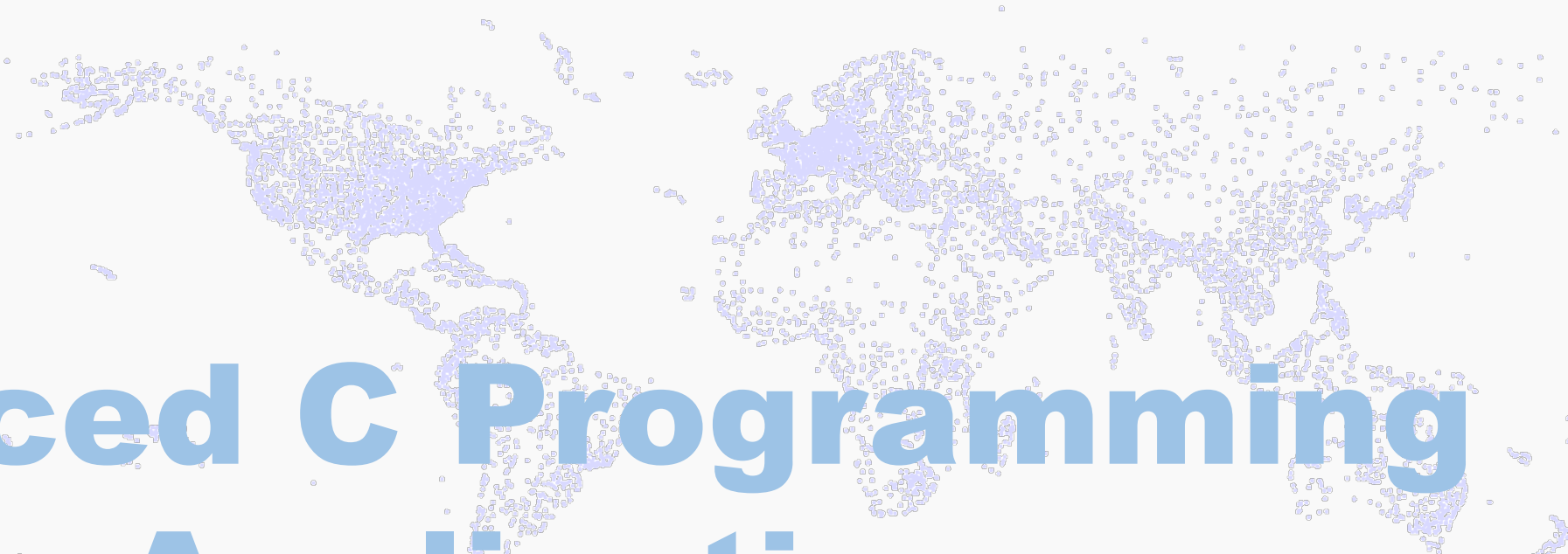


<Adv C & App/>



# Advanced C Programming And It's Application

**Function Advance**

Assistant Prof. Chan, Chun-Hsiang

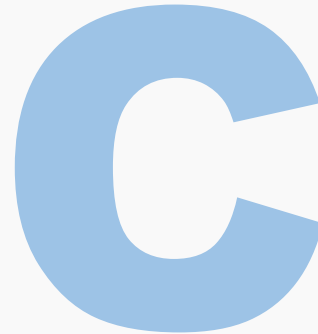
*Department of Artificial Intelligence, Tamkang University*

*Oct. 20, 2021*

</ Adv C & App >

# 大綱

- [1] Global and local variables
- [2] Recursive function
- [3] Standard Library
- [4] Assignments



## <Global and local variables/>

# Global and local variables

What's the definition of global and local variables?

```
#include <stdio.h>
```

```
//global variables
```

```
int foo(void){
```

```
    //local variables
```

```
}
```

```
int main(void){
```

```
    //local variables
```

```
}
```

Those variables declared **inside** function and **outside** function body are **local** and **global** variables, respectively.

所以簡單的區分就是，這個變數是在函數裡面還是外面被宣告？

**Local variable** 區域變數

**Global variable** 全域變數

# Global and local variables

What's the characteristics of global and local variables?

```
#include <stdio.h>
//global variables
```

```
int foo(void){
    //local variables
}
```

```
int main(void){
    //local variables
}
```

## Local variable 區域變數

- (1) 只有函數內可以呼叫的到；反之，跨函數就無法呼叫到。
- (2) 只有當開始定義函數的時候，才會初始化變數；當離開被定義函數時，該變數就會被刪除。

## Global variable 全域變數

- (1) 整個程式都可以呼叫的到。
- (2) 整個程式一開始就會初始化變數 (before main())，當離開程式才會被刪除 (after main())。

<Global and local variables/>

# Global and local variables

換一個視角來看global and local variables ◦

```
#include <stdio.h>
```

```
//global variables declare here!
```

```
int foo(void){
```

```
    //local variables declare here!
```

```
    //we can see global variables!}
```

```
int main(void){
```

```
    //local variables declare here!
```

```
    //we can see global variables!}
```

