

第三章 流程控制與例外處理



3.1 選擇結構

3.4 程式除錯

3.2 重複結構

3.5 例外處理

3.3 break與continue陳述式

備註：可依進度點選小節



3.1 選擇結構

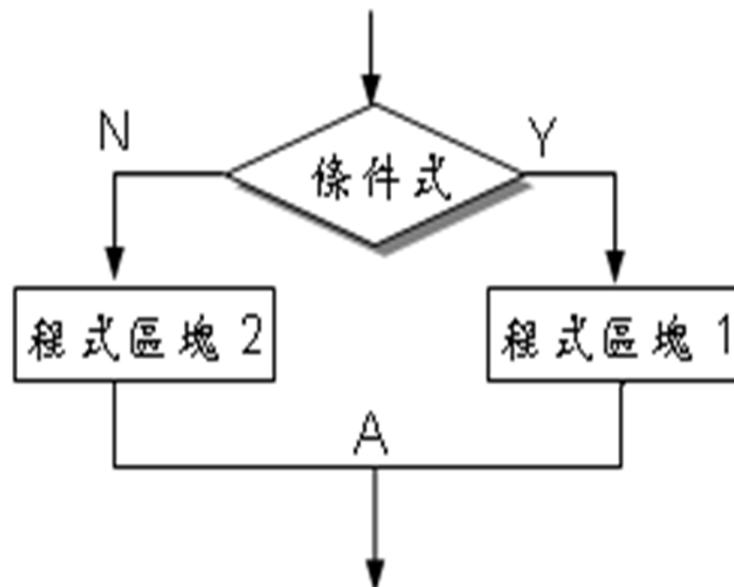
三種方式：

1. **if… else** (雙重或單一選擇)
2. **if… else if… else** (多重選擇兩種以上)
3. **switch** (多重選擇兩種以上)



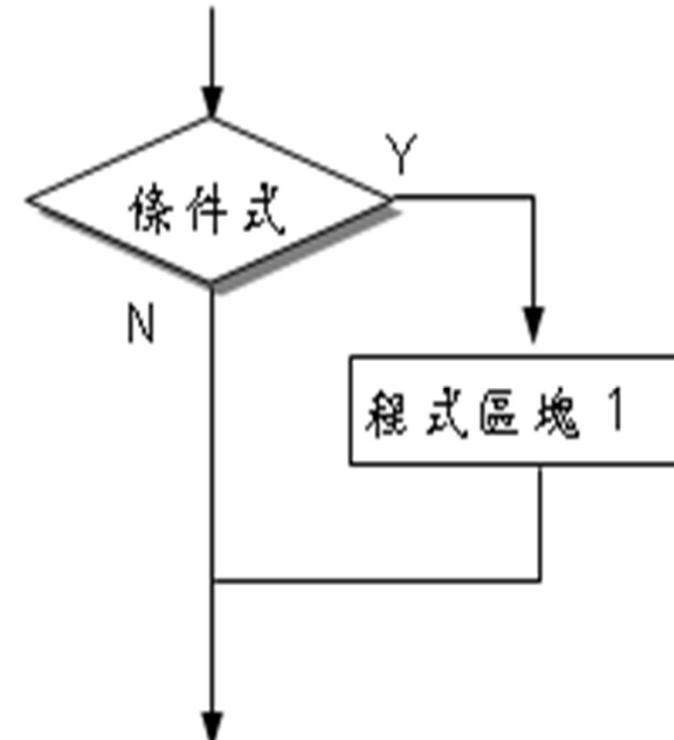
3.1.1 if...else 選擇陳述式

```
if (條件式)
{
    程式區塊 1;
}
else
{
    程式區塊 2;
}
```





```
if (條件式)
{
    程式區塊 1;
}
```





If (條件式) 陳述式；

【例1】

若單價(price)大於等於1,000元，折扣(discount)八折，
否則折扣九折。

```
if ( price>=1000)
{
    discount=0.8 ; // price大於等於1,000執行此敘述
}
else
{
    discount=0.9 ; // price小於1,000 執行此敘述
}
```



【例2】

年齡(age)是10歲(含)以下或60歲(不含)以上則票價(price)
為100元，否則為200元。寫法如下：

```
if (age<=10 || age>60)
{
    price=100 ;
}
else
{
    price=200 ;
}
```



巢狀 if

若 **if** 或 **else** 程式區塊內 還有 **if...else** 陳述式。

範例演練

試寫一個程式由鍵盤輸入兩個整數(**num1**和**num2**)，

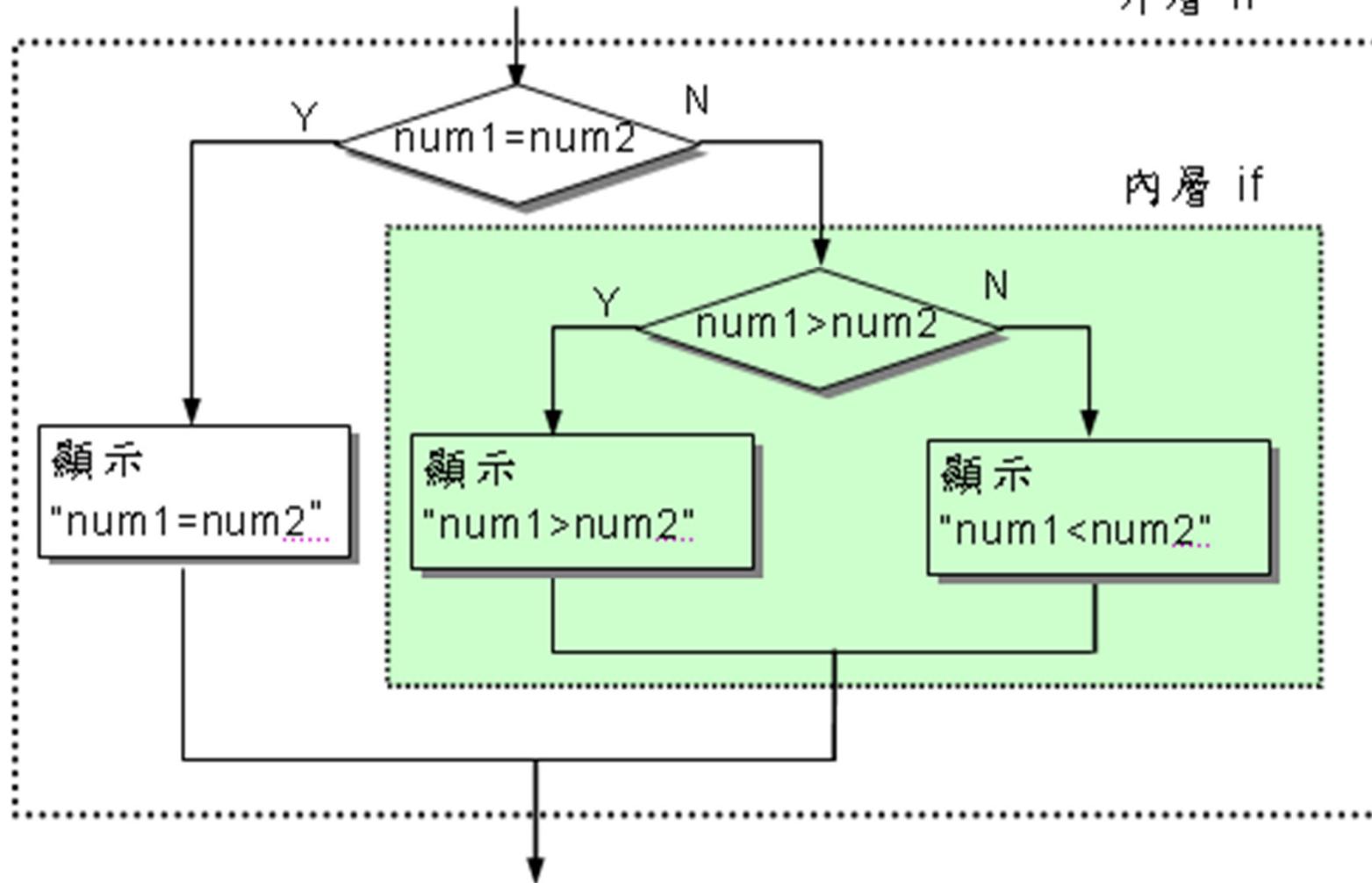
- ① 若**num1=num2**，則顯示 “**num1=num2**”。
- ② 若**num1>num2**，則顯示 “**num1>num2**”。
- ③ 若**num1<num2**，則顯示 “**num1<num2**”。

```
c:\ file:///C:/CSharp/chap03/ifelse1/bin/Debug/ifelse1.EXE - □ X
請輸入第一個整數(num1) : 45
請輸入第二個整數(num2) : 30
45 > 30
```



