■ ソケット関連

■ プログラムをリストに格納する

dbm とか

■ dbm 変換

gdbm で変換していたファイルがあったのですが、Gauche-0.7.2 でサポートされた
fsdbm の方が都合よさげだったので、一括変換してみました。

```
#!/usr/local/bin/gosh

(use dbm)
(use dbm.gdbm)
(use dbm.fsdbm)

; open the database
(define *gdb* (dbm-open <gdbm>
     :path "/home/www/wiliki/wiki.dbm" :rw-mode :read))
(define *fsdb* (dbm-open <fsdbm>
     :path "/home/miyo/src/gauche/test" :rw-mode :write))

(dbm-for-each *gdb*
          (lambda (key value)
        (dbm-put! *fsdb* key value)))

(dbm-close *gdb*)
(dbm-close *fsdb*)
```

束縛とか

■ letrec

再帰的に呼び出す関数を局所的に定義したいときは
letrec を使えばいい

```
;; @desc リストの要素の全ての組み合わせをリストにして返す
;; @param given-list リスト
(define (bisl.list.util.get-all-list-combinated given-list)
  (cond ((= (length given-list) 1) given-list)
  ((= (length given-list) 2)
  (list given-list (reverse given-list)))
 (#t
  (letrec ((f (lambda (a ll)
         (if (null? (cdr ll)) ; リストの最終
       (bisl.list.util.get-all-list-add-atom a (car ll))
       (append
        (bisl.list.util.get-all-list-add-atom a (car ll))
        (f a (cdr ll)))))))
   (f (car given-list)
      (bisl.list.util.get-all-list-combinated
       (cdr given-list)))))))
```

---

socket とか

■ Socket 通信 UDP

```
;; @desc UPL/UDP 通信のためのソケットを開く
;; @param port ポート番号
;; @param addr ホスト名 ( 文字列 )
(define (fefe port hostname)
  (socket-bind
   (make-socket AF_INET SOCK_DGRAM)
   (make <sockaddr-in> :host hostname :port port)))
```

続 Socket 通信 UDP

上のでは実はうまくいかない
きちんと fd を取得して、ポートを開く必要あり

```
(use srfi-1)
(use gauche.net)

(define (send-udp-packet sockaddr packet)
  (let* ((socket (make-socket AF_INET SOCK_DGRAM)))
    (socket-connect socket sockaddr)
    (let* ((fd (socket-fd socket))
     (port (open-output-fd-port fd)))
      (for-each (lambda (x)
      (write-byte x port))
    packet)
      (flush port))))

(define (main args)
  (send-udp-packet
   (make <sockaddr-in> :host "moss" :port 12345)
   '(254 2 3 4 5 6)))
```

■ socket-bind

上のコードの場合にはローカル側が適当なポートに bind される。
なので socket-bind でローカルで bind することが必要

---

list 遊び

■ slice

作ろうとおもってたら、組み込まれてました

```
(use util.list)

Function: slices list k &optional fill? padding

  (slices '(a b c d e f g) 3)
  => ((a b c) (d e f) (g))
(slices '(a b c d e f g) 3 #t 'z)
  => ((a b c) (d e f) (g z z))
```

list を、それぞれの長さが k であるようなサブリスト (スライス) に分割します。 list の長さが k の整数倍でない場合は、最後のスライスは take* と同じ方法で扱われます。つまり、デフォルトでは k より短いもの、あるいは fill? が真ならば padding が追加されます。