CHAPTER 5 - FUNCTIONS

思考下面問題

- 你的程式在許多地方必須將三個數字由小到大印出的功能。你要怎麼辦？
  - 目前你的作法只能這樣...

/*將三個數字x1,x2,x3由小到大印出 */

....

/*將三個數字a,b,c由小到大印出 */

```
if (x1 >= x2) {
      if (x1 >= x3) {
            if (x2 >= x3) {
                  printf("%d %d %d\n",x3,x2,x1);
            } else {
                  printf("%d %d %d\n",x2,x3,x1);
            }
   } else {
            printf("%d %d %d\n",x2,x1,x3);
   }
} else if (x1 >= x3) {
      printf("%d %d %d\n",x3,x1,x2);
                                          x3) {
                                    intf("%d %d %d\n",x1,x3,x2);
                                    intf("%d %d %d\n",x1,x2,x3);
```

```
x1=a; x2=b; x3=c;
if (x1 >= x2) {
      if (x1 >= x3) {
            if (x2 >= x3) {
                  printf("%d %d %d\n",x3,x2,x1);
            } else {
                  printf("%d %d %d\n",x2,x3,x1);
            }
   } else {
            printf("%d %d %d\n",x2,x1,x3);
   }
} else if (x1 >= x3) {
      printf("%d %d %d\n",x3,x1,x2);
} else {
      if (x2 >= x3) {
            printf("%d %d %d\n",x1,x3,x2);
      } else {
            printf("%d %d %d\n",x1,x2,x3);
      }
}
```

  - 這種作法有沒有問題？

較好的作法---使用函式function

```
/*將三個數字x1,x2,x3由小到大印出 */
print_3int(x1,x2,x3);
....
 /*將三個數字a,b,c由小到大印出 */
print_3int(a,b,c);
```

```
void print_3int(int x1,int x2,int x3) {
    if (x1 >= x2) {
        if (x1 >= x3) {
            if (x2 >= x3) {
                printf("%d %d %d\n",x3,x2,x1);
            } else {
                printf("%d %d %d\n",x2,x3,x1);
            }
        } else {
            printf("%d %d %d\n",x2,x1,x3);
        }
    } else if (x1 >= x3) {
        printf("%d %d %d\n",x3,x1,x2);
    } else {
        if (x2 >= x3) {
            printf("%d %d %d\n",x1,x3,x2);
        } else {
            printf("%d %d %d\n",x1,x2,x3);
        }
    }
}
```

這種寫法會有哪些好處？

5.2 Program Modules in C

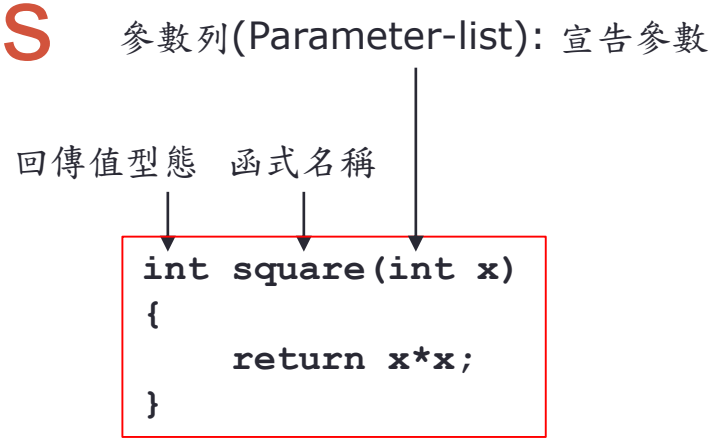
- 程式可以由許多小的模組（modules）構成。
- 每個小模組會比整個大程式較容易管理。
- 函式(Functions)
  - C的模組(Modules)
  - C程式乃是結合使用者定義的函式與程式庫函式而成
- 函式呼叫(Function calls)
  - 函式呼叫就像老闆叫員工去完成某件事，回傳結果。
- 設計函式需要設計下面項目
  - 函式名稱與參數(arguments (data))
  - 函式進行的運算
  - 函式回傳的結果

5.3 Math Library Functions

- 數學函式庫(Math library functions)
  - `#include <math.h>`
- 函式呼叫語法
  - `FunctionName(`*argument*`);`
    - 多個參數以逗點,隔開
  - `printf( "%.2f", sqrt( 900.0 ) );`
    - Calls function `sqrt`, which returns the square root of its argument
    - All math functions return data type `double`
- 參數可以是常數，變數，運算式等(須視函式宣告)。

Routine	Use
abs	Return absolute value of **int**
acos	Calculate arccosine
asin	Calculate arcsine
atan, atan2	Calculate arctangent
atof	Convert character string to **double**-precision floating-point value
Bessel functions	Calculate Bessel functions **_j0**, **_j1**, **_jn**, **_y0**, **_y1**, **_yn**
_cabs	Find absolute value of complex number
ceil	Find integer ceiling
_chgsign	Reverse sign of double-precision floating-point argument
_clear87, _clearfp	Get and clear floating-point status word
_control87, _controlfp	Get and modify floating-point control word and set new control-word value
_copysign	Return one value with sign of another
cos	Calculate cosine
cosh	Calculate hyperbolic cosine
difftime	Compute difference between two specified time values
div	Divide one integer by another, returning quotient and remainder
_ecvt	Convert **double** to string of specified length
exp	Calculate exponential
fabs	Find absolute value
_fcvt	Convert **double** to string with specified number of digits following decimal point
_finite	Determine whether given double-precision floating-point value is finite
floor	Find largest integer less than or equal to argument
fmod	Find floating-point remainder
_fpclass	Return status word containing information on floating-point class

5.5 Function Definitions



- 函式定義語法：

  return-value-type function-name(parameter-list)
  {
  declarations and statements
  }

  - 函式名稱(Function-name): any valid identifier
  - Declarations and statements: function body (block)
    - 可宣告變數
    - 函式內部不可以再宣告函式
  - 回傳值型態(Return-value-type) (default **int**)
    - 有回傳值的函式結束
      - **return** *expression*;
      - 例子

        ```
        return 0;
        return x;
        return n>= 0? 1 : 0;
        ```
    - 不用回傳值的函式結束
      - (return-value-type: void myfunc() { })
      - **使用return;**
      - 或是到達函式最尾端的}
  - 參數列(Parameter-list): 宣告參數

- **所有的在函式內宣告變數皆是區域變數(local variables)**
  - **函式參數(Parameters)**
    - **用來傳遞函式間資料**
    - **也是區域變數(Local variables)**
- 使用函式的好處
  - 將問題分割處理
  - 已設計好的函式可重複使用
  - 避免程式碼重複

程式結束(呼叫exit函式)

- 你可以在任何地方呼叫exit函式，**程式就會結束**。

```
// #include<stdlib.h>
void exit(int value);
```

其中value是指程式結束狀態

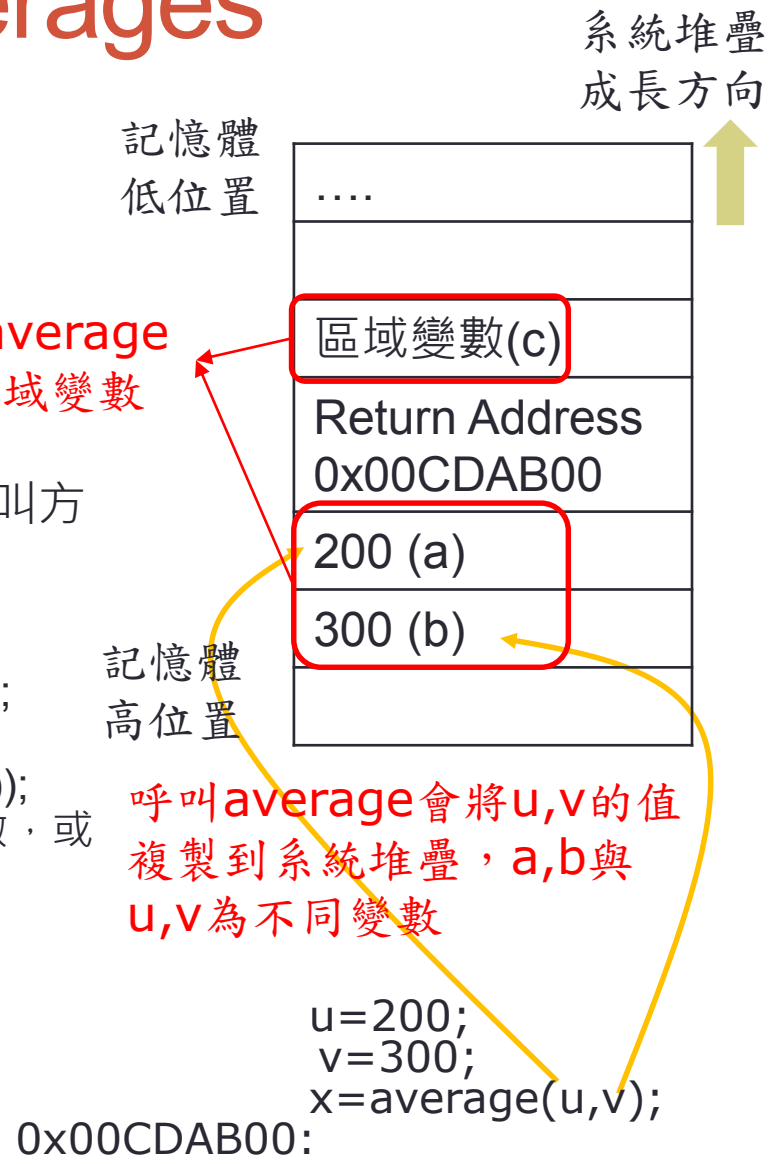
- 在main函式使用return *expression*;與呼叫exit(*expression*)效果一樣。

Program: compute averages

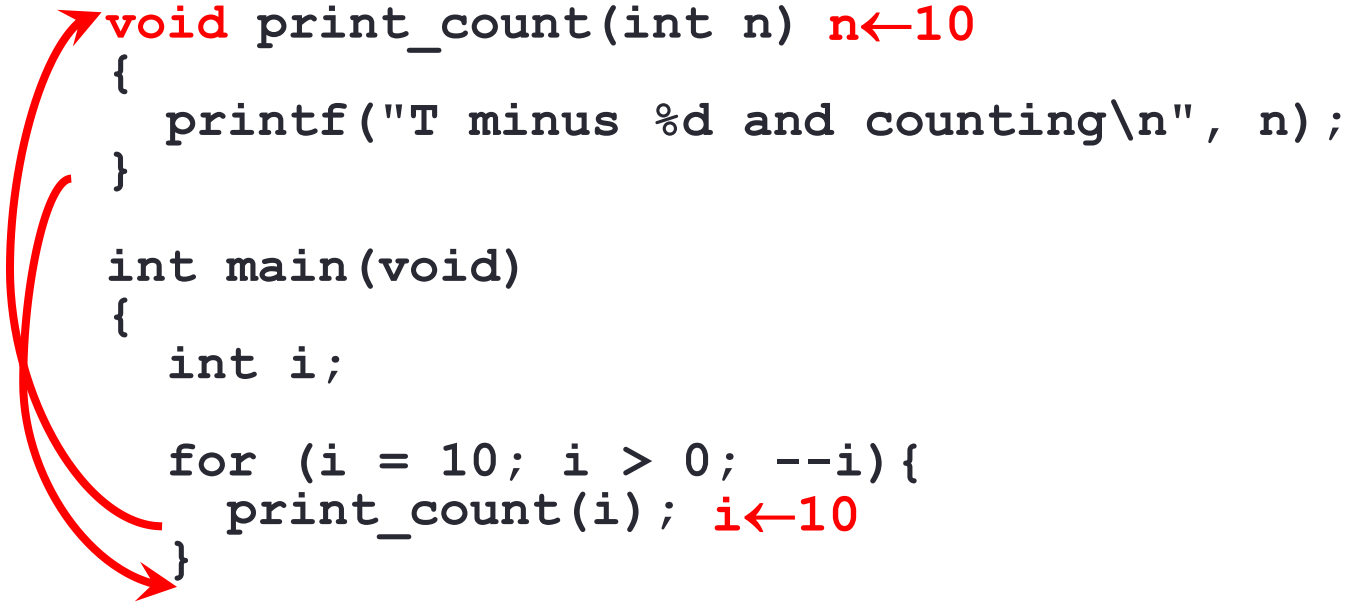
```
double average(double a, double b)
{
   int c;
   c = (a+b)/2;
   return c;
}
```

