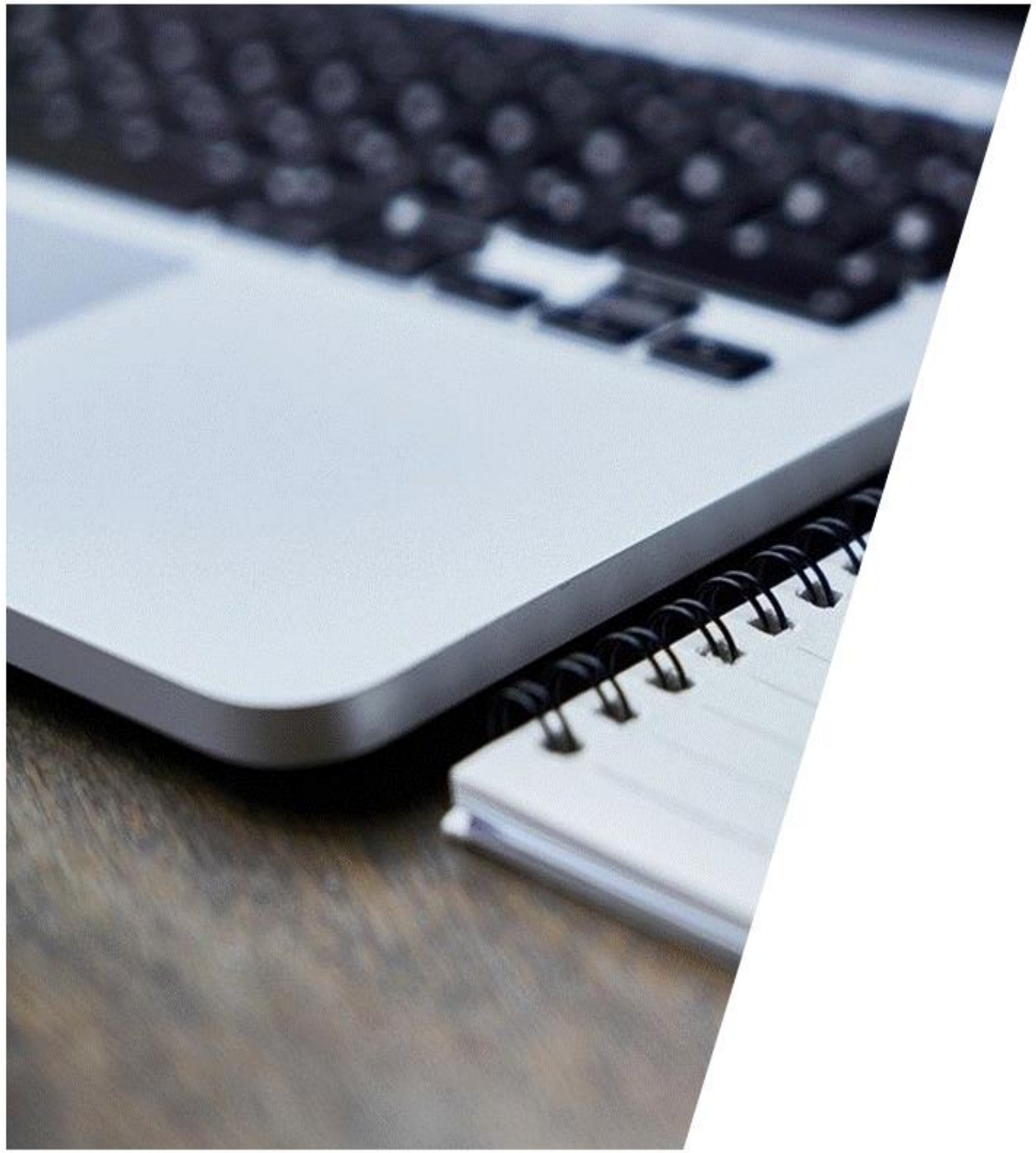


C++

函式傳遞陣列

函式傳遞基本資料型態變數

```
#include <iostream>
using namespace std;

float area3(float r,float h){
    return 3.14*r*r*h;
}

int main(){
    float r,h;
    cin >> r >> h;
    cout << "圓柱體體積" << area3(r,h) << endl;
    return 0;
}
```

