

計算機程式設計

C語言 Basic

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

基本C程式

- 每一個C程式一定有一個main function(函式)
 - C程式執行的進入點
 - C程式可由許多function組成，但只能有一個main function
- function 語法

```
回傳值資料型態 函式名稱(參數資料型態 參數名稱, ...) {  
    return 回傳值;  
}
```

- 回傳值資料型態: int(整數), void(無), double(浮點小數), ...
- 參數資料型態: int(整數), void(無), double(浮點小數), ...
- 函式名稱: 除了main之外，其他自訂
- 回傳值: 回傳一個計算後得到的數值，給呼叫此函式的程式
- return是保留字，後續會進一步介紹函式

基本C程式

□ main function

```
#include <stdio.h>
int main(void) {
    int i;          // 宣告變數 i，這一行是註解
    /* 註解有兩種，這是多行註解
     */
    i=5;
    printf("%d\n", i);
    return 0;
}
```

- {} 大括號裡面稱為 程式區塊 (Block)
- 程式的指令/敘述，稱為 instruction, statement
- 每一行指令，後面一定要以分號(;)結束
- 程式不同層次，最好能空四個space以增加可讀性

基本C程式

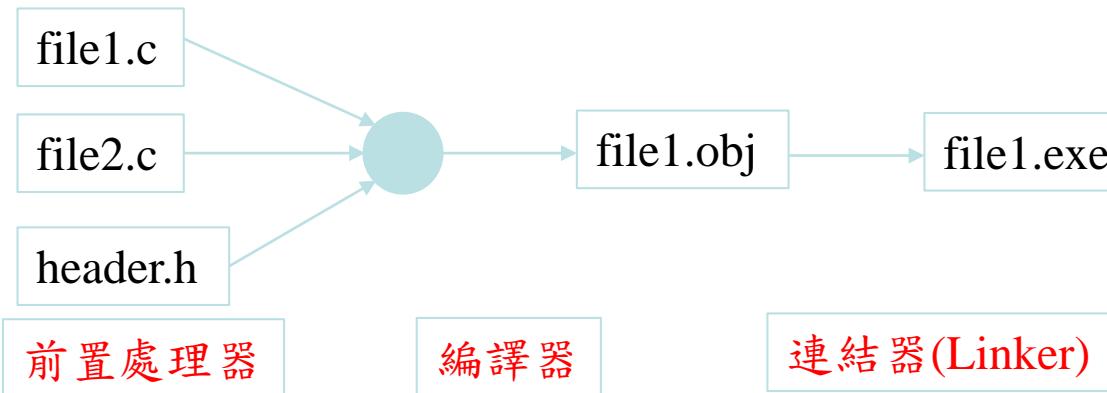
□ main function

- stdio.h是標準輸出輸入(**standard input out**)函式庫，.h是附屬檔名，h是標頭檔(header file)
- int i; 是宣告變數(variable) i 的資料型態是整數，i 是程式師命名
- i = 5; 是將 5 這個值指定(= Assigment)給 i。
- printf是stdio.h函式庫提供的一個函式功能，輸出值到螢幕(標準輸出)
- printf()，括號內是參數，有兩個，一個是格式化旗標，第二個是要輸出變數資料
- main後面括號，DOS傳遞參數，void表示不傳參數
- return 0 是將0傳回給DOS。

基本C程式

□ 前置處理器(preprocessor)

- # 後程式由前置處理器處理，之後再由編譯器(compiler)處理
- #define，定義一個符號的值，程式之後使用這個符號代表此值
 - #define start 0，定義start代表0，前置處理器先將程式中start全換成0
- #include<內建函式庫檔名>表示載入一個內建函式庫 (library)
- #include"自訂函式庫檔名"表示載入一個自訂函式庫 (library)
- #後面不需要分號;結束



變數名稱的使用

- 變數(Variable)，在程式中值是可以改變的，可以指定各種不同的值。
 - 常數(Constant)，在程式中一經指定，值是不可以改變的。
 - const int i = 9; #define i 9
- C語言變數(Variable)名稱使用，須以下列三種字元做開頭：
 - 大寫字母。
 - 小寫字母。
 - 底線(_)
- 變數名稱由下列四種字元所構成：
 - 大寫字母
 - 小寫字母
 - 底線(_)
 - 阿拉伯數字0~9

變數名稱的使用

- 大寫和小寫字母代表不同變數，下列代表三個不同變數。
 - sum
 - Sum
 - SUM
- 系統保留字（又稱關鍵字），在C編譯程式中代表特別意義，不可使用這些字為變數名稱。下列是ANSI C語言保留字。
 - auto break case char continue default
 - do double else enum extern float
 - for goto if int long register
 - return short sizeof static struct switch
 - typedef union unsigned void while

Exercise

□ 以下那些變數是合法，哪些是非法

- SUM
- Hung
- sum_1
- Fg
- X5
- y61
- sum,1
- 3y
- x\$2

變數名稱的使用

□ 合法的變數名稱：

- SUM, Hung, sum_1, Fg, X5, y61

□ 不合法的變數名稱：

- sum,1 ← 變數名稱不可有"，"符號
- 3y ← 變數名稱不可由阿拉伯數字開頭
- x\$2 ← 變數名稱不可含有"\$"符號

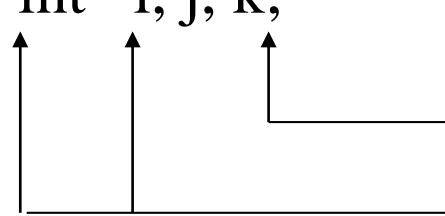
□ 大寫和小寫字母代表不同變數，下列代表三個不同變數。

- sum
- Sum
- SUM

變數宣告

- 將i,j,k三個數宣告為整數，下列宣告均是合法的。

```
int i, j, k;
```



整行宣告以";"為結束

各變數間用",,"隔開

```
int i;  
int j;  
int k;
```

- 由於i和j間用",,"隔開，所以上述是合法宣告方式。

- 宣告完由";"結束，也可在宣告變數的同時，設定變數的值。
 - 將i宣告成整數，並將其設定成7。
 - int i = 7 ;
- 把所有的變數集中在程式的開頭宣告，有利於程式的維護。

Exercise

□ 撰寫一個main function

- 設定矩形的高與寬兩個整數變數值分別為15, 12，
- 計算面積和周長並輸出

基本資料型態

□ C語言的基本資料型態有：

- void：無型別
- int：整數
- char：字元
- float：單精度浮點數
- double：雙倍精度浮點數

基本資料型態

□ void：無型別

- 非屬任何型別，可以轉成任何型別
- 放在函式宣告參數中，表示函數不需要傳入任何參數
- 放在函式宣告回傳資料型態中，表示函數不需要回傳任何值

```
回傳值資料型態 函式名稱(參數資料型態 參數名稱, ...){  
    return 回傳值;  
}
```

```
void 函式名稱(void) {  
    return;      //可省略此行  
}
```

資料型態

關鍵字	bit	scope	printf chars
char	8	-128..127 (或0..255與體系相關)	%c
unsigned char	8	0..255	
signed char	8	-128..127	
int	32(win32/unix)	-2147483648..2147483647	%i, %d
unsigned int/unsigned	32(win32/unix)	0..4294967295	%u
signed int	32(win32/unix)	-2147483648..2147483647	%i, %d
short int	16	-32768..32767	%hi
unsigned short	16	0..65535	%hu
signed short	16	-32768..32767	
long int	32	-2147483648..2147483647	%li, %ld
unsigned long	32	0..4294967295	%lu
signed long	32	-2147483648..2147483647	
long long	64	-9223372036854775808.. 9223372036854775807	%lli
unsigned long long	64	0..18446744073709551615	%llu
float	32	$3.4 \times 10^{-38} \dots 3.4 \times 10^{+38}$ (7 sf)	%f, %e, %g
double	64	$1.7 \times 10^{-308} \dots 1.7 \times 10^{+308}$ (15 sf)	%f, %e, %g
long double	64 或以上	編譯器相關	%Lf, %Le, %Lg

整數常數

- 以0（零）為開頭的整數視為8進位數字。
- 試說明013和026的10進位值。
 - 013 等於 11
 - 026 等於 22
- 以0x開頭的整數，視為16進位整數。
- 試說明0x28和0x3A的10進位值。
 - 0x28等於40
 - 0x3A等於58
- 在數字後加l或L，表示長整數。

常數 Constant

- 常數 (Constant) 是 "不會在程式執行過程中改變的數"。
- 定義常數的好處
 - 使常數的意義一目瞭然。
 - 使用常數易於維修程式。

```
#include <stdio.h>

int main(void) {
    double radiusA, areaA, radiusB, areaB;
    radiusA = 3.0;
    radiusB = 5.0;
    areaA = 3.14 * radiusA * radiusA;
    areaB = 3.14 * radiusB * radiusB;
    printf("%f, %f\n", areaA, areaB);
    return 0;
}
```

```
#include <stdio.h>
#define PI 3.14159
int main(void) {
    double radiusA, areaA, radiusB, areaB;
    radiusA = 3.0;
    radiusB = 5.0;
    areaA = PI * radiusA * radiusA;
    areaB = PI * radiusB * radiusB;
    printf("%f, %f\n", areaA, areaB);
    return 0;
}
```

字元char

- 一個char宣告的變數，佔記憶體空間是8位元。 $2^8 = 256$
 - 代表256個不同的值。
 - ASCII碼值，含大小寫字母、數字、標點符號及其它特殊符號。
- 宣告字元變數single_char
 - char single_char ;
- 宣告字元變數single_char，將其值設定為 'a'。
 - char single_char = 'a' ;
 - 單引號間的字元，如： 'a'， ';'， '3'。
 - '\0' 值是0， '0' 值是48
- 字元和一般整數混合進行加法和減法運算。
 - ch ='a' ;
 - ch = 'a' + 1 ;
 - 因 'a' 值是97，執行加法後ch值是98，所以最後ch值是 'b' 。

字元char

- 無法列印字元，如：‘\0’，雖在單引號中有\和0，但合併只算一個字元，這些字元為逸出（escape）字元。

整數值	字元表示方式	字元名稱
0	'\0'	空格 (null space)
7	'\a'	響鈴 (bell ring)
8	'\b'	退格 (backspace)
9	'\t'	標識 (tab)
10	'\n'	新列 (newline)
12	'\f'	送表 (form feed)
13	'\r'	回轉 (carriage return)
34	'\"'	雙引號 (double quote)
39	'\''	單引號 (single quote)
92	'\\'	倒斜線 (back slash)

字元常數

- 單引號間的字元，稱字元常數。例如：‘a’，‘;’，‘3’。
 - '0'的ASCII值是0
 - '0'的ASCII值是48
- 有時也將字元常數和一般整數混合進行加法和減法運算。
 - 假設有一字元變數 $ch = 'a'$ ；
 - 假設有一指令是 $ch = 'a' + 1$ ；
 - 因 'a' 值是97，執行加法後 ch 值是98，最後 ch 值是 'b' 。

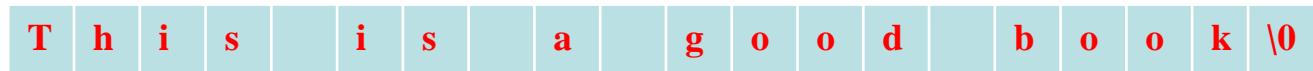
字串常數

□ 雙引號間任意個數的字元符號。例如：

- "This is a good book" ← 一個字串
- "" ← 一個空字串

□ 字串

- 是一個陣列，
- 每個元素存一個字元，編譯程式自動把'\0'放入陣列最後，代表字串結束。



浮點數

- 宣告一個浮點變數average，則其宣告如下：
 - float average；
- double雙倍精度浮點數，容量是浮點數的一倍。而long double則稱長雙倍浮點數，長度是80位元。
- 宣告tmp為雙倍精度浮點數，tmp1為長雙倍精度浮點數。
 - double tmp；
 - long double tmp1；
- 下表是三種浮點數/實數有關資料表

浮點數宣告型態	長度	值的範圍
float	32	$3.4 \times 10^{-38} \sim 3.4 \times 10^{38}$
double	64	$1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$
long double	80	$3.4 \times 10^{-4932} \sim 1.1 \times 10^{4932}$

字串資料型態

- 字串指在兩個雙引號中任意字元。例如：
 - " ", "hello , How are you ? "
- 若雙引號中沒有字元，稱空字串。
- 字串存入記憶體會將 ‘\0’（空字元）加在字串最後，表示字串結束。
- 假設有一字串是"hello !"，實際記憶體儲存此字串如下：

h	e			o	!	\0
---	---	--	--	---	---	----

- 雙引號非字串一部份，若字串是He say, "Hello ! ，"，則此字串表示法： "He say , \ "Hello ! ， \ ""

基本算術運算

- $\text{sum} = a + b;$
- $s = a - b;$
- $s = a * b;$
- $s=a/b;$
- $s = a\%b;$
- $s=-a-b;$
- $s = a *(b + c);$

Homework I

- 寫一程式計算一元二次方程式的兩個實根

型別轉換

□ 不同型態間運算，如整數轉換成浮點來運算。

○ 會轉換成最長空間後再運算

○ $s = a + b;$

➤ 整數 $a = 3$ ，浮點數 $b = 2.5$ ，運算後 $s = 5.5$

○ $s = a + b;$

➤ $a = 3$ ， b 是浮點數 $b = 2.5$ ， $s=?$

○ $a = 3$ ， $b = 2$ ， $s=a/b$, $s=?$

➤ $s=(float) a/(float)b;$ $s=?$

○ a 和 b 是浮點數， $a = 4.6$ ， $b = 2.1$ ， $s = ?$

➤ $i='a' - 'A';$

➤ i 是整數，先將 ‘ a ’ 轉成ASCII碼97，再將 ‘ A ’ 轉成ASCII碼65，運算完後 i 的值是32。

遞增和遞減運算式

□ "++" 將某個運算元加1，"--" 會主動將某個運算元減1。

○ $i++$ ；執行前 $i = 2$ ，則執行後 $i = 3$

○ $i--$ ；執行前 $i = 2$ ，則執行後 $i = 1$

□ $i++$ ，後置運算

○ 先取出 i 的值， i 再加。

□ $++i$ ，稱前置運算。

○ 先加1，再取出 i 的值

○ $a = ++i + 3;$

➤ $a = 3, i = 5,$

➤ 先做 i 加1， i 變為6，再進行加算， a 值是9。

○ $a = 3 + i++;$

➤ $a = 3, i = 5,$

➤ 先執行 $3 + i$ ， a 值是8， i 本身再加1，值是6。

遞增和遞減運算式

□ ++ 和 -- 前置運算子和後置運算子應用

```
#include <stdio.h>
int main() {
    int x,y,z;
    x = y = z = 0;
    /* testing the increment by 1 */
    x = ++y + ++z;
    printf("\1: line 14 result ... %d %d %d\n",x,y,z);
    x = y++ + z++;
    printf("\2: line 16 result ... %d %d %d\n",x,y,z);
    /* testing the decrement by 1 */
    x = y = z = 0;
    x = --y + --z;
    printf("\1: line 20 result ... %d %d %d\n",x,y,z);
    x = y-- + z--;
    printf("\2: line 22 result ... %d %d %d\n",x,y,z);
}
```

輸入和輸出函數

□ 加、減、乘、除及求餘數（ $+=$ ， $-=$ ， $*=$ ， $/=$ ， $\% =$ ）等特殊運算子的程式應用

○ $a *= c$;

➤ $a = 3$, $c = 2$, 則執行後 $c = 2$, $a = 6$

○ $a += c * d$;

➤ $a = a + (c * d)$;

○ $a *= c + d$;

➤ $a = 2$, $c = 3$, $d = 4$,

○ $a = a * (c + d)$

➤ $c + d$ 等於 7 , $7 * 2 = 14$,

➤ 可得 $a = 14$

```
#include <stdio.h>
int main() {
    int a,b,c,d,e;
    a = b = c = d = e = 0;
    a += 2;
    printf("a is %d\n",a);
    b -= 2;
    printf("b is %d\n",b);
    c *= c = 2;
    printf("c is %d\n",c);
    d %= d = 3;
    printf("d is %d\n",d);
    e /= e = 4;
    printf("e is %d\n",e);
}
```

位移運算子(Shift Operator)

- 位移運算，把位元(bit)向左移($<<$)或向右移($>>$)幾個位置。
 - 向左移n個位元，相當於乘2的n次方；
 - 向右移n個位元，相當於除以2的n次方。

```
#include<stdio.h>
int main(){
    int a = 5, b = 13;
    a = a << 2;
    b = b >> 1;
    printf(" 5 << 2 = %i \n", a);
    printf("13 >> 1 = %i \n", b);
    return 0;
}
```

5 << 2 = 20
13 >> 1 = 6

位元運算子(Bit Operator)

□ 位元邏輯運算

```
#include<stdio.h>
int main(){
    int A = 3, B = 5;
    printf("A & B = %i \n", A & B);
    printf("A | B = %i \n", A | B);
    printf("A ^ B = %i \n", A ^ B);
    printf(" ~A = %i \n", ~A );
    return 0;
}
```

A & B = 1
A | B = 7
A ^ B = 6
~A = -4

運算子	使用方式	功能敘述
&	i & j	i AND j
	i j	i OR j
^	i ^ j	i XOR j
~	~ i	NOT i

sizeof

- 算出型別資料佔用記憶體位置數量（以byte為單位）
 - sizeof (某個資料型態)
 - n = sizeof (char) ;
 - char字元是一個位元組 (byte) ，執行完後n值是1。

```
#include <stdio.h>
int main() {
    printf("The size of char  is %d\n",sizeof(char));
    printf("The size of int   is %d\n",sizeof(int));
    printf("The size of float is %d\n",sizeof(float));
    printf("The size of double is %d\n",sizeof(double));
}
```

The size of char is 1
The size of int is 4
The size of float is 4
The size of double is 8

基本輸入輸出

□ 標準輸入與輸出函數基本定義：

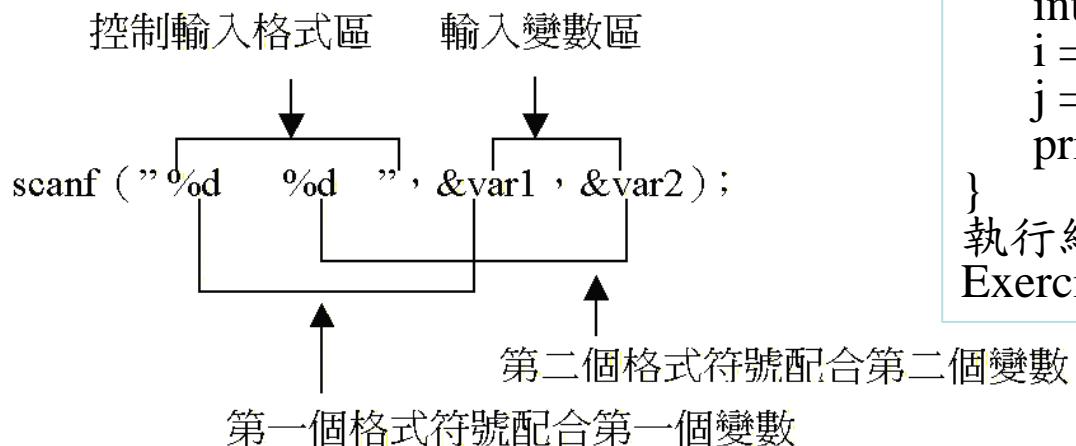
- #include <stdio.h>
- printf () : 這是一個最常用的輸出函數
- scanf () : 這是一個最常用的輸入函數
- putchar (c) : 列印字元的輸出函數
- getche () : 讀取字元的輸入函數
- getchar () : 讀取字元的輸入函數
- getch () : 讀取字元的輸入函數

```
//依不同格式列印字串，  
//Introduction to C language字串兩次，依不同格列印。  
#include <stdio.h>  
int main() {  
    printf("Introduction\n to C language.\n");  
    printf("Introduction to\n C language.\n");  
}
```

執行結果
Introduction
To C language.
C language.

%d十進位整數的列印

□ 格式化控制輸出的結果



```
#include <stdio.h>
int main() {
    int i,j;
    i = 3;
    j = 4;
    printf("exercise ch%d_%d.c \n",i,j);
}
```

執行結果
Exercie ch3-4.c

- 第一個格式符號配合第一個欲列印的變數。
- 控制輸入格式區的格式符號間，可有許多空格，或沒有空格。
- 輸入變數區，各變數間要有逗點隔開。
- 控制輸入格式區需用雙引號""包夾起來。
- 控制輸入格式區和輸入變數區間需用逗號隔開。

格式化控制輸出

□ %5d

```
#include <stdio.h>
int main() {
    int i;
    i = 356;
    printf("%d\n",i);
    printf("%2d\n",i);
    printf("%5d\n",i);
    printf("%-5d\n",i);
}
//控制字元 '\n'
```

執行結果
/356/
/356/
/ 356/
/356 /

% f 浮點數的列印

```
#include <stdio.h>
int main() {
    float i;
    i = 123.56;
    printf("%f\n",i);
    printf("%3.2f\n",i);
    printf("%8.2f\n",i);
    printf("%-8.2f\n",i);
}
```

執行結果
/123.559998/
/123.56/
/ 123.56/
/123.56 /

% c 字元的列印

- 格式化出某一字元變數。

```
#include <stdio.h>
int main() {08
    char i;
    i = 'a';
    printf("%c\n",i);
    printf("%3c\n",i);
    printf("%-3c\n",i);
}
```

執行結果

```
/a/
/ a/
/a /
```

其他格式化資料列印

□ printf() 提供格式化列印方式：

- % e : 以e記號（科學符號）表示法，輸出浮點數。
- % u : 不帶符號的10進位整數輸出。
- % X : 16進位整數輸出。
- % o : 8進位整數輸出。
- % s : 列印字串。
- %lld : long long
- %llu : unsigned long long

```
format octal value output  
/12/  
/12  /  
format hexadecimal value output  
/a/  
/ a/  
format unsigned value output  
/10/  
/ 10/  
format scientific symbol value output  
/1.235600e+002/  
/1.236e+002/
```

```
#include <stdio.h>  
int main() {  
    int i = 10;  
    float j = 123.56;  
    printf("format octal value output\n");  
    printf("%o\n",i);  
    printf("%-8o\n",i);  
    printf("format hexadecimal value output\n");  
    printf("%x\n",i);  
    printf("%8x\n",i);  
    printf("format unsigned value output\n");  
    printf("%u\n",i);  
    printf("%8u\n",i);  
    printf("format scientific symbol value output\n");  
    printf("%e\n",j);  
    printf("%8.3e\n",j);  
}
```

scanf ()

□ 使用兩次scanf()的問題

- 執行結果錯誤，輸入一個值(例如 1)，按下 enter 就結束。
- scanf() 未處理換行符號，造成第二個 scanf("%c",&b) 吃到換行符號 enter 。

```
int a;  
char b;  
scanf( "%c", &a );  
scanf( "%c", &b );  
printf( "a = %c, b = %c\n", a, b );
```

1
a = 1, b =

□ 解決方法：

- 在第二個 scanf() 的 %c 之前加入空白字元，scanf(" %c", &b);
- 在第一個 scanf() 的 %d 之後使用 %*c 跳脫一個字元，
scanf("%d%*c", &a);
- 在兩個 scanf() 之間加入 getchar() 吸收換行

```
getchar();  
scanf( "%c", &b );
```

scanf ()

□ scanf 可自定欲接收的字元，包括空白等字元：

- `scanf("%[^\\n]",str);` // 接收除了 \n 以外的所有字元
- `printf("%s",str);` // 輸出完整的「hello world」

□ 如果輸入不是integer時，程式會跑個不停。

- 因 scanf 透過stdin 讀入資料，存放在memory一個queue中。
- 只要queue裡有資料，程式就須scanf 將裡面資料消化掉。
- 「%」符號是match，「%d」表示queue中資料須是integer才可。
- 若輸入char，會使queue中資料沒有辦法消化，程式將在回圈中不繼尋找一個可以消化的scanf。

```
int a=0;
while(a!=1){
    scanf("%d",&a);
}
```

scanf ()

□ 解決方法：清掉queue裡的資料

- 用fflush(stdin)，把 input queue清除，但此函式偶而有誤。可以使用getche把queue裡的值讀完。

```
while(a!=1){  
    scanf("%d",&a);  
    while(getche() != '\n');  
}
```

- 若輸入資料不符合scanf 時，getche可把queue裡的值讀完，
- 若輸入資料符合scanf，不會因多getche那一行程式就停在那邊不動，因enter剛好可留給它消化，如此程式就不會出問題。

scanf ()

- 輸入兩字元，顛倒順序輸出，輸入三個整數求總和，將它列印出，輸入兩個浮點數，將平均浮點數值印出

```
#include <stdio.h>
int main()
{
    int i,j,k,sum;
    char ch1,ch2;
    float x1,x2,ave;
    printf("Enter 2 characters ==>");
    scanf("%c%c",&ch1,&ch2);
    printf("Reverse of 2 char ==>");
    printf("%c%c\n",ch2,ch1);
    printf("Enter 3 integer ==>");
    scanf("%d %d %d",&i,&j,&k);
    sum = i + j + k;
    printf("The sum of input ==> %d\n",sum);
    printf("Enter 2 floating ==>");
    scanf("%f %f",&x1,&x2);
    ave = ( x1 + x2 ) / 2.0;
    printf("Average of input ==> %6.2f\n",ave);
}
```

Enter 2 characters ==>ab
Reverse of 2 char ==>ba
Enter 3 integer ==>3 1 5
The sum of input ==> 9
Enter 2 floating ==>1.6 3.8
Average of input ==> 2.70

注意：輸入整數或浮點數時，可用空格區別所輸入的資料。但輸入字元時，字元間不可有空格。

字元輸入和輸出函數

□ Function

- getche () 讀取一個字元，putchar (ch) 每次輸出一個字元。
- getche () 函數式不包含任何參數，使用：ch = getche () ；
- putchar (ch) 須包含一個字元變數：
- getche () 函数注意事項
 - 以 scanf () 讀取字元值，按enter鍵後，才正式讀取字元資料，以 getche () 讀取字元值，只要一有輸入，輸入值立刻讀入所設定的字元變數內。
- getchar () ，使用格式：ch = getchar () ；
 - 讀取的字元會被存至變數ch內。
- 以getche () 讀取字元時，不必按enter鍵，程式自動讀該字元。
- 以getchar () 讀取字元時，輸入完字元後，須按enter鍵。

字元輸入和輸出函數

- 以getch（）讀取字元時，所輸入的字元不顯示在螢幕上。

```
#include <stdio.h>
int main() {
    char ch1, ch2;
    printf("Please enter 2 characters \n==>");
    ch1 = getch();
    ch2 = getch();
    printf("\n");
    printf("The first character is \n==>");
    putchar(ch1);
    printf("\n");
    printf("The second character is \n==>");
    putchar(ch2);
}
```

Please enter 2 characters

==>

The first character is

==>p

The second character is

==>q

輸入和輸出函數

- 輸入英哩和碼數，轉換成公里。1英哩=1.609公里，1英哩=1760碼

```
#include <stdio.h>
int main() {
    int mile, yard;
    float km;
    printf("Convert the mile and yard to kilometer\n");
    printf("Please enter mile here \n==> ");
    scanf("%d",&mile);
    printf("Please enter yard here \n==> ");
    scanf("%d",&yard);
    km = 1.609 * ( mile + (float) yard / 1760 );
    printf("The result is % 8.3f \n",km);
}
```

輸入和輸出函數

□ 不同型態資料運算與強制運算元的基本應用

```
#include <stdio.h>
int main() {
    float x = 5.3;
    int y = 9;
    int z = 4;
    x = y / z;
    printf("The result is %6.2f\n",x);
    x = (float) y / (float) z;
    printf("The result is %6.2f\n",x);
}
```

The result is 2.00

The result is 2.25

輸入和輸出函數

- 輸入華氏溫度，轉換成攝氏溫度輸出，溫度轉換公式：
 - 攝氏溫度 = $(5.0 / 9.0) * (\text{華氏溫度} - 32.0)$

```
#include <stdio.h>
int main() {
    float f,c;
    printf("Input fahrenheit degree \n==>");
    scanf("%f",&f);
    c = ( 5.0 / 9.0 ) * ( f - 32.0 );
    printf("The celsius is %6.2f \n",c);
}
```

```
Input fahrenheit degree
==>103
The celsius is 39.44
```

Exercise

- BMI值計算公式 (正常範圍 BMI=18.5~24)
 - $BMI = \text{體重(公斤)} / \text{身高}^2(\text{公尺}^2)$
 - 例如：一個52公斤的人，身高是155公分，則BMI為: $52(\text{公斤}) / 1.552^2(\text{公尺}^2) = 21.6$

```
#include <stdio.h>
int main() {
    double BMI = 0, weight=0, height=0;

    return 0;
}
```

Exercise

- 工資計算，假設工作一小時12元，而薪資所得的12%是稅金，請輸入工作時數，列出扣完稅之後實際所得。

```
#include <stdio.h>
int main() {
    int hourpay = 12;
    int hour, totalpay;
    float taxrate = 0.12;
    float netpay;
    printf("\1: Please input the working hours. ==> ");
    scanf("%d",&hour);
    totalpay = hour * hourpay;
    netpay = totalpay - (totalpay * taxrate);
    printf("\2: The netpay is %8.2f \n",netpay);
}
```

Homework II

- 工資計算，假設工作一小時x元(整數)，而全部薪資所得的8%取到小數第一位是稅金，每個月勞保費為最低工資y元(整數)的5%取到小數第一位。
- 請輸入每月工作時數(整數)，輸入二個月， x, y 。
- 輸出扣完稅與勞保後實際所得、勞保費、稅金。

程式執行時間

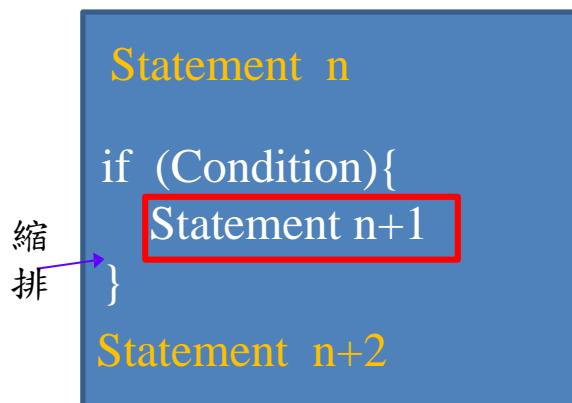
□ 計算程式執行時間

```
#include <stdio.h>
#include <time.h>
long g(int n) {
    if (n<1) return 1;
    else return (g(n-1)+g(n-2)+g(n-3));
}
int main() {
    long begin, end;
    begin = clock();
    g(32); g(32);
    end = clock();
    printf("%d ms\n", (end-begin)*100/CLK_TCK); //毫秒
    return 0;
}
```

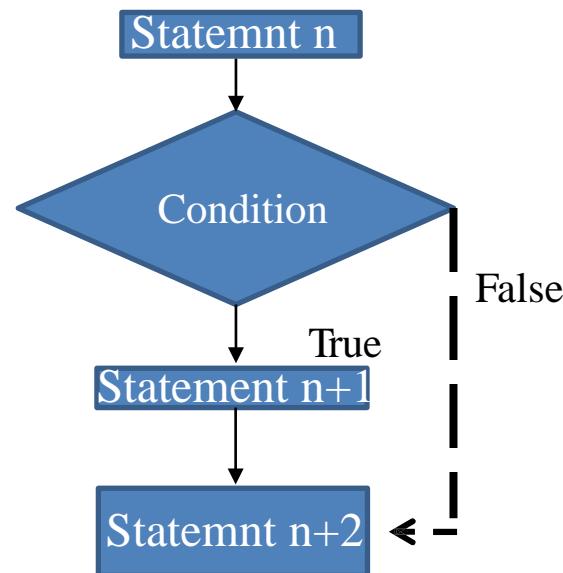
選擇 - if 條件判斷

□ 基本語法

- if (condition) , if 後跟隨"條件判斷"(Condition)
- statements , if 本體區塊 (body block) , 縮排一層 , 若"條件判斷" "True , 執行此區指令



```
if (2*2 == 4) {  
    printf("2^2=4");  
}
```



Exercise

□ 105APCS Q4

- 輸入棒球隊球員打擊結果，計算出隊得分。假設球員打擊情況：
 - 安打：以1, 2, 3 和4代表一、二、三和全(四)壘打。
 - 出局：以 0表示。
- 簡化版的規則如下：
 - 球場上有四個壘包，稱為本壘、一、二和三壘。
 - 本壘握球棒打的稱「擊球員」，在另外三個壘包的稱為「跑員」。
 - 當擊球員打擊「安打」時，擊球員與跑壘員可移動；「出局」時，跑壘員不動，擊球員離場換下一位。
 - **比賽開始由第 1位打擊，接著2, 3位球員。**
 - 打出 K 壘打時，場上球員(擊球員和跑壘員)會前進 K個壘包。本壘到一壘，接著二、三壘，最後回到本壘。回到本壘可得 1分。

Exercise

□ 105APCS Q4

○ 輸出格式

- 輸入3位打者打擊資料，輸出最後一、二、三壘狀況，有人為1，沒人為0。

輸入範例一

1

0

1

正確輸出

1 1 0

輸入範例二

1

4

0

正確輸出

0 0 0

輸入範例三

1

0

0

正確輸出

1 0 0

輸入範例四

1

2

0

正確輸出

0 1 1

Exercise

```
#include <stdio.h>
int main() {
    int state=0, input=0, score=0;
    scanf("%d", &input);
    state = (state <<input) | (1<<(input-1));
    state = state&7;
    scanf("%d", &input);
    state = (state <<input) | (1<<(input-1));
    state = state&7;
    scanf("%d", &input);
    state = (state <<input) | (1<<(input-1));
    state = state&7;
    printf("%d %d %d\n", state&1, (state>>1)&1, (state>>2)&1);
    return 0;
}
```

input = 2	
s =	0 0 0 0 0 0 1 1
s<<2 =	0 0 0 0 1 1 0 0
1<<(input-1) =	0 0 0 0 0 0 1 0

s =	0 0 0 0 1 1 1 0
7 =	0 0 0 0 0 1 1 1

s&7=	0 0 0 0 0 1 1 0

Homework III

□ 105APCS Q4

- 輸入棒球隊球員打擊結果，計算出隊得分。假設球員打擊情況：
 - 安打：以 1B, 2B, 3B 和 HB 代表一、二、三和全(四)壘打。
 - 出局：以 O1, O2, O3 表示 1, 2, 3 人出局 (OUT)。
- 簡化版的規則如下：
 - 球場上有四個壘包，稱為本壘、一、二和三壘。
 - 本壘握球棒打的稱「擊球員」，在另外三個壘包的稱為「跑員」。
 - 當擊球員打擊「安打」時，擊球員與跑壘員可移動；「出局」時，跑壘員不動，擊球員離場換下一位。
 - **比賽開始由第 1 位打擊，接著 2, 3, ..., 9 位球員。**
 - 打出 K 壘打時，場上球員(擊球員和跑壘員)會前進 K 個壘包。本壘到一壘，接著二、三壘，最後回到本壘。回到本壘可得 1 分。
 - 每達到三個出局數時，壘包清空(跑壘員都得離開)，重新開始。

Exercise

○ 輸入格式

- 1. 每組測試資料固定有十行。
- 2. 第一到九行，第一到九行，依照球員順序，每一行代表一位球員打擊資訊。每一行開始有一個正整數 a ($1 \leq a \leq 5$)，代表球員總共打 a 次。接下來有 a 個字串(均為兩字元)，依序代表每次打擊結果。資料間均以一個空白隔開代表每次打擊結果。球員打擊資訊不會有錯誤也不會缺漏。
- 3. 第十行有一個正整數 b ($1 \leq b \leq 27$)，表示要計算當總出局數累計到 b 時，該球隊的得分。輸入的打擊資訊中至少包含 b 個出局。

○ 輸出：計算在第 b 個出局數發生時的總得分，並將此得分輸出。

5 1B 1B FO GO 1B
5 1B 2B FO FO SO
4 SO HR SO 1B
4 FO FO FO HR
4 1B 1B 1B 1B
4 GO GO 3B GO
4 1B GO GO SO
4 SO GO 2B 2B
4 3B GO FO FO
3

輸出
0

5 1B 1B FO GO 1B
5 1B 2B FO FO SO
4 SO HR SO 1B
4 FO FO FO HR
4 1B 1B 1B 1B
4 GO GO 3B GO
4 1B GO GO SO
4 SO GO 2B 2B
4 3B GO FO FO
6

輸出
5

輸入範例二
1
4
0
3
1
正確輸出
0 0 0

計算機程式設計

C語言 Function

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

Function

□ 程式設計使用函式的好處

- 將大程式切割由多人撰寫，利於團隊分工，縮短程式開發時間。
- 可縮短程式長度，具結構化可讀性高，利於測試驗證、除錯維護、重複使用
- 當再開發類似功能產品，只需稍微修改即可套用。

零件:輪子、儀錶板、引擎



工廠組裝汽車



輸出:汽車



Function 呼叫

□ 呼叫function

- `x = abs(-5);`
- `printf("Hello, world");`

□ 內建函式可#include後使用

- `scanf(...), printf(...)`

□ 函式的定義可獨立成一個原始檔，個別編譯後再聯結一起。

運算後回傳的資料型態

命名方法如同變數

引數/參數

```
data_type Function_name (para1, para2, ... ){
```

函式主體指令 (body statement)

```
}
```

任何函式主體指令要縮排

必先定義而後使用

Function 定義與使用

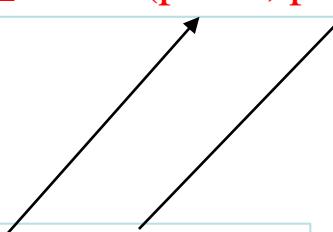
- 函式建立後不會執行，須在程式中呼叫函式才會執行
 - [變數=] 函式名稱([參數宣告串列])
- parameter / 參數 / formal parameter
 - 宣告，被呼叫時必須接收什麼資料

data_type Function_name (para1, para2, ...);

- argument / 引數 / actual argument
 - 呼叫函式時，放在括號內的變數/值

Function_name (arg1, arg2, ...);

傳遞的動作為賦值運算， $pra1 = arg1$,
 $pra2 = arg2$
...
 $praN = argN$



Function 定義

□ 自行設計與定義函式

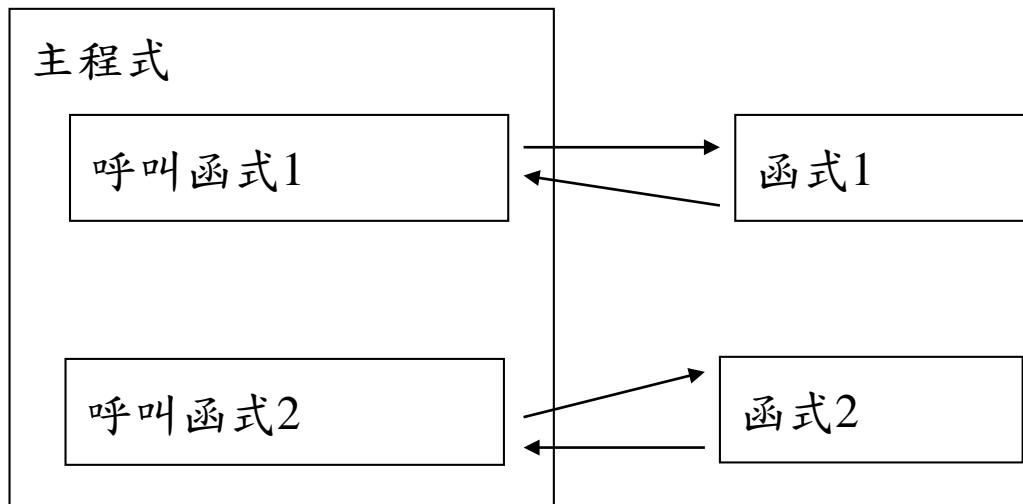
```
1. int CreateCar(int wheels, int dashboard, int engine) {  
2.     car = dashboard + wheels + engine  
3.     return car  
4. }  
5. int main() {  
6.     carA = CreateCar(wheels, dashboard, engine);  
7.     printf("%d", carA);  
8.     return 0;  
9. }
```

執行順序: 5, 6 (=右邊), 1, 2, 3, 4, 6 (=左邊), 7, 8, 9

Function 定義與使用

□ 函式執行結束

- 執行到return指令、或最後}時，
- 會離開結束函式，
- 回傳結果(值)，也可不回傳。



Function

- 計算三個數a, b, c 的標準差 sd
 - 計算三個數的平均值 average
 - 三個數分別與平均值的差之平方，三個加總後平均
 - 再開根號
 - #include <math.h>
 - x = sqrt(y)

$$SD = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

數學函式的使用

- 在程式前加：`# include < math.h >`
 - exp函式是指數函式ex，使用格式：`exp(x)`
 - 使用前將x宣告成double或float。
 - log函式，自然對數函式，使用格式：`log(x)`
 - 使用前將x宣告成double或float。
 - log10函式，以10為底的對數函式，使用格式：`log10(x)`
 - 使用前將x宣告成double或float。
 - sqrt函式，求某數平方根值，使用格式：`sqrt(x)`
 - 使用前將x宣告成double或float。

數學函式的使用

- floor函式，傳回不大於x的最大整數，使用格式：`floor(x)`
- ceil函式，傳回不小於x的最小整數，使用格式：`ceil(x)`
- fabs函式，傳回x的絕對值，使用方式：`fabs(x)`
- `pow(double x, double y)`
- `sin(x), cos(x), tan(x)`

Function

□ 計算三個數a, b, c 的標準差 sd

○ 不使用 function , 算三次

```
#include <math.h>
#include <stdio.h>
int main() {
    int a=3, b=4, c=5, sum = 0;
    double average=0.0, sd= 0.0;
    average = (a + b + c)/3.0;
    sum = (average -a)*(average -a) + (average -b)*(average -b) + (average -b)*(average -b);
    sd = sqrt(sum/3);
    a = 7; b = 8; c = 9;
    average = (a + b + c)/3.0;
    sum = (average -a)*(average -a) + (average -b)*(average -b) + (average -b)*(average -b);
    sd = sqrt(sum/3);
    a = 17; b = 18; c = 19;
    average = (a + b + c)/3.0;
    sum = (average -a)*(average -a) + (average -b)*(average -b) + (average -b)*(average -b);
    sd = sqrt(sum/3);
    return 0;
}
```

Function

□ 計算三個數a, b, c 的標準差 sd

○ 使用 function , 算三次

```
#include <math.h>
#include <stdio.h>
double getSD(int a, int b, int c) {
    double average = (a + b + c)/3.0;
    double sum = (average -a)*(average -a) + (average -b)*(average -b) + (average -b)*(average -b);
    return (sqrt(sum/3));
}
int main() {
    int a=3, b=4, c=5, sum = 0;
    double average=0.0, sd= 0.0;
    sd = getSD(a, b, c);
    a = 6; b = 7; c = 8;
    sd = getSD(a, b, c);
    a = 17; b = 18; c = 19;
    sd = getSD(a, b, c);
    return 0;
}
```

Function

- 計算三個數a, b, c 的樣本標準差 s
 - 使用 function , 改一個地方

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

```
#include <math.h>
#include <stdio.h>
double getSD(int a, int b, int c) {
    average = (a + b + c)/3.0;
    sum = (average -a)*(average -a) + (average -b)*(average -b) + (average -c)*(average -c);
    return (sqrt(sum/(3-1)));
}
int main() {
    int a=3, b=4, c=5, sum = 0;
    double average=0.0, sd= 0.0;
    sd = getSD(a, b, c);
    a = 6; b = 7; c = 8;
    sd = getSD(a, b, c);
    a = 17; b = 18; c = 19;
    sd = getSD(a, b, c);
    return 0;
}
```

命名規則

- 函數之命名應有意義，指出函數目的或回傳資料
 - 以駝峰式(Camel-Case)命名、或以小寫或輔以底線命名
 - 避免非慣用縮寫
- 若由多個單字組成，第一個單字要小寫，從第二個單字開始，每個單字的第一個字母大寫
 - studentName, bianryData, dataLength。
- 名稱不使用縮寫，以清楚、明確為原則。
 - 例如 distance 比 d 清楚
 - dataLength 比 length 更明確易理解。
- Boolean 命名用 is開頭
 - isPrime
- 函式表示一種操作，使用動詞作為第一個單字。
 - computeAverage()

命名規則

□ 命名

```
double f() {  
    double x1;  
    int x2, x3, x4;  
    scanf("%d", &x2);  
    scanf("%d", &x3);  
    scanf("%d", &x4);  
    x1=(x2+x3+x4)/3.0;  
    return x1;  
}
```

```
double ComputeAverage() {  
    double average = 0.0;  
    int englishGrade, mathGrade, chineseGrade;  
    scanf("%d", &englishGrade);  
    scanf("%d", &mathGrade);  
    scanf("%d", &chineseGrade);  
    average=(englishGrade+mathGrade+chineseGrade)/3.0;  
    return average;  
}
```



Function

□ function種類

- 有參數、有回傳值 (傳入資料，處理完後回傳結果)
- 無參數、有回傳值
- 無參數、無回傳值
- 有參數、無回傳值

Function

□ 有參數、有回傳值

○ 輸入參數為何？回傳值為何？

```
#include <math.h>
double getSD(int a, int b, int c) {
    average = (a + b + c)/3.0;
    sum = (average -a)*(average -a) + (average -b)*(average -b) + (average -c)*(average -c);
    return (sqrt(sum/2));
}
```

□ 無參數、有回傳值：輸入資料

```
double ComputeAverage() {
    double average = 0.0;
    int englishGrade, mathGrade, chineseGrade;
    scanf("%d", &englishGrade);
    scanf("%d", &mathGrade);
    scanf("%d", &chineseGrade);
    average=(englishGrade+mathGrade+chineseGrade)/3.0;
    return average;
}
```

Function

□ 有參數、無回傳值：輸出報表、結果

```
void myFunction(char name) {  
    printf("Hello~ %c", name);  
}  
int main() {  
    myFunction('K');  
    myFunction('O');  
    return 0;  
}
```

□ 無參數、無回傳值

```
void myFunction() {  
    printf("Hello~\n");  
    printf("Hi~\n");  
}  
int main() {  
    myFunction();  
    myFunction();  
    return 0;  
}
```

可維護程式

□ 程式品質

*Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.*
- Martin Fowler.

任何傻瓜都能寫出電腦能懂的程式碼。而只有好的程
式設計師才能寫出人能懂的程式碼

- Martin Fowler.

Quality is not an act. It is a habit.

- Aristotle

品質不是動作，是一種習慣

- Aristotle

Exercise

- 程式要切割模組
 - 好的設計
 - MVC架構
- 將計算三人成績總和、與標準差，練習寫成至少三個function
 - input
 - compute
 - output

Exercise

- 計算BMI公式： $BMI = \text{體重(公斤)} / \text{身高}^2(\text{公尺平方})$
 - 測試案例：52公斤的人，身高155公分，BMI為： $52(\text{公斤}) / (1.55 * 1.55)(\text{公尺平方}) = 21.64412$ 。兩位小數 21.64

```
double computeBMI(int kg, int M):  
    #BMI = round(kg/(M*M),2)          #四捨五入取兩位小數  
    BMI = ((100*kg/(M*M))/1.0)/100;  #乘 100取整數，再除100取兩位小數  
    return BMI;  
}
```

Function 不定參數

□ 不定長度引數關鍵字

- va_list: 一個特殊的型態 (type)，在 va_start、va_arg 與 va_end 三個巨集 (macro) 中當作參數使用。
- va_start: 啟始不定長度引數的巨集，第一個引數是 va_list，第二個引數是最後一個具名參數。
- va_arg: 讀取不定長度引數的巨集。
- va_end: 終止不定長度引數的巨集。

□ 宣告不定長度引數時，函式定義參數 ... 前至少要有一個具名參數，之後使用 ... 表示將使用不定長度引數，例如：

- void foo(int, ...);

□ 使用 va_arg 巨集取出引數內容時，須指定以何種資料型態取出，例如：

- va_arg(num_list, double);

Function 不定參數

□ 不定長度引數使用

```
#include <stdio.h>
#include <stdarg.h>
void foo(int len, ...) {
    va_list args;
    va_start(args, len);
    for(int j = 0; j < len; j++) {
        printf("%.1f\n", va_arg(args, double));
    }
    va_end(args);
}
int main() {
    double x = 1.1, y = 2.1, z = 3.9;
    double a = 0.1, b = 0.2, c = 0.3;
    foo(3, x, y, z);
    foo(6, x, y, z, a, b, c);
    return 0;
}
```

計算機程式設計

C語言 選擇 if/switch

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

選擇 - if 條件判斷

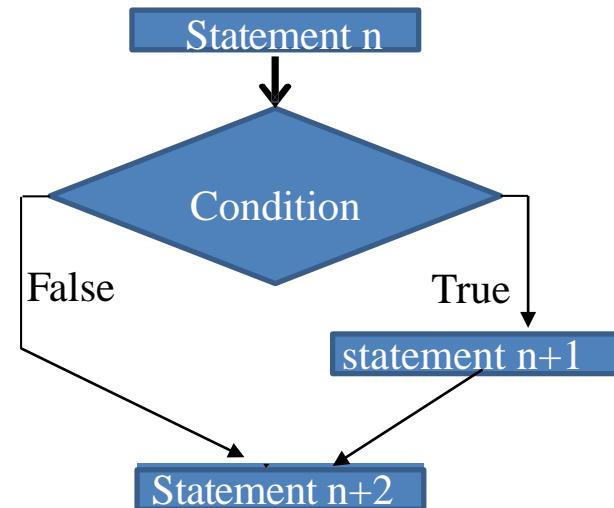
□ 基本語法

- if (condition) , if 後跟隨"條件判斷"(Condition)
- statements , if 本體區塊 (body block) , 縮排一層 , 若"條件判斷" "True , 執行此區指令

```
Statement n  
if (Condition){  
    Statement n+1;  
}  
Statement n+2;
```

縮排 → Statement n+1;

```
if (2*2 == 4) {  
    printf("2^2=4");  
}
```



If 條件判斷

□ 條件判斷，求值是 True/False，1/0

○ 關係比較運算: $x > 10$, $x < 10$, $x == 10$, $x != 10$

比較運算	語法
相等	$A == B$
不等於	$A != B$
大於	$A > B$
小於	$A < B$
大於等於	$A >= B$
小於等於	$A <= B$

```
int main() {  
    printf("%d", (2==3));  
    printf("%d", (2!=3));  
    printf("%d", (2 >= 3));  
    printf("%d", (2 <= 3));  
    return 0;  
}
```

If條件判斷

□ 邏輯運算子

not !	A
False	True
True	False

and &&	A	B
True	True	True
False	True	False
False	False	True
False	False	False

or	A	B
True	True	True
True	True	False
True	False	True
False	False	False

```
int main() {
    printf("%d", (2==3) || (3<7));
    printf("%d", (2==3) && (3<7));
    printf("%d", !(3<7));
    return 0;
}
```

□ 三種關係比較

```
if ((10 < x) && (x< 20)) {
    printf("%d 在 10~20 範圍內", x);
}
```

運算子優先順序

- 位置高表示有高優先權
- 同一行，運算時，由左至右
- $a > b + 2$ ， "+" 優先順序較 > 號高，可表示為 $a > (b + 2)$
- 設計程式最好用括號區別。

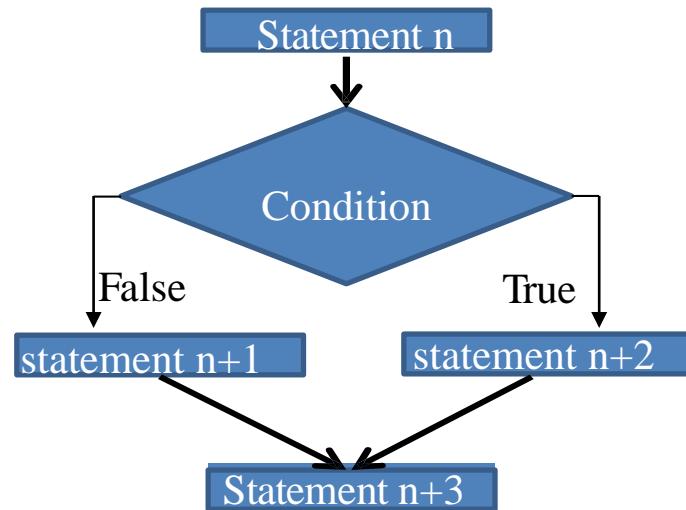
基本運算子優先順序表
! , — (負號) ++ , --
× , ÷ , %
+ , —
< , <= , > , >=
== , !=

if-else條件判斷

□ 基本語法

- if (condition) , if 後跟隨"條件"(Condition)
- if "條件"判斷True , 執行statement n+1
- if "條件"判斷True , 執行else 的 statement n+2
- statement n+1, n+2 縮排一層

```
Statement n;  
if (Condition) {  
    statement n+1;  
}  
else {  
    statement n+2;  
}
```



if 條件判斷

□ 基本指令

```
void f() {  
    int num;  
    scanf("%d", &num);  
    if (num % 2 == 0) {  
        printf("%d 是偶數", num);  
    }  
    else {  
        printf("%d 是奇數", num);  
    }  
}
```

○ if else 縮寫

- 單行可以不需要 {}
- 一般建議均使用 {}
- 可維護性

```
if (x>y)  
    maxValue = x;  
else  
    maxValue = y;
```

maxValue = x ? x>y : y

if 條件判斷

- if 條件判斷可以是一個變數

```
void f(int num) {  
    if (num!=0) {  
        printf("%d 不為 0", num);  
    }  
    if (num) {  
        printf("%d 不為 0", num);  
    }  
    if (!num) {  
        printf("%d 為 0", num);  
    }  
}
```

if 條件判斷

□ 輸入分數，判斷是否及格

```
void myFunction() {  
    int score;  
    printf("Hello~\n輸入分數: ");  
    scanf("%d", &score);  
    if (score>=60)  
        printf("恭喜你及格");  
    else  
        printf("不及格，要加油");  
}  
  
int main() {  
    myFunction();  
    myFunction();  
    return 0;  
}
```

```
Hello~  
輸入分數:61  
恭喜你及格  
Hello~  
輸入分數:59  
不及格，要加油
```

□ 輸入超過100，或負數要如何修正？

Exercise

- 若溫度(temperature)高於30而且沒有風wind=0，或濕度(humidity)大於85，印出"開冷氣"，若溫度小於10度，印出"開暖氣"。
 - CODE

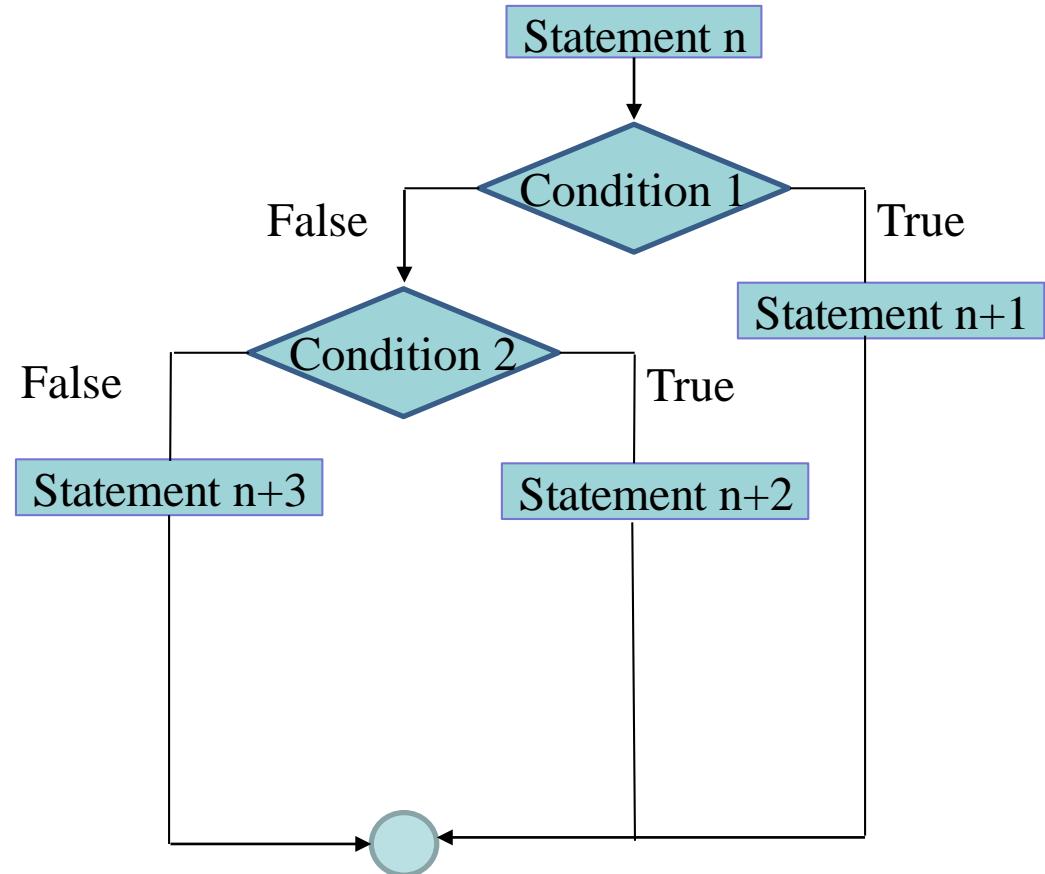
Exercise

- 寫一程式計算一元二次方程式的根 (實根或虛根)

If 條件判斷

□ if, else if, else (多選1)

```
Statement n;  
if (Condition 1)  
    Statement n+1;  
else if (Condition 2)  
    Statement n+2;  
else  
    Statement n+3;
```



if 條件判斷

□ if, else if, else (多選1)範例(比大小)

```
void ops() {  
    scanf("%d", &num1);  
    scanf("%d", &num2);  
    if (num1 == num2)  
        printf("%d 等於 %d", num1, num2);  
    else if (num1 < num2)  
        printf("%d 小於 %d", num1, num2);  
    else  
        printf("%d 大於 %d", num1, num2);  
}
```

if條件判斷

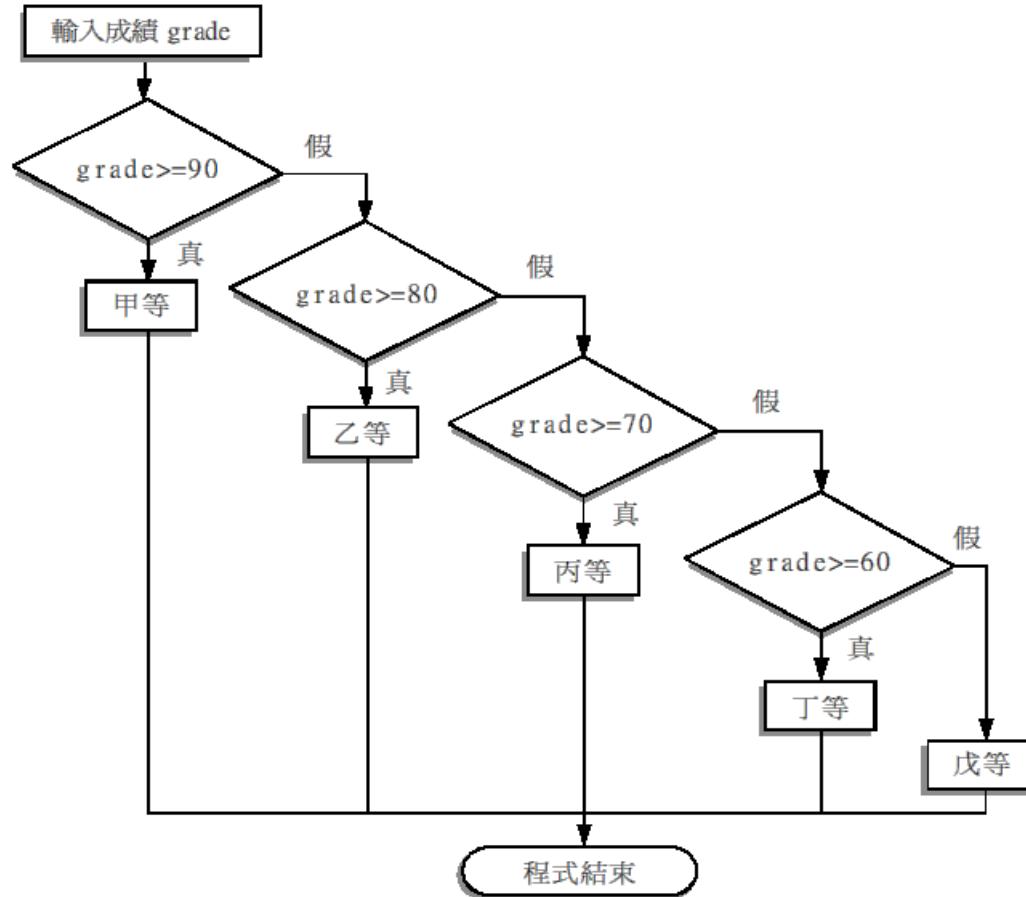
□ 判斷分數等第A, B, C, D, E, F

```
void myFunction() {  
    int score;  
    scanf("%d", &score);  
    if (score>=90)  
        printf ("得 A");  
    else if (score>=80 && score <90)  
        printf("得 B");  
    else if (score>=70 && score <80)  
        printf("得 C");  
    else if (score>=60 && score <70)  
        printf("得 D");  
    else  
        printf("不及格");  
}  
  
int main(){  
    myFunction()  
    myFunction()  
    myFunction()  
    return 0;  
}
```

