1

回顧

- 動態多維陣列
  - 不夠彈性的多維動態記憶體管理 (多維陣列指標)
  - 指標的指標 (指標陣列)
  - 轉換為一維陣列
- 多維陣列的傳遞
- 變數的儲存等級 / 生命週期
- 變數的視野

2

摘要 – 三種多維陣列的動態記憶體管理

- 假設 nRow 為列數、nCol為行數。
- 不夠彈性的多維動態記憶體管理 (p.s. nCol 必需為常數變數)

```
int (*a)[nCol] = new int[nRow][nCol];
for(int i=0;i<nRow;++i) {
    for(int j=0;j<nCol;++j) {
        a[i][j] = 0;
    }
}
delete []a;
```

3

摘要 – 三種多維陣列的動態記憶體管理

- 假設 nRow 為列數、nCol為行數。
- 以一維陣列模擬 (p.s. 使用上有點麻煩)

```
int *a = new int[nRow * nCol];
for(int i=0;i<nRow;++i) {
    for(int j=0;j<nCol;++j) {
        a[i*nCol + j] = 0;
    }
}
delete []a;
```

4

摘要 – 三種多維陣列的動態記憶體管理

- 假設 nRow 為列數、nCol為行數。
- 使用指標陣列 (p.s. **配置**與**刪除**都有點麻煩)

```
int **a = new int*[nRow];
for(i=0;i<nRow;i++) a[i] = new int[nCol];
for(int i=0;i<nRow;++i) {
    for(int j=0;j<nCol;++j) {
        a[i][j] = 0;
    }
}
for(i=0;i<nRow;i++) delete []a[i];
delete []a;
```

5

函式呼叫時的多維陣列傳遞

- 有時，我們想讓函式處理陣列資料，陣列在傳遞給函式時，都是以指標處理。

```
void array2D(int a[2][3]) {
      …
}
int p[2][3] = { {1,2,3}, {4,5,6} };
array2D(p);
```

`int (*a)[3] = p;`

- 函式參數列的參數，都可視為變數宣告
- 但是在參數列宣告的陣列，都會宣告出來陣列的指標，用來記錄所傳入陣列的記憶體位置。(i.e. 以下三種原型宣告的寫法同義!)
  - `void array2D(int a[2][3]);`
  - `void array2D(int (*a)[3]);`
  - `void array2D(int  a[][3]);`

6

15-7.cpp & 15-8.cpp

func 被呼叫 5次；
C 變數被創造出來 5 次，
每次被初始化為 5 ；
然後被消滅了 5 次。

```
void func() {
  int c = 5;
  cout << ++c << " ";
}

int main() {
  int a=3;

  cout << a << " ";
  for(int i=0;i<5;i++)
    func();

  return 0;
}
```

程式執行結果
3 6 6 6 6 6

func 被呼叫 5次；
C 變數被創造出來 1次，
並初始化為 1 ；
然後被消滅了 1 次。

```
void func() {
  static int c = 5;
  cout << ++c << " ";
}

int main() {
  int a=3;

  cout << a << " ";
  for(int i=0;i<5;i++)
    func();

  return 0;
}
```

程式執行結果
3 6 7 8 9 10

7

15-9.cpp

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

int a;

void func2() {
  int a = 8;
  int b = ::a + 7;
  a++;
  cout<<a<<","<<b<<endl;
}
```

此 a 為一全域變數

```
void func1() {
  int b = a*2;
  a++;
  cout<<a<<","<<b<<endl;
}
int main() {
  int b=5;
  a=4;
  func1();
  func2();
  func1();
  cout<<a<<","<<b<<endl;
  return 0;
}
```

5,8
9,12
6,10
6,5

8

Lecture 16

函式的回傳值
檔案操作

9

函式呼叫時的參數列資料傳遞

- 呼叫函式時，可以傳值、傳址、或傳參考的方式，透過參數列將變數資料由呼叫者傳遞給被呼叫的函式，供函式運算或回傳資料。
  - 傳值的原型宣告範例：void func1(int a, double b, float c);
  - 傳址的原型宣告範例：void func2(int *a, double *b, float *c);
  - 傳參考的原型宣告範例： void func3(int &a, double &b, double &c);
- 用途：
  - 傳值：參數列的資料只進不出
  - 傳址：參數列的資料可進可出，較適合用來傳遞陣列資料
  - 傳參考：參數列的資料可進可出，較適合用來傳遞一般非陣列資料
- 函式時回傳值，則有**四種**可能。

10

函式的回傳值

- `void func1(int a, double b, float c);`

回傳值型別　函式名稱　參數列: 可視為區域變數的宣告，而傳入之參數則為它們的初始值。

- 回傳值型別：
  - void: void 為 C/C++ 中一個特別的型別，代表沒有回傳值。
  - 回傳值：將資料以傳值的方式回傳給呼叫方。
  - 回傳址：將資料以地址的方式回傳給呼叫方。
  - 回傳參考：將資料以參考的方式回傳呼叫方。
- 除了回傳型別為 void 外，呼叫方可使用 = (指定運算子) 或是+=, -=, ... 等(複合指定運算子)去記錄或應用回傳的資料。
- **傳址或是傳參考，必需額外注意變數的生命週期的問題！**

11

16-1.cpp - 回傳值

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

int max(int a, int b) {
  if(a>b) return a;
  return b;
}

int main() {
  int c=5, d=7, q;
  q = max(c, d);
  cout << "q=" << q << ", c=" << c << ", d=" << d << endl;

  return 0;
}
```

