

```

1:  【Lisp(JunLispInterpreter)マニュアル】                2010/11/09現在
2:
3:  最小なLispの処理系(インタプリタ)です。プロパティリストを具備し
4:  ていますので、apval,subr,fsubr,expr,fexprなどの属性を利用して処理
5:  系が構築されています。浅い束縛を行ないます。
6:
7:  アトム
8:
9:  Lispのアトムには以下のようなものがあります。
10:
11:      記号
12:      数
13:      文字列
14:      オブジェクト
15:
16:  記号は、英字と「+-<>=~/_:,¥」で始まり、二文字目以降には、これ
17:  らの文字に加えて数字も使用できます。
18:      +
19:      _private
20:      aoki
21:      atsushi
22:      symbol12345
23:
24:  数は、Smalltalkの数オブジェクトと同様な書き方をします。
25:      123
26:      -123
27:      123.456
28:      -123.456
29:      123.456e7
30:      -123.456e-7
31:      16r123
32:
33:  文字列は、「"」と「"」で囲って記述します。「"」を文字列の中に
34:  書きたい場合には「"""」というように重ねて書きます。
35:      "AOKI Atushi"
36:      "You are ""cool""."
37:
38:  オブジェクトは、「{」と「}」で囲ってSmalltalkのメッセージ式を
39:  記述します。「{」と「}」で囲まれた部分は、まずSmalltalk内部で実
40:  行され、その実行結果がLispのアトムとして扱われます。なお、「{」
41:  は「{nil}」と同義です。
42:      {100@100}
43:      {Image fromUser}
44:      {('aaa') at: 2 put: $z; yourself}
45:
46:  リスト
47:
48:  リストは、「(」 「.」 「)」そしてスペースなどの区切り文字を使用
49:  して表わします。空リストは「()」のように角カッコを二つ合わせて書
50:  いたり、「nil」と書いたりします。ドットペアは「(AOKI . Atsushi)」
51:  のように表わします。「AOKI」が頭部(car)であり、「Atsushi」が尾部
52:  (cdr)です。「.」の代わりにスペースなどの区切り文字を用いて
53:  「(AOKI Atsushi)」と表わすと、「AOKI」が頭部になり、「(Atsushi)」
54:  が尾部になります。「(AOKI Atsushi)」と「(AOKI . (Atsushi))」は同
55:  義です。また、「()」または「nil」はSmalltalkオブジェクトの「nil」

```

```

56: と異なります。
57:     nil
58:     ()
59:     (AOKI . Atsushi)
60:     (AOKI Atsushi)
61:     (AOKI . (Atsushi))
62:
63: システム関数
64:
65:   Lispのシステム内にあらかじめ組み込まれている関数をシステム関数
66:   と呼び、それらは以下の4個のグループに大別されます。
67:
68:       SUBRシステム関数
69:       FSUBRシステム関数
70:       EXPRシステム関数
71:       FEXPRシステム関数
72:
73:   上記の4個のグループに大別されて組み込まれているシステム関数を
74:   それぞれ順番に説明していきます。なお、引数は次のような規則で記述
75:   します。
76:
77:       X, X1, X2, ... , Xn   式
78:       Y, Y1, Y2, ... , Yn   式
79:       L, L1, L2, ... , Ln   リスト
80:       A   アトム
81:       M, N   数アトム
82:       Fn   関数
83:       Alist   連想リスト
84:
85: SUBRシステム関数
86:
87:   SUBRシステム関数は、引数の数が固定で、関数本体はSmalltalkで記
88:   述されています。引数を評価します。実際の関数は、以下の式を評価し
89:   て求めることができます。
90:   (getprop 'append 'subr)
91:
92:   <   (< M N)
93:       M<Nならばtを答え、それ以外ならばnilを答えます。
94:
95:   <=   (<= M N)
96:       M<=Nならばtを答え、それ以外ならばnilを答えます。
97:
98:   =   (= X Y)
99:       X=Yならばtを答え、それ以外ならばnilを答えます。equalと同じです。
100:
101:   ==   (== X Y)
102:       X==Yならばtを答え、それ以外ならばnilを答えます。eqと同じです。
103:
104:   >   (> M N)
105:       M>Nならばtを答え、それ以外ならばnilを答えます。
106:
107:   >=   (>= M N)
108:       M>=Nならばtを答え、それ以外ならばnilを答えます。
109:
110:   append   (append L1 L2)

