C++程式設計：指標

**目錄**

[C++程式設計：遞迴 1](#_Toc204799122)

[1.1 範例：費波那契數列（Fibonacci） **錯誤! 尚未定義書籤。**](#_Toc204799123)

[1.2 範例：最大公因數（GCD） **錯誤! 尚未定義書籤。**](#_Toc204799124)

[1.3 範例：河內塔（Tower of Hanoi） **錯誤! 尚未定義書籤。**](#_Toc204799125)

# 基本概念

指標（Pointer）是一種變數，用來儲存另一個變數的記憶體位址。我們先來複習複習。當我們跟記憶體宣告一個變數後，會有以下三個東西可以使用：

1. 空間(存取變數)
2. 變數名稱(例如 成功大學)
3. 空間地址(例如 台南市東區大學路1號)

我們會有兩個常用的運算子：

* 解參考運算子（\*）：用來宣告指標變數，或用來存取某個位址所儲存的值。
* 位址運算子（&）用來取得變數的位址，或把某變數的位址傳給指標使用。

## 範例：印出地址

#include <iostream>

using namespace std;

int main() {

// int\* ptr 宣告一個指標

// &m：取得變數m的地址

    int m = 100;

    int\* ptr = &m;

    cout << "Value of Variable m is: " << m << endl; // 100

    cout << "Memory Address of Variable m is: " << &m << endl; // 0x7ffffcbf4

    cout << "Memory Address of Variable m is using ptr: " << ptr << endl; // 0x7ffffcbf4

    return 0;

}

## 範例：印出指標值與地址

接著，我們示範將int\* ptr = &m;拆成兩步：

#include <iostream>

using namespace std;

int main() {

    int a = 10;

    int\* ptr;

    ptr = &a;

    cout << "ptr = " << ptr << endl; // 0x7ffffcbf4

    cout << "\*ptr = " << \*ptr << endl; // 10

    return 0;

}

# 宣告指標

當我們宣告完指標後，是無法更改指標所指向的data type。宣告指標方式如下：

int \*x;

int \*x, \*y;

int \*y=NULL;

float \*fptr;

double \*dptr;

## 範例：查看不同型態與指標大小

這個範例示範不同data type類型的指標

#include <iostream>

using namespace std;

int main() {

    int a1 = 40, a2 = 50, a3 = 60;

    int\* p1 = &a1;

    int\* p2 = &a2;

    int\* p3 = &a3;

    cout << p1 << "\t" << \*p1 << endl; // 0x7ffffcbe4     40

    cout << p2 << "\t" << \*p2 << endl; // 0x7ffffcbe0     50

    cout << p3 << "\t" << \*p3 << endl; // 0x7ffffcbdc     60

    // 查看不同型態的指標大小

    int\* iptr;

    float\* fptr;

    double\* dptr;

    cout << "sizeof(iptr) = " << sizeof(iptr) << endl; // sizeof(iptr) = 8

    cout << "sizeof(fptr) = " << sizeof(fptr) << endl; // sizeof(fptr) = 8

    cout << "sizeof(dptr) = " << sizeof(dptr) << endl; // sizeof(dptr) = 8

    return 0;

}

// 雖然他們是不同值，但因為指標型態都是int，因此指標的資料型態都是int -> 8

# Integer Pointers(整數指標)

## 範例：call by value（傳值，無法交換）

#include <iostream>

using namespace std;

void swap(int a, int b) {

    int temp = a;

    a = b;

    b = temp;

}

int main() {

    int i = 1;

    int j = 2;

    swap(i, j);

    cout << "i = " << i << ", j = " << j << endl; // i = 1, j = 2

    return 0;

}

## 範例：Call by Pointer（透過指標交換）

#include <iostream>

using namespace std;

void swap(int\* a, int\* b) {

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int main() {

    int i = 1;

    int j = 2;

    swap(&i, &j);

    cout << "i = " << i << ", j = " << j << endl; // i = 2, j = 1

    return 0;

}

# 元素指標（Element Pointer）

元素指標（Element Pointer）是指陣列第0個元素的地址(也就是整個陣列的統稱地址)

當你寫int \*ptr = arr;，代表 ptr 指向arr[0]，也就是陣列的第 0 個元素。

#include <iostream>

using namespace std;

int main() {

    int\* ptr; // 宣告整數指針

    int arr[5] = {1, 2, 3, 4, 5};

