



# C/C++基礎程式設計

流程控制敘述

講師：張傑帆

CSIE, NTU

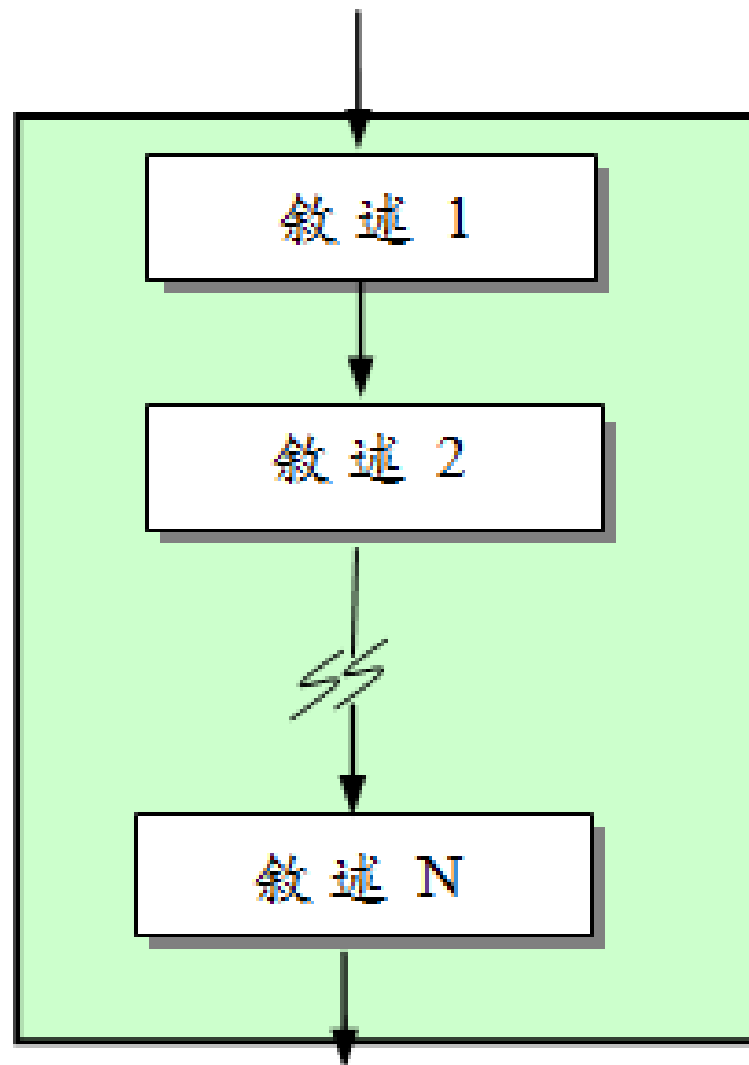
追隨我的好奇與直覺，大部分我所投入過的事務，後來都成了無比珍貴的經歷。

*Much of what I stumbled into by following my curiosity and intuition turned out to be priceless later on. -Steve Jobs*

# 課程大綱

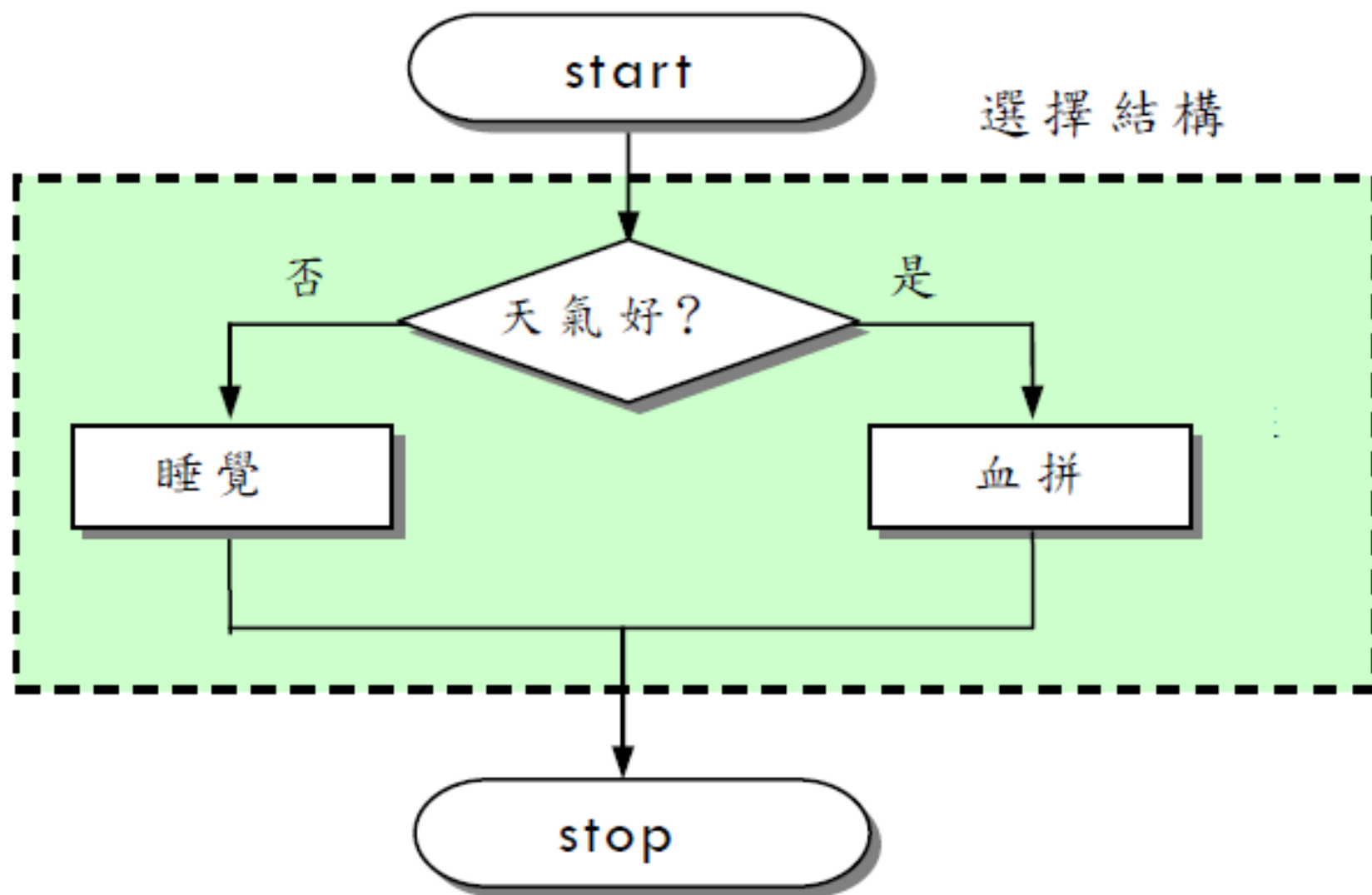
- 選擇控制 (流程控制)
  - IF-ELSE
  - SWITCH-CASE
- 重覆控制 (迴圈)
  - FOR
  - WHILE
  - 迴圈中的流程控制: break, continue
  - 巢狀迴圈
- 作業

