第 5 章
流程控制

本章提要

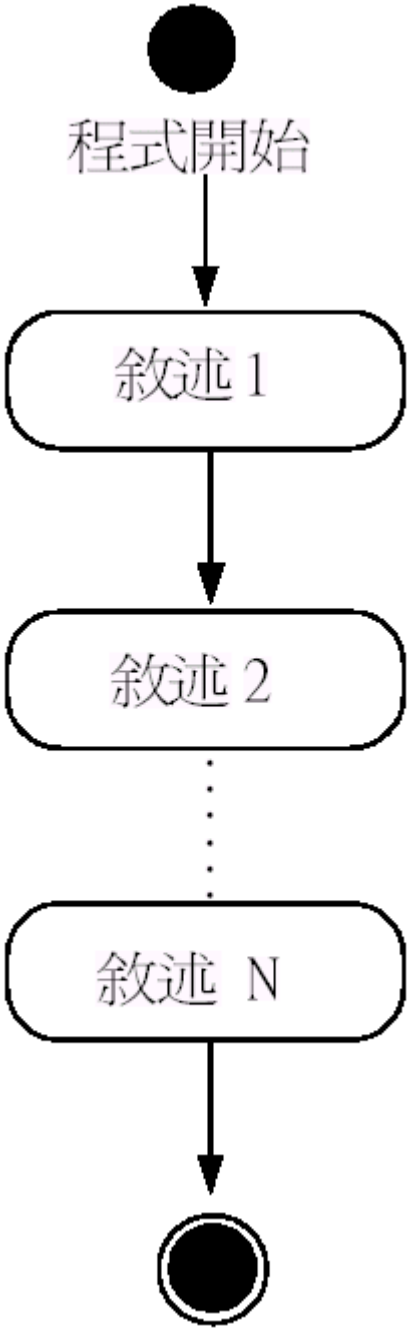
- 5-1 甚麼是流程控制？
- 5-2 if 條件分支
- 5-3 switch 多條件分支
- 5-4 迴圈
- 5-5 變更迴圈流程的 break 與 continue
- 5-6 綜合演練

5-1 甚麼是流程控制？

- 在說明流程控制前，先了解何謂**流程**？以一天的生活為例，**早上起床後，先刷牙洗臉，吃完早餐後就出門上課，上完了早上的三堂課，在餐廳吃自助餐，午休後繼續上下午的課...，晚上回宿舍唸書，最後上床睡覺**，結束一天的流程。程式的執行也是相同的，如同第 2 章曾提及，程式在執行時是以敘述為單元，由上往下循序進行。如下圖：

甚麼是流程控制？

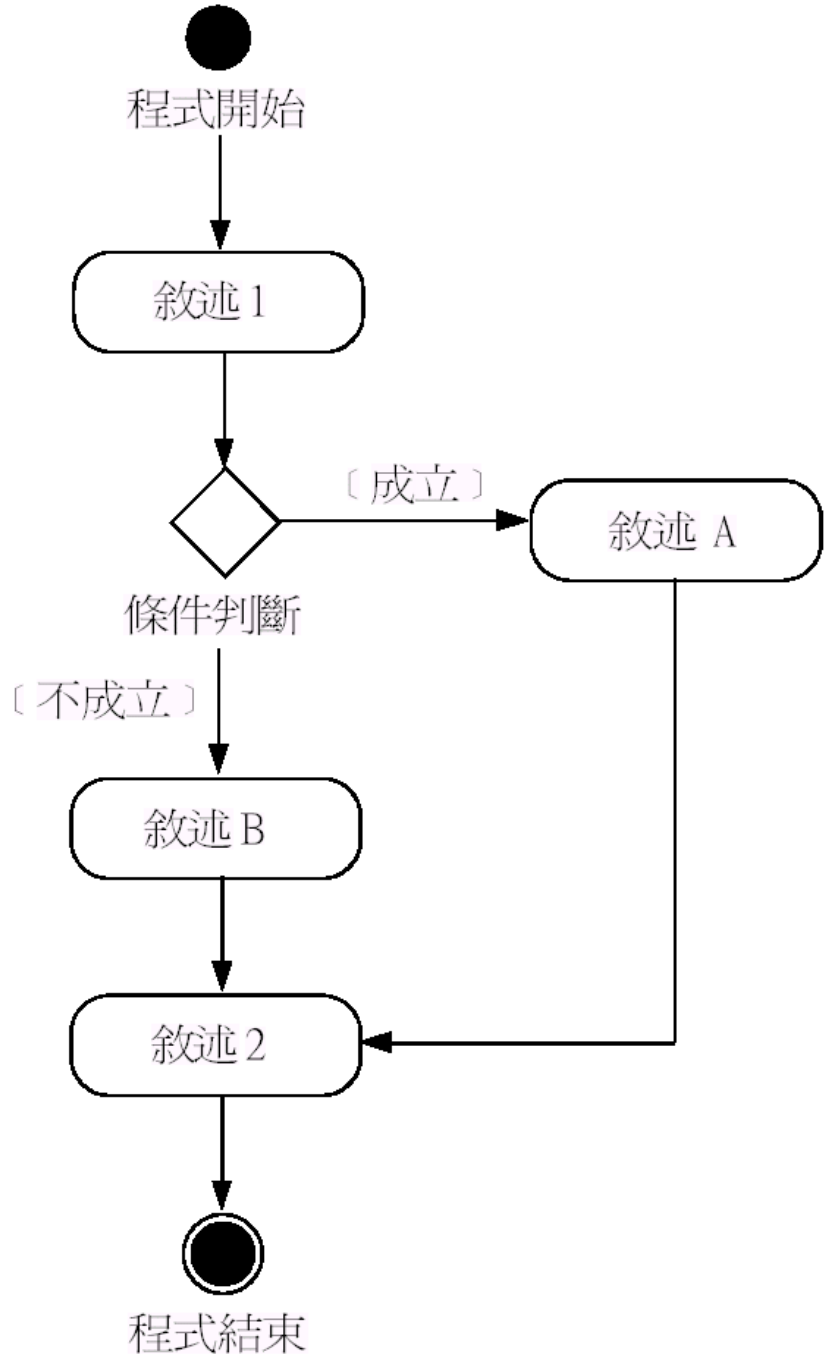
- 由右圖不難發現，程式的執行就如同平常的生活一樣，是有順序性地在執行，整個執行的順序與過程，就是**流程**。



甚麼是流程控制？

- 但是流程並非僅僅依序進行，它可能會因為一些狀況而變化。以一天的生活為例，如果下午老師請假沒來上課，下午的課就會取消，因而更改流程，變成**早上起床後，先刷牙洗臉，... 上完了早上的三堂課，由於下午老師請假，因此決定去學校外面吃午餐，並在市區逛街，下午再回學校打球...**』。程式中的流程也是一樣，可能會因為狀況不同而改變，執行不同的敘述，如下圖：

甚麼是流程控制？



甚麼是流程控制？

- 因此, 對於程式執行的流程順序以及因應不同狀況而選取不同的流程, 即為**流程控制** (flow control)。流程控制可說是電腦程式的靈魂, 它包含：條件判斷、迴圈控制及無條件跳躍三大類：

  1. **條件判斷控制**：判斷條件的真偽, 然後程式依真偽的情形至指定的地方去執行程式。C++ 這方面的敘述有：if-else、switch-case 等 2 種。

甚麼是流程控制？

2. **迴圈控制**：程式依指定的條件做判斷, 若條件成立則進入迴圈執行迴圈內的動作。每執行完一次迴圈內動作便再回頭做一次條件判斷, 直到條件不成立後才結束迴圈, C++ 屬於這方面的流程控制敘述有：for 、while 和 do-while 3 種。
3. **無條件跳躍**：當程式執行到無條件跳躍敘述時, 程式立即依該敘述的指示跳到目的位置執行, 由於無條件跳躍的強制性, 容易使我們無法由程式本身看出其前因後果, 造成閱讀及偵錯的困難, 一般也都盡量不用。

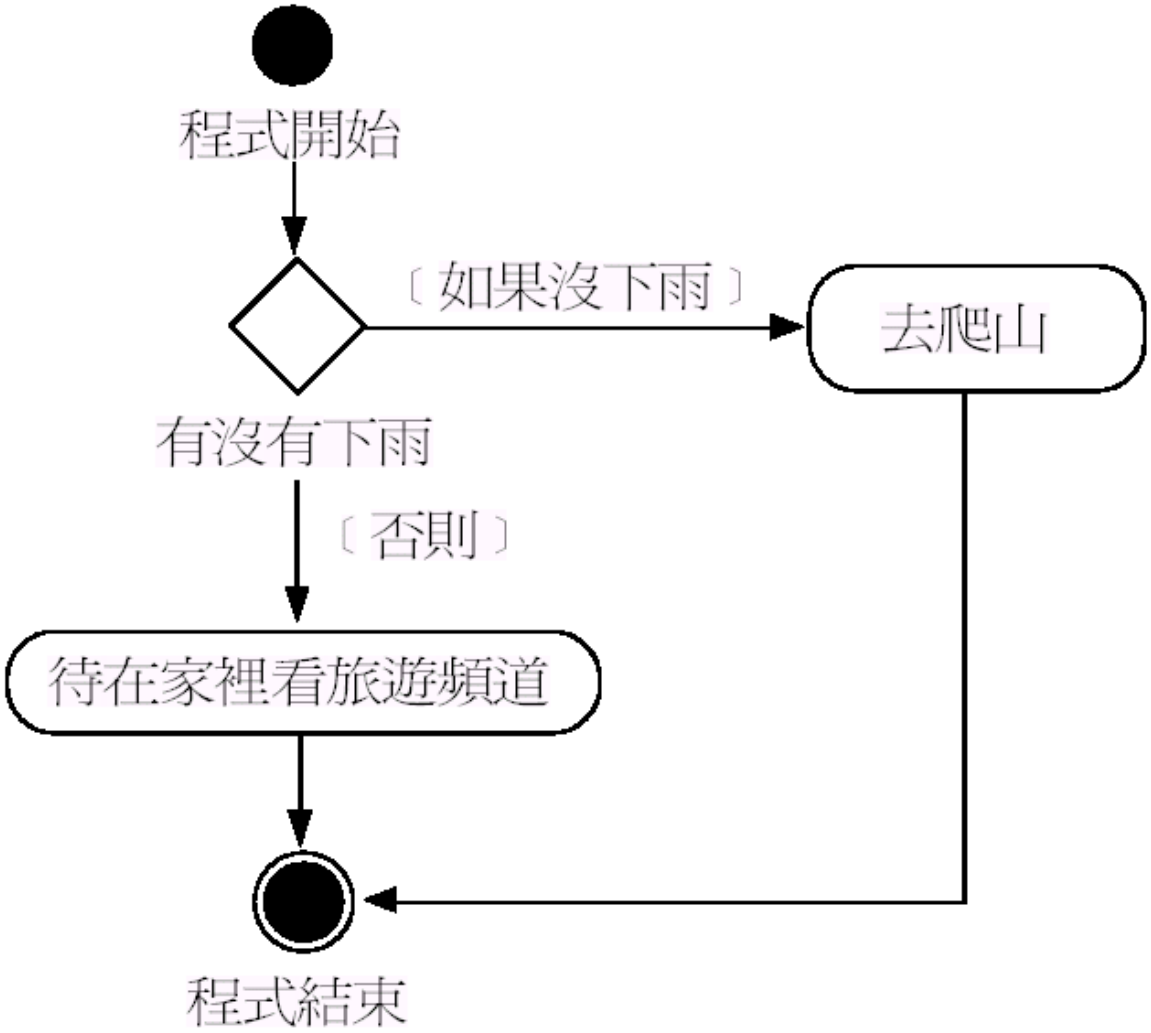
5-2 if 條件分支

- 語法與執行流程
- 多條件運算式與巢狀 if
- 以 else 處理另一種狀況
- 條件運算子

語法與執行流程

- 在條件判斷敘述中，最常用到的就是 if 敘述了，它就如同日常生活中常使用的**如果...就...** 是一樣的意思。比方說**如果明天沒下雨，就去爬山**以圖形來表示就是：

語法與執行流程



語法與執行流程

- 在 C++ 程式中判斷當然不是使用這麼口語的說法, 而是使用 if 敘述來依條件判斷的結果執行對應的程式敘述。if 敘述的語法如下：

```
if (條件運算式)
    敘述                  // 條件運算式爲 true 時要執行的敘述

if (條件運算式) {
    敘述                  // 如果要執行的敘述有好幾行
    ....                  // 可用大括號括起來
}
```

