流程控制的認識

- 程式的執行方式有循序以及跳躍執行的方式，循序式是由上往下一一執行，而跳躍式則可依據條件判斷來處理。
- 流程控制的指令分為兩類：
- 有條件跳躍指令：根據關係運算或邏輯運算的條件式來判斷程式執行的流程，若條件式結果為true，就執行跳躍。有條件跳躍指令包括：

```
if…else
switch…case
try…catch…finally
```

- 迴圈：根據關係運算或邏輯運算條件式的結果為 true 或 false 來判斷，以決定是否執行指定的程式。迴圈指令包括：

```
for…
for each
while
```

單向選擇 if...

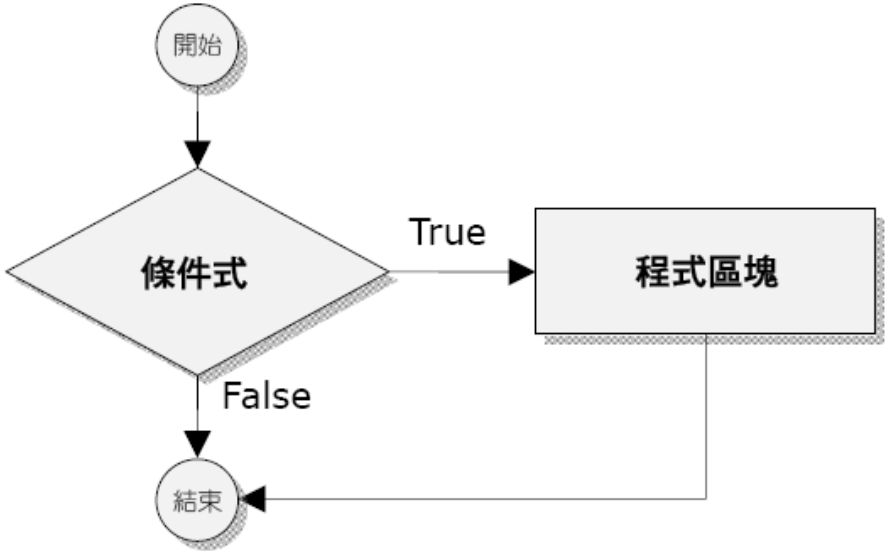
- if...為單向選擇結構，當條件式為 true 時，就會執行程式區塊的敘述；當條件式為 false 時，則不會執行程式區塊的敘述。

```
if (<條件式>)
{
  <程式區塊>;
}
```

- 條件式可以是關係運算式，例如：x>2；也可以是邏輯運算式，例如：x>2 && x<5，如果程式區塊只有一行程式碼，則大括號也可以省略，直接寫成。

```
if (<條件式>)
    <程式區塊>;
```

