

程式設計 (一)

CH8. 陣列

Ming-Hung Wang 王銘宏

tonymhwang@cs.ccu.edu.tw

Department of Computer Science and Information Engineering
National Chung Cheng University

Fall Semester, 2021

本章目錄

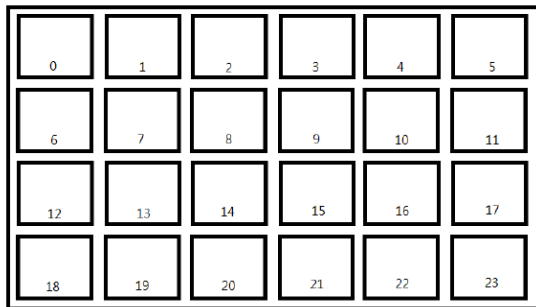
1. 什麼是陣列
2. 定義陣列
3. 使用陣列
4. 傳遞陣列記憶體位址給函式
5. 排序與搜尋
6. 多維陣列

什麼是陣列

陣列 (array)= 具有相同型別的連續的記憶體位置

什麼是陣列

如果把記憶體比喻成有一堆抽屜的一個大櫃子

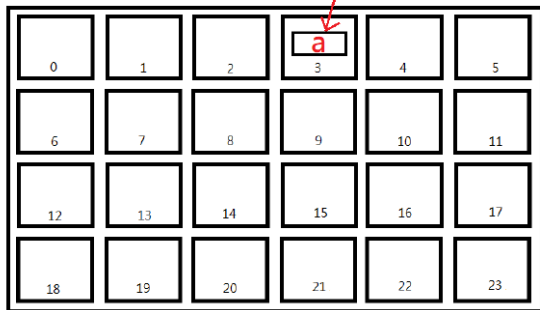


0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

什麼是陣列

「宣告一個變數」可以比喻成
「跟櫃子要了一個抽屜」

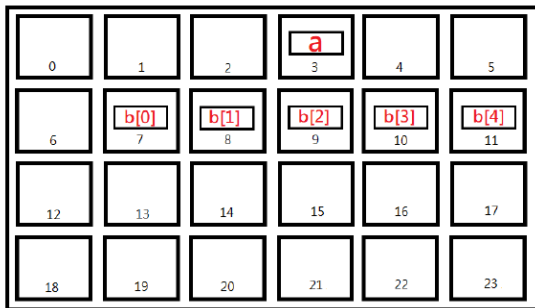
`int a;` 跟櫃子(記憶體)屜要了其中一個抽屜(空間)



什麼是陣列

「宣告一個陣列」就能比喻成「跟櫃子 (記憶體) 要了編號連續的一堆抽屜 (位址連續的空間)」

```
int b[5];
```



定義陣列

定義陣列

宣告陣列格式 (宣告陣列的中括號內不能放變數):
變數型態 陣列名稱 [陣列大小];

- 宣告一個大小為 5 的 int 陣列 a
int a[5];
- 宣告一個大小為 10 的 unsigned char 陣列 b
unsigned char a[10];

定義陣列

- 在宣告時初始化陣列 (給定陣列初始值串列):

int a[5] = {12, 56, 7, 18, -3};

a[0]	a[1]	a[2]	a[3]	a[4]
12	56	7	18	-3

- 給定的陣列大小大於初始化的數值個數，未指定數值的空間會初始化為 0:

int a[8] = {12, 56, 7, 18};

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
12	56	7	18	0	0	0	0

定義陣列

- 在 C99 版本以後的 C，初始值串列可以跳著指定第 N 個元素要被初始化的值

int a[9] = {12, 56, [3] = 18, [5] = 7, 10};

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
12	56	0	18	0	7	10	0	0

定義陣列

- 如果省略陣列大小，則此陣列大小等於初始值串列的元素個數：

int a[] = {12, 56, 7, 18, -3};

a[0]	a[1]	a[2]	a[3]	a[4]
12	56	7	18	-3

int a[] = {12, 56, [3] = 18, [5] = 7, 10};

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
12	56	0	18	0	7	10

定義陣列

注意，只有在宣告陣列時，才能使用大括號初始化陣列。

```
int array[10] = {1, 2, 3, 4, 5};  
int array2[10];  
array2 = {1, 2, 3, 4, 5}; //ERROR!  
//陣列宣告之後不能再如同宣告時使用初始值串列指派
```

