【第七章】

指標

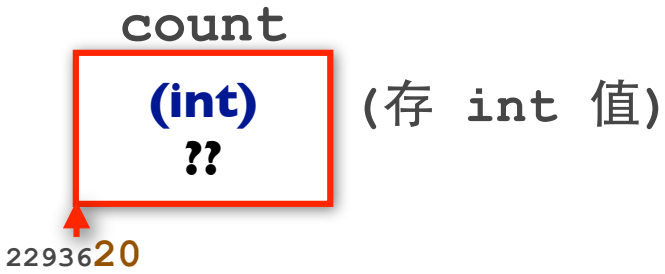
講師: 李根逸 (Ken-Yi Lee), E-mail: feis.tw@gmail.com

課程大綱

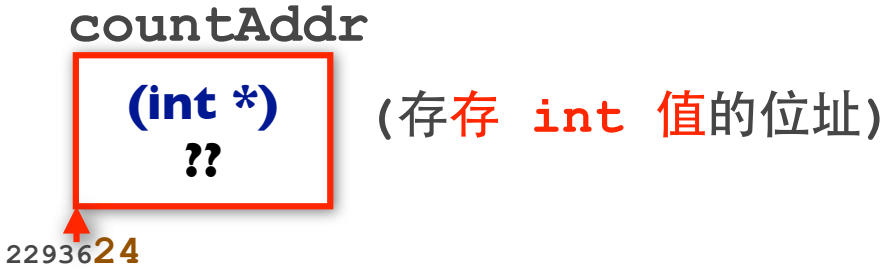
- 指標變數宣告 (type *) [P203]
- 取址運算子 (&) [P204]
- 間接運算子 (*) [P205]
- 指標與函式
  - 函式傳值 [P206]
  - 函式傳址 [P207]
  - 傳值還是傳址？[P209]
- 指標與陣列 [P211]

指標變數宣告 (type *)

- 指標 (Pointer) 是 C 語言的一大特色，是儲存記憶體位址的資料型態
- 指標變數宣告語法：
  - 資料型態 *變數名稱;
    - 表示變數名稱內存放的是一存放該資料型態值的記憶體位址
- 宣告指標變數與一般變數的差別：
  - **int count;**

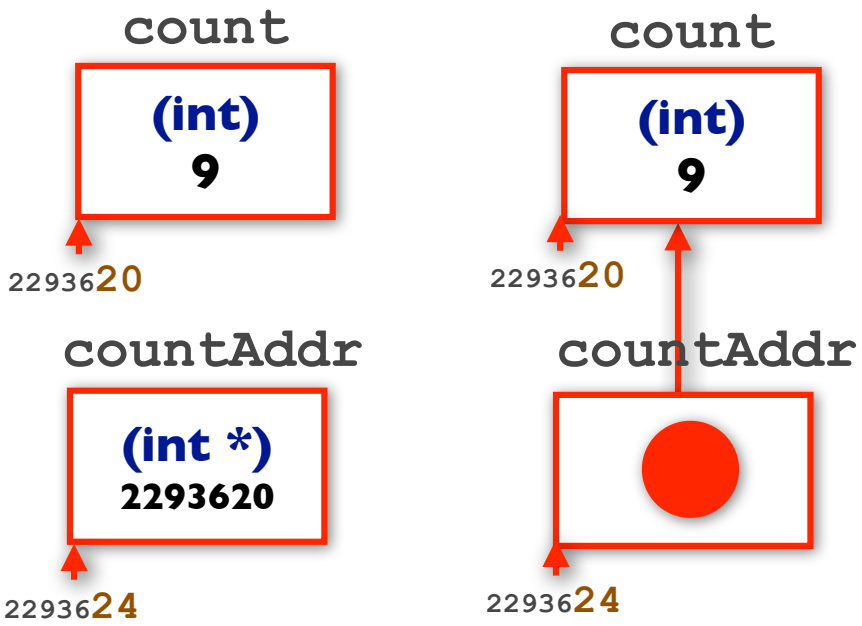


  - **int *countAddr;**



取址運算子 (&)