# 選擇結構簡介



循序結構

# 人會做判斷



# 選擇敘述 if-else敘述

- 程式中的選擇結構有如口語中的  
「如果.....就.....否則.....」  
在C語言中是使用if-else敘述來達成。
- 如語法

若 條件 成立時

則執行接在if後面的 敘述區段1。

否則(條件不成立)

執行接在else後面的 敘述區段2。

# IF-ELSE

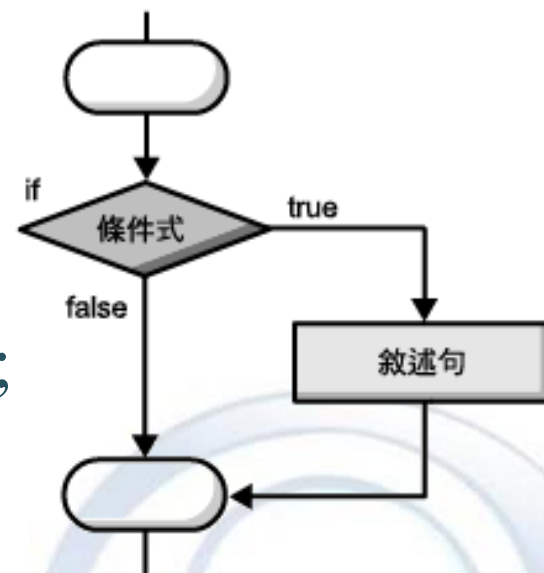
- 用途：程式執行時根據條件情況選擇要執行的程式碼。

- 語法：

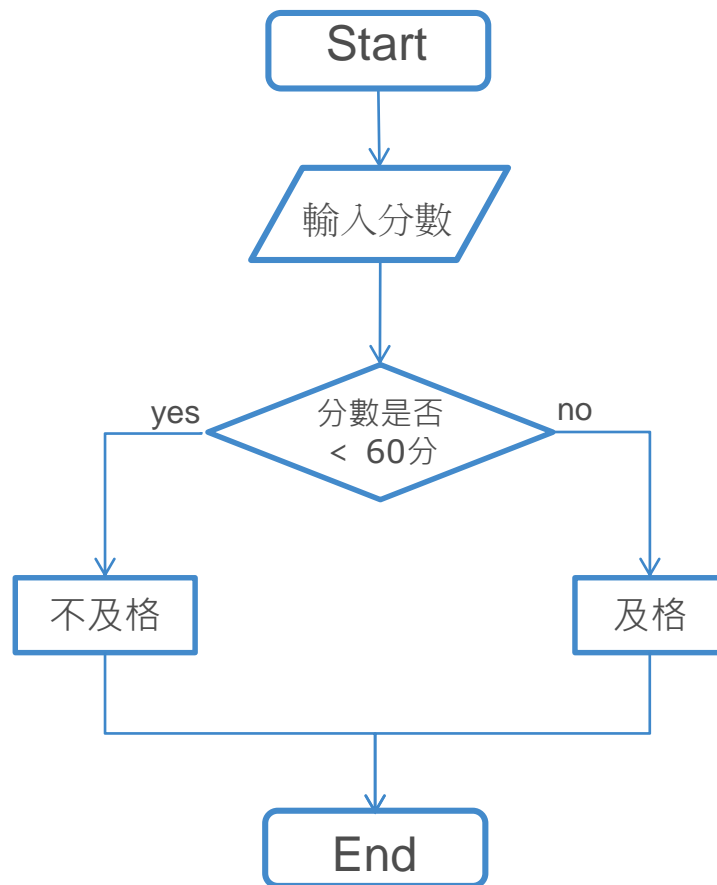
- **if**( 條件判斷式 ) {  
    條件成立的話要做的程式碼;  
}  
**else** {  
    條件不成立的話要做的程式碼;  
}

註：*else* 可視情況省略不寫;

當{ } 中程式只有一行(敘述), 可省略{ }



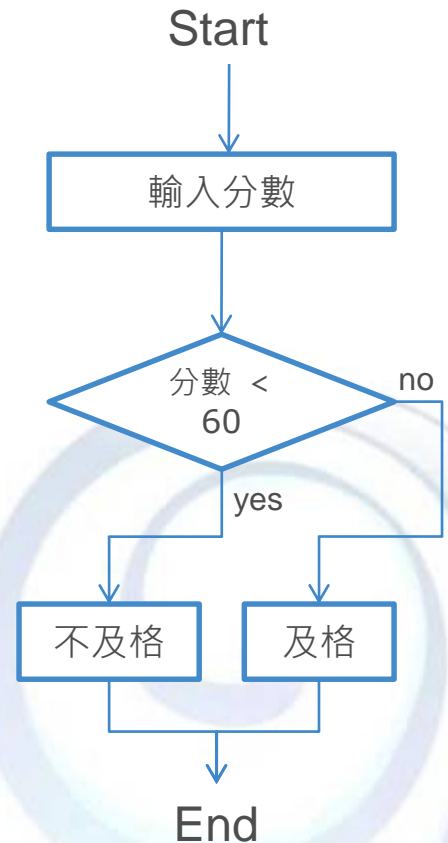
# 想判斷分數是否及格



# IF-ELSE

- 範例：寫個程式，判斷一個人的成績是否及格（及格分數為60分）

```
#include <stdio.h>
int main()
{
    int score;
    printf("輸入您的分數:");
    scanf("%d",&score);
    if ( score < 60 ) {
        printf("不及格\n");
    }
    else {
        printf("及格\n");
    }
    return 0;
}
```





