計算機プログラミング'93　＃1

久野　靖*

1993.9.1

今年度の「計算機プログラミング」は予告通り Lisp 言語を取り上げます。Lisp を取り上げる 1 つの理由は「C や Pascal のような手続き型言語とはまた違った言語があることを体験して頂く」ことにありますが、併せて講義全体の目標として次のことを掲げることにします。

- 基本的なアルゴリズム (定石) を体験し理解すること。
- 様々なプログラミングスタイル (パラダイム?) を体験すること。
- 言語処理系の内幕について理解すること。

実はこれはかなり欲張りな目標なので、すべてを十分達成できるかどうかは分かりません。まあ進み具合を見て調整していくことにします。テキスト (参考書) としては、予告通り「Essential Lisp」(アンダーソン他、玉井訳、これが LISP だ、サイエンス社、1989) を使用しますが、順番などはこれにこだわらず毎回資料を用意して進めていきます。評価については、試験は行なわず、代わりにこまめに小レポート課題を用意しますのでできる範囲で提出して下さい。これも様子によっては、最後に試験がわりのまとまった課題を追加するかも知れません。ではよろしく。

1　Lisp 言語の紹介と KCL

Lisp は 1950 年代に、MIT のマッカーシーによって考案された。Fortran と並んで最も古い高水準言語の 1 つ。その特徴は:

- リストと呼ばれる柔軟なデータ構造を扱える。
- 記号処理を得意とする。
- インタプリタベースの処理系であり、対話的利用に向く。

Lisp にはいくつもの「方言」があるが、ここでは CommonLisp と呼ばれる言語仕様を持つ処理系である KCL(Kyoto Common Lisp) を用いる。KCL を起動するには

```
% kcl
KCl (Kyoto Common Lisp)  May 15, 1989
>
```

この「>」というプロンプトに向かって、式 (Lisp のプログラムの断片) を打ち込むとそれに対応する動作 (計算) がただちに行なわれ、結果が表示される。終りにするときは^D を打つ。

```
>^D
Bye.
%
```

*筑波大学大学院経営システム科学専攻

2　Lisp と S 式

Lisp では処理系に向かって打ち込むものはすべて S 式と呼ばれる。とりあえず、S 式とは次のようなものだと定義しておく。

- S 式とはアトムまたはリストである。
- 数値 (整数や小数点つき数など) はアトムである。
- 記号 (名前や+、-などの文字) はアトムである。
- リストとは、「(S 式 S 式 … S 式)」の形のものである。

これをよく見ると、「S 式」の定義に「S 式」が使われているので循環的 (堂々めぐり) みたいに思えるが、実はそうではない。(なぜか) このような定義を再帰的定義と呼ぶ。再帰的定義を使うと無限要素の集合をうまく定めることができる。(テキスト 1.4 節)

3　S 式と形式と評価

S 式のうちで、Lisp 処理系によって実行されるプログラム (の断片) として適した形になっているものを形式 (form) という。形式を実行すること (というのは、式の値を計算すること) を、Lisp の世界では「評価する」という。(「値を求める」くらいの意味だと思えばよい。) 評価の仕方は次の通り。

- 数値アトムでは、その数値そのものが結果の値となる。
- 記号アトムでは、それを変数として扱い、その中に格納されている値が結果の値となる。
- リストの場合は、そのリストは「(操作名 引数 1 引数 2 …引数 n)」の形でなければならない。結果の値は、各引数を評価した後、それらにたいして操作を適用した結果の値である。

なお、操作の大部分は関数である。まず算術関数から順に見てみる (テキスト 1.1〜1.3 節)。

- (+ 引数 1 … 引数 n): 引数の和を返す。
- (- 引数 1 引数 2): 引数 1 と引数 2 の差を返す。
- (* 引数 1 … 引数 n): 引数の積を返す。
- (/ 引数 1 引数 2): 引数 1 を引数 2 で割った商を返す。
- (floor 引数 1): -∞へ向かって切捨て。
- (truncate 引数 1): 0 へ向かって切捨て。
- (ceiling 引数 1): +∞へ向かって切上げ。
- (round 引数 1): 丸め。
- (rem 引数 1 引数 2): 剰余。
- (mod 引数 1 引数 2): 剰余。

例えば次のような具合ですね。

```
>(+ 1.0 3.14)
4.14
>(+ (* 2.0 2.0) 1.0)
5.0
```

操作名が必ず先頭に来ること (前置記法) と、常に全体をかっこで囲むことに違和感があるでしょう? Lisp とはそういうものだと思ってください。なお、引数は評価されるので、引数として関数呼び出しを書けることにも注意。