- 以下是單向選擇流程控制的流程圖：



◆ if 條件控制流程圖

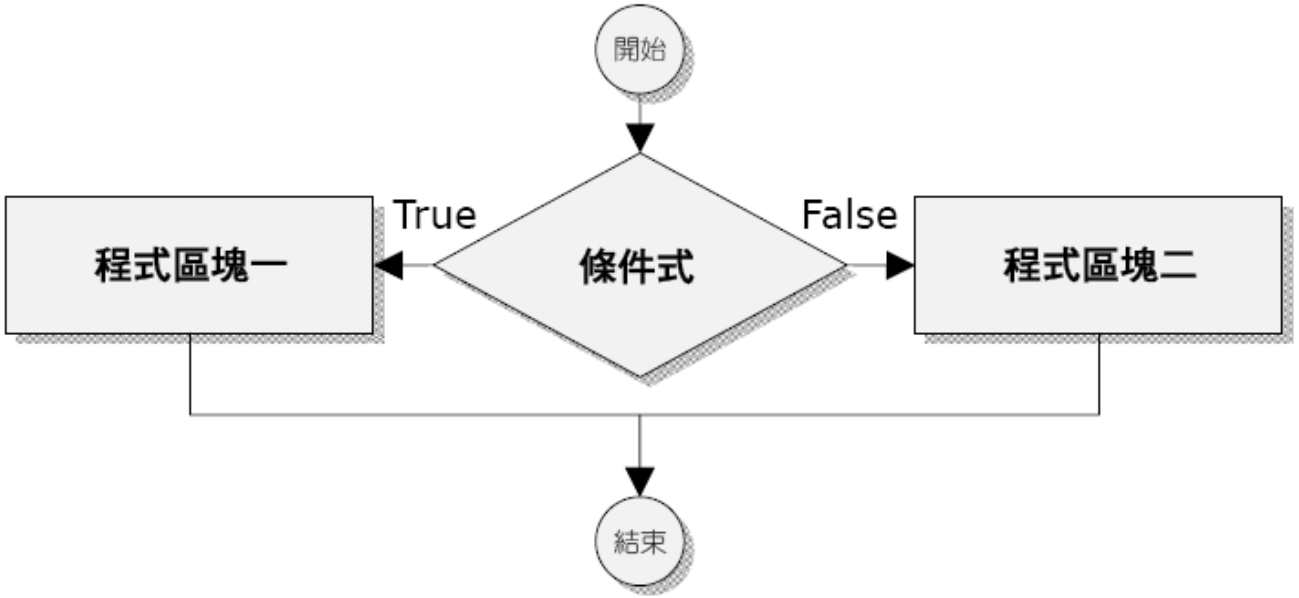
雙向選擇 if...else

- if...else...為雙向選擇結構，當條件式為 true 時，會執行 if 後的敘述 (程式區塊一)。當條件式為 false 時會執行 else 後的敘述 (程式區塊二)，程式區塊可以是一行或多行的敘述，如果程式區塊的敘述只有一行則可以省略大括號。