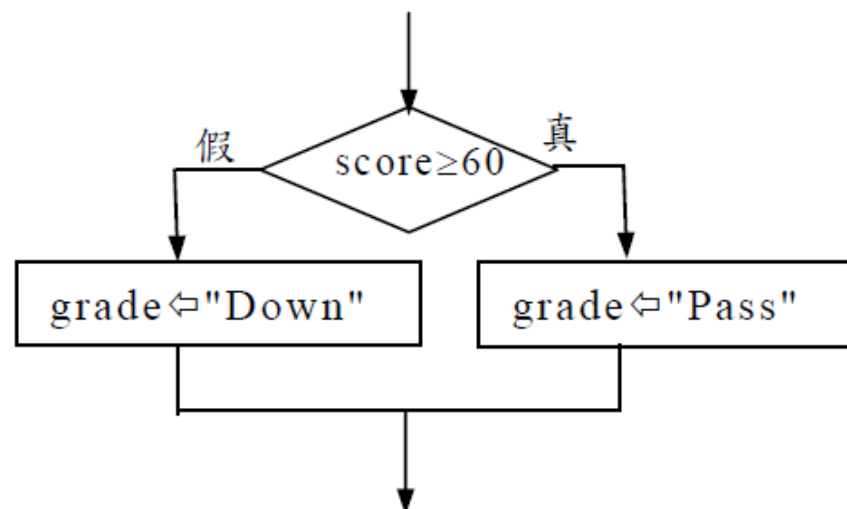
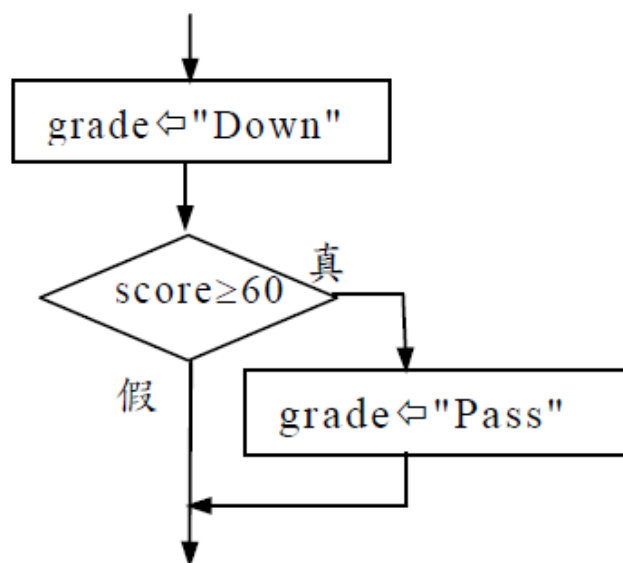
3. 上面語法中的 [...] 中括號內的敘述是當不滿足條件且不執行任何敘述時，此部份可省略。
4. 譬如：由分數 score 來判斷是否 Pass(及格)或 Down(不及格)？若  $\text{score} \geq 60$  顯示 "Pass"；如果  $\text{score} < 60$  顯示 "Down"，有下列兩種撰寫方式：

使用單一選擇(省略 else 敘述)

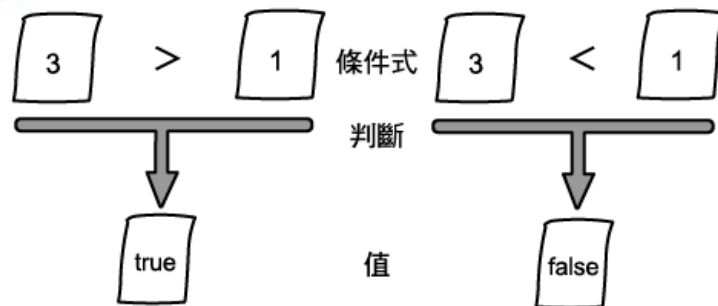
```
grade = "Down" ;  
if (score >= 60)  
    grade = "Pass" ;
```

使用 if...else 敘述：

```
if (score >= 60)  
    grade = "Pass" ;  
else  
    grade = "Down";
```

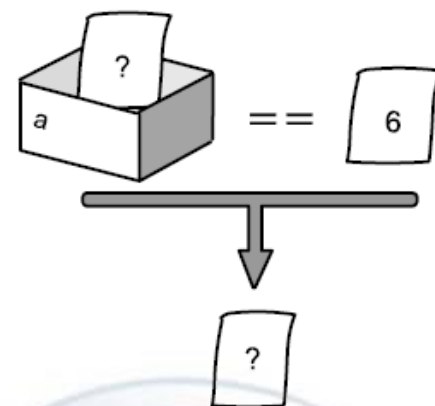


# IF-ELSE



- 邏輯判斷可以使用的運算符號如下

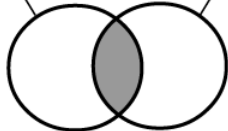
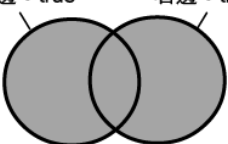

| 運算符號 | 意義                    |
|------|-----------------------|
| >    | 大於                    |
| <    | 小於                    |
| >=   | 大於或等於                 |
| <=   | 小於或等於                 |
| ==   | 等於 ※ 和「=」(指定運算子) 不同 ! |
| !=   | 不等於                   |



- 用來連結邏輯判斷的符號有

| 符號 | 意義                                  |
|----|-------------------------------------|
| && | " <b>而且</b> ", 所有的條件都要成立, 整個判斷才會成立  |
|    | " <b>或</b> ", 只要有任何一個條件成立, 整個判斷就會成立 |
| !  | " <b>非</b> ", 條件不成立時, 整個判斷就會成立      |

| 運算子          | 說明                          | 使用例          | 結果   |
|--------------|-----------------------------|--------------|------|
| ==<br>(相等)   | 判斷此運算子左右兩邊運算式的值是否相等。        | 18 == 18     | 1(真) |
|              |                             | 18 == 35     | 0(假) |
|              |                             | 3+2 == 1+4   | 1(真) |
| !=<br>(不相等)  | 判斷此運算子左右兩邊運算式的值是否不相等。       | 17 != 18     | 1(真) |
|              |                             | 56 != 56     | 0(假) |
|              |                             | 12*3 != 3*12 | 0(假) |
| <<br>(小於)    | 判斷此運算子左邊運算式的值是否小於右邊運算式的值。   | 17 < 18      | 1(真) |
|              |                             | 42 < 30      | 0(假) |
|              |                             | 2 < 10-7     | 1(真) |
| ><br>(大於)    | 判斷此運算子左邊運算式的值是否大於右邊運算式的值。   | 19 > 18      | 1(真) |
|              |                             | 26 > 36      | 0(假) |
|              |                             | 12*3 > 12*2  | 1(真) |
| <=<br>(小於等於) | 判斷此運算子左邊運算式的值是否小於等於右邊運算式的值。 | 17 <= 18     | 1(真) |
|              |                             | 18 <= 18     | 1(真) |
|              |                             | 10+3 <= 12   | 0(假) |
| >=<br>(大於等於) | 判斷此運算子左邊運算式的值是否大於等於右邊運算式的值。 | 17 >= 18     | 0(假) |
|              |                             | 18 >= 18     | 1(真) |
|              |                             | 12*3 >= 35   | 1(真) |

| 邏輯運算子                                     | 說明  | 真值表   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|-------|-------|----|---|---|---|---|---|---|---|---|---|---|---|---|
| <div>&amp;&amp;</div> <div>(AND, 且)</div> | <div>此運算子左右兩邊的運算式結果若不為零值，結果為1(真)；否則為零值(假)。</div> <div><div>左邊：true</div><div>右邊：true</div></div>                              | <table><tr><th>運算式 1</th><th>運算式 2</th><th>結果</th></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table> | 運算式 1 | 運算式 2 | 結果 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 運算式 1                                     | 運算式 2   | 結果  |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 1   | 1   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 0   | 0   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 1   | 0   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 0   | 0   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| <div>  </div> <div>(OR, 或)</div>          | <div>此運算子左右兩邊的運算式結果只要其中有一個不為零值，結果就是1；兩個都為零結果才是零值</div> <div><div>左邊：true</div><div>右邊：true</div></div>                       | <table><tr><th>運算式 1</th><th>運算式 2</th><th>結果</th></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table> | 運算式 1 | 運算式 2 | 結果 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 運算式 1                                     | 運算式 2   | 結果  |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 1   | 1   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 0   | 1   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 1   | 1   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 0   | 0   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| <div>!</div> <div>(NOT)</div>             | <div>此運算子是單一的運算，主要是將敘述結果相反，即 <math>1 \Rightarrow 0</math>，<math>0 \Rightarrow 1</math>。</div> <div><div>右邊：true</div></div> | <table><tr><th>運算式</th><th>結果</th></tr><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table>   | 運算式   | 結果    | 1  | 0 | 0 | 1 |   |   |   |   |   |   |   |   |   |
| 運算式                                       | 結果  |   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 1   | 0   |   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |
| 0   | 1   |   |       |       |    |   |   |   |   |   |   |   |   |   |   |   |   |

