1

回顧

```
int a = 3;          // 宣告一整數變數，變數名稱為 a
int *b = &a;        // b 為一指標變數，指向一整數 a
                    // &a: & 為取址運算子，取得 a 的地址。
int &c = a;         // 宣告 c 為 a 變數的參考/別名
cout << *b;         // 印出 3
*b = 100;           // 將 100 存入 b 指向的地址去
cout << a++;        // 印出 100
cout << c;          // 印出 101

cout << b;          // 印出 a 變數的位址 (e.g. 301 室)
cout << &c;         // 印出 c 變數的位址 (也是 301 室!)
```

2

回顧

```
int a[] = {1,2,3,4,5};  // 宣告一陣列 a, 初始化為 1,2,3,4,5
int *b = a;             // b 為一指標變數，a 陣列的開頭
int *c = &a[2];         // c 為一指標變數，指向 a[2] 的地址
int *d = a + 2;         // d 為一指標變數，也指向 a[2] 的地址

for(int i=0;i<3;++i) {
  cout << b[i] << ", " << c[i] << endl;
}
```

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

3

回顧

- 指標乃利用記憶體地址來存取資料的工具

```
int A=4, *ptrA = &A;
cout << *ptrA;                  // 印出 4
*ptrA = 7;
cout << A;                      // 印出 7
```

- 一維陣列 vs. 指標

```
int a[5] = {1, 2, 3, 4, 5};
int *ptrA = a;
cout << ptrA[3];                // 印出 4
```

4

參考型別：變數的分身、別名
指標型別：陣列的分身、別名

回顧

```
void fun(int n, double *a) {
  for(int i=0;i<n;++i) a[i] *= 2.0;
}

int main() {
  double q[] = {1.1, 2.2, 3.3, 4.4, 5.5};
  fun(5, q);        // {2.2, 4.4, 6.6, 8.8, 11.0}
  fun(3, &q[2]);    // {2.2, 4.4, 13.2, 17.6, 22.0}
  fun(2, q);        // {4.4, 8.8, 13.2, 17.6, 22.0}
  return 0;
}
```

5

Lecture 13

動態記憶體管理
變數的生命週期
檔案輸入輸出
多維陣列之動態記憶體管理

6

動態記憶體管理

Dynamic Memory Management

7

動態記憶體管理

- 記憶體配置 (memory allocation)
- 判斷記憶體配置的成功與否
- 記憶體釋放 (memory de-allocation/release)

8

13-1.cpp

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

int main() {
  int a=3;
  int *b = new int(4);

  cout << a << ", " << *b;

  delete b;
  return 0;
}
```

配置一個整數的記憶空間，並初始化為4。將配置取得的記憶空間存入指標變數 b

將 b 指向的記憶空間釋放歸還給作業系統。

9

13-1a.cpp

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

int main() {
  int a=3;
  int *b = new int;
  *b = 4;

  cout << a << ", " << *b;

  delete b;
  return 0;
}
```

配置一個整數的記憶空間，並將配置取得的記憶空間存入指標變數 b

將 4 存入指標變數 b 指向的記憶體空間

將 b 指向的記憶空間釋放歸還給作業系統。

10

說明：13-1.cpp、13-1a.cpp

- 在以前範例中使用指標變數，都是用別的變數的儲存空間來儲存資料
  - int a;
  - int *b = &a; *b = 3;　　// 將 3 存入 a 變數的儲存空間內
- 使用動態記憶體配置 (new)，指標變數可以取得一個獨立的資料儲存空間，不會將資料存到其到變數的儲存空間。
  - int *b = new int(3);　　// 配置一整數空間，並初始其資料為 3。
- 動態配置出來的儲存空間必需歸還給作業系統，一個 new 一定會有一個相對應的 delete 才是負責任的作法。否則，會發生記憶體漏失 (memory leak) 的現象。
  - delete b;

