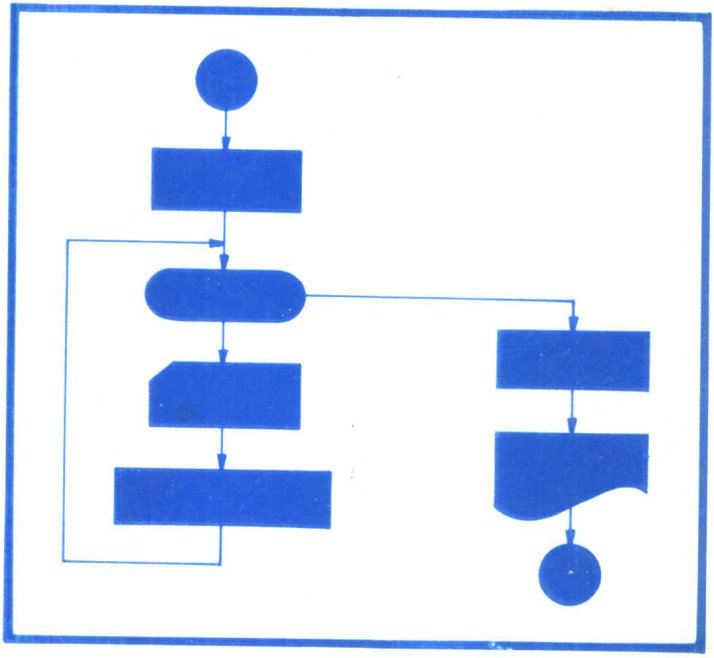
電子計算機科學叢書

郭德盛　主編

計算機科學概論

下　冊



原著

A. I. Forsythe
T. A. Keenan
E. I. Organick
W. Stenberg

李學養
編譯

田野出版社　印行

計算機科學概論

下　册

李學養 編譯

國立台灣大學

資訊工程學系系主任

田野出版社 印行

計算機科學概論

下 冊

每本定價新台幣 160 元整

編著者：李 學 養

發行人：陳 碧 玉

發行所：田野出版社

台北市仁愛路二段一一〇號三樓

電話：*3930255 · 3930249*

總經銷：松崗電腦圖書資料有限公司

台北市仁愛路二段一一〇號三樓

電話：*3930255 · 3930249*

郵政劃撥：**0109030-8**

經 銷：文 笙 書 局

台北市重慶南路一段六十九號

承印者：泉崗印刷設計股份有限公司

台北市仁愛路二段一一〇號三樓

電話：*3930255 · 3930249*

中華民國六十六年九月 初 版

中華民國六十九年九月 第二版

中華民國七十三年八月 第五版

本出版社經行政院新聞局核准登記，

登記證為局版台業字第〇二〇七號。

電子計算機科學叢書

郭德盛博士主編

1. 電子計算機名詞字典，許照校訂，郭德盛主編
2. 電子計算機程式語言——COBOL ，吳建平編譯
3. 數字方法，陳秋發編著
4. 電子計算機原理，郭德盛編著
5. 系統分析與設計，鍾英明編著
6. COBOL 程式設計範例，鍾英明編著，吳建平校閱
7. 電子資料處理，吳建平編著
8. 福傳程式設計技巧，陳秋發編著
9. 福傳程式設計，李學養編著
10. 微計算機基本原理，于惠中編譯
11. 商用程式語言COBOL，鍾英明、陳秋發合著
12. 計算機科學概論（上册），李學養編譯
13. 計算機科學概論（下册），李學養編譯
14. 資料結構，劉乃誠、吳建平合著
15. 培基語言—— BASIC，于惠中、周慶榮合著
16. 程式語言FORTRAN 77 ，郭德盛、許舜欽合著