# 邏輯運算子使用範例

- $5 > 3 \ \&\& \ 3 == 4$

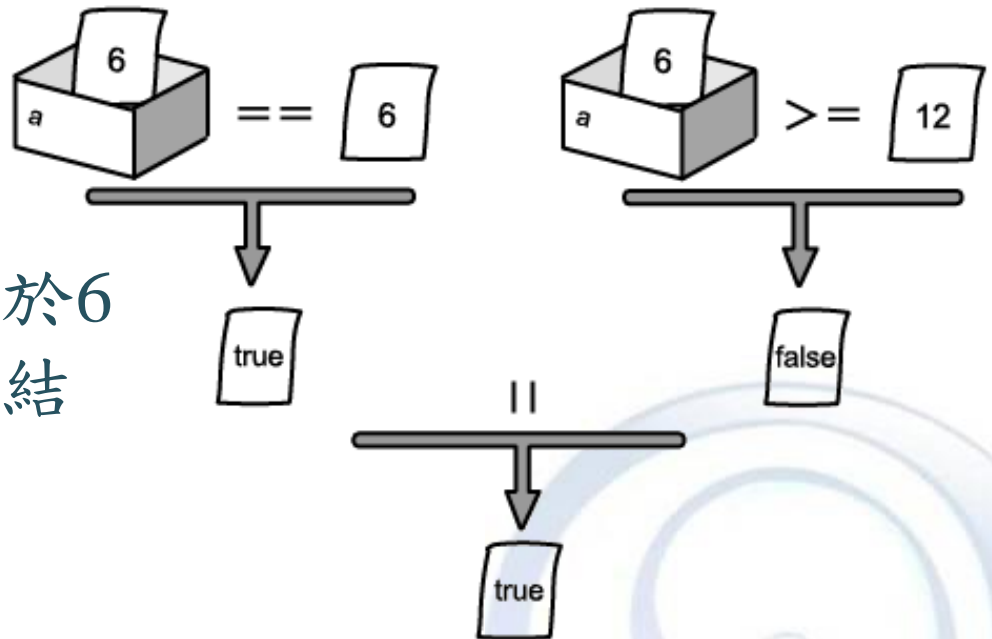
- (結果為false)

- $a == 6 \ || \ a \geq 12$

- (如果變數a的值等於6或是大於等於12，結果就會是true)

- $!(a == 6)$

- (當變數a等於6以外的其他值時，結果就會是true)

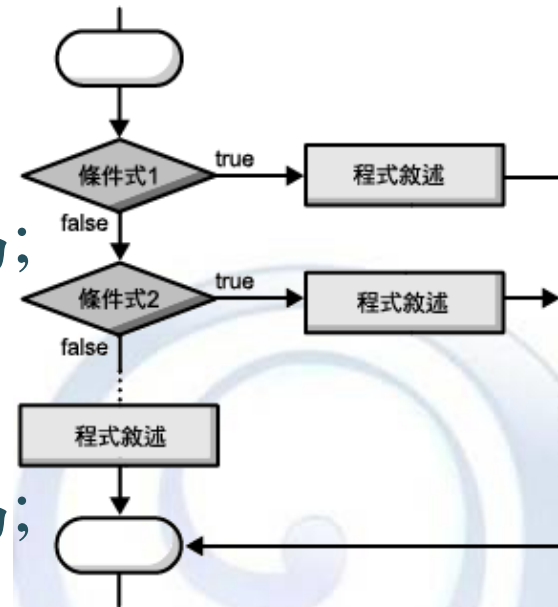


# IF-ELSE IF

- 若我們要判斷的條件不只是做“對”或“錯”的二分法時，可以在if後增加else if來做多重判斷。

- 語法：

- **if( 條件判斷式1 )**{  
    條件1成立的話要做的程式碼;  
}  
**else if ( 條件判斷式2 )** {  
    條件2成立的話要做的程式碼;  
}  
**else**{  
    以上條件都不成立的話要做的程式碼;  
}



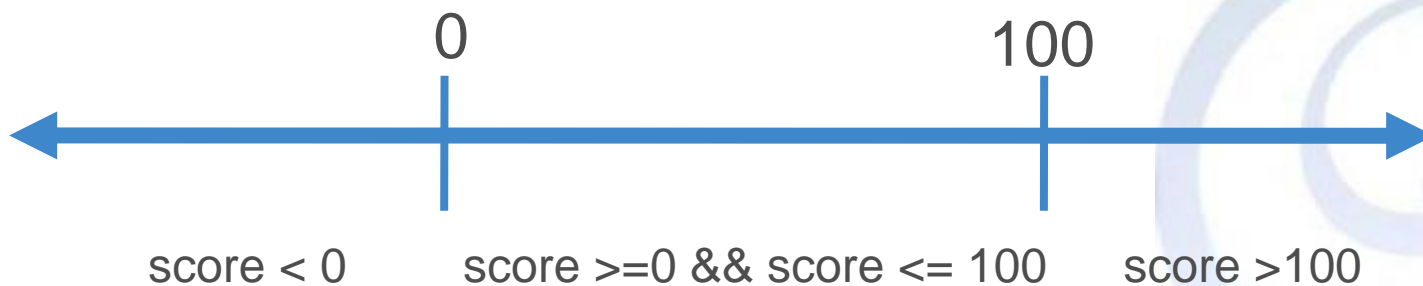
# IF-ELSE IF

- 範例：寫個程式，判斷一個人的成績是否及格  
(大學部及格分數為60分，研究所為70分)

```
#include <stdio.h>
int main()
{
    int score, n;
    printf("(1)大學部 (2)研究所:");
    scanf("%d",&n);
    printf("輸入您的分數:");
    scanf("%d",&score);
    if ( score < 60 && n == 1) {
        printf("不及格\n");
    }
    else if ( score < 70 && n == 2) {
        printf("不及格\n");
    }
    else {
        printf("及格\n");
    }
    return 0;
}
```

## 小練習

- 修改上頁程式, 若使用者輸入值不合下列格式則不做任何輸出, 並提示使用者輸入分數錯誤
  - 只能輸入 1 或 2 代表大學部或研究所
  - 成績只能輸入 0~100





## If else 大刮號{ }的恩怨情仇

- if else 用大刮號{ }表示其程式碼範圍
- 當{ }中程式只有一行(敘述), 可省略{ }
- if( 條件判斷式 )  
    裡面只可以寫一行
- 行但請務必記得改成多行時要加上{ }不然....
- 就會變成爆肝工程師！

```
int date = 0;
printf("Please enter date(1-7):");
scanf("%d", &date);
if(date==6 || date==7)
    printf("Yes, holiday");
else
    printf("work day");
    printf(", go to work.");
```

# 課程大綱

- 選擇控制 (流程控制)
  - IF-ELSE
  - SWITCH-CASE
- 重覆控制 (迴圈)
  - FOR
  - WHILE
  - 迴圈中的流程控制: break, continue
  - 巢狀迴圈
- 作業

# SWITCH-CASE

- 用途：程式執行時根據資料數值選擇要執行的程式。
- 語法：(註：default: 可有可無)

- switch( 變數或運算式 )  
{

- case 值1:

- 程式碼;

- break; //可不加, 結果不同!

