

第四章

結構化程式設計的流程控制





本章內容

4-1 流程控制的三種結構

4-2 循序結構(Sequential)

4-3 迴圈結構(Loop)

4-4 選擇結構(Switch)

4-1 何謂結構化程式設計

雖然JAVA是屬於物件導向程式設計的撰寫方式，但是，在撰寫程式的技巧上，還是以「結構化」的基本結構方式來撰寫。何謂結構化程式設計呢？它是利用「由上而下」技巧，將程式分解成多個具有獨立功能的模組，再使用「模組化程式設計」方式。如圖4-1所示：

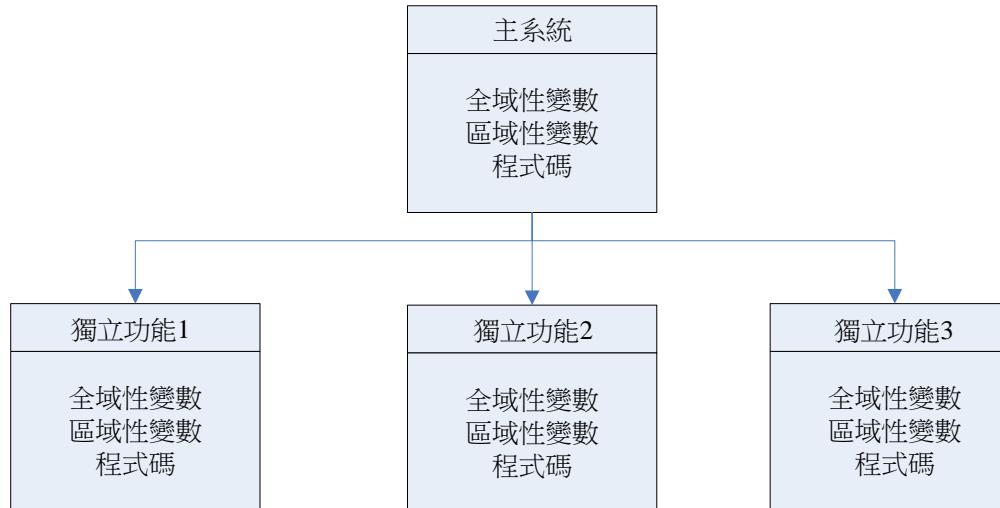


圖 4-1 模組化程式設計的示意圖

將每一個模組的功能單元自成(僅用三種基本結構：循序、選擇及迴圈)一段程式，最後再做整合測試的一種程式設計的方法。如圖4-2所示：

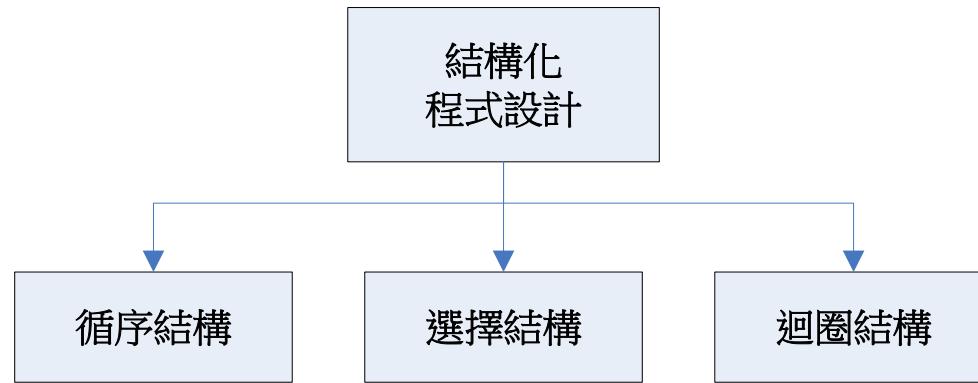


圖 4-2 結構化程式設計分類圖



一、優點：

1. 程式容易閱讀與了解。
2. 減少程式維護成本。
3. 減少程式邏輯錯誤。
4. 提高程式設計的生產力。
5. 降低程式的複雜性及測試時間。

二、缺點：

1. 程式比較多，故比較佔用記憶體空間。
2. 因為程式多，故執行速度比較慢。

三、使用結構化程式設計必須遵守下列幾個原則：

1. 每一種結構只能有一個入口及出口。
2. 少用goto的指令。
3. 採用「由上而下」的觀念設計程式。

【實例】某一所大專院校的計算機中心，欲開發一套「網路教學」系統，以提供學生加值的服務。其計算機中心首先必須要將該系統細分為：學生介面、老師介面及管理者介面等功能，並且再依照不同的介面細分各種不同身份的子系統。如圖4-3所示：

