Chapter 6.
Functions

I-Fen Chao

What is "Function"?

- **Top-down design**
  - **Simplify the question**
  - **Divide a big problem into several small problems**
  - **All methods for the small problems, are written by small function blocks**

The goal of "Function"

- Maintainable: 函數可以使程式的設計與維護更加容易
- Readable: 可提高程式的可讀性
- Reuse: 函數可重複被呼叫，提高程式的再利用率
- Module: 每一個函數有自己的任務，可按任務來規劃函數

Function (如同黑盒子)

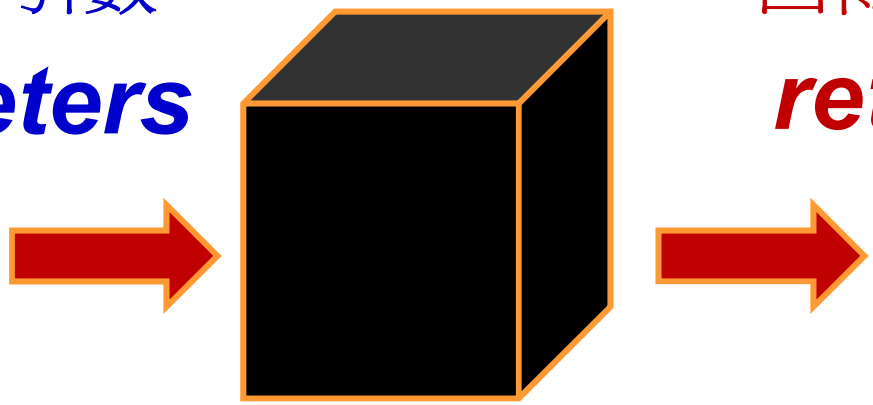
return fun_name(parameter,…)

參數，或引數

parameters

回傳值

return



Input

Output

A simple example : star()

```
02  #include <stdio.h>
03  #include <stdlib.h>
04  void star(void);      /* star()函數的原型 */
05  int main(void)
06  {
07     star();                          /* 呼叫 star 函數 */
08     printf("歡迎使用 C 語言\n");
09     star();                          /* 呼叫 star 函數 */
10     system("pause");
11     return 0;
12  }
13
14  void star(void)
15  {
16     printf("**************\n");
17     return;
18  }
```

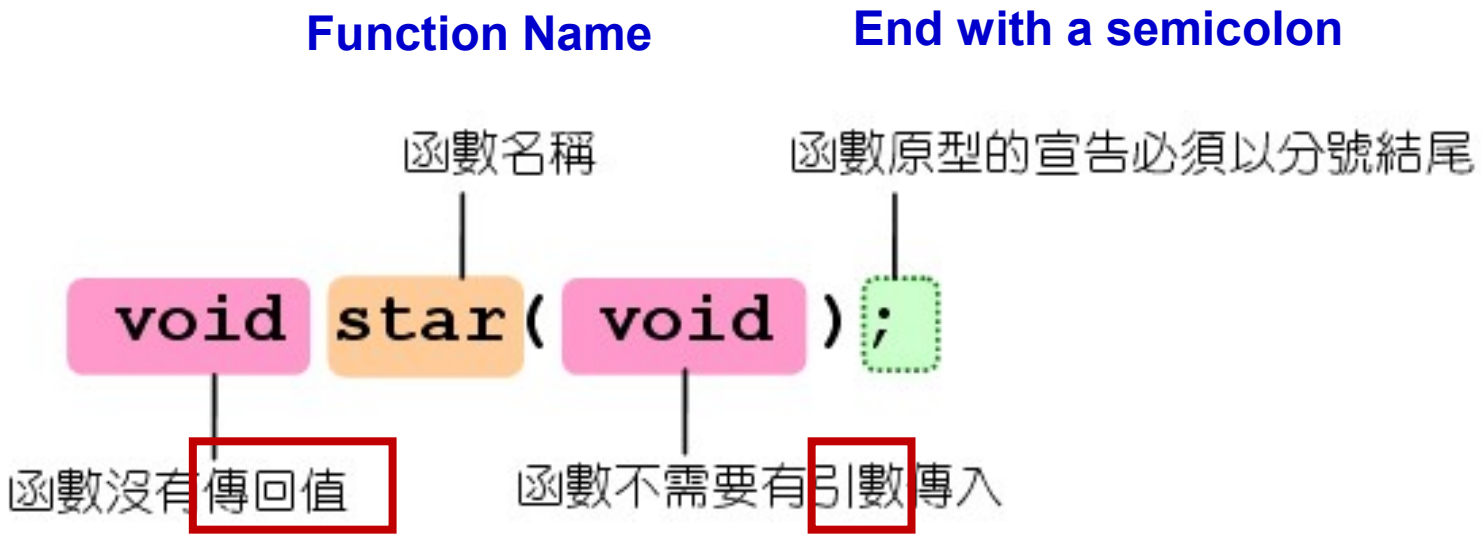
A function prototype

A function call

A function definition

Function Prototype

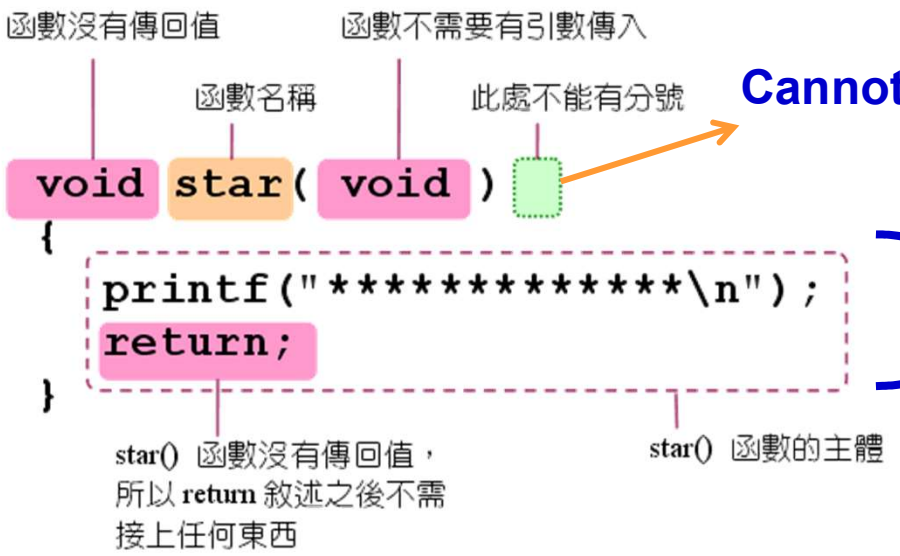
Function Prototype：star() 函數原型宣告的語法：



參數，或引數

Function Definition (Implementation, Program Code)

- star() 函數的定義：



Cannot have a semicolon here!!

Program here

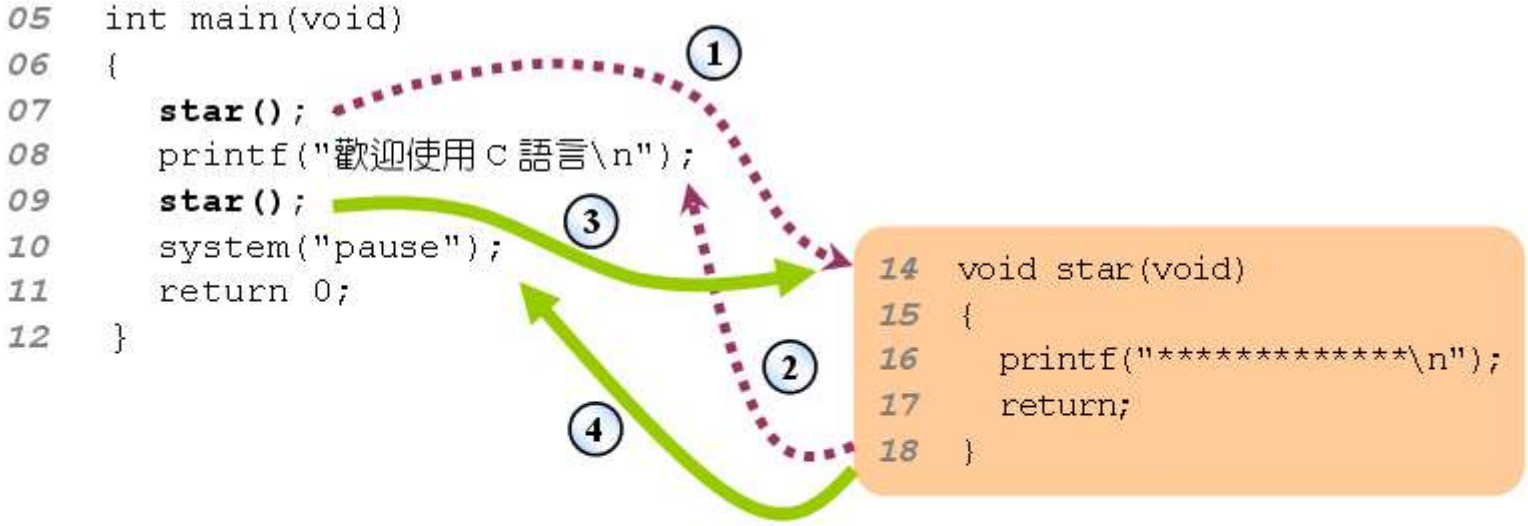
Since no return value for this function,

so you can write “return; ”

Or “}” also indicates return but no return value

Call a function (Execute a program)

