

Lisp 入門

田村直之 (神戸大学工学部情報知能工学科)

(tamura@kobe-u.ac.jp)

1 LISP の概要

- AI 用言語 (AI プログラムの記述・開発に適している言語) の一種
- 多くの AI システムは LISP で記述されている (特に米国で)
- 1957 年頃に米国 MIT の John McCarthy らにより開発された
- LISP Processor (リスト処理言語) の省略
- 多数の方言が存在するが、標準版として 1984 年に Guy L. Steele Jr. が中心となって Common Lisp を設計した

2 LISP の特徴

- **記号処理言語, リスト処理言語**
データとして、記号 (シンボル) を取り扱うことができる。また、リストと呼ばれる可変長のデータの列を取り扱うことができる。
- **関数型言語**
LISP では新たな関数を定義することによってプログラムを作り上げていく。すなわち、LISP のプログラムは関数定義の集まりである。LISP や Prolog は、FORTRAN や BASIC などの手続き型言語とは異なり、非手続き型言語と呼ばれる。
- **対話的使用, 会話形使用**
LISP システムの使用形態は対話的である。すなわち、ユーザは LISP システムを立ち上げたあと、システムと会話するような形で命令を与え、関数を定義したり実行したりできる。

3 記号

LISP, Prolog などの言語では、**記号** (シンボル, symbol) をデータとして取り扱うことができる。記号を取り扱える言語は**記号処理言語**と呼ばれることがある。

Common Lisp では通常、記号は英字 (a, b, ..., z, A, B, ..., Z および +, -, * など) も英字と見なされる) または数字 (0, 1, ..., 9) を 1 文字以上続けた名前で見なされる。ただし、数値として見なされるもの (123, +1 など) は省く。

```
X coffee B52 - N-Tamura
```

Common Lisp では、英字の大文字と小文字は区別しない。したがって、coffee, COFFEE, cOfFeE はすべて同じ記号である。いくつかの LISP システムでは漢字も使用できる。

記号処理言語はデータとして数値だけしか取り扱えない言語に比較して、より自然に問題を表現できる。

4 リスト

LISP, Prolog などの AI 用言語では、記号や数値の列をデータとして取り扱うことができる。この列は **リスト (list)** と呼ばれる。リストは、その **要素** を空白で区切って並べたものをカッコでくくって表す。たとえば

(JAN 31 1957 THU)

(PI 3.14)

であり、一般的には

(要素₁ 要素₂ ... 要素_n)

と書く。リストの要素の個数 n は、そのリストの **長さ** と呼ばれる。上の例では、リストの長さはそれぞれ 4 と 2 である。要素 _{i} をリストの **第 i 要素** と呼ぶ。たとえば、最初の例での第 3 要素は 1957 である。

要素のないリスト (長さが 0 のリスト) は **空リスト** と呼ばれる。空リストは記号 NIL で表すこともできる。

()

NIL

リストの要素が、またリストであってもよい。

(COFFEE MILK (ORANGE JUICE))

((A (B) (C (D))) (E () F))

上の例でのリストの長さはそれぞれ 3 と 2 である。

Common Lisp では、記号、数値、リスト以外に、文字、文字列、配列、ベクタ、ハッシュ表などをデータとして使用できる。また、数値としても任意倍長の整数、分数 (有理数)、浮動小数点数、複素数を使用できる。

練習問題

問 1 リスト (A () B) の長さはいくつか。

問 2 リスト ((A B) C) の第 1 番目の要素は何か。

5 LISP を使ってみる

NEC PC-9801 上で利用できる muLISP-87 システムを例にとって説明する。ただし、説明する機能は Common Lisp で標準とされているものだけに限る。

LISP システムを立ち上げると、プロンプト (入力促進記号) として \$ が表示される (カーソルは下線で示す)。

```
$ _
```

ユーザが LISP システムに実行させたい命令 (計算させたい式) を入力すると、それが実行した結果が表示される。

```
$ (+ 1 2)
3
$ -
```

このように、ユーザがシステムと対話しながらシステムを利用することから、このような使用形態は**対話的使用**と呼ばれる。

ユーザが入力する式は、一般的には

(関数名 式₁ 式₂ ... 式_n)

