【第八章】

進階指標

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

課程大綱

- 陣列的複製 [P220]
- 字串的特殊性 [P223]
- 指標陣列 [P225]
- 指標與二維陣列 [P228]
- 動態記憶體
  - C 語言中動態記憶體的配置 [P234]
  - C 語言中動態記憶體的釋放 [P235]

```
int v[5];
int *vptr;

v          = v;    // 語法錯誤
vptr       = vptr; // 合法

v[0]       = 1;    // 合法
*vptr      = 1;    // 邏輯錯誤

v          = vptr; // 語法錯誤
vptr       = v;    // 隱性轉型
```

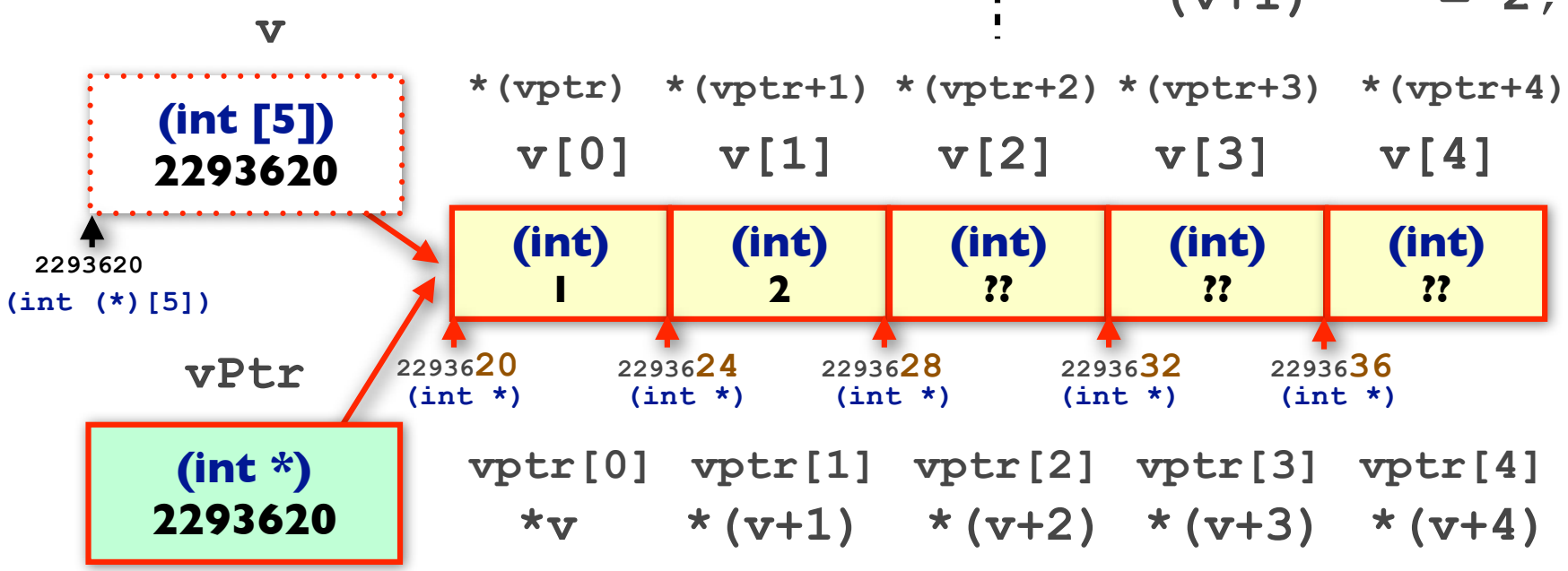
```
int v[5];
int *vptr = v;

v[0]       = 1;
vptr[0]    = 1;

v[1]       = 2;
vptr[1]    = 2;

*vptr      = 1;
*v         = 1;

*(vptr+1)  = 2;
*(v+1)     = 2;
```



陣列的複製

- 陣列的複製並不能用陣列名稱直覺的做：
  - **int v[5] = {1, 2, 3, 4, 5};**
  - **int n[5];**
  - **n = v;** /* 此指定不被允許！(雖然同型) */
    - 真的要複製陣列需要使用迴圈 (deep copy) ：
      - for (int i = 0; i < 5; i++) { n[i] = v[i]; }
    - 或者要使用 memcpy 這類的記憶體複製函式
- 陣列名稱是個常數，無法放在指定運算子左方。但是指標變數可以：
  - **int *p = v;** /* 此語法合法 (隱性轉型)！ */
    - 使用時 p 與 v 指的是同一個陣列 (shallow copy)