- 主函數main與star之間，程式執行的流程：



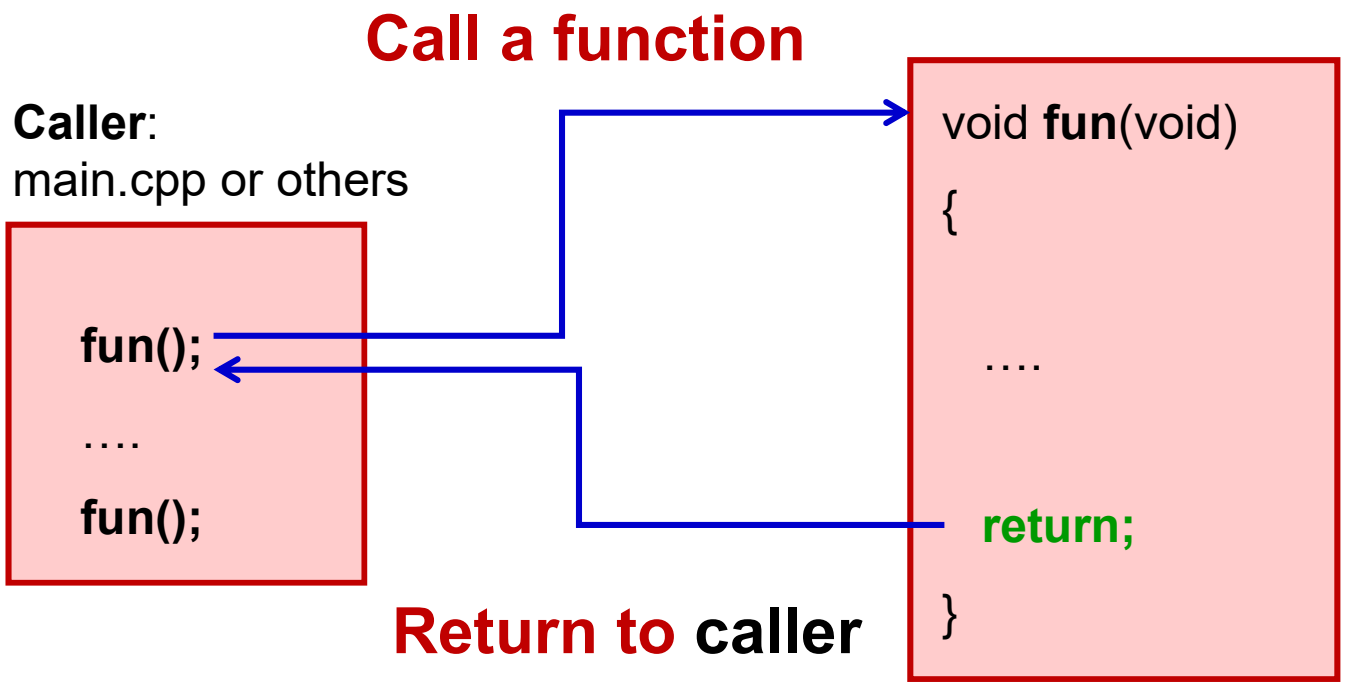
① 第 7 行呼叫 star() 函數，此時程式跳到第 14 行執行

② star() 函數執行完畢，此時返回主程式，繼續執行第 8 行

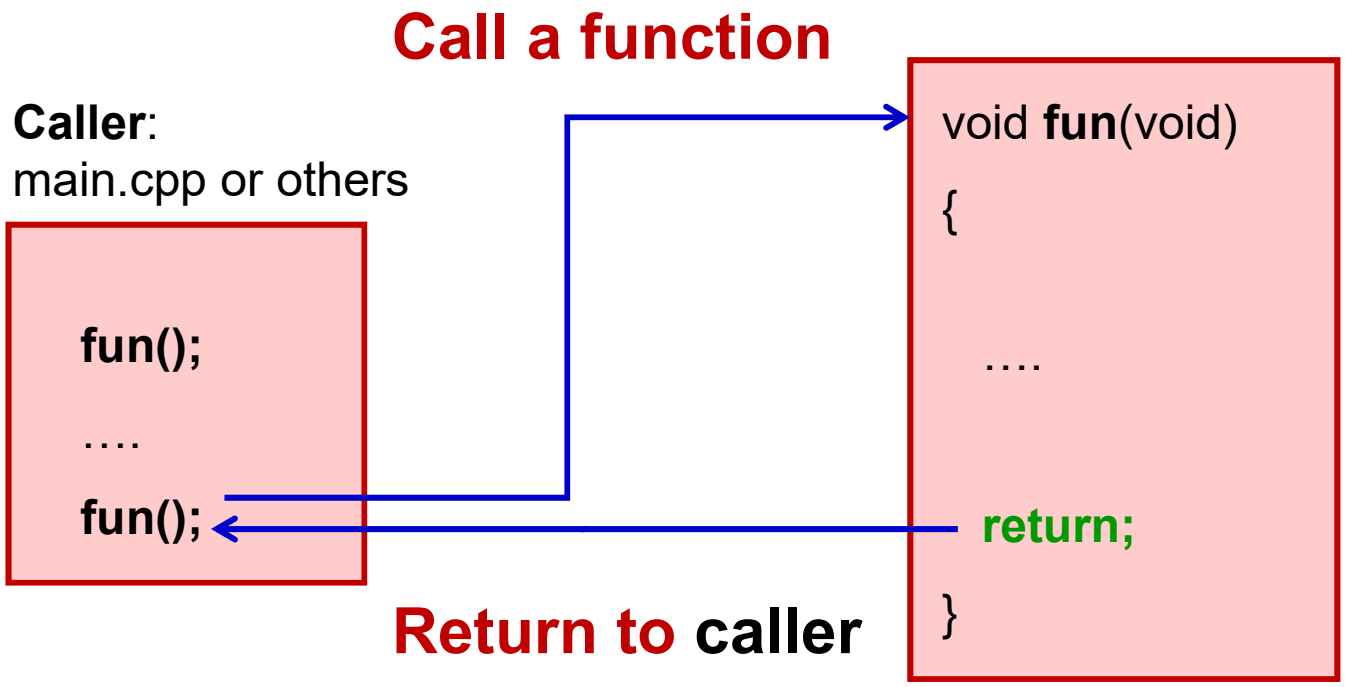
③ 第 9 行呼叫 star() 函數，此時程式再度跳到第 14 行執行

④ star() 函數執行完畢，此時返回主程式，繼續執行第 10 行

Call a simple function



Call a simple function



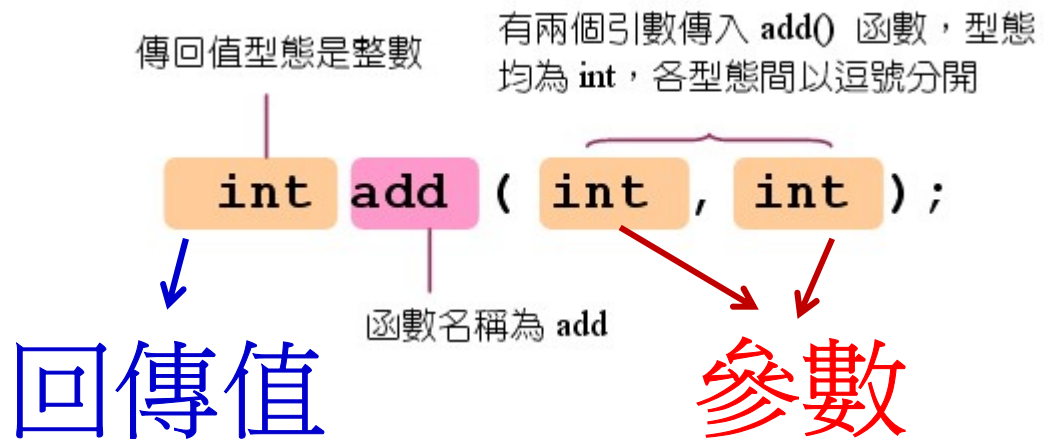
函數的基本架構

- 函數原型宣告

函數原型宣告的格式

```
傳回值型態  函數名稱(引數型態1, 引數型態2,...);
```

- **add()** 可接收二整數，傳回值為整數之和，其原型為：



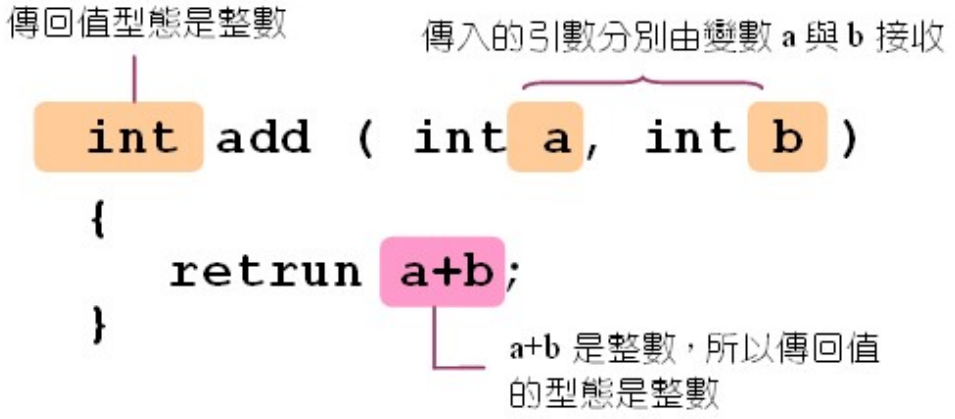
函數的基本架構

- 函數的定義

函數的定義

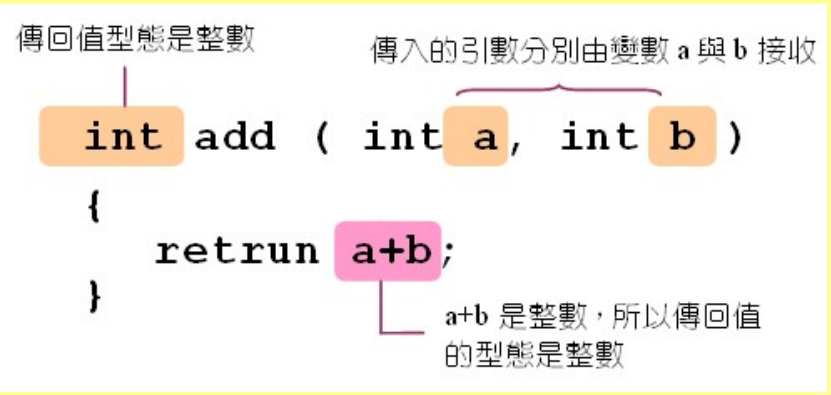
```
傳回值型態 函數名稱(型態1 引數1, …, 型態n 引數n)
{
    變數宣告;
    敘述主體;
    return 運算式;      /* 傳回運算式的值 */
}
```

- **add() 函數的定義：**



程式裡呼叫函數

add() 函數的定義：




```
01  /* prog8_2, 使用 add()函數 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  int add(int,int);           /* add()函數的原型 */
05  int main(void)
06  {
07     int sum, a=5, b=3;
08     sum=add(a,b);            /* 呼叫 add()函數，並把傳回值設給 sum */
09     printf("%d+%d=%d\n",a,b,sum);
10     system("pause");
11     return 0;
12  }
13  int add(int num1, int num2)    /* add()函數的定義 */
14  {
15     int a;                      /* 於 add()函數裡宣告變數 a */
16     a=num1+num2;
17     return a;                   /* 傳回 num1+num2 的值 */
18  }
/* prog8_2 OUTPUT---

5+3=8
--------------------*/
```

請寫出
Function Prototype

- **Function: unique**：有兩個整數引數**a, n**，並回傳一個也是整數的結果
- **Function: compare**：有兩個整數引數**n, m**，不回傳值
- **Function: compute_bin**：有兩個小數引數：**x, y**，及一個整數引數**z**，回傳一個小數的結果
- **Function**：**menu**：沒有引數，沒有回傳值

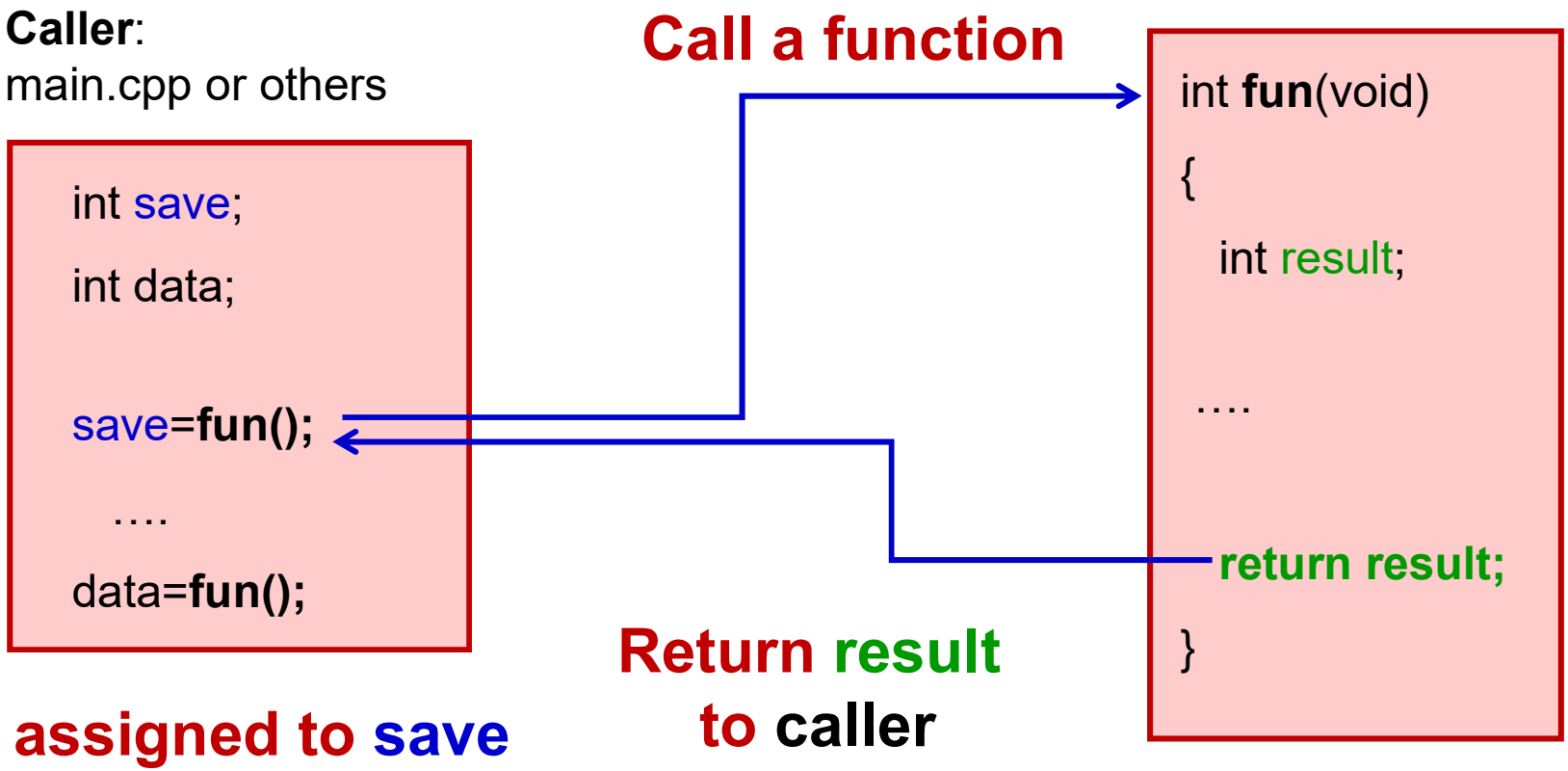
A Practice- function

- 撰寫一函數，**repeat(3)印出3次”What a beautiful day!”**
- **repeat(10)印出10次”What a beautiful day!”**