という**関数**の形である。式_iは**引数**と呼ばれる。引数が、また式であってもよい。たとえば、関数 $\log \sin 1$ の値は次のようにして計算できる。

```
$ (LOG (SIN 1))
-0.1726038
```

また、ほとんどの LISP システムでは任意倍長の整数演算ができる。

```
$ (* 123456789 987654321)
121932631112635269
```

リストを引数とする関数も使用できる。リストを引数とする場合には、リストの前に**クォート** (引用符, クォーテーションマーク) をつける。たとえば、

```
$ (LENGTH '(COFFEE MILK (ORANGE JUICE)))
3
$ (LENGTH '((A (B) (C (D)))) (E () F])
2
```

ここで、LENGTH はリストの長さを求める関数である。また、右角カッコ] はたくさんのカッコを一度に閉じるために用いられる。

+, *, LOG, LENGTH などのように、LISP システムであらかじめ用意されている関数は**組込関数** (built-in function) と呼ばれる。

練習問題

問 3 与えられた 2 つのリストの長さの合計を求めるにはどうすればよいか。

6 変数

関数 SETF を使うと、記号に値を代入できる。

```

$ (SETF X (* 12 34))
408
$ (+ X 56)
464
$ (SETF MENU '(TEA COFFEE MILK))
(TEA COFFEE MILK)
$ (LENGTH MENU)
3
$ (SETF X (LENGTH MENU))
3
$ (+ X 1)
4

```

一般的には

(SETF 変数 値)

と書く。変数としては任意の記号、値としては任意の式が書ける。SETF の代わりに、SETQ という関数を使用してもよい。

7 リスト処理関数

数多くあるリストを処理する組込関数のうち、いくつかを紹介する。
APPEND は 2 つのリストを連結したリストを求める関数である。

```

$ (APPEND '(A B) '(C D))
(A B C D)
$ (APPEND '(A (B C)) '(D () E))
(A (B C) D () E)
$ (APPEND '(A B C) '())
(A B C)
$ (APPEND () '(A B C))
(A B C)

```

空リスト () はクォートを省いてもよい。

REVERSE はリストの要素を逆順に並べ変えたリストを求める関数である。

```

$ (REVERSE '(A B C D))
(D C B A)
$ (REVERSE '(A (B C) D))
(D (B C) A)

```

SORT はリストを一定の順序 (昇順, 降順, アルファベット順など) にしたがって並べ変えたリストを求める関数である。

```
$ (SORT '(3 1 4 2) '<')
(1 2 3 4)
$ (SORT '(3 1 4 2) '>')
(4 3 2 1)
$ (SORT '(SHIRAI SUGIHARA ISHIZUKA) 'STRING<')
(ISHIZUKA SHIRAI SUGIHARA)
```

これらの関数は、以下で述べるようなより基本的な関数の組合せで作られている。

8 基本リスト処理関数

先頭要素を求める関数 CAR

関数 CAR はリストの先頭の要素 (第 1 要素) を求める。

```
$ (CAR '(A B C))
A
$ (CAR '((A B) (C D)))
(A B)
$ (SETF MENU '(TEA COFFEE MILK))
(TEA COFFEE MILK)
$ (CAR MENU)
TEA
```

残りのリストを求める関数 CDR

関数 CDR はリストの先頭の要素を除いた残りのリスト (第 2 要素以降のリスト) を求める。

```
$ (CDR MENU)
(COFFEE MILK)
$ (CDR (CDR MENU))
(MILK)
$ (CDR (CDR (CDR MENU)))
NIL
$ (CAR (CDR MENU))
COFFEE
```

(CAR (CDR MENU)) で MENU の第 2 要素を求めることができる。

先頭に要素を加えたリストを求める関数 CONS

関数 CONS は、第 1 引数を第 2 引数のリストの前につけたリストを求める。

```
$ (CONS 'COCOA MENU)
(COCOA TEA COFFEE MILK)
$ (CONS 'COCOA (CDR MENU))
(COCOA COFFEE MILK)
```

