國立屏東大學 資訊工程學系 C++程式設計入門教材

7. 選擇

早在上一世紀60年代，學者們就已經證明了從簡單到複雜的各式應用問題都可以由順序(Sequence)、選擇(Selection)與迴圈(Loop)三種基本結構所組成的程式加以解決[1)]。截至目前為止，本書所有範例程式的執行都是從main()函式開始，一行、一行的執行，直到main()結束為止。這種 “ 一條腸子通到底 ” 的線性執行動線就是順序結構的意思 — 簡單有效、但還不夠，我們還需要學習其它兩種結構才能夠解決所有的問題，這就是本章與下一章的目的 — 分別為讀者介紹選擇與迴圈結構。

C++語言支援讓程式依據特定條件執行不同動線的選擇敘述(Selection Statement)，我們將可以為程式的執行定義特定條件，讓程式視不同情況執行不同的程式碼。本章將先介紹與定義條件相關的邏輯運算式(Logical Expression)，以及兩個常用的條件敘述 — if及switch。

隨著我們開始使用條件敘述，程式設計的複雜性也隨之提升，原本所使用的IPO模型已經不敷所需；因此，本章也將介紹第二種程式設計的思維模型工具 — 流程圖(flowchart)，透過此一工具，我們將能更容易地設計出結構與動線都更為複雜的程式。

7.1 邏輯運算式

所謂的邏輯運算式(Logical Expression)亦稱為布林運算式(Boolean Expression)，由著名數學家George Boole所提出，是當代電腦科學的重要基礎。邏輯運算式的運算結果稱為布林值(Boolean Value)，只能是true與false兩種可能，分別代表某種情況、情境或是狀態、條件的「正確」與「錯誤」、「成立」與「不成立」、「真」與「偽」等「正面的」或「負面的」兩種可能。我們在4.3.4 布林型態已經介紹過，C++語言所支援的布林型態為bool，其數值true與false亦可以非0(預設為1)以及0的整數值表達[2)]，並且在6.4.4 布林型態的數值介紹過其相關的輸入與輸出格式設定相關函式與操控子，請讀者自行加以回顧。

邏輯運算式的運算元(Operand)可以是數值、常數、變數、函式呼叫、甚至是其它的運算式；至於在運算子(Operator)方面可再區分為關係運算子(Relational Operator)、相等運算子(Equality Operator)、不相等運算子(Inequality Operator)與邏輯運算子(Logical Operator)等多項分類，我們將在本節中分別加以介紹。

7.1.1 關係運算子

關係運算子(Relational Operator)是一個左關聯的二元運算子，用以判斷兩個運算元(數值、函式、變數或運算式)之間的關係，其可能的關係有：大於、小於、等於、或不等於，C語言提供以下的關係運算子，如table 1。

符號	範例	意義
>	a > b	a 是否大於 b
<	a < b	a 是否小於 b
>=	a >= b	a 是否大於或等於 b
<=	a <= b	a 是否小於或等於 b

Tab. 1: Relational Operators

可以搭配table 1的關係運算子使用的運算元與一般運算式一樣，可以是數值、變數、常數、函式呼叫，甚至是運算式。舉例來說，假設a=5與b=10分別為兩個int整數變數，c=15為整數常數，下列的邏輯運算式示範了不同運算元的使用：

Example 1

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
    int a=5, b=10;
    const int c=15;
    cout << boolalpha;
    cout << setw(21) << "(a>0)="                 << (a>0)                 <<
endl;
    cout << setw(21) << "(12<b)="                << (12<b)                <<
endl;
    cout << setw(21) << "(a+b>=c+2)="            << (a+b>=c+2)            <<
endl;
    cout << setw(21) << "(a*b>=sizeof(int)*4)=" << (a*b>=sizeof(int)*4) <<
endl;
    cout << setw(21) << "(round(3.48)>3)="       << (round(3.48)>3)       <<
endl;
}
```

在上面的例子當中，其中做為運算元的部份包含了數值、變數、常數、函式呼叫以及運算式。請試著計算出這些邏輯運算式的運算結果為何？然後再比對下面的執行結果：