- case 值2:

- 程式碼;

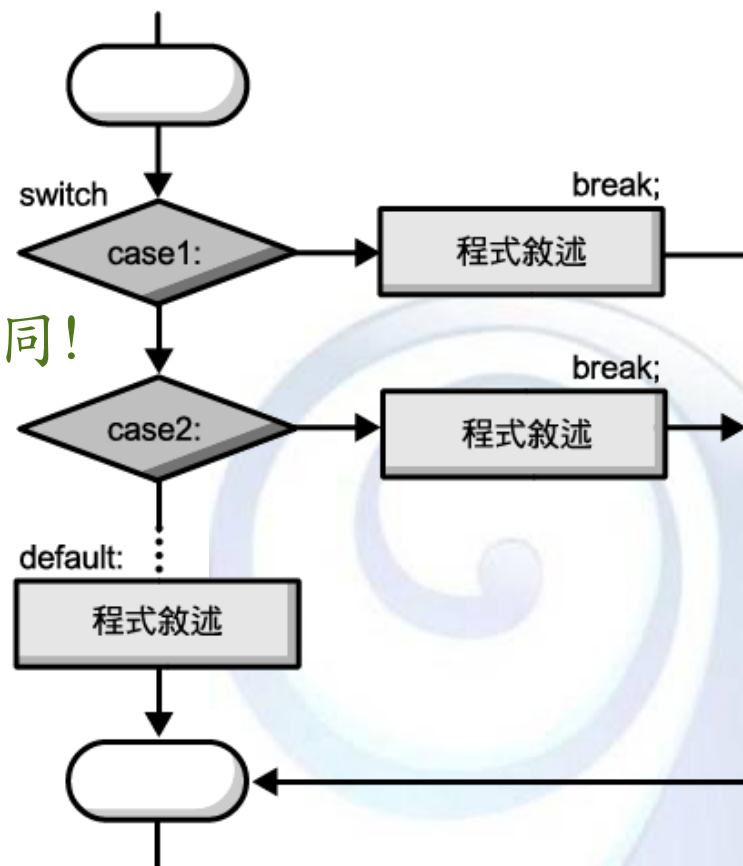
- break;

- default:

- 程式碼;

- break;

- }



```
switch (exp)
```

```
{
```

```
    case 常數 1 :
```

```
        ⋮
```

```
        /* 敘述區段 1 */
```

```
        break ;
```

```
    case 常數 2 :
```

```
        ⋮
```

```
        /* 敘述區段 2 */
```

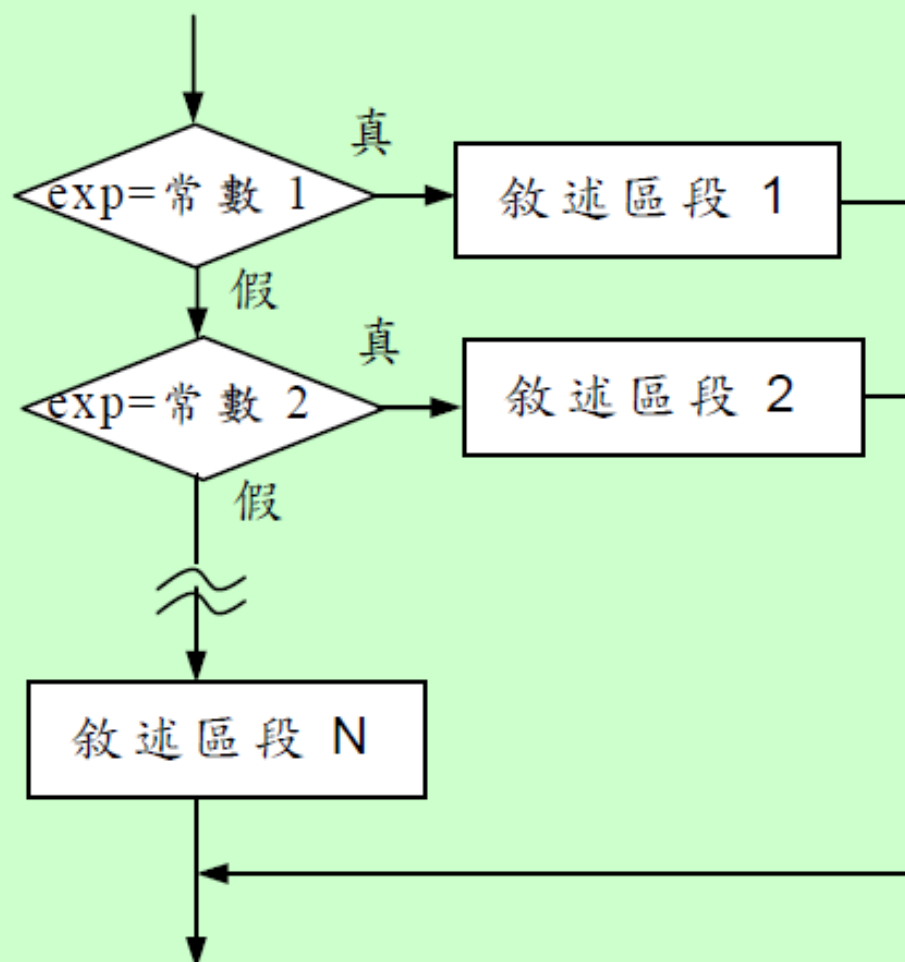
```
        break ;
```

```
        ⋮
```

```
    [default :]
```

```
        /* 敘述區段 N */
```

```
}
```



# SWITCH-CASE

- 範例, 輸入兩個數字, 再輸入+, -, \*, /任一鍵, 根據輸入的鍵內容顯示兩數計算結果

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

```
int main()
{
    double a, b, ans;
    char key;

    printf("input two number:");
    scanf("%lf %lf", &a, &b);
    printf("press +, -, *, / : ");
    //key = getch();
    scanf(" %c", &key);
```

```
    switch( key ) {
        case '+':
            ans = a + b;
            break;
        case '-':
            ans = a - b;
            break;
        case '*':
            ans = a * b;
            break;
        case '/':
            ans = a / b;
            break;
        default:
            printf("Undefined key\n");
            return 0;
    }
    printf( "%lf %c %lf = %lf \n", a, key, b, ans );
    return 0;
}
```