<global, local, and block variables/>

# Global and local variables

```

1  #include <stdio.h>
2  //global variables
3  int a = 1;
4  int b = 5;
5  char c[2] = "s";
6
7  int foo(void){
8      int a = 3;
9      int x = 10;
10
11     printf("\nWithin foo function!\n");
12     printf("a = %d; b = %d; c = %s; x = %d\n\n", a, b, c, x);
13 }
14
15 int main(void){
16     /*Ex 4-1: Global and local variables */
17     /* Demonstration - The differences between global and local variables*/
18
19     //local variables
20     float h = 1.67;
21     float w = 75;
22
23     printf("Ex 4-1: Global and local variables\n");
24     printf("Before calling foo function!\n");
25     printf("a = %d; b = %d; c = %s\n", a, b, c); //x = %d, x
26     printf("h = %f; w = %f\n", h, w);
27
28     foo();
29
30     printf("After calling foo function!\n");
31     printf("a = %d; b = %d; c = %s\n", a, b, c);
32     printf("h = %f; w = %f\n", h, w);
33 }

```

全域變數宣告在此

子函數foo  
函數內變數為區域變數

主程式main  
函數內變數為區域變數

</global, local, and block variables>

## &lt;Global and local variables/&gt;

## Global and local variables

**Lab 4-1:**

請嘗試回答  
這些程式的  
輸出結果:

```

1  #include <stdio.h>
2  int a = 7;
3  int c = 12;
4
5  int foo(void){
6      int a = 5;
7      int b = 10;
8  }
9
10 int main(void){
11     /*Lab 4-1-1: Global and local variables */
12     /* Demonstration - The differences between
13     global and local variables*/
14     int x = a + c;
15     int y = a + b;
16
17     printf("Lab 4-1-1: Global and local variables\n");
18     printf("Before calling foo()\n");
19     printf("x = a + c = %d\n", x);
20     printf("y = a + b = %d\n", y);
21
22     foo();
23     x = a + c;
24     y = a + b;
25     printf("After calling foo()\n");
26     printf("x = a + c = %d\n", x);
27     printf("y = a + b = %d\n", y);
28 }

```

(a)

```

1  #include <stdio.h>
2  int a = 7;
3  int c = 12;
4  int b = 15;
5
6  int foo(void){
7      int a = 5;
8      int b = 10;
9      return b;
10 }
11
12 int main(void){
13     /*Lab 4-1-2: Global and local variables */
14     /* Demonstration - The differences between
15     global and local variables*/
16     int x = a + c;
17     int y = a + b;
18
19     printf("Lab 4-1-2: Global and local variables\n");
20     printf("Before calling foo()\n");
21     printf("x = a + c = %d\n", x);
22     printf("y = a + b = %d\n", y);
23
24     foo();
25     x = a + c;
26     y = a + b;
27     printf("After calling foo()\n");
28     printf("x = a + c = %d\n", x);
29     printf("y = a + b = %d\n", y);
30 }

```

(b)

## Global and local variables

**Global variable** 很好用，所以大部分人剛學程式喜歡大量使用；然而在程式設計中，通常會希望大家不要那麼頻繁的定義**Global variable**，或許第一個閃過你頭腦的問題是：

“唉，不用**global**，反而使用很多**local**不是會佔用更多的記憶體空間嗎？”

那你可以換個角度想一下，現在的程式都很簡單，所以用到的變數與計算也很少。如果今天你需要計算你股票組合的獲利率，假設你有買**10**種不同的股票，難道你會每一種股票都另一個變數嗎？

如果不會，你會怎麼做呢？



# Global and local variables

這時候我們要提到 **Global variable minimization**，**Global variable**的壞處就是，一旦宣告(除非你把記憶體空間釋出)，否則他將會一直佔用該記憶體空間與變數名稱。

Think一下!

# Recursive function

遞迴函數(Recursive function)是一個會自己不停迭代的函數，這個或許大家有點難想像。我們來用一個高中數學中的排列組合來舉例，可能較比較能理解迭代的概念。

**Q:** 今天有三個顏色的球: 藍色、紅色、黃色，要排成一直線請問有幾種排法?

**A:**  $3! = 3 \times 2 \times 1 = 6$

**!** → 階乘 (factorial) 定義:  $n! \equiv n \times (n-1) \times (n-2) \times \dots \times 1$   
 $\equiv \prod_{i=1}^n i$

# Recursive function

$\prod_{i=1}^n i$  如果我們要做出一個自己用的階乘函數，你會怎麼做呢？

一般來說，我們應該會想要 **loop** 來做階乘的計算對吧？  
因為**loop**可以讓變數慢慢增加，我們再將這些數字相乘再一起就可以做到階乘的效果。

# Recursive function

## Lab 4-2:

請嘗試自定義一個階乘的函數，名稱為`my_factorial()`。在`main()`主函數中，呼叫`my_factorial`並計算下列三個數學題目，印出結果。

(1) 5!

(2) 8!

(3) 10!