リストに対する基本的な操作は、以上の CAR, CDR, CONS の 3 つの関数で行える。

練習問題

- 問 4 MENU の 3 番目の要素を求めるにはどうすればよいか。
問 5 MENU の第 1 要素を JUICE に変更したリストを作るにはどうすればよいか。
問 6 MENU の第 1 要素と第 2 要素の間に JUICE を挿入したリストを作るにはどうすればよいか。
問 7 MENU の第 2 要素を取り除いたリストを作るにはどうすればよいか。
問 8 MENU の第 1 要素と第 2 要素を交換したリストを作るにはどうすればよいか。

9 関数定義

LISP では既存の関数を用いて、新たな関数を自分で定義することができる。たとえば、引数として与えられたリストの 2 番目の要素を求める関数 NIBANME は、関数 DEFUN を使って次のようにして定義する。

```
$ (DEFUN NIBANME (X) (CAR (CDR X)))
NIBANME
```

一般的には

(DEFUN 関数名 (仮引数₁ 仮引数₂ ... 仮引数_n) 関数本体)

と書く。このように定義した関数は、LISP システムにすでにある関数と同様に使用できる。

以下では、さらにリストの 2 番目の要素を取り除いたリストを求める関数 DEL2 を定義し、それと関数 NIBANME を用いて、リストの 1 番目の要素と 2 番目の要素を交換したリストを求める関数 EX12 を定義する。

```
$ (NIBANME '(A B C))
B
$ (DEFUN DEL2 (X) (CONS (CAR X) (CDR (CDR X)]
DEL2
$ (DEFUN EX12 (X) (CONS (NIBANME X) (DEL2 X]
EX12
$ (DEL2 '(A B C))
(A C)
$ (EX12 '(A B C))
(B A C)
```

実は、LISP では以下のような組込関数があらかじめ用意されている。

```

(DEFUN CAAR (X) (CAR (CAR X)))
(DEFUN CADR (X) (CAR (CDR X)))
(DEFUN CDAR (X) (CDR (CAR X)))
(DEFUN CDDR (X) (CDR (CDR X)))
(DEFUN FIRST (X) (CAR X))
(DEFUN SECOND (X) (CAR (CDR X)))
(DEFUN THIRD (X) (CAR (CDR (CDR X))))
(DEFUN REST (X) (CDR X))

```

練習問題

- 問 9 リストの 2 番目の要素と 3 番目の要素を交換したリストを求める関数 EX23 を定義せよ。
- 問 10 リストを左へ 1 つ回転させたリストを求める関数 ROTATE を定義せよ。たとえば (ROTATE '(A B C D))=(B C D A) である。与えられるリストの長さは 1 以上としてよい。(ヒント : (A B C D) から (B C D A) を得るには, (APPEND '(B C D) '(A)) と考える)

10 述語と条件

LISP のある種の関数は条件判断を行うためのもので、特に **述語** と呼ばれる。述語は判断の結果が真ならば T を、偽ならば NIL を返す。

NULL は引数が空リストかどうか調べる述語である。

```

$ (NULL ())
T
$ (NULL 'A)
NIL
$ (NULL NIL)
T
$ (NULL '(A))
NIL

```

= は 2 つの引数が同一の数値であるかどうか調べる述語である。

```

$ (= 1 2)
NIL
$ (= 2 (+ 1 1))
T

```

EQUAL は 2 つの引数が同一の記号やリストであるかどうか調べる述語である。

```

$ (EQUAL 'A 'B)
NIL
$ (EQUAL 'A (CAR '(A B)))
T
$ (EQUAL '(B) (CDR '(A B)))
T

```

(NULL x) と (EQUAL x ()) は同じ意味である。

<, >, <=, >= は 2 つの数値の大小関係を調べる。STRING<, STRING>, STRING<=, STRING>= は 2 つの記号の大小関係 (アルファベット順) を調べる。

```

$ (< 1 2)
T
$ (STRING> 'A 'B)
NIL

```

述語の判断結果によって、すなわち条件によって異なる値を返す関数を作るには、関数 IF を用いる。

(IF 条件式 式₁ 式₂)

は、条件式の結果が T の時は式₁ の計算結果を、NIL の時は式₂ の計算結果を値とする (正確には、条件式の結果が NIL 以外なら式₁ を選ぶ)。式₂ を省略した場合は、(IF 条件式 式₁ NIL) と同じである。

```

$ (IF (NULL 'A) 1 2)
2
$ (SETF X '(A B))
(A B)
$ (IF (>= (LENGTH X) 2) (CAR (CDR X)) X)
B

