1

回顧

- 函式呼叫時資料如何傳遞
  - 傳值呼叫
  - 參考型別
  - 傳參考呼叫
- 基本檔案操作

2

Lecture 11

函式呼叫的資料傳遞 (II) – 傳址
指標與陣列

3

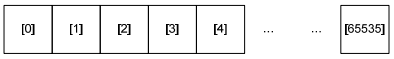
指標/指位器

Pointer

4

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

5

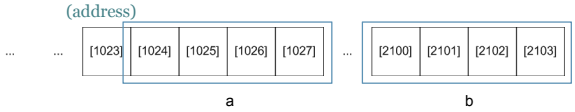
問題

- 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

6

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)

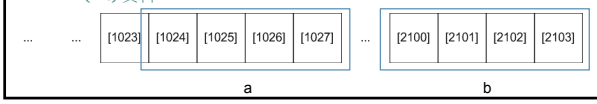


  - a 變數佔了四個 byte, 從位址 1024 開始
  - b 變數佔了四個 byte, 從位址 2100 開始

7

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。



8

11-1.cpp

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

int *ptrA: 宣告變數 ptrA為一指標，且其指向的資料為一整數。

&a: 取得變數 a 的記憶體位置，稱為取址運算子

```
#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;
}
```

9

11-1.cpp 的說明

	變數名稱	變數所佔記憶體位置	變數記錄的值
int	a	21F878	3
float	b	21F86C	3.0
int *	ptrA	??????	0021F878
float *	ptrB	??????	0021F86C

10

11-1.cpp 的說明

- 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。
- 指標變數的值，通常使用「&」這個運算子取得某一個已宣告、已存在的變數所在的記憶體位置。注意到這裡的&不是用來標記一個變數的宣告，而是作為一個運算子來使用，此運算子稱為「**取址運算子**」。
  - int a = 3;
  - int *ptrA = &a; ← → 不要和參考型別 int &A = a; 弄混
- 指標在輸出時以16進位數字表示: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, ...。16進位數字的一位數代表了4個位元
- 32位元的電腦: 記憶體位置以32位元 (4 bytes) 記錄, e.g. ffe01234; 最多可有 2^{32} 個不同記憶體位置 → 4GB。

11

11-2.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

11-2.cpp 的說明

	變數名稱	變數所佔記憶體位置	變數記錄的值
int	a	15FCB8	3
int *	ptrA	15FCAC	0015FCB8
int**	pptrA	??????	0015FCAC

13

11-3.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;
}
```

變數宣告時，加上「＊」代表該變數為一指標變數。

在程式可執行的敘述中，「＆」代表「取址運算子」，用來取得一變數的記憶體位置。

在程式可執行的敘述中，一元運算子「＊」代表「取值運算子」，用來將一記憶體位置「還原」成可使用的變數。

```
3
0021F794
3

4
```

16

11-4.cpp

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

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

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

  return 0;
}
```

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。

```
0026F8F8
0026F8F8
```

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。

14

11-3.cpp 的說明

- 程式中宣告了一個指標變數 ptrA，其儲存的記憶體位置為 a 變數的位置 (0021F794)。
  - int a=3;
  - int *ptrA = &a;
- 取值運算子 * 在讀取變數值時，所給予的記憶體位置將所儲存的資料取出來
  - cout << *ptrA;
  - 去 21f794 位置上，取得該位置所儲存的資料 (3)，並將其列印出來。
- 取值運算子*在儲存資料時，是將所給予的資料存放至所指定的記憶體位置上。
  - *ptrA = 4;
- 由於 ptrA 所儲存的記憶體位置即為變數 a 所在的記憶體位置 21f794，因此將 4 存入 21f794 位置上，會把變數 a 的值改變為 4。

17

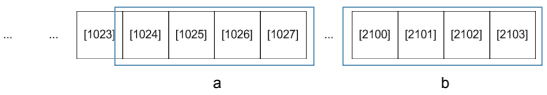
指標與參考的用途

- 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)
  - 用來操作陣列內的資料
  - 用來進行動態的記憶體配置

15

指標 (Pointer) 相關之運算子

- **&**
  - 宣告時 (e.g. int b; int &a = b)
    - 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、別名
  - 執行時 (e.g. int b = 3; int *a; a = &b)
    - 取得一變數之記憶體位置 (e.g. **&**a, **&**b)，稱為**取址**運算子。



18

11-5.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;
}
```

19

11-5.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。

20

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
呼叫方以一變數利用 = 加以接收

21

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 ...

22

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。

23

陣列與指標

Array vs. Pointer

24

11-6.cpp

```
#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;
}
```

```
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
```

指標很像在幫陣列取別名！

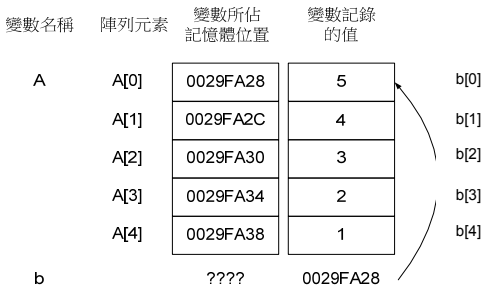
25

11-6.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。也可說令 `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。

26

11-6.cpp 的說明



27

11-7.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;
}
```

其實，陣列的名字本身即為一個指標

28

11-7.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 陣列的開頭記憶體位置) 設給 b：
  - int *b = A; ➔ int *b; b = A;
- 所以，b 可視為 A 陣列的別名、分身 ...。換句話說，指標變數可用來產生一維陣列的分身、別名。
- 指標可當成一維陣列來存取、使用