語法與執行流程

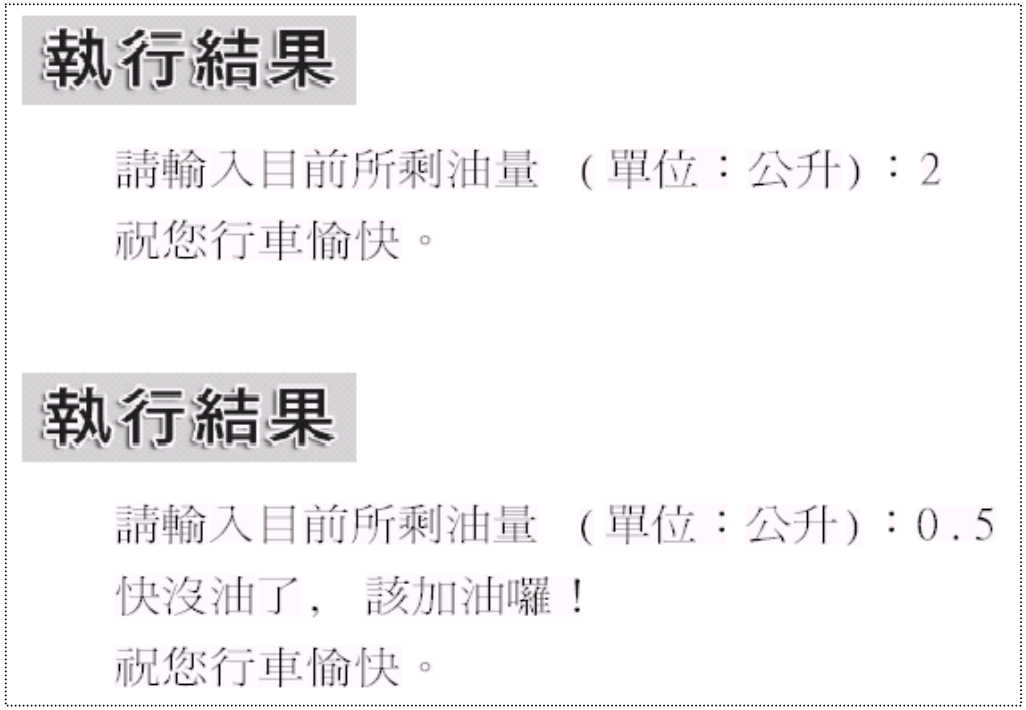
- **if**：**如果**的意思。會根據條件運算式的結果, 來判斷是否執行 { } (稱為區塊, block) 中的程式。如果條件運算式的結果為 true, 則執行區塊內的敘述；如果結果為 false, 則跳過區塊。
- **條件運算式**：通常由比較運算或邏輯運算所組成。
- **敘述**：條件運算式結果為 true 時所要執行的動作。如果只有單一敘述, 則大括號可省略；如果有好幾行敘述, 則需用大括號將它們括起來。

語法與執行流程

程式 Ch05-01.cpp 檢查油量的程式

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   float gasoline;
07   cout << "請輸入目前所剩油量 (單位：公升)：";
08   cin >> gasoline;
09
10   if (gasoline < 1)                    // 如果 gasoline 小於 1
11     cout << "快沒油了, 該加油囉！\n";    // 就執行指定的敘述
12
13   cout << "祝您行車愉快。";
14 }
```

語法與執行流程



- 以生活中常見的例子來說, 使用 if 判斷汽車是否該加油的程式可以寫成這樣：

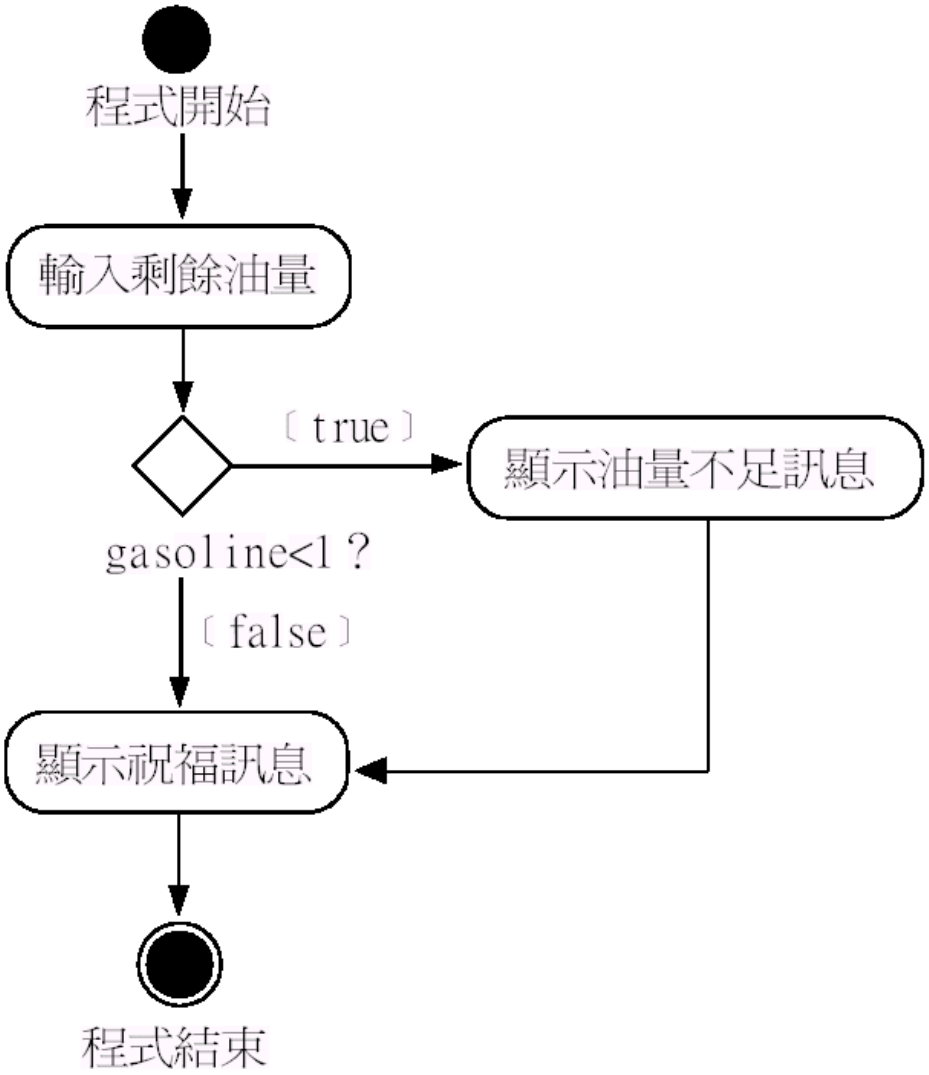
語法與執行流程

1. 第 8 行由鍵盤取得使用者輸入代表目前油量的數值, 並存入變數 gasoline 中。
2. 第 10 行就 是 if 敘述, 它的條件式是 "gasoline < 1" , 也就是判斷 gasoline 的值是否小於 1 若 gasoline 小於 1, 此運算式結果為 true, 此時將會執行第 11 行的敘述；若 gasoline 不小於 1, 此運算式結果為 false, 此時將會跳過第 11 行的敘述, 執行第 13 行的敘述。
3. 第 11 行為條件成立時要執行的敘述, 由於只有一行敘述, 所以未使用大括號。

語法與執行流程

- 在第 1 個執行結果中, 輸入的剩餘汽油量為 2 公升, if 的條件運算式 (gasoline < 1) 的運算結果為 false, 因此第 11 行的敘述不會被執行, 而是直接執行之後的程式。
- 在第 2 個執行結果中, 輸入的剩餘汽油量小於 1 公升, 此 時 if 的條件運算式結果為 true, 因此就會執行第 11 行的敘述, 輸出 " 沒油了, 該加油囉！\n" 這個字串。
- 程式流程如下：

語法與執行流程



語法與執行流程

- 要提醒的是，如果符合條件時所要執行的敘述不只一個，就必須使用一對大括號將這些敘述括起來成為一個區塊，例如：

程式 Ch05-02.cpp 檢查油量的程式

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   float gasoline;
```

語法與執行流程

```
07    cout << "請輸入目前所剩油量 (單位：公升)：";
08    cin >> gasoline;
09
10    if (gasoline < 1) {   // 如果 gasoline 小於 1
11        cout << "快沒油了！\n";
12        cout << "該加油囉！\n";
13    }
14    cout << "祝您行車愉快。";
15 }
```

執行結果

```
請輸入目前所剩油量 (單位：公升)：0.5
快沒油了！
該加油囉！
祝您行車愉快。
```

語法與執行流程

- 如果忘記加上大括號, 而將 if 敘述寫成這樣：

```
if (gasoline < 1)
    cout << "快沒油了！\n";
    cout << "該加油囉！\n";
```

語法與執行流程

- 那麼不管 if 條件式的結果為何, 都一定會執行 **cout << "加油囉！\n";** 這行敘述。這是因為省略括號時, 對 if 敘述而言, 就只有緊接在條件式後面的敘述, 才是條件為 true 時要執行、false 時要跳過的敘述, 此後的敘述都不受 if 條件的影響, 一定都會執行。

多條件運算式與巢狀 if

- 由於 if 是藉由條件運算式的結果來決定是否繼續指定的敘述，而條件運算式通常是由比較運算子以及邏輯運算子所構成。例如前面的例子就是用 < 比較運算子來判斷。如果要判斷的狀況比較複雜，我們可用多個比較運算子或是邏輯運算子來組成條件運算式：

多條件運算式與巢狀 if

程式 Ch05-03.cpp 用多個比較運式組成條件運算式

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   float gasoline;
07   cout << "請輸入目前所剩油量 (單位：公升)：";
08   cin >> gasoline;
09
10   if ((gasoline >= 1) && (gasoline < 5)) // 兩個條件
11       cout << "油量尚足, 但需注意油表！\n";
12
13   cout << "祝您行車愉快。";
14 }
```

多條件運算式與巢狀 if

執行結果

```
請輸入目前所剩油量 （單位：公升）：4
油量尚足， 但需注意油表！
祝您行車愉快。
```

- 其中第 10 行的 **if ((gasoline >= 1) && (gasoline < 5))** 就是使用邏輯運算子 && 將兩個比較運算式結合成條件運算式, 只有在 gasoline 的值大於或等於 1 而且小於 5 的時候運算結果才是 true。

巢狀 if

- 上例是將兩個比較運算式用 && 結合在一起，我們也可以把兩個比較運算式分開，放在連續的 if 敘述，也具有相同的作用：

程式 Ch05-04.cpp 用巢狀 if 設計檢查油量的程式

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    float gasoline;
07    cout << "請輸入目前所剩油量 (單位：公升)：";
```

巢狀 if

```
08    cin >> gasoline;
09
10    if (gasoline >= 1)        // 第 1 個 if
11        if (gasoline < 5)     // 第 2 個 if
12          cout << "油量尚足, 但需注意油表！\n";
13
14    cout << "祝您行車愉快。";
15 }
```

執行結果

```
請輸入目前所剩油量 (單位：公升)：3.5
油量尚足, 但需注意油表！
祝您行車愉快。
```

巢狀 if

- 第 10、11 行兩個 if 的寫法, 又稱為巢狀 if (nested if), 也就是說將 if 敘述包在另一個 if 敘述中執行。如果加上大括號, 就可以看出巢狀 if 敘述的結構：

```
if (...) {
   if (...) // 這個 if 敘述是放在另一個 if 敘述的大括號下執行
      ...
}
```

巢狀 if

- 雖然寫法不同, 但效果相同。第 1 個 if 敘述先檢查油量是否大於等於 1, 若為 true 才會繼續執行第 2 個 if 敘述, 並在滿足其條件 (油量小於 5) 時, 才會執行第 12 行的敘述。

以 else 處理另一種狀況

- if 敘述就像**如果...就...** 的處理方式, 但平常我們也常做另一種形式的選擇：**如果...就..., 要不然就...** 。C++ 當然也提供這種**要不然就...** 的處理方式, 其語法是在 if 敘述後加上 else 子句, 如下：

```
if ( 條件運算式 )
    { 動作 1 }      // 條件式運算式爲 true 時, 執行動作 1
else
    { 動作 2 }      // 條件式運算式爲 false 時, 執行動作 2
```

以 else 處理另一種狀況

- 這個敘述是說, 當條件運算式成立時, 則執行動 作 1, 然後略過 else 的部份 (動作 2), 接著往下執行。當條件式不成立時, 則略過 if 的部份 (動作 1) 而執 行 else 的動作 2, 然後再往下執行。也就是說, 動作 1 與動作 2 只會因條件式的真假由二者選一來執行。