外層 if

內層 if





```
// FileName: ifelse1.sln
01 namespace ifelse1
02 {
03     class Program
04     {
05         static void Main(string[] args)
06         {
07             int num1, num2;
08             Console.Write("請輸入第一個整數(num1) : ");
09             num1 = int.Parse(Console.ReadLine());
10             Console.Write("請輸入第二個整數(num2) : ");
11             num2 = int.Parse(Console.ReadLine());
12             if (num1 == num2)
13             {
14                 Console.WriteLine("{0} = {1}", num1, num2);
15             }
16         }
17     }
18 }
```



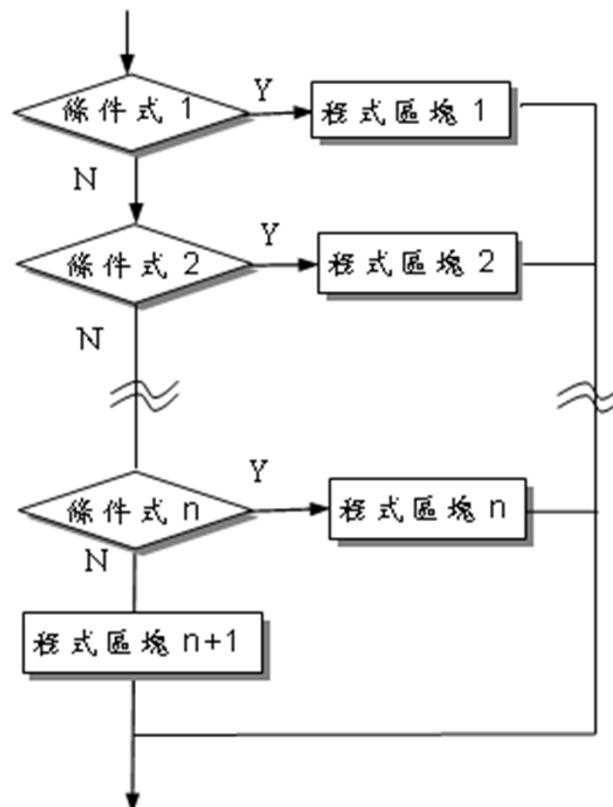
```
16     else
17     {
18         if (num1 > num2)
19         {
20             Console.WriteLine("{0} > {1}", num1, num2);
21         }
22     else
23     {
24         Console.WriteLine("{0} < {1}", num1, num2);
25     }
26 }
27     Console.Read();
28 }
29 }
30 }
```



3.1.2 if…else if…else 多重選擇陳述式

- 有兩個以上的條件式需要連續做判斷時

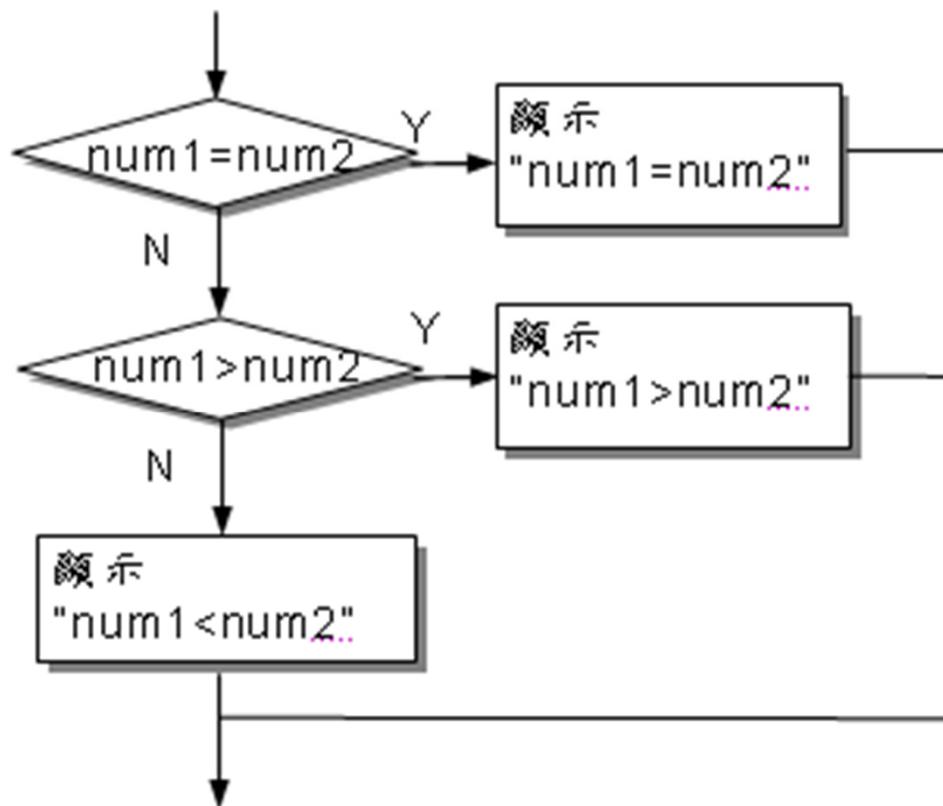
```
if(條件式 1)
{
    程式區塊 1
}
else if(條件式 2)
{
    程式區塊 2
}
.
.
.
else if(條件式 n)
{
    程式區塊 n
}
else
{
    程式區塊 n+1
}
```