函式參數必須提供資料類型以及變數名稱

傳遞陣列內的單一元素

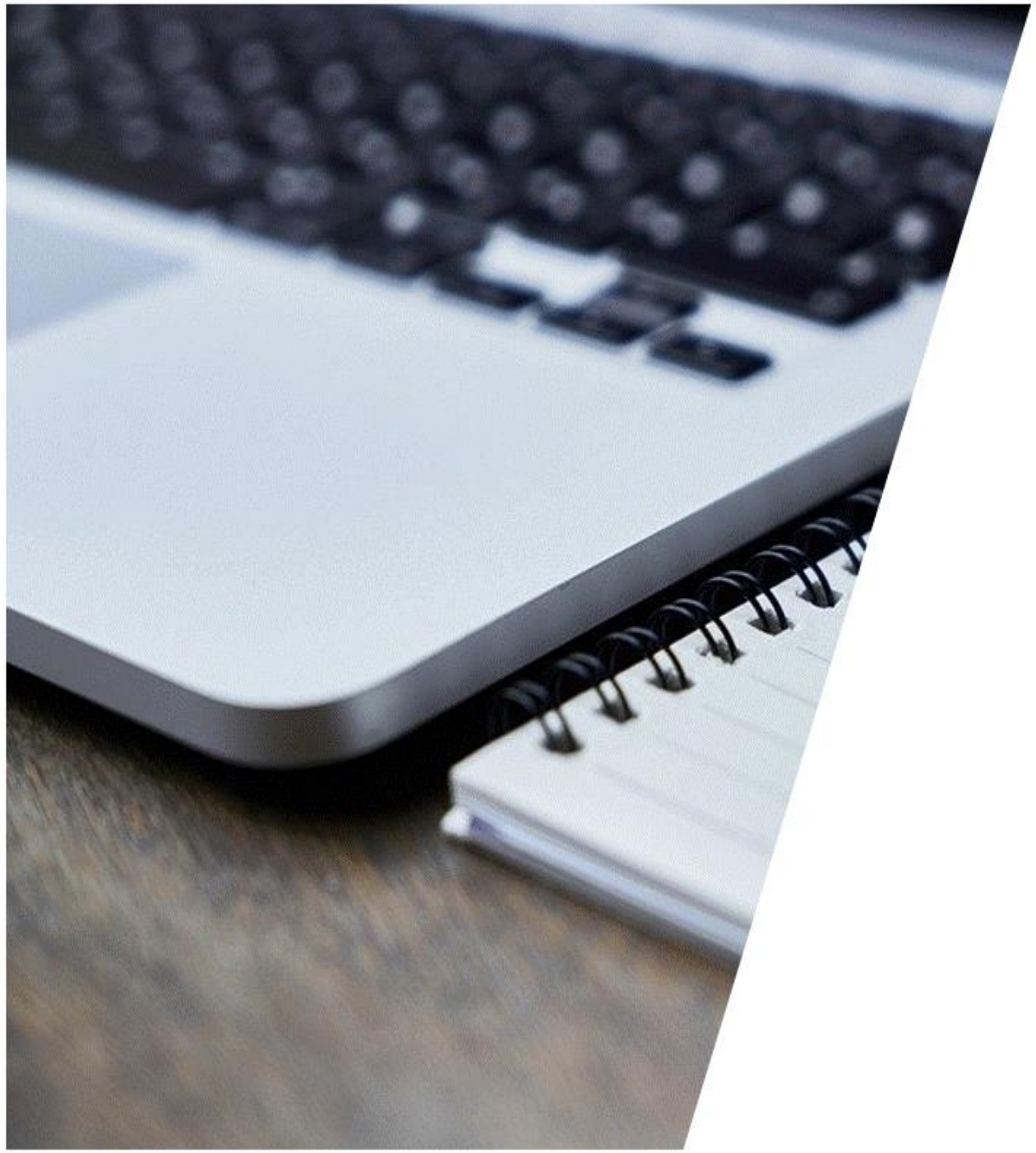
```cpp
#include <iostream>
using namespace std;

void out(int value) {
    cout << value << endl;
}

int main(){
    int list[3] = {10,20,30};
    int i;
    for(i=0;i<3;i++){
        out(list[i]);
    }
    return 0;
}
```