以 else 處理另一種狀況

程式 Ch05-05.cpp 用 if-else 處理不同狀況

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   float height;
07   cout << "請輸入身高 (單位：公分)：";
08   cin >> height;
09
10    if (height > 110)
11       cout << "身高超過標準，請購票上車！\n";
12    else                          // 條件式不成立時才會執行此部份
13       cout << "身高低於標準，可免購票！\n";
```

以 else 處理另一種狀況

```
14
15      cout << "祝旅途愉快。";
16  }
```

執行結果

```
請輸入身高 (單位：公分)：179.5
身高超過標準，請購票上車！
祝旅途愉快。
```

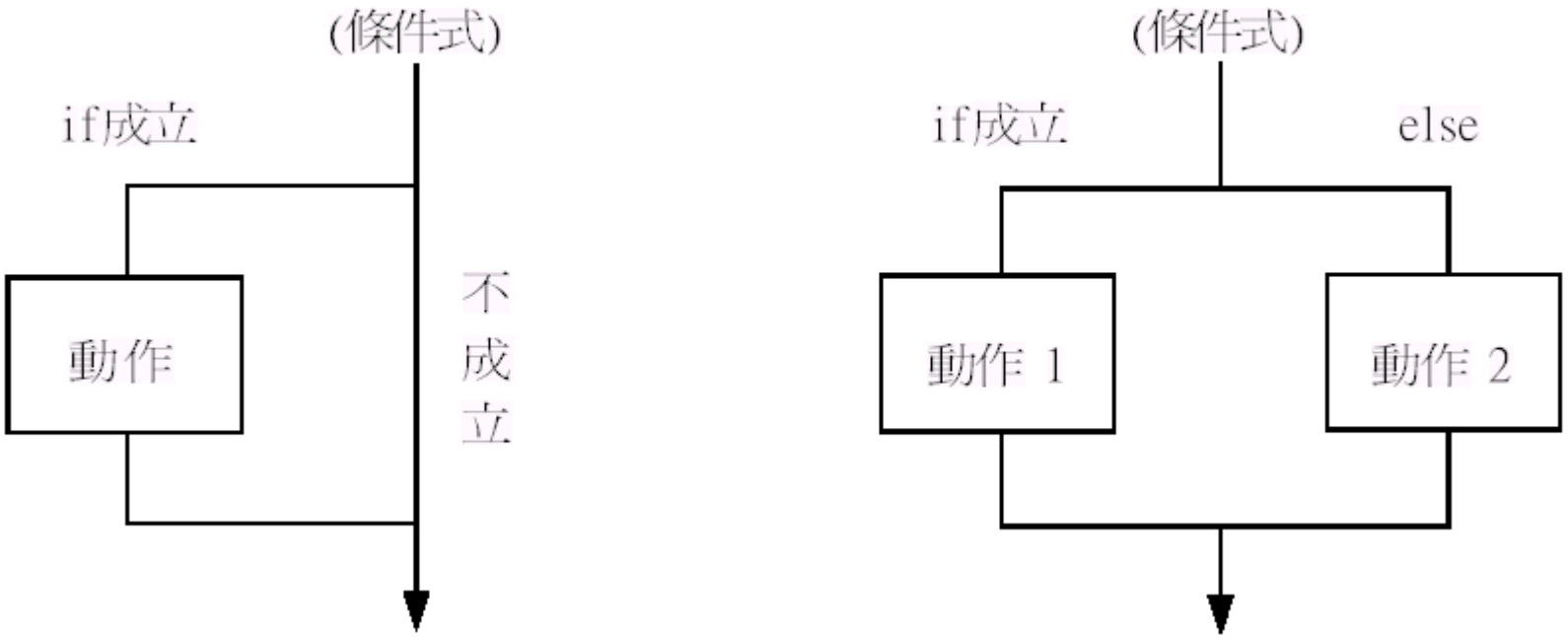
執行結果

```
請輸入身高 (單位：公分)：109
身高低於標準，可免購票！
祝旅途愉快。
```

以 else 處理另一種狀況

- 第 12、13 行就是 else 子句的部份, 當第 10 行 的 if 條件運算式結果為 false 時, 就會執行 else 的敘述。本例中 else 下僅有第 13 行的單一敘述。如果希 望 else 的部份有多行敘述, 也是需要大括號括起來。
- 由下圖可看出 if 與 if-else 的差異所在：

以 else 處理另一種狀況



- 利用 if-else 我們可以讓程式處理 2 種不同的狀況, 如果要處理的狀況有 2 種以上, 可使用 巢狀 的 if-else 來處理。

巢狀的 if-else

- 前面我們看過將 if 敘述放在另一個 if 中的用法, 而 if-else 當然也可以放在另一個 if 或 if-else 之中, 形成巢狀的 if-else。例如我們可以在 if 及 else 下分別再放另外一組 if-else, 如此共可處理 4 種不同的狀況, 例如：

巢狀的 if-else

```
if (a > 0)   // 這個 if 下有一對 if-else 敘述
   if (b > 0)
      cout << "a>0 and b>0";
   else
      cout << "a>0 and b<=0";
else          // 這個 else 下也有一對 if-else 敘述
   if (b > 0)
     cout << "a<=0 and b>0";
   else
     cout << "a<=0 and b<=0";
```

巢狀的 if-else

- 這種寫法是 以 if-else 來當成 if 及 else 的動作部份, 但有時會讓看程式的人搞不清楚各 else 究竟是對應到哪一個 if 敘述。此時要掌握的閱讀原則是：**else 要與前面最接近它, 且尚未與別的 else 配對 的 if 配成一對**。如果怕無法分辨清楚, 可在適當位置加上大括號幫助閱讀, 或改變配對的方式, 例如下面這段程式：

巢狀的 if-else

```
if (a > 0)
   if (b > 0)
      cout << "a>0 and b>0";
else      // 這個 else 其實是在 a>0 且 b<=0 時會被執行
   cout << "a<=0";
```

- 上述的寫法乍看之下, else 好像是與第一個 if 配對, 也就是當 a>0 為 false 時會被執行。其實不然, 這個 else 事實上是與第二個 if 配對 (與最接近且未配對的 if) 配對, 換言之, 它應是在 a>0 成立, 且 b>0 為 false 時才會被執行。

巢狀的 if-else

因此, 如果希望這個 else 是在 "a<=0" (也就是 a>0 為 false) 時, 應在第 1 個 if 下加上一對大括號, 以使 else 和第一個 if 配對：

```
if (a > 0) {
   if (b > 0)
      cout << "a>0 and b>0";
}
else   // 與第 1 個 if 配對, 處理 "a<=0" 的狀況
    cout << "a<=0";
```

巢狀的 if-else

- 巢狀 if-else 還有另一種較常見的用法, 就是連續判斷式的 if-else, 也就是只在 else 的部份放進巢狀的 if-else, 而且視狀況的多寡, 可一直在巢狀的 if-else 下再加一個巢狀的 if-else 而通常我們會把它寫成 if....else if...else if..., 如下所示：

```
if (條件式1)
    { 動作 A }          // 條件式 1 成立時
else if (條件式2)
    { 動作 B }          // 條件式 2 成立時
```

巢狀的 if-else

```
else if (條件式3)
    { 動作 C }          // 條件式 3 成立時
else
    { 動作 D }          // 其它狀況
```

- 這種寫法是以 if-else 來取代原來的 else 動作部份, 也可稱之為過濾式或多重選擇式的 if-else 寫法。例如同樣以購買車票, 若除了全票及免票外, 又多一種半票, 則程式可寫成：

巢狀的 if-else

程式 Ch05-06.cpp 用 if-else 處理不同狀況

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06     float height;
07     cout << "請輸入身高 (單位：公分)：";
08     cin >> height;
09
10     if (height < 110)
11        cout << "身高低於標準，可免購票！\n";
12     else if (height < 140) // 身高在 110 及 140 間的狀況
13        cout << "身高超過 110，請買半票！\n";
```

巢狀的 if-else

```
14      else                                    // 身高超過 140 的狀況
15          cout << "身高超過 140，請買全票！\n";
16
17      cout << "祝旅途愉快。";
18  }
```

執行結果

```
請輸入身高 (單位：公分)：139.5
身高超過 110，請買半票！
祝旅途愉快。
```

巢狀的 if-else

1. 第 12 行的 else if 會在 height 大於等於 110 小於 140 的狀況下執行。
2. 第 14 行的 else 會在前面所有 if 都不成立的情況下, 也就是 height 大於等於 140 時執行。

- 這種 if...else if...else if... 的寫法可針對多種不同條件進行處理, 而如果所有的條件運算式要判斷的都是同一運算式 (變數) 是否為某數值, 則可用 5-3 節介紹的 switch...case 敘述來處理。

條件運算子

- 除了用 if...else 敘述來處理 2 種不同的狀況外, C++ 也提供一個較特別的**條件運算子**, 適用於不同條件的處理動作, 都只是一簡單的運算式的情況。條件運算子是由問號及冒號 2 個符號所構成, 其語法如下：

```
(條件運算式) ? (運算式 1) : (運算式 2)
```

條件運算子

- 當條件運算式為 true 時, 就會執行 (運算式 1)；反之若為 false, 則執行 (運算式 2)。至於整個運算式的運算結果, 則為 (運算式 1) 或 (運算式 2) 的結果。
- 舉例來說, 若某遊樂園入場券每人 100 元, 但 10 人及以上可享 8 折優惠, 則要依購票數計算總票價時, 即可應用條件運算子設計成如下的程式：

條件運算子

程式 Ch05-07.cpp 用條件運算子處理不同狀況

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06   double fee = 100;          // 票價 100 元
07   int ticket;
08   cout << "要買幾張票？";
09   cin >> ticket;
10
11   fee *= (ticket <10) ? (ticket) : (ticket*0.8);
12   cout << "您要購買 " << ticket << " 張票" << endl
```

條件運算子

```
13              << "共計 " << fee << " 元";
14  }
```

執行結果

```
要買幾張票？9
您要購買 9 張票
共計 900 元
```

執行結果

```
要買幾張票？18
您要購買 18 張票
共計 1440 元
```

條件運算子

1. 第 6、7 行分別宣告票價變數 fee 及購票張數變數 ticket。由於稍後計算時會用到小數, 所以將 fee 宣告為 double 型別, 以免編譯時出現警告。
2. 第 11 行即使用條件運算子的運算式, 我們讓 fee 乘上由條件運算子構成的運算式。而條件運算子會在 ticket 小於 10 的情況, 傳回 ticket 的值；否則傳回 (ticket*0.8)。這 1 行若改用 if...else 語法, 則可寫成：

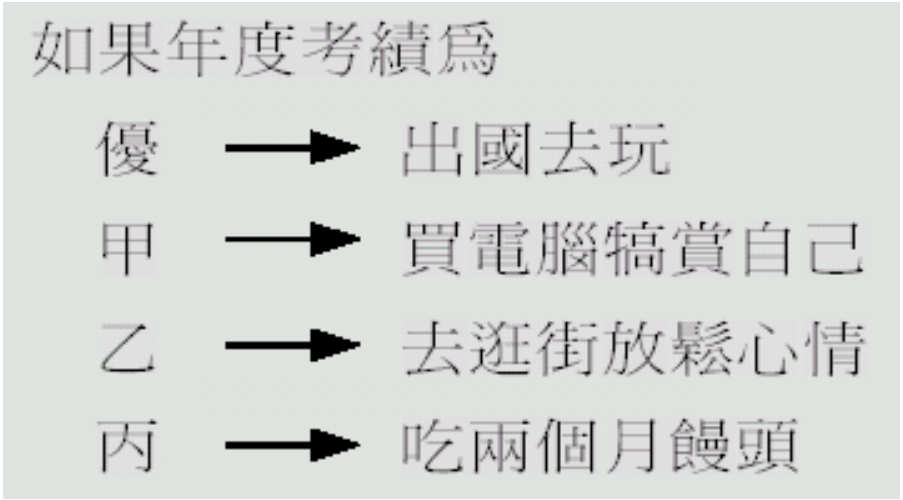
條件運算子

```
if (ticket < 10)    // 購票數未滿十張
   fee *= ticket;
else                // 購票數滿十張以上
   fee *= ticket * 0.8;
```

- 對本例而言，使用條件運算子的確使程式精簡不少。但如果不同條件下要處理的動作較複雜，就不適合使用條件運算子，仍是使用 if/else 為宜。

5-3 switch 多條件分支

- switch 是一種多選一的敘述。舉個例子來說, 在本年度初, 我們對自己訂了幾個目標, 如果年度考績拿到優, 就出國去玩；如果拿到甲, 就買台電腦犒賞自己；拿到乙, 就去逛個街放鬆一下；如果考績是丙, 就要罰自己吃兩個月饅頭：



switch
多條件分支

- switch 多條件分支的用法與上述的情況十分類似, 是由一個條件運算式的值來決定應執行的對應敘述, 語法如右：