- 每當需要計算平均時可呼叫average, 可能的呼叫方式如下:
  - 計算u與v的平均並將結果給x: x=average(u,v);
  - 計算1.0與b的平均並將結果給x: x=average(1.0,b);
  - 計算3與5的平均並將結果給x: x=average(3,5);
  - 印出3與5的平均:printf("average %f",average(3,5));
  - 對於一個非void的函式，它所產生的值可以給變數，或在運算式中。

```
if(average(a,b)>=60) printf("passed\n")

x=average(a,b)+30;
```



Program: print a countdown



```
#include <stdio.h>

void print_count(int n) n←10
{
  printf("T minus %d and counting\n", n);
}

int main(void)
{
  int i;

  for (i = 10; i > 0; --i){
    print_count(i); i←10
  }

  return 0;
}
```

Program: print a pun

```
#include <stdio.h>

void print_pun(void)
{
  printf("To C, or not to C: that is the
  question.\n");
}

int main(void)
{
  print_pun();
  return 0;
}
```

Program: test whether a number is prime

```
#include <stdio.h>

int is_prime(int n)
{
  int divisor;

  if (n <= 1)
    return 0;
  for (divisor = 2;
     divisor * divisor <= n;
     divisor++)
       if (n % divisor == 0)
      return 0;
  return 1;
}
```

```
int main(void)
{
  int n;

  printf("Enter a number: ");
  scanf("%d", &n);
  if (is_prime(n))
    printf("Prime\n");
  else
    printf("Not prime\n");

  return 0;
}
```

5.6 Function Prototypes

- 函式原型(Function prototype)
  - 用來確認函式是否被正確引用
  - Function name
  - Parameters – what the function takes in
  - Return type – data type function returns (default **`int`**)

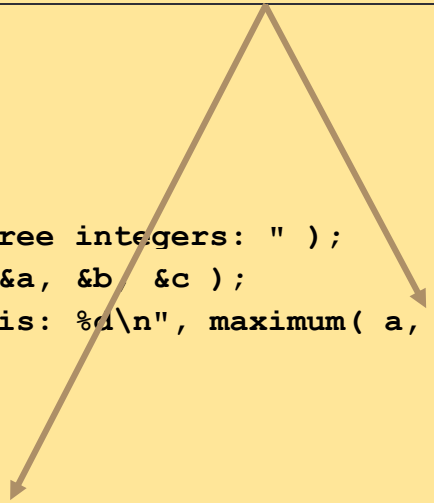
  - The function with the prototype

```
int maximum( int, int, int );
```

    - Takes in 3 **`int`**s
    - Returns an **`int`**

在這行之後的程式maximum函式原型的定義就生效

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

int maximum( int, int, int );   /* function prototype */

int main()
{
   int a, b, c;

   printf( "Enter three integers: " );
   scanf( "%d%d%d", &a, &b, &c );
   printf( "Maximum is: %d\n", maximum( a, b, c ) );

   return 0;
}

/* Function maximum definition */
int maximum( int x, int y, int z )
{
   int max = x;

   if ( y > max )
      max = y;

   if ( z > max )
      max = z;

   return max;
}
```



```
Enter three integers: 22 85 17
Maximum is: 85
```

參數型態轉換

- 萬一呼叫函式時其參數型態與原本定義不同，C編譯器會套用下面規則自動轉換參數型態。
  - 轉換發生在函式呼叫前。
  - 若函式呼叫前已定義函式prototype，C編譯器會根據函式prototype裡參數的型態來轉換。

    double average(double,double);

    x=average(1, 5); // 1→1.0; 5→5.0 (int->double)

  - 若函式呼叫前未定義函式prototype，C編譯器會根據下面規則轉換參數型態
    - float → double
    - char, short → int
    - 當然可能轉錯!!

```
#include<stdio.h>
int main(void)
{
   double x=3.0;
   printf("square: %d\n", square(x));
   printf("square: %d\n", square((int)x));
}
int square(int n)
{
   return n*n;
}
```

n會得到錯誤的值

n會得到正確的值

5.7 Function Call Stack and Stack Frames

- 函式呼叫堆疊(Call Stack)

  函式呼叫時會將函式結束後要回到的位置與區域變數等資料，存放在一個先進後出的資料結構(堆疊(Stack))裡。

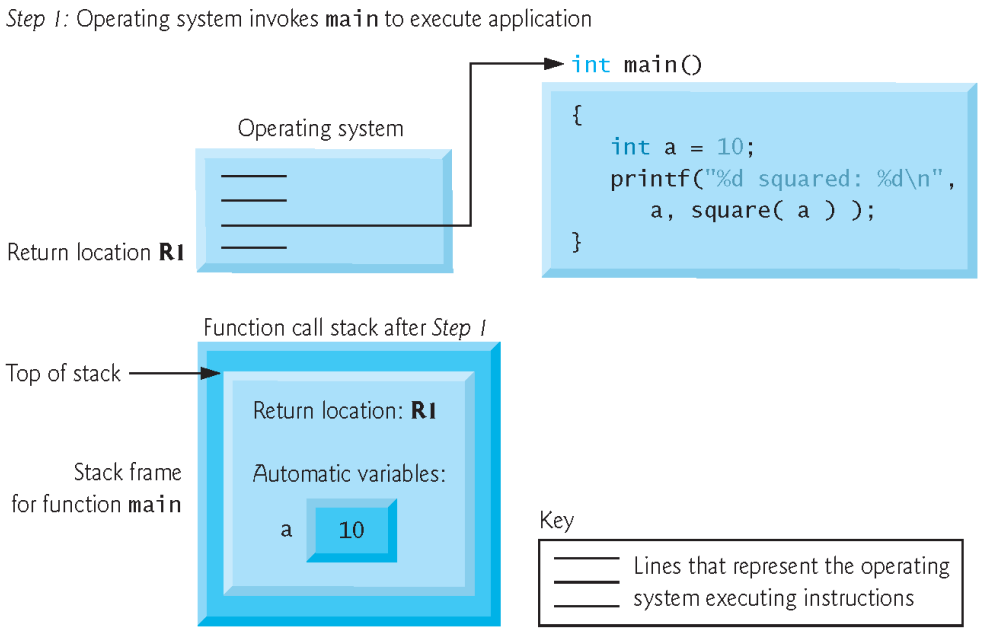


Fig. 5.7 | Function call stack after the operating system invokes `main` to execute the program.

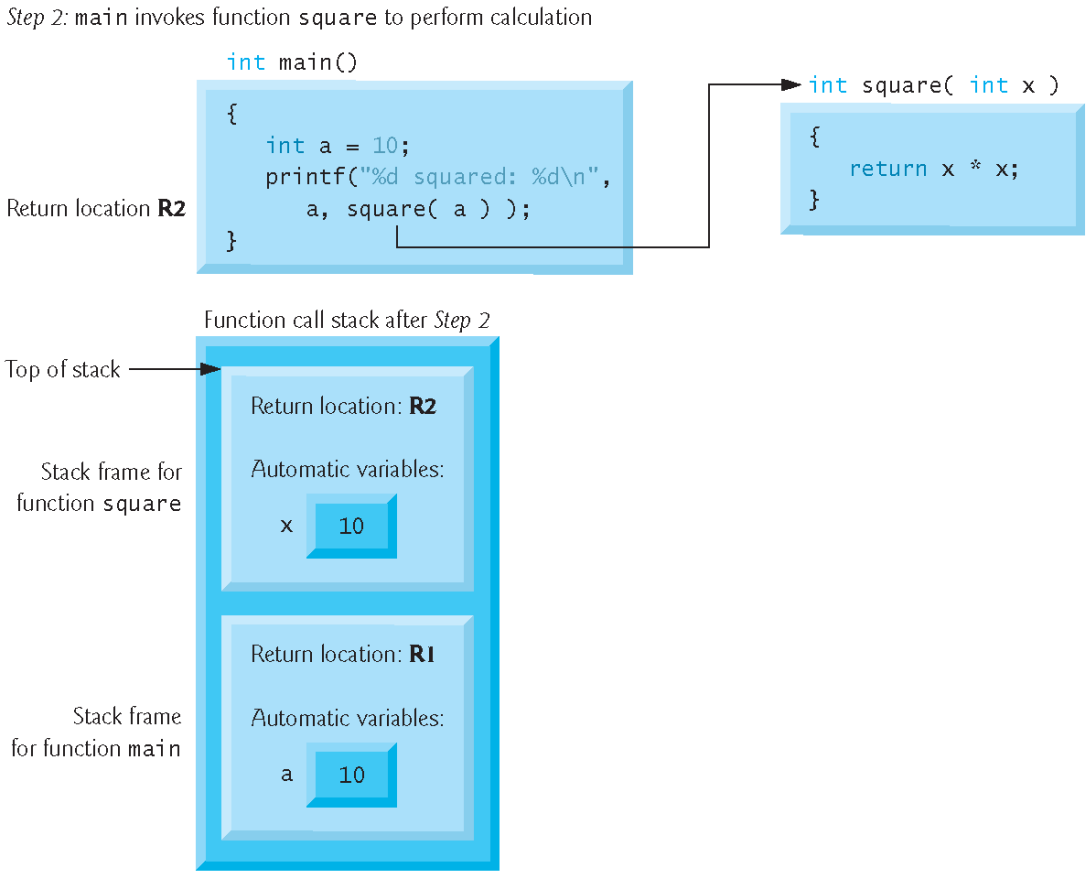


Fig. 5.8 | Function call stack after `main` invokes `square` to perform the calculation.

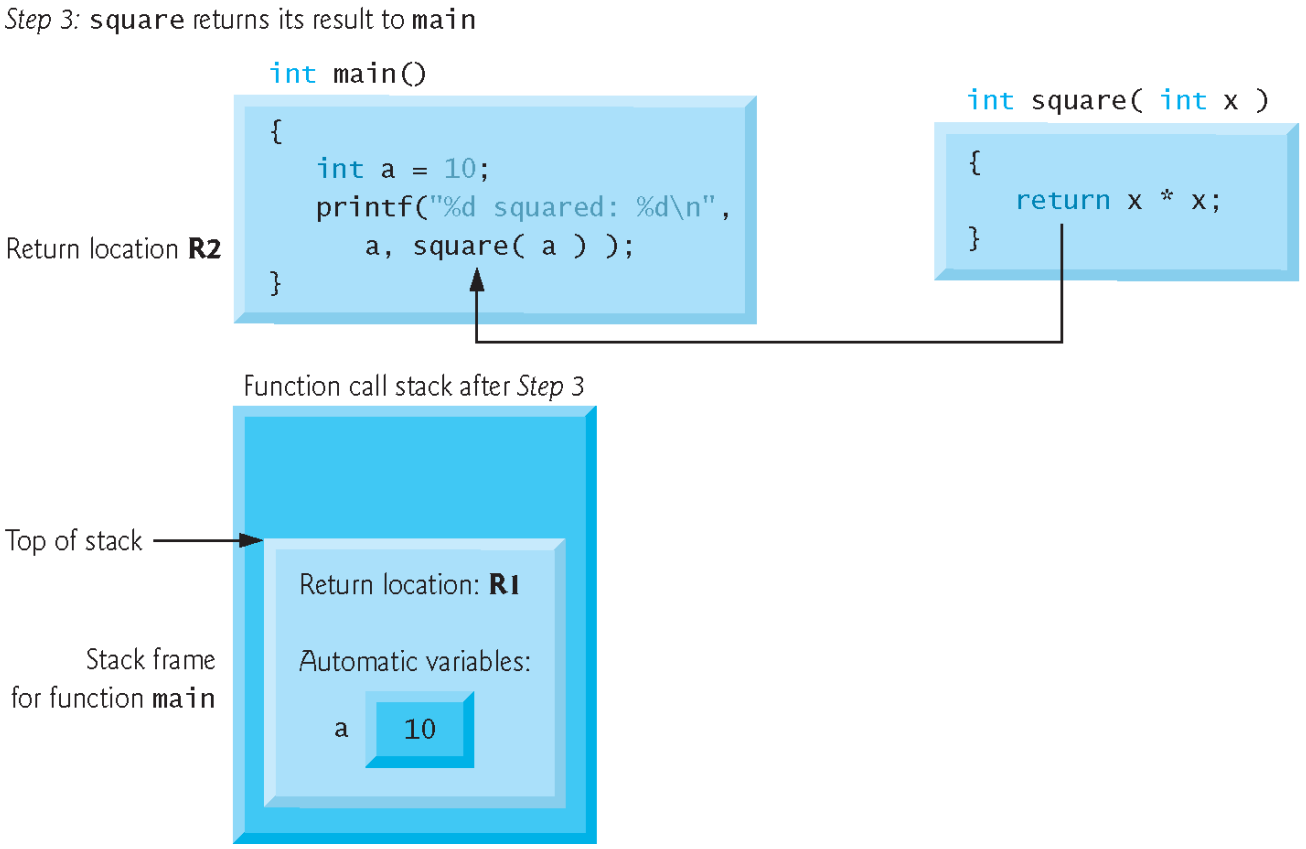


Fig. 5.9 | Function call stack after function `square` returns to `main`.

5.8 Header Files

- C標準程式庫標頭檔(Header files)
  - 用來包含資料型態與函式原型
  - 例如 **<stdio.h>** ,**<stdlib.h>** , **<math.h>** , etc
  - 載入命令語法 **#include <filename>**

    #include <math.h>

- 客製標頭檔
  - Create file with functions
  - 慣用的命名規則**filename.h**
  - 載入命令語法 **#include "filename.h"**
  - Reuse functions

Separated Compilation