語法與控制流程

```
if (<條件式>)
{
<程式區塊一>;
}
else
{
<程式區塊二>;
}
```

- 以下是雙向選擇流程控制的流程圖：



◆ if / else 條件控制流程圖

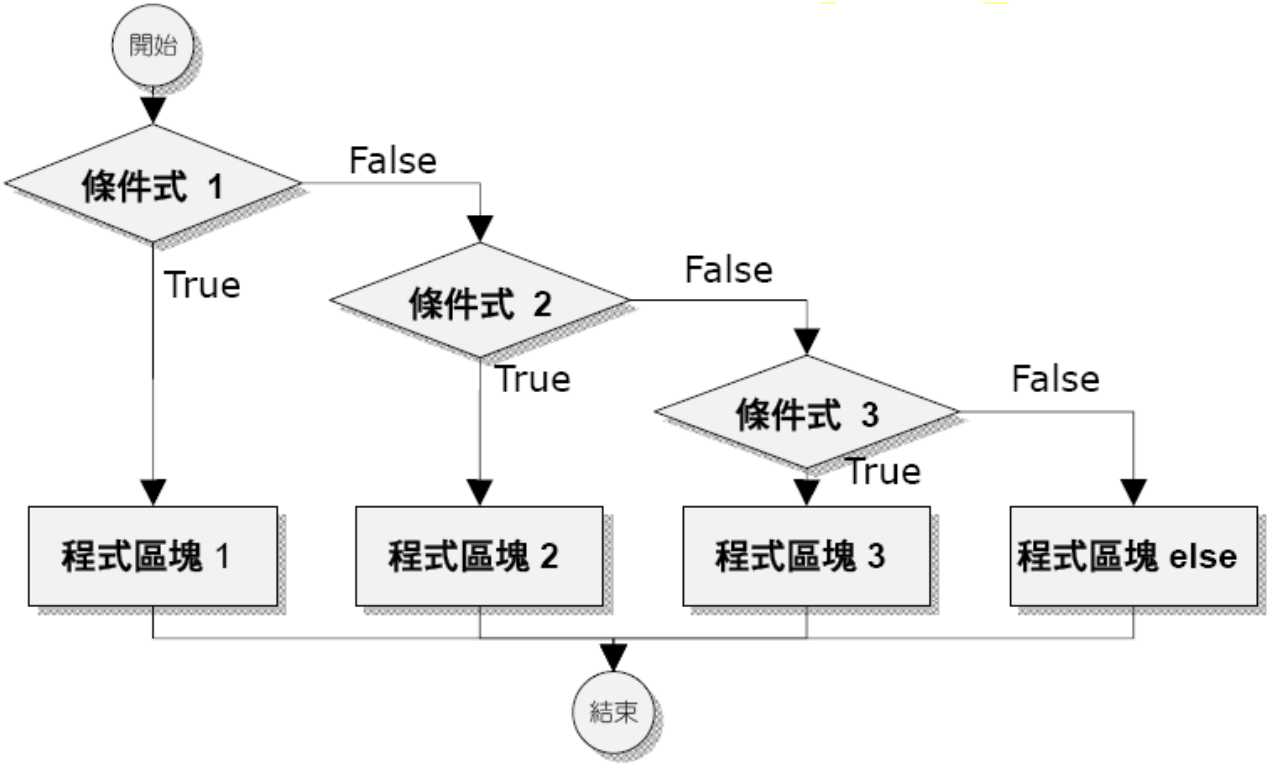
多向選擇

if……else If…

- 如果條件式為 true 時，就執行相對應的程式區塊，如果所有條件式都是 false，則執行 else 後的程式區塊。（若省略 else 敘述，則條件式都是 false 時將不執行任何程式區塊）

```
if (<條件式一>)
{
   <程式區塊一>;
}
else if (<條件式二>)
{
  <程式區塊二>;
}
else if (<條件式三>)
   ……
[else]
{
  [<程式區塊 else>];
}
```

