

程式設計 (一)

CH11. 指標

Ming-Hung Wang 王銘宏

tonymhwang@cs.ccu.edu.tw

Department of Computer Science and Information Engineering
National Chung Cheng University

Last Semester, 2021

本章目錄

1. 取址運算子 (Address Operator)
2. 指標變數 (Pointer Variable)
3. 反參考運算子 (Dereferencing Operator)
4. 指標的算術運算
5. 傳參考呼叫 (Call by Reference)
6. `const` 修飾詞在指標上的使用
7. 指標陣列
8. 函式指標

取址運算子

(Address Operator)

取址運算子 (Address Operator)

& 運算子，或稱為取址運算子 (address operator)，
是一個會傳回運算元位址的一元運算子。

```
1 // & operator
2 #include <stdio.h>
3
4 int main() {
5     int a = 10;
6     printf("%p\n", &a);
7 }
```

"D:\codeblocks\& operator.exe"

0061FF1C

| | | | | | | | | |
|-----|----------|----------|----------|----------|----------|----------|----------|-----|
| ... | 00B3FEC6 | 00B3FEC7 | 00B3FEC8 | 00B3FEC9 | 00B3FECA | 00B3FECB | 00B4FECC | ... |
| ... | garbage | garbage | 00000000 | 00000000 | 00000000 | 00001010 | garbage | ... |

取址運算子 (Address Operator)

其實我們使用 `scanf` 時，就已經多次使用過取址運算子。
例如 `scanf("%d", &a);` 中的 `&a` 就是取變數 `a` 的
記憶體位址的意思。

指標變數 (Pointer Variable)

指標變數 (Pointer Variable)

指標 (pointer) 是 C 程式語言最強大的功能之一。指標能讓程式模擬傳參考讓函式之間能互相傳遞，以及產生和操作動態的資料結構，亦即在執行時期會增大和減小的資料結構，如鏈結串列 (linked lists)、佇列、堆疊和樹。

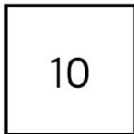
指標變數 (Pointer Variable)

指標是存放**記憶體位址**的變數。通常一個變數都會存放某個特定的數值。而指標所存放的卻是某個變數的位址。

在這種認知下，我們可以說一個變數名稱直接 (directly) 參考一個值，而一個指標則間接 (indirectly) 參考這個值。

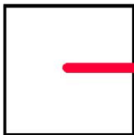
指標變數 (Pointer Variable)

count

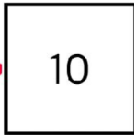


直接參考
count直接參考
一個值為10的變數

pointer



count



間接參考
指標變數pointer間接參考
一個值為10的變數

指標變數 (Pointer Variable)

宣告指標變數

宣告指標變數，只要將變數名稱前面加上 * 即可。
宣告一個 int 變數 count 與一個 int 指標變數 pointer，
並將 count 設為 10，並將 pointer 指向 count:

```
1 // pointer
2 #include <stdio.h>
3
4 int main() {
5     int count, *pointer;
6
7     count = 10;
8     pointer = &count;
9 }
```

對指標初始化及設定值

在宣告指標變數時，建議直接對指標初始化為某個位址或是初始化為 NULL。

NULL 的指標表示不指向任何東西。(NULL 是定義在 `<stddef.h>` 標頭檔中的符號常數，其值為指標型態的 0。)

```
int *point = NULL;
```

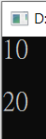
反參考運算子

(Dereferencing Operator)

反參考運算子 (Dereferencing Operator)

* 運算子通常稱為間接運算子 (indirection operator) 或反參考運算子 (dereferencing operator)，它會傳回其運算元 (即指標變數) 所指向的物件的數值。

```
1 // pointer
2 #include <stdio.h>
3
4 int main() {
5     int count = 10;
6     int *pointer = &count;
7     printf("%d\n\n", *pointer);
8     count = 20;
9     printf("%d\n", *pointer);
10 }
```

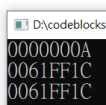


反參考運算子 (Dereferencing Operator)

* 運算子與 & 運算子的互補關係

- * 運算子作用是取記憶體位址對應的變數的值，而 & 運算子的作用則是取變數的記憶體位址，兩者為相反的動作。

```
1 // pointer
2 #include <stdio.h>
3
4 int main() {
5     int count = 10;
6     int *pointer = &count;
7     printf("%p\n", *pointer);
8     printf("%p\n", &*pointer);
9     printf("%p\n", *&pointer);
10 }
```