# 課程大綱

- 選擇控制 (流程控制)
  - IF-ELSE
  - SWITCH-CASE
- 重覆控制 (迴圈)
  - FOR
  - WHILE
  - 迴圈中的流程控制: break, continue
  - 巢狀迴圈
- 作業

# 印出10行hello

- ?

```
printf("hello %d\n",i);  
printf("hello %d\n",i);  
printf("hello %d\n",i);  
printf("hello %d\n",i);  
printf("hello %d\n",i);  
printf("hello %d\n",i);  
printf("hello %d\n",i);  
printf("hello %d\n",i);  
printf("hello %d\n",i);  
printf("hello %d\n",i);
```

- 那100行呢？

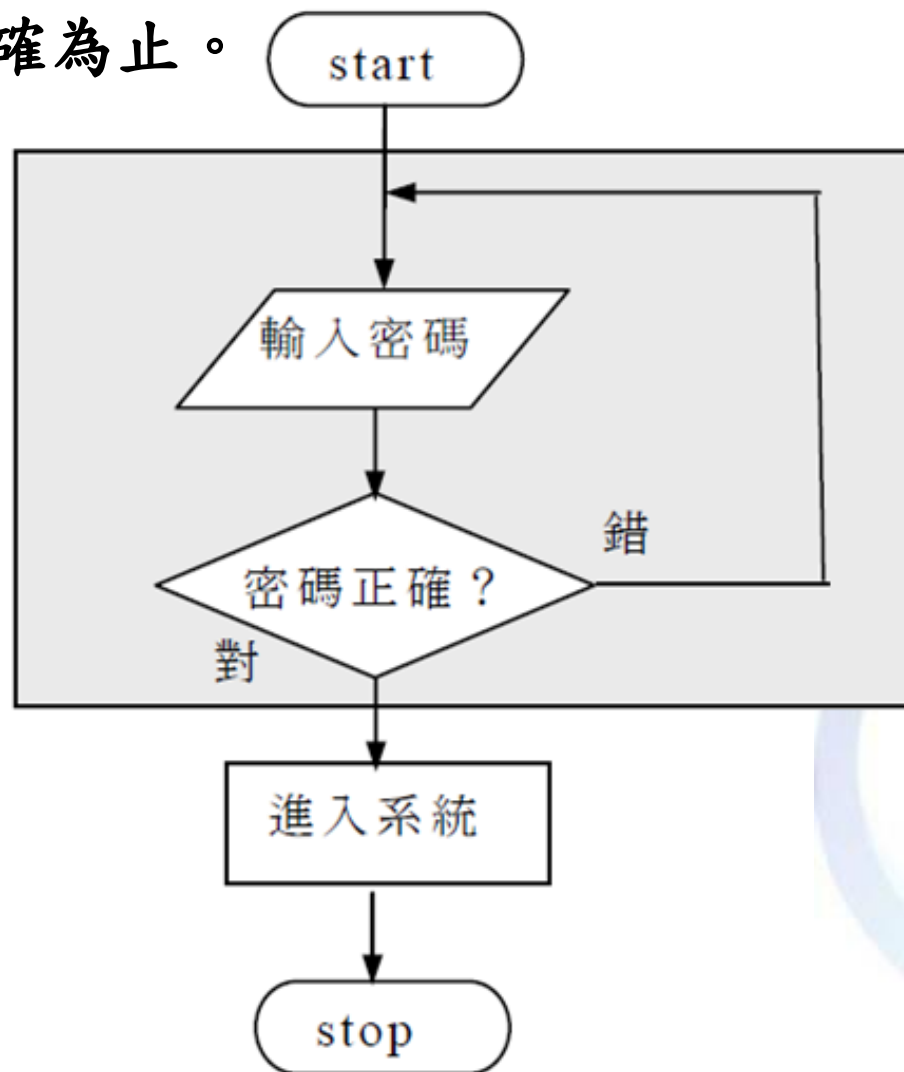


# 重複結構簡介

- C語言允許將需要連續重複執行的敘述區段改用「迴圈敘述」，不但可縮短程式的長度，而且程式易維護及增加程式可讀性。
- 此種程式架構稱為「重複結構」或「迴圈」(Loop)。



下圖是密碼檢查的流程圖，若密碼正確，則進入系統繼續往下執行；反之，密碼輸入錯誤，重新輸入密碼，一直詢問到輸入的密碼正確為止。



# for迴圈敘述

- C語言提供三種迴圈敘述：**for**、**while**、**do-while**敘述
- 若迴圈的次數可以預知，**for**敘述是最好的選擇
- 若迴圈次數無法確定，則可使用**while**、**do-while**敘述來達成

# FOR迴圈

- 用途：當程式需要來回重複執行某一段程式碼時
  - for通常用在已知重覆執行次數時。

- 語法：

```
for ( 進入迴圈前要做的事; 繼續執行迴圈的條件; 每跑完一次迴圈會做的事 )  
{  
    要被重覆執行的程式碼;  
}
```

- for迴圈會重複執行數次{ }中所包含的程式碼，  
至於是執行幾次則由for(;;)當中的東西來決定。

# for迴圈

## 語法

```
for (初始運算式; 條件運算式; 控制運算式)
{
    [敘述區段]
}
```

for( i=1; i<=3; i++ )

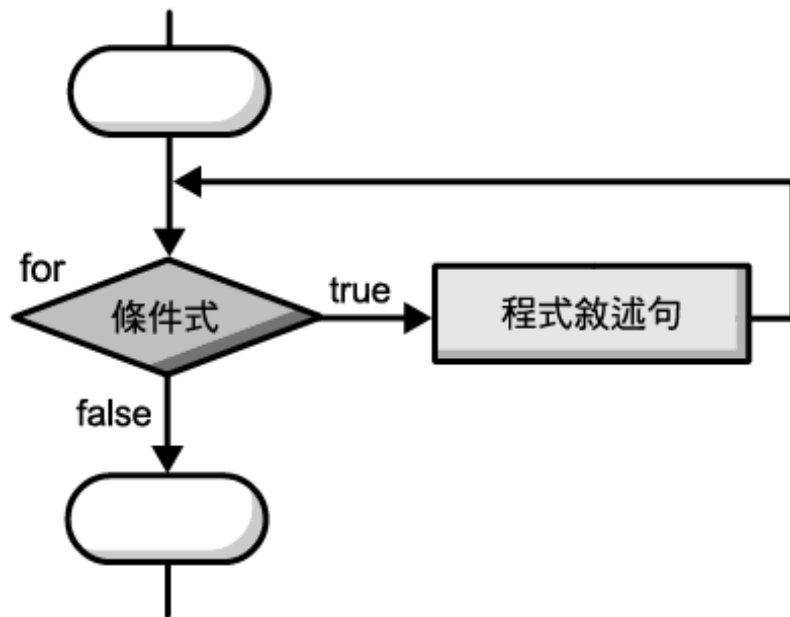
## 說明

### 1. 初始運算式

用來設定迴圈控制變數的初始值。這個運算式只有在第一次進入迴圈時才會執行，一直到離開迴圈前都不會再執行。

### 2. 條件運算式

判斷是否要離開 for 迴圈，此運算式會在每次執行 for 迴圈之前進行判斷。若條件運算式的結果為零(假)表示不滿足條件，則離開 for 迴圈，執行接在 for 迴圈後面的敘述。若條件運算式的結果不為零(真)表示滿足條件，會執行 for 迴圈內的敘述區段一次，再回到 for 敘述中，先執行控制運算式，接著執行條件運算式，再依滿足條件與否決定執行迴圈內的敘述區段或離開迴圈。



```
printf("hello %d\n",i);  
printf("hello %d\n",i);  
printf("hello %d\n",i);
```

.....  
?

