1

Lecture 13

指標
函式呼叫的資料傳遞 (III) – 傳址
指標與陣列

2

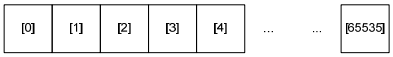
指標

Pointer

3

指標/指位器 (Pointer)

- 變數
  - int a; → 整數型別，名稱為 a
  - 變數是為了使用記憶體資源來儲存資料與進行運算
  - 所有的變數都佔有記憶體空間
- 記憶體
  - 可視為一個很大的一維陣列，單位是 byte



  - 1GB → 1,024 MB → 1,048,576 KB → 1,073,741,824 Bytes

4

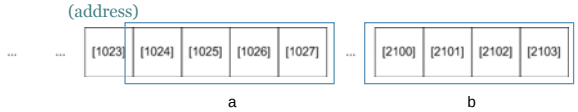
問題

- 一個 4KB 的電腦，其記憶體位置(編號)從 0 至？
  - 4 x 1024 – 1 = 4095
- 一台 1MB的電腦，其記憶體位置從 0 至？
  - 1 x 1024 x 1024 – 1 = 1048575

5

指標 (Pointer)

- 變數宣告
  - int a; → 整數型別，名稱為 a，由型別知其佔記憶體大小。sizeof()
  - float b; → 浮點數，名稱為 b ，由型別可知其佔記憶體大小。
- 記憶體
  - 在記憶體 (陣列) 裡每個元素都有一個索引 ← 稱為記憶體位置 (address)



  - a 變數佔了四個 byte, 從位址 1024 開始
  - b 變數佔了四個 byte, 從位址 2100 開始

6

指標/指位器 (Pointer)

- 指標/指位器
  - `int a, b;`
  - `int *ptrA, *ptrB;`
  - ptrA 是一個變數，其型別為整數的指標 (int *)，此變數的值為一個記憶體位址 (address)，且該記憶體位址上所儲存的資料為整數 (int) 資料。我們常說 ptrA 指向一個整數。
  - ptrB 是一個變數，其型別為整數的指標 (int *)，此變數的值為一個記憶體位址 (address)，且該記憶體位址上所儲存的資料為整數 (int) 資料。

7

13-1.cpp

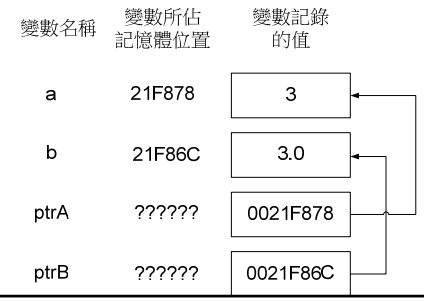
```
a=3
b=3
ptrA=0021F878
ptrB=0021F86C
```

```
#include <iostream>
using namespace std;
int main() {
  int a = 3;
  float b = 3;
  int *ptrA = &a;
  float *ptrB = &b;
  cout << "a=" << a << endl;
  cout << "b=" << b << endl;
  cout << "ptrA=" << ptrA << endl;
  cout << "ptrB=" << ptrB << endl;
  return 0;
}
```

- 每個變數「通常」有獨立的記憶體位置
- 在印出指標的值時，是以16進位方式印出的。

8

13-1.cpp 的說明



9

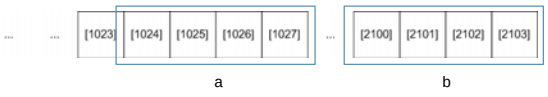
指標 (Pointer) 相關之運算子

- *****
  - 宣告時
    - int ***ptrA;** → 1) 宣告名稱為 ptrA 的變數為一「指標變數」，以後簡稱為「指標」，2) ptrA 變數內容存放的值為一個記憶體位置，3) 其指到的位址所存放的資料為整數型別 int 的資料。
  - 執行時
    - C = (*ptrA); → 將 ptrA 所指到的資料取出並存入 C。
    - *ptrA = 3; → 將 3 存入 ptrA 所指到的變數裡。
    - * 稱為 dereference (反參照、或翻為**取值**運算子)

10

指標 (Pointer) 相關之運算子

- **&**
  - 宣告時
    - 宣告一個變數為其它的變數的參考 (reference)
    - e.g. int &a = b;
  - 執行時
    - 取得一變數之記憶體位置 (e.g. **&**a, **&**b)，稱為**取址**運算子。