11

new 運算子的語法一: 配置基本變數

- 用來配置 (allocate) 一塊記憶體
- 配置到的記憶體應該要用 delete 運算子釋放掉(release, de-allocate)。

- 語法一: 指標變數 = new 變數型別**(初始值)**;
  - int *myData;
  - myData = new int(100);
  - myData = new int;
    - 註：這種寫法的意義不大，幾乎等同於 int myData=100; 而且還要記得用 delete 釋放記憶體。但等以後學到了撰寫類別與物件時，就會有差別。
  - double *another = new double(3.14159);

12

動態陣列的配置

- 陣列 (Array)
  - 用來儲存大量相同型別的資料
    - int a[100];
    - double b[40];
  - 但陣列在宣告時即必需知道大小，且大小必需是**常數**
  - 且陣列大小一經宣告即無法改變
- 有時，所需要的陣列大小無法預知。
- 透過動態記憶體管理，我們可依程式與處理資料量的需求，適當地配置記憶體空間。

13-2.cpp

```
請問你要處理多少個數字：5
請輸入第1個數字：5
請輸入第2個數字：4
請輸入第3個數字：3
請輸入第4個數字：2
請輸入第5個數字：1
1, 2, 3, 4, 5,
```

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

int main() {
  int *myData;
  int size;

  cout << "請問你要處理多少個數字：";
  cin >> size;

  myData = new int[size];
```


13-2.cpp (cont'd)

```
  for(int i=0;i<size;i++) {
      cout << "請輸入第" << i+1 << "個數字：";
      cin >> myData[i];
  }

  for(int i=size-1;i>=0;i--) {
      cout << myData[i] << ", ";
  }

  delete []myData;

  return 0;
}
```


new 運算子的語法二: 配置陣列

- 用來配置 (allocate) 一塊記憶體
- 配置到的記憶體應該要用 delete 運算子釋放掉(release, de-allocate)。

- 語法二: 指標變數 = new 變數型別 **[陣列大小]**;
  - e.g. myData = new int[100];
    - 動態地向作業系統要一可容納 100 個整數的記憶體空間。
  - 配置完成後即可將指標變數當成一般的一維陣列進行操作。
  - 這比較是我們需要的動態記憶體配置
  - 無法指定初始值！

delete 運算子

- 用來釋放之前利用 new 配置到的一塊記憶體。

- 語法一: delete 指標變數;
  - int *myData = new int; // 配置一個整數的空間
  - delete myData; // 釋放掉 myData 指到的記憶體位置

- 語法二: delete **[]**指標變數;
  - int *myData = new int[100]; // 配置容納100個整數的空間
  - delete **[]**myData; // 釋放掉 myData 指到的陣列

- 只要在 new 時有 [] 出現，在delete時就應有 []出現。!!!!

17

更多範例

- 需要存放 200 個字元 (199 個字元的C-字串)

```
char *str = new char[200];
str[3] = 'C'; …
delete []str;
```

- 需放 500 個 float 浮點數

```
float *data = new float[500];
data[10] = 3.1416; …
delete []data;
```

- 需放 n 個 double 浮點數

```
double *data = new double[n];
data[3] = 2.178281828; …
delete []data;
```

18

記憶體配置的成功與失敗

- 記憶體配置不保證成功！
- 成功時，代表程式已成功地向作業系統從 heap memory (堆積記憶體, 亦有書籍用 free store 這個詞) 中配置一塊記憶體資源供程式應用
  - heap 記憶體獨立於程式運行之記憶體空間，可取得的大小遠多於直接宣告出來的陣列 (使用 stack memory) 。
- 在 C++ 裡，有兩種方式處理記憶體配置失敗
  - (nothrow) 參數 → 12-6.cpp
  - 例外處理 (Exception handling) → 12-7.cpp (僅供參考 …)

19

13-3.cpp

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

int main() {
  char *mem;
  unsigned long size, i;

  cout << "請問你要配置幾 MB 的記憶體？";
  cin >> size;

  size *= 1024 * 1024;
  cout << "\n配置" << size << " bytes 的記憶體";
```