```
#include<stdio.h>

int maximum(int, int, int);

int main()
{
   int a,b,c;
   printf("Enter three integers:");
   scanf("%d%d%d",&a,&b,&c);
   printf("Maximum is: %d\n",maximum(a,b,c));
    return 0;
}


int maximum(int x,int y,int z)
{
    int max = x;
    if (y > max) max = y;
    if (z > max) max = z;
    return max;
}
```

A single C file

```
#include<stdio.h>
#include"maximum.h"
int main()
{
   int a,b,c;
   printf("Enter three integers:");
   scanf("%d%d%d",&a,&b,&c);
   printf("Maximum is: %d\n",
                maximum(a,b,c));
    return 0;
}
```

Main program file: main.c

```
#ifndef MAXIMUM_H
#define MAXIMUM_H
// function prototype
int maximum(int,int,int);
#endif
```

```
#pragma once
// function prototype
int maximum(int,int,int);
```

Header file:maximum.h

```
#include"maximum.h"
int maximum(int x,int y,int z)
{
    int max = x;
    if (y > max) max = y;
    if (z > max) max = z;
    return max;
}
```

Program file: maximum.c

5.9 Calling Functions: Call by Value and Call by Reference

- 呼叫函式時有時必須傳遞參數

- **傳值呼叫(Call by value)—複製一份傳給函式(分身)**
  - Copy of argument passed to function
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes

C, C++

- **傳址呼叫(Call by reference)—將參數本尊送過去**
  - Passes original argument
  - Changes in function effect original
  - Only used with trusted functions

C++

例子

- p=power(x,m); //呼叫power後m值沒變

```
int power(int x, int n)
{
   int result=1;
   for(;n>=1; --n) result *= x;
   return result;
}
```

- decompose(34.25,intp,fracp); //intp=? , fracp=?

```
void decompose(double x, long int_part, double frac_part)
{
   int_part = x;
    frac_part=x-int_part;
}
```

intp的值複製給int_part後
兩者就沒關係了

5.10 Random Number Generation

- **`rand` function**
  - **`#include<stdlib.h>`**
  - 回傳一個介於`0`與**`RAND_MAX`**的亂數

    ```
    i = rand();
    ```
  - 假亂數(Pseudorandom)
    - Preset sequence of "random" numbers
    - Same sequence for every function call
- Scaling
  - 產生1與 n間的亂數

    ```
    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`

5.10 Random Number Generation

- **srand function**
  - **<stdlib.h>**
  - 設定產生亂數序列啟始資訊

    ```
    srand( seed );
      //seed:種子
    ```

  - **srand(time(NULL)); //load <time.h>**
    - 取得目前時間來當序列啟始種子
    - **time(NULL)**
      - 得到目前的時間
      - 讓啟始種子每次都不一樣

```c
/* Fig. 5.9: fig05_09.c
   Randomizing die-rolling program */
#include <stdlib.h>
#include <stdio.h>

int main()
{
   int i;
   unsigned seed;

   printf( "Enter seed: " );
   scanf( "%u", &seed );
   srand( seed );

   for ( i = 1; i <= 10; i++ ) {
      printf( "%10d", 1 + ( rand() % 6 ) );

      if ( i % 5 == 0 )
         printf( "\n" );
   }

   return 0;
}
```

```
Enter seed: 67
        6       1       4       6       2
        1       6       1       6       4
```

```
Enter seed: 867
        2       4       6       1       6
        1       1       3       6       2
```

```
Enter seed: 67
        6       1       4       6       2
        1       6       1       6       4
```

5.11Example: A Game of Chance

- Craps simulator
- Rules
  - 丟兩個骰子(Roll two dice)
    - 第一次丟7 or 11 玩家贏
    - 第一次丟2, 3, or 12玩家輸
    - 第一次丟4, 5, 6, 8, 9, 10記下這個值 - value becomes player's "point"
  - 一直丟直到丟出他記的值(贏)或丟出七(輸)

```c
/* Fig. 5.10: fig05_10.c
   Craps */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int rollDice( void );

int main()
{
   int gameStatus, sum, myPoint;

   srand( time( NULL ) );
   sum = rollDice();             /* first roll of the dice */

   switch ( sum ) {
      case 7: case 11:           /* win on first roll */
         gameStatus = 1;
         break;
      case 2: case 3: case 12:   /* lose on first roll */
         gameStatus = 2;
         break;
      default:                   /* remember point */
         gameStatus = 0;
         myPoint = sum;
         printf( "Point is %d\n", myPoint );
         break;
   }

   while ( gameStatus == 0 ) {    /* keep rolling */
      sum = rollDice();

```

```
33       if ( sum == myPoint )       /* win by making point */
34          gameStatus = 1;
35       else
36          if ( sum == 7 )          /* lose by rolling 7 */
37             gameStatus = 2;
38    }
39
40    if ( gameStatus == 1 )
41       printf( "Player wins\n" );
42    else
43       printf( "Player loses\n" );
44
45    return 0;
46 }
47
48 int rollDice( void )
49 {
50    int die1, die2, workSum;
51
52    die1 = 1 + ( rand() % 6 );
53    die2 = 1 + ( rand() % 6 );
54    workSum = die1 + die2;
55    printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );
56    return workSum;
57 }
```

```
Player rolled 6 + 5 = 11
Player wins
```

```
Player rolled 6 + 6 = 12
Player loses
```

```
Player rolled 4 + 6 = 10
Point is 10
Player rolled 2 + 4 = 6
Player rolled 6 + 5 = 11
Player rolled 3 + 3 = 6
Player rolled 6 + 4 = 10
Player wins
```

```
Player rolled 1 + 3 = 4
Point is 4
Player rolled 1 + 4 = 5
Player rolled 5 + 4 = 9
Player rolled 4 + 6 = 10
Player rolled 6 + 3 = 9
Player rolled 1 + 2 = 3
Player rolled 5 + 2 = 7
Player loses
```

問題

下面程式產生的100個亂數夠亂嗎？

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main()
{
        int k;
        for(k = 0; k < 100; ++k) {
                srand(time(NULL));
                printf("%d\n",rand());
        }
        return 0;
}
```

5.12 Storage Classes

- **Storage class specifiers**
  - 變數（物件）儲存持續期間(Storage duration)
    - how long an object exists in memory
      - Automatic storage duration
      - Static storage duration
  - 領域(Scope) –
    - where object can be referenced in program
      - File scope
      - Function scope
      - Block scope
      - Function prototype score
  - 連結(Linkage) –
    - specifies the files in which an identifier is known

Storage duration

- **Automatic storage duration**
  - 變數(物件)在區塊內建立與消滅，離開區塊就不能用了
  - **auto**:
    ```
    auto double x, y; //在C++裡 auto有其他意思
    等同於double x, y;
    ```
  - **register**:
    - 將變數放在宣告在暫存器內
    - Can only be used for automatic variables
      ```
      register int counter = 1;
      ```
- **Static storage duration**
  - 變數(物件)在程式執行時一直存在直到程式結束就不能用了
  - 內定值是0(Default value of zero)
    - 若是要其它值呢？
    - static int count = 1;
  - **static**: local variables defined in functions.
    - 函式結束時值仍然保留

    ```
    void func()
    {
       static int x = 1;
    }
    ```
  - **extern**: default for global variables and functions
    - Known in any function

Example

```
// this function sets the seed for the random
// number generator only at the first call
unsigned myrand()
{
   static int flag = 0;
   if (flag == 0) {
        srand(time(NULL));
        flag = 1;
   }
   return rand();
}
```

5.13領域規則(Scope Rules)

- **檔案領域(File scope)**
  - 定義在函式外面，在定義之後的指令都可以使用
    - 全域變數(global variables)
    - 函式定義(function definitions)
    - 函式原型(function prototypes)

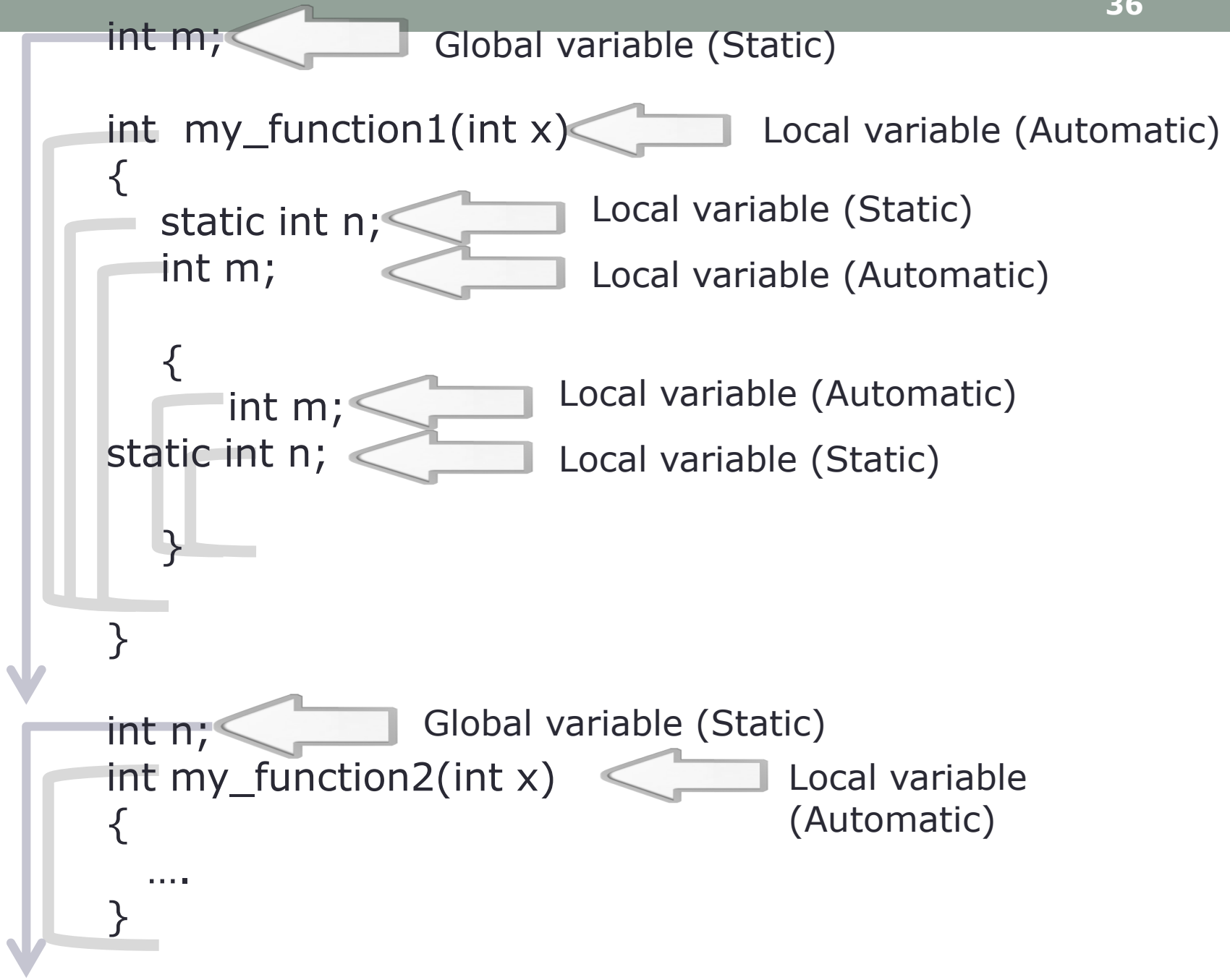
- **函式領域(Function scope)**
  - 只能在函式內被使用
  - 使用標籤(labels) (`start:`, `case:` , etc.)