參考 **copy.cpp**

《範例》字串複製

- 試寫一函式 **(strcpy)** 將一字串內容複製到另一字元陣列中
  - void strcpy(char *dst, const char *src);
  - 提示: 字串是個以 '\0' 字元表示結尾的字元陣列
  - 方法一:

```
for (int i = 0; i != 0 && src[i-1] != '0'; i++) {
  dst[i] = src[i];
}
```

  - 方法二:

```
while ( (*dst++ = *src++) != '\0') {}
```

    - 或

```
while ( (*dst++ = *src++) != '\0');
```

《練習》泡沫排序法

- 試寫一名為 **bubbleSort** 的函式，將傳入大小為 **size** 的整數陣列 **array** 內容，變成由小到大排序：

```
void bubbleSort(int *array, int size);
```

  - 例如：

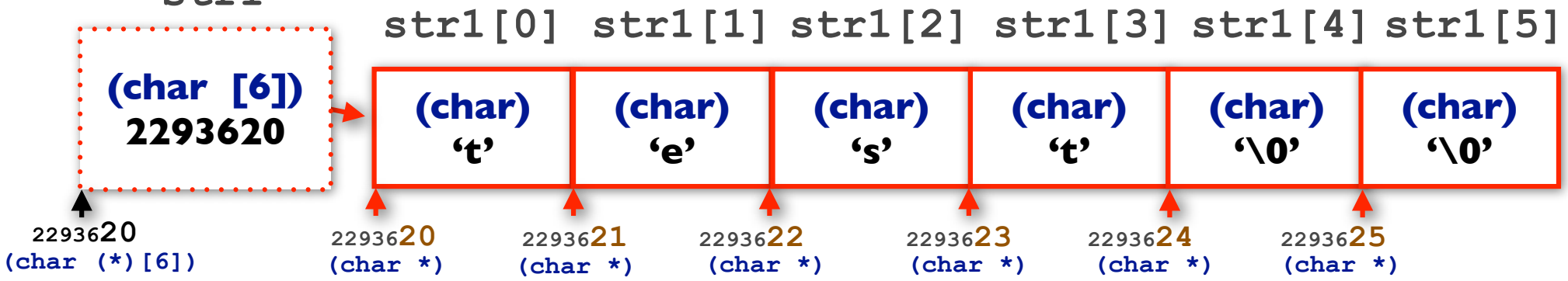
```
int values[3] = {3, 4, 1};
bubbleSort(values, 3);
printf("%d %d %d\n",
   values[0], values[1], values[2]);
```

    - 結果： 1 3 4

字串的特殊性

```
char str1[6] = "test";  // 特殊的初始化方式
```

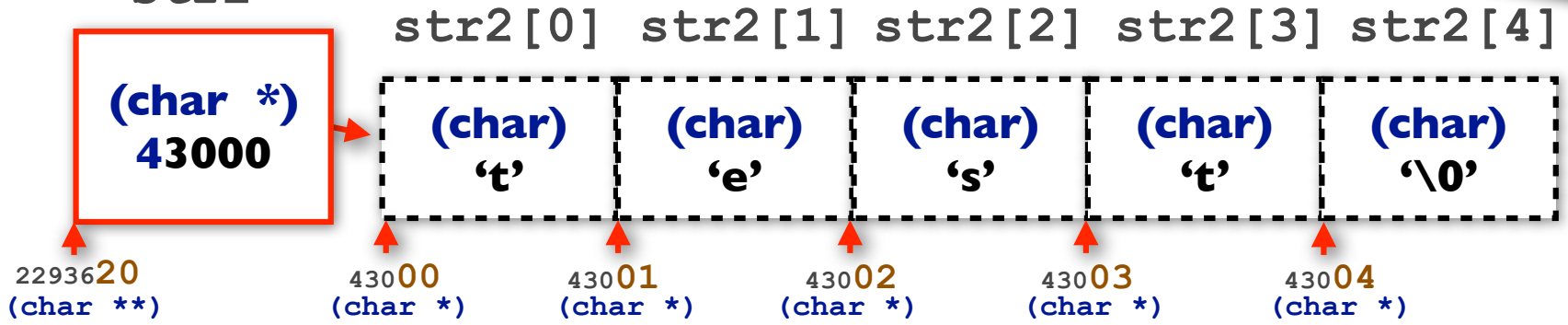
str1




```
char *str2 = "test";  // 在記憶體裡哪裡？
```