參數的資料型態必須與陣列的資料型態一致

傳遞陣列元素時，使用 **陣列變數[索引值]** 的方式指定要傳遞的陣列元素

傳遞陣列內的單一元素

- 陣列單一元素內的值會複製到函式內的接收參數變數內，操作就如同傳遞基本資料型態變數一樣。
- 參數的資料型態必須與陣列的資料型態一致。
- 函式接收陣列單一元素值後，函式內對接收變數的任何操作都不會影響到傳遞方的陣列內容值。
- 是一種**傳值呼叫（call by value）**：只傳送變數的值給函式，傳遞後，傳遞方與接收方兩邊的變數就互不影響。



C++

函式傳遞陣列

傳遞陣列：第一種寫法

```
#include <iostream>
using namespace std;

int add(int list[5]){
    int i=0, sum=0;
    for (i=0; i<5; i++) {
        sum = sum + list[i];
    }
    return sum;
}

int main(){
    int list[5] = {1, 2, 3, 4, 5};
    cout << add(list);
    return 0;
}
```

與傳遞一般變數相似，資料型態必須一致，也要是同樣長度的陣列型態。

傳遞陣列：第二種寫法

```cpp
#include <iostream>
using namespace std;

int add(int list[]){
    int i=0, sum=0;
    for (i=0; i<5; i++) {
        sum = sum + list[i];
    }
    return sum;
}

int main(){
    int list[5] = {1, 2, 3, 4, 5};
    cout << add(list);
    return 0;
}
```

參數變數不指定
陣列大小也可以

傳遞陣列：第三種寫法

```
#include <iostream>
using namespace std;

int add(int list[], int num){
    int i=0, sum=0;
    for (i=0; i<num; i++) {
        sum = sum + list[i];
    }
    return sum;
}

int main(){
    int list[5] = {1, 2, 3, 4, 5};
    cout << add(list, 5);
    return 0;
}
```

除了傳遞陣列變數本身之外，額外再傳遞陣列長度

若有參數傳遞陣列大小，函式內就可以透過變數控制陣列存取

傳遞陣列：傳址呼叫

函式參數	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
陣列內容 (同一個位址)	10	20	30	40	50	60	70
傳送方(呼叫方)	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]

- 函式傳遞陣列參數時，只會傳遞陣列第一個元素的位址，不會將整個陣列內的各個元素值複製一份給函式，因此函式陣列參數可以不設定陣列大小。
- 陣列只有傳遞位址給函式，所以函式內仍是存取呼叫方使用的陣列空間，不會在函式內額外使用另一個陣列空間。
- 是一種**傳址呼叫（call by address）**：傳送變數的位址給函式，傳遞後，函式內若有更改陣列的內容值，呼叫方也會拿到更改後的值．

傳遞陣列：提供陣列大小

- 因為陣列傳遞是一種傳址呼叫（call by address），可以不設定陣列大小，但函式內存取陣列時，仍必須注意不可存取超出陣列範圍。

- 如何避免存取超出陣列範圍？
  ① 在函式內使用固定的數字，如: while(i<5)
  ② 傳遞一個紀錄陣列長度的變數給函式，如: while(i<num)