    // 正確寫法

    ptr = arr;  // ptr 指向 arr[0]

    cout << ptr << endl; // 0x7ffffcbe0

    // 正確寫法

    ptr = &arr[0];

    cout << ptr << endl; // 0x7ffffcbe0

    // 錯誤寫法

    // ptr = &arr;  // &arr是「指向整個陣列」的指標，而上面都是「指向單一 int 元素」的指標

    int (\*arrayPtr)[5] = &arr;  // 若你真的需要取得整個陣列的指標，則必須使用正確的型別

    cout << arrayPtr << endl; // 0x7ffffcbe0

    return 0;

}

這邊就可以來個小整理：

|  |  |
| --- | --- |
| int\* ptr = arr; | 指向 arr[0] 的指標（陣列退化） |
| int\* ptr = &arr[0]; | 明確取得第一個元素的指標 |
| int (\*arrayPtr)[5] = &arr; | 指向整個陣列的指標 |

至於我們在範例中，會發現說，其實三者輸出的值都是會一樣。但它們的型別和語意是不一樣的。後面我們會在位移運算中再來討論一次~

# 陣列位移

這邊我們將【取值/地址】跟【陣列位移】整理在一起做討論。

## 範例：存取陣列元素的位址

我們宣告int\* ptr;

* ptr + i代表陣列第i 個元素的位址。
* &ptr[i]則是一種語法糖(這邊可以理解為ptr[i]是陣列取元素，那&ptr[i]就是陣列取元素取地址)

#include <iostream>

using namespace std;

int main() {

    int arr[5] = {1, 2, 3, 4, 5};

    int\* ptr = arr; // 指向 arr[0]

    // int\* ptr = &arr[0]; // 也可以

    // 指標運算

    cout << "Pointer arithmetic (ptr + i):\n";

    for (int i = 0; i < 5; ++i) {

        cout << (ptr + i) << '\n';

    }

    // 語法糖：&ptr[i]

    cout << "\nAddress using &ptr[i]:\n";

    for (int i = 0; i < 5; ++i) {

        cout << &ptr[i] << '\n';

    }

    return 0;

}

// Pointer arithmetic (ptr + i):

// 0x7ffffcbd0

// 0x7ffffcbd4

// 0x7ffffcbd8

// 0x7ffffcbdc

// 0x7ffffcbe0

// Address using &ptr[i]:

// 0x7ffffcbd0

// 0x7ffffcbd4

// 0x7ffffcbd8

// 0x7ffffcbdc

// 0x7ffffcbe0

## 範例：存取陣列元素的值

我們宣告int\* ptr;

* \*(ptr + i)代表陣列第i 個元素的位址的值。
* ptr[i] 則是一種語法糖(這邊可以理解成，ptr就是陣列，ptr[i]就是陣列取值)

#include <iostream>

using namespace std;

int main() {

    int arr[5] = {1, 2, 3, 4, 5};

    int\* ptr = arr;

    // int\* ptr = &arr[0]; // 也可以

    // 指標算術

    cout << "Values via \*(ptr + i): ";

    for (int i = 0; i < 5; ++i) {

        cout << \*(ptr + i) << " "; // 1 2 3 4 5

    }

    cout << "\n";

    // 語法糖 ptr[i]

    cout << "Values via ptr[i]:     ";

    for (int i = 0; i < 5; ++i) {

        cout << ptr[i] << " "; // 1 2 3 4 5

    }

    cout << "\n";

    return 0;

}

# 陣列與指標作為函數參數做傳遞

## 範例：將陣列作為函數參數傳遞

#include <iostream>

using namespace std;

// 這邊都是用語法糖

void f1(int arr[]) {

    // 位移運算

    for (int i = 0; i < 5; ++i) {

        cout << arr+i << " "; // 取地址

    }

    cout << "\n";

    for (int i = 0; i < 5; ++i) {

        cout << \*(arr+i) << " "; // 取值

    }

    cout << "\n";

    // 語法糖

    for (int i = 0; i < 5; ++i) {

        cout << arr[i] << " "; // 取值

    }

    cout << "\n";

    for (int i = 0; i < 5; ++i) {

        cout << &arr[i] << " "; // 取地址

    }

    cout << "\n";

}

void f2(int\* arr) {

    // 位移運算

    for (int i = 0; i < 5; ++i) {

        cout << arr + i << " "; // 取地址

    }

    cout << "\n";

    for (int i = 0; i < 5; ++i) {

        cout << \*(arr + i) << " "; // 取值

    }

    cout << "\n";