- 此 max 函式規定了會回傳一個整數
- 因此，函式的內容必需使用 return 指令回傳一個整數
- 而撰寫者必需確定函式離開前一定會執行到 return 這個指令！

- 此例中，呼叫方 (caller) 使用 = 將 max 回傳的資料存入 q 變數中。
- 此例中可視為 q = 7;

q=7, c=5, d=7

12

16-2.cpp – 回傳值

```
#include <iostream>
using namespace std;
int max(int n, int *array);
int main() {
  int a[] = {5,4,3,2,1};
  int b[] = {1,2,3,4,5,6,7,8};
  int c[] = {10,20,30};
  cout << max(5, a) << endl;
  cout << max(8, b) << endl;
  cout << max(3, c) << endl;

  return 0;
}
```

5
8
30

```
int max(int n, int *array) {
  int ans = array[0];
  for(int i=1;i<n;i++) {
    if(array[i] > ans)
      ans = array[i];
  }
  return ans;
}
```

13

16-2.cpp 說明

- 在此範例中，傳遞一陣列至 max 函式找尋陣列中最大的元素。
- 雖然說是傳遞陣列，其實我們傳的是陣列中的第一個元素的記憶體位置 (指標指向陣列的第一個元素)。我們也必需將陣列中的元素個數傳至函式中，不然函式裡不知道陣列有幾個元素。
- 此範例中，原型宣告或是函式定義的介面，亦可採以下語法：
  - `int max(int n, int *array);` // 指標
  - `int max(int n, int array[]);` // 陣列
  - `int max(int n, const int *array);` // 唯讀指標
  - `int max(int n, const int array[]);` // 唯讀陣列
- 使用唯讀指標，可使函式只能讀出陣列的內容而無法對其作修改，可達到保護陣列資料的目的，確保其不會被函式內的程式所修改。

14

16-3.cpp – 回傳址

```
#include <iostream>
using namespace std;
int *even(int *it) {
  do {
    ++it;
  } while(*it!=0 && *it%2!=0);
  return it;
}
int main() {
  int a[] = {1,2,3,4,5,6,0};
  int *it = a;
  do {
    it = even(it);
    cout << *it << " ";
  } while(*it != 0);
  return 0;
}
```

此函式傳入一指標，並回傳下一個出現偶數或是 0 的指標 ...

2 4 6 0

15

16-3.cpp 說明

- 假設 a 陣列中的資料住在 301, 302, ... 307 號房。
- 主程式一開始令 it 指標記得 301 號房
- do 迴圈開始
  - 呼叫 even 函式，傳入 it 指標 (301 號房)，並將 even 函式回傳的結果存回 it 指標。
  - 取回 it 指標記錄房內所住的資料，並列印出來。
- 當 it 指標記錄房內所住的資料不為 0 時，繼續執行迴圈內容
- even 函式：
  - 傳入一房間號碼 (it 指標)
  - do 迴圈開始
    將 it 記得的房間號碼移向下一間房間
  - 當 it 房間號碼內住的數字不為 0 且其為奇數時，重複執行迴圈
  - // 所以，離開函式時，it裡住得必為偶數或是 0
  - 將房間號碼 (it 指標) 回傳

16

16-4.cpp – 回傳參考

```
#include <iostream>
using namespace std;
int& printStar(int n) {
  static int count=0;
  for(int i=0;i<n;++i) cout << "*";
  count += n;
  return count;
}
int main() {
  for(int i=1;i<=10;++i) {
    printStar(i*i);
    cout << "\n";
  }
  int &count = printStar(0);
  cout << "\n印出 " << count << " 顆星.";
  return 0;
}
```

17

16-4 輸出結果

```
*
****
*********
****************
************************
********************************
****************************************************
************************************************************
****************************************************************************
************************************************************************************************
```

印出 385 顆星.