```
switch (條件運算式) {
   case 條件值 1:
      敘述 1
      // 其他敘述...
      break;
   case 條件值 2:
      敘述 2
      // 其他敘述...
      break;
      .
      .
      .
   case 條件值 N:
      敘述 N
      // 其他敘述...
      break;
}
```

switch 多條件分支

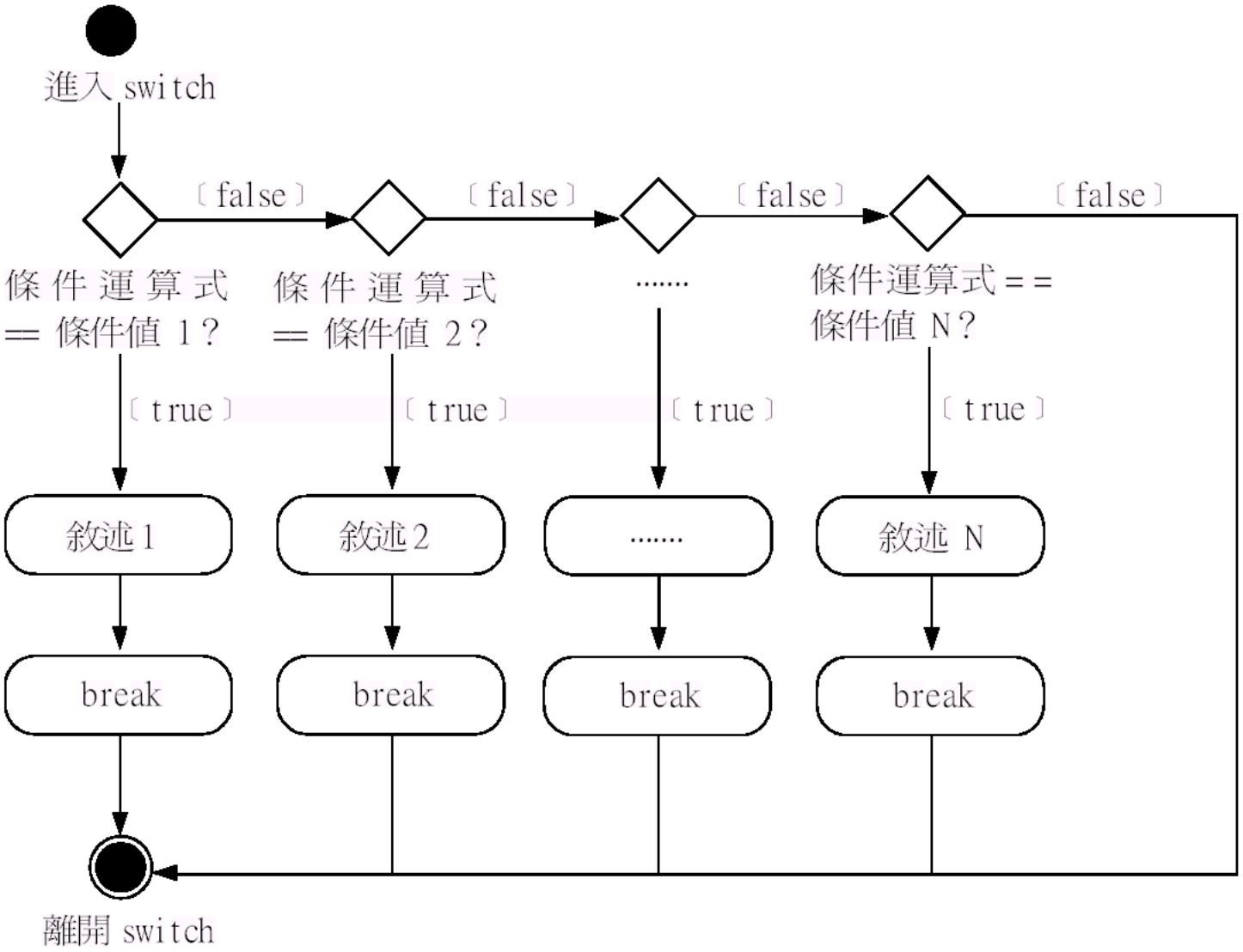
- **switch**：**選擇..** 的意思, 表示要根據條件運算式的結果, 選擇接下來要執行哪一個 case 內的動作。switch 後的條件運算式其運算結果必須是整數或字元 (也算是整數), 否則編譯時會出現錯誤。
- **case**：列出個別的條件值, case 之後的條件值必須是常數或是由常數所構成的運算式, 且不同 case 的條件值運算結果不能相同。switch 會根據條件運算式的運算結果, 從各個 case 中挑選相同的條件值, 並執行其下所列的敘述。

switch 多條件分支

- **break**：結束這一段 case 的處理動作。

- 雖然 switch 表面上看起來跟使用判斷式的 if 條件運算式完全不同, 但是 switch 私底下仍然是使用判斷式的真假來作為其控制流程的機制。如下圖：

switch 多條件分支



switch
多條件分支

- 根據上述 switch 語法，年度考績的例子就可以寫成如右的 switch 程式片段：

```
switch (年度考績) {
        case 'A': // 代表 '優'
                出國去玩
                break;
        case 'B': // 代表 '甲'
                買台電腦犒賞自己
                break;
        case 'C': // 代表 '乙'
                去逛街放鬆心情
                break;
        case 'D': // 代表 '丙'
                吃兩個月饅頭
                break;
}
```

switch 多條件分支

程式 Ch05-08.cpp 可依輸入顯示不同季節應穿著的服裝

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int season;
07    cout << "請選擇季節：1.春 2.夏 3.秋 4.冬：";
08    cin >> season;
09
10    switch (season)
11    {
12       case 1:   // 當 season 的數值爲 1
```

switch 多條件分支

```
13          cout << "請穿著長袖出門";
14          break; // 結束此 case
15        case 2:  // 當 season 的數值爲 2
16          cout << "請穿著短袖出門";
17          break; // 結束此 case
18        case 3:  // 當 season 的數值爲 3
19          cout << "請加件長袖輕薄外套出門";
20          break; // 結束此 case
21        case 4:  // 當 season 的數值爲 4
22          cout << "請穿著毛衣或大衣出門";
23          break; // 結束此 case
24      }
25  }
```

switch 多條件分支

執行結果

```
請選擇季節：1.春 2.夏 3.秋 4.冬：1
請穿著長袖出門
```

執行結果

```
請選擇季節：1.春 2.夏 3.秋 4.冬：3
請加件長袖輕薄外套出門
```

switch 多條件分支

- 如上的程式範例, 會根據使用者輸入數字的不同, 顯示不同的訊息：
  1. 第 6～8 行先顯示訊息請使用者選擇季節, 並且將使用者輸入代表季節的數字指定給 season 變數。
  2. 第 10～24 行即是 switch 敘述的區塊, 在 第 10 行中的 switch 括號內放的是 season 變數, 也就表示此 switch 敘述是依 season 的值, 去執行對應 case 內的敘述, 也就是顯示適合該季節穿著的訊息。

switch 多條件分支

3. 第 12 行的 "case 1" , 即表示 season 的值為 1 時, 要執行以下的敘述, 此時將會先執行第 13 行的 cout 敘述輸出訊息, 接著執行第 14 行的 break 敘述, 表示結束此 case 的處理, 也就是結束此 switch 區塊。如果程式在 switch 後面還有其它敘述, 程式就會跳到該處 (第 24 行後) 繼續執行。

break 敘述的重要性

- 前面提過 break 是用來結束單一個 case, 如果不加上 break, 程式也能執行, 但此時程式的執行方式是接著往下一個 case 對應的敘述繼續執行。例如我們把前一個範例程式的第 1 個 break 拿掉：

break 敘述的重要性

程式 Ch05-09.cpp 遺漏 break 會影響執行結果

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int season;
07    cout << "請選擇季節：1.春 2.夏 3.秋 4.冬：";
08    cin >> season;
09
```

break 敘述的重要性

```
10    switch (season) {
11      case 1:   // 當 season 的數值爲 1
12        cout << "請穿著長袖出門";
13        // break; 故意把這行標示成註解, 觀察沒有 break 的情況
14      case 2:   // 當 season 的數值爲 2
15        cout << "請穿著短袖出門";
16        break; // 結束此 case
17      case 3:   // 當 season 的數值爲 3
18        cout << "請加件長袖輕薄外套出門";
19        break; // 結束此 case
20      case 4:   // 當 season 的數值爲 4
21        cout << "請穿著毛衣或大衣出門";
22        break; // 結束此 case
23    }
24  }
```

break 敘述的重要性

執行結果

請選擇季節：1.春 2.夏 3.秋 4.冬：1
請穿著長袖出門請穿著短袖出門

執行結果

請選擇季節：1.春 2.夏 3.秋 4.冬：2
請穿著短袖出門

break 敘述的重要性

- 由於 case 1 的敘述中沒有 break, 因此, 當使用者輸入 1 時, 程式就會進入第 11 行開始執行, 但因為沒有 break 敘述, 所以又會繼續執行第 15 行的程式, 也就是執行到 "case 2" 的部份, 直到遇到第 16 行的 break 才跳出, 造成選擇春天時程式輸出錯亂的情況。
- 雖然遺漏 break 可能會讓程式執行的結果不正確, 但有些情況下, 我們會故意讓不同的 case 執行相同的敘述, 此時就需故意省掉 break 以精簡程式碼。請看以下這個範例：

break 敘述的重要性

程式 Ch05-10.cpp 移除 break 敘述，讓 case 共用程式碼

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int choice;
07    cout << "我們有四種餐, 請選擇：\n";
08    cout << "1.炸雞餐 2.漢堡餐 3.起司堡餐 4.薯條餐：";
09    cin >> choice;
10
11    switch (choice) {
```

break 敘述的重要性

```
12        case 1: // 炸雞餐價錢 109 元
13            cout << "您點的餐點價錢爲 109 元";
14            break;
15        case 2: // 漢堡餐和起司堡餐同價
16        case 3: // 起司堡餐價錢爲 99 元
17            cout << "您點的餐點價錢爲 99 元";
18            break;
19        case 4: // 薯條餐價錢爲 69 元
20            cout << "您點的餐點價錢爲 69 元";
21            break;
22    }
23 }
```

break 敘述的重要性

執行結果

我們有四種餐， 請選擇：
1.炸雞餐 2.漢堡餐 3.起司堡餐 4.薯條餐：2
您點的餐點價錢爲 99 元

執行結果

我們有四種餐， 請選擇：
1.炸雞餐 2.漢堡餐 3.起司堡餐 4.薯條餐：3
您點的餐點價錢爲 99 元

break 敘述的重要性

- 由於漢堡餐及起司堡餐的價錢一樣, 因此我們故意將 "case 2" 及 "case 3" 放在一起, 使用者輸入 2 號餐及 3 號餐時, switch 都會執行到第 17 行的敘述。如果不這樣寫, 就得在第 15 行之下, 也插入和 17、18 行一樣的敘述, 程式才會有相同的效果。

捕捉其餘狀況的 default

- 在 switch 內還可以加上一個 default 項目，用來捕捉條件運算式與所有 case 條件值都不相符的狀況，就像是當 if 條件不成立時會進入 else 的部份執行一樣。語法如下：

捕捉其餘狀況的 default

```
switch (條件運算式) {
   case 條件值 1:
      ...
      break;
   case 條件值 2:
      ...
      break;
   .
   .
   .
   case 條件值 N:
      ...
      break;
   default:
      ...
    // 因爲後面已沒有 case 了, 所以不用加 break 敘述
}
```

捕捉其餘狀況的 default

- 以之前的點餐程式為例, 如果使用者輸入了非 1～4 的數值, switch 敘述找不到相符的 case, 只會簡單的跳出 switch 敘述, 因此我們可加上 default 來處理這樣的狀況：

程式 Ch05-11.cpp 使用 default 處理未點餐選項的情況

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
```

捕捉其餘狀況的 default

```
06    int choice;
07    cout << "我們有四種餐, 請選擇：\n";
08    cout << "1.炸雞餐 2.漢堡餐 3.起司堡餐 4.薯條餐：";
09    cin >> choice;
10
11    switch (choice) {
12      case 1: // 炸雞餐價錢 109 元
13        cout << "您點的餐點價錢爲 109 元";
14        break;
15      case 2: // 漢堡餐和起司堡餐同價
16      case 3: // 起司堡餐價錢爲 99 元
17        cout << "您點的餐點價錢爲 99 元";
18        break;
19      case 4: // 薯條餐價錢爲 69 元
20        cout << "您點的餐點價錢爲 69 元";
```

捕捉其餘狀況的 default