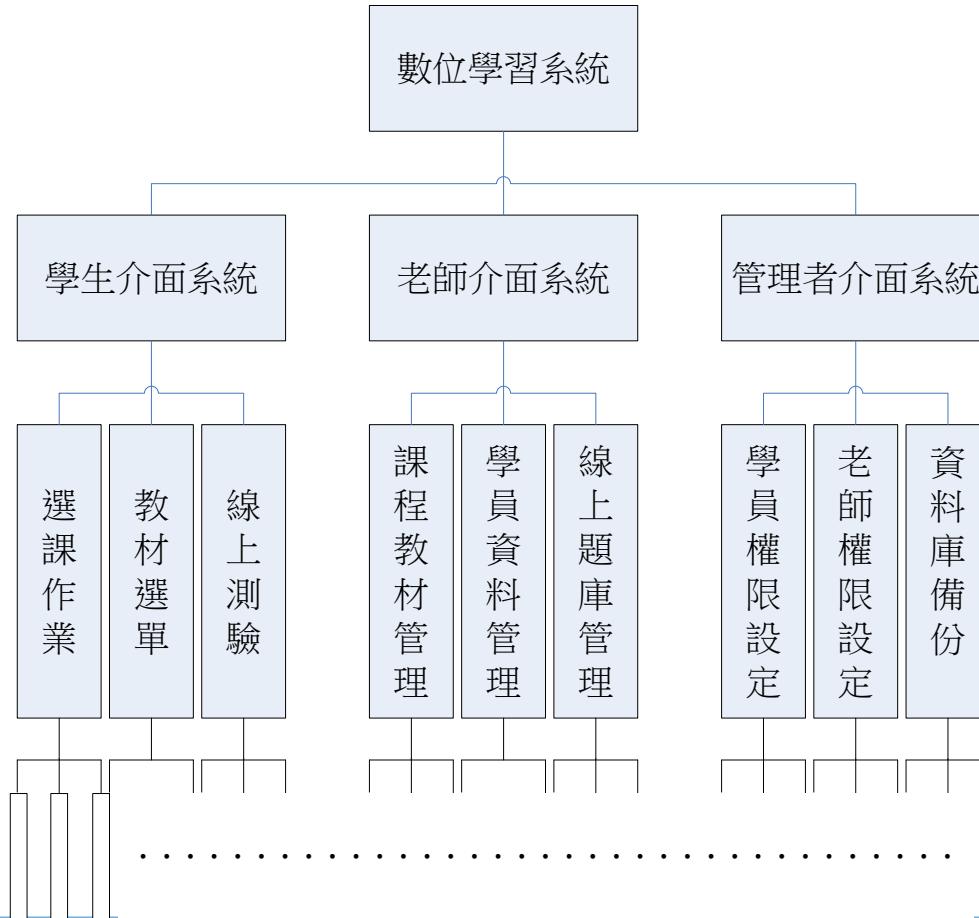


圖 4-3 真理模塊化設計的實例分析圖



4-2 流程圖的介紹

初學者在學習程式設計之前，最好先利用流程圖方式來表示問題的步驟，這對於程式邏輯比較容易理解。雖然程式設計時，不一定要畫流程圖，但是當我們遇到比較複雜的問題時，如果沒有事先規劃出解決問題的步驟，很容易產生錯誤，因此筆者非常建議初學者一定要養成繪製流程圖的習慣。

