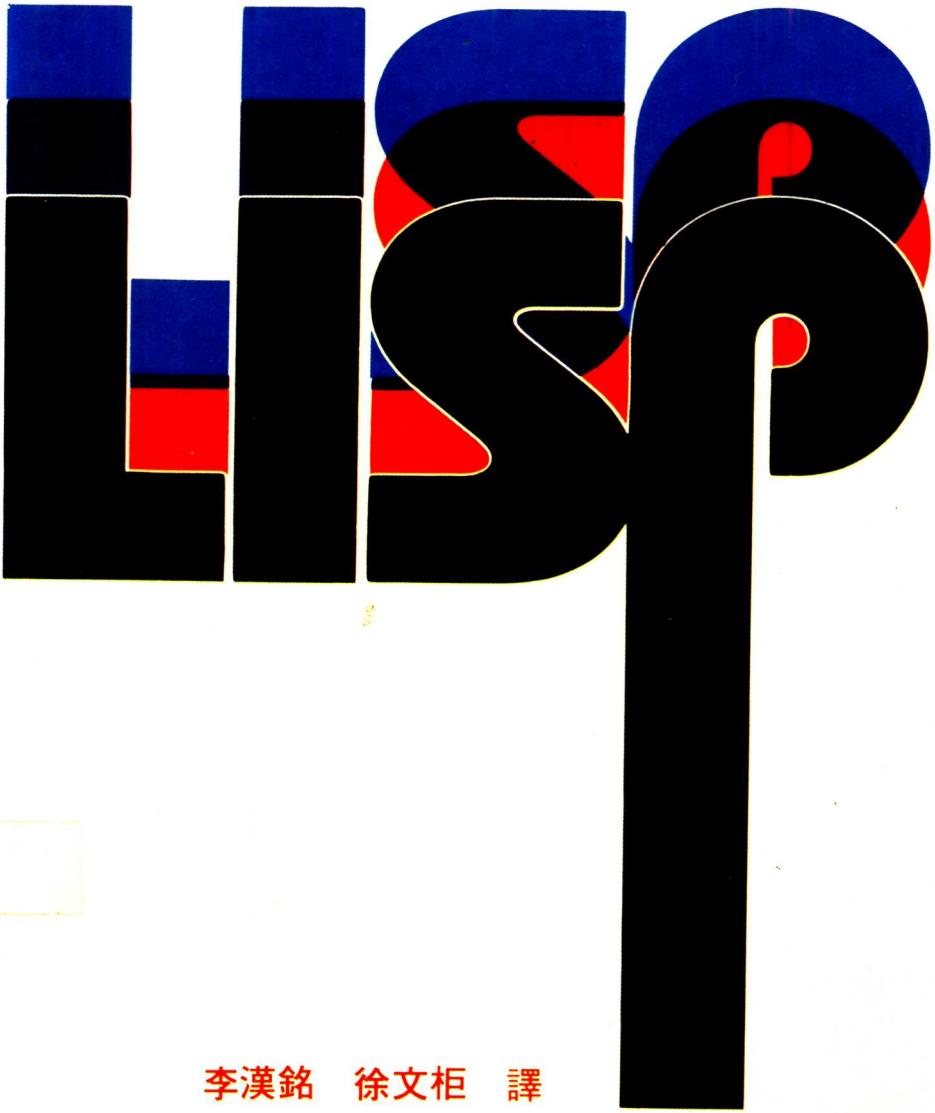
LISP基礎



李漢銘　徐文柜　譯

LISP基礎

李漢銘　徐文柜　譯

儒林圖書公司　印行

LISP基礎

譯　者：李漢銘・徐文柜
發行人：楊　鏡　秋
出版者：儒　林　圖　書　有　限　公　司
地　址：台北市重慶南路一段111號
電　話：3118971-3　3144000
郵政劃撥：0106792-1號
吉豐印刷廠有限公司承印
板橋市三民路二段正隆巷46弄7號

行政院新聞局局版台業字第1492號
中華民國七十五年十月初版
定價新台幣　140　元正

前言

本書是Lisp程式的入門書，是爲不論有無計算基礎而想學習Lisp的學生寫的。一般而言，討論的層次適合高中、大學學生，或是電腦從業人員。若具備基本微積分的知識，將有助於了解後面所附的範例，但在本書的其他部份則無此必要。