```
21            break;
22        default:
23            cout  <<  "您並未輸入正確的號碼！";
24    }
25 }
```

執行結果

```
我們有四種餐，  請選擇：
1.炸雞餐  2.漢堡餐  3.起司堡餐  4.薯條餐：5  ←輸入 1～4 以外的值
您並未輸入正確的號碼！
```

捕捉其餘狀況的 default

- 第 22 行加入 default 的項目, 這樣一來, 當使用者輸入的數值不符合所有 case 的條件值 (1～4), switch 就會選擇執 行 default 的部份, 在本範例就是顯示訊息告知使用者輸入不正確。

5-4 迴圈

- 迴圈是一種能快速解決重複性工作 (重複的執行動作) 的敘述。在日常生活中，我們都會遇到一些例行性 (routine) 的工作：辦公人員每天重複的收發表格、操作員重複地把原料放到機器上等。這種重複性的工作即使是在寫程式時也很容易發生，此時我們就需要利用迴圈來解決此類程式問題。

迴圈

- 簡單的說, 迴圈就是用來執行需要重複執行的敘述。例如我們要用程式計算從 1 加到 100, 而且每次加完都要顯示目前的累加成果。此時我們不必寫 100 行做加法運算的敘述、100 行 cout 敘述, 只需將 2 行敘述放在迴圈中, 讓迴圈執行 100 次即可。

for 迴圈

- for 迴圈就是適用在我們已明確知道迴圈要執行幾次的場合, 例如剛才提到要將 1 到 100 的數字相加, 顯然就是要加 100 次, 所以就適合用 for 迴圈。for 迴圈的語法如下：

```
for (初始運算式; 條件運算式; 控制運算式) {
    迴圈動作敘述;
}
```

for 迴圈

- **初始運算式**：在第一次進入迴圈時, 會先執行此處的運算式。在一般的情況下, 我們都是在此設定**條件運算式**中會用到的變數之初始值。
- **條件運算式**：用來判斷是否應執行迴圈中的動作敘述, 傳回值需為布林值。此條件運算式會在**每次迴圈開始時**執行, 以重新檢查讓迴圈執行的條件是否仍成立。比方說, 若條件運算式為 "i < 10", 那麼只有在 i 小於 10 的情形下, 才會執行迴圈內的動作敘述；一旦 i 大於或等於 10, 迴圈便結束。

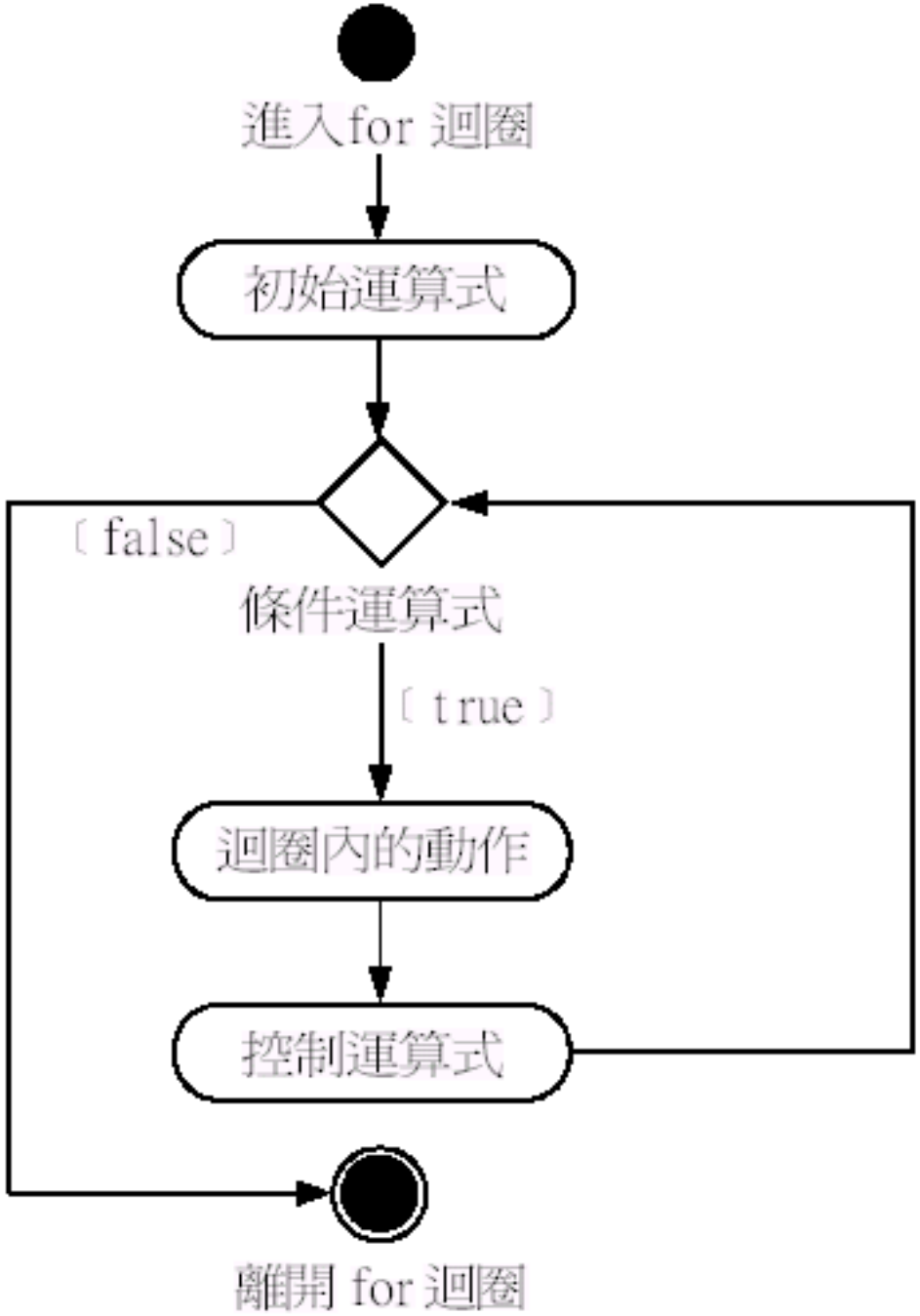
for 迴圈

- **控制運算式**：每次執行完 for 迴圈中的動作後, 就會先執行此運算式, 我們通常都是用它來調整條件運算式中會用到的變數值。以上例來說, 在條件運算式為 i < 10 的情況下, 要想讓此迴圈可以執行十次, 就會利用控制運算式來改變 i 值, 例如 i++, 讓 i 以每次加 1 的方式不斷累加, 等到 i 的值加到 10 的時候, 條件運算式 i < 10 的結果即為 false, 此時迴圈結束, 也完成我們想要迴圈執行 10 次的目標。

for 迴圈

- **迴圈動作敘述：**將您希望利用迴圈重複執行的敘述放在此處，如果要執行的敘述只有一個，也可省略前後的大括號{}。

- 整個 for 迴圈的執行流程圖如右：



for 迴圈

- 前面提到一個計算 1 到 100 相加並輸出過程中所有計算結果的例子, 我們就來看如何用迴圈解決這個問題：

程式 Ch05-12.cpp 用迴圈計算 1 加到 100

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int sum = 0;              // 儲存總和的變數
```

for 迴圈

```
07
08      for (int i=1; i<=100; i++) {
09          sum += i;              // 將 sum 加上目前的 i 值
10          cout << "1 加到 " << i << " 的總和爲 " << sum << endl;
11      }
12  }
```

執行結果

```
1 加到 1 的總和等於 1
1 加到 2 的總和等於 3
1 加到 3 的總和等於 6
...
1 加到 99 的總和等於 4950
1 加到 100 的總和等於 5050
```

for 迴圈

1. 第 6 行宣告的變數 sum, 稍後即用它來計算及儲存 1 到 100 相加的總和。
2. 第 8 行即為 for 迴圈敘述。在初始運算式中宣告了整數變 數 i, 並設定初始值為 1；條件運算式則是設定當 i 的值小於等於 100 時才會執行迴圈；控制運算式則設定每次執行完迴圈的動作時, 就將 i 的值遞增。
3. 第 9～10 行大括號中的敘述, 就是每一輪迴圈要執行的動作。 第 9 行就是將 sum 的值加上目前變數 i 的值, 第 10 行則是輸出目前的累加結果。

for 迴圈

- 請注意, 在第 8 行 for 敘述中宣告的變數 i, 因為是在 for 敘述開始時才宣告的, 所以只能用在 for 迴圈之中 (大括號內), 若離開 for 迴圈後, 程式仍使用這個變數, 將會出現編譯錯誤。因此若想在迴圈後仍使用變數 i, 則需在 for 敘述之前就先宣告該變數。
- 我們就以 for (int i=1;i<=100;i++) 這個迴圈為例, 來看看 for 迴圈的執行步驟：

for 迴圈

第 1 次：i 值爲 1 → i<=100 成立 → 執行迴圈內動作 → i++ (i 變成 2)
第 2 次：i 值爲 2 → i<=100 成立 → 執行迴圈內動作 → i++ (i 變成 3)
.
.
.
第 100 次：i 值爲 100 → i<=100 成立 → 執行迴圈內動作 → i++ (i 變成 101)
第 101 次：i 值爲 101 → i=<100 不成立 → 跳出迴圈

- 由上述例子可知, 我們只要善用迴圈的條件運算式及控制運算式, 就可以控制迴圈的執行次數。由此亦可得知, 如果迴圈執行的次數可以預先判斷出來, 使用 for 迴圈將是最好的選擇。

while 迴圈

- while 迴圈的結構和前面介紹的 if 條件判斷式看起來很類似, 兩者都有個用括號括住的條件判斷式, 加上一組由大括號括住的的動作敘述。
- while 迴圈有別於 for 迴圈的地方在於, while 迴圈不需要初始運算式及控制運算式, 只需要條件運算式即可。如下述：

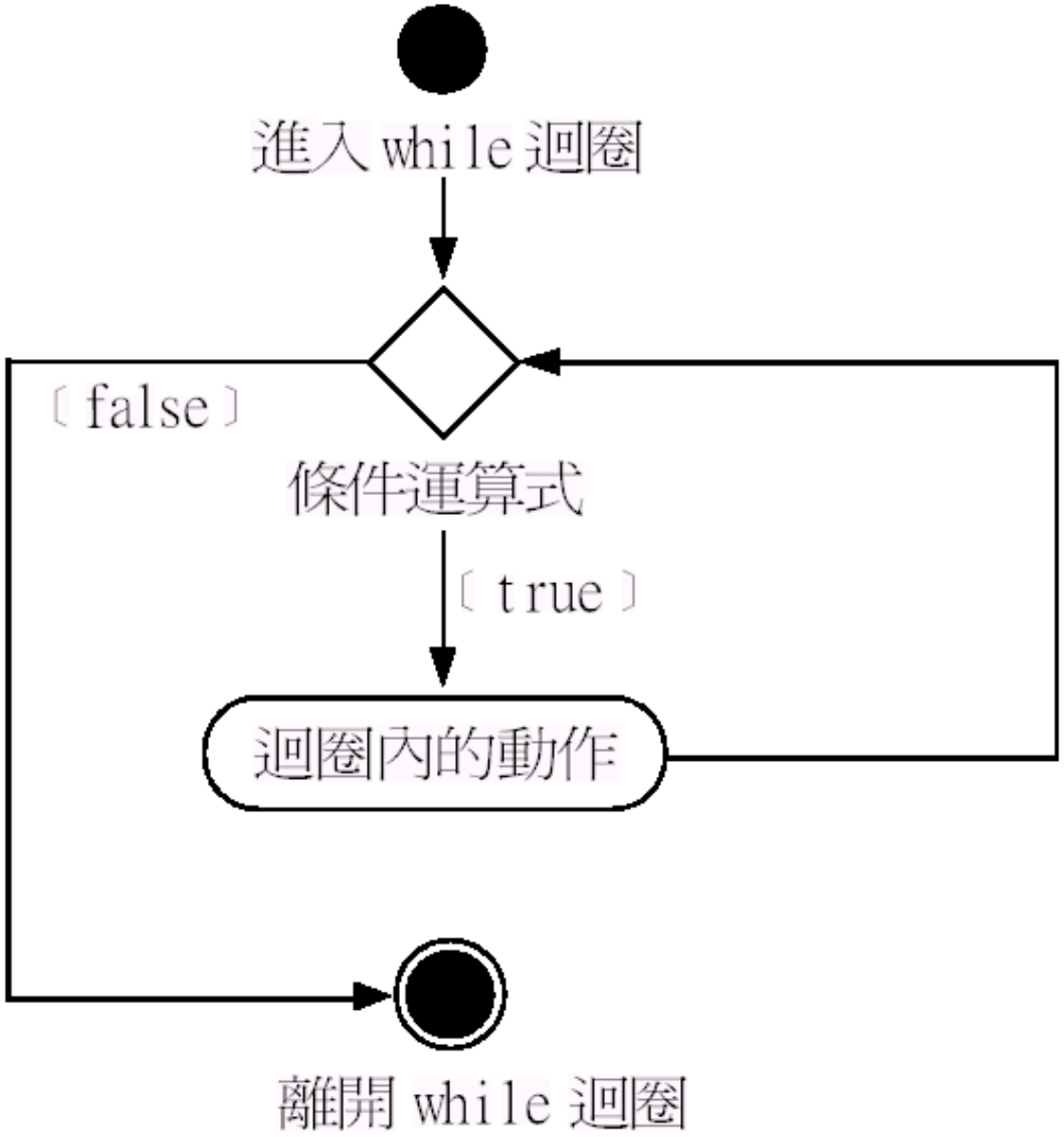
```
while (條件運算式) {
    迴圈動作敘述
}
```

while 迴圈

- **while**：**當..** 的意思。會根據條件運算式的真假, 來決定是否執行迴圈內的動作。也就是說當條件運算式結果為真 (true), 就執行迴圈內的動作；若為假, 則不執行括號內的敘述 (跳出迴圈)。
- **條件運算式**：可以為任何運算式、變數或數值。如果結果為真或非 0 的數值, 則表示為 true, 反之則為 false。
- **迴圈動作敘述**：要用迴圈重複執行的敘述, 如果要重複執行的敘述只有一個, 則可省略大括號。