這個程式邏輯是否有問題?

if條件判斷

- 判斷分數等第A, B, C, D, E, F，修正判斷邏輯
 - 超過100分或低於0分之處理



Exercise

- 以下code會輸出？

```
#include <stdio.h>
void f(int key) {
    if (key <1000 && key >=0)
        if (key < 100)
            if (key < 10)
                printf ("1 digit");
            else
                printf("2 digits");
            else
                printf("3 digits");
            else
                printf("Not allowed!");
    }
int main() {
    f(99);
    f(0);
    f(7);
    return 0;
}
```

Exercise

□ 判斷何種三角形

○ 當三個邊長能構成三角形時，再判斷該三角形為鈍角、銳角或是直角三角形，其判別方法如下：

- 1. 直角三角形：其中有兩個邊的平方和等於第三邊的平方。
- 2. 鈍角三角形：其中有兩個邊的平方和小於第三邊的平方。
- 3. 銳角三角形：任兩邊的平方和大於第三邊的平方。

○ 輸入三個整數

○ 輸出：顯示直角三角形(Right Triangle)、鈍角三角形(Obtuse Triangle)、銳角三角形(Acute Triangle)或無法構成三角形(Not Triangle)。

□ 測試資料：

input
5 12 13

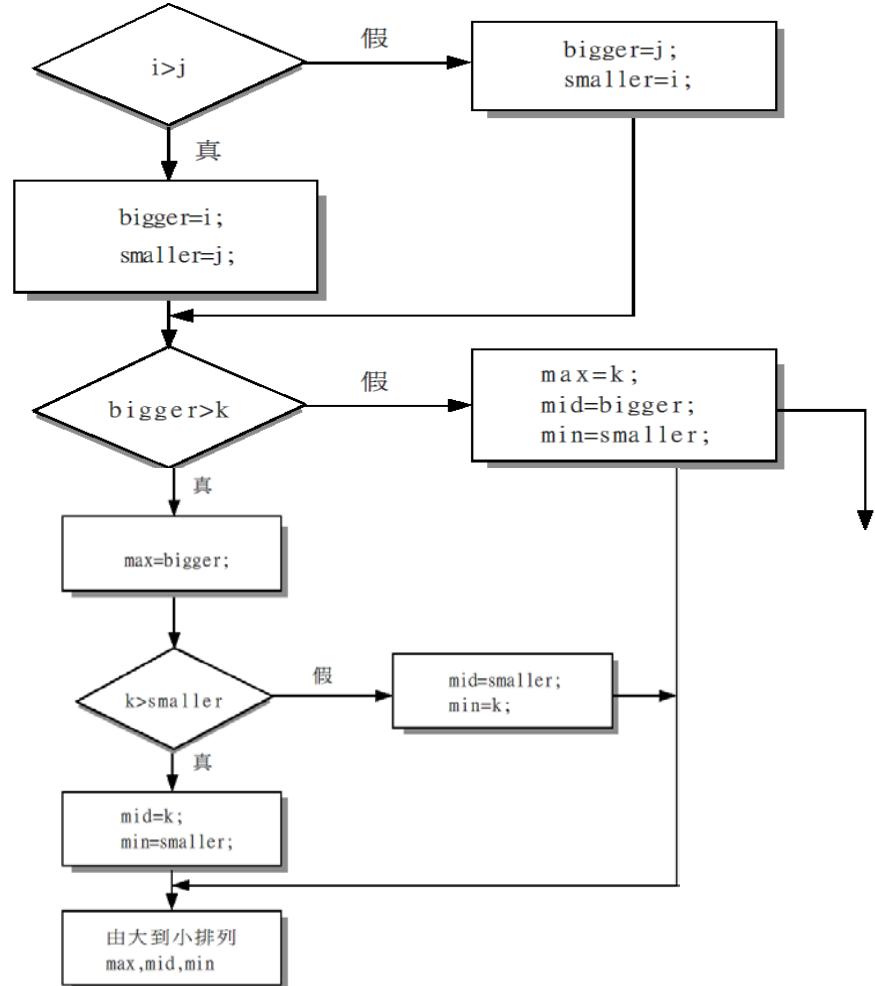
output
Right Triangle

input
3 4 5
output
Right Triangle

input
1 2 3
output
Not Triangle

Exercise

- 比較三個數大小流程圖
- 寫一個function比較



Exercise

□ 計算BMI並輸出分級值， $BMI = \text{體重 (kg)} / \text{身高 (m}^2)$

分 級	身體質量指數
體重過輕	$BMI < 18.5$
正常範圍	$18.5 \leq BMI < 24$
過 重	$24 \leq BMI < 27$
輕度肥胖	$27 \leq BMI < 30$
中度肥胖	$30 \leq BMI < 35$
重度肥胖	$BMI \geq 35$

Exercise

- A、B、C三本書價格及折扣表如下，一顧客欲購買A: x 本、B: y 本、C: z 本（x、y、z 為使用者輸入），請計算需花費多少錢？

	定價	1~10本	11~20本	21~30本	31本以上
A	380	原價	打9折	打8.5折	打8折
B	1200	原價	打9.5折	打8.5折	打8折
C	180	原價	打8.5折	打8 折	打7折

Exercise

□ Code

```
scanf("%d", &x);
scanf("%d", &y);
scanf("%d", &z);
A_discount = 0;
if (x>=31)
    A_discount = 0.8;
else if (x>=21)
    A_discount = 0.85;
else if (x>=11)
    A_discount = 0.9;
else if (x>=1)
    A_discount = 1;
```

```
B_discount = 0
if (y>=31)
    B_discount = 0.8;
else if (y>=21)
    B_discount = 0.85;
else if (y>=11)
    B_discount = 0.95;
else if (y>=1)
    B_discount = 1;
```

```
C_discount = 0;
if (z>=31)
    C_discount = 0.7;
else if (z>=21)
    C_discount = 0.8;
else if (z>=11)
    C_discount = 0.85;
else if (z>=1)
    C_discount = 1;
```

```
cost= x*380*A_discount + y*1200*B_discount + z*180*C_discount;
printf("The total cost is %d", cost);
```

Exercise

□ 有使用 function 的 Code

```
def getDiscount(int x, double d0,  
double d1, double d2, double d3){  
    double discount = 0.0;  
    if (x>=31)  
        discount = d0;  
    else if (x>=21)  
        discount = d1;  
    else if (x>=11)  
        discount = d2;  
    else if (x>=1)  
        discount = d3;  
    return discount;  
}
```

```
double A_d0=8.0, A_d1 =8.5, A_d2=9.0, A_d3=1.0;  
double B_d0=8.0, B_d1= 8.5, B_d2= 9.5, B_d =1.0;  
double C_d07, C_d =8.0, C_d2= 8.5, C_d3= 1.0;  
scanf("%d", &x);  
scanf("%d", &y);  
scanf("%d", &z);  
  
A_discount = getDiscount(x, A_d0, A_d1, A_d2, A_d3);  
B_discount = getDiscount(y, B_d0, B_d1, B_d2, B_d3);  
C_discount = getDiscount(z, C_d0, C_d1, C_d2, C_d3);
```

```
cost= x*380*A_discount + y*1200*B_discount + z*180*C_discount  
printf("The total cost is %d" %(cost))
```

Exercise

- A、B、C三本書價格及折扣表如下，一顧客欲購買A: x本、B: y本、C: z本（x、y、z為使用者輸入），請計算需花費多少錢？
- 幾本區間是否可以輸入，定價表格(區間個數)是否可以輸入？

	定價	1~10本	11~20本	21~30本	31本以上
A	380	原價	打9折	打8.5折	打8折
B	1200	原價	打9.5折	打8.5折	打8折
C	180	原價	打8.5折	打8 折	打7折

HOMEWORK I

- 輸入每月網內、網外、市話、通話時間(sec)及網內、網外簡訊則數，求最佳資費。費率如下表：

資費類型	183型	383型	983型
月租費	183元	383元	983元
優惠內容	月租費可抵等額通信費		
語音 網內	0.08	0.07	0.06
(元/秒) 網外	0.1393	0.1304	0.1087
市話(元/秒)	0.1349	0.1217	0.1018
簡訊 網內	1.1287	1.1127	0.9572
(元/則) 網外	1.4803	1.2458	1.1243

- 輸入

- 網內語音(sec)、網外語音(sec)、市話(sec)、網內簡訊數、網內簡訊數測試資料：

```
input  
500 120 13 2 5  
output
```

HOMEWORK II

□ 撲克牌

- A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K
- A~10 點數為 1~10，J, K, Q 為 0.5。

□ X, Y 兩個人各發三張撲克牌，加總點數接近 10.5 則贏。

- 超過 10.5 爆掉分數為 0。

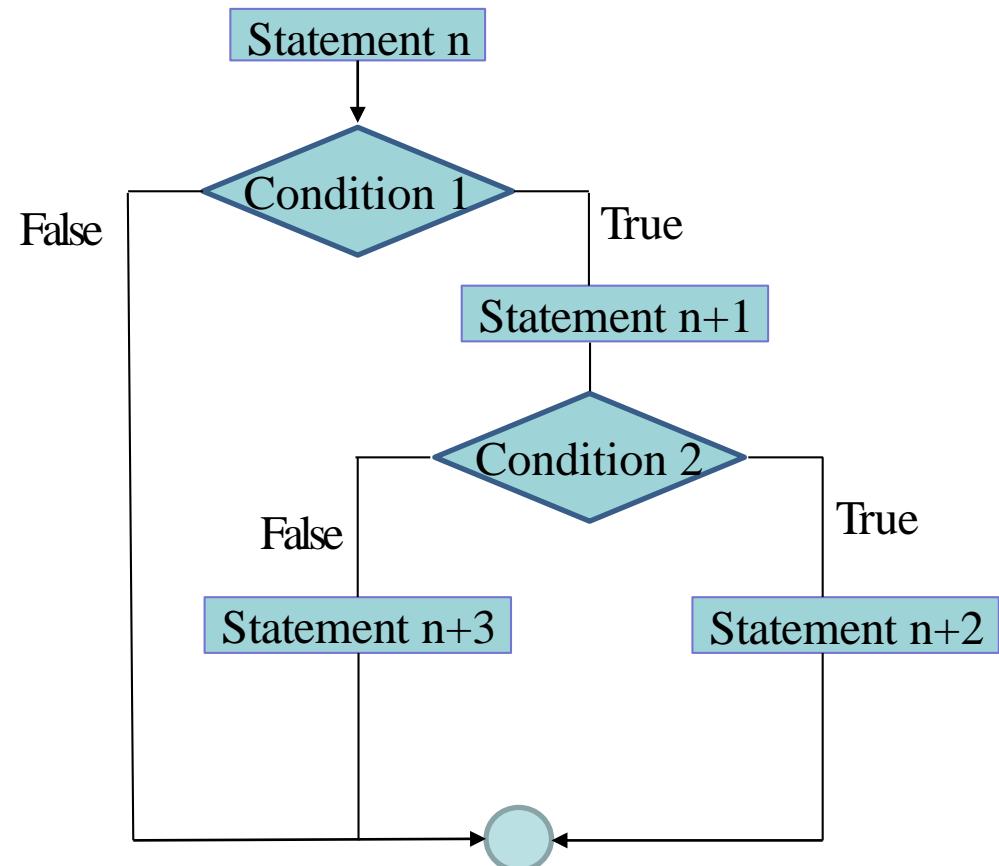
□ 程式

- 輸入X, Y 兩個人各發的五張撲克牌。
- 輸出兩個人的點數，以及A贏或B贏或平手。

巢狀 if

- if 內可以再有一層 if

```
Statement n;  
if (Condition 1) {  
    Statement n+1;  
    if (Condition 2)  
        Statement n+2;  
    else  
        Statement n+3;  
}
```



switch-case

- 從數個個案中，選擇出一個執行。

```
switch (條件算式) {  
    case 條件算式值1:  
        動作1;  
        break;  
    case 條件算式值2:  
        動作2;  
        break;  
    default:  
        最後動作;  
        break;  
}
```

switch-case

- 從數個個案中，選擇出一個執行。
 - 條件算式：結果為數值或字元的算式，也可以是變數名稱。
 - 根據條件算式，判斷執行哪個 "case"。
 - 適合多選一的條件判斷式。
 - case：在 switch 內的敘述，可同時存在兩個以上。每個 case 都有不同的條件算式值與動作，且應與他 case 不同。
 - break：結束 case 動作，**若沒有break，會一直執行直到遇見break**。
 - default：沒有條件算式值。當 switch 找不到符合的 case，便會執行 default 內的程式碼。

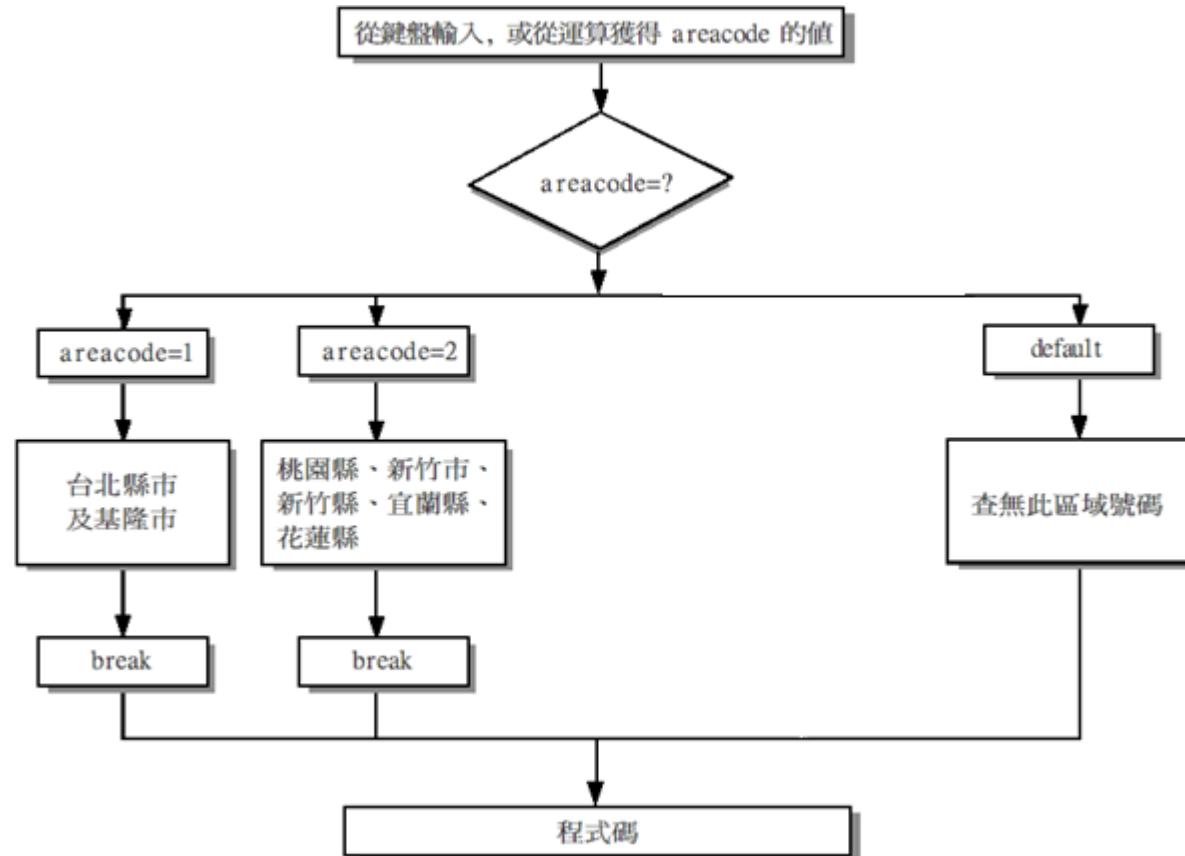
switch-case

```
#include <stdio.h>
void findCode(int areaCode) {
    switch(areaCode) { //依區域號碼判斷
        case 2:
            printf("02 是台北市、新北市與基隆市區域號碼\n");
            break;
        case 3:
            printf("03 是桃園市、新竹縣市與宜蘭花蓮縣區域號碼\n");
            break;
        default:
            printf("資料庫無此資料\n");
    }
}
int main() {
    findCode(2);
    findCode(3);
    findCode(9);
    return 0;
}
```

switch-case

□ switch 根據 areaCode 的值，選擇執行哪一個 case。

- 若 areaCode 等於 2，執行 case 2 程式碼；若 areacode 等於 3，則執行 case 3 程式碼。無對應 case 可執行，執行 default 內程式。



Exercise

□ 程式會輸出甚麼？

```
#include <stdio.h>
void findCode(int areaCode) {
    switch(areaCode) { //依區域號碼判斷
        case 2:
            printf("02 是台北市、新北市與基隆市區域號碼\n");
        case 3:
            printf("03 是桃園市、新竹縣市與宜蘭花蓮縣區域號碼\n");
            break;
        default:
            printf("資料庫無此資料\n");
    }
}
int main() {
    findCode(2);
    findCode(3);
    findCode(9);
    return 0;
}
```

Exercise

- 假設 A~F六個字元由長度為 4的二元序列編碼，個程式要從編碼辨識這六個字元。

字元	A	B	C	D	E	F
編碼	0 1 0 1	0 1 1 1	0 0 1 0	1 1 0 1	1 0 0 0	1 1 0 0

輸入

0 1 0 1

正確輸出

A

輸入

0 0 1 0

正確輸出

C

輸入

1 0 0 0

正確輸出

E

Exercise

- 假設 A~F六個字元由長度為 4的二元序列編碼，個程式要從編碼辨識這六個字元 。

```
#include <stdio.h>
char encode(int a, int b, int c, int d){
    switch(a*1000+b*100+c*10+d) {
        case 101:
            return 'A';
        case 111:
            return 'B';
        case 10:
            return 'C';
        case 1101:
            return 'D';
        case 1000:
            return 'E';
        case 1100:
            return 'F';
    }
}
```

```
int main(){
    int a=0, b=0, c=0, d=0;
    scanf("%d %d %d %d",&a, &b, &c, &d);
    printf("%c\n", encode(a,b,c,d));
    return 0;
}
```

HOMEWORK III

□ 檢查三門課程是否衝堂

- 依序輸入課程編號(數字)、上課小時數(1-3小時)、上課時間(ao
星期1-5, 第1-9節)

輸入說明

1001 (第二門課課程編號)

3 (3小時)

11 (星期1 第1節課)

59 (星期5 第9節課)

25 (星期2 第5節課)

2020 (第二門課課程編號)

...

2030 (第三門課課程編號)

...

輸出說明

(兩課程編號衝突在哪幾節)

1001 and 2020 conflict on 25

計算機程式設計

C語言 Loop

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

程式結構

□ 計算機程式有三種結構

- 循序結構

- 選擇結構

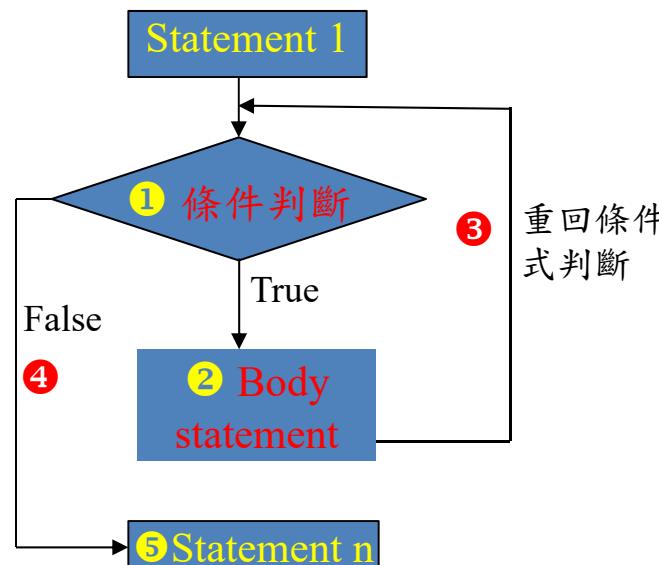
- 根據邏輯條件判斷執行正確指令。
- 以邏輯條件判斷決定之後執行哪一段程式。

- 重複結構

- 程式不斷重複執行相同程式碼，利用迴圈完成重複計算。
- 以邏輯條件判斷決定是否重複執行同一段程式碼。

迴圈原理

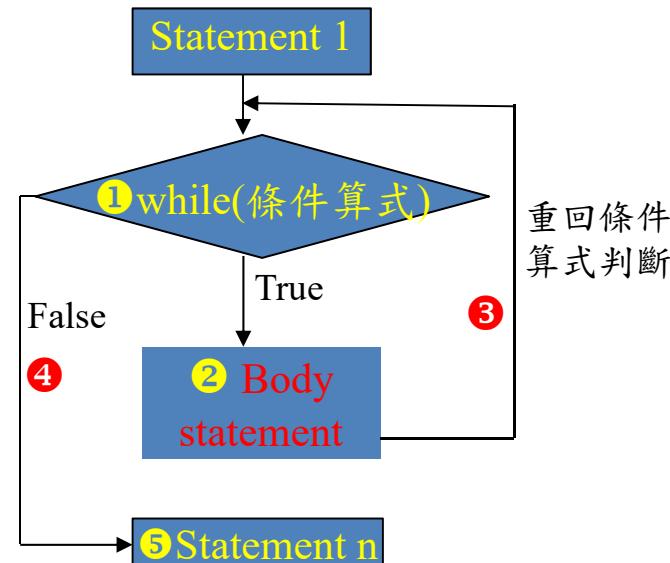
- 利用條件判斷真假，決定程式碼是否重複。
- 當條件判斷結果為真，程式會執行迴圈內容，之後重複回到條件判斷。
- 當條件算式結果為假，程式會跳出迴圈，繼續執行迴圈後的程式碼：



預先條件算式迴圈：while

- while 是預先條件式迴圈，檢查條件式結果是否真(不為 0)。
 - 條件式：可以為任何運算式、變數或數值。如果結果為非 0 的數值，則表示為真；否則為假
 - 若為真(非0)，則執行一次迴圈內動作，然後跳回條件式再檢查。
 - 如此一直執行到條件算式不成立為止(0) 才離開迴圈。
 - 迴圈內動作：可以為任何合法程式指令。

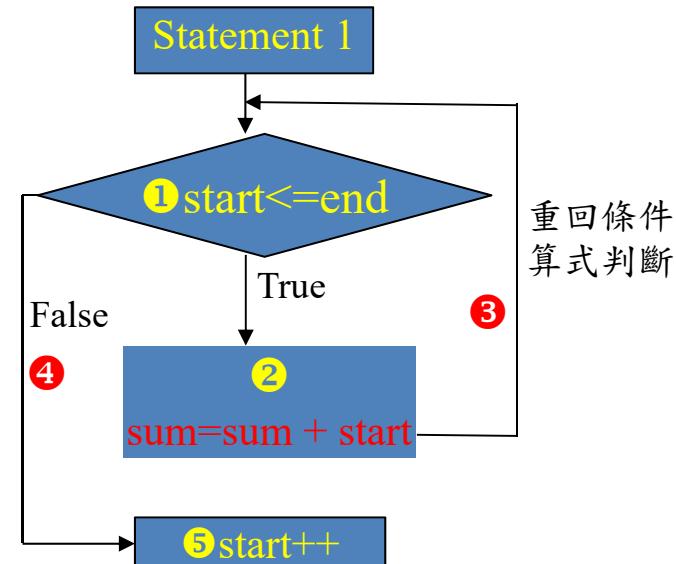
```
while (條件算式) {  
    迴圈內動作  
}
```



預先條件算式迴圈：while

- 適用於當迴圈內容不一定需執行時，因 while 條件式會先判斷，再決定是否執行迴圈動作。

```
#include <stdio.h>
int getSum(int start, int end) {
    int sum = 0;
    while (start<=end) {
        sum = sum + start;
        start = start + 1;
    }
    return sum;
}
int main() {
    printf("%d\n", getSum(1,10));
    return 0;
}
```



執行結果

- 迴圈條件算式，根據輸入的 start 跟 end 值判斷。如果 start 值小於或等於 end，則會執行迴圈動作，開始累加計算；如果 start 值大於 end，表示條件為假，不會執行迴圈動作。
 - 3~8 ?
 - 7~4 ?

求兩數的最大公因數

- 兩數的最大公因數，可用輾轉相除法。
 - 兩數先相除一次後，用除數當新的被除數，餘數當新的除數。
 - 如此不斷相除，直到除數大於被除數為止，除數即為最大公因數。
 - 要避免除以 0 的情況，須判斷除數是否為 0，才能進行相除。

```
#include <stdio.h>
int gcd (int n1, int n2) {
    while ((n1!=0)&&(n2!=0)) {
        if (n1>n2) n1=n1%n2;
        else n2=n2%n1;
    }
    if (n1>n2) return (n1);
    else return (n2);
}
int main() {
    printf("%d\n", gcd(12,10));
    printf("%d\n", gcd(54,48));
    return 0;
}
```

程式執行說明

- 迴圈 while 的條件算式，在 $n1!=0$ 和 $n2!=0$ 為真的情況下，程式會執行while 底下大括號內的迴圈內容。
- 迴圈內容是輾轉相除法的程式碼。每次除完就以除數當成被除數，餘數當成除數一直除下去，直到除盡為止。
- 除盡後， $n1$ 或 $n2$ 值會為 0，使 while 的條件算式結果為假，會跳出迴圈執行迴圈後的程式碼。

Exercise

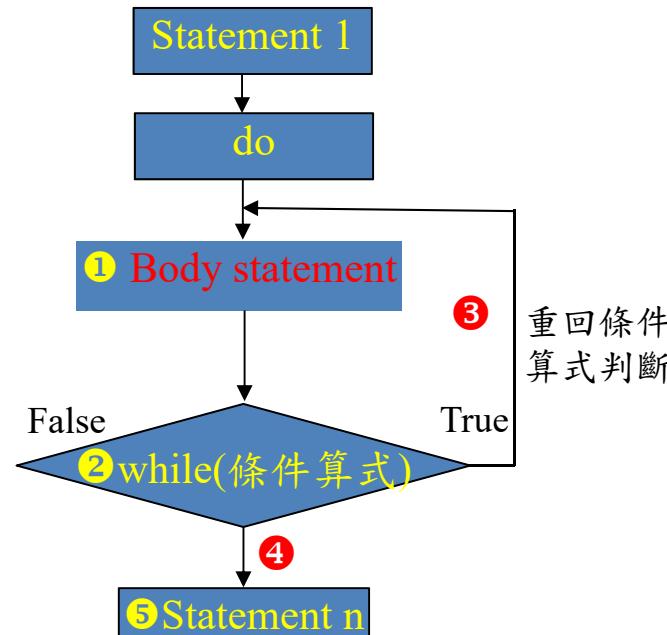
- 印出 $1, 2, 4, 9, 16, 25 \dots 225$ 數字和

- 求三個數的最小公倍數

後設條件算式迴圈：do-while

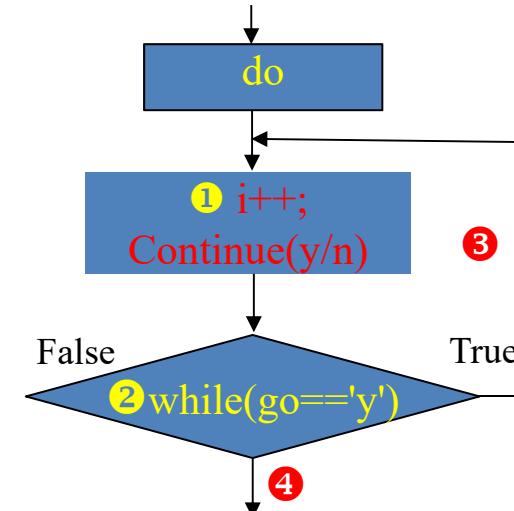
- do-while 是後設判斷式迴圈，先執行一次動作後，再判斷迴圈控制條件，若條件成立再回到前面執行 {} 內動作，如此重複直到條件算式結果為假。

```
do {  
    迴圈內動作  
} while (條件算式);
```



累計迴圈執行的次數

```
#include <stdio.h>
int main() {
    int i=0;
    char go='y';
    do {
        i++;
        printf("; Loop %d\n", i);
        printf("Continue(y/n):");
        go = getche();
    } while (go=='y');
    printf("\nTotal Loop %d\n", i);
    return 0;
}
```



; Loop 1
Continue(y/n):y; Loop 2
Continue(y/n):y; Loop 3
Continue(y/n):y; Loop 4
Continue(y/n):y; Loop 5
Continue(y/n):y; Loop 6
Continue(y/n):y; Loop 7
Continue(y/n):n
Total Loop 7

Exercise 累加值

- 輸入非0，印出目前累加值，輸入0，印出累加值並停止。

```
目前累加值:0  
輸入下一個值: 25  
目前累加值:25  
輸入下一個值: 18  
目前累加值:43  
輸入下一個值: 33  
目前累加值:76  
輸入下一個值: 64  
目前累加值:140  
輸入下一個值: 0  
累加值:140
```

執行結果

- 5~10 行是 do-while 迴圈程式碼，10 行條件算式會判斷，在第 8 行輸入的 number 值是否為 0。若不為 0，會跳回第 5 行，再將迴圈內容執行一次；若為 0，則跳出迴圈，執行第 11 行的程式碼。

```
1 #include <stdio.h>
2 int main() {
3     int sum=0;    //數值總和
4     int number=0; //累加數值
5     do {
6         printf("目前累加值:%d\n", sum);
7         printf("輸入下一個值: ");
8         scanf("%d", &number);
9         sum = sum + number;
10    } while (number!=0);
11    printf("累加值:%d\n", sum);
12    return 0;
13 }
```

```
目前累加值:0
輸入下一個值: 25
目前累加值:25
輸入下一個值: 18
目前累加值:43
輸入下一個值: 33
目前累加值:76
輸入下一個值: 64
目前累加值:140
輸入下一個值: 0
累加值:140
```

無限迴圈/無窮迴圈

- 當迴圈的條件算式設定有誤，使迴圈的條件算式結果恆真，迴圈動作就會不斷執行，直到程式被強迫中止(按下組合鍵 Ctrl + C)或硬體停止回應(電腦當機)程式才會被終止。

```
while (1) {  
    printf("無限迴圈\n");  
}
```

```
do {  
    printf("無限迴圈\n");  
} while (1);
```

跳離迴圈：break

□ break 跳出一層迴圈

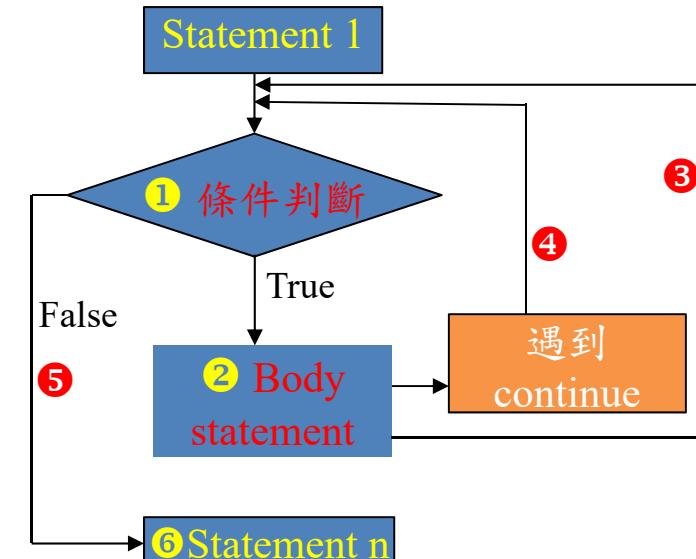
- 當 while、do-while 迴圈在執行中，想跳出迴圈有兩種方法
 - 一是使判斷條件不成立，另一是使用 break。
- break 讓程式立即跳出迴圈，繼續執行迴圈之後的程式。
 - 將 break 放置於想要跳離的迴圈內即可：
- 一個 break 只會跳出一層迴圈。
 - 如果是三層巢狀迴圈，就需三個 break 才能完全跳脫迴圈。

```
while (1) {  
    printf("無限迴圈\n"); ←—— 這訊息只出現一次  
    break;  
}
```

跳出一輪迴圈

- continue 跳出一輪迴圈，但未必跳出一層迴圈
 - while, for 都可以使用

```
#當number 沒超過20 不印@，超過印@
void test03() {
    int i=0, number =0;
    for (i=1; i<30; i++) {
        number = number +i;
        if (number<20)
            continue
        print("@ %d, %d", i, number);
    }
}
```



利用continue在任何時候略過迴圈
(略過本次迴圈剩餘的運算)

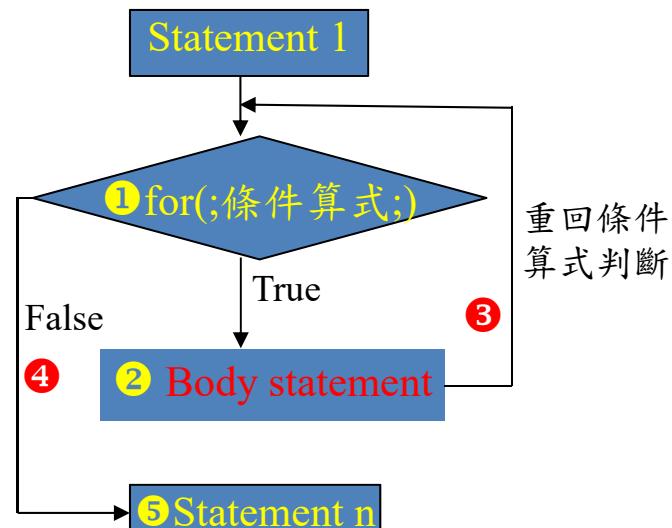
範圍設定式迴圈：for

□ for 迴圈

- 利用索引變數值的累加或累減控制迴圈執行次數，等該變數值達到設定標準，就會跳出迴圈。
- for 迴圈最好不要用浮點數來控制

```
for (初始值設定;終止條件判斷;每次迴圈後的運算) {  
    迴圈內的指令動作  
}
```

初始值設定
while (終止條件判斷) {
 迴圈內的指令動作
 每次迴圈後的運算
}



範圍設定式迴圈：for

- 初始值設定：設定條件式中用到變數之初始值。
 - 例如 $i=1$
- 終止條件判斷：判斷是否執行迴圈中的程式。
 - 每次迴圈開始時檢查一次。
 - 例如設 $i < 3$ ，表示只有在 i 小於 3 才會執行迴圈動作；若 i 大於等於 3，則不會執行迴圈。
- 每次迴圈後的運算：用於調整條件算式中的變數值
 - 例如條件算式為 $i < 3$ ，用此運算來改變 i 的值，使終止條件成立
 - 例如 $i++$ ，使 i 的值最終大於等於 10，進而使迴圈結束。
 - 控制算式會在每次迴圈執行完畢時執行 1 次。

```
for (i=1; i<3;i++) {  
    //迴圈內指令動作  
}
```

for 迴圈執行步驟

1. 從初始算式開始，之後執行條件算式判斷是否執行迴圈。若結果為真，則執行迴圈動作。
2. 執行過一次迴圈動作後，執行每次迴圈後的運算，再次判斷條件算式中的結果。
3. 重複步驟 1、2，直到條件算式的判斷結果為假，跳出迴圈。

i = 0 → i < 3 為 True → 執行迴圈指令 → i++, i值變為 1
i = 1 → i < 3 為 True → 執行迴圈指令 → i++, i值變為 2
i = 2 → i < 3 為 True → 執行迴圈指令 → i++, i值變為 3
i = 3 → i < 3 為 False → 跳出迴圈

```
for (i=1; i<3;i++) {  
    //迴圈內指令動作  
}
```

Exercise for 迴圈累加

- 計算 1~100 間所有奇數的和。
- 如何設定迴圈的執行條件，
 - 初始算式設定用於累加的變數 i 其初始值為 1、
 - 條件算式設定 $i < 100$ 、
 - 控制算式設為每次將 i 的值加 2 (因為只計算奇數) ，
 - 在迴圈中做累加的動作。

簡單的 for 迴圈累加

- 第 4、5 行，為 for 迴圈，程式碼只有一行，可省略大括號 {}。
 - 第47 行，初始算式 i=1 將 i 值設為 1；條件式 i 小於 100 下為真，持續執行迴圈動作；i+=2 表示每次迴圈執行後，將 i 加 2。
 - 當 i 值變成 100 時，結束迴圈。
 - 1~100，2500

```
1 #include <stdio.h>
2 int main() {
3     int sum=0; //計算總和
4     for (int i=0; i<100; i+=2)
5         sum = sum+i; //迴圈每次跳 2
6     printf("\nTotal %d\n", sum);
7     return 0;
8 }
```

for 迴圈中可有兩組算式

- for 迴圈允許使用兩組以上的初始算式、條件算式及控制算式，每組間以逗號隔開：

```
for (初始值設定1, 2;終止條件判斷1, 2;每次迴圈後的運算1, 2) {  
    迴圈內的指令動作  
}
```

```
for (i=0, j=0; i<3, j<3; i++, j++)
```

- 變數 i、j 初始值均為 0，每執行一次迴圈，i 會依控制算式 i++，將 i 加 1。
- 同時，j 也會依控制算式 j++，將 j 加 1。
- 一直到兩個條件算式 i<3 且 j<3 同時不成立，才會跳出迴圈。
- 寫一個程式計算一個多項式 $(1+2) + (2+4) + (3+6) \dots + (n+2*n)$ ，可使用此種 for 迴圈完成。

for 迴圈可用浮點數控制

- for 迴圈的迴圈變數可以每次加0.1。(不建議使用)
 - 迴圈有兩個迴圈變數，sum 與 i。每次迴圈執行後，將當時 i 值加到 sum，且 i 加 0.1。

```
#include <stdio.h>
int main() {
    double sum=0; //計算總和
    for (double i=0.1; i<1.05; sum+=i, i+=0.1)
        printf(" %.1f + ", i);
    printf(" = %.1f\n", sum);
    return 0;
}
```

0.1 + 0.2 + 0.3 + 0.4 + 0.5 + 0.6 + 0.7 + 0.8 + 0.9 + 1.0 + = 5.5

跳出一輪迴圈

- break 是跳脫整個迴圈
- continue是跳脫 "這一輪" 迴圈。
 - 第 3~7 行迴圈，輸出從 1 到 10 除了 5 以外的數值。
 - 當迴圈進行到第 5 圈，會使第 4 行 if 條件判斷式的判斷為真。而執行第 5 行的 contiune，跳過 $i=5$ 這一輪，繼續執行下一輪。

```
1 #include <stdio.h>
2 int main() {
3     for (int i=0; i<=10; i++){
4         if (i==5)    // i=5 時成立
5             continue; //跳脫第5次迴圈
6         printf("%d ", i);
7     }
8     return 0;
9 }
```

0 1 2 3 4 6 7 8 9 10

for 的無限迴圈

- for 迴圈中有三個設定項目，若缺少條件算式，或程式邏輯錯誤使條件算式永遠為真時，會導致一直執行不停的無限迴圈：

```
#include <stdio.h>
int main() {
    for (int i=1;;)
        printf("cannot stop\n");
    return 0;
}
```

使用迴圈的注意事項

□ 條件算式的設定要合理

- 不當的條件算式設定會產生無限迴圈，或者根本未執行到迴圈的內容。所以在設定迴圈的條件式時，請仔細檢查條件算式的推演結果，以下是一些條件算式不合理的例子：

while(1) 條件算式為非 0 數值，導致無窮迴圈

for (i=0; i<100;) 缺項導致 $i < 100$ 永遠為真，形成無窮迴圈

for (i=0; i<0; i++) 條件算式不可能為真，永遠不會執行迴圈

□ 依照程式需求，選擇使用特性適合的迴圈

- 先判斷再決定是否執行時，使用 while 迴圈
- 先執行一次再決定是否繼續時，使用 do-while 迴圈
- 準確控制迴圈內容的執行次數，使用 for 迴圈

巢狀迴圈

- 巢狀迴圈是在迴圈的條件算式為真時，所執行的動作內還有其他迴圈。

```
for (初始運算式1;條件算式1;控制算式1) {
```

 第一個迴圈動作指令1

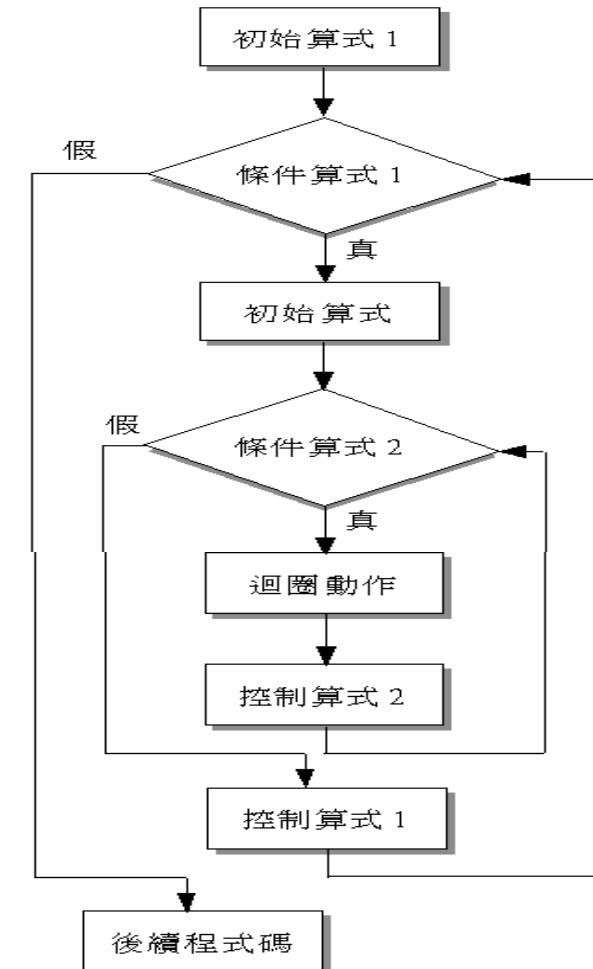
```
        for (初始運算式2;條件算式2;控制算式2) {
```

 第二個迴圈動作指令

```
}
```

 第一個迴圈動作指令2

```
}
```



巢狀迴圈

- 2層巢狀迴圈為例，外迴圈每執行一圈，就把所有內迴圈執行一次。
 - 每次執行時都會先執行外迴圈的第一圈，
 - 然後把內迴圈所有迴圈執行完後，
 - 再執行外迴圈的第二圈。

```
1 #include <stdio.h>
2 int main() {
3     for (int i=2; i<10; i++) {
4         for (int j=2; j<10; j++)
5             printf(" %d*%d=%2d ", i, j, i*j);
6         printf("\n");
7     }
8     return 0;
9 }
```

程式執行說明

- 第 3~7 行，為巢狀迴圈。第 4、5 行，為內迴圈。
 - 第 3 行，外迴圈變數 $i=2$ ，進入內迴圈連續執行 8 次。分別為 $i=2 j=2, i=2 j=3, \dots, i=2 j=9$ 。
 - 執行完第 6 行，輸出換行字元後，回到第 3 行，以 $i=3$ 再次進入內迴圈，執行 $i=3 j=2, i=3 j=3 \dots i=3 j=9$ 8 次後，再以 $i=4$ 帶入。
 - 以此類推，直到 i 的值不小於 10 為止。

```
2*2=4 2*3=6 2*4=8 2*5=10 2*6=12 2*7=14 2*8=16 2*9=18  
3*2=6 3*3=9 3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27  
4*2=8 4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36  
5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45  
6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54  
7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63  
8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72  
9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

Homework I

□ 迴圈與 if 條件判斷式混合應用：求 1~N 之間的質數

- 質數是除了 1 與本身之外，無法被其他數整除的數。
- 可利用巢狀迴圈計算 1~100 間任一個數字，是否會被比它小的數字(從 2 開始) 整除，會整除的就不是質數。找不到可整除的，就是質數。
- 輸出，質數每三個輸出一個。

輸出等腰三角形圖案

- 巢狀迴圈，可將外迴圈的迴圈變數，放入內迴圈的條件變數。
隨著外迴圈執行次數增加，內迴圈的執行次數也跟著改變。
- 利用內外迴圈變數間的關係，輸出星號等腰三角形。

```
1 #include <stdio.h>
2 int main() {
3     int n; // 三角形底的星號數
4     do {
5         scanf("%d", &n);
6     } while (n%2==0);
7     for (int i=0; i<=n/2; i++){
8         for (int j=n/2; j>i; j--) //控制輸出空白
9             printf(" ");
10        for (int k=0; k<=2*i; k++) //控制輸出*
11            printf("*");
12        printf("\n");
13    }
14    return 0;
15 }
```

4	6	9
*	***	*****
*****	*****	*****

程式執行說明

- 第 7~13 行，外迴圈控制換行。依輸入三角形的底，設定行數。
- 第 8、9 行，第 1 個內迴圈，用來控制每行輸出空白字元數。
- 第 10、11 行，第 2 個內迴圈，用來控制每行輸出的星號數。
輸出的星號，會印在第 9 行迴圈所輸出的空白字元後。

Exercise等腰三角形圖案

- 使用一層迴圈與function，輸出等腰三角形圖案

```
void printStar(int n) {  
    for (int i=1; i<=n; i++)  
        printf("*");  
}
```

n=5

```
void printStar(int n, char mark) {  
    for (int i=1; i<=n; i++)  
        printf("%c", mark);  
}
```

```
#####*  
###***  
##*****  
#*****  
*****
```

```
void printTriangle(int n) { // n 是高度  
  
}
```

巢狀迴圈應用：輸出字母圖形

□ 輸出以字母組成的直角三角形

- 第 3~7 行是外迴圈，迴圈變數 i 用來控制內迴圈的輸出數。
- 第 4、5 行是內迴圈，用來輸出英文字母。
- 第 5 行的 printf() 會將數字轉換成字元從螢幕輸出。
- 外迴圈每執行一次，將 i 值代入內迴圈。內迴圈就會執行 i 次，並輸出 i 順序的字母。

```
1 #include <stdio.h>
2 int main() {
3     for (int i=1; i<7; i++){
4         for (int j=0; j<i; j++)
5             printf("%c", j+65);
6         printf("\n");
7     }
8     return 0;
9 }
```

```
A
AB
ABC
ABCD
ABCDE
ABCDEF
```

巢狀迴圈應用：輸出字母圖形

- 輸出以字母組成的直角三角形

```
1 void printLine(int n);  
2  
3 void print(int n) {  
4     for (int i=1; i<n; i++) {  
5         printLine();  
6         printf("\n");  
7     }  
8 }  
9 }
```

```
A  
AB  
ABC  
ABCD  
ABCDE  
ABCDEF
```

Exercise

□ 以下程式輸出？

```
#include <stdio.h>
int main() {
    int a,b;
    for(b=1;b<=3;b=b+2 ) {
        for( a=6;a>=2;a=a-2 )
            printf("%d,%d,%d\n",a,a+b,a*b);
        printf("\n");
    }
}
```

Homework II

- 將Code寫成二個function，每一個function使用一層迴圈
 - 輸入圖案編號與層數，輸出各種圖形

1
12
123
1234
12345

54321
4321

321
21
1

1
22
333
4444
55555

1
121
12321
1234321
123454321

—1—
—212—
—32123—
4321234

4321234
—32123—
—212—
—1—

Exercise

□ APCS 2019

- A, B 兩隊比賽籃球，每場籃球賽有四節，輸入A, B雙方每一節的比分，求最終比賽結果，其規則為若A兩場全贏：Win；兩場全敗：Lose；一勝一敗：Draw。
- 輸入皆為正整。
- 每場比賽雙方比分不同。
- 輸入 A:B 二場、每場四節的比分
- 使用 function 改善

11 2

22 12

13 17

16 18

22 33

41 15

32 19

26 28

輸出

Win

```
#include <stdio.h>
int main(){
    int i=0, a = 0, b = 0, ans = 0, ai=0, bi=0;
    for (i = 0 ; i < 4 ; i++){
        scanf("%d %d",&ai, &bi);
        a += ai;      b += bi;
    }
    ans += ( a > b ? 1 : -1 );
    a = b = 0;
    for (i = 0 ; i < 4 ; i++){
        scanf("%d %d",&ai, &bi);
        a += ai;      b += bi;
    }
    ans += ( a > b ? 1 : -1 );
    if (!ans)      printf("Draw");
    else if ( ans > 0 )  printf("Win");
    else          printf("Lose");
    printf("\n");
    return 0; }
```

迴圈混合應用：計算階乘

- 從鍵盤輸入正整數，輸出該數字的階乘，再詢問是否要繼續。

```
1 #include <stdio.h>
2 int factorial(int n) {
3     for (int i=n-1; i>0; i--)
4         n = n*i;
5     return n;
6 }
7 int main() {
8     int number=0; //計算階層
9     char go='y'; //判斷迴圈是否繼續
10    do {
11        printf("\n計算階乘，請輸入一個值:");
12        scanf("%d", &number);
13        printf("Answer=%d\n", factorial(number));
14        printf("是否繼續(y/n):");
15        go = getche();
16    } while (go=='y');
17    return 0;
18 }
```

計算階乘，請輸入一個值:12
Answer=479001600
是否繼續(y/n):y
計算階乘，請輸入一個值:23
Answer=862453760
是否繼續(y/n):n

程式執行說明

- 2~6，factorial，計算階乘。範圍由迴圈判斷條件設定為輸入的數值到 1。
- 10~16，後設條件判斷式迴圈 do-while，用來判斷是否繼續輸入數字。
- 15，輸入判斷字元。再由 16 回圈的條件算式判斷，決定是否繼續執行迴圈。

Exercise

□ 計算BMI值的function

- BMI值計算公式: $BMI = \text{體重(公斤)} / \text{身高}^2(\text{公尺}^2)$
- 例如：一個52公斤的人，身高是155公分，則BMI為：
- $52(\text{公斤}) / 1.552 (\text{公尺}^2) = 21.6$
- 正常範圍為 $BMI=18.5 \sim 24$
- 輸入身高、體重，輸出BMI值。
- 身高正常範圍 0.5~2.50 公尺，體重正常20~300 公斤，若輸入不在正常範圍，輸出 "Input Error (0.5~2.50)" / "Input Error (20~300)"，請重新輸入。
- 若BMI值太高，輸出 "BMI too hight"，太低輸出 "BMI too low"。
- 可以接受不斷輸入計算，直到輸入-1停止。

Exercise

- 猜數字，隨機產生一個介於1~10的答案，使用者猜中則停止輸入，根據使用者輸入提示以下訊息：
 - 1.猜太大
 - 2.猜太小
 - 3.猜中了

```
#include <stdio.h>
void myFunction() {
    int input=0;
    int ans = random.randint(1,10)
    while (true) {
        printf("Guess 1~10: ");
        scanf("%d", &input);
        if (inputData == ans) {
            print("Right")
            break;
        }
    }
}
int main() {
    myFunction();
}
```

Homework III

□ 撲克牌

- A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K
- A~10 點數為 1~10，J, K, Q 為 0.5。

□ 電腦與玩家各隨機(測資輸入)發撲克牌，加總點數接近 10.5 則贏。

- 超過 10.5 爆掉分數為 0。

□ 程式

- 隨機(測資)發一張撲克牌給玩家，玩家選擇要牌或不要牌。
- 隨機(測資)發一張撲克牌給電腦，電腦判斷是否停發牌。
 - 牌面比玩家小，要牌
 - 小於8點(含)，要牌
 - 其餘情況不要牌
- 輸出電腦與玩家的點數，以及電腦贏或玩家贏或平手。

Homework III

A 先發一張給給玩家

J 再發一張給電腦

Y 玩家選擇要牌

9 發一張給玩家

8 電腦牌面 0.5 點，未超過 8 點，再發一張給電腦

N 玩家選擇不要牌

3 電腦牌面小於玩家，要牌

10.0 vs. 0.0

player win

9 先發一張給給玩家

8 再發一張給電腦

N 玩家選擇不要牌

9 電腦牌面比玩家少，要牌

9.0 vs. 0.0

player win

4 先發一張給給玩家

6 再發一張給電腦

Y 玩家選擇要牌

1 發一張給玩家

2 電腦牌面 6 點，未超過 8 點，要牌

Y 玩家選擇要牌

5 發一張給玩家

J 電腦牌面比玩家少，要牌

2 電腦牌面比玩家少，要牌

10.0 vs. 10.5

computer win

Homework III

```
#include <stdio.h>
double input(char x, char y) {
    if (x=='1') return 10;
    else if ((x>='2')&&(x<='9')) return (x-'0');
    else if (x=='A') return (1);
    else return 0.5;
}
int isDeal(int sum) {
    if (sum<=8) return 1;
    return 0;
}
int main() {
    char x, y;
    double in=0, scoreA=0, scoreB=0;
    scanf("%c%c", &x, &y);
    in = input(x, y);
    scoreA = scoreA + in;
    printf("%.1f, %.1f\n", input(x,y), scoreA);
```

```
if (isDeal(scoreA)) {
    scanf("%c%c", &x, &y);
    in = input(x, y);
    scoreA = scoreA + in;
    printf("%.1f, %.1f\n", input(x,y), scoreA);
}
if (isDeal(scoreA)) {
    scanf("%c%c", &x, &y);
    in = input(x, y);
    scoreA = scoreA + in;
    printf("%.1f, %.1f\n", input(x,y), scoreA);
}
return 0;
```

Homework IV

- 輸入10進位整數，轉成二進位
- 輸入二進位，轉成10進位
- 輸入 b_1 進位 x ，轉成 b_2 進位 y

計算機程式設計

C語言 Unit Test

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

單元測試 (Unit Test)

□ 測試

- 一個程式如何算完成，如何證明程式沒有問題。
 - 應有測試程式(test driver)與測試案例(test case)驗證，程式通過這些test case。
 - 單元測試用簡單明確方法，驗證某功能在測試案例如預期運作。
- 計算BMI公式： BMI = 體重(公斤) / 身高^2(公尺平方)
- 測試案例：52公斤的人，身高155公分，BMI為： $52/(1.55 \times 1.55) = 21.64412$ 。兩位小數 21.64

```
double computeBMI(int kg, int height) {  
    double M = height/100.0;  
    double BMI = 0.0;  
    if (kg<=0 || height<=0)  
        return -1;  
    BMI = round(kg/(M*M),2); //四捨五入取兩位小數  
    return BMI;  
}
```

單元測試 (Unit Test)

- #include <assert.h> , assert(int expression) , 錯誤會中斷程式。

```
#include <stdio.h>          // main.c
#include <math.h>
#include <assert.h>
double computeBMI(int kg, int height) {
    double BMI = 0.0, M = height/100.0;
    if (kg<=0 || height<=0)
        return -1;
    BMI = round(100*kg/(M*M))/100; //四捨五入取兩位小數
    return BMI;
}
int main() {
    int kg = 52, height = 155;
    double expectedResult = 21.64f;
    double result = computeBMI(kg, height);
    assert(fabs(result-expectedResult)<0.0001);
    printf("Hi\n");
    return 0;
}
```

程式碼涵蓋度 (Code Coverage)

- 程式碼涵蓋度 (Code Coverage) – 指令涵蓋與分支涵蓋
 - 每條指令是否執行過? 每個分支判斷 True/False 是否執行過?
- gcov (GCC Coverage) , 測試程式碼覆蓋率的工具
 - 分析哪幾行程式被執行過，統計每一行程式的執行次數
- gprof 程式分析工具(profiling tool) , 分析程式耗費時間
 - 藉以改善程式執行效率

程式碼涵蓋度 (Code Coverage)

□ 指令行執行步驟

- 編輯main.c
- 在指令行執行gcc -fprofile-arcs -ftest-coverage -o main main.c
 - 在目標檔中加入gcov所需extra profiling information
 - 產生main.exe、main.gcno(gcov 所需檔案)
- 在指令行執行main.exe
 - 產生test.gcda檔案(gcov 所需data檔案)
- 在指令行執行gcov -b -c main.c
 - 顯示code coverage資訊
 - 產生main.c.gcda，紀錄每行程式碼執行的次數
- 使用記事本打開main.c.gcov，查看每行程式碼執行的次數

程式碼涵蓋度 (Code Coverage)

□ 指令行執行步驟

- 在e:\Test目錄，編輯main.c



```
main.c - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)
#include <stdio.h>
#include <math.h>
#include <assert.h>
double computeBMI(int kg, int height) {
    double M = height/100.0;
    double BMI = 0.0;
    if (kg<=0 || height<=0)
        return -1;
    BMI = round(100*kg/(M*M))/100; //四捨五入取兩位小數
    return BMI;
}
int main() {
    int kg = 52, height = 155;
    double expectedResult = 21.64f;
    double result = computeBMI(kg, height);
    assert(fabs(result-expectedResult)<0.0001);
    printf("Hi\n");
    return 0;
}
```

程式碼涵蓋度 (Code Coverage)

- 在指令行執行 `gcc -fprofile-arcs -ftest-coverage -o main main.c`
 - 開啟命令提示字元
 - e:
 - cd e:\Test
 - dir main.c
 - path=%path%;C:\Program Files (x86)\CodeBlocks\MinGW\bin
 - gcc -fprofile-arcs -ftest-coverage -o main main.c
 - dir
 - 確認產生 main.exe 、 main.gcno

程式碼涵蓋度 (Code Coverage)

- 在指令行執行 `gcc -fprofile-arcs -ftest-coverage -o main main.c`

```
命令提示字元

C:\Users\jykuo>e:

E:\>cd Test

E:\Test>dir main.c
磁碟區 E 中的磁碟是 DTAT3T
磁碟區序號: B613-6B32

E:\Test 的目錄

2020/02/16 上午 10:42      499 main.c
          1 個檔案          499 位元組
          0 個目錄  845,378,142,208 位元組可用

E:\Test>path=%path%;C:\Program Files (x86)\CodeBlocks\MinGW\bin

E:\Test>gcc -fprofile-arcs -ftest-coverage -o main main.c

E:\Test>dir
磁碟區 E 中的磁碟是 DTAT3T
磁碟區序號: B613-6B32

E:\Test 的目錄

2020/02/16 上午 11:22    <DIR>   .
2020/02/16 上午 11:22    <DIR>   ..
2020/02/16 上午 10:42          499 main.c
2020/02/16 上午 11:22          132,620 main.exe  -
2020/02/16 上午 11:22          796 main.gcno  -
          3 個檔案          133,915 位元組
          2 個目錄  845,378,142,208 位元組可用
```

程式碼涵蓋度 (Code Coverage)

- 在指令行執行 main.exe
 - 產生 main.gcda 檔案 (gcov 所需 data 檔案)

```
E:\Test>main.exe
Hi

E:\Test>dir
磁碟區 E 中的磁碟是 DTAT3T
磁碟區序號: B613-6B32

E:\Test 的目錄

2020/02/16 上午 11:27    <DIR>    .
2020/02/16 上午 11:27    <DIR>    ..
2020/02/16 上午 10:42            499  main.c
2020/02/16 上午 11:22          132,620  main.exe
2020/02/16 上午 11:27          244  main.gcda
2020/02/16 上午 11:22          796  main.gcno
                                4 個檔案          134,159 位元組
                                2 個目錄   845,378,142,208 位元組可用
```

程式碼涵蓋度 (Code Coverage)

- 在指令行執行 gcov -b -c main.c
 - 顯示 code coverage 資訊
 - 產生 main.c.gcov，紀錄每行程式碼執行的次數

```
E:\Test>gcov -b -c main.c
File 'main.c'
Lines executed:92.86% of 14
Branches executed:100.00% of 6
Taken at least once:50.00% of 6
Calls executed:66.67% of 3
Creating 'main.c.gcov'

