1

回顧 – 函式

- 函式 – 用來將程式獨立成為小單元、模組
  - 標準函式庫函式 – 提供 C/C++ 認為基本、必要的功能 – 輸入輸出、數學運算、 ...
  - 自訂函式 (user-defined functions)
- 函式的目的主要主於簡化程式發展的難度、並使多人可容易可以互相合作以發展大型程式
  - 各別擊破法 (divide and concur)

2

10-8.cpp

```
#include <iostream>
using namespace std;
bool isTriangle(double, double, double);
int main() {
    double a, b, c;
    cout << "請輸入三邊長: ";
    cin >> a >> b >> c;
    cout << a << ", " << b << ", " << c;
    if( isTriangle(a,b,c) )
        cout << " 可";
    else
        cout << " 不可";
    cout << "構成一個三角形。";

    return 0;
}
```

```
bool isTriangle(double a, double b, double c) {
    if(c >= (a+b)) return false;
    if(a >= (b+c)) return false;
    if(b >= (a+c)) return false;
    if((a-b) >= c) return false;
    if((b-c) >= a) return false;
    if((a-c) >= b) return false;
    return true;
}
```

1. 先想好主要的程式流程
2. 找出可獨立出來的程式單元 (isTriangle)
3. 定義每一程式單元的介面。(e.g. 函式的原型宣告 – 函式需要的傳入資料與回傳資料)
4. 分別實現不同的程式單元。(e.g. 函式定義)

此乃所謂的 top-down design (由上而下的程式設計方法)

3

回顧 – 函式

- 由於函式之間互相為獨立的單元，因此：
  - 函式定義時可以互相呼叫，但是任何一個函式定義不會把別的函式定義包起來 。
  - 函式定義時所使用的變數大部份是 local 的，亦即函式內所使用的變數互相獨立而不互相干擾 。
  - 一個函式要呼叫任何其它函式之前，至少要先有它的原型宣告被定義出來。

```
void funA(int, double);
void funB(void);
```

```
int main() {
    .
    funA(3, 5.0);
    funB();
    .
    .
}
```

```
void funA(int a, double b) {
    int c;
    double d;
    …
}
```

```
void funB() {
    double c;
    int d;
    …
}
```

這些區塊出現在同一個檔案裡哦！

4

補充

- 目前為止，所有的函式原型與函式定義都請大家寫在一個原始程式碼裡 (source code)。而真正大的程式開發：
  - 原型宣告被放在標頭檔 (header file) 裡, 通常為副檔名為 .h，或沒有副檔名 – e.g. iostream, cstdlib, cmath, ... 。用到的程式把它含括進來 (#include ...)
  - 而函式定義被放在
    - 分開的原始程式檔裡
    - 函式庫檔案裡 (library file)
- 例如大家作業使用的堆土機專案 ...

5

回顧

- 而為了能在不同的函式之間傳遞資料，而介紹了
  - 傳值呼叫 (call by value)
  - 傳址呼叫 (call by address)
  - 傳參考呼叫 (call by reference)
- 而為了達到傳址與傳參考呼叫，則先介紹了
  - 指標型別 (pointer)
  - 參考型別 (reference)

6

回顧 – 指標

- 為一特別的資料型別，代表一個變數存放的資料是記憶體位址，用來記錄一個資料存在電腦裡的位置 (印出時為16進位數字)。
- 宣告：
  - int A=3; double B=3.14; float D=1.732;
  - int *ptrA; double *ptrB, *ptrC; float *ptrD, **pptrD, ***ppptrD;
- 使用：
  - 給指標變數值 – 使用取址運算子 **&**：ptrA = **&**A; ptrB = **&**B; ptrC=ptrB; ptrD=**&**D; pptrD = **&**ptrD; ppptrD = **&**pptrD;
  - 使用指標變數的值 – 使用取值運算子 * (deferencing)：

```
cout << ptrA << ": " << *ptrA;          // 印出 0x7000fe24: 3
cout << *ptrB;                          // 印出 3.14
cout << *ptrC;                          // 一樣印出 3.14
cout << *ptrD;                          // 印出 1.732
cout << *pptrD << ": " << **pptrD;      // 印出 0x7000a000: 1.732
cout << ***ppptrD;                      // 印出 1.732
```

7

回顧 – 傳址呼叫 (以傳遞陣列為例)

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

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

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

8

回顧 – 參考

- 也是一特別的資料型別，代表一個變數為另一個變數的分身、別名。
- 宣告：
  - int A=3; double B=3.14; float D=1.732;
  - int &secretA=A; double &agentB = B; float &aliasD=D;
- 使用上和一般變數沒有兩樣！

```
int a = 3;
int &b = a;

cout << a << endl;
cout << b << endl;
b = 4;
cout << a << endl;
```

```
3
3

4
```

9

回顧 – 傳參考的函式呼叫

```
void circle(double, double &, double &);
int main() {
  double r, area, perimeter;
  cout << "請輸入半徑:";
  cin >> r;

  circle(r, area, perimeter);

  cout << "面積 = " << area << endl;
  cout << "圓周 = " << perimeter << endl;
  return 0;
}
```