```
               (a>0)=true
              (12<b)=false
          (a+b>=c+2)=false
(a*b>=sizeof(int)*4)=true
     (round(3.48)>3)=false
```

最後還要注意關係運算子較算術運算子(Arithmetic Operator)優先順序低，所以x+y<i-j等同於(x+y) < (i-j)另外，由於關係運算子是左關聯，因此x<y<z等同於(x<y)<z

7.1.2 相等與不相等運算子

相等運算子(Equality Operator)與不相等運算子(Inequality Operator)同樣是左關聯的二元運算子，其優先順序也都較算術運算子來得低C++語言提供==與!=運算子，用以判斷兩個運算元(數值、函式、變數或運算式)之值是否相等，如table 2

符號	範例	意義
==	a == b	a 是否等於 b
!=	a != b	a 是否不等於 b

Tab. 2: Equality Operators

是 ==， 不是 =

千萬不要將比較兩數是否相等的==寫成=，這實在是一個常常會遇到的錯誤!建議你以後如果遇到程式執行結果錯誤，但找不出任何問題時，試試檢查一下所有的 = 與 ==，有很高的機會可以改正你的程式。

7.1.3 邏輯運算子

C++語言提供三種邏輯運算子(Logical Operator)，如table 3，

符號	意義	一元或二元
!	NOT	一元
&&	AND	二元
||	OR	二元

Tab. 3: Logical Operators

其運算結果請參考table 4的真值表：

X	Y	NOT X	X AND Y	X OR Y
0	0	1	0	0
0	1		0	1
1	0	0	0	1
1	1		1	1

Tab. 4: Truth Table

AND與OR都是二元運算子，必須與兩個運算元搭配運算，其運算元通常為邏輯運算式，但也可以是具有布林值或整數值的變數、常數、或可傳回布林值或整數值的函式呼叫等。至於NOT則是一元運算子，僅和一個運算元相關。以下我們分別說明三者的用途：

- AND(&&)，AND用以處理兩個條件皆成立的情形。假設變數score代表C語言的修課成績，以下使用AND的邏輯運算即為檢查成績是否介於0~100：

```
((score >= 0) && (score <=100))
```

- OR(||)，OR是用以判斷兩個條件中任意一個成立的情形，例如學生的修習某門課程的期末考成績大於等於60分或是作業成績不低於80分，就可以通過該課程，那麼可以表達為：

```
(( final >= 60 ) || ( homework >=80 ))
```

- NOT(□)□與前述兩者不同□NOT是一個一元的運算，它只與一個運算元相關。例如quit是一個整數變數，其值為false代表不要離開(或結束)程式的執行，所以在程式中可以依使用者的設定或其它狀況將quit變數設定為true□代表要離開程式的執行。因此，程式中可以使用