使用陣列

使用陣列

陣列是一群具有相同型別的連續記憶體位置。若要引用陣列的某個位置或元素，我們必須指定陣列名稱，以及此元素在陣列中的位置編號 (position number，或叫 index、subscript)。

若有一個陣列 `a` 的大小為 10，代表它有 10 個元素 (elements)，我們可以使用中括號 (`[]`) 來引用陣列裡的元素，例如陣列 `a` 的第 5 個元素寫成 `a[5]`。

注意，每個陣列最開頭的元素均是**第零個元素** (zeroth element)

使用陣列

中括號內的 index 可以是整數或整數變數或運算式：

```
1 // array
2 #include <stdio.h>
3
4 int main() {
5     int n = 3, a[5] = {12, 33, 45, 60, 85};
6     printf("%d\n", a[1]);
7     printf("%d\n", a[n]);
8     printf("%d\n", a[n + 1]);
9 }
10
11 D:\codeblocks\array.exe
33
60
85
```

使用陣列

在宣告時若要將陣列初始化歸零，可給定初始值串列為 {0}：

```
1 // array
2 #include <stdio.h>
3
4 int main() {
5     int a[5] = {0}; //歸零
6     for (int i = 0; i < 5; i++) {
7         printf("a[%d] = %d\n", i, a[i]);
8     }
9 }
10
11 a[0] = 0
12 a[1] = 0
13 a[2] = 0
14 a[3] = 0
15 a[4] = 0
```


使用陣列

若沒有初始化，陣列的元素並不會自動歸零：

```
1 // array
2 #include <stdio.h>
3
4 int main() {
5     int a[5]; //沒歸零
6     for (int i = 0; i < 5; i++) {
7         printf("a[%d] = %d\n", i, a[i]);
8     }
9 }
10
11 a[0] = 6422352
12 a[1] = 4200891
13 a[2] = 4200800
14 a[3] = 0
15 a[4] = 2617344
```

使用迴圈存取陣列

範例：輸入 10 個數值，並倒著輸入順序印出

```
1 // array
2 #include <stdio.h>
3
4 int main() {
5     int a[5]; //沒歸零
6     for (int i = 0; i < 5; i++) {
7         printf("a[%d] = %d\n", i, a[i]);
8     }
9 }
10
11 a[0] = 6422352
12 a[1] = 4200891
13 a[2] = 4200800
14 a[3] = 0
15 a[4] = 2617344
```

使用陣列

範例：輸入 10 個數值，尋找最大數值、最小數值與平均數

```
1 // array
2 #include <stdio.h>
3 #include <limits.h>
4
5 int main() {
6     int a[10] = {0};
7     int min = INT_MAX, max = INT_MIN, sum = 0;
8     for (int i = 0; i < 10; i++){
9         scanf("%d", &a[i]);
10    }
11    for (int i = 0; i < 10; i++) {
12        min = a[i] < min ? a[i] : min;
13        max = a[i] > max ? a[i] : max;
14        sum += a[i];
15    }
16    printf("MIN : %d\tMAX: %d\tVAR: %.2f\n", min, max, (double)sum / 10);
17 }
18
19 D:\codeblocks\array.exe
20 10 55 3 8 23 66 13 9 41 36
    MIN : 3 MAX: 66 VAR: 26.40
```

使用陣列

我們可以使用 `#define` 來定義符號常數，符號常數如同字面常數都可以放入宣告陣列時的中括號中。

```
1 // array
2 #include <stdio.h>
3 #define SIZE 10
4
5 int main() {
6     int a[SIZE] = {0};
7     for (int i = 1; i < SIZE; i++) {
8         a[i] = i + a[i - 1];
9     }
10    printf("%d\n", a[SIZE - 1]);
11 }
12
13 D:\codeblocks\array.exe
45
```

※ 常數的命名只使用大寫和底線

練習：費氏數列

請寫一個程式，隨意輸入 5 個整數 N_0 到 N_4 (範圍在 0 到 59 之間)，計算並印出費氏數列的 $Fib(N_0)$ 到 $Fib(N_4)$ 這 5 項分別是多少，並將計算時間控制在 3 毫秒內

※ 限制：除了第 0 項為 0 與第 1 項為 1 外，都必須靠計算得出

D:\codeblocks\Practice.exe

```
1 8 45 58 50
fib(1) = 1
fib(8) = 21
fib(45) = 1134903170
fib(58) = 591286729879
fib(50) = 12586269025
cost time: 1 ms
```