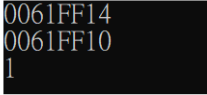
```
D:\codeblocks\
0000000A
0061FF1C
0061FF1C
```

指標的算術運算

指標的算術運算

指標可以與整數做加法或是與另一個指標做減法，
但指標不能與另一個指標做加法。

```
1 // pointer
2 #include <stdio.h>
3
4 int main() {
5     int val1, val2;
6     int *ptr1, *ptr2;
7     ptr1 = &val1;
8     ptr2 = &val2;
9     printf("%p\n", ptr1);
10    printf("%p\n", ptr2);
11    printf("%d\n", ptr1 - ptr2);
12    // printf("%p\n", ptr1 + ptr2); //ERROR
13 }
```



指標的算術運算

如上頁，ptr1 的值為 0061FF14(16 進位)，ptr2 的值為 0061FF10(16 進位)，兩者相差了 4(10 進位)，但相減的值為 1。

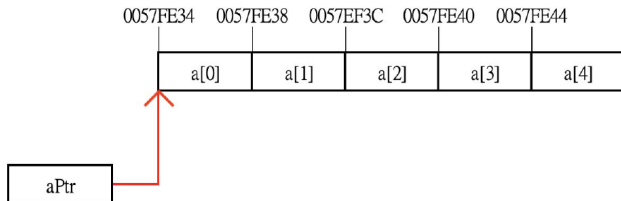
這是因為 int 大小為 4 bytes，而相差的 1 指的是 1 個 int 大小 ($1 * \text{sizeof}(\text{int}) = 4$) 的意思。

指標的算術運算

指標與整數的加法及指標與陣列的關係

若一個 `int` 指標 `aPtr` 指向一個大小為 5 的 `int` 陣列 `a` 的第 0 個元素:

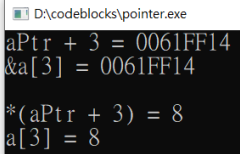
```
int a[5] = {2, 4, 6, 8, 10};  
int* aPtr = &a[0];
```



指標的算術運算

若對指標做加法後使用 * 運算子取值，也可以達到陣列下標取值一樣的作用。

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



```
D:\codeblocks\pointer.exe
aPtr + 3 = 0061FF14
&a[3] = 0061FF14

*(aPtr + 3) = 8
a[3] = 8
```

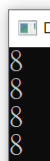
指標的算術運算

事實上，總共有 4 種參考陣列的方法，分別是：

1. 陣列下標法
2. 以陣列名稱作為指標的位移法
- 3. 指標下標法 (pointer subscripting)**
4. 使用真正的指標位移法

指標的算術運算

```
1 // pointer
2 #include <stdio.h>
3
4 int main() {
5     int a[5] = {2, 4, 6, 8, 10};
6     int * aPtr = a;
7     //陣列下標法
8     printf("%d\n", a[3]);
9     //以陣列名稱作為指標的位移法
10    printf("%d\n", *(a + 3));
11    //指標下標法
12    printf("%d\n", aPtr[3]);
13    //使用真正指標的位移法
14    printf("%d\n", *(aPtr + 3));
15 }
```



傳參考呼叫

(Call by Reference)

傳參考呼叫 (Call by Reference)

我們可以利用指標和 * 運算子來模擬傳參考的動作。

若傳給某個函式的引數應該要更改的話，我們可以傳遞引數的「位址」給函式。當傳遞變數的位址給函式時，函式可以利用 * 運算子來存取位於呼叫者記憶體內的數值。

傳參考呼叫 (Call by Reference)

以下是使用 call by value 與 call by reference 來達成
將數值乘 2 的函式:

```
18 int callByValue(int x) {  
19     return x * 2;  
20 }  
21  
22 void callByReference(int * x) {  
23     *x = *x * 2;  
24 }
```


傳參考呼叫 (Call by Reference)

```
1 // Call by reference
2 #include <stdio.h>
3
4 int callByValue(int x) {
5     return x * 2;
6 }
7
8 void callByReference(int * x) {
9     *x = *x * 2;
10 }
11
12 int main() {
13     int a = 2, b = 2;
14     a = callByValue(a);
15     callByReference(&b);
16     printf("a = %d\nb = %d\n", a, b);
17 }
```