通常一個較專業的程式設計師，在繪製流程圖時，常利用專業的軟體來製作，例如Microsoft Visio繪圖軟體，它有提供非常多的工具來使用。如圖4-4所示：

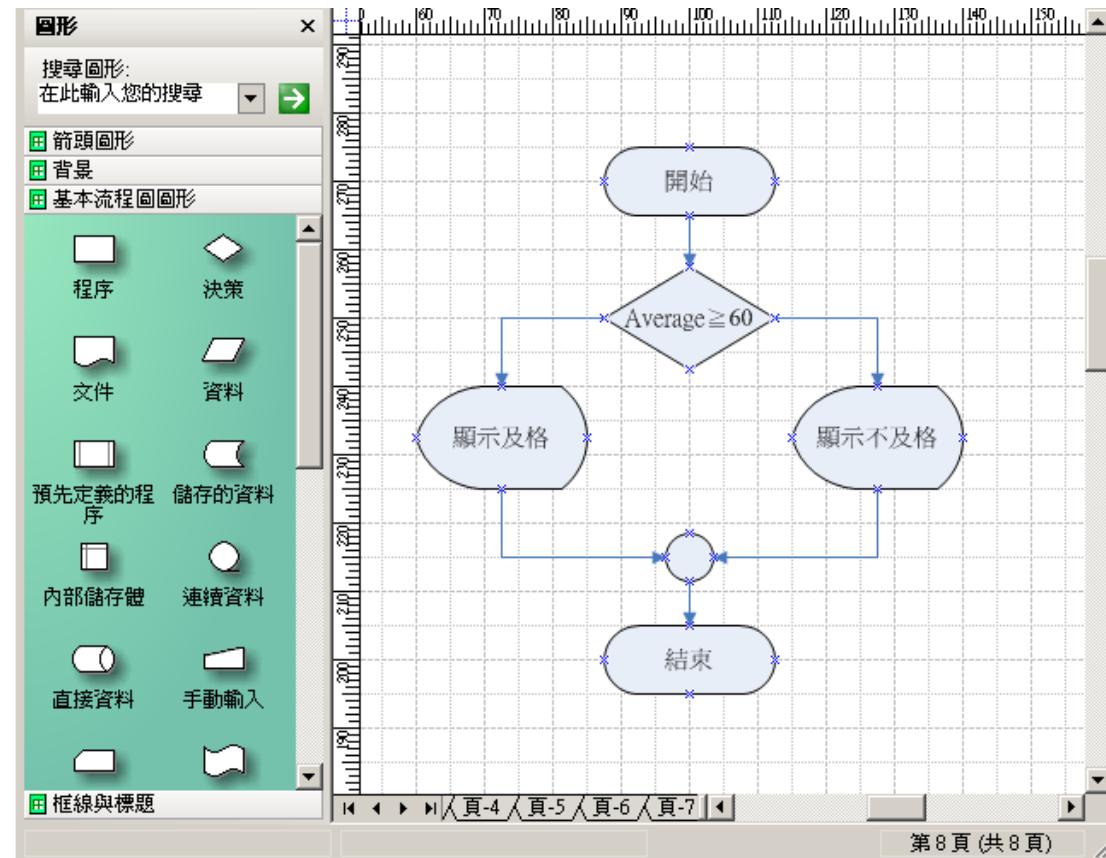


圖 4-4 繪製流程圖工具

何謂流程圖(Flow Chart)？它是利用各種專門的圖形符號，以圖表的方式來說明整個問題的邏輯、推演的方法及步驟。而在我們要開始程式設計時，一定要先了解下面幾個步驟：

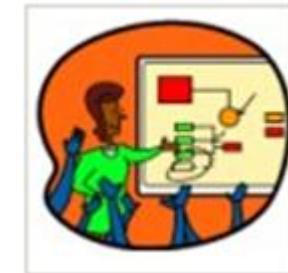
1. 分析所要解決的問題

- (1)首先要瞭解問題的需求及條件。
- (2)確定要輸入那些資料。
- (3)確定要輸出那些資訊報表。



2. 設計解題的步驟

撰寫演算法可以利用文字敘述、「流程圖」或虛擬碼來表示解決問題的步驟。



3.編寫程式

選擇適當的程式語言，將演算法的步驟寫成一個完整的程式。



4.上機測試、偵測錯誤

一個有用性、易用性的程式，必須要經過多次的測試，若有錯誤，立即更正，直到正確無誤為止。



5. 編寫程式說明書

一個功能強而完整的程式，使用者就會願意使用，因此必須有使用說明書，以便於別人使用或日後的維護；一般而言，在程式書面資料中，一般包括有下列三項：

- (1) 程式的功能、輸入需求及輸出格式。
- (2) 演算法或程式流程圖。
- (3) 測試結果或數據。



◆ 其中畫出流程圖是我們在本章節中研究的範圍，而它有下列幾個重要性：

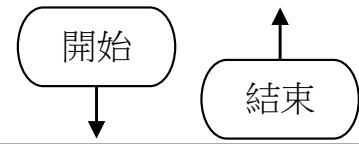
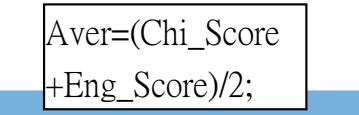
- 1. 讓程式設計師了解整個程式的流程。
- 2. 有助於程式的維護及除錯。

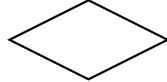
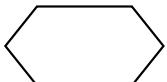
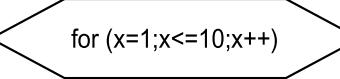
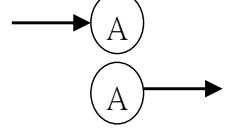
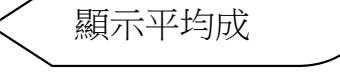
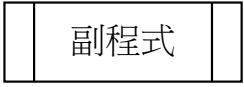
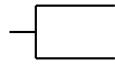
4-2.1 流程圖常用的符號表



當我們想利用電腦程式語言來處理問題時，先要了解問題並想出許多方案來解決問題，並且分析那些資料是要「輸入」，經過「處理」之後，要「輸出」那些結果。因此，我們必須先利用「流程圖」來進行分析與設計，以便快速了解整個程式來解決問題的流程，這對於往後程式在偵錯及維護上有莫大的助益。如表4-1所示。

表4-1 流程圖常用的符號表

符號	名稱	意義	實例說明
	開始 / 結束符號	表示流程圖的起點或終點，每一個流程圖只有一個起點，但可以一個以上的終點。	
	輸入 / 輸出符號	表示資料的輸入或輸出。	
	處理符號	表示程式正在執行。	