5.13領域規則(Scope Rules)

- **區塊領域(Block scope)**
  - 區塊({ … })內宣告的變數
    - Block scope begins at declaration, ends at right brace
  - 區域變數(local variables)
  - Used for variables, function parameters (local variables of function)
  - 外面區塊的變數會被裡面區塊的變數 “遮住”

```
{ int x;
    x = 1;
    {
            int x;
            x = 2;
            printf(“%d\n”,x);          /*2 will be output*/
    }
}
```

- **函式原型領域(Function prototype scope)**
  - void func(int x, int y);
  - Used for identifiers in parameter list


int m;
Global variable (Static)
int my_function1(int x)
Local variable (Automatic)
{
static int n;
Local variable (Static)
int m;
Local variable (Automatic)
{
int m;
Local variable (Automatic)
static int n;
Local variable (Static)
}
}
int n;
Global variable (Static)
int my_function2(int x)
Local variable (Automatic)
{
....
}

```
#include<stdio.h>
void a(void);
void b(void);
void c(void);

int x=1;

int main(void)
{
   int x = 5; // local variable
   printf("local x in outer scope of main is %d\n", x );
   {
      int x = 7;
      printf( "local x in inner scope of main is %d\n", x );
   }
   printf( "local x in outer scope of main is %d\n", x );
   a();
   b();
   c();
   a();
   b();
   c();
   printf( "local x in main is %d\n", x );
   return 0;
}
```

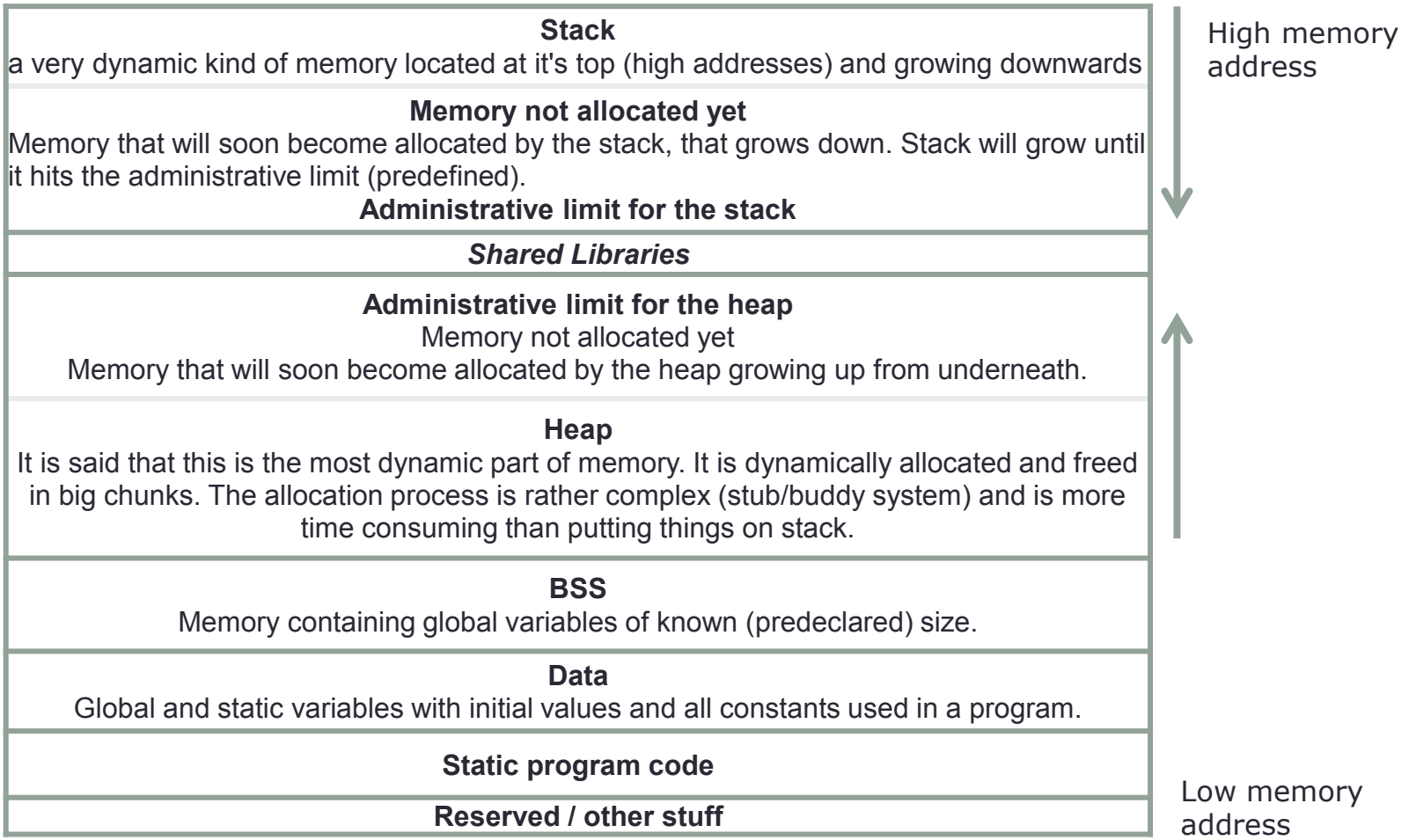
```
void a(void)
{
   int x = 25;
   printf( "\nlocal x in a is %d after entering a\n", x );
   x++;
   printf( "local x in a is %d before exiting a\n", x );
}
void b(void)
{
   static int x=50;
   printf( "\nlocal static x is %d after entering b\n", x );
   x++;
   printf( "local static is %d before exiting b\n", x );
}
void c(void)
{
   printf( "\nglobal x is %d after entering c\n", x );
   x*=10;
   printf( "global x is %d before exiting c\n", x );

}
```

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5
local x in a is 25 after entering a
local x in a is 26 before exiting a
local static x is 50 on entering b
local static x is 51 on exiting b
global x is 1 on entering c
global x is 10 on exiting c
```

```
local x in a is 25 after entering a
local x in a is 26 before exiting a
local static x is 51 on entering b
local static x is 52 on exiting b
global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```

Program memory

- 程式在記憶體會被組織成下面段落
  - Data segment
    - Data: initialized global and static variables
    - BSS: uninitialized global and static variables
    - Heap: dynamically allocated variables
      - managed by malloc and free
  - Stack segment
    - Automatic variables
    - Return addresses
  - Code (Text) segment


Stack
a very dynamic kind of memory located at it's top (high addresses) and growing downwards
Memory not allocated yet
Memory that will soon become allocated by the stack, that grows down. Stack will grow until it hits the administrative limit (predefined).
Administrative limit for the stack
Shared Libraries
Administrative limit for the heap
Memory not allocated yet
Memory that will soon become allocated by the heap growing up from underneath.
Heap
It is said that this is the most dynamic part of memory. It is dynamically allocated and freed in big chunks. The allocation process is rather complex (stub/buddy system) and is more time consuming than putting things on stack.
BSS
Memory containing global variables of known (predeclared) size.
Data
Global and static variables with initial values and all constants used in a program.
Static program code
Reserved / other stuff
High memory address
Low memory address

Global Variables and Side Effects

```
#include<stdio.h>
int g;
int f(int x)
{
   g++;
   return x;
}
```

```
int main()
{
      int z0,z1, z2;
      g   = 0;
      printf("%d,%d\n",f(1),g);
      g   = 0;
      z0  = f(1);
      printf("%d,%d\n",z0,g);
      g   = 0;
      z1 = f(2)*f(g);
      g   = 0;
      z2 = f(g)*f(2);
      printf("z1=%d,z2=%d\n",z1,z2);
      return 0;
}
```

(Code::Block) Output

```
1,0
1,1
z1=2,z2=0
```

遞迴函式(Recursive functions)

- 函式自己呼叫自己（間接或直接）。
- 有些問題用遞迴函式較容易解決。
- 範例：產生1…n的所有排列。

若n=3

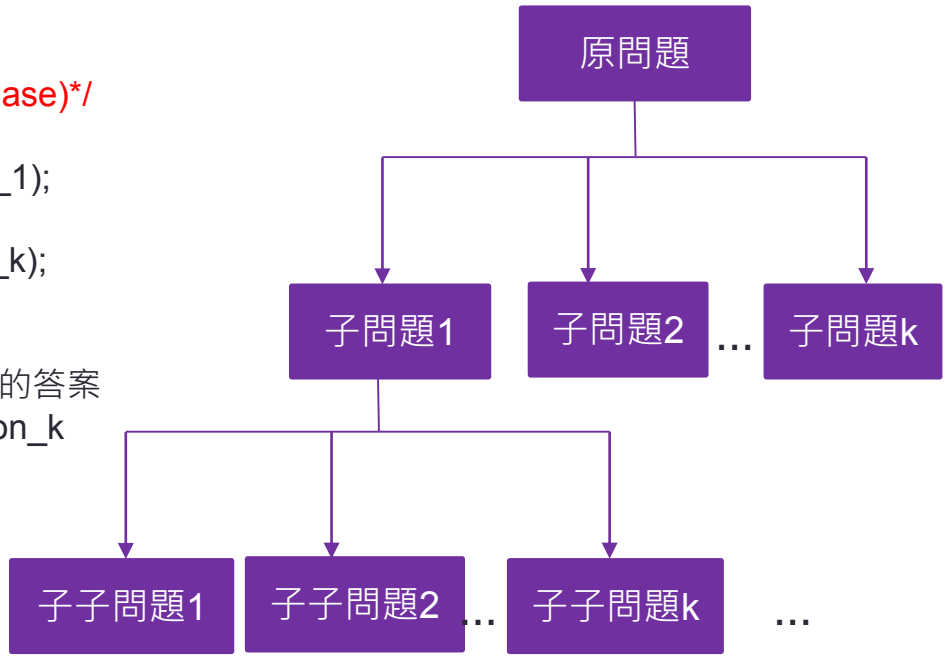
```
for(u1=1;u1<=3;u1++) {
        for(u2=1;u2<=3;u2++) {
                for(u3=1; u3<=3; u3++) {
                        if (u1!=u2&&u1!=u3&&u2!=u3) {
                                printf("(%d %d %d)\n",u1,u2,u3);
                        }
                }
        }
}
```

若n=10?

遞迴函式(Recursive functions)

- 函式自己呼叫自己（間接或直接）。
- 有些問題用遞迴函式較容易設計程式。
- 有些問題可用把大問題切成若干子問題，分別解決子問題後，再整合成原本問題答案的方式計算。

```
solution recursive_solver(problem)
{
    /* Base Case */
    /* 處理 Base Case */

    /* 將原本問題切成較小的子問題 (朝Base Case)*/
    /* 解決各個子問題 */
    solution_1 = recursive_solver(subproblem_1);
    /* … */
    solution_k = recursive_solver(subproblem_k);

    /*
        藉由結合子問題的答案，產生原本問題的答案
    solution <- solution_1, solution_2,…,solution_k
     */
    return solution;
}
```



範例: 階乘(factorial)

- 5! = 5 * 4 * 3 * 2 * 1
- 它有遞迴定義

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times factorial(n-1) & \text{elsewhere} \end{cases}$$

根據遞迴公式用手推算

f(5)=5xf(4)=5x4xf(3)=5x4x3xf(2)
=5x4x3x2xf(1)=5x4x3x2x1xf(0)=5x4x3x2x1x1
=5x4x3x2x1=5x4x3x2=5x4x6=5x24=120