29

11-8.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;
}
```

指標不但可以當成陣列的別名，而且可以變成「子陣列」、或是另一個帶有偏移量的陣列分身。

30

11-8.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];
  - 此敘述將 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]

31

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] << " ";
}
```

32

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。
  - 參考型別：變數的分身、別名
  - 指標型別：陣列的分身、別名
- int a = 3; int &b = a; b=5; (a→5); a++; (b→ 6);
- int c[10] = {0}; int *d = c; c[3] = 4; (d[3] → 4); d[0]++; (c[0]→ 1);
- 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。

33

11-9.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;
}
```

34

11-9.cpp 補充說明

- 一般而言，陣列都是以指標方式 (e.g. 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，其實就是對原始的陣列作操作，呼叫方會「看到」改變。
- 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。

35

11-10.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;
  }
}
```

36

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;
```

37

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。

38

隨堂練習

1. 請寫一函式 void random(int n, int *a); ，它會將傳入的陣列 a 中的前 n 個元素都放入介於 1 – 10 的亂數。 (參考 11-10.cpp)
   e.g. int q[10]; random(10, q); → q[] = {9, 0, 1, 5, 4, 2, 8, 5, 1, 3};
2. 請寫一函式 int maxLoc(int n, const int *a); ，它會找出 a 陣列中的前 n 個元素裡的最大值之索引值並回傳。
   e.g. int m = maxLoc(10, q); → m = 0; int n = maxLoc(9, &q[1]) → n = 5;
3. 請寫一函式 void swapElem(int m, int n, int *a); 它會將傳入陣列 a 中的索引 m 與索引 n 的兩個元素互換。
   e.g. swap(1, 5, q); → q[] = {9, **2**, 1, 5, 4, **0**, 8, 5, 1, 3};
4. 請寫一函式 void printArray(int n, const int *a); ，它會將 a 陣列中的前 n 個元素印出來。(之前某個範例已經有囉～)
5. 請寫主程式，宣告一20個元素的陣 列 q，並以 random 函式將其內容初始化為 1-10的亂數後並列印出來。接著，請撰寫一重複19次的迴圈，迴圈計數器 i 從 0 變化至18，並在迴圈內作以下兩件事情：
   1. 找出 a 陣列中從索引 i 開始的最大值所在位置 → maxP
   2. 將 a 陣列中索引 i 的元素與索引 maxP + i 的元素互換

Google: Insertion Sort

39

作業十

Due: 6/12/2015

40

1. 排序

請撰寫一函式 sort，其原型宣告如下。此函式將傳入的陣列 a 中前面 n 個元素由小排到大作排序。

void sort(int n, int *a);

e.g. int a[] = {5, 3, 1, 2, 4}; sort(5, a); → a[] = {1, 2, 3, 4, 5};

請撰寫一主程式，讓使用者輸入20個數字，之後將此20個數字排序後列印出來，並輸出此20個數字的中位數是多少。

p.s. 中位數：若資料個數為奇數 n 時，則中位數為排序後數列的 (n+1)/2個；若資料個數為偶數 n 時，則中位數為排序後數的 n/2 個 與 n/2+1個的平均。以此題來說，即為 a[9] 與 a[10] 的平均。

41

```
請輸入第 1 個數字：20
請輸入第 2 個數字：19
請輸入第 3 個數字：18
請輸入第 4 個數字：17
請輸入第 5 個數字：16
請輸入第 6 個數字：15
請輸入第 7 個數字：14
請輸入第 8 個數字：13
請輸入第 9 個數字：12
請輸入第 10 個數字：11
請輸入第 11 個數字：10
請輸入第 12 個數字：9
請輸入第 13 個數字：8
請輸入第 14 個數字：7
請輸入第 15 個數字：6
請輸入第 16 個數字：5
請輸入第 17 個數字：4
請輸入第 18 個數字：3
請輸入第 19 個數字：2
請輸入第 20 個數字：1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
其中的中位數為：10.5
```

```
請輸入第 1 個數字：1
請輸入第 2 個數字：21
請輸入第 3 個數字：2
請輸入第 4 個數字：22
請輸入第 5 個數字：3
請輸入第 6 個數字：34
請輸入第 7 個數字：4
請輸入第 8 個數字：24
請輸入第 9 個數字：5
請輸入第 10 個數字：25
請輸入第 11 個數字：6
請輸入第 12 個數字：26
請輸入第 13 個數字：7
請輸入第 14 個數字：27
請輸入第 15 個數字：8
請輸入第 16 個數字：28
請輸入第 17 個數字：9
請輸入第 18 個數字：29
請輸入第 19 個數字：10
請輸入第 20 個數字：30
1 2 3 4 5 6 7 8 9 10 21 22 24 25 26 27 28 29 30 34
其中的中位數為：15.5
```

42

2. 數個數

請撰寫一函式 count，其原型宣告如下：

- void count(int n, const int *a, int &nOdd, int &nEven);
- 其中，n 為傳入值，代表總共有幾個數字；*a 為存放資料的陣列；nOdd為輸出值，代表 a 陣列中有幾個奇數；nEven為輸出值，代表 a 陣列中有幾個偶數。
- e.g. int a[] = {3, 1, 5, 4, 7, 7, 1, 4, 2, 0}, nOdd, nEven;
  count(10, a, nOdd, nEven); → nOdd = 6, nEven = 4;
  count(4, &a[6], nOdd, nEven); → nOdd = 0, nEven = 4;

請撰寫一主程式，讓使用者輸入20個數字，之後呼叫 count 函式，最後列印出來使用者輸入了幾個偶數、幾個奇數。