高級中學科學課程試用教材
電 子 計 算 機
第五章　程式基本概念

5-1 資料與變數

從上一章我們了解計算機如何儲存資料，在這一章我們要進一步討論資料處理的問題。計算機所要處理的資料可歸納爲兩類：

1. 數字資料

整數	如	247	−1982
定點數	如	46.8	−25.0
浮點數	如	7.09E−06	−0.9432 E 05

2. 文字資料

字串　如“ ABC”“ MICROCOMPUTER ”

一般而言，資料都儲存在記憶單元裏，必要時才把它們找出來，放到算術邏輯運算單元加以運算，然後把結果放回記憶單元。換句話說，對記憶單元要不斷的做“ 讀 ”和“ 寫 ”的工作。這些資料在記憶單元裏都有自己的位址，如果每次都要把位址講清楚，使用計算機就變成了一件煩人的事，因此人們才想到要利用符號來代表資料的位址，這類符號稱爲變數（ variables ），而變數所代表的實際位址就交給計算機系統去處理。

寫程式的人可以自己選擇變數的符號，通常用數個字符表達，但第一個字符必須是英文字母，例如X，VALUE，A1，DATE，COUNT等都可做爲變數的符號，最好是具有字面上的意義，便於閱讀和指認。

5-2 基本運算

一般計算機能做下列三種運算：

1. 算術運算

對數字做加法、減法、乘法、除法和次方就是做算術運算，它們的符號和優先順序如下：

算術運算	代表符號
次方	∧或↑
負號	−
乘，除	*，／
加，減	＋，−

下面左邊的代數式，必須利用運算符號寫成右邊的算術式，才能在計算機上運算。

$x+2y$	x + 2 * y
$x-\frac{y}{z}$	x − y ／ z
$\frac{xy}{z}$	x * y ／ z
$\frac{x+y}{z}$	(x + y) ／ z
$(x^2)^y$	(x ∧ 2) ∧ y
x^{y^z}	x ∧ (y ∧ z)
x (− y)	x * (− y)

在上面的例子裏，有些地方使用了小括弧，而某些地方則省略掉，這樣會不會產生混淆呢？當我們做計算的時候，習慣上是按照優先順序從左到右，但如果有必須先做的部分，就用括弧括起來，而如果有兩個以上的運算沒用括弧分開的話，就按照運算的優先順序去做，這樣一來就不會產生困擾，且可以避免有太多的小括弧。例如（ x + y ）／ z 代表的是 $\frac{x+y}{z}$，而 x + y ／ z 代表的是 x+(y/z) 。

2. 關係運算

關係運算用以比較兩項資料的大小，也是一種基本運算，在計算機上用下列符號表示：

關係運算	代表符號
等於	=
小於	<
大於	>
小於等於	<=
大於等於	>=
不等於	<>

例、要判別一個二次方程式是否有兩個實根，可利用下面的一些關係式子：

B ∧ 2 − 4 * A * C > 0
B ∧ 2 − 4 * A * C = 0
B ∧ 2 − 4 * A * C < 0
B ∧ 2 > 4 * A * C
B ∧ 2 = 4 * A * C
B ∧ 2 < 4 * A * C

例、字串可按字典排列次序比較大小，如

" JOHN " < " MARY "
" ABCD " > " ABC "

3. 邏輯運算

NOT（非）、AND（且）、OR（或）等叫做邏輯運算，用來組合簡單的關係式子，使成爲複雜的關係式子。運算順序類似算術運算，按照（　）、NOT、AND、OR 的順序，由左至右運算。

例、X＞0　AND　Y＜0

（L＞M　AND　L＞N）OR（M＞L　AND　M＞N）　NOT　COUNT＜＝0

都是關係式子。

5-3 演算法則

要利用計算機幫我們處理資料時，必須把我們的意思透過某種媒介告訴計算機，而這個媒介就是計算機程式（computer program），換言之，我們的責任是程式設計（programming），而計算機則負責程式的執行（execution）。就像一件工程要順利進行，必須有很好的準備，程式設計如果縝密週全，則設計出來的程式才能夠正確有效的執行，以達到解決問題的目的。

先看一個很簡單的例子：例如有三個數字，希望把其中最大的找出來，應該怎樣進行？首先，用三個變數L，M，N代表（或說是儲存）這三個數字，先比較L和M的大小，如果L大的話，則比較L和N，否則就比較M和N，經過這些過程，最大的數字自然就找出來了。如果上面的構想要計算機代爲執行，可做下面的安排，讓計算機一步一步的去做：