範例演練

延續上一範例，改用 **if … else if … else** 多重選擇陳述式來撰寫兩數比大小。





```
// FileName: ifelseif1.sln
01 static void Main(string[] args)
02 {
03     int num1, num2;
04     Console.Write("請輸入第一個整數(num1) :");
05     num1 = int.Parse(Console.ReadLine());
06     Console.Write("請輸入第二個整數(num2) :");
07     num2 = int.Parse(Console.ReadLine());
08     if (num1 == num2)
09     {
10         Console.WriteLine("{0} = {1}", num1, num2);
11     }
12     else if (num1 > num2)
13     {
14         Console.WriteLine("{0} > {1}", num1, num2);
15     }
16     else
17     {
18         Console.WriteLine("{0} < {1}", num1, num2);
19     }
20     Console.Read();
21 }
```



3.1.3 switch 多重選擇陳述式

■ 使用上差異

① **if … else if … else** 可使用多個不同條件式。

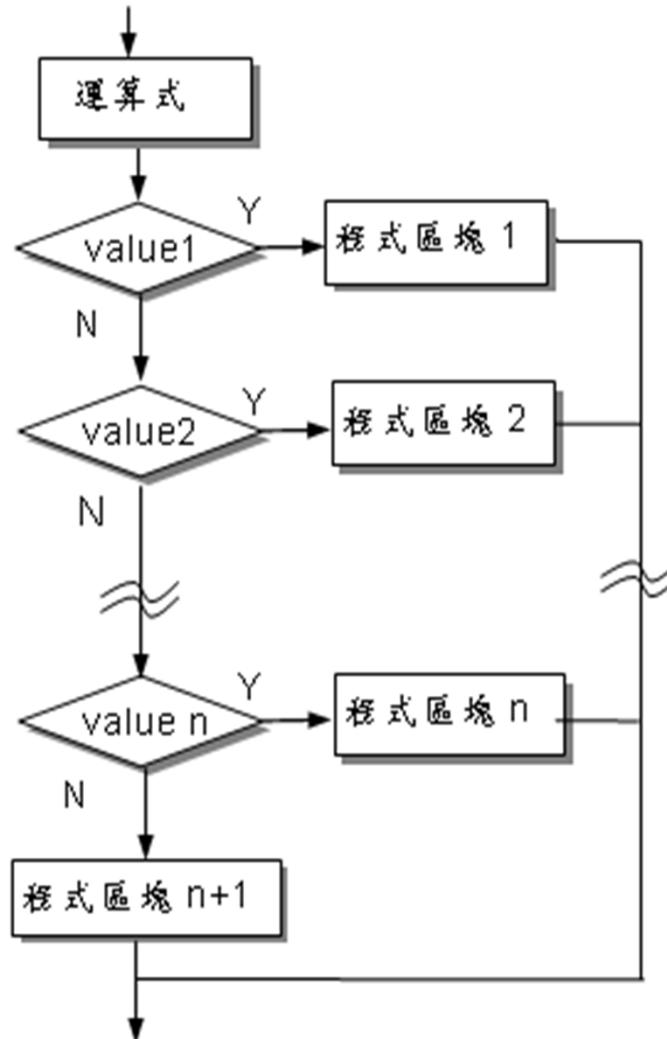
② **switch** 只允許用一個運算式。

依其運算式結果 判斷其值是落在哪個範圍。

■ 使用太多 **if** 使得程式看起來複雜且不易維護， **switch** 多重選擇陳述式則不會。



```
switch (運算式)
{
    case value1:
        程式區塊 1;
        break;
    case value2:
        程式區塊 2;
        break;
    ...
    case value n:
        程式區塊 n
        break;
    default:
        程式區塊 n+1
}
```





■ Case 各種寫法

① 若條件式為 1、2、4 為真：

```
case 1:  
case 2:  
case 4:  
    ...  
    程式區塊;  
break;
```

② 若條件式結果為 "Y" 或 "y" 為真：

```
case "y":  
case "Y":  
    ...  
    程式區塊;  
break;
```



範例演練

試使用 **switch**，由鍵盤輸入現在的月份(1~12)，

- ① 輸入5，表5月份，由程式判斷5月份是屬於哪一季？顯示“現在是第二季”；
- ② 若輸入值超出範圍，則顯示“... 輸入值超出範圍...”。

```
file:///C:/CSharp/chap03/switch1/bin/Debug/switch1.EXE
==> 請輸入現在的月份: 5
...
... 現在是第二季...

file:///C:/CSharp/chap03/switch1/bin/Debug/switch1.EXE
==> 請輸入現在的月份: 20
...
... 輸入值超出範圍....
```



```
// FileName: switch1.sln
01 static void Main(string[] args)
02 {
03     string month;
04     Console.Write("==> 請輸入現在的月份: ");
05     month = Console.ReadLine();
06     switch (month)
07     {
08         case "1":
09         case "2":
10         case "3":
11             Console.WriteLine("\n ... 現在是第一季...");
12             break;
13         case "4":
14         case "5":
15         case "6":
16             Console.WriteLine("\n ... 現在是第二季...");
17             break;
    }
```

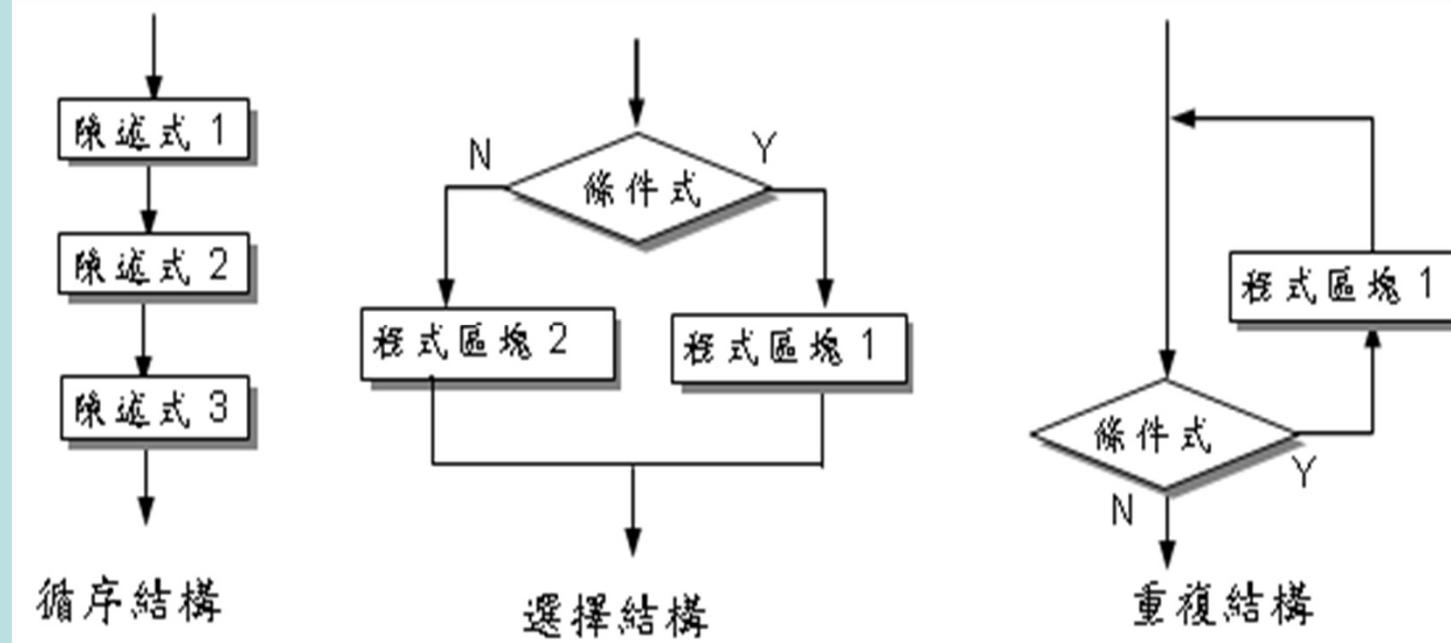


```
18    case "7":  
19    case "8":  
20    case "9":  
21        Console.WriteLine(" \n ... 現在是第三季...");  
22        break;  
23    case "10":  
24    case "11":  
25    case "12":  
26        Console.WriteLine(" \n ... 現在是第四季...");  
27        break;  
28    default:  
29        Console.WriteLine(" \n ... 輸入值超出範圍....");  
30        break;  
31    }  
32    Console.Read();  
33 }
```