- 變數宣告後依照資料型態會佔據一定的記憶體空間，並具有一個在記憶體的位址。我們可以利用取址運算子 (&) 去取得某變數的記憶體位址：
  - **int count** = 9;
  - **int *countAddr = &count**;



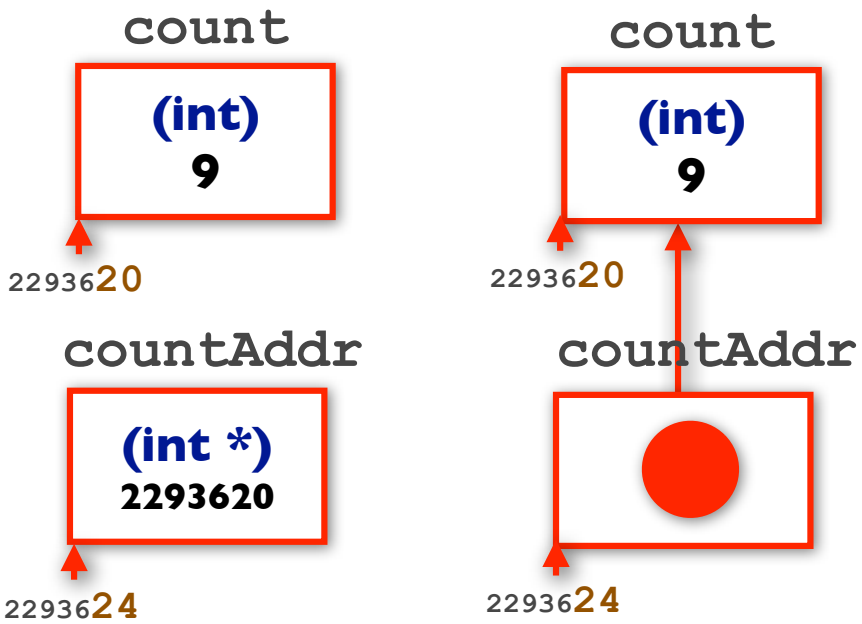
表示式	資料型態	值
count	int	9
&count	int *	2293620
countAddr	int *	2293620

間接運算子 (*)

- 相對地，我們可以利用間接運算子 (*) 從位址取得或指定某個記憶體位址存放的值 (別跟宣告指標用的*搞混)
  - **int count** = 9;
  - **int *countAddr** = **&count**;
  - **int result** = ***countAddr**;

注意這裡出現的兩個星號 (*) 代表的意思不同！(重要)

參考 pointer.cpp



表示式	資料型態	值
count	int	9
&count	int *	2293620
countAddr	int *	2293620
*countAddr	int	9
result	int	9

函式傳值

- 當我們將變數送入函式執行時，是使用『傳值呼叫』的方式。意味著只有變數的值會被複製一份進函式。在函式內部對參數做任何的變動不會改變到原本的變數。

```
void test(int n) {
  n = 5;
  return;
}

int main() {
  int a = 3;
  test(a);            /* 傳 a 的值給 test */
  printf("%d", a);
  return 0;
}
```

對 **test** 來說他只是得到一個整數，無法知道整數原本存放的地方或來源

函式傳址

- 我們可以將變數的『記憶體位址』作為值複製進入函式執行。此時在函式內部對參數用『間接運算子』指定新的值時就會改變原本的變數值。

```
void test(int *n) {
  *n = 5;
  return;
}

int main() {
  int a = 3;
  test(&a);          /* 傳 a 所在位址給 test */
  printf("%d", a);
  return 0;
}
```