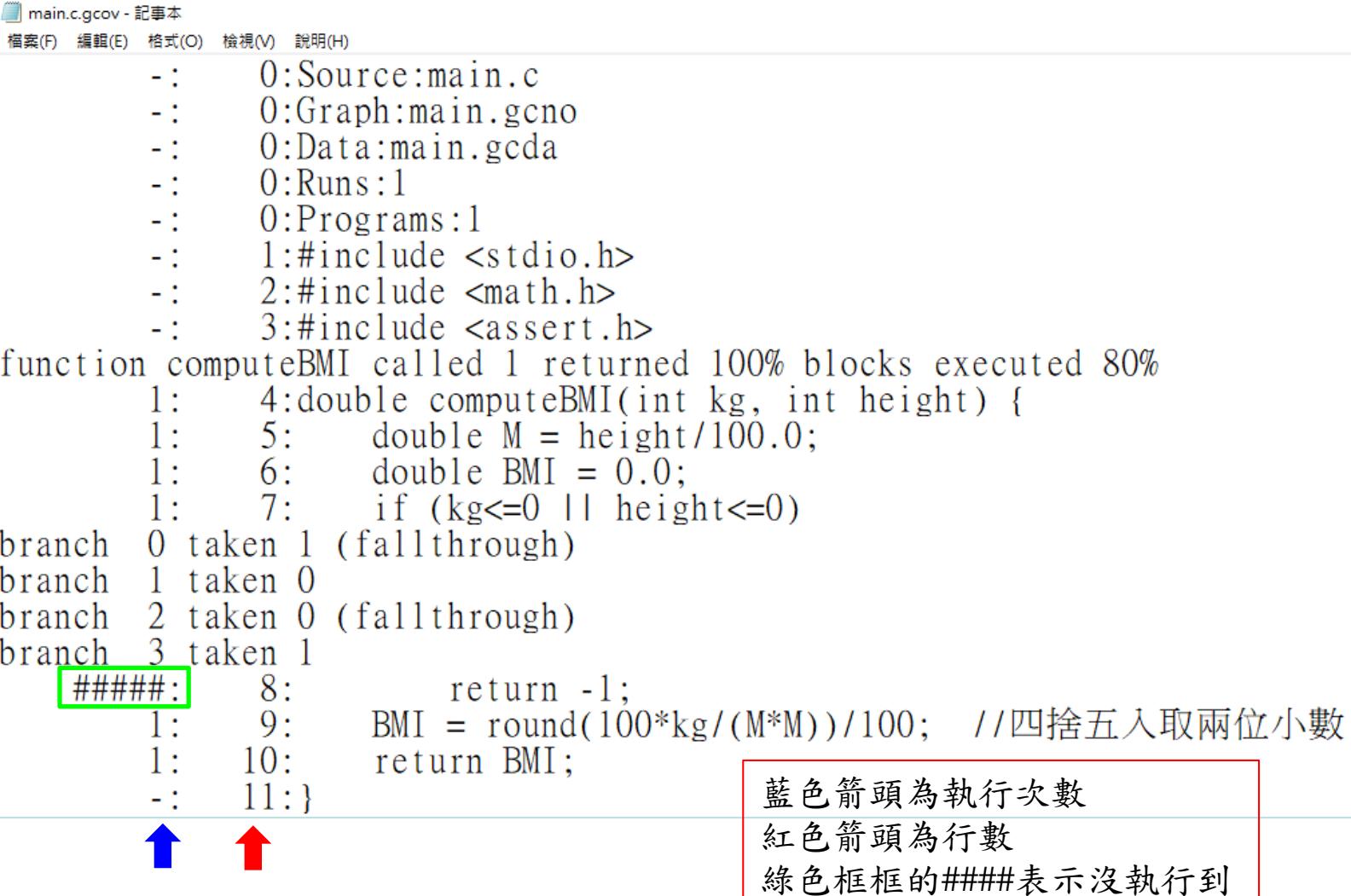
E:\Test>dir
磁碟區 E 中的磁碟是 DTAT3T
磁碟區序號: B613-6B32

E:\Test 的目錄

2020/02/16 上午 11:27 <DIR> .
2020/02/16 上午 11:27 <DIR> ..
2020/02/16 上午 10:42 499 main.c
2020/02/16 上午 11:27 1,315 main.c.gcov
2020/02/16 上午 11:22 132,620 main.exe
2020/02/16 上午 11:27 244 main.gcda
2020/02/16 上午 11:22 796 main.gcno
      5 個檔案 135,474 位元組
      2 個目錄 845,378,138,112 位元組可用
```

程式碼涵蓋度 (Code Coverage)

- 使用記事本打開main.c.gcov，查看每行程式碼執行的次數



```
main.c.gcov - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明(H)

-: 0:Source:main.c
-: 0:Graph:main.gcno
-: 0:Data:main.gcda
-: 0:Runs:1
-: 0:Programs:1
-: 1:#include <stdio.h>
-: 2:#include <math.h>
-: 3:#include <assert.h>
function computeBMI called 1 returned 100% blocks executed 80%
  1: 4:double computeBMI(int kg, int height) {
  1: 5:   double M = height/100.0;
  1: 6:   double BMI = 0.0;
  1: 7:   if (kg<=0 || height<=0)
branch 0 taken 1 (fallthrough)
branch 1 taken 0
branch 2 taken 0 (fallthrough)
branch 3 taken 1
  #####: 8:       return -1;
  1: 9:   BMI = round(100*kg/(M*M))/100; //四捨五入取兩位小數
  1: 10:  return BMI;
-: 11:}

藍色箭頭為執行次數
紅色箭頭為行數
綠色框框的#####表示沒執行到
```

程式碼涵蓋度 (Code Coverage)

- 使用記事本打開main.c.gcov，查看每行程式碼執行的次數

```
function main called 1 returned 100% blocks executed 83%
    1: 12: int main() {
    1: 13:     int kg = 52, height = 155;
    1: 14:     double expectedResult = 21.64f;
    1: 15:     double result = computeBMI(kg, height);
call 0 returned 1
    1: 16:     assert(fabs(result - expectedResult) < 0.0001);
branch 0 taken 0 (fallthrough)
branch 1 taken 1
call 2 never executed
    1: 17:     printf("Hi\n");
call 0 returned 1
    1: 18:     return 0;
    -: 19: }
    -: 20:
```

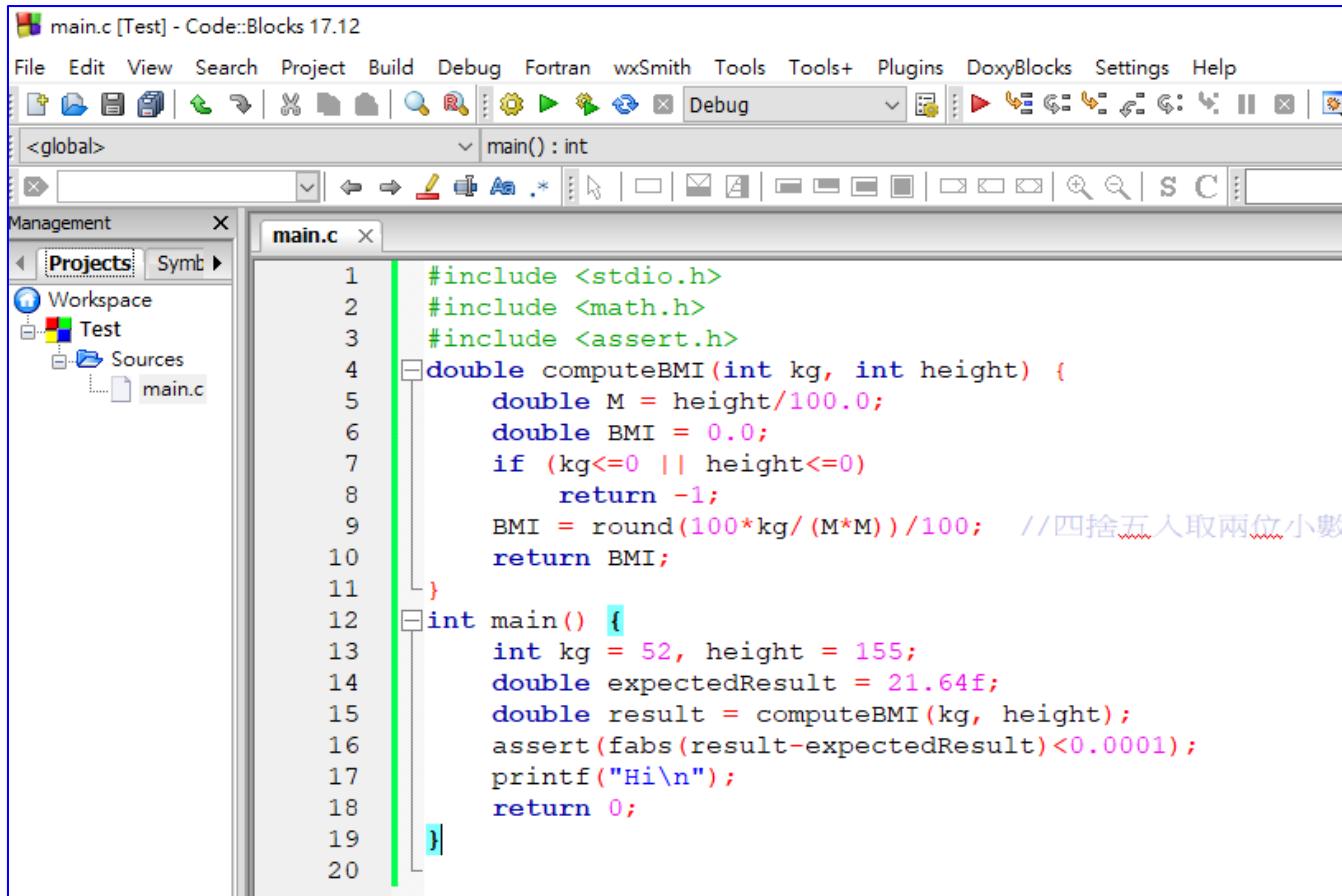
Exercise

- 如何使得涵蓋度 100%

程式碼涵蓋度 (Code Coverage)

□ 使用Code::Blocks

- 新增一個 Console Application 專案，編輯main.c

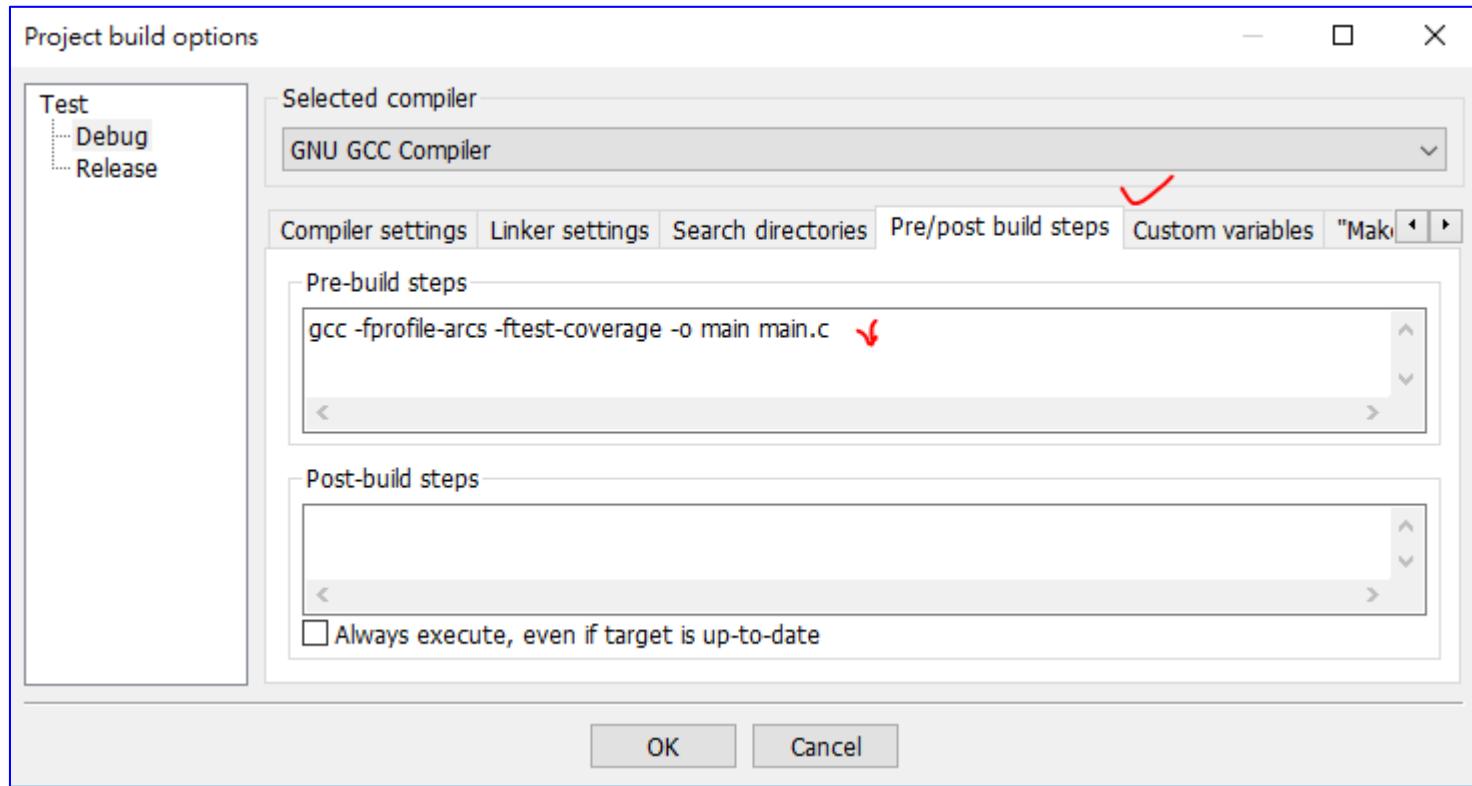


The screenshot shows the Code::Blocks IDE interface. The title bar reads "main.c [Test] - Code::Blocks 17.12". The menu bar includes File, Edit, View, Search, Project, Build, Debug, Fortran, wxSmith, Tools, Tools+, Plugins, Doxygen, Settings, and Help. The toolbar has various icons for file operations like Open, Save, and Build. The left sidebar has a "Management" section with "Projects" selected, showing a "Workspace" with a "Test" project containing a "Sources" folder with "main.c". The main code editor window displays the following C code:

```
#include <stdio.h>
#include <math.h>
#include <assert.h>
double computeBMI(int kg, int height) {
    double M = height/100.0;
    double BMI = 0.0;
    if (kg<=0 || height<=0)
        return -1;
    BMI = round(100*kg/(M*M)) /100; //四捨五入取兩位小數
    return BMI;
}
int main() {
    int kg = 52, height = 155;
    double expectedResult = 21.64f;
    double result = computeBMI(kg, height);
    assert(fabs(result-expectedResult)<0.0001);
    printf("Hi\n");
    return 0;
}
```

程式碼涵蓋度 (Code Coverage)

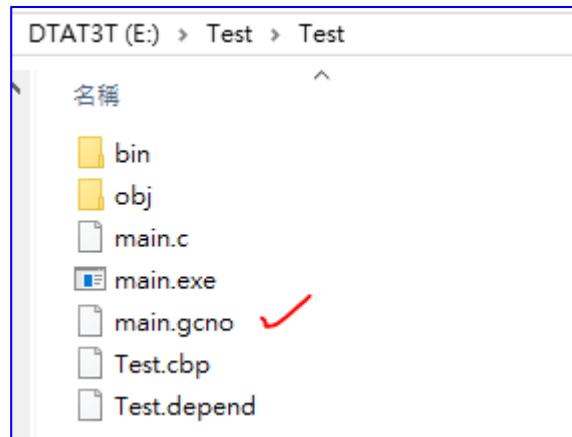
- 選單 project – build options – pre/post build steps，輸入
 - `gcc -fprofile-arcs -ftest-coverage -o main main.c`



- 按 OK
- 選單 File – save project

程式碼涵蓋度 (Code Coverage)

- 選單 Build – Build and run
- 查看檔案總管，產生main.gcno



Exercise

□ 測試計算BMI值的function

- BMI值計算公式： $BMI = \text{體重(公斤)} / \text{身高}^2(\text{公尺平方})$
- 例如：一個52公斤的人，身高是155公分，則BMI為：
- $52(\text{公斤}) / (1.55 * 1.55)(\text{公尺平方}) = 21.64412$
- 正常範圍為 $BMI = 18.5 \sim 24$
- 請設計一個 function，傳入身高(公分)、體重(公斤)，回傳BMI，取兩位小數四捨五入。
- 當 $BMI < 18.5$ ，輸出 -1。
- 當 $BMI > 24$ ，輸出 -2。
- 當身高或體重 < 0 ，輸出 0。

Exercise

□ 測試計算BMI值的function

```
#include <stdio.h>          // main.c
#include <math.h>
#include <assert.h>
double computeBMI(int kg, int height) {
    double BMI = 0.0, M = height/100.0;
    if (kg<=0 || height<=0)
        return 0;
    BMI = round(100*kg/(M*M))/100; //四捨五入取兩位小數
    if (BMI <18.5)
        return -1;
    if (BMI >24)
        return -2;
    return BMI;
}
```

```
int main() {
    int kg = 52, height = 155;
    double expectedResult = 21.64f;
    double result = computeBMI(kg, height);
    assert(fabs(result-expectedResult)<0.0001);
    printf("Hi\n");
    return 0;
}
```

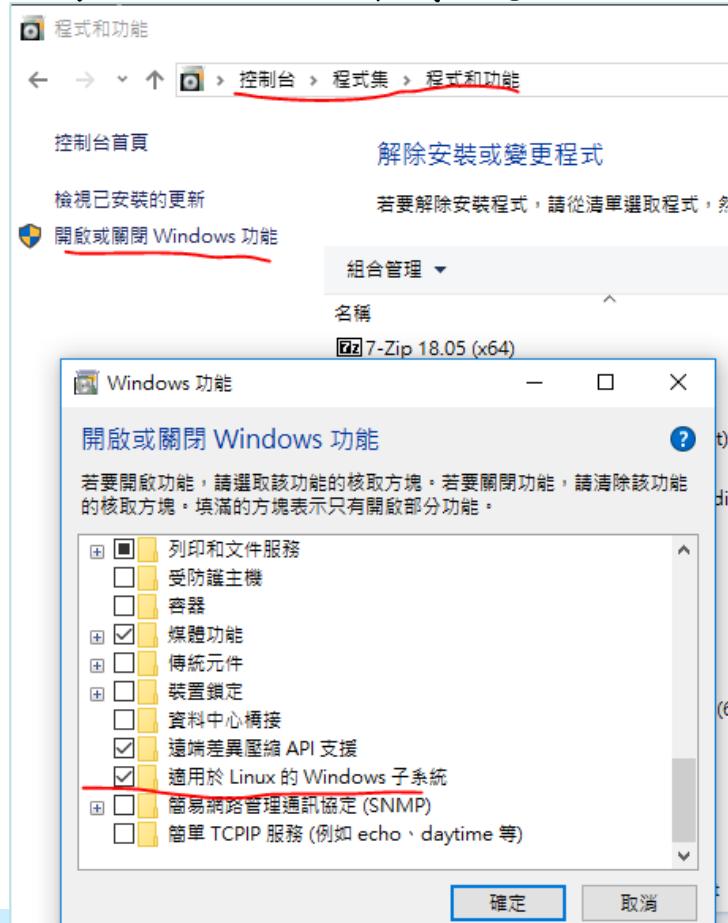
Exercise

- 紿予一組學生名單，包括名字、學號以及其三科成績，計算每位學生的平均分數，並將最高分與最低分的學生姓名分數印出。

Google Test

□ Windows 10 安裝 Ubuntu bash

- 控制台 -> 程式集 -> 程式和功能 -> 開啟或關閉 Windows 功能
- 勾選『適用於 Linux 的 Windows 子系統』



Google Test

- 搜尋 Microsoft Store 安裝 ubuntu ，安裝完成後按「啟動」



Google Test

□ 在 Ubuntu (win10-Ubuntu) 設定 Google test

- 左下角程式集，開啟 Ubuntu bash



- 一開始設定帳號、密碼

jykuo@DESKTOP-CE8JI9C: /usr/src/gtest

```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: jykuo
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

jykuo@DESKTOP-CE8JI9C:~\$ sudo apt-get update
[sudo] password for jykuo:

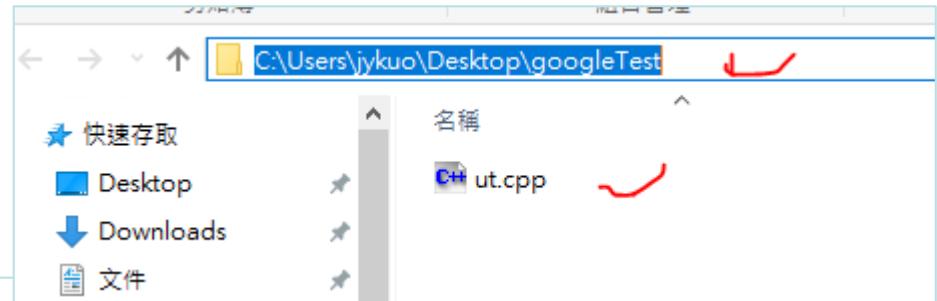
Google Test

- 在 Ubuntu bash 一行一行 輸入以下指令(需要十幾分鐘)
 - sudo apt-get update
 - sudo apt-get install g++
 - sudo apt-get install make
 - sudo apt-get install libgtest-dev
 - sudo apt-get install cmake
 - cd /usr/src/gtest
 - sudo cmake CMakeLists.txt
 - sudo make
 - sudo cp *.a /usr/lib

Google Test

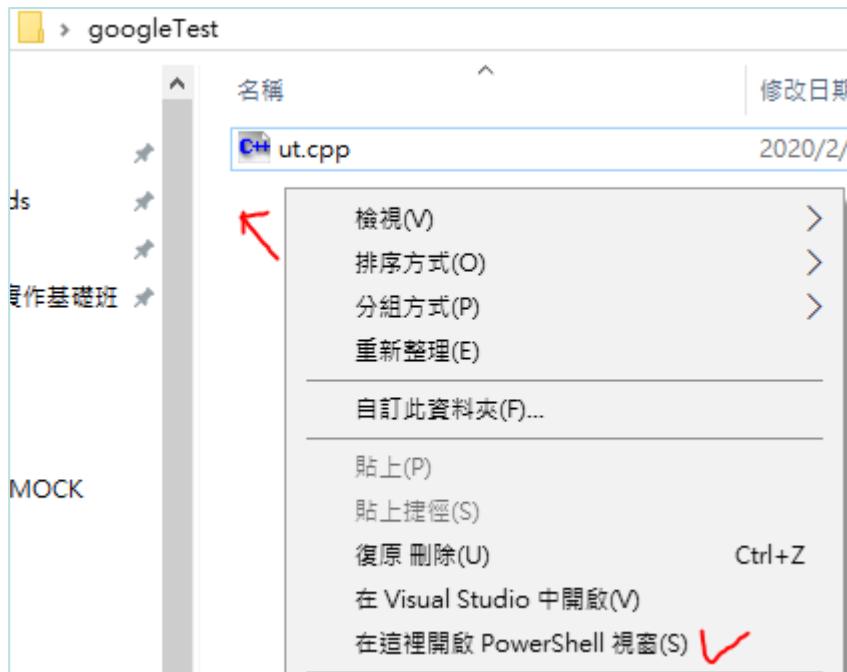
- 在 Windows 10 桌面新增googleTest目錄
- 在目錄內編輯 ut.cpp

```
#include <cstdlib>
#include <gtest/gtest.h>
int mul( int a , int b ) {
    return a * b ;
}
TEST( multest , HandleNoneZeroInput ) {
    ASSERT_EQ( 21 , mul( 3 , 7 ) );
    ASSERT_EQ( -24 , mul( -6 , 4 ) );
}
int main( int argc , char **argv ){
    testing :: InitGoogleTest( &argc , argv );
    return RUN_ALL_TESTS();
}
```



Google Test

- 在 桌面googleTest目錄，按shift + 滑鼠右鍵
 - 選開啟powerShell

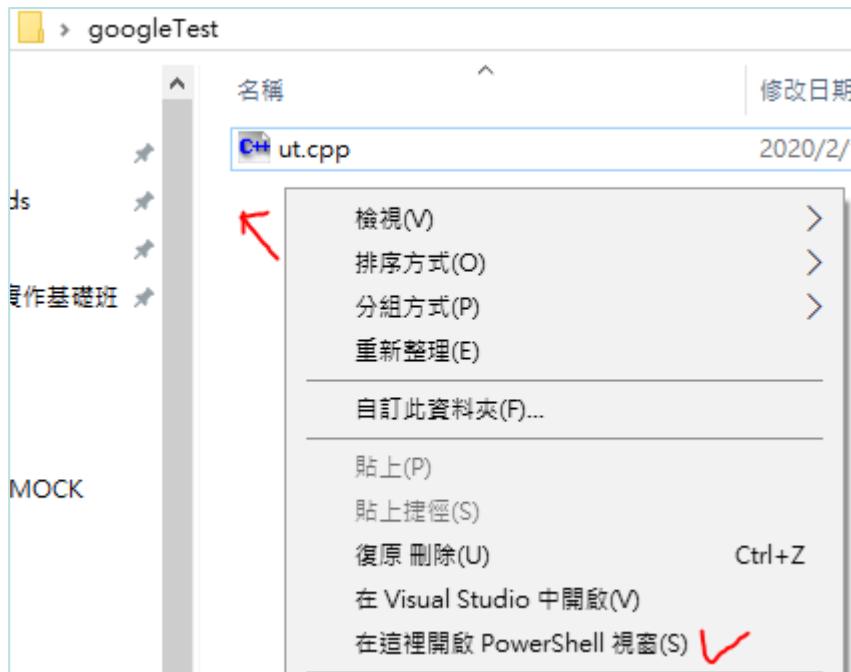


```
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest
```

```
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$
```

Google Test

- 在 桌面googleTest目錄，按shift + 滑鼠右鍵
 - 選開啟powerShell，在指令行輸入bash，進入Linux Ubuntu。



```
jykuo@DESKTOP-CE8JI9C: /mnt/c/Users/jykuo/Desktop/googleTest
PS C:\Users\jykuo\Desktop\googleTest> bash ✓
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$
```

Google Test

□ 輸入指令，編譯、執行、看code coverage報表

- g++ -pg -fprofile-arcs -ftest-coverage ut.cpp -o ut -lgtest -lpthread
- ./ut

```
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$ g++ -pg -fprofile-arcs -ftest-coverage ut.cpp -o ut -lgtest -lpthread
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$ ./ut
[=====] Running 1 test from 1 test case.
[-----] Global test environment set-up.
[-----] 1 test from multest
[RUN    ] multest.HandleNoneZeroInput
[      OK] multest.HandleNoneZeroInput (0 ms)
[-----] 1 test from multest (1 ms total)

[-----] Global test environment tear-down
[-----] 1 test from 1 test case ran. (2 ms total)
[PASSED ] 1 test.

jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$
```

- gcov -c -b ut.cpp

```
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$ gcov -c -b ut.cpp
File 'ut.cpp'
Lines executed:100.00% of 9
Branches executed:57.89% of 38
Taken at least once:28.95% of 38
Calls executed:53.19% of 47
Creating 'ut.cpp.gcov'
```

Google Test - 安裝html報表

□ 安裝 lcov , sudo apt-get install -y lcov

```
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$ sudo apt-get install -y lcov
[sudo] password for jykuo:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

□ 產生html報表 ,

- lcov -c -o ut.info -d . --rc lcov_branch_coverage=1

```
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$ lcov -c -o ut.info -d . --rc lcov_branch_coverage=1
Capturing coverage data from .
Found gcov version: 7.4.0
Scanning . for .gcda files ...
Found 1 data files in .
Processing ut.gcda
Finished .info-file creation
```

- genhtml ut.info -o report --branch-coverage

```
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$ genhtml ut.info -o report --branch-coverage
Reading data file ut.info
Found 8 entries.
Found common filename prefix "/usr/include"
Writing .css and .png files.
Generating output.
Processing file /mnt/c/Users/jykuo/Desktop/googleTest/ut.cpp
Processing file c++/7/iostream
```

Google Test - 安裝html報表

- 在googleTest\report 目錄點選index.html



- ## □ 產生html報表

Current view: top level - mnt/c/Users/jykuo/Desktop/google Test - ut.cpp (source / functions)

Test: ut.info

Date: 2020-03-31 15:32:43

	Hit	Total	Coverage
Lines:	20	20	100.0 %
Functions:	8	8	100.0 %
Branches:	28	88	31.8 %

Google Test - 分開測試檔案

□ 編輯三個檔案

```
// bmi.h  
double computeBMI(int kg, int height);
```

```
//bmi.cpp  
#include <math.h>  
double computeBMI(int kg, int height) {  
    double BMI = 0.0, M = height/100.0;  
    if (kg<=0 || height<=0)  
        return 0;  
    BMI = round(100*kg/(M*M))/100; //四捨五入取兩位小數  
    if (BMI <18.5)  
        return -1;  
    if (BMI >24)  
        return -2;  
    return BMI;  
}
```

Google Test - 分開測試檔案

□ 編輯三個檔案

```
//ut.cpp
//#include <cstdlib>
#include <gtest/gtest.h>
#include "bmi.h"

TEST( multest , HandleNoneZeroInput ) {
    ASSERT_EQ( 0 , computeBMI( 0 ,0 ) );
    ASSERT_EQ( 0 , computeBMI( 100 , 0) );
    ASSERT_EQ( -2 , computeBMI( 52 , 100) );
    ASSERT_EQ( -1 , computeBMI( 42 , 155) );
    ASSERT_EQ( 21.64 , computeBMI( 52 , 155) );
}

int main( int argc , char **argv ){
    testing :: InitGoogleTest( &argc , argv );
    return RUN_ALL_TESTS();
}
```

Google Test - 分開測試檔案

- g++ -pg -fprofile-arcs -ftest-coverage bmi.cpp ut.cpp -o ut -lgtest -lpthread
- ./ut

```
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$ g++ -pg -fprofile-arcs -ftest-coverage bmi.cpp ut.cpp -o ut -lgtest -lpthread
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$ ./ut
[=====] Running 1 test from 1 test case.
[-----] Global test environment set-up.
[-----] 1 test from multest
[RUN     ] multest.HandleNoneZeroInput
[OK      ] multest.HandleNoneZeroInput (0 ms)
[-----] 1 test from multest (5 ms total)

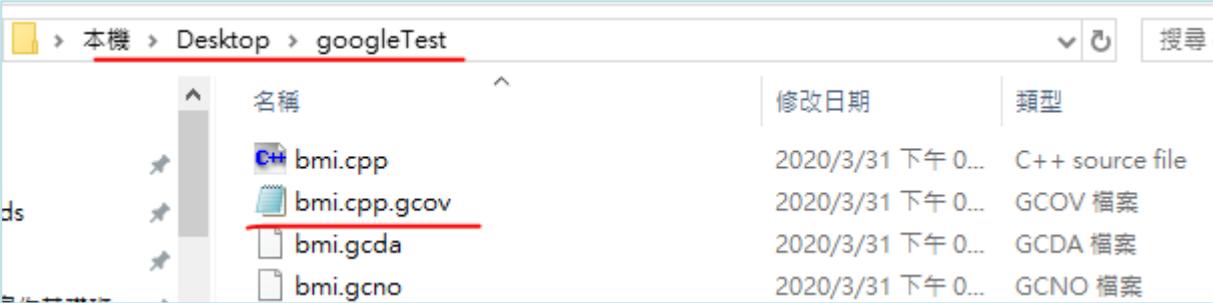
[-----] Global test environment tear-down
[=====] 1 test from 1 test case ran. (9 ms total)
[PASSED  ] 1 test.
```

- gcov -c -b ut.cpp
- gcov -c -b bmi.cpp

```
jykuo@DESKTOP-CE8JI9C:/mnt/c/Users/jykuo/Desktop/googleTest$ gcov -c -b bmi.cpp
File 'bmi.cpp'
Lines executed:100.00% of 10
Branches executed:100.00% of 8
Taken at least once:100.00% of 8
No calls
Creating 'bmi.cpp.gcov'
```

Google Test - 分開測試檔案

- 在 googleTest 目錄，檢視 bmi.cpp.gcov



bmi.cpp.gcov

名稱	修改日期	類型
bmi.cpp	2020/3/31 下午 0...	C++ source file
bmi.cpp.gcov	2020/3/31 下午 0...	GCOV 檔案
bmi.gcda	2020/3/31 下午 0...	GCDA 檔案
bmi.gcno	2020/3/31 下午 0...	GCNO 檔案

```
1      -: 0:Source:bmi.cpp
2      -: 0:Graph:bmi.gcno
3      -: 0:Data:bmi.gcda
4      -: 0:Runs:1
5      -: 0:Programs:1
6      -: 1:#include <math.h>
7 function _Z10computeBMIii called 5 returned 100% blocks executed 100%
8      5:    2:double computeBMI(int kg, int height) {
9      5:      3:      double BMI = 0.0, M = height/100.0;
10     5:      4:      if (kg<=0 || height<=0)
11 branch 0 taken 4 (fallthrough)
12 branch 1 taken 1
13 branch 2 taken 1 (fallthrough)
14 branch 3 taken 3
15     2:      5:          return 0;
16     3:      6:          BMI = round(100*kg/ (M*M)) /100; //四捨五入取兩位小數
17     3:      7:          if (BMI <18.5)
18 branch 0 taken 1 (fallthrough)
19 branch 1 taken 2
20     1:      8:          return -1;
21     2:      9:          if (BMI >24)
22 branch 0 taken 1 (fallthrough)
23 branch 1 taken 1
24     1:      10:         return -2;
25     1:      11:         return BMI;
26     -: 12:{}
```

Google Test - 分開測試檔案

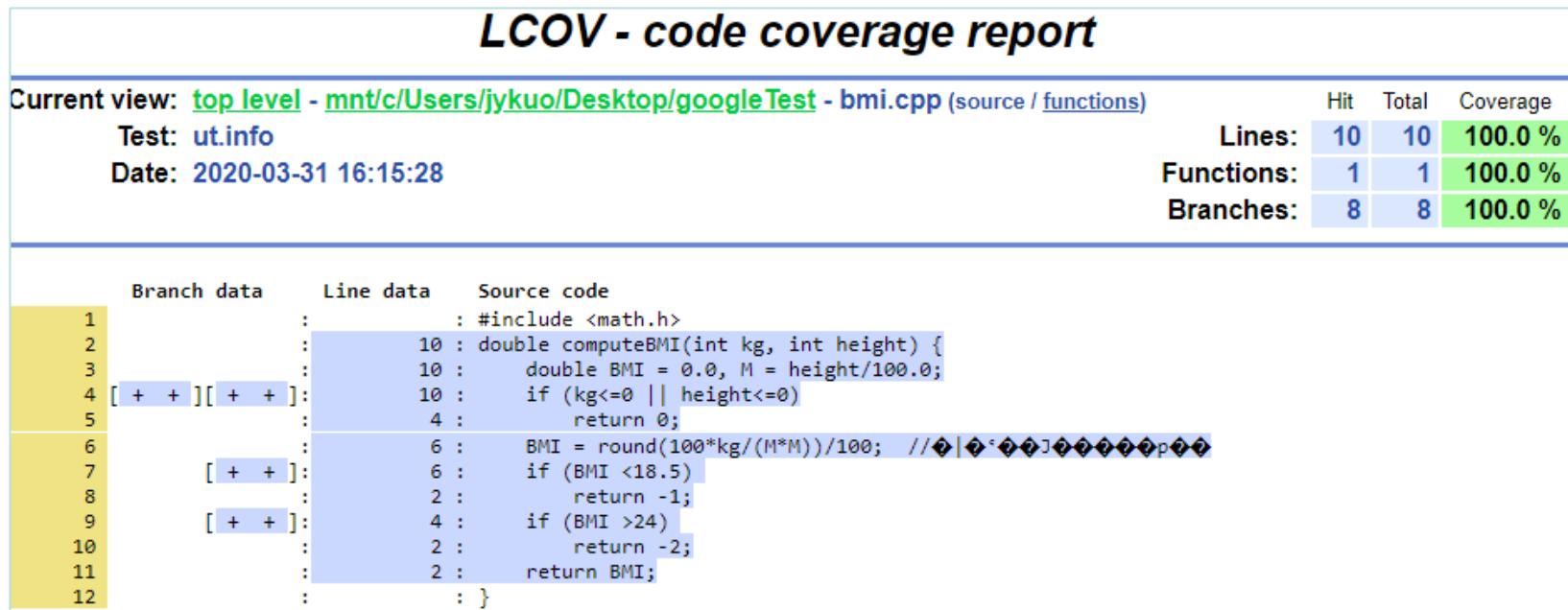
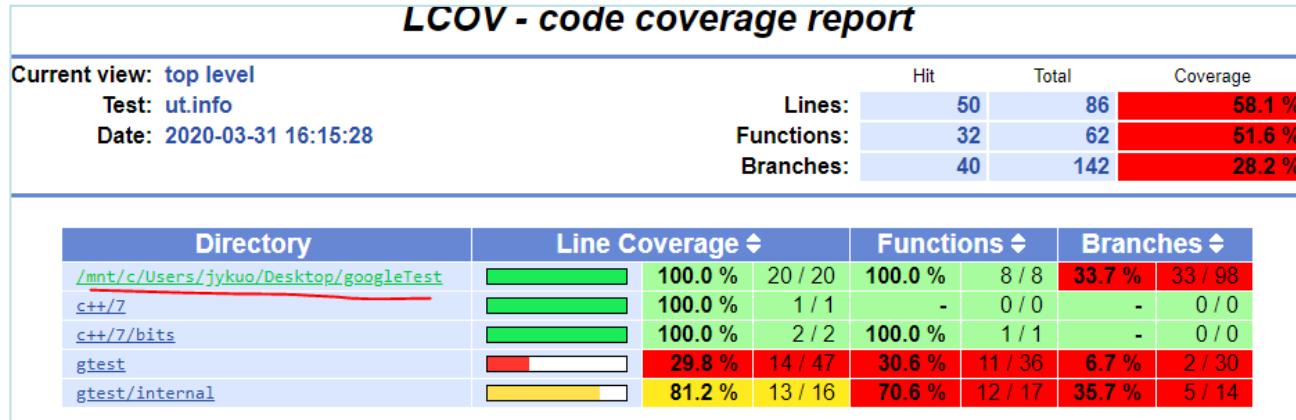
□ 產生html報表

- lcov -c -o ut.info -d . --rc lcov_branch_coverage=1
- genhtml ut.info -o report --branch-coverage
- 在 googleTest\report 目錄點選 index.html



Google Test - 分開測試檔案

□ 產生html報表



設計測試案例(Test case)方法

□ 白箱測試(white box testing)

- 檢視程式碼，設計測試案例，執行所有程式碼可能路徑
- 評估測試案例的涵蓋度準則(coverage criteria)
 - Line coverage (statement coverage)，測試案例能執行所有程式碼行
 - branch coverage，測試案例能執行所有條件判斷狀況，一個條件判斷有二種狀況(True, False)，兩個條件組合有四種狀況(True/True, True/False, False/True, False False)

□ 黑箱測試(black box testing)

- 檢視題目需求，設計測試案例，執行題目需求可能狀況、功能
- 設計測試案例方法
 - 分類功能、規則、輸入條件，例如BMI太高或太低情況
 - 邊界值分析，考慮邊界情況，例如BMI太高或太低剛好邊界情況
 - 負向，不合規範的情況，例如負數體重、身高

黑箱測試方法 Homework

□ 題目需求

- 辨識輸入符號，Identifier是C語言的變數
- Input: rate R2D2 -2 time 55566 0.23 -1.2 #
- Output:
 - rate - Identifier
 - R2D2 - Identifier
 - -22 - Negative Integer
 - 555666 – Positive Integer
 - 0.23 - Positive Floating
 - -1.2 - Negative Floating
- Exercise: 考慮負向情況的測試案例

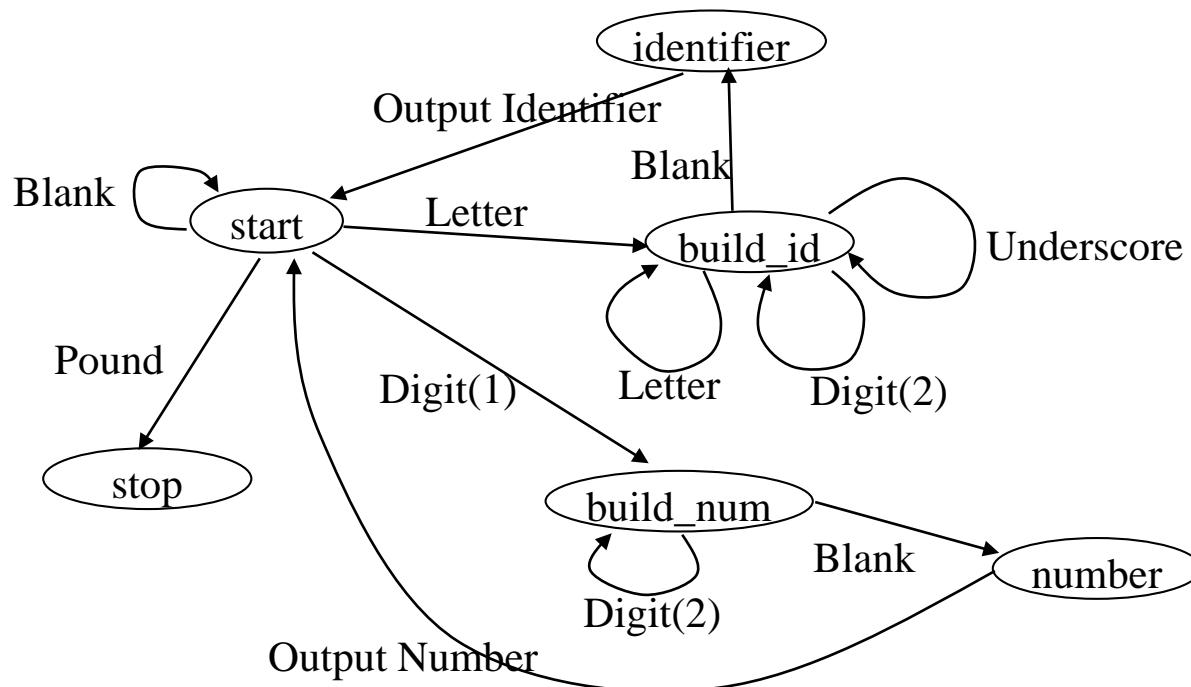
黑箱測試方法 Homework

□ 使用有限狀態機解題

○ 簡化題目

➤ 只處理identifier, Number

Digit(1)	1, .., 9
Pound	#
Blank	empty
Letter	A,.., Z, a, .., z
Digit(2)	0, 1, .., 9
Underscore	_



黑箱測試方法 Homework

□ `typedef` 自訂資料型別 data type

- 自訂資料型別，命名以`_t`結束，可以以此宣告變數

```
typedef int myInt_t;  
myInt x1=0, x2=3;  
x1 = x2 + 4;
```

□ 列舉型別 (enumeration)，使程式易於閱讀、了解、修改

- 列舉所宣告的變數，有哪些值，第一個符號其值預設為0，後面依序加1
- `state_t`，是自訂資料型別，值有 `start`, `build_id`, `build_num`，其值分別為 0, 1, 2。

```
typedef enum { start, build_id, build_num } state_t;  
state_t x1, x2;  
x1 = start;  
x2 = build_num;  
printf("%d, %d\\", x1, x2); //印出 0, 1
```

黑箱測試方法 Homework

```
#include <stdio.h>          // FMS.c
#include <ctype.h>
typedef enum {start, build_id, build_num, build_invalid, identifier, number, invalid, stop} state_t;
state_t getNextState(state_t current_state, char ch) {
    if (current_state == start) {
        if (ch == ' ') return start;
        else if (isalpha(ch)) return build_id;
        else if (isdigit(ch)) return build_num;
        else if (ch=='#') return stop;
    }
    if (current_state == build_id) {
        if (isalpha(ch)||isdigit(ch)|| (ch=='_')) return build_id;
        else if (ch==' ') return identifier;
    }
    if (current_state == build_num) {
        if (isdigit(ch)) return build_num;
        else if (ch==' ') return number;
    }
    else return -1;
}
```

黑箱測試方法 Homework

- 使用有限狀態機解題，只處理identifier, Number

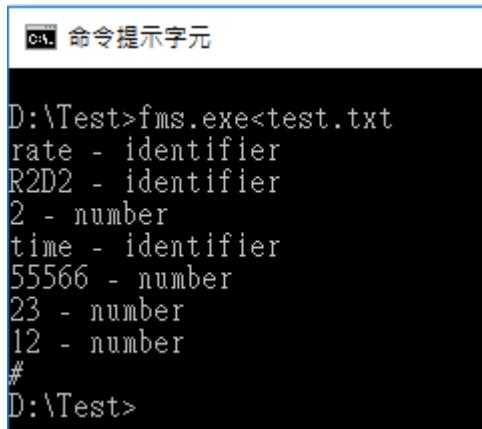
```
void FMS() {  
    char input_char;  
    state_t current_state;  
    current_state = start;  
    do {  
        if (current_state==identifier) {  
            printf(" - identifier\n");  
            current_state = start;  
        }  
        else if (current_state==number) {  
            printf(" - number\n");  
            current_state = start;  
        }  
        scanf("%c", &input_char);  
        if (input_char != ' ') printf("%c", input_char);  
        current_state = getNextState(current_state, input_char);  
    } while (current_state!=stop);  
}  
int main() {    FMS();    return 0; }
```

黑箱測試方法 Homework

- 設計 test.txt

```
rate R2D2 2 time 55566 23 12 #
```

- 編譯 fms.c
- 在 cmd 模式，執行 fms.c 測試



```
D:\Test>fms.exe<test.txt
rate - identifier
R2D2 - identifier
2 - number
time - identifier
55566 - number
23 - number
12 - number
#
D:\Test>
```

APPENDIX Google Test白箱測試

```
#include <stdio.h>          // fms.cpp
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
typedef enum {start, build_id, build_num, build_invalid, identifier, number, invalid, stop} state_t;
state_t getNextState(state_t current_state, char ch) {
    if (current_state == start) {
        if (ch == ' ')
            return start;
        else if (isalpha(ch))
            return build_id;
        else if (isdigit(ch)) return build_num;
        else if (ch=='#') return stop;
    }
    if (current_state == build_id) {
        if (isalpha(ch)||isdigit(ch)|| (ch=='_'))
            return build_id;
        else if (ch==' ')
            return identifier;
    }
    if (current_state == build_num) {
        if (isdigit(ch))
            return build_num;
        else if (ch==' ')
            return number;
    }
    else return stop;
}
```

APPENDIX Google Test白箱測試

```
char * FMS(char *s) { // fms.cpp
    char out[80];
    char *output = (char*) malloc(sizeof(char)*500);
    char input_char;
    int index=0;
    state_t current_state;
    current_state = start;
    strcpy(output,"");
    do {
        if (current_state==identifier) {
            current_state = start;
            sprintf(out," - identifier\n");
            strcat(output, out);
        }
        else if (current_state==number) {
            current_state = start;
            sprintf(out," - number\n");
            strcat(output, out);
        }
        //scanf("%c", &input_char);
        input_char = s[index++];
        if ((input_char != ' ')&&(input_char != '#')) {
            sprintf(out, "%c", input_char);
            strcat(output, out);
        }
        current_state = getNextState(current_state, input_char);
    } while (current_state!=stop);
    return output;
}
```

黑箱測試方法 Homework

run.sh

```
#!/bin/bashg++ -pg -fprofile-arcs -ftest-coverage fms.cpp ut.cpp -o ut -lgtest -lpthread  
.ut  
gcov -c -b fms.cpp  
lcov -c -o ut.info -d . --rc lcov_branch_coverage=1  
genhtml ut.info -o report --branch-coverage
```

```
//fms.h  
char * FMS(char *s);
```

```
// ut.cpp  
#include <cstdlib>  
#include <gtest/gtest.h>  
#include "fms.h"  
TEST( fmsTest , HandleInput ) {  
    char s[]="rate R2D2 2 time 55566 23 12 #";  
    char expOutput[] = "rate - identifier\nR2D2 - identifier\n2 - number\n time - identifier\n55566 - number\n23 - number\n12 - number\n";  
    ASSERT_STREQ(expOutput, FMS(s));  
}  
int main( int argc , char **argv ){  
    testing :: InitGoogleTest( &argc , argv );  
    return RUN_ALL_TESTS();  
}
```

黑箱測試方法 Homework

□ ./run.sh

```
jykuo@DESKTOP-CE8JI9C:/mnt/d/Test/googleTest$ ./run.sh
[=====] Running 1 test from 1 test case.
[-----] Global test environment set-up.
[-----] 1 test from fmsTest
[ RUN   ] fmsTest.HandleInput
[       ] fmsTest.HandleInput (0 ms)
[-----] 1 test from fmsTest (14 ms total)

[-----] Global test environment tear-down
[-----] 1 test from 1 test case ran. (381 ms total)
[ PASSED ] 1 test.

File 'fms.cpp'
Lines executed:95.12% of 41
Branches executed:100.00% of 40
Taken at least once:87.50% of 40
Calls executed:100.00% of 1
```

APPENDIX Google Test白箱測試

LCOV - code coverage report

Current view: top level - mnt/d/Test/googleTest - fms.cpp (source / functions)			Hit	Total	Coverage
Test:	ut.info		Lines:	39	41
Date:	2020-04-02 17:09:09		Functions:	2	2
			Branches:	35	40

Branch data	Line data	Source code
	1 :	#include <stdio.h>
	2 :	#include <stdlib.h>
	3 :	#include <ctype.h>
	4 :	#include <string.h>
	5 :	typedef enum
	6 :	{start, build_id, build_num, build_invalid,
	7 :	identifier, number, invalid, stop}
	8 :	state_t;
	9 :	31 : state_t getNextState(state_t current_state, char ch) {
[+ +]:	31 :	if (current_state == start) {
[+ +]:	9 :	if (ch == ' ')
	1 :	return start;
[+ +]:	8 :	else if (isalpha(ch))
	3 :	return build_id;
[+ +][+ +]:	5 :	else if (isdigit(ch)) return build_num;
[+ -]:	1 :	else if (ch=='#') return stop;
	17 :	}
[+ +]:	22 :	if (current_state == build_id) {
[+ +][+ +]:	12 :	if (isalpha(ch) isdigit(ch) ((ch=='_'))
[- +]:	9 :	return build_id;
[+ -]:	3 :	else if (ch==' ')
	3 :	return identifier;
	23 :	}
[+ -]:	10 :	if (current_state == build_num) {
[+ +][+ +]:	10 :	if (isdigit(ch))
	6 :	return build_num;
[+ -]:	4 :	else if (ch==' ')
	4 :	return number;
	30 :	}
	0 :	else return stop;
	0 :	}
	1 :	char * FMS(char *s) {
	33 :	char out[80];
	34 :	char *output = (char*) malloc(sizeof(char)*500);
	35 :	char input_char;
	36 :	int index=0;
	37 :	state_t current_state;
	38 :	current_state = start;
	39 :	strcpy(output,"");
	40 :	do {
[+ +]:	31 :	if (current_state==identifier) {
	3 :	current_state = start;
	3 :	sprintf(out," - identifier\n");
	3 :	strcat(output, out);
	45 :	}
[+ +]:	28 :	else if (current_state==number) {
	4 :	current_state = start;
	4 :	sprintf(out," - number\n");
	4 :	strcat(output, out);
	50 :	}

Exercise

```
int main() {  
    int kg = 52, height = 155;  
    double expectedResult = 21.64f;  
    double result = computeBMI(kg, height);  
    assert(fabs(result - expectedResult) < 0.0001);  
    assert(computeBMI(0,0) == 0);  
    assert(computeBMI(100,0) == 0);  
    assert(computeBMI(52,100) == -2);  
    assert(computeBMI(42,155) == -1);  
    printf("Hi\n");  
    return 0;  
}
```

6rte - Invalid
r_yg - Identifier
t#ee - Invalid

計算機程式設計

C語言 Recursive

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

遞迴概念

- 遞迴(Recursive)是程式設計重要觀念
 - 遞迴函式是使用遞迴觀念建立的函式，可讓程式碼變簡潔。
 - 設計遞迴函式需小心，否則易掉入無窮迴圈陷阱。
- 遞迴設計
 - 問題要可拆解成規模較小但性質和原問題完全一樣的問題
 - 範圍逐漸縮小到一個終止條件。
- 遞迴函式特性
 - 每次呼叫時，都可使問題範圍逐漸縮小。
 - 有一個終止條件，以便結束遞迴執行，否則會有無窮迴圈。

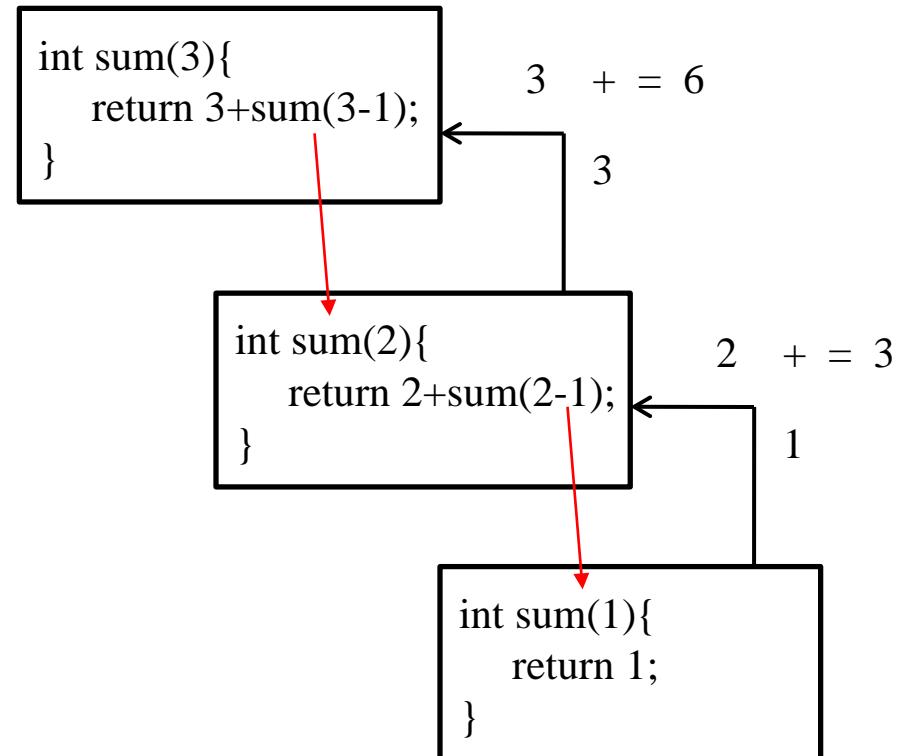
	優點	缺點
遞迴	程式簡潔，節省記憶體空間	參數的堆疊存取較費時
非遞迴	節省執行時間	程式較長，浪費記憶體空間

疊代(iterative)與遞迴(recursive)

由 1 加到 3

```
01 int sum(int n) {  
02     int i;  
03     int tmp=0;  
04     for (i=0; i<n; i++)  
05         tmp = tmp + (i+1);  
06     return tmp;  
07 }
```

```
01 int sum(int n) {  
02     if (n==1)  
03         return 1;  
04     else  
05         return n+sum(n-1);  
06 }
```



階層

□ 計算階層

- 計算 $4!$ ， $n > 0$ ，使用第2條計算

$$n! = \begin{cases} 1 & n=0 \\ n*(n-1)*(n-2)*...*1 & n>0 \end{cases}$$

$$○ 4! = 4 * 3 * 2 * 1 = 24$$

$$○ 將 4! 的計算分解成子問題，4! = 4 * (4-1)! = 4 * 3!$$

○ 將子問題 $3!$ 繼續分解，

$$\triangleright 3! = 3 * (3-1)! = 3 * 2! ,$$

$$\triangleright 2! = 2 * (2-1)! = 2 * 1! ,$$

$$\triangleright 1! = 1 * (1-1)! = 1 * 0! = 1 * 1 = 1$$

○ 最後知道 $1!$ 值，可計算出 $2! \sim 4!$ 的值

$$\triangleright 2! = 2 * (2-1)! = 2 * 1! = 2 ,$$

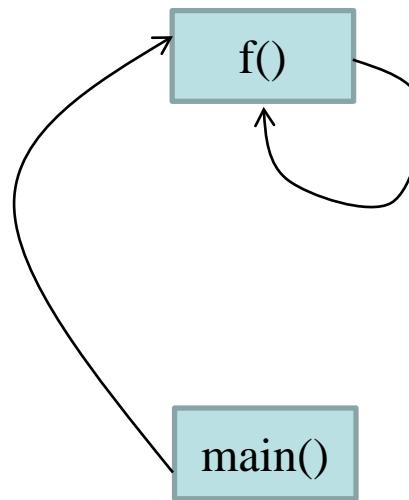
$$\triangleright 3! = 3 * (3-1)! = 3 * 2! = 3 * 2 = 6 ,$$

$$\triangleright 4! = 4 * (4-1)! = 4 * 3! = 24$$

遞迴種類

- 依呼叫遞迴函式位置，分為兩種：
 - 直接遞迴(Direct Recursion)：自己呼叫自己。

```
01 #include <stdio.h>
02 int f(int x) {
03     if (x<0) return 1;
04     return f(x-1)+1;
05 }
06 int main() {
07     printf("%d",f(5));
08     return 0;
09 }
```



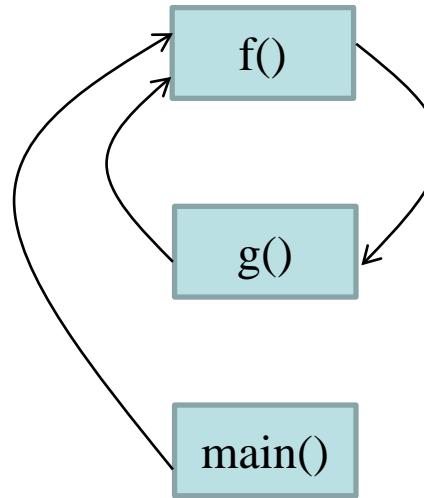
執行順序?執行結果?

遞迴種類

□ 依呼叫遞迴函式位置，分為兩種：

- 間接遞迴(Indirect Recursion)：至少需2個函式a()和b()，函式a()呼叫函式b()；函式b()呼叫函數a()。

```
01 #include <stdio.h>
02 int f(int x) {
03     if (x<0) return 1;
04     return g(x-1)+1;
05 }
06 int g(int y) {
07     if (y<0) return 2;
08     return f(y-1)+1;
09 }
10 int main() {
11     printf("%d",f(5));
12     return 0;
13 }
```

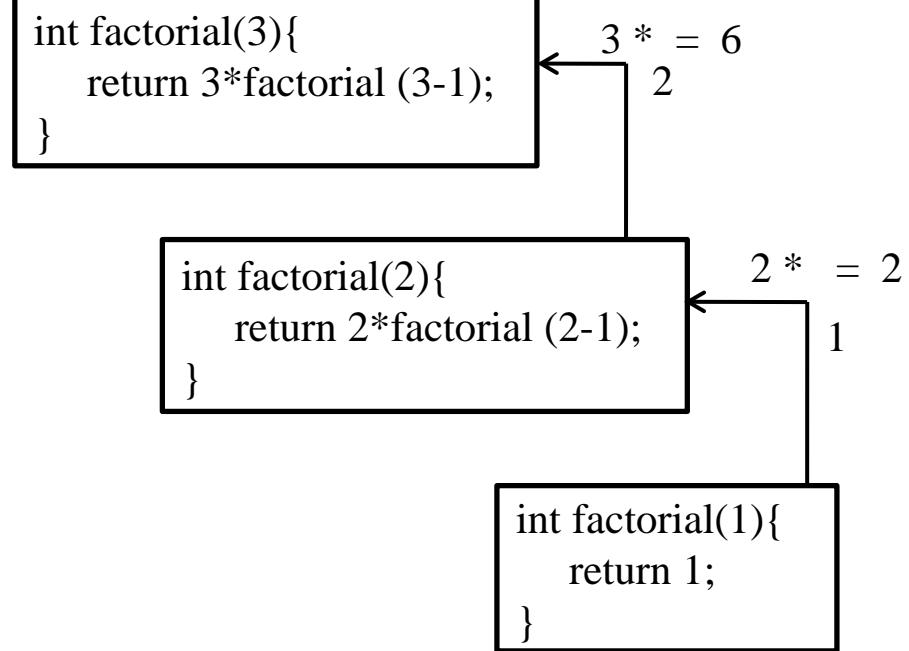


執行順序? 執行結果?

階層

□ 計算階層

```
01 int factorial(int n) {  
02     if ( n == 1)  
03         return 1;  
04     else  
05         return n * factorial(n-1);  
06 }
```



最大公因數

□ 最大公因數(Greatest Common Divisor, GCD)

- 某幾個整數共有因數中最大的一個。
- 求兩個整數最大公因數：各自列出因數，找出最大公因數。

□ 39, 27的最大公因數 $\Rightarrow 3$

- 39: 1, 3, 13, 39
- 27: 1, 3, 9, 27

□ 輾轉相除法

- $39 \% 27 \Rightarrow 12$
- $27 \% 12 \Rightarrow 3$
- $12 \% 3 \Rightarrow 0$

最大公因數

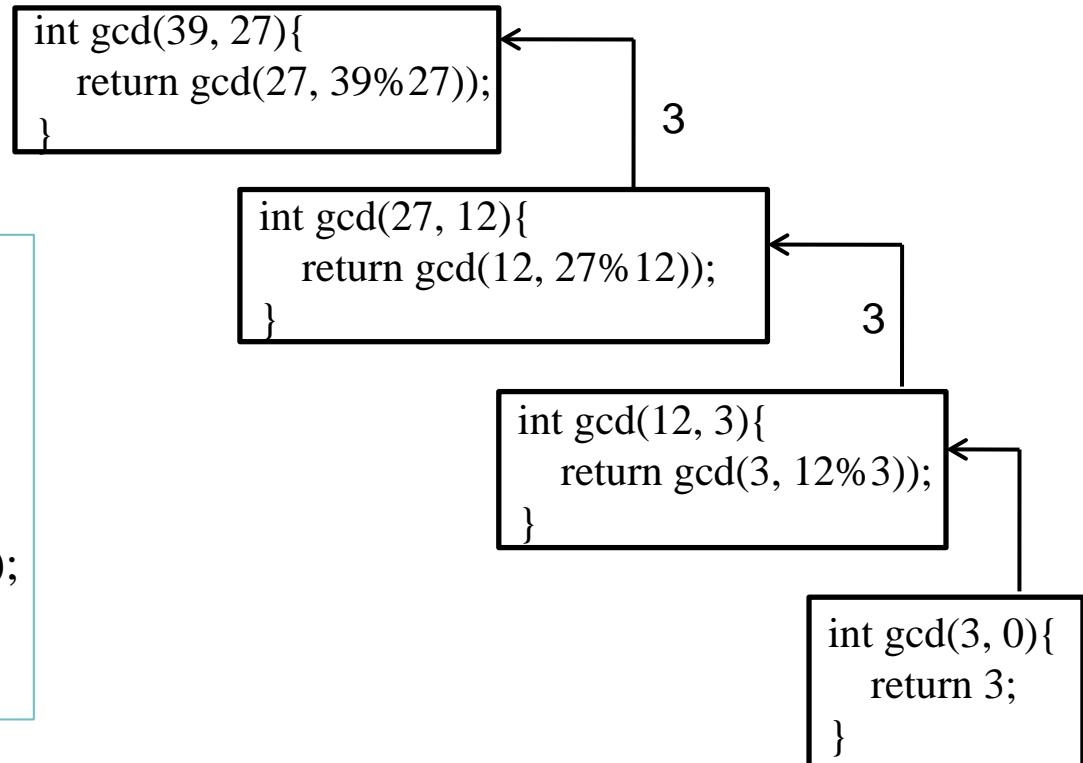
□ 輾轉相除法

○ $39 \% 27 \Rightarrow 12$

○ $27 \% 12 \Rightarrow 3$

○ $12 \% 3 \Rightarrow 0$