3.2 重複結構

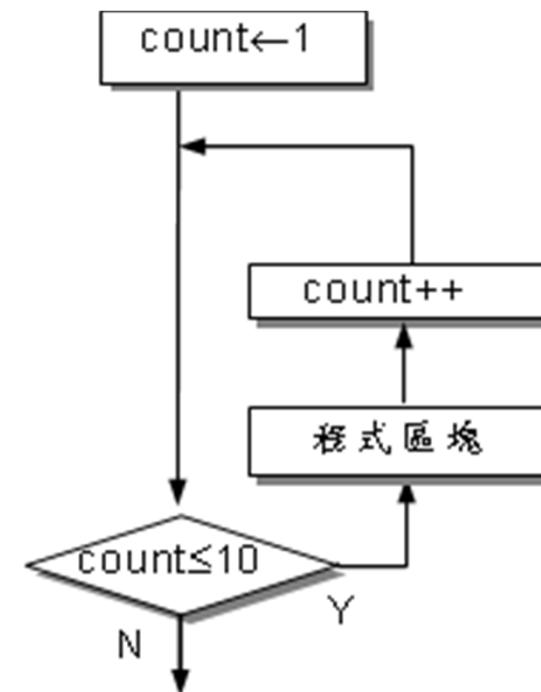
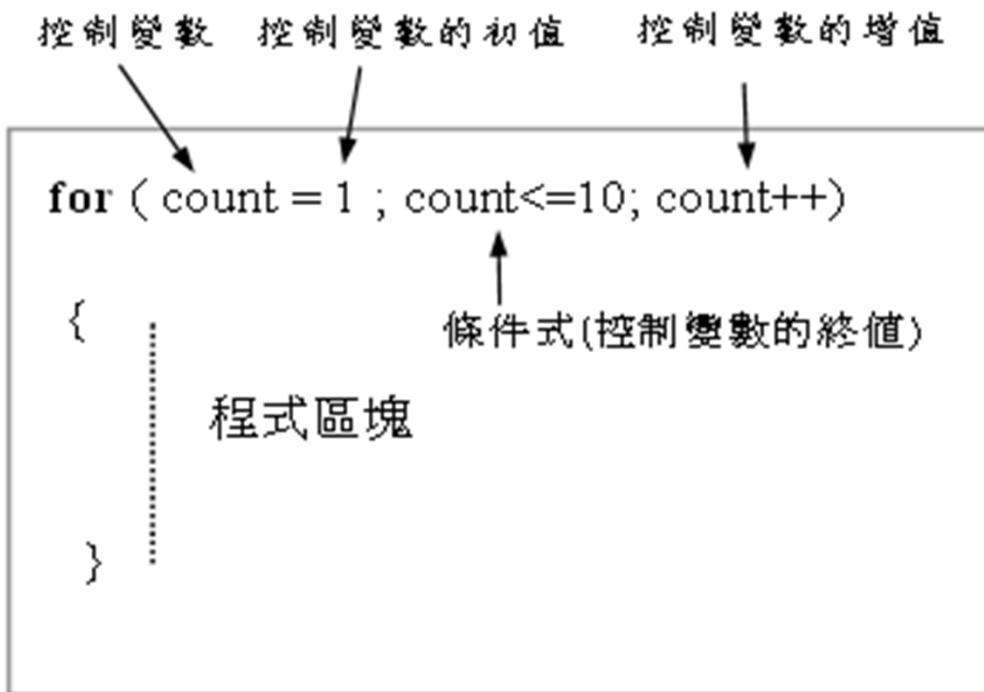
- 重複結構 是指設計程式時需要將某部份程式區塊重複執行指定的次數，或一直執行到不滿足條件為止。
- 重複結構亦稱 **迴圈 (Loop)**。
- **for**陳述式 指定次數者稱為「計數器」控制迴圈。
- **while**陳述依條件者稱為「條件式」控制迴圈。
- 三種結構的流程圖如下圖所示：





3.2.1 for 迴圈

- 計數器控制迴圈以 for 的 左大括號開始，以右大括號結束。
- 語法：





- 中途欲離開 for 迴圈，使用 **break** 陳述式。

- 使用 **continue**，立即跳回 for 的開頭繼續執行。



一般 for 迴圈的常用寫法：

① **for (k=1 ; k<= 5 ; k++)**

$k = 1, 2, 3, 4, 5$ 共執行迴圈內的程式區塊5次。

② **for (k=1 ; k<=5 ; k+=2)**

$k = 1, 3, 5$ 共執行迴圈內的程式區塊3次。

③ 初值、增值可為小數

for (k=-0.5 ; k<=1.5 ; k+= 0.5)

$k = -0.5, 0, 0.5, 1.0, 1.5$ 共執行迴圈內程式區塊5次。

④ 增值採遞減

for (k=6 ; k>=1 ; k-=2)

$k = 6, 4, 2$ 共執行迴圈內的程式區塊3次。



⑤ 若初值、增值都有兩個以上，中間使用逗號分開：

```
for (x=1, y=5 ; x<3 && y>2 ;x++ , y-- )
```

x=1 & y=5 ; x=2 & y=4 ； 共執行迴圈內的程式區塊2次。

⑥ 初值和終值可以為運算式

```
for (k=x ; k<=y+9 ;k+=2)
```

若x=1、y=-2，則 k = 1,3,5,7 共執行迴圈內的程式區塊 4 次。

⑦ 無窮迴圈

```
for ( ; ; )
```



範例演練

試求下列級數的和。

$$\sum_{x=1}^5 (2x+1) = \frac{3}{x=1} + \frac{5}{x=2} + \frac{7}{x=3} + \frac{9}{x=4} + \frac{11}{x=5} = ?$$

The screenshot shows a Windows command-line interface window titled "file:///C:/CSharp/chap03/series/bin/Debug/series.EXE". The window displays the following text:
==== 求級數的總和 ====
===== x ===== 2x+1
===== 1 ===== 3
===== 2 ===== 5
===== 3 ===== 7
===== 4 ===== 9
===== 5 ===== 11