	決策符號	表示條件式是否成立與否。	
	迴圈符號	固定次數或前測試迴路及測測試迴路。	
	連結符號	表示流程圖的出口或入口的連接點。	
	流向符號	表示程式所執行的方向。	
	顯示符號	表示顯示輸出的結果到螢幕	
	預定處理符號	表示已定義的副程式	
	註解符號	此符號可對程式加一些說明	



4-2.2 繪製流程圖的原則

一般而言，要繪製一個好的流程圖，必須符合下面幾個原則：

1. 流程圖必須使用標準符號，便於閱讀和分析。
2. 流程圖中的文字力求簡潔、扼要，而且明確可行。
3. 繪製方向應由上而下，由左至右。
4. 流程線條避免太長或交叉，可多用連接符號。

【實例】欲設計一個計算國文與英文的平均成績，並依照平均成績來分成五個等級，其所需進行的步驟為何？

【解答】

1. 分析及定義問題。



五個等級分別如下：

- (1)甲等的範圍：90(含)以上。
- (2)乙等的範圍：80~89。
- (3)丙等的範圍：70~79。
- (4)丁等的範圍：60~69。
- (5)戊等的範圍：60以下。

2. 畫出整合問題的流程圖或撰寫問題的演算法。如圖4-5所示：

在繪製流程圖時，必須要考慮資料的處理流程，亦即「輸入---處理---輸出」三大步驟。

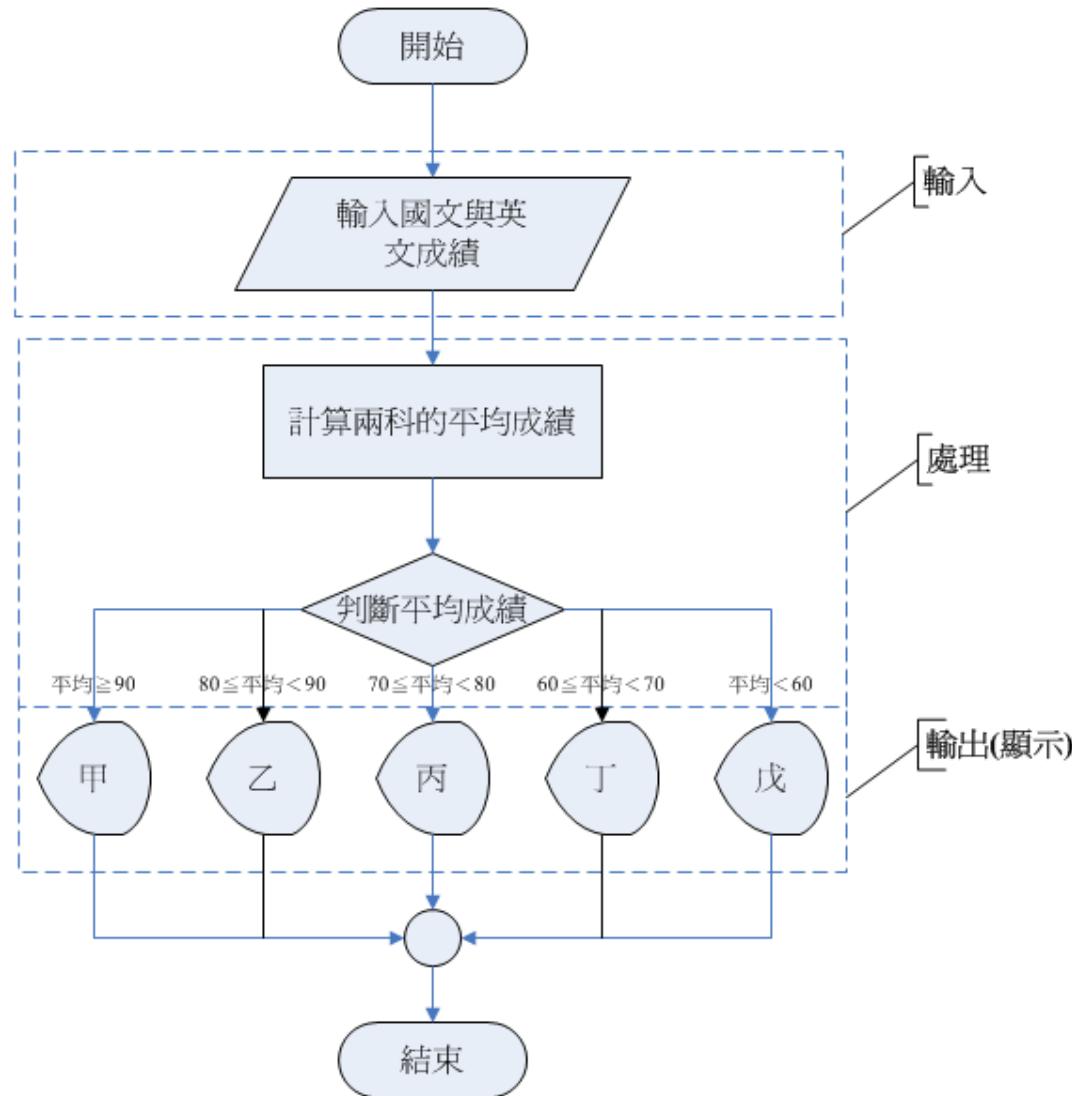


圖 4-5 整合問題的流程圖

3. 撰寫及建立程式模組。

4. 對每一個程式模組進行測試及除錯，直到沒有錯誤為止。

當使用者輸入國文為60分，英文為61分時，是否可以計算出平均成績為60.5，如果沒有則必須要進行除錯，亦即要將Average的資料型態改為double(倍精準度)