20

13-3.cpp (cont'd)

```
  mem = new (nothrow) char[size];
  if( mem == 0 ) {
      cout << "記憶體配置失敗，程式結束!";
      return 255;
  }
  for(i=0;i<size;i++) {
      mem[i] = (char) (i%6);
  }
  cout << "輸入任意資料後按 Enter 結束";
  cin >> mem;

  delete []mem;
  return 0;
}
```

21

13-3.cpp 的說明

- mem = new **(nothrow)** char[size];
  - 透過讓程式執行時，若記憶體配置失敗時，不要拋出例外狀況 (exception)。
- 若記憶體配置失敗且程式不允許拋出例外狀況時，會回傳一個 null pointer，目前來說 null pointer 其值為 0。

22

13-4.cpp

```
#include <iostream>
using namespace std;
int main() {
  char *mem;
  unsigned long size, i;

  try {
    cout << "請問你要配置幾 MB 的記憶體？";
    cin >> size;

    size *= 1024 * 1024;
    cout << "\n配置" << size << " bytes 的記憶體";
```

23

13-4.cpp (cont'd)

```
    mem = new char[size];
    for(i=0;i<size;i++) {
      mem[i] = i % 256;
    }
    cout << "按Enter 結束";
    cin >> mem;
    delete []mem;
    return 0;
  }
  catch(bad_alloc&) {
    cout << "記憶體配置失敗，程式結束!";
    return 255;
  }
}
```

24

13-4.cpp 的說明

- 此範例利用 C++ 的例外處理的機制來處理記憶體配置的錯誤。
  - 例外處理是 C++ 相對較新加入的功能，用來將所有程式錯誤狀況 (e.g. 除以0) 當成例外狀況 (exception)，在正常程式流程的最後再一起加以處理。
  - 例外處理機制可使正常程式與錯誤處理程式分開撰寫而不會交織在一起，使程式更容易理解與閱讀。
  - 目前還沒有正式學到例外處理，以後請自修 … :p
- 當記憶體配置失敗時，程式會跳躍到 catch (bad_alloc&) 的區塊執行，印出記憶體配置失敗的訊息。

25

摘要

- 動態記憶體管理
  - new ← → delete
    - a = new int; ← → delete a;
  - new ...[] ← → delete []...
    - a = new int[100]; ← → delete []a;
- 記憶體配置失敗
  - a = new (nothrow) int;← → if(a==0) 表失敗
  - 例外處理 (exception handling)

26

變數的儲存等級/生命週期

27

變數的儲存等級/生命週期

- 變數為運算電腦記憶體 (RAM) 的工具
- 變數儲存等級(storage class) / 生命週期代表了
  - 變數何時開始被創造出來 (並開始佔據記憶體)
  - 變數何時被消滅 (歸還其佔據的記憶體給作業系統)
- 在 C/C++ 裡，變數的儲存等級有三種：
  - **自動 (automatic)**：變數在該變數所屬的函式/區塊開始執行時創造出來，在函式/區塊結束執行後被消滅。此為預設之儲存等級，到目前為止我們大部份都是使用這種儲存等級的變數。
  - **動態 (dynamic)**：由程式作者決定一變數何時被創造出來、何時被消滅(new, delete, new[], delete[])。
  - **靜態 (static)**：變數在其被宣告的函式開始執行即被創造出來，並賦予初始值 (沒指定時為 0)，在程式結束執行後被消滅。

28

13-5.cpp

```
#include <iostream>
using namespace std;
void func() {
  int c = 5;
  cout << ++c << " ";
}

int main() {
  int a=3;

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

  return 0;
}
```

