計算機プログラミング'93　＃6

久野　靖*

1993.10.13

今回は前回の最後に駆け足でやったラムダ式とマップ関数の復習からです。

1　ラムダ式

関数を引数として渡す時にごく簡単な関数までいちいち名前をつけて defun で定義するのはやっかいである。そこで、名前をつけてその名前を指定する代わりに、「ラムダ式」と呼ばれる形式で関数本体を直接指定することもできる。その形は次の通り。

```
#'(lambda (引数 …) 式 …)
```

ラムダ式とは要するに「defun 関数名」の代わりに lambda と書いたものだと思えばよい。これを使って 2 の平方根を求めるには次のようにする。

```
>(getroot #'(lambda (x) (- (* x x) 2.0)) 0.0 2.0 0.0000001)
1.4142135679721 83
```

これを利用すると、一般の数の平方根を求める関数は次のようにして作れる。

```
(defun sqroot (n)
  (getroot #'(lambda (x) (- (* x x) n)) 0.0 n 0.0000001))
```

なお、ラムダ式の中に使われている「n」は sqroot の引数の「n」で、その値がずっと覚えられていて getroot の中からこのラムダ式 (に対応する関数) が呼ばれた時に使われる。これはちょっと不思議な感じがするけれど、なかなか便利である。

2　マップ関数

まずは復習であるが。

- (mapcar 関数 リスト): リストの各要素に関数を適用し、その結果をリストにして返す。
- (mapcan 関数 リスト): 上と同様だけれど、関数の結果はリスト (nil でもよい) である必要がある。mapcan の値は各関数呼び出しが返した値を連結したものになる。

mapcar、mapcan などの関数をマップ関数と呼ぶ。これは、リストの各要素に引数として与えた関数を適用した写像 (マップ) を集めて結果にするからである。では練習問題。

練習 18 次の関数をマップ関数を利用して書け。

*筑波大学大学院経営システム科学専攻

- 数値のリスト L を受け取り、各要素の値を 1 増やしたリストを返す関数 add1list。
- 数値のリスト L を受け取り、各要素の正、負、零に対応して P、M、Z という記号を並べたリストを返す関数 pmz。
- 数値のリスト L を受け取り、その中の正の要素のみを集めたリストを返す関数 selectplus。
- 0 以上の数値のリスト L を受け取り、その各要素をその数だけ繰り返したリストを返す関数 repnlist。

しかし、単にそういう意味だけでなく、マップ関数は「リストの各要素を順に処理する」という意味合いで使うこともできる。例えば次の例を見ていただく。

```
>(defun accum (l &aux a)
    (setq a 0)
    (mapcar #'(lambda (x) (setq a (+ a x)) a) l))
ACCUM
>(accum '(1 2 3 4 5))
(1 3 6 10 15)
```

3 setf

これまで、「こういうリストを作れ」という時は常に cons、list、append などでそういうリストを部品から組み立てていた。しかし、既にあるリストの一部分を直接変更できたらいいと思ったことはないだろうか? 実は setf というのを使うとそれができる。

(setf 場所 値): 「場所」のところに「値」を書き込む。

例えば次の通り。

```
>(setq x '(1 2 3))
(1 2 3)
>(setf (cadr x) 4)
4
>x
(1 4 3)                    ←書き変わっている!
>(setf (cddr x) '(5 6 7))
(5 6 7)
>x
(1 4 5 6 7)                ←書き変わっている!
```

ただし、こういう副作用は時として危険なことがあるので注意して使わないといけない。例えば次のは何がおきるか?

```
(setf (cddr x) x)
```

4 属性リスト

実は Lisp にも配列やレコードは存在して、setf で値を格納する「場所」としてはそういうのを使う方が普通である。でも配列やレコードはあまり Lisp らしくないから略。その代わりに属性リストというのをやっておく。

実は、すべての記号アトムには属性リストという格納場所が付随している。属性リストには<属性、値>の対を格納しておける。格納されている値を参照するのには

(get 記号 属性)

を使用する。(属性もまた記号アトムである。) そして、値を設定するにはこれを「場所」として setf を用いる。例えば次の通り。

```
>(setf (get 'kuno 'age) 20)
20
>(get 'kuno 'age)
20
```