此級數總和為 : 35



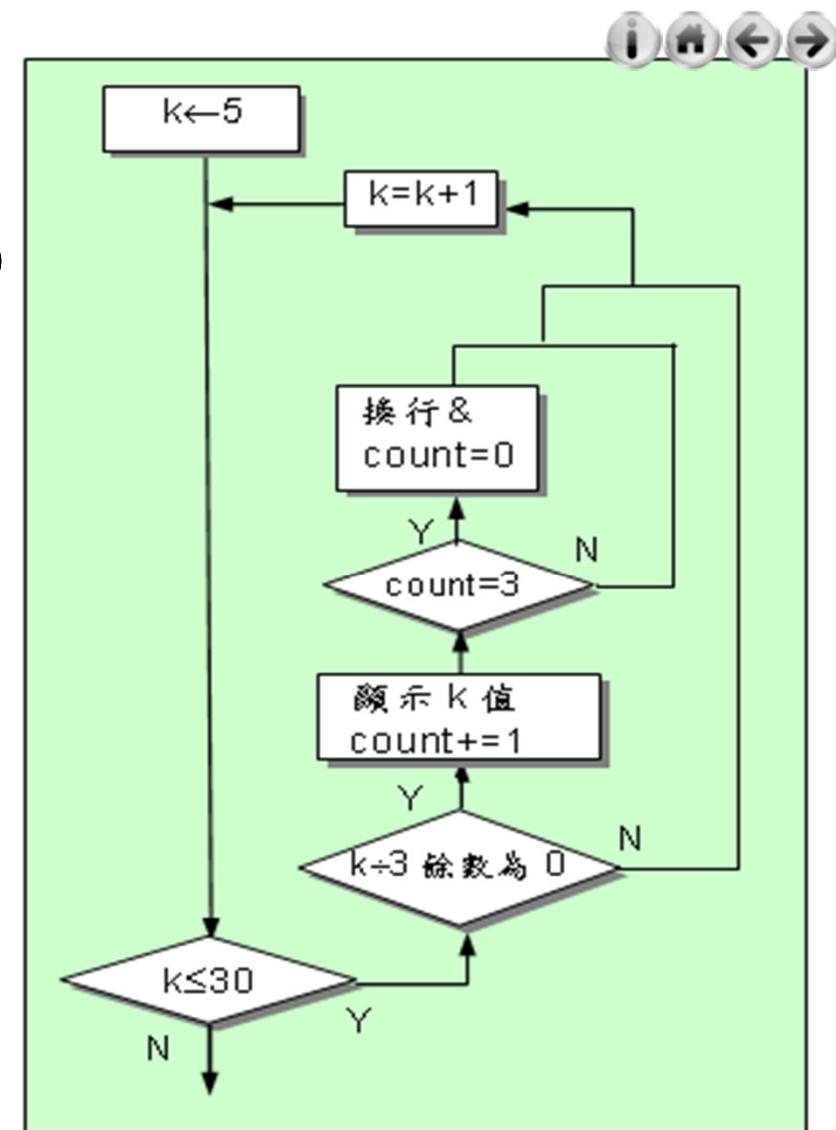
```
// FileName: series.sln
01 static void Main(string[] args)
02 {
03     int x, sum = 0;
04     Console.WriteLine ("\n === 求級數的總和 === \n");
05     Console.WriteLine ("    x      2x+1 ");
06     Console.WriteLine (" ======      ======");
07     for (x = 1; x <= 5; x++)
08     {
09         Console.WriteLine ("    {0}      {1}", x, 2 * x + 1);
10         sum += 2 * x + 1;
11     }
12     Console.WriteLine (" ----- ");
13     Console.WriteLine (" 此級數總和為 :{0} \n", sum);
14     Console.Read();
15 }
```



範例演練

試寫一個程式將介於5到30(含)
是3的倍數顯示出來，顯示時
每3個倍數印一行。

```
c:\ file:///C:/... - □ X
6 9 12
15 18 21
24 27 30
```





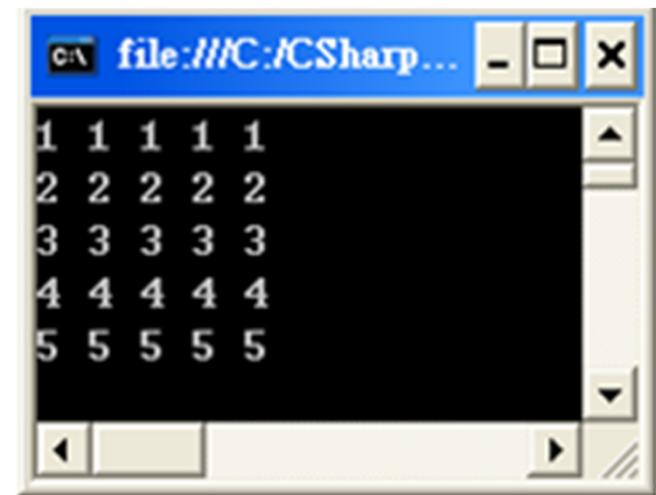
```
// FileName: for1.sln
01 static void Main(string[] args)
02 {
03     int k, count = 0;
04     for (k = 5; k <= 30; k++)
05     {
06         if ((k % 3) == 0)
07         {
08             Console.WriteLine("{0} ", k);
09             count++;
10             if (count == 3)
11             {
12                 Console.WriteLine();
13                 count = 0;
14             }
15         }
16     }
17     Console.Read();
18 }
```



3.2.2 巢狀迴圈

- 若迴圈內還有迴圈就構成巢狀迴圈，一般應用在二維資料列表。
- 下例外迴圈的變數，每個變數值印五次後，再將游標移到下一列最前面，總共印五列。

```
for (int i = 1; i <= 5; i++)      外迴圈
{
    for (int k = 1; k <= 5; k++)
    {
        Console.Write("{0} ", i);   內迴圈
    }
    Console.WriteLine();
}
```





3.2.3 前測式迴圈

- 前測式迴圈 就是將 條件式 放在迴圈的 最前面 。
- 依條件式的真假決定是否進入迴圈：
 - ① 若滿足條件將迴圈內的程式區塊執行一次，然後再回迴圈最前面條件式。
 - ② 不滿足時才離開迴圈。
- 前測式迴圈，若第一次進入迴圈時便不滿足條件式，馬上離開迴圈，連一次都沒執行迴圈內的程式區塊。



語法	流程圖
<pre>while(條件式) { ... }</pre> <p>程式區塊</p>	<pre>graph TD Start(()) --> Cond{條件式} Cond -- N --> End(()) Cond -- Y --> Block[程式區塊] Block --> Cond</pre>



3.2.4 後測式迴圈

- 後測式迴圈 就是將 條件式 放在迴圈的 最後面。
- 第一次不用檢查條件式，直接進入迴圈執行裡面的程式區塊，才判斷條件式的真假：
 - ① 若滿足條件會將迴圈內的程式碼執行一次，再檢查位於迴圈最後面的條件式。
 - ② 一直到不滿足條件時才離開迴圈。
- 此種架構迴圈內的程式區塊至少會執行一次。



語法	流程圖
<pre>do { 程式區塊 } while (條件式);</pre>	<pre>graph TD A[程式區塊] --> B{條件式} B -- N --> C B -- Y --> A</pre>