```

IF を使うと、2 つの数値の大きい方を返す関数 OOKIIHOU が定義できる。

```

$ (DEFUN OOKIIHOU (X Y) (IF (> X Y) X Y))
OOKIIHOU
$ (OOKIIHOU 3 7)
7

```

組込関数 MAX を利用すると同じことが行える。

練習問題

問 11 2 つの数値を要素とする長さ 2 のリストを昇順に並べ変えたリストを求める関数 SORT2 を定義せよ。たとえば (SORT '(3 2))=(2 3), (SORT '(2 3))=(2 3) である。

11 再帰的定義

数学では、定義の中に自分自身が現れることがある。たとえば、 n の階乗は次のように定義される。

$$n! = \begin{cases} 1 & (n = 0) \\ n \times (n - 1)! & (n \geq 1) \end{cases}$$

このように定義の中に自分自身が現れる定義を、LISP では**再帰的定義**と呼ぶ。たとえば、階乗を求める関数 FACT は次のように定義できる。

```
$ (DEFUN FACT (N) (IF (= N 0) 1 (* N (FACT (- N 1))
FACT
$ (FACT 10)
3628800
$ (FACT 100)
9332621544394415268169923885626670049071
5968264381621468592963895217599993229915
6089414639761565182862536979208272237582
51185210916864000000000000000000000000
```

リストの要素の合計を求める関数 SUM は次のように定義できる。

```
$ (DEFUN SUM (L)
  (IF (NULL L) 0 (+ (CAR L) (SUM (CDR L))
SUM
$ (SUM '(1 9 8 9))
27
```

2つのリストを連結したリストを求める関数 APP は次のように定義できる。

```
$ (DEFUN APP (X Y)
  (IF (NULL X) Y (CONS (CAR X) (APP (CDR X) Y))
APP
$ (APP '(A B) '(C D))
(A B C D)
```

実は、この関数は APPEND と同じことを行う。

練習問題

- 問 12 関数 FACT を変更して、1 から n の和 $1 + 2 + \dots + n$ を求める関数 SIGMA を定義せよ。
問 13 関数 FACT を変更して、1 から n の平方和 $1^2 + 2^2 + \dots + n^2$ を求める関数 SIGMA2 を定義せよ。
問 14 次の漸化式で定義されるフィボナッチ数を計算する関数 FIB を定義せよ。

$$fib(n) = \begin{cases} n & (n < 2) \\ fib(n - 1) + fib(n - 2) & (n \geq 2) \end{cases}$$

- 問 15 リストの全要素の積を求める関数 PROD を定義せよ。
- 問 16 リストの中で一番大きい要素を求める関数 MAXELEM を定義せよ。(ヒント: リストの長さが 1 の時は CAR が最大要素である。長さが 2 以上の時は CDR の最大要素と CAR との大きい方が最大要素である)
- 問 17 リストを逆順にしたリストを求める関数 REV を定義せよ。(ヒント: 空リストの逆順は空リストである。空リストでないときは CDR の逆順と、CAR だけからなる長さ 1 のリストとを APPEND したものが全体の逆順である)
- 問 18 第 2 引数のリストの要素として第 1 引数と同じ記号が現れるかどうか調べる述語 MEM を定義せよ。たとえば (MEM 'A '(B A C))=T, (MEM 'A '(B (A) C))=NIL である。(ヒント: 第 2 引数が空リストなら結果は NIL である。第 2 引数が空リストでないときは、第 2 引数の CAR が第 1 引数と EQUAL なら結果は T, EQUAL でないなら第 2 引数の CDR について調べる)

12 応用プログラム例

Lisp のプログラム例として、次の宣教師と土人のパズルを A*アルゴリズムで解くプログラムを紹介する。

3 人の宣教師と 3 人の人喰い土人が川の一方向の岸にいます。1 隻の 2 人乗りボートを使って、向こう岸に渡るにはどうすれば良いでしょうか。ただし土人の人数が宣教師の人数よりも多くなると、宣教師は喰われてしまうので、そうならないようにしてください。