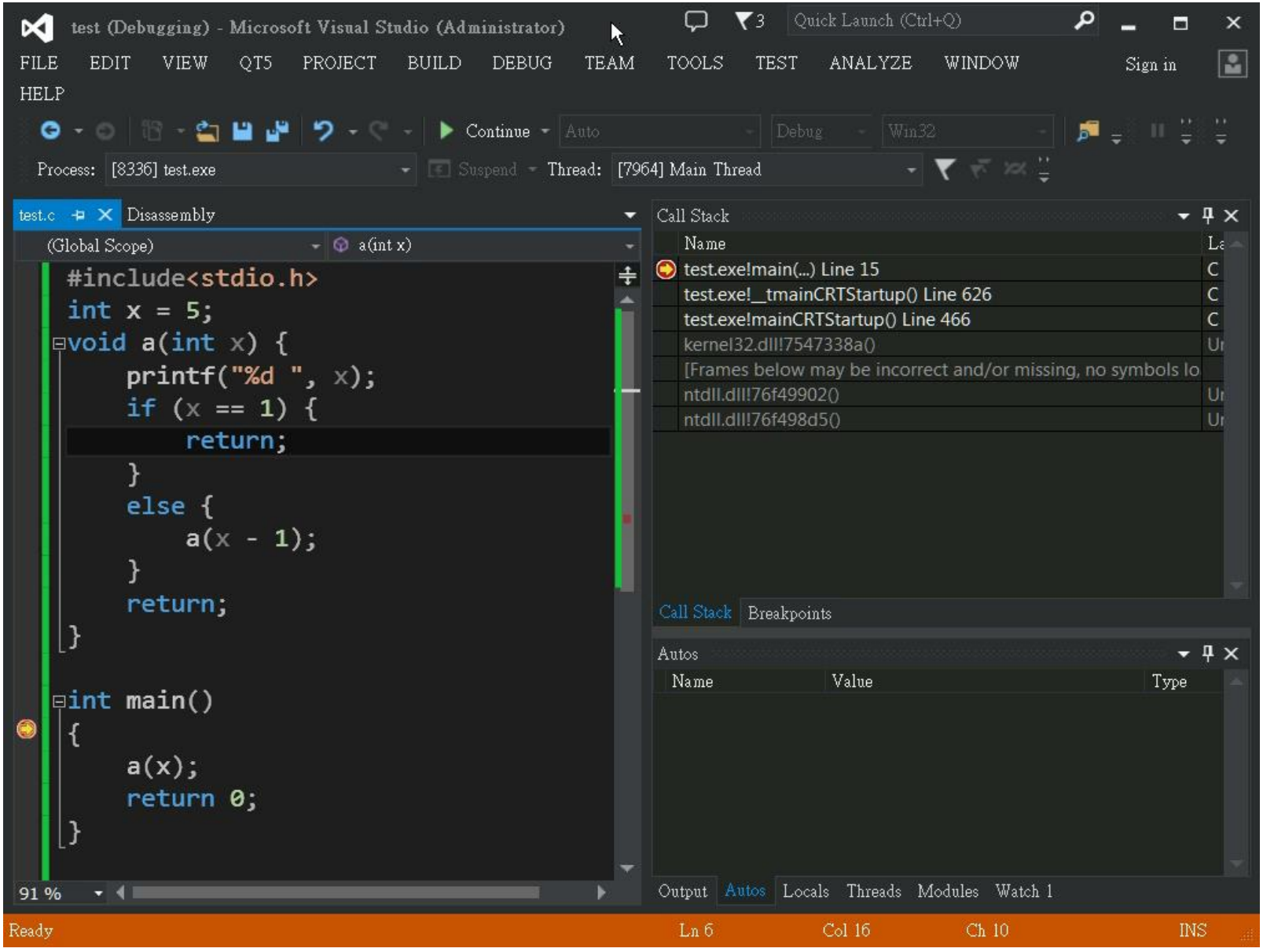
```
//C程式遞迴版本
int factorial(int n)
{
    if (n==0) return 1;
    return n*factorial(n-1);
}
```

Base case

遞迴呼叫

```
//C程式迴圈版本
int factorial(int n)
{
    int f = 1;
    for(;n>1; n--) f *= n;
    return f;
}
```

以除錯器追蹤遞迴程式計算過程



練習:推算遞迴函式

```
int a(int m,int n)
{
    if (m<3) {
        if (n < 3) {
                return m+n;
        } else {
                return a(m,n-2) + n;
        }
    } else {
        return a(m-1,n) + m;
    }
}
```

用手推算a(5,5)

a(5,5)
=a(4,5)+5
=(a(3,5)+4)+5
=((a(2,5)+3)+4)+5
=(((a(2,3)+5)+3)+4)+5
=((((a(2,1)+3)+5)+3)+4)+5
=((((3+3)+5)+3)+4)+5
=23

練習:計算gcd(x,y),其中x>=y

- 以遞迴函式方式設計:
  - 若x是y的倍數: gcd(x,y) = y;
  - 否則: gcd(x,y)=gcd(y, x%y)
    - 假設x和y的gcd等於k
    - $y = k \times q$
    - $x = k \times p = k \times \left(\left\lfloor \frac{p}{q} \right\rfloor \times q + p\%q\right) = \left\lfloor \frac{p}{q} \right\rfloor \times k \times q + k \times (p\%q)$

$$\Rightarrow x\%y = k \times (p\%q)$$

遞迴公式1:

$$\gcd(x,y) = \begin{cases} y & if\ x\%y = 0 \\ \gcd(y, x\%y) & otherwise \end{cases}$$

遞迴公式2:(只用減法)

$$\gcd(x,y) = \begin{cases} y & if\ x = y \\ \gcd(x-y, y) & if\ x > y \\ \gcd(x, y-x) & if\ x < y \end{cases}$$

練習:計算次方

- 計算f(x,n)=x^n，其中n為大於等於0的整數。
- 以遞迴函式方式設計:

遞迴公式1:

$$f(x,n)=\begin{cases}1 & if\ n=0\\ x\times f(x,n-1) & otherwise\end{cases}$$

遞迴公式2:

$$f(x,n)=\begin{cases}1 & if\ n=0\\ x\times f(x,\lfloor n/2\rfloor)^2 & if\ n\ 是奇數\\ f(x,\lfloor n/2\rfloor)^2 & if\ n\ 是偶數\end{cases}$$

練習:計算次方 (Cont'd)

遞迴公式

$$f(x,n)=\begin{cases}1 & if\ n=0\\ x\times f(x,\lfloor n/2\rfloor)^2 & if\ n\ 是奇數\\ f(x,\lfloor n/2\rfloor)^2 & if\ n\ 是偶數\end{cases}$$

```
long long int power(int x,int n)
{
    long long int y;
    if (n == 0) return 1;
    if (n == 1) return x;
    y = power(x,n/2);
    return n%2 ? y*y*x : y*y;
}
```

另一個算法 $x^n = x^{a_m 2^m + a_{m-1}2^{m-1}+\cdots+a_2 2^2+a_1 2^1+a_0 2^0}$

$= x^{a_m 2^m} \times x^{a_{m-1}2^{m-1}} \times \ldots \times x^{a_2 2^2} \times x^{a_1 2^1} \times x^{a_0 2^0}$

$a_i = 1$ x^{2^m} $x^{2^{m-1}}$ ..... x^4 x^2 x^1

$a_i = 0$ 1 1 1 1 1

```
answer = 1;
for(;n>0;n/=2) {
    if(n%2) answer *= x;
     x*=x;
}
```

費氏級數(The Fibonacci Series)

- **費氏級數:**每項是前兩項的和
  - **0, 1, 1, 2, 3, 5, 8...**
- 費氏級數遞迴定義

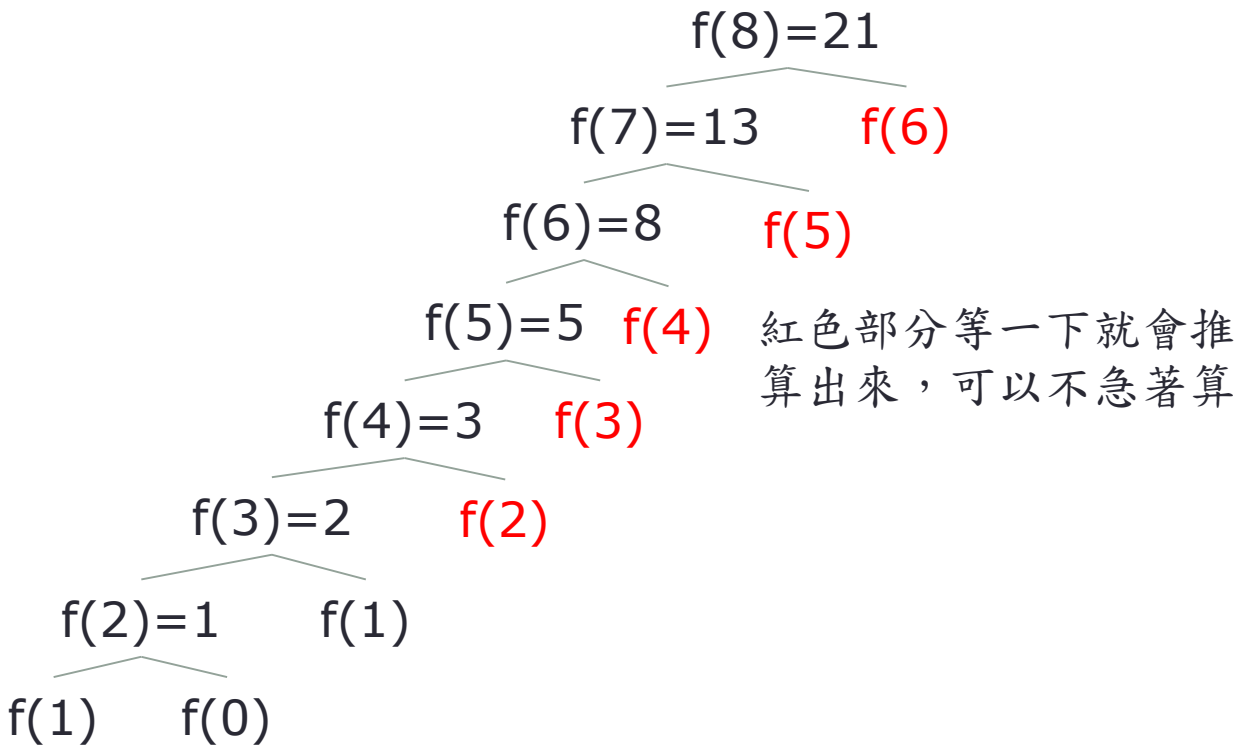
$$fibonacci(n) = \begin{cases} n & \text{if } n \le 1 \\ fibonacci(n-1) + fibonacci(n-2) & \text{elsewhere} \end{cases}$$

```
long fibonacci( long n )
{
   if (n == 0 || n == 1)  // base case
     return n;
   else
     return fibonacci( n - 1) + fibonacci( n - 2 );
}
```

用遞迴公式推算費氏級數

- 用手推算fibonacci(8)

$$f(n) = \begin{cases} n & \text{if } n \le 1 \\ f(n-1) + f(n-2) & \text{elsewhere} \end{cases}$$

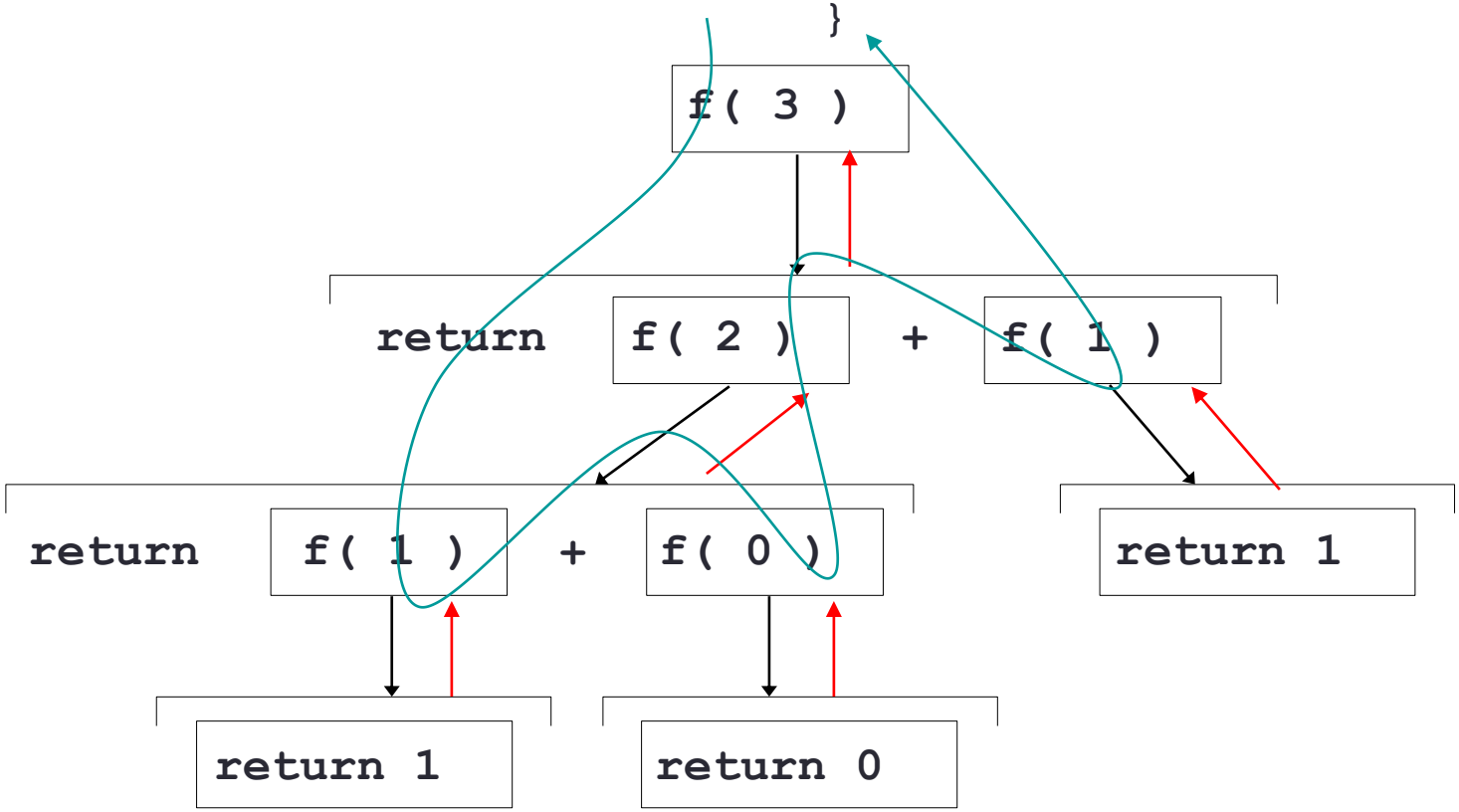


紅色部分等一下就會推算出來，可以不急著算

檢視費氏級數遞迴函式計算過程

- 計算f(3)的過程

```
long fibonacci( long n )
{
      if (n == 0 || n == 1)  // base case
            return n;
      else
            return fibonacci( n - 1) + fibonacci( n – 2 );
}
```



練習:爬樓梯

- 有一個n階的樓梯，每次可踏上一階或兩階。那麼爬上此n階樓梯有幾總走法?