練習 1 以下の値を Lisp により計算させてみよ。

- 半径 5cm の円の面積
- 半径 5cm の球の体積
- 半径 5cm、高さ 10cm の円柱の体積
- 半径 5cm、高さ 10cm の円錐の体積

ついでに、`mod` と `rem` はどう違うかも探求してみよ。

4 setq と quote

変数に値を入れるには

```
(setq 変数名 値)
```

による。これを使うと値を変数に覚えさせておくことができる。

ところで、記号アトムとか一般の S 式を値として覚えさせたりデータとして使ったりしたい時はどうするか? それには、「以下はデータだよ」という書き方が必要。Lisp ではこれは

```
(quote 値)
```

によって表せる。これはひどくよく使うので、もっと簡便に「' 値」と書いてもかまわないように仕掛けがしてある (読み込む時に Lisp が quote に変換してくれる)。

5 関数定義

ここまでは、関数として Lisp 内部に予め用意されているものだけを使用してきた。しかし、自分で新しく関数を定義することももちろんできるし、実は Lisp でプログラムを作るというのは自分がやりたい計算のための関数群を用意することに他ならない。関数を作るには操作 `defun` を使用する。

```
(defun 関数名 (引数名 …) 式 … 式)
```

ここで「引数名」は関数が呼び出される時に渡される引数にそれぞれ対応させられる。そして、式は関数の値を計算するためのものである。式は複数個書けるが、その最後のものが値として返される (当分は式は 1 つだけで済むであろう)。

例えば、数を 1 つ引数として受け取り、その 2 乗を計算する関数 `square` を定義してみる。

```
>(defun square (num) (* num num))
SQUARE
>(square 2)
4
>(square 1.5)
2.25
```

関数が呼び出されて実行される過程をよく考えてみて欲しい。まず引数として与えた値「2」が num に対応させられ、それから「(* num num)」が評価されるので、結果として 4 が求まるわけである。

練習 2 以下の関数を定義せよ。

- 半径 r の円の面積を求める関数 circle
- 半径 r の球の体積を求める関数 sphere
- 半径 r、高さ h の円柱の体積を求める関数 cylinder
- 半径 r、高さ h の円錐の体積を求める関数 corn

6 条件判断と枝分かれ

ここまでにやった関数の場合、計算はすべて「1 本道」だったが、プログラムを組む上では当然、条件判断と枝分かれが必要である。それは cond 形式によって表現できる。

```
(cond (条件 1 式 … 式)
      (条件 2 式 … 式)
      …
      (条件 n 式 … 式))
```

これは次のような動作を表す。まず条件 1 を調べ、それが成り立っていればそれに続く式を順に評価し、最後の値が cond 形式の値となる。成り立っていなければ今度は条件 2 を調べ、それが成り立っていればればそれに続く式を順に評価し、最後の値が cond 形式の値となる。成り立っていなければ…のように続く。

では条件とは何だろう? これもやっぱり、関数であり、ただし結果として「はい」か「いいえ」のいずれかの値を返す。このような関数のことを「述語」という。なお、Lisp の世界では「いいえ」は特別な記号 nil で表し、nil 以外のものはすべて「はい」であるとみなす。(特に「はい」であることを表したい場合には特別な記号 t を使う。)

とりあえず数値的な条件についてまとめておく。

- (= 引数 1 引数 2): 引数 1 が引数 2 と数値的に等しいとき「はい」。
- (> 引数 1 引数 2): 引数 1 が引数 2 より大きいとき「はい」。
- (< 引数 1 引数 2): 引数 1 が引数 2 より小さいとき「はい」。
- (>= 引数 1 引数 2): 引数 1 が引数 2 以上のとき「はい」。
- (<= 引数 1 引数 2): 引数 1 が引数 2 以下のとき「はい」。
- (and 条件 1 … 条件 n): 条件 1〜条件 n がすべて「はい」のとき「はい」。
- (or 条件 1 … 条件 n): 条件のどれかが「はい」のとき「はい」。
- (not 条件): 条件の反転。
- (zerop 数値): 数が 0 なら「はい」。
- (plusp 数値): 数が正なら「はい」。
- (minusp 数値): 数が負なら「はい」。

これを使って、絶対値を計算する関数を作ってみる。

```
(defun absolute (num)
  (cond ((minusp num) (- num))
        (t num)))
```