    // 語法糖

    for (int i = 0; i < 5; ++i) {

        cout << arr[i] << " "; // 取值

    }

    cout << "\n";

    for (int i = 0; i < 5; ++i) {

        cout << &arr[i] << " "; // 取地址

    }

    cout << "\n";

}

int main() {

    int arr[5] = {1, 2, 3, 4, 5};

    f1(arr);

    cout << "========================\n";

    f2(arr);

    return 0;

}

## 範例：將陣列作為函數參數傳遞(較正式版)

我們有提過，通常傳陣列會傳兩格東西：

1. 陣列名稱(同時也代表陣列本身地址，也代表陣列第0個元素的地址)。
2. 陣列長度

#include <iostream>

using namespace std;

// 使用陣列語法

void printArray01(int arr[], int size) {

    // 邏輯位移

    for (int i = 0; i < size; ++i) {

        cout << arr + i << " ";

    }

    cout << "\n";

    for (int i = 0; i < size; ++i) {

        cout << \*(arr + i) << " ";

    }

    cout << "\n";

    // 語法糖

    for (int i = 0; i < size; ++i) {

        cout << arr[i] << " ";

    }

    cout << "\n";

    for (int i = 0; i < size; ++i) {

        cout << &arr[i] << " ";

    }

    cout << "\n";

}

int main() {

    int arr[] = {1, 2, 3, 4, 5};

    printArray01(arr, 5); // 傳遞

    return 0;

}

# 我請GPT幫我生成練習的題目：

## 範例01：Memory Address Inspector

|  |  |
| --- | --- |
| 題目描述：  給定一個整數 x，請輸出該變數的值與記憶體位址（模擬輸出即可，實際位址用 placeholder 模擬）。 | |
| Input  單一整數 x（0 ≤ x ≤ 1000） | Output  輸出兩行：  第一行輸出 Value = x  第二行輸出 Address = 0xABCDEF01（用此值模擬地址即可） |
| Sample Input  123 | Sample Output  Value = 123  Address = 0xABCDEF01 |

## 範例02：Pointer Addition

|  |  |
| --- | --- |
| 題目描述：  輸入一個含 n 個整數的陣列，請用指標運算輸出每個元素的值（不可使用陣列語法）。 | |
| Input  第一行一個整數 n（1 ≤ n ≤ 100） 第二行 n 個整數 | Output  輸出一行，陣列元素值以空格分隔，使用指標加法存取 |
| Sample Input  5  1 2 3 4 5 | Sample Output  1 2 3 4 5 |

## 範例03：Pointer vs Array Access

|  |  |
| --- | --- |
| 題目描述：  請同時使用 ptr[i] 與 \*(ptr + i) 兩種方式，輸出同一陣列的所有值。 | |
| Input  一行輸入 5 個整數 | Output  輸出兩行： 第一行：使用 ptr[i] 輸出陣列內容 第二行：使用 \*(ptr + i) 輸出陣列內容 |
| Sample Input  10 20 30 40 50 | Sample Output  10 20 30 40 50  10 20 30 40 50 |

## 範例04：Array as Function Argument

|  |  |
| --- | --- |
| 題目描述：  設計一個函數 printArray(int\* arr, int size)，輸入陣列與長度後，請用三種方式輸出陣列內容：   * arr[i] * \*(arr + i) * &arr[i] | |
| Input  第一行整數 n（1 ≤ n ≤ 100） 第二行輸入 n 個整數 | Output  三行：依序使用上述三種方法輸出，空格分隔 |
| Sample Input  4  7 8 9 10 | Sample Output  7 8 9 10  7 8 9 10  0x1 0x2 0x3 0x4 |

## 範例05：Pointer to Max

|  |  |
| --- | --- |
| 題目描述：  請實作一個函數 int\* findMax(int\* arr, int n) 回傳指向最大值的指標，主程式輸出其值與位址。 | |
| Input  第一行一個整數 n（1 ≤ n ≤ 100） 第二行 n 個整數 | Output  兩行： 第一行最大值 第二行記憶體地址（模擬輸出） |
| Sample Input  5  3 9 2 7 1 | Sample Output  9  0x12345678 |