```

```

111:  L1とL2のリストを結合して一つのリストにします。(L1 . L2)のL1の
112:  部分は新たに複製されます。nconcと比較してみてください。
113:      > (append '(1 2) '(3 4))
114:      (1 2 3 4)
115:      > (do
116:          (x y)
117:          (setq x '(1 2))
118:          (setq y (append x '(3 4)))
119:          (eq x y))
120:      nil
121:
122:  atom    (atom X)
123:  Xがアトムならばtを答え、それ以外ならばnilを答えます。
124:      > (atom 'nil)
125:      t
126:      > (atom '123)
127:      t
128:      > (atom '"string")
129:      t
130:      > (atom 'symbol)
131:      t
132:      > (atom '{100@100})
133:      t
134:      > (atom '(1 2 3))
135:      nil
136:
137:  car      (car L)
138:  リストLの先頭の要素を答えます。
139:      > (car '(1 2 3))
140:      1
141:
142:  cdr      (cdr L)
143:  リストLの先頭の要素を抜いたリストを答えます。
144:      > (cdr '(1 2 3))
145:      (2 3)
146:
147:  clear     (clear)
148:  出力をきれいにします。常にtを答えます。
149:      > (clear)
150:
151:      t
152:
153:  cons      (cons X L)
154:  リストLの先頭に要素Xを加えたリストを答えます。
155:      > (cons 1 '(2 3))
156:      (1 2 3)
157:
158:  consp     (consp X)
159:  Xがドットペア(コンセル)であればtを答え、それ以外ならばnilを
160:  答えます。dtprと同等です。
161:      > (consp '(1 2 3))
162:      t
163:      > (consp '())
164:      nil
165:

```

```

166: dtpr (dtpr X)
167: Xがドットペア(コンセル)であればtを答え、それ以外ならばnilを
168: 答えます。conspと同等です。
169:
170: doublep (doublep X)
171: Xが単精度浮動小数点数ならばtを答え、それ以外ならばnilを答えます。
172: > (doublep 123)
173: nil
174: > (doublep 123.456)
175: nil
176: > (doublep 123.456d)
177: t
178: > (doublep 123.456e7)
179: nil
180: > (doublep 123.456d7)
181: t
182: > (doublep "string")
183: nil
184: > (doublep '(1 2 3))
185: nil
186:
187: eq (eq X Y)
188: X==Yならばtを答え、それ以外ならばnilを答えます。==と同じです。
189: 厳密な等値チェックなので、複製はnilとなります。
190: > (eq 123 123)
191: t
192: > (eq "symbol" "symbol")
193: nil
194: > (eq 'symbol 'symbol)
195: t
196: > (eq {100@100} {100@100})
197: nil
198: > (do
199:     (x y)
200:     (setq x "string")
201:     (setq y x)
202:     (eq x y))
203: t
204:
205: equal (equal X Y)
206: X=Yならばtを答え、それ以外ならばnilを答えます。=と同じです。
207: 厳密な等値チェックではないので、複製はtとなります。
208: > (eq 123 123)
209: t
210: > (equal "symbol" "symbol")
211: t
212: > (equal 'symbol 'symbol)
213: t
214: > (equal {100@100} {100@100})
215: t
216: > (do
217:     (x y)
218:     (setq x "string")
219:     (setq y x)
220:     (equal x y))

```

```

221:      t
222:
223: eval    (eval X)
224:   Xを評価したものを答えます。
225:   > (eval '(append '(1 2) '(3 4)))
226:   (1 2 3 4)
227:
228: exprs    (exprs)
229:   EXPR関数の記号をリストにして答えます。
230:   > (exprs)
231:   (++) -- assoc copy ... )
232:
233: fexprs   (fexprs)
234:   FEXPR関数の記号をリストにして答えます。
235:   > (fexprs)
236:   (and or ... )
237:
238: floatp   (floatp X)
239:   Xが単精度浮動小数点数ならばtを答え、それ以外ならばnilを答えます。
240:   > (floatp 123)
241:   nil
242:   > (floatp 123.456)
243:   t
244:   > (floatp 123.456d)
245:   nil
246:   > (floatp 123.456e7)
247:   t
248:   > (floatp 123.456d7)
249:   nil
250:   > (floatp "string")
251:   nil
252:   > (floatp '(1 2 3))
253:   nil
254:
255: fsubrs   (fsubrs)
256:   FSUBR関数の記号をリストにして答えます。
257:   > (fsubrs)
258:   (* + - / // cond defun ... )
259:
260: gc       (gc)
261:   ガベジコレクションを行ない、常にtを答えます。
262:   > (gc)
263:   t
264:
265: gensym   (gensym)
266:   新たな記号を答えます。
267:   > (gensym)
268:   id8253100
269:
270: getprop  (getprop A X)
271:   AのプロパティリストにXの属性があれば、その属性値を答えます。な
272:   ければnilを答えます。
273:   > (getprop '++ 'expr)
274:   (++ lambda (x) (+ x 1))
275:

```

```

276: integerp (integerp X)
277:   Xが整数ならばtを答え、それ以外ならばnilを答えます。
278:   > (integerp 123)
279:   t
280:   > (integerp 123.456)
281:   nil
282:   > (integerp 123.456d)
283:   nil
284:   > (integerp 123.456e7)
285:   nil
286:   > (integerp 123.456d7)
287:   nil
288:   > (integerp "string")
289:   nil
290:   > (integerp '(1 2 3))
291:   nil
292:
293: last (last L)
294:   リストLの一番最後の要素を含むリストを答えます。
295:   > (last '(1 2 3 4))
296:   (4)
297:
298: length (length L)
299:   リストLの長さを答えます。
300:   > (length '(1 2 3))
301:   3
302:   > (length '(1 (2 3) 4))
303:   3
304:
305: listp (listp X)
306:   Xがリストならばtを答え、それ以外ならばnilを答えます。
307:   > (listp '(1 2 3))
308:   t
309:   > (listp nil)
310:   t
311:   > (listp 'symbol)
312:   nil
313:
314: member (member X L)
315:   XがリストLの要素であれば、その要素が先頭となるリストを答えます。
316:   もし、なければnilを答えます。等値チェックにはequalが使用されます。
317:   > (member 2 '(1 2 3))
318:   (2 3)
319:   > (member '(2) '(1 (2) 3))
320:   ((2) 3)
321:   > (member 4 '(1 2 3))
322:   nil
323:
324: memq (memq X L)
325:   XがリストLの要素であれば、その要素が先頭となるリストを答えます。
326:   もし、なければnilを答えます。等値チェックにはeqが使用されます。
327:   > (memq 2 '(1 2 3))
328:   (2 3)
329:   > (memq '(2) '(1 (2) 3))
330:   nil

```

```

331:      > (memq 4 '(1 2 3))
332:      nil
333:
334: nconc  (nconc L1 L2)
335:  L1とL2のリストを結合して一つのリストにします。(L1 . L2)のL1の
336:  部分は複製されず、元々のL1の後ろにL2が結合されます。appendと比較
337:  してみてください。
338:      > (nconc '(1 2) '(3 4))
339:      (1 2 3 4)
340:      > (do
341:          (x y)
342:          (setq x '(1 2))
343:          (setq y (nconc x '(3 4)))
344:          (eq x y))
345:      t
346:
347: neq  (neq X Y)
348:  eqの否定です。
349:
350: nequal  (nequal X Y)
351:  equalの否定です。
352:
353: nospy  (nospy X)
354:  Xのトレース情報を出力しないように設定し、そのXを答えます。
355:      > (nospy 'sigma)
356:      sigma
357:
358: not  (not X)
359:  Xの否定を答えます。Xがnilならばtを答え、それ以外ならばnilを答
360:  えます。nullと同様です。
361:
362: notrace  (notrace)
363:  全トレース情報を出力しないように設定します。常にtを答えます。
364:      > (notrace)
365:      t
366:
367: nth  (nth X L)
368:  リストLのX番目の要素を答えます。XがリストLの要素を指示できない
369:  ならばnilを答えます。
370:      > (nth 2 '(1 2 3))
371:      2
372:      > (nth 4 '(1 2 3))
373:      nil
374:
375: null  (null X)
376:  Xがnilならばtを答え、それ以外ならばnilを答えます。
377:      > (null '(1 2 3))
378:      nil
379:      > (null nil)
380:      t
381:      > (null 'symbol)
382:      nil
383:
384: numberp  (numberp X)
385:  Xが数ならばtを答え、それ以外ならばnilを答えます。