使用陣列

呼叫兩次 `<time.h>` 的 `clock` 函式的回傳值相減，可獲得 2 次之間的時間差 (單位為毫秒)

```
1  #include <stdio.h>
2  #include <time.h>
3  //You can define functions here if you want
4
5  int main() {
6      int num[5] = { 0 };
7      for (int i = 0; i < 5; i++) {
8          scanf("%d", &num[i]);
9      }
10     double startTime = clock();
11     //Please write your code here
12
13     printf("cost time: %.0f ms\n", clock() - startTim
14 }
```

傳遞陣列記憶體位址給函式

傳遞陣列記憶體位址給函式

在函式章節，我們簡單介紹**傳值呼叫** (call by value) 與**傳參考呼叫** (call by reference)。而到目前為止，我們實作的都是傳值呼叫。
若要將陣列作為函式參數，則為傳參考呼叫。

傳遞陣列記憶體位址給函式

一個簡單的傳遞陣列到函式的程式，
由此還看不出傳參考呼叫的特性。

```
1 // pass array - call by reference
2 #include <stdio.h>
3 #define SIZE 5
4
5 void printfArray(int array[], int size) {
6     for (int i = 0; i < size; i++){
7         printf("%d ", array[i]);
8     }
9 }
10
11 int main() {
12     int score[SIZE] = {80, 85, 76, 90, 88};
13     printfArray(score, SIZE);
14     printf("\n");
15 }
16
```

"D:\codeblocks\pass array - call by reference.exe"

80 85 76 90 88

傳遞陣列記憶體位址給函式

若在傳參考呼叫的函式中改變陣列中的數值，則會反映到原本的陣列中。這是因為我們傳入的引數是陣列的記憶體位址，不是一般的數值。

```
1 // pass array - call by reference
2 #include <stdio.h>
3 #define SIZE 5
4
5 void arrayAdding(int array[], int size, int add) {
6     for (int i = 0; i < size; i++){
7         array[i] += add;
8     }
9 }
10
11 int main() {
12     int score[SIZE] = {80, 85, 76, 90, 88};
13     arrayAdding(score, SIZE, 10);
14     for(int i = 0; i < SIZE; i++)
15         printf("%d ", score[i]);
16     printf("\n");
17 }
18 "D:\codeblocks\pass array - call by reference.exe"
90 95 86 100 98
```

傳遞陣列記憶體位址給函式

在函式的引數中，我們填入了陣列名稱 `score`。若實際將 `score` 印出來，會發現 `score` 的值跟 `&score[0]` (`score` 的第 0 個元素的位址) 一樣。由此，我們可以知道我們是將陣列的開頭位址傳給了函式。

```
1 // address of array
2 #include <stdio.h>
3 #define SIZE 5
4
5 int main() {
6     int score[SIZE] = {80, 85, 76, 90, 88};
7     printf("score      = %p\n", score);
8     printf("&score[0] = %p\n", &score[0]);
9 }
10
```

"D:\codeblocks\address of array.exe"

```
score      = 0061FF0C
&score[0] = 0061FF0C
```

傳遞陣列記憶體位址給函式

傳參考呼叫函式中的陣列位址與傳入的陣列的位址是同一個，代表兩者使用的記憶體空間是同一個，所以在傳參考呼叫函式中若改變陣列的數值，在原本的陣列也會一起改變。

傳遞陣列記憶體位址給函式

以下比較傳遞整個陣列與傳遞單一陣列元素

- 傳參考呼叫的陣列參數位址與引數是相同的
- 傳值呼叫的參數位址與引數是不相同的

```
1 // address of array
2 #include <stdio.h>
3 #define SIZE 5
4
5 void callByReference(int array[], int size) {
6     printf("array = %p\t", array);
7     printf("&array[0] = %p\n", &array[0]);
8 }
9
10 int main() {
11     int score[SIZE] = {80, 85, 76, 90, 88};
12     printf("score = %p\t", score);
13     printf("&score[0] = %p\n", &score[0]);
14     callByReference(score, SIZE);
15 }
```

"D:\codeblocks\address of array.exe"

score	= 0061FF0C	&score[0]	= 0061FF0C
array	= 0061FF0C	&array[0]	= 0061FF0C

```
1 // address of array
2 #include <stdio.h>
3 #define SIZE 5
4
5 void callByValue(int value) {
6     printf("&value = %p\n", &value);
7 }
8
9 int main() {
10     int score[SIZE] = {80, 85, 76, 90, 88};
11     printf("&score[0] = %p\n", &score[0]);
12     callByValue(score[0]);
13 }
```