目　錄

下冊

第八章　解譯與編譯

8.1　解譯與編譯

用**程序語言**（ Procedual Language ）如FORTRAN或ALGOL 寫的程式，可用下列二種方式之一，在計算機上執行。(1)程式的敍述，用**主控員**（ Master Computer ）及輔助機器，一次**解譯**（ Interpret ）一個敍述後，再執行之。也就是說，每當要執行一個敍述（流程圖框）時，先決定該敍述的意思，然後再予執行。像在廻圈的狀況，不管相同的敍述出現多少次，都得作相同的解譯工作。(2)程式的敍述可以**轉換**成一相當於機器語言（如SAMOS ）的指令群，構成**演算法**（ Algorithm ）的新表示方式。在這種方式中，機器所執行的，已不是原有的程式，而是轉換過的新程式。這種將程序語言寫的程式轉換成機器可以直接執行語言的過程，稱爲**編譯**（ Compilation ）。

第一種方式（即解譯）的最大好處是容易偵錯。由程式師的觀點看，程式好像是在一部高階層語言的機器上執行。實際上，程式師看不到此機器是由一低階層語言機器所摹擬（程式師無法看到相對應的低階層程式）。因此當執行時，程式如有錯誤，則此錯誤，以高階層語言機器的形式及執行的敍述表達。解譯過的程式和高階層敍述，則很少有直接關連。

第二種方式（即編譯）的主要好處是效率。機器語言程式的指令，可以由計算機硬體直接**解碼**（ Decode），但高階層語言的指令，一般須由程序（軟體）解碼，而這些程序本身則用機器語言表示。

目前在計算機設計方面的趨勢，是用硬體作高階層語言敍述的解碼，而只需很少的軟體。此種型式的機器綜合了解譯程式的彈性和編

譯程式的效率。有許多這種型式的機器，實驗型或商用型都已建造完成，而且可以執行像ALGOL， APL， COBOL， RPG等高階層指令。

不管是採用何種方法，編譯也好，解譯也好，或是二種的綜合形式，主要必須執行相似的演算法，以推論程式敍述的意義。本章强調這些分析和演算法結構的共同點。另外，上述在計算機設計的改變，其趨勢由漣漪而成大浪時，更多人會想了解這些新的計算機模型。本章將提供一些基本的觀念。

8.2 波蘭字串

在如SAMOS的機器上，編譯或解譯一個程式時，每一敍述是一個字元一個字元地掃視，以決定其意義。以編譯程式爲例，這種分析的目的，是爲被掃視的敍述，產生一串機器語言的指令。

例如，在SAMOS中，程序語言敍述

$$A \leftarrow A + B \times C$$

被轉換成如下的機器語言指令。（假定A，B和C的儲存位址爲1001，1002和1003）

±	運算指令		位　　址	附　　註
1	2 3 4	5 6 7	8 9 10 11	
	L D A		1 0 0 2	將B放在累積器
	M P Y		1 0 0 3	累積器的值乘以C
	A D D		1 0 0 1	把A加在累積器
	S T O		1 0 0 1	將累積器的值指定到A

即使採用2.3節中所用的改進表示法，也很難將慣用的數學式轉換成機器語言指令。

考慮下列的算術式

$$\frac{B+\sqrt{B^2-4\times A\times C}}{2\times A}$$

此式代表下面方程式一個根的負值

$$A\times X^2+B\times X+C=0$$

採用 2.3 節的修正符號，上述的表示變成

$$(B+(B\uparrow 2-4\times A\times C)\uparrow .5)/(2\times A)$$

對陳式中的運算符號，依運算的優先等級記上號碼。這是執行機器語言指令所須的次序。

```
(B+ (B↑2−4×A×C) ↑ . 5) / (2×A)
 ↑   ↑  ↑  ↑  ↑   ↑    ↑   ↑
 ⑥   ①  ④  ②  ③   ⑤    ⑧   ⑦
```

由此可見處理的困難，機器必須對此數學式，作多次往返的掃視，俾能發現下一執行的運算。如果去掉運算間的優先關係，而祇用括號決定其次序，則困難依然存在。如在APL中，不用運算優先等級，而由右至左掃視，則有下列表示：

```
(B+ ( (B↑2) − ( (4×A) ×C) ) ↑ .5) / (2×A)
 ↑   ↑   ↑    ↑   ↑   ↑   ↑   ↑
 ⑦   ④   ⑤    ②   ③   ⑥   ⑧   ①
```