在此範例中，變數 a, c 的儲存等級皆為自動，簡稱為自動變數 (auto variables)

func 函式 開始執行：
- **自動**配置一個 4 bytes 記憶體給變數 c，並給初始值 5
- 將 c 加一後，列印出來
- 函式結束執行，**自動**將變數 c 消滅掉，將 4 bytes 的記憶體歸還給作業系統

main 函式 / 主程式開始執行：
- **自動**配置一個 4 bytes 記憶體給變數 a，並給初始值 3
- 將 a 變數的值列印出來
- 重複呼叫 func() 函式 5 次
- 程式結束執行，**自動**將 變數 a 消滅並將 4 bytes記憶體歸還給作業系統

程式執行結果
3 6 6 6 6 6

29

13-6.cpp

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

在此範例中，變數 a 為自動變數；變數 c 的儲存等級則為靜態，簡稱為靜態變數 (static variables)

func 函式 開始執行：
- 若此函式為第一次被呼叫，配置4 bytes 記憶體給變數 c。
- 將 c 加一後，列印出來。

main 函式 / 主程式開始執行：
- **自動**配置一個 4 bytes 記憶體給變數 a，並給初始值 3
- 將 a 變數的值列印出來
- 重複呼叫 func() 函式 5 次
- 主程式結束執行，**自動**將 變數 a 消滅並將4 bytes記憶體歸還給作業系統
- 將靜態變數 c 消滅，將 4 bytes 記憶體歸還

程式執行結果
3 6 7 8 9 10

30

靜態變數

- 在函式中的靜態變數在該函式第一次被呼叫時，被配置記憶體並進行初始化的動作，而待整個程式結束執行時，靜態變數的記憶體才會被釋放掉。
- 雖然靜態變數宣告在函式內，並不會隨著函式開始執行而被重新創造出來，也不會因為函式結束執行而被消滅掉。此與自動變數不同。
- 由於變數的初始化 (int c=5;) 是在變數被創造出來佔據記憶體時所執行的動作，因此在函式被重複呼叫時，變數的初始化是不會再被執行的。
- 靜態變數若沒有指定初始值時，會被初始化為 0。

31

檔案操作

32

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。

33

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

34

13-7.cpp - 檔案輸出

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  ofstream file;
  file.open("d:\\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 d:\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

35

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 (insertion) 與抽取 (extraction) 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("d:\\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

36

13-8.cpp - 將運算結果輸出至檔案

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

int main() {
  int i;
  unsigned long sum=0;
  ofstream outp;

  outp.open("d:\\test.dat");
  if(!outp) {
      cout << "檔案開啟錯誤";
      return 255;
  }
  for(i=1;i<=20000;i++) {
    sum += i;
    outp<< i <<", "<<sum<<"\n";
  }
  outp.close();

  return 0;
}
```

1, 1
2, 3
3, 6
4, 10
5, 15
.
.

37

13-9.cpp – 讀取檔案

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  ifstream file;                          宣告 file 為一輸入檔案串流物件
  file.open("d:\\test.txt");              嘗試開啟 d:\test.txt
  if (!file) {                            檢查檔案開啟是否成功
    cout << "檔案開啟錯誤!";               失敗時輸出一訊息、回傳非零值以代
    return 255;                             表程式異常結束並結束程式執行。
  }
  for (;;) {                              無窮迴圈
    int n, sum;                             宣告變數 (n, sum, c) 以記錄
    char c;                                 讀出的資料
    file >> n >> c >> sum;                  嘗試從串流取出資料存入n,c,sum
    if (file.eof()) break;                  若已達檔案結尾則跳離迴圈
    cout << n << ":" << sum << "\n";        輸出由檔案讀取的資料
  }
  file.close();                           檔案關閉
  return 0;                               程式正常結束
}
```

38

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
  - **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。
  - **eof**(): 回傳 true 時表示已到達檔案結尾。
  - **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。
- 前範例中僅使用了 eof() 檢查是否到達檔案結尾。

39

13-10.cpp – 對一檔案有讀也有寫 (1/2)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  int i;
  unsigned long sum;
  char comma;
  fstream file;

  file.open("d:\\test.dat", ios::in | ios::out);
  if (!file) {
    cout << "檔案開啟錯誤";
    return 255;
  }
```

- 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)

40

13-10.cpp – 對一檔案有讀也有寫 (2/2)