```
01 int gcd(int m, int n) {  
02     if(n == 0) {  
03         return m;  
04     }  
05     else {  
06         return gcd(n, m % n);  
07     }  
08 }
```



Exercise 登山

- 小英要為登玉山準備，練習爬 101 大樓的樓梯。
 - 可以一步踏一階、二階、最長跨三階。因此有很多種爬階方法。
 - 要爬到第一階，有一種爬法: $f(1)=1$
 - (1)一次一階。
 - 要爬到第二階，有二種爬法: $f(2)=2$
 - (1)一次一階，爬二次。
 - (2)一次爬二階
 - 要爬到第三階，有四種爬法: $f(3)=2$
 - (1)一次一階，爬三次。
 - (2)先爬一階，再爬二階
 - (3)先爬二階，再爬一階
 - (4)一次三階。

Exercise 登山

- 小英要為登玉山準備，練習爬 101 大樓的樓梯。
 - 要爬到第四階，有七種爬法。
 - (1) 爬到第三階後，再一次爬一階， $f(3) = 4$
 - (2) 爬到第二階後，再一次爬二階， $f(2) = 2$
 - (3) 爬到第一階後，再一次爬三階， $f(1) = 1$
 - 以上三種皆不重複
 - 一次可跨 $1 \sim n$ 階時($n \geq 1$)，從第 $n+1$ 階開始，其全部方法為前 n 階方法數總和。
 - 若 $n \geq 4$, $f(n) = f(n-1) + f(n-2) + f(n-3)$
 - 請考慮使用遞迴與非遞迴方法，或配合指標使用。
 - 請使用 `unsigned long long int x; printf("%lld", x);`

Exercise 登山

- 最大要爬 70 層。

輸入說明

整數 K，K<70，代表共爬 K 階。

輸出說明

整數 M，代表共有幾種爬法。

Sample Input

4

Sample Output

7

Sample Input

12

Sample Output

927

Sample Input

67

Sample Output

33326997224634006

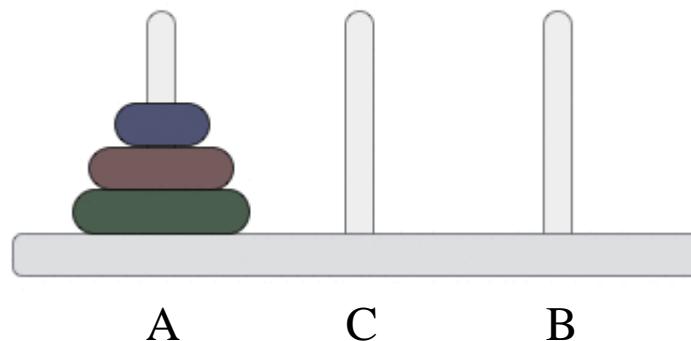
8

河內塔

問題

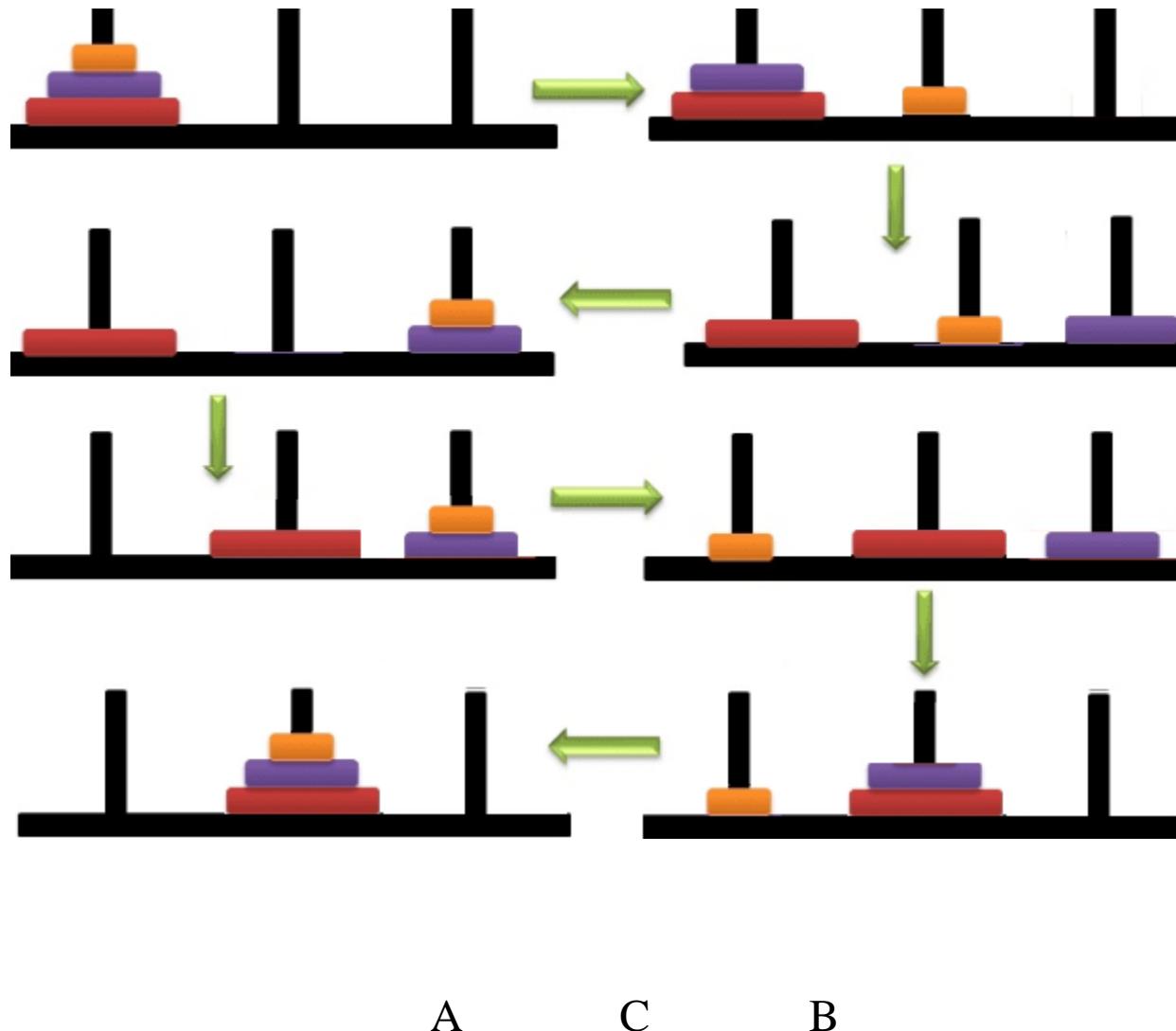
- 有三根竿子A，B，C。A竿上有N個($N > 1$)穿孔圓盤，尺寸由下到上依次變小。按下列規則將所有圓盤移至C杆：
 - 每次只能移動一個圓盤。
 - 大盤不能疊在小盤上面。
 - 圓盤可以在任意一個竿子上。
- 可將圓盤臨時置於B竿，也可將從A竿移出的圓盤重新移回A竿，但都須遵循上述規則。

Step: 0



河內塔

問題



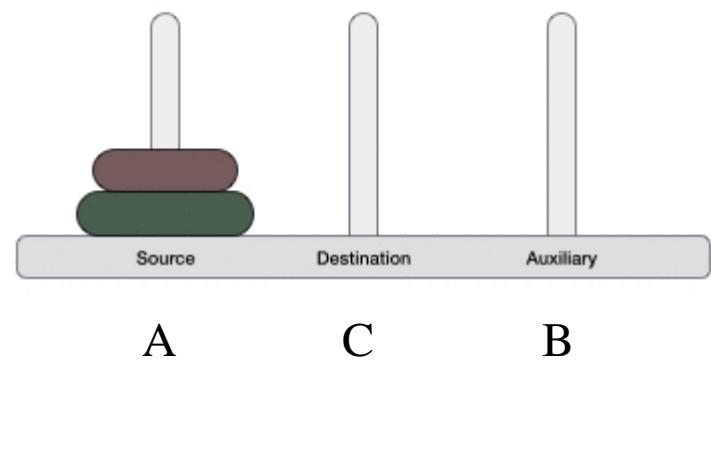
河內塔

□ 解決

- 將較小的(頂部)圓盤移動到輔助竿子B。
- 將較大的(底部)圓盤移動到目標竿子C。
- 將較小的圓盤從輔助竿子B移動到目標竿子C。

```
void move(int i , char x , char y) {  
    static int c = 1;  
    printf("%d: %d from %c > %c\n", c++ , i , x , y);  
}  
void hanoi(int i , char A , char C , char B) {  
    if(i == 1) move(i , A , C);  
    else {  
        hanoi(i - 1 , A , B , C);  
        move(i , A , C);  
        hanoi(i - 1 , B , C , A);  
    }  
}  
// Hanoi(n, 'A', 'C', 'B');
```

Step: 0



Exercise 葛雷碼 (Gray code)

- 反射二進位編碼-葛雷碼 (Gray code) , 是編碼成兩個連續的不同位元。
 - 輸入 n , 編碼範圍 $0 \leq i \leq 2^n - 1$ 。
 - $n = 3$, 編碼 $0 \sim 7$ 為 $000, 001, 011, 010, 110, 111, 101, 100$ 。
 - 其編碼規則

$$G_1 = \{0, 1\}$$

$$G_{1_r} = \{1, 0\}$$

$$G_n = \{0G_{(n-1)}, 1G_{(n-1)}_r\}$$

$$\text{if } G_n = \{g_1, g_2, g_3, \dots, g_n\}$$

$$G_{n_r} = \{g_n, g_{(n-1)}, g_{(n-2)}, \dots, g_1\}$$

[G_{n_r} 是 G_n 的逆向順序]

$$G_{(n+1)} = \{0G_n, 1G_{n_r}\}$$

葛雷碼 (Gray code)

□ 反射二進位編碼-葛雷碼 (Gray code)

例如

$$G_2 = \{0G_1, 1G_1_r\} = \{00, 01, 11, 10\}$$

$$G_{2_r} = \{10, 11, 01, 00\}$$

$$G_3 = \{0G_2, 1G_{2_r}\} = \{000, 001, 011, 010, 110, 111, 101, 100\}$$

$$G_{3_r} = \{100, 101, 111, 110, 010, 011, 001, 000\}$$

其遞迴公式為，

$$G(n, k) = k \quad \text{if } n=1$$

$$G(n, k) = 0G(n-1, k) \quad \text{if } k < 2^{n-1}$$

$$G(n, k) = 1G(n-1, 2^{n-1}-k) \quad \text{if } k \geq 2^{n-1}$$

當 $G(4, 7) = 0G(4-1, 7) = 0G(3, 7) = 01G(3-1, 2^{3-1}-7) = 01G(2, 0) = 010G(2-1, 0) = 010G(1, 0) = 0100$

依此撰寫遞迴程式，輸入n, k，輸出 Gray code。

葛雷碼 (Gray code)

□ 反射二進位編碼-葛雷碼 (Gray code)

輸入說明:

第一行是一個測試案例資料，整數 $n\ k$
接著是一行 0 分隔測試資料
第三行是第二個測試案例資料
最後 -1 結束

輸出說明:

二進位 Gray code
每一行是一個測試案例資料的結果

Sample Input:

1 1
0
2 3
0
3 6
0
4 12
-1

Sample Output:

1
10
101
1010

Exercise 數位電路模擬

□ 模擬一個數位電路。

- 輸入 n 是二進位 8 位元，輸出是二進位 4 位元。
- 輸入範圍從 00000000 到 11111111 (十進位 0~255).

○ 此數位電路內具有回饋迴路，其功能如下：

- $C(m) = m \quad \text{if } m = 0 \text{ or } m = 1$
- $C(m) = C(m/2) \quad \text{if } m \text{ is even 偶數}$
- $C(m) = C((m+1)/2) \quad \text{if } m \text{ is odd 奇數}$

○ 此電路有一個紀錄器，會記錄跑過幾次回饋迴路，最後輸出為回饋電路跑過的次數。

- 例如 $m=00001010$ (十進位 10)，則電路內部運算回饋電路輸入依序為十進位 5, 3, 2, 1 。
- $C(10)=C(5)=C(3)=C(2)=C(1)=1$
- 共跑過 4 次。則此電路輸出為 0100 (十進位 4)。

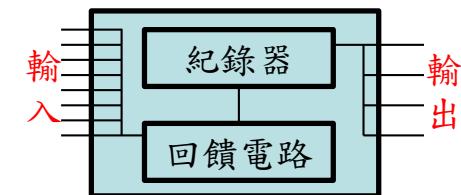
Exercise 數位電路模擬I

□ 模擬一個數位IC，內有回饋電路與紀錄器電路。

- 輸入 m 是二進位 8 位元，輸出是二進位 4 位元。
- 輸入範圍從 00000000 到 11111111 (十進位 0~255).

○ 數位IC內有一個回饋電路，回饋方式：

- $C(m) = m$ if $m = 0$ or $m = 1$ (十進位)
- $C(m) = C(m/2)$ if m 偶數(十進位)
- $C(m) = C((m+1)/2)$ if m 奇數(十進位)
- 例如 $m=00001010$ (十進位 10)，則電路回饋依序為十進位 5, 3, 2, 1 。
- $C(10)=C(5)=C(3)=C(2)=C(1)=1$ ，共回饋 4 次。



○ 數位IC內有一個紀錄器，會記錄回饋電路的回饋次數。

- $R(m) = [C(m) \text{ 的回饋次數}]$ ，例如 $R(10) = 4$ 。

○ 數位IC的輸出為紀錄器所記錄之回饋電路的回饋次數。

- 若數位IC的輸入為 $m=00001010$ (十進位 10)，因回饋電路的回饋次數為4，則此數位IC輸出為 0100 (十進位 4)。

Exercise 數位電路模擬I

- 模擬一個數位IC，內有回饋電路與紀錄器電路。

輸入說明:

二進位 8 bit 位元

第一行是第一個測試案例資料

接著是一行 0 分隔測試資料

第三行是第二個測試案例資料

....

最後 -1 結束

輸出說明:

二進位 4 bit 位元

每一行是一個測試案例資料的結果

Sample Input:

00000000

0

11111111

0

00000001

0

10000000

0

00111111

-1

Sample Output:

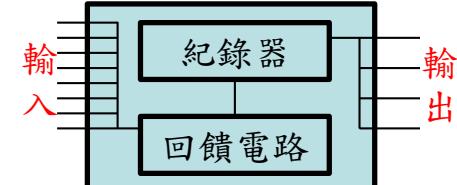
0000

1000

0000

0111

0110



Exercise 數位電路模擬II

□ 模擬一個數位IC，內有回饋電路與紀錄器電路。

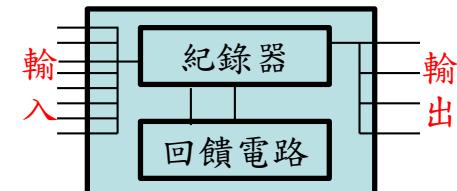
- 輸入 m 是二進位 8 位元，輸出是二進位 4 位元。
- 輸入範圍從 00000000 到 11111111 (十進位 0~255).

○ 數位IC內有一個回饋電路，回饋方式：

- $C(m) = m$ if $m = 0$ or $m = 1$ (十進位)
- $C(m) = C(m/2)$ if m 偶數(十進位)
- $C(m) = C((m+1)/2)$ if m 奇數(十進位)
- 例如 $m=00001010$ (十進位 10)，則電路回饋依序為十進位 5, 3, 2, 1 。
- $C(10)=C(5)=C(3)=C(2)=C(1)=1$ ，共回饋 4 次。

○ 數位IC內有一個紀錄器，其功能

- 純予回饋電路輸入 $0, 1, 2, \dots, m$ ，並記錄每次回饋次數， $R(0), R(1), \dots, R(m)$ 。例如 $R(10)=4$ [$C(10)$ 的回饋次數] 。
- 會累加所有回饋電路的回饋次數。 $Out(10) = R(0)+R(1)+\dots+R(10)$ 。



Exercise 數位電路模擬II

□ 模擬一個數位IC，內有回饋電路與紀錄器電路。

○ 數位IC輸出為記錄器累加回饋次數。

➤ $m = 3$ ， $R(0)+R(1)+R(2)+R(3) = 0+0+1+2=3$ (二進位 0011)。

輸入說明:

二進位 8 bit 位元

第一行是第一個測試案例資料

接著是一行 0 分隔測試資料

第三行是第二個測試案例資料

....

最後 -1 結束

輸出說明:

二進位 11 bit 位元

每一行是一個測試案例資料的結果

Sample Input:

00000000

0

11111111

0

10101010

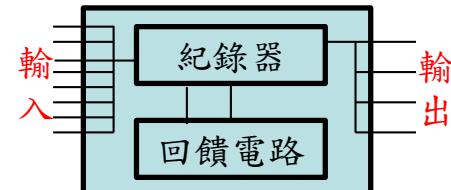
-1

Sample Output:

00000000000

11011111001

10001010001



計算機程式設計

C語言 Pointer

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

指標

□ 指標變數 ptr 指向(儲存)變數 num 記憶體位址。

○ 指標變數的資料型別規定要在一般資料型別加上*

○ 才可以存變數記憶體位址

○ 變數的記憶體位址由編譯器、作業系統配置。

○ &運算子，取出變數被編譯器、作業系統所配置的記憶體位址

○ *運算子，取出指標變數所指向記憶體位址，裡面存的值

➤ ptr指向 num 記憶體位址，num裡面存100。

二行可以合併寫成：

```
int *ptr=&num;
```

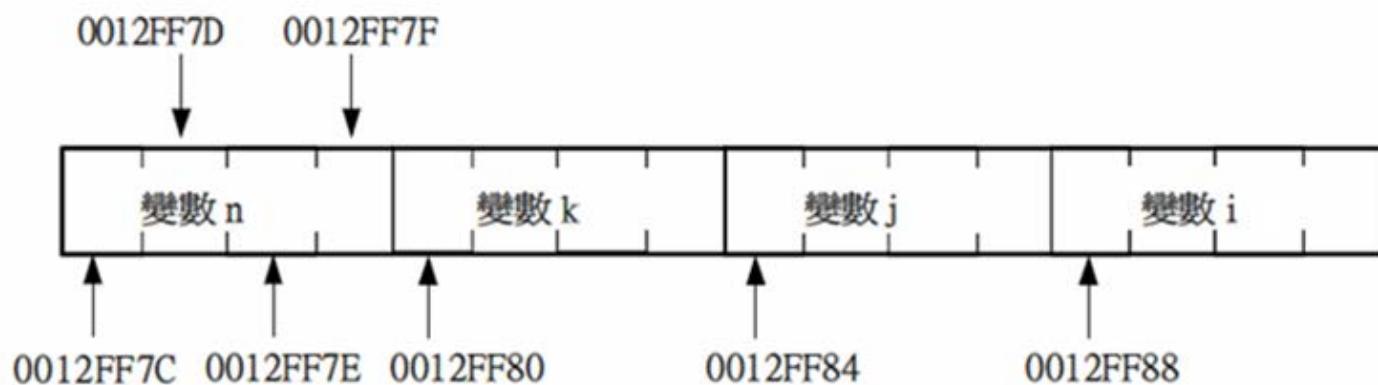
```
int main() {  
    int num = 100;  
    int *ptr;          // int *ptr = &num;  
    ptr = &num;        // *ptr == num  
    printf("num=%d, ptr=%p", num, ptr); //100, 1015  
    return 0;  
}
```

值	變數名	記憶體位址 (假設)
		1001
		...
100	num	1015
1015	ptr	1019
		...

記憶體位址

- 印出變數的記憶體位址。

```
int main() {  
    int i=1, j=2, k=3, n=4;  
    printf("i 記憶體位置=%p\n", &i);  
    printf("j 記憶體位置=%p\n", &j);  
    printf("k 記憶體位置=%p\n", &k);  
    printf("n 記憶體位置=%p\n", &n);  
    return 0;  
}
```



記憶體大小

□ 印出變數的記憶體位址。

```
01 char c = '1'; int i = 1;  
02 float f = 1.0f; double d = 1.0f;  
03 printf("c 記憶體位大小%d\n", sizeof(c));  
04 printf("i 記憶體位大小%d\n", sizeof(i));  
05 printf("f 記憶體位大小%d\n", sizeof(f));  
06 printf("d 記憶體位大小%d\n", sizeof(d));
```

c 記憶體位大小1
i 記憶體位大小4
f 記憶體位大小4
d 記憶體位大小8

```
01 char c = '1'; int i = 1;  
02 float f = 1.0f; double d = 1.0f;  
03 char *cPtr = &c; int *nPtr = &i;  
04 float *fPtr = &f; double *dPtr = &d;  
05 printf("cPtr 記憶體位大小%d\n", sizeof(cPtr));  
06 printf("nPtr 記憶體位大小%d\n", sizeof(nPtr));  
07 printf("fPtr 記憶體位大小%d\n", sizeof(fPtr));  
08 printf("dPtr 記憶體位大小%d\n", sizeof(dPtr));
```

cPtr 記憶體位大小4
nPtr 記憶體位大小4
fPtr 記憶體位大小4
dPtr 記憶體位大小4

設定指標的值

```
01 int *ptr;  
02 *ptr = 35;
```



錯誤的指標初始值(記憶體位址)設定方法。
記憶體位址由編譯器、作業系統配置。

```
01 int number = 100;  
01 int *ptr;  
03 ptr = &number;  
04 *ptr = 35;  
05 printf("*ptr = %d, *ptr);  
06 printf("number = %d, number);
```

輸出結果:

*ptr = 35
number = 35

Exercise

□ 以下輸出

```
int i = 4, *p = &i, *q = &i;  
printf("%d %d\n", *p, *&i);  
i += (*q) * (*p);  
printf("%d %d\n", i, q);
```

```
double i = 4, j = 6, *p = &j, *q = &i, *r;  
printf("%f %f\n", *p, *q);  
r = p; p = q; q = r;  
printf("%f %f\n", *p, *q);
```

```
int i,j=21,*p=&j,*q=p;  
printf("%d %d\n",*p,*q);  
for (i=0;i<4; i++)  
    *(p++);  
(*q)++;  
printf("%d %d %d",*p,*q,i);
```

```
int i=4,j,number=3,*p=&number;  
for (j=0;j<number;j++)  
    (*p)+=i--;  
printf("%d %d %d %d\n",*p,number,j,i);
```

指標的轉型

```
void test() {  
    int x=1001, *p;  
    p=x;  
    printf("%d", *p);  
}
```

錯誤結果，p指向記憶體位址為10的地方，但位址為10 (000A)，裡面存的值未知。

值	變數名	記憶體位址
?		1001
		...
1001	x	1015
1001	p	1019

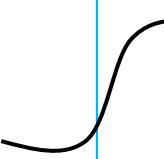
```
void test() {  
    int x=1001, *p;  
    *p=x;  
    printf("%d", *p);  
}
```

可能造成系統問題，p指向記憶體位址未知，在未知的位址內，存入10。

值	變數名	記憶體位址
		1001
		...
1001	x	1015
?	p	1019
		...
1001		?

指標的轉型

```
01 int *ptr;  
02 float f = 100.01f;  
03 ptr = &f;  
04 printf("%0.2f", *(float *)ptr);
```



[Execution Error] *ptr value is unexpected.

```
void *ptr;  
float f = 100.01f;  
ptr = &f;  
printf("%0.2f", *(float *)ptr);
```

若想將指標指向不特定型別變數，
可使用void型別宣告。可指向任何
型別，編譯器不會發出警告。

字串常數

□ 字串常數表示。

```
#define MAX "max"
int main() {
    const char* str = "str";
    printf("%s hello world!!", str);
    return 0;
} str hello world!!
```

相同Global記憶體空間
輸出一樣

```
#include <stdio.h>
void f() {
    const char* str1 = "test";
    const char* str2 = "test";
    printf("%d\n%d\n", str1, str2);
}
int main() {
    const char* str = "test";
    f();
    printf("%d", str);
    return 0;
}
```

- 編譯階段在Global區段建立資料空間，生命週期和程式共存亡。
- 程式執行時，不論在任何地方取得字串常數都有效。
- 字串常數一旦建立，在程式執行中只能讀，不可被修改。

□ 非字串常數

```
char str[] = "hello";           // (使用字串常數初始一個字串陣列)
char str[] = { 'h', 'e', 'l', 'l', 'o', '\0' }; // 字串是字元陣列 + 字串結束符號
```

字串常數

□ 字串常數。

- 不可透過指標解參考(dereference *)修改某個字元。
- 指向字串常數的指標，若被設定指向另一個字串常數或字串陣列，或任何記憶體位址後，就無法再指向本來字串常數，因已遺失字串常數的位址。

```
#include <stdio.h>
int main() {
    char *p = "lose";
    char *str = "test";
    p = str;      // "lose"字串遺失
    str++;        //指標可以移動
    (*str) = 'p'; //不可以指標解參考修改，
                   //執行會錯誤，中斷執行
    printf("%c", *str);
    return 0;
}
```

```
#include <stdio.h>
int main() {
    char *p = "lose";
    printf("%c", *p); //印出一個字元
    printf("%s", p); //印出一個字串
    return 0;
}
```

傳遞指標到函數

- 可以讓函式回傳兩個值以上

```
01 void swap(int a, int b){  
02     int temp = b;  
03     b = a;  
04     a = temp;  
05 }  
06  
07 int main(){  
08     int a = 10, b = 12;  
09     swap(a, b);  
10     printf("a=%d, b=%d", a, b);  
11     return 0;  
12 }  
13 }
```

a=10, b=12

```
void swap(int *a, int *b){  
    int temp = *b;  
    *b = *a;  
    *a = temp;  
}  
  
int main(){  
    int a = 10, b = 12;  
    swap(&a, &b);  
    printf("a=%d, b=%d", a, b);  
    return 0;  
}
```

a=12, b=10

Exercise

□ 寫出以下輸出

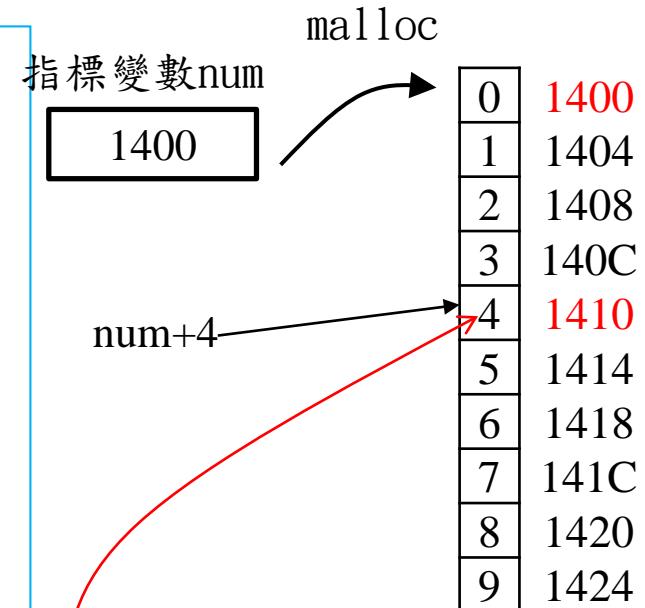
```
void f(int a, int *b, int *c) {  
    int d; a = 2;  
    *b = 3; c = &a; d = 5;  
}  
int main() {  
    int a=1,b=2,c=3,d=4;  
    f(a, &b, &c);  
    printf("%i %i %i %i\n", a, b, c, d);  
    return 0;  
}
```

```
void function(int *a,int *b,int *c){  
    int *temp=a;  
    *b=(*c)*(*temp);  
    *c=*temp;  
    *a=10;  
    a=b;  
    b=c;  
    c=temp;  
}  
void main(void){  
    int i=-1,j=4,k=2,*p=&i,*q=&j;  
    function(p, q, &k);  
    printf("%d %d %d\n",i,j,k);  
}
```

動態記憶體配置

語法:(資料型別 *)malloc(sizeof(資料型別) * 個數)

```
#include <stdio.h>
int main(){
    int *num = (int *)malloc(sizeof(int)*10);
    if (num == NULL) exit(1);
    for(int i=0; i<10; i++) *(num+i) = i;
    for(int i=0; i<10; i++) printf("%d\n", *(num+i));
    free(num);
}
```



不再使用的空間必須還回系統

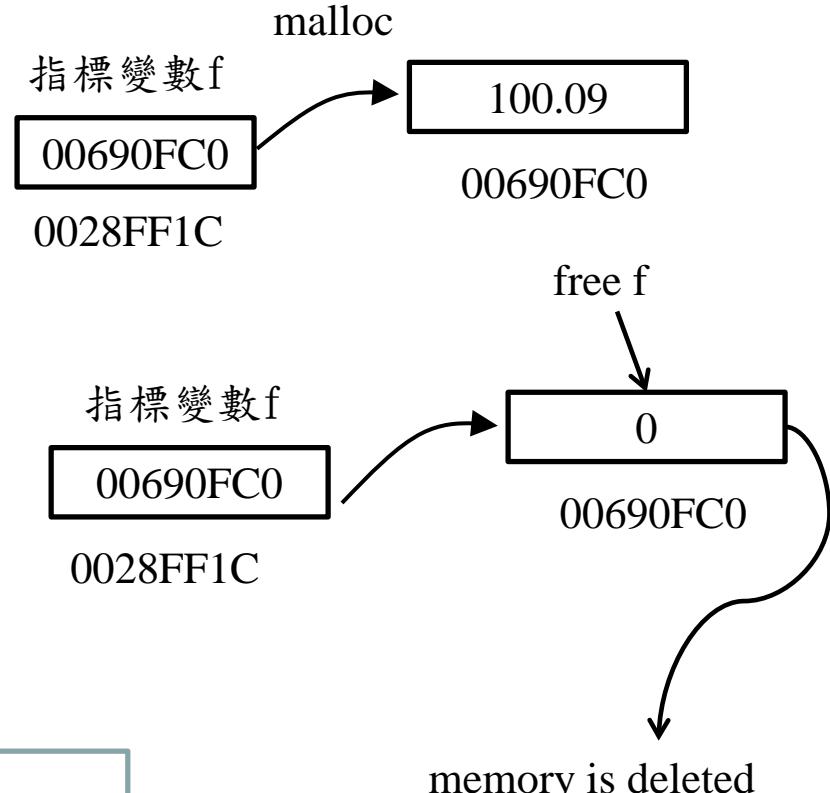
$*(\text{num}+4)$ ，是 $\text{num}+4$ 指向記憶體內存的值

動態記憶體配置

- 印出變數的記憶體位址。

```
#include <stdio.h>
int main(){
    float *f = (float *)malloc(sizeof(float));
    *f = 100.09;
    printf("f=%p &f=%p *f=%+.2f\n", f, &f, *f);
    free(f);
    printf("f=%p &f=%p *f=%+.2f\n", f, &f, *f);
}
```

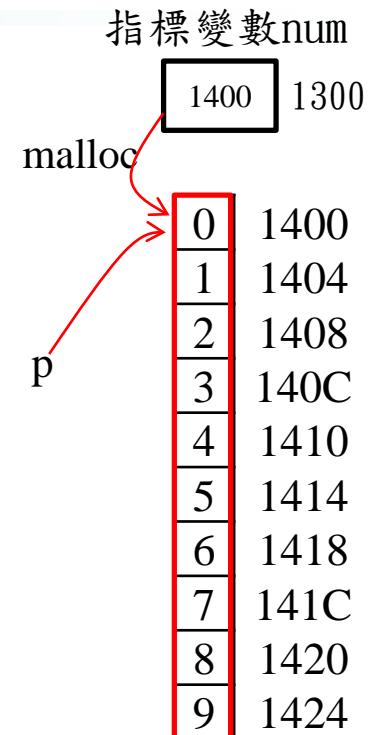
```
f=00690FC0 &f=0028FF1C *f=100.09
f=00690FC0 &f=0028FF1C *f=0.00
```



動態記憶體配置

□ 指標++。

```
#include <stdio.h>
int main(){
    int *p;
    int *num = (int *)malloc(sizeof(int)*10);
    for(int i=0; i<10; i++) *(num+i) = i;
    p = num;
    p++; // p 值會變，指向下一個 1
    printf("%d, ", (*p)++);
    // p 值不會變，1 變2
    printf("%d, ", ++(*p));
    // p 值不會變，2 變3
    printf("%d, ", *(p++));
    // p 值會變，指向下一個，再取值
    printf("%d, ", *p++);
    // 要看 * 或 ++ 哪一個運算子優先
    free(num);
}
```



Exercise

□ 分數輸入

- 輸入 $3/5$ ，輸出 $3, 5$

```
void scan_fraction(int *nump, int *denomp) {  
    char slash; /* character between numerator and denominator */  
    int status; /* status code returned by scanf indicating number of valid values obtained */  
    int error; /* flag indicating presence of an error */  
    char discard; /* unprocessed character from input line */  
    do { /* No errors detected yet */  
        error = 0;  
        /* Get a fraction from the user */  
        printf("Enter a common fraction as two integers separated ");  
        printf("by a slash> ");  
        status = scanf("%d %c%d", _____, _____, _____);  
        /* Validate the fraction */  
        if (status < 3) {  
            error = 1;  
            printf("Invalid-please read directions carefully\n");  
        }  
    }
```

Exercise

□ 分數輸入

- 輸入 $3/5$ ，輸出 $3, 5$

```
else if (slash != '/') {
    error = 1;
    printf("Invalid-separate numerator and denominator");
    printf(" by a slash (/)\n");
} else if (*denomp <= 0) {
    error = 1;
    printf("Invalid—denominator must be positive\n");
}
/* Discard extra input characters */
do {
    scanf("%c", &discard);
} while (discard != '\n');
} while (error);
}
```

Homework I

□ 分數運算

○ 輸入1

- int n1, d1 /* 第一個分數的分子與分母 */
- int n2, d2 /* 第二個分數的分子與分母 */
- char op /* 數學運算 + - * or / */
- char again /* y or n 是否繼續 */

○ 輸出1

- int n_ans /* 答案分子 */
- int d_ans /* 答案分母 */

○ 輸入2

- I(n/d)opI(n/d) , 例如 $-2(2/3)*3(1/4)$

○ 輸出2

- I(n/d) , $-6(1/6)$

Homework II

□ 輸入平面上兩個點，求直線方程式

○ 輸入兩個點，整數， $(x_1, y_1), (x_2, y_2)$

○ 輸出 $y = mx + b$

➤ $m = (y_1 - y_2) / (x_1 - x_2)$

➤ $b = (x_2 y_1 - x_1 y_2) / (x_2 - x_1)$

```
int main() {
    equation(1,0,0,-1);
    equation(1,0,0,1);
    equation(1,1,2,2);
    equation(1,1,2,4);
    equation(2,3,4,5);
    equation(0,1,3,3);
    return 0;
}
```

```
y = x -1
y = -x + 1
y = x
y = 3 x -2
y = x + 1
y = 2/3 x + 1
```

指標的指標

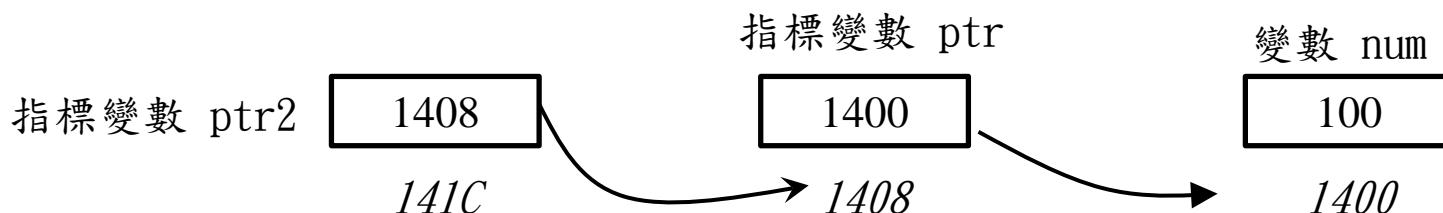
```
01 int num = 100;  
02 int *ptr;  
03 int **ptr2  
04 ptr = &num;  
05 ptr2 = &ptr;  
06 printf("num=%d &num=%p\n", num, &num);  
07 printf("*ptr=%d ptr=%p\n", *ptr, ptr);  
08 printf("**ptr2=%d *ptr2=%p ptr2=%p\n", **ptr2, *ptr2, ptr2);
```

輸出結果:

num=100 &num=0028FF18

*ptr=100 ptr=0028FF18

**ptr2=100 *ptr2=0028FF18 ptr2=0028FF14



Exercise

□ 寫出以下輸出

```
void function(int **a,int **b, int **c,int *d){  
    *a=d;  
    **b=(**a)/(**c);  
    *d=(**b)*(**a);  
}  
void s(int **a, int **b, int **c, int *d){  
    int **temp=a;  
    a = b;      b = temp;  
    *a = d;  
    (**a) = (**b)+(**c);  
}  
void main(void){  
    int i=5,j=-2,k=9,*p=&i,*q=&j,*x=&k;  
    //function(&p,&q,&x,&k);  
    s(&p,&q,&x,&k);  
    printf("%d %d %d",i,j,k);  
}
```

計算機程式設計

C語言 Array

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

陣列

- 陣列是相同型態之元素所組成的集合
- 在 C 語言中，陣列使用前必須先宣告
- 一維陣列宣告
 - 陣列個數必須是整數常數

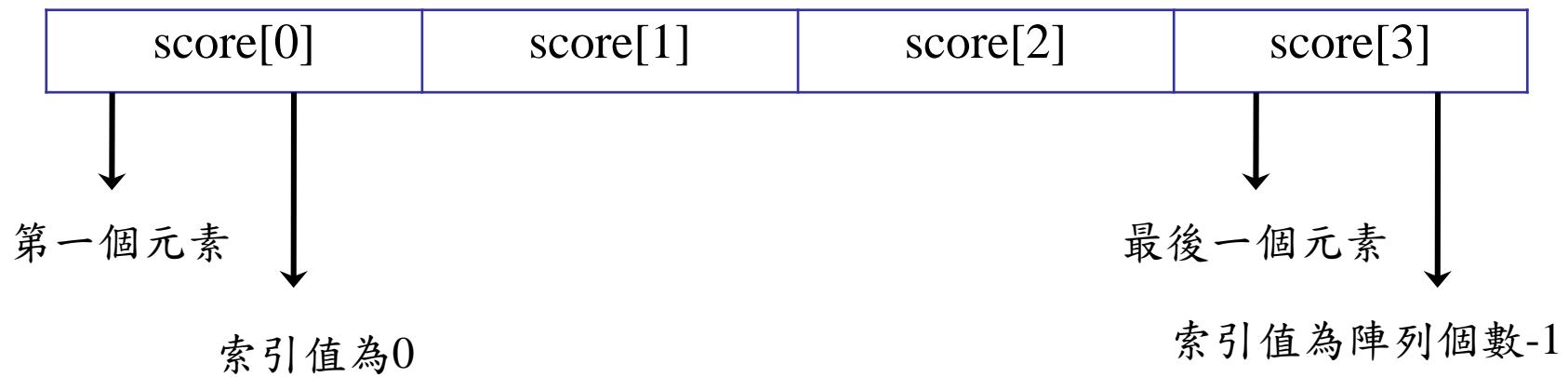
資料型態 陣列名稱[個數];

```
int score[4];      /* 宣告整數陣列score，可存放4個元素 */  
float temp[7];     /* 宣告浮點數陣列temp，可存放7個元素 */  
char name[12];     /* 宣告字元陣列name，可存放12個元素 */
```

陣列

- 陣列中的元素是以索引值來標示存放的位置
- 陣列索引值必須由0開始

```
int score[4];
```



陣列的記憶空間及大小

□ 查詢陣列所佔的記憶空間

`sizeof(陣列名稱)` 檢查陣列所佔的位元組

```
01 int main() {  
02     double data[4];  
03     printf("陣列所佔位元組%d", sizeof(data));  
04     printf("陣列元素所佔位元組%d", sizeof(data[0]));  
05     printf("陣列個數%d", sizeof(data)/sizeof(double));  
06     return 0;  
07 }
```

陣列所佔位元組32
陣列元素所佔位元組8
陣列個數4

陣列的位址

- 陣列的名稱是一個指標常數，它指向該陣列的位址
 - 陣列名稱A是一個指標常數，其值不能被更改

```
int A[4] = {5, 3, 6, 1};
```

A[0]	5	0022fe30
A[1]	3	0022fe34
A[2]	6	0022fe38
A[3]	1	0022fe3c

A diagram illustrating the memory layout of a one-dimensional array A. On the right, the variable A is shown in a box with the value 0022fe30. An arrow points from this box to the first element of the array table. Another arrow points from the value 0022fe30 in the first row of the array table back to the variable A.

一維陣列A

陣列的位址

- C語言是以陣列第一個元素的位址當成是陣列的位址
- 陣列名稱本身就是存放陣列位址的變數

```
01 #define SIZE 4
02 int main() {
03     int A[SIZE] = {5, 3, 6, 1};
04     int i;
05     for(i=0; i<SIZE; i++)
06         printf("A[%d]=%2d,位址為%p\n", i , A[i], &A[i]);
07     printf("陣列A的位址=%p\n", A);
08     return 0;
09 }
```

A[0]= 5,位址為0022fe30
A[1]= 3,位址為0022fe34
A[2]= 6,位址為0022fe38
A[3]= 1,位址為0022fe3c
陣列A的位址=0022fe30

一維陣列基本操作

□ 初始值設定

```
int score[4]={66,23,22,1};  
int score[]={66,23,22,1};  
int score[4]={0}; // 紿第一個元素值
```

```
01 #include <stdio.h>  
02 int main() {  
03     int score[4];  
04     score[0] = 66;  
05     score[1] = 23;  
06     2[score] = 22;    //也可以這樣寫//score[2] = 22;  
07     score[3] = 1;  
08     for(int i=0; i<=3; i++)  
09         printf("score[%d]=%d\n", i, score[i]);  
10     return 0;  
11 }
```

```
score[0] = 66  
score[1] = 23  
score[2] = 22  
score[3] = 1
```

一維陣列基本操作

□ 初始值設定

```
01 #include <stdio.h>
02 int main() {
03     int score[10]={0};      // 只會設定第一個元素為0
04     for(int i=0; i<10; i++)
05         score[i]=0;        // 使用迴圈設定每一個元素為0
06     for(i=0; i<10; i++)
07         printf("score[%d]=%d\n", i, score[i]);
08     return 0;
09 }
```

一維陣列基本操作

□ 由鍵盤輸入資料來設定陣列元素

```
01 int main() {  
02     int i, score[4];  
03     for(i=0; i<4; i++){  
04         printf("請輸入score[%d]的值", i);  
05         scanf(" %d ", &score[i]);  
06     }  
07     for(i=0; i<4; i++)  
08         printf("score[%d]=%d", i, score[i]);  
09     return 0;  
10 }
```

```
請輸入score[0]的值: 95  
請輸入score[1]的值: 100  
請輸入score[2]的值: 63  
請輸入score[3]的值: 77  
score[0]=95  
score[1]=100  
score[2]=63  
score[3]=77
```

傳遞一維陣列到函式

□ 一維陣列名稱為記憶體位址

- 呼叫函式時，傳遞陣列名稱，即傳遞陣列位址，函式不會為陣列配置空間，會為陣列變數配置空間
- 定義函式時，定義陣列變數，或指標變數
- $\text{arr}[i] = *(\text{arr}+i)$

```
01 #define SIZE 4
02 void show(int arr[]){ //int arr[4], int *arr 等價
03     for(int i=0; i<SIZE; i++)
04         printf("%d ", arr[i]);
05     //printf(" %d", *(arr+i));
06 }
07 void change(int arr[]){
08     for(int i=0; i<SIZE; i++)
09         arr[i] = i;
10 }
```

```
12 int main() {
13     int A[SIZE] = {5, 3, 6, 1};
14     //int *A = {5, 3, 6, 1}; 錯誤
15     show(A); // show(&A[0])
16     change(A);
17     show(A);
18     return 0;
19 }
```

搜尋(Search)一維陣列

□ 線性搜尋(Linear Search)/循序式搜尋(Sequential Search)

○ 從第一筆資料開始搜尋比對，如果找到則傳回該值或該位置，
如果沒找到則往下一筆資料搜尋比對，

○ 例如，有一整數陣列的資料內容如下：

	0	1	2	3	4	5
data	13	25	16	23	57	66

□ 如欲找出57，則：

- Step 1: 與陣列中第一筆資料比對， $57 \neq 13$
- Step 2: 與陣列中第二筆資料比對， $57 \neq 25$
- Step 3: 與陣列中第三筆資料比對， $57 \neq 16$
- Step 4: 與陣列中第四筆資料比對， $57 \neq 23$
- Step 5: 與陣列中第五筆資料比對， $57 = 57$

搜尋(Search)一維陣列

□ 線性搜尋(Linear Search)/循序式搜尋(Sequential Search)

```
01 int search(int d[], int size, int key) {  
02     int i=0;  
03     for (i=0; i<size; i++)  
04         if (d[i]==key) return i;  
05     return -1;  
06 }  
07 int main() {  
08     int data[]={5,2,8,1,7,9,4,3,6};  
09     printf("%d, %d\n", search(data, 9, 1), 1);  
10     return 0;  
11 }
```

搜尋(Search)一維陣列

□ 二元搜尋(Binary Search)

- 陣列資料須要排序
- 每次都從範圍(left~right)的中間點 $mid=(left+right)/2$ 找
- 中間點太大，往左找，中間點太小，往右找

假設找9

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	2	5	7	9	14	23	26
left=0			mid = 3 7<9 left = 3				right=7
					mid = 5 (7+3)/2 14>9 right= 5		
				mid = 4 (3+5)/2			

搜尋(Search)一維陣列

□ 二元搜尋(Binary Search)

```
01 #include <stdio.h>
02 int binarySearch(int d[], int left, int right, int key) {
03     int mid = 0;
04     while (left<right) {
05         mid = (left+right)/2;
06         if (d[mid]==key) return mid;
07         else if (d[mid]>key) right = mid;
08         else left = mid;
09     }
10     return -1;
11 }
12 int main() {
13     int data[]={1,2,5,7,9,14,23,26};
14     printf("%d, %d\n", binarySearch(data, 0,7,9), 9);
15     return 0;
16 }
```

大數相加

- 20位數以上整數加1，C語言 long long 無法表示30為整數

○ 使用整數陣列表示


```
#include<stdio.h>
int Add(int a[],int b[],int size){
    int i=0, carry=1;
    for (i=0; i<size; i++) {
        b[i] = (a[i] + carry)%10;
        carry = (a[i] + carry)/10;
    }
    if (carry ==1) {
        b[i] =1;
        size++;
    }
    return size;
}
```

```
void print(int a[], int size) {  
    int i=0;  
    for (i=size-1; i>=0; i--) printf("%d", a[i]);  
    printf("\n");  
}  
void test01(){  
    int a[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};  
    int c[]={9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9};  
    int b[22];  
    Add(a, b, 20);  
    print(a, 20);    print(b, 20);  
    Add(c, b, 20);  
    print(c, 20);    print(b, 21);  
    return 0;  
}
```

Exercise

- 20位數以上整數加1，C語言 long long 無法表示30位數整數
 - 使用整數陣列表示

➤ $10000000000000000000000000000000 + 99999999999999999999 =$
 $11000000000000000000000000000000$

```
#include<stdio.h>
int Add(int a[], int b[], int c[], int size){
    int i=0, carry=1;
    for (i=0; i<size; i++) {
    }
    if (carry ==1) {
    }
    return size;
}
```

```
void print(int a[], int size) {
    int i=0;
    for (i=size-1; i>=0; i--) printf("%d", a[i]);
    printf("\n");
}
void test01(){
    int a[]={1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
    int c[]={9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9};
    int b[22];
    Add(a, b, c, 20);
    print(a, 21);  print(b, 21);  print(c, 21);
}
```

Homework I

- 20位數以上整數加1，C語言 long long 無法表示30位數整數
 - 使用整數陣列表示50位數長整數
 - 計算兩個長整數的 +, -, *
 - 整數可以為正數與負數

二進位轉十進位

□ 運用位移運算子

```
#include <stdio.h>
int test(char s[]) {
    int sum=0, i=0;
    unsigned index=0x80; //1000 0000 的 16 進位
    for (i=0; i<=7; i++) {
        sum = sum + (s[i]-'0')*index; // s[i] * 2 的n次方，加總
        index >>= 1;
    }
    return sum;
}
int main() {
    printf("%d",test("01001010"));
    return 0;
}
```

Homework II

□ 分散度

- 輸入一串整數序列，計算此序列的m分散度。
- m分散度定義為，序列中擁有長度為 m 且有 m 種不同數字的連續子序列之數量。
- 例如， $m = 3$ ，序列 {1 2 3 5 4 5 4}，3 分散度數量 {1 2 3}, {2 3 5}, {3 5 4}，四個。

排序 (sorting) — 維陣列

- 將相同性質資料，由小至大或由大至小排列；身高、英文字典字詞順序、事件發生遠近、...，都可排序。
- 依資料存放位置的不同，可分：
 - 內部排序法 (internal sorting)：資料全部在主記憶體中。
 - 外部排序法 (external sorting)；主記憶體中只存放部分資料，大部分資料皆在外部記憶體如硬碟檔案中。

選擇(selection)排序法

- 每次迴圈都挑出目前為排序資料的最小值
 - 利用n次迴圈完成排序。
 - 在第i次迴圈時，挑出第i小的資料將之與陣列中第i筆資料對調，重整陣列使成為由小至大排序的數列。

```
#include <stdio.h>
#define SWAP(x,y) { int t; t = x; x = y; y = t;}
int getMinIndex(int d[], int left, int right) {
    int i=0, minIndex = left;
    for ((i=left+1); i<right; i++) {
        if (d[i]<d[minIndex]) minIndex=i;
    }
    return minIndex;
}
void selectSort(int d[], int n) {
    int i=0, index=0;
    for (i=0; i<n; i++) {
        index = getMinIndex(d, i, n);
        SWAP(d[i], d[index]);
    }
}
```

插入排序法 (insertion sort)

- 將未排序的資料逐一插入已排序的部分資料中。

```
for (i=1; i<n; i++) {  
    target = data[i];  
    j = i;  
    while ((data[j-1]>target) && (j>0)) {  
        a[j] = a[j-1];  
        j--;  
    }  
    a[j] = target;  
}
```

Homework III

□ 105APCS Q4

- 輸入棒球隊球員打擊結果，計算出隊得分。假設球員打擊情況：
 - 安打：以 1, 2, 3 和 H 代表一、二、三和全(四)壘打。
 - 出局：以 O 表示 (OUT)。
- 簡化版的規則如下：
 - 球場上有四個壘包，稱為本壘、一、二和三壘。
 - 本壘握球棒打的稱「擊球員」，在另外三個壘包的稱為「跑員」。
 - 當擊球員打擊「安打」時，擊球員與跑壘員可移動；「出局」時，跑壘員不動，擊球員離場換下一位。
 - 比賽開始由第 1 位打擊，當第 i 位球員打擊完後，由第 $(i+1)$ 位球員打擊。當第九位球員打擊完，則輪回第一位球員。
 - 打出 K 壘打時，場上球員(擊球員和跑壘員)會前進 K 個壘包。本壘到一壘，接著二、三壘，最後回到本壘。回到本壘可得 1 分。
 - 每達到三個出局數時，壘包清空(跑壘員都得離開)，重新開始。

Homework III

□ 105APCS Q4

○ 輸入格式

- 每組測試資料固定有十一行。
- 第一到九行，依照球員順序，每一行代表一位球員的打擊資訊。每一行開始有一個正整數 a ($1 \leq a \leq 5$)，代表球員總共打 a 次。接下來有 a 個字元，依序代表每次打擊結果。資料間均以一個空白字元隔開。球員打擊資訊不會有錯誤與缺漏。
- 第十行有一個正整數 b ($1 \leq b \leq 27$)，表示想計算當總出局數累計到 b 時，該球隊得分。輸入打擊資訊中至少包含 b 個出局。
- 第十一行有一個正整數 m ($1 \leq m \leq 9$)，表示想計算第 m 個球員在總計第 b 個出局數安打數與到達本壘的次數。

○ 輸出格式

- 計算在總計第 b 個出局數發生時的總得分。
- 計算第 m 個球員在總計第 b 個出局數的安打數與到達本壘的次數。

Homework III

□ 105APCS Q4

輸入範例一

5 1 1 O O 1

5 1 2 O O O

4 O 4 O 1

4 O O O 4

4 1 1 1 1

4 O O 3 O

4 1 O O O

4 O O 2 2

4 3 O O O

3

4

正確輸出

0

?

輸入範例一

5 1 1 O O 1

5 1 2 O O O

4 O 4 O 1

4 O O O 4

4 1 1 1 1

4 O O 3 O

4 1 O O O

4 O O 2 2

4 3 O O O

6

7

正確輸出

5

?

氣泡排序法(bubble sort)

- 相鄰的兩元素要維持「上小下大」的順序關係；相鄰的兩元素會互相比較大小，將較大的資料往下放。

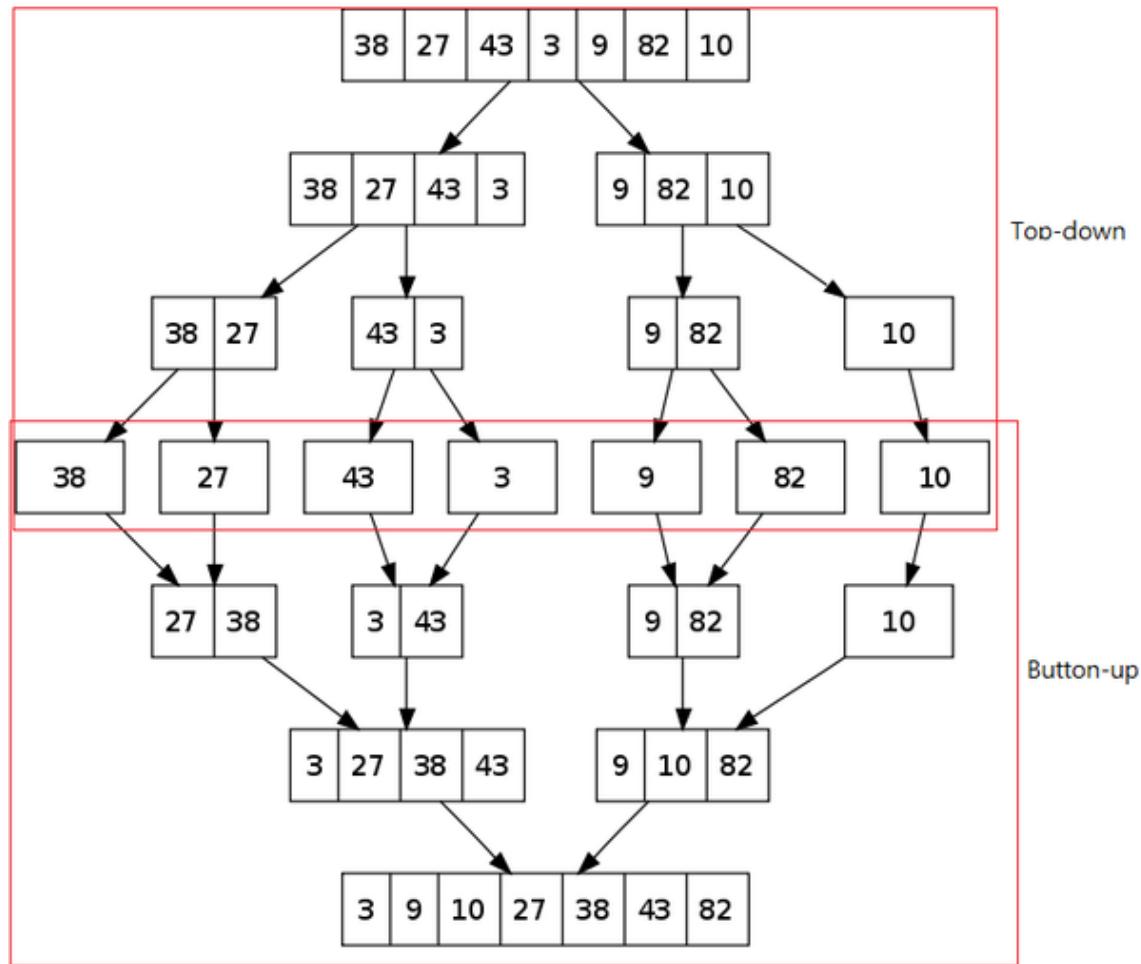
```
1 for (i=n-1; i>0; i--)  
2   for (j=1; j<=i; j++)  
3     if (data[j-1]>data[j])  
4       swap(&data[j-1], &data[j]);
```

索引	0	1	2	3	4	5	6	7	8	9
數值	69	81	30	38	9	2	47	61	32	79

播放動畫

合併排序法(merge sort)

- 將兩組已各自排序好的數列予以合併，使成為一完整的排列數列。



合併排序法(merge sort)

- 將兩組已各自排序好的數列予以合併，使成為一完整的排列數列。

```
1 void merge(int C[], int k, int A[], int i, int m, int B[], int j, int n) {  
2     while ((i<=m) &&(j<=n)) {  
3         if(A[i]<=B[j])  
4             C[k++] = A[i++];  
5         else  
6             C[k++] = B[j++];}  
7     }  
7     while (i<=m) C[k++] = A[i++];  
8     while (j<=n) C[k++] = B[j++];  
9 }
```

Homework I

問問題敘述

- 考慮一個數列 $A = (a[1], a[2], a[3], \dots, a[n])$ 。如果 A 中兩個數 $a[i]$ 和 $a[j]$ 滿足 $i < j$ 且 $a[i] > a[j]$ ，則說 $(a[i], a[j])$ 是 A 中的一個反序 (inversion)。定義 $W(A)$ 為數列 A 中反序數量。例如，在數列 $A = (3, 1, 9, 8, 9, 2)$ 中，共有 $(3, 1)$ 、 $(3, 2)$ 、 $(9, 8)$ 、 $(9, 2)$ 、 $(8, 2)$ 、 $(9, 2)$ 6 個反序，所以 $W(A) = 6$ 。
- 給定數列 A ，計算 $W(A)$ 簡單方法是 對所有 $1 \leq i < j \leq n$ 檢查數對 $(a[i], a[j])$ ，但在序列太長時，計算時間會超過給定期限。以下運用分而治之 (divide and conquer) 策略所設計更有效率方法。
 - 將 A 等分為前後兩個數列 X 與 Y ，其中 X 的長度是 $n/2$ 。
 - 遞迴計算 $W(X)$ 和 $W(Y)$ 。
 - 計算 $W(A) = W(X) + W(Y) + S(X, Y)$ ，其中 $S(X, Y)$ 是由 X 中的數字與 Y 中的數字構成的反序數量。

Homework I

- 以 $A = (3, 1, 9, 8, 9, 2)$ 為例， $W(A)$ 計算如下。
 - 將 A 分為兩個數列 $X = (3, 1, 9)$ 與 $Y = (8, 9, 2)$ 。
 - 遞迴計算得到 $W(X) = 1$ 和 $W(Y) = 2$ 。
 - 計算 $S(X, Y) = 3$ 。因為有三個反序 $(3, 2)$ 、 $(9, 8)$ 、 $(9, 2)$ 是由 X 中的數字與 Y 中的數字所構成。所以得到 $W(A) = W(X) + W(Y) + S(X, Y) = 1+2+3 = 6$ 。
- 請撰寫一個程式，計算一個數列 A 的反序 數量 $W(A)$ 。
- 輸入格式：測試資料有兩列，第一列為一個正整數 n ，代表 A 的長度。第二列有 n 個不大於 10^6 的非負整數，代表 $a[1], a[2], a[3], \dots, a[n]$ ，數字間以空白隔開。
- 輸出格式： A 的反序 數量 $W(A)$ 。可能超過32-bit 整數範圍。

範例一：輸入

6

3 1 9 8 9 2

正確輸出

6

範例二：輸入

5

5 5 4 3 1

正確輸出

9

快速排序法(quick sort)

□ 概念

- 選取某個元素做為基準值，令此基準值為target。
- 將所有比target小的資料，都放在target左邊；
- 所有不比target小的資料都放在target右邊。

□ 步驟

- 選取第0個元素 69為基準
- 從最右邊往左找比基準69還小的元素32
- 從最左邊往右找比基準69還大的元素81
- 兩個元素交換



索引	0	1	2	3	4	5	6	7	8	9
數值	69	81	30	38	9	2	47	61	32	79

快速排序法(quick sort)