11

13-2.cpp

```
int **pptrA;
  •宣告一變數，名稱為 pptrA
  •pptrA 為一個指標變數
  •其指向的資料型別為 int*
  •視為int* *pptrA; 較好理解
```

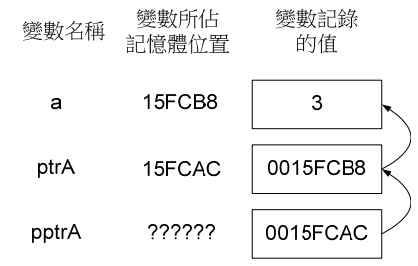
```
a=3
ptrA=0015FCB8
pptrA=0015FCAC
```

```
#include <iostream>
using namespace std;
int main() {
  int a=3;
  int *ptrA  = &a;
  int **pptrA = &ptrA;

  cout << "a=" << a << endl;
  cout << "ptrA=" << ptrA << endl;
  cout << "pptrA=" << pptrA << endl;

  return 0;
}
```

- 每個變數「通常」佔據獨立的記憶體位置。
- 每個變數都可以透過取址運算子得到其記憶體位址。

12

13-2.cpp 的說明



13

指標 (Pointer) 相關之運算子

- *
  - 宣告時
    - int *ptrA; → 1) 宣告名稱為 ptrA 的變數為一指標變數，2) 其內容存放的值為記憶體位置，3) 其指到的位址所存放的資料為整數型別的資料。
  - 執行時
    - (*ptrA); → 得到 ptrA 所指到的記憶體位址內的資料。
    - *ptrA = 3; → 將 3 存入 ptrA 所指到的記憶體位址。
    - * 稱為 dereference (翻為**取值**運算子)

14

13-3.cpp

```
#include <iostream>
using namespace std;
int main() {
  int a=3;
  int *ptrA = &a;

  cout << a;
  cout << ptrA;
  cout << *ptrA;
  *ptrA = 4;
  cout << a;

  return 0;
}
```

輸出 a 的值
輸出 ptrA 的值
輸出 ptrA 指到的記憶體的值
將 ptrA 指到的記憶體內容填入4
輸出 a 的值, 此時 a 的值為 ?

有了指標後，一個變數 (a) 可以被其它變數 (*ptrA) 所改變或使用。

15

練習

```
#include <iostream>
using namespace std;
int main() {
  float b;
  // 1. 請宣告一ptrB 變數, 其型別為指向 float 型別的指標
  // 2. 請宣告一ptrC 變數, 其型別為指向 int 型別的指標

  // 3. 請透過取址運算子(&), 將 b 變數的記憶體位置存入ptrB
  // 4. 請透過取址運算子(&), 將 b 變數的記憶體位置存入ptrC
  // 5. 請利用取值運算子(*), 將3.14 存入 ptrB 所指向的記憶體位置
  // 6. 請將 b 變數的值列印出來
  // 7. 請將ptrC 指標所指向記憶體位置之值列印出來
  return 0;
}
```

```
float *ptrB;
int *ptrC;
ptrB = &b;
ptrC=(int*)&b;
(*ptrB) = 3.14;
cout << b;
cout << (*ptrC);
```

16

請解釋或回答

```
double qq = 3.14;
double *q = &qq;
int a = 5;
int *b = &a;
int **c = &b;
int ***d = &c;
int *z = 0;

*q = 3.1415;
q = 0;
***d = 200;
cout << a;
```

以下述敍何者 ok ? 何者不 ok?

```
b=6.02;
*b = 9.03;
b=q;
b = (int*) q;
**c = 4;
&a = 0;
c = 0;
d = 200;
d = (int***) 200;
c = &z;
b = *c;
b = **d;
```

17

參考 (Reference)

* 在程式執行時，為**取值運算子**
* 在變數宣告時，代表其**宣告的變數為指標變數**
  int b = 4; int *a = &b; (*a) = 3;
  int *q, r, s; ← q 為指標變數; r, s 為兩個整數變數
  int *q, *r, *s; ← q, r, s 為三個指標變數
  cout << b;