18

錯誤範例

```
int *fun(int a) {
  a *= 2;
  return &a;
}
```

```
int &fun(int a) {
  a *= 2;
  return a;
}
```

在此會錯誤的原因是，此兩範例的 a 皆為儲存等級為自動的區域變數，a 只有在此函式在執行時才擁有一記憶空間儲存資料，等函式結束後即被消滅。故回傳之 a 的位置或參考皆無法正常使用。
語法正確、可編譯無誤，但執行會錯！

正確範例

```
int fun(int a) {
  a *= 2;
  return a;
}
```

```
int &fun(int &a) {
  int b=0;
  a *= 2;
  return a;
}
```

```
int *fun(int a) {
  static int b=0;
  a *= 2;
  return &b;
}
```

19

函式回傳值

- 函式呼叫時，參數列可將參數以傳值、傳址、傳參考的方式傳入函式內。
- 函式在回傳運算結果時，亦可回傳值、回傳位址、回傳參考。
  - 回傳值：用來回傳一個運算結果，類似於數學函數。
  - 回傳址：用來回傳一個指標，此指標需指向合法的記憶體位址 (動態記憶體配置、傳回陣列中的某個元素、 ...) (如上次的隨堂練習)
  - 回傳參考：回傳一個合法資料的參考 (e.g. 靜態變數、傳入陣列中的某個元素 ...)
- 注意回傳址或參考時，不可參考到自動之區域變數 (不回傳區域變數的位址、不回傳區域變數的參考)
- 但可以回傳靜態區域變數的位址或是參考

20

檔案操作

21

檔案輸入與輸出
file I/O (input/output)

- 為什麼需要電腦檔案？
  - 所有我們目前學到的，都是在利用記憶體 (變數) 來進行運算，而運算的資料來源為：
    - 1) 內建於程式內的變數 (data = 3;)
    - 2) 由使用者輸入 (cin >> data);
  - 運算得到的結果常常是輸出至螢幕上
    - cout << result;
- 問題
  - 運算結果無法長期保存下來、無法檢查
  - 使用者會忘記要輸入什麼資料
  - 重複輸入資料很煩、會出錯。
- 在 C/C++ 來說，電腦檔案是一種串流 (stream) 或資料流。

22

檔案串流
file stream

- 檔案串流，即為其來源或目的為電腦檔案
  - cin: 來源為鍵盤輸入的串流物件
  - cout: 來源為螢幕輸出的串流物件
  - 檔案串流需有相關的檔案名稱！
- 檔案輸入輸出串流
  - **ifstream** (input file stream)
  - **ofstream** (output file stream)
  - **fstream**(file stream) – 可設定作輸入、輸出的串流

23

16-5.cpp – 檔案輸出

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  ofstream file;
  file.open("c:\\test.txt");
  if(!file) {
      cout << "檔案開啟錯誤!";
      return 255;
  }
  file << "Hello File!";
  file.close();
  return 0;
}
```

含括入檔案串流的相關原型宣告

宣告 file 為一**輸出**檔案串流物件
開啟 c:\test.txt
檢查檔案串流開啟狀態，一個好的程式應該要檢查檔案是否有開啟成功，以免後面的程式出錯。

將 "Hello File!" 插入至檔案串流
關閉檔案串流

24

檔案串流物件

- cin, cout, ofstream, ifstream 都是一種串流物件
- 之前 cin, cout 的操作 <<, >> 都可直接套用在檔案操作上。
  - `file << "Hello File!";`
  - `file << "\nA=" << A << ", " << "B=" << B;`
  - << 和>>稱為串流的插入與抽取運算子
- 檔案串流只增加了
  - #include <**fstream**>
  - ifstream abc; 或是 ofstream abc; ← 用來產生輸入/輸出串流物件
  - abc.open("c:\\test.dat"); ← 將串流物件關連於一個電腦檔案
  - abc.close(); ← 關閉檔案
  - 其中 abc 為任意合法的變數名稱

25

16-6.cpp – 讀取檔案

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  ifstream file;
  char a;
  file.open("c:\\test.txt");
  if(!file) {
      cout << "檔案開啟錯誤!";
      return 255;
  }
  while(file.good()) {
      file >> a;
      cout << a;
  }
  file.close();
  return 0;
}
```

