1

回顧

- 變數的視野/範疇
  - 全域變數
  - 區域變數
- 函式呼叫時資料如何傳遞
  - 傳值呼叫
  - 參考型別
  - 傳參考呼叫

2

全域變數

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

int pig;
void fun1() {
  cout << pig << endl;
  pig++;
}
void fun2() {
  cout << pig << endl;
  pig *= 2;
}
int main() {
  pig = 3;
  fun1();
  fun2();
  fun1();
  fun2();
  cout << pig << endl;
}
```

```
3
4
8
9
18
```

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

int pig;
void fun1() {
  int pig = 3;
  cout << pig << endl;
  pig++;
}
void fun2() {
  int pig = 5;
  cout << pig << endl;
  pig *= 2;
}
int main() {
  pig = 3;
  fun1();
  fun2();
  fun1();
  fun2();
  cout << pig << endl;
}
```

```
3
5
3
5
3
```

3

參考型別 (reference)

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

int main() {
    string Superman;
    string &ClarkKent = Superman;

    Superman = "strong";
    cout << ClarkKent << endl;
    cout << Superman << endl;

    ClarkKent = "Kryptonian";
    cout << ClarkKent << endl;
    cout << Superman << endl;

    return 0;
}
```

```
strong
strong

news reporter
news reporter
```

4

傳參考呼叫 (call by reference)

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

void whoIsSpiderman(string who) {
    who = "Peter Parker";
}

void whoIsFlash(string &who) {
    who = "Barry Allen";
}

int main() {
    string man = "???";
    whoIsSpiderman(man);
    cout << man << endl;
    whoIsFlash(man);
    cout << man << endl;
    return 0;
}
```

```
???
Barry Allen
```

5

Lecture 12

指標
函式呼叫的資料傳遞 (II) – 傳址
指標與陣列

6

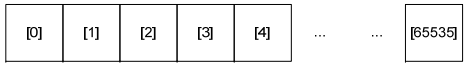
指標/指位器

Pointer

7

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

8

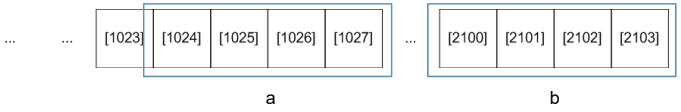
問題

- 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

9

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)

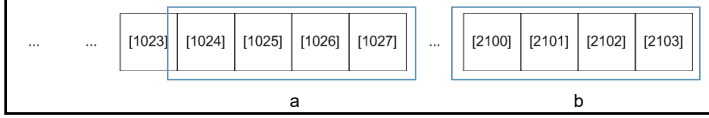


  - a 變數佔了四個 byte, 從位址 1024 開始
  - b 變數佔了四個 byte, 從位址 2100 開始

10

指標/指位器 (Pointer)

- `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。



11

12-1.cpp

```
#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;
}
```

```
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

12

12-1.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

13

12-1.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) 其記錄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「裝飾」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
- 指標以 cout 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 ($2^4 = 16$)
- 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。

14

12-2.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; 較好解讀
```

```
#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;
}
```

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

•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。

15

12-2.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

16

12-3.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。

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

在程式可執行的敍述中，一元運算子「＊」代表「取值運算子」，用來取出一記憶體位置上的資料、或將資料存入記憶體內。

```
3
0021F794
3

4
```

17

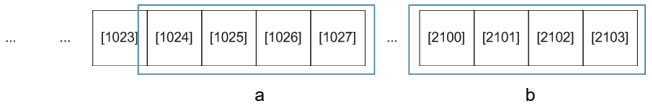
12-3.cpp 的說明

- 程式中宣告了一個指標變數 ptrA，其儲存的記憶體位置為 a 變數的地址 (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 地址上，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地址上。
  - *ptrA = 4;
- 由於 ptrA 所儲存的記憶體位置即為變數 a 所在的記憶體地址 21f794，因此將 4 存入 21f794 地址上，會把變數 a 的值改變為 4。

18

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。



19

12-4.cpp

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

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

  cout << a << endl;
  cout << b << endl;
  cout << *c << endl;

  return 0;
}
```

```
0026F8F8
0026F8F8
0026F8F8

3
3
3
```

宣告 b 為一參考型別的變數，其參考的變數為 a，也就是 b 為 a 變數的另一個名字。

宣告 c 為一整數型別的指標，其記錄的記憶體位置為變數 a 的記憶體位置。

分別將 a, b 兩個變數的記憶體位址透過取址運算子**&**得到後列印出來，發現它們完全一樣。故說 b 為 a 的別名或分身。

變數 c 在宣告時即將其值初始化為變數 a 的記憶體位置，所以將其列印出來時也和相同於 &a 與 &b 的結果。

20

指標與參考的用途

- 參考 (reference)
  - 用來在不同的函式之間傳遞變數資料
- 指標 (pointer)
  - 用來在不同的函式之間傳遞資料 (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*

21

12-5.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;
}
```

22

12-5.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。

23

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) {      int a = b;
  a*=10;
  return a;
}

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

24

小結

- 傳址呼叫

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

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 ...