まず、A*アルゴリズムによる探索プログラムの骨格部分を示す。理解しやすいように、変数名、関数名にはできるだけ日本語を使用する。

```

;
; A*アルゴリズムのプログラム
;
(DEFUN A* (開始節点)
  (LET (節点1 枝リスト 枝)
    (SETF OPENリスト (LIST 開始節点))
    (SETF CLOSEDリスト ())
    (SETF 経路・コスト表 (LIST (LIST 開始節点 () 0)))
    (LOOP
      ((NULL OPENリスト) NIL)
      (SETF 節点1 (CAR OPENリスト))
      (SETF OPENリスト (CDR OPENリスト))
      (SETF CLOSEDリスト (CONS 節点1 CLOSEDリスト))
      ((目標節点? 節点1) (経路 節点1))
      (SETF 枝リスト (枝リスト 節点1))
      (LOOP
        ((NULL 枝リスト) NIL)
        (SETF 枝 (CAR 枝リスト))
        (SETF 枝リスト (CDR 枝リスト))
        (リスト更新 節点1 枝)
      )
      (OPENリスト並べ替え)
    )
  )
)

```

ここで、LIST は引数を要素として持つリストを求める関数である。たとえば、(LIST (LIST 開始節点 () 0)) は変数 開始節点の値が (1 3 3) ならば、(((1 3 3) () 0)) というリストを返す。

LET は、

(LET (局所変数₁ 局所変数₂ …) 式₁ 式₂ … 式_n)

において、式₁、式₂、…、式_nを、その順序で実行し、式_nの値を返す関数である。ただし、局所変数₁、局所変数₂、…は、LETの中だけの局所変数として取り扱われる。

LOOPは、

(LOOP 式₁ 式₂ … 式_n)

において、式₁、式₂、…、式_nを、繰り返し実行する。ただし、式_iのうちに

(条件式 結果式)

という形の式があり、条件式の結果がNILでなければ、繰り返しを終了し、LOOPの結果として結果式の値を返す。たとえば、(NULL OPENリスト) NIL)は変数OPENリストの値が空リストならば、NILをLOOPの値として返す。

次に、“リスト更新”関数の定義を示す。

```
;
; 節点1から枝を経由した場合について
; OPENリストとCLOSEDリストを更新する
;
(DEFUN リスト更新 (節点1 枝)
  (LET (子節点 枝コスト 新経路 新コスト)
    (SETF 子節点 (CAR 枝))
    (SETF 枝コスト (CADR 枝))
    (SETF 新経路 (APPEND (経路 節点1) (LIST 節点1)))
    (SETF 新コスト (+ (コスト 節点1) 枝コスト))
    (IF (OR (NULL (コスト 子節点)) (< 新コスト (コスト 子節点)))
      (LET ()
        (SETF 経路・コスト表
          (表更新 (LIST 子節点 新経路 新コスト) 経路・コスト表))
        (IF (要素? 子節点 CLOSEDリスト)
          (SETF CLOSEDリスト (削除 子節点 CLOSEDリスト)))
        (IF (NOT (要素? 子節点 OPENリスト))
          (SETF OPENリスト (CONS 子節点 OPENリスト)))
      ))
  ))
```

ORは

(OR 式₁ 式₂ … 式_n)

において、式₁、式₂、…、式_nを、その順序で実行していくが、途中、式_iの値がNILでなければそれを返し、すべてNILならNILを返す関数である。

同様の関数にANDがある。

(AND 式₁ 式₂ … 式_n)

