1

回顧 –
自訂函式 (user-defined function) 包含:

- 函式的原型宣告 (prototype)
```
int APlusB (int A, int B);
```
- 函式的定義 (definition)
```
int APlusB(int A, int B) {
  return A+B;
}
```
- 函式呼叫 (function invocation)
```
A = APlusB(3, 5);
```

2

函式 (user-defined function)

- 函式可以有預設/內定參數值
```
int APlusB (int A, int B=4);
```
- 函式內宣告的變數為區域變數，不同函式之間的區域變數互相獨立，即使它們同名也沒有關係。
```
int APlusB(int A, int B) {
  return A+B;
}
int AMinusB(int A, int B) {
  return A-B;
}
```
- 函式覆載 (overloading)
```
double area(double r);
double area(double width, double height);
```

3

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)

4

11-6.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)

5

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！

6

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 ...

7

Lecture 12

函式呼叫的資料傳遞 (I) – 傳值
參考型別
函式呼叫的資料傳遞 (II) – 傳參考
遞迴函式

8

函式的資料傳遞 (I)

傳值呼叫 (call by value)

9

函式定義

```
int sum(int x) {
  int i, sum = 0;
  for(i=1;i<=x;i++) {
      sum += i;
  }
  return sum;
}
```

函式定義的介面決定了資料如何**進入**此函式、以及運算結果回傳給呼叫者 (caller)。

此參數列可以將資料傳入進行運算，也可用來將資料傳出！(這個範例不行！)

此為函式傳出資料的途徑一，使用 **=** 將回傳的資料存入一指定變數。
e.g. q = sum(200);

10

12-1.cpp

```
#include <iostream>
using namespace std;
int test(int b) {
  b = b * 2;
  return b+1;
}
int main() {
  int a = 3;
  int b;
  b = test(a);
  cout << "a=" << a << endl;
  cout << "b=" << b << endl;
  return 0;
}
```

int b = a;

- 在 C/C++ 裡, 參數的傳遞內定的方法是**傳值呼叫 (call by value)**。
- 在 test 函式被呼叫時, 主程式傳入的 a 變數的值被**複製**一份到函式裡區域變數 b ，然後才開始執行 test 函式的內容。
- test 函式結束時將 b+1 的值透過 return 的功能傳回主程式中，並在此範例中存入主程式中的 b 變數，然後test 函式中的 b 變數即被消滅。
- 因為是**傳值呼叫**，無論test函式裡的 b 變數如何改變，都不會影響主程式裡 a 變數的值。

a=3
b=7

11

參考型別

12

12-2.cpp

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

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

  cout << a << endl;
  cout << b << endl;
  b = 4;
  cout << a << endl;
  return 0;
}
```

變數 b 為變數 a 的參考
也可說變數 b 為變數 a 的別名、分身

3
3
4

13

12-2.cpp 的說明

- 宣告 b 為一參考型別的變數，其參考的變數為 a
- 參考型別在宣告時，一定要指定參考那一個變數!
- 宣告出來的參考型別變數 b, 為原來變數 a 的分身。
- 有時, 亦可說 b 為 a 的別名 (alias)
- 所以，改變參考型別變數 b 的值，被參考到的變數 a 的值也會一起跟著改變。

14

12-3.cpp

```
#include <iostream>
using namespace std;
int test(int &b) {
  b = b * 2;
  return b+1;
}
int main() {
  int a = 3;
  int b;
  b = test(a);
  cout << "a=" << a << endl;
  cout << "b=" << b << endl;
  return 0;
}
```

int &b = a;

- 此範例為**傳參考**的函式呼叫。
- 在 test 函式被呼叫時, 主程式傳入的 a 變數成為 b 這個參考型別的變數初始值 ，因此 b 變數在 test 函式中，為主程式中的 a 變數的分身。
- test 函式結束時將 b+1 的值透過 return 的功能傳回主程式中，並在此範例中存入主程式中的 b 變數。
- 因為是傳參考呼叫，所以在 test 函式中對於 b 變數的改變，實際上就是對主程式中的 a 變數作改變。

a=6
b=7

15

函式的資料傳遞 (II)

傳參考呼叫 (call by reference)

16

參數傳遞的方式有三種

- **傳值 (call by value)**
  - 此為預設方式, 將傳入的變數值**拷貝**一份到函式內的區域變數 (local variable).
- **傳參考 (call by reference)**
  - 使用參考型別的變數, 將函式的區域參考變數參考到呼叫者的變數
- ***傳址 (call by address)***
  - *使用指標型別的變數, 將變數的記憶體位址拷貝一份到函式內的區域指標變數*

17

12-4.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;
}
```

18

12-4.cpp (cont'd)

```
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=3, b=7
```

此範例例亦為傳值呼叫 (call by value)，所以雖然在 swap 函式裡成功將傳入的兩個變數交換了，在主程式中的兩個變數並沒有交換。

19

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;
}
```

20

12-5.cpp (cont'd)

```
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
```

此範例亦為傳參考呼叫 (call by reference)，所以在 swap 函式裡成功將傳入的兩個變數交換了，在主程式中的兩個變數也成功地交換，因為函式裡的 a, b 為主程式中 a, b 的分身。

21

小結

- 函式呼叫使用傳值法 (C/C++ 的內定方法) 時，我們只能傳回一個運算結果。
- 傳值呼叫**不可能**意外改到呼叫者 (e.g. main) 內的變數值。
- 傳參考呼叫方法則可能可以傳回一個以上的運算結果，並可以改變呼叫者內的變數值。

22

12-6.cpp

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

  circle(r, area, perimeter);// 如何寫這個函式？

  cout << "面積 = " << area << endl;
  cout << "圓周 = " << perimeter << endl;

  return 0;
}
```

23

12-6.cpp (cont'd)

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

24

遞迴函式

Recursive Function

25

遞迴函式
Recursive Function

- 何謂遞迴函式?
  - 一個函式呼叫自己，即為遞迴函式。
    `int fun(int a) { return fun(a-1); }`
  - 有些數學上的定義本來就以遞迴定義。
    等差級數、 等比級數、 Fibonacci 級數、 ...