本書最初是Jeff Shrager（目前在Carnegie-Mellon大學）和Steve Bagley（目前在MIT）在Pennsylvania大學求學時寫成的，Pennsylvania大學的工程及應用科學的Moore學院提供計算機設備供他們使用，以便在電腦上發展Lisp。增加的部份則是由Gnosis公司職員Stewart Schiffman和本書範例所使用的P-LISP作者Steve Cherry完成。

爲什麼要學LISP？

Lisp是重要的，它是人們目前還在使用最古老語言中的一種。計算機科學家們設計且讓人們將其應用在“人工智慧”上。人工智慧AI，如其名稱的涵意，乃是目前計算機科學研究的一個領域。爲了完成這件工作，Lisp是不可或缺的。

Lisp是簡單的，很多電腦語言要求使用者處理其所要電腦執行的煩人細節部份；但在Lisp中，你不需爲計算機結構擔心，而Lisp運算式（expression）的語法或格式有其規則且是一致

的。

Lisp是有趣的,它經常處理的問題包含了遊戲和迷宮，且完全以交談式作業，交談式使得使用者更能控制機器，並使電腦成爲其“思考的夥伴”。在很多年前，所有計算機系統，都是在打孔卡片上執行工作，而P-Lisp在計算機上的執行，比大部份早期機器的功能更強且更容易使用。

LISP發展簡介

Lisp在1950年代末期由John McCarthy於MIT發展，以其充當一種代數串列處理語言（LISP-LISt 處理程式 ，以便在人工智慧的新領域中運作。第一個產品Lisp 1在1958年完成，隨後幾年則推出第二版Lisp 1.5 ，Lisp 1.5是目前既存Lisp系統的先驅。在1960年代，幾個其他版本被發展在不同機器上使用。MIT的MacLisp 、InterLisp（其前身爲BBN Lisp）和Michigan大學的MTSLisp是目前所使用的三種版本。

本書所要討論的P-Lisp，是另一種Lisp系統，可在Apple和各種微電腦系統上執行。個人Lisp 電腦是計算機界的一個精巧發明，MIT以大型的微電腦執行由MacLisp衍生的語言。雖然P-Lisp不具有像微電腦那麼強的功能（主要是因爲用來執行的電腦比較小），但擁有Lisp 處理程式的理想仍然不變，尤其本書使用的是P-Lisp版本3.1，如果你沒有這版本，一些小細節將不同（例如，可能沒有浮點算術）。

如果你使用另一種Lisp,則不致於有太多問題，因爲我們

對書中的大部份範例，都使用與Lisp函數共通的子集。

本書的風格

我們相信學習本書是有趣的，本書一直是依循此種旨趣寫成。因此，有時我們會使用“逗趣的”範例型態，以避免讀者因枯躁的內容而不耐煩。

我們使用基本而具體的範例，減少使用者對Lisp 中一些抽象觀念在理解上的困難，我們建議你仔細看完書中的範例，然後利用你的電腦直接執行Lisp，這是很是必要的，因爲沒有其他方法比立即回饋更能刺激學習。

這本書章節都很簡短，一口氣讀完好幾章並瞭解其內容是可能的，但本書並不要求你一口氣讀完，所以不必太急。書中有某幾章最後附有一些問題，如果想就去做做看，而某些問題只是提供你思考而不要求有解，所以如果你只是推想而沒有實際去做它們，也已經足夠了。

印刷上的習慣

讀者會發現本書有時（特別是在較長的範例）會以小寫字母表示使用者鍵入的一列文字，而以大寫字母表示Lisp 系統本身顯示的文字。如果依循這個習慣，則有一個小小的例外，就是字母“ L ”總是以大寫形式出現，這是因爲它的小寫形式跟1一樣，而這將會使程式範例產生混淆。很不幸的是，L時常在Lisp程式中代表“ List ”。

本書的備註是以方括弧〔“〔〕”〕而非正常括弧表示，因爲Lisp中使用了很多這種括弧。

儲存本書範例的磁片

我們使用一軟式磁碟片儲存本書用到的所有函數，如此可節省鍵入的時間並提供一個預先建立的工作環境來撰寫程式。如果你已知道Lisp的工作環境，當然就沒有問題；如果沒有，也不要擔心，本書會在後面介紹。