- 使用固定數字控制陣列範圍會降低程式的可用性，透過參數傳遞方式就可以提升程式碼的使用彈性。

傳遞陣列：三種寫法

第一種寫法：

```
函釋回傳型態　函式名稱(陣列型態 陣列名稱[陣列大小])
```

第二種寫法：

```
函釋回傳型態　函式名稱(陣列型態 陣列名稱[])
```

第三種寫法：

```
函釋回傳型態　函式名稱(陣列型態 陣列名稱[], int 陣列長度變數)
```

函式傳遞參數

- 傳遞基本資料型態變數：傳值呼叫 call by value
  - 傳送方與接收方各自使用不同空間
  - 函式內更動參數內容值，呼叫方不會知道

- 傳遞陣列變數：傳址呼叫 call by address
  - 傳送方與接收方共用同一個內容
  - 函式內更動陣列參數內容值，呼叫方會知道

C++

函式傳遞多維陣列

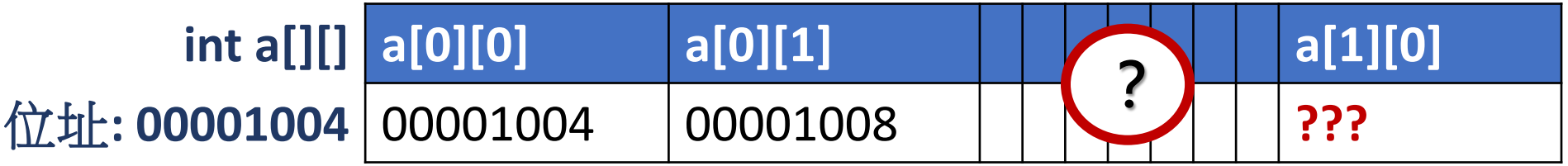
一維陣列長度

- 一維陣列：可以透過陣列的資料型態，知道陣列內的第 N 個元素位址會是陣列起始位址往後移多少空間，因此函式參數可以不用設定一維陣列的陣列大小。

int a[]	a[0]	a[1]	a[2]	a[3]
位址: **00001004**	00001004	00001008	00001012	00001016

二維陣列長度

- 二維陣列：陣列每一列的空間大小會根據陣列內層的長度而定，因此函式傳遞二維陣列時，必須設定內層的陣列大小。
  - 每一列空間大小：內層陣列長度 * 陣列資料型態



傳遞二維陣列

```
#include <iostream>
using namespace std;

int add(int list[][3], int rows, int cols){
    int i, j, sum=0;
    for (i=0; i<rows; i++) {
        for (j=0; j<cols; j++) {
            sum = sum + list[i][j];
        }
    }
    return sum;
}

int main(){
    int list[2][3] = {{1,2,3}, {9,8,7}};
    cout << add(list, 2, 3);
    return 0;
}
```

二維陣列參數至少要設定第二層陣列的長度大小

多維陣列長度

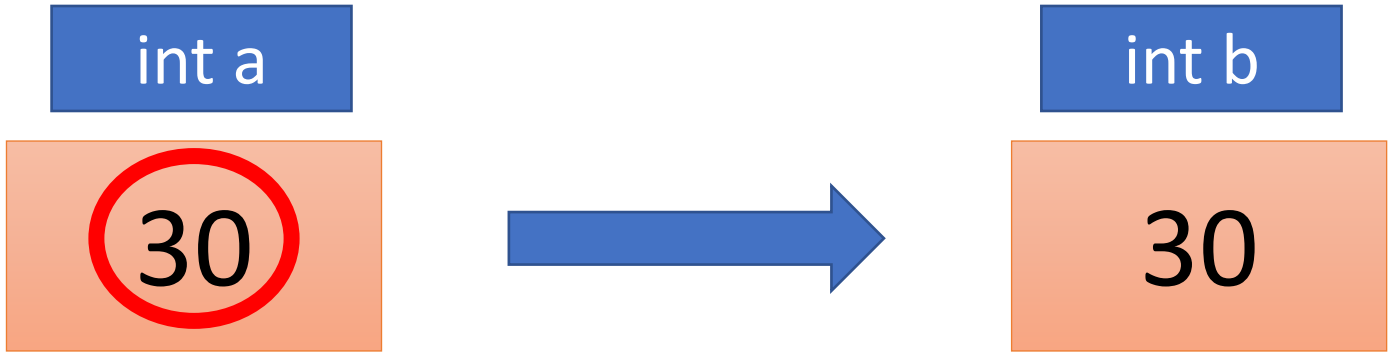
```
#include <iostream>
using namespace std;

int add(int list[][3][2], int num1, int num2, int num3){
    int i, j, k, sum=0;
    for (i=0; i<num1; i++) {
        for (j=0; j<num2; j++) {
            for (k=0; k<num3; k++) {
                sum = sum + list[i][j][k];
            }
        }
    }
    return sum;
}

int main(){
    int list[2][3][2] = {{{1,1},{2,2},{3,3}}, {{11,11},{22,22},{33,33}}};
    cout << add(list, 2, 3, 2);
    return 0;
}
```