- 思考方式:

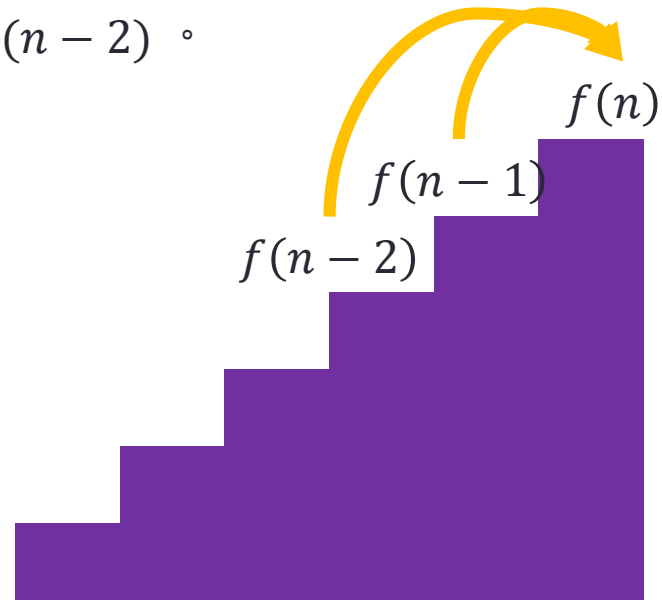
假設$f(n)$為爬上此n階樓梯總走法數。先試試小例子:

$$f(1) = 1$$
$$f(2) = 2$$

想看看一般狀況$f(n)$:因為最後一步可以踏一階或兩階到達第n階。因此得到

$$f(n) = f(n-1) + f(n-2) \text{ 。}$$

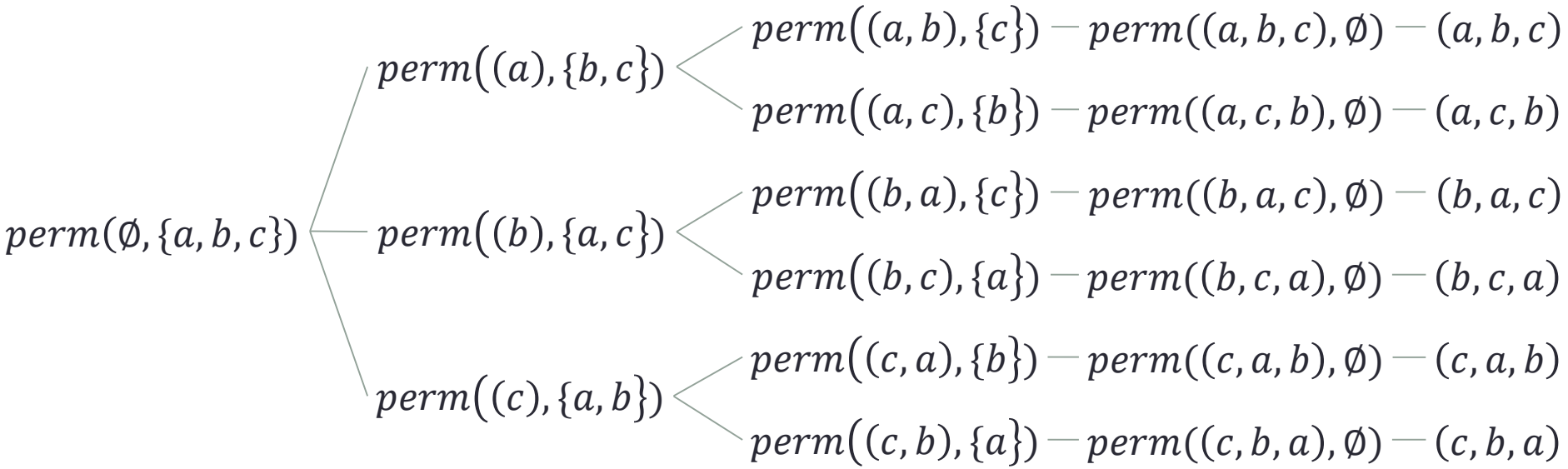
$$f(n) = \begin{cases} n & if\ n \le 2 \\ f(n-1) + f(n-2) & otherwise \end{cases}$$



例子:產生所有排列(1/2)

產生$\{a_0, a_1, \ldots, a_{n-1}\}$所有排列的遞迴公式

$$perm(P, \{a_0, a_1, \ldots, a_{n-1}\}) = \begin{cases} P & if\ n = 0 \\ perm((P, a_0), \{a_1, \ldots, a_{n-1}\}) & \\ perm((P, a_1), \{a_0, a_2 \ldots, a_{n-1}\}) & otherwise \\ \ldots & \\ perm((P, a_{n-1}), \{a_0, \ldots, a_{n-2}\}) & \end{cases}$$



例子:產生所有排列(2/2)

- 產生以字母順序的n字元所有排列

```
#include<stdio.h>
#define MAX_SIZE 20
void permutation(char data[],int idx, int size)
{
   int i;
   if (idx == size-1) {
      for(i = 0; i < size; ++i) {
         printf("%c",data[i]);
      }
      printf("\n");
   } else {
      char hold = data[idx];
      for(i = idx; i < size; ++i) {
         data[idx] = data[i];
         data[i]   = hold;
         permutation(data,idx+1,size);
         data[i]   = data[idx];
         data[idx] = hold;
      }
   }
   return;
}
```

```
int main()
{
   char data[MAX_SIZE];
   int n, i;
   printf("# of letters:");
   scanf("%d",&n);
   for(i = 0; i < n; ++i) {
      data[i] = 'a'+i;
   }

   permutation(data,0,n);
   return 0;
}
```

例子:產生組合(m裡面挑n個)

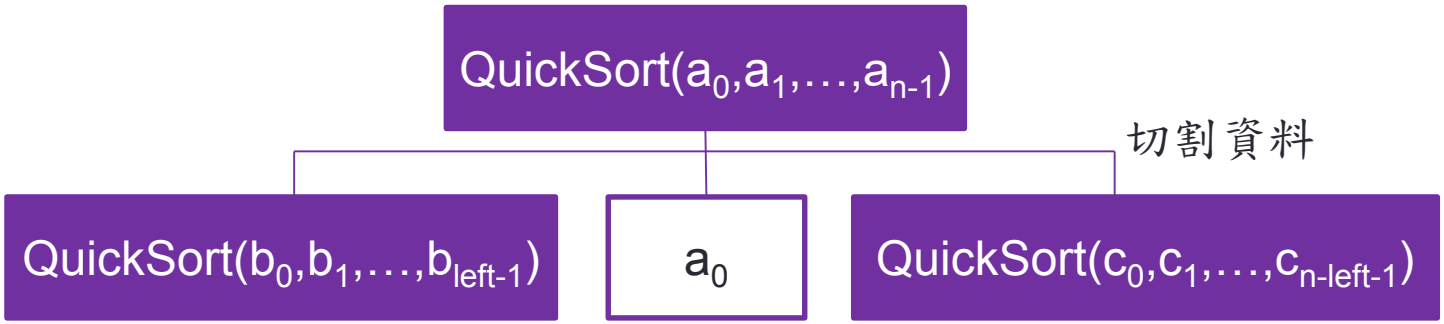
```
#include<stdio.h>
#define MAX_SIZE 20
void combination(char data[],char sel[],int idx, int sel_n, int n, int m)
{
    int i;
    if (n == sel_n) {
        for(i = 0; i < n; ++i) {
            printf("%c",sel[i]);
        }
        printf("\n");
    } else if (n > sel_n) {
        // data[idx] is selected.
        sel[sel_n] = data[idx];
        combination(data,sel,idx+1,sel_n+1,n,m);
        if (m-idx-1>=n-sel_n){
            // data[idx] is not selected.
            combination(data,sel,idx+1,sel_n,n,m);
        }
    }
    return;
}
```

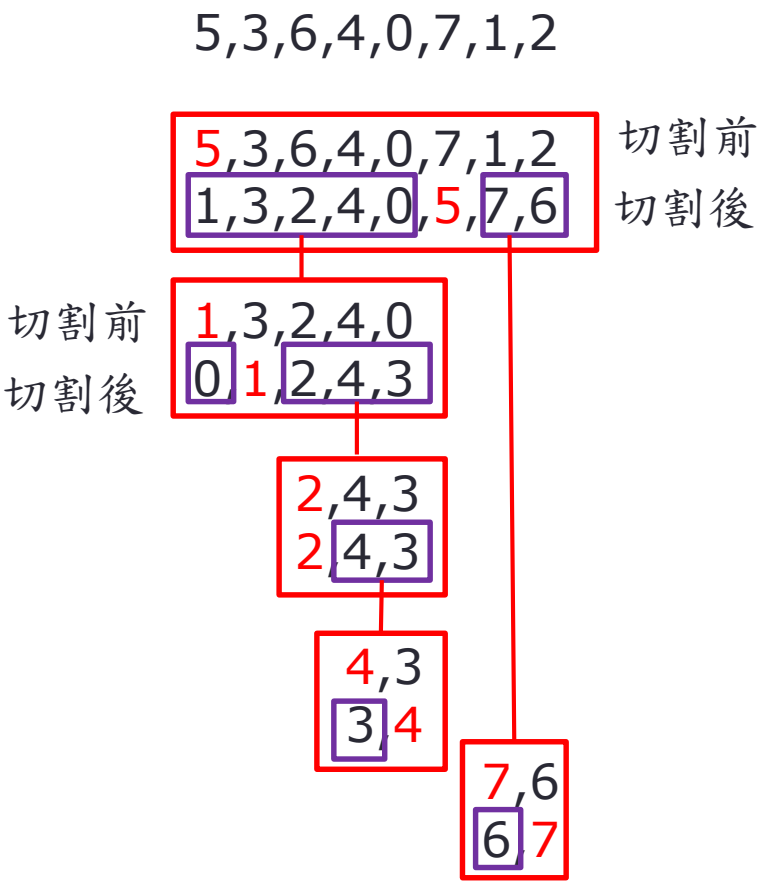
```
int main()
{
    char data[MAX_SIZE];
    char sel[MAX_SIZE];
    int m, n, i;
    printf("# of letters (m):");
    scanf("%d",&m);
    printf("n out of m, n:");
    scanf("%d",&n);
    for(i = 0; i < m; ++i) {
        data[i] = 'a'+i;
    }

    combination(data,sel,0,0,n,m);
    return 0;
}
```

練習:快速排序法

- 將資料$a_0,a_1,\ldots,a_{n-1}$由小到大排好。
- 想法:

1. 將資料$a_0,a_1,\ldots,a_{n-1}$安排成$b_0,b_1,\ldots,b_{left-1},a_0,c_0,c_1,\ldots,c_{n-left-1}$，其中
   $b_0,b_1,\ldots,b_{left-1}$皆不大於$a_0$
   $c_0,c_1,\ldots,c_{n-left-1}$皆不小於$a_0$
2. 然後套用相同方式分別對$b_0,b_1,\ldots,b_{left-1}$排序與$c_0,c_1,\ldots,c_{n-left-1}$排序