```
(□quit)
```

來表示Not Quit — 意即不要離開程式的執行。由於quit=false表示不想離開，所以Not運算就將(!quit)變成「不要離開」之意！

注意：變數x是否的值介於a與b之間

許多初學者在學習邏輯運算式時，通常會犯下一種常見的錯誤：以□a < x < b□來表達「變數x是否的值介於a與b之間」！這個式子看似正確，實則不然。因為這裡所使用的 < 為左關聯的關係運算子，所以□a < x < b□其實會變成□(a < x) < b□意即先判斷x是否大於a□然後將其判斷結果再與b進行比較。假設a=1□b=10□x=50□我們想要以羅輯運算式□a < x < b□來判斷變數x是否的值介於a與b之間，由於左關聯的緣故□□a < x < b□將會等於□(a < x) < b□□也就是「(1<50) <10」；由於「(1<50)」的結果是true(也就是整數值1)，所以「(1<50)<10」就變成了「1<10」，也因此其最終的運算結果將會是true□

但是請你冷靜地想一想在 a=10□b=1與x=5 的情況下，變數x的數值其實是沒有介於a與b之間的。如果要正確的判斷變數x的數值有沒有介於a與b之間的話，應該寫成□ (a<x)&&(x<b)□才是正確的寫法！

最後還要說明的是，由於邏輯運算子(例如上面的&&)的優先順序比關係運算子(例如上面的<)來得低，所以不論a<x與x<b有沒有使用括號其運算結果都是正確的。

7.1.4 優先順序

我們將目前為止介紹過的運算子之優先順序(Precedence)整理如table 5(表中是以優先權高至低依序列示)：

運算子	符號
一元運算子	+(正)、-(負)□++□--□!(NOT)
算術運算子(乘除)	*□/□%
算術運算子(加減)	+□-
關係運算子	>=□<=□>□<
相等運算子	==□!=
邏輯運算子	&&□||
條件運算子	?:
指定運算子	=□*=□/=□%=□+=□-=

Tab. 5: 各運算子的優先順序(由高至低)

關於C++語言所有運算子的關聯性及優先順序，亦可參考本書附錄B□

7.2 if敘述

當我們在程式寫作時，某些功能可能是要視情況來決定是否要加以執行的。在C++語言中，提供一個if敘述，可以做到依特定條件成立與否，來決定該執行哪些程式碼。if的語法如下：

if敘述語法

if (測試條件) 敘述 | { 敘述* }

依據此語法在if之後必須以一組括號將所謂的「測試條件(Test Condition)」包裹於其中 — 測試條件即為本章前面所介紹的邏輯運算式，或者也可以直接給定能表示為布林值或整數值的變數、常數或函式呼叫(當使用整數時，以非0的數值為true，以0為false)。當測試條件經檢測其運算結果成立時(也就是其布林值為true，或是其整數值不為0時)，就會執行接在其後的一行敘述或是使用一組大括號包裹的多行敘述。

請參考下面的程式：

```
if (score >= 60) cout << "You pass!" << endl;;
```

if敘述除了上面的寫法以外，比較常見的是利用一個換行，將測試條件成立時所要執行的敘述寫在下一行，並且將它往內縮排：

```
if (score >= 60)
   cout << "You pass!" << endl;;
```

不論是上面這兩個中的何者，其意義是完全相同的，差別只在於閱讀時是否容易“看出來”敘述是不是屬於哪個if敘述而已。它們兩者在執行時，經判斷測試條件(score>=60)後，若其結果為true，則印出“You are pass!”。當然，若條件不成立時，後面所接的cout敘述是不會被執行的。

如果測試條件成立時，想要進行的處理需要一行以上的程式碼該怎麼辦呢？我們可以在if敘述後，以一組大括號來將要執行的程式碼包裹起來。這種被包裹起來的程式碼又稱為複合敘述(Compound Statment)。請參考下面的例子：

```
if (score >= 60)
{
   cout << "Your score is " << score << endl;
   cout << "You pass!" << endl;
}
```

如果我們想判斷的條件不只一個，那又該怎麼辦呢？其實if敘述也是敘述，所以可以通過測試條件後所要

執行的敘述裡，再寫另一個if的敘述 — 如此一來，就是在一個if敘述裡再包含另一個if的敘述。請參考下面的程式：

```
if (score >= 60)
{
   cout << "Your score is " << score << endl;
   cout << "You pass!" << endl;
   if(score >= 90)
   {
      cout << "You are outstanding!" << endl;
   }
}
```

下面是另一個例子：

```
if (score >= 0)
{
   if(score <= 100)
   {
      cout << "The score " << score << " is valid!" << endl;
   }
}
```

不過這個例子，還可以改寫成：

```
if ((score >= 0)&&(score <=100))
{
   cout << "The score " << score << " is valid!" << endl;
}
```

我曾經看過有人把程式這樣寫:

```
if ( 0 <= score <= 100)
{
   cout << "The score " << score << " is valid!" << endl;
}
```

雖然我可以瞭解他的明白，但其實這個程式是錯誤的！因為關係運算子是左關聯□0<=score<=100在執行順序上其實是(0<=score)<=100□假設score的值是110(不合理的分數，應判定為false才是)，此測試條件就會等同於 (true)<=100□由於true等同於非0的整數，但預設值為1，所以此測試條件就又變成了 1 <=100，因此

最終的運算結果為true — 可是這是錯的！110分不介於0到100之間，是不合理的分數！

延續上面的例子，若是想要在score超出範圍時，印出錯誤訊息，那又該如何設計呢？請參考下面的程式：

```
if ((score >= 0)&&(score <=100))
{
   cout << "The score " << score << " is valid!" << endl;
}

if((score<0) || (score>100))
{
   cout << "Error! The score " << score << " is out of range!" << endl;
}
```

在這段程式碼中的兩個if敘述，其實是互斥的，也就是當第一個if的條件成立時，第二個if的條件絕不會成立，反之亦然。這種情況可以利用下面的語法，把兩個if敘述整合成一個：

if敘述語法

if (測試條件) 敘述 else 敘述

或

if (測試條件) {敘述*} else {敘述*}

else保留字可以再指定一個敘述或是複合敘述，來表明當if條件不成立時，所欲進行的處理。請參考下面的例子：

```
if ((score < 0) || (score >100))
{
   cout << "Error! The score " << score << " is out of range!"  << endl;
}
else
{
   cout << "The score " << score << " is valid!" << endl;
}
```

再一次考慮到if敘述也是一種敘述，在else的後面，我們也可以再接一個if敘述，例如下面的例子：

```
if ((score < 0) || (score >100))
{
   cout << "Error! The score " << score << " is out of range!"  << endl;
}
else
{
   if(score>=60)
   {
      cout << "You pass!" << endl;
   }
}
```

類似的結構延伸，下面的程式碼也是正確的：

```
if ((score < 0) || (score >100))
{
   cout << "Error! The score " << score << " is out of range!"  << endl;
}
else
{
   if(score>=60)
   {
      cout << "You pass!" << endl;
   }
   else
   {
      cout << "You fail!" << endl;
   }
}
```

上述的程式碼，也可以利用if敘述及else保留字後面可以接一個敘述(只有一個敘述時，大括號可以省略)，我們可以將部份的大括號去掉，請參考下面的程式碼:

```
if ((score < 0) || (score >100))
{
   cout << "Error! The score " << score << " is out of range!"  << endl;
}
else if(score>=60)
{
   cout << "You pass!" << endl;
}
else
{
      cout << "You fail!" << endl;
}
```

在本節結束以前，讓我們來看一些使用if完成的程式範例：

Example 2

某百貨公司週年慶舉辦「滿五千折五百」活動，請設計一C++程式，讓使用者輸入購物總金額，並計算其(不論是否有)折扣後的金額：

```
#include <iostream>
using namespace std;

int main()
{
   float total;
   cout << "Please input the total:";
   cin >> total;
   if(total >= 5000 )
   {
      total*=0.95;
   }
   cout << "Total=" << total << endl;
   return 0;
}
```

Example 3

屏東火車站有開往屏東大學不同校區的接駁車服務，車資如下：

- 屏東火車站←→屏商(1) 30元
- 屏東火車站←→民生(2) 25元
- 屏東火車站←→林森(3) 20元

請設計一個C++語言程式，不論是從火車站到屏東大學，或是從屏東大學到火車站，都讓使用者輸入屏東大學的校區(以整數1, 2, 3 分別代表屏商、民生及林森校區)