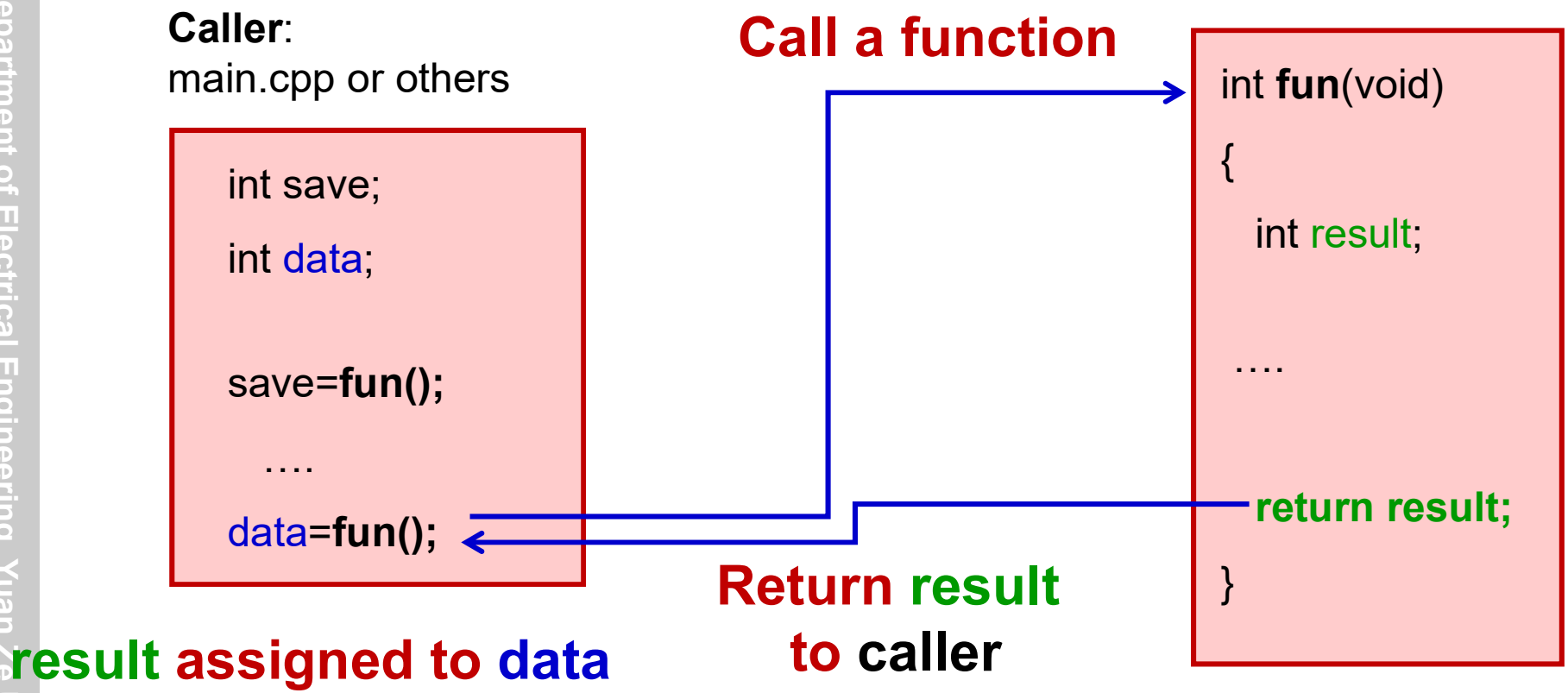
請查詢出以下函式的Function Prototype
請列印p的數值

- p=sqrt(25.5);
- p=cbrt(82.0);
- p=floor(98.52);
- p=exp(5);
- p=pow(3.5);
- p=fabs(12.76);
- p=abs(-ceil(-15+sqrt(16.05)));

Call a simple function with return



Call a simple function with return



An Example

- **Function declaration:**

```
int fun1(void);
float fun2(void);
```

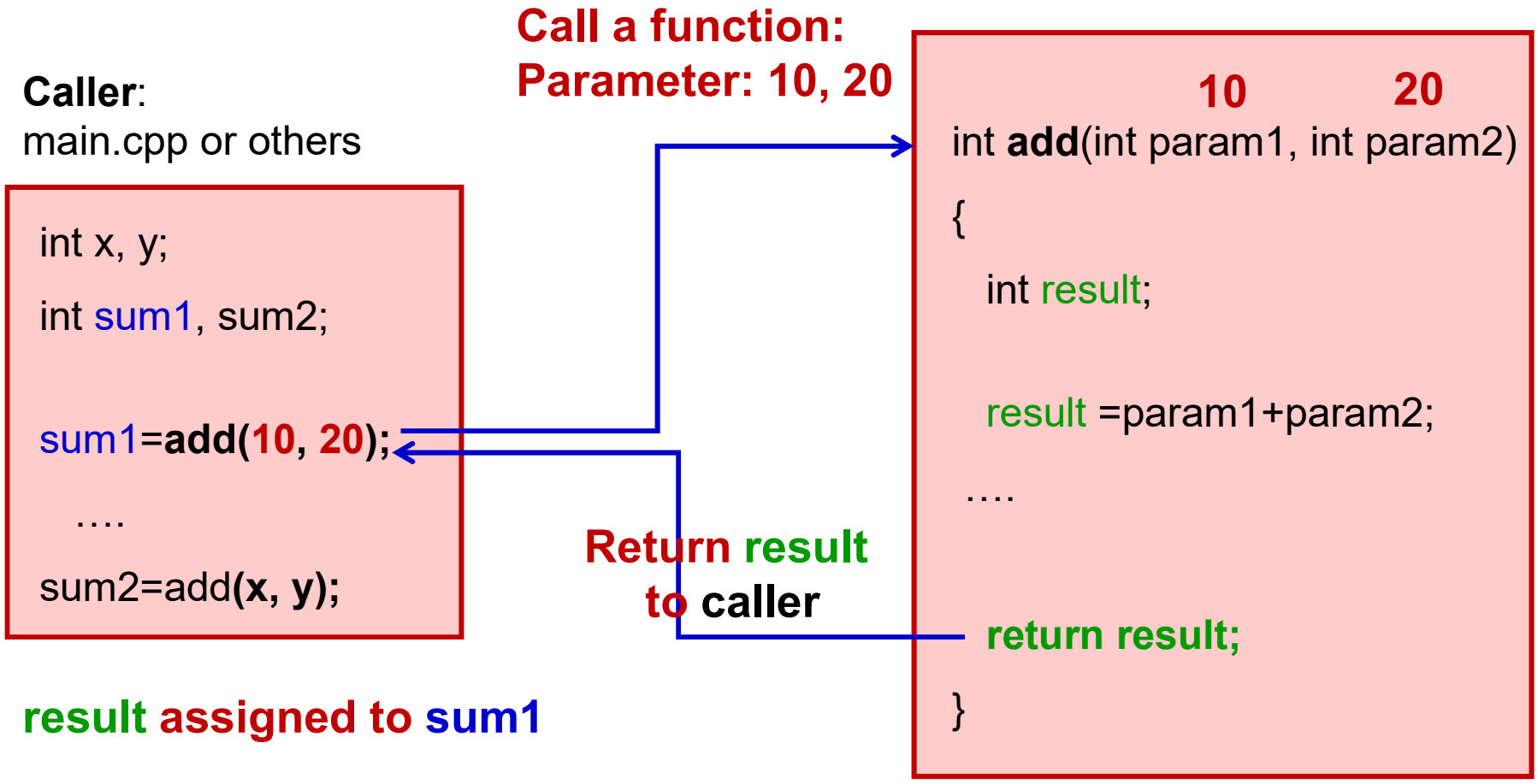
- **Call function:**

```
void main(void)
{
  int result1;
  float result2;

  result1=fun1();

  result2=fun2();

  printf("we call fun2 and get %f\n", fun2());
}
```

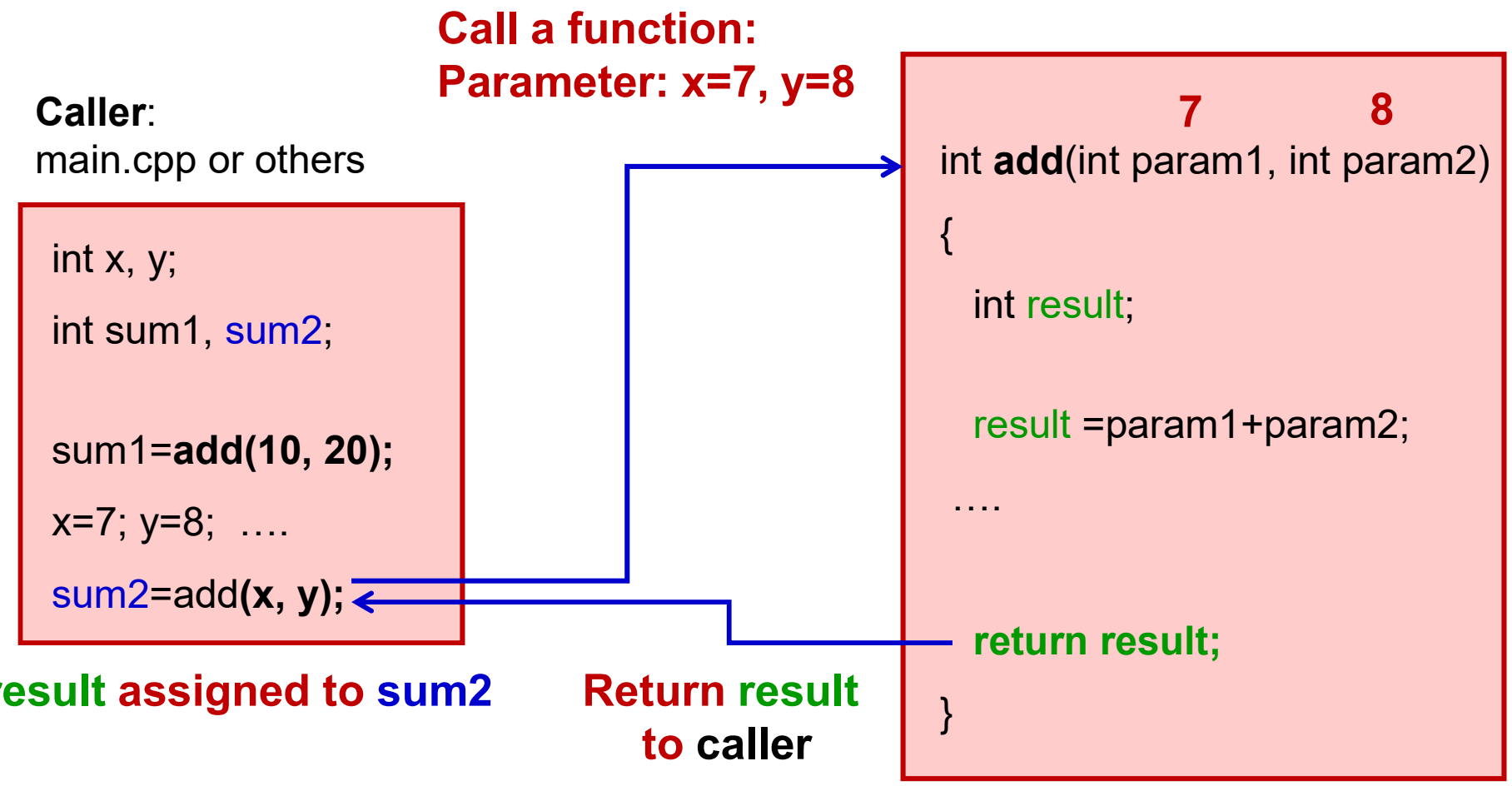
- **Function definition:**

```
int fun1(void)
{
   return 1;
}

float fun2(void);
{
   return 2.0;
}
```

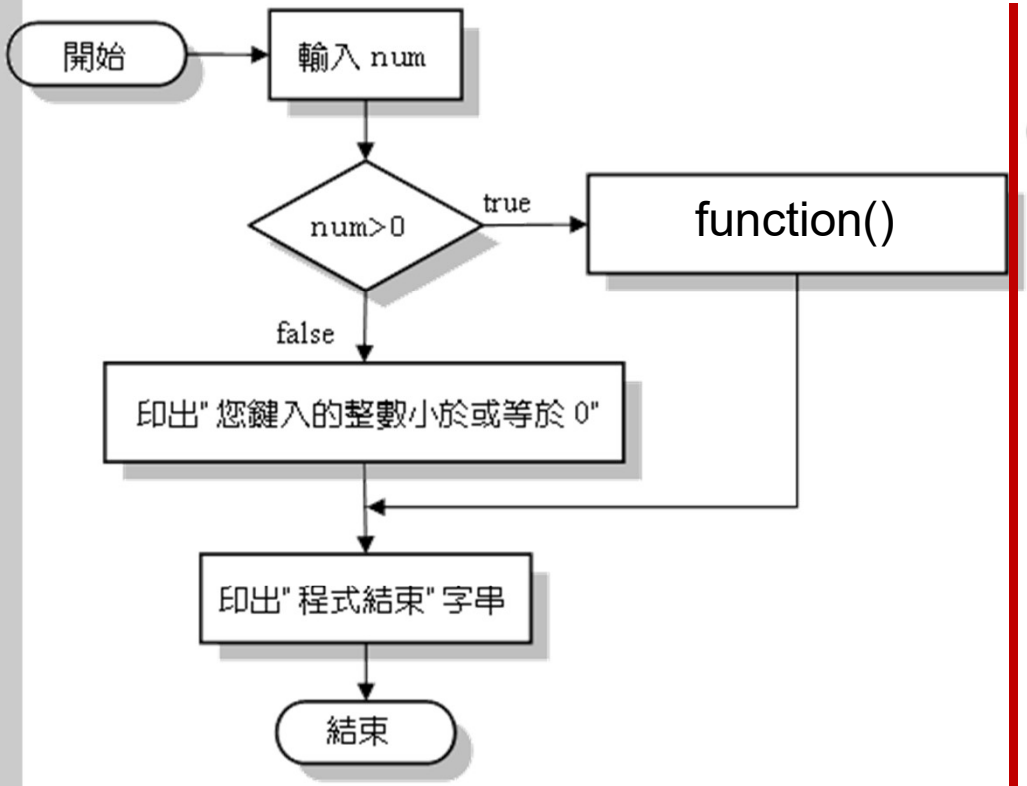
Call a simple function with parameter

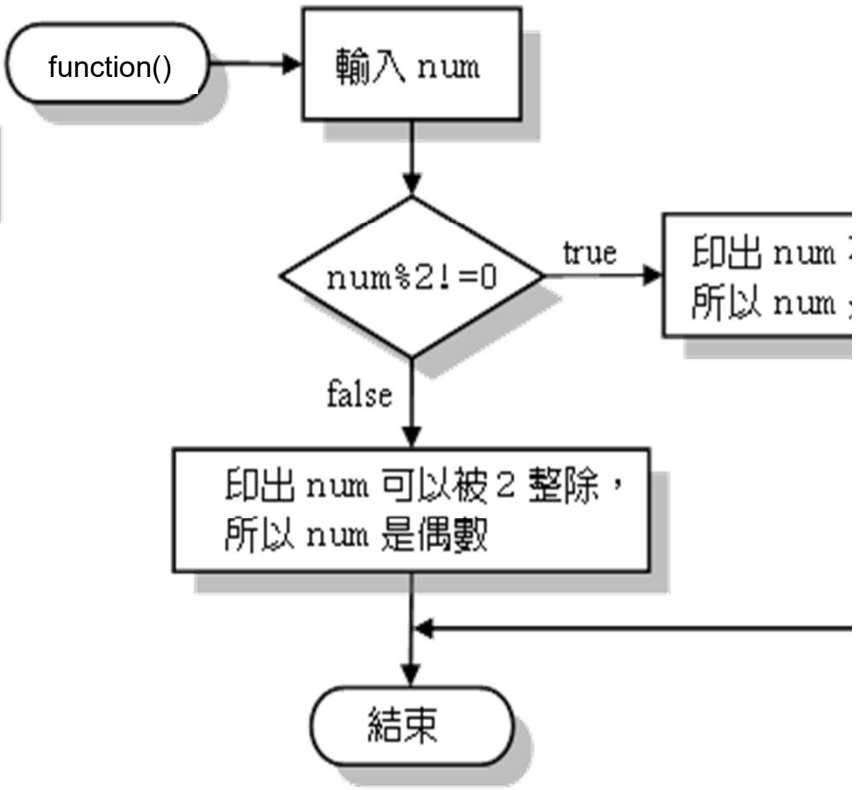


Call a simple function with parameter



Function - flow chart





A Practice- function with parameters & return type

- Write a function abs
  - One integer parameters
  - One return type
- Main program: call abs()
- Output:

Enter one number=10

abs(10)=10

Enter one number=-3

abs(-3)=3

Another way to put the "Function" Before main()

```
01  /* prog8_3, 將 add() 函數放在 main() 函數的前面 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  int add(int num1, int num2)
05  {
06     int a;
07     a= num1+num2;
08     return a;
09  }
10  int main(void)
11  {
12     int sum, a=5, b=3;
13     sum=add(a,b);
14     printf("%d+%d=%d\n",a,b,sum);
15
16     system("pause");
17     return 0;
18  }
```

將 add() 放在 main() 函數的前面

main() 函數置於 add() 的後面

Similarities between functions and variables in C

- Both function names and variable names are considered to be identifiers
  - function names must conform to the rules for identifiers
- Functions (like variables) have types associated with them
  - such as **`int`** or **`double`**
- Like variables, functions and their type must be declared prior to their use in a program.

Functions

- **Using Function for modularity structured programming**

  (程式模組化)

  - **Suggestion: One work for one function. Do not put all work altogether.**
  - **Do not do input/output in function, only do the calculation. e.g., sqrt();**

- **Parameters and return value depends on what you need**
  - **0~many parameters**
  - **0~1 return value**

Another example: Function (1)

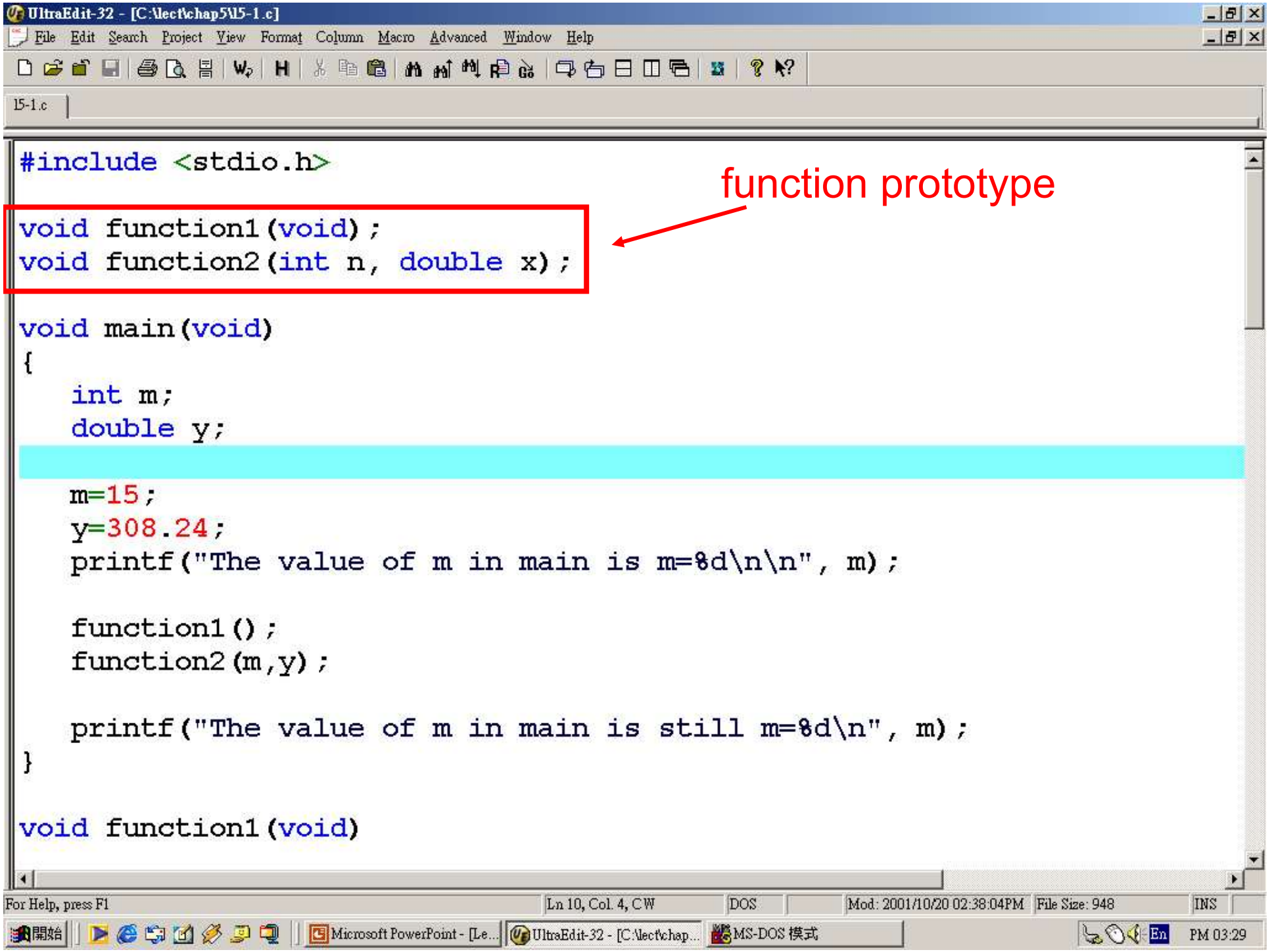
```
01  /* prog8_4, display()的練習 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  void display(char,int);   /* display()函數的原型 */
05  int main(void)
06  {
07     int n;
08     char ch;
09     printf("請輸入欲列印的字元:");
10     scanf("%c",&ch);
11     printf("請問要印出幾個字元:");
12     scanf("%d",&n);
13     display(ch,n);          /* 呼叫自訂的函數，印出 n 個 ch 字元 */
14
15     system("pause");
16     return 0;
17  }
```

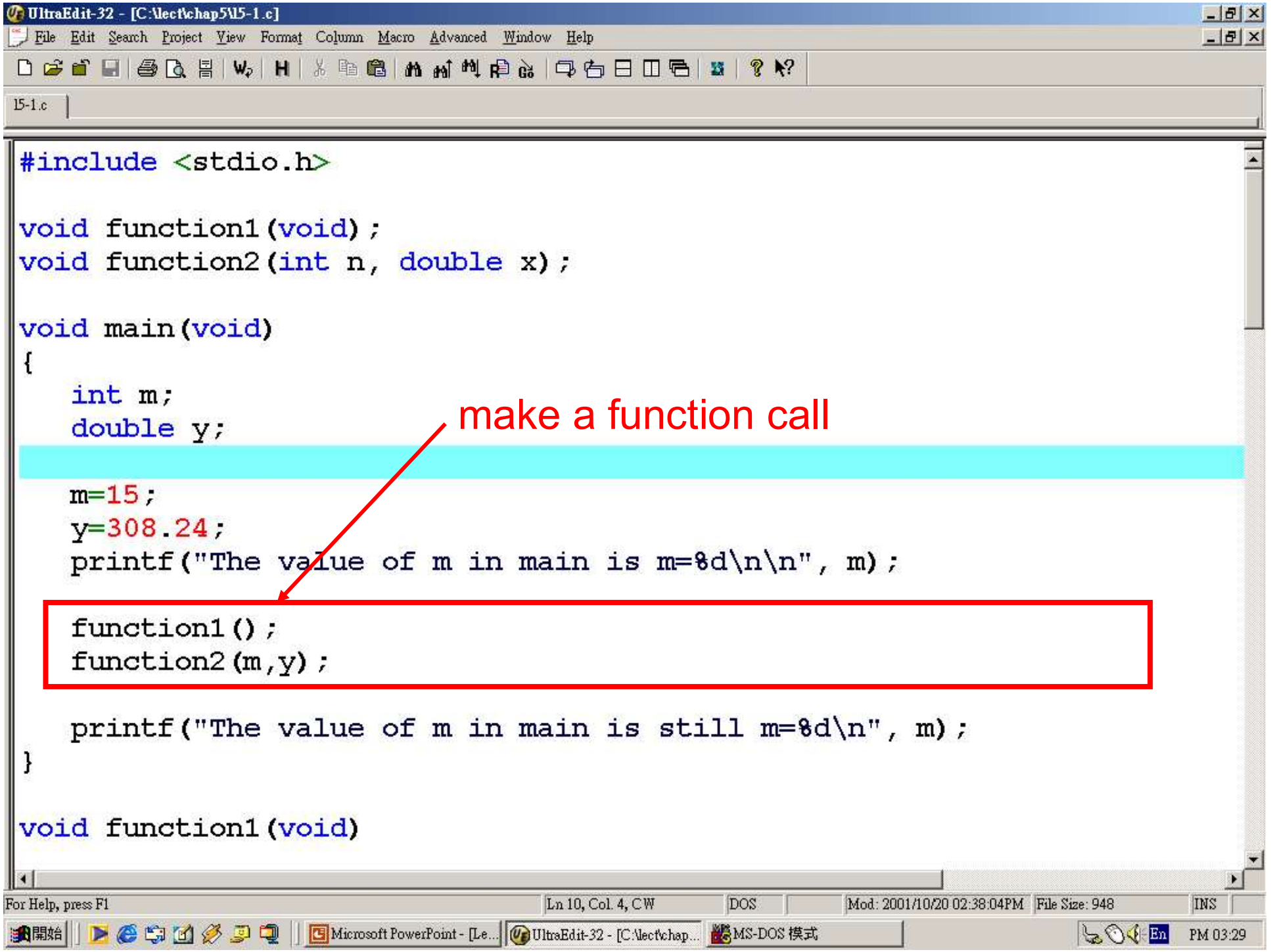
Another example: Function (2)

```
19  void display(char ch,int n)    /* 自訂的函數 display() */
20  {
21     int i;
22     for(i=1;i<=n;i++)           /* for 迴圈，可印出 n 個 ch 字元 */
23        printf("%c",ch);              /* 印出 ch 字元 */
24     printf("\n");
25     return;
26  }
```

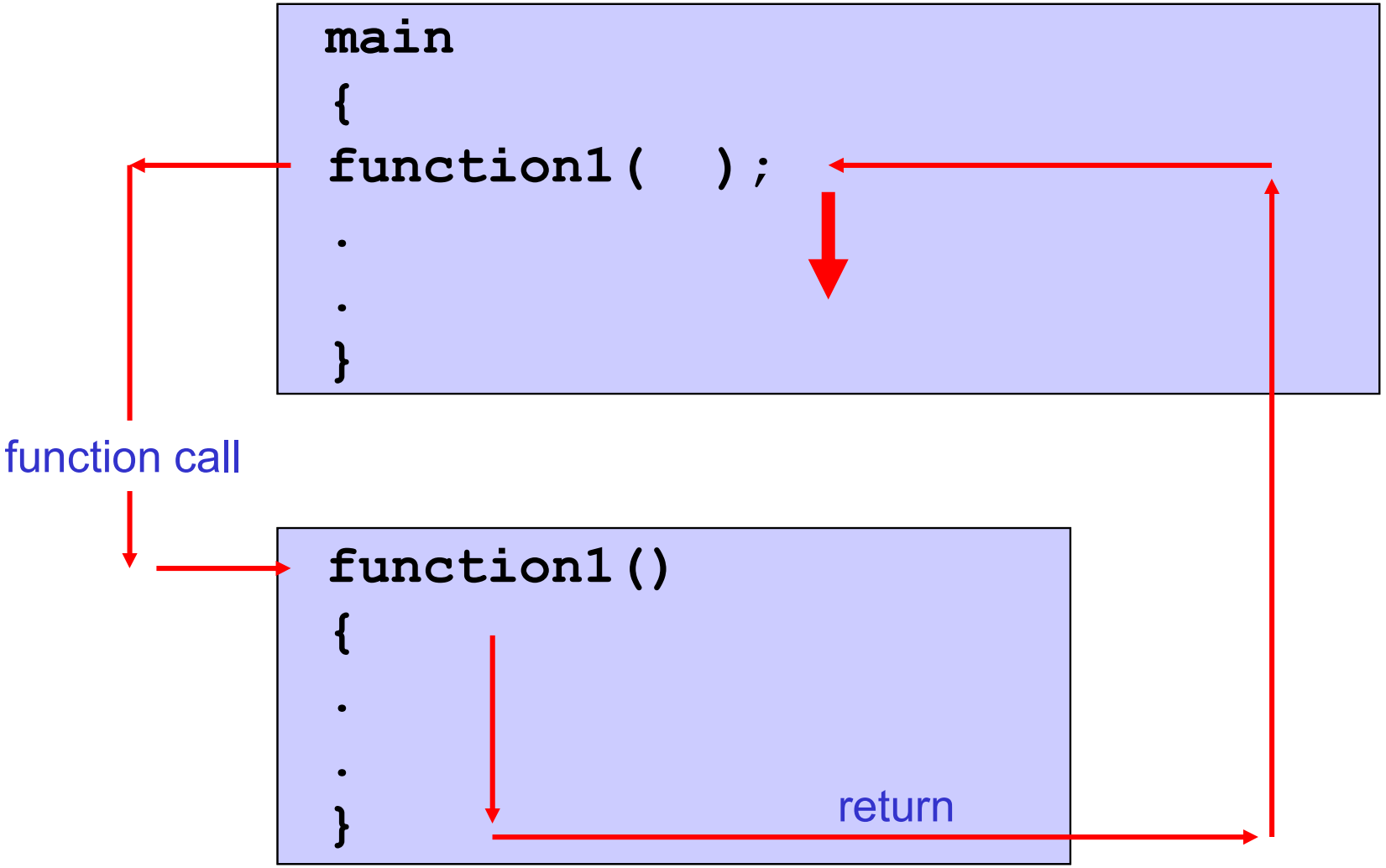
A practice

- Function: power(x,n) $-x^n$
  - Two parameters
  - Return result
- main
  - Input : x, n
  - Call power function (print the result)
  - Input : x, n
  - Call power function (print the result)