5,3,6,4,0,7,1,2
5,3,6,4,0,7,1,2
1,3,2,4,0,5,7,6
切割前
切割後
切割前
切割後
1,3,2,4,0
0,1,2,4,3
2,4,3
2,4,3
4,3
3,4
7,6
6,7
0,1,2,3,4,5,6,7

```
int partition(int a[],int n)
{
   int left = 0, right = n, hold, pivot=a[0],i;

   do {
     do {left++; } while (left < right && a[left] <= pivot);
     do {right--; } while(right >= left && a[right] >= pivot);
     if (left<right) { hold = a[left]; a[left] = a[right]; a[right]=hold; }
   } while(left<right);

   a[0] = a[right]; a[right]=pivot;

   return right;
}
void Qsort(int a[],int n)
{
     int left_size;

     if(n<=1) return;
    /*資料安排成b0,b1,…,bleft_size-1,a0,c0,c1,…,cn-left_size-1 */
    left_size = partition(a,n);

    /* */
     Qsort(a,left_size);
     Qsort(a+left_size+1,n-left_size-1);

     return;
}
```

數列表示方式

- a_n寫成n的函數: $a_n = f(n)$
  - 等差數列$a_n = b + \mathrm{c} \times n$
  - 等比數列$a_n = b \times c^n$
  - 階乘$a_n = 1 \times 2 \times \ldots \times n$
  - 費氏數列$a_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$
  - 質數數列:2,3,5,7,11,13,17,…**目前找不到規律的數列**

- a_n寫成與前面項的關係(遞迴關係式):
  - 等差數列$a_n = \begin{cases} b & \text{if } n = 0 \\ a_{n-1} + c & \text{elsewhere} \end{cases}$
  - 等比數列$a_n = \begin{cases} b & \text{if } n = 0 \\ c \times a_{n-1} & \text{elsewhere} \end{cases}$
  - 階乘$a_n = \begin{cases} 1 & \text{if } n = 1 \\ n \times a_{n-1} & \text{elsewhere} \end{cases}$
  - 費氏數列$a_n = \begin{cases} n & \text{if } n \leq 2 \\ a_{n-1} + a_{n-2} & \text{elsewhere} \end{cases}$

遞迴 vs. 疊代 (1/3)

- 都可以表示反覆的控制結構
  - Iteration: 迴圈
  - Recursion: 反覆的函式呼叫
- 結束方式
  - Iteration: 迴圈條件不成立
  - Recursion: 邊界條件成立
- 使用時機
  - 效能(iteration)
  - 可讀性及維護 (recursion)

```
int sum_input(int n) {
    int k,s=0,x;
    for(k = 0; k < n; ++k) {
            printf("input:");
            scanf("%d",&x);
            s +=x;
    }
    return s;
}
```

```
int sum_input(int n) {
    int x;
    if(n==0)  return 0;
    printf("input:");
    scanf("%d",&x);
    return x+sum_input(n-1);
}
```

遞迴 vs. 疊代 (2/3)

如何完成下面運算?

sum = a[0]+a[1]+…+a[n-1]

可以這麼算

sum = a[0]+a[1]+…+a[n-1]

```
for(sum = 0,i=0; i <n; i++) sum +=a[i];
```

也可以這麼算

sum = a[0]+a[1]+…+a[n-1]

```
for(sum = 0,i=n-1; i >=0; i--) sum +=a[i];
```

還可以這麼算

sum = a[0]+a[1]+…+a[n/2-1]+a[n/2+1]+…+a[n-1]

使用遞迴函式比較
容易實現這種計算
方式。

Base case

```
int sum(int a[],int n)
{
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return a[0];
    } else {
        int h = n/2;
        return sum(a,h)+sum(a+h,n-h);
    }
}
```

結合子問題的解成為原本問題的解

往Base case方向拆解成較小問題

Tail Recursion

- 下面這種遞迴我們稱它為Tail Recursion。有別於一般遞迴，它可以輕易地用迴圈改寫。

```
int factorial(int n)
{
    if (n==0) return 1;
    return n*factorial(n-1);
}
```

通常稍微改寫一下，tail recursion函式就會呈現當遞迴到邊界條件時，答案也同時算出來了!

```
int factorial(int n,int ans)
{
    if (n==0) return ans;// n!
    return factorial(n-1,n * ans);
}
x=factorial(n,1); // calculate n!
```

Tail recursion特徵

很容易用迴圈改寫

```
int factorial(int n)
{   int u,ans=1;
    if (n==0) return 1;
    for(u=2; u <=n; ++u) {
      ans*=u;
    }
    return ans;
}
```

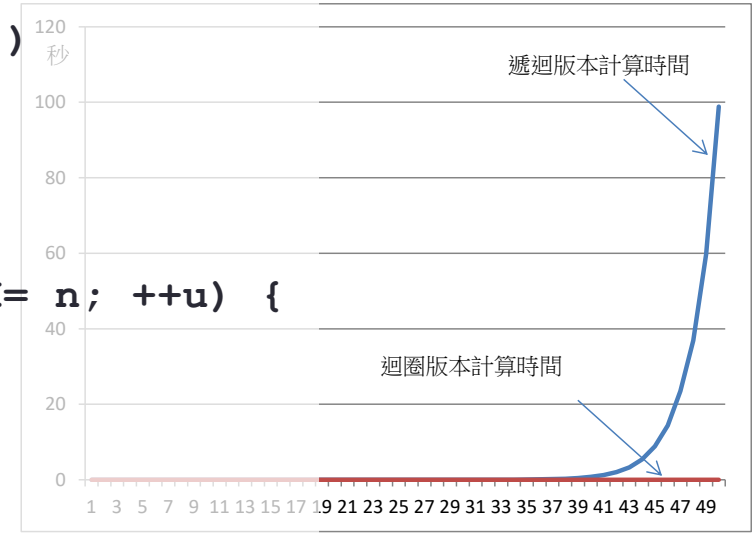
再看費氏級數

- 遞迴版本

```
long long unsigned fibonacci( long n )
{
   if (n == 0 || n == 1)  // base case
        return n;
   else
        return fibonacci(n - 1) + fibonacci( n - 2 );
}
```

- 改寫

```
long long unsigned fibonacci( long n )
{
   long long unsigned f[n+1]; //C99
   long u;
   if (n <= 1) return n;
   for(f[0] = 0, f[1] = 1, u = 2; u <= n; ++u) {
        f[u] = f[u-1]+f[u-2];
   }
   return f[n];
}
```



第一次改寫

訣竅:用陣列記住已經算過的答案，避免重算。

```
long long unsigned fibonacci( long n )
{
    long long unsigned f[n+1]; //C99
    long u;
    if (n <= 1) return n;
    for(f[0] = 0, f[1] = 1, u = 2; u <= n; ++u) {
        f[u] = f[u-1]+f[u-2];
    }
    return f[n];
}
```

f[6]在計算f[7]與f[8]時都會用到，所以記住它就不需要重算。

f[0]	f[1]	f[2]	f[3]	f[4]	f[5]	f[6]	f[7]	f[8]
0	1	1	2	3	5	8	13	21

第二次改寫

訣竅:捨棄陣列。因為只會參考前兩項，其他都不再需要。因此只要保留最近兩項，以便計算下一項就足夠了。

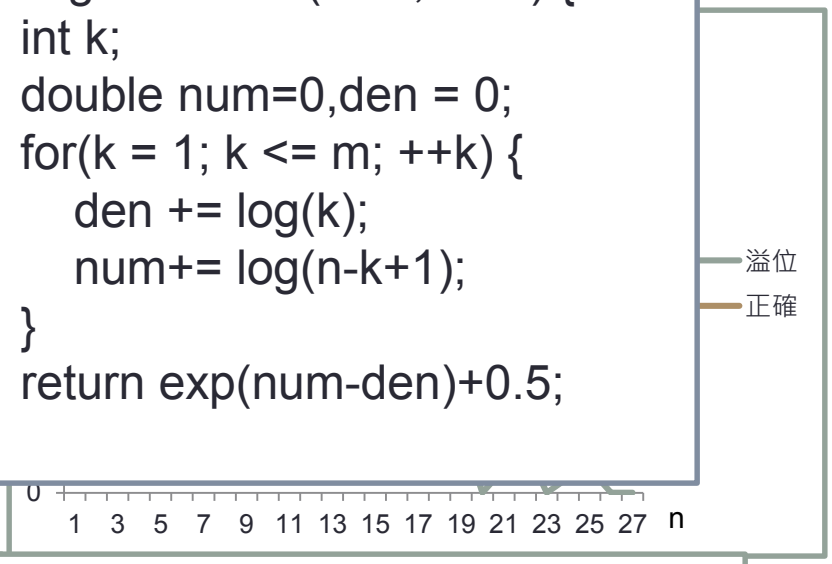
```
long long unsigned fibonacci( long n )
{
    long u;
    long long unsigned f0,f1,f;
    if (n <= 1) return n;
    for(f0 = 0, f1 = 1, u = 2; u <= n; ++u) {
        f = f1+f0;
        f0= f1;
        f1= f;
    }
    return f;
}
```

範例: 計算組合數(1/3)

$$C(n,m)=\frac{n(n-1)\times\ldots\times(n-m+1)}{m\times(m-1)\times\ldots\times 1}$$

```
unsigned comb(int n,int m) {
   int k;
   unsigned num=1,den = 1;
   for(k = 1; k <= m; ++k) {
       den *= k;
       num*=(n-k+1);
   }
   return num/den;
}
```

```
#include<math.h>
unsigned comb(int n,int m) {
   int k;
   double num=0,den = 0;
   for(k = 1; k <= m; ++k) {
       den += log(k);
       num+= log(n-k+1);
   }
   return exp(num-den)+0.5;
}
```



當m,n有點大，計算就會溢位(overflow)。

若不需要算的很準，那麼可以這樣寫得到差不多的答案。

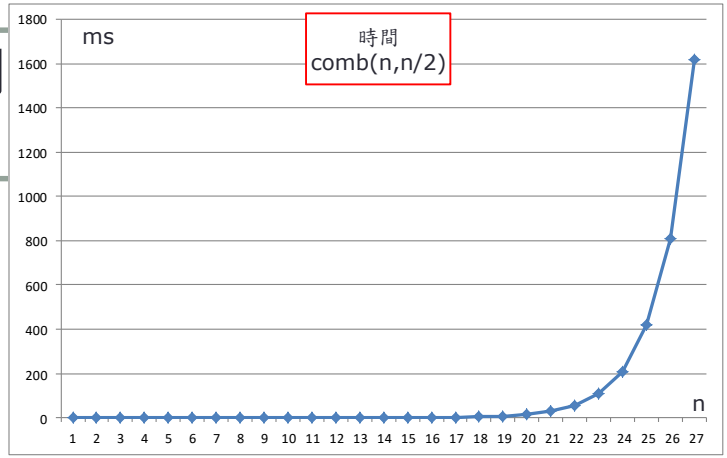
範例: 計算組合數(2/3)

$$C(n,m)=\begin{cases}1 & \text{if } m=0 \text{ or } m=n\\ C(n-1,m)+C(n-1,m-1) & \text{if } 1\le m<n\end{cases}$$

```
unsigned comb(int n,int m) {
    if (m==n || m ==0) return 1;
    return comb(n-1,m)+comb(n-1,m-1);
}
```

採用遞迴公式，計算只會用法。可以應付較大數字。

缺點: 很慢!!!



