

函式

各個擊破 是解決真實問題電腦程式的好 方法

真實的電腦程式比我們的練習題大很多，維護大型程式的方法：
以較小的單元或模組來建構程式

小單元要比大程式好管理




函式

C/C++ 語言 = C 標準函式庫 + 程式設計師所寫
的新函式

main()是一個函式


```
#include <iostream>
using namespace std;
int main() {
    int x;
    cin>>x;
    cout<<x;
    return 0;
}
```



main()是程式設計師自行發展的主程式，主程式是函式的一種類型，主程式配合標準函式庫stdio.h產生一個程式的新功能

#include <iostream>是什麼？

```
#include <iostream>
using namespace std;
int main() {
    int x;
    cin>>x;
    cout<<x;
    return 0;
}
```



函式庫可以想像
成一個人才庫，
有許多專家在那。

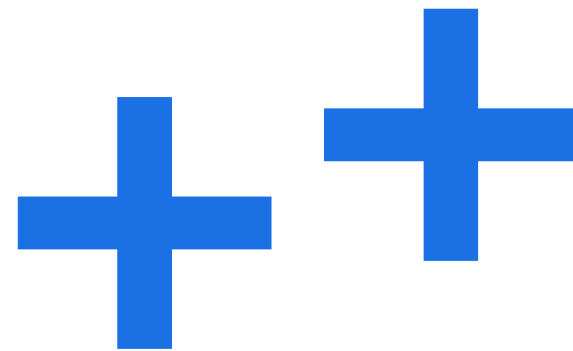
- 輸出/入函式來自於iostream函式庫。
- include是將iostream函式庫引入程式，因為cin與cout來自於iostream函式庫。

標準函式庫

- 包含
 - 常用的數學運算
 - 字串處理
 - 字元處理
 - 輸入輸出
- 目的
 - 提供程式設計師所需要的大部分功能，減輕程式設計師的負擔



自訂函式



自訂函式-又稱為副程式(子程式)結構

- **副程式(子程式)結構**

- 如果有一些程式段在不同程式的地方反覆出現，可以將這些程式段做為相對獨立的整體

- **好處**

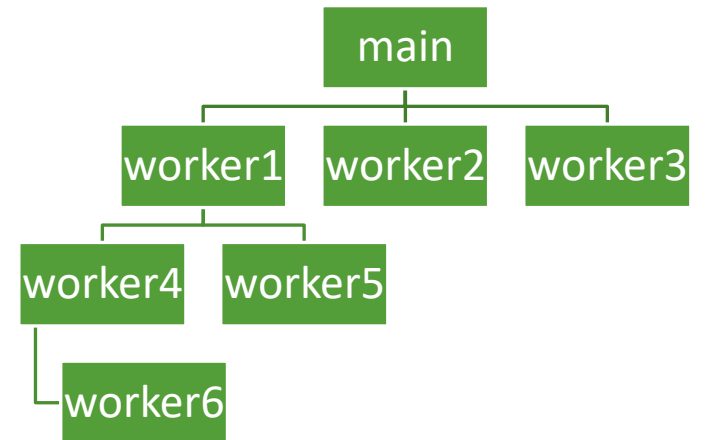
- 各個擊破式使得程式發展更容易被管理
- 增加軟體的重複使用性
- 可以減少重複程式的撰寫

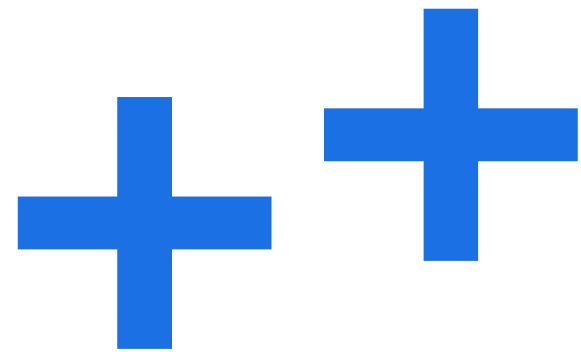
函式的使用概念

函式經由呼叫(function call)的方式引用(invoked)，函式的呼叫指明了要引用的函式名稱，並提供資訊（當作引數）給受呼叫函式，以執行受呼叫函式的工作