UltraEdit-32 - [C:\lect\chap5\l5-1.c]
File Edit Search Project View Format Column Macro Advanced Window Help
l5-1.c
#include <stdio.h>
void function1(void);
void function2(int n, double x);
void main(void)
{
int m;
double y;
m=15;
y=308.24;
printf("The value of m in main is m=%d\n\n", m);
function1();
function2(m,y);
printf("The value of m in main is still m=%d\n", m);
}
void function1(void)
function prototype
For Help, press F1
Ln 10, Col. 4, CW
DOS
Mod: 2001/10/20 02:38:04PM
File Size: 948
INS
開始
Microsoft PowerPoint - [Le...
UltraEdit-32 - [C:\lect\chap...
MS-DOS 模式
En
PM 03:29


UltraEdit-32 - [C:\lect\chap5\l5-1.c]
File Edit Search Project View Format Column Macro Advanced Window Help
l5-1.c
#include <stdio.h>
void function1(void);
void function2(int n, double x);
void main(void)
{
int m;
double y;
m=15;
y=308.24;
printf("The value of m in main is m=%d\n\n", m);
function1();
function2(m,y);
printf("The value of m in main is still m=%d\n", m);
}
void function1(void)
make a function call
For Help, press F1
Ln 10, Col. 4, CW
DOS
Mod: 2001/10/20 02:38:04PM
File Size: 948
INS
開始
Microsoft PowerPoint - [Le...
UltraEdit-32 - [C:\lect\chap...
MS-DOS 模式
PM 03:29

Direction of program flow when function1 is called



Function type

- The type of a function is the type of value that it *returns* to it's calling function
  - Therefore, function can be of type:
    - int
    - float
    - double
    - char
    - void
    - … (other valid data types)
- Not return value→ void

Arguments (Parameters)

- The information passed into the function
- No arguments function call

```
function1( );
```

- 2 arguments function call

```
function2(m,y);
```

Function Prototype

- **To declare a function, means that it indicates that the function exists and probably will be used in the program.**
  - **function's name**
  - **return type**
  - **arguments**

function2 has 2 arguments

```
#include <stdio.h>

void function1(void);
void function2(int n, double x);

void main(void)
```

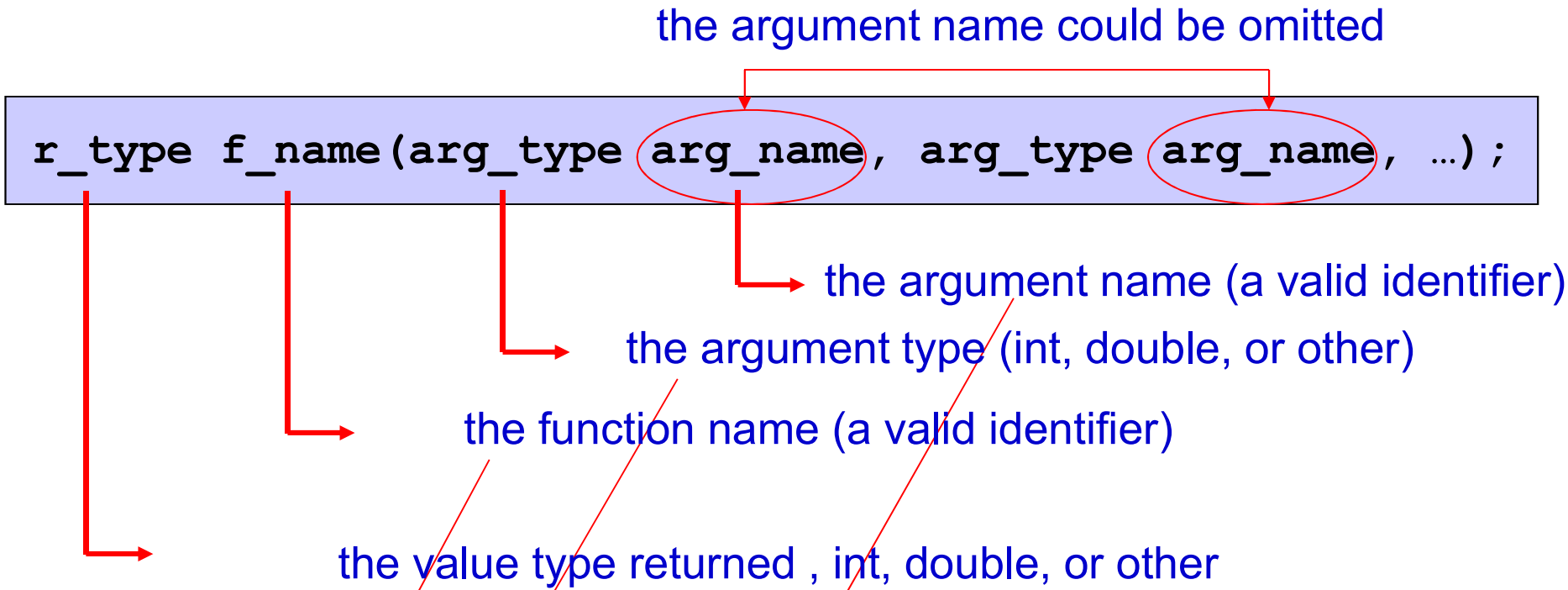
function2 returns no value

Function Prototypes (Function Declaration)

```
/* function prototype */
int maximum( int x, int y, int z );
or
int maximum( int , int , int  );
```

- Function prototypes: tell the compiler how the function looks like: function name, parameters data type, return value data type.
- There are lots of system function prototype in the header file

Function Prototypes (Function Declaration)




```
#include <stdio.h>

void function1(void);
void function2(int n, double x);

void main(void)
```

Function Prototypes (Function Declaration)

```
r_type f_name(arg_type , arg_type , …);
```

the argument name could be omitted

```
void function2(int n, double x);
```

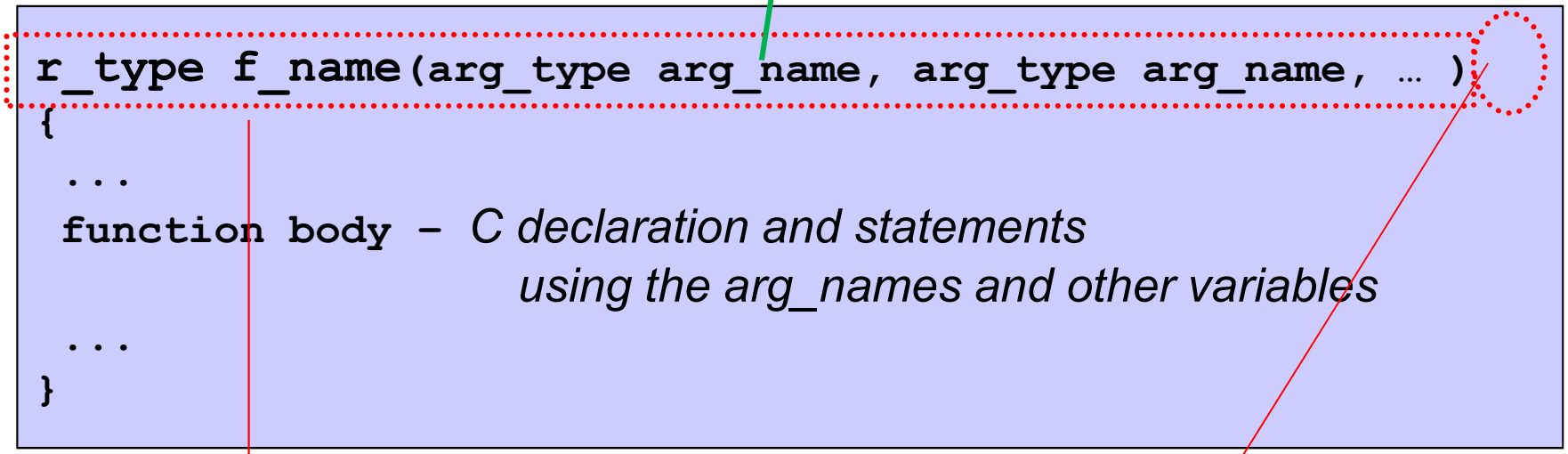
The prototype of the function could be rewritten as followed:

```
void function2(int, double );
```

The function definition

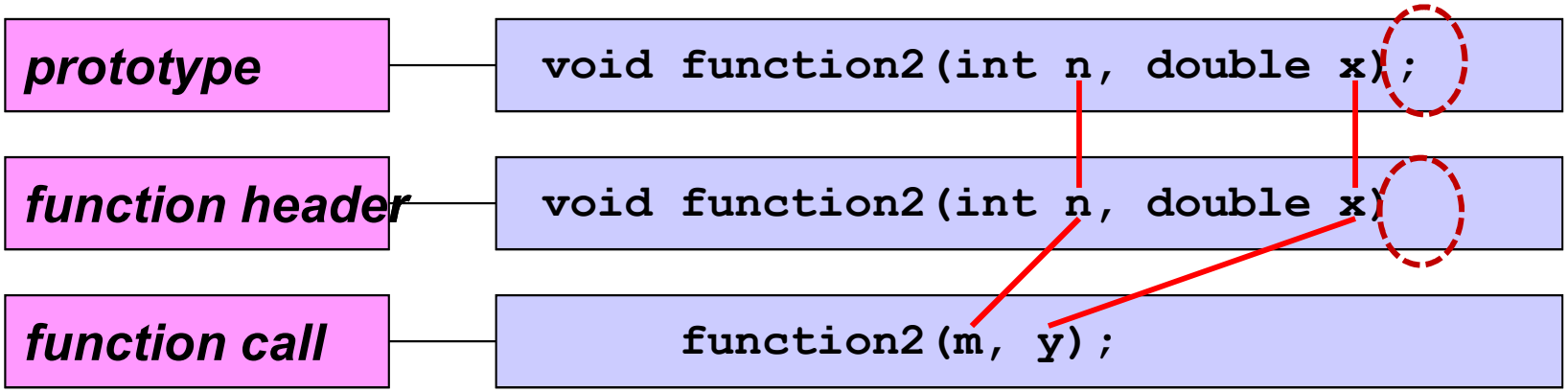
The argument names cannot be omitted!

- It includes the body of the function.



Argument list

- The number, order, and type of parameters in the argument lists of a function call and its *must* match.



More about functions

- main()
  - Do not have to put at the beginning of the program
  - The execution start point of the program
  - The program can not be executed if without main()
- Function
  - Function can call other functions
  - Only the declared functions can be called (the function prototype is known)

- 函式的宣告可以出現在其他函式內
  - 只有經過宣告的函式可以呼叫該函式

`return` statement[1]

- The **`return`** statement could be used in a *void* function.
  - In a void function, we can use a return statement with the following form:

```
return;
```

- 「 } 」
  - ANSI C中，that is “return” statement

`return` statement[2]

- It could be placed in anywhere of the code.
  - The "return" does not have to be put at the end of the program
  - More than one "**`return`**" statement is allowed

Something reminds

- A *void* function could not appear on the right side of an assignment statement.