```

```

386:      > (numberp 123)
387:      t
388:      > (numberp 123.456)
389:      t
390:      > (numberp 123.456d)
391:      t
392:      > (numberp 123.456e7)
393:      t
394:      > (numberp 123.456d7)
395:      t
396:      > (numberp "string")
397:      nil
398:      > (numberp '(1 2 3))
399:      nil
400:
401: oblist (oblist)
402:   登録されているすべての記号をリストにして答えます。
403:   > (oblist)
404:   (* + ++ - -- / // < <= = == > >= and append ... )
405:
406: pp (pp X)
407:   Xをプリティプリントします。常にXを答えます。
408:   > (pp '(lambda (x) (cond ((null x) t) (t nil))))
409:   (lambda (x)
410:     (cond ((null x) t)
411:           (t nil)))
412:   (lambda (x) (cond ((null x) t) (t nil)))
413:
414: princ (princ X)
415:   Xをプリントします。復改を出力しません。常にXを答えます。
416:   > (princ '(1 2 3))
417:   (1 2 3)(1 2 3)
418:
419: print (princ X)
420:   Xをプリントし、復改を出力します。常にXを答えます。
421:   > (print '(1 2 3))
422:   (1 2 3)
423:   (1 2 3)
424:
425: putprop (putprop A X Y)
426:   AのプロパティリストにXの属性名でYを属性値として登録します。常
427:   にYを答えます。
428:
429: remprop (remprop A X)
430:   AのプロパティリストからXの属性を削除し、その属性値を答えます。
431:   対応する属性値が見つからないならばnilを答えます。
432:
433: reverse (reverse L)
434:   リストLをリバースしたリストを答えます。
435:
436: rplaca (rplaca L X)
437:   リストLの頭部(car部)の要素をXに置き換え、そのリストを答えます。
438:
439: rplacd (rplacd L X)
440:   リストLの尾部(cdr部)の要素をXに置き換え、そのリストを答えます。

```



```

441:
442: spy      (spy X)
443:   Xのトレース情報出力するように設定し、そのXを答えます。
444:       > (spy 'sigma)
445:       sigma
446:
447: stringp   (stringp X)
448:   Xが文字列ならばtを答え、それ以外ならばnilを答えます。
449:       > (stringp 123)
450:       nil
451:       > (stringp 'symbol)
452:       nil
453:       > (stringp "string")
454:       t
455:       > (stringp '(1 2 3))
456:       nil
457:
458: subrs     (subrs)
459:   SUBR関数の記号をリストにして答えます。
460:       > (subrs)
461:       (< <= = == > >= append atom ... )
462:
463: symbolp   (symbolp X)
464:   Xが記号ならばtを答え、それ以外ならばnilを答えます。
465:       > (symbolp 123)
466:       nil
467:       > (symbolp 'symbol)
468:       t
469:       > (symbolp "string")
470:       nil
471:       > (symbolp '(1 2 3))
472:       nil
473:
474: terpri    (terpri)
475:   復改出力します。常にtを答えます。
476:       > (terpri)
477:
478:       t
479:
480: trace     (trace)
481:   全トレース情報出力するように設定します。常にtを答えます。
482:       > (trace)
483:       t
484:
485: ~=        (~= X Y)
486:   equalの否定です。
487:
488: ~~        (~~ X Y)
489:   eqの否定です。
490:
491: FSUBRシステム関数
492:
493:   FSUBRシステム関数は、引数の数が不定で、関数本体はSmalltalkで記
494:   述されています。引数を評価しません。実際の関数は、以下の式を評価
495:   して求めることができます。