において、式₁、式₂、…、式_nを、その順序で実行していくが、途中、式_iの値がNILならNILを返し、すべてNILでないなら式_nの値を返す関数である。

NOTは、引数の値がNILならTを、NILでないならNILを返す関数である。実は、NULLとNOTは常に同じ結果を返す。

残りの関数定義を示す。

```

;
; OPENリストの要素を評価値の昇順に並べ替える
;
(DEFUN OPENリスト並べ替え ()
  (SETF OPENリスト (SORT OPENリスト '評価値<)))
(DEFUN 評価値< (節点1 節点2) (< (評価値 節点1) (評価値 節点2)))
;
; 評価値 = これまでたどってきた経路のコスト + 目標までの推定コスト
;
(DEFUN 評価値 (節点) (+ (コスト 節点) (推定コスト 節点)))
(DEFUN 経路 (節点) (CADR (表検索 節点 経路・コスト表)))
(DEFUN コスト (節点) (CADDR (表検索 節点 経路・コスト表)))
(DEFUN 表検索 (節点 表)
  (IF (NULL 表)
    NIL
    (IF (EQUAL 節点 (CAAR 表))
      (CAR 表)
      (表検索 節点 (CDR 表))))))
(DEFUN 表更新 (要素 表)
  (IF (NULL 表)
    (LIST 要素)
    (IF (EQUAL (CAR 要素) (CAAR 表))
      (CONS 要素 (CDR 表))
      (CONS (CAR 表) (表更新 要素 (CDR 表)))))))
(DEFUN 要素? (要素 リスト)
  (IF (NULL リスト)
    NIL
    (IF (EQUAL 要素 (CAR リスト))
      T
      (要素? 要素 (CDR リスト))))))
(DEFUN 削除 (要素 リスト)
  (IF (NULL リスト)
    NIL
    (IF (EQUAL 要素 (CAR リスト))
      (CDR リスト)
      (CONS (CAR リスト) (削除 要素 (CDR リスト))))))

```

次に、A*のプログラムを利用して宣教師と土人のパズルを解くプログラムを示す。

```

;
; 宣教師と土人のパズルをA*アルゴリズムで解くためのプログラム
;
(DEFUN PUZZLE () (A* '(1 3 3)))
(DEFUN 目標節点? (節点) (EQUAL 節点 '(0 0 0)))
(DEFUN 推定コスト (節点) 0)
(DEFUN 枝リスト (節点)
  (枝リスト2 節点
    '( ( 1 1 0) ( 1 0 1) ( 1 1 1) ( 1 2 0) ( 1 0 2)
        (-1 -1 0) (-1 0 -1) (-1 -1 -1) (-1 -2 0) (-1 0 -2) )
  ) )
(DEFUN 枝リスト2 (節点 移動方法リスト)
  (IF (NULL 移動方法リスト)
    ()
    (IF (移動可能? 節点 (CAR 移動方法リスト))
      (CONS (LIST (移動 節点 (CAR 移動方法リスト)) 1)
            (枝リスト2 節点 (CDR 移動方法リスト)))
      (枝リスト2 節点 (CDR 移動方法リスト)))
  ) ) )

```

```

(DEFUN 移動 (節点 移動方法)
  (LIST (- (NTH 0 節点) (NTH 0 移動方法))
        (- (NTH 1 節点) (NTH 1 移動方法))
        (- (NTH 2 節点) (NTH 2 移動方法))) )
(DEFUN 移動可能? (節点 移動方法)
  (LET (左岸ボート数 左岸宣教師数 左岸土人数
        右岸ボート数 右岸宣教師数 右岸土人数)
    (SETF 左岸ボート数 (- (NTH 0 節点) (NTH 0 移動方法)))
    (SETF 左岸宣教師数 (- (NTH 1 節点) (NTH 1 移動方法)))
    (SETF 左岸土人数 (- (NTH 2 節点) (NTH 2 移動方法)))
    (SETF 右岸ボート数 (- 1 左岸ボート数))
    (SETF 右岸宣教師数 (- 3 左岸宣教師数))
    (SETF 右岸土人数 (- 3 左岸土人数))
    (AND (OK? 左岸ボート数 左岸宣教師数 左岸土人数)
         (OK? 右岸ボート数 右岸宣教師数 右岸土人数))
  ) )
(DEFUN OK? (ボート数 宣教師数 土人数)
  (AND
    (>= ボート数 0) (>= 宣教師数 0) (>= 土人数 0)
    (OR (= 宣教師数 0) (>= 宣教師数 土人数))
  ) )

```

(NTH n リスト) はリストの n 番目の要素を求める関数である。(NTH 0 x) は (CAR x) と同じ, (NTH 1 x) は (CADR x) と同じである.

実行結果は次のようになる.

```

$ (PUZZLE)
((1 3 3) (0 3 1) (1 3 2) (0 3 0) (1 3 1) (0 1 1)
 (1 2 2) (0 0 2) (1 0 3) (0 0 1) (1 1 1))

```

なお変数 OPEN リストの値は次のように変化する.