- `int main()`
  - the returning value is passing to operating system


```
return (0);
```

```c
/* Fig. 5.4: fig05_04.c
   Finding the maximum of three integers */
#include <stdio.h>

int maximum( int x, int y, int z ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   int number1; /* first integer */
   int number2; /* second integer */
   int number3; /* third integer */

   printf( "Enter three integers: " );
   scanf( "%d%d%d", &number1, &number2, &number3 );

   /* number1, number2 and number3 are arguments
      to the maximum function call */
   printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );
   return 0; /* indicates successful termination */
} /* end main */

```

```
23  /* Function maximum definition */
24  /* x, y and z are parameters */
25  int maximum( int x, int y, int z )
26  {
27     int max = x; /* assume x is largest */
28
29     if ( y > max ) { /* if y is larger than max, assign y to max */
30        max = y;
31     } /* end if */
32
33     if ( z > max ) { /* if z is larger than max, assign z to max */
34        max = z;
35     } /* end if */
36
37     return max; /* max is largest value */
38  } /* end function maximum */
```

Data type: bool

當資料是兩種答案(是/錯，成功/失敗,...)，可以宣告成bool這個資料型態

- true (value=1)
- false (value=0)

```
bool complete=false;
while(!complete){
   if(….)
      complete=true;
}
```

```
bool fun(void);
void main(void)
{
    if(fun()==true)
       printf("we get true\n");
}
bool fun(void)
{
   if(…)
     return true;
  return false;
}
```

practice

1. abs
2. add
3. is_odd

定義function

- **abs:** 傳回?
- **add** 三個整數: 傳回?
- **add** 三個小數: 傳回?
- **is_odd:** 傳回?
- **max:** 找三個整數的最大數，並傳回

The Type of Variables

- **Local variable (區域變數)**
  - **The lifecycle of a local variable is only within a function**

    區域變數的生命週期只在於主控權在函數手上
- **Global variable (全域變數)**
  - **All function is allowed to access a global variable (read/write)**

    所有的函數模組皆可使用全域變數
- **Static variable (靜態變數)**
  - **A static variable is still remained within a function, even if the program leaves the function**

    函數執行完時，函數內的靜態變數之值不會消失，而會保留在函數內

Local Variables (1)

```
01  /* prog8_13, 區域變數的範例 (一) */
02  #include <stdio.h>
03  #include <stdlib.h>
04  int fac(int);                  /* fac()函數的原型 */
05  int main(void)
06  {
07     int ans;
08     ans=fac(5);
09     printf("fac(5)=%d\n",ans);
10     system("pause");
11     return 0;
12  }
13  int fac(int n)
14  {
15     int i, total=1;
16     for(i=1; i<=n; i++)
17        total=total*i;
18     return total;
19  }
```

區域變數 ans 的活動範圍

區域變數 i 與 total 的活動範圍

區域變數 n 的活動範圍

Local Variables (2)

```
01  /* prog8_14, 區域變數的範例（二）*/
02  #include <stdio.h>
03  #include <stdlib.h>
04  void func(void);
05  int main(void)
06  {
07     int a=100;      /* 宣告 main()函數裡的區域變數 a */
08
09     printf("呼叫 func()之前,a=%d\n",a);    /* 印出 main()中 a 的值 */
10     func();         /* 呼叫自訂的函數 */
11     printf("呼叫 func()之後,a=%d\n",a);    /* 印出 a 的值 */
12
13     system("pause");
14     return 0;
15  }
16  void func(void)       /* 函數 func() */
17  {
18     int a=300;          /* 宣告 func()函數裡的區域變數 a */
19     printf("於 func()函數裡,a=%d\n",a);    /* 印出 func 函數中 a 的值 */
20  }
```

Global Variables (1)

```
01  /* prog8_15, 全域變數的範例(一) */
02  #include <stdio.h>
03  #include <stdlib.h>
04  void func(void);        /* 函數 func()的原型 */
05  int a;                  /* 宣告全域變數 a */
06  int main(void)
07  {
08     a=100;               /* 設定全域變數 a 的值為 100 */
09     printf("呼叫 func()之前,a=%d\n",a);
10     func();              /* 呼叫自訂的函數 */
11     printf("呼叫 func()之後,a=%d\n",a);
12
13     system("pause");
14     return 0;
15  }
16  void func(void)         /* 自訂的函數 func() */
17  {
18     a=300;               /* 設定全域變數 a 的值為 300 */
19     printf("於 func()函數裡,a=%d\n",a);
20  }
```

全域變數 a 的活動範圍

Global Variables (2)

```
01  /* prog8_16, 全域變數的範例(二) */
02  #include <stdio.h>
03  #include <stdlib.h>
04  void func(void);
05  int a=50;                          /* 定義全域變數 a */
06
07  int main(void)
08  {
09     int a=100;                      /* 定義區域變數 a */
10     printf("呼叫 func()之前,a=%d\n",a);
11     func();          /* 呼叫自訂的函數 */
12     printf("呼叫 func()之後,a=%d\n",a);
13     system("pause");
14     return 0;
15  }
16  void func(void)
17  {
18     a=a+300;                        /* 這是全域變數 a */
19     printf("於 func()函數裡,a=%d\n",a);
20  }
```

全域變數 a 的活動範圍

區域變數 a 的活動範圍

全域變數 a 的活動範圍

Global Variables (3)

```
01  /* prog8_17, 全域變數的使用範例（三） */
02  #include <stdio.h>
03  #include <stdlib.h>
04  double pi=3.14;        /* 宣告全域變數 pi */
05  void peri(double),area(double);
06  int main(void)
07  {
08     double r=1.0;
09     printf("pi=%.2f\n",pi);     /* 於 main() 裡使用全域變數 pi*/
10     printf("radius=%.2f\n",r);
11     peri(r);        /* 呼叫自訂的函數 */
12     area(r);
13     system("pause");
14     return 0;
15  }
16  void peri(double r)  /* 自訂的函數 peri()，印出圓周 */
17  {
18     printf("圓周長=%.2f\n",2*pi*r);  /* 於 peri() 裡使用全域變數 pi */
19  }
20  void area(double r)  /* 自訂的函數 area()，印出圓面積 */
21  {
22     printf("圓面積=%.2f\n",pi*r*r);  /* 於 area() 裡使用全域變數 pi */
23  }
```

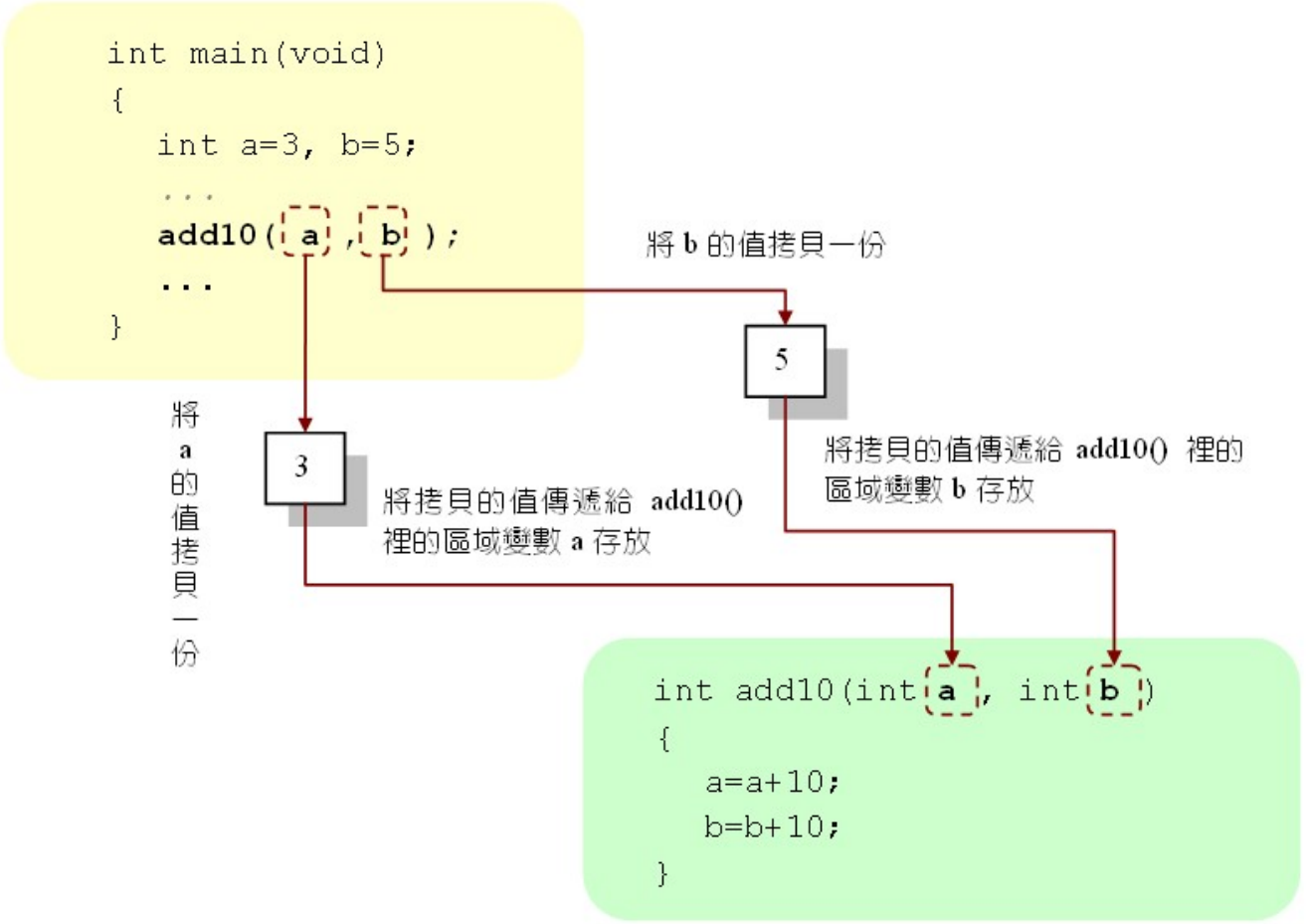
Static Variables

```
01  /* prog8_18, 區域靜態變數使用的範例 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  void func(void);      /* 宣告 func()函數的原型 */
05  int main(void)
06  {
07     func();            /* 呼叫函數 func() */
08     func();            /* 呼叫函數 func() */
09     func();            /* 呼叫函數 func() */
10
11     system("pause");
12     return 0;
13  }
14  void func(void)
15  {
16     static int a=100;                /* 宣告靜態變數 a */
17     printf("In func(),a=%d\n",a);    /* 印出 func()函數中 a 的值 */
18     a+=200;
19  }
```

Parameters (1)

```
01  /* prog8_19, 函數的傳值機制 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  void add10(int,int);                    /* add10()的原型 */
05  int main(void)
06  {
07     int a=3, b=5;                        /* 宣告區域變數 a 與 b */
08     printf("呼叫函數 add10()之前: ");
09     printf("a=%d, b=%d\n",a,b);          /* 印出 a、b 的值 */
10     add10(a,b);
11     printf("呼叫函數 add10()之後: ");
12     printf("a=%d, b=%d\n",a,b);          /* 印出 a、b 的值 */
13     system("pause");
14     return 0;
15  }
16  void add10(int a,int b)
17  {
18     a=a+10;                  /* 將變數 a 的值加 10 之後，設回給 a */
19     b=b+10;                  /* 將變數 b 的值加 10 之後，設回給 b */
20  }
```

Parameters (2)



Headers

- Header files
  - Contain function prototypes for library functions
  - `<stdlib.h>` , `<math.h>` , etc
  - Load with `#include <filename>`
    `#include <math.h>`
  - System header files : `<` `>`
- Custom header files
  - Create file with functions
  - Save as `filename.h`
  - Load in other files with `#include "filename.h"`
  - Reuse functions
  - Your own header files : “ ”

Scope Rules

- 一個變數要由duration跟scope去分析
- Variable Duration (生命週期)
  - An variable's duration is the period during which the variable exists in memory.
  - Some exist briefly, some are repeatedly created and destroyed, and others exist for the entire execution of a program.
- Variable Scope (可見範圍)
  - An variable's scope is where the variable can be referenced in a program.
  - Some can be referenced throughout a program, others from only portions of a program.

Storage Duration

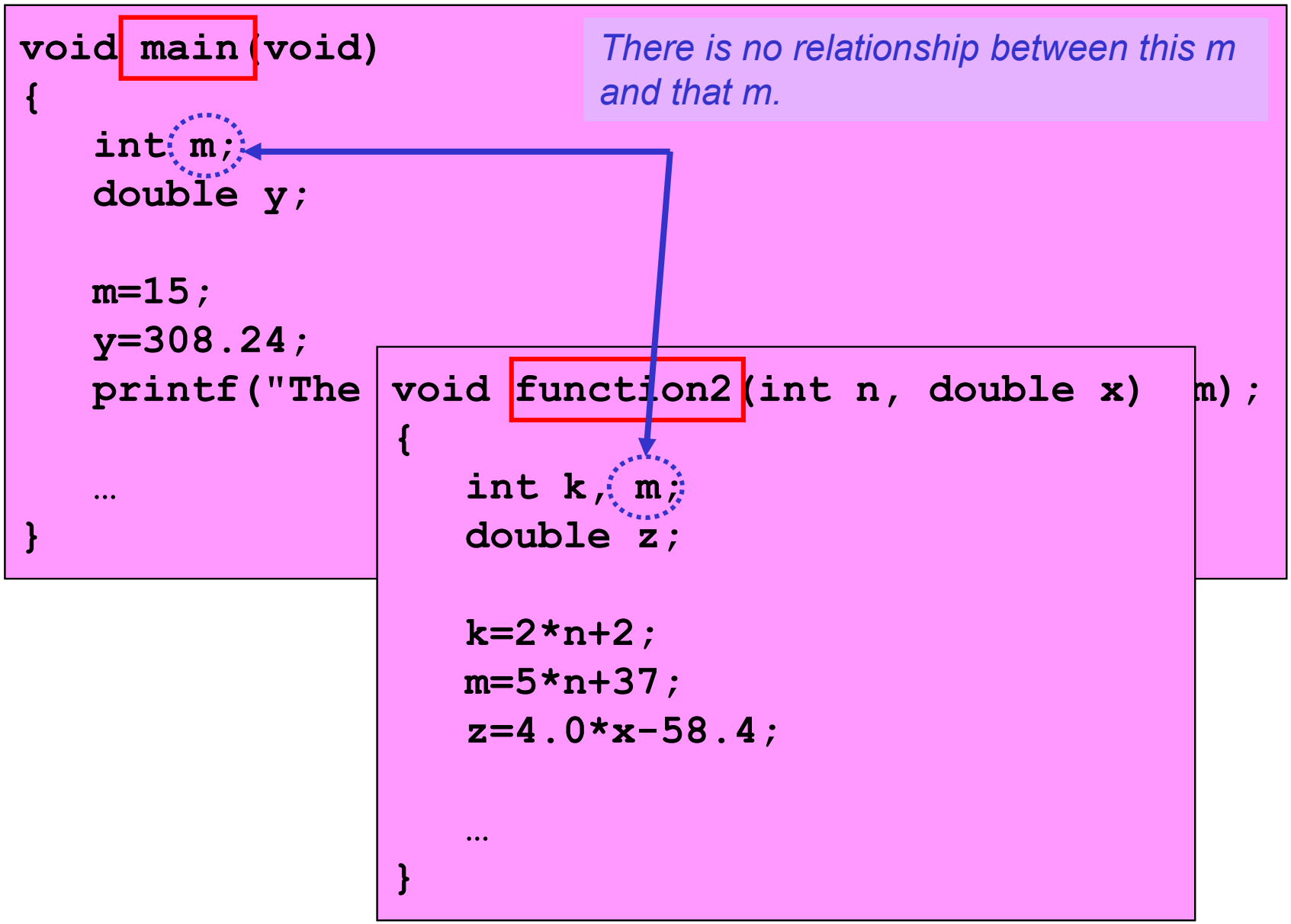
- automatic storage duration (短暫生命週期)
  - Variables with automatic storage duration are created when the block in which they're defined is entered;
  - they exist while the block is active, and they're destroyed when the block is exited.
  - A function's local variables (those declared in the parameter list or function body) normally have automatic storage duration.
- static storage duration (永久生命週期)
  - Variables of static storage duration exist from the time at which the program begins execution.
  - For static variables, storage is allocated and initialized once, when the program begins execution.