D:\code\et

a = 4
b = 4

傳參考呼叫 (Call by Reference)

我們也可以使用指標作為參數來存取陣列，以下是將陣列內容複製的函式：

```
1 // Call by reference
2 #include <stdio.h>
3 #define SIZE 5
4
5 void arrayCopy(int *arr1, int *arr2, int n) {
6     for (int i = 0; i < n; i++) {
7         arr1[i] = arr2[i];
8     }
9 }
10
11 int main() {
12     int a[SIZE] = {2, 4, 6, 8, 10};
13     int b[SIZE];
14     arrayCopy(b, a, SIZE);
15     for (int i = 0; i < SIZE; i++)
16         printf("%d ", b[i]);
17     puts(" ");
18 }
```



const 修飾詞在指標上的使用

const 修飾詞在指標上的使用

const 修飾詞讓你能夠告訴編譯器，
某個變數的值不應該進行更改。

const 在函式的參數上共有 6 種形式，其中
傳值呼叫有 2 種 (使用 const 與不使用 const)，
傳參考呼叫有 4 種。

如何從中挑出適合自己使用的呢？我們可以用
最小權限原則 (principle of least privilege)
來做為挑選的準則。

const 修飾詞在指標上的使用

傳一個指標給函式的方式:

- 指向非常數資料的非常數指標
`int *ptr`
- 指向非常數資料的常數指標
`int *const ptr`
- 指向常數資料的非常數指標
`const int *ptr`
- 指向常數資料的常數指標
`const int *const ptr`

const 修飾詞在指標上的使用

指向非常數資料的非常數指標

存取權等級最高的形式，資料可透過對指標的求值而遭到更改，指標也可能改為指向其他的資料。

```
1 // Call by reference
2 #include <stdio.h>
3 #include <ctype.h>
4
5 void stringToUpper(char * a) {
6     while (*a != '\0') {
7         *a = toupper(*a);
8         a++;
9     }
10 }
11
12 int main() {
13     char s[] = "apple and banana.";
14     stringToUpper(s);
15     puts(s);
16 }
```


D:\codeblocks\pointer.exe

APPLE AND BANANA.

指向非常數資料的常數指標

指標永遠都會指到同一個記憶體位置，不過我們可透過這個指標來更改它所指向的數值。

```
1 // Call by reference
2 #include <stdio.h>
3 #include <ctype.h>
4
5 void stringToUpper(char * a) {
6     for (int i = 0; a[i] != '\0'; i++) {
7         a[i] = toupper(a[i]);
8     }
9 }
10
11 int main() {
12     char s[] = "apple and banana.";
13     stringToUpper(s);
14     puts(s);
15 }
```



The terminal window shows the output of the program: APPLE AND BANANA.

const 修飾詞在指標上的使用

指向常數資料的非常數指標

指標可改為指向適當型別的任何資料，但它所指向的資料卻不能夠進行更改。

```
1 // Call by reference
2 #include <stdio.h>
3 #include <ctype.h>
4
5 void stringCopy(char *a, const char *b) {
6     while (*b != '\0') {
7         *a = *b;
8         a++;
9         b++;
10    }
11    *a = '\0'; //The end of string
12 }
13
14 int main() {
15     char s1[] = "apple and banana.";
16     char s2[50];
17     stringCopy(s2, s1);
18     puts(s2);
19 }
```

D:\codeblocks\pointer.exe

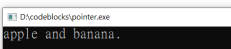
apple and banana.

const 修飾詞在指標上的使用

指向常數資料的常數指標

這種指標會永遠指到同一個記憶體位置，並且該記憶體位置中的數值不能夠受到更改。

```
1 // Call by reference
2 #include <stdio.h>
3 #include <ctype.h>
4
5 void stringCopy(char *const a, const char *const b) {
6     int i;
7     for (i = 0; b[i] != '\0'; i++) {
8         a[i] = b[i];
9     }
10    a[i] = '\0'; //The end of string
11 }
12
13 int main() {
14     char s1[] = "apple and banana.";
15     char s2[50];
16     stringCopy(s2, s1);
17     puts(s2);
18 }
```