```
#define SWAP(x,y) {int t; t = x; x = y; y = t;}  
void QuickSort (int data[], int left, int right) {  
    int i, j, target;  
    if (left>=right) return;  
    i = left;  
    j = right;  
    target = data[left];  
    while (i!=j) {  
        while ((data[j]>target)&&(i<j)) j--;//從右邊開始找  
        while ((data[i]<=target)&&(i<j)) i++;//從左邊開始找  
        // 左邊開始找比基準點大，如果有找到又沒與從右邊的相遇  
        // 表示 data[i]一定可以換到比較小的  
        // 否則 data[i]一定是小的最邊緣，可以跟中間值交換  
        if(i<j) SWAP(data[i], data[j]);//左右沒相遇則可交換  
    }  
    SWAP(data[left], data[i]) //i是中間值  
    QuickSort(data, left, i-1); //處理左半邊  
    QuickSort(data, i+1, right); //處理右半邊  
}
```

快速排序法(quick sort)

- 須從右邊right開始往左找 j--，找比中間點小的準備交換
 - 之後才可以左邊left開始往右找 i++，找比中間點大的交換

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
6	9 3	7	0	1	3 9
i=1					j=5
6	3	7 1	0	4 7	9
i=2					j=4
6	3	1	0	7	9
i=2		j=3			
		j=3, i=3			

先由右往左找 j--，再由左往右找 i++， $a[3] < 6$, $i = 2 + 1 = 3$
此時 $a[i=3]$ 與 $a[left=0]$ 交換，正確

6 0	3	1	0 6	7	9
-----	---	---	-----	---	---

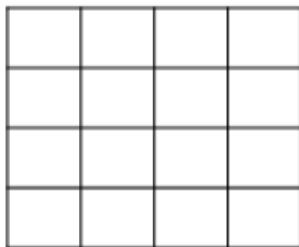
6	3	1	0	7	9
					j=4
i=2		i=3	i=4		

先由左往右找 i++， $i=3, j=4$ 時還可再 $i++$ ， $i=4$ ，
再由右往左找 $i==j$ ，仍然 $j=4$
此時 $a[i=4]$ 與 $a[left=0]$ 交換將發生錯誤

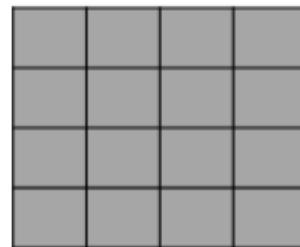
6 7	3	1	0	7 6	9
-----	---	---	---	-----	---

Exercise APCS

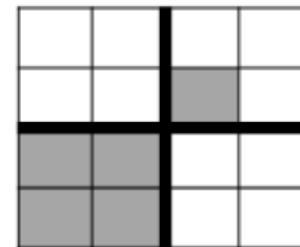
- DF - expression是儲存圖片資訊的表達方式。在一個 $n \times n$ 方陣中，若方格是白色記為0；黑色記為1；若方格可分為更小方格(左上、右上、左下、右下)，則記2，再依序(左上→右上→左下→右下)記錄這四個方格的資訊。
- 給定DF - expression的輸入，與這張圖像寬度(必為2的非負整數次方)，輸出這張圖中黑色格子的數量。



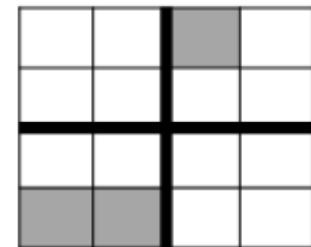
0



1



202001010



2021000200110

Exercise

□ Input

- 輸入第一行包含由0~2組成數組，代表某 DF - expression 結果
- 第二行輸入一個正整數，代表圖像寬度，為2的非負整數次方
- 第一行必定為一個2後面接續4個0或1

□ Output

- 輸出整數，代表圖中黑色格子的數量

Sample Input 1

2020010120110

4

Sample Output 1

7

Sample Input 2

202002020100010

8

Sample Output 2

17

Exercise 遷迴

```
#include<stdio.h>
#include<string.h>
int square(char data[100] ,int length ,int n,int *i){
    int time=0, total=0;
    while((*i)<length){
        time++;
        if(data[*i]=='2'){
            (*i)++;
            total = total+ square(data ,length ,n/2 ,i);
        }
        else if(data[*i]=='1'){
            total = total+(n*n);
            (*i)++;
        }
        else if(data[*i]=='0'){
            (*i)++;
        }
        if(time==4) break;
    }
    return total;
}
```

```
void test01(){
    int n=0 ,length=0, i=0;
    char data[100];
    //scanf("%s",&data);  scanf("%d",&n);length = strlen(data);
    //printf("%d",square(data ,length ,n ,&i));
    printf("%d\n",square("202001010" ,9 ,4 ,&i)); i=0;//5
    printf("%d\n",square("2020020100010" ,13 ,8 ,&i));i=0;//17
    printf("%d\n",square("2020020100010" ,13 ,4 ,&i));i=0;//7
    printf("%d\n",square("2020010120110" ,13 ,8 ,&i));i=0;//28
    printf("%d\n",square("2021000200110" ,13 ,4 ,&i));i=0;//3
}
int main(){  test01();  return 0;}
```

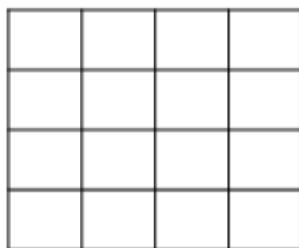
Exercise 非遞迴

```
#include <stdio.h>
#include <string.h>
int count(int level) {
    int ans=1;
    for (int i=0; i<level;i++) ans = ans*2;
    return (ans*ans);
}
int f(int N, char data[]) {
    int level =0, index =0, sum=0;
    int len = strlen(data);
    int comp[100]={0};
    for (int i=0; i<100; i++) comp[i]=0;
    while (index <len) {
        if (data[index]=='2') level++;
        else if (data[index]=='1') {
            sum = sum + (N*N)/count(level);
        }
        if (data[index]!='2') comp[level]++;
    }
}
```

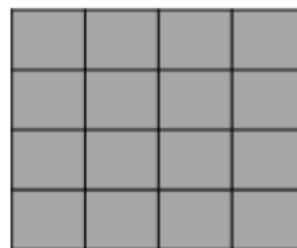
```
if (comp[level]>=4) {
    comp[level]=0;
    level--;
    comp[level]++;
}
index++;
}
return sum;
}
int main() {
    printf("=>%d\n", f(4, "0"));      //0
    printf("=>%d\n", f(4, "1"));      //16
    printf("=>%d\n", f(2, "21100"));   //2
    printf("=>%d\n", f(4, "202001010")); //5
    printf("=>%d\n", f(8, "2020020100010")); //17
    printf("=>%d\n", f(4, "2020010120110")); //7
    printf("=>%d\n", f(8, "2020010120110")); //28
    printf("=>%d\n", f(4, "2021000200110")); //3
    return 0;
}
```

Homework II

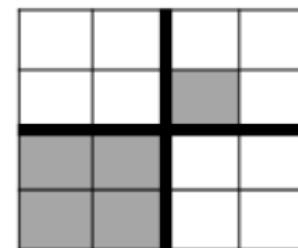
- DF - expression是儲存圖片資訊的表達方式。在一個 $n \times n$ 方陣中，若方格是白色記為0；黑色記為1；若方格可分為更小方格(左上、右上、左下、右下)，則記2，再依序(左上→右上→左下→右下)記錄這四個方格的資訊。
- 給定DF - expression的輸入，與這張圖像寬度(必為2的非負整數次方)，輸出這張圖中黑色格子的座標位置。



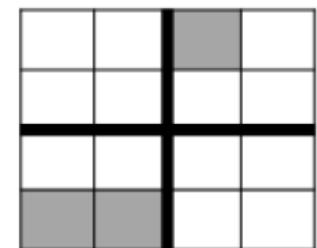
0



1



202001010



2021000200110

二維陣列

□ 二維陣列的宣告

資料型態 陣列名稱[列的個數][行的個數];

```
int data[10][5];      /* 可存放10列5行個整數 */  
float score[4][3];    /* 可存放4列3行個浮點數 */
```

□ 二維陣列的宣告與資料初始化

```
int data[4][3] = {{1, 2, 3}, {4, 5, 6}, 7, 7, 7}, {7, 8, 9}};
```

Array Index

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)
(3,0)	(3,1)	(3,2)

Array Value

1	2	3
4	5	6
7	7	7
7	8	9

二維陣列

□ 二維陣列資料初始化

```
#include <string.h>
void *memset(void *str, int c, size_t n)
複製字元c (unsigned char) 參數str指向的字串的前n個字元(單位1Byte)。
```

```
#include <stdio.h>
#include <string.h>
void test() {
    char s1[] = "congratulation";
    char s2[][20] = {"Hi", "Hello", "Good"};
    printf("%s\n", memset(s1, '#', 10));
    printf("%s\n", s1);
    printf("%s\n", memset(s2, '#', 22));
    printf("%s\n", s2[0]);
    printf("%s\n", s2[1]);
}
```

```
#####
#####tion  
#####
#####tion  
#####
#####Hello  
#####
#####Hello  
##Hello
```

多維陣列

- C語言把多維陣列看成是由低一維的陣列所組成的一維陣列
 - n維陣列是n-1維陣列所組成的一維陣列
 - 3維陣列是2維陣列所組成的一維陣列
 - 多維陣列名所指的元素是其低一維的陣列

```
int data[4][3] = {{1, 2, 3}, {4, 5, 6}, {7, 7, 7}, {7, 8, 9}};
```

Array Index			
data[0]	(0,0)	(0,1)	(0,2)
data[1]	(1,0)	(1,1)	(1,2)
data[2]	(2,0)	(2,1)	(2,2)
data[3]	(3,0)	(3,1)	(3,2)

Array Value		
1	2	3
4	5	6
7	7	7
7	8	9

```
data[0] = {1, 2, 3}  
data[1] = {4, 5, 6}  
data[2] = {7, 7, 7}  
data[3] = {7, 8, 9};
```

二維陣列基本操作

□ 二維陣列的給值與輸出

```
#define ROW 2
#define COL 4
int main() {
    int i, j;
    int arr[ROW][COL];
    for(i=0; i<ROW; i++){
        for(j=0; j<COL; j++){
            arr[i][j] = i*j;
        }
    }
    for(i=0; i<ROW; i++){
        for(j=0; j<COL; j++) {
            printf("%d*%d=%d\t", i, j, arr[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

二維陣列操作

□ 傳遞二維陣列到函式

```
01 #define ROW 4
02 #define COL 3
03 void search(int arr[][COL], int p[]){
04     int i, j;
05     p[0] = p[1]=arr[0][0];
06     for(i=0; i<ROW; i++){
07         for(j=0; j<COL; j++){
08             if(p[0]<arr[i][j]) p[0] = arr[i][j];
09             if(p[1]>arr[i][j]) p[1] = arr[i][j];
10         }
11     }
12 }
13 void show(int arr[][COL] {
14     int i, j;
15     for(i=0; i<ROW; i++){
16         for(j=0; j<COL; j++)
17             printf("%02d ", a[i][j]);
18         printf("\n");
19     }
20 }
```

```
21 int main() {
22     int a[ROW][COL]= {{1, 2, 3}, {4, 5, 6},
23                         {7, 8, 9}, {5, 4, 3}};
24     int i, j, b[2];
25     show(a);
26     search(a, b);
27     printf("max = %d, min = %d\n", b[0], b[1]);
28 }
```

Exercise

□ APCS2019

- 給定一張二維平面圖，起點為整張圖權重最小的點，在此平面圖上從起點開始走出一條路徑，求此路徑經過點的總權重。往下一個點走避須遵循規則：
 - (1) 從相鄰的點選擇一個權重最小的點往下走，相鄰的點是上、下、左、右。
 - (2) 走過的點不能重複。
- 例如以下平面圖，從1開始走，接著走6，再走13，最後走到9。
- 總權重為 $1+6+13+4+12+5+14+7+11+3+16+9 = 101$

3	16	9	1
11	7	14	6
10	15	5	13
2	8	12	4

Exercise

□ APCS2019

```
#include <stdio.h>
#define maxN 105
const int INF = 0x3f3f3f3f;           // INF = 1061109567 , 大於值域
typedef long long LL;
int mp[maxN][maxN];
int min(int x, int y) {
    int r = x>y?y:x;  return r;
}
LL dfs ( int x, int y, LL sum ){      // 先找出最低點
    int mi = min(min(mp[x+1][y], mp[x-1][y]), min(mp[x][y+1], mp[x][y-1]));
    sum += mp[x][y];
    mp[x][y] = INF;
    if ( mi == INF )
        return sum;
    if ( mp[x + 1][y] == mi )
        return dfs ( x + 1, y, sum );
    if ( mp[x - 1][y] == mi )
        return dfs ( x - 1, y, sum );
    if ( mp[x][y + 1] == mi )
        return dfs ( x, y + 1, sum );
    if ( mp[x][y - 1] == mi )
        return dfs ( x, y - 1, sum );
}
```

Exercise

□ APCS2019

```
int main(){
    int n, m, x, y, mi = INF, i=0, j=0;
    memset ( mp, INF, sizeof mp ); // INF 可以 0x3f 取代，因一次寫一個char
    scanf("%d %d", &n, &m);
    // 1 index，簡化邊界判斷，邊界都是 INF = 1061109567
    for (i = 1 ; i <= n ; i++ ) {
        for (j = 1 ; j <= m ; j++) {
            scanf("%d", &mp[i][j]);
            if (mi > mp[i][j])
                mi = mp[i][j], x=i, y=j;
        }
    }
    printf("%d\n",dfs(x, y, 0));
    return 0;
}
```

INF	INF	INF	INF	INF	INF
INF	INF	INF	INF	INF	INF
INF	INF	INF	INF	INF	INF
INF	INF	INF	INF	INF	INF
INF	INF	INF	INF	INF	INF

Exercise

□ memset

- 以位元組Byte為單位設定其值，即後8位元二進位進行賦值。
- 1的二進位是(00000000 00000000 00000000 00000001)，取後8位(00000001)，int型占4個Byte，當初始化為1時，它把一個int的每個位元組都設置為1，也就是0x01010101,二進位是00000001 00000001 00000001 00000001，十進位就是16843009。
- 輸入0,-1時正確初始化0, -1，純屬巧合。
 - 0，二進位是(00000000 00000000 00000000 00000000)，取後8位(00000000)，初始化後00000000 00000000 00000000 00000000是0
 - -1，負數在電腦中以補數存儲，二進位是(11111111 11111111 11111111 11111111)，取後8位(11111111)，則是11111111 11111111 11111111 11111111結果也是-1
- 總結：memset()只有在初始化-1, 0時才會正確。

Exercise

- 三個 5×5 矩陣相加，相減
- 隨意矩陣大小(10以下)的加、減、乘。考慮矩陣維度適當性，若使用者填錯要顯示錯誤訊息並要求重來。

Homework III

□ 五子棋

- 檢查 10×10 五子棋可以構成5個連為一線的位置。1表示有放棋子，0表示沒有放棋子，如右圖。
- 可以增加5個連為一線，以下圖表示。

000000 x 0 1 0
000000 1 1 0 0
000000 1 0 0 0
00000 x 1 1 1 1
00001 0 1 0 0 0
000x00x000
0001000000
0001000000
0 x 1 1 1 1 x 0 0 0
0001000000

其位置為
0 6
3 5
5 3
5 6
8 1
8 6

0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0
0 0 0 1 0 0 0 0 0 0

Homework IV

□ 旋轉矩陣

○ 設一個 $n \times n$ 的矩陣，由左而右，由上而下標示自 1 到 $n \times n$ 的數，如下圖為 4×4 的。

○ 讀入旋轉序列後，將該矩陣的資料輸出。

○ 右圖表示向右旋轉一次 R。

➤ (順時鐘)

13	9	5	1
14	10	6	2
15	11	7	3
16	12	8	4

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

○ 右圖表示向左旋轉一次 L，

➤ (逆時鐘)

4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

○ 右圖表示上下對翻一次，N

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Homework I

□ 騎車

- 賴先生騎腳踏車挑戰一日N塔， $N < 10$ 。每一個塔位在編號 1, 2, 3, ..., N 城市中。兩兩個城市都有一段距離的公路相連。請計算從第 1 個城市出發，騎過每一個城市的最短距離。例如 $N = 5$ ，以下是兩兩城市之間公路的距離。例如

- 城市 1 和城市 2 的距離是 4，
- 城市 1 和城市 3 的距離是 2，
- 城市 3 和城市 4 的距離是 2，
- 城市 5 和城市 4 的距離是 3，

--	1	2	3	4	5
1	0	4	2	3	6
2	4	0	3	1	4
3	2	3	0	2	5
4	3	1	2	0	3
5	6	4	5	3	0

- 則從城市 1 出發，騎完所有城市的距離最短是，經由 13245 的距離 $= 2+3+1+3=9$ 。

輸入說明

第 1 筆資料是 N ，

第 2 筆資料是第 1 個城市和其他城市的公路距離。

第 3 筆資料是第 2 個城市和其他城市的公路距離。

....

第 $N+1$ 筆資料是第 N 個城市和其他城市的公路距離。

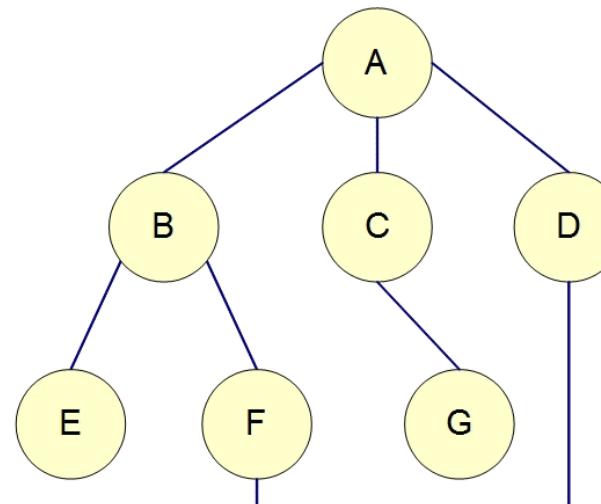
輸出說明

輸出從第 1 個城市出發，騎完所有城市最短距離。

圖論 – 廣度優先搜尋法

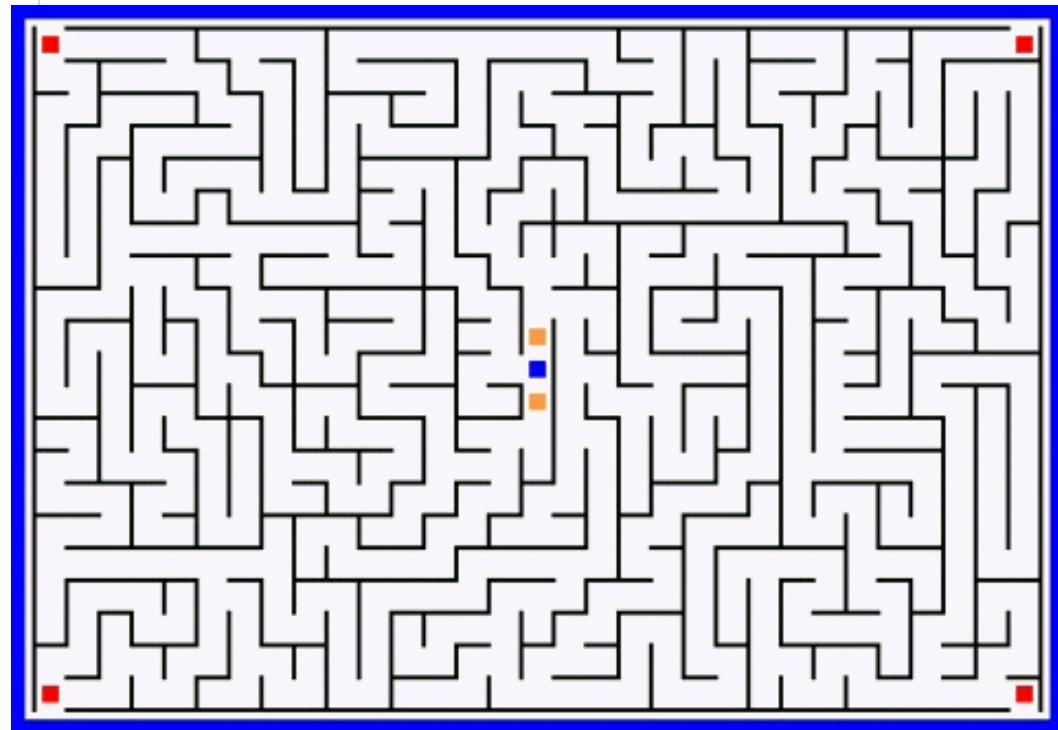
□ 廣度優先搜尋法 (Breadth-first Search)

- 從圖某節點(vertex, node)開始走訪，接著走訪此節點所有相鄰且未拜訪過的節點，
- 由走訪過節點繼續進行先廣後深搜尋。把同一深度(level)節點走訪完，再繼續向下個深度搜尋，直到找到目的節點或遍尋全部節點。
- 廣度優先搜尋法屬於盲目搜索(uninformed search)，可利用佇列(Queue)處理。



圖論 – 廣度優先搜尋法

```
procedure BFS(vertex s) {  
    create a queue Q  
    enqueue s onto Q  
    mark s as visited  
    while Q is not empty {  
        dequeue a vertex from Q into v  
        for each w adjacent to v {  
            if w unvisited {  
                mark w as visited  
                enqueue w onto Q  
            }  
        }  
    }  
}
```



圖論 – 廣度優先搜尋法

```
#include<stdio.h>
#define Max 100
#define N 6
void printMaze(int maze[N][N]){
    int i=0,j=0;
    for(i=0;i<N;i++){
        for(j=0;j<N;j++) printf("%2d ",maze[i][j]);
        printf("\n");
    }
    int isempty(int data[Max], int fe[]){
        if(fe[0]==fe[1]) return 1; //if(front==back) return 1;
        return 0;
    }
    void enqueue(int data[Max], int fe[],int x){
        if((fe[1]+1)%Max != fe[0]){ //if((back+1)%Max != front)
            fe[1]=(fe[1]+1)%Max; //back=(back+1)%Max;
            data[fe[1]]=x; // data[back]=x;
        }
    }
}
```

圖論 – 廣度優先搜尋法

```
int dequeue(int data[Max], int fe[]){
    if(isempty(data, fe)==0){
        fe[0]=(fe[0]+1)%Max;      //front=(front+1)%Max;
        return data[fe[0]];
    }
}
void find_path(int maze[N][N]){
    int x=4,y=4;
    while(1){
        printf("(%d,%d)\n", x, y);
        if (x==1 && y==1) break;
        if(maze[x+1][y]>maze[x][y])  x = x+1;
        else if(maze[x-1][y]>maze[x][y])  x = x-1;
        else if(maze[x][y+1]>maze[x][y])  y = y+1;
        else if(maze[x][y-1]>maze[x][y])  y = y-1;
    }
}
```

圖論 – 廣度優先搜尋法

```
void go_search(int maze[N][N], int data[], int fe[]){  
    int r=0,x=0,y=0;  
    int level =20;  
    while(isempty(data, fe)==0){  
        r = dequeue(data, fe);  
        x = r/10;  
        y = r%10;  
        maze[x][y]=level--;           //printf("(%d,%d)\n", x, y);  
        if(x==4 && y==4){      break;      }  
        else{  
            if(maze[x+1][y]==0){      enqueue(data, fe,(x+1)*10+y);      }  
            if(maze[x-1][y]==0){      enqueue(data, fe,(x-1)*10+y);      }  
            if(maze[x][y+1]==0){      enqueue(data,fe, (x)*10+y+1);      }  
            if(maze[x][y-1]==0){      enqueue(data,fe, (x)*10+y-1);      }  
        }  
    }  
}
```

圖論 – 廣度優先搜尋法

```
test020{
    int maze[N][N]={
        {1,1,1,1,1},
        {1,0,1,0,1},
        {1,0,0,0,1},
        {1,0,0,0,1},
        {1,1,1,1,1}
    };
    int data[Max];
    int fe[2]={0,0};
    enqueue(data, fe, 11);
    go_search(maze, data, fe);
    printMaze(maze);
}
```

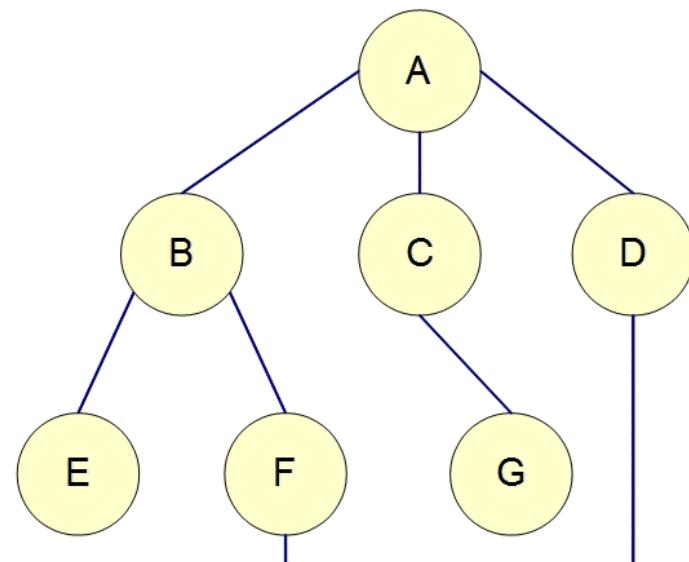
```
test01(){
    int maze[N][N]={
        {1,1,1,1,1,1},
        {1,0,1,0,0,1},
        {1,0,1,0,1,1},
        {1,0,0,0,1,1},
        {1,0,1,0,0,1},
        {1,1,1,1,1,1}
    };
    int data[Max];
    int fe[2] = {0, 0};
    enqueue(data, fe, 11);
    go_search(maze, data, fe);
    printMaze(maze);
    find_path(maze);
}

int main(){
    test01();
}
```

圖論 – 深度優先搜尋法

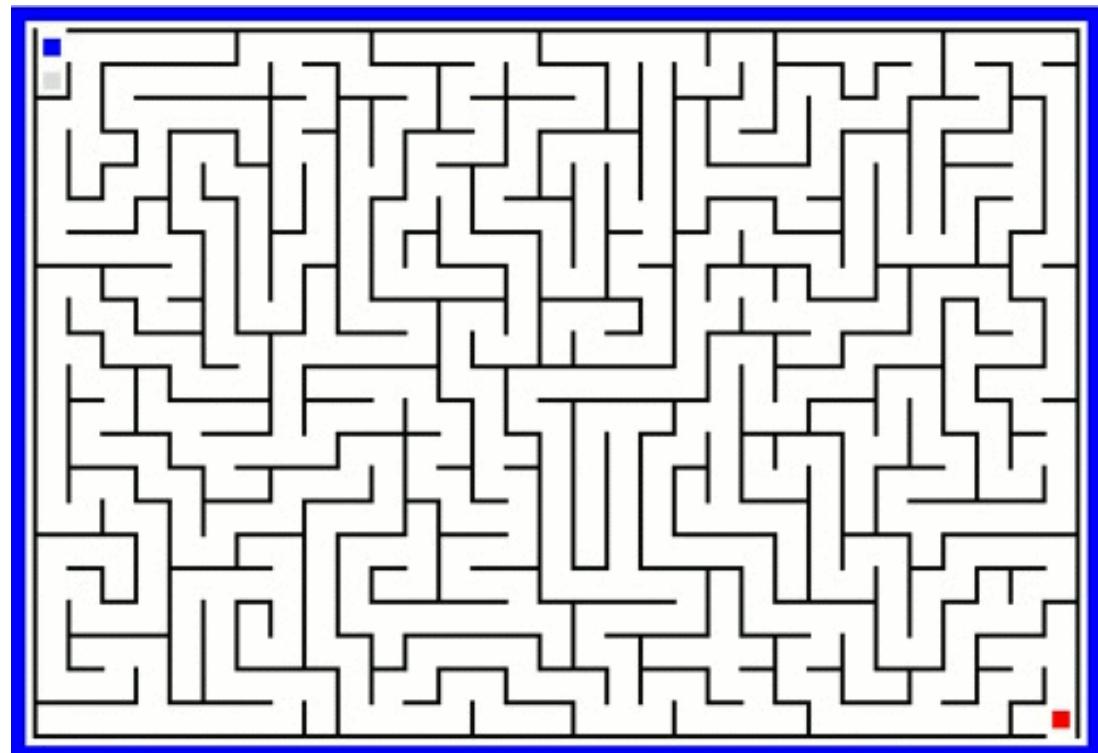
□ 深度優先搜尋法 (Depth-first Search)

- 從圖某節點開始走訪，先探尋邊(edge)上未搜尋的一節點，並儘可能深的搜索，直到該節點的所有邊上節點都已探尋；
- 回溯(backtracking)到前一節點，重覆探尋未搜尋節點，直到找到目的節點或遍尋全部節點。
- 屬盲目搜索，利用堆疊(Stack)處理



圖論 – 深度優先搜尋法

```
procedure dfs(vertex v) {  
    mark v as visited  
    for each w adjacent to v {  
        if w unvisited {  
            dfs(w)  
        }  
    }  
}
```



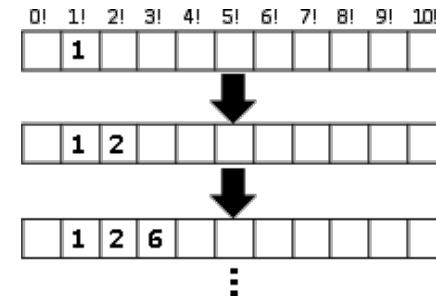
動態程式規劃

□ 動態程式規劃(Dynamic Programming)

○ 階乘 (Factorial)

```
void factorial(int f[10], int N) {  
    int i=0;  
    f[0] = 0;  f[1] = 1;  
    for (i=2; i<=N; ++i) {  
        f[i] = f[i-1] * i;  
    }  
}
```

```
void factorial(int N) {  
    int i=0, f=1;  
    for (i=2; i<=N; ++i) {  
        f = f * i;  
    }  
}
```



動態程式規劃

□ 動態規劃是分治法的延伸。

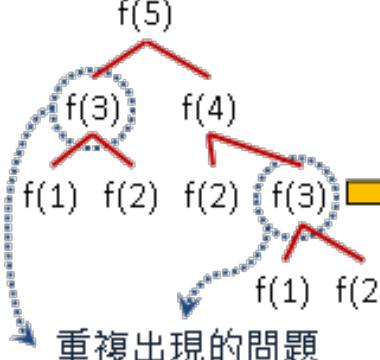
- 當遞迴分割出來的問題，一而再、再而三出現，就運用記憶法儲存這些問題的答案，避免重複求解，以空間換取時間。
- 規劃的過程是反覆讀取資料、計算、儲存資料。
- 時間複雜度 $O(N)$
- 空間複雜度 $O(N)$

```
int f(int n) {  
    if (n == 0 || n == 1)      return 1;  
    else          return f(n-1) + f(n-2);  
}
```

Recurrence

$$f(n) = \begin{cases} 1 & , \text{if } n = 1 \\ 2 & , \text{if } n = 2 \\ f(n-2) + f(n-1) & , \text{if } n \geq 3 \end{cases}$$

Divide and Conquer



Memoization

$f(1)$	$f(2)$	$f(3)$	$f(4)$	$f(5)$	$f(6)$
1	2	3	5	8	

記錄答案

重複出現的問題

動態程式規劃

- 動態規劃是分治法的延伸。

```
int f(int n, int s[]) {  
    if (n == 0 || n == 1) return 1;  
    // 用 0 代表該問題還未計算答案  
    if (s[n]) return s[n];  
    return s[n] = f(n-1, s) + f(n-2, s);  
}  
void stairs_climbing(){  
    int stairs[20];  
    for (int i=0; i<=20; i++) {  
        stairs[i] = 0;  
    }  
    printf("%d", f(15, stairs));  
}
```

貪婪演算法

- 貪婪演算法 (Greedy Algorithm) - 換零錢遊戲
 - 有 71 個 1 元，幣值分別為 29 元、22 元、5 元、1 元，請用最少零錢個數兌換零錢。
 - 局部解：29 元 2 個，22 元 0 個，5 元 2 個，1 元 3 個， $2+2+3=7$
- 動態程式規劃
 - 最佳解：29 元 0 個，22 元 3 個，5 元 1 個，1 元 0 個， $3+1=4$
- 貪婪演算法不一定是最佳解，但效率高
 - 一種短視/近利/貪婪的想法，每一步都不管大局，只求局部解決
 - 透過一步步的選擇局部最佳解來得到問題解答。
 - 每個選擇是根據某種準則決定，前次決定不會影響後次決定。
- 動態規劃演算法可以求出最佳解，但效率略差

動態程式規劃

□ 71元，最佳解：29元0個，22元3個，5元1個，1元0個，共4

○ $f(n) = \min(1+f(n-29), 1+f(n-22), 1+f(n-5), 1+f(n-1))$

○ 71元以29元兌換，剩 $71-29=42$ ，總兌換數=42元可兌換個數+1

○ $f(0)=0, f(n) = 1 + \min(f(n-c_1), f(n-c_2), \dots, f(n-c_k))$

○ $n > c_i, 1 < i < k, c_i$ 是硬幣面額， k 是總共有幾種面額

○ $c_1=29, c_2=22, c_3=5, c_4=1$

○ $f(1)=1, f(2)=1+f(1)=2, f(3)=1+\min(f(2))=3, f(4)=1+\min(f(3))=4$

○ $f(5) = 1+\min(f(5-5), f(5-1)) = 1+\min(f(0), f(4)) = 1$

➤ 用一個1元換；或用一個5元換；之後可以如何換最少。

○ $f(6) = 1+\min(f(6-5), f(6-1)) = 1+\min(f(1), f(5)) = 1+\min(1, 1) = 2$

○ $f(7) = 1+\min(f(7-5), f(7-1)) = 1+\min(f(2), f(6)) = 1+\min(2, 2) = 3$

○ $f(8) = \dots$

○ 要宣告 int $f[n]$ 空間，換取計算時間，並計算各種可能性。

動態程式規劃

```
#include <stdio.h>
int f(int n, int coinType[], int k) {
    int p=0, i=0, coin=0, min_coin=0;
    int min_number[100]={0}, min_first_element[100];
    for (p=0; p<100; p++) min_number[p]=0;
    for (p=1;p<=n;p++){
        min_coin = n;
        for(i=0;i<k;i++){
            coin = coinType[i];
            if (((p-coin)>=0)&&((1+min_number[p-coin])<min_coin))
                min_coin = 1 + min_number[p-coin];
        }
        min_number[p] = min_coin;
        min_first_element[p] = coin;
    }
    for (p=1; p<=n; p++) {
        printf("(%d, %d)\n", p, min_number[p]);
    }
}
```

```
int main(){
    int coinType[10]={29, 22, 5, 1};
    int k=4;
    //int coinType[10]={5, 4, 2, 1};
    int n =71; //8
    f(n, coinType, k);
    return 0;
}
```

動態程式規劃-TSP

□ TSP問題（Travelling Salesman Problem）

○一個商人要拜訪n個城市，路徑選擇限制是每個城市只拜訪一次，且最後回到原來出發的城市。目標要求最小值。

○將有向圖轉為鄰接矩陣

➤ V 表示所有頂點 $\{v_0 \sim v_{n-1}\}$ 集合，例如 $\{v_0, v_1, v_2\}$ 。

➤ S 表示 V 的子集，例如 (v_1, v_2) 。

➤ $d[i][j]$ 表示從 v_i 到 v_j 的直接距離，例如 $d[0][1]$ 。

➤ $R[v_0][S]$ ，從頂點 v_0 經 S 所有頂點回到 v_0 ，且僅經一次，最短距離。

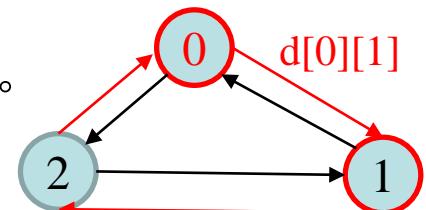
– 假設 v_1 是 v_0 的下個經過頂點，則最短距離，則

$$- R[v_0][S] = d[0][1] + R[v_1][S-v_1]; \text{ 其中 } S=\{1, 2\}, S-v_1=\{2\}$$

$$- R[v_0][\{1, 2\}] = d[0][1] + R[v_1][\{2\}]$$

– $R[v_1][S-v_1]$ ，從頂點 v_1 經 $(S-v_1)$ 所有頂點回到 v_0 ，且僅經一次，最短距離，

$$- R[v_1][\{2\}] = d[1][2] + R[v_2][\{\}]$$



動態程式規劃-TSP

□ TSP是以下三種方案，選最小值(最優結果)

- 0出發，到1，再從1出發，經過{2}城市，回到0花費最少。
- 0出發，到2，再從2出發，經過{1}城市，回到0花費最少。

□ 最優結果，記錄在R表內，避免重複計算

- 設計一個二維的動態規劃表R, R[0]{1, 2}表示從0出發，經過1, 2回到0花費最少。
- 上面二方案最小值

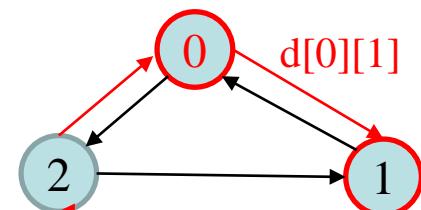
$$\triangleright R[0][\{1,2\}] = \min\{ d[0][1]+R[1]\{2\}, d[0][2]+R[2]\{1\},$$

$$\triangleright R[1]\{2\} = \min\{ d[1][2]+R[2]\{\}, d[1][2]+d[2][0] \}$$

$$\triangleright R[2]\{1\} = \min\{ d[2][1]+R[1]\{\}, d[2][1]+d[1][0] \}$$

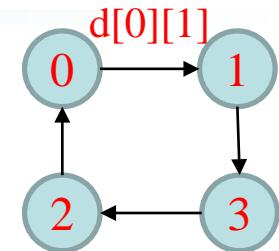
– $R[2]\{\}$ ，從2沒有經過任何點到0, $d[2][0]$

– $R[1]\{\}$ ，從1沒有經過任何點到0, $d[1][0]$



動態程式規劃-TSP

- $R[0]\{\{1,2,3\}\} = \min\{ d[0][1]+R[1]\{2,3\},$
 $d[0][2]+R[2]\{1,2\},$
 $d[0][3]+R[3]\{1,3\} \}$



- $R[1]\{2,3\} = \min\{ d[1][2]+R[2]\{3\},$
 $d[1][3]+R[3]\{2\} \}$
- $R[2]\{3\} = \min\{ d[2][3]+R[3]\{\}, \} = d[2][3]+d[3][0]$

○ $R[v_0][S]$ ，從 v_0 經 S 所有頂點回到 v_0 ，且僅經一次，最短距離。

- 等於從 v_0 到 v_j 的距離，加上從 v_j 經 S 中所有點到 v_0 的最短距離，的最小值（其中 v_j 從 S 中取，且是最短距離的點）

○ $R[v_i][S]$ ，從 v_i 經 S 所有頂點回到 v_0 ，且僅經一次，最短距離。

- 等於從 v_i 到 v_j 的距離，加上從 v_j 經 $S-v_j$ 中所有點到 v_0 的最短距離，的最小值（其中 v_j 從 S 中取，且是最短距離的點）

- $R[v_i][S]=\min\{d[i][j]+R[v_j][S-v_j]\}(v_j \in S)$ ，

- 初始化， $v=v_0$ ， $S=\{v_1 \sim v_{n-1}\}$

動態程式規劃-TSP

□ TSP問題 (Travelling Salesman Problem)

```
#include <stdio.h>           #include <string.h>
#define MAX_N 4                #define INF 0x3f3f3f3f
void print(int R[][1<<MAX_N]) {
    for (int i=0; i<(1<<MAX_N); i++)
        for (int j=0; j<MAX_N; j++) printf("%d ", R[i][j]);
    printf("\n");
}
int minValue(int x, int y) { return (x>y?y:x); }
int Rec(int R[][1<<MAX_N], int d[][MAX_N], int v,int S, int n){
    int t=0, ans = INF;
    if(R[v][S] >= 0) { return R[v][S]; } //已經計算過有紀錄，不用再計算
    for(int i = 0 ; i < n ; i++) {
        if(!(S>>i&1)) {           //集合 S，{0, 1, 2, 3}哪一個存在(i)，不為0
            t = Rec(R, d, i, S|(1<<i),n); //集合 S，去掉存在的那一個，第i個設為1
            ans = minValue(ans,d[v][i]+ t);
        }
    }
    return R[v][S] = ans;
}
```

動態程式規劃-TSP

□ TSP問題 (Travelling Salesman Problem)

```
void solve() {                                // {0, 1, 2, 3}編碼 0000
    int v = 0, S=(0|1<<v);                // 從v=0出發，經過{1, 2, 3}->{0001}
    int R[MAX_N][1<<MAX_N];               // 從v=1出發，經過{0, 2, 3}->{0010}
    int d[MAX_N][MAX_N] = {
        {0, 3, 7, 3},
        {4, 0, 3, 6},
        {6, 2, 0, 3},
        {8, 3, 4, 0}};
    memset(R,-1,sizeof(R));
    for (int i=0; i<MAX_N; i++)           // R[1][空集合 1111]=d[1][0]
        R[i][(1<<MAX_N)-1]=d[i][0];      // R[0][空集合 1111]=d[0][0]=0
    printf("%d\n", Rec(R, d, v, S, MAX_N));
}
```

```
int main() {    solve();    return 0; }
```

動態程式規劃-TSP

□ TSP問題（Travelling Salesman Problem）

```
void travel(int n,const number W[][],index P[][],number& minlenth) {  
    index i,j,k;  
    number D[n][V-{v1}];  
    for(i=2;i<=n;i++)  
        D[i][空集]=W[i][1];//從頂點vi到v1的直接距離  
    for(k=1;k<=n-2;k++)//遍歷A的每個子集  
        for(包含k個頂點的所有子集A屬於V-{vi}) //從小到大遍歷  
            for(滿足i != 1且vi不在v中的i)//算出任意點出發到v1結束的最短距離 {  
                D[i][A]=min{W[i][j]+D[j][A-{vj}]};//vj∈A  
                P[i][A]=最小值的j值;  
            }  
    D[1][V-{vi}]=min{W[i][j]+D[j][A-{v1 , vj}]};  
    //2<=j<=n 從v1出發到v1結束的最短距離  
    P[1][V-{vi}]=最小值的j值；  
    minlenth=D[1][V-{vi}];  
}
```

計算機程式設計

C語言 String

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

字串宣告與初值設定

□ 字元以單引號包圍，而字串則是以雙引號包圍：

- 'a' /* 這是字元常數 a */
- "a" /* 這是字串常數 a */
- "Sweet home" /* 這是字串常數 Sweet home */

char 字元陣列名稱[陣列大小] = 字串常數;

char str[]="Sweet home";

0	1	2	3	4	5	6	7	8	9	10
S	w	e	e	t		h	o	m	e	\0

字元與字串之比較

- 字元以單引號包圍，而字串則是以雙引號包圍：

```
01 #include <stdio.h>
02 int main() {
03     char ch = 'a';
04     char str1[] = "a";
05     char str2[] = "Sweet home";
06     printf("ch size=%d\n", sizeof(ch));
07     printf("str1 size=%d\n", sizeof(str1));
08     printf("str2 size=%d\n", sizeof(str2));
09 }
```

ch size=1
str1 size=2
str2 size=11

字串輸入與輸出函數

□ gets() 與 puts()

- gets(字元陣列名稱);
- puts(字元陣列名稱); 或 puts(字串常數);

```
01 #include <stdio.h>
02 int main() {
03     char name[15];
04     puts("What's your name?");
05     gets(name);
06     puts("Hi!");
07     puts(name);
08 }
```

What's your
name?
JJ
Hi!
JJ

字串輸入與輸出函數

□ 大小寫的轉換

```
01 #include <stdio.h>
02 void toUpper(char s[]){
03     int i = 0;
04     while(s[i] != '\0'){
05         if(s[i]>=97 && s[i]<=122){
06             s[i] -= 32;
07             i++;
08         }
09     }
10 }
11 int main() {
12     char str[15];
13     puts("請輸入一個字串");
14     gets(str);
15     toUpper(str);
16     printf("轉成大寫後: %s", str);
17     return 0;
18 }
```

請輸入一個字串
abcdefg
轉成大寫後:
ABCDEFG

Exercise

- 大小寫的轉換 – 若未知英文字元編碼編號，但已知a~z 依序編碼, A~Z 依序編碼。

```
01 #include <stdio.h>
02 void toUpper(char s[]){
03     int i = 0;
04     while(s[i] != '\0'){
05         if(s[i]>=97 && s[i]<=122){
06             s[i] -= 32;
07             i++;
08         }
09     }
10 }
```

```
11 int main() {
12     char str[15];
13     puts("請輸入一個字串");
14     gets(str);
15     toUpper(str);
16     printf("轉成大寫後: %s", str);
17     return 0;
18 }
```

請輸入一個字串
abcdefg
轉成大寫後:
ABCDEFG

字串陣列

- 字串陣列的宣告
 - char 字元陣列名稱[字串的個數][字串長度];
- 字串陣列的宣告與初值設定
 - char 字元陣列名稱[字串的個數][字串長度]= {"字串常數1", "字串常數2", ..., "字串常數n"};

```
char customer[6][15];
char S[3][10]={ "Tom","Lily","James Lee"};
```

字串陣列元素的存取

□ 字串陣列

S[0]	0022FE20
S[1]	0022FE2A
S[2]	0022FE34

0	1	2	3	4	5	6	7	8	9
T	o	m	\0						
L	i	l	y	\0					
J	a	m	e	s	\0				

```
01 #include <stdio.h>
02 int main() {
03     char S[3][10] = {"Tom", "Lily", "James"};
04     int i;
05     for(i=0; i<3; i++)
06         printf("S[%d]=%s\n", i, S[i]);
07     printf("\n");
08     for(i=0; i<3; i++){
09         printf("S[%d]=%p\n", i, S[i]);
10         printf("S[%d][0]=%p\n", i, &S[i][0]);
11     }
12 }
```

S[0]=Tom

S[1]=Lily

S[2]=James

S[0]=0022FE20

S[0][0]=0022FE20

S[1]=0022FE2A

S[1][0]=0022FE2A

S[2]=0022FE34

S[2][0]=0022FE34

複製字串陣列

□ 字串陣列

```
01 #include <stdio.h>
02 #define MAX 3
03 #define LENGTH 10
04 int main() {
05     char S1[MAX][LENGTH] = {"Tom", "Lily", "James"};
06     char S2[MAX][LENGTH];
07     int i, j;
08     for(i=0; i<MAX; i++){
09         for(j=0; j<LENGTH; j++){
10             if(S1[i][j] == '\0') break;
11             else S2[i][j] = S1[i][j];
12         }
13         S2[i][j]='\0';
14     }
15     for(i=0; i<MAX; i++)
16         printf("S2[%d]=%s\n", i, S2[i]);
17 }
```

S2[0]=Tom
S2[1]=Lily
S2[2]=James

字串轉換函數

函數	說明
double atof(char *)	將參數字串轉換成浮點數，如果字串不能轉換傳回0.0
int atoi(char *)	將參數字串轉換成整數，如果字串不能轉換傳回0
long atol(char *)	將參數字串轉換成長整數，如果字串不能轉換傳回0

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main() {
04     char buffer1[] = "2.675";
05     char buffer2[] = "1234";
06     double f = atof(buffer1);
07     int n = atoi(buffer2);
08     printf("%0.3f\n", f);
09     printf("%d\n", n);
10     return 0;
11 }
```



字串處理函數

函數	說明
size_t strlen(char *)	傳回參數字串的長度
char *strcpy(char *d, char *s)	將參數字串s複製到字串d，傳回字串d的指標
char *strncpy(char *d, char *s, size_t n)	將參數字串s複製最多n個字元到字串d，傳回字串d的指標，需要自行加上字串結束字元'\0'
char *strcat(char *d, char *s)	將參數字串s連接到字串d之後，傳回字串d的指標
char *strncat(char *d, char *s, size_t n)	將參數字串s連接最多n個字元到字串d之後，傳回字串d的指標
int strcmp(char *d, char *s)	比較參數字串s與字串d，d < s傳回負值；d = s傳回0；d > s傳回正值
int strncmp(char *d, char *s, size_t n)	比較參數字串s與字串d的前n個字元，d < s傳回負值；d = s傳回0；d > s傳回正值
char *strchr(char *d, char c)	傳回指標指向在參數字串d中第1次出現字元c的位置，如果沒有傳回NULL
char * strrchr(char *d, char c)	傳回指標指向在參數字串d中最後1次出現字元c的位置，沒有傳回NULL
char * strstr(char *d, char *s)	傳回指標指向在參數字串d中第1次出現字串s的位置，沒有傳回NULL

字串處理函數 – 複製

```
01 #include <stdio.h>
02 #include <string.h>
03 int main() {
04     char str[50];
05     char str2[50] = "I am John!";
06     int len;
07     strcpy(str, "Hello World!");
08     len = strlen(str);
09     printf("%s(size=%d)", str, len);
10     memset(str, '\0', sizeof(str));
11     strncpy(str, str2, 4);
12     len = strlen(str);
13     printf("%s(size=%d)", str, len);
14 }
```



Hello World!(size=12)
I am(size=4)

字串處理函數 - 複製

□ 自訂複製函式

```
#include <stdio.h>
#include <string.h>
void myStrcpy(char * s1, char *s2) {
    for (;(*s1=*s2)!='0';s1++, s2++);
}
int main() {
    char str[50];
    char str2[50] = "I am John!";
    myStrcpy(str, "Hello World!");
    printf("%s\n", str);
    myStrcpy(str, str2);
    printf("%s\n", str);
    return 0;
}
```



Hello World!
I am John!

Exercise

□ 自訂strlen計算字串長度函式

```
#include <stdio.h>
#include <string.h>
int myStrlen(char * s) {
    int r=0;
    for (;*s]!='\0';s++) {
        }
    return r;
}
int main() {
    char str[50] = "I am John!";
    int len = myStrlen("Hello World!");
    printf("%d\n", len);
    len = myStrlen(str);
    printf("%d\n", len);
    return 0;
}
```

Exercise

- 自訂strlen計算字串長度函式，遞迴

```
#include <stdio.h>
#include <string.h>
int myStrlen(char * s) {
    int r=0;

    return r;
}
int main() {
    char str[50] = "I am John!";
    int len = myStrlen("Hello World!");
    printf("%d\n", len);
    len = myStrlen(str);
    printf("%d\n", len);
    return 0;
}
```

字串處理函數 – 比較

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[10]="1234";
    char str2[10]="123456";
    int r = strcmp(str1,str2);
    printf("result=%d\n", r);
    r = strncmp(str1,str2,4);
    printf("result=%d\n", r);
}
```



result=-1
result=0

字串處理函數 – 比較

□ 自訂比較函式

```
#include <stdio.h>
#include <string.h>
int mystrcmp(char *s1, char *s2) {
    for (;(*s1]!='\0')&&(*s2]!='\0');s1++, s2++) {
        if (*s1>*s2) return 1;
        else if (*s1<*s2) return -1;
    }
    if (*s1!='\0') return 1;
    else if (*s2!='\0') return -1;
    return 0;
}
int main() {
    char str1[10]="1234";
    char str2[10]="123";
    int r = mystrcmp(str1,str2);
    printf("result=%d\n", r);
}
```

Exercise

□ 自訂比較函式-遞迴

```
#include <stdio.h>
#include <string.h>
int myStrcmp(char *s1, char *s2) {
    if (*s1 < *s2)
        return -1;
    else if (*s1 > *s2)
        return 1;
    else if (*s1 == '\0' && *s2 == '\0')
        return 0;
    else
        return myStrcmp(s1 + 1, s2 + 1);
}

int main() {
    char str1[10] = "1234";
    char str2[10] = "123";
    int r = myStrcmp(str1, str2);
    printf("result=%d\n", r);
}
```

字串處理函數-切割

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[]="00:22:33:4B:55:5A";
    char *delim = ":";  
    char * pch;
    pch = strtok(str,delim);
    while (pch != NULL) {
        printf ("%s\n",pch);
        pch = strtok (NULL, delim);
    }
}
```



```
00
22
33
4B
55
5A
```

Exercise

□ 自訂切割函式

```
#include <stdio.h>
#include <string.h>
char* myStrtok(char *s1, char *s2) {
    static char *p;
    if (s1==NULL) s1 = p;
    else p = s1;
    while (*p!=(*s2) || *p!="\0") {
        p++;
    }
    if (*p=='\0') return NULL;
    p++;
    return s1;
}
```

```
int main() {
    char str[]="00:22:33:4B:55:5A";
    char *delim = ":";
    char * pch;
    pch = strtok(str, delim);
    while (pch != NULL) {
        printf ("%s\n",pch);
        pch = strtok (NULL, delim);
    }
}
```

字串交換

- 使用區域變數指標，無法達到交換字串目的。
 - 必須使用指標的指標

```
#include<stdio.h>
void swap(char *str1, char *str2) {
    char *temp = str1;
    str1 = str2;
    str2 = temp;
}
int main() {
    char *str1 = "geeks";
    char *str2 = "forgeeks";
    swap(str1, str2);
    printf("str1 is %s, str2 is %s", str1, str2);
    return 0;
}
```

```
void swap1(char **str1_ptr, char **str2_ptr) {
    char *temp = *str1_ptr;
    *str1_ptr = *str2_ptr;
    *str2_ptr = temp;
}
```

Homework II

□ 英文字分析、取代、插入、刪除

○ 輸入一篇英文文章 A，文章中英文字以一個空白間隔。另外輸入 2 個英文字(word) P、Q。

- (1) 將文章 A 中 P 字串以 Q 字串取代，並輸出。
- (2) 在文章 A 中 P 字串前插入 Q 字串，並輸出。
- (3) 將文章 A 中 P 字串刪除，並輸出。
- (4) 分析文章 A 所有英文字 (word) 的頻率，依頻率由大自小排序，頻率相同則以 word 由小自大排序 (That > This....) 輸出。

○ 輸入範例說明：

- 第一行，文章 A
- 第二行，英文字 P
- 第三行，英文字 Q

Homework II

○輸出範例說明：

- 第一行，文章 A 將 P 替換成 Q。
- 第二行，文章 A 將 Q 插入 P 前面。
- 第三行，文章 A 將 P 刪除。
- 第四行之後，每一行依序為英文字、出現頻率次數，中間以逗號
間隔。

Sample Input

```
This is a book That is a cook  
is  
was
```

Sample Outpt

```
This was a book That was a cook  
This was is a book That was is a cook  
This a book That a cook  
a 2  
is 2  
That 1  
This 1  
book 1  
cook 1
```

範例一 互補字串I

□ 互補字串

- 互補字串 S_1, S_2 的定義是字串 S_1, S_2 沒有重複出現的字元，且 S_1 和 S_2 內的字元包含所有前 M 個字元，字元由 'A' 開始。此處，字串是一個集合，亦即，元素有重複只算一個，也不管排列情況。例如 AABAAB 與 ABAB 與 BABA 都是相同的字串。輸入 N 個字串，輸出這 N 個字串互補的個數。
- 輸入 (m, n) n 個字串
 - 7 4
 - AABAAB
 - ABCABCDE
 - CDECD
 - GFFGF

○ 輸出

➤ 4

範例—互補字串I

□ 互補字串

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int isIn(char *s, char c, int n) { //判斷c 是否在 s 裡面
    int i=0;
    for (i=0; i<n; i++)
        if (c==s[i]) return 1;
    return 0;
}
char *erase(char *s, int m) { //將重複的部分刪除，變成集合
    int i=0, j=0;
    char *p = (char*) malloc(strlen(s)*sizeof(char));
    char ch = 'A' + m -1;
    for (; s[i]!='\0'; i++)
        if ((s[i]<=ch)&&(isIn(p, s[i], j)==0)) p[j++] = s[i];
    p[j]='\0';
    return p;
}
```

範例—互補字串I

□ 互補字串

```
int isDuplicat(char *s1, char *s2) { //判斷 s1, s2 是否有重複字元
    int i=0, j=0;
    printf("%s, %s\n", s1, s2);
    for (i=0; s1[i]!='\0'; i++) {
        for (j=0;s2[j]!='\0'; j++) {
            if (s1[i]==s2[j]) return 1;
        }
    }
    return 0;
}
int isFitLength(char *s1, char *s2, int m) { // 判斷s1+s2的長度是否剛好符合
    if (m==(strlen(s1)+strlen(s2))) return 1;
    else return 0;
}
```

範例—互補字串I

□ 互補字串

```
void compute(char s[][10], int n, int m) {  
    char *p[10];  
    int i=0, j=0, sum=0;  
    for (i=0; i<n; i++) p[i] = erase(s[i], m);  
    for (i=0; i<n; i++) {  
        for (j=i+1; j<n; j++) { // p[i], p[j] 沒有重複，總和長度又等於 m  
            if ((isDuplicate(p[i], p[j])==0) && (isFitLength(p[i], p[j],m)))  
                sum++;  
        }  
    }  
    printf("sum = %d\n", sum);  
}  
void test01() {  
    char s[][10] = {"AABAAB", "ABCDE", "CDECD", "GFFGF"};  
    compute(s, 4, 7);  
    char s1[][10] = {"ABB", "AB", "C", "CC"};  
    compute(s1, 4, 3);  
}
```

範例—互補字串II

□ 互補字串 C++ STL

```
#include <iostream>
#include <string.h>
#include <map>      // map 函式庫
#include <algorithm> // sort 函式庫
#define F first
#define S second
using namespace std;
int solve(int m, int n, char s[][10]) {
    map < string, int > lib;
    int ans = 0;
    string str, basic;
    char c = 'A';
    for ( int i = 0 ; i < m ; i++, c++ )
        basic += c;    //前 m 個形成的基本集合
    cout<<basic<<endl;
```

範例—互補字串II

□ 互補字串C++ STL

```
while ( n-- ){
    //cin >> str;
    str.assign(s[n]);
    cout<<str<<endl;
    sort ( str.begin(), str.end() ); // 排序統一順序
    //unique 會將重複放在最後面，回傳n為前n個字元沒重複
    //erase 會刪除(i,j) i~j字元，兩者配合，可刪除重複字元
    str.erase ( unique ( str.begin(), str.end() ), str.end() );
    lib[str]++; //放入map，計數加一
}
for (auto j:lib) { //一一尋訪 map lib
    cout<<"j:( "<<j.F<<","<<j.S<<") "<<endl;
    for ( auto i: j.F ) //針對尋訪到的 map 元素，取 first，一一尋訪
        cout<<"i:"<<i<<endl;
}
// 例如 {{"ABC", 1},{CD}} 會取出 ABC，再取出 A, B, C ,
// 第二次取出 CD，再取出 C, D
```

範例—互補字串II

□ 互補字串C++ STL

```
for ( auto j: lib ){
    str = basic;
    // 把有出現從基本集合中刪除，即其互補，
    // 比對是否有互補資料，將出現個數相乘，即配對個數
    for ( auto i: j.F )
        str.erase ( lower_bound ( str.begin(), str.end(), i ) );
    ans += j.S * lib[str];
}
return (ans/2); // 重複計算兩次
}
int main(){
ios::sync_with_stdio ( false );
cin.tie ( 0 );
cout.tie ( 0 );
int n = 4, m = 3; //cin >> n >> m;
char s[][10] = {"ABB", "AB", "C", "CC"};
cout<<solve(m, n, s);
return 0; }
```

範例—互補字串III

□ 互補字串C

```
#include <stdio.h> #include <string.h>
void print(int n) {
    for (;n>0; n=n/2)  printf("%d ", n%2);
    printf("\n");
}
int encode(char *s) { // 編碼 A***E = 10001
    int i=0, code=0; // 向左位移，重複不算，使用 |
    for (i = 0; i != strlen(s); ++i)  code |= (1LL << (s[i] - 'A'));
    return code;
}
int match(int code[], int c, int n, int mask) {
    int answer =0;
    for (int i=0; i<n; i++) {
        //判斷互補 10001 + 01110 = 11111
        // 10001 ^ 11111 = 01110 (XOR)
        //if ((code[i]+c)==mask) answer++;
        if (code[i]==(c^mask)) answer++;
    }
    return answer;
}
```

範例—互補字串III

□ 互補字串C

```
int solve(int charSet, int amount, char s[][][10]){
    int basic = 0, answer=0, i=0, code[10];
    // 基礎集合編碼 ABCDE = 11111 , 1 向左移後累加
    for (int i = 0; i < charSet; ++i)  basic += (1LL << i);
    print(basic);
    while (i<amount) {
        code[i] = encode(s[i]);
        print(code[i]);
        // 目前處理的輸入，跟之前的比對互補
        answer+= match(code, code[i], i, basic);
        i++;
    }
    return answer;
}
```

範例—互補字串III

□ 互補字串C

```
int main() {  
    char s1[][10] = {"ABB", "AB", "C", "CC"};  
    printf("%d\n", solve(3, 4, s1));  
    char s2[][10] = {"AABAAB", "ABCABCDE", "CDECD", "GFFGF"};  
    printf("%d\n", solve(7, 4, s2));  
    return 0;  
}
```

Homework I—互補字word

□ 互補字串

- 互補字 S_1, S_2 的定義是字串 S_1, S_2 沒有重複出現的字。字串是英文字的一個集合，亦即，元素有重複只算一個，也不管排列情況。例如 "Happy Happy Day" 與 "Day Happy Day" 是相同的字串。輸入 N 個字串，輸出這 N 個字串互補的個數。
- 輸入 (m, n) n 個字串
- 輸出

