計算機プログラミング'93　＃3

久野　靖*

1993.9.22

小レポートはいかがでしたか。さて、重要なところなので、cdr 再帰のところの復習からやろうと思います。

1　cdr 再帰の復習

cdr 再帰とは、リストの各要素を先頭から順番に処理していくことで結果を求めるが、その時次のような考え方で処理するような手順である。

- まずリストが空のときの値を定める。
- リストが空でない時は、先頭要素について処理をする。
- 先頭要素の処理が終ったら、リストの残りの部分 (cdr) については、自分自身を再帰的に呼び出すことで処理させる。

例として、前回の練習 7 の回答例を見てみる。まず「リスト l を受け取り、その中の数値のみの合計を返す関数 sumnum」。

```
(defun sumnum (l)
   (cond ((null l) 0)                                    ; (1)
         ((numberp (car l)) (+ (car l) (sumnum (cdr l)))) ; (2)
         (t (sumnum (cdr l)))))                          ; (3)
```

すなわち、(1) l が空なら合計は 0。(2) l の先頭要素 (`(car l)`) が数ならば、『残りのリストの中の数値の合計』にこの先頭要素を加えたものが答え。(3) l の先頭要素が数でなければ、この先頭要素は答えに影響しないので無視して良い。従って『残りのリストの中の数値の合計』が答え。ここで『』で囲んだ部分が自分自身を再帰的に呼び出すことによって計算される。countx もパターンとしては同様ですね。

「リスト l とアトム x を受け取り、l 中に x と同じものがあるかどうか調べる述語 xmember」ですが。

```
(defun xmember (l x)
   (cond ((null l) nil)                ; (1)
         ((eq (car l) x) t)            ; (2)
         (t (xmember (cdr l) x))))     ; (3)
```

*筑波大学大学院経営システム科学専攻

すなわち、(1) l が空なら答えは「いいえ」。(2) l の先頭が x と等しい (記号だから eq) で比べること) ならば「はい」。(3) そうでなければ、先頭は関係ないので『残りのリストの中に x と同じものがあるかどうか調べ』てその結果を返せば良い。このように先頭を処理した結果答が求まるならそのような場合には再帰は不要になる。saigo もこれと似たパターンになる。

ではもう少し練習しておきましょう。

練習 8 以下の関数を定義せよ。

- 数のリスト l を受け取り、その全要素の積を返す関数 seki。
- リスト l を受け取り、その長さが偶数かどうか調べる述語 evenlist。
- リスト l と記号 x を受け取り、l の中に x が現れればそこから後のリストを返し、そうでなければ nil を返す関数 ymember。
- リスト l を受け取り、その中の (トップレベルにある) アトムの数を数える関数 countatoms。

2 cdr 再帰によるリスト組み立て

これまでは 1 個の数値や特定の値を返すだけだったが、そうではなく cdr 再帰のパターンで新しいリストを組み立てることも用意である。要は、car と cdr で分解することの反対は cons でくっつけることだと理解していればよい。例えば次の例を見てみる。

```
(defun add1list (l)
   (cond ((null l) nil)
         (t (cons (+ (car l) 1) (add1list (cdr l))))))
```

練習 9 以下の関数を定義せよ。

- リスト l を受け取り、その各要素を () の中に入れたリストを返す関数 enpar。
- リスト l を受け取り、その各要素が 2 回ずつ現れるリストを返す関数 doublelist。
- リスト l を受け取り、その中の数値を取り除いたリストを返す関数 removenum。
- 数が昇順に並んだリスト l と数値 x を受け取り、x を l の適切な位置に挿入したリストを返す関数 insertnum。
- 与えられたリスト l を逆順にする関数 xreverse。

終ってしまって暇な人はさらにテキストの練習問題をどうぞ。

練習 10 テキストの練習問題 7.1〜7.18。ただし 7.1、7.2、7.6 はやってしまった (はず)。7.5 はテキストの説明とちょっと違うやり方になる。7.15、7.16 は今のところできないのでとばす。

3 別のプログラミングスタイル

ここまででは、Lisp のプログラミングスタイルとして、すべての必要な計算を 1 つの関数として書き、関数の値は (枝分かれした後は)1 つの式で表すという関数型のスタイルを取ってきた。しかし場合によっては、(C や Pascal などと同様) 変数とか入出力を駆使したプログラミングスタイルの方が使いやすいこともある。そのための道具立てについてここで勉強しておく。

3.1 局所変数

変数に値を入れることは setq でできる、というのは既にやった。ただしそうやっていきなり setq を使って代入するとその変数は広域変数になってしまう。広域変数にしたくない時、つまり局所変数を使いたい時は