道理十分類似人事管理，老闆（呼叫函式或呼叫者）要求某位員工（受呼叫的函式）去執行某項工作，並在工作後完成回報。

例如老闆函式呼叫員工函式printf去執行這項工作，然後printf將資訊顯示出來，並且在顯示之後回報、返回或呼叫函式，但老闆函式並不知道員工函式如何執行，老闆函式也可以再呼叫其他函式





函式的定義



函式定義的語法形式

傳回值的資料類型 函式名稱(參數列表)

{

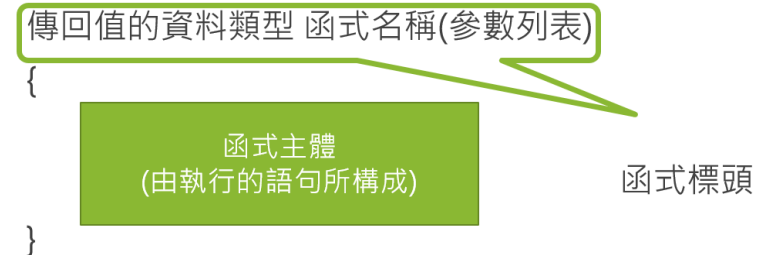
函式主體
(由執行的語句所構成)

}

函式標頭

函式定義的語法形式

- **傳回值的資料類型**：
函式傳回主程式值的類型，若是void則表示無返回值
- **函式名稱**
主程式的名字一定是main，其餘函式的名字則依照標示符的取名規則
- **參數列表**
 - 可以是空的，不管有沒有參數，()都不能去掉
 - 也可以多個，參數之間以逗號隔開
 - 參數必須有類型說明，可以是變數、陣列或指標
- 函式最外層是一對{ }



函式的宣告和呼叫

函式的宣告

寫法1：

先宣告，副程式寫在主程式後

```
#include <iostream>
using namespace std;
int slave();
int main() {
    cout<<slave()<<endl;
    return 0;
}
int slave() {
    return (1+10)*10/2;
}
```

寫法2：

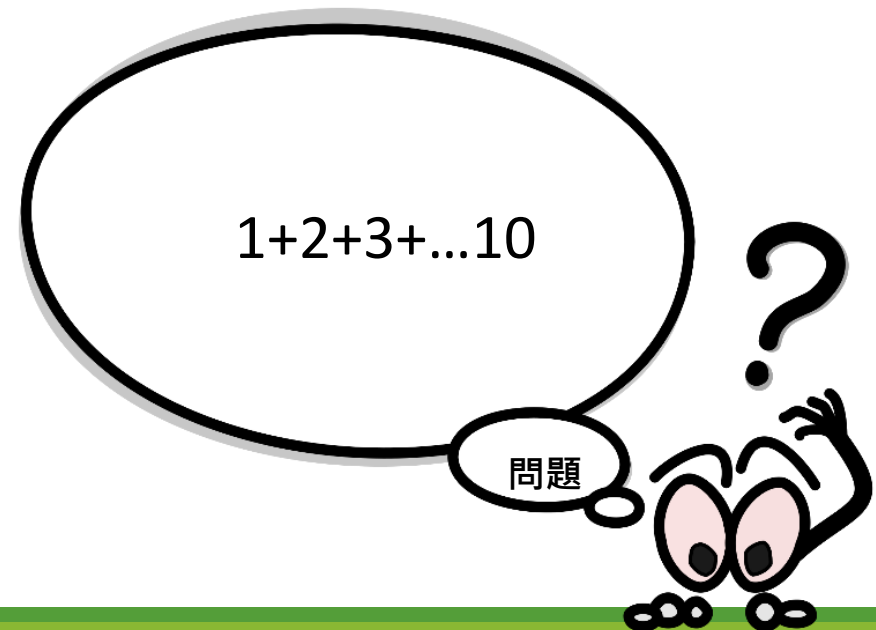
將副程式提到主程式之前

```
#include <iostream>
using namespace std;
int slave() {
    return (1+10)*10/2;
}
int main() {
    cout<<slave()<<endl;
    return 0;
}
```

函式的形式(類型1)- 無參數函式，有傳值回主程式

```
#include <iostream>
using namespace std;
int slave() 1
int main() { 2
    cout<<slave()<<endl;
    return 0;
}
int slave() {
    return (1+10)*10/2;
}
```