while 迴圈

- while 迴圈的執行流程如右圖所示，由於只有條件運算式，因此執行流程較 for 迴圈簡潔：



while 迴圈

- 雖然 while 敘述中只有條件運算式, 但只要經過設計, 它仍能適用於會執行特定次數的迴圈, 但它更適合用於執行次數不定的狀況。舉例來說, 如果我們要讓程式持續執行某項動作, 直到使用者說不再執行才停止, 由於我們無法預期使用者想執行該動作幾次, 所以屬於執行次數不定的情況。

while 迴圈

- 例如以下就是用 while 迴圈所設計的程式，程式會根據使用者輸入的身高、體重計算體脂率，且可一再重複請使用者輸入另一組數據來計算，直到使用者不想再輸入為止：

程式 Ch05-13.cpp 利用 while 迴圈讓使用者可重複計算體脂率

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
```

while 迴圈

```
06    char go_again = 'Y';
07    float height, weight;
08
09    while (go_again == 'Y' || go_again == 'y') { // 大小寫 Y 都會
10    cout << "請輸入身高 (公分)：";                    // 繼續迴圈
11    cin >> height;
12    cout << "請輸入體重 (公斤)：";
13    cin >> weight;
14    cout << "您的體脂率爲：" << weight / (height * height) * 10000
15          << '%' << endl;          // 體脂率爲體重除以身高 (公尺) 平方
16    cout << "要繼續計算另一位嗎？(要請輸入 Y 或 y )：";
17    cin >> go_again;
18    }
19 }
```

while 迴圈

執行結果

```
請輸入身高 (公分)：170
請輸入體重 (公斤)：80
您的體脂率爲：27.6817%
要繼續計算另一位嗎？(要請輸入 Y 或 y )：y
請輸入身高 (公分)：160
請輸入體重 (公斤)：50
您的體脂率爲：19.5313%
要繼續計算另一位嗎？(要請輸入 Y 或 y )：n
```

while 迴圈

- 第 9 行的 while 敘述中用以判斷是否執行迴圈的條件運算式, 是判斷字元變數 go_again 是否為大寫或小寫的 'Y' 此變數的初始值設為 'Y', 所以第 1 次一定會進入迴圈執行。但執行到第 17 行時則是由使用者輸入新的 go_again 值, 若使用者輸入大寫或小寫的 'Y', 迴圈才會繼續執行；否則將跳出迴圈並結束程式 (因為迴圈後已無其它程式)。

while 迴圈

- 除了這類由使用者控制執行次數的情況外，我們也常會遇到另一種執行次數不確定的狀況，像是許多特定的計算，都是依參與計算的數字來決定重複計算次數。舉例來說，我們在小時候求最大公因數所用的輾轉相除法，輾轉相除的次數並非一定，而是由用來算最大公因數的 2 個數值所決定。以 while 迴圈設計輾轉相除法的程式，可以寫成下面這個樣子：

while 迴圈

程式 Ch05-14.cpp 使用 while 迴圈處理輾轉相除法求最大公因數

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int num1,num2;
07    cout << "計算兩整數的最大公因數\n";
08    cout << "請輸入第1個數字：";
09    cin >> num1;
10    cout << "請輸入第2個數字：";
11    cin >> num2;
12
13    int a, b = num2, c=num1%num2;    // c 的值就是第 1 次相除的餘數
14
```

while 迴圈

```
15    while (c!=0) {                    // 當餘數爲 0 時,
16       a=b;                           // b 就是最大公因數
17       b=c;
18       c=a%b;                         // 輾轉相『除』, 取餘數
19    }
20
21    cout << num1 << " 和 " << num2 << " 的最大公因數是：" << b;
22 }
```

執行結果

```
計算兩整數的最大公因數
請輸入第 1 個數字：246
請輸入第 2 個數字：135
246 和 135 的最大公因數是：3
```

while 迴圈

- 輾轉相除法的計算方式, 就是將要求最大公因數的數值相除, 然後再用餘數繼續參與相除的計算, 當餘數為 0 時, 則最後一次計算的除數就是最大公因數。程式中第 13 行即是先做第 1 次除法計算, 若餘數不為 0, while 迴圈就會繼續進行計算。迴圈內的第 16、17 行是在做下次除法運算前, 先將前次計算的除數設為被除數、餘數設定為除數；18 行才是做取餘數的運算。

do/while 迴圈

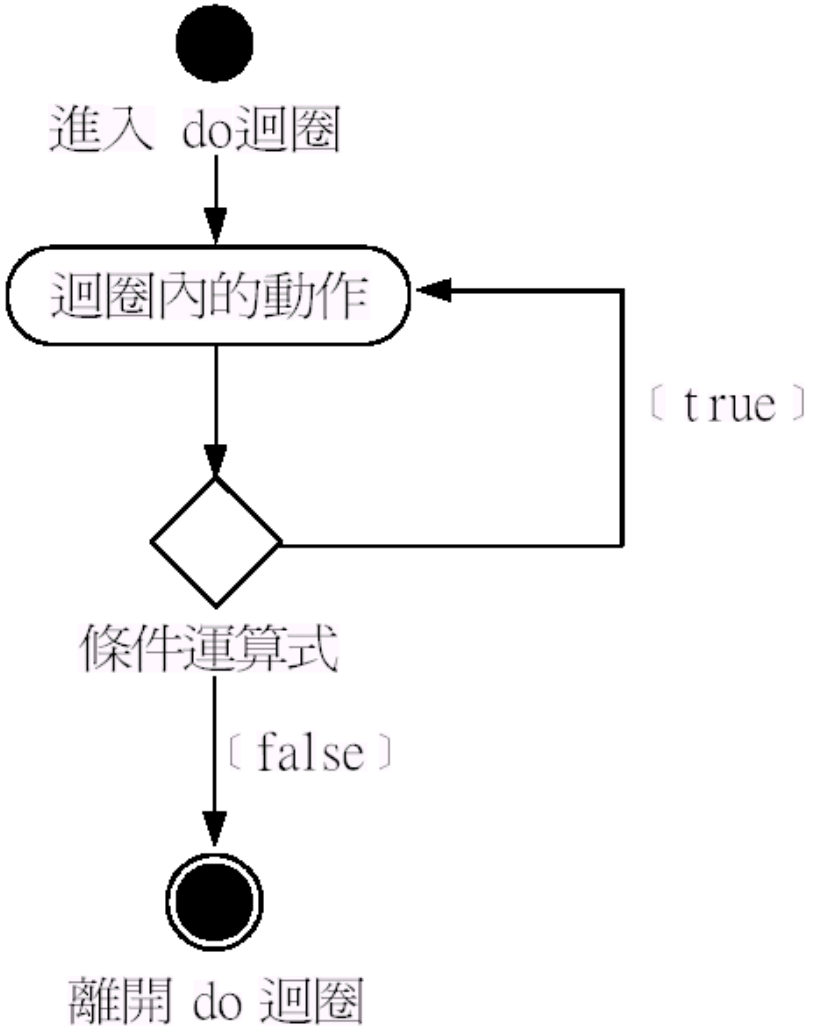
- do/while 迴圈和 while 迴圈相似。前面介紹的 forwhile 迴圈有時被稱為**預先條件迴圈**，也就是說在進入迴圈前就要先檢查條件是否成立；至於 do/while 迴圈則被稱為**後設判斷式迴圈**，它會先執行一次迴圈的動作，然後才檢查條件式是否成立，以決定是否再執行一次迴圈。所以 do/while 迴圈的條件運算式是放在迴圈的大括號後面，如下所示：

do/while 迴圈

```
do {
    敘述...                    //  先執行一次迴圈動作
} while (條件運算式);          //  再檢查條件運算式是否成立
```

- 換句話說, 不論條件式的結果為何, 大括號內的敘述至少會執行一次, 這是 do/while 迴圈的特性, 其執行流程如圖所示：

do/while 迴圈



do/while 迴圈

- 由於 do/while 迴圈是不論如何都會執行 1 次, 所以像剛剛用輾轉相除法求最大公因數的計算, 就很適合用 do/while 迴圈來處理。前面以 while 設計的程式, 改用 do/while 重新設計的結果如下：

do/while 迴圈

程式 Ch05-15.cpp 使用 do/ while 迴圈以輾轉相除法求最大公因數

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int num1,num2;
07    cout << "計算兩整數的最大公因數\n";
08    cout << "請輸入第1個數字：";
09    cin >> num1;
10    cout << "請輸入第2個數字：";
11    cin >> num2;
12
13    int a, b = num1, c=num2;   // c 的值就是第 1 次相除的餘數
14    do {
```

do/while 迴圈

```
15        a=b;
16        b=c;
17        c=a%b;                    //  輾轉相除，取餘數
18    } while (c!=0);               //  當餘數爲 0 時，b 就是最大公因數
19
20    cout << num1 << " 和 " << num2 << " 的最大公因數是："
21         << b;
22 }
```

執行結果

```
計算兩整數的最大公因數
請輸入第 1 個數字：147
請輸入第 2 個數字：258
147 和 258 的最大公因數是：3
```

巢狀迴圈

- 在上述的迴圈範例中, 我們都是以一維的方式去思考, 比如說 1 加到 100 這種只需一個累加變數就能解決的問題。但是如果我們想要解決一個像九九乘法表這種二維的問題 (x,y 兩累加變數相乘的情況)就必須將使用迴圈的方式做一些變化, 也就是在迴圈中放入另一個迴圈, 這種迴圈就稱為**巢狀迴圈 (nested loops)**。

巢狀迴圈

- 巢狀迴圈就是迴圈的大括號之中, 還有其它迴圈, 而且這些迴圈不一定要是同一種, 例如 for 迴圈中可以是 while 迴圈等。以下就是用巢狀的 for 迴圈來設計可輸出九九乘法表的範例程式：

巢狀迴圈

程式 Ch05-16.cpp 利用巢狀迴圈輸出九九乘法表

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    for (int x=1; x < 10; x++) {   // 外迴圈, x 的值由 1 到 9
07       for (int y=1; y < 10; y++)  // 內迴圈, y 的值由 1 到 9
08          cout << x << '*' << y << '=' << x*y << '\t';
09       cout << endl;                // 外迴圈每執行一次就換行
10    }
11 }
```