```
#include <iostream>

int main()
{
    int loc;
    int payment=0;

    cin >> loc;

    if(loc==1)
```

```
    {
        cout << "30" << endl;
    }
    else if(loc==2)
    {
        cout << "25" << endl;
    }
    else if(loc==3)
    {
        cout << "20" << endl;
    }
    else
    {
        cout << "Error!" << endl;
    }
}
```

7.3 switch 敘述

除了if敘述以外，C++還提供另一種選擇敘述 — switch敘述，正如其名稱一樣，它就像一個 “ 開關 ”，依據特定的數值決定程式不同的執行動線。具體來說，使用switch敘述時，必須給定一個特定的整數變數或運算結果為整數的運算式，然後可以針對該整數可能的各種數值，執行不同的程式敘述。其語法如下：

switch敘述語法

switch (整數變數 | 整數運算式)
{
    [case 整數值|整數常數運算式: 敘述*]*
    [default: 敘述*]?
}

switch敘述與if敘述類似，都可以讓程式視情況執行不同的程式碼，但switch針對整數變數或整數型態的運算式可能的不同數值，使用列舉式的方式提供不同的處置操作，其語法詳細說明如下：

1. switch (整數變數 | 整數運算式)：以switch開頭，緊接著以一組小括號將一個「整數變數」或是一個「整數運算式」加以包裹。
    - 整數變數:整數型態的變數(包含short、long、int等各種整數型態皆可)，但也可以使用char字元型態的變數，因為字元的本質就是整數。
    - 整數運算式：運算結果為整數的運算式。
2. { … }：後續則以一組大括號與其內的敘述來定義不同情況下的處理方法。可以接受的敘述包含「case標籤敘述」與「預設標籤敘述」兩種，分述如下：
    - case標籤敘述(case Label Statement)
        - 依其語法[case 整數值|整數常數運算式: 敘述*]*，此部份為可選擇性的(可使用0次或多次)， 換言之，在一個switch敘述內可以完全沒有任何case標籤敘述，也可以有多個case標籤敘述)。
        - 每次使用必須以 case 開頭，其後接一個標籤數值(可使用「整數值」或「整數常數運算式」)與一個冒號：，並於其後接0個或多個程式敘述。

        - 整數值:就是整數型態的數值，或是等同於整數值的字元，例如數值1、2、3，或是'A'、'B'、'C'。
        - 整數常數運算式:就是運算結果為整數的運算式，但其運算元僅為整數值或常數(不可以使用變數或函式呼叫)。
      - 可以有0個或多個敘述(statement)敘述。
      - 當接在switch後的「整數變數」或「整數運算式」的數值與此處的「整數值」或「整數常數運算式」相同時，程式的執行就會從swithc處跳躍到此case標籤處開始，並依序執行後續的程式敘述。
      - 一旦開始執行，就會一直往下執行(遇到別的case標籤也不會停止)，直到遇到switch敘述結尾處的右大括號，或是使用break敘述來跳離switch敘述為止。
    - 預設標籤敘述(Default Label Statement)
      - 依其語法[default: 敘述*]?，此部份是選擇性的，至多可以寫一次，但也可以省略不寫。其用意就在於為「整數值」或「整數運算式」的數值，提供一個預設的處理方法。換句話說，若前面的各個case標籤的「整數值」或「整數常數運算式」都不等同於switch的「整數值」或「整數運算式」時，那麼程式就會略過前述的各個case標籤敘述，直接跳躍到一個名為default的標籤處執行敘述，直到遇到switch敘述結尾的右大括號或是使用break中斷為止，才會跳離switch敘述。

請特別注意在上述語法說明中所提到的「break敘述」，其使用方式就是在需要跳離switch敘述的地方，寫下「break;」即可。

現在，讓我們以前一小節所介紹的Example 3程式為例，將它以switch敘述改下如下：

Example 4

```
#include <iostream>
using namespace std;

int main()
{
    int loc;
    int payment=0;

    cin >> loc;

    switch(loc)
    {
        case 1:
           cout << "30" << endl;
           break;
        case 2:
           cout << "25" << endl;
           break;
        case 3:
```