- 以下是多向選擇流程控制的流程圖 (以設定 3 個條件式為例)：

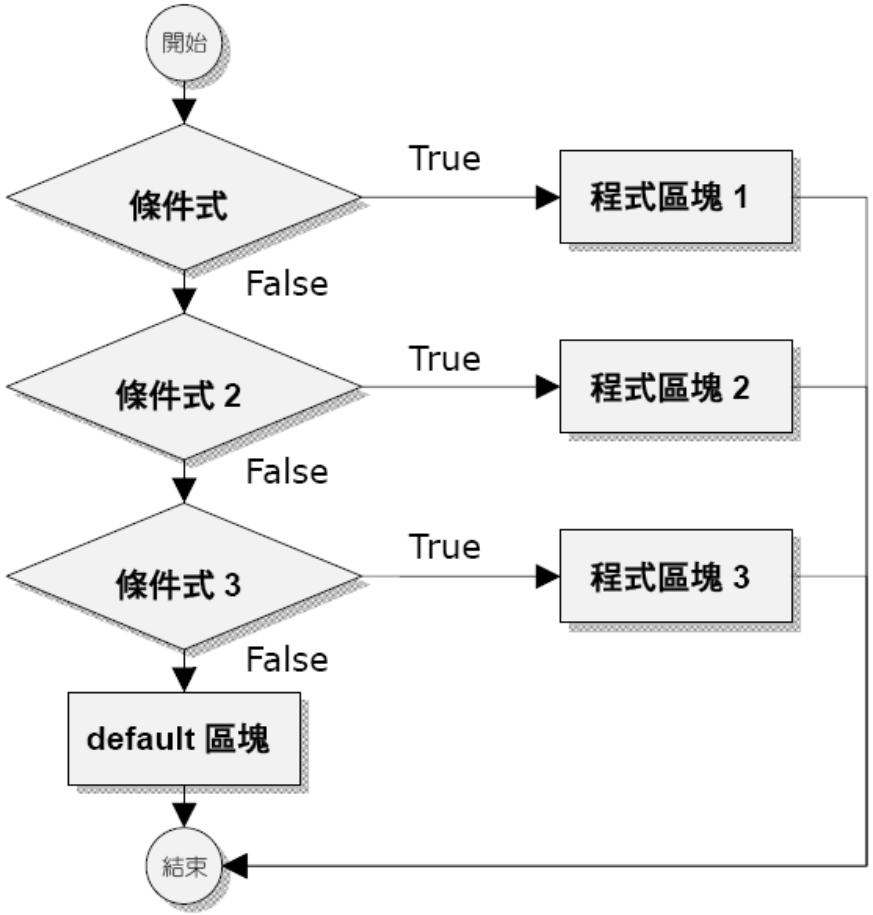


switch…case

- 使用 if…else if… 多向選擇有一個缺點，當判斷的條件式較多時，會使程式顯得複雜且不易閱讀，若使用 switch…case 則可以改善程式碼複雜的狀況，但是 switch 只能判斷整數或字串條件式，使用上限制也較多，而 if …else if 則無此限制。

```
switch (<表示式>)
{
    case Value1:
      <程式區塊一>;
      break;
    case Value2:
      <程式區塊二>;
      break;
    ..............
    case ValueN:
      <程式區塊 N>;
      break;
    [default]:
      [<程式區塊 default >];
      break;
}
```

- 以下是 switch 多向選擇流程控制的流程圖 (以設定 3 個條件式為例)：



◆ switch 條件控制流程圖

重複結構

- 重複結構會重複執行指定區塊的程式碼，包括指定重複次數的 for...和依前後條件測試的while...、do...while。

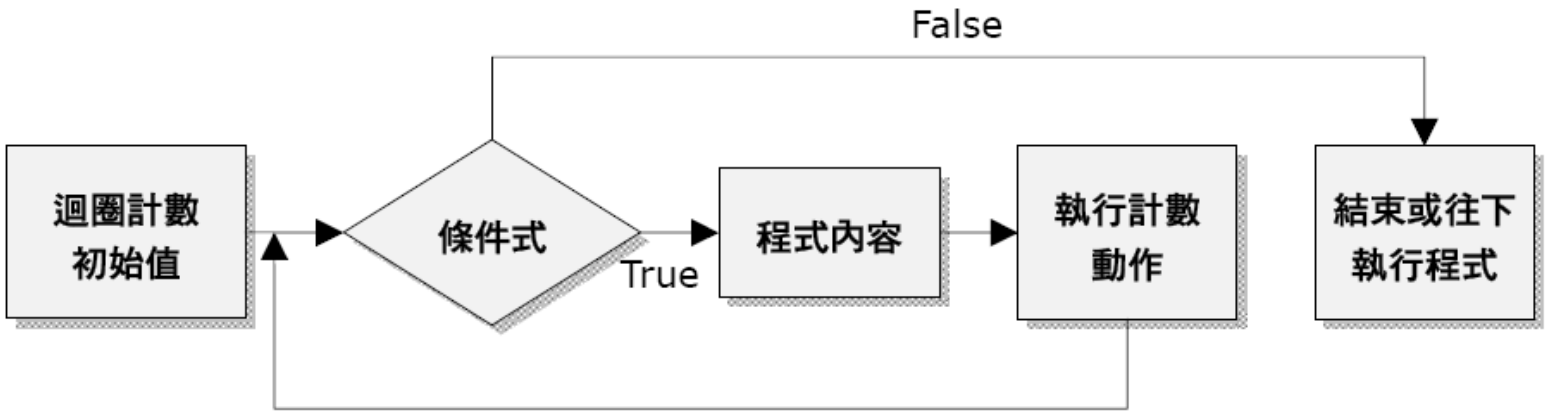
for...迴圈

- for...迴圈自初值開始判斷 <條件判斷> 是否為 true，若為 true 則執行 for 迴圈內的敘述，並依 <增量值>，每次增加 (或減少) 指定的增量值，直至 <條件判斷> 為 false 為止，break則可以強迫結束 for 迴圈。