巢狀迴圈

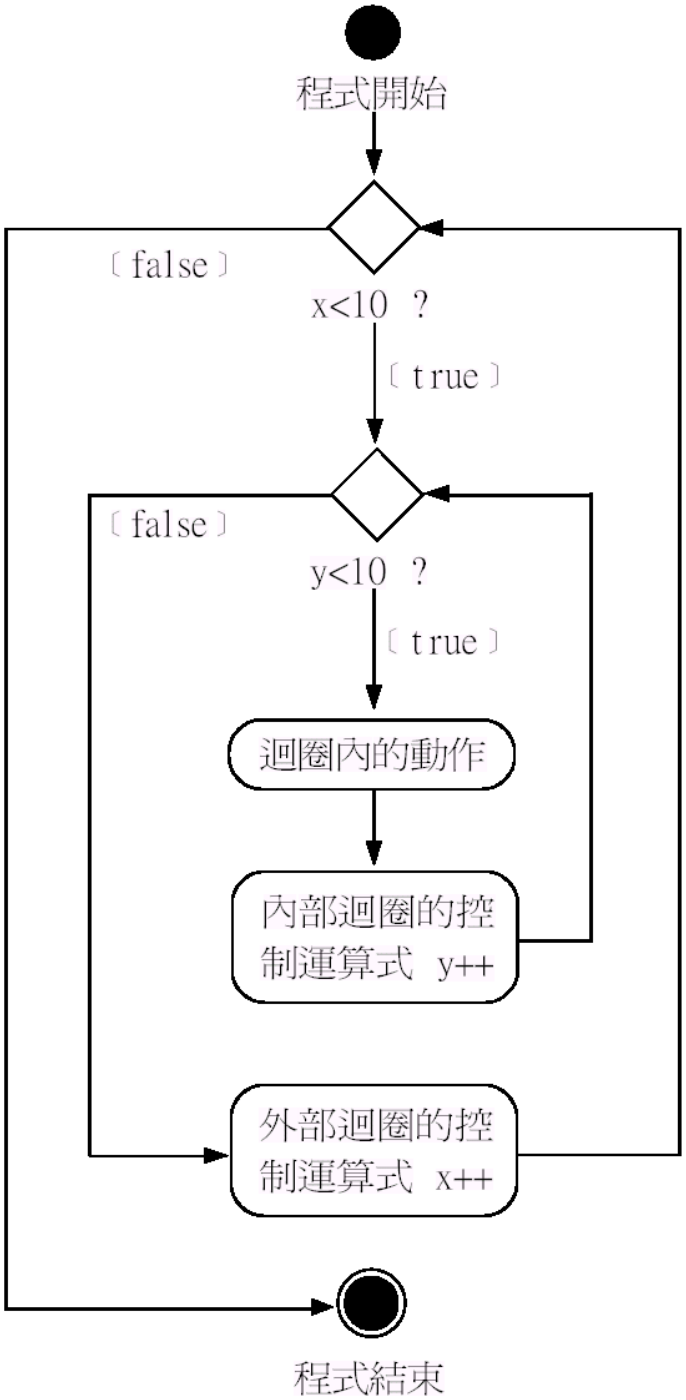
執行結果

```
1*1=1   1*2=2   1*3=3   1*4=4   1*5=5   1*6=6   1*7=7   1*8=8   1*9=9
2*1=2   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*1=3   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*1=4   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*1=5   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*1=6   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*1=7   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*1=8   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*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

巢狀迴圈

- 在上述的例子中, 我們利用兩個迴圈分別來處理九九乘法表的 (x, y) 變數的累加相乘動作。第 6 行的 for 迴圈宣告整數變數由 x 等於 1 開始, 必須分別乘內迴圈從 1 到 9 的 y 當 x 等於 2 時, 又是分別乘上 1 到 9 的 y..., 也就是在外部的 for 迴圈每執行一輪時, 內迴圈就會執行 9 次, 以此類推。執行流程如下圖：

巢狀迴圈


程式開始
〔false〕
x<10 ?
〔true〕
〔false〕
y<10 ?
〔true〕
迴圈內的動作
內部迴圈的控制運算式 y++
外部迴圈的控制運算式 x++
程式結束

巢狀迴圈

內迴圈控制橫向的數字增加 →

外迴圈控制縱向的數字變化 ↓

1*1=1	1*2=2	1*3=3	1*4=4	1*5=5	1*6=6	1*7=7	1*8=8	1*9=9
2*1=2	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*1=3	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*1=4	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*1=5	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*1=6	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*1=7	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*1=8	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*1=9	9*2=18	9*3=27	9*4=36	9*5=45	9*6=54	9*7=63	9*8=72	9*9=81

無窮迴圈

- 在設計迴圈程式時, 要注意一點：一定要讓迴圈的條件運算式有可能為 false, 以使迴圈有機會結束執行。否則迴圈無止盡的執行, 將形成所謂的**無窮迴圈**, 如此程式將不會停止執行, 因此一直耗用系統資源, 此時只好以其它方式強迫程式中止。
- 以下程式示範無窮迴圈的執行情況, 及可能的中止方式：

無窮迴圈

程式 Ch05-17.cpp 無窮迴圈

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    while (true) // 條件永遠爲真, 程式不會結束
07       cout << "無窮迴圈執行中...\n";
08 }
```

無窮迴圈

執行結果

```
無窮迴圈執行中..
無窮迴圈執行中..
無窮迴圈執行中..
無窮迴圈執行中..
無窮迴圈執行中..
```

- 如範例所示, 若不予以理會, 程式將會不斷執行迴圈中的動作, 造成程式耗用大量 CPU 資源的狀況, 必須以強制的方式才能將程式中止。在設計程式時, 一定要讓迴圈的條件運算式有機會為 false, 否則就會造成無窮迴圈的情況。

5-5 變更迴圈流程的 break 與 continue

- 原則上程式流程在進入迴圈後, 都會把迴圈大括號內的所有敘述執行完, 之後才檢查條件式是否成立, 以決定是否繼續執行迴圈。但某些情況下, 我們可打斷這個標準的執行流程, 立即跳出整個迴圈 (中止執行迴圈), 或是跳出該輪迴圈。

用 break 敘述跳出一層迴圈

- 前面介紹 switch 敘述時, 提到要用 break 敘述放在每個 case 後面, 讓程式執行流程會跳出該 switch 敘述外。同理, break 敘述也能放在迴圈之中, 此時它將會中斷目前的迴圈執行, 或者說是**跳出**迴圈。
- 當程式中遇到某種狀況, 而不要繼續執行迴圈時, 即可用 break 來中斷迴圈。方法如下：

用 break 敘述跳出一層迴圈

程式 Ch05-18.cpp 使用 break 跳脫無窮迴圈

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    int i=1;
07
08    while (i>0) { // 無窮迴圈
09       cout << "無窮迴圈執行中...\n";
10       if (i == 5) // 當 i 爲 5 時,
11          break;      // 就跳出迴圈
```

用 break 敘述跳出一層迴圈

```
12          i++;
13      }
14      cout << "成功的跳出迴圈了！";
15  }
```

執行結果

```
無窮迴圈執行中...
無窮迴圈執行中...
無窮迴圈執行中...
無窮迴圈執行中...
無窮迴圈執行中...
成功的跳出迴圈了！
```

← 這行訊息僅出現 5 次，表示迴圈只執行 5 次

用 break 敘述跳出一層迴圈

- 由於在第 8 行的條件運算式 "i > 0" 恆為真, 所以程式中的 while 迴圈是個無窮迴圈。不過由於程式在實際執行時, i 變數會持續累加, 等累加到 i 的值等於 5 時, 第 10 行的 if 條件運算式其值為 true, 所以會執行第 11 行的 break 來跳出此層迴圈。
- 請注意, break 敘述只能跳出一層迴圈, 換句話說, 如果程式是巢狀迴圈, 而 break 敘述是在最內層的迴圈中, 則 break 的功用只能跳出最內層的迴圈, 外層的迴圈則不受影響。

結束這一輪迴圈的 continue

- 除了 break 敘述外, 迴圈還有一個 continue 敘述, continue 的功能與 break 相似, 不同之處在於 break 會跳脫整個迴圈, continue 僅跳出**這一輪**的迴圈。換言之, continue 的功能是立即跳到迴圈的條件運算式 (若是 for 迴圈則是跳到控制運算式), 而不管迴圈中其它尚未執行的敘述。請看以下的程式示範：

結束這一輪迴圈的 continue

程式 Ch05-19.cpp 使用 continue 來跳脫此輪迴圈

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    for (int i=1;i<=10;i++) { // 由 1 到 10 跑 10 次的迴圈
07       if (i == 5)    // 迴圈執行到第 5 輪時, if 條件運算式成立
08          continue;   // 跳出第 5 輪的迴圈, 繼續第 6 輪的迴圈
09       cout << i << '_';
10    }
11 }
```

執行結果

```
1_2_3_4_6_7_8_9_10_
```

結束這一輪迴圈的 continue

- 執行結果獨缺了 5, 是因為迴圈執行第 5 次時, 第 7 行的 if 條件運算式為 true, 因此程式會執行第 8 行的 continue, 該輪迴圈就被結束掉了, 導致第 9 行敘述未被執行, 而直接進行第 6 圈的迴圈。

強制性的流程控制：goto

- C++ 還有個承襲自 C 的流程控制敘述：goto, 如其英文字面意思所示, 它的功用是讓式流程直接跳到指定的位置繼續執行, 不需有任何的條件運算式來判斷真假。goto 敘述的語法如下：

```
goto 標籤名稱;
```

- 標籤名稱的寫法和變數一樣, 我們可在程式另一處放置標籤名稱並加上一個冒號, 如此一來 goto 就會讓程式就會跳到該處繼續執行, 例如：

強制性的流程控制：goto

```
void main()
{
    .....
    if(err) goto pgm_err;

    ...

    pgm_err: cout << " system error:\n";

}
```

強制性的流程控制：goto

- 不當的使用 goto 敘述將造成程式閱讀上的諸多困難並難以理解, 因此目前一般都不會使用這個敘述。理論上, 我們可完全不使用 goto 敘述, 也能達到相同的控制程式流程功效。

5-6 綜合演練

- 判斷是否可為三角形三邊長
- 各種迴圈的混合應用：計算階乘
- 迴圈與 if 條件式判斷式混合應用：取出 1 到指定數值之間的質數

判斷是否可為三角形三邊長

- 在學數學時曾學過, 三角形的兩邊長加起來一定大於第三邊, 我們可以應用這個定理寫一個測試使用者輸入的 3 個值, 是否能為三角形三邊長的程式。判斷的方式當然是用 if 敘述, 由於需反複比較 3 邊的邊長, 所以需用到巢狀 if 的架構。程式如下：

判斷是否可為三角形三邊長

程式 Ch05-20.cpp 判斷是否可爲三角形三邊長

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    float i,j,k;              // 用來儲存 3 邊的邊長
07
08    cout << "請依序輸入三角形的三邊長：\n";
09    cout << "邊長 1 →";
10    cin >> i;                 // 輸入第 1 邊邊長
```

判斷是否可為三角形三邊長

```
11    cout << "邊長 2 →";
12    cin >> j;                  // 輸入第 2 邊邊長
13    cout << "邊長 3 →";
14    cin >> k;                  // 輸入第 3 邊邊長
15
16    if ((i+j) > k)             // 判斷第 1, 2 邊的和是否大於第 3 邊
17      if ((i+k) > j)           // 判斷第 1, 3 邊的和是否大於第 2 邊
18        if ((j+k) > i)         // 判斷第 2, 3 邊的和是否大於第 1 邊
19          cout << "可以爲三角形的三邊長。";
20        else
21          cout << "第 2、3 邊的和小於或等於第 1 邊";
22      else
```

判斷是否可為三角形三邊長

```
23          cout << "第 1、3 邊的和小於或等於第 2 邊";
24      else
25          cout << "第 1、2 邊的和小於或等於第 3 邊";
26  }
```

執行結果

```
請依序輸入三角形的三邊長：
邊長 1 → 3.6
邊長 2 → 8.9
邊長 3 → 6
可以爲三角形的三邊長。
```

判斷是否可為三角形三邊長

執行結果