概　觀

本書格式如下：以一章簡單的範例開始，以使讀者輕鬆地使用Lisp系統，然後用幾章介紹資料結構和函數，之後告訴你如何定義和編修你自己的函數；其次，再介紹遞歸(recursion)概念、Lisp程式原則，利用幾章探究遞歸的不同用法，並且舉一些複雜的Lisp程式範例。書中最後一節並以數個章節討論高等Lisp技巧，以及Lisp系統本身的內容。

請於閱讀本書後，讓我們知道你的問題所在或其他你想要了解的事情。

LISP基礎

本書是針對人們經常提到的人工智慧語言LISP(LIST Processor）作一個新且具有指導性的介紹。

不管你的計算機背景如何，這本易懂、清晰的入門書將會教你使用雖是最古老但仍然爲人使用的語言，其學習的過程簡單而且有趣。讀完本書後，你將會熟悉：

- LISP的資料結構和函數
- 如何來定義和編修函數
- 樹狀結構和遞歸
- 較精深的LISP程式
- 更多的技巧

你同時可以找到一些具體且基本的例子，幫助你領悟某些抽象的觀點，而每章後面的習題則可加強學習的效果。本書還有一個特點，就是包括了一個由Joseph Weizenbaum所寫的LISP程式ELIZA所完成的對話範例。

不要猜測LISP是什麽以及它可以爲你作些什麽。讓這本LISP基礎（Learning LISP）來教你學習語言應該知道的知識，而這語言將會使你的計算機參與你的思考。

儒林圖書公司 印行

目 錄

第一章

如何開始使用LISP

本章將提供一些使用P-Lisp 系統的經驗，目的是幫助你熟悉該語言的基本運作。

我們假設你正坐在電腦前面，並且在執行Lisp（參見P-Lisp手册中“如何做這件事”的說明）。你將在螢幕頂端看到如下的畫面：

```
        GNOSIS INC.
P-LISP    VER. 3.1.2
--------------------

COPYRIGHT 1982'BY STEVEN CHERRY
      ALL RIGHTS RESERVED
```

在看到這畫面時，表示你已成功地進入Lisp，提示號“：”告訴你Lisp正等著你鍵入某些東西，此時可鍵入所要執行的資料Lisp會在你按下RETURN鍵後，執行鍵入的指令並顯示結果這種“讀取－求值－輸出”的程序，組成了交談式Lisp系統的核心，我們待會兒會介紹更多的讀取－求值－輸出程序。

注意，如果你按數次RETURN鍵，則Lisp每次都會回應一個“：”。如果你不要它做任何事，它就什麼也不做，然後要求輸入另一列。

如果你鍵入一個數字，Lisp會回應這數字。所有輸入都跟在“：”提示號後，而所有Lisp的回應，都留有兩個空白。

```
:3
  3
:0
  0
:-2
  -2
```

讓我們看一個將某些數字相加的範例。在Lisp 中相加數字時是使用ADD函數，要將1和2相加，只需鍵入

```
:(ADD 1 2)
  3
```

的確，Lisp能相加。實際上發生了什麼事呢？我們在Lisp顯示“：”提示號後，鍵入“(add 1 2)”,應注意下列幾件事:

- 以空白分開字語ADD和數字“1”及“2”。
- 以括弧將運算式括起來。括弧是Lisp 語言的整合要素，所以你應該學習去使用括弧。
- Lisp會立即回應解答。它是一個交談式系統，並且立

即顯示解答，除非你叫它不要顯示，我們稍後說明令其不要顯示解答的方法。

讓我們再看看一些加法的範例。

```
:(ADD 12 3
:)
   15
:(ADD 1)
     ** ERROR: TOO FEW ARGS **
       ADD :: (1 )
+()
  NIL
:(ADD 11 8 3)
     ** ERROR: TOO MANY ARGS **
       ADD :: (11 8 3 )
+()
  NIL
```

此處我們首先使用先前的範例，但忘了鍵入右括弧，而Lisp正等著右括弧，所以它回應一個“：”提示號。在我們輸入右括弧後，Lisp就得到了滿足條件,所以開始執行這加法運算。一般而言，讀者可以隨意將輸入放在好幾列，待會你會發現這種處理方式非常有用。

