ソフトウェア工学実験'93　課題2 (Lisp 言語) ＃1の捕捉

久野　靖 *

1993.5.26,1993.6.7

はじめに

Lisp の体験はいかがでしたか。いくつか気がついたことを。

- emacs を使っていない人がいますが… 課題になったら絶対使わないでは済まないですから、早く慣れてください。再度説明すると:
  1. 「emacs &」で emacs を別の窓で動く独立プロセスとして動かす。
  2. ^X^F でファイル名を聞いてきたら test2.lsp[RET] などとして最後が.lsp で終るファイル名をつける。
  3. defun は 1 つのファイルにいくつ書いてもよい。
  4. 適当なところまで書いたら^X^F で保存する。
  5. Lisp の窓へ行って「(load "test2")」などとするとロードできる。
- XtermLog.xxxx の制御文字を取り除くコマンドを作りました。勝手に持っていって使ってください。
- xterm の logging がいやなら emacs の ESC x run-lisp RET を使って Lisp を起動してもよいです。その場合は Lisp のバッファですから普通に^X^W でファイルを指定して保存できます。
- アンケートを書かなかった人、プリントアウトの裏を使わなかった人、5 枚も 6 枚もプリントした人、「指定に合致していません」よ。

練習問題の回答例

練習 1

最初のはあまりに簡単ですが。

```
(* 5 5 3.14)
(* 3/4 3.14 5 5 5)
(* 5 5 3.14 15)
(* 5 5 3.14 15 1/3)
```

*筑波大学大学院経営システム科学専攻

練習 1

これも簡単ですね。既に作った関数をどんどん利用しましょう。

```
(defun circle (r) (* r r 3.14))
(defun sphere (r) (* 3/4 3.14 r r r))
(defun cylinder (r h) (* (circle r) h))
(defun corn (r h) (* (cylinder r h) 1/3))
```

練習 3

max3 は maximum を利用するとスマートですね。

```
(defun sign (x)
   (if (< x 0) -1 (if (> x 0) 1 0)))
(defun maximum (x y)
   (if (> x y) x y))
(defun max3 (x y z)
   (maximum x (maximum y z)))
```

練習 4

再帰関数には慣れましたか?

```
(defun power1 (x n)
   (if (< n 1) 1 (* x (power1 x (- n 1)))))
(defun power (x n)
   (if (< n 0) (/ 1 (power1 x (- n))) (power1 x n)))
(defun crsum (x n)
   (if (< n 1) 1 (+ 1(* x (crsum x (- n 1))))))
```

練習 5+7

既に lambda をやったので、どんどん使いました。

```
(defun nprod (x f)
   (if (< x 1) 1 (* (funcall f x) (nprod (- x 1) f))))
(defun sigma (x f)
   (if (< x 1) 0 (+ (funcall f x) (sigma (- x 1) f))))
(defun newsqsum (x)
   (sigma x #'(lambda (y) (* y y))))
(defun newpower1 (x n)
   (nprod n #'(lambda (y) x)))
(defun newcrsum (x n)
   (+ 1 (sigma n #'(lambda (y) (newpower1 x y)))))
(defun sigmamn (m x f)
   (if (< x m) 0 (+ (funcall f x) (sigmamn m (- x 1) f))))
```

なお、power1 を作るには sigma ではなく nprod がいりますね。なおかつ lambda を使わないとできませんでしたね、これは。すいません。

練習 6

これはお楽しみ問題だったのですが、できましたか。

```
(defun findroot (a b f)
   (if (< (- b a) 0.0000001)
       a
       (if (> (funcall f (* 0.5 (+ a b))) 0)
           (findroot a (* 0.5 (+ a b)) f)
           (findroot (* 0.5 (+ a b)) b f))))
```

2 の平方根とかも求めてみました。

```
>(findroot 0 2 #'(lambda (x) (- (* x x) 2)))
1.414213538169861
```