範例: 計算組合數(3/3)

$$C(n,m)=\begin{cases}1 & \text{if } m=0 \text{ or } m=n\\ C(n-1,m)+C(n-1,m-1) & \text{if } 1\le m<n\end{cases}$$

Pascal's triangle

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1

```
#define MAXN 51
unsigned table[MAXN][MAXN];
int min(int x,int y)
{
    return (x < y) ? x : y;
}
int comb(int n,int m)
{
   int u,v;
   for(u = 1; u < MAXN; ++u) {
        table[u][u] = table[u][0] = 1;
   }
   for(u = 1; u <= n; ++u) {
     for(v = 1; v < min(u,m+1); v++) {
          table[u][v] = table[u-1][v]+table[u-1][v-1];
     }
   }
   return table[n][m];
}
```

避免直接使用遞迴公式產生的重複計算問題。

n\m	0	1	2	3	4	5
0	1					
1	1	1				
2	1		1			
3	1					
4	1				1	
5	1					1

comb(n-1,m-1)

comb(n-1,m)

comb(n,m)=comb(n-1,m-1)+comb(n-1,m)

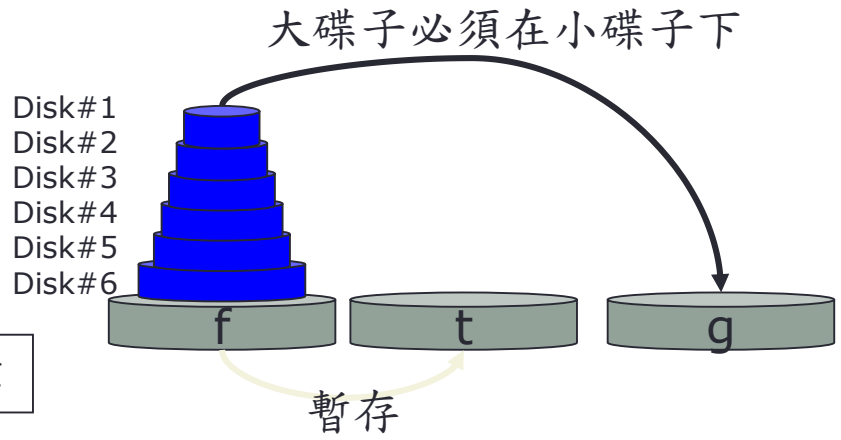
範例: 計算comb(300,150)

comb(300,150)=
9375970277282745279319375443906408487923265570008135892047235271297517002183959167586142 4

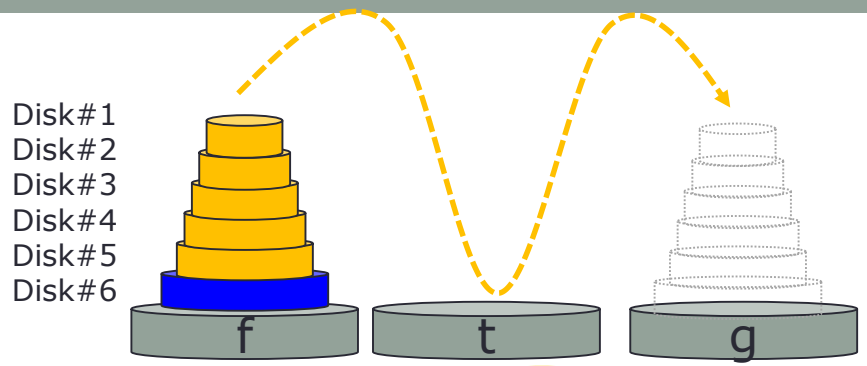
只需數十 milliseconds!! 當然你要寫一個"長長" 整數，來處理這麼大的數字。

例子: 河內塔(Hanoi tower)

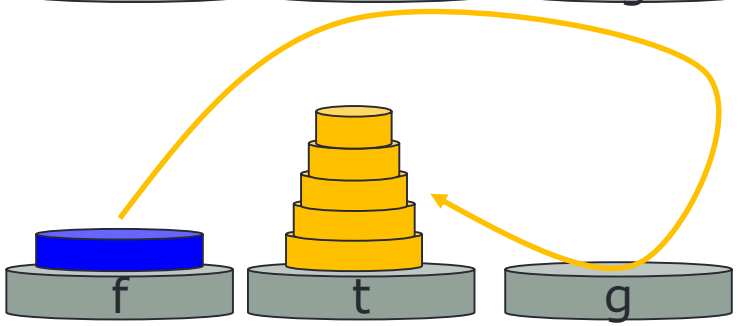
```
void hanoi(int disks,int f,int t,int g)
/* move disks from peg  f to peg g using t as a temporary peg */
{
    if(disks==1) {
        printf("move from %d to %d\n",f,g);
        return;
    }
    hanoi(disks-1,f,g,t);
    printf("move from %d to %d\n",f,g);
    hanoi(disks-1,t,f,g);
}
```



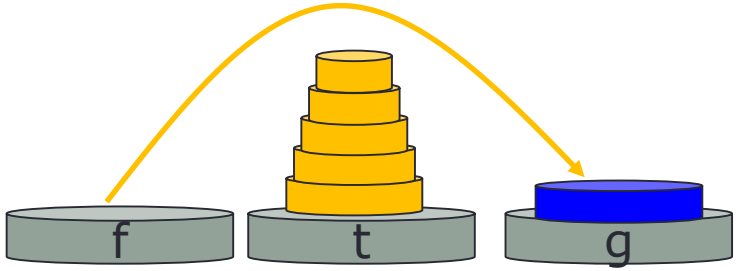
亦有迴圈版本:hint 觀察碟子移動是否有規律



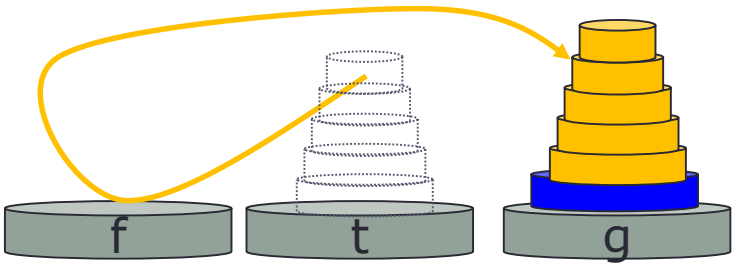
hanoi(disks,f,t,g);



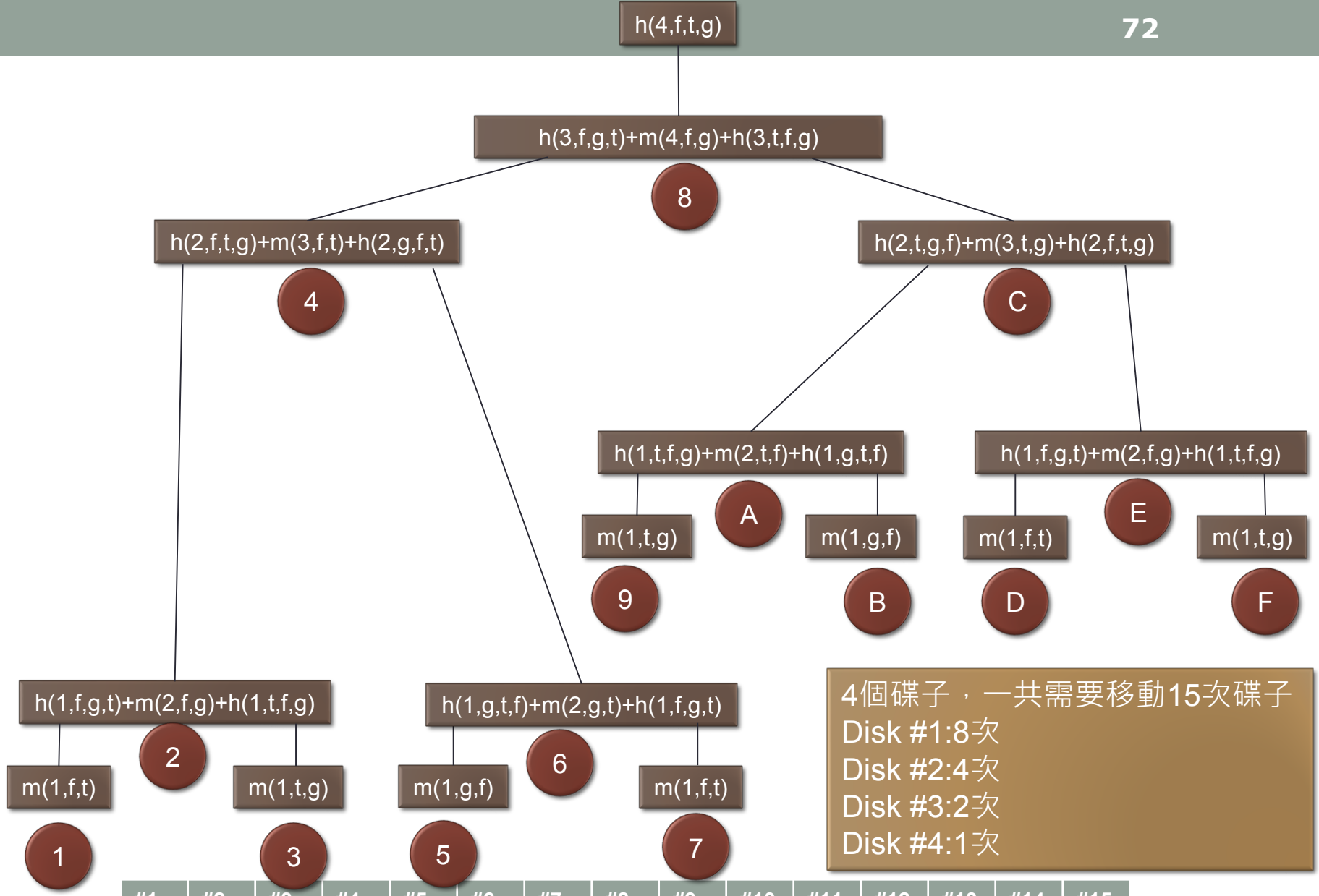
hanoi(disks-1,f,g,t);



move from f to g



hanoi(disks-1,t,f,g);



4個碟子，一共需要移動15次碟子
Disk #1:8次
Disk #2:4次
Disk #3:2次
Disk #4:1次

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
D1	D2	D1	D3	D1	D2	D1	D4	D1	D2	D1	D3	D1	D2	D1
F	F	T	F	G	G	F	F	T	T	G	T	F	F	T
T	G	G	T	F	T	T	G	G	F	F	G	T	G	G

河內塔碟子移動次數的遞迴公式

移動次數的遞迴公式：

$$f(n) = \begin{cases} 1 & if\ n = 1 \\ 2f(n-1)+1 & otherwise \end{cases}$$

```
void hanoi(int disks,int f,int t,int g)
{
    if(disks==1) {
        printf("move from %d to %d\n",f,g);
        return;
    }
    hanoi(disks-1,f,g,t);
    printf("move from %d to %d\n",f,g);
    hanoi(disks-1,t,f,g);
}
```

$f(n)$ 有公式解

$$\begin{aligned} f(n) &= 2(2f(n-2)+1)+1 \\ &= 2^2 f(n-2)+2+1 \\ &= 2^2(2f(n-3)+1)+2+1 \\ &= 2^3 f(n-3)+2^2+2+1 \\ &= 2^{n-1} f\big(n-(n-1)\big)+2^{n-2}+\cdots+2+1 \\ &= 2^n-1 \end{aligned}$$

河內塔問題疊代版本程式

```
#include<stdio.h>
int main()
{
    int total_disk,i, j;
    unsigned total_move;
    int disk_id;
    char peg[3];

    printf("total disk:");
    scanf("%d",&total_disk);

    /* calculate total number of moves */
    total_move = (1u<<total_disk)-1;

    for(i = 1; i <= total_move; ++i) {
        unsigned moves,step;
        for(disk_id=1;disk_id<=total_disk;++disk_id){
            if (i & (1u<<(disk_id-1))) {
                break;
            }
        }
        step = 1u<<disk_id;
        moves= i/step % 3;
        peg[0] = 'F';
        peg[1] = (disk_id %2 == total_disk %2)? 'G' : 'T';
        peg[2] = (disk_id %2 == total_disk %2)? 'T' : 'G';
        printf("step:%3d,disk:%2d,from %c to %c\n",i,disk_id,peg[moves],peg[(moves+1)%3]);
    }
    return 0;
}
```

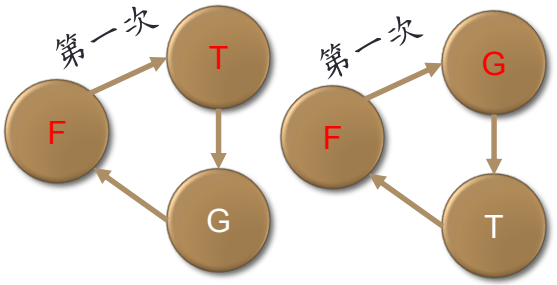
間隔2　間隔4　間隔8

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
D1	D2	D1	D3	D1	D2	D1	D4	D1	D2	D1	D3	D1	D2	D1
F	F	T	F	G	G	F	F	T	T	G	T	F	F	T
T	G	G	T	F	T	T	G	G	F	F	G	T	G	G

碟子第一次移動方式

次數	移動碟子
#1:	0001->D1
#2:	0010->D2
#3:	0011->D1
#4:	0100->D3
#5:	0101->D1
#6:	0110->D2
#7:	0111->D1
#8:	1000->D4
#9:	1001->D1
#10:	1010->D2
#11:	1011->D1
#12:	1100->D3
#13:	1101->D1
#14:	1110->D2
#15:	1111->D1

碟子在栓間移動方式:



Disk#1
Disk#3
與Disk#4
不同奇偶數

Disk#2
與Disk#4
同樣是偶數