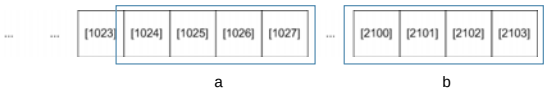
&在程式執行時，為取址運算子

& 在變數宣告時？

18

指標 (Pointer) 相關之運算子

- &
  - 宣告時
    - 宣告一個變數為其它的變數的參考 (reference)
    - 可以把它想為一個變數的分身、
  - 執行時
    - 取得一變數之記憶體位置 (e.g. &a, &b)，稱為**取址**運算子。



19

13-4.cpp

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

int main() {
  int a = 3;
  int &b = a;

  cout << &a << endl;
  cout << &b << endl;

  return 0;
}
```

```
0026F8F8
0026F8F8
```

20

13-4.cpp 的說明

- 宣告 b 為一參考型別的變數，其參考的變數為 a
- 分別將 a, b 兩個變數的記憶體位址印出來，發現它們完全一樣。故說 b 為 a 的分身或別名。
- 也因此，改變其中一個變數的值，另一個出來的值也跟著改變，因為他們實質上為同一個變數，只是他們有著不一樣的名字。

21

指標與參考的用途

- 參考 (reference)
  - 用來在不同的函式之間傳遞資料
- 指標 (pointer)
  - 用來在不同的函式之間傳遞資料 (一般變數、陣列)
  - 用來進行動態的記憶體配置

22

13-5.cpp

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

void swap(int *a, int *b);

int main() {
  int a = 3;
  int b = 7;

  cout << "a=" << a << ", b=" << b << endl;
  swap(&a, &b);
  cout << "a=" << a << ", b=" << b << endl;

  return 0;
}
```

23

13-5.cpp

```
void swap(int *a, int *b) {
  cout << *a << ", " << *b << endl;
  int tmp = *a;
  *a = *b;
  *b = tmp;
  cout << *a << ", " << *b << endl;
}
```

```
a=3, b=7
3, 7
7, 3
a=7, b=3
```

此範例亦為傳址呼叫 (call by address)，
在 swap 函式得到兩個變數的地址，並將此兩地址上的變數值作交換，所以主程式中的兩個變數值也跟著被交換了。

24

小結

- 呼叫方 (caller) 呼叫函式時，傳入資料的方式有三種：
  - 傳值呼叫 (call by value)
  - 傳址呼叫 (call by address)
  - 傳參考呼叫 (call by reference)
- 傳值呼叫

```
int func(int a) { a*=10; return a; }

        int a = b;

int b=4, int a = func(b);
cout << b;
```

只能回傳一個資料，並在
呼叫方以一變數透過 = 加以接收

25

小結

- 傳址呼叫

```
int func(int *a) { (*a)*=10; return (*a); }
                 ↘
          int *a = &b;
                 ↖
int b=4, int a = func(&b);
cout << b;
```

可以回傳一個以上的資料，只要在參數列之傳址方式傳入的資料就可以被函式裡修改，而呼叫方的資料也因此被修改。
另外也可return 資料給呼叫方。
此種方法寫出來的程式比較不美麗 ...

26

小結

- 傳參考呼叫

```
int func(int &a) { a*=10; return a; }
                 ↘
          int &a = b;
                 ↖
int b=4, int a = func(b);
cout << b;
```

可以回傳一個以上的資料，只要在參數列之傳址方式傳入的資料就可以被函式裡修改，且結果會反應在呼叫方。另外也可return 資料給呼叫方。

27

陣列與指標

Array vs. Pointer

28

13-6.cpp

```
i=0, A[i]=5, &A[i]=0029FA28, b[i]=5
i=1, A[i]=4, &A[i]=0029FA2C, b[i]=4
i=2, A[i]=3, &A[i]=0029FA30, b[i]=3
i=3, A[i]=2, &A[i]=0029FA34, b[i]=2
i=4, A[i]=1, &A[i]=0029FA38, b[i]=1
```

```
#include <iostream>
using namespace std;
int main() {
  int A[5] = {5,4,3,2,1};
  int *b=&A[0];

  for(int i=0;i<5;i++) {
      cout << "i=" << i;
      cout << ", A[i]=" << A[i];
      cout << ", &A[i]=" << &A[i];
      cout << ", b[i]=" << b[i] << endl;
  }
  return 0;
}
```