輸入　L，M，N，

若L＞M則做：

若L＞N則輸出L否則輸出N；

若L＜M則做：

若M＞N則輸出M否則輸出N。

這裏仍然有兩個模糊不清的地方，在執行第二行的時候，如果L比M小的話，計算機怎麼知道要跳到第四行呢？如果做第三行，做完之後是否要做第四行呢？爲了避免這些困擾，可以給每個步驟一個行號（line number），如果要跳行，就標明跳到那一行。依照這個原則，上面的安排就可以改寫成：

1　輸入L，M，N

2　若L＜M則跳到6

3　若L＞N則輸出L　跳到8

4　輸出N

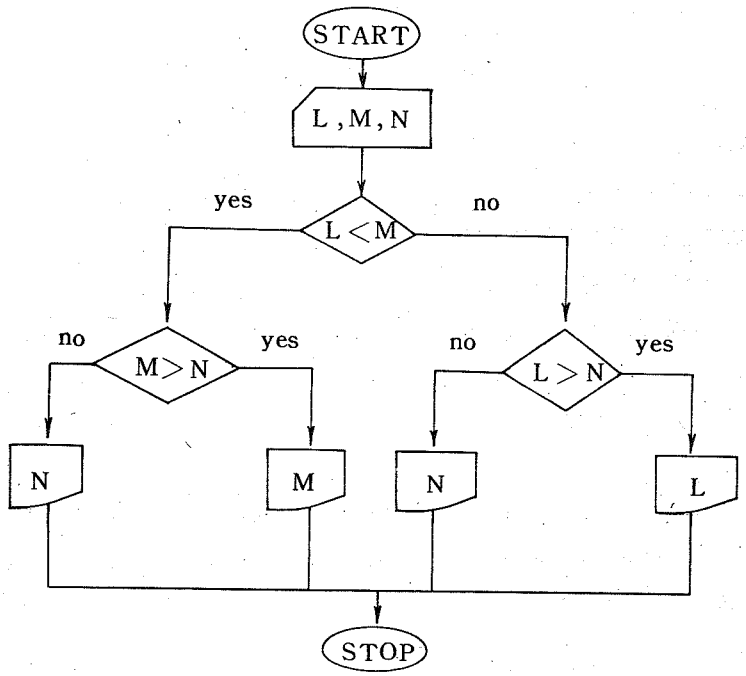
5　跳到8

6　若M＞N則輸出M　跳到8

7　輸出N

8　停止

讀者可以看出，我們的設計一次比一次嚴密，這種過程很自然且很有條理，同時也不容易有疏漏。第三個設計也可以用一種流程圖（flow-chart）來表示，在視覺上或許比較直接：



流程圖中 ⬭ 代表開始或結束，◇ 代表關係運算，▱ 代表輸入，▱ 代表輸出，→ 則代表步驟與步驟間的流程方向。

不管是用文字敍述或用流程圖，它們所表示的都是演算的實際過程，稱之爲演算法則（algorithm）。既然一個演算法則是用來描寫解決一個問題的整個過程，它就必須具備有幾個重要的特性：

1. 由有限個步驟構成。
2. 連接各步驟的流向明確。
3. 每一步驟意義明確而可執行。
4. 開始執行後，經過有限個步驟結束。

有了以上的認識後，我們再來看一個例子：如果輸入一個正整數 n，而欲求出 1，2，3，……到 n 的和，應該怎麼做？首先會想到使用幾個變數，輸入的數字是其中之一，設它爲 N，而最後求出的和設爲 SUM。剛開始 SUM 爲 0，因爲還沒有任何數加進去。要加進去的數從 1 開始，然後 2，3，…，也設一個變數 NUMBER 來代表。NUMBER 加到 SUM 後，NUMBER 本身要加 1，然後重覆上面的步驟，重覆多少次呢？那就視 n 而定，不能讓 NUMBER 超過 N。把以上的構想寫下來，就得到了一個演算法則：