我們下一列顯示一有趣的範例，要求Lisp 對一個數字作

加法運算，因爲單獨一個數字的加法沒有特別意義和用處，所以Lisp傳回一個錯誤訊息，告訴你相加的數字個數太少，這是相當合理的，因爲進行加法運算時，至少需要兩個數字。

注意，Lisp顯示的是一個“＋”，而非正常的“：”提示號，此時不必擔心，只要鍵入“()”即可，我們等一下會介紹這種運算。

最後一列顯示對三個數字作加法的情形也是錯誤的！對兩個以上數字作加法運算，似乎比單獨一個數字相加合理，我們待會將介紹更多能實際定義加法完成這項工作（或其他我們要做的事情）的方法。

一對括弧中的第一個元素是函數名稱，之後便是引數（arguments）〔因此上面錯誤訊息說“引數太少”〕。這是非常重要的，而且本書都是使用這兩個字語。第一個範例的函數名稱是ADD，它的引數是“1”和“2”，所以在此範例中，ADD有兩個引數。

除了加法外，Lisp也能執行乘法運算，乘法函數的名稱是MULT，讓我們試試它吧！

```
:(MULT 2 3)

  6

:(MULT 9 2)

  18

:(MULT 1 2 3 4)

    ** ERROR: TOO MANY ARGS **
      MULT :: (1 2 3 4 )
```

```
+()
  NIL

:(MULT 1.2 4)
  4.8
:(MULT 2 (ADD 1 2))
  6
```

前兩個例子證明Lisp 能夠做乘法運算。

然而，Lisp 僅能對兩個數值進行乘法運算。如果用兩個以上數值相乘，你將得到一個TOO MANY ARGS（太多引數）的錯誤訊息。同樣地，如果MULT僅有一個引數或沒有引數，則會得到TOO FEW ARGS（引數太少）的錯誤訊息，此時必須再鍵入“()”，以便回到正常提示號。

下一個範例顯示Lisp能處理非整數數值。浮點運算雖是好的運算，但並不重要，我們很快會發現Lisp 的長處不是在算術上。

上面的最後一個範例最有趣。Lisp 要執行MULT 函數，但發現第二個引數是子運算式（subexpression），該子運算式“(add 1 2)”的值爲3。現在有一數值可以取代這子運算式，所以可以繼續乘法運算，Lisp 現在實際上執行的是“(mult 2 3)”。

因爲這種運算在Lisp 中是非常普遍的，所以我們再練習一些範例，試看你是否能算出每個運算式的結果。

```
:(ADD (MULT 3 4) (MULT 2 6))
  24
:(MULT (MULT(ADD 1 0)(ADD 1 1)) (MULT (ADD 2 1) (ADD 1 3)))
  24
:(MULT 1 (MULT 2 (MULT 3 (MULT 4 1))))))))))
  24
```

在上面範例中要注意的一件重要事情是，最後一個例子有很多右括弧，這是很好的一點，事實上當你不想進行運算括弧數目時是非常方便的。你要做的是鍵入很多右括弧，然後就可順利地回到提示號“：”下。

現在討論一個概念：述語(predicates)。述語是傳回答案爲眞或假的一種特別函數。在Lisp中,眞是以“T”表示，而假是以“NIL”表示。

```
:(GREATER 3 4)
  NIL
:(GREATER 4 3)
  T
:(GREATER 100 -100)
  T
:(NUMBER 47)
  T
:(NUMBER 'LETTERS)
```

```
  NIL
:(NUMBER 'SEVEN)
  NIL
:(ZERO 0)
  T
:(ZERO (ADD 2 -1))
  NIL
:(ZERO (ADD 2 -2))
  T
```

如果數值是以絕對遞減順序排列那麼述語GREATER就傳回爲眞“T”；否則就傳回假“NIL”。如果引數是數值，則述語NUMBER爲“T”，否則爲“NIL”。很明顯地，“seven”是字母〔關於引號在前面的意義待會再作說明〕而非數字。如果引數的值爲零，ZERO就傳回“T”。

動手做習題

現在開始練習Lisp並鍵入運算式。你可能需花些時間完成本章所有範例，順便做些算術以確定Lisp也能執行一般的算術運算，試看能否找出一種將代數運算式轉換成等效Lisp 數學運算式的方法。