指向指標的指標

處理字串

- 可用來指向任意個任意長度的字串
- compiler 編譯函數的指向指標的指標時，會給足夠空間

```
#include <stdio.h>
void f() {
    char **p;
    char *pa[4]={"YOUD", "ME", "HI", "STAR"};
    p = &pa[0];           //同 p = pa
    printf("%c\n", **p);   //Y
    printf("%s\n", *p);    //YOUD
    printf("%s\n", *(p+1)); //ME
    printf("%c\n", *(*p+1)); //O
    printf("%s\n", *p+2);   //UD
    printf("%c\n", *(*(p+2)+1)); //I
}
```

			+0	+1	+2	+3	
*(p+0)	*(pa+0)	pa[0]	Y	O	U	D	\0
*(p+1)	*(pa+1)	pa[1]	M	E	\0		
*(p+2)	*(pa+2)	pa[2]	H	I	\0		
*(p+3)	*(pa+3)	pa[3]	S	T	A	R	\0

$$*(*(p+0)+1)=*(*p+1)$$

指向指標的指標

□ 一般作為函數的引數

- 長度及個數不固定，例如 * argv[]
- 宣告：char **p; 或 char *p[];

```
#include <stdio.h>
#include <string.h>
void f(char **p) {          // 同 char *p[]
    printf("%c\n", **p);    // 印出 Y
    printf("%s\n", *p);     // 印出 YOUD
}
void test() {
    char *pa[4]={"YOUD", "ME", "HI", "STAR"};
    f(pa);
}
```

指向指標的指標

要修改內容

- 宣告成字串陣列 d[][][5]，而非字串常數

```
#include <stdio.h>
#include <string.h>
void f(char *p[]) {
    printf("%c\n", **p);      //Y
    strcpy(*(p+1), "PK");   //ME -> HELLO
    *(*(p+2)+1) = 'Y';     //HI -> HO
}
void test() {
    char d[][5]={"YOUD", "ME", "HI", "STAR"};
    char *pa[4]={d[0], d[1], d[2], d[3]};
    f(pa);
    printf("%s\n", pa[1]);
    printf("%s\n", pa[2]);
}
```

f			+0	+1	+2	+3	
*(p+0)	*(pa[0])	d[0]	Y	O	U	D	\0
*(p+1)	*(pa[1])	d[1]	M	E	\0		
*(p+2)	*(pa[2])	d[2]	H	I	\0		
*(p+3)	*(pa[3])	d[3]	S	T	A	R	\0

指向指標的指標

- 字串存在字元陣列，要交換字串須要另外配置空間。

```
#include<string.h>
/* Swaps strings by swapping data*/
void swap(char *str1, char *str2) {
    int strLen = strlen(str1)>strlen(str2)?strlen(str1):strlen(str2);
    char *temp = (char *)malloc((strLen + 1) * sizeof(char));
    strcpy(temp, str1);
    strcpy(str1, str2);
    strcpy(str2, temp);
    free(temp);
}
int main() {
    char str1[] = "geeks";
    char str2[] = "forgeeks";
    swap(str1, str2);
    printf("str1 is %s, str2 is %s", str1, str2);
    return 0;
}
```

Exercise

- 字串存在字元陣列，要交換字串，以下Code問題？

```
#include<string.h>
void swap2(char *x, char *y) {
    char buf[80];
    int i, len;
    len = strlen(x);
    for (i=0; i<len; i++)      buf[i] = x[i];
    for (i=0; i<len; i++)      x[i] = y[i];
    for (i=0; i<len; i++)      y[i] = buf[i];
}
int main() {
    char str1[] = "geeks";
    char str2[] = "forgeeks";
    swap2(str1, str2);
    printf("str1 is %s, str2 is %s", str1, str2);
    return 0;
}
```

Exercise

- 字串存在字元陣列，要交換字串，以下Code問題？

```
#include<string.h>
void swap(char *x, char *y) {
    char tmp;
    tmp = *x;
    *x = *y;
    *y = tmp;
}
int main() {
    char name1[] = "hello";
    char name2[] = "world";
    int i;
    printf("name1=%s\nname2=%s\n", name1, name2);
    for (i=0; i<strlen(name1); i++)
        swap(&name1[i], &name2[i]);
    printf("name1=%s\nname2=%s\n", name1, name2);
    return 0;
}
```

Homework II

問題描述

一個字串如果全由大寫英文字母組成，我們稱為大寫字串；如果全由小寫字母組成則稱為小寫字串。字串的長度是它所包含字母的個數，在本題中，字串均由大小寫英文字母組成。假設 k 是一個自然數，一個字串被稱為「 k -交錯字串」，如果它是由長度為 k 的大寫字串與長度為 k 的小寫字串交錯串接組成。

舉例來說，「StRiNg」是一個 1-交錯字串，因為它是一個大寫一個小寫交替出現；而「heLLow」是一個 2-交錯字串，因為它是兩個小寫接兩個大寫再接兩個小寫。但不管 k 是多少，「aBBaaa」、「BaBaBB」、「aaaAAbbCCCC」都不是 k -交錯字串。

本題的目標是對於給定 k 值，在一個輸入字串找出最長一段連續子字串滿足 k -交錯字串的要求。例如 $k=2$ 且輸入「aBBaaa」，最長的 k -交錯字串是「BBaa」，長度為 4。又如 $k=1$ 且輸入「BaBaBB」，最長的 k -交錯字串是「BaBaB」，長度為 5。

請注意，滿足條件的子字串可能只包含一段小寫或大寫字母而無交替，如範例二。此外，也可能不存在滿足條件的子字串，如範例四。

例一：1 aBBdaaa 例二：3 DDaasAAbbCC 例三：2 aafAXbbCDCCC 例四：3 DDaaAAabbCC

範例一：2 範例二：3 範例三：8 範例四：0

Homework

```
#include <stdio.h>
#include <string.h>
void print(int b, int length){
    for (int i=0; i<length; i++) {      printf("%d ", b&1);      b = b>>1;  }
    printf("\n");
}
int match(int data, int bit, int mask) {
    if ((bit ==0) && ;)
        ;
    else if ((bit==1)           return 1;
    else return 0;
}
void f(int data, int k, int length) {
    int mask= (1<<k)-1;
    int currentCount=0, maxCount=0, bit =0;
    print(mask,length);
    print(data,length);
    while (length>0) {
        if ((currentCount ==0) && (match(data, 0, mask)==1)) {  bit = 1;      currentCount++;  }
        else if ((currentCount ==0) && (match(data, 1, mask)==1)) {  bit = 0;      currentCount++;  }
        else if ((currentCount>0) && (match(data, bit, mask)==1)) {  currentCount++;  bit = !bit;  }
        else currentCount=0;
```

Homework

```
if (currentCount==0) {           data = data>>1;           length=length-1;      }
else {           data = data>>k;           length=length-k;      }
if (currentCount>maxCount) maxCount=currentCount;
}
printf("%d %d %d %d\n", data, mask, currentCount, maxCount);

}

int input(char s[]) {

}

int main() {
//int k=2, length = 6, data=51;
int k=3, length = 6, data=7;
//int k=2, length = 6, data=50;
f(data, k, length);
//printf("%d\n", match(51, 1, 3));
print(input("AAaaaBBB"), 8);
return 0;
}
```

計算機程式設計

C語言 Pointer II

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

指標陣列(pointers of array)

□ 指標陣列：由指標組成的陣列

- 宣告：int *x[10]; // 一維陣列
- 運用：int var=5; x[2]=&var;
- 傳送給函數：利用陣列名稱傳送整個指標陣列
- trap 的 x :指向一維整數指標陣列的指標

```
int trap(int * x[ ]) {
    return (*x[0] + *x[1])* (*x[2])/2;
}
void test() {
    int *x[3];
    int base1=3, base2=5, height=4, area=0;
    x[0] = &base1;           // *(x+0) == x[0]
    x[1] = &base2;           // *(x+1) == x[1]
    x[2] = &height;
    area=trap(x);
    printf(" %d", area); // 4*(3+5)/2 = 16
}
```

trap		
值	變數名	記憶體位址 (假設)
100D	x	1031

test		
值	變數名	記憶體位址 (假設)
3	base1	1001
5	base2	1005
4	height	1009
1001	x[0]	100D
1005	x[1]	1011
1009	x[2]	1015
100D	x	

指標陣列(pointers of array)

□ `int *x[3][2]; // 二維陣列`

○ 傳送給函數：利用陣列名稱傳送整個指標陣列

○ `x[0][0]`的type是 `int*`

此處記憶體空間須要有code配置

○ `x[0]`的type是 `int**` 或 `int *[2]`

➤ 指向2個元素的一維陣列

○ `x`的type是 `int *[3][2]`

➤ 指向2維陣列的指標

變數名	記憶體位址 (假設)	值 (記憶體位址)
<code>x[0][0]</code>	1001	
<code>x[0][1]</code>	1005	
<code>x[1][0]</code>	1009	
<code>x[1][1]</code>	100D	
<code>x[2][0]</code>	1011	
<code>x[2][1]</code>	1015	

變數名	記憶體位址 (假設)	值 (記憶體位址)
	2001	
	2005	
	2009	
	200D	
	2011	

此處記憶體空間須要有code配置

變數名	記憶體位址 (假設)	值 (記憶體位址)
	2101	
	2105	
	2109	
	210D	
	2111	

二維指標動態記憶體配置

```
#define ROW 2
#define COL 4
int main() {
    int i, j;
    int **ptr2 = NULL;
    ptr2 = (int**)malloc(sizeof(int*)*ROW);
    for(i=0; i<ROW; i++){
        ptr2[i] = (int*)malloc(sizeof(int)*COL);
    }
    for(i=0; i<ROW; i++){
        for(j=0; j<COL; j++) ptr2[i][j] = i*j;
    }
}
```

```
for(i=0; i<ROW; i++){
    for(j=0; j<COL; j++) {
        printf("%d\t%d=%d\n", i, j, ptr2[i][j]);
    }
    printf("\n");
}

for(i=0; i<ROW; i++) free(ptr2[i]);
free(ptr2);
return 0;
}
```

Exercise

□ 指標++，寫下程式輸出

```
#include <stdio.h>
void print(int a[], int n) {
    int i=0;
    for (i=0; i<n; i++) {
        printf("(%x - %d)", &a[i], a[i]);
        if (i%3==2) printf("\n");
    }
}
void f1() {
    int i=0, a[]={6,5,4,3,2,1};
    int *p = a;
    *(p++) = 0;
    print(a, 6);
    printf("%x, %d\n", p, *p);
    *(++p)= -1;
    print(a, 6);
    printf("%x, %d\n", p, *p);
    a[0]=*(p++);
}
```

```
print(a, 6);
printf("%x, %d\n", p, *p);
}
int main() {
f1();
return 0;
}
```

(60fee0 - 0)(60fee4 - 5)(60fee8 - 4)
(60feec - 3)(60fef0 - 2)(60fef4 - 1)
60fee4, 5
(60fee0 - 0)(60fee4 - 5)(60fee8 - -1)
(60feec - 3)(60fef0 - 2)(60fef4 - 1)
60fee8, -1
(60fee0 - -1)(60fee4 - 5)(60fee8 - -1)
(60feec - 3)(60fef0 - 2)(60fef4 - 1)
60feec, 3

Exercise

□ 指標++，寫下程式輸出

```
#include <stdio.h>
#include <stdlib.h>
void test01(int a[]) {
    int *p=a;
    *(p++) = 99;
}
void test02(int a[]) {
    int *p=a;
    *(++p) = 99;
}
void print(int a[], int size) {
    int i=0;
    for (i=0; i<size; i++) {
        printf("%d,", a[i]);
    }
    printf("\n");
}
```

```
void test001(){
    int a[]={0,1,2,3,4,5};
    int b[]={0,1,2,3,4,5};
    test01(a);print(a,6);
    test02(b);print(b,6);
}
```

99,1,2,3,4,5,
0,99,2,3,4,5,

Exercise

□ 指標++，寫下程式輸出

```
void test002() {  
    int a[]={5,4,3,2,1,0};  
    int b[]={0};  
    int *p;  
    p = a;  
    b[1] = (*(p++))++;  
    printf("%d, %d\n", b[1], *p);  
    print(a,6);  
}  
  
void test03() {  
    int a[]={5,4,3,2,1,0};  
    int b[]={0};  
    int *p;  
    p = a;  
    b[1] = ++(*(p++));  
    printf("%d, %d\n", b[1], *p);  
    print(a,6);  
}
```

5, 4
5,4,3,2,1,0,
6, 4
6,4,3,2,1,0,

Exercise

□ 指標++，寫下程式輸出

```
void test004() {  
    int a[]={5,4,3,2,1,0};  
    int b[]={0};  
    int k=0;  
    int *p;  
    p = a;  
    k = *(p++);  
    b[1] = (k)++;  
    printf("%d, %d\n", b[1], *p);  
    print(a,6);  
}  
  
int main() {  
    test004();  
    return 0;  
}
```

5, 4
5,4,3,2,1,0,

指向函數的指標

- 宣告：`int (*fptr)()`
- 函式指標指向函式的位址，函式的位址是指函式入口點。
- 可以利用函式名稱獲得函式位址。(與陣列名稱同)
- 把函式傳給函式：函式的引數宣告成函式指位器，把函式名作為引數傳入

指向函數的指標

```
#include <stdio.h>
#include <string.h>
void check (char *a, char *b, int (*cmp)()) {
    if (!(*cmp) (a,b)) printf("equal");
    else printf("not equal");
}
void test() {
    char s1[80], s2[80], *p;
    p= (char *)strcmp;
    gets(s1);
    gets(s2);
    check(s1, s2, p);
}
```

指向函數的指標

```
int square(int len){  
    return len*len;  
}  
  
int main() {  
    int (*ptr) (int) = square  
    int area = (*ptr)(20);  
    printf("area=%d", area);  
    return 0;  
}
```

輸出結果:
area=400

指向函數的指標

□ 計算面積(積分)

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

double square(double x) {    return x * x;} /* 計算平方 */
double cube(double x) {    return x * x * x;} /* 計算三次方 */
/* 計算f()在(x,y)間以n等份逼近的積分數值，使用梯形法 */
double integral(double (*f)(double), int n, double x, double y) {
    int i;
    double gap = (y - x) / n;
    double fy1 = (*f)(x);
    double fy2 = (*f)(x + gap);
    double area = 0;
    for (i = 0; i < n; i++) {
        area += (fy1 + fy2) * gap / 2; // 使用梯形面積公式
        fy1 = fy2;
        fy2 = (*f)(x + gap * (i + 1)); //下底
    }
    return area;
}
```

指向函數的指標

□ 計算面積(積分)

```
int main() {
    char fun[100];
    int n;
    double x, y;
    double (*f)(double); // f: a pointer to function(double) returning double
    while (scanf("%99s",fun) != EOF) { // EOF定義於stdio.h內,一般為-1
        if (strcmp(fun,"square")==0) { f = square; }
        else if (strcmp(fun,"cube")==0) { f = cube; }
        else if (strcmp(fun,"sqrt")==0) { f = sqrt; } // sqrt is defined in math.h
        } else if (strcmp(fun,"end")==0) { break; }
        else {
            printf("Unknown function\n");
            continue;
        }
        scanf("%d%lf%lf", &n, &x, &y);
        printf("Integral of %s from %lf to %lf is: %lf\n", fun, x, y, integral(f, n, x, y));
    }
    return 0;
}
```

指向函數的指標

□ 計算面積(積分)

- 讓積分算得更快，迴圈內的Code越簡單越好

```
double integral(double (*f)(double), int n, double x, double y) {  
    int i;  
    double area = ((*f)(x) + (*f)(y)) / 2.0L;  
    double gap = (y - x) / n;  
    double next = x;  
    for (i = 1; i < n; i++) {  
        area += (*f)(next += gap);  
    }  
    return area * gap;  
}
```

Homework I

□ 計算面積(積分)

- 使用切割面積加總法公式， $T = (h/2)(f(p) + f(q) + 2 \sum f(x_i))$, $h = (q-p)/n$ ，求解 $f(x)$ ， x 值從區間起始 p 到區間終點 q 的面積， n 為切割數。
- $f1(x) = \sqrt{a - x^2}$ ， a 為常數。
- $f2(x) = (ax^3 + 7x)/\sqrt{a + x}$ ， a 為常數。
- double area(double (*f)(double x), double a, double p, double q, int n)，計算 $f(x)$ 從 p 到 q 切成 n 塊的面積，即計算面積加總公式 T 值。
- 增加 n 切割數(切割數 n 每次 $*2$)，精確到小數第err位。輸入1求 $f1$ ，2求 $f2$ ，程式要讓使用者重複輸入，直到輸入-1結束程式。
其他顯示錯誤訊息

Homework I

□ 計算面積(積分)

○ 1: f1 2:f2 0:exit

○ >1

○ Input a, p, q, err:4 -2 1 9

○ answer= $\sqrt{a-x^2}$ =5.054815608319

○ 1: f1 2:f2 0:exit $\sqrt{a+x}$

○ >2

○ Input a, p, q, err:1 0 3 9

○ answer=29.752380952687

○ 1: f1 2:f2 0:exit

○ >0

○ 注意：答案需輸出到小數點第12位

陣列指標(pointers of array)

□ 陣列指標，指向一個陣列的指標

- 可以作為傳多維陣列的引數
- C語言把多維陣列看成是由低一維的陣列所組成的一維陣列
- 接收n 維陣列的函數的引數，可宣告成指向n-1維陣列的指標

➤ int x[][10][20]; <=> int (*x)[10][20];

➤ int x[][10]; <=> int (*x)[10];

➤ int x[]; <=> int *x;

➤ (X) int x[][][20]

➤ (X) int (x*)[][20]

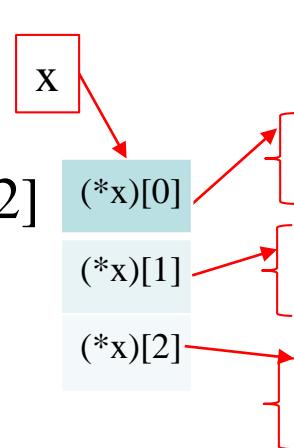
➤ int (*x)[10]; <> int *x[10];

```
#include <stdio.h>
void trap(int (*x)[3]) { // int x[][3]
    int i=0, j=0;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++) printf("%d ", x[i][j]);
        printf("\n");
    }
}
int main() {
    int x[3][3]={{1,2,3}, {4,5,6}, {7,8,9}};
    trap(x);
}
```

陣列指標(pointers of array)

□ 陣列指標，指向一個陣列的指標

- `int (*x)[3][2];`
- `x` 的 type 是 `int(*)[3][2]`
- `*x` 的 type 是 `int[3][2]` 或 `int(*)[2]`
- `(*x)[0]` 的 type 是 `int*` 或 `int[2]`
- `(*x)[0][0]` 的 type 是 `int`



變數名	記憶體位址 (假設)	值
<code>(*x)[0][0]</code>	1001	
<code>(*x)[0][1]</code>	1005	
<code>(*x)[1][0]</code>	1009	
<code>(*x)[1][1]</code>	100D	
<code>(*x)[2][0]</code>	1011	
<code>(*x)[2][1]</code>	1015	

陣列指標(pointers of array)

```
void test() {  
    // a是一個陣列，有兩個元素，每個元素是一個陣列  
    int a[2][2] = {55,66,77,88};  
    // a[0]是一個陣列，有兩個元素，每個元素是一個整數  
    int b[2]={11,22};  
    //p是指標，指向一個陣列(的記憶體)，該陣列有兩元素，P是指標的指標  
    int (*p)[2];  
    //p指向一個陣列 b  
    p=&b;  
    printf("(00)=>%d\n",**p);      // 印出 b[0]  
    printf("(01)=>%d\n",*(p+1));  // 印出 b[1]  
    printf("(02)=>%d\n",(*p)[0]);  // 印出 b[0]  
    printf("(03)=>%d\n",(*p)[1]);  // 印出 b[1]  
    printf("(04)=>%d\n", p[0][0]); // 印出 b[0]  
    printf("(05)=>%d\n", p[0][1]); // 印出 b[1]  
    printf("(06)=>%d\n",*(p[0]));  // 印出 b[0]  
    printf("(07)=>%d\n\n",*(p[0]+1)); // 印出 b[1]  
    printf("(08)=>%d\n",**(p+1));  // 錯誤指向，超出範圍  
    printf("(09)=>%d\n",p[1][0]);  // 錯誤指向，超出範圍  
    printf("(0a)=>%d\n\n",*(p[1])); // 錯誤指向，超出範圍
```

陣列指標(pointers of array)

□ 陣列指標

```
p=&a[0];
// a[0]是一個陣列，每個陣列有兩個元素，每個元素是整數
// p指向一個陣列 a[0]
printf("(10)=>%d\n",**p);      // 印出 a[0][0]
printf("(11)=>%d\n",*(p+1));   // 印出 a[0][1]
printf("(12)=>%d\n",(*p)[0]);   // 印出 a[0][0]
printf("(13)=>%d\n",(*p)[1]);   // 印出 a[0][1]
printf("(14)=>%d\n",p[0][0]);   // 印出 a[0][0]
printf("(15)=>%d\n",p[0][1]);   // 印出 a[0][1]
printf("(16)=>%d\n",*(p[0]));  // 印出 a[0][0]
printf("(17)=>%d\n\n",*(p[0]+1)); // 印出 a[0][1]

printf("(18)=>%d\n",**(p+1)); // 錯誤指向，超出範圍a[1][0]
printf("(19)=>%d\n",p[1][0]); // 錯誤指向，超出範圍a[1][0]
printf("(1a)=>%d\n\n",*(p[1])); // 錯誤指向，超出範圍a[1][0]
```

陣列指標(pointers of array)

```
p=a;           // p指向一個陣列 a
printf("(20)=>%d\n",**p);    // 印出 a[0][0]
printf("(21)=>%d\n",*(p+1)); // 印出 a[0][1]
printf("(22)=>%d\n",(*p)[0]); // 印出 a[0][0]
printf("(23)=>%d\n",(*p)[1]); // 印出 a[0][1]
printf("(24)=>%d\n", p[0][0]); // 印出 a[0][0]
printf("(25)=>%d\n", p[0][1]); // 印出 a[0][1]
printf("(26)=>%d\n",*(p[0])); // 印出 a[0][0]
printf("(27)=>%d\n\n",*(p[0]+1)); // 印出 a[0][1]

printf("(30)=>%d\n",**(p+1)); // 印出 a[0][0]
printf("(31)=>%d\n",*(p+1)+1)); // 印出 a[0][1]
printf("(32)=>%d\n",(*p+1)[0]); // 印出 a[0][0]
printf("(33)=>%d\n",(*p+1)[1]); // 印出 a[0][1]
printf("(34)=>%d\n", p[1][0]); // 印出 a[0][0]
printf("(35)=>%d\n", p[1][1]); // 印出 a[0][1]
printf("(36)=>%d\n",*(p[1])); // 印出 a[0][0]
printf("(37)=>%d\n\n",*(p[1]+1)); // 印出 a[0][1]
}
```

指標型態

- ()[] 是第一優先權左結合，* 是第二優先權右結合
 - [] — array[] of
 - * — pointer to
 - 變數後面的() — function() returning

```
char *x;           // x: a pointer to char
char x[3];         // x: an array[3] of char
char x();          // x: a function() returning char
char *x[3];        // x: an array[3] of pointer to char
char (*x)[3];      // x: a pointer to array[3] of char
char **x;          // x: a pointer to pointer to char
char *x();          // x: a function() returning pointer to char
char *x()[3];       // x: a function() returning array[3] of pointer to char
char (*x[])();      // x: an array[] of pointer to function() returning char
char (*x())();      // x: a function() returning pointer to function() returning char
char (*(*x)[]>(int, int)); // x: a pointer to array[] of pointer to function(int,int) returning char
```

指標型態

□ 表達式的資料型態辨識原則

- 觀察表達式前面的資料型態

```
char *x;      // x: a pointer to char
*x : a char
char *x[3];   // x: an array[3] of pointer to char
x[0] : a pointer to char
char **x;     // x: a pointer to pointer to char
*x : a pointer to char
char *x();    // x: a function() returning pointer to char
x() : a pointer to char
char *x()[3]; // x: a function() returning array[3] of pointer to char
x()[1] : a pointer to char
```

Exercise

- 表達式的資料型態辨識原則
 - 觀察表達式前面的資料型態

```
char **argv;
int (*daytab)[13];
int *daytab[13];
void *comp();
void (*comp)();
char (*(*x())[])();
char (*(*x[3])())[5];
```

不定長參數的函數

- 處理不定長參數要用<stdarg.h>。va_list型態存取每個參數

```
#include <stdarg.h>
// or #include <sys/varargs.h>
void minprintf(char *fmt, ...) {
    va_list ap; /* pointer to each unnamed arg in turn */
    char *p, *sval;
    int ival;
    double dval;
    va_start(ap, fmt); /* make ap point to 1st unnamed arg */
    for (p = fmt; *p; p++) { // check each character
        if (*p != '%') { // 不是特殊字元,直接輸出即可
            putchar(*p);
            continue;
    }}
```

不定長參數的函數

```
switch(*++p) { // 檢查%的下一個字元是甚麼
    case 'd':
        ival = va_arg(ap, int);
        printf("%d", ival);
        break;
    case 'f':
        dval = va_arg(ap, double);
        printf("%f", dval);
        break;
    case 's':
        for (sval = va_arg(ap, char *); *sval; sval++) { // 印出sval所指到的所有字元
            putchar(*sval);
        }
        break;
    default :
        putchar(*p);
        break;
    }
}
va_end(ap); // clean up when done */
```

計算機程式設計

C語言 union enum

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

列舉集合 enum

- 列舉集合(enum)：列舉所有可能的元素形成一種資料型態
 - 限定變數的內容，使變數值可以名稱化。
 - 使程式可讀性高，利於撰寫智慧型程式。
 - 元素以整數儲存，預設從0開始，之後 +1，也可設定整數值

```
enum 型別名 {列舉所有元素} 變數名, 變數名, ...;
```

```
enum animal {tiger, horse, bird =100 , dog};
```

```
enum animal x; // x 是列舉型別變數，值只有四種可能
```

```
typedef enum animal {tiger, horse, bird =100 , dog} animal_t;  
animal_t x; // x 是列舉型別變數，值只有四種可能
```

```
x = dog;
```

```
printf("%d", x); // 101
```

元素名	值
tiger	0
horse	1
bird	100
dog	101

列舉集合 enum

```
#include <stdio.h>
typedef enum animal {tiger, horse, bird , dog}
animal_t;
void test(animal_t x) {
    char *name[]={ "tiger","horse","bird","dog"};
    printf("%s, ",name[(int)x]);
    if (x==tiger) printf ("Escape!\n");
    else if ( x==bird) printf("watch!\n");
    else printf("Go ....\n");
}
int main() {
    test(tiger);
    test(bird);
    test(horse);
    return 0;
}
```

tiger, Escape!
bird, watch!
horse, Go

列舉集合 enum

```
#include <stdio.h>
typedef enum grade_s{PASS = 60, GOOD = 80,
EXCELLENT = 90} grade_t;
void test(int score) {
    if (score>=EXCELLENT) printf("Wonderful~\n");
    else if (score>=GOOD) printf("Nice~\n");
    else if (score>=PASS) printf("Work Hard~\n");
    else printf("See you next year~\n");
}
int main() {
    test(59);
    test(90);
    test(85);
    test(60);
    return 0;
}
```

See you next year~
Wonderful~
Nice~
Work Hard~

共同空間 (union)

利用一段共用的空間來容納不同的資料型態。

- 當一個變數的資料型別不確定時
- 不同的資料型別存在相同的空間
- 同時只能有一種資料型別存在

高位元組

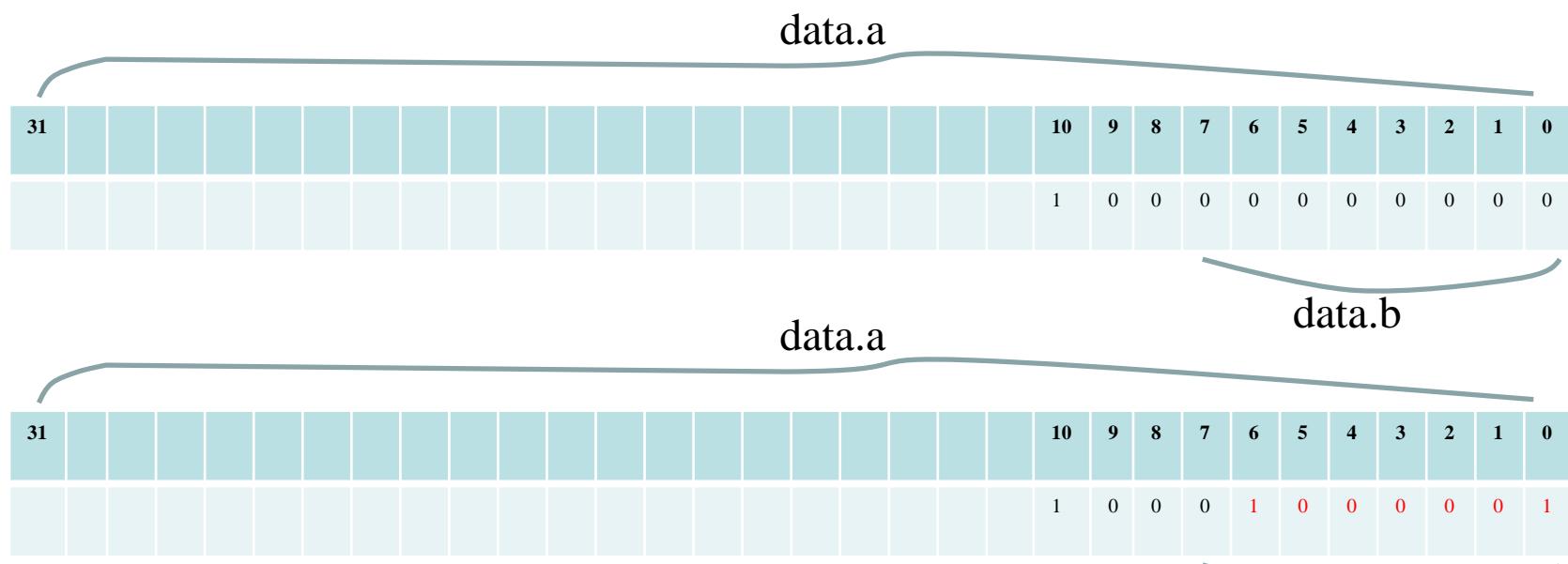
data.a	1 byte	
	1 byte	
	1 byte	
	1 byte	data.b

低位元組

```
#include <stdio.h>
typedef union data {
    int a; // 4 byte
    char b; // 1 byte
} data_t; // 佔 4 byte
void test() {
    data_t x;
    x.a = 65;
    printf("%d\n", x.a); //印出 65
    printf("%c\n", x.b); //印出 A (編碼 65)
    x.a = 1024;
    x.b = 'A';
    printf("%d\n", x.a); //印出 1089 = 1024 + 65
}
```

共同空間 (union)

- 利用一段共用的空間來容納不同的資料型態。
 - 兩個變數共用一個空間



```
x.a = 1024;      // 2進位10000000000  
x.b = 'A';        // 65 的2進位10000001  
printf("%d",x.a); // 最後變成 10001000001 = 1089
```

共同空間與列舉集合

□ 可選兩種不同評分方式 scoreType_t {G, S}

- grade_t grade; (等第評分 G) {A, B, C}
- int score; (百分評分 S) {0~100}

```
typedef enum scoreType_s {G, S} scoreType_t;
typedef enum grade_s{A, B, C} grade_t;
typedef union score_s{
    int score;
    grade_t grade;
} score_t; // 兩種評分選一種
```

```
// 印出評分結果
void print(score_t s, scoreType_t t) {
    char g[] = {'A', 'B', 'C'};
    if (t == G) printf("%c\n", g[s.grade]);
    else printf("%d\n", s.score);
}
```

```
void test() {
    int i=0;
    score_t student[3];
    scoreType_t type[3];
    char s[]={ 'A', 'B', 'C' };
    student[0].score=90;
    type[0]=S;
    student[1].grade=A;
    type[1]=G;
    student[2].grade=B;
    type[2]=G;
    for (i=0; i<3; i++) print(student[i], type[i]);
}
```

Homework I

- 一個班級有N位同學(ID)，M門課，每一門課可以設定等地評分或百分制評分。可選兩種不同評分方式
 - grade_t grade; (等第評分 G) {A, B, C}
 - int score; (百分評分 S) {0~100}

```
typedef enum scoreType_s {G, S} scoreType_t;
typedef enum grade_s{A, B, C} grade_t;
typedef union score_s{
    int score;
    grade_t grade;
} score_t;           // 兩種評分選一種
```

- 請使用enum定義不同評分制資料型別，使用union定義分數資料型別。
- 設定M門課評分制，輸入N位同學M門課成績，根據下表換算，計算每一位同學百分制平均成績，印出前三名ID與平均成績，四捨五入到整數。

Homework I

○分數對照表

等第	GPA	百分制區間	百分制對照
A+	4.3	90-100	95
A	4.0	85-89	87
A-	3.7	80-84	82
B+	3.3	77-79	78
B	3.0	73-76	75
B-	2.7	70-72	70
C+	2.3	67-69	68
C	2.0	63-66	65
C-	1.7	60-62	60
F	0	59 以下	50
X	0	0	0

計算機程式設計

C語言 Struct

郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

結構 (structure)

□ 結構

- 使用者自定的資料型別
- 由基本資料型別構成的複合資料型別。
- 結構與陣列都屬複合資料型別，但陣列是相同型別資料集合，結構體可為不同型別資料集合

□ 結構型別 (person) 定義與結構型別變數 (Tom)宣告

```
struct 結構名稱 {  
    資料型別 變數名稱;  
    ....  
};
```

```
struct person {  
    char name[30];  
    int age;  
};  
int main() {  
    struct person student;  
    return 0;  
}
```

```
int main() {  
    struct person {  
        char name[30];  
        int age;  
    } student;  
    return 0;  
}
```

結構 (structure)

- 使用 `typedef` 定義結構型別 (`person_t`)
 - `_t` 資料型別的命名原則

```
typedef struct 結構名稱 {  
    資料型別 變數名稱;  
    ....  
} 結構型別名稱;
```

```
typedef struct person {  
    char name[30];  
    int age;  
} person_t;  
int main() {  
    person_t student;  
}
```

struct person == person_t

計算結構大小

□ 使用typedef定義結構型別(person_t)

```
#include <stdio.h>
//預設以最大的元素byte對齊
//此處為 int 4 byte 對齊，
//每一個結構元素不能跨過4byte。
typedef struct person {
    char name[30];
    int age;
} person_t;
int main() {
    person_t student;
    printf("%d\n", sizeof(student)); // 36
    printf("%d\n", sizeof(person_t)); // 36
    return 0;
}
```

name[30]	age
32 bytes (4的乘方)	4 bytes

```
#include <stdio.h>
#pragma pack(1) //以一個 byte 對齊
typedef struct person {
    char name[30];
    int age;
} person_t;
int main() {
    person_t student, teacher[20] ;
    printf("%d\n", sizeof(student)); // 34
    printf("%d\n", sizeof(person_t)); // 34
    printf("%d\n", sizeof(teacher)); // 34*20
    return 0;
}
```

name[30]	age
30 bytes	4 bytes

結構體初始值設定

- 初始值以大括號設定
 - 陣列多一個大括號

```
typedef struct person {  
    char name[30];  
    int age;  
} person_t;  
int main() {  
    person_t teacher = {"Mary", 36};  
    person_t student[2] = {  
        {"Tom", 18},  
        {"John", 19}  
    };  
}
```

存取結構體資料

- 結構體資料的存取法一：'.' 運算子(operator)
 - 一般變數使用直接運算子'.'

```
typedef struct person {  
    char name[30];  
    int age;  
} person_t;  
int main() {  
    person_t teacher;  
    scanf("%s", teacher.name); //陣列名稱是記憶體位址  
    scanf("%d", &teacher.age); //一般變數要加&  
    printf("%s, %d", teacher.name, teacher.age);  
}
```

結構體指標

- 結構體資料的存取法二：'->' 運算子(operator)
 - 指標變數使用間接運算子'->'

```
typedef struct person {  
    char name[30];  
    int age;  
} person_t;  
int main() {  
    person_t teacher[20];  
    person_t *p = teacher;  
    int i=0;  
    for (i=0; i<5; i++, p++) {    // p++ 指到下一個元素  
        scanf("%s", p->name);    // 陣列名稱是記憶體位址  
        scanf("%d", &p->age);    // 一般變數要加&  
        printf("%s, %d\n", p->name, p->age);  
        printf("%s, %d\n", (*p).name, (*p).age); // 使用直接運算子  
        printf("%d - %p\n", i, p); // 記憶體位址相差32byte  
    }  
}
```

Exercise

- 請寫一段CODE，輸入兩個人的資料，印出年紀比較大的資料。

將結構體值設給另一個結構體

□ 直接=設定

```
typedef struct person {  
    char name[30];  
    int age;  
} person_t;  
int main() {  
    person_t teacher1 = {"Lin", 38};  
    person_t teacher2;  
    teacher2 = teacher1;  
    printf("%s, %d\n", teacher2.name, teacher2.age);  
}
```

結構體的結構體

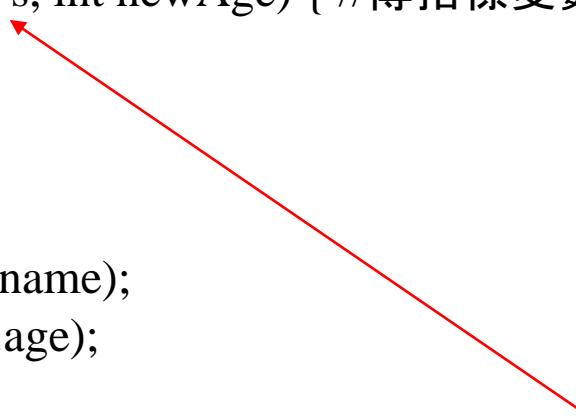
- 結構內的元素，其資料型別可以是另一個結構型別

```
typedef struct person {  
    char name[30];  
    int age;  
} person_t;  
typedef struct class_s {  
    person_t student[100];  
    int num;  
} class_t;  
int main() {  
    class_t bee;  
    scanf("%d", &bee.num);          //班上幾個人  
    scanf("%s", bee.student[0].name); //第一位同學的名字  
    scanf("%d", &bee.student[0].age); //第一位同學的年齡  
    printf("%s, %d\n", bee.student[0].name, bee.student[0].age);  
}
```

結構體與函式

□ 結構指標傳入函式

```
typedef struct person {  
    char name[30];  
    int age;  
} person_t;  
void show(person_t s) { //傳一般變數, 傳值  
    printf("%s, %d\n", s.name, s.age);  
}  
void change(person_t *s, int newAge) { //傳指標變數, 值的改變會影響主程式  
    s->age = newAge;  
}  
int main() {  
    person_t student;  
    scanf("%s", student.name);  
    scanf("%d", &student.age);  
    show(student);  
    change(&student, 12); // 傳記憶體位址, 指標變數  
    show(student);  
}
```



Exercise

- 請增加學生struct資料，新增一個程式設計pd資料。
- 請寫一個function，輸入一個班級兩位學生的資料，no是班級學生數。
 - void input(person_t s[2], int no);
- 請寫一個function，輸入一個班級一位學生的資料。
void input(person_t *s);
- 請寫一個function，比較班級兩位學生資料，回傳成績高的姓名(參數 char name[])。
 - void getHigh(person_t s[2], int no, char name[]);
- 請寫一個function，計算回傳班級兩位學生平均成績(使用 return 回傳)。
 - int getAverage(person_t s[2], int no);

Exercise

- 請增加學生struct資料，新增一個程式設計pd資料。

```
int main() {  
    person_t s[2];  
    char name[30];  
    input1(s, 2);  
    input2(&s[0]);  
    getHigh(s, 2, name);  
    printf("%s\n", name);  
    printf("%d\n", getAverage(s, no));  
}
```

Exercise

□ 105APCS Q4

- 簡化版模擬讀入棒球隊每位球員打擊結果，計算得分。假設球員打擊結果只有以下情況：
 - 安打：以1B, 2B, 3B 和 HR代表一壘打、二三和全(四)壘打。
 - 出局：以 FO, SO, GO表示。
- 簡化版的規則如下：
 - 球場上有四個壘包，稱為本壘、一、二和三壘。
 - 在本壘 握球棒打的稱「擊球員」，站在另外三個壘包稱「跑員」。
 - 擊球員打擊結果為「安打」時，場上擊球員與跑壘員可移動；「出局」時，跑壘員不動，擊球員離場換下一位。
 - 球隊共九位球員，依序排列。比賽開始由第 1位打擊，當第 i 位球員打擊完，由第 $(i+1)$ 位球員打擊。當第九位球員完後，輪回第 1位球員。

Exercise

□ 105APCS Q4

- 當打出 K 壘打時，場上擊球員和跑壘員前進 K 個壘包。從本壘到一壘，接著二、三壘，最後回到本壘。
- 每位球員回到本壘時可得 1 分。
- 每達三出局時，一、二、三壘會清空，跑壘員離開，重新開始。

○ 輸入格式

- 每組測試資料固定十行。
- 第一到九行，依照球員順序，每一行代表每位球員打擊資訊。每一行開始有一個正整數 a ($1 \leq a \leq 5$)，代表球員總共打 a 次。接下來有 a 個字串(均為兩個字元)，依序代表每次打擊結果。資料間均以一個空白字元隔開。球員打擊資訊不會錯、缺漏。
- 第十行有一個正整數 b ($1 \leq b \leq 27$)，表示要計算當總出局數累計到 b 時，該球隊的得分。輸入的打擊資訊中至少包含 b 個出局。

Exercise

- 練習：請設計球員的struct資料型別
- 輸出：計算第 b個出局數發生時總得分，將此得分輸出於一行。

輸入範例一

5 1B 1B FO GO 1B
5 1B 2B FO FO SO
4 SO HR SO 1B
4 FO FO FO HR
4 1B 1B 1B 1B
4 GO GO 3B GO
4 1B GO GO SO
4 SO GO 2B 2B
4 3B GO FO FO

3

正確輸出

0

輸入範例二

5 1B 1B FO GO 1B
5 1B 2B FO FO SO
4 SO HR SO 1B
4 FO FO FO HR
4 1B 1B 1B 1B
4 GO GO 3B GO
4 1B GO GO SO
4 SO GO 2B 2B
4 3B GO FO FO

6

正確輸出

5

Exercise

□ 105APCS Q4

```
#include <stdio.h>
typedef struct member_s{
    int no;
    int data[10];
} member_t;
void input(member_t m[9], int *goal) {
    char temp[10];
    for (int i=0; i<9; i++) {
        scanf("%d", &m[i].no);
        for (int j=0; j<m[i].no; j++) {
            scanf("%s", temp);
            if (temp[1]=='O') m[i].data[j]=0;
            else if (temp[0]=='H') m[i].data[j]=4;
            else m[i].data[j]=(temp[0]-'0');
        }
    }
    scanf("%d", goal);
}
```

Exercise

□ 105APCS Q4

```
void f() {
    member_t m[9];
    int goal=0, out=0, index=0, score=0;
    int state=0, r=0, c=0, reset=0;
    input(m, &goal);
    while (out<goal) {
        r = index%9; c = index/9;
        if (m[r].data[c]==0) {
            out++;
            if (out%3==0) state=0;
        }
        else {
            state = (state<<m[r].data[c])|(1<<(m[r].data[c]-1));
            for (int i=0; i<4; i++) score = score + ((state>>(3+i))&1);
            state = state&7;
        }
        index++;
    }
    printf("score=%d, mem=%d", score, index);
}
int main() {    f();    return 0; }
```

位元欄位 (bit field)

宣告

struct-declarator :

 type-specifier declarator : constant-expression

- constant-expression 指定欄位的位元寬度，非負整數。若該值是零，宣告沒有 declarator。
- 標準 ANSI C，type-specifier 須是 **unsigned int**、**signed int** 或 **int**。
- Microsoft 的 ANSI C 標準延伸模組允許 **char**, **long**, **unsigned**。
- 不允許位元欄位陣列、位元欄位指標與傳回位元欄位的函式。
- address-of 運算子 (&) 無法套用至位元欄位元件。
- 位元欄位可以有名字、或沒有命名。
- 未命名位元欄位內容在執行期無法預測，一般作為「虛設」欄位，用於**補空間到對齊位元**。
- 寬度為 0，規範 struct-declaration-list 在其後的成員儲存區以下一個 int 界限開始。

位元欄位 (bit field)

- 位元欄位只能宣告為結構的成員。

```
short a:17; /* Illegal! 不能用於一般變數*/  
int long y:33; /* Illegal! */
```

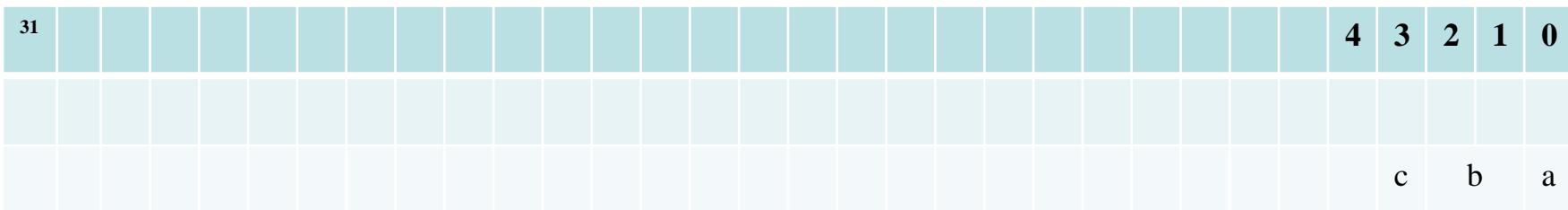
- 將資料以位元形式儲存，節省空間。
- 可存取整數值的部分內容，以簡化程式碼。
- 移植性不高 - 不同作業系統與硬體平台可能有不同結果。
- 一個位元欄位成員不允許跨越兩個 unsigned int 邊界，若成員總位數超過一個 unsigned int 大小，編譯器會自動移位成員，使其對齊unsigned int 的邊界。
- $\text{field1} + \text{field2} = 34$ Bits，超出 32 Bits。
- field2移位至下一个 unsigned int。
- stuff佔64 bit。

```
struct stuff {  
    unsigned int field1: 30;  
    unsigned int field2: 4;  
};
```

位元欄位 (bit field)

- 28 bit 未使用。

```
struct data {  
    unsigned int a: 1;  
    unsigned int b: 2;  
    unsigned int c: 1;  
};
```



```
struct data {  
    unsigned int a: 4;  
    unsigned int : 0; // 空的位元欄位  
    unsigned int b: 1; // 從下一個單元開始存放  
};
```

位元欄位 (bit field)

31																9	8	7	6	5	4	3	2	1	0
																0	0	1	1	1	1	1	0	1	0
																		status	b4	b3	b2	b1	b0		

```
#include <stdio.h>
typedef struct flag_s{
    unsigned b0 : 1;
    unsigned b1 : 1;
    unsigned b2 : 1;
    unsigned b3 : 1;
    unsigned b4 : 1;
    unsigned status : 4;
} flag_t;
int main() {
    flag_t flag;
    scanf("%x", &flag); //輸入 16進位 fa = 1111 1010
    printf("%d\n", sizeof(unsigned), sizeof(flag_t));// 4 byte
    printf("%d %d %d\n", flag.b2, flag.b1, flag.b0); // 0 1 0 byte
    printf("%d %d %d\n", flag.status, flag.b4, flag.b3); // 7 1 1 byte
    return 0;
}
```

位元欄位 (bit field)

○ 共占用 byte 數。

```
#include <stdio.h>
struct on_off {
    unsigned light : 1;
    unsigned toaster : 1;
    int count; /* 4 bytes */
    unsigned ac : 4;
    unsigned : 4;
    unsigned clock : 1;
    unsigned : 0;
    unsigned flag : 1;
} kitchen;
struct flag{
    unsigned short F1 : 1;
    unsigned short F2 : 1;
    unsigned short F3 : 1;
    unsigned short F4 : 1;
    unsigned short F5 : 1;
} room;
```

```
struct house_s{
    unsigned int F1 : 1;
    unsigned int F2 : 1;
    unsigned int F3 : 1;
} house;
int main() {
    printf("kitchen=%d\n", sizeof(kitchen)); // 16 byte
    printf("%d\n", sizeof(room));           // 2 byte
    printf("%d\n", sizeof(unsigned short)); // 2 byte
    printf("%d\n", sizeof(house));         // 4 byte
    return 0;
}
```

Exercise bit 位元欄位

```
//#pragma pack(1)
#include <stdio.h>
struct on_off1 {
    unsigned light : 1;
    unsigned toaster : 1;
    int count; /* 4 bytes */
    unsigned ac : 4;
    unsigned : 4;
    unsigned clock : 1;
    unsigned : 0;
    unsigned flag : 1;
}kitchen1;
struct on_off2 {
    unsigned light : 1;
    unsigned toaster : 1;
    int count; /* 4 bytes */
    unsigned ac : 4;
}kitchen2;
struct on_off3 {
    unsigned light : 1;
    unsigned toaster : 1;
    int count; /* 4 bytes */
    unsigned ac : 4;
    unsigned flag : 1;
}kitchen3;
```

```
struct on_off4 {
    unsigned light : 1;
    unsigned : 0;
    unsigned toaster : 1;
    int count; /* 4 bytes */
    unsigned ac : 4;
}kitchen4;
struct on_off5 {
    unsigned light : 1;
    unsigned : 2;
    unsigned toaster : 1;
    int count; /* 4 bytes */
    unsigned ac : 4;
}kitchen5;
int main() {
    printf("kitchen=%d\n", sizeof(kitchen1)); // 16 byte
    printf("kitchen=%d\n", sizeof(kitchen2)); // 12 byte
    printf("kitchen=%d\n", sizeof(kitchen3)); // 12 byte
    printf("kitchen=%d\n", sizeof(kitchen4)); // 16 byte
    printf("kitchen=%d\n", sizeof(kitchen5)); // 12 byte
    return 0;
}
```

Exercise bit 位元欄位

```
#pragma pack(1)
#include <stdio.h>
struct on_off1 {
    unsigned light : 1; // 1 byte
    int count;          // 4 bytes
}kitchen1;
struct on_off2 {
    unsigned light : 1; // 1 byte
    int count;          // 4 bytes
    unsigned ac : 3;   // 4 bytes
}kitchen2;
struct on_off3 {
    int count;          // 4 bytes
    unsigned ac : 3;   // 4 bytes
}kitchen3;
struct on_off4 {
    unsigned light : 1; // 1 bytes
    unsigned : 0;
    unsigned toaster : 1; // 4 bytes
    int count;          // 4 bytes
    unsigned ac : 3;   // 4 bytes
}kitchen4;
```

```
struct on_off5 {
    unsigned light : 1;
    unsigned toaster : 1; // 1 bytes
    int count;           // 4 bytes
    unsigned ac : 3;
    unsigned : 4;
    unsigned clock : 1; // 4 bytes
    unsigned : 0;
    unsigned flag : 1;  // 4 bytes
}kitchen5;
int main() {
    printf("kitchen=%d\n", sizeof(kitchen1)); // 5 byte
    printf("kitchen=%d\n", sizeof(kitchen2)); // 9 byte
    printf("kitchen=%d\n", sizeof(kitchen3)); // 8 byte
    printf("kitchen=%d\n", sizeof(kitchen4)); // 13 byte
    printf("kitchen=%d\n", sizeof(kitchen5)); // 13 byte
    return 0;
}
```

Exercise bit 位元欄位

```
#include "stdio.h"
union {
    int id;
    struct {
        unsigned int x : 1;
        unsigned int y : 2;
        unsigned int z : 2;
    } bits;
} number;
void main(){
    for (number.id=0;number.id<16; number.id++) {
        printf("id=%3d,bits=%3d%3d%3d\n",number.id,
        number.bits.z, number.bits.y, number.bits.x);
    }
}
```

函式指標-talk

- 結構內的元素，是函式指標。

```
#include <stdio.h>
#include <string.h>
typedef struct cat {
    char name[10];
    void (*talk)(int money);
} cat_t;
void talkLarge(int money) {
    printf("LARGE %d\n", money);
}
void talkSmall(int money) {
    printf("SMALL %d\n", money);
}
void CatTalk(cat_t catX, int money) {
    catX.talk(money);
}
```

函式指標-talk

- 結構內的元素，是函式指標。

```
void test01() {  
    cat_t cat001;  
    strcpy(cat001.name, "Tom");  
    cat001.talk = talkLarge;  
    CatTalk(cat001, 1000);  
    cat001.talk = talkSmall;  
    CatTalk(cat001, 10);  
}  
  
void talkEnglish(char name[]) {  
    printf("Hello! %s, ", name);  
}  
  
void talkDeutsch(char name[]) {  
    printf("Hallo! %s, ", name);  
}
```

函式指標-talk

- 結構內的元素，是函式指標。

```
void run02(void talk(char *), char name[]) {  
    void (*f)(char *);  
    talk(name);  
    talk = talkEnglish;  
    talk(name);  
    f = talk;  
    f(name);  
}  
void test02() {  
    run01(talkEnglish, "John");  
    run02(talkDeutsch, "John");  
}  
int main() {  
    test01();  
    test02();  
    return 0;  
}
```

函式指標-Shape

```
#define ShapeText(TYPE) \
    char name[10];\
    float (*perimeter)(struct TYPE*);\
typedef struct _Shape { // Shape 物件無欄位
    ShapeText(_Shape);
} Shape;
typedef struct _Circle {
    ShapeText(_Circle);
    float radius;
} Circle;
float ShapeArea(Shape *obj) { return 0; }
float ShapePerimeter(Shape *obj) { return 0; }
void ShapeNew(Shape *obj) {
    strcpy(obj->name,"shape");
    obj->perimeter = ShapePerimeter;
}
float CircleArea(Circle *obj)
{ return 3.14 * obj->radius * obj->radius; }
```

```
float CirclePerimeter(Circle *obj)
{ return 2*3.14 * obj->radius; }
void CircleNew(Circle *obj) {
    strcpy(obj->name,"circle");
    obj->perimeter = CirclePerimeter;
}
int main() {
    int i;
    Shape s;
    Circle c;
    ShapeNew(&s);
    CircleNew(&c);
    c.radius=3.0;
    Shape *list[] = { &s, (Shape*) &c };
    for (i=0; i<2; i++) {
        Shape *o = list[i];
        printf("%s.perimeter()=%G\n", o->name,
               o->perimeter(o));
    }}
```

函式指標-Shape draw

- 結構內的元素，是函式指標。不定長度參數，造出不同物件。

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
typedef enum {point,circle} shape_type;
typedef struct {
    shape_type type;
    void (*destroy)();
    void (*draw)();
}shape_t;
typedef struct {
    shape_t common;
    int x,y;
}point_t;
```

函式指標-Shape draw

□ 定義各種運算函式

```
typedef struct {
    shape_t common;
    point_t *center;
    int radius;
}circle_t;
void destroyPoint(point_t *this){
    int x=this->x,y=this->y;
    free(this);
    printf("P(%d,%d) destroyed!\n",x,y);
}
void drawPoint(point_t *this){
    printf("P(%d,%d)",this->x,this->y);
}
```

函式指標-Shape draw

```
point_t * createPoint(va_list *ap){  
    point_t *p_point;  
    if ((p_point=(point_t*)malloc(sizeof (point_t)))==NULL)  
        return NULL;  
    p_point->common.type=point;  
    p_point->common.destroy=destroyPoint;  
    p_point->common.draw=drawPoint;  
    p_point->x=va_arg(*ap,int);  
    p_point->y=va_arg(*ap,int);  
    return p_point;  
}  
void destroyCircle(circle_t *this){  
    int x=this->center->x;  
    int y=this->center->y;  
    int r=this->radius;  
    this->center->common.destroy(this->center);  
    free(this);  
    printf("C(P(%d,%d),%d) destroyed!\n",x,y,r);  
}
```

函式指標-Shape draw

```
void drawCircle(circle_t *this){  
    printf("C(");  
    this->center->common.draw(this->center);  
    printf(",%d)",this->radius);  
}  
circle_t* createCircle(va_list* ap){  
    circle_t *p_circle;  
    if ((p_circle=(circle_t*)malloc(sizeof(circle_t)))==NULL)  
        return NULL;  
    p_circle->common.type=circle;  
    p_circle->common.destroy=destroyCircle;  
    p_circle->common.draw=drawCircle;  
    p_circle->center=createPoint(ap);  
    p_circle->radius=va_arg(*ap,int);  
    return p_circle;  
}
```

函式指標-Shape draw

```
shape_t* createShape(shape_type st,...){  
    va_list ap;  
    shape_t* p_shape=NULL;  
    va_start(ap,st);  
    if (st==point) p_shape=(shape_t*)createPoint(&ap);  
    if (st==circle) p_shape=(shape_t*)createCircle(&ap);  
    va_end(ap);  
    return p_shape;  
}  
int main(){  
    int i;  
    shape_t *shapes[2];  
    shapes[0]=createShape(point,2,3);  
    shapes[1]=createShape(circle,20,40,10);  
    for (i=0;i<2;i++) {  
        shapes[i]->draw(shapes[i]);  
        printf("\n");  
    }  
    for (i=1;i>=0;i--)    shapes[i]->destroy(shapes[i]);  
    return 0;}
```

Exercise函式指標-電路圖

□ 定義各種運算函式

```
#include <stdio.h>
#include <stdlib.h>
#define GATEVALUE(TYPE) int(*GateValue)(void)
typedef struct _Gate{
    GATEVALUE();
}Gate;
int GateGetValue(){return 0;}
typedef struct _GateAnd{
    GATEVALUE();
}GateAnd;
int GateAndValue(){return 1;}
void CreateGate(Gate *obj){
    obj->GateValue = GateGetValue;
}
```

Exercise函式指標-電路圖

□ 定義各種運算函式

```
void CreateGateAND(GateAnd *obj){  
    obj->GateValue = GateAndValue;  
}  
int main(int argc, char *argv[]){  
    Gate gate;  
    CreateGate(&gate);  
    GateAnd and;  
    CreateGateAND(&and);  
    printf("Gate = %d, GateAND = %d\n", gate.GateValue(),  
and.GateValue());  
    return 0;  
}
```

Homework

- 以下邏輯電路圖，輸入為 X1, X2, X3，輸出為 Y1, Y2, Y3。
 - $X_1 \rightarrow P_Gate \rightarrow Y_1$
 - $X_2 \rightarrow Q_Gate \rightarrow Y_2$
 - $X_3 \rightarrow R_Gate \rightarrow Y_3$
 - P_Gate 邏輯閘可設定為 NOT或空，輸入為 X1，輸出為 Y1和 Q_Gate 邏輯閘的輸入。
 - Q_Gate 邏輯閘可設定為 AND 或 OR，輸入為 X2 和 P_Gate 邏輯閘的輸出，輸出為 Y2 和 R_Gate 邏輯閘的輸入。
 - R_Gate 邏輯閘可設定為 AND 或 OR，輸入為 X3 和 Q_Gate 邏輯閘的輸出，輸出為 Y3。
 - 輸入 X1、X2、X3，以及設定 P、Q、R、三個邏輯閘的種類。

Homework

○ 輸入說明:

- 第一行依次輸入 X1、X2、X3 為 0 或 1，中間以逗號間隔。
- 第二行輸入 P、Q、R 邏輯閘的設定，A 代表 AND 邏輯閘，O 代表 OR 邏輯閘，N 代表 NOT 邏輯閘，E 代表空的邏輯閘，中間以逗號間隔。

○ 輸出說明:

- 輸出 Y1、Y2、Y3 為 0 或 1，中間以逗號間隔。

○ 範例:

- Sample Input:

– 0,1,0

– N,A,O

- Sample Output:

– 1,1,1

Homework

□ 利用結構 struct 定義

- Shape (圖形) , Circle (圓) , Rectangle (矩形) , Square (正方形) , Triangle (三角形) 。
- 圓有半徑，矩形有長和寬，正方形有邊長，三角形有三個邊。
- 計算各個圖形的周長，以及所有圖形的周長加總。
- 此題須使用以下 struct 及 function pointer 實作，否則不予計分。

```
#define ShapeText(TYPE) char name[10];
    double (*perimeter)(struct TYPE*);
    double (*area)(struct TYPE*);

typedef struct shape_s {
    ShapeText(shape_s);
} shape_t;

typedef struct circle_s {
    ShapeText(circle_s);
    double radius;
} circle_t;
```

Homework

□ 利用結構 struct 定義

- Shape (圖形) , Circle (圓) , Rectangle (矩形) , Square (正方形) , Triangle (三角形) 。
- 圓有半徑，矩形有長和寬，正方形有邊長，三角形有三個邊。
- 計算各個圖形的周長，以及所有圖形的周長加總。
- 此題須使用以下 struct 及 function pointer 實作，否則不予計分。

輸入說明	輸出說明
第一行輸入圖形個數N。第二行到第N+1行輸入圖形種類、及該圖形所需整數資料，以空白間隔。圖形種類以字元表示，C代表圓、R代表矩形、S代表正方形、T代表三角形。輸入C跟隨一個數值為半徑，輸入R跟隨兩個數值長和寬，輸入S跟隨一個數值代表邊長，輸入T跟隨三個數值代表三個邊。	PI設為4。輸出N+1行，前N行，輸出第N個圖形的周長。第N+1行，輸出N個圖形的周長總和。
Sample Input	Sample Output
5 T 3 4 5 S 1 R 2 3 C 1 T 3 4 6	12 4 10 8 13 47