危險！勿修改內容

str2




```
const char *str3 = "test";  // 使用 const 修飾字
```

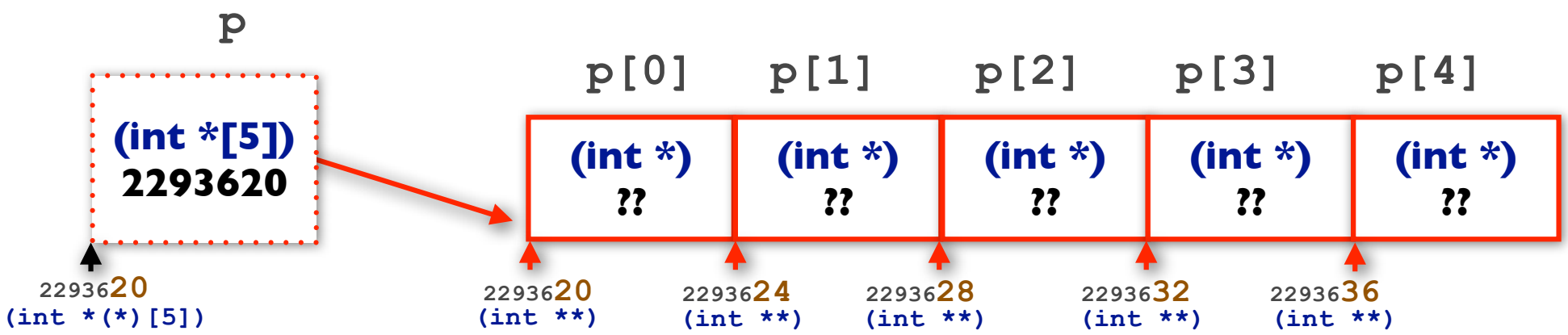
const 修飾字

- **變數或函式宣告時可以在資料型別前加上 const 修飾詞，之後此變數的值就無法直接改變！**
  - 嘗試使用指定運算子 (=) 改變宣告為 const 的變數值時會造成編譯錯誤 (無法產生執行檔)
  - 使用 const 修飾字可以避免不小心修改到不應修改的變數值，此外也有機會可以增加程式的效率。

指標陣列

- 指標陣列顧名思義就是指標所構成的陣列：

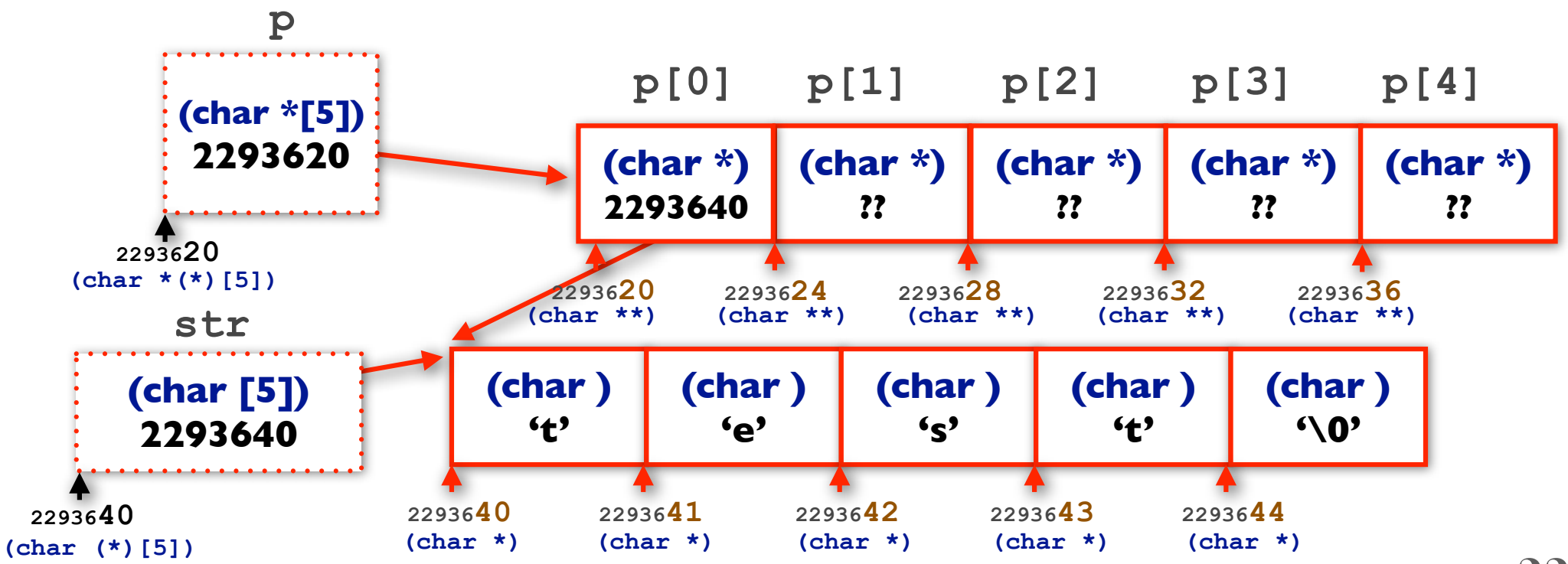
```
int *p[5];  /* 宣告名稱為 p 的陣列，元素型別為 int* */
```