Function variables and parameters

- All variables defined in function definitions are local variables—they're known only in the function in which they're defined.
- A function's parameters are also local variables of that function.

Variable duration and scope

	Scope (可見範圍)	Duration (生命週期)
Local variable	block	block
Global variable (全域變數)	file	program execution
extern global declaration	project	program execution
static local variable	block	program execution
static global variable	file	program execution


void main(void)
{
int m;
double y;
m=15;
y=308.24;
printf("The
…
}
There is no relationship between this m and that m.
void function2(int n, double x)
m);
{
int k, m;
double z;
k=2*n+2;
m=5*n+37;
z=4.0*x-58.4;
…
}

Mismatch of argument list

```
#include <stdio.h>
void function2 (int i, int j);
void main(void)
{
  double a=12.34985, b=-2.5678;
  function2(a,b);
  printf("a= %lf  b= %lf \n", a, b);
}
void function2 (int i, int j)
{
  printf("i= %d  j= %d \n", I, j);
}
```

Mismatch!

12 -2

Global or local variables ??

- Global variable seems to be easily programming
- But, a better structured and modular programming (better maintenance)
  - A function should be independent with other programs
  - If the variable is only related to a function, declare it as a local variable
  - If not necessary, do not declare global variable

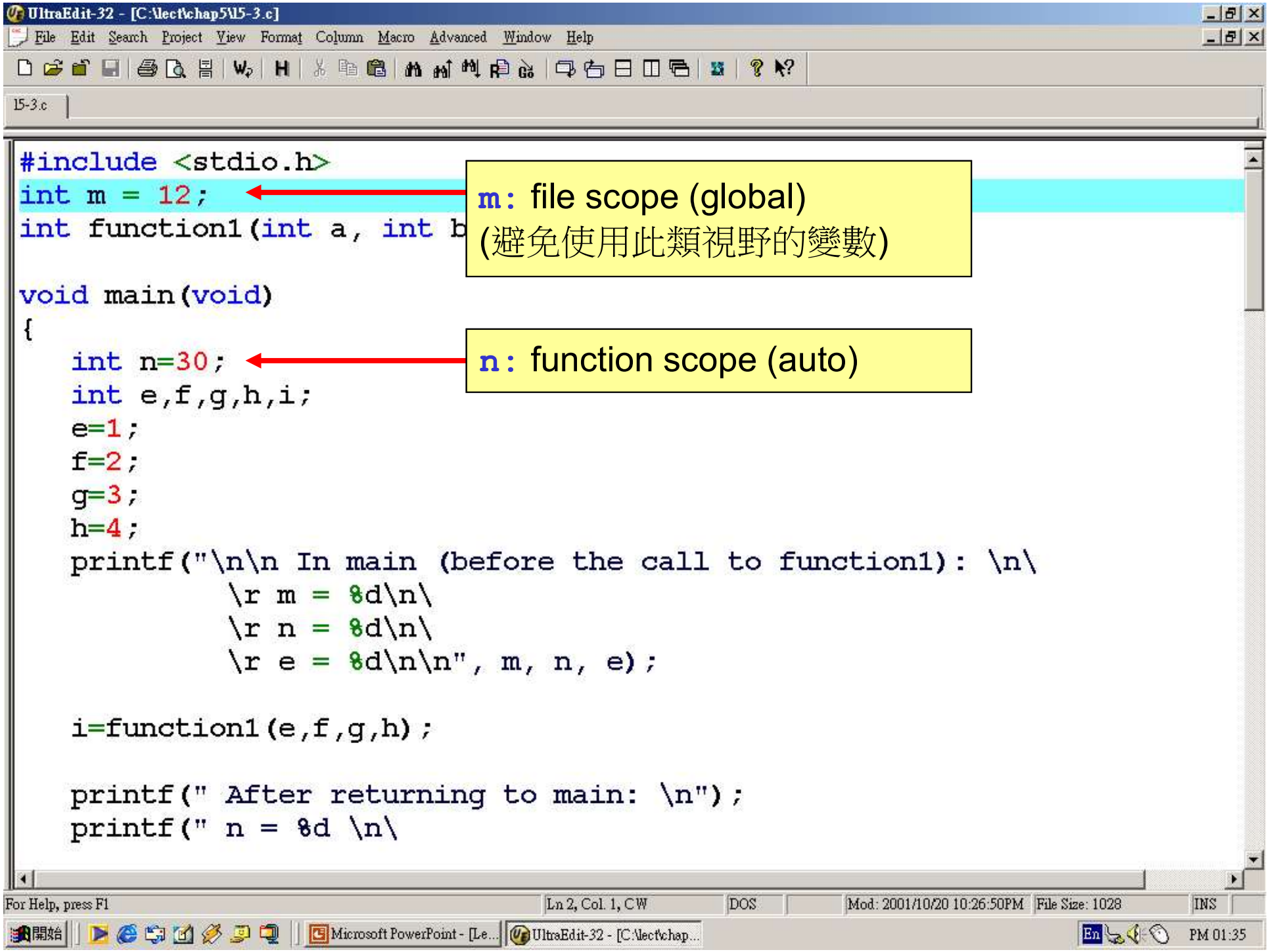
The concept of Object-oriented programming!

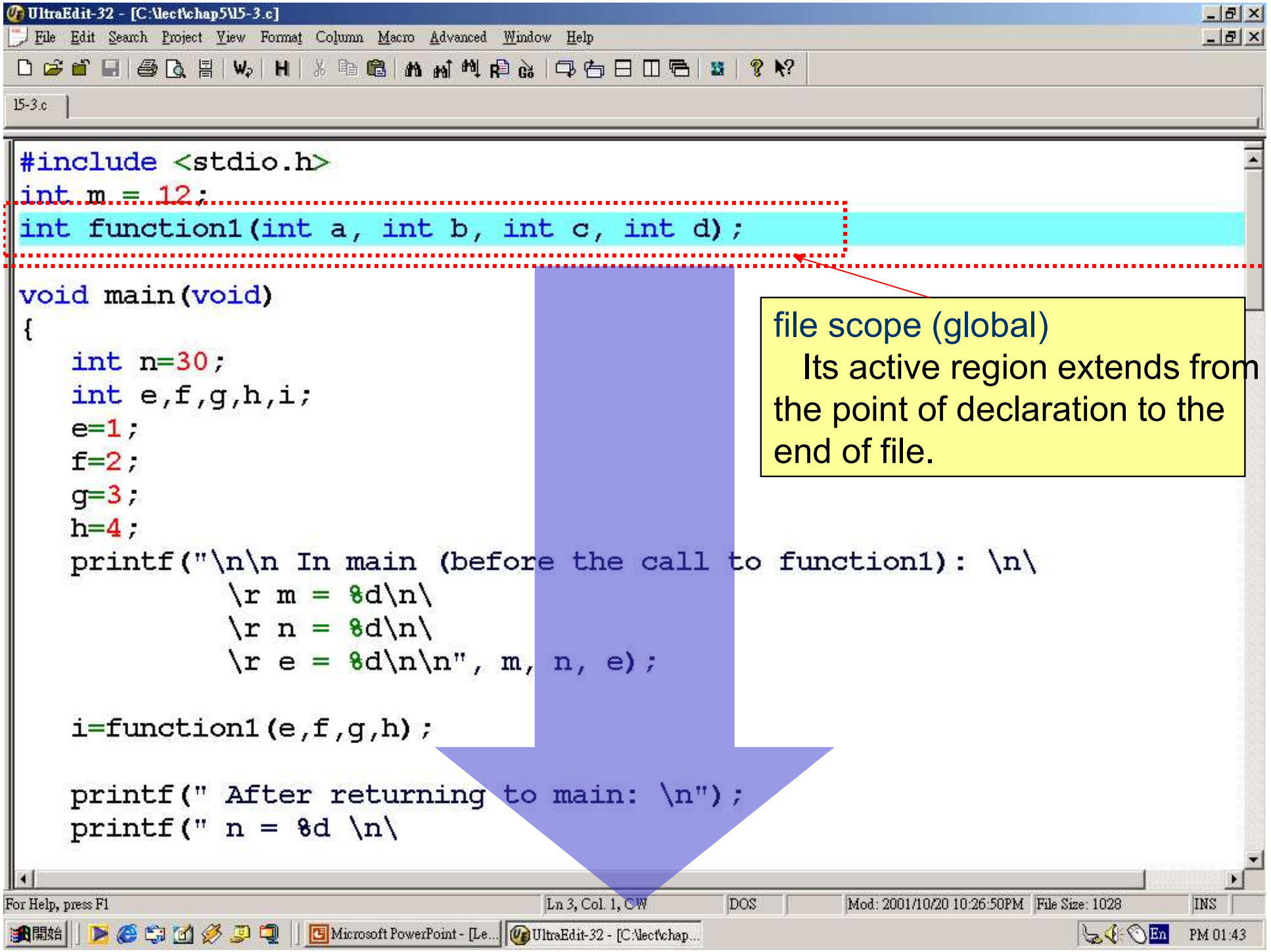
Scope and mechanics of passing values to functions

- Scope
- *Call by value*
- Function with many arguments returning one value
- Multiple returns a function
- Mechanics of passing values to functions

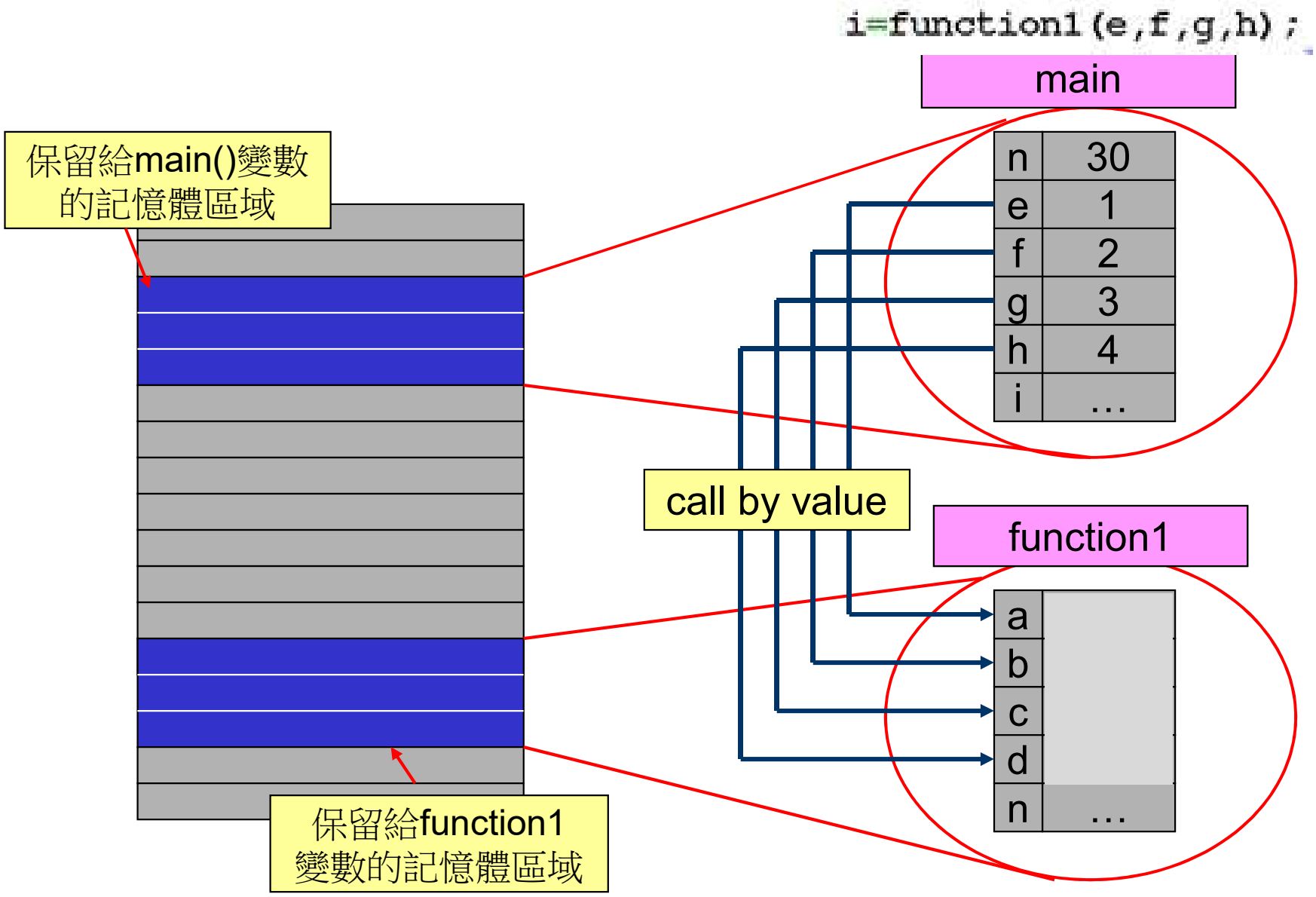
Scope

- *Scope* refers to the region in which a declaration is active
- In C, there are four kinds of scope:
  - block, function, file, prototype
- Scope for an identifier is determined by the location of the identifier's declaration