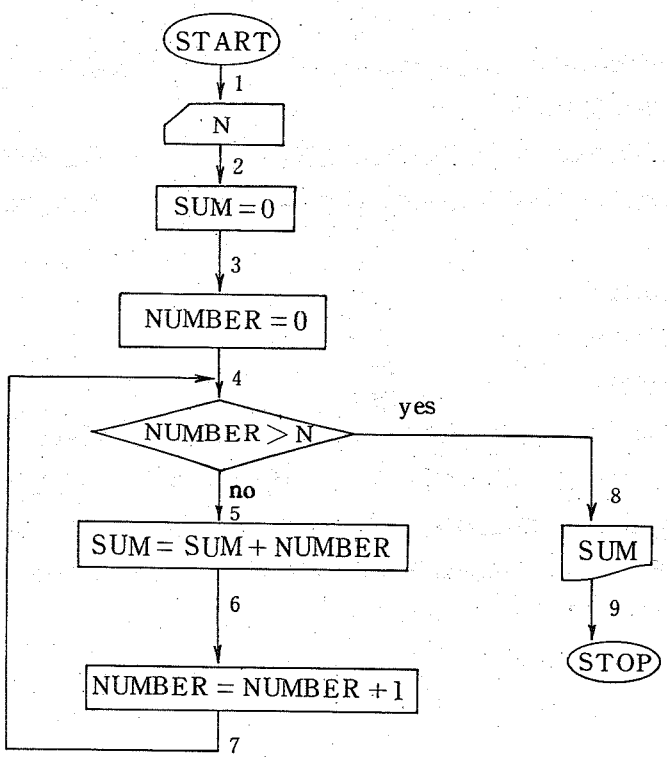
輸入 N
SUM 設爲 0
NUMBER 設爲 0
重覆下面兩個步驟直到 NUMBER ＞ N：
　更改 SUM 爲 SUM+NUMBER
　更改 NUMBER 爲 NUMBER + 1

輸出SUM

將以上的敍述進一步改寫如下：

```
1  INPUT N
2  LET SUM = 0
3  LET NUMBER = 0
4  IF NUMBER > N GOTO 8
5    LET SUM = SUM + NUMBER
6    LET NUMBER = NUMBER + 1
7  GOTO 4
8  PRINT SUM
9  END
```

這裏把“輸入”、“輸出”、“跳到”、“停止”、“更改成”等寫成INPUT、PRINT、GOTO、END、LET 以符合下一章所介紹的程式語言的寫法。“＝”號右邊的運算結果用來改變左邊變數的內容。上面的演算法則也可以用流程圖表達，其中LET 步驟全部用 ▭ 表示。



由流程圖可看出，步驟7回到步驟4產生了一個循環的路徑，它恰好對應“重覆下面兩個步驟直到NUMBER＞N ”這個敍述，這種循環的路徑叫做迴路（ loop ）。

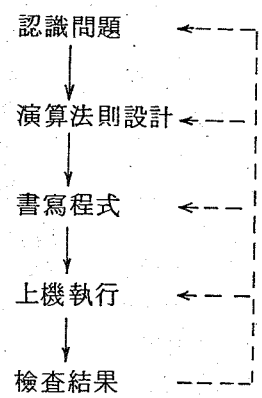
由步驟4跳到步驟8必須滿足NUMBER＞N 的條件，這種步驟叫做條件分支（ conditional branching），而步驟7跳到步驟4則沒有任何條件限制，這種步驟叫做無條件分支（ uncondi-

tional branching）。

分支和迴路是程式的基本架構，能夠駕馭分支和迴路，而將它們靈活應用的人，對程式設計的工作必定勝任愉快。

5-4 程式設計的步調

程式設計有人把它說成是程式發展，它的確是依靠累積的經驗，經由系統化的程序去創造程式的。我們可以用五個階段來概分其過程，底下用圖說明這五個階段的關係：



在“認識問題”階段把問題廓清，確定要得到什麼結果，然後在“演算法則設計”階段思考採取何種方式去獲得結果。“書寫程式”就是把演算法則轉換成正式的計算機程式。程式在計算機上執行，難免發現各種錯誤，而這些錯誤可能發生在上面任何一個階段，上圖中的虛線表示，在偵錯（debugging）之後，必須回到產生錯誤的地方加以更正，直到去除了所有的錯誤。

上一節所討論的，是眞正書寫程式上機之前的準備工作，所得到的是一個解決問題的演算法則，而非程式本身，但強調演算法則設計，可避免或降低上機後偵錯、更改的困難。爲了把演算法則轉換成程式，必須進一步學習程式語言，也只有完全按照程式語言規則所寫的程式，計算機才能了解而加以執行。

第五章　問題

1. 將下面的代數式利用計算機符號改寫成算術式：
   (a) $x(y+z)$
   (b) ax^2+bx+c
   (c) $c/x+yk^3$
   (d) $(1+\frac{1}{n})^n$
2. 以演算法則規畫本星期日的活動，在計畫中必須利用條件分支來表達你的決策。
3. 寫一個求二次方程式根的演算法則，其中三個係數都當作輸入的數值處理。