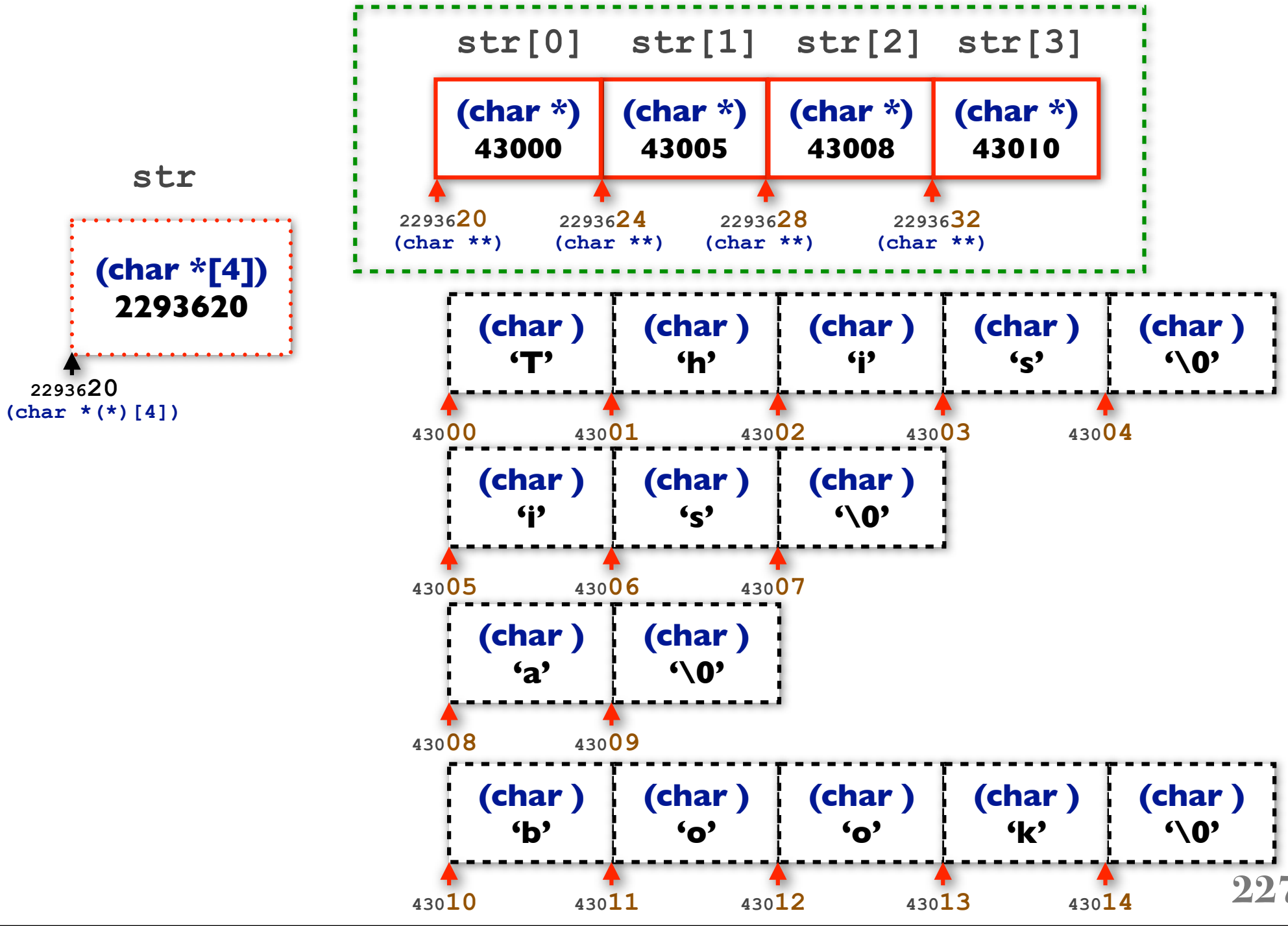
字串陣列

- 字串陣列是一個指向 **char** 的指標陣列：

```
char *p[5];  /* 宣告名稱為 p 的陣列，元素型別為 char* */
char str[] = "test";
p[0] = str;
```



```
const char *str[4] = {"This", "is", "a", "book" };
```