29

13-6.cpp 的說明

- `int *b=&A[0]` ➔ `int *b; b = &A[0];`
  將 `A[0]` 元素的記憶體位址，以取址運算子得到，並將之存入 `b` 這個指標裡。亦可說令 `b` 指標指向A陣列的第1個元素(索引為0)，即其陣列的開頭。
- `cout << ", &A[i]=" << &A[i];`
  可以利用取址運算子 `&` 取出陣列每個元素的記憶體位置，可以從程式執行結果發現陣列的每個相鄰元素其記憶體位置為「連續的」。
- `cout << ", b[i]=" << b[i] << endl;`
  使用指標時，除了可使用取值運算子 `*` 取得其指向的記憶體內的資料外，亦可把它當陣列使用，取出連續記憶體內的資料。

30

13-6.cpp 的說明

變數名稱	陣列元素	變數所佔記憶體位置	變數記錄的值	
A	A[0]	0029FA28	5	b[0]
	A[1]	0029FA2C	4	b[1]
	A[2]	0029FA30	3	b[2]
	A[3]	0029FA34	2	b[3]
	A[4]	0029FA38	1	b[4]
b		????	0029FA28	

31

13-7.cpp

```
i=0, A[i]=5, &A[i]=0029FA28, b[i]=5
i=1, A[i]=4, &A[i]=0029FA2C, b[i]=4
i=2, A[i]=3, &A[i]=0029FA30, b[i]=3
i=3, A[i]=2, &A[i]=0029FA34, b[i]=2
i=4, A[i]=1, &A[i]=0029FA38, b[i]=1
```

```
#include <iostream>
using namespace std;
int main() {
  int A[5] = {5,4,3,2,1};
  int *b=A;

  for(int i=0;i<5;i++) {
      cout << "i=" << i;
      cout << ", A[i]=" << A[i];
      cout << ", &A[i]=" << &A[i];
      cout << ", b[i]=" << b[i] << endl;
  }
  return 0;
}
```

32

13-7.cpp

- 陣列名 A 本身即為一個指標，指到陣列最開始元素位置
  - A[0] 與 *A 的動作完全一樣，即取得 A 這個指標所指到的位置上之資料
  - A[1] 則是取出 A 這個指標指向的元素的下一個元素資料 ... 會等同於 *(A+1) ← 我們之後會介紹指標的算術運算 ...
- 在宣告 int *b 時，我們將 A 的值 (A 陣列的開頭位置) 設給 b：
  - int *b = A; ➔ int *b; b = A;
- 所以，b 可視為 A 陣列的別名、分身 ...
- 指標可當成一維陣列來存取、使用

33

13-8.cpp

```
i=0, A[i+2]=3, &A[i+2]=002EFCFC, &b[i]=002EFCFC, b[i]=3
i=1, A[i+2]=2, &A[i+2]=002EFD00, &b[i]=002EFD00, b[i]=2
i=2, A[i+2]=1, &A[i+2]=002EFD04, &b[i]=002EFD04, b[i]=1
```

```
#include <iostream>
using namespace std;
int main() {
  int A[5] = {5,4,3,2,1};
  int *b=&A[2];
  for(int i=0;i<3;i++) {
      cout << "i=" << i;
      cout << ", A[i+2]=" << A[i+2];
      cout << ", &A[i+2]=" << &A[i+2];
      cout << ", &b[i]=" << &b[i];
      cout << ", b[i]=" << b[i] << endl;
  }
  return 0;
}
```

34

13-8.cpp

- 陣列 int A[5];
  - A[0], A[1], A[2], A[3], A[4]
- int *b = &A[2]; ➔ int *b; b = &A[2];
  - 此敍述將 A[2] 這個陣列元素的記憶體位置取出，並存入 b 指標中。
  - 也因此，*b、b[0]、A[2] 皆為相同的資料 (3)
- 又因為，陣列資料在記憶體內是連續存放，且 C/C++ 允許指標當陣列來使用，所以：
  - b[0] ← → A[2]
  - b[1] ← → A[3]
  - b[2] ← → A[4]

35

練習

```
int *b;
```

```
for(int i=0;i<10;i++) {
    cout << a[i] << " ";
}
```

```
b = &A[2];
```

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

int main() {
  int A[] = {1,2,3,4,5,6,7,8,9,10};
  //1. 宣告一個整數指標變數b(一指標變數,其指向資料為整數型別)