透過傳參考呼叫，可讓函式回傳一個以上的資料！

```
void circle(double r, double &area, double &perimeter) {
    const double pi = 3.141592654;
    area = pi*r*r;
    perimeter = 2*pi*r;
}
```

10

Lecture 13

變數的儲存等級/ 生命週期
靜態變數
動態記憶體管理

11

今日內容

- 變數的儲存等級/ 生命週期
  - 自動變數、靜態變數、動態記憶體管理
- 靜態變數 (static) 的使用
- 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)
  - 多維陣列的記憶體管理

12

變數的儲存等級/生命週期

13

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，變數的儲存等級主要有三種：
  - **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/內定之儲存等級，到目前為止我們都是使用這種儲存等級的變數。
  - **靜態 (static)**：變數在程式開始執行即被創造出來，在程式結束執行後被消滅。
  - **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。

14

13-1.cpp

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)

```
#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;
}
```

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 記憶歸還給作業系統

程式執行結果
3 6 6 6 6 6

15

靜態變數 (static) 的使用

16

13-2.cpp

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)

```
#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;
}
```

func 函式 開始執行：
- 將 c 加一後，列印出來

程式開始執行前，配置記憶給所有的靜態變數 (此例為func() 中的變數 c)，並將之初始化。
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

17

靜態變數

- 靜態變數在程式載入開始執行**前**，即已被創造出來並佔據記憶體
- 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。
- 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。
- 靜態變數若沒有初始化時，會被初始化為 0。

18

動態記憶體管理

Dynamic Memory Management

19

13-3.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。

20

13-3a.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。

21

13-3.cpp

- 在之前使用指標變數時，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 才正確。
  - delete b;

22

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);

23

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。

24

13-4.cpp

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

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

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

  myData = new int[size];
```

```
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,
```

25

13-4.cpp

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

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

  delete []myData;

  return 0;
}
```

26

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。
  - 無法指定初始值！

27

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;
  - double *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。!!!!

28

更多範例

- 需要存放 200 個字元 (199 個字的字串)
  - 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;

29

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 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程式內宣告的陣列 (使用 stack memory)

- 在 C++ 裡，有兩種方式處理記憶體配置失敗
  - (nothrow) 參數 → 13-5.cpp
  - 例外處理 (Exception handling) → 13-6.cpp (僅供參考 ...)

30

13-5.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";
```

31

13-5.cpp

```
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;
}
```

32

13-5.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 0 pointer (null pointer)，null pointer 其值為 0。

33

13-6.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";
```

34

13-6.cpp

```
    mem = new char[size];
    for(i=0;i<size;i++) {
      mem[i] = i;
    }
    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;
  }
}
```

35

13-6.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++ 新加入的功能，用來將所有程式錯誤狀況 (e.g. 除以0) 當成一個一個的例外狀況 (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。

36

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

37

隨堂練習

38

動態記憶體配置

- 主程式已貼在 BB 的 5/27 隨堂練習，請完成以下兩個函式的定義：

```
void showData(int n, int *data);
int* getEven(int &nEven, int n, int *data);
```

- 其中，showData 會將傳入 data 指標位置上的前 n 個整數印出來。
  - int a[] = {5,4,3,2,1};
  - showData(5, a); // 會印出 5 4 3 2 1 後換行
- 而 getEven 比較複雜，它會將傳入 data 指標位置上的前 n 個整數中所有的偶數找出來存入一新的大小剛剛好的陣列裡 (動態配置出來的記憶體位置)，並將偶數個數放在 nEven 變數裡回傳回來。
  - int nEven;
  - int *evenData = getEven(nEven, 5, a);
  - showData(nEven, evenData); // 會印出 4 2 後換行

39

```
.
.
.

  cout << "\n請輸入你要輸入的整數資料個數: ";
  cin >> n;
  data = new int [n];
  for(int i=0;i<n;i++) {
    cout << "請輸入第 " << i+1 << " 個資料: ";
    cin >> data[i];
  }

  cout << "\n你輸入的資料為: ";
  showData(n, data);
  evenData = getEven(nEven, n, data);

  cout << "\n其中共有 " << nEven << " 個偶數。他們為: ";
  showData(nEven, evenData);

  delete []data; delete []evenData;
  return 0;
}
```

40

```
請輸入你要輸入的整數資料個數: 10
請輸入第 1 個資料: 1
請輸入第 2 個資料: 2
請輸入第 3 個資料: 4
請輸入第 4 個資料: 5
請輸入第 5 個資料: 7
請輸入第 6 個資料: 9
請輸入第 7 個資料: 10
請輸入第 8 個資料: 12
請輸入第 9 個資料: 14
請輸入第 10 個資料: 15

你輸入的資料為: 1 2 4 5 7 9 10 12 14 15

其中共有 5 個偶數。他們為: 2 4 10 12 14
```