練習 19 テキストの練習問題 11.1〜11.6。ただし、やりやすくするために予めデータを用意した。

/ua/kuno/work/ex6data.lsp

に入っているので、コピーしてきて load されたい。年齢や性別は自分で add-data を作った後で入れること。

練習 20 先のデータファイルの情報をもとに、次のような計算をする関数を作れ。

- 商品名を与えると、その商品がどことどこのストアにあるかをリストとして返す関数 query-goods-store。
- 商品ごとにその商品とそれが何箇所のストアにあるかを長さ 2 のリストにしたもののを各商品について集めたリストを返す関数 count-goods-stores。

5 eval、apply

ごく最初の方で「Lisp ではプログラムを実行することを『評価する』という」などという説明をしたことをご記憶だろうか。例えば

(+ 1 2 3) →評価→ 6

のようになる。これは要するに、左側の S 式を KCL に向かって打ち込むと右側の式が返ってくる、というのに等しい。ところで、Lisp には評価を強制する (余計に行なわせる) 関数 eval というのが備わっている。すなわち、

'(+ 1 2 3) →評価→ (+ 1 2 3)

ですよね? この右側はさらに評価すると先に書いたように 6 になる。従って、

(eval '(+ 1 2 3)) →評価→ 6

となるわけである。これを利用して、次のようなことを考えてみる。例えば「(1 2 3 4 5)」のような数のリストを受け取って、その合計を計算するのに、これまでだと再帰的関数を使って計算していた。しかし、次のように考えたらどうか?

- そのリストの頭に'+をつけ加える。
- できあがったリスト ((+ 数値…)) を評価する。

```
(defun listsum1 (l) (eval (cons '+ l)))
```

ちょっと面白いでしょう? ところで、こういうことは時々必要になるので、いちいち cons で関数名をくっつける代わりに apply というのが用意されている。

```
(apply 関数 引数のリスト)
```

これを使うと上の例は次のように書ける。

```
(defun listsum2 (l) (apply #'+ l))
```

ところで、前回やった funcall と apply の違いがお分かりか?

```
(apply 関数 引数1 引数2 引数3 …)
```

このように、引数を 1 つのリストにまとめて渡すのが apply、ばらばらに与えるのが funcall ということになる。

練習 21 次のような関数を作れ。

- 変数のリストを渡すと、それぞれの変数に入っている値のリストを返す関数 varvalues。
- Lisp の式の形をした S 式を渡すと、その S 式の内容を 3 回実行する関数 exec3times。
- 数が入った変数の名前と数 N を渡すと、その変数の内容を N ふやす関数 addvartonum。
- 変数のリストを渡すと、それぞれの変数に値 nil を設定する関数 nilvars。

6 小課題その 4 — 属性リストによるデータベース

そろそろ次の課題を出してもいいかな? 属性とかをやって、ようやく Lisp らしくなったので、それを駆使してみましょう。これは単一の大きな課題ではなく、まず属性を使ったデータベースを用意し、それの上で問い合わせを計算するいくつかの関数を (好みにより選択して) 書くというものです。

まず、あなたの親戚や知合いとその親子を 20 人程度リストアップします。3〜4 世代いることが望ましいです。20 人思いつかなければ架空の人を入れて結構です。以下で正確なデータが分からなければ適当にでっちあげても構いません。そして…

- すべての人の名前を変数*persons*に入れます。(同名の人は適当に避けてください。)
- それぞれの人について、年齢、性別、友達のリスト、(いれば) 配偶者、(いれば) 子供のリストのデータを属性として設定してください。

この上で次のような問い合わせを行なう関数を作ってください。

- 人の名前を受け取り、その人の親のリストを返す関数 query-parent。
- 人の名前を受け取り、その人の孫のリストを返す関数 query-grandsons。
- 人の名前を受け取り、その人の血縁者すべてのリストを返す関数 query-sinseki。
- 人の名前を受け取り、「友達の友達はみな友達だ」という原理のもとでその人のすべての友達を返す関数 query-friends。
- 人の名前を受け取り、その人が結婚しているか否かを返す述語 is-married。
- 人の名前を受け取り、その人のいとこを返す関数 query-itoko。

できれば 4 つ以上、最低 3 つはやって頂きたいです。結果には使用したデータと実行例を含めること。

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＃ 3)、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 (10/13) の直前までとしますが、遅れても受理はします。小課題の提出は義務ではありません。