範例演練

試寫一個使用前測式迴圈計算階乘的程式，由使用者先輸入一個整數，接著再計算該數的階乘值。
如下圖輸入5，結果計算出 $5! = 120$ 。





```
// FileName: factorial1.sln
01 static void Main(string[] args)
02 {
03     int keyin, num, factorial = 1;
04     Console.Write("請輸入整數: ");
05     keyin = int.Parse(Console.ReadLine());
06     num = keyin;
07     while (num >= 1)
08     {
09         factorial *= num;
10         num -= 1;
11     }
12     Console.WriteLine("{0}! = {1}", keyin, factorial);
13     Console.Read();
14 }
```



3.3 break 與 continue 陳述式

- 當使用 **for**、**while** 或 **do…while** 迴圈時，在迴圈內要中途離開迴圈時，可在欲離開處插入 **break** 陳述式，便可直接離開迴圈，繼續執行接在迴圈後面的陳述式。
- 若要中途返回迴圈開始處，可在欲返回處插入 **continue** 陳述式。
- **Break** 和 **continue** 都是用來改變迴圈的執行流程。
- 迴圈內接在 **break** 或 **continue** 後面的陳述式是不被執行。
- 譬如一個無窮迴圈，可在迴圈內適當位置插入 **if** 陳述式，藉由在條件中插入 **continue** 和 **break** 來控制迴圈。



break	continue
<pre>for (.....) { 陳述式 1; 陳述式 2; ... break; ━━━━━━ 陳述式 n-1; 陳述式 n; }</pre>	<pre>for (.....) ← { 陳述式 1; 陳述式 2; ... continue; ━━━━━━ 陳述式 n-1; 陳述式 n; }</pre>
<pre>while (條件式) { 陳述式 1; 陳述式 2; ... break; ━━━━━━ 陳述式 n-1; 陳述式 n; }</pre>	<pre>while (條件式) ← { 陳述式 1; 陳述式 2; ... continue; ━━━━━━ 陳述式 n-1; 陳述式 n; }</pre>



```
do
{
    陳述式 1;
    陳述式 2;
    ...
    break;
    陳述式 n-1;
    陳述式 n;
} while (條件式);
```

```
do
{
    陳述式 1;
    陳述式 2;
    ...
    continue;
    陳述式 n-1;
    陳述式 n;
} while (條件式);
```

由上表可知 陳述式 n-1 及 陳述式 n 都不會被執行到。



範例演練

試寫一個連續輸入數值累加總和的程式，在無窮迴圈
do…while 中，透過**break**和**continue**來判斷是否繼續
累加輸入值。

```
c:\ file:///C:/CSharp/chap03/breakcontinue1/... - □ ×  
== 請輸入一個數值: 7  
== 是否繼續輸入(Y/N) ? : y  
== 請輸入一個數值: 88  
== 是否繼續輸入(Y/N) ? : y  
== 請輸入一個數值: 12  
== 是否繼續輸入(Y/N) ? : n  
== 3 個數的總和: 107
```



```
// FileName: breakcontinue1.sln
01 static void Main(string[] args)
02 {
03     int count = 0, keyin = 0, sum = 0;
04     string str1;
05     do
06     {
07         Console.Write(" == 請輸入一個數值: ");
08         keyin = Convert.ToInt32(Console.ReadLine());
09         sum += keyin;
10         count++;
11         Console.Write(" == 是否繼續輸入(Y/N) ? : ");
12         str1 = Console.ReadLine();
13         if ((str1 == "y") || (str1 == "Y"))
14         {
15             continue;
```



```
16      }
17  else
18  {
19      break;
20  }
21 } while (true);
22 Console.WriteLine("\n == {0}個數的總和: {1} ", count, sum);
23 Console.Read();
24 }
```



3.4 程式除錯

- 當程式執行若結果不符合預期，錯誤可能發生在編譯階段或執行階段。
- 一般編譯發生錯誤大都是語法錯誤，表示所撰寫的陳述式不符合C# 2008 所規定語法。
- 此時在該識別字正下方出現**藍色波浪線**，表示該識別字C# 2008無法辨別。此時便要做**除錯(Debug)**工作，直到發生錯誤地方無誤時，藍色波浪線才會消失。
- 當一個程式在編譯時沒錯誤發生，在執行階段若無法得到預期的結果，表示發生邏輯上錯誤
- **邏輯錯誤** 並不是語法錯誤，而是程式的流程、運算式、變數誤用等錯誤，此時就需要使用「區域變數」視窗來做逐行偵錯，觀察每行執行結果是否正確？以找出發生錯誤的地方。

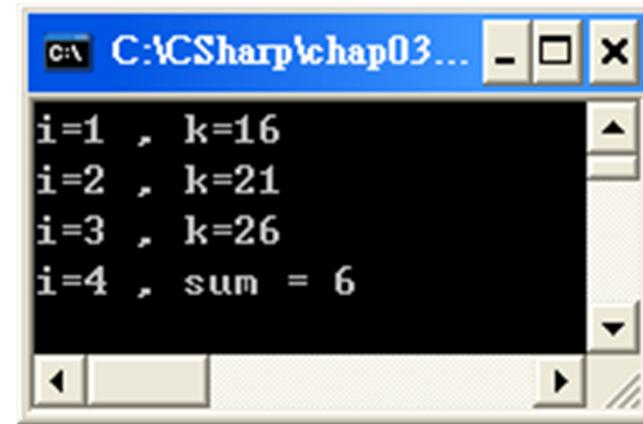


3.4.1 逐行偵錯

- Visual C# 2008提供「區域變數」視窗來評估變數和運算式，並保存其結果。
- 可用「區域變數」視窗來編輯變數或暫存器的數值。
- 下例透過 for 迴圈來學習程式如何做逐行偵錯。
- 首先自行鍵入下列程式，或由書附光碟中載入 chap03/debug1.sln來練習。



```
// FileName: debug1.sln
01 static void Main(string[] args)
02 {
03     int i, k, sum = 0;
04     k = 11;
05     for (i = 1; i <= 3; i++)
06     {
07         k += 5;
08         sum += i;
09         Console.WriteLine("i={0} , k={1}", i, k);
10     }
11     Console.WriteLine("i={0} , sum = {1} ", i, sum);
12     Console.Read();
13 }
```



上面程式執行時，請按照下面步驟來對程式做逐行偵錯工作。



Step1

先點選功能表的【偵錯(D)/逐步執行(I)】或直接按 **F11** 鍵，此時在Main()方法的下一行”{”會出現 向右箭頭，表示下次執行由此行陳述式開始，進入逐行偵錯：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace debug1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, k, sum = 0;
            k = 11;
            for (i = 1; i <= 3; i++)
            {
                k += 5;
                sum += i;
                Console.WriteLine("i={0}, k={1}", i, k);
            }
            Console.WriteLine("i={0}, sum = {1}", i, sum);
            Console.Read();
        }
    }
}
```



Step2

點選【偵錯(D)/視窗(W)/區域變數(L)】開啟「區域變數」監看視窗。

The screenshot shows the Visual Studio 2008 IDE during a debugging session. The code editor displays the following C# code:

```
debug1 Program
Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace debug1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, k, sum = 0;
            k = 11;
            for (i = 1; i <= 3; i++)
            {
                k += 5;
                sum += i;
                Console.WriteLine("i={0} , k={1}", i, k);
            }
            Console.WriteLine("i={0} , sum = {1} ", i, sum);
            Console.Read();
        }
    }
}
```

A yellow arrow points to the first line of the for loop. To the right of the code editor, the 'Locals' window is open, showing the current values of variables:

名稱	值	型別
args	{string[0]}	string[]
i	0	int
k	0	int
sum	0	int



Step3

按 **F11** 功能鍵三次，已執行過 $k = 11$ ，區域變數
監看視窗內的 K 值由 $0 \rightarrow 11$ 。

The screenshot shows the Visual Studio IDE during debugging. On the left, the code editor displays the following C# code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace debug1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, k, sum = 0;
            k = 11;
            for (i = 1; i <= 3; i++)
            {
                k += 5;
                sum += i;
                Console.WriteLine("i={0} , k={1}", i, k);
            }
            Console.WriteLine("i={0} , sum = {1}", i, sum);
            Console.Read();
        }
    }
}
```

A yellow oval highlights the assignment statement `k = 11;`. Another yellow oval highlights the loop control variable `i` in the `for` loop condition `i = 1`. To the right, the "Locals" window is open, showing the current values of variables:

名稱	值	型別
args	(string[]) string[]	
i	0	int
k	11	int
sum	0	int

A pink oval highlights the value `11` in the `k` row of the locals window.



Step4

接著按 **F11** 功能鍵六次，執行 **sum += 1** 和 **Console...**，此時區域變數監看視窗內的 **sum** 由 $0 \rightarrow 1$ ，**i** 由 $0 \rightarrow 1$ ，**k** 由 $11 \rightarrow 16$ 。並執行**Console**陳述式，此時會將 **i** 和 **k** 值按照設定顯示格式顯示在主控台視窗。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace debug1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, k, sum = 0;
            k = 11;
            for (i = 1; i <= 3; i++)
            {
                k += 5;
                sum += i;
                Console.WriteLine("i={0} , k={1}", i, k);
            }
            Console.WriteLine("i={0} , sum = {1} ", i, sum);
            Console.Read();
        }
    }
}
```



Step5

接著按 **F11** 功能鍵一次，跳回執行**for...陳述式**，
將*i*值1→2，以此類推下去，便可看到各陳述式中，
各變數的變化情形。

若要中斷逐行偵錯，將尚未執行的陳述式一次執行
完畢，可按 開始偵錯鈕，會顯示最後結果。



3.4.2 設定中斷點

- 上節逐行偵錯是屬於細部除錯。
- C# 2008 另提供中斷點設定做大範圍除錯。

■ 設定中斷點做法：

在程式中欲監看的陳述式前面設定中斷點，程式每次執行到所設定的中斷點會停止執行，此時可透過區域變數監看視窗或移動滑鼠到該變數上會顯示該變數目前的值。



The screenshot shows a Microsoft Visual Studio 2008 interface. The title bar says "debug1 Program" and the tab bar shows "Main(string[] args)". The code editor displays the following C# program:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace debug1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, k, sum = 0;
            k = 11;
            for (i = 1; i <= 3; i++)
            {
                k += 5;
                sum += i;
                Console.WriteLine("i={0} , k={1}", i, k);
            }
            Console.WriteLine("i={0} , sum = {1} ", i, sum);
            Console.Read();
        }
    }
}
```



■ Step1 設定中斷點

先移動滑鼠到下圖程式中有 中斷點圖示處按一下，將指定的三行陳述式設成中斷點。有設定中斷點的陳述式會預設紅色底顯示：

```
debug1.Program
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace debug1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, k, sum = 0;
            k = 11;
            for (i = 1; i <= 3; i++)
            {
                k += 5;
                sum += i;
                Console.WriteLine("i={0} , k={1}", i, k);
            }
            Console.WriteLine("i={0} , sum = {1} ", i, sum);
            Console.Read();
        }
    }
}
```



■ Step2 開始偵錯

- ① 功能表【偵錯(D)/開始偵錯(S)】或直接按 **F5** 功能鍵，開始執行到第一個中斷點處暫停。
- ② 此時 `k+=5` 前出現 向右箭頭，表下次由此陳述式開始往下執行。
- ③ 可從「區域變數」監看視窗觀看目前變數的值。

若螢幕未出現「區域變數」監看視窗，行功能表【偵錯(D)/視窗(W)/區域變數(L)】開啟「區域變數」視窗。

名稱	值	類型
i	1	int
k	11	int
sum	0	int



Step3 直接觀看

直接移動滑鼠到編碼視窗任一個變數上停一會兒，
在該變數的右下方出現目前該變數的值。

名稱	值	型別
args	{string[0]}	string[]
i	1	int
k	11	int
sum	0	int



■ Step4 繼續執行

按 **F5** 功能鍵或 鈕，執行到” }” 符號便暫停執行。
各行陳述式中變數變化情形如下圖箭頭。

```
debug1 Program
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace debug1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, k, sum = 0;
            k = 11;
            for (i = 1; i <= 3; i++)
            {
                i += 5;
                sum += i;
                Console.WriteLine("i={0} , k={1}", i, k);
            }
            Console.WriteLine("i={0} , sum = {1} ", i, sum);
            Console.Read();
        }
    }
}
```

區域變數

名稱	值	類型
args	{string[0]}	string
i	2	int
k	16	int
sum	1	int

i=1 , k=16



■ Step6 停止偵錯

若中途欲停止偵錯可以執行【偵錯(D)/停止偵錯(E)】或直接按 停止偵錯圖示即可。

■ Step7 取消中繼點

移動滑鼠到欲取消中斷點陳述式前面的  圖示上按一下，中斷點便消失，該行陳述式即恢復正常狀態。



3.5 例外處理

■ 例外(Exception)

就是指當程式在執行時期所發生的錯誤。

■ C# 2008 提供一個具有結構且易控制的機制來處理執行時期原程式未考慮的狀況所發生的錯誤稱為「例外處理」。

■ 設計良好的錯誤處理程式碼區塊，可讓程式更為穩定，且更不易因應用程式處理此類錯誤而當機。



■ 例外處理主要由 **try**、**catch**、**throw**、**finally** 四個關鍵字構成。

- ① 方式是將要監看是否發生錯誤的程式區塊放在 **try** 區塊內
- ② 當 **try** 區塊內的任何陳述式執行時發生錯誤，該例外會被丟出 **throw**，在程式碼中利用 **catch** 抓取此例外情況
- ③ C# 會由上而下逐一檢查每個 **catch** 陳述式，當找到符合的 **catch** 陳述式，會將控制權移轉到該 **catch** 陳述式內程式區塊的第一列陳述式去執行。
- ④ 當該 **catch** 程式區塊執行完畢，不再繼續往下檢查 **catch** 陳述式
- ⑤ 直接跳到 **finally** 內執行 **finally** 程式區塊。
- ⑥ 若未找到符合的 **catch** 陳述式，最後也會執行 **finally** 內的 **finally** 程式區塊後才離開 **try**。



```
try
{
    [try 程式區塊] // 需例外處理的程式區塊
}
catch(exception1 ex)
{
    [catch 程式區塊]
    // 當需例外處理的程式區塊發生錯誤，符合exception1時
    // 執行此程式區塊1；
}
catch(exception2 ex)
{
    [catch 程式區塊]
    // 當需例外處理的程式區塊發生錯誤，符合exception2時
    // 執行此程式區塊2；
}
finally
{
    [finally 程式區塊] // 無論是否發生例外，都會執行此程區塊
}
```