```
請輸入三角形的三邊長：
邊長 1 → 3
邊長 2 → 2
邊長 3 → 1
第 2、3 邊的和小於或等於第 1 邊
```

- 程式一開始使用了之前提過的方式，讓使用者依序輸入三角形三個邊的邊長，接著在第 16～25 行以三層的巢狀 if 敘述判斷是否任兩邊的和都大於第三邊，並且顯示適當的訊息。

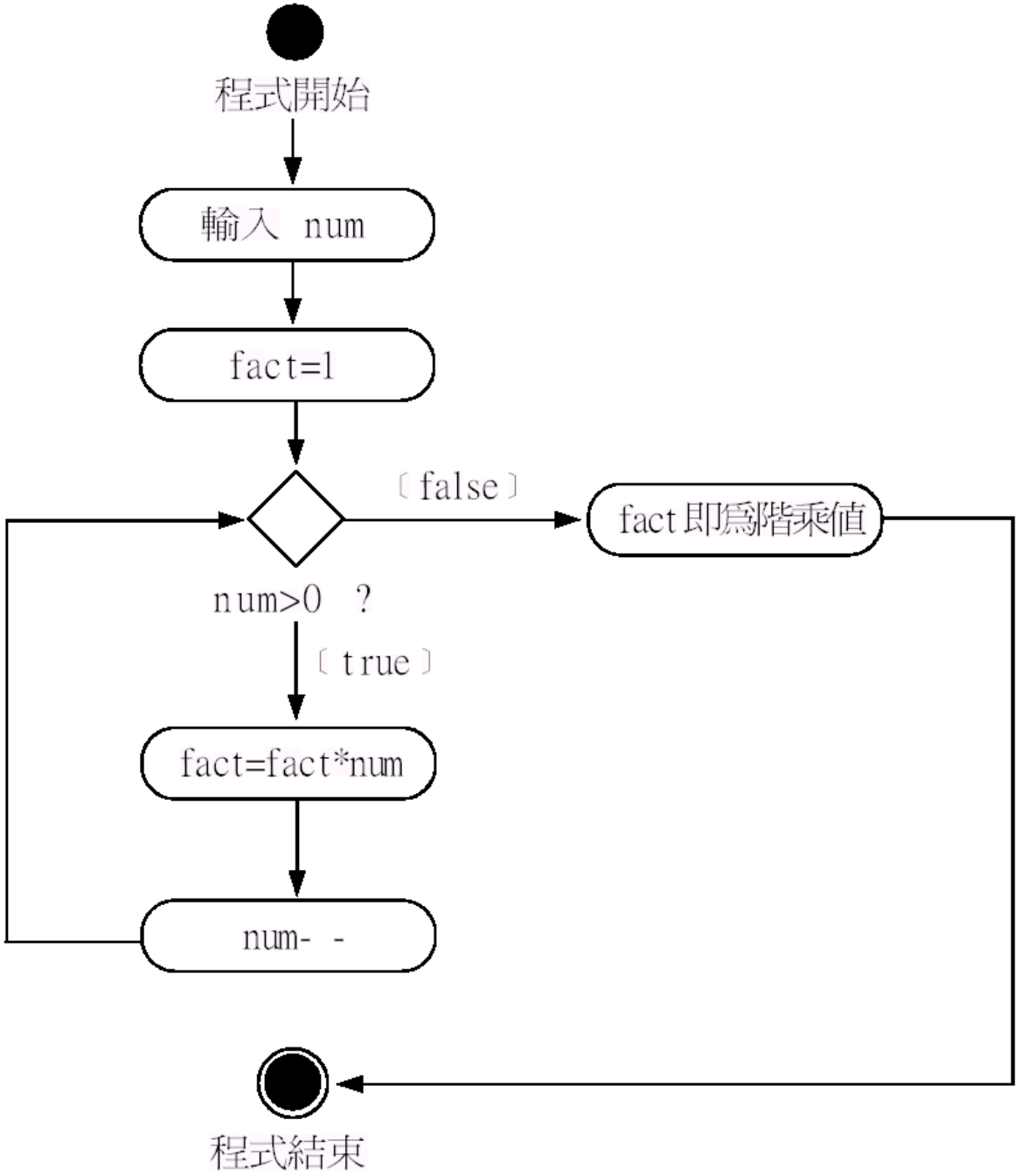
各種迴圈的混合應用：計算階乘

- 在前面的範例中，應該不難發現一個程式中是可以同時使用各種迴圈的。比方說，我們要設計一個可以處理階乘問題的程式，讓使用者可以輸入任意整數，程式會計算該數字的階乘並詢問是否要繼續輸入數字做計算。階乘的算法就是將數字從 1 開始依序相乘：

```
N! = 1*2*3*...*(N-2)*(N-1)*N
```

- 換言之，我們只要用迴圈持續將 1 到 N 的數字相乘，或反過來從 N 乘到 1 即可。以從 N 乘到 1 為例，流程如下：

各種迴圈的混合應用：計算階乘



各種迴圈的混合應用：計算階乘

- 依上述流程設計的程式如下：

程式 Ch05-21.cpp 計算使用者輸入數字的階乘值

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
06    while(true) {
07       cout << "請輸入 1-170 間的整數(輸入 0 即結束程式)：";
08       int num = 0;
09       cin >> num;
10
```

各種迴圈的混合應用：計算階乘

```
11      if (num == 0)
12        break;                  // 若使用者輸入 0，就跳出迴圈
13
14      cout << num << "! 等於 ";
15
16      double fact;                    // 用來儲存、計算階乘值的變數
17      for(fact=1;num>0;num--)  // 計算階乘的迴圈
18        fact = fact * num;        // 每輪皆將 fact 乘上目前的 num
19
20      cout << fact << "\n\n"; // 輸出計算所得的階乘值
21    }
22    cout << "謝謝您使用階乘計算程式。";
23  }
```

各種迴圈的混合應用：計算階乘

執行結果

```
請輸入 1-170 間的整數(輸入 0 即結束程式)：5
5! 等於 120

請輸入 1-170 間的整數(輸入 0 即結束程式)：15
15! 等於 1.30767e+012

請輸入 1-170 間的整數(輸入 0 即結束程式)：0
謝謝您使用階乘計算程式。
```

各種迴圈的混合應用：計算階乘

1. 第 6 到 21 行的 while 迴圈內含主要的階乘計算程式, 迴圈讓程式可重複請使用者輸入新的值來計算。迴圈條件運算式故意直接寫成 true, 表示恆為真的無窮迴圈, 但迴圈內部 (第 11 行) 則有跳出迴圈的機制。
2. 第 11 行用 if 判斷使用者是否輸入 0, 若是即跳出迴圈, 結束程式。

各種迴圈的混合應用：計算階乘

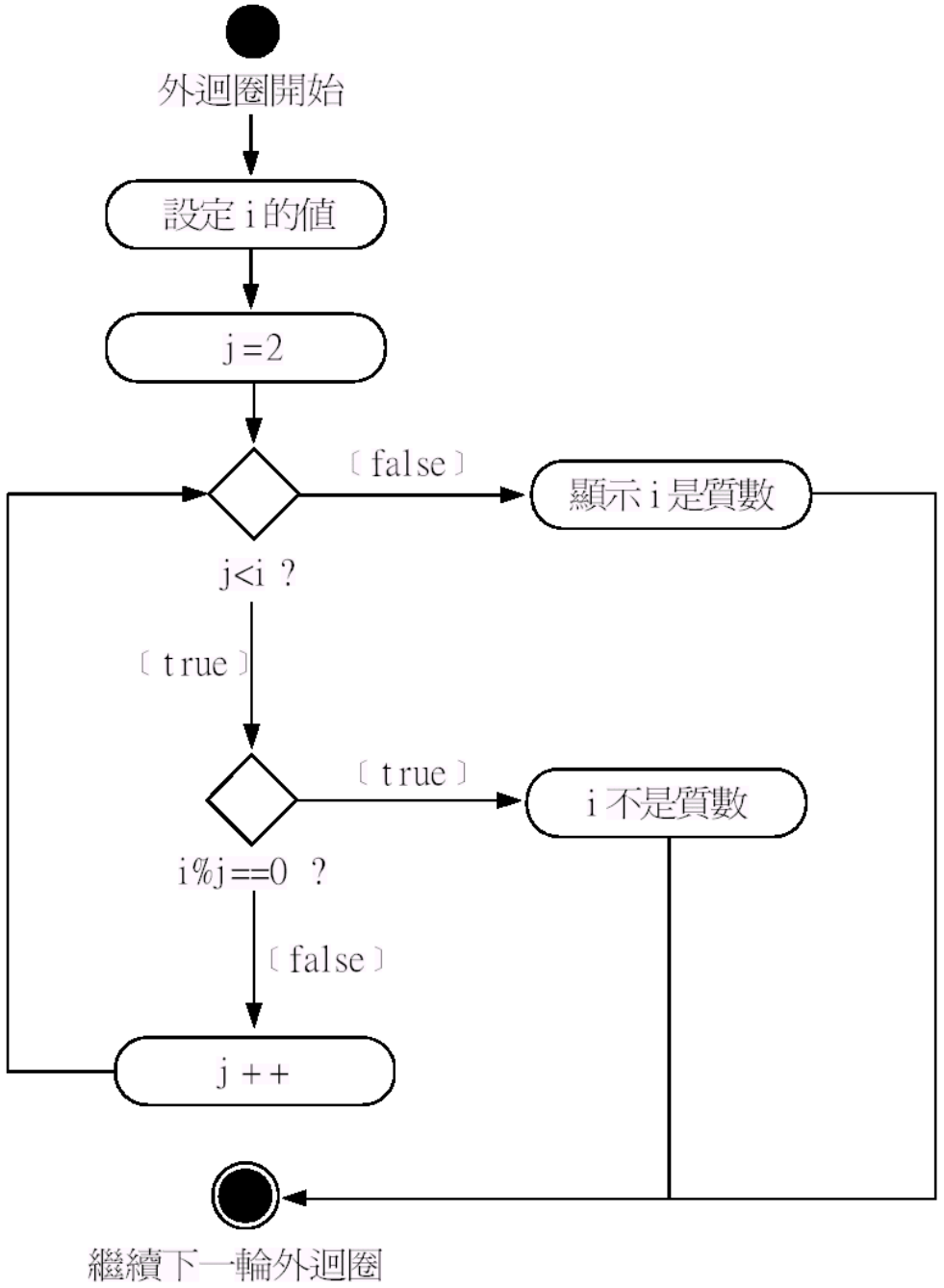
3. 第 16 行用 double 型別來宣告存放階乘值的 fact, 以便程式能計算較大的階乘值。但即使使用了 double 型別, 也只能計算到 170! 的值。
4. 第 17、18 行才是真正在計算階乘值的 for 迴圈, 計算方式如前面的流程圖所示。

迴圈與 if 條件式判斷式混合應用

- 在國中的數學課程裡, 我們有時會為了一個數值是不是質數而花時間去計算。質數就是只能被 1 和自己整除的整數, 所以要自己用手計算的話, 數字越大往往就越難辨別。但是現在我們可以利用巢狀迴圈來解決這類的問題。原理很簡單, 將數字除以每一個比它小的數值 (從 2 開始), 只要能被任 1 比它小的數值整除者就不是質數, 反之則為質數。

迴圈與 if 條件式判斷式混合應用

- 假設 i 是要判斷是否為質數的正整數 (且大於等於 2), 則我們可用如右的流程來判斷 I 是不是質數。



迴圈與 if 條件式判斷式混合應用

- 以上述的流程為基礎，我們設計一個可計算出 2 到指定數值間所有質數的程式。此程式使用巢狀迴圈，外迴圈從 2 到指定數值，內迴圈則實作上述的流程，檢查目前的 i 值是否為質數。

程式 Ch05-22.cpp 計算 2 到指定數值間的所有質數

```
01 #include<iostream>
02 using namespace std;
03
04 int main()
05 {
```

迴圈與 if 條件式判斷式混合應用

```
06    int range;
07    cout << "請輸入要尋找的範圍：";
08    cin >> range;
09
10    int count = 0;                    // 用來計算共有幾個質數
11
12    for (int i=2;i<=range;i++) {
13      bool isPrime = true;            // 表示目前的 i 是否爲質數的布林值
14
15      for (int j=2;j<i;j++) {         // 做除法運算的內迴圈
16        if ((i%j) == 0) {             // 餘數爲 0 表示可以整除,
17          isPrime = false;            // 所以不是質數,
18          break;                      // 也不用繼續除了
```

迴圈與 if 條件式判斷式混合應用

```
19          }
20
21      if (isPrime) {                  // 若是質數, 即輸出該數值
22          cout << i << '\t';
23          count++;
24          if ((count%5) == 0)         // 設定每輸出五個質數就換行
25            cout << endl;
26      }
27    }
28
29    cout << "\n小於等於 "  << range << " 的質數共有 "
30         << count << " 個";
31  }
```

迴圈與 if 條件式判斷式混合應用

執行結果

```
請輸入要尋找的範圍：110
2       3       5       7       11
13      17      19      23      29
31      37      41      43      47
53      59      61      67      71
73      79      83      89      97
101     103     107     109
小於等於 110 的質數共有 29 個
```

迴圈與 if 條件式判斷式混合應用

1. 第 12～27 行的外迴圈, 是用來重複計算 2 到指定數值間的所有整數是否為質數。
2. 第 15～19 行為內迴圈, 功能是將目前的 i 值, 依序除以所有小於它的整數。若無法除盡, 則不變動 isPrime 的值；若可以除盡, 則將 isPrime 設為 false, 表示此數為非質數, 並跳出此迴圈 (因為已經確定它不是質數, 就不用再繼續做除法運算)。

迴圈與 if 條件式判斷式混合應用

3. 第 21～26 行的程式會先檢查 isPrime 的值是否為 true, 是就將目前的 i (質數) 輸出到螢幕上, 同時將 count 的值加 1在 第 25 行則用 count 除以 5, 檢查目前是否為 5 的倍數, 是就輸出一個換行字元, 達到每行只呈現 5 個質數的效果。
4. 第 29、30 行是在外迴圈執行完後, 輸出 count 值, 以顯示在指定的範圍內共有幾個質數。