"D:\codeblocks\address of array.exe"

&score[0]	= 0061FF0C
&value	= 0061FEF0

傳遞陣列記憶體位址給函式

我們可以將陣列的所有元素的位址列出來觀察

```
1 // address of array
2 #include <stdio.h>
3 #define SIZE 5
4
5 int main() {
6     int score[SIZE] = {80, 85, 76, 90, 88};
7     for (int i = 0; i < SIZE; i++) {
8         printf("&score[%d] = %p\t", i, &score[i]);
9         printf("score + %d = %p\n", i, score + i);
10    }
11 }
```

12 "D:\codeblocks\address of array.exe"

```
&score[0] = 0061FF08    score + 0 = 0061FF08
&score[1] = 0061FF0C    score + 1 = 0061FF0C
&score[2] = 0061FF10    score + 2 = 0061FF10
&score[3] = 0061FF14    score + 3 = 0061FF14
&score[4] = 0061FF18    score + 4 = 0061FF18
```

傳遞陣列記憶體位址給函式

依照上頁的結果，陣列元素的位址 `&score[n]` 會等於 `score+n`，並且 `+n` 並不是將記憶體位置加上 `n` 個 bytes，而是依據元素的型別大小為單位來做加法。

所以，因為 `score` 是 `int` 陣列，`score+n` 實際上是 `score` 的值 (陣列起始位址) 加上 `4*n` bytes。

傳遞陣列記憶體位址給函式

將陣列從中間部分開始傳入：

```
1 // address of array
2 #include <stdio.h>
3 #define SIZE 5
4
5 int printMark(int num[], int size) {
6     for (int i = 0; i < size; i++)
7         printf("%d ", num[i]);
8     printf("\n");
9 }
10
11 int main() {
12     int num[SIZE] = {5, 4, 3, 2, 1};
13     for (int i = SIZE - 1; i >= 0; i--)
14         printMark(num + i, SIZE - i); //num + i可替換成&num[i]
15 }
```

"D:\codeblocks\address of array.exe"

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```


傳遞陣列記憶體位址給函式

參數使用 `const` 修飾字

在第三章我們有提到在宣告變數時加上 `const` 修飾字，會讓變數變成無法修改的常數。同樣的，若將參數的宣告加上 `const`，可以防止在該函式內修改到該參數的值。

```
void printArray(const int array[], int size) {  
    for (int i = 0; i < size; i++) {  
        printf("%d ", array[i]);  
    }  
}
```

傳遞陣列記憶體位址給函式

若嘗試修改 `const` 變數或陣列，會造成編譯錯誤

```
1 // address of array
2 #include <stdio.h>
3 #define SIZE 5
4
5 void printArray(const int array[], int size) {
6     for (int i = 0; i < size; i++) {
7         printf("%d ", ++array[i]);
8     }
9 }
10
11 int main() {
12     int score[SIZE] = {80, 85, 76, 90, 88};
13     printArray(score, SIZE);
14     printf("\n");
15 }
16
17 Logs & others
18 Build messages x
19 File Line Message
20 D:\codeblocks... 7 error: increment of read-only location '*(array + (sizetype)((unsigned int)i * 4u))'
```

排序與搜尋

氣泡排序法 (bubble sort)

排序 (Sorting) 資料 (也就是照特定的順序放置資料，例如遞增或遞減順序) 是電腦最重要的應用之一。而氣泡排序法 (bubble sort) 是眾多排序法之中最容易理解的一個。

42 30 7 2 5

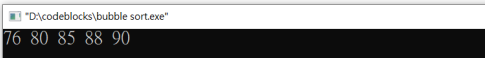
排序與搜尋

42 30 7 2 5

從動畫可以發現, 氣泡排序法排序大小為 N 的陣列時, 首先比較第 0 項與第 1 項, 接著比較第 1 項與第 2 項, 直到比較完第 $N-1$ 項與第 N 項時, 才結束第一回合。接著第 2 回合之後, 每回合都比前一回合少比較 1 項, 直到無法比較為止。而且, 若某回合需要比較 X 項時, 該回合總共只比較了 $X-1$ 次。