```

```

496: (getprop 'cond 'fsubr)
497:
498: *   (* X1 X2 ... Xn)
499:   X1, X2, ... , Xnを順次乗算した結果を答えます。
500:
501: +   (+ X1 X2 ... Xn)
502:   X1, X2, ... , Xnを順次加算した結果を答えます。
503:
504: -   (- X1 X2 ... Xn)
505:   X1, X2, ... , Xnを順次減算した結果を答えます。
506:
507: /   (/ X1 X2 ... Xn)
508:   X1, X2, ... , Xnを順次除算した結果を答えます。
509:
510: //  (// X1 X2 ... Xn)
511:   X1, X2, ... , Xnを順次余り切り捨て除算した結果を答えます。
512:
513: cond (cond (X1 Y11 Y12 ... Y1n) ... (Xn Yn1 Yn2 ... Ynn))
514:   XiがnilでないならばYi1 Yi2 ... Yinが順番に評価され、関数の値は
515:   Yinとなります。Xiがnilならば次のXi+1を評価していきます。すべての
516:   Xiがnilならばこの関数もnilを答えます。
517:   > (lambda (x)
518:       (cond
519:         ((null x) "nil")
520:         ((numberp x) "number")
521:         ((stringp x) "string")
522:         ((symbolp x) "symbol")
523:         ((atom x) "object")
524:         ((consp x) "cell")
525:         (t "unknown")))) {100@100})
526:   "object"
527:
528: defun (defun A L X1 X2 ... Xn)
529:   (defun A lambda L X1 X2 ... Xn)
530:   (defun A nlambda L X1 X2 ... Xn)
531:   関数の定義をします。Aは関数名、Lは引数リスト、X1, X2, ... , Xn
532:   が関数の本体になります。AとLの間にlambdaを書けばEXPR関数の定義で
533:   あり、nlambdaを書けばFEXPR関数の定義となります。これらを省略すれ
534:   ばEXPR関数の定義とみなされます。
535:   次の例はオブジェクトアトムであるかどうかを答える関数objectpの
536:   定義です。その後、その関数をチェックしています。
537:   > (defun objectp lambda (x)
538:       (cond
539:         ((null x) nil)
540:         ((numberp x) nil)
541:         ((stringp x) nil)
542:         ((symbolp x) nil)
543:         ((atom x) t)
544:         (t nil)))
545:   objectp
546:   > (objectp nil)
547:   nil
548:   > (objectp 123)
549:   nil
550:   > (objectp 123.456)

```

```

551:      nil
552:      > (objectp 123.456d)
553:      nil
554:      > (objectp 'symbol)
555:      nil
556:      > (objectp "string")
557:      nil
558:      > (objectp {100@100})
559:      t
560:      > (objectp '(1 2 3))
561:      nil

```

次の例は分数アトムであるかどうかを答える関数fractionpの定義です。その後、その関数をチェックしています。

```

564:      > (defun fractionp lambda (x)
565:          (cond
566:              ((numberp x)
567:               (cond
568:                   ((integerp x) nil)
569:                   ((floatp x) nil)
570:                   ((doublep x) nil)
571:                   (t t)))
572:              (t nil)))
573:      fractionp
574:      > (fractionp nil)
575:      nil
576:      > (fractionp 123)
577:      nil
578:      > (fractionp 123.456)
579:      nil
580:      > (fractionp 123.456d)
581:      nil
582:      > (fractionp (/ 3 4))
583:      t
584:      > (fractionp 'symbol)
585:      nil
586:      > (fractionp "string")
587:      nil
588:      > (fractionp {100@100})
589:      nil
590:      > (fractionp '(1 2 3))
591:      nil

```

次の例はFEXPR関数を使用した例です。EXPR関数定義を専門に行なうde関数です。そして、そのdeを利用して、引数が二つのplusExprを作成し、実験しています。

```

595:      > (defun de nlambda (x)
596:          (eval (cons
597:                  'defun
598:                  (cons
599:                      (car x)
600:                      (cons 'lambda (cdr x))))))
601:      de
602:      > (de plusExpr (x y) (+ x y))
603:      plusExpr
604:      > (plusExpr 3 (+ 5 4))
605:      12

```

```

606:  次の例もFEXPR関数を使用した例です。FEXPR関数定義を専門に行なう
607:  df関数です。そして、そのdfを利用して、引数が不定のplusFexprを作
608:  成し、実験しています。
609:      > (defun df nlambda (x)
610:          (eval (cons
611:                  'defun
612:                  (cons
613:                    (car x)
614:                    (cons 'nlambda (cdr x)))))))
615:  df
616:      > (df plusFexpr (x)
617:          (do
618:            (y z)
619:            (setq y x)
620:            (setq z 0)
621:            (while
622:              (not (null y))
623:              do
624:                (setq z (+ z (eval (car y))))
625:                (setq y (cdr y)))
626:            z))
627:  plusFexpr
628:      > (plusFexpr 3 (+ 4 5) 5)
629:  17
630:  FEXPR関数はただ一つの引数を持ち、実引数は評価をされずに、その
631:  ただ一つ引数にリストとして束縛されます。一方、EXPR関数は固定個の
632:  引数を持ち、実引数は評価されて、対応する引数に束縛されます。
633:      > ((nlambda (x) (print x) nil) 1 (+ 2 3) 4)
634:  (1 (+ 2 3) 4)
635:  nil
636:      > ((lambda (x y) (print x) (print y) nil) 1 (+ 2 3))
637:  1
638:  5
639:  nil
640:  このdefun関数を利用して、このLispにどんどん新しい関数を加えて、
641:  皆さんの好みのLispにしてみてください。
642:
643:  do (do L X1 X2 ... Xn)
644:  リストLは局所変数リストで、X1, X2, ... , Xnを順次実行します。
645:  Xnの評価結果を答えます。
646:      > (do
647:          (x y)
648:          (setq x 100)
649:          (setq y 200)
650:          (do
651:            (x y) % local variables
652:            (setq x 10000)
653:            (setq y 20000)
654:            (print x)
655:            (print y))
656:          (print x)
657:          (print y)
658:          'end)
659:  10000
660:  20000