$a_i = a_{i-1} + b$ $a_i = b \times a_{i-1}$ $a_i = a_{i-1} + a_{i-2}$

- 範例：
  - 12-7.cpp: 計算階乘
    - 5! =
  - 12-8.cpp: Fibonacci 數列
    - 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

26

12-7.cpp

```
int fact(int n) {
  int ans = 1;
  for(;n>0;n--) ans *= n;
  return ans;
}
```

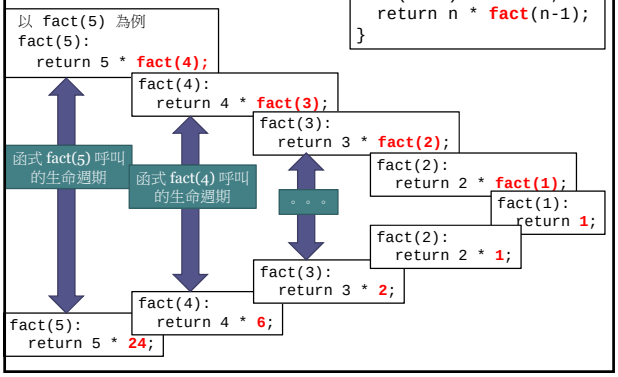
```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

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

int fact(int n) {
  if(n==1) return 1;
  return n * fact(n-1);
}

int main() {
  for(int i=1;i<=10;i++) {
    cout << i << "! = " << fact(i) << endl;
  }
  return 0;
}
```

27

12-7.cpp

```
int fact(int n) {
  if(n==1) return 1;
  return n * fact(n-1);
}
```



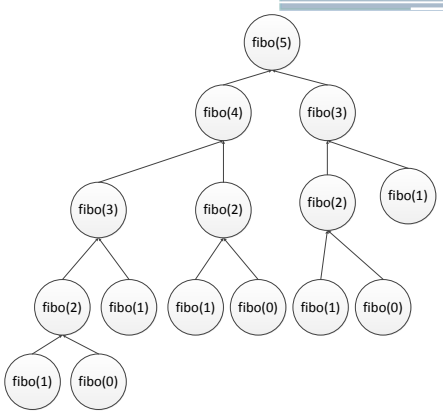
28

12-8.cpp

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

unsigned int fibo(unsigned int n) {
  if(n==0) return 0;
  else if(n==1) return 1;
  else return fibo(n-1) + fibo(n-2);
}

int main() {
  cout << fibo(5) << endl;

  return 0;
}
```

```
5
```

29



30

撰寫遞迴函式的注意事項

- 遞迴函式必需有「終止」條件，電腦不能無窮遞迴下去!
  - if(n==1) return 1; else ...
  - if(n==0) return 0; else if(n==1) return 1; else return ...
- 遞迴的「深度」不能太深，否則程式執行會發生錯誤
  - stack overflow! (堆疊溢位)
- 遞迴的寫法會讓程式看起來比較簡短，但執行效率不見得比較高；遞迴常可使用迴圈取代。

31

作業七

Due: 5/18/2012

32

隨堂練習

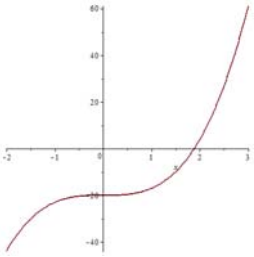
33

以二分逼近法尋求方程式的根

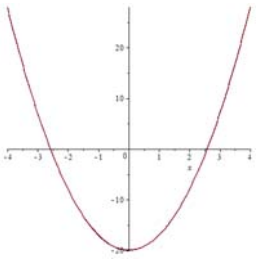
1. 請撰寫一函式 f，傳入 x，並回傳 $3x^3-20$ 的值。
2. 撰寫一函式 findRoot，以在[x1,x2] 區間內找解 x，找到解回傳 true，否則回傳 false ，其原型宣告如下：
   bool findRoot(const double x1, const double x2, double &x);
   其中：
   a) 令區間 [x1, x2] 的中點為 x3
   b) 判斷 f(x3) 是否為解，若是，則令 x = x3，並回傳 true。
   c) 若 [x1,x3] 區間有解，則以二分搜尋法找 [x1,x3] 區間內的解
   d) 否則若 [x2,x3] 區間有解，則以二分搜尋法找 [x2,x3] 區間內的解
   e) 否則回傳無解
3. 請撰寫主程式，利用 findRoot 函式，找出 $3x^3-20$ 在 [-1,4] 是否有解，以及解的位置。

34

y=3x³-20, x=[-2,3]

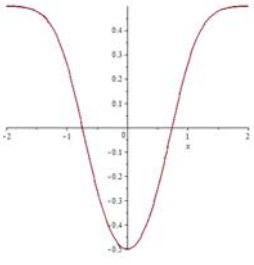


y=3x²-20, x=[-4,4]



35

y=tanh(x²)-0.5, x=[-2,2]



y=sin(x)+0.5*cos(2*x), x=[-2,4]