例外類別	發生錯誤原因
ArgumentOutOfRangeException	當參數值超過某個方法所允許的範圍時所產生的例外。
DivideByZeroException	當除數為零時所產生的例外。
IndexOutOfRangeException	當陣列索引值超出範例時所產生的例外。
InvalidCastException	資料型別轉換錯誤時所產生的例外。
OverflowException	資料發生溢位時所產生的例外。
Exception	執行時期發生錯誤時所產生的例外。



範例演練

試寫一個會發生除數為零 **DivideByZeroException** 例外程式或直接開啟下面 **try1.sln** 範例程式。本程式中先宣告 i、k、p 為整數變數，並設定 i 初值為 5，k 初值為 0。當執行 i/k 時會發生除數為零的 **DivideByZeroException** 例外此時程式即會終止執行。

The screenshot shows the Visual Studio IDE with a code editor containing the following C# code:

```
try1.Program
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace try1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, k, p;
            i = 5;
            k = 0;
            p = i / k;
            Console.WriteLine(p);
            Console.Read();
        }
    }
}
```

A yellow callout points from the line `p = i / k;` in the code editor to a **DivideByZeroException 未處理** (Unhandled DivideByZeroException) dialog box. The dialog box contains the following text:

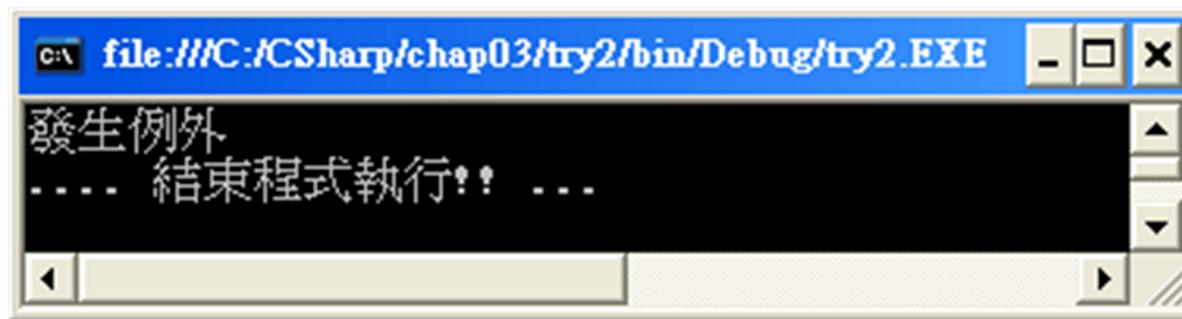
嘗試以零除。
疑難排解提示:
執行除法運算前請確定分母不是零。
取得這項例外狀況的一般說明。
動作:
檢視詳細資料...
將例外狀況詳細資料複製到剪貼簿



```
// FileName: try2.sln
01 static void Main(string[] args)
02 {
03     int i, k, p;
04     i = 5;
05     k = 0;
06     try
07     {
08         p = i / k;
09     }
10     catch (Exception ex)
11     {
12         Console.WriteLine("發生例外");
13     }
14     finally
15     {
16         Console.WriteLine(".... 結束程式執行!! ...");
17     }
18     Console.Read();
19 }
```



- 只要此行發生錯誤，會自動去找符合**catch**陳述式。
- 執行第8行，由於分母為零產生錯誤，此時會被第10行的 **Exception** 例外捕捉到而產生**Exception** 類別的ex 例外物件，接著在第12行直接印出 “發生例外” 訊息。
- 最後執行第14~17行**finally**程式區塊，顯示“.... 結束程式執行!! ...”。結果：





```
// FileName: try2.sln
01 static void Main(string[] args)
02 {
03     int i, k, p;
04     i = 5;
05     k = 0;
06     try
07     {
08         p = i / k;
09     }
10     catch (DivideByZeroException ex)
11     {
12         Console.WriteLine(ex.Message);
13     }
14     catch (Exception ex)
15     {
16         Console.WriteLine(ex.Message);
17     }
18     finally
19     {
20         Console.WriteLine(".... 結束程式執行!! ...");
21     }
22     Console.Read();
23 }
```



- 由於先符合 `catch(DivideByZeroException ex)`陳述式，所以執行第一個 `catch` 內的程式區塊後，跳過第二個 `catch(Exception ex)`陳述式，直接執行 `finally` 陳述式的程式區塊。
- 由於 `catch (Exception ex)` 是當上面所有 `catch` 陳述式中的 `Exception` 類別不符合時才執行，也就是發生其它的錯誤才接受，因此 `catch(Exception ex)` 必須放在所有 `catch` 陳述式的最後面及 `finally` 前面。
- 執行結果





■ 下表列出幾個例外物件常用的屬性與方法，透過這些方法可供你了解一些例外的資訊。

例外物件的成員	說明
GetType 方法	取得目前例外物件的資料型別。
ToString 方法	取得目前例外狀況的文字說明。
Message 屬性	取得目前例外的訊息。
Source 屬性	取得造成錯誤的應用程式或物件的名稱。
StackTrace 屬性	取得發生例外的方法或函式。



範例演練

延續上例，使用例外物件的 **GetType**、**ToString**、**Message**、**Source**、**StackTrace** 成員將例外的資訊顯示出來。

```
file:///C:/CSharp/chap03/try3/bin/Debug/try3.EXE
例外訊息：嘗試以零除。
發生例外的函式：    於 try3.Program.Main(String[] args) 於 C:\CSharp\chap03\try3\
Program.cs: 行 18
發生例外的物件：try3
發生例外的物件型別：System.DivideByZeroException
發生例外的文字說明：System.DivideByZeroException: 嘗試以零除。
    於 try3.Program.Main(String[] args) 於 C:\CSharp\chap03\try3\Program.cs: 行 1
8
.... 結束程式執行!! ...
```



```
// FileName: try3.sln
01 static void Main(string[] args)
02 {
03     int i, k, p;
04     i = 5;
05     k = 0;
06     try
07     {
08         p = i / k;
09     }
10     catch (DivideByZeroException ex)
11     {
12         Console.WriteLine("例外訊息 : {0}", ex.Message);
13         Console.WriteLine("發生例外的函式 : {0}", ex.StackTrace);
14         Console.WriteLine("發生例外的物件 : {0}", ex.Source);
15         Console.WriteLine("發生例外的物件型別 : {0}", ex.GetType());
16         Console.WriteLine("發生例外的文字說明 : {0}", ex.ToString());
17     }
18     finally
19     {
20         Console.WriteLine(".... 結束程式執行!! ...");
21     }
22     Console.Read();
23 }
```