# FOR迴圈

- 範例:印出10行hello

- 所以在下面的例子中，進入迴圈前會令*i*=0;  
當*i*還是<10的時候，迴圈會繼續跑。
- 迴圈每跑完一圈，就會做*i++*的動作。
- 執行程式時可以看到*i*由一開始的0會每次都累加上1，  
到最後變成10之後就結束程式。

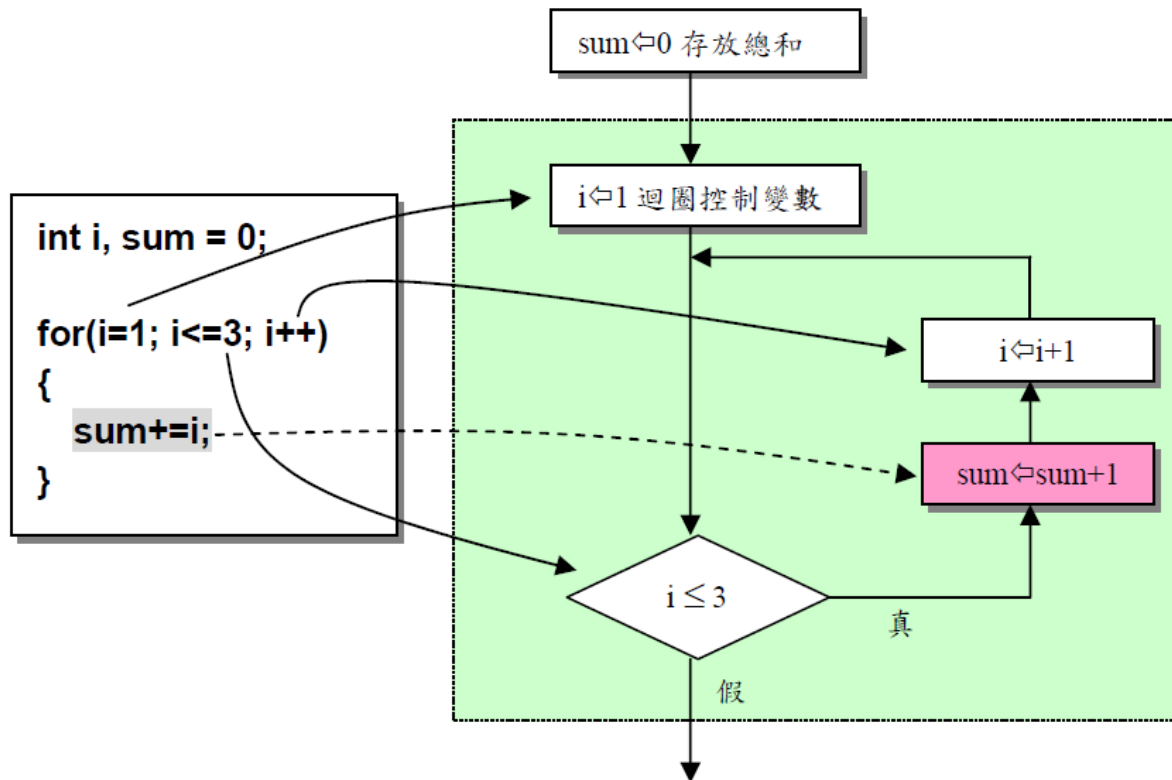
```
#include <stdio.h>  
int main()  
{  
    int i;  
    for ( i=0; i<10; i++ ) {  
        printf("hello %d\n",i);  
    }  
    return 0;  
}
```

# FOR迴圈

- 範例：算出 $1+2+3+...+9+10$

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

5. 下例是利用 for 迴圈，計算  $1+2+3=?$  總和的流程圖以及執行過程：



### 程式追蹤

| 敘述<br>迴圈 | i 值 | $i \leq 3$       | $sum \leftarrow sum + i$       | $i++$ |
|----------|-----|------------------|--------------------------------|-------|
| 第 1 次    | 1   | $1 \leq 3$ (成立)  | $sum \leftarrow 0 + 1$         | 2     |
| 第 2 次    | 2   | $2 \leq 3$ (成立)  | $sum \leftarrow 0 + 1 + 2$     | 3     |
| 第 3 次    | 3   | $3 \leq 3$ (成立)  | $sum \leftarrow 0 + 1 + 2 + 3$ | 4     |
| 第 4 次    | 4   | $4 \leq 3$ (不成立) | 離開迴圈                           |       |

## 練習

- 輸入一個大於0的整數n  
印出" $1+2+3+\dots+n = \text{結果}$ "
- 例如：輸入5，印出  $1+2+3+4+5 = 15$
- hint：想在中間的數字加上「+」號該怎麼做呢？



# 課程大綱

- 選擇控制 (流程控制)
  - IF-ELSE
  - SWITCH-CASE
- 重覆控制 (迴圈)
  - FOR
  - WHILE
  - 迴圈中的流程控制: break, continue
  - 巢狀迴圈
- 作業

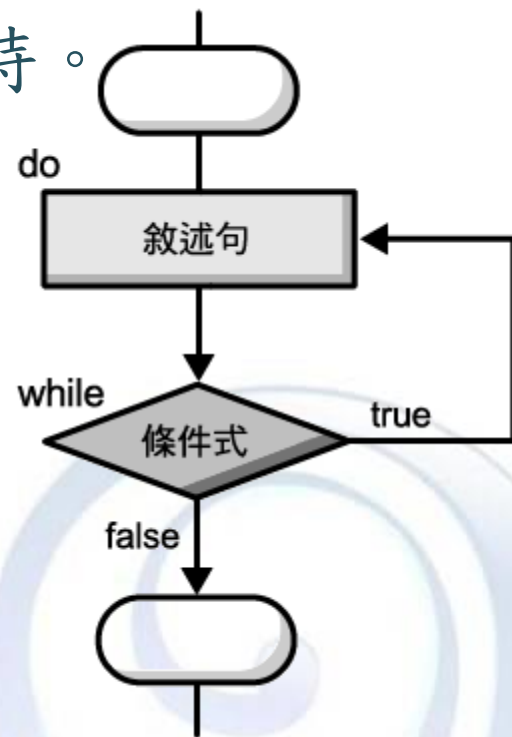
# WHILE迴圈(前測式)