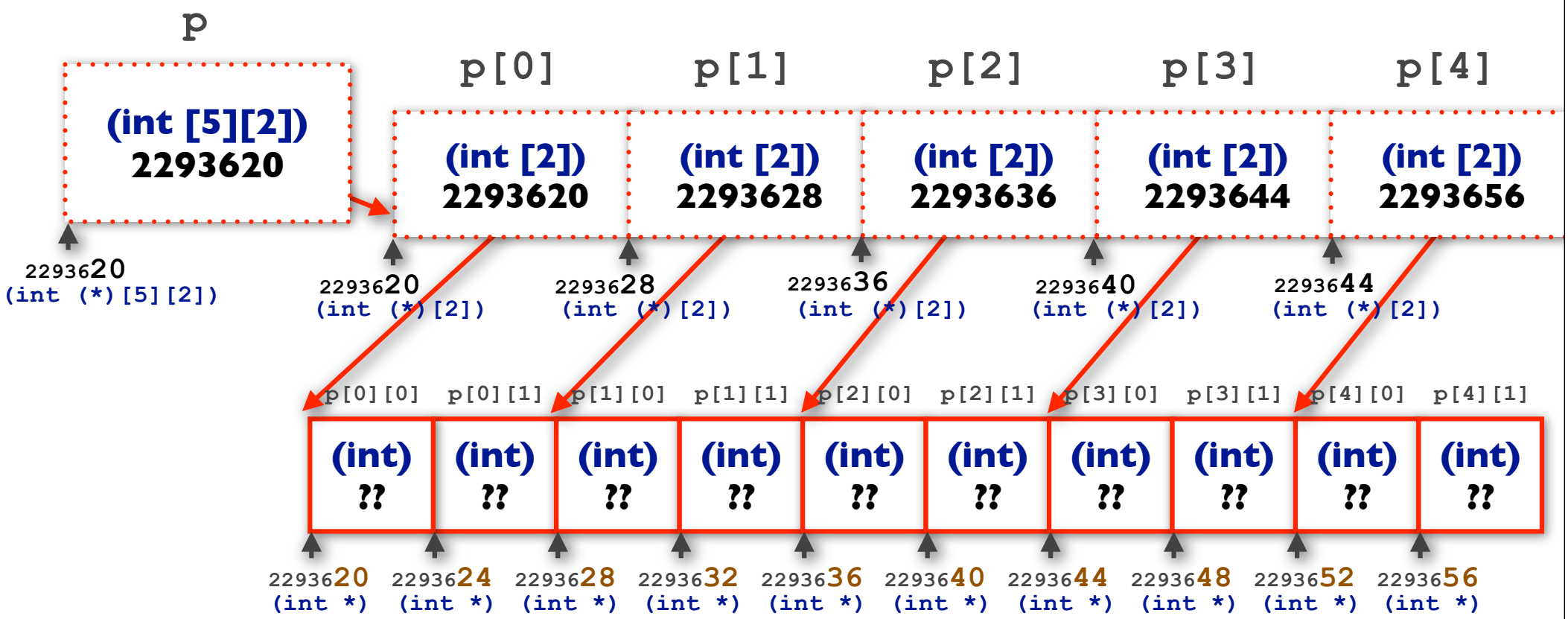


指標與二維陣列

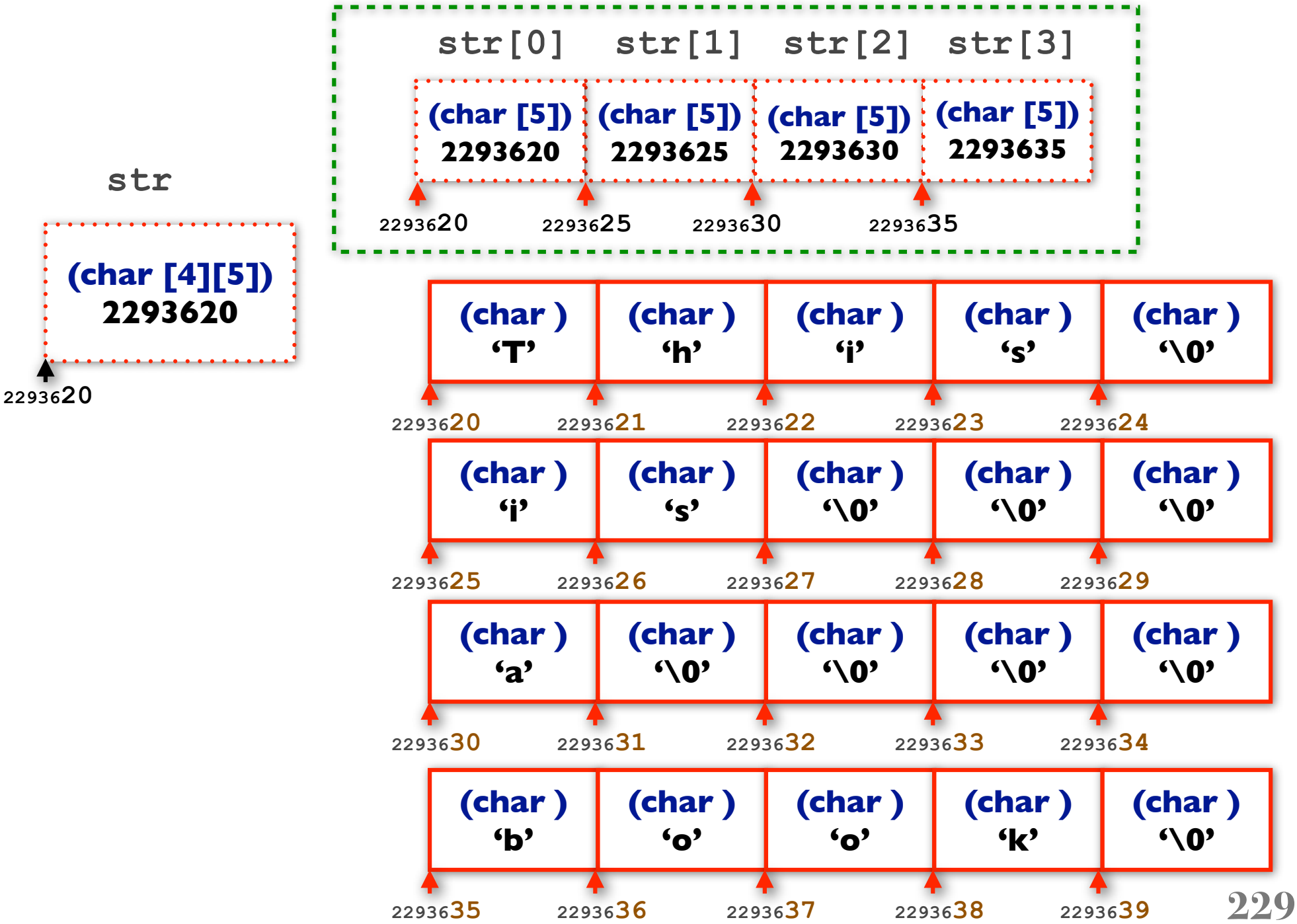
對指標常數取址有特殊性

- 使用二維陣列時也可以想成是一個常數指標陣列：

```
int p[5][2];  /* 宣告名稱為 p 的二維陣列 */
```



```
char str[4][5] = {"This", "is", "a", "book" };
```



```
const char *str1[4] = {“This”, “is”, “a”, “book”};
char str2[4][5] = {“This”, “is”, “a”, “book”};

printf(“%s\n”, str1[1]);
printf(“%s\n”, str2[1]);

printf(“%c\n”, str1[0][0]);
printf(“%c\n”, str2[0][0]);

str1[3] = “pig”;
str2[3] = “pig”;

str1[3][0] = ‘l’;
str2[3][0] = ‘l’;
```