要消除此困難，機器將算術式轉換成**波蘭字串記法**（Polish String Notation），這是爲了紀念發明此種記法的波蘭數學家 Lukasiewicz。此種記法的好處是運算符號以實際執行的次序出現。

波蘭字串記法的基本觀念是運算子寫在陳式的末端，而不是夾在陳式的中間。如A＋B寫成AB＋。運算子＋，在此祇爲將其前面二

個運算變數值相加的命令。

我們允許變數是一串字元。如果不對這種情形留意的話，於使用波蘭記法時，將會引起混淆。例如

AA／AAA

這個表示，如果轉換成波蘭記法，則成爲

AAAAA／

但是AAA／AA，A／AAAA，AAAA／A 等陳式，也會有相同的波蘭記法。因此除非有特別的防範方法，否則以波蘭字串表示的陳式，就不夠清楚，無法辨認出其對應於普通記法的唯一陳式。

爲避免這種模凌兩可的情形，每一變數應予定界，也就是說開頭和結尾必須指明。普通記法中，一部份是靠運算子作定界。但用波蘭記法時，將**定界符號**（ Delimiter ），如空格（□），加在每一變數的前面或後面。如陳式AA／AAA轉變成波蘭陳式 AA□AAA／，就不再有模凌兩可的情形發生。

幾乎所有現存的編譯或解譯程式，可以用相當於上述的技巧，預先編譯輸入的陳式。對每一個多字元的變數，常數或運算子，可以用通常爲固定長度的唯一記碼符號來取代。（有時會加入特殊的定界碼，以指示符號的結束。）

在下述較簡單的例子，只使用單字元變數的陳式。如此在波蘭記法時，免去在陳式中插入定界記號的麻煩。

此例說明習慣用的**中置式**（ Infix Notation ，即運算子在中間），如何用**後置式**（ Postfix Notation ，即運算子在最後）來代替：

中置式	後置式
(A+B)×(C−D)	(AB+) (CD−)×

在後置式中的運算子"×"，視爲將其前面二個陳式（AB+）和（

CD－）之值相乘的命令。

我們如何能導出將中置式變成後置式的一般規則呢？如果用**樹**（Tree）來代表中置式，那麼作此轉換時，祇須看這樹的結構，再把連於**節點**（Node）的**運算子枝**（Operator Branch），移到最右邊，如圖8.1所示。