```
            cout << "20" << endl;
            break;
        default:
            cout << "Error!" << endl;
            break;
    }
}
```

請將上述程式碼編輯、編譯並加以執行，看看結果為何？注意，在這個程式中，每個case的敘述後都加了一個break敘述，其作用是讓程式的執行跳離其所屬的程式區塊中，一個程式區塊(block)是一組左右對稱的大括號與其內的敘述所組成。假設deptID的輸入值為2，請參考figure1，紅色粗線即為程式執行的動線，當loc等於2時，程式的執行會跳過一部份，直接到case 2:的地方再加以執行，直到遇到break時，則跳出所屬的程式區塊，意即結束了這個switch敘述的執行。

Fig. 1: switch敘述執行動線示意圖(假設loc=2)

當程式具有類似此程式範例的處理需求時，使用switch敘述將可以讓程式的架構更為簡單易讀。請再思考看看，還有哪些應用適合使用switch敘述呢？

- 程式提供操作選項，而各選項負責執行不同的功能:

```
switch (choice)
{
   case 'i':
             insert_data();
             break;
   case 'x':
             execute();
             break;
   case 'q':
             exit(0);
             break;
```

```
}
```

- 國小學生週一至週五，每天下課時間不同，有時半天、有時整天：

```
switch (weekday)
{
   case 1:
   case 2:
   case 4:
   case 5:
           printf("After school at 4:00pm\n");
           break;
   case 3:
           printf("After school at 12:00am\n");
}
```

- 設計一程式，輸入一整數N□計算並印出1+2+...+N的結果：

```
int n=0, sum=0;
cint >> n;

switch (n)
{
   case 10: sum+=10;
   case  9: sum+=9;
   case  8: sum+=8;
   case  7: sum+=7;
   case  6: sum+=6;
   case  5: sum+=5;
   case  4: sum+=4;
   case  3: sum+=3;
   case  2: sum+=2;
   case  1: sum+=1;
}
cout << "Sum=" << sum << endl;
```

- 設計一程式，讓使用者輸入一個學生成績，再依據以下的規則輸出其對應的成績等第：
  - A□成績大於等於90
  - B□成績大於等於80但小於90
  - C□成績大於等於70但小於80
  - D□成績大於等於60但小於70
  - E□成績小於60

```
int score;
cout << "Please input your score: ";
cin >> score;

switch(score/10)
{
   case 10:
   case 9:  cout << "Your grade is A." << endl; break;
   case 8:  cout << "Your grade is B." << endl; break;
   case 7:  cout << "Your grade is C." << endl; break;
   case 6:  cout << "Your grade is D." << endl; break;
   default:  cout << "Your grade is E."<< endl; break;
}
```

7.4 條件運算式

C++語言還有提供一種特別的運算式，稱為條件運算式(Conditional Expression)，可依條件決定運算式的運算結果。最簡單的條件運算式語法如下：

條件運算式語法

測試條件 ? true值運算式 : false值運算式

條件運算式的運算結果依測試條件的結果而定 — 當測試結果為true時，以true值運算式的運算結果做為結果；當測試結果為false時，則以false值運算式的運算結果做為結果。

- 測試條件：運算結果為布林值的邏輯運算式，亦可為其它類型的運算式或數值，只要其值為布林值即可。
- true值運算式：當前述的測試條件為true時，所要執行的運算式，並以其運算結果做為此條件運算式的運算結果。
- false值運算式：當前述的測試條件為false時，所要執行的運算式，並以其運算結果做為此條件運算式的運算結果。

條件運算式可以思考為if-else敘述的縮寫

依據條件運算式的語法「測試條件 ? true值運算式 : false值運算式」，可以轉換為以下等價的if-else敘述：