```
  for (;;) {
    file >> i >> comma >> sum;
    if (file.eof()) break;
  }
  cout << "\n最後一筆為" << i << ":" << sum;

  file.clear();
  for (int count = 0; count < 10000; ++count) {
    ++i;
    sum += i;
    file << i << ", " << sum << "\n";
  }

  file.close();
  return 0;
}
```

由於在讀取資料時，最後串流狀態已變為不是 good (eof!)，所以必需先將該不好的狀況清除掉後才能對檔案作操作。

41

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

42

格式化的輸入輸出

- 以上介紹的皆為格式化輸入輸出 (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。
  - 格式化指的是檔案內容可讓人容易讀得懂，對於電腦的處理上來說，實際上比較費事 (較慢)。電腦內部處理的二進位格式的資料，為了要作格式化的輸入輸出，要經過轉換的動作。
- 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

43

循序存取 vs. 隨機存取

- 以上所介紹的檔案存取方式為所謂的循序存取 (sequential access)
  - 當讀取資料時，是一筆一筆地從頭讀起
  - 即使僅需讀取第100個資料時，還是必需要先讀過之前的 99 筆資料。
- 隨機存取 (Random access)
  - 顧名思義，隨機存取為可以任意存取一筆資料而無需讀取不必要的資料。
  - 隨機存取的先決條件：可以透過計算取得想要資料的位置，亦即每一筆資料所佔的空間數目 (e.g. bytes) 要先知道。

44

多維陣列之動態記憶體管理

45

基本二維動態記憶體配置的方式

- 使用**不夠彈性**的多維動態記憶體管理 (行數不能修改)

```
int (*a)[5]= new int [10][5];
delete [] a;
```

- 使用指標陣列 (pointer array)、或稱為指標的指標 (pointer to pointers)

```
int **a = new int*[10];
for(int i=0;i<10;i++) a[i] = new int[5];
  …
for(int i=0;i<10;i++) delete []a[i];
delete []a;
```

46

不夠彈性的多維動態記憶體管理

13-11.cpp

```
#include <iostream>
using namespace std;
int main() {
  int (*a)[3] = new int[5][3];

  for(int i=0;i<5;i++) {
    for(int j=0;j<3;j++) {
      a[i][j] = i*3 + j;
    }
  }
  for(int i=0;i<5;i++) {
    for(int j=0;j<3;j++) {
      cout << a[i][j] << " ";
    }
    cout << endl;
  }

  delete []a;
  return 0;
}
```

(*a) 中的括弧不能省略!!

此方式無法完全動態地配置多維陣列
其中**僅有第一個維度**可以變數指定，
其它維度必需為常數 ... Orz

```
0 1 2
3 4 5
6 7 8
9 10 11
12 13 14
```

47

指標陣列

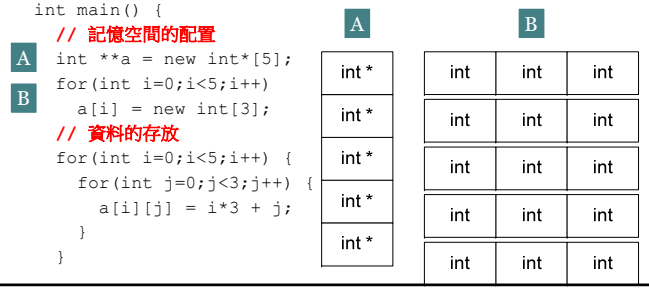
13-12.cpp

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

int main() {
  // 記憶空間的配置
  int **a = new int*[5];
  for(int i=0;i<5;i++)
    a[i] = new int[3];
  // 資料的存放
  for(int i=0;i<5;i++) {
    for(int j=0;j<3;j++) {
      a[i][j] = i*3 + j;
    }
  }
```