† 利用For loop設計`my_factorial()`。

<Recursive function/>

# Recursive function

經過剛剛的 Lab 4-2 有沒有覺得，這樣的設計很沒有效率？  
在程式設計的世界裡，程式碼越少越好。Recursive function在此就扮演一個重要的角色，看看以下的範例：

```
#include <stdio.h>
```

```
int my_factorial(int n){  
    if(n<=1){  
        return 1;  
    } else {  
        return n*my_factorial(n-1);  
    }  
}
```

```
int main(void){  
    /*Ex 4-2: Recursive function */  
    /* Factorial – recursive version*/  
    printf( "Ex 4-2: Global and local  
variables\n");  
    printf("5! = %d\n", my_factorial(5));  
    printf("8! = %d\n", my_factorial(8));  
    printf("10! = %d\n", my_factorial(10));  
}
```

# Recursive function

## Lab 4-3:

請嘗試自定義一個加總的函數，名稱為my\_sum()。在main()主函數中，呼叫my\_sum並計算下列三個數學題目，印出結果。

(1)  $\sum_{i=1}^{10} i$

(2)  $\sum_{i=1}^{30} i$

(3)  $\sum_{i=1}^{100} i$

† 利用While loop設計my\_sum()。

# Recursive function

## Lab 4-4:

請嘗試自定義一個加總的函數，名稱為my\_sum()。在main()主函數中，呼叫my\_sum並計算下列三個數學題目，印出結果。

(1)  $\sum_{i=1}^{10} i$

(2)  $\sum_{i=1}^{30} i$

(3)  $\sum_{i=1}^{100} i$

† 利用Recursive function設計my\_sum()。

# Standard Library in C

**Standard Library**就是基本函數庫，舉凡我們平常用的**printf**都是基本函數之一。在這堂課我們會介紹一些常用且比較簡單的函數，另外會教大家如何看懂**library**的**document**。

包含：

(1) **ctype.h**

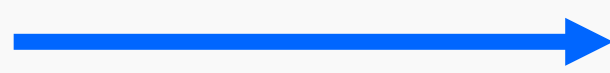
(2) **math.h**

(3) **stdlib.h**

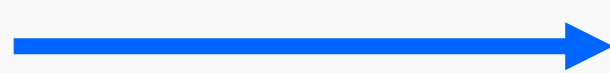
(4) **stdio.h**

(5) **string.h**

(6) **time.h**



之前的課程已經介紹了許多!



之後的課程會介紹!



## 作業一 Fibonacci number

請利用今天上課教的兩種方式完成my\_fibon()。

**my\_fibon()** :: accomplished with **loop approaches** (e.g., for or while loop)

(1) Call **my\_fibon()** and calculate questions in **main()**

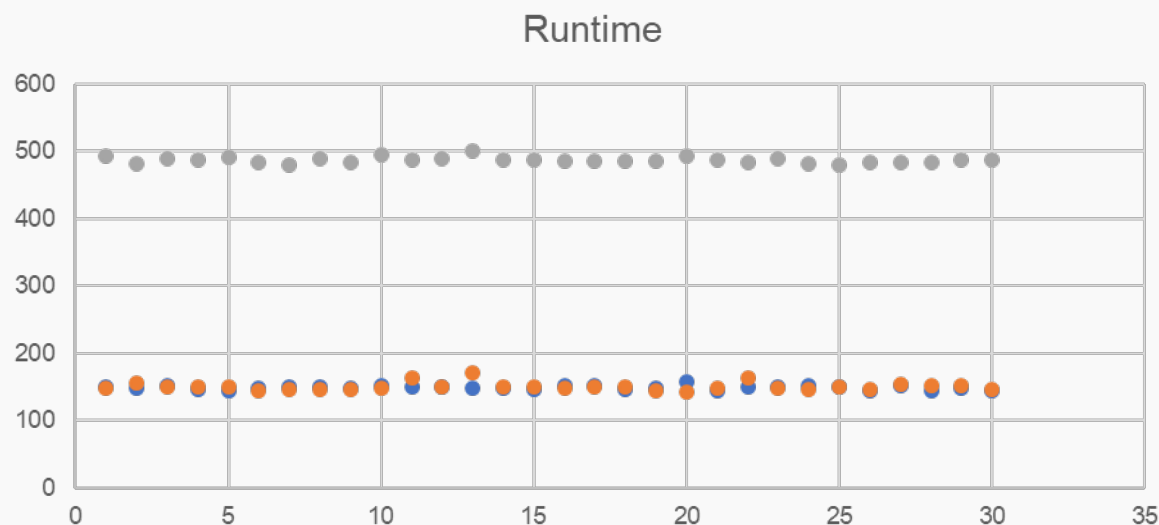
**my\_fibon()** :: accomplished with **recursive function**

(2) Call **my\_fibon()** and calculate questions in **main()**

## &lt;Assignments/&gt;

## 作業一 Fibonacci number

比較for/while loop與recursive function的計算速度快慢，請跑30次，將程式運行的時間記錄下來，並利用MS Excel/ Apple Numbers畫成scatter plot (如下圖)。



# References

[http://tw.gitbook.net/c\\_standard\\_library/index.html](http://tw.gitbook.net/c_standard_library/index.html)

<https://bit.ly/3CR2Gri>