  //2. 請寫一個迴圈, 將 A 陣列完整列印出來
  //3. 請將 b 指標指向 A 陣列內第 3 個元素

  //4. 請寫迴圈, 把 b 指標當成一維陣列使用, 印出b[0]至b[4]

  return 0;
}
```

```
for(int i=0;i<5;i++) {
    cout << b[i] << " ";
}
```

36

指標與陣列

- 在前面已經看到我們可以把陣列的名字當指標用，也可以把指標當陣列使用。
- 所以，我們可以把陣列當成是指標變數傳遞給函式去使用。
- 在以下範例中，即將陣列當成是指標傳入函式中。由於指標指向陣列的開頭，函式裡可以存取到陣列中的每一個元素。

37

13-9.cpp

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

```
#include <iostream>
using namespace std;
void printArray(int n, const int *a);
int main() {
  int A[5] = {5,4,3,2,1};

  printArray(5, A);
  A[1] = 3;
  printArray(5, A);
  printArray(3, A);
  printArray(2, &A[2]);

  return 0;
}
```

```
void printArray(int n, const int *a)
{
  for(int i=0;i<n;i++) {
    cout << a[i] << " ";
  }
  cout << endl;
}
```

38

13-9.cpp 補充說明

- 一般而言，陣列都是以指標方式 (int *a) 傳入函式作運算
- 因為指標為記憶體位置，所以是無法知道該陣列大小的，也因此通常都會再傳遞一個陣列大小的資訊 (n) ，以讓函式作參考，也讓函式可以處理任意大小的陣列。
- 而由於陣列是以指標方式傳入，函式內對陣列進行的操作，是會反應到呼叫方的。
- 在 13-9.cpp 中，陣列傳入函式時是以 const int * 的型別傳入。在函式內而言， a 是一個指標，指向 const int，亦即常數整數，因此函式內不能對陣列的內容作修改。
- 若函式內欲對陣列進行操作與修改，則以一般的指標型別傳入。

39

13-10.cpp

```
1 2 3 4 5
```

```
#include <iostream>
using namespace std;
void initArray(int n, int *a);

int main() {
  int A[5];

  initArray(5, A);
  for(int i=0;i<5;++i) cout << A[i] << " ";

  return 0;
}
```

```
void initArray(int n, int *a) {
  for(int i=0;i<n;i++) {
    a[i] = i+1;
  }
}
```

40

回顧 – 指標與陣列

- 所有變數都存在電腦的記憶空間，其所在地方稱為記憶體位址 (地址, address)。
- 指標 (pointer)，即為用來記錄地址的特殊型別。
- 指標的宣告

```
int A=3; int *ptrA:
double B=4.0, *ptrB;
float  C; float *ptrC;
```

- 指標變數亦有型別，用來讓電腦知道所記錄的地址 (所指向的位置上) 存放資料的型別。

- 取址運算子，用來取一個變數的記憶體位置。

```
ptrA = &A;
ptrB = &B;
ptrC = &C;
```

- 欲存取一記憶體位址上的資料或值，則使用取值運算子 (deference)。

```
cout << (*ptrA)<<endl;
(*ptrB)=B+3.0;
*ptrC = 5.0;
```

41

回顧 – 指標與陣列

- 一維陣列裡所存放的資料，常使用索引用來指定要存取那一個元素。
- int q[5] = {0, 1, 2, 3, 4};

```
cout << q[2];
q[3] = 7;
q[1] += 5;
```

- 陣列的名字本身即為一指標，而其後附加的索引 i (e.g. q[i]) 即代表要存取由該記憶體地址開始的第 i+1 個元素。

- 所以，我們可以亦可用指標來存取陣列中的元素。

```
int *r; r=q;
cout << r[4];
r[1] = 7;
```

- 也因此，我們可以將陣列當成函式呼叫的參數傳入函式加以使用。

42

作業八

隨堂練習

1. 請寫一函式 void plusOne(int n, int *a); ，會將傳入的陣列 a 中的前 n 個元素都加 1。
2. 請寫一函式 void minus (int n, int *a, int b); ，會將 a 陣列中的前 n 個元素都減掉 b 值。
3. 請寫一函式 void printArray(int n, const int *a); 會將 a 陣列中的前 n 個元素印出來。(之前某個範例已經有囉～)
4. 請寫一主程式，宣告一個10個元素的陣 列 q，並填入 1, 2, 3, ... 10 的值。想辦法利用 plusOne() 函式以及 minus()函式將 q 陣列的內容變成 [2,3,4,5,6,2,3,4,5,6]，並利用 printArray 將過程和最後結果印出來。