5. 建立操作說明文件。

牛刀小試：請繪出使用者登入帳號與密碼時，系統檢查的流程圖。

提示：

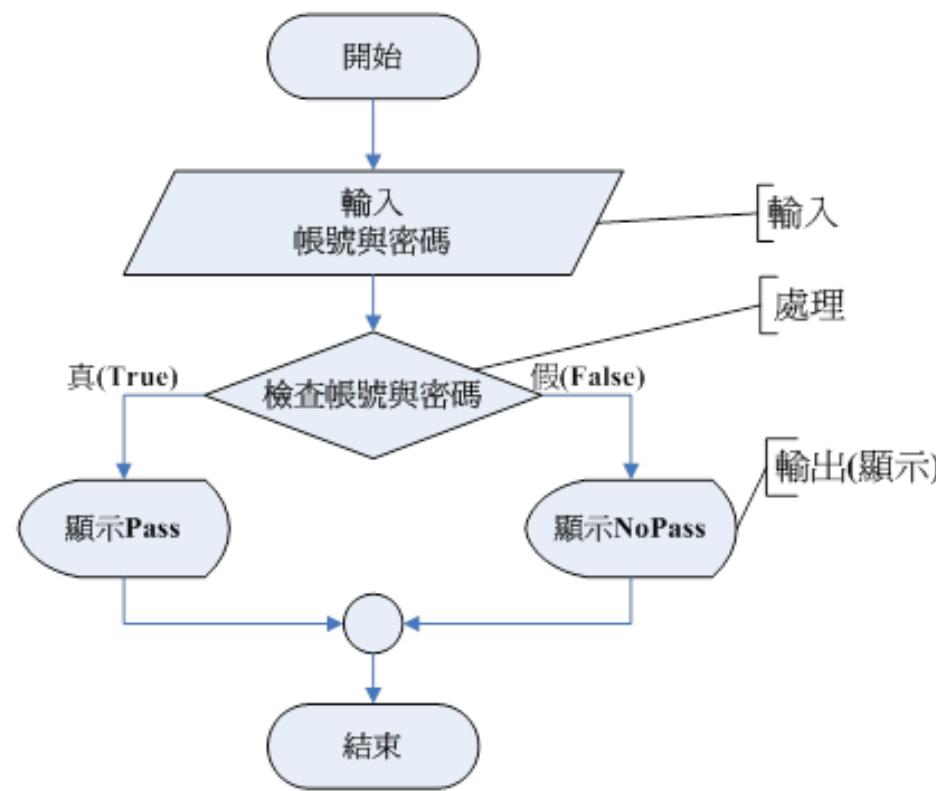
步驟1：輸入帳號與密碼

步驟2：檢查是否正確

步驟3.1：正確時，則顯示Pass。

步驟3.2：不正確時，則顯示NoPass。

【解答】



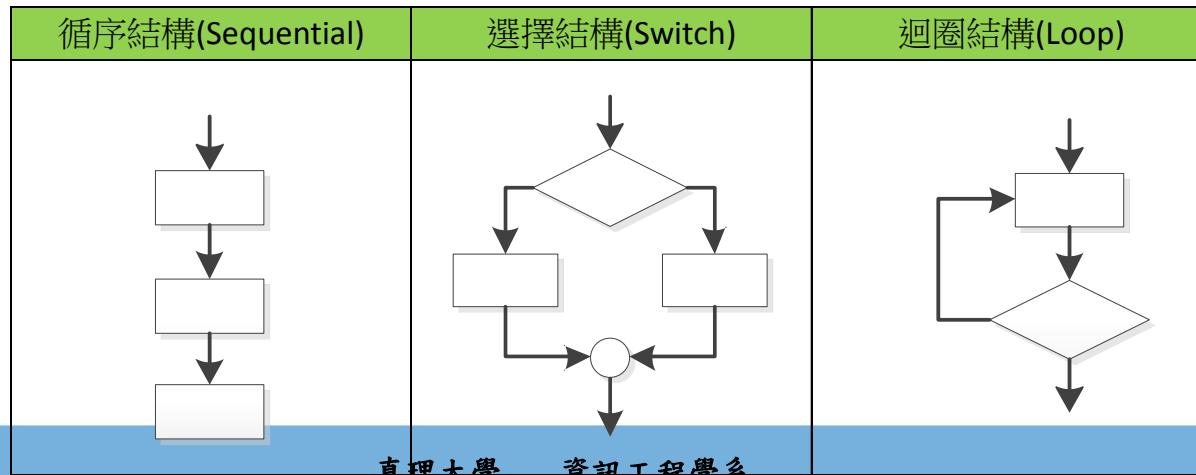
4-3 流程控制的三種結構



【引言】

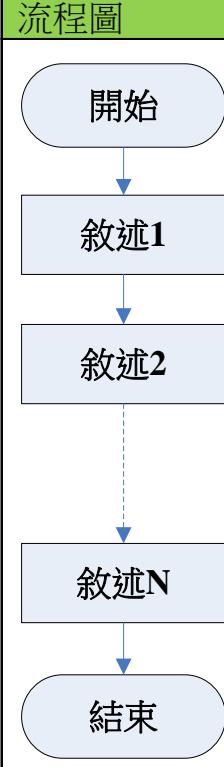
當我們在撰寫JAVA程式時，往往會依照題目的需求，必須要撰寫一連串的指令區塊，並且當某一事件發生時，它會根據「不同情況」來選擇不同的執行動作，而且要反覆的檢查環境變化。因此，我們想要完成以上的程序，就必須要學會程式的流程控制的三種結構。

【流程控制的三種結構】



【說明】

1. 循序結構(Sequential)：是指程式由左至右，逐一執行。

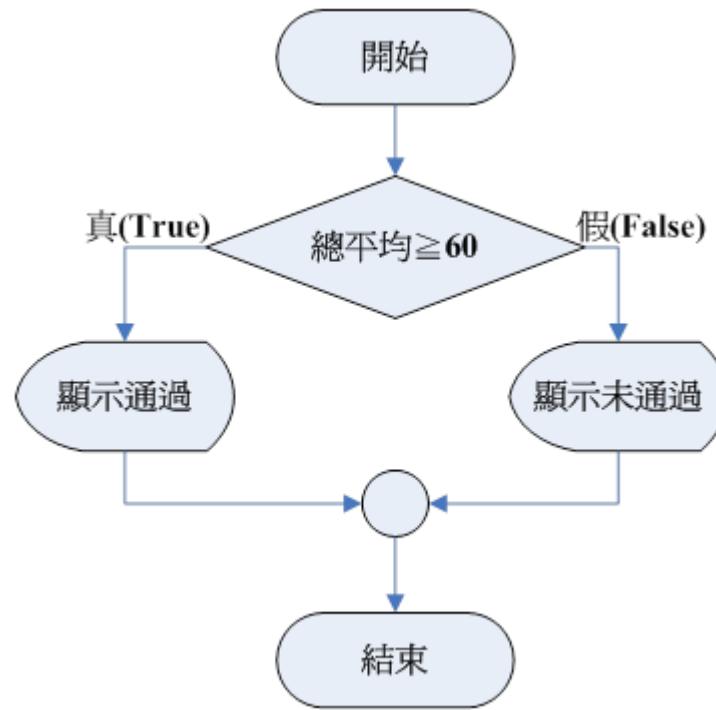
流程圖	程式碼
 <pre>graph TD; Start([開始]) --> Step1[敘述1]; Step1 --> Step2[敘述2]; Step2 --> StepN[敘述N]; StepN --> End([結束]);</pre>	<pre>public static void main(String[] args) { int A,B,C,D; A=20; //將 20 指定給 A B=40; //將 40 指定給 B C=A+B; //將 A 與 B 的值相加後，再指定給 C D=A*B; //將 A 與 B 的值相乘後，再指定給 D System.out.println("C=" + C); System.out.println("D=" + D); }</pre>

2. 選擇結構(Switch)：是指根據「條件式」來選擇不同的執行路徑。

【例如】老師想要了解那些同學及格與不及格時，如果只利用循序結構時，則只能顯示全班的平均成績，但無法顯示那些同學及格與不及格。那我們就必須要使用「選擇結構」的方式了。

學生成績單							
學號	姓名	國文	英文	數學	程式設計	總平均	結果
001	張三	90	90	90	90	90	及格
002	李四	60	50	60	30	50	不及格
003	王五	80	80	80	80	80	及格