語法與控制流程

```
for ([變數初始化宣告];[條件判斷];[增量值])
{
    程式碼;
    [break;]
    程式碼;
}
```

- 以下是 for 計次迴圈流程控制的流程圖：



◆ for 迴圈流程圖

for each…迴圈

- for each …是陣列 (Arrays) 或集合 (Collections) 的專用迴圈，它會依序對陣列中的每個元素執行。

語法

```
for each(資料型別 變數 in Group)
{
  程式碼;
  [break];
  程式碼;
}
```

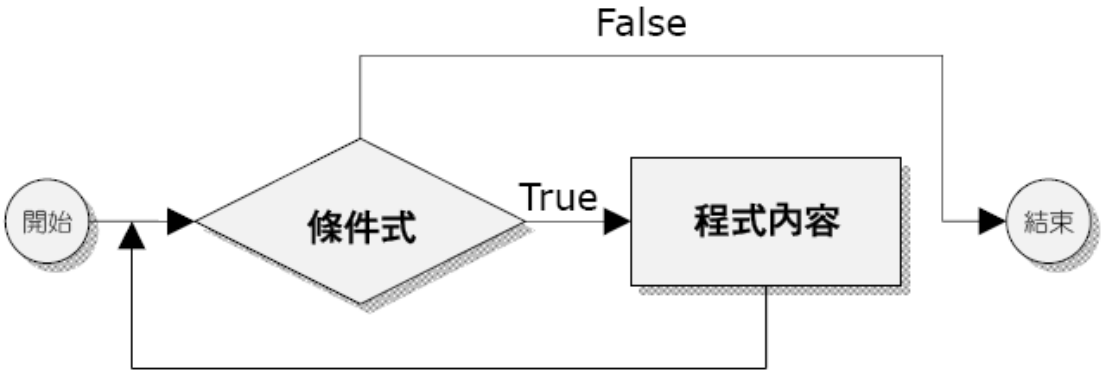
while 前測試迴圈

- 前測試迴圈會先測試 <條件式> 是否滿足，以決定是否要執行迴圈內的程式區塊。若條件成立就執行程式區塊一次，然後再回到迴圈最前面的條件式繼續測試；若條件不成立就結束迴圈的執行。在前測試迴圈中，如果連第一次的測試條件都不成立，則程式區塊將完全未被執行。

語法與控制流程

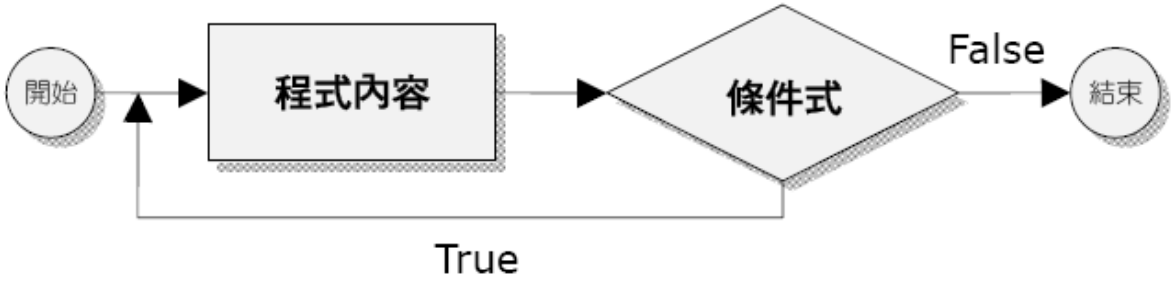
```
while(<條件式>)
{
  程式碼;
  [break;]
  程式碼;
}
```

- 以下是 while 迴圈流程控制的流程圖：



◆ while 迴圈流程圖

- 以下是 do...while 迴圈流程控制的流程圖：



◆ do…while 迴圈流程圖

Do..while

try...catch...finally 錯誤處理

- 應用程式在開發過程中難免發生錯誤，如何發現錯誤所在並加以修正，是每一個程式設計者必須具備的基本能力。C++ 和 C++/CLR 對於錯誤處理的語法稍有不同，所以分開來說明。

語法

- C++中的例外處理使用三個關鍵字來進行：try、throw、catch，其語法架構如下：

```
try
{
  可能發生錯誤的程式碼;
  throw Type;
}
catch(Type 1)
{
  處理錯誤程式碼;
}
catch(Type 2)
{
  處理錯誤程式碼;
}
```

C++/CLR 結構化錯誤處理

- 感覺上，C++ 的錯誤處理功能似乎稍嫌不足，幸好 CLR 的錯誤處理做了改進。CLR 的錯誤處理語法和 C++ 稍有不同，其語法如下：

語法

```
try
{
    可能發生錯誤的程式碼;
}
catch(exception 物件型別 變數]
{
     處理錯誤程式碼;
    [exit try;]
}
[finally]
{
     [程式碼;]
}
```