【Lisp(JunLispInterpreter)マニュアル】　　2010/11/09現在

　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ていますので，apval,subr,fsubr,expr,fexprなどの属性を利用して処理系が構築されています。浅い束縛を行ないます。

アトム

　Lispのアトムには以下のようなものがあります。

　　記号
　　数
　　文字列
　　オブジェクト

　記号は，英字と「+-<>=~*/_:,¥」で始まり，二文字目以降には，これらの文字に加えて数字も使用できます。

```
+
_private
aoki
atsushi
symbol12345
```

　数は，Smalltalkの数オブジェクトと同様な書き方をします。

```
123
-123
123.456
-123.456
123.456e7
-123.456e-7
16r123
```

　文字列は，「"」と「"」で囲って記述します。「"」を文字列の中に書きたい場合には「""」というように重ねて書きます。

```
"AOKI Atushi"
"You are ""cool""."
```

　オブジェクトは，「{」と「}」で囲ってSmalltalkのメッセージ式を記述します。「{」と「}」で囲まれた部分は，まずSmalltalk内部で実行され，その実行結果がLispのアトムとして扱われます。なお，「{}」は「{nil}」と同義です。

```
{100@100}
{Image fromUser}
{('aaa') at: 2 put: $z; yourself}
```

リスト

　リストは，「(」「.」「)」そしてスペースなどの区切り文字を使用して表わします。空リストは「()」のように角カッコを二つ合わせて書いたり，「nil」と書いたりします。ドットペアは「(AOKI . Atsushi)」のように表わします。「AOKI」が頭部(car)であり，「Atsushi」が尾部(cdr)です。「.」の代わりにスペースなどの区切り文字を用いて「(AOKI Atsushi)」と表わすと，「AOKI」が頭部になり，「(Atsushi)」が尾部になります。「(AOKI Atsushi)」と「(AOKI . (Atsushi))」は同義です。また，「()」または「nil」はSmalltalkオブジェクトの「nil」

と異なります。

```
        nil
        ()
        (AOKI . Atsushi)
        (AOKI Atsushi)
        (AOKI . (Atsushi))
```

システム関数

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と呼び，それらは以下の4個のグループに大別されます。

```
        SUBRシステム関数
        FSUBRシステム関数
        EXPRシステム関数
        FEXPRシステム関数
```

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します。

```
        X, X1, X2, ... , Xn   式
        Y, Y1, Y2, ... , Yn   式
        L, L1, L2, ... , Ln   リスト
        A   アトム
        M, N   数アトム
        Fn   関数
        Alist   連想リスト
```

SUBRシステム関数

SUBRシステム関数は，引数の数が固定で，関数本体はSmalltalkで記述されています。引数を評価します。実際の関数は，以下の式を評価して求めることができます。

```
(getprop 'append 'subr)
```

`<`　(< M N)
M<Nならばtを答え，それ以外ならばnilを答えます。

`<=`　(<= M N)
M<=Nならばtを答え，それ以外ならばnilを答えます。

`=`　(= X Y)
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。

`==`　(== X Y)
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。

`>`　(>= M N)
M>Nならばtを答え，それ以外ならばnilを答えます。

`>=`　(> M N)
M>=Nならばtを答え，それ以外ならばnilを答えます。

append　(append L1 L2)

  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
部分は新たに複製されます。nconcと比較してみて下さい。

```
        > (append '(1 2) '(3 4))
        (1 2 3 4)
        > (do
                (x y)
                (setq x '(1 2))
                (setq y (append x '(3 4)))
                (eq x y))
        nil
```

atom   (atom X)
  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。

```
        > (atom 'nil)
        t
        > (atom '123)
        t
        > (atom '"string")
        t
        > (atom 'symbol)
        t
        > (atom '{100@100})
        t
        > (atom '(1 2 3))
        nil
```

car   (car L)
  リストLの先頭の要素を答えます。

```
        > (car '(1 2 3))
        1
```

cdr   (cdr L)
  リストLの先頭の要素を抜いたリストを答えます。

```
        > (cdr '(1 2 3))
        (2 3)
```

clear   (clear)
  出力をきれいにします。常にtを答えます。

```
        > (clear)

        t
```

cons   (cons X L)
  リストLの先頭に要素Xを加えたリストを答えます。

```
        > (cons 1 '(2 3))
        (1 2 3)
```