```

```

661:      100
662:      200
663:      end
664:
665:  if (if X then X1 X2 ... Xn else Y1 Y2 ... Yn)
666:    Xがnil以外ならば, X1, X2, ... , Xnを順次実行し, Xnの評価結果を
667:    答えます。Xがnilならば, Y1, Y2, ... , Ynを順次実行し, Ynの評価結
668:    果を答えます。else以降を省略することもできます。
669:    > (progn
670:        (if
671:          (= (send {Dialog} `confirm: "Are you happy?")
672:            {true}))
673:          then
674:            (print "yes")
675:          else
676:            (print "no"))
677:        'end)
678:    yes
679:  end
680:
681:  list (list X1 X2 ... Xn)
682:    X1からXnまでをリストにします。Xがないならばnilを答えます。
683:    > (list 1 '(2 3) '(4 (5) 6))
684:    (1 (2 3) (4 (5) 6))
685:
686:  progn (progn X1 X2 ... Xn)
687:    X1, X2, ... , Xnを順次実行します。Xnの評価結果を答えます。
688:    > (progn
689:        (print 1)
690:        (print 2)
691:        (print 3)
692:        'end)
693:    1
694:    2
695:    3
696:  end
697:
698:  quote (quote X)
699:    Xそのものを答えます。シングルクォートを使って「X」のように書
700:    くこともできますし、バッククォートを使って「`X」のように書くこ
701:    ともできます。
702:    > (quote x)
703:    x
704:    > 'x
705:    x
706:    > `x
707:    x
708:    > x
709:    *** Error: x is unbound atom
710:    nil
711:
712:  read (read X Y)
713:    入力します。Xはその際のメッセージ, Yはデフォルトの値です。X
714:    もYもこの関数の中で評価されます。入力されたLispの表現を答えます。
715:    > (do

```

```

715:          (m d)
716:          (setq m "What's your name?")
717:          (setq d "default")
718:          (read m d))
719:      Aoki
720:
721: repeat (repeat X1 X2 ... Xn until Y)
722:   Yがnil以外に評価される間, X1, X2, ..., Xnを順次実行します。Y
723: の評価はループの最後に起こります。最終のXnの評価結果を答えます。
724:   > (do
725:       (count)
726:       (setq count 1)
727:       (repeat
728:         (print count)
729:         (setq count (+ count 1))
730:         until
731:         (<= count 10))
732:       'end)
733:   1
734:   2
735:   3
736:   4
737:   5
738:   6
739:   7
740:   8
741:   9
742:   10
743:   end
744:   > (do
745:       (count)
746:       (setq count 1)
747:       (repeat
748:         (print count)
749:         (setq count (+ count 1))
750:         until
751:         nil)
752:       'end)
753:   1
754:   end
755:
756: send (send X A Y1 Y2 ... Yn)
757:   XにAというメッセージを引数Y1, Y2, ..., Ynを付けて送信します。
758:   メッセージの送信結果が, この関数の値となります。
759:   > (do
760:       (image message canvas point)
761:       (setq image {Image fromUser})
762:       (setq message `displayOn:at:)
763:       (setq canvas
764:         {ScheduledControllers activeController
765:          view graphicsContext})
766:       (setq point {30@50})
767:       (send (send image message canvas point) `inspect))
768:   {Depth8Image(extent: 57@57 depth: 8)}
769:

```

```

770: setq (setq X1 Y1 X2 Y2 ... Xn Yn)
771:   Xiそれ自身にYiを束縛します。(ただし、浅い束縛ゆえに、局所変数
772:   に対してのみ束縛になり、トップレベルにおいては代入となってしまう
773:   ことに注意されたし。)
774:
775: while (while X do Y1 Y2 ... Yn)
776:   Xがnil以外に評価される間、Y1, Y2, ..., Ynを順次実行します。X
777:   の評価はループの先頭に起こります。最終のYnの評価結果を答えます。
778:   > (do
779:       (count)
780:       (setq count 1)
781:       (while
782:           (<= count 10)
783:           do
784:             (print count)
785:             (setq count (+ count 1))))
786:       'end)
787:   1
788:   2
789:   3
790:   4
791:   5
792:   6
793:   7
794:   8
795:   9
796:   10
797:   end
798:   > (do
799:       (count)
800:       (setq count 1)
801:       (while
802:           nil
803:           do
804:             (print count)
805:             (setq count (+ count 1))))
806:       'end)
807:   end
808:
809: ¥¥ (¥¥ X1 X2 ... Xn)
810:   X1, X2, ..., Xnを順次モジュロ演算した結果を答えます。
811:
812: EXPRシステム関数
813:
814:   EXPRシステム関数は、引数の数が固定で、関数本体はLispで記述され
815:   ています。引数を実評価します。実際の関数は、以下の式を実評価して求め
816:   ることができます。
817:   (getprop 'assoc 'expr)
818:
819: ++ (++ N)
820:   Nに1を加えます。
821:
822: -- (-- N)
823:   Nから1を減じます。
824:

```

```

825: assoc (assoc X Alist)
826: 連想リストAlistからXで指定される項目を検索し、それを答えます。
827: ないならばnilを答えます。
828: > (do
829:     (alist)
830:     (setq agelist '((father 38) (mother 38) (son 10)))
831:     (assoc 'son agelist))
832: (son 10)
833:
834: copy (copy L)
835: リストLの複製を答えます。
836:
837: mapc (mapc Fn L)
838: リストLの各々の要素にFnで指定される関数を適用し、nilを答えます。
839: > (do
840:     (alist)
841:     (setq agelist '((father 38) (mother 38) (son 10)))
842:     (mapc 'print agelist))
843: (father 38)
844: (mother 38)
845: (son 10)
846: nil
847: > (do
848:     (alist)
849:     (setq agelist '((father 38) (mother 38) (son 10)))
850:     (mapc '(lambda (x) (print x)) agelist))
851: (father 38)
852: (mother 38)
853: (son 10)
854: nil
855:
856: mapcar (mapcar Fn L)
857: リストLの各々の要素にFnで指定される関数を適用し、その値のリス
858: トを答えます。
859: > (do
860:     (alist)
861:     (setq agelist '((father 38) (mother 38) (son 10)))
862:     (mapcar 'cdr agelist))
863: ((38) (38) (10))
864: > (do
865:     (alist)
866:     (setq agelist '((father 38) (mother 38) (son 10)))
867:     (mapcar '(lambda (x) (car (cdr x))) agelist))
868: (38 38 10)
869:
870: FEXPRシステム関数
871:
872: FEXPRシステム関数は、引数の数が不定で、関数本体はLispで記述さ
873: れています。引数を評価しません。実際の関数は、以下の式を評価して
874: 求めることができます。
875: (getprop 'and 'fexpr)
876:
877: and (and X1 X2 ... Xn)
878: X1, X2, ..., Xnを順次評価し、nilのなるものがあればnilを答えま
879: す。それ以外ならばtを答えます。順次評価中にnilになった時点で、引

```



```

880: 数の評価を止めます。
881:      > (and
882:          (print 100)
883:          (print 200)
884:          (print 300))
885:      100
886:      200
887:      300
888:      t
889:      > (and
890:          (print 100)
891:          (progn (print 200) nil)
892:          (print 300))
893:      100
894:      200
895:      nil
896:
897: or (or X1 X2 ... Xn)
898:   X1, X2, ..., Xnを順次評価して、すべてがnilのならばnilを答えま
899: す。それ以外ならばtを答えます。順次評価中にnilでなくなった時点で、
900: 引数の評価を止めます。
901:      > (or
902:          (progn (print 100) nil)
903:          (progn (print 200) nil)
904:          (progn (print 300) nil))
905:      100
906:      200
907:      300
908:      nil
909:      > (or
910:          (progn (print 100) nil)
911:          (progn (print 200))
912:          (progn (print 300) nil))
913:      100
914:      200
915:      t
916:
917: システム関数一覧
918:
919:      * (FSUBR)
920:      + (FSUBR)
921:      ++ (EXPR)
922:      - (FSUBR)
923:      -- (EXPR)
924:      / (FSUBR)
925:      // (FSUBR)
926:      < (SUBR)
927:      <= (SUBR)
928:      = (SUBR)
929:      == (SUBR)
930:      > (SUBR)
931:      >= (SUBR)
932:      and (FEXPR)
933:      append (SUBR)
934:      assoc (SUBR)