【流程圖】



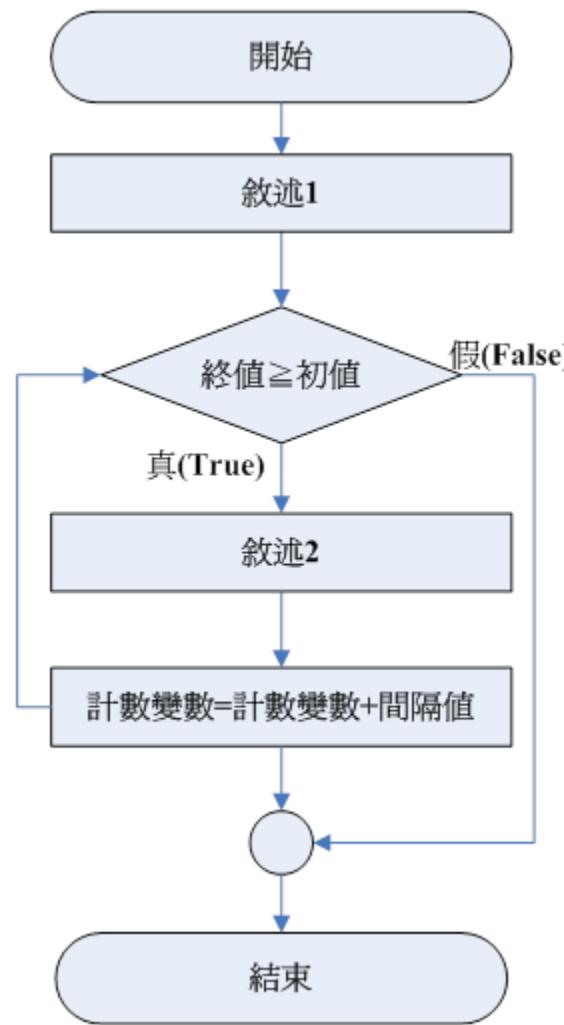
3. 迴圈結構(Loop)：是指某一段「拼圖方塊」反覆執行多次。

【例如】我們想要撰寫一段程式，來輸出 $1, 2, \dots, 100$ 時，怎麼辦？目前最少有兩種方法可以使用。如圖4-8所示：

循序結構	迴圈結構
System.out.println("1"); System.out.println("2"); System.out.println("3"); System.out.println("100");	for (i=1;i<=100;i++) System.out.println(i);

【說明】利用「循序結構」必須要撰寫100行程式碼，但是如果改用「迴圈結構」，則只須要撰寫2行，即可達到此功能。

【流程圖】



4-4 循序結構(Sequential)

【定義】是指程式由左至右，逐一執行一連串的拼圖程式，其間並沒有分岔及迴圈的情況，稱之。

- 【優點】**
- 1.由左至右，非常容易閱讀。
 - 2.結構比較單純，沒有複雜的變化。

- 【缺點】**
- 1.無法表達複雜性的條件結構。
 - 2.雖然可以表達重複性的迴圈結構，但是往往要撰寫較長的拼圖程式。

【適用時機】

- 1.不須進行判斷的情況。
- 2.沒有重複撰寫的情況。

【實例】請計算國文(chi)與英文(eng)成績兩科的平均成績(average)。

分析：(1)輸入：國文(chi)與英文(eng)成績

(2)處理：計算平均成績(Average)

(3)輸出：顯示結果



【解答】

程式檔案名稱		ch4-4A.java
01	import	java.util.Scanner; //載入Scanner類別
02	public	class ch4_4A
03	{	
04	public	static void main(String[] args)
05	{ //輸入	
06	Scanner	inScore=new Scanner(System.in);
07	System.out.print	("請輸入國文成績 : ");
08	String	chiScore=inScore.next(); //國文成績
09	System.out.print	("請輸入英文成績 : ");
10	String	engScore=inScore.next(); //英文成績
11	//處理	
12	int	chi=Integer.parseInt(chiScore);
13	int	eng=Integer.parseInt(engScore);
14	int	average=(chi+eng)/2;
15	//輸出	
16	System.out.println	();//換行
17	System.out.println	("國文 : " + chi);
18	System.out.println	("英文 : " + eng);
19	System.out.println	("平均 : " + average);
20	}	
21	}	

【說明】

行號01：載入Scanner類別套件，用來提供使用者輸入資料所需要的物件及方法。

行號06：利用Scanner類別來建立inScore物件，並需要使用System.in物件。

行號07：利用System.out.print()方法來提醒使用者「輸入國文成績」。

行號08：利用Scanner類別的next()方法來取得使用者輸入的成績。

行號09：利用System.out.print()方法來提醒使用者「輸入英文成績」。

行號12~13：利用Integer.parseInt()方法，將字串資料轉換成整數。

行號16：利用System.out.println()方法來「換行」

行號17~19：顯示國文、英文成績及平均成績。



【執行結果】

請輸入國文成績：80

請輸入英文成績：90

國文：80

英文：90

平均：85

【隨堂練習】請設計攝氏轉換華氏的程式。

分析：(1)輸入：攝氏(C)

(2)處理：轉換公式為 $F=C*9/5+32$

(3)輸出：顯示華氏

【解答】

程式檔案名稱	ch4-4B.java
	<pre>01 import java.util.Scanner; //載入Scanner類別套件 02 public class ch4_4B 03 { 04 public static void main(String[] args) 05 { //輸入 06 Scanner inTempC=new Scanner(System.in); 07 System.out.print("請輸入攝氏C : "); 08 String TempC=inTempC.next(); //攝氏C的溫度 09 //處理 10 int C=Integer.parseInt(TempC); 11 int F=C*9/5+32; 12 //輸出 13 System.out.println();//換行 14 System.out.println("攝氏C :" + C); 15 System.out.println("華氏F :" + F); 16 } 17 }</pre>