25

小結

- 傳參考呼叫

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

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。

26

陣列與指標

Array vs. Pointer

27

12-6.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;
}
```

指標很像在幫陣列取別名！

28

12-6.cpp 的說明

- `int *b=&A[0]`
  以取址運算子得到 `A[0]` 元素的記憶體位址，並將其作為 `b` 指標變數的初始值。也可說令 `b` 指標指向 `A` 陣列的開頭元素(索引為 0) 。
- `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」，其數值的差異即為該陣列的變數型別所佔的 byte 數。
- `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。
- 所以，「[]」可以當成是一種帶偏差量 (offset) 的取值運算子。

29

12-6.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	

30

12-7.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

31

12-7.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 的結果是一模一樣的，即取得 A 這個指標所指到的位置上之資料、或是取得A位址上偏移量為0的元素。
  - A[1] 則是取出 A 這個指標指向的元素的下「1」個元素資料、或是從 A 開始偏移量為 1 的元素資料，會等同於 *(A+1)
  - A[i] 則是取出 A 這個指標指向的元素的下「i」個元素資料、或是從 A 開始偏移量為 2 的元素資料，會等同於 *(A+i)
  - 在對指標變數作加減 n 的算術運算時 (A++, A+5, A-2...) ，會得到下或上 n 個元素的位置。
- 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

32

12-8.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;
  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。

33

12-8.cpp

- 陣列 int A[5];
  - A[0], A[1], A[2], A[3], A[4]
- int *b = &A[2]; ←→ int *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]

34

練習

```
#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 陣列內的第 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;
}
```

```
int *b;
```

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

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

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

35

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會變成5); a++; (b會變成6);
- int c[10] = {0}; int *d = c; c[3] = 4; (d[3] 會是 4); d[0]++; (c[0] 會是 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。

36

12-9.cpp

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

```
#include <iostream>
using namespace std;
void printArray(int, const int *);
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;
}
```

37

12-9.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，也因此通常會再另外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 12-9.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。

38

12-10.cpp

```
0 0 0 0 0 0 0 0 0 0
1 2 3 4 5
1 2 3 4 5 1 2 3 4 5
1 2 3 1 2 3 2 3 4 5
1 2 1 2 3 4 5 6 4 5
```

```
#include <iostream>
using namespace std;
void setArray(int, int *);
void printArray(int n, const int *array);
int main() {
  int A[10] = {0};          printArray(10, A);
  setArray(5, A);           printArray(5, A);
  setArray(5, A+5);         printArray(10, A);
  setArray(3, A+3);         printArray(10, A);
  setArray(6, &A[2]);       printArray(10, A);
  return 0;
}
void setArray(int n, int *a) {
  for(int i=0;i<n;i++) a[i] = i+1;
}
void printArray(int n, const int *a) {
  for(int i=0;i<n;i++) cout << a[i] << " "; cout << "\n";
}
```

39

回顧 - 指標與陣列

- 所有變數都存在電腦的記憶空間，其所在的位置稱為記憶體位置 (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;
```

40

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偏移 i 個元素的資料。

- 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。

41

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。

42

作業十一

本次作業的程式 a) 讓使用者任意輸入十個正整數，輸入完後 b) 印出使用者輸入的十個數字、c) 以這個十個數字繪製橫條圖、d) 接著將這十個數字從小到大排序、排好後 e) 印出這十個數字的排序結果、 f)並依排序結果繪製橫條圖。其中，主程式如下，並請完成其所需的函式，包括 inputArray、printArray、horizontalBarChart、sortArray。

```
int main() {
    int a[10];
    inputArray(10, a);

    printArray(10, a);
    horizontalBarChart(10, a);

    sortArray(10, a);

    printArray(10, a);
    horizontalBarChart(10, a);

    return 0;
}
```

43

- inputArray: 輸入資料至陣列，此函式傳入兩參數，第一個參數為要輸入幾個數字、第二個參數則是儲存使用者輸入數字的陣列。
- printArray: 印列陣列資料，此函式傳入兩參數，第一個參數為要印幾個數字，第二個參數則是要被列印的陣列。
- horizontalBarChart：以陣列資料繪製橫條圖，此函式傳入兩參數，第一個參數為幾個資料、第二個參數為儲存資料的陣列。
- sortArray: 將陣列內容由小到大排序，此函式入傳入兩參數，第一個參數為陣列內有幾個資料，第二個參數為欲被排序的陣列。

演算法

```
for i = 0, 1, 2, ... n-2
        令 pos 為 data 陣列中 i ~ n-1 之間最小元素所在的位置
        交換 data[i] 與 data[pos]
end for
```

44

排序演算法的範例

若 data[] = {1, 5, 2, 4, 3}

i=0:
- pos=0，因為 data[0], data[1], data[2], data[3], data[4] 中 data[0] 的值為最小
- 交換 data[0] 與 data[0]
- 結果 data[] = {1, 5, 2, 4, 3}

i=1:
- pos=2，因為 data[1], data[2], data[3], data[4] 中 data[2] 的值最小
- 交換 data[1] 與 data[2]
- 結果 data[] = {1, 2, 5, 4, 3}

i=2:
- pos=4，因為 data[2], data[3], data[4] 中 data[4] 的值最小
- 交換 data[2] 與 data[4]
- 結果 data[] = {1, 2, 3, 4, 5}

i=3:
- pos=3，因為 data[3], data[4] 中 data[3] 的值最小
- 交換 data[3] 與 data[3]
- 結果 data[] = {1, 2, 3, 4, 5}

完成

45

輸入資料

請輸入第 1 個正整數:81
請輸入第 2 個正整數:64
請輸入第 3 個正整數:49
請輸入第 4 個正整數:36
請輸入第 5 個正整數:25
請輸入第 6 個正整數:16
請輸入第 7 個正整數:9
請輸入第 8 個正整數:4
請輸入第 9 個正整數:1
請輸入第 10 個正整數:1

46

輸出結果

```
81, 64, 49, 36, 25, 16, 9, 4, 1, 1,
*********************************************************************************
*****************************************************************
*************************************************
*********************************
************************
***************
*********
****
*
*
1, 1, 4, 9, 16, 25, 36, 49, 64, 81,
*
*
****
*********
***************
************************
*********************************
*************************************************
*****************************************************************
*********************************************************************************
```