宣告 file 為一輸入檔案串流物件

嘗試開啟 c:\test.txt
檢查檔案開啟是否成功
失敗時輸出一訊息
結束程式執行，回傳一非零值乃一習慣
以代表程式異常結束
當檔案串流的「狀態」良好時作：
由檔案讀入一個字元
透過 cout 輸出至螢幕上。

檔案關閉
程式正常結束

26

串流的狀態
stream state

- 串流有以下的狀態，通常是經過輸入或輸出的操作後會被設定。
  - **eof** (end of file): 串流已至檔案結尾，表所有資料都已經流過了，不會再有更多資料進來了。
  - **fail**: 失敗，表示有輸入或操作的動作失敗。例如：
    - cin >> a; (其中 a 為整數，但輸入的資料為 "$AC"，無法轉換)
  - **bad**: 壞掉，表串流本身之完整性 (integrity) 有問題，通常代表即使將狀態清除後換一種操作仍然會發生問題。
- 可透以下相關函式檢查或設定
  - **clear**(): 將以上三種不好的狀態清除。
  - **good**(): 回傳 true 表示以上三種狀態都沒有發生, 可以進行操作。
  - **bad**(): 回傳 true 時表示 bad 的狀態發生了。
  - **fail**(): 回傳 true 時表示 bad 或 fail 的狀態發生了。

27

16-7.cpp

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

int main() {
  int a;

  // 插入 A)
  // 插入 B)

  cout << "請輸入 a:";
  cin >> a;
  cout << "a=" << a << endl;

  // 插入 C)
  return 0;
}
```

A)
```
ifstream cin("c:\\test.txt");
if(!cin) {
  cout << "c:\\test.txt 開啟失敗";
  return 255;
}
```

B)
```
ofstream cout("c:\\test1.txt");
if(!cout) {
  cout << "c:\\test1.txt 開啟失敗";
  return 255;
}
```

C)
```
cin.close();
cout.close();
```

28

16-7.cpp 的說明

- cin, cout 為 C++ 幫我們宣告好的全域變數！
- 所以，我們可以透過遮蔽的方式，利用區域變數的 cin、cout 去遮蔽掉全域變數的 cin、cout ，使 cin、cout 變成檔案串流。所以可以很容易地把原來輸出在螢幕上或是由鍵盤輸入的方式，轉變成使用檔案作輸入輸出。
- 檔案操作的三步驟
  1. 開啟檔案、檢查是否開啟成功
  2. 進行串流輸入輸出
  3. 關閉檔案串流
- 以下兩範例程式說明透過 iomanip 對輸出進行美化，僅供參考！

29

16-8.cpp – 附加資料到檔案後

- ifstream: 專門讀取資料
- ofstream: 專門寫入資料
- fstream: 由開啟時的開啟模式決定寫入、讀取、或是附加 (**app**end)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  fstream file;

  file.open("c:\\test.txt", ios::app);
  if(!file) {
    cout << "檔案開啟錯誤!";
    return 255;
  }
  for(int i=0;i<100;++i)
    file << "Hello File!" << i << "\n";
  file.close();
  return 0;
}
```

30

檔案開啟模式

- ios::in – 可讀取
- ios::out – 可寫入
- ios::app – 每次寫入資料前將檔案操作指標指向檔案尾。
- ios::ate – 開檔時將檔案操作指標指向檔尾。
- ios::binary – 寫檔時使用二進位模式 (未格式化格式, unformatted)
- ios::trunc – 將檔案既有內容清空。

- ifstream 的 open 檔案開啟模式為 ios::in
- ofstream 的 open 檔案開啟模式為 ios::out
- fstream 的 open 檔案開啟模式內定為 ios::in | ios::out

31

格式化的輸入輸出

- 格式化輸入輸出 (formatted input/output)
  - <<: 將右手邊變數的值經過**轉換**成為文字後插入左手邊之串流
  - >>: 將由串流輸入的資料以空格 (white space) 或換行將輸入的文字資料分割開來**轉換**成為適當格式後存入右手邊的變數。
- 若欲使輸出的檔案更加美觀 (對人來說!) ➔ 使用串流操控器 iomanip
  - #include <iomanip>
  - 使用 << 插入格式控制

32

串流操控器	效　果
setw(n)	將欄位寬度設定為 n。
setfill('*')	設定欄位內空白處填入的字元。
left	在欄位內靠左對齊。
right	在欄位內靠右對齊。