對 **test** 來說他得到了一個位址，經由間接運算子 **(*)** 可以取得並指定該位址上的值

可以看成 **C** 語言只能用複製傳值的方式呼叫函式，而位址也是一種值

《範例》交換與排序

- 試寫一函式 **`void swap(int *, int *)`**，將輸入的兩個整數參數的值交換 **(swap.cpp)**。
  - 例如:
    - a = 1，b = 2，執行完 swap 後，a = 2, b = 1
- 試寫一函式 **`void sort(int *, int *)`**，將輸入的兩個整數參數的值由小到大排 **(sort.cpp)**。
  - 例如:
    - a = 1，b = 2，執行完 sort2 後，a = 1, b = 2
    - a = 2，b = 1，執行完 sort2 後，a = 1, b = 2

傳值還是傳址？

- 簡言之，在 C 語言裡如果你想要讓其他函式可以幫你修改變數的值時就需要傳址
  - 傳值的設計讓不同函式之間的非全域變數是完全沒有關係的、是不會互相影響的，可以避免污染與干涉！
  - 要藉由『呼叫函式』的方式來修改目前所在函式的變數值時有兩個方式：
    - 接收回傳值：

```
var = func();
```

    - 傳送變數的位址 ：

```
func(&var);
```

傳陣列只能傳址

- 能傳值就傳值，可以避免函式之間的間接汙染！
  - 變數位址就好像變數真實的名字一樣，得到的函式可以為所欲為！

《範例》讀出與印出

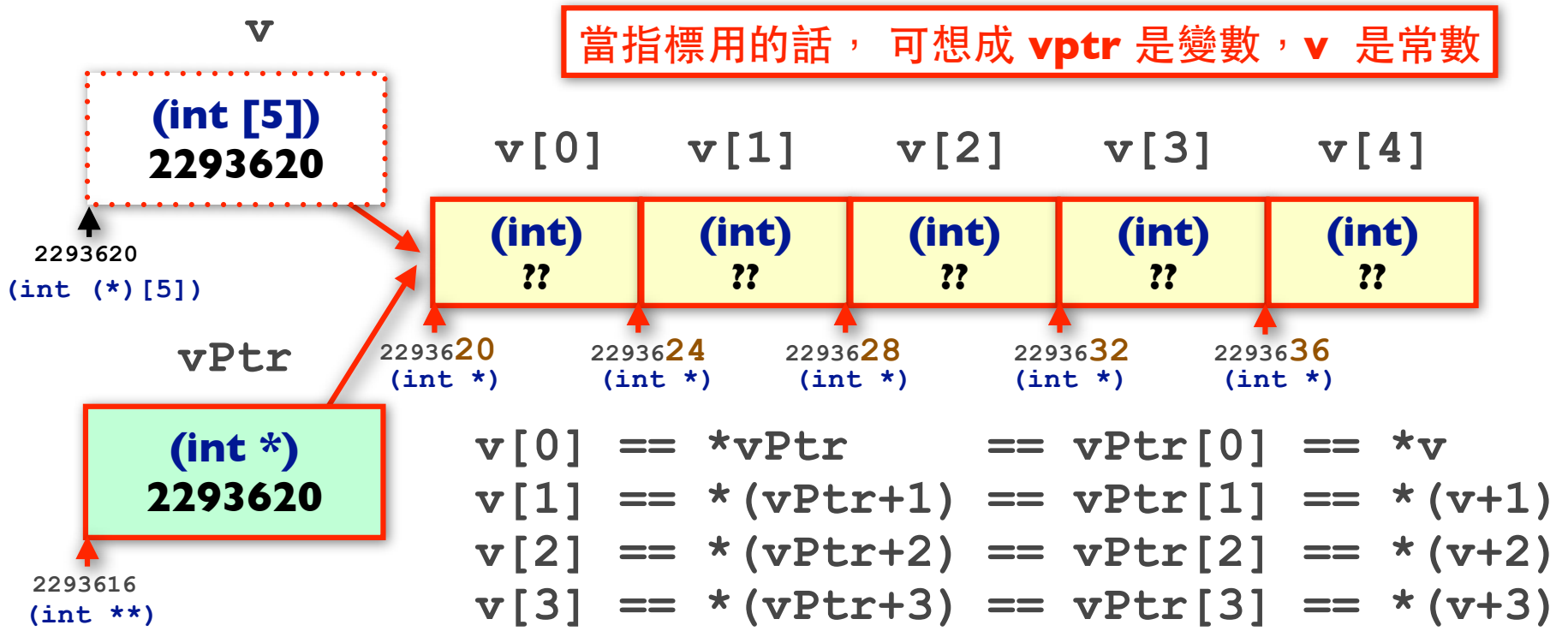
- 試寫兩函式，讓 **read(...)** 可讀入一成績，並用 **show(...)** 印出 **(read_show.cpp)**

```
#include <stdio.h>
#include <stdlib.h>
int read();
void show(int);

int main() {
  int grade = read();
  show(grade);
  system("pause");
  return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
void read(int *);
void show(int);

int main() {
  int grade;
  read(&grade);
  show(grade);
  system("pause");
  return 0;
}
```