UltraEdit-32 - [C:\lect\chap5\l5-3.c]
File Edit Search Project View Format Column Macro Advanced Window Help
l5-3.c
#include <stdio.h>
int m = 12;
int function1(int a, int b
void main(void)
{
int n=30;
int e,f,g,h,i;
e=1;
f=2;
g=3;
h=4;
printf("\n\n In main (before the call to function1): \n\
\r m = %d\n\
\r n = %d\n\
\r e = %d\n\n", m, n, e);
i=function1(e,f,g,h);
printf(" After returning to main: \n");
printf(" n = %d \n\
m: file scope (global)
(避免使用此類視野的變數)
n: function scope (auto)
For Help, press F1
Ln 2, Col. 1, CW
DOS
Mod: 2001/10/20 10:26:50PM
File Size: 1028
INS
開始
Microsoft PowerPoint - [Le...
UltraEdit-32 - [C:\lect\chap...
PM 01:35

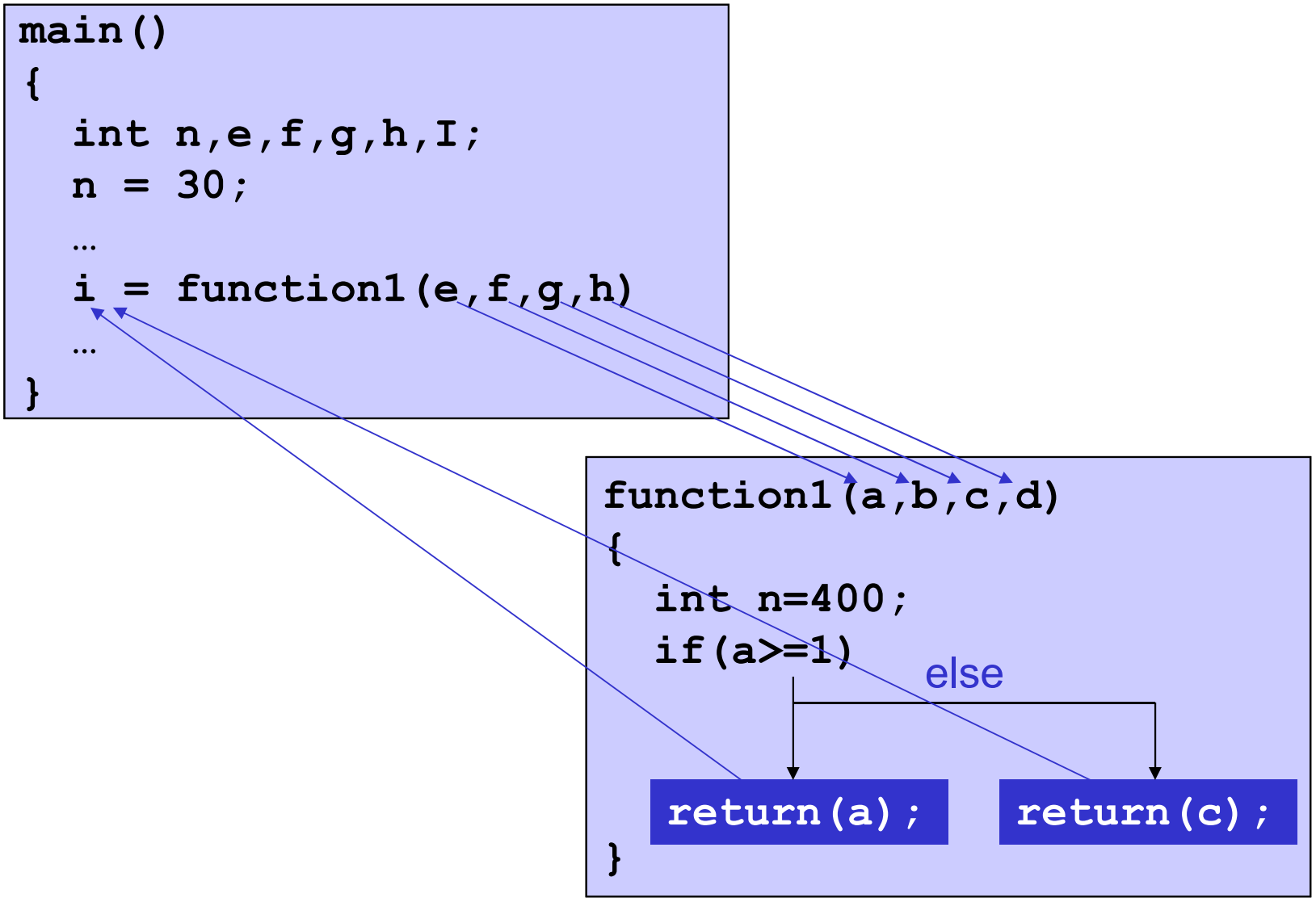

UltraEdit-32 - [C:\lect\chap5\l5-3.c]
File Edit Search Project View Format Column Macro Advanced Window Help
l5-3.c
#include <stdio.h>
int m = 12;
int function1(int a, int b, int c, int d);
void main(void)
{
int n=30;
int e,f,g,h,i;
e=1;
f=2;
g=3;
h=4;
printf("\n\n In main (before the call to function1): \n\
\r m = %d\n\
\r n = %d\n\
\r e = %d\n\n", m, n, e);
i=function1(e,f,g,h);
printf(" After returning to main: \n");
printf(" n = %d \n\
file scope (global)
Its active region extends from the point of declaration to the end of file.
For Help, press F1
Ln 3, Col. 1, CW
DOS
Mod: 2001/10/20 10:26:50PM
File Size: 1028
INS
開始
Microsoft PowerPoint - [Le...
UltraEdit-32 - [C:\lect\chap...
En
PM 01:43

The passing of arguments: Call by Value




```
int function1(int a, int b, int c, int d);
```

The returning value



Local and global variables

- Loosely speaking
  - *global* variables→file scope
  - *local* variables → function scope

A practice

Complete the program:

1. **Use global variables**
2. **Use static variables**

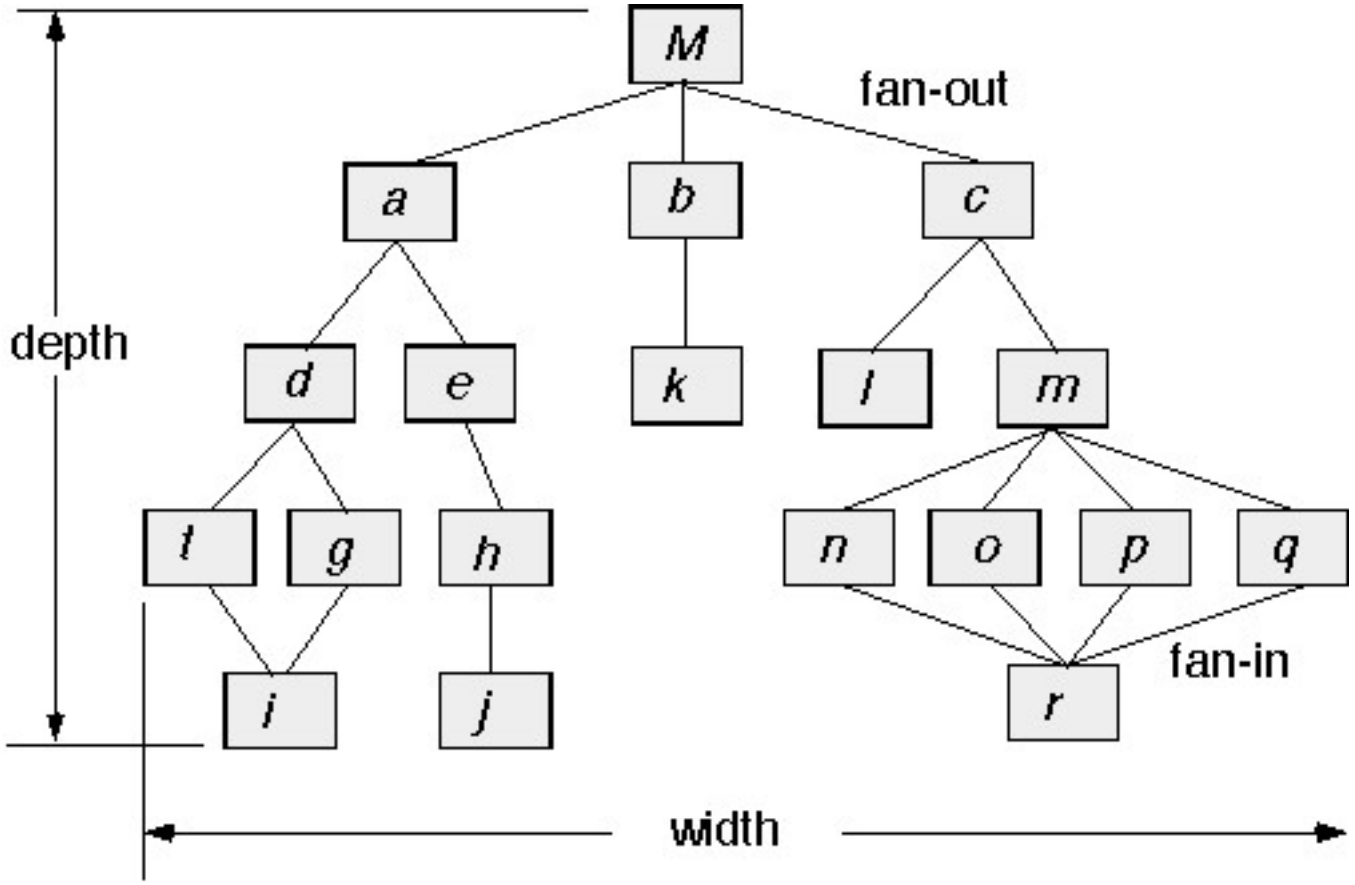
```
02  #include <stdio.h>
03  #include <stdlib.h>
04  void counter(void);
05  int main(void)
06  {
07     counter();
08     counter();
09     system("pause");
10     return 0;
11  }
12  void counter(void)
13  {
14
15  }
```

本習題的執行結果應如下所示：

```
counter()已被呼叫 1 次了...
counter()已被呼叫 2 次了...
```

Control Hierarchy

- Function呼叫要儘量扁平化
  - 要width-oriented, 不要depth-oriented



Something remains

- The memory space allocated to the variables in function() is freed up after the execution of function() has finished.
- To call function() a second time, the memory will be reserved again
  - possibly not at the same location
- Function definition could be also considered as function declaration

Random Number Generation (亂數產生器)

- rand() function
  - #include <stdlib.h>
  - Returns "random" number between 0 and RAND_MAX (at least 32767)

```
i = rand();
```

  - Pseudorandom
    - 每次呼叫rand()所傳回的亂數順序都一樣
- To get a random number between 1 and n

```
1 + ( rand() % n )
```

  - rand() % n returns a number between 0 and n - 1
  - Add 1 to make random number between 1 and n

    1 + (rand() % 6) ➔ number between 1 and 6

Random Number Generation

讓每次產生的亂數順序不一樣

- srand() function
  - #include <stdlib.h>
  - Function srand takes an unsigned integer argument and seeds function rand to produce a different sequence of random numbers for each execution of the program.

    ```
    srand( seed );
    ```

  - srand(time(NULL));
  - This causes the computer to read its clock to obtain the value for the seed automatically.
- time(NULL) function
  - #include<time.h>
  - Function time returns the number of seconds that have passed since midnight on January 1, 1970.

NULL就是0
通常用在指標

=

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int gen(int max); //generate a number between 1 to max


void main(void)
{
    int i;

    srand(time(NULL));  //請放在main program 中，只能呼叫一次

    //call function
    printf("call function\n");
    for(i=0;i<=4;i++)
        printf("generate = %d\n", gen(100)); //generate number between 1-100

    system("pause");
}

//generate a number between 1 to max
int gen(int max)
{
    return 1 + rand() % max;
}
```