- 用途：當程式需要來回重複執行某一段程式碼時

- while通常用在未知重覆執行次數時。

- 語法：

- while(繼續執行迴圈的條件)  
{  
    條件成立的話要做的程式碼;  
}



- 把for中的進入迴圈前要做的事與每跑完一次迴圈會做的事拿掉，就是while迴圈。

# WHILE迴圈(前測式)

- 範例：輸入鍵盤按鍵，直到輸入q後程式結束。

```
#include <stdio.h>
#include <conio.h>
int main()
{
    char key=0;

    while(key!='q')
    {
        key=getche();
        printf("\n%c\n", key);
    }
    return 0;
}
```

## DO-WHILE迴圈(後測式)

- 用途：當程式需要**先做一次**某一段程式碼**再判斷**是否要重複執行該一段程式碼時
- 語法：
  - **do**  
{  
    條件成立的話要做的程式碼;  
} **while**(**繼續執行迴圈的條件**); //記得加「;」號
- *while迴圈中的判斷式可以放在迴圈的最後端, 形成一個do-while迴圈*  
*這樣子的迴圈至少會跑一次*

# DO-WHILE迴圈

- 範例：輸入鍵盤按鍵，直到輸入q後程式結束。

```
#include <conio.h>
int main()
{
    char key;
    do
    {
        key=getche();
        printf("\n%c\n", key);
    }while(key!='q');

    return 0;
}
```

## 動動腦

- `while(exp){...}` 與 `do{...}while(exp);` 兩者在應用上有什麼不同？
- 剛才的例子是按下 **q** 離開程式，但
- 假如我想要讓使用者按下 **y** 才重覆執行程式呢？

# 課程大綱

- 選擇控制 (流程控制)
  - IF-ELSE
  - SWITCH-CASE
- 重覆控制 (迴圈)
  - FOR
  - WHILE
  - 迴圈中的流程控制: break, continue
  - 巢狀迴圈
- 作業

## 迴圈中的流程控制

- 在迴圈中，有兩個指令可方便做控制：
  - break：直接結束迴圈
  - continue：直接跳到迴圈開頭處繼續下一次執行
- 用途：常用來設定在迴圈中某些情形下，選擇做結束(break)或回頭(continue)



while (條件式)

{

[敘述區段 1]

break;

[敘述區段 2]

}

[敘述區段 3]

圖一

while (條件式)

{

[敘述區段 1]

continue ;

[敘述區段 2]

}

[敘述區段 3]

圖二

## 迴圈中的流程控制: break

- 範例：輸入鍵盤按鍵，直到輸入q後程式結束。

```
#include <conio.h>
int main()
{
    char key;

    while(1) //while中條件為 1 稱為無窮迴圈
    {
        key=getche();
        if ( key=='q' )
            break;
    }
    return 0;
}
```

## 迴圈中的流程控制: continue

- 範例：假如有一棟大樓沒有4樓這個樓層，寫一個程式顯示從1樓坐電梯到10樓所經過的樓層

```
#include <stdio.h>
int main()
{
    int i;
    for ( i=1; i<=10; i++ ){
        // 若是4樓就跳過, 回到迴圈開頭繼續執行
        if ( i==4 )
            continue;
        printf("floor %d\n",i);
    }
    return 0;
}
```

## 練習

- 寫個程式，判斷一個人的成績是否及格  
(及格分數為60分)
- 每次判斷完成績後可輸入
  - 'y': 繼續
  - 其他: 離開



# 課程大綱

- 選擇控制 (流程控制)
  - IF-ELSE
  - SWITCH-CASE
- 重覆控制 (迴圈)
  - FOR
  - WHILE
  - 迴圈中的流程控制: break, continue
  - 巢狀迴圈
- 作業

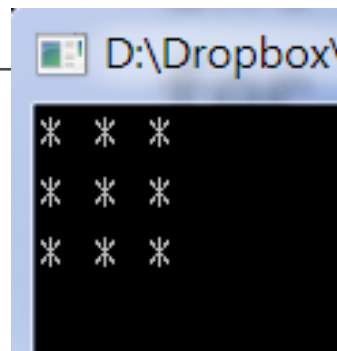
# 巢狀迴圈

- 迴圈中的迴圈

- 下面範例中, i 每加三次, j 才加一次  
試著觀察輸出結果

```
#include <stdio.h>
int main()
{
    int i, j;
    for ( j=0; j<3; j++ ) {
        for ( i=0; i<3; i++ ) {
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

```
for( ){
    for( ){
    }
}
```



```
D:\Dropbox\
* * *
* * *
* * *
```

## 練習

- 輸入一個整數, 印出n列\*號, 每列各含1~n個\*
- 例如:
  - 輸入5
  - 輸出:

```
✕  
✕  ✕  
✕  ✕  ✕  
✕  ✕  ✕  ✕  
✕  ✕  ✕  ✕  ✕
```

# 課程大綱

- 選擇控制 (流程控制)
  - IF-ELSE
  - SWITCH-CASE
- 重覆控制 (迴圈)
  - FOR
  - WHILE
  - 迴圈中的流程控制: break, continue
  - 巢狀迴圈
- 作業



## 回家作業：

- 印出下列九九乘法表：

```
1×1= 1 2×1= 2 3×1= 3 4×1= 4 5×1= 5 6×1= 6 7×1= 7 8×1= 8 9×1= 9
1×2= 2 2×2= 4 3×2= 6 4×2= 8 5×2=10 6×2=12 7×2=14 8×2=16 9×2=18
1×3= 3 2×3= 6 3×3= 9 4×3=12 5×3=15 6×3=18 7×3=21 8×3=24 9×3=27
1×4= 4 2×4= 8 3×4=12 4×4=16 5×4=20 6×4=24 7×4=28 8×4=32 9×4=36
1×5= 5 2×5=10 3×5=15 4×5=20 5×5=25 6×5=30 7×5=35 8×5=40 9×5=45
1×6= 6 2×6=12 3×6=18 4×6=24 5×6=30 6×6=36 7×6=42 8×6=48 9×6=54
1×7= 7 2×7=14 3×7=21 4×7=28 5×7=35 6×7=42 7×7=49 8×7=56 9×7=63
1×8= 8 2×8=16 3×8=24 4×8=32 5×8=40 6×8=48 7×8=56 8×8=64 9×8=72
1×9= 9 2×9=18 3×9=27 4×9=36 5×9=45 6×9=54 7×9=63 8×9=72 9×9=81
請按任意鍵繼續 . . . ■
```

- 提示：在印星號的例子中，將迴圈控制變數*i*, *j*印出

## 作業(加分題): 中文大寫數字輸出

- 輸入一個金額1~99999整數, 印中文大寫數字金額
  - 中文大寫1~9: 壹, 貳, 參, 肆, 伍, 陸, 柒, 捌, 玖
  - 需要印出單位: 萬, 仟, 佰, 拾
  - 不需輸出"零"
  - 最後要印出"元整"
  - 輸入錯誤數值範圍, 顯示錯誤後程式直接結束
- 輸入輸出格式 (格式必須與下面程式結果一致)

```
請輸入金額: 12345  
壹萬貳仟參佰肆拾伍元整
```