	串流操控器	效　果	範例
int	showpos	永遠顯示正負號，即正數之前加上+號。	+1
	noshowpos	不強制顯示正號。（預設值）	1
	dec	以10進位法表示。（預設值）	17
	oct	以8進位法表示。	21
	hex	以16進位法表示。	11
	noshowbase	不顯示數字的基底格式（預設值）	11
	showbase	顯示數字的基底格式	0x11

33

16-9.cpp

```
const int i=100;

cout << "|1234567890123456789O\n";
cout << "|" << setw(10) << i << "|\n";
cout << "|" << showpos  << i << "|\n";
cout << "|" << setw(5)  << hex << i << "|\n";
cout << "|" << setw(10) << showbase << i << "|\n";
cout << "|" << setw(10) << dec << left << i << "|\n";
cout << right;
cout << "|" << setw(7) << noshowpos << i << "|\n";
cout << "|" << setw(8) << oct << i << "|\n";
```

34

16-9.cpp 執行結果

```
|1234567890123456789O            "|1234567890123456789O\n"
|       100|                   "|" << setw(10) << i <<"|\n"
|+100|                         "|" << showpos  << i <<"|\n"
|   64|                      "|"<<setw(5)<< hex << i <<"|\n"
|      0x64|          "|"<<setw(10)<< showbase << i <<"|\n"
|+100      |           "|"<<setw(10)<<dec<<left<< i <<"|\n"
|   _100|        right<<"|"<<setw(7)<<noshowpos<< i <<"|\n"
|    0144|               "|" << setw(8) << oct << i <<"|\n"
```

35

	串流操控器	效　果	範例
float, double	setprecision(n)	將浮點精度設為 n 位數。(內定為6)	12.300000
	fixed	使用一般的浮點數表示法。	
	scientific	使用科學記號表示法。	1.230000e+001
	showpoint	永遠顯示小數點	1.0
	noshowpoint	不強制顯示小數點（預設值）	1
	showpos	永遠顯示正負號，即正數之前加上+號。	+1
	noshowpos	不強制顯示正號	1

	串流操控器	效　果	範例
bool	boolalpha	以true、false字串表示bool值。	true
	noboolalpha	以1、0表示bool值（預設值）。	1

36

16-10.cpp

```
double pi=1;

cout << showpoint << pi << endl;
cout << noshowpoint  << pi << endl;

pi = 314.159265358979323846;
cout << pi << endl;
cout << scientific << pi << endl;
cout << fixed << pi << endl;
cout << setprecision(10);
cout << setw(6) << scientific << pi << endl;
cout << setw(6) << fixed << pi << endl;
```

37

16-10.cpp 執行結果

```
1.00000                                  showpoint << pi
1                                      noshowpoint  << pi
314.159                                             pi
3.141593e+002                          scientific << pi
314.159265                                 fixed << pi
3.1415926536e+002 setprecision(10)<<setw(6)<<scientific<<pi
314.1592653590                      setw(6) << fixed << pi
```

38

隨堂練習

39

隨堂練習

$$dx = v_x \times dt$$

$$dy = v_y \times dt - \frac{g}{2} \times dt^2$$

$$v_y = v_y - gdt$$

$$g = 9.81$$

- 假設一物體之初始位置為 (0, 0)，請寫一程式：
  - 讓使用者輸入一物體之水平與垂直方向的初始速度
  - 每隔 0.1 秒，輸出時間 (0.1, 0.2, 0.3, ...) 以及物體之水平座標、以及物體的垂直座標，這三個數字 (t, x, y) 之間以逗號隔開。
  - 重複前述動作直到物體的 y 座標小於 0 為止。
  - 將計算結果輸出至「 fly.csv 」檔案內。
- 找到輸出之「fly.csv」 檔案，點兩下後可由 Excel 自動開啟。請利用 x y 散佈圖的方式畫出該物體的飛行軌跡。

40

參考輸出

$$dx = v_x \times dt$$

$$dy = v_y \times dt - \frac{g}{2} \times dt^2$$

$$v_y = v_y - gdt$$

$$g = 9.81$$

```
請輸入初速度 Vx 與 Vy: 5 5
0, 0, 0
0.1, 0.5, 0.45095
0.2, 1, 0.8038
0.3, 1.5, 1.05855
0.4, 2, 1.2152
0.5, 2.5, 1.27375
0.6, 3, 1.2342
0.7, 3.5, 1.09655
0.8, 4, 0.8608
0.9, 4.5, 0.52695
1, 5, 0.095
1.1, 5.5, -0.43505
```