Homework 工作排程

□ 問題描述

- 有M個工作要在N台機器上加工，每個工作*i*包含若干個工序 O_{ij} ，這些工序須依序加工，也就是前一道工序 $O_{i(j-1)}$ 完成後才可開始下一道工序 O_{ij} 。每道工序 O_{ij} 可用一個有序對 (k_{ij}, t_{ij}) 表示它需在機器 k_{ij} 上面花費 t_{ij} 小時完成。每台機器一次只能處理一道工序。
- 所謂一道工序 O_{ij} 的「最早完成時間的 c_{ij}^* 」是指考慮目前排程中機器 k_{ij} 之可用性以及前一道工序 $O_{i(j-1)}$ (若該工序存在)之完成時間後可得的最早完成時間。工廠經理安排所有工廠經理安排所有工序的排程規則如下：
 - 針對每一個工作的第一個尚未排程的工序，計算出此工序的「最早完成時間」，然後挑選出最早完成時間最小的工序納入排程，如果有多個完成時間都是最小，則挑選其中工作編號最小之工序。一個工序一旦納入排程就不會再更改，重複以上步驟直到所有工序皆納入排程。

Exercise 工作排程

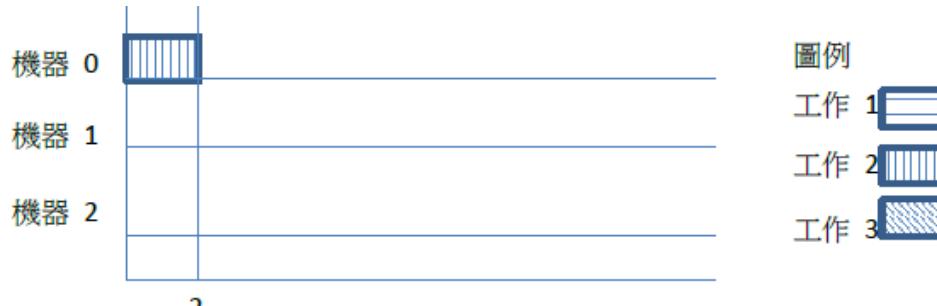
- 總是從時間0開始排程，每個工作的完成時間為其最後一個工序的完成時間，序的完成時間，本題的目標是計算出每個工作的完成時間並輸出其總和。
- 以下例子說明，此例中有三個工作要在三台機器上排程，各工作的資料如下。

	工序	說明
工作1	$o11 = (2, 4)$ $o12 = (1, 1)$	此工作有兩道工序，第一道需要在機器2執行4小時，第二道需要在機器1執行1小時。
工作2	$o21 = (0, 2)$ $o22 = (2, 2)$ $o23 = (0, 1)$	有三道工序，第一道需要在機器0執行2小時，餘類推。
工作3	$o31 = (0, 7)$	有一道工序需要在機器0執行7小時。

Exercise 工作排程

○排程過程說明如下：

- 1. 在開始時，每個工作都是考慮第一道工序，三個工作第 1 道工序需要的時間分別是 $t_{11} = 4$ 、 $t_{21} = 2$ 、 $t_{31} = 7$ ，這也是它們的最早完成時間，也就是 $c_{11}=4$ 、 $c_{21}=2$ 、 $c_{31}=7$ ，因此會先排 o_{21} 。



- 2. 接下來，三個工作要考慮的順序分別是第 1、2、1 個工序，即 o_{11} 、 o_{22} 和 o_{31} 。
 - (1) o_{11} 需要機器 2 執行 4 小時，而機器 2 可以開始加工的時間點是 0； o_{11} 沒有前一道工序。因此，這工序可以開始的時間是 $\max(0, 0) = 0$ 。是故，其最早完成時間 $c_{11} = \max(0, 0) + 4 = 4$ 。
 - (2) o_{22} 需機器 2 執行 2 小時，而機器 2 可開始加工時間點是 0； o_{22} 前一道工序 o_{21} 完成時間是 2。因此，這工序可以開始的時間是 $\max(0, 2) = 2$ 。最早完成時間 $c_{22} = \max(0, 2) + 2 = 4$ 。

Exercise 工作排程

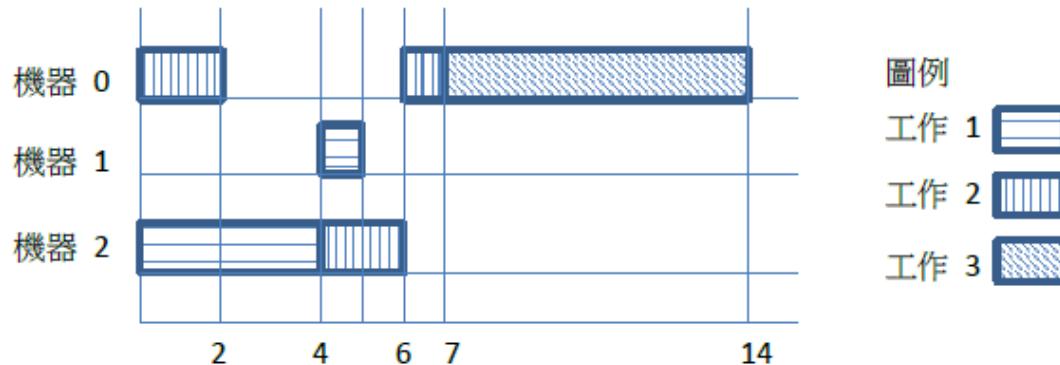
- (3) o₃₁ 需機器 0 執行 7 小時，而機器 0 可開始加工的時間點是 2；o₃₁ 沒有前一道工序。因此，這工序可開始的時間是 $\max(2, 0) = 2$ 。其最早完成時間 $c_{31}^* = \max(2, 0) + 7 = 9$ 。
- 因此，由於 c_{11}^* 和 c_{22}^* 都是最小，根據規則，工作編號小的先排，所以會排 o₁₁。



- 3. 三個工作目前要考慮的順序分別第 2、2、1 個工序。依照類似推論，可得到 $c_{12} = 5$ ， $c_{22} = 6$ ， $c_{31} = 9$ ，因此排 o₁₂。工作 1 的工序均已排完，所以它的完成時間是 5。
- 4. 剩下工作 2 與 3。 $c_{22} = 6$ ， $c_{31} = 9$ ，因此先排 o₂₂。
- 5. $c_{23} = 7$ 而 $c_{31} = 9$ ，因此排 o₂₃，工作 2 的工序已排完，所以它的完成時間是 7。

Exercise 工作排程

- 6. 剩下工作 3，因為機器 0 的下一個可以開始時間是 7，o31的完成時間是 $7+7=14$ 。



- 三個工作完成時間分別是 5、7、14，輸出答案 $5+7+14=26$ 。
- 輸入格式

- 第一行有兩個整數 N 與 M，代表 N 台機器與 M 個工作，接下來有 M 個工作資訊，輸入順序即是工作編號順序。每個工作資訊包含兩行，第一是整數 P，代表到工序數量；第二行是 $2 \times P$ 個整數，每兩個一組依序代表一道工序的機器編號與需求時間。機器的編號由 0 開始。參數 N、M、P 以及每個工序的需求時間都是不超過 100 的正整數。

Exercise 工作排程

➤ 輸出格式

– 輸出每個工作的完成時間的總和。

範例一：輸入

3 3
2
2 4 1 1
3
0 2 2 2 0 1
1
0 7
正確輸出
26

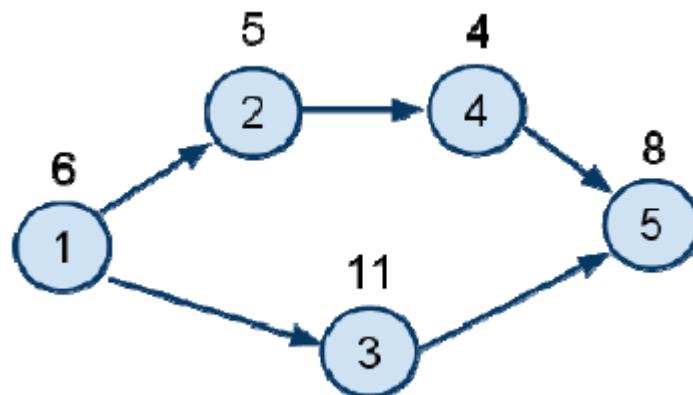
範例二：輸入

2 3
1
0 4
1
1 5
1
1 3
正確輸出
15

Homework

□ 題目

- 開發專案時，專案會被分割為許多項目，分配給多組程式設計師開發。但這些項目是有順序關係，只有當順序在前的項目完成，才能開始開發順序在後的項目。用一個有向圖，表示這些開發順序。每個節點代表一個項目，節點內的數字為節點編號，上方數字代表開發這個項目所需天數；邊表示開發順序。
- 以圖為例，只有在節點 2 完成後，才能開始節點 4 的開發。
- 一軟體公司的專案準備開始開發，你是公司專案經理，根據開發流程圖，老闆想知道專案能否在限制時間內完工。



Exercise

□ 輸入與輸出

- 輸入第一行是整數，代表後續測試資料組數。每組測試資料代表專案有向圖。每組測試資料的第一行是正整數N，代表專案共有 N 個工作事項(節點)， $N \leq 1000$ 。
- 接下來N 行測試資料，每一行依序代表一個項目節點(從 1 開始)，第一個正整數表示完成這個項目所需天數，第二個正整數 K 表示這個節點有 K 條指向其他節點的邊，接下來 K 個正整數表示所指向的項目節點編號。

範例輸入 #1

2

2

8 1 2

2 0

5

6 2 2 3

5 1 4

11 1 5

4 1 5

8 0

範例輸出 #1

10

25

2 共有兩組專案測試資料

2 第一組專案有兩個工作項目（節點）

8 1 2 第一個工作項目需8天完成，有一個工作項目(第二個工作項目)需等第一個工作項目完成後才能進行。

2 0 第二個工作項目需要2天才能完成

5 第二組專案有五個工作項目（節點）

6 2 2 3 第一個工作項目需6天才能完成，有兩個工作項目（第二、三個工作項目）需等第一個工作項目完成後才能進行。

5 1 4 第二個工作項目需要5天才能完成，有一個工作項目（第四個工作項目）需等第二個工作項目完成後才能進行。

11 1 5 第三個工作項目需要11天才能完成，有一個工作項目（第五個工作項目）需等第三個工作項目完成後才能進行。

4 1 5 第四個工作項目需要4天才能完成，有一個工作項目（第五個工作項目）需等第四個工作項目完成後才能進行。

8 0 第五個工作項目需要8天才能完成

Exercise

```
#include <stdio.h>
typedef struct task_s{
    int pTask[10];
    int pNo;
    int day;
    int tDay;
} task_t;
int isCanCompute(task_t tasks[], task_t t) {
    int flag=1, id=0;
    for (int i=0; i<t.pNo; i++) {

    }
    return flag;
}
int isYetCompute(task_t t) {
    if (t.tDay===-1) return 1;
    else return 0;
}
```

```
int computeDay(task_t tasks[], task_t t) {
    int maxDay=0, id=0;
    for (int i=0; i<t.pNo; i++) {
        }

    return (maxDay+t.day);
}
void print(int n, task_t data[]) {
    for (int i=1; i<=n; i++) {
        printf("%d, %d, %d, %d\n", i,
data[i].pNo, data[i].day, data[i].tDay);
        for (int j=0; j<data[i].pNo; j++) {
            printf("%d ", data[i].pTask[j]);
        }
        printf("\n");
    }
}
```

Exercise

```
void input(int n, task_t data[]) {  
    int c=0, id=0, index=0;  
    for (int i=1; i<=n; i++) {  
        data[i].pNo=0;  
        data[i].tDay = -1;  
    }  
    for (int i=1; i<=n; i++) {  
        scanf("%d", &data[i].day);  
        scanf("%d",&c);  
        for (int j=0; j<c;j++) {  
              
        }  
    }  
}
```

```
void printAnswer(int n, task_t data[]) {  
    int maxDay=0;  
    printf("%d", maxDay);  
}  
int getId(int n, task_t data[]) {  
    for (int i=1; i<=n; i++) {  
        if (isYetCompute(data[i])&&isCanCompute(data,  
data[i]))  
            return i;  
    }  
    return -1;  
}
```

Exercise

```
void f() {  
    int n=0, count=0, id=0;  
    task_t data[10];  
    scanf("%d", &n);  
    input(n, data);  
    print(n, data);  
  
    while (count<5) {  
        id = getId(n, data);  
        if (id>=1) {  
            data[id].tDay = computeDay(data, data[id]);  
            count++;  
        }  
        else break;  
    }  
    print(n, data);  
    printAnswer(n, data);  
}
```

```
int main() {  
    int no =0;  
    scanf("%d", &no);  
    for (int i=0; i<no; i++)  
        f();  
  
    return 0;  
}
```

計算機程式設計

C語言 File

郭忠義

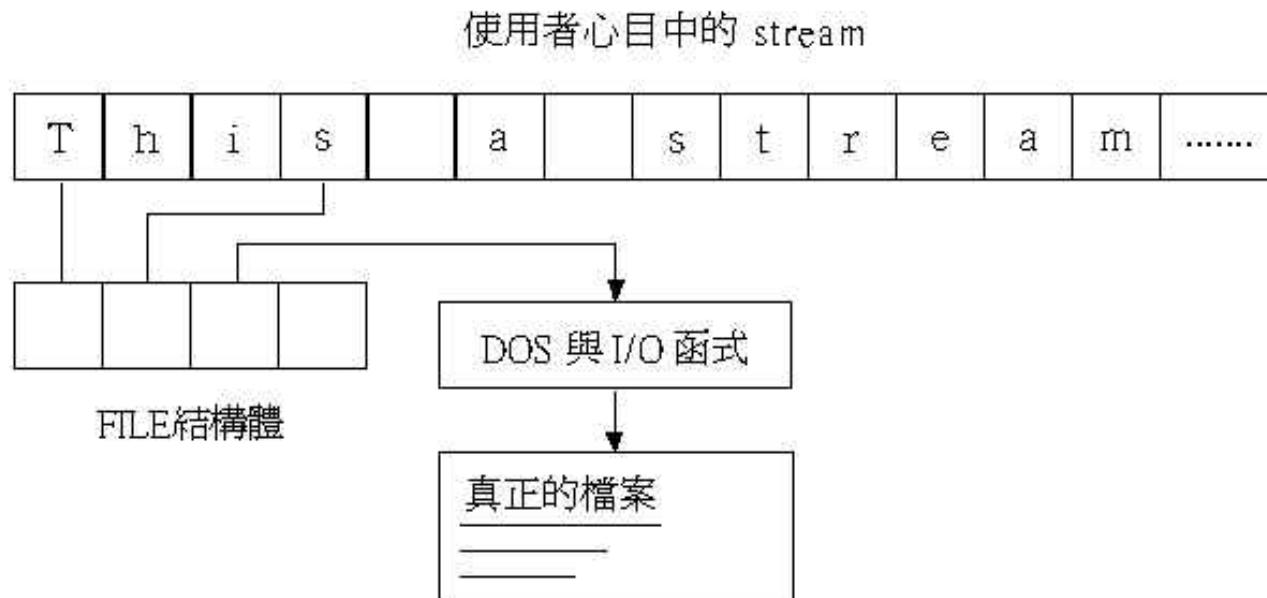
jykuo@ntut.edu.tw

臺北科技大學資訊工程系

檔案處理

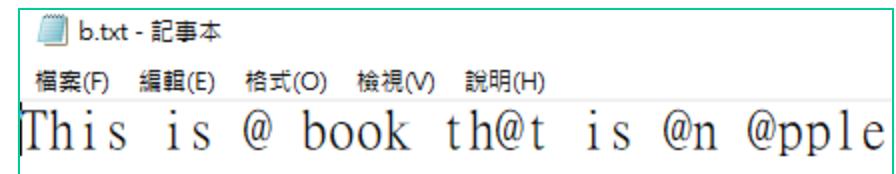
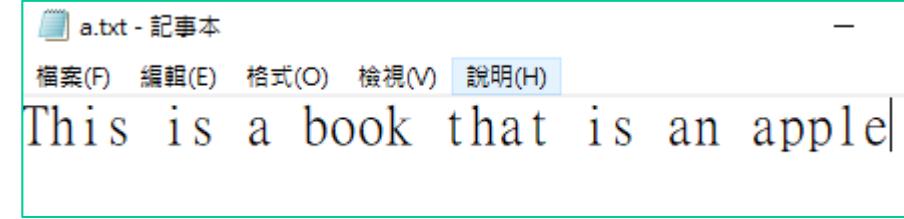
□ FILE 型別

- Stream是已經open的檔案



檔案開啟關閉與複製

```
#include <stdio.h>
#include <stdlib.h>
void fun(char fileName1[], char fileName2[]) {
    FILE *fp1, *fp2;
    char ch,full[20];
    if ((fp1=fopen(fileName1,"r"))==NULL) {
        printf("cannot open file %s",fileName1);
        exit(0);
    }
    fp2=fopen(fileName2,"w");
    do {
        if ((ch=fgetc(fp1))=='a') ch='@';
        printf("%c",ch);
        fputc(ch,fp2);
    } while (ch!=EOF);
    fclose(fp1);
    fclose(fp2);
}
int main(void) {
    fun("a.txt", "b.txt");
    return 0;
}
```



檔案開啟關閉

- FILE *fopen(const char *filename, const char *mode)
 - filename – 檔案名稱。
 - mode –
 - "r" – 讀取，檔案必須存在。
 - "w" – 創建空的檔案。若已存，內容被刪除。
 - "a" – 附加到檔案中。寫入資料加在檔案末尾。檔案若不存在則新創。
 - "r+" – 開啟更新檔案讀取和寫入。檔案必須存在。
 - "w+" – 創建空檔案，讀取和寫入。
 - "a+" – 開啟檔案讀取和追加。
 - 回傳值 – FILE指標，若失敗將回傳NULL。
- int fclose(FILE *stream)

檔案讀取與寫入

- int fgetc(FILE *stream)
 - stream – 檔案指標，標識要執行的操作串流。
 - 回傳值，讀取的字元為unsigned char轉換為int或EOF檔案結束或錯誤。
- int fputc(int char, FILE *stream)
 - char – 要寫入的字元。
 - stream – 檔案指標。
 - 回傳值-有錯誤，已寫入相同的字元被回傳。出現錯誤，回傳EOF。

檔案格式化讀取與寫入

- int fscanf(FILE *stream, const char *format, ...)
 - stream – 檔案指標。
 - format – 與 scanf 用法相同。
- int fprintf(FILE *stream, const char *format, ...)
 - format – 與 printf 用法相同。
- char *fgets(char *str, int n, FILE *stream)
 - n – 最大被讀取字元數(含結束符號)
 - str – 將檔案資料讀到 str 字串
- int fputs(const char *str, FILE *stream)

檔案格式化讀取與寫入

```
#include <stdio.h>
#include <stdlib.h>
void fun(char fileName[]) {
    char s[3][10];
    int year;
    FILE * fp;
    fp = fopen (fileName, "w+");
    fputs("We are in 2022", fp);    //寫入檔案
    rewind(fp); //重回檔案開頭，準備從檔案讀出
    fscanf(fp, "%s %s %s %d", s[0], s[1], s[2], &year);
    printf("%s %s %s %d", s[0], s[1], s[2], year);
    fclose(fp);
}
int main(void) {
    fun("a.txt");
    return 0;
}
```

We are in 2022

檔案格式化讀取與寫入

```
#include <stdio.h>
#include <stdlib.h>
void fun(char fileName[]) {
    char s[80];
    int year;
    FILE * fp;
    fp = fopen (fileName, "w+");
    fprintf(fp, "%s %s in $%d.", "we", "are", 2002); //格式化寫入資料
    rewind(fp);      //重回檔案開頭
    fgets(s, 50, fp); // 從檔案讀出資料
    printf("=> %s", s);
    fclose(fp);
}
int main(void) {
    fun("a.txt");
    return 0;
}
```

=> we are in \$2002.

循序與隨機讀寫

- int fseek(FILE *stream, long int offset, int where)
 - stream – 檔案指標。
 - offset – 從 where 開始的位移量。
 - where – 指定起始點：
 - SEEK_SET 檔案開頭
 - SEEK_CUR 檔案指標目前位址
 - SEEK_END 檔案結尾
 - 成功回傳 0，否則回傳非 0。

檔案格式化讀取與寫入

```
#include <stdio.h>
#include <stdlib.h>
void fun(char fileName[]) {
    FILE *fp;
    char s[80];
    fp = fopen(fileName, "w+");
    fputs("This is a book", fp);
    fseek( fp, 7, SEEK_SET );
    fputs(" C Programming", fp);
    fseek( fp, 0, SEEK_SET );
    fgets(s, 80, fp);
    printf("=>%s\n", s);
    fclose(fp);
}
int main(void) {
    fun("a.txt");
    return 0;
}
```

=> This is C Programming

Binary檔案 fread

- fopen() 設定模式
 - fopen(filename, "rb")；讀取 binary 檔案
 - fopen(filename, "wb")；將資料寫入 binary 檔案中
 - fopen(filename, "ab")；將資料加到 binary 檔案的最後面
- size_t fread(void *p, size_t size, size_t n, FILE *fp)
 - p 從檔案讀資料，將資料存入變數位址。
 - size 從檔案讀資料，每次讀取size大小(單位byte)。
 - n 讀取的個數，每一個都是size個byte。
 - fp 檔案指標。
 - 成功回傳讀取的元素總數。若不同於n，為發生錯誤或達到檔案末端。

檔案fread

```
#include <stdio.h>
#include <stdlib.h>
void fun(char fileName[]) {
FILE *fp;
char s2[80], s1[40] = "This is an apple";
fp = fopen(fileName, "w+");
fwrite(s1, 2*sizeof(char),5, fp ); //從 s1 每次讀 2 byte , 讀 5 個
rewind(fp);
fgets(s2, 60, fp);
printf("%s\n", s2);
fclose(fp);
}
int main(void) {
fun("a.txt");
return 0;
}
```

This is an

檔案fwrite

- `size_t fwrite(void *p, size_t size, size_t n, FILE *fp)`
 - p 將變數位址資料寫入檔案。
 - size 每次從p所指位址，讀取size byte資料寫入檔案。
 - n 寫入的個數，每一個都是size byte。
 - fp 檔案指標。
 - 成功回傳寫入的元素總數。若不同於n，為發生錯誤。

檔案fwrite

```
#include <stdio.h>
#include <string.h>
void fun(char fileName[]) {
    FILE *fp;
    char s1[80], s2[] = "This is an apple";
    long lSize;
    fp = fopen(fileName, "w+");
    fwrite(s2, strlen(s2) + 1, 1, fp);
    fseek(fp, 0, SEEK_END); //檔案指標指到檔案結尾
    lSize = ftell(fp);      //目前檔案指標位置距離
    rewind(fp);             //檔案指標回到檔案開頭
    fread(s1, lSize, 1, fp); //讀檔案
    printf("%s\n", s1);
    fclose(fp);
}
int main() {
    fun("a.txt");
    return(0);
}
```

This is an apple

struct 檔案 fwrite fread

```
#include <stdio.h>
#include <string.h>
typedef struct student_s {
    char name[8];
    int id;
    double score;
} student_t;
void fun(char fileName[]) {
    FILE *fp;
    student_t s2, s1 = {"John", 10590011, 90.5};
    long lSize;
    fp = fopen(fileName, "wb+");
    fwrite(&s1, sizeof(student_t), 1, fp);
    fseek(fp, 0, SEEK_END); // 檔案指標指到檔案結尾
    lSize = ftell(fp);      // 目前檔案指標位置距離
    rewind(fp);             // 檔案指標回到檔案開頭
    fread(&s2, sizeof(student_t), 1, fp); // 讀檔案
    printf("%s, %d, %.2f\n", s2.name, s2.id, s2.score);
    fclose(fp);
}
int main() { fun("a.bin"); return(0);}
```

John, 10590011, 90.50

計算機程式設計

C語言 Program in Large

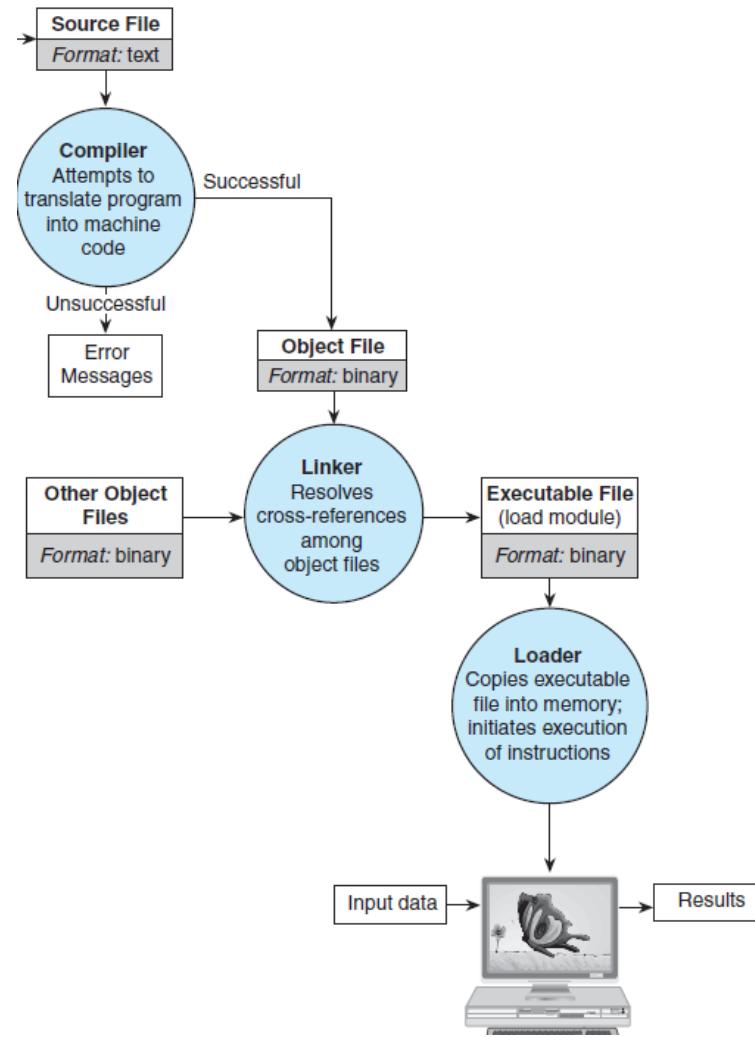
郭忠義

jykuo@ntut.edu.tw

臺北科技大學資訊工程系

程式執行

□ 程式撰寫、編譯、連結



自訂函式庫

□ extern

- 宣告變數、函式在其他地方定義，"其他地方"可能是同一個檔案，或其他檔案。

□ 步驟

- (Code:Block)造一個project檔案，加入 main.c 和 some.c
- 編譯執行

```
// some.c
double some_var = 1000.0;
```

```
// main.c
#include <stdio.h>
int main() {
    extern double some_var;
    printf("%f\n", some_var);
    return 0;
}
```

自訂函式庫

□ extern

- 在使用 `extern` 同時指定值，則視為在該位置宣告定義變數，將造成重覆定義錯誤。
- 須先宣告 `extern` 找到變數，再重新指定其值。

```
// main.c
#include <stdio.h>
int main() {
    extern double some_var = 100; // 錯誤
    printf("%f\n", some_var);
    return 0;
}
```

```
// main.c
#include <stdio.h>
int main() {
    extern double some_var;
    some_var = 100; // 正確
    printf("%f\n", some_var);
    return 0;
}
```

自訂函式庫

□ 標頭檔案 Header

- 註解區塊，說明函式庫的功能目的
- #define 宣告、命名、常數和巨集
- 資料型別定義 typedef
- 針對函式庫的每一個函式，有一個註解區塊，說明該函式的功能，以及使用外部宣告(extern)定義的函式

自訂函式庫

□ 標頭檔案 Header

```
// planet.h - abstract data type planet
// Type planet_t has these components:
// name, diameter, moons, orbit_time, rotation_time
// Operators:
// print_planet, planet_equal, scan_planet
#define PLANET_STRSIZ 10
typedef struct {      // planet structure
    char name[PLANET_STRSIZ];
    double diameter;   // equatorial diameter in km
    int moons;         // number of moons
    double orbit_time, // years to orbit sun once
           rotation_time; // hours to complete one revolution on axis
} planet_t;
// Displays with labels all components of a planet_t structure
extern void print_planet(planet_t pl); // input - one planet structure
```

自訂函式庫

□ 標頭檔案 Header

```
// Determines whether or not the components of planet_1 and planet_2 match  
// input - planets to compare  
  
extern int planet_equal(planet_t planet_1, planet_t planet_2);  
// Fills a type planet_t structure with input data. Integer returned as  
// function result is success/failure/EOF indicator.  
// 1 => successful input of planet  
// 0 => error encountered  
// EOF => insufficient data before end of file  
// In case of error or EOF, value of type planet_t output argument is undefined.  
extern int  
scan_planet(planet_t *plnp); /* output - address of planet_t structure to fill */
```

自訂函式庫

□ 實做 Implementation

```
// planet.c
#include <stdio.h>
#include <string.h>
#include "planet.h"
// Displays with labels all components of a planet_t structure
void print_planet(planet_t pl) {          // input - one planet structure
    printf("%s\n", pl.name);
    printf(" Equatorial diameter: %.0f km\n", pl.diameter);
    printf(" Number of moons: %d\n", pl.moons);
    printf(" Time to complete one orbit of the sun: %.2f years\n", pl.orbit_time);
    printf(" Time to complete one rotation on axis: %.4f hours\n", pl.rotation_time);
}
```

自訂函式庫

□ 實做 Implementation

```
// Determines whether or not the components of planet_1 and planet_2 match
// input - planets to compare
int planet_equal(planet_t planet_1, planet_t planet_2) {
    return (strcmp(planet_1.name, planet_2.name) == 0 &&
    planet_1.diameter == planet_2.diameter &&
    planet_1.moons == planet_2.moons && planet_1.orbit_time ==
    planet_2.orbit_time && planet_1.rotation_time == planet_2.rotation_time);
}
```

自訂函式庫

□ 實做 Implementation

```
// Fills a type planet_t structure with input data. Integer returned as
// function result is success/failure/EOF indicator.
// 1 => successful input of planet
// 0 => error encountered
// EOF => insufficient data before end of file
// In case of error or EOF, value of type planet_t output argument is undefined.
// output - address of planet_t structure tofill
int scan_planet(planet_t *plnp) {
    int result;
    result = scanf("%s%lf%d%lf%lf", plnp->name, &plnp->diameter,
                  &plnp->moons, &plnp->orbit_time, &plnp->rotation_time);
    if (result == 5)
        result = 1;
    else if (result != EOF)
        result = 0;
    return (result);
}
```

使用自訂函式庫

□ 標頭檔案 Header

```
// Beginning of source file in which a personal library and system I/O library  
// are used.
```

```
#include <stdio.h> // system's standard I/O functions  
#include "planet.h" // personal library with planet_t data type and operators  
int main() {  
    return 0;  
}
```

變數等級(Storage Class)

□ 全域變數 (global variable)

- 宣告在所有函式外面。
- 生命週期，編譯完就配置空間，直到程式結束。
- 避免使用全域變數，因每個函式都可變更其值
- 全域常數，可以使用。

```
// eg1.c
int global_var_x;
const size = 10;
const char *months[12] = {"January", "February", "March", "April", "May",
    "June", "July", "August", "September", "October", "November", "December"};
void f() {    global_var_x = 10; }
int main() {
    global_var_x = 20;
    f();
    return 0;
}
```

變數等級(Storage Class)

□ 外在全域變數 (global variable)

- extern 全域變數，宣告在其他地方，故在此不配置記憶體空間。

```
// eg2.c
extern int global_var_x;
void g() {
    global_var_x = 5;
}
int main() {
    g();
    return 0;
}
```

變數等級(Storage Class)

□ 區域變數 auto

- many，一般區域變數，auto等級，存放於堆疊stack。
- 生命週期，宣告開始，直到第一個Block結束(**遇到右大括號**)。
- 每次呼叫f函式，many會重新配置記憶體空間，被重新初始化為 0。

```
void f() {  
    int many = 0;  
    many = many + 1;  
    if (many>=1) {  
        int many = 0;  
        printf("%d\n", many); //0  
    }  
    printf("%d\n", many); // 1  
}  
int main() {  
    f();  
    return 0;  
}
```

變數等級(Storage Class)

□ 靜態變數 static

- once , static , 編譯後程式執行之前就(只有)會配置一次記憶體空間，只初始化其值一次，每次f函式變更其值，都會被記住。
- 生命週期，程式開始執行，直到程式結束。

```
int f () {  
    static int once = 0;  
    once = once +1;  
    return once;  
}  
int main() {  
    printf("%d\n", f()); //1  
    printf("%d\n", f()); //2  
    printf("%d\n", f()); //3  
    return 0;  
}
```

變數等級(Storage Class)

- 暫存器變數 register
 - 使用CPU的暫存器，存取速度快

```
static double matrix[50][40];  
register int row, col;
```

條件式編譯

- 追蹤 printf 偵錯，#if - #endif
 - 偵錯模式，#define TRACE

```
// Computes an integer quotient (m/n) using subtraction
// #define TRACE
int quotient(int m, int n){
    int ans;
#if defined (TRACE)
    printf("Entering quotient with m = %d, n = %d\n", m, n);
#endif
    if (n > m) ans = 0;
    else ans = 1 + quotient(m - n, n);
#if defined (TRACE)
    printf("Leaving quotient(%d, %d) with result = %d\n", m, n, ans);
#endif
    return (ans);
}
```

條件式編譯

□ 追蹤 printf 偵錯，#if - #elif - #endif

```
// #define TRACE_VERBOSE #define TRACE_BRIEF
int quotient(int m, int n) {
    int ans;
#if defined (TRACE_VERBOSE)
    printf("Entering quotient with m = %d, n = %d\n", m, n);
#elif defined (TRACE_BRIEF)
    printf(" => quotient(%d, %d)\n", m, n);
#endif
    if (n > m) ans = 0;
    else ans = 1 + quotient(m - n, n);
#if defined (TRACE_VERBOSE)
    printf("Leaving quotient(%d, %d) with result = %d\n", m, n, ans);
#elif defined (TRACE_BRIEF)
    printf("quotient(%d, %d) => %d\n", m, n, ans);
#endif
    return (ans);
}}
```

自訂函式庫

□ 標頭檔案 Header ，防止重複被包含

```
// planet.h - abstract data type planet
// Type planet_t has these components:
// name, diameter, moons, orbit_time, rotation_time
// Operators:
// print_planet, planet_equal, scan_planet
#if !defined (PLANET_H_INCL)
#define PLANET_H_INCL
#define PLANET_STRSIZ 10
typedef struct {      // planet structure
    char name[PLANET_STRSIZ];
    double diameter;   // equatorial diameter in km
    int moons;         // number of moons
    double orbit_time, // years to orbit sun once
           rotation_time; // hours to complete one revolution on axis
} planet_t;
// Displays with labels all components of a planet_t structure
extern void print_planet(planet_t pl); // input - one planet structure
```

自訂函式庫

□ 標頭檔案 Header ，防止重複被包含

```
// Determines whether or not the components of planet_1 and planet_2 match  
// input - planets to compare  
  
extern int planet_equal(planet_t planet_1, planet_t planet_2);  
// Fills a type planet_t structure with input data. Integer returned as  
// function result is success/failure/EOF indicator.  
// 1 => successful input of planet  
// 0 => error encountered  
// EOF => insufficient data before end of file  
// In case of error or EOF, value of type planet_t output argument is undefined.  
  
extern int  
scan_planet(planet_t *plnp); // output - address of planet_t structure to fill  
  
#endif
```

argc & argv

- 命令列引數，只有main可以使用；是字串型態
 - argc:引數的個數(整數)，argv: 指向字元指標陣列的指標
 - 命令列引數必須由空格分開
 - argv[0]表示程式名稱，argv[1]表示第一個引數

```
// Test.c      Test.c Tom John
#include <stdio.h>
int main(int argc, char * argv[]) {
    int t,i;
    if (argc<2) {
        printf("you forgot to type your name\n");
        exit(0);
    }
    for (t=0; t<argc; t++) {
        printf("Hi~ %s\n", argv[t]);
    }
    return 0;
}
```

```
Hi~ Test.c
Hi~ Tom
Hi~ John
```

巨集(macro)

□ 巨集處理

- 前置處理器，將程式中巨集定義的符號，換成巨集定義後面的符號

```
#include <stdio.h>
#define LABEL_PRINT_INT(label, num) printf("%s = %d", (label), (num))
int main() {
    int r = 5, t = 12;
    LABEL_PRINT_INT("rabbit", r);
    // 變成 printf("%s=%d", ("rabbit"), (r));
    printf(" ");
    LABEL_PRINT_INT("tiger", t + 2); // t+22; 當成 num，換成 (t+2)
    // 變成 printf("%s=%d", ("tiger"), (t+2));
    printf("\n");
    return(0);
}

// rabbit = 5 tiger = 14
```

巨集(macro)

□ 巨集與函式

- 巨集純粹使用符號取代，不是函式的呼叫
- 巨集會使程式碼變大，函式會使用堆疊stack，呼叫執行讓程式較慢

□ 巨集換行

```
#include <stdio.h>
#define INDEXED_FOR(ct, st, end) \
for ((ct) = (st); (ct) < (end); ++(ct))

int main() {
    int i = 0;
    INDEXED_FOR(i, 0, X_MAX)
        printf("x[%2d] = %6.2f\n", i, x[i]);
    return(0);
}
```

```
for ((i) = (0); (i) < (X_MAX); ++(i))
    printf("x[%2d] = %6.2f\n", i, x[i]);
```

巨集(macro)

□ 巨集處理要加括號

```
#define SQUARE(n) n * n // 應該要寫成 #define SQUARE(n) ((n) * (n))
int main() {
    double x = 0.5, y = 2.0;
    int n = 4, m = 12;
    printf("(%.2f + %.2f) squared = %.2f\n\n", x, y, SQUARE(x + y));
    printf("%d squared divided by ", m);
    printf("%d squared is %d\n", n, SQUARE(m) / SQUARE(n));
    return 0;
}

// SQUARE(x + y)          變成 x + y * x + y
// SQUARE(m) / SQUARE(n)) 變成 m * m / n * n
//      印出 (0.5 + 2.0)squared = 3.5, 12 squared divided by 4 squared is 144
// 應該印出 (0.5 + 2.0)squared = 6.25, 12 squared divided by 4 squared is 9
```

計算機程式設計

C語言 Link List

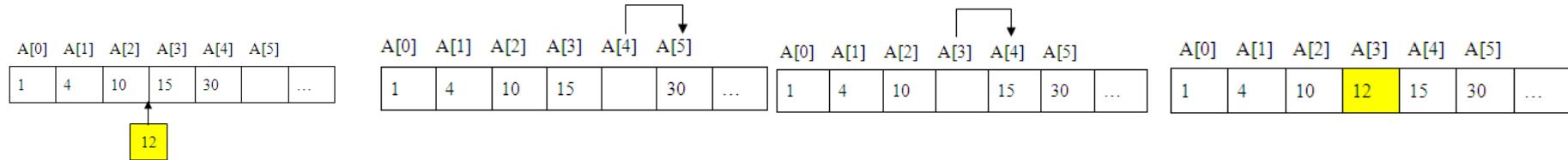
郭忠義

jykuo@ntut.edu.tw

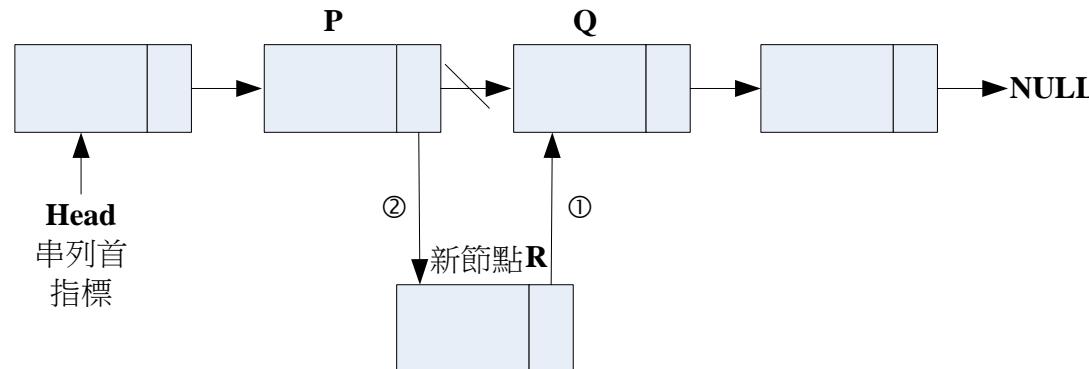
臺北科技大學資訊工程系

有序串列與無序串列

- 有序串列(ordered list)：如陣列(靜態資料結構)
 - 資料儲存在連續記憶空間，無法任意增/刪空間。
 - 陣列欲插入數字12到10與15之間，搬移過程。



- 無序串列(unordered list)-**鏈結串列(Link List)** 動態資料結構
 - 資料儲存在非連續性記憶空間，以指標彈性在串列增減元素。



陣列與鏈結串列

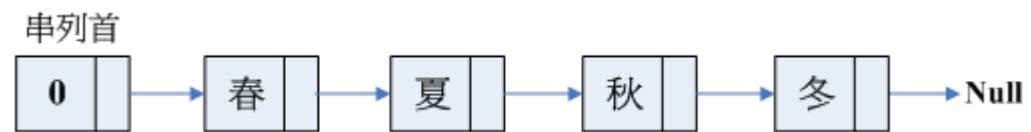
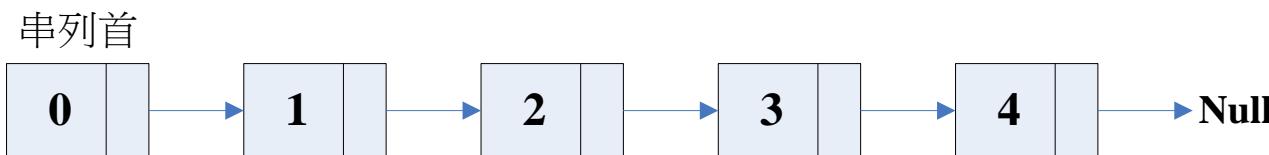
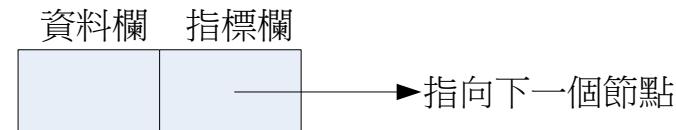
□ 利用指標串接節點

靜態資料結構(如：陣列)	動態資料結構(如：鏈結串列)
較節省記憶體空間	較浪費記憶體空間，須多出一個指標
加入、刪除及合併須大量資料移動	記憶點配置較有彈性 加入、刪除及合併，只須改變指標即可
可以直接存取	不可以直接存取
可進行二分法搜尋	搜尋某元素可能較耗時 指標(point)斷裂時，資料會遺失(lost)

動態資料結構

- 每個節點儲存資料、指向 下一個節點位置，最後節點指向 Null。

```
typedef struct node_s {  
    int data ;  
    struct node_s * next ;  
} node_t;  
typedef node_t * nodep_t
```



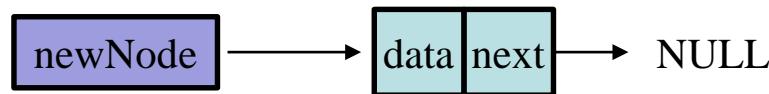
記憶體動態配置

```
#include <stdlib.h>
#include <stdio.h>
int main() {
    char *str;
    if (( str = (char *) malloc(80*sizeof(char))) == NULL) {
        printf("\1: unable to allocate memory for str.\n");
        exit(1);
    }
    printf("\1: Please input any sentence.\n");
    gets(str);
    printf("\2: The string that you input is.\n");
    puts(str);
}
```

動態資料結構

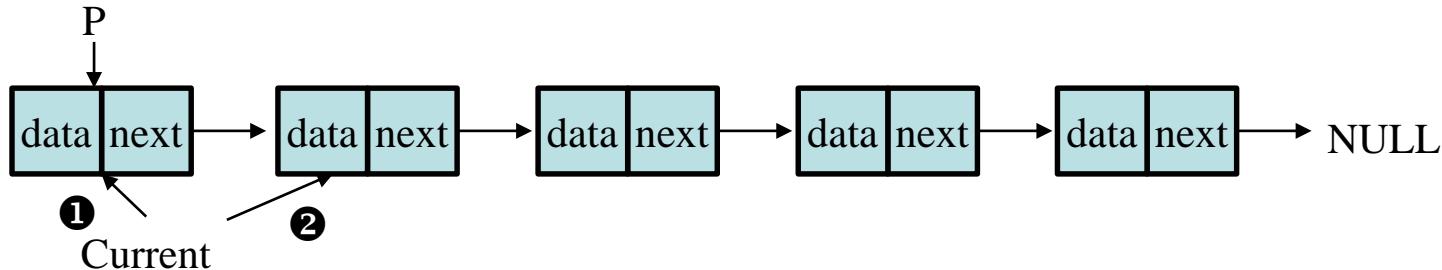
```
typedef struct node {                                //定義 node_t
    int data;
    struct node *next;
} node_t;
typedef node_t * nodep_t;                          //定義 nodep_t

nodep_t create(int data) {                         //造出 newNode
    nodep_t newNode;                               //newNode是指向 node_t 的指標變數
    newNode=(nodep_t)malloc(sizeof(node_t)); //配置記憶體空間
    newNode->data = data;
    newNode->next=NULL;                           //將新 node指標回傳
    return newNode;
}
```



串列尋訪列印

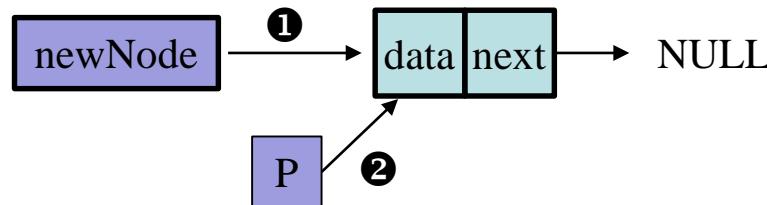
- 設定current指標指向串列首，再一一的往下一個node移動。



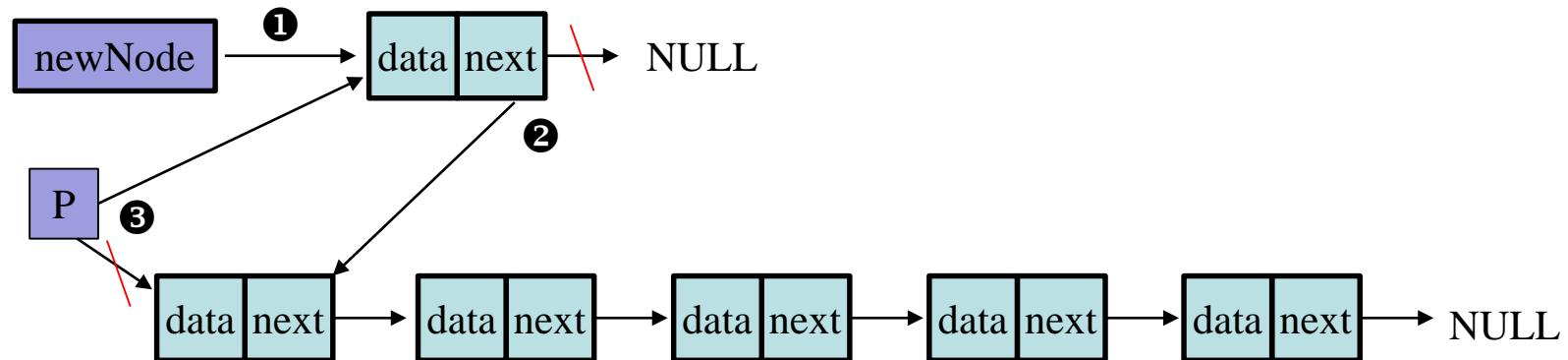
```
void printList(nodep_t p) {                                //尋訪串列並列印
    nodep_t current = p;
    printf("\n The list is: ");
    while(current!=NULL) {
        printf("%d, ",current->data);
        current = current->next;
    }
}
```

串列從前面加入節點

- 空串列，把根指標p指向新造出的節點newNode。



- 非空串列，把新節點指標指向串列首，再把串列首指到新節點上。



串列從前面加入節點

- 空串列，把根指標p指向新的節點。
- 非空串列，把新節點指標指向串列首，再把串列首移到新節點上。

```
void insertFromFront(nodep_t *p, int data) {  
    nodep_t newNode = create(data);  
    if ((*p)==NULL) {  
        (*p) = newNode;  
    }  
    else {  
        newNode->next = (*p);  
        (*p) = newNode;  
    }  
}
```

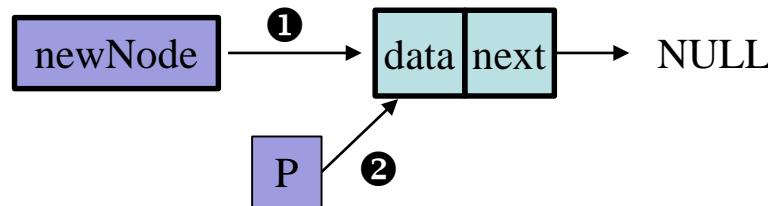
```
nodep_t p;  
insertFromFront(&p, 15);
```

```
nodep_t void insertFromFront(nodep_t p, int data)  
{  
    nodep_t newNode = create(data);  
    if (p==NULL) {  
        p = newNode;  
    }  
    else {  
        newNode->next = p;  
        p = newNode;  
    }  
    return p;  
}
```

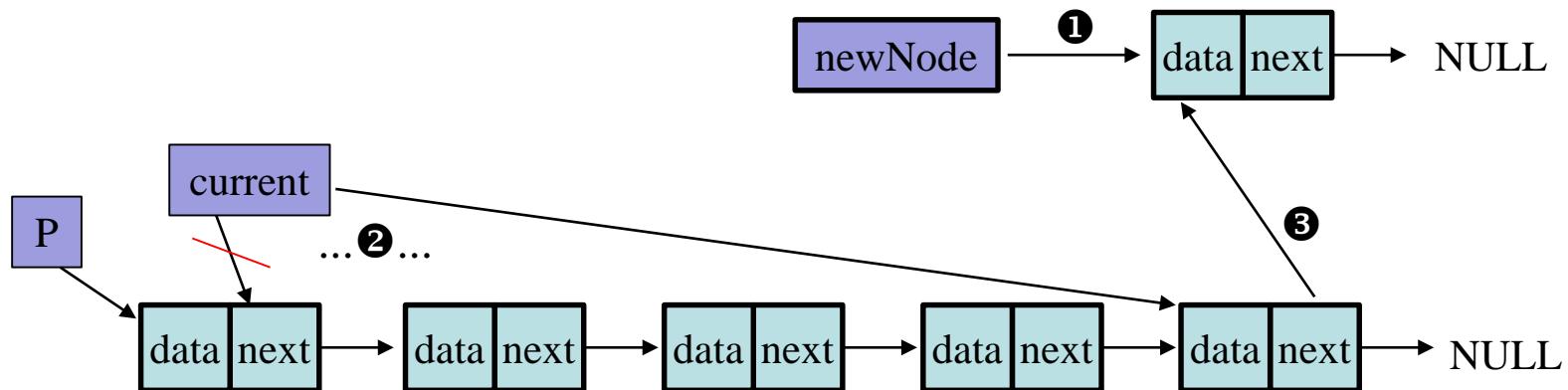
```
nodep_t p;  
p = insertFromFront(p, 15);
```

串列從後面加入節點

- 空串列，把根指標p指向新造出的節點newNode。



- 非空串列，設定current指標變數從頭開始尋訪到尾，把current指標的next指向新節點上。



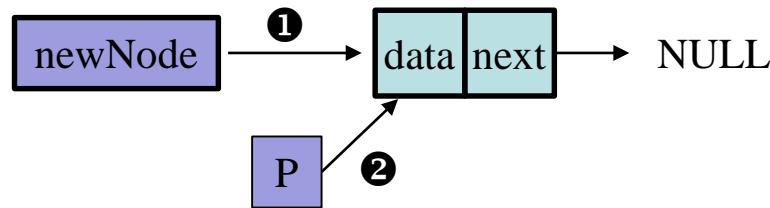
串列從後面加入節點

- 空串列，把根指標p指向新造出的節點newNode。
- 非空串列，設定current指標變數從頭開始尋訪到尾，把current指標的next指向新節點上。

```
void insertFromBack(nodep_t *p, int data) {  
    nodep_t current;  
    nodep_t newNode = create(data);  
    if ((*p)==NULL) {  
        (*p) = newNode;  
    }  
    else {  
        current = (*p);  
        while (current->next!=NULL)  
            current = current->next;  
        current->next = newNode;  
    }  
}
```

串列加入節點排序(疊代)

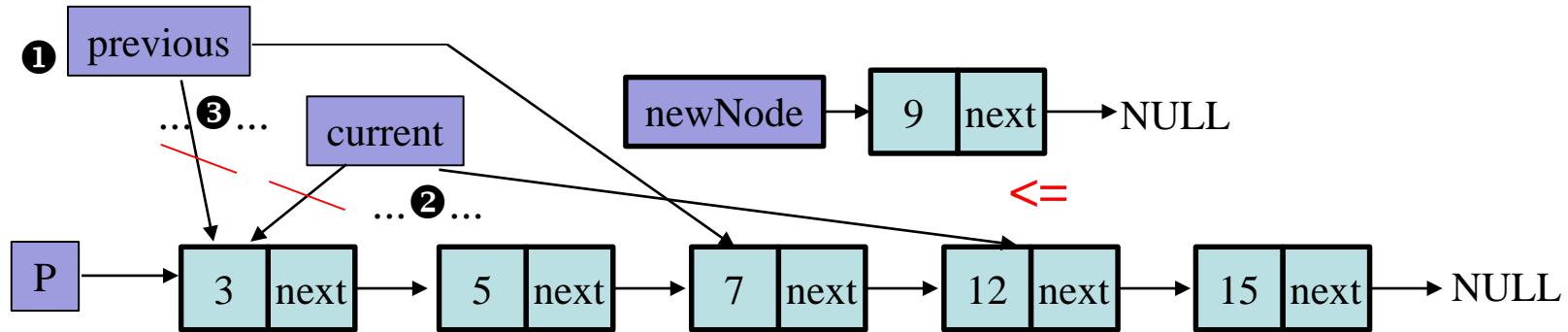
- 空串列，把根指標p指向新造出的節點newNode。



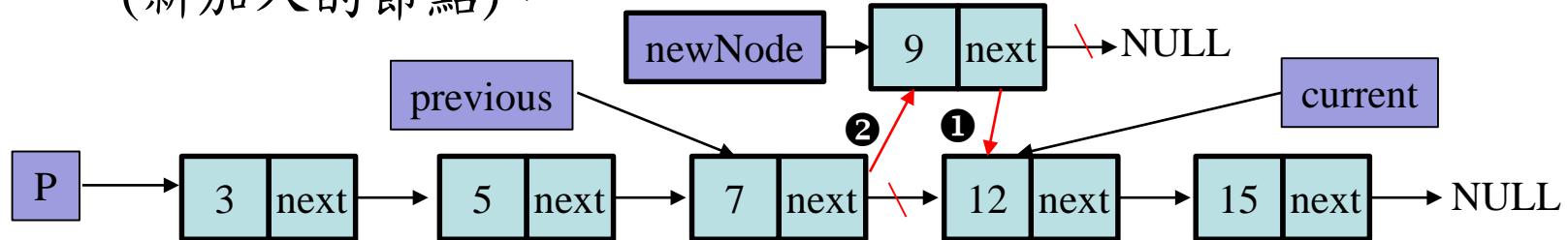
```
nodep_t newNode, current, previous;  
newNode = create(data);  
if (p ==NULL) { return newNode; }  
else if ((p->data) >= data) {  
    newNode->next = p;  
    return newNode;  
}
```

串列加入節點排序(疊代)

- 非空串列，設定current指標從頭開始尋訪，直到newNode的data(新加入 9)小於current指標指向node的data (12)。
 - 設定previous指標，在current前面



- 將newNode指向節點 9 的next，指向current指標所指節點 12，
- 將previous指向的節點(7)的next，指向newNode指標所指位址
(新加入的節點)，

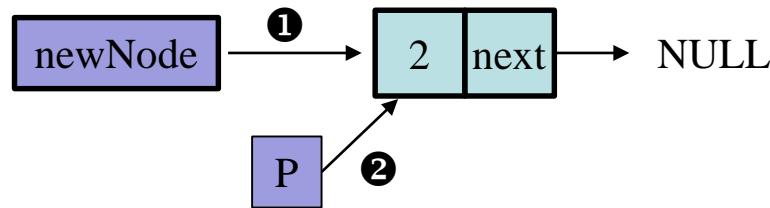


串列加入節點排序(疊代)

```
nodep_t insertInOrderI(nodep_t p, int data) {  
    nodep_t newNode, current, previous;  
    newNode = create(data);  
    if (p ==NULL) { return newNode; }  
    else if ((p->data) >= data) {  
        newNode->next = p;  
        return newNode;  
    }  
    else {  
        current = previous = p;  
        while (current !=NULL) {  
            if ((current->data) < data) {  
                previous = current;  
                current = current->next;  
            }  
            else break;  
        }  
        previous->next = newNode;  
        newNode->next = current;  
        return p;  
    }  
    return (p);  
}
```

串列加入節點排序(遞迴)

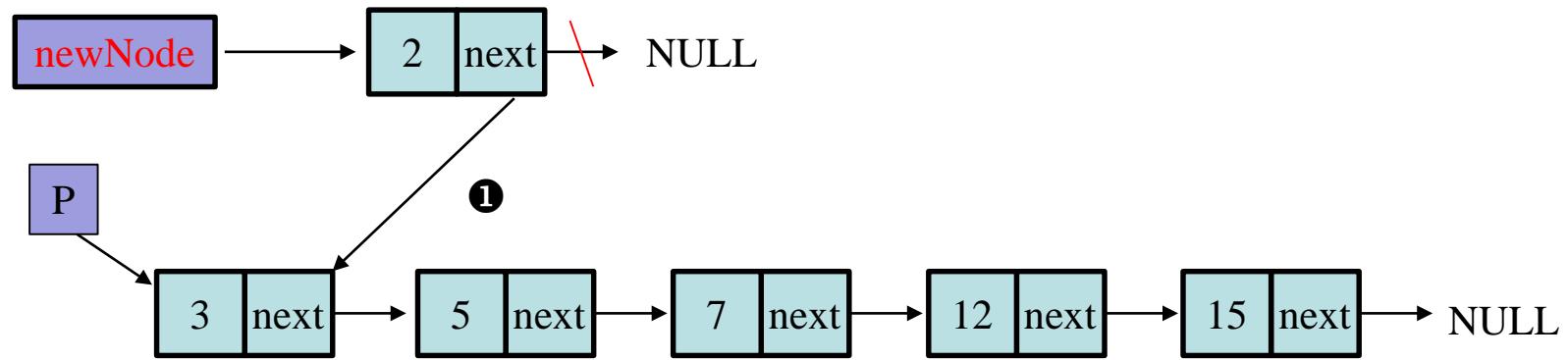
- 空串列，把根指標p指向新造出的節點newNode。



```
nodep_t newNode, current, previous;  
if (p ==NULL) { return create(data); }
```

串列加入節點排序(遞迴)

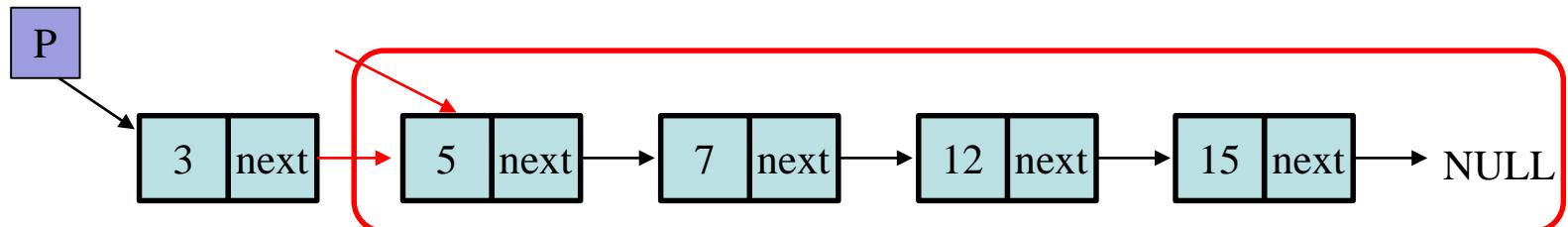
- 非空字串，新節點資料小於串列頭的資料。
 - 從前面加入新節點



```
else if ((p->data) >= data) {  
    newNode = create(data);  
    newNode->next = p;  
    return newNode;  
}
```

串列加入節點排序(遞迴)

- 非空字串，新節點資料不小於串列頭的資料。
 - 將問題變小 - 指標往下 next 移動，
 - 呼叫本身函式處理，完成回傳指標接到 p->next



```
else {  
    p->next = insertInOrderR(p->next, data);  
    return p;  
}
```