氣泡排序法範例程式：

```
1 // bubble sort
2 #include <stdio.h>
3 #define SIZE 5
4
5 int main() {
6     int score[SIZE] = {80, 85, 76, 90, 88}, tmp;
7     for (int i = 0; i < SIZE - 1; i++){ //氣泡排序
8         for (int j = 0; j < SIZE - i - 1; j++){ //比較(SIZE - i)項，(SIZE - i - 1)次
9             if (score[j] > score[j + 1]){ //注意j不要寫成i
10                 tmp = score[j];
11                 score[j] = score[j + 1];
12                 score[j + 1] = tmp;
13             }
14         }
15     }
16     for (int i = 0; i < SIZE; i++) {
17         printf("%d ", score[i]);
18     }
19     printf("\n");
20 }
```



線性搜尋法 (linear search)

有時我們會想要知道陣列中是否有一個符合某個關鍵值 (key value) 的數值。找出陣列中某個元素的過程稱為搜尋 (searching)。線性搜尋法是最直接暴力的搜尋法, 在搜尋未經排序過的資料時, 可使用線性搜尋法。

線性搜尋法動畫：

search : 2

30 42 7 2 5

線性搜尋法範例程式：

```
1 // linear search
2 #include <stdio.h>
3 #define SIZE 5
4
5 int main() {
6     int score[SIZE] = {80, 85, 76, 90, 88};
7     int key = 76, index = -1;
8     for (int i = 0; i < SIZE; i++){ //線性搜尋法
9         if (score[i] == key) {
10             index = i;
11             break;
12         }
13     }
14     if (index != -1)
15         printf("found %d at score[%d]: %d\n", key, index, score[index]);
16     else
17         printf("not found\n");
18 }
19
```

D:\codeblocks\linear search.exe

found 76 at score[2]: 76

二元搜尋法 (binary search)

對於排序過的陣列，我們可以使用二元搜尋法會比線性搜尋法來的更加快速。

二元搜尋法的概念是，每個回合先將數列的中央數字與目標數字相比，若中央數字不等於目標數字，則保留目標數值可能會存在的那半邊，下個回合使用上個回合所保留的區間繼續縮小範圍。

search : 42

2	4	7	10	20	33	42	56	78	90
---	---	---	----	----	----	----	----	----	----

二元搜尋法範例程式 (1/2 - 副程式):

```
1  // binary search
2  #include <stdio.h>
3  #define SIZE 10
4
5  int binarySearch(int array[], int size, int key) {
6      int left = 0, right = size, mid;
7      while (left < right) {
8          mid = (left + right) / 2;
9          if (array[mid] < key)
10             left = mid + 1;
11         else if (array[mid] > key)
12             right = mid;
13         else
14             return mid;
15     }
16     return -1;
17 }
```

二元搜尋法範例程式 (2/2 - 主程式):

```
19 int main() {  
20     int score[SIZE] = {45, 50, 66, 76, 78, 80, 85, 88, 90, 95};  
21     int key = 88, index;  
22     index = binarySearch(score, SIZE, key);  
23     if (index != -1)  
24         printf("found %d at score[%d]: %d\n", key, index, score[index]);  
25     else  
26         printf("not found\n");  
27 }  
28
```

"D:\codeblocks\binary search.exe"
found 88 at score[7]: 88

二元搜尋法有很多種寫法，試試看使用遞迴函式寫出二元搜尋法。

多維陣列

多維陣列

多維陣列 (multidimensional arrays) 能夠有兩個以上的下標 (subscript, 或稱 index)。

使用兩個下標的陣列, 稱為二維陣列 (double-subscripted arrays), 經常會用來表示表格 (tables), 其數值是依照列 (rows) 和行 (columns) 排列的資料所組成。

多維陣列

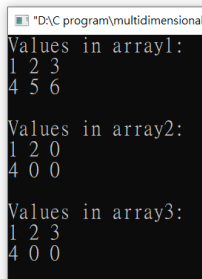
二維陣列的第一個下標代表列, 第二個下標代表行

	第0行	第1行	第2行	第3行
第0列	a[0][0]	a[0][1]	a[0][2]	a[0][3]
第1列	a[1][0]	a[1][1]	a[1][2]	a[1][3]
第2列	a[2][0]	a[2][1]	a[2][2]	a[2][3]