序號	說明
1	呼叫副程式slave
2	傳回slave副程式執行結果
3	傳回值是一個整數類型



函式的形式- (類型2)有參數函式與傳值回主程式

```
#include<iostream>
#include<iomanip>
using namespace std;
float area3(float r, float h);
float area2(float r);
int main()
{
    float p=3.14, r, h;
    int i;
    for(i=1; i<=10; i++) {
        cin>>r>>h;
        cout<<"圓柱體體積"<<fixed<<setprecision(2)<<area3(r, h)<<endl;
    }
    return 0;
}
float area3(float r, float h){
    return area2(r)*h;
}
float area2(float r){
    float p=3.14;
    return p*r*r;
}
```

序號	說明
1	呼叫副程式area3(r,h) area3含有2個參數，分別代表半徑與高
2	有參函式中必須定義半徑與高的資料類型
3	副程式可以再細分工作呼叫另一個副程式

由主程式呼叫一個圓柱體體積的函式area3()，再由area3()呼叫計算面積函式area2()而完成體積計算，程式可以重複運行10次。

問題



函式的形式- (類型3) -沒有傳值回主程式

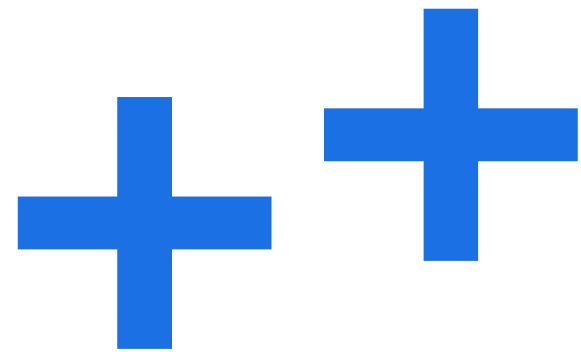
```
#include<iostream>
using namespace std;
void crazyprint();
void crazyprint() {
    cout<<"為什麼要副程式，方便我瘋狂列印。\\n";
}
int main()
{
    int i,n;
    cin>>n;
    for(i=1;i<=n;i++){
        crazyprint();
    }
    return 0;
}
```

void代表沒有傳回值

使用自定函式的方法，寫一個 crazyprint() 函式，輸出「為什麼要副程式，方便我瘋狂列印。」
輸入一個整數，決定輸出次數。

問題





細說函式-全域與區域變數



全域變數

- 定義：
在函式外部，沒有被括弧括起來的變數。
- 作用域：
是從變數的定義開始到檔案結束。
- 程式中的任何函式都可以使用。

全域變數

```
#include <iostream>
using namespace std;
int x,y;
int gcd(int x, int y)
{
    int r=x%y;
    while (r!=0)
    {
        x=y;
        y=r;
        r=x%y;
    }
    return y;
}
int lcm()
{
    return x*y/gcd(x,y);
}
int main()
{
    cin>>x>>y;
    cout<<lcm()<<endl;
    return 0;
}
```

區域變數

全域變數的說明

- 在一個函式內部，既可以使用函式定義的區域變數，也可以使用在此函式前定義的全域變數。
- 使用時機
使得函式間多一種傳遞資訊的方式，如果一個程式多個函式都要對一個變數進行處理。
- 若沒有設定賦值，其預設值是0。
- 執行過程中一直佔用記憶體。
- 副作用
 - 會增加調試困難
 - 降低程式的通用性

區域變數

- 作用域是在定義該變數的函式內部。函式執行完畢，區域變數的空間就被釋放，值無法被保留到下次使用。
- 在不同的函式中變數名可以相同，記憶體中佔據不同的記憶體單元，互不干擾。
- 區域變數與全域變數是可以重名的，在相同作用域內，區域變數有效時，對全域變數無效。
- 副程式中定義的變數的存在時間和作用會被限制在該區塊中。
- 主程式`main()`中的定義是一種區域變數。
- 若沒有給賦值，區域變數值是隨機的，區域變數受棧空間大小限制，大陣列要注意。

全域變數 / 區域變數

- 相同：

- 需要遵守變數的命名規則

- 相異：

- 變數範圍(scope)不同：scope 指的是變數可被使用或呼叫的地方，全域變數是在整份程式內都可以使用，區域變數只能在被定義的函示中使用。