```
if (測試條件)
   true值運算式;
else
   false值運算式;
```

關於true值運算式與false值運算式並沒有太多規範，任何運算式皆可，甚至是不包含任何運算子、只由單一運算元(變數、常數、數值、函式等皆可)所組成的運算式亦可。以下讓我們就一些使用範例進行討論(並為幫助讀者理解起見，我們也將對應的if-else敘述加以並列)。首先是一個比較簡單的使用情境：

使用條件運算式	使用if敘述	說明

`a=(b%2==0)?0:1;`	`if(b%2==0)`
	`  a = 0;`
	`else`
	`  a=1;`
		當b是偶數時，將a的數值設為0；當b不為偶數時，則將a的值設定為1。

`max = a>b?a:b;`	`if(a>b)`
	`  max = a;`
	`else`
	`  max=b;`
		讓max變數的數值等於變數a與b兩者中的最大值 — 依據a是否大於b，將a或b的數值做為此條件運算式的運算結果並賦值給變數max。

`score = score>=50?score+10:score;`	`if(score>=50)`
	`  score = score+10;`
	`else`
	`  score = score;`
		讓score已經大於(含)50分的同學加10分 — 若score大於等於50，將score=score+10，否則score值維持不變。

除了上述幾個簡單的示範以外，在條件運算式裡的true值運算式與flase值運算式也可以再寫成一個條件運算式，也就是形成在條件運算式裡還有其它條件運算式的複合式使用，請參考下面的例子：

```
score = score<50? score : score<=90? score+10 : 100;
```

綜合上述的範例，條件運算式都是放在賦值運算子的右側，並將其運算結果做為左側變數的數值。但是我們其實也可以把條件運算式放在賦值運算子的左側：

```
int a, b;
cin >> a >> b;
(a<b?a:b)=0;
```

由於條件運算式(a<b?a:b)會視a與b的數值決定採用a或b做為此運算式的值，所以它的值不是a就是b，然後再使用=0，把a或b的值設定為0。

同樣的程式還可以這樣寫：

```
int a, b;
cin >> a >> b;
a<b?a=0:b=0;
```

在上面這個程式裡，視a是否小於b進行a=0或b=0的運算操作。我們甚至還可以再改寫如下：

```
int a, b;
cin >> a >> b;
a<b?cout << a:cout << b;
```

同樣是判斷a是否小於b但是直接在true值運算式與false值運算式裡，使用cout將結果輸出。相信讀者們透過上述這些範例，應該對條件運算式有了清楚的認識。

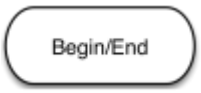
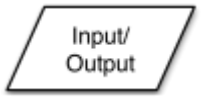
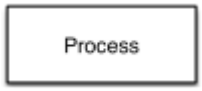
最後，讓讀者動動腦，想一想下面的運算式在做些什麼呢?