為了可以計算出陣列內各個元素的位址，傳遞多維陣列時，只有最外層可以不用設定長度。

C++

傳值呼叫與傳址呼叫

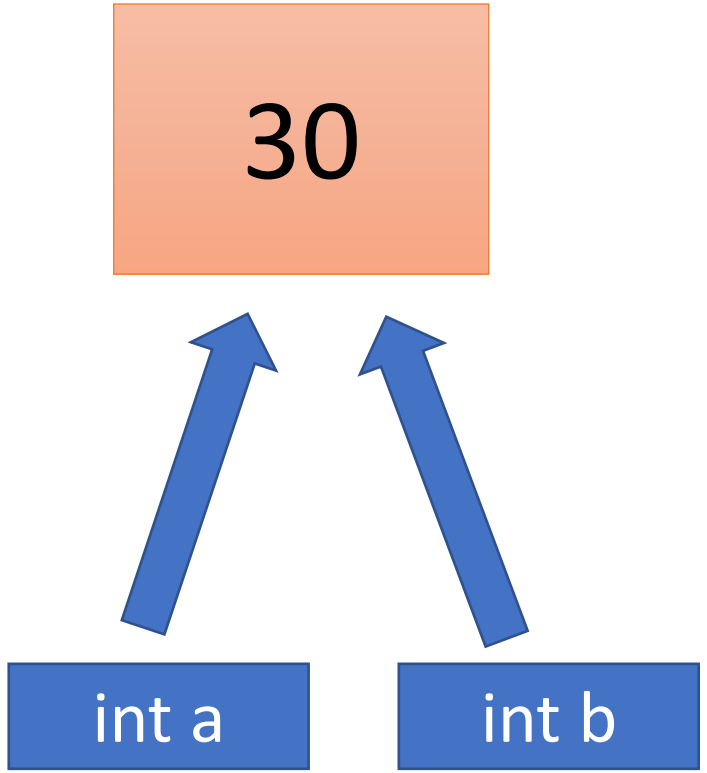
傳值呼叫 call by value



- 是一種**單向**的傳遞
- 將傳送方的值複製給接收方，複製後，兩方就各自操作，不會相互影響。
- 傳送方與接收方各自有獨立的空間，也就是說會使用到兩個同樣大小的空間來各別儲存。
- 變數a 內的值 30 複製給變數b 後，不論對變數a 執行任何加減乘除等運算都不會影響到變數b，同樣的，變數b執行任何加減乘除等運算也都不會影響到變數a。

傳值呼叫 call by value

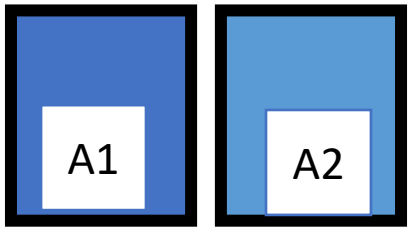
- 會產生**雙向**的影響
- 將傳送方存放的地址傳給接收方，接收方儲存那份地址。
- 傳送方與接收方其實都是儲存同一個地址，所以操作時會是操作這個地址內的資料，因此不論哪一方更動到地址內部的資料，都會影響到另一方拿到的資料內容。
- 變數a 存放的位址傳給變數b 後，如果變數a 對資料執行任何加減乘除都會讓變數b拿到被更改後的資料，同樣的，變數b執行任何加減乘除等運算也會讓變數a拿到更動後的資料。



傳值與傳址

傳值呼叫 CALL BY VALUE

- 單向的傳遞
- 各自使用獨立空間的兩個變數
- 兩棟空間完全一樣的獨立房子，分別有有各自的門牌號碼，打開不同的門就進入不同的房子。



傳址呼叫 CALL BY ADDRESS

- 雙向的影響
- 通向同一個空間的兩個變數
- 一棟空間很大的房子，使用了兩個門牌號碼，但房子內部空間是打通的，不論進入哪個門牌號碼，都是走進同一棟房子。