((1 3 3))
((0 3 2) (0 2 2) (0 3 1))
((0 3 1) (0 2 2))
((0 2 2) (1 3 2))
((1 3 2))
((0 3 0))
((1 3 1))
((0 1 1))
((1 2 2))
((0 0 2))
((1 0 3))
((0 0 1))
((1 1 1) (1 0 2))
((1 0 2) (0 0 0))
((0 0 0))

13 参考文献

1. 竹内郁雄 著,「初めての人のための LISP」, ソフトウェア ライブラリ 3, サイエンス社
(楽しく LISP を学べる本)
2. ウィンストン ほか著, 白井良明 ほか訳,「LISP」, 情報処理シリーズ 4, 培風館
(ウィンストンの有名な教科書の翻訳)
3. 黒川利明 著,「LISP 入門」, 電子計算機のプログラミング 7, 培風館
(LISP 入門としてよくまとまった教科書)
4. シャピロ 著, 松田利夫 訳,「LISP による人工知能の基礎技法」, 共立出版
(人工知能の基礎技法のための LISP プログラムをやさしく解説してある)
5. 安西祐一郎 ほか著,「LISP で学ぶ認知心理学 1-3」, 東京大学出版会
(学習, 問題解決, 言語理解についての 3 部作)
6. 後藤滋樹 著,「記号処理プログラミング」, 岩波講座 ソフトウェア科学 8, 岩波書店
(LISP と Prolog についての教科書)

Lisp 練習問題解答

問 1 3

問 2 (A B)

問 3 たとえば, (A B C) と (D E F) の長さの合計を求めるには (+ (LENGTH '(A B C)) (LENGTH '(D E F))) とすればよい.

問 4 たとえば, (CAR (CDR (CDR MENU))) とする. 後で述べるように (CADDR MENU), (THIRD MENU), (NTH 2 MENU) などでもよい.

問 5 たとえば,

```
(CONS 'JUICE (CDR MENU))
```

問 6 たとえば,

```
(CONS (CAR MENU) (CONS 'JUICE (CDR MENU)))
```

問 7 たとえば,

```
(CONS (CAR MENU) (CDR (CDR MENU)))
```

問 8 たとえば,

```
(CONS (CAR (CDR MENU)) (CONS (CAR MENU) (CDR (CDR MENU))))
```

問 9 たとえば,

```
(DEFUN EX23 (X) (CONS (CAR X) (EX12 (CDR X))))
```

あるいは

```
(DEFUN EX23 (X)
```

```
  (CONS (CAR X) (CONS (CADDR X) (CONS (CADR X) (CDDDR X)))))
```

などでもよい.

問 10 たとえば,

```
(DEFUN ROTATE (X) (APPEND (CDR X) (CONS (CAR X) ())))
```

問 11 たとえば,

```
(DEFUN SORT2 (X)
```

```
  (IF (< (CAR X) (CADR X)) X (CONS (CADR X) (CONS (CAR X) ())))))
```

問 12 たとえば,

```
(DEFUN SIGMA (N) (IF (= N 0) 0 (+ N (SIGMA (- N 1)))))
```

問 13 たとえば,

```
(DEFUN SIGMA2 (N) (IF (= N 0) 0 (+ (* N N) (SIGMA2 (- N 1)))))
```

問 14 たとえば,

```
(DEFUN FIB (N) (IF (< N 2) N (+ (FIB (- N 1)) (FIB (- N 2)))))
```

問 15 たとえば,

```
(DEFUN PROD (L) (IF (NULL L) 1 (* (CAR L) (PROD (CDR L)))))
```

問 16 たとえば,

```
(DEFUN MAXELEM (L)
  (IF (NULL (CDR L)) (CAR L) (OOKIIHOU (CAR L) (MAXELEM (CDR L)))))
```

問 17 たとえば,

```
(DEFUN REV (L)
  (IF (NULL L) () (APPEND (REV (CDR L)) (CONS (CAR L) ())))))
```

問 18 たとえば,

```
(DEFUN MEM (X L)
  (IF (NULL L) NIL (IF (EQUAL X (CAR L)) T (MEM X (CDR L)))))
```