指標與陣列

- C 語言的指標與陣列關係相當密切，似乎是一體的兩面，但是又有著蠻多的不同：
  - `int v[5];` /* 會配置五個 int 的記憶體空間 */
  - `int *vptr = v;` /* 會配置一個 int *的記憶體空間 */



《範例》比大小

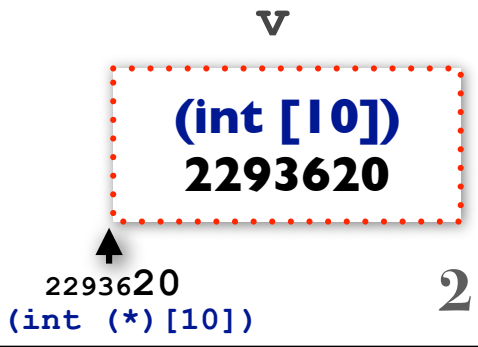
- 試修改 **max_1.cpp** 為 **max_2.cpp** ，寫一可以讓使用者輸入五個整數，回傳最大值的函式 **max1v**
  - `int max1v( int *);`
- 試修改 **max_2.cpp** 為 **max_3.cpp**，寫一可以讓使用者輸入任意個整數，回傳最大值的函式 **max2v**
  - `int max2v( int, int *);`

可以用指標變數來接收陣列名稱的位址值

《補充》陣列名稱的特殊性

- 陣列名稱一般情況下可以視作是一個指標常數：
  - 陣列名稱可以隱性轉型成指標，其值為該陣列開頭元素的記憶體位置
  - 陣列名稱無法放在指定運算子 (=) 左方去指定成其他記憶體位址 (常數性質)
- 對陣列名稱取址：
  - 一般我們無法對字面常數取址。例如 &3 是不合法的，但是陣列名稱並不是個字面常數，卻又不需要另外配置記憶體儲存，所以對陣列名稱做取址運算 (&) 採用了特別的處理方式：
    - 對陣列名稱取址後的值也是陣列開頭位址

```
int v[10];
(int) &v == (int) v
```



《範例》指標與陣列 (1)

- 將下述程式迴圈的部份用指標改寫：

```
int main() {
  int v[10];
  for (int i = 0; i < 10; i++) {
      v[i] = 0;
  }
  return 0;
}
```

- 結果：

```
int main() {
  int v[10];
  int *p = v;
  for (int i = 0; i < 10; i++) {
      *p = 0;
      p++;
  }
  return 0;
}
```

《範例》指標與陣列 (2)

- 將下述程式迴圈的部份用指標改寫：

```
int main() {
  int v[10];
  for (int i = 0; i < 10; i++) {
      v[i] = 0;
  }
  return 0;
}
```

- 結果：

```
int main() {
  int v[10];
  for (int *p = v; p != &v[10]; p++) {
      *p = 0;
  }
  return 0;
}
```

習題 (1)

- **[E0701]** 試寫一函式 **inc(int *v)** ，呼叫後讓 **v** 的值加一