## 範例06：Pointer Search

|  |  |
| --- | --- |
| 題目描述：  請實作一個 int\* binarySearch(int\* arr, int size, int target) 回傳指向 target 的指標，若未找到則回傳 nullptr，並輸出其值或 Not Found。 | |
| Input  第一行一個整數 n（1 ≤ n ≤ 1000） 第二行 n 個整數 | Output  若找到 x，輸出 Found x at address 0xABCDEF01，否則輸出 Not Found |
| Sample Input  5 7  1 3 5 7 9 | Sample Output  Found 7 at address 0xABCDEF01 |

# 陣列指標（Array Pointer）

陣列指標是指向整個陣列的指標，不是指向單一元素。最常用於多維陣列工作與保留陣列大小資訊。

|  |  |
| --- | --- |
| **宣告方式** | **說明** |
| int \*p = arr; | 指向 arr[0]，第 0 個元素 |
| int (\*p)[5] = &arr; | 指向整個 arr[5] 陣列 |

這邊可以先提一下，在存取時：

(\*p) [i]：取指向陣列的第 i 個元素(就是arr[i] 的值)

\*p[i]：

* 會先讀p[i]：從指標p，視為陣列p(就是陣列指標)，再去做解參考運算子(\*) ，❗️結果是未定義行為

## 範例：區分【指向元素】跟【指向整個陣列】

#include <iostream>

using namespace std;

int main() {

    int arr[5] = {10, 20, 30, 40, 50};

    int\* p1 = arr;         // 指向 arr[0]

    int (\*p2)[5] = &arr;   // 指向整個陣列 arr[5]

    // 存取元素

    cout << p1[2] << endl;        // 印出 30

    cout << (\*p2)[2] << endl;     // 印出 30

    // 指向資料的大小

    cout << "sizeof(\*p1) = " << sizeof(\*p1) << endl;  // 4（int）

    cout << "sizeof(\*p2) = " << sizeof(\*p2) << endl;  // 20（5 ints）

    // 指標本身的大小（通常是 8 bytes on 64-bit systems）

    cout << "sizeof(p1) = " << sizeof(p1) << endl; // 8

    cout << "sizeof(p2) = " << sizeof(p2) << endl; // 8

    return 0;

}

## 範例：區分【指向元素】跟【指向整個陣列】

#include <iostream>

using namespace std;

int main() {

    int arr[5] = {1, 2, 3, 4, 5};

    int\* p1;         // 指向單一整數（arr[0]）

    int (\*p2)[5];    // 指向整個陣列（arr）

    p1 = arr; // 也可以寫p1 = &arr[0];

    p2 = &arr;

    // 印出地址

    cout << "p1  = " << p1 << endl; // 0x7ffffcbd0

    cout << "p2  = " << p2 << endl; // 0x7ffffcbd0

    cout << "\*p2 = " << \*p2 << endl; // 0x7ffffcbd0

    cout << endl;

    // 印出值

    cout << "p1[0] = " << \*p1 << endl;       // 1

    cout << "p1[1] = " << \*(p1 + 1) << endl; // 2

    cout << "p1[0] = " << \*p2 << endl;       // 0x7ffffcbd0

    cout << "p1[0] = " << \*p2[0] << endl;    // 1

    // 向後移動

    p1++;  // 前進 1 個 int（4 bytes）

    p2++;  // 前進 1 個陣列（5 \* sizeof(int) = 20 bytes）

    // 印出地址

    cout << "p1  = " << p1 << endl; // 0x7ffffcbd0

    cout << "p2  = " << p2 << endl; // 0x7ffffcbd0

    cout << "\*p2 = " << \*p2 << endl; // 0x7ffffcbd0

    cout << endl;

    // 印出值

    cout << "p1[0] = " << \*p1 << endl;       // 1

    cout << "p1[1] = " << \*(p1 + 1) << endl; // 2

    cout << "p1[0] = " << \*p2 << endl;       // 0x7ffffcbd0

    cout << "p1[0] = " << \*p2[0] << endl;    // 7(未知)

    cout << "p1[0] = " << (\*p2)[0] << endl;  // 7(未知)

    return 0;

}

## 範例：將 Array Pointer 傳入函式

#include <iostream>

using namespace std;

// 接收一個指向含有 5 個 int 的陣列指標

void printArray(int (\*p)[5]) {

    for (int i = 0; i < 5; ++i) {

        cout << (\*p)[i] << " "; // 取陣列的值

        // cout << \*p[i] << " "; // 為定義行為

    }

    cout << endl;

}

int main() {

    int arr[5] = {1, 2, 3, 4, 5};

    printArray(&arr);  // 傳整個陣列的位址

    return 0;

}