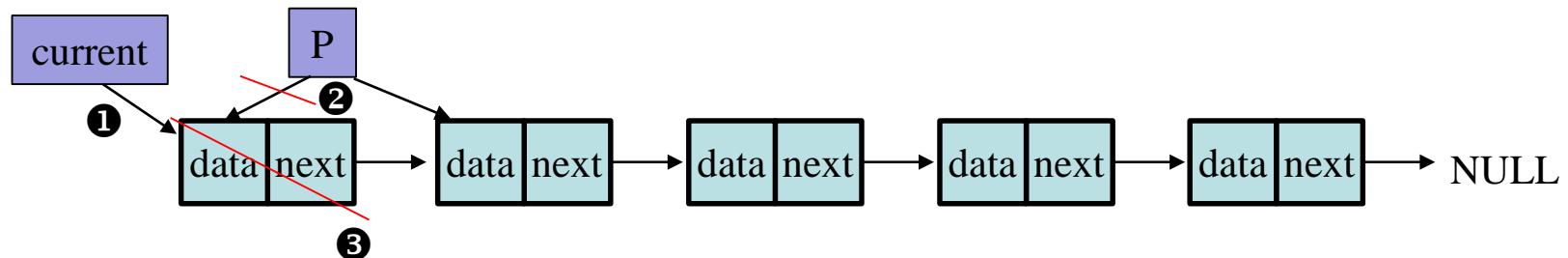
串列加入節點排序(遞迴)

- 加入新節點，把新節點指標指向串列首，再把串列首移到新節點上。

```
nodep_t insertInOrderR(nodep_t p, int data) {  
    nodep_t newp, current, prev;  
    if (p ==NULL) { return create(data); }  
    else if ((p->data) >= data) {  
        newp = create(data);  
        newp->next = p;  
        return newp;  
    }  
    else {  
        p->next = insertInOrderR(p->next, data);  
        return p;  
    }  
}
```

串列從前面刪除節點

- 設定current指標指向 p 所指位址，串列頭，把p指標指向第二個節點，釋放current指標指向的空間，原串列頭。



串列從前面刪除節點

- 設定current指標指向 p 所指位址，串列頭，把p指標指向第二個節點，釋放current指標指向的空間，原串列頭。

```
int delFromFront(nodep_t *p) {  
    int data = -1;  
    nodep_t current;  
    if ((*p)!=NULL) {  
        current = (*p);  
        (*p) = current->next;  
        data = current->data;  
        free(current);  
        return data;  
    }  
    return data;  
}
```

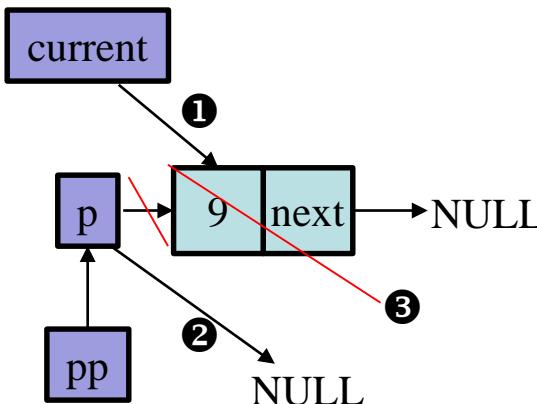
串列從後面刪除節點

- 空串列，回傳-1。

```
nodep_t previous, current;           // (1)
int data = -1;
previous = current = (*pp);
if (current==NULL) return data;
```

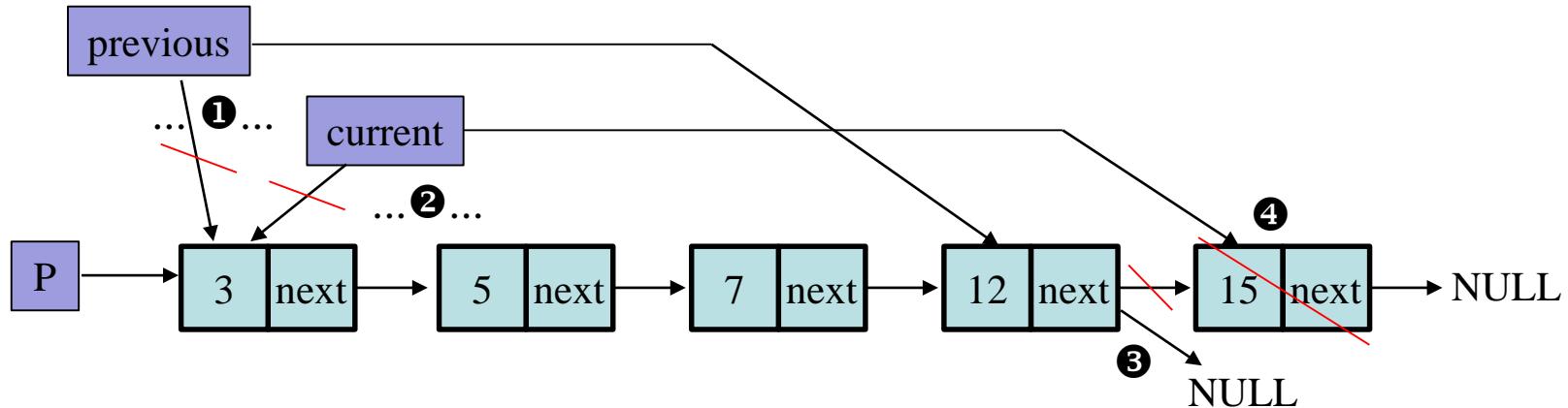
- 串列只有一個，刪除後free記憶體空間，p (*pp) 指向NULL。

```
else if (current->next==NULL) {
    (*pp) = NULL;                  // (2)
    data = current->data;
    free(current);                // (3)
    return data;
}
```



串列從後面刪除節點

- 串列至少二個，
 - 刪除後free記憶體空間， $p(*pp)$ 指向NULL。



```
while (current->next!=NULL) {
    previous = current;          // (1)
    current=current->next;       // (2)
}
data = current->data;
previous->next = NULL;         // (3)
free(current);                // (4)
```

串列從後面刪除節點

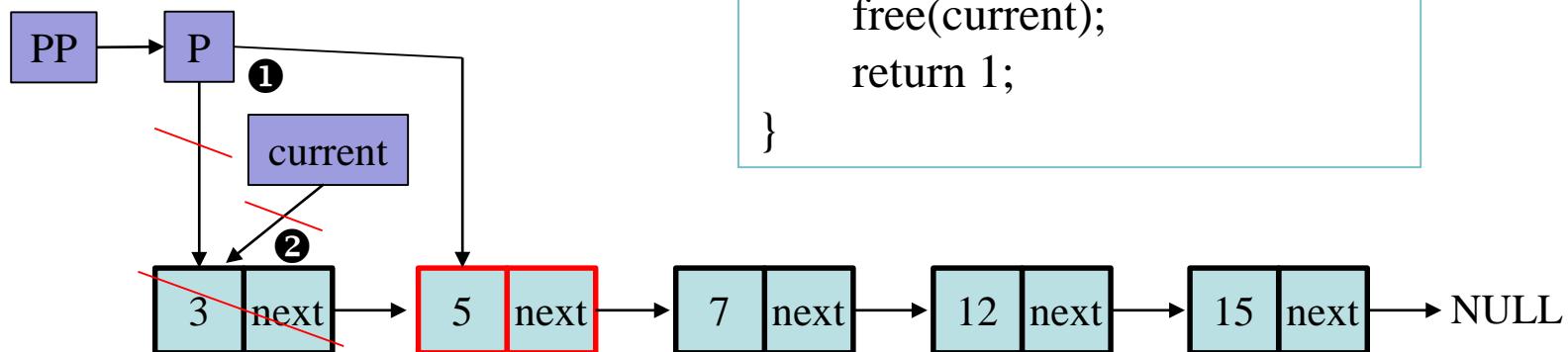
```
int delFromBack(nodep_t *pp) {  
    nodep_t previous, current;  
    int data = -1;  
    previous = current = (*pp);  
    if (current==NULL) return data;  
    else if (current->next==NULL) {  
        (*pp)=NULL;  
        data = current->data;  
        free(current);  
        return data;  
    }  
    while (current->next!=NULL) {  
        previous = current;  
        current=current->next;  
    }  
    data = current->data;  
    previous->next = NULL;  
    free(current);  
    return data;  
}
```

串列根據key刪除節點

- 空串列，回傳-1。

```
nodep_t previous, current;           // (1)  
int data = -1;  
previous = current = (*pp);  
if (current==NULL) return data;
```

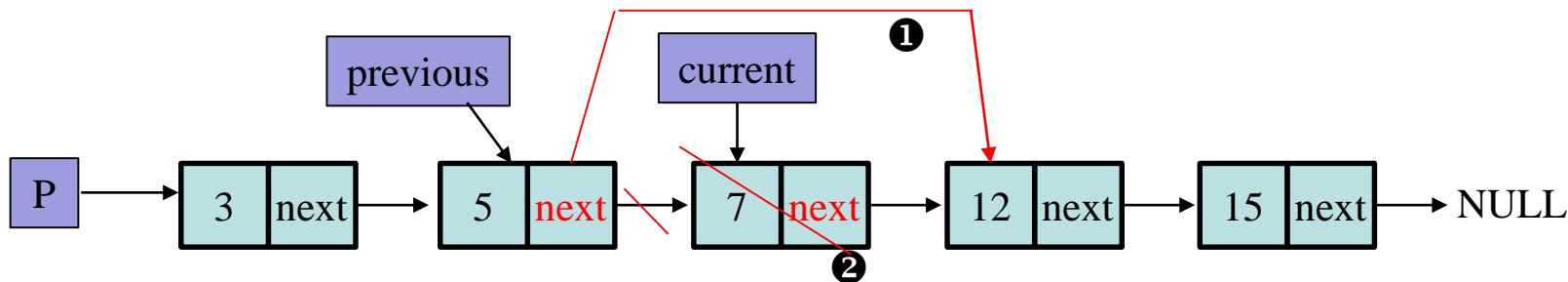
- 第一個找到，刪除後free記憶體空間，p (*pp) 指向下一個。



串列根據key刪除節點

- 一直尋訪直到找到(假設 7)，刪除後free記憶體空間，p(*pp) 指向下一個。

```
while (current!=NULL) {  
    if ((current->data) == key) { // 找到  
        previous->next = current->next; // (1)  
        free(current); // (2)  
        return 1;  
    }  
    else { // 沒找到  
        previous = current; // 一直往後找  
        current = current->next;  
    }  
}
```



串列根據key刪除節點

```
int deleteByKey(nodep_t *pp, int key) {  
    nodep_t previous, current;  
    previous = current = (*pp);  
    if (current==NULL) return -1;  
    else if ((current->data) == key) {  
        (*pp) = (*pp)->next;  
        free(current);  
        return 1;  
    }  
    while (current!=NULL) {  
        if ((current->data) == key) {  
            previous->next = current->next;  
            free(current);  
            return 1;  
        }  
        else {  
            previous = current;  
            current = current->next;  
        }  
    }  
    return -1;  
}
```

抽象資料型態

- 抽象資料型態(Abstract Data Type, ADT)特性與優點
 - 介面操作規範 (The specification of the operation)
 - 函式名稱 (Function Name)
 - 參數型態 (The type of its argument)
 - 結果型態 (The type of its result)
 - 函式功能的描述 (不必說明內部表示法或實作細節)
 - 介面是"封裝隔離"機制，
 - 客戶端程式呼叫存取，僅需操作其所提供的介面函式。
 - 不用也不需要知道內部如何實作。
 - 系統具高維護性與彈性，不論擴充或重構，客戶端程式僅受最小幅度影響。

抽象資料型態

- 抽象資料型態實作獨立(Implementation-Independent)。
- 堆疊STACK實作方法
 - 陣列，會受到大小須事先宣告的限制。
 - 鏈結(node-based List)，以動態記憶體配置方式新增每個元素。
- 靜態記憶體配置，編譯階段要求配置所需記憶體空間。
- 動態記憶體配置，執行階段要求配置所需記憶體空間。
 - 指標變數 = (資料型態*) malloc(sizeof(資料型態));
 - int *pt;
 - pt = (int *) malloc(sizeof(int));
- 釋放記憶體運算子
 - free(指標變數); free(pt);

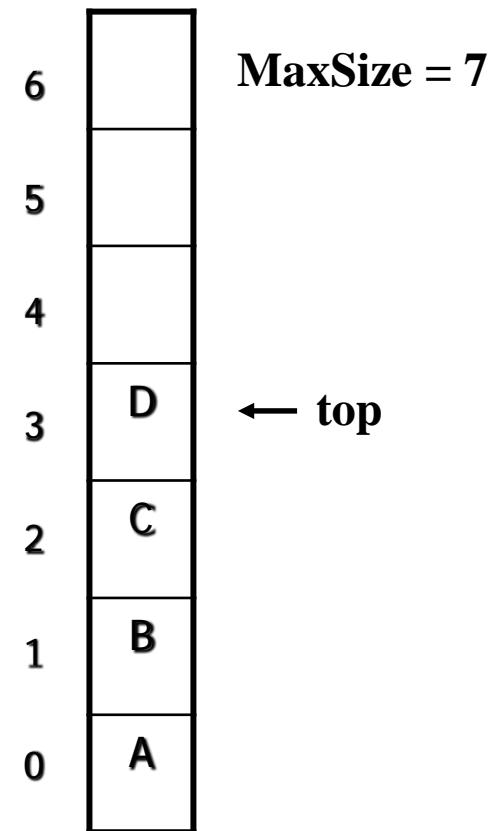
堆疊(Stack)

□ 後進先出(Last In First Out, LIFO)

- 將資料依序從下面(bottom)儲存，從上面(top)將資料取出 (Stack)。

□ 動作：

- create：建立一個空堆疊。
- push：將資料存入堆疊。
- pop：將資料從堆疊中取出。
- empty：判斷堆疊是否為空堆疊。
- full：判斷堆疊是否已滿。



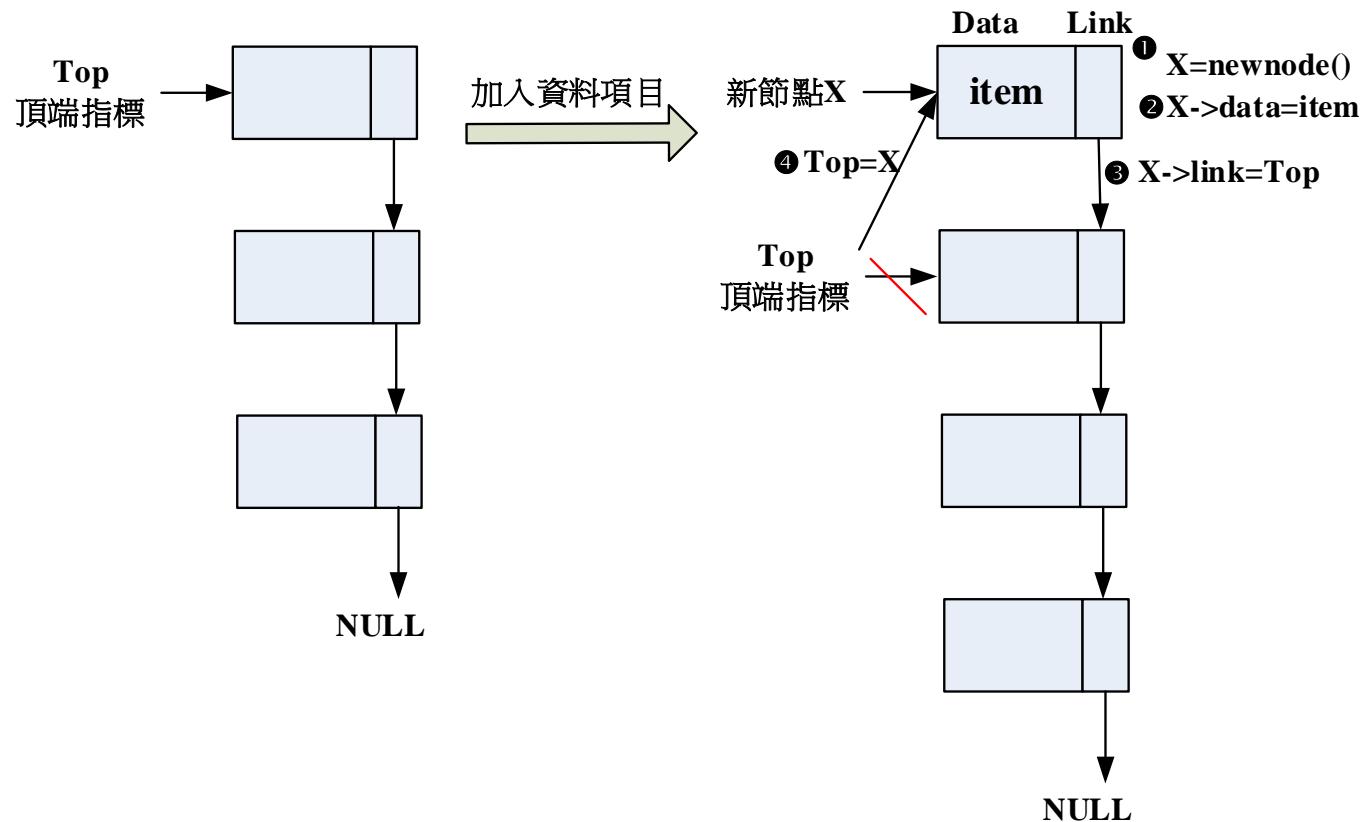
堆疊 – 使用陣列

```
int push(int stack[MaxSize], int *top, int n){  
    if ((*top)<MaxSize) {  
        stack[*top]=n;  
        (*top)++;  
        return 0;  
    }  
    else { return -1; }  
}  
  
int pop(int stack[MaxSize], int *top) {  
    int k;  
    if ((*top)>0) {  
        return (stack[(*top)--]);  
    }  
    else { return -1; }  
}
```

```
int empty(int stack[MaxSize], int *top) {  
    if ((*top)==0) { return 1; }  
    else { return 0; }  
}  
  
int full(int *top){  
    if ((*top)==MaxSize) { return 1; }  
    else { return 0; }  
}
```

堆疊 – 使用鏈結串列加入資料

- 先產生一個新資料項item的新節點，假設此節點為X，其加入新資料項目到鏈結堆疊中的過程如下。

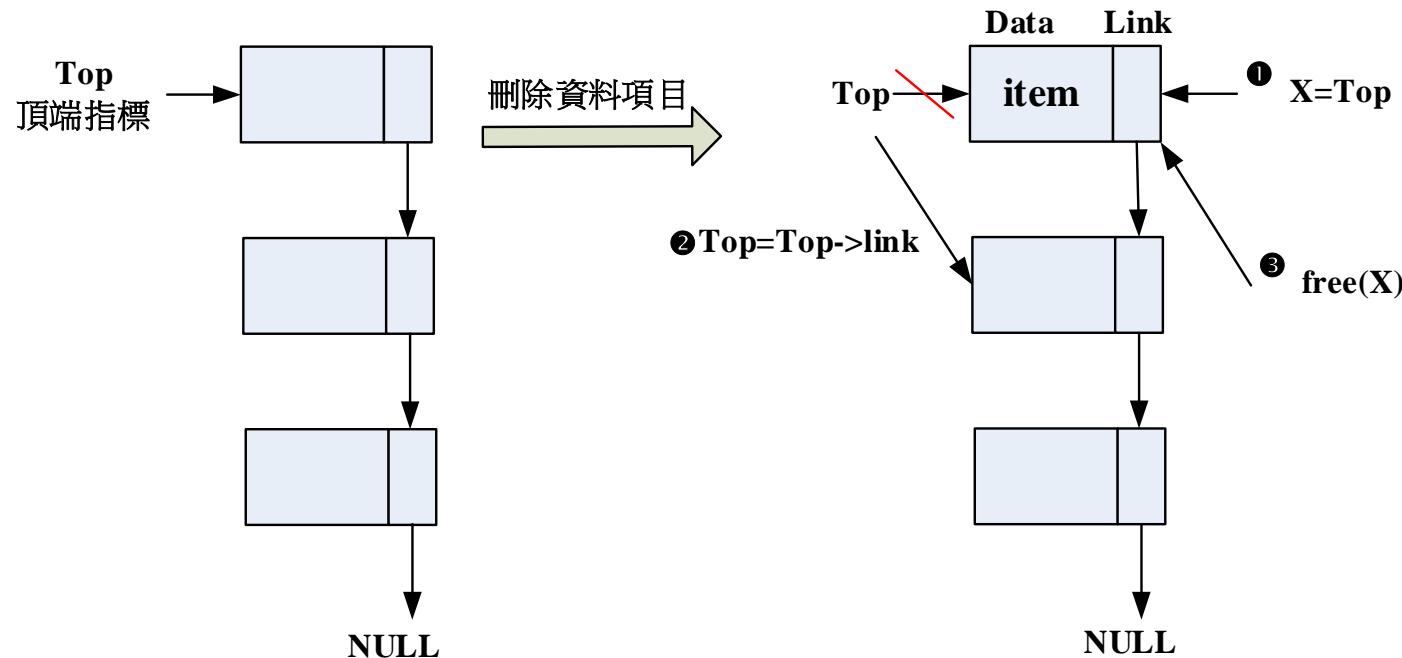


Exercise

- 堆疊 – 使用鏈結串列加入資料

堆疊 – 使用鏈結串列刪除資料

- 刪除頂端資料時，須利用一個指標指向頂端節點，以便在頂端指標Top改為指向第二個節點後，還能參照到原來的頂端節點，且將其空間歸還系統。

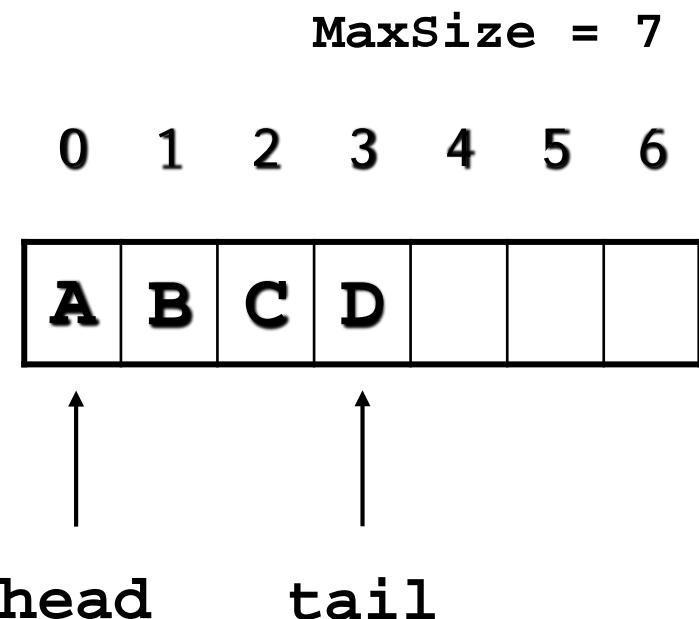


Exercise

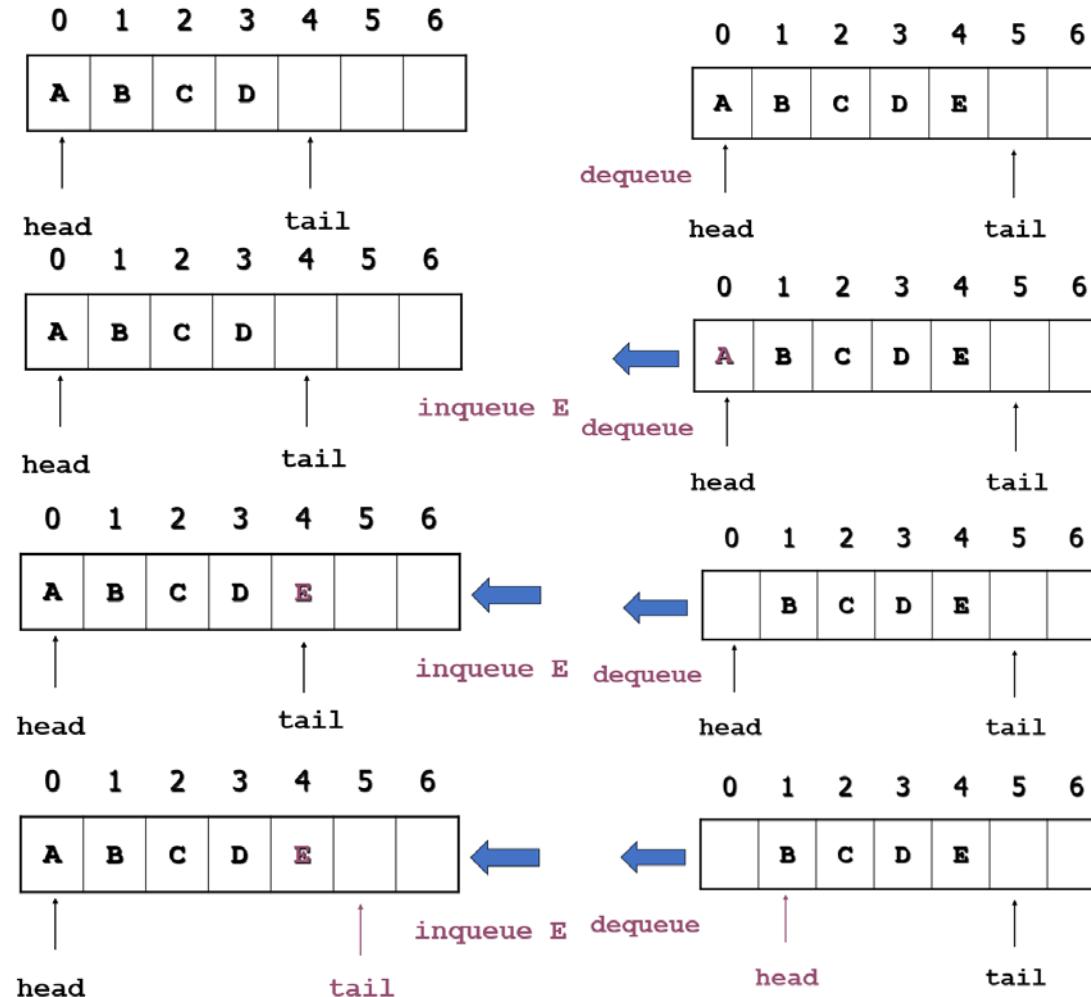
- 堆疊 – 使用鏈結串列刪除資料

佇列的操作

- 佇列(Queue)，後進先出(First In First Out, FIFO)
 - 將資料依序從尾端(tail) 儲存，從開頭(head) 將資料取出。
- 動作：
 - create : 建立一個空佇列。
 - inqueue : 將資料存入佇列中。
 - dequeue : 將資料從佇列中取出。
 - empty : 判斷佇列是否為空佇列。
 - front : 傳回佇列前端 head 的值。



佇列的操作



佇列的操作 – 使用陣列

```
int enqueue(n) {  
    if((tail+1)% MaxSize != head) {  
        queue[tail] = n;  
        tail++;  
        tail = tail % MaxSize;  
        return 0;  
    }  
    else {  
        return -1;  
    }  
}
```

```
int dequeue(int *n) {  
    if (tail != head) {  
        *n = queue[head];  
        head++;  
        head = head % MaxSize;  
        return 0;  
    }  
    else {  
        return -1;  
    }  
}
```

環狀佇列 – 使用陣列

□ 重複利用空間

```
#include <stdio.h>
#define SIZE 3
//#define TRUE 1
//#define FALSE 0
typedef enum{FALSE, TRUE} bool;
bool isEmpty(int front, int back) {
    return (front==back);
}
bool isFull(int front, int back) {
    return (((back+1)%SIZE)==front);
}
bool enqueue(int data[], int index[], int key) {
    //front = index[0];    back = index[1];
    if (isFull(index[0], index[1])) return FALSE;
    index[1] = (index[1]+1)%SIZE;
    data[index[1]] = key;
    return TRUE;
}
```

環狀佇列 – 使用陣列

□ 重複利用空間

```
int dequeue(int data[], int index[]) {  
    //front = index[0];  back = index[1]; dequeue data = index[2]  
    if (isEmpty(index[0], index[1])) return FALSE;  
    index[0] = (index[0]+1)%SIZE;  
    index[2] = data[index[0]];  
    return TRUE;  
}  
int main() {  
    int k=0, i=0;  
    //front = index[0];  back = index[1]; dequeue data = index[2]  
    int index[3]={0, 0, 0};  
    int data[SIZE];  
    bool result;  
    for (i=0; i<5; i++) {  
        result=enqueue(data, index, k++);  
        if (!result) printf("Queue Full!\n");  
        else printf("enqueue %d\n", k-1);  
    }  
}
```

環狀佇列 – 使用陣列

□ 重複利用空間

```
result = dequeue(data, index);
if (!result) printf("Queue Empty!\n");
else printf("%d\n",index[2]);
```

```
result = dequeue(data, index);
if (!result) printf("Queue Empty!\n");
else printf("%d\n",index[2]);
```

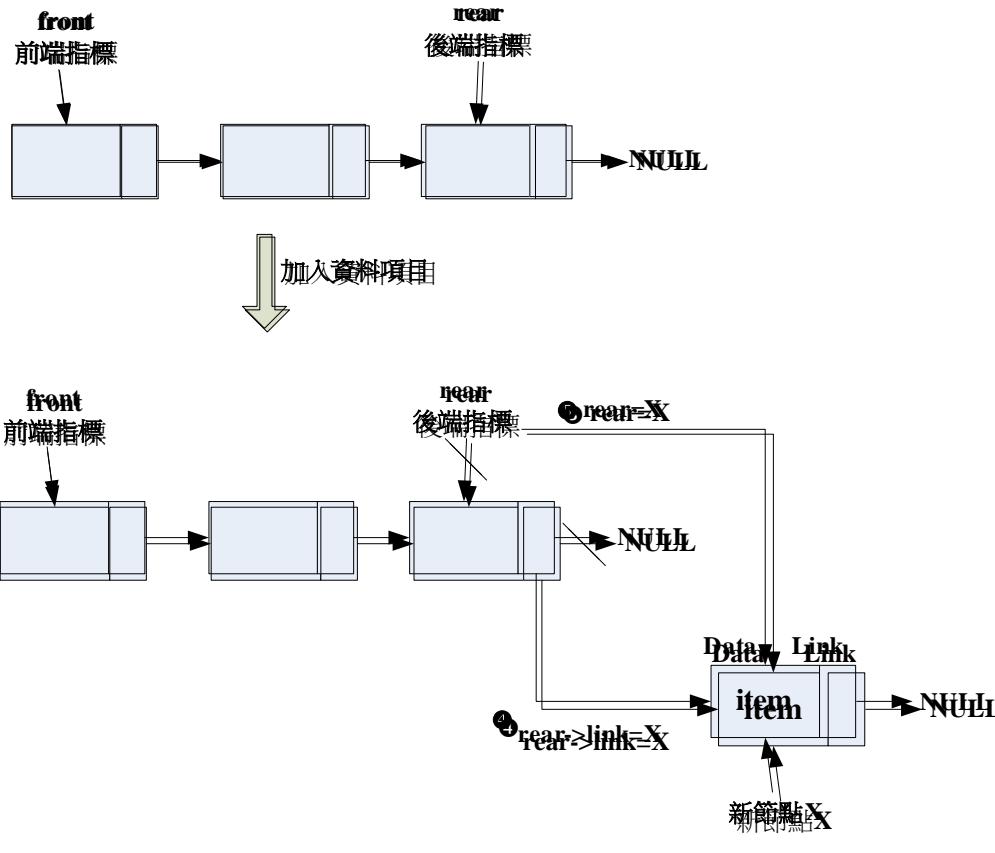
```
result = dequeue(data, index);
if (!result) printf("Queue Empty!\n");
else printf("%d\n",index[2]);
```

```
return 0;
```

```
}
```

鏈結佇列的資料加入

- 產生新資料項新節點X，加到鏈結佇列的尾端



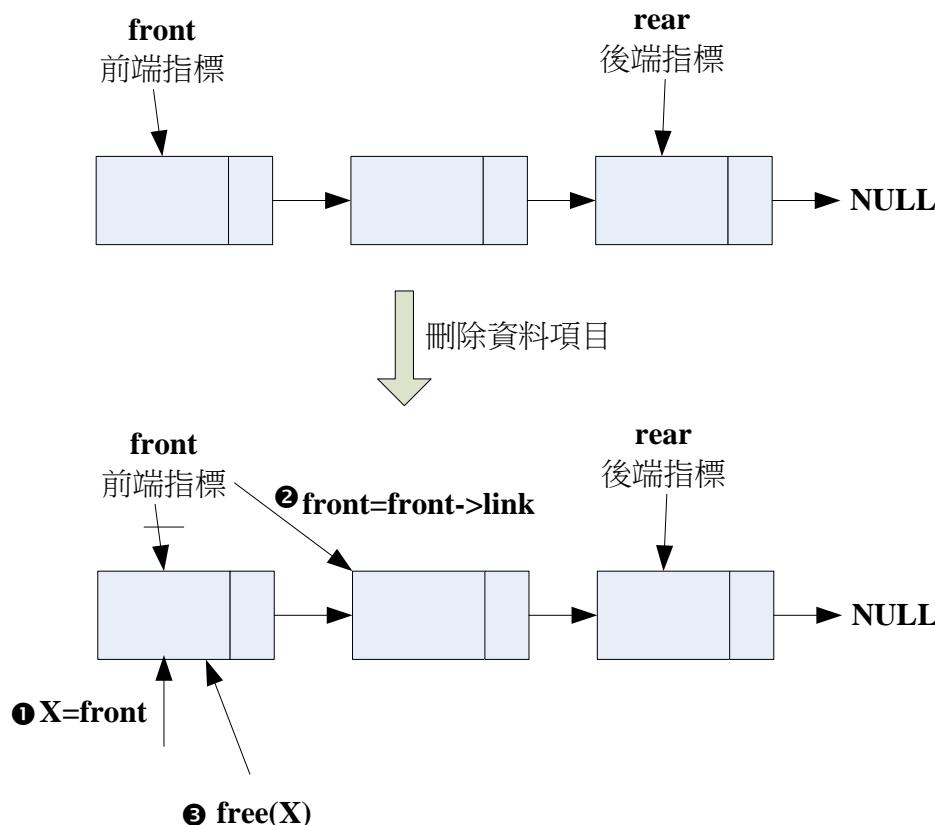
① $X = \text{newnode}()$
② $X->\text{data} = \text{item}$
③ $X->\text{link} = \text{NULL}$

Exercise

- 鏈結佇列的資料加入

鏈結佇列的資料刪除

- 判斷鏈結堆疊「是否為空」，若不為空，則改變前端指標front至下一個節點，歸還原來前端節點空間給系統



Exercise

- 鏈結佇列的資料刪除

反向列印串列

```
int main() {
    nodep_t ptr, head, tail;
    int num,i;
    tail = (nodep_t) malloc(sizeof(node_t));
    tail->next = NULL;
    ptr = tail;
    printf("\1: Please input 5 different data\n");
    for ( i = 0; i <= 4; i++ )  {
        scanf("%d",&num);
        ptr->data = num;
        head = (nodep_t) malloc(sizeof(node_t));
        head->next = ptr;
        ptr = head;
    }
    ptr = ptr->next; /* because final head does not have data */
    printf("\2: Reverse print the list\n");
    while ( ptr != NULL )  {
        printf("\2: The value is ==> %d\n",ptr->data);
        ptr = ptr->next;
    }
}
```

Homework I

- 使用 Link List，輸入兩個多項式，輸出相加、相減、相乘的結果。例如：
 - $2x^4 + 3x^3 + x - 1$
 - $x^5 - x^3 + 4x^2 - 3x + 2$
 - 結果：
 - $2x^9 + 3x^8 - 2x^7 + 6x^6 + 5x^5 - 6x^4 + 11x^3 - 7x^2 + 5x - 2$
 - 輸入說明
 - 輸入兩筆資料，分別代表兩個多項數。
 - 每一筆輸入 n 個整數，第一個代表 $n-1$ 次方的係數，第 n 個代表 0 次方係數。

Homework I

○ 輸出說明

- 兩個多項式相乘後從最高次方到0次方的係數，相乘結果為0的係數也需印出。

- Sample Input

– 2 3 0 1 -1

– 1 0 -1 4 -3 2

- Sample Output

– 2 3 -2 6 5 -6 11 -7 5 -2

Homework II

□ 使用 Link List 實作 stack

- 在一端進行後進先出（LIFO, Last In First Out）的原理運作。
- 兩種基本操作：push 和 pop
 - push：將數據放入堆疊的頂端（串列形式），堆疊頂端top指標加一。
 - pop：將頂端數據資料輸出（回傳），堆疊頂端top指標減一。
- 每一次push和pop的一筆資料都包含姓名、年齡、生日(年、月、日)
- 輸入說明：
 - 1 代表 push，再依序輸入姓名、年齡、生日(年、月、日)，參數之間以空白相隔
 - 2 代表 pop，再輸入一個數字，進行不同的操作，操作數字如下：

Homework II

- 1:印出該次pop的資料中的姓名
- 2:印出該次pop的資料中的年齡
- 3:印出該次pop的資料中的生日(年、月、日之間以底線連結)
- 若stack中為空則印出 The Stack is empty\n
- 3 結束程式。

Sample Input

```
1 "Marry Hu" 19 1989 7 16
1 "Tom Chen" 22 1996 10 19
2 1
1 "Billy Wu" 15 2005 3 18
2 3
2 2
1 "Lucas Su" 24 1993 5 21
2 3
2 1
3
```

Sample Output

```
Tom Chen
2005_3_18
19
1993_5_21
The Stack is empty
```

Homework III

- 使用 Link List 實作queue
 - 在一端進行先進先出（FIFO, First In First Out）的原理運作。

Homework IV

□ 建構唯一二元樹

- 紿定(1)中序，(2)前序或後序，產生唯一個Binary Tree，依序印出Tree的內容，印出順序，Tree元素由上而下，由左而右印出。
- 假設一二元樹如下圖
 - A
 - B C
 - D E F G
 - (A為根節點，B為A之左子樹，C為A之右子樹)
- 前序(Pre-order)：
 - 1.訪問根節點
 - 2.訪問左子樹
 - 3.訪問右子樹
 - 上圖的走訪順序為：ABDECFG

Homework IV

- 中序(In-order)：
 - 1. 訪問左子樹
 - 2. 訪問根節點
 - 3. 訪問右子樹
 - 上圖的走訪順序為：DBEAFCG
- 後序(Post-order)：
 - 1. 訪問左子樹
 - 2. 訪問右子數
 - 3. 訪問根節點
 - 上圖的走訪順序為：DEBFGCA
- 前序代碼：P
- 中序代碼：I
- 後序代碼：O

Homework III

- *每次輸入一定有一個是中序。(中序搭配前序，或是中序搭配後序，不會有前序搭配後序)。
- 輸入說明
 - 第一筆輸入前序、中序或後序代碼。
 - 第二筆輸入上一筆輸入種類尋訪的結果，大寫英文字母。
 - 第三筆輸入前序、中序或後序代碼。
 - 第四筆輸入上一筆輸入種類尋訪的結果，大寫英文字母。
- 輸出說明
 - 輸出唯一二元樹的內容，由上而下，由左而右。

Sample input

P
ABCDEFGHI
I
BCAEDGHFI

Sample output

ABDCEFGIH

計算機程式設計

C語言 tree

郭忠義

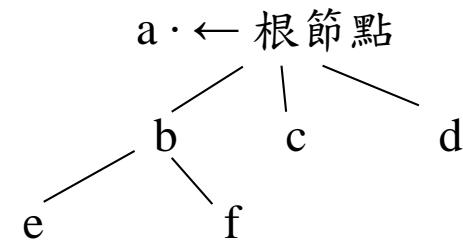
jykuo@ntut.edu.tw

臺北科技大學資訊工程系

二元樹

□ 樹是一種資料結構

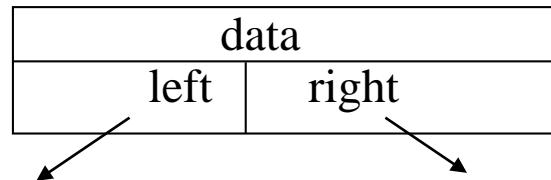
- 每一顆樹有一個根節點(root node)，
- 根節點下有零到n個子節點。
- 子節點也可以擁有自己的子節點。
- e和f是b的子節點，a是b的父節點。
- 一個節點最多有n個子節點，則稱n元樹。
- 某樹一個節點最多有二個子節點，則稱二元樹。
- 某節點沒有子節點，稱葉節點。
- 沒有父節點的節點，稱根節點。
- a是根節點，e，f，c...d是葉節點。



二元樹的節點結構

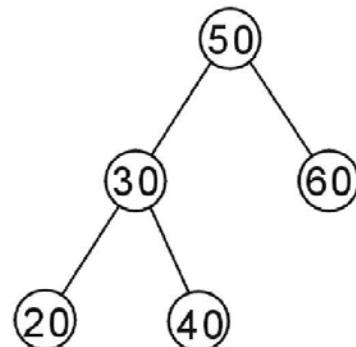
- data欄位存放二元樹節點的基本資料，right和left指向右子樹和左子樹的指標。

```
typedef struct node_s {  
    int data ;  
    struct node_s * right, * left;  
} tree_t;  
typedef tree_t * btree;
```

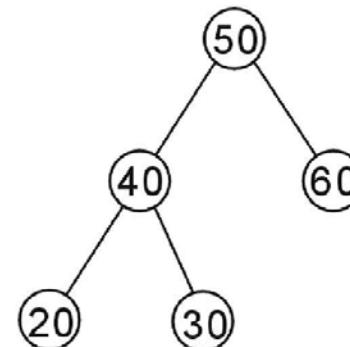


二元搜尋樹

- 二元搜尋樹可以是空集點，假使不是空集合，樹中每一節點（node）均含有一鍵值（key value），且具有下列特性：
 - 在左子樹的所有鍵值均小於樹根的鍵值。
 - 在右子樹的所有鍵值均大於樹根的鍵值。
 - 左子樹和右子樹亦是二元搜尋樹。
 - 每個鍵值都不一樣。



(a)

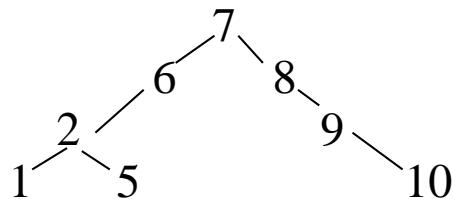


(b)

二元搜尋樹的建立

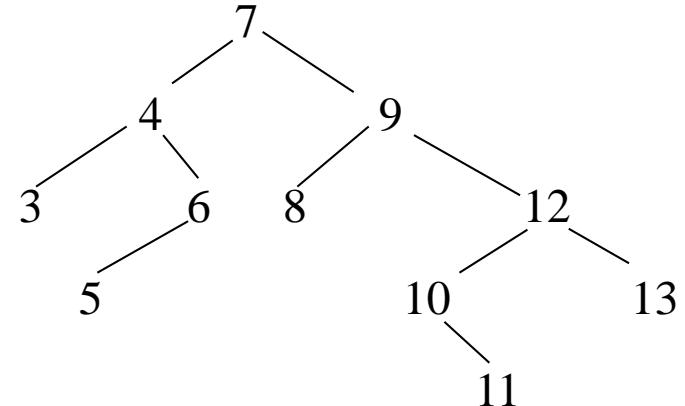
□ 二元搜尋樹建立

- 將第一個欲建的元素放在根節點
- 比較元素值與節點值，若元素值大於節點值，則將此元素值送往右子節點，若右邊子節點不是NULL則重複比較，否則建立一個新節點存放這筆資料，然後將新節點的右邊子節點和左邊子節點設成NULL。
- 若元素值小於節點值，將此元素值送往左子節點，若此左子節點不是NULL，則重複比較。否則建立一個新節點存放這筆資料，然後將新節點的右子節點，和左子節點設成NULL。
- 範例：依資料順序7，6，2，8，9，10，1，5建立樹結構。



二元樹的列印

- 線性串列只有從頭到尾或從尾到頭的列印
- 二元樹有三種不同列印方式：
 - 中序(inorder)列印方式。
 - 以從小到大的方式列印
 - 3 4 5 6 7 8 9 10 11 12 13
 - 前序 (preorder) 列印方式。
 - 後序 (postorder) 列印方式。



```
void inorder(btreetype root) {  
    if (root!=NULL) {  
        inorder(root->left);  
        printf("\2: %d\n", root->data);  
        inorder(root->right);  
    }  
}
```

二元樹的列印

```
btree create_btree(btree root, int val) {  
    btree newnode, current, back;  
    newnode = (btree) malloc(sizeof(node));  
    newnode->data = val;  
    newnode->left = newnode->right = NULL;  
    if ( root == NULL ) { /* insert root node */  
        root = newnode;      return root;  
    }  
    else {                /* insert other node */  
        current = root;  
        while ( current != NULL ) {  
            back = current;  
            if ( current->data > val )  current = current->left;  
            else                  current = current->right;  
        }  
        if ( back->data > val )  back->left = newnode;  
        else                  back->right = newnode;  
    }  
    return root;  
}
```

二元樹的列印

```
int main() {  
    int arr[] = { 7, 4, 1, 5, 12, 8, 13, 11 };  
    btree ptr;  
    int val, i;  
    ptr = NULL; /* initial the root pointer */  
    printf("\1: Create binary tree for following value.\n");  
    for ( i = 0; i < 8; i++ ) {  
        ptr = create_btree(ptr,arr[i]);  
        printf("\2: %d\n",arr[i]);  
    }  
    printf("\1: Using inorder to print the binary tree.\n");  
    inorder(ptr);  
}  
  
@: Create binary tree for following value.  
@: 7  
@: 4  
@: 1  
@: 5  
@: 12  
@: 8  
@: 13  
@: 11  
@: Using inorder to print the binary tree.  
@: 1  
@: 4  
@: 5  
@: 7  
@: 8  
@: 11  
@: 12  
@: 13  
Press any key to continue
```

二元樹的列印

□ 前序列印

- 每個節點皆會比它左子節點及右子節點先列印，左子節點又比右子節點有更高優先順序。
- 若以前序列印方式，得到結果：7，4，3，6，5，9，8，12，10，11，13

□ 後序(postorder)列印

- 在列印某節點前，先列印左節點，然後右節點，等到兩個子節點列印完後，才列印此節點。
- 列印此資料結構，得到結果：3，5，6，4，8，11，10，13，12，9，7

```
void postorder(btree root) {  
    if (root!=NULL) {  
        postorder(root->left);  
        postorder(root->right);  
        printf("\2: %d\n",root->data);  
    }  
}
```

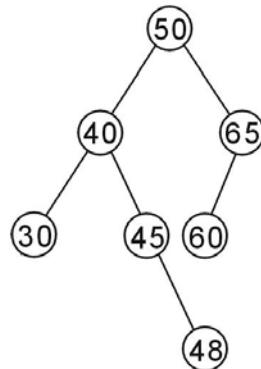
二元樹的列印

```
void preorder(btreet root) {  
    if ( root != NULL ) {  
        printf("\2: %d\n",root->data);  
        preorder(root->left);  
        preorder(root->right);  
    }  
}  
  
int main() {  
    int arr[] = { 7, 4, 1, 5, 12, 8, 13, 11 };  
    btreet ptr;  
    int val, i;  
    ptr = NULL; /* initial the root pointer */  
    printf("\1: Create binary tree for following value.\n");  
    for ( i = 0; i < 8; i++ ) {  
        ptr = create_btreet(ptr,arr[i]);  
        printf("\2: %d\n",arr[i]);  
    }  
    printf("\1: Using preorder to print the binary tree.\n");  
    preorder(ptr);  
}
```

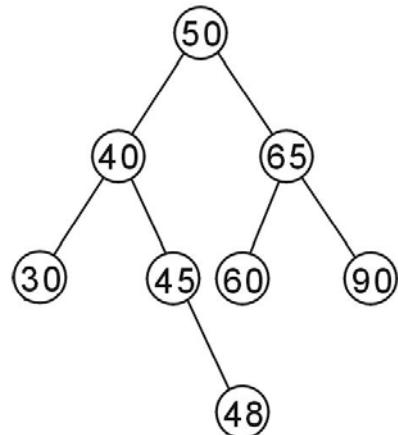
```
①: Create binary tree for following value.  
①: 7  
①: 4  
①: 1  
①: 5  
①: 12  
①: 8  
①: 13  
①: 11  
①: Using preorder to print the binary tree.  
①: 7  
①: 4  
①: 1  
①: 5  
①: 12  
①: 8  
①: 11  
①: 13  
Press any key to continue
```

二元搜尋樹加入與刪除

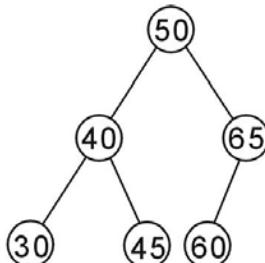
1. 今欲加入48



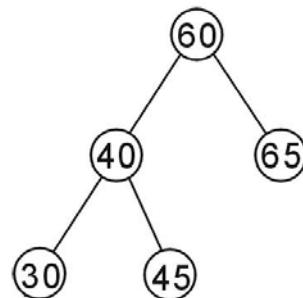
2. 繼續加入90則



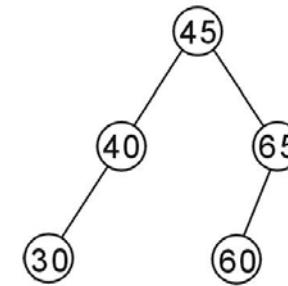
3. 刪除某一節點時，若刪除的是樹葉節點，
則直接刪除之，假若刪除不是樹葉節點，
則在左子樹找一最大的節點或在右子樹找
一最小的節點，取代將被刪除的節點



4. 刪除50，則可用下列二種方法之一：



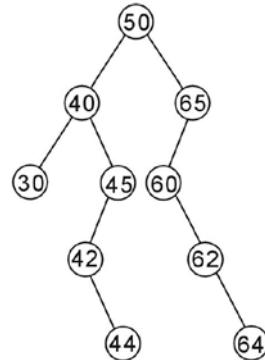
(以右子樹最小的節點取代)



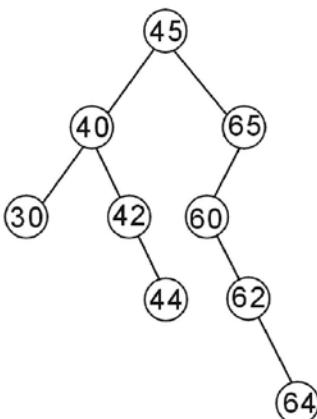
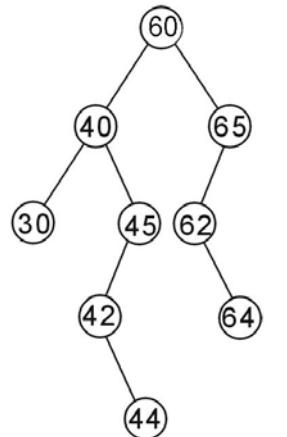
(以左子樹最大的節點取代)

二元搜尋樹加入與刪除

5. 若取代的節點有右子樹或左子樹時，則必須加以調整其子節點

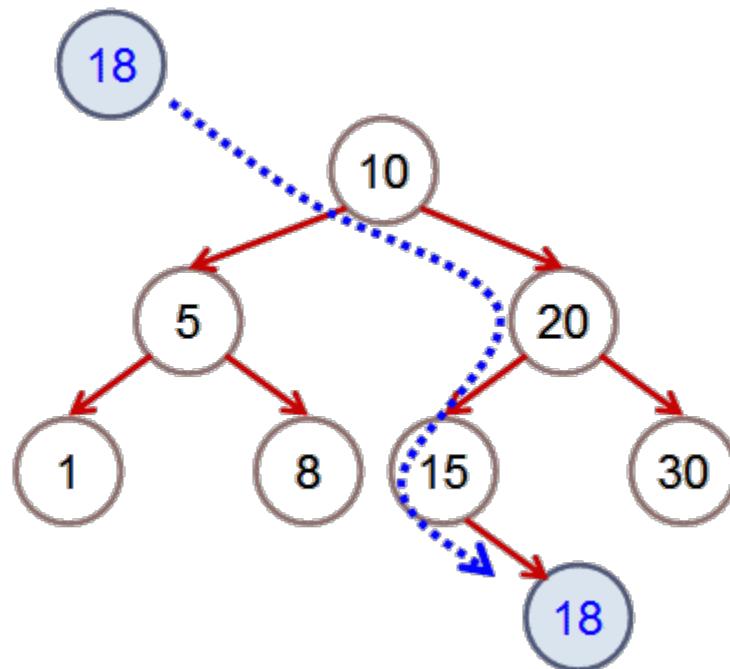


- 今欲刪除50的兩種做法



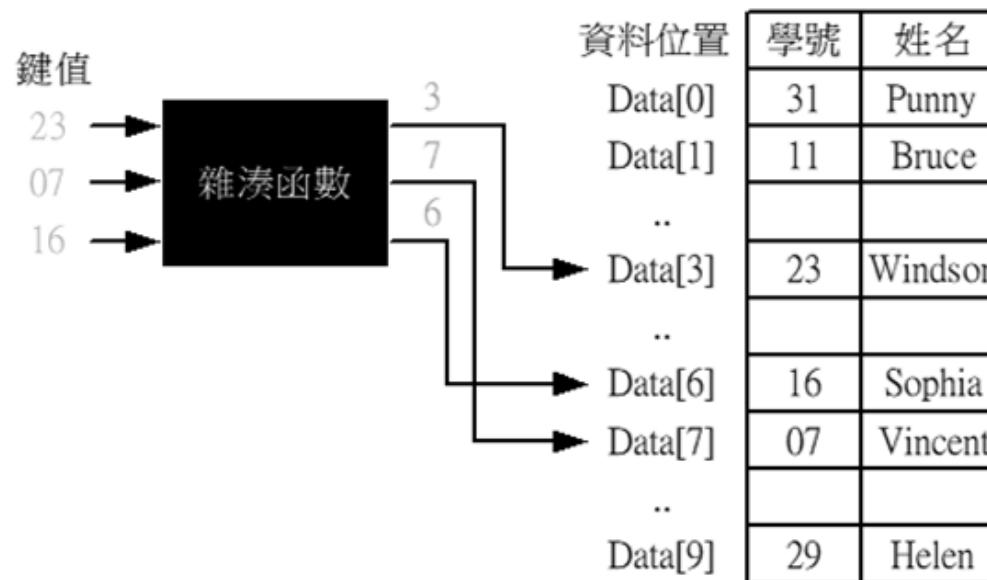
二元搜尋(Binary Search)

- 適用於已排序完成資料之搜尋
- 如欲搜尋18，藍色虛線是搜尋路徑



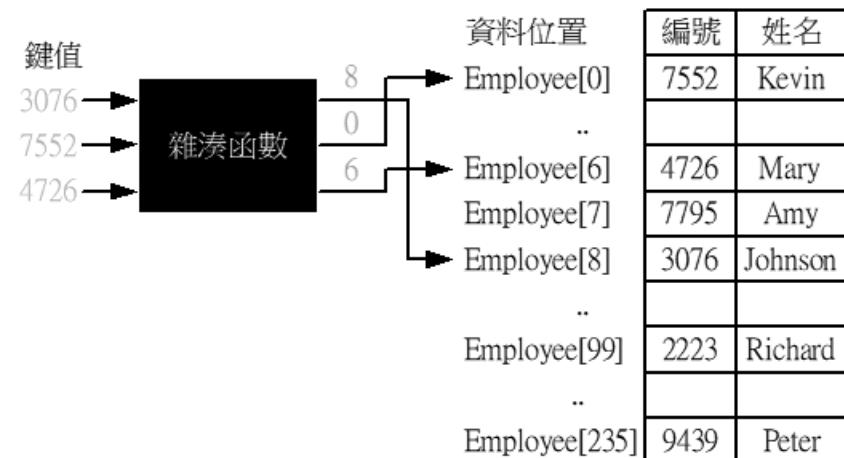
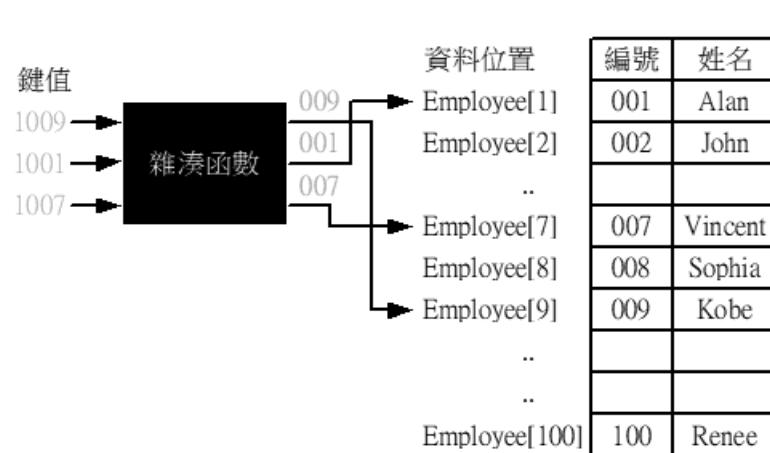
雜湊搜尋(Hash Search)

- 雜湊搜尋可將資料比較次數減少到每次搜尋只須一次
- 雜湊表中資料儲存位置是透過特定雜湊函數運算而來
- 在雜湊結構中，稱輸入資料的值為鍵值(Key)
- 例如，利用雜湊表儲存學生資料，其中學生的學號即為雜湊表的鍵值，示意圖表如下



雜湊搜尋的減去與餘數法

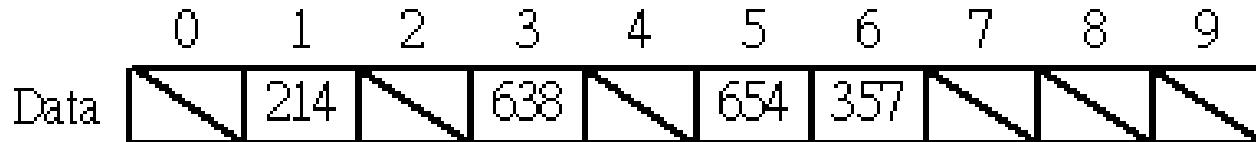
- 減去法(Subtraction Method)
 - 資料鍵值減去一個特定的數值，以求得資料儲存的位置
- 餘數法(Modulo-Division Method)
 - 將資料的鍵值除以陣列大小後，取其餘數做為資料儲存的位置



雜湊碰撞解決法

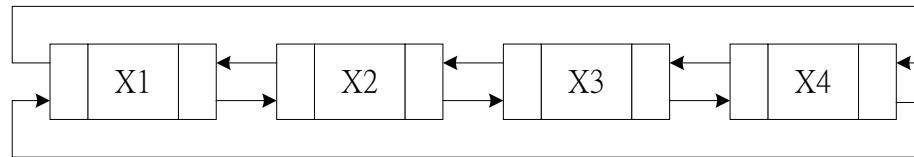
□ 線性探索法(Linear Probe)

- 往下一筆資料位置尋找可用空間儲存資料
- 假設現有資料如下：654、638、214、357，採用數值抽出法，取出第二位，並將資料放入大小為10的雜湊表中
- 插入資料654，位置為5、插入資料638，位置為3、插入資料214，位置為1
- 插入資料357，位置為5，發生碰撞忘下一筆資料儲存空間尋找可用空間，位置為6



雙向鏈結串列(Double nodep_ted List)

- 資料欄，儲存節點資料，
- 兩個指標，一個指向前一個節點，另一個指向下一個節點。
- 單向鏈結串列，在搜尋串列時，只能沿一個方向搜尋，無法往回搜尋。雙向鏈結串列可克服此問題。



```
typedef struct dnode_s {  
    int data;  
    struct dnode_s * front;  
    struct dnode_s * back;  
} node_t;  
  
typedef node_t * nodep_t;
```

雙向鏈結串列

```
#include <stdlib.h>
#include <stdio.h>
typedef struct dnode_s {
    int data;
    struct dnode_s * front;
    struct dnode_s * back;
} node_t;
typedef node_t * nodep_t;

void reversePrint(nodep_t ptr) {
    while (ptr) {
        printf("=>%d\n",ptr->data);
        ptr = ptr->back;
    }
}
```

雙向鏈結串列

```
void insert(nodep_t* tailp, nodep_t* headp, int num) {  
    nodep_t ptr;  
    ptr = (nodep_t) malloc(sizeof(node_t));  
    ptr->data = num;  
    ptr->front = ptr->back=NULL;  
    if ((*headp)==NULL) {  
        (*tailp) = (*headp) = ptr;  
        return;  
    }  
    (*headp)->front = ptr;  
    ptr->back=(*headp);  
    (*headp)=(*headp)->front;  
}  
void print(nodep_t ptr) {  
    while (ptr) {  
        printf("=> %d\n",ptr->data);  
        ptr = ptr->front;  
    }  
}
```

雙向鏈結串列

```
int main() {  
    nodep_t ptr=NULL, tail=NULL, head=NULL;  
    int num;  
    char c;  
    while (1) {  
        scanf("%c",&c);  
        if (c=='e') break;  
        else if (c=='i') {  
            scanf("%d",&num);  
            insert(&tail, &head, num);  
        }  
        else if (c=='p') {      print(tail);      }  
        else if (c=='r') {      reversePrint(head);      }  
    }  
    return 0;  
}
```