【執行結果】

請輸入攝氏 C : 30

攝氏 C : 30

華氏 F : 86

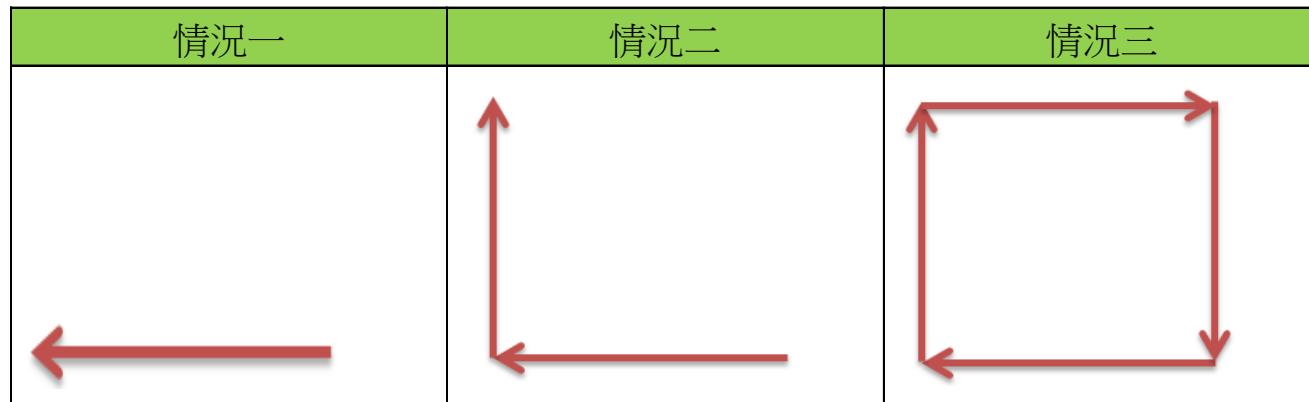
【延伸學習】在機器人設計上的應用

情況一：讓機器人馬達前進2圈後，自動停止。

情況二：讓機器人馬達前進2圈後，向右轉，再向前走2圈

情況三：讓機器人繞一個正方形。

【圖解說明】



在上圖中，「情況三」作法雖然可以使用「循序結構」，但是，程式會較長，並且非常不夠專業。因此，建議使用「迴圈結構」。

【機器人繞一個正方形】的兩種方法之比較：

【第一種方法】循序結構(沒有使用迴圈)

行號	使用 Java 撰寫機器人程式(LeJOS)
01	<code>import lejos.nxt.Motor;</code>
02	<code>public class ch5_2A {</code>
03	<code> public static void main(String[] args) {</code>
04	<code> //前進2圈後，再向右轉</code>
05	<code> Motor.B.rotate(720,true);</code>
06	<code> Motor.C.rotate(720,false);</code>
07	<code> Motor.B.rotate(1200,false);</code>
08	<code> //前進2圈後，再向右轉</code>
09	<code> Motor.B.rotate(720,true);</code>
10	<code> Motor.C.rotate(720,false);</code>
11	<code> Motor.B.rotate(1200,false);</code>
12	<code> //前進2圈後，再向右轉</code>
13	<code> Motor.B.rotate(720,true);</code>
14	<code> Motor.C.rotate(720,false);</code>
15	<code> Motor.B.rotate(1200,false);</code>
16	<code> //前進2圈後，再向右轉</code>
17	<code> Motor.B.rotate(720,true);</code>
18	<code> Motor.C.rotate(720,false);</code>
19	<code> Motor.B.rotate(1200,false);</code>
20	<code>}</code>
21	<code>}</code>

【說明】以上共有八個，但是，重複出現四次「前進2圈，向右轉」。

【第二種方法】使用「Loop迴圈」結構

【JAVA程式】

行號	使用 Java 撰寫機器人程式(1eJOS)
01	<code>import lejos.nxt.Motor;</code>
02	<code>public class ch5_2B {</code>
03	<code> public static void main(String[] args) {</code>
04	<code> int i=0;</code>
05	<code> while(i<4)</code>
06	<code> {</code>
07	<code> //前進2圈後，再向右轉</code>
08	<code> Motor.B.rotate(720,true);</code>
09	<code> Motor.C.rotate(720,false);</code>
10	<code> Motor.B.rotate(1200,false);</code>
11	<code> i++;</code>
12	<code> }</code>
13	<code> }</code>
14	<code>}</code>

【說明】

將【第一種方法】中的前3個指令「前進2圈，向右轉」抽出來，外層加入一個「for迴圈」來控制迴圈的次數即可。

【註】以上的機器人的應用，詳細介紹請參考第十一章「開發NXT樂高