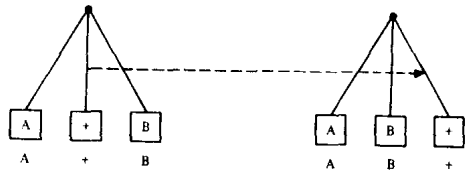


圖8.1

另外看一個例子，中置式A＋（B×C）及（A＋B）×C，如圖8.2所示。

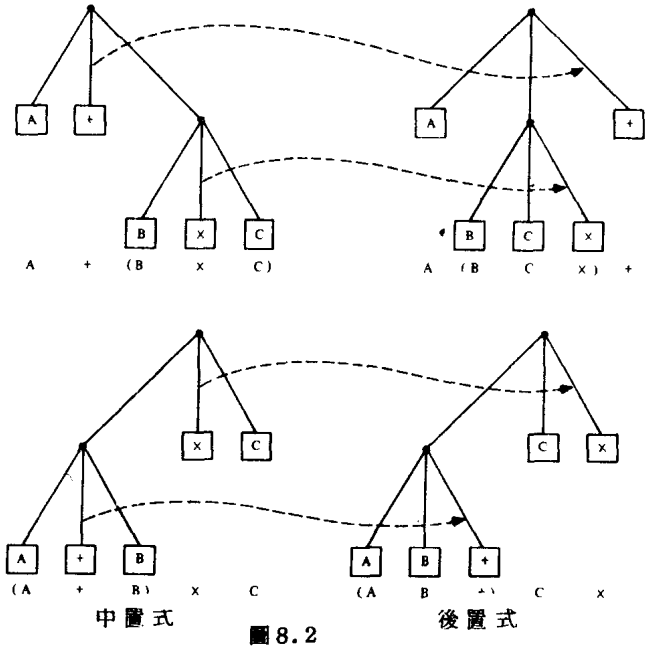


圖8.2

原有的這二個中置式，外觀上只是括號插入的位置不同。但換成後置

式

$$A\ (BC\times)\ +和\ (AB+)\ C\times$$

則不一樣。這二種表示，運算子的位置不相同。運算子的位置已夠指示要做的運算，而不須要括弧的存在。即可寫：

$$ABC\times+及AB+C\times$$

以代替上述的式子，且不致引起混淆。如此的表示就不需要優先次序的規則。

另外一種說法是，對任何波蘭字串陳式，祇有一種括弧插入法使所得的結果有意義。也就是說祇有一種括弧插入法，可以使全部的陳式及每個**副陳式**（ Sub expression ）有如下的形式：

陳式1　陳式2　運算子

可以看出上述的例子就是這種形式。

要了解波蘭字串的意義，一定要能指認出每個運算子要運算的副陳式。以本節開始的二次式爲例，用T代替2，F代替4，H代替5，則其中置式爲

$$(B+(B\uparrow T-F\times A\times C)\uparrow H)/(T\times A)$$

將此式變成波蘭字串，則成

$$BBT\uparrow FA\times C\times -H\uparrow +TA\times /$$

使用方括號，依下列規則可以指認出副陳式和有關的運算子。

1. 變數爲副陳式時，在每一變數後加一方括號。
2. 每次碰到運算子時，在運算子及其前面二個副陳式之後，放一個方括號，以形成新的副陳式。

開始如下：

$$BBT\uparrow FA\times C\times -H\uparrow +TA\times /$$

然後依上述規則，最後完成下式：

$$BBT\uparrow FA\times C\times -H\uparrow +TA\times /$$

完成波蘭字串記法的訣竅在於每個運算子恰爲一個副陳式的終結符號。

在中置及後置式中，變數的次序相同，但是運算子的次序則大大不同。如圖 8.3所示，爲中置式及後置式運算次序的比較。

後置式的運算次序在實際計算時非常重要。因爲這種表示法不需要前後往返的掃視。

執行運算的次序

中置式	後置式
(B↑(B↑T−F×A×C)↑H)　(T×A)	BBT↑FA×C×−H↑　+TA×
⑥ ① ④ ② ③ ⑤ ⑧ ⑦	① ② ③④ ⑤⑥ ⑦⑧

圖 8.3　二次式的計算程序

8.3　波蘭字串的計算

爲瞭解波蘭字串陳式及其效率，我們檢視簡單的單一掃視的計值方法。當由左而右，一字元一字元地掃視陳式時，建一個叫**堆疊**（Stack）的特殊表列。每次碰到一個運算子，就把最近堆到疊裏的二個值拿來運算。

建疊的規則如下：

1. 掃視到一變數時，將其值置於堆疊的頂端（即開口端）。
2. 掃視到一運算子時，

(a)將最近堆入的二個值移出堆疊。

(b)將這二值，用此運算子作運算。

(c)運算所得的結果，置於堆疊的頂端。

再回頭看看以波蘭字串表示的二次式：

$$BBT\uparrow FA\times C\times -H\uparrow +TA\times /$$

而其值如下：

A	B	C	F	H	T
3	7	−20	4	0.5	2

（記住，此值爲方程式 $3x^2+7x-20=0$，一個根的負值。）

在圖8.4中，陳式寫在左邊的垂直方向，以顯示堆疊的建立。右邊表示右邊字元掃視後，堆疊的狀況。圖8.4是顯示演算過程中，堆疊在所有階段的狀況。實際應用時，則不須如此，最後結果示於圖8.5。到運算的最後（限正確的後置式），除了最後一個運算子計算的值外，其餘的都劃掉，這就是陳式的值。

採用「堆疊」一詞的理由很明顯。這裏堆疊的工作狀況很像自助餐館裏的盤疊。供應午餐時，顧客拿走盤子，服務員就把盤子再放在盤疊上。由於都是由疊的同一端取走或加入，因此最後放入的，最先被移出，稱爲**後入先出**（Last In, First Out）。