このように、cond では最後まで全部条件が「いいえ」だと困る (困らないけれど全体の値が nil になってしまう) ので、最後の条件は「t」にするのが普通である。

ところで、ついにプログラム (関数) が 3 行にもなったので、これからは直接 Lisp に向かってプログラムを打ち込むのではなく、まず nemacs を使って defun 形式を「なんとか.lsp」というファイルに打ち込み、あとで load という関数を使って読み込むことにする。こうすれば修正も簡単だし、nemacs の機能でかっこの対応などもよくわかる。例えば上の定義が test.lsp というファイルに入れてあったとして、次のようにして実行できる。

```
>(load "test")
Loading test.lsp
Finished loading test.lsp
T
>(absolute -2)
2
>(absolute 2)
2
```

練習 3 以下の関数を定義せよ。

- 1 つの引数 x を取り、その正/負/零に対応して 1/-1/0 を返す関数 sign
- 1 つの引数 x を取り、その逆数 (ただし x が 0 なら 0) を返す関数 inv
- 2 つの引数 x、y を取り、その大きい方を返す関数 max2
- 3 つの引数 x、y、z を取り、その最大のものを返す関数 max3
- 引数 x を取り、それを 10 進表示した時の 1 の位を返す関数 dig1
- 引数 x を取り、それを 10 進表示した時の十の位を返す関数 dig10
- 引数 a, b, x を取り、$a < x < b$ であるかどうか判定する述語 between

7 再帰的関数

さて、分岐までできたけれど、まだこれだけでは大した計算はできない。そこで次に、Pascal や C ならループをやるのだけれど、Lisp ではその代わりに再帰をやる。再帰というのは要するに自分に渡された問題を「ちょっとだけ」簡単にして、残っている部分をもう 1 度自分自身 (のコピー) にたらいまわしにするというやり方である。例えば階乗の計算を考えてみる。5! = 5 * 4 * 3 * 2 * 1 = 5 * 4!ですよね。だから:

```
(defun factorial (x)
  (cond ((zerop x) 1)
        (t (* x (factorial (- x 1))))))
```

つまり、ごく簡単な場合 (x = 0) は自分で解いてしまうが、そうでない場合はまず (x-1)!を求めて、それに x を掛け合わせることで x!を求めるわけである。なお、「簡単な場合は自分で解く」というのがないと無限にたらいまわししようとして止まらなくなるので注意すること。

練習 4 以下の関数を定義せよ。

- 非負整数 n を受け取り、$1 + 2 + \ldots + n$ を返す関数 `nsum`
- 数 x、n(n は非負整数) を取り、x^n を返す関数 `power1`
- 数 x、n(n は任意整数) を取り、x^n を返す関数 `power`
- 数 x、n(n は自然数) を取り、$1 + x + x^2 + \ldots + x^n$ を返す関数 `crsum`

A　本日の小課題 — 2 分法による求根

関数 $f(x)$ が区間 $[a, b]$ で連続かつ単調増大 (実は単調減少でも原理は同じ) であり、なおかつこの区間に $f(x) = 0$ の根があるものとする。その時、任意の小ささの誤差 e を指定して、その誤差範囲内でこの根の値を求める方法がある。それには次のような関数 (`getroot` a b e) を書けばよい。

- $(b - a) < e$ ならば、$\frac{a+b}{2}$ が答え。
- そうでなくて、$f(\frac{a+b}{2}) < 0$ なら、(`getroot` $\frac{a+b}{2}$ b e) が答え。
- そうでなければ (`getroot` a $\frac{a+b}{2}$ e) が答え。

なぜそうか? よく考えてみよう。納得したら、この方法で $f(x) = x^2 - 2$ として、つまり 2 の平方根を求めるプログラムを作って動かせ。納得したら、別の関数の根もいろいろ試してみよ。また任意の数の平方根を求める関数 `sqroot` を作ってみよ (誤差は適当に決めてよい)。

結果をレポートとして提出する場合には、必ず A4 版の紙を使用し、次のような順で構成し、全体を綴じて提出すること。

- 表紙。課題名 (1993 年度計算機プログラミング課題＃ 1)、学籍番号、氏名、提出日付) のみを記すこと。
- 方針。どのような考え方で課題を解こうと思ったか記す。
- 回答。作成したプログラムとその解説、実行例など。
- 考察。課題をやった結果どんなことがわかったか、何が問題か、など。(感想も入れてほしいが、感想ばかりでは内容として不足。)

〆切は一応次回の授業の直前までとしますが、遅れても受理はします。最初に述べたように小課題の提出は義務ではありません。