```

```
935:      atom    (SUBR)
936:      car      (SUBR)
937:      cdr      (SUBR)
938:      clear    (SUBR)
939:      cond     (FSUBR)
940:      cons     (SUBR)
941:      consp    (SUBR)
942:      copy     (SUBR)
943:      defun    (FSUBR)
944:      do       (FSUBR)
945:      doublep  (SUBR)
946:      dtp      (SUBR)
947:      eq       (SUBR)
948:      equal    (SUBR)
949:      eval     (SUBR)
950:      exprs    (SUBR)
951:      fexprs   (SUBR)
952:      floatp   (SUBR)
953:      fsubrs   (SUBR)
954:      gc       (SUBR)
955:      gensym   (SUBR)
956:      getprop  (SUBR)
957:      if       (FSUBR)
958:      integerp (SUBR)
959:      last     (SUBR)
960:      length   (SUBR)
961:      list     (FSUBR)
962:      listp    (SUBR)
963:      mapc     (FEXPR)
964:      mapcar   (FEXPR)
965:      member   (SUBR)
966:      memq     (SUBR)
967:      nconc    (SUBR)
968:      neq      (SUBR)
969:      nequal   (SUBR)
970:      nosp     (SUBR)
971:      not      (SUBR)
972:      notrace  (SUBR)
973:      nth      (SUBR)
974:      null     (SUBR)
975:      numberp  (SUBR)
976:      oblist   (SUBR)
977:      or       (FEXPR)
978:      pp       (SUBR)
979:      princ   (SUBR)
980:      print   (SUBR)
981:      progn   (FSUBR)
982:      putprop (SUBR)
983:      quote   (FSUBR)
984:      read    (FSUBR)
985:      remprop (SUBR)
986:      repeat  (FSUBR)
987:      reverse (SUBR)
988:      rplaca  (SUBR)
989:      rplacd  (SUBR)
```

```

990:      send      (FSUBR)
991:      setq      (FSUBR)
992:      spy       (SUBR)
993:      stringp   (SUBR)
994:      subrs     (SUBR)
995:      symbolp   (SUBR)
996:      terpri    (SUBR)
997:      trace     (SUBR)
998:      while     (FSUBR)
999:      ¥¥       (FSUBR)
1000:     ~=        (SUBR)
1001:     ~~        (SUBR)

```

1002: 1003: SUBRシステム関数一覧

```

1004:      <
1005:      <=
1006:      =
1007:      ==
1008:      >
1009:      >=
1010:      append
1011:      atom
1012:      car
1013:      cdr
1014:      clear
1015:      cons
1016:      consp
1017:      doublep
1018:      dtp
1019:      eq
1020:      equal
1021:      eval
1022:      exprs
1023:      fexprs
1024:      floatp
1025:      fsubrs
1026:      gc
1027:      gensym
1028:      getprop
1029:      integerp
1030:      last
1031:      length
1032:      listp
1033:      member
1034:      memq
1035:      nconc
1036:      neq
1037:      nequal
1038:      nospy
1039:      not
1040:      notrace
1041:      nth
1042:      null
1043:      numberp
1044:      oblist

```

```
1045:      pp
1046:      princ
1047:      print
1048:      putprop
1049:      remprop
1050:      reverse
1051:      rplaca
1052:      rplacd
1053:      spy
1054:      stringp
1055:      subrs
1056:      symbolp
1057:      terpri
1058:      trace
1059:      ~=
1060:      ~~
1061:
1062: FSUBRシステム関数一覧
1063:      *
1064:      +
1065:      -
1066:      /
1067:      //
1068:      cond
1069:      defun
1070:      do
1071:      if
1072:      list
1073:      progn
1074:      quote
1075:      read
1076:      repeat
1077:      send
1078:      setq
1079:      while
1080:      ¥¥
1081:
1082: EXPRシステム関数一覧
1083:      ++
1084:      --
1085:      assoc
1086:      copy
1087:      mapc
1088:      mapcar
1089:
1090: FEXPRシステム関数一覧
1091:      and
1092:      or
1093:
1094: Copyright (C) 1984-2010 AOKI Atsushi
```