`int* *a;`
- 和 a 緊靠在一起的 * 表示 a 為一指標變數(使用起來 ~= 一維陣列)
- int* 代表一維陣列中所存放的資料為整數的指標 (~= 整數的一維陣列)

new int *[5]; ← 配置一個可以存放 5 個 int* 的記憶空間



48

13-12.cpp (cont'd)

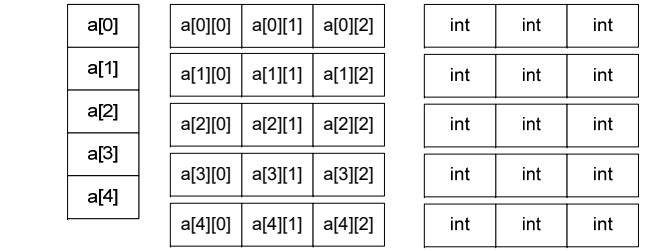
```
  // 資料的輸出
  for(int i=0;i<5;i++) {
    for(int j=0;j<3;j++) {
      cout << a[i][j] << " ";
    }
    cout << endl;
  }

  // 記憶空間的釋放
  for(int i=0;i<5;i++)
    delete []a[i];
  delete []a;
  return 0;
}
```

```
0 1 2
3 4 5
6 7 8
9 10 11
12 13 14
```

49

```
int* *a = new int*[5];
```




```
for(int i=0;i<5;i++)
    a[i] = new int[3];
```

```
a[0] = new int[3];
a[1] = new int[3];
a[2] = new int[3];
a[3] = new int[3];
a[4] = new int[3];
```

50

摘要 - 二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];
…
delete []a;
```

- 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(i=0;i<nRow;i++) delete []a[i];
delete []a;
```

51

課程總結

- ✓基本程式架構與輸入輸出
- ✓變數與算術運算
- ✓程式流程控制
- ✓迴圈與陣列
- ✓函式與標準函式庫 → API, Application Programming Interface
- ✓函式覆載與遞迴函式
- ✓參考型別、指標型別、函式呼叫時的資料傳遞
- ✓檔案操作
- ✓指標與動態記憶體配置
- ✓變數之儲存等級

52

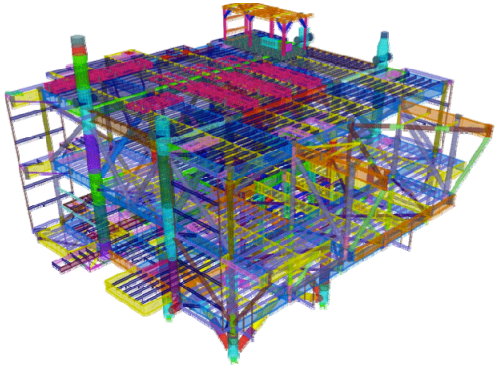
課程目的

學習電腦程式之寫作以

- 強化解析問題與電腦應用之能力;
- 提升資料/資訊處理的能力與速度;
- 進一步了解電腦程式運作之原理;
- 訓練表達能力;
- 期望將來能充份運用電腦協助解決工程問題。