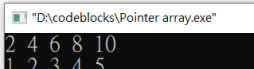


指標陣列

指標陣列

陣列所存放的物件也可以是指標。


```
1 // Pointer array
2 #include <stdio.h>
3 #define SIZE 5
4
5 int main() {
6     int a[SIZE] = {2, 4, 6, 8, 10};
7     int b[SIZE] = {1, 2, 3, 4, 5};
8     int *ptrArr[2];
9     ptrArr[0] = a;
10    ptrArr[1] = b;
11    for (int i = 0; i < 2; i++) {
12        for (int j = 0; j < SIZE; j++)
13            printf("%d ", ptrArr[i][j]);
14        puts("");
15    }
16 }
```



指標陣列

指標陣列常用來建構字串陣列。在這種情形下，陣列中的每一個項目都是字串，而 C 的字串本質上是一個指向字串第一個字元的指標。

```
1 // String array
2 #include <stdio.h>
3 #define SIZE 4
4
5 int main() {
6     char* strArr[SIZE] = { "heart",
7                             "diamond",
8                             "club",
9                             "spade" };
10    for (int i = 0; i < SIZE; i++) {
11        puts(strArr[i]);
12    }
13 }
```



函式指標

陣列名稱實際上是此陣列第一個元素在記憶體內的位址。同樣的，函式的名稱是執行此函式之程式碼的起始位址。

指向函式的指標 (pointer to a function) 可儲存函式在記憶體中的位址。

函式指標的宣告

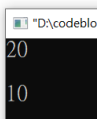
回傳值型別 (* 變數名)(參數列)

```
1 // Function pointer
2 #include <stdio.h>
3
4 int max(int a, int b) {
5     if (a > b)
6         return a;
7     return b;
8 }
9
10 int main() {
11     int (*funcPtr)(int, int) = max;
12     printf("%d\n", funcPtr(10, 20));
13 }
```



在宣告後進行指派

```
1 // Function pointer
2 #include <stdio.h>
3
4 int max(int a, int b) {
5     return a > b ? a : b;
6 }
7
8 int min(int a, int b) {
9     return a < b ? a : b;
10 }
11
12 int main() {
13     int (*funcPtr)(int, int) = max;
14     printf("%d\n\n", funcPtr(10, 20));
15     funcPtr = min;
16     printf("%d\n", funcPtr(10, 20));
17 }
```



"D:\codeblo
20
10

函式指標作為參數

若需要多個功能相近的函式 (如遞增排序與遞減排序)
可將功能相異的部分使用函式指標參數取代。

```
1 // Function pointer
2 #include <stdio.h>
3
4 int bigger(int a, int b) {
5     return a > b;
6 }
7
8 int smaller(int a, int b) {
9     return a < b;
10 }
11
12 void sort(int* arr, int n, int (*cmp)(int, int)) {
13     for (int i = 0; i < n - 1; i++)
14         for (int j = 0; j < n - i - 1; j++)
15             if (cmp(arr[j], arr[j + 1])) {
16                 int tmp = arr[j];
17                 arr[j] = arr[j + 1];
18                 arr[j + 1] = tmp;
19             }
20 }
```


```
22 void printArray(int* arr, int n) {
23     for (int i = 0; i < n; i++)
24         printf("%d ", arr[i]);
25     puts("");
26 }
27
28 int main() {
29     int nums[] = {1, 3, 58, 2, 65, 4, 2, 3, 50};
30     sort(nums, 9, bigger);
31     printArray(nums, 9);
32     sort(nums, 9, smaller);
33     printArray(nums, 9);
34 }
```

D:\codeblocks\Function pointer.exe

```
1 2 2 3 3 4 50 58 65
65 58 50 4 3 3 2 2 1
```

函式指標陣列

```
1 // Function pointer
2 #include <stdio.h>
3
4 int plus(int a, int b) {
5     return a + b;
6 }
7
8 int times(int a, int b) {
9     return a * b;
10 }
11
12 int main() {
13     int (*funcArr[2])(int, int) = {plus, times};
14     for (int i = 0; i < 2; i++)
15         printf("%d\n", funcArr[i](36, 64));
16 }
```



A small terminal window titled "D:\codeblc" is shown in the bottom right corner. It displays the output of the program: 100 on the first line and 2304 on the second line.

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