《補充》字串轉換函式

- C 標準函式庫在 **<stdlib.h>** 中提供了許多字串轉換相關的函式：
  - **int atoi**(const char*)
    - 將字串內容轉換成 int 值傳回
    - int num = atoi(“30”);
  - **float atof**(const char*)
    - 將字串內容轉換成 float 值傳回
    - float num = atoi(“30.5”);
  - **long atol**(const char*)
    - 將字串內容轉換成 double 值傳回
    - double num = atol(“30.5”);

《補充》main 函式的參數

- 標準 **main** 的函式宣告：

```
int main(int argc, char* argv[]) {
  ...
  return 0;
}
```

- **main** 函式有兩個參數：
  - argc: 程式在執行時具有幾個參數
  - argv: 字串指標陣列
    - argv[0]： 第一個參數內容 (字串) [程式本身的執行路徑]
  - 將程式編譯後，可以在指令列模式 (cmd) [MS-DOS] 執行執行檔時加入參數
  - 將檔案拖曳到執行檔上時，也會將檔案路徑作為參數傳入

《範例》main 的引數

- 試寫一程式 **(main)**，讓使用者在指令列輸入下列指令後，顯示最大值：

```
main 3 4
4
```

```
system("main 3 4");
```

  - 補充:
    - 在「開始 > 執行」內輸入 cmd 可以啟動 MS-DOS 指令列模式
    - 在 MS-DOS 指令列模式下，可以用 cd 指令切換目錄
- 試寫一程式 **(path)**，在 **Windows** 中拖曳檔案後顯示檔案路徑

C 語言中動態記憶體的配置

- 寫一個函式幫我配置記憶體？
  - 能找到未使用的記憶體位址
  - 讓以後的變數不會重複使用到

```
int *f(int N) {
  int a[N];
  return a;
}
```

函式回傳值型態不能是陣列

- 負責提供可在系統動態記憶體池內配置與管理記憶體的函式:

```
#include <stdlib.h>
void *malloc(size_t size);
```

參考 malloc_3.cpp

```
int *f(int N) {
  int *p =
   (int *)malloc(
     N*sizeof(int));
  return p;
}
```

  - malloc 在記憶體池中找到 size 個位元組大小的未使用記憶體空間，註冊後並回傳指向該空間的位址
  - malloc 配置的記憶體可以存放任意資料型態 (void *)

C 語言中動態記憶體的釋放

- 程式在宣告變數時會依照資料型態自動配置記憶體空間。在變數生命週期結束時，該空間自動的被釋放。但是在使用 **malloc** 這類函式動態配置記憶體後，該空間會一直被佔用直到 **free** 函式被呼叫

```
#include <stdlib.h>
void free(void *ptr);
```

```
#include <stdlib.h>
void func() {
  int *buf =
    (int*)malloc(100*sizeof(int));
  /* Do something */
  free(buf);
  return;
}
```

```
#include <stdlib.h>

void func() {
  int buf[100];
  /* Do something */
  return;
}
```

參考 **malloc_4.cpp**

《範例》動態陣列

- **試寫一程式 (malloc)，讓使用者輸入任意個學生成績，直到使用者輸入負數時停止。輸入完後請印出所有學生的成績。**
  - 提示：這裡我們沒有限制學生個數，無法預設最大個數。我們必須隨著使用者輸入的成績越多，而產生越大的陣列去存。

《範例》儲存字串

- 試寫一程式讓使用者輸入五個名字後，讓使用者輸入編號查詢姓名：

```
請輸入第 1 個名字: Lee
請輸入第 2 個名字: Lin
請輸入第 3 個名字: Huang
請輸入第 4 個名字: Chiang
請輸入第 5 個名字: Wang

請輸入你要查詢的編號: 3
第 3 個名字是 Huang

請輸入你要查詢的編號: 5
第 5 個名字是 Wang
```