consp   (consp X)
  Xがドットペア(コンスセル)であればtを答え，それ以外ならばnilを
答えます。dtprと同等です。

```
        > (consp '(1 2 3))
        t
        > (consp '())
        nil
```

```
dtpr   (dtpr X)
  Xがドットペア(コンスセル)であればtを答え，それ以外ならばnilを
答えます。conspと同等です。

doublep   (doublep X)
  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。
        > (doublep 123)
        nil
        > (doublep 123.456)
        nil
        > (doublep 123.456d)
        t
        > (doublep 123.456e7)
        nil
        > (doublep 123.456d7)
        t
        > (doublep "string")
        nil
        > (doublep '(1 2 3))
        nil

eq   (eq X Y)
  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。
厳密な等値チェックなので，複製はnilとなります。
        > (eq 123 123)
        t
        > (eq "symbol" "symbol")
        nil
        > (eq 'symbol 'symbol)
        t
        > (eq {100@100} {100@100})
        nil
        > (do
                (x y)
                (setq x "string")
                (setq y x)
                (eq x y))
        t

equal   (equal X Y)
  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。
厳密な等値チェックではないので，複製はtとなります。
        > (eq 123 123)
        t
        > (equal "symbol" "symbol")
        t
        > (equal 'symbol 'symbol)
        t
        > (equal {100@100} {100@100})
        t
        > (do
                (x y)
                (setq x "string")
                (setq y x)
                (equal x y))
```

```
        t

eval   (eval X)
  Xを評価したものを答えます。
        > (eval '(append '(1 2) '(3 4)))
        (1 2 3 4)

exprs   (exprs)
  EXPR関数の記号をリストにして答えます。
        > (exprs)
        (++ -- assoc copy ... )

fexprs   (fexprs)
  FEXPR関数の記号をリストにして答えます。
        > (fexprs)
        (and or ... )

floatp   (floatp X)
  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。
        > (floatp 123)
        nil
        > (floatp 123.456)
        t
        > (floatp 123.456d)
        nil
        > (floatp 123.456e7)
        t
        > (floatp 123.456d7)
        nil
        > (floatp "string")
        nil
        > (floatp '(1 2 3))
        nil

fsubrs   (fsubrs)
  FSUBR関数の記号をリストにして答えます。
        > (fsubrs)
        (* + - / // cond defun ... )

gc   (gc)
  ガベジコレクションを行ない，常にtを答えます。
        > (gc)
        t

gensym   (gensym)
  新たな記号を答えます。
        > (gensym)
        id8253100

getprop   (getporp A X)
  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
ければnilを答えます。
        > (getprop '++ 'expr)
        (++ lambda (x) (+ x 1))
```

integerp　(integerp X)
　Xが整数ならばtを答え，それ以外ならばnilを答えます。
　　> (integerp 123)
　　t
　　> (integerp 123.456)
　　nil
　　> (integerp 123.456d)
　　nil
　　> (integerp 123.456e7)
　　nil
　　> (integerp 123.456d7)
　　nil
　　> (integerp "string")
　　nil
　　> (integerp '(1 2 3))
　　nil

last　(last L)
　リストLの一番最後の要素を含むリストを答えます。
　　> (last '(1 2 3 4))
　　(4)

length　(length L)
　リストLの長さを答えます。
　　> (length '(1 2 3))
　　3
　　> (length '(1 (2 3) 4))
　　3

listp　(listp X)
　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。
　　> (listp '(1 2 3))
　　t
　　> (listp nil)
　　t
　　> (listp 'symbol)
　　nil

member　(member X L)
　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。
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。
　　> (member 2 '(1 2 3))
　　(2 3)
　　> (member '(2) '(1 (2) 3))
　　((2) 3)
　　> (member 4 '(1 2 3))
　　nil