後置機器（POSTOS）

後置機器（Postfix Machine）的結構是用後置陳式做爲機器的語言。最熟悉的實際後置機器即爲掌上型計算器及圖1.36所述的計算機。了解這種型式的計算機，即**POSTOS**（Postfix-Oriented SAMOS），如何工作，將很有助益。

考慮中置式 （A＋B）×（C－D）及其等效的後置式

AB＋CD－×

BBT↑FA×C×−H↑+TA×／的各值已知，作單一掃視計值

掃視字元	計算	堆疊的狀況（右邊爲開端）
B		7
B		7 7
T		7 7 2
↑	7↑2 = 49	7 7 2 49
F		7 7 2 49 1
A		7 7 2 49 4 3
×	4×3 = 12	7 7 2 49 4 3 12
C		7 7 2 49 4 3 12 −20
×	12×(−20) = −240	7 7 2 49 4 3 12 −20 −240
−	49 − (−240) = 289	7 7 2 49 4 3 12 −20 −240 289
H		7 7 2 49 4 3 12 −20 −240 289 .5
↑	289↑.5 = 17	7 7 2 49 4 3 12 −20 −240 289 .5 17
+	7 + 17 = 24	7 7 2 49 4 3 12 −20 −240 289 .5 17 24
T		7 7 2 49 4 3 12 −20 −240 289 .5 17 24 2
A		7 7 2 49 4 3 12 −20 −240 289 .5 17 24 2 3
×	2×3 = 6	7 7 2 49 4 3 12 −20 −240 289 .5 17 24 2 3 6
.	24／6 = 4	7 7 2 49 4 3 12 −20 −240 289 .5 17 24 2 3 6 4

圖8.4　二次式的詳細掃視圖

B	B	T	↑	F	A	×	C	×	−	H	↑	+	T	A	×	.
7	7	2	49	4	3	12	−20	−240	289	.5	17	24	2	3	6	4

圖8.5　簡易的堆疊運算狀況

在後置機器語言中，產生所需後置式值的指令列，如下方所示：

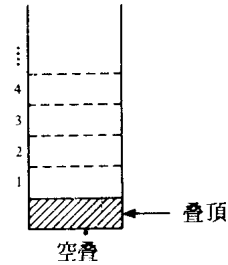
疊　A		疊1001
疊　B		疊1002
ADD	⟹	ADD
疊　C		疊1003
疊　D		疊1004
SUB		SUB
MPY		MPY

假定A，B，C，D的儲存位置爲1001，1002，1003，1004，則指令列如箭頭右邊所示。

此機器語言用堆疊運算說明，甚易了解。此機器的堆疊可視爲一

列儲存記錄器，以取代一般機器（如SAMOS）的累積記錄器。計算機綫路在任何時刻都知道最後填入值的記錄器在串列中的位置，也就是知道堆疊最頂端的位置。

假設在執行上述指令列之前，堆疊是空的。此空疊示於右圖，**疊頂指示器**（Top-of-stack indicator）指到未使用的一特殊記錄器。

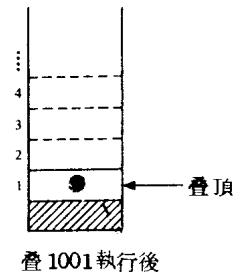


爲說明起見，假設A，B，C和D的位置及現值如下表：

變數	位置	值
A	1001	2
B	1002	4
C	1003	3
D	1004	7

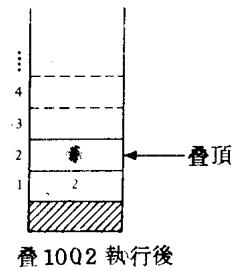
疊1001

這個指令意指取出1001位置的值，「**堆**」（Push）到疊頂端，圖示於右。



疊1002

這個指令是取出4這個值，指定到疊的下一位置，如右圖



下一指令

ADD

不需要運算元（位址），其功用是(1)將疊最頂端二個記錄器內的值相加，再把相加的