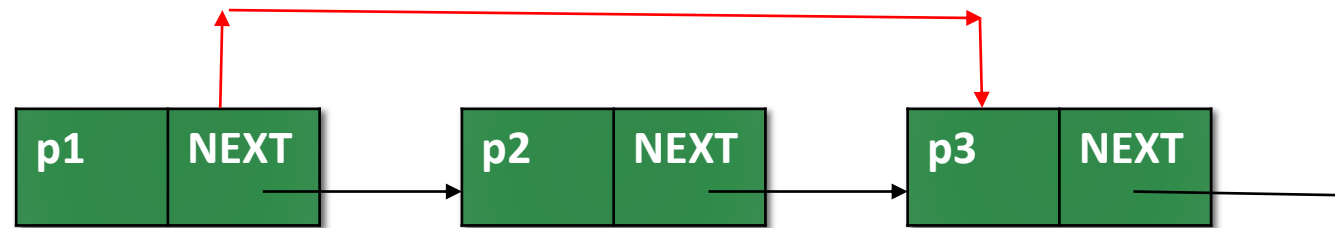
```
node *ptr = new node;//新增一個節點  
ptr->num = 40;  
p1->next = ptr;  
ptr->next = p2;
```



Linked list – 新增(2)



Linked list – 删除



Linked list – 刪除

```
node *ptr = p1->next;  
p1->next = p1->next->next;
```

delete ptr;//刪除ptr所在位置的節點