(let（局所変数名 …）式 …)

という形式を使う。これによって、本体の「式 …」を実行している中で指定した局所変数が使える。局所変数の初期値は nil である。なお、let 式の値は本体の式の並びのうちで最後に実行された式の値である。let と類似したものに prog がある。

(prog（局所変数名 …）式 …)

prog が let と違うのは、その内部で (return 式) というのを実行するとそこで実行が終って、その式が prog 全体の値になるところである。

以上がテキストに書いてある事柄だったが、CommonLisp では加えて次のようなこともできる。まず局所変数は let を使わなくても、

(defun 関数名（引数 … &aux 局所変数名 …）式 …)

のように、defun の引数部で指定できる。また、関数の実行から抜け出るには prog を使わなくても

(return-from 関数名 式)

によることができる。

3.2 入力と出力その他

変数を使ったプログラミングでは入出力の関数もいろいろと欲しくなるはずである。

- (read): 引数を持たない関数 read は、キーボードから S 式を 1 つ読み込み、その S 式を値として返す。
- (print X): 関数 print は、その引数 X を画面に表示する。
- (princ X): 関数 princ の機能は print と類似しているが、ただし文字列定数を打ち出す時に""をつけたりしない。また余分な改行も行なわない。
- (terpri): 関数 terpri は単に出力を改行する。

練習 11 テキストの練習問題 5.1〜5.13。

3.3 ループ

ループは次回にもっと詳しくやりますが、とりあえず 1 つだけ。

(loop 式 …)

は式の並びを繰り返し実行する。繰り返しから抜け出すには return か return-from を使う。例えば次の通り。

```
(defun yes-or-no (mesg &aux ans)
  (loop (terpri) (princ mesg) (setq ans (read))
        (cond ((eq ans 'yes) (return-from yes-or-no t))
              ((eq ans 'no) (return-from yes-or-no nil)))
        (terpri) (princ "Please say yes or no.")))
```

なお、この関数を再帰に直すこともできる。

```
(defun yes-or-no (mesg &aux ans)
  (terpri) (princ mesg) (setq ans (read))
  (cond ((eq ans 'yes) (return-from yes-or-no t))
        ((eq ans 'no) (return-from yes-or-no nil)))
  (terpri) (princ "Please say yes or no.") (yes-or-no mesg))
```

4　小課題その 2 — 3 目並べ

今度はテキストにも載っていることですし、3 目並べを課題にしてみましょう。一気に作るのは大変ですから、順番に考えて作っていき、できたところまでで小レポートにして下さっていいです。なおテキストには回答例が載っていますが、そのまんま打ち込んで動かそうとするのはあまり得策でありません。なぜなら、1 箇所でも打ち込み間違うと自分ではどこが違うかわからなくなるからです。だから参考にするのはいいですが、プログラムとしてはあくまでも自分で順を追って独自に作るのがいいわけです。まず、盤は 3 × 3 のます目ですが、これを

```
'((nil nil nil) (nil nil nil) (nil nil nil))
```

のようなリストで表すことにします。○や×を打った時は、`nil` を記号 `x` や `o` で置き換えるわけです。その上で、次の関数を作ってみてください。

- 上のリストから、n 行目の m 列目を取り出す関数 `tic-ref` を作る。`nth` とかいう関数を利用すれば簡単ですね? ただし `nth` は 0 から数えるのに注意。
- 上のようなリストを、それらしく画面に表示する関数 `tic-out` を作る。(テキスト問題 5-12。)
- 上のリストで○の勝ち/×の勝ち/どちらも勝っていない、に対応して `o`、`x`、`nil` のいずれかを返す関数 `winner` を作る。
- 上のリストの、n 行目の m 列目を指定された記号 (たぶん `o` か `x` のはず) で置き換える関数 `enter` を作る。
- 上のリストについて、適当な「自分の手」を選んで打つ関数 `choose-move` を作る。例えばテキスト問題 5-13 の解説のように、決まった順番に空いているところを探して最初に見つかったところに打つというのでもよい。
- 利用者と計算機で交互に手を打って行き勝負がついたら終るという関数 `play-role` を作る。
- これらを適当に組み合わせて、3 目並べを行なう関数 `play-tic-tac-toe` を作る。

これら小部分の 1 つでもものによっては結構骨があると思います。まあ頑張ってください。虫取りの相談はいつでもどうぞ。最後までできてゲームが動いたら結構感動しますよ。その後はもちろん「強く」できれば言うことありません。

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＃ 2)、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。)

〆切は一応次々回の授業 (10/6) 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。