多維陣列的宣告與使用

多維陣列可以和一維陣列一樣，在宣告時指定其初始值。

```
1 // multidimensional array
2 #include <stdio.h>
3
4 int printArray(int array[][3], int r_size, int c_size, int n);
5
6 int main() {
7     int array1[2][3] = {{1, 2, 3}, {4, 5, 6}};
8     //未指定數值的元素會初始化為0
9     int array2[2][3] = {{1, 2}, {4}};
10    //使用一維初始值串列會照順序填入
11    int array3[2][3] = {1, 2, 3, 4};
12    printArray(array1, 2, 3, 1);
13    printArray(array2, 2, 3, 2);
14    printArray(array3, 2, 3, 3);
15 }
```



```
"D:\C program\multidimensional
Values in array1:
1 2 3
4 5 6

Values in array2:
1 2 0
4 0 0

Values in array3:
1 2 3
4 0 0
```

(接續上一頁程式碼)
使用巢狀迴圈將二維陣列印出的函式：

```
17 int printArray(int array[][3], int r_size, int c_size, int n) {  
18     printf("Values in array%d:\n", n);  
19     for (int i = 0; i < r_size; i++){  
20         for (int j = 0; j < c_size; j++) {  
21             printf("%d |", array[i][j]);  
22         }  
23         printf("\n");  
24     }  
25     printf("\n");  
26 }
```

多維陣列

多維陣列的宣告，除了第一維以外的維度，都一定要指定大小

```
int array2_1[][3] = {{1, 2, 3}, {4, 5, 6}};  
int array2_2[][] = {{1, 2, 3}, {4, 5, 6}}; //ERROR  
  
int array3_1[][3][3] = {{{1, 2, 3}, {4, 5, 6}},  
                           {{6, 5, 4}, {3, 2, 1}}};  
int array3_2[][][3] = {{{1, 2, 3}, {4, 5, 6}}, //ERROR  
                        {{6, 5, 4}, {3, 2, 1}}};  
int array3_2[3][3][] = {{{1, 2, 3}, {4, 5, 6}}, //ERROR  
                        {{6, 5, 4}, {3, 2, 1}}};
```

多維陣列元素的記憶體位址

多維陣列的記憶體空間也是連續的。

第二行以後的每行第一個元素的位址剛好接續在上一行的最後一個元素位址後面。

```
1 // multidimensional array
2 #include <stdio.h>
3
4 int main() {
5     int array[3][3];
6     for (int i = 0; i < 3; i++){
7         for (int j = 0; j < 3; j++){
8             printf("%p ", &array[i][j]);
9         }
10        printf("\n");
11    }
12 }
```


"D:\C program\multidimensional array.exe"

```
0061FEF4 0061FEF8 0061FEFC
0061FF00 0061FF04 0061FF08
0061FF0C 0061FF10 0061FF14
```

(補充) 多維陣列位址加法

多維陣列的位址加法是以自己小一個維度的大小為單位

```
1 // multidimensional array
2 #include <stdio.h>
3
4 int main() {
5     int array[3][5];
6     printf("&array[1][0]: \n");
7     printf("%p\n", &array[1][0]);
8     printf("%p\n", array + 1);
9     printf("\n");
10    printf("&array[2][3]: \n");
11    printf("%p\n", &array[2][3]);
12    printf("%p\n", array[2] + 3);
13 }
```



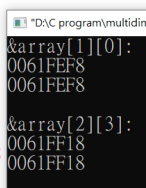
```
"D:\C program\multidin
&array[1][0]:
0061FEF8
0061FEF8

&array[2][3]:
0061FF18
0061FF18
```

多維陣列

如果要讓單位強制變成一個元素的大小, 需要用到
第 11 章才會教到的指標型態

```
1 // multidimensional array
2 #include <stdio.h>
3
4 int main() {
5     int array[3][5];
6     printf("&array[1][0]: \n");
7     printf("%p\n", &array[1][0]);
8     printf("%p\n", (int *)array + 1 * 5);
9     printf("\n");
10    printf("&array[2][3]: \n");
11    printf("%p\n", &array[2][3]);
12    printf("%p\n", (int *)array + 2 * 5 + 3);
13 }
```



```
"D:\C program\multidin
&array[1][0]:
0061FEF8
0061FEF8

&array[2][3]:
0061FF18
0061FF18
```

參考資料： Deitel, H. M., & Deitel, P. J. (2015). C: How to program.
Upper Saddle River, N.J: Prentice Hall.