53

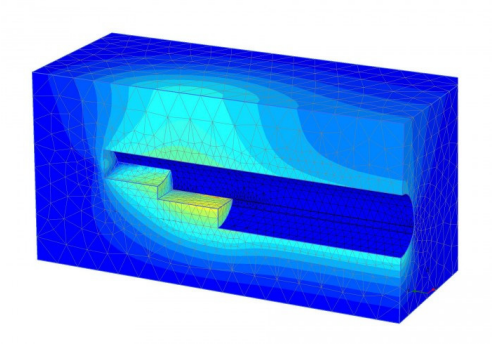
土木營建工程之應用電腦軟體 - 結構工程



https://www.csiamerica.com/products/sap2000/oil-and-gas

54

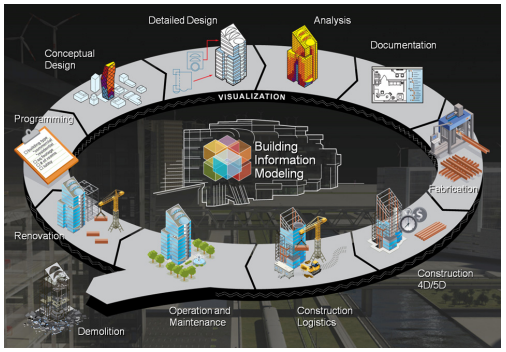
土木營建工程之應用電腦軟體 - 大地工程



http://www.plaxis.nl/files/images/img/img_000130_01.jpg

55

Building Information Modeling



56

6/9 期末考

- 檔案操作與之後的內容不會考
- 期中考之前的內容很難不考到
- 範例程式、隨堂練習、作業要懂，用背的絕對沒有用

57

隨堂練習

動態記憶體管理
檔案操作

58

1. 矩形梁的尺寸組合與重量限制

請撰寫一程式，由使用者輸入檔案名稱，之後讀取使用者指定檔案的內容。內容格式如下述，之後列出所有寬度(b)、高度(h)、長度(L)、密度(D)滿足重量限制的所有組合，以及各組合所對應的矩形梁之Ixx、Iyy、與重量。$Ixx=bh^3/12$、$Iyy=hb^3/12$、重量＝bhLD。

p.s. 1: 注意到由於每種參數的個數未知，必需使用動態記憶體配置。
p.s. 2: 本程式需動態配置四個一維陣列，以分別儲存b, h, L, D等不同的值
p.s. 3: 由於需要四種參數的所有組合，所以會需要四層的巢狀迴圈
p.s. 4: 有沒有注意到，我們到目前為止所有的程式，其實裡面都是沒有單位的！大家未來使用軟體時，務必注意手冊上說明的單位。有的軟體不管單位時，自己要注意各個輸入值使用的單位是否一致。

```
3 5 6 4 1000000
12 18 24
15 21 27 33 36
100 200 300 400 600 800
12.4 15.6 21.3 44.2
```

```
3 種寬度、5 種高度、6 種長度、4 種密度、重量限制
3 種寬度的值
5 種高度的值
6 種長度的值
4 種密度的值
```

59

ex.txt

```
3 5 7 4 500000
12 18 24
15 21 27 33 36
100 200 300 400 600 800 1000
12.4 15.6 21.3 44.2
```

請輸入檔案名稱：D:\ex.txt

```
  12   15  100      12.4      3375      2160         223200
  12   15  100      15.6      3375      2160         280800
  12   15  100      21.3      3375      2160         383400
  12   15  200      12.4      3375      2160         446400
  12   21  100      12.4      9261      3024         312480
  12   21  100      15.6      9261      3024         393120
  12   27  100      12.4     19683      3888         401760
  12   33  100      12.4     35937      4752         491040
  18   15  100      12.4    5062.5      7290         334800
  18   15  100      15.6    5062.5      7290         421200
  18   21  100      12.4   13891.5     10206         468720
  24   15  100      12.4      6750     17280         446400
```

總共組合有 420 組，滿足重量限制的有 12 組。

60

ex2.txt

```
5 7 3 3 100000
11.1 22.2 33.3 44.4 55.5
15 21 27 33 36 39 42
25.4 50.8 76.2
12.4 15.6 21.3
```

請輸入檔案名稱：d:\ex2.txt

```
 11.1   15 25.4      12.4   3121.88   1709.54        52440.8
 11.1   15 25.4      15.6   3121.88   1709.54          65974
 11.1   15 25.4      21.3   3121.88   1709.54        90079.8
 11.1   21 25.4      12.4   8566.42   2393.35        73417.2
 11.1   21 25.4      15.6   8566.42   2393.35        92363.5
 11.1   27 25.4      12.4   18206.8   3077.17        94393.5
```

總共組合有 315 組，滿足重量限制的有 6 組。