```
x = (x%10)!=0 ? (x-x%10+10) : x;
```

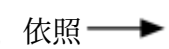
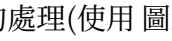
7.5 流程圖

隨著我們學習到愈來愈多C++語言的語法與功能後，過去曾用來思考程式設計的IPO模型似乎慢慢地跟不上需求，你將會發現有愈來愈多的程式無法只靠著IPO模型完成開發。本節將介紹另一個常用的程式設計思維工具 — 流程圖(Flowchart)透過它應該就可以涵蓋絕大多數的程式設計問題了。

流程圖是一種表達流程的圖示法，透過流程圖可以清楚地描述C語言程式的執行動線，非常適合使用它來規劃與設計程式的功能。以下我們先將流程圖的幾個重要圖示進行說明，請參考table 6:

圖示	意義
	程式或功能的開始或結束
	資料輸入或輸出
	邏輯與資料處理
	決策(依特定條件決定後續的程式動線)
	流程(也就是程式的動線)

Tab. 6: Relational Operators

table 6僅節錄部份常用的圖示，更完整的流程圖圖示請自行參考相關資料。figure 2是將IPO模型以流程圖加以表達，從圖中可看出，自  圖示開始，依照  的指示展開資料收集的動作(使用  圖示代表資料輸入)，後續再依照  展開資料的處理(使用 圖示表示資料的處理)，再接下來則是資料的輸出(與輸入一樣，都使用  圖示)，以及最後的結束圖示(也就是  圖示)。在這個例子中，程式的執行由開始到結束都是使用單一的動線。

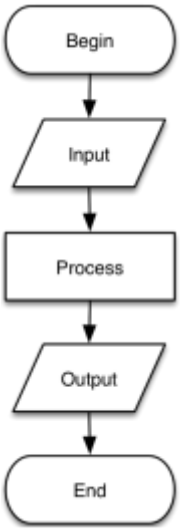


Fig. 2: 使用流程圖來表達一般的IPO模型

現在讓我們以一個簡單的例子來示範流程圖應用在程式設計的做法，首先請看figure 3(a)程式開始後隨即進行資料的取得，我們在 圖示中標示了「取得x(取得變數x)接著依照 的指引，接著進行的是處理的部份，本例在 圖示中，標明了「將x乘上2」，接下來則是再使用 圖示，將x的值印出(其中標示為「輸出x)然後程式的執行就到此結束。從這個例子中可以看出，流程圖是一種圖示法的表達工具，透過其 的指引，程式的執行動線以及輸入/輸出處理等全部都可以被清楚地表達。請再繼續參考figure 3(b)我們還可以把同樣的例子改成使用C語言的程式碼加以表達，如此一來，使用流程圖就如同IPO模型一樣，也能夠輕易地轉換為C語言的程式。

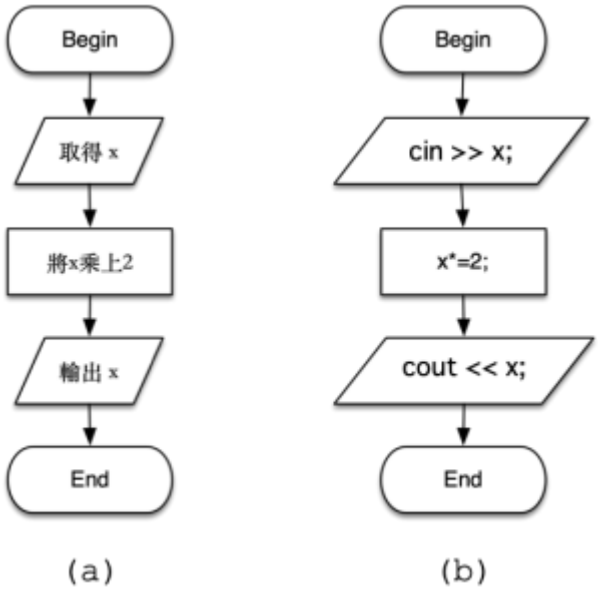


Fig. 3: 一個簡單的流程圖範例（將取得的資料，乘以2後輸出）

由於流程圖是由簡單的幾種圖示來分別表達程式相關的資料輸入與輸出、處理、以及決策等程式設計要素，它所可以組合出的程式邏輯與處理流程比起IPO模型來得更為豐富的多，現在讓我們使用流程圖把前面所討論過的滿五千折五百程式範例分析如下，請參考figure 4

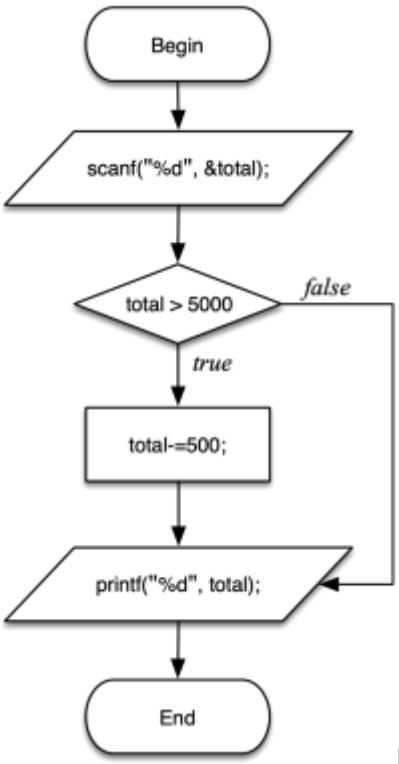


Fig. 4: 滿五千折五百的流程圖範例

[1)] Corrado Böhm and Giuseppe Jacopini,“Flow diagrams, Turing machines and languages with only two formation rules.” Communications of the ACM, Vol.9, pp. 366–371. 1966.
[2)] 使用非0整數與0代表布林數值是承襲自C語言的做法。