memq　(memq X L)
　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。
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。
　　> (memq 2 '(1 2 3))
　　(2 3)
　　> (memq '(2) '(1 (2) 3))
　　nil

```
        > (memq 4 '(1 2 3))
        nil

nconc   (nconc L1 L2)
  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
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
してみて下さい。
        > (nconc '(1 2) '(3 4))
        (1 2 3 4)
        > (do
                (x y)
                (setq x '(1 2))
                (setq y (nconc x '(3 4)))
                (eq x y))
        t

neq   (neq X Y)
  eqの否定です。

nequal   (nequal X Y)
  equalの否定です。

nospy   (nospy X)
  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。
        > (nospy 'sigma)
        sigma

not   (not X)
  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
えます。nullと同様です。

notrace   (notrace)
  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。
        > (notrace)
        t

nth   (nth X L)
  リストLのX番目の要素を答えます。Xがリストの要素を指示できない
ならばnilを答えます。
        > (nth 2 '(1 2 3))
        2
        > (nth 4 '(1 2 3))
        nil

null   (null X)
  Xがnilならばtを答え，それ以外ならばnilを答えます。
        > (null '(1 2 3))
        nil
        > (null nil)
        t
        > (null 'symbol)
        nil

numberp   (numberp X)
  Xが数ならばtを答え，それ以外ならばnilを答えます。
```

```
        > (numberp 123)
        t
        > (numberp 123.456)
        t
        > (numberp 123.456d)
        t
        > (numberp 123.456e7)
        t
        > (numberp 123.456d7)
        t
        > (numberp "string")
        nil
        > (numberp '(1 2 3))
        nil

oblist   (oblist)
  登録されているすべての記号をリストにして答えます。
        > (oblist)
        (* + ++ - -- / // < <= = == > >= and append ... )

pp   (pp X)
  Xをプリティプリントします。常にXを答えます。
        > (pp '(lambda (x) (cond ((null x) t) (t nil))))
        (lambda (x)
                 (cond ((null x)  t)
                       (t nil)))
        (lambda (x) (cond ((null x) t) (t nil)))

princ   (princ X)
  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。
        > (princ '(1 2 3))
        (1 2 3)(1 2 3)

print   (princ X)
  Xをプリントし，復改を出力します。常にXを答えます。
        > (print '(1 2 3))
        (1 2 3)
        (1 2 3)

putprop   (putprop A X Y)
  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
にYを答えます。

remprop   (remprop A X)
  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。
対応する属性値が見つからないならばnilを答えます。

reverse   (reverse L)
  リストLをリバースしたリストを答えます。

rplaca   (rplaca L X)
  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。

rplacd   (rplacd L X)
  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。
```

```
spy   (spy X)
  Xのトレース情報を出力するように設定し、そのXを答えます。
        > (spy 'sigma)
        sigma

stringp   (stringp X)
  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。
        > (stringp 123)
        nil
        > (stringp 'symbol)
        nil
        > (stringp "string")
        t
        > (stringp '(1 2 3))
        nil

subrs   (subrs)
  SUBR関数の記号をリストにして答えます。
        > (subrs)
        (< <= = == > >= append atom ... )

symbolp   (symbolp X)
  Xが記号ならばtを答え，それ以外ならばnilを答えます。
        > (symbolp 123)
        nil
        > (symbolp 'symbol)
        t
        > (symbolp "string")
        nil
        > (symbolp '(1 2 3))
        nil

terpri   (terpri)
  復改を出力します。常にtを答えます。
        > (terpri)

        t

trace   (trace)
  全トレース情報を出力するように設定します。常にtを答えます。
        > (trace)
        t

~=   (~= X Y)
  equalの否定です。

~~   (~~ X Y)
  eqの否定です。

FSUBRシステム関数

  FSUBRシステム関数は，引数の数が不定で，関数本体はSmalltalkで記
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
して求めることができます。
```

```
(getprop 'cond 'fsubr)

*   (* X1 X2 ... Xn)
  X1, X2, ... , Xnを順次乗算した結果を答えます。

+   (+ X1 X2 ... Xn)
  X1, X2, ... , Xnを順次加算した結果を答えます。

-   (- X1 X2 ... Xn)
  X1, X2, ... , Xnを順次減算した結果を答えます。

/   (/ X1 X2 ... Xn)
  X1, X2, ... , Xnを順次除算した結果を答えます。

//   (// X1 X2 ... Xn)
  X1, X2, ... , Xnを順次余り切り捨て除算した結果を答えます。

cond   (cond (X1 Y11 Y12 ... Y1n) ... (Xn Yn1 Yn2 ... Ynn))
  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
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
Xiがnilならばこの関数もnilを答えます。
        > ((lambda (x)
                (cond
                        ((null x) "nil")
                        ((numberp x) "number")
                        ((stringp x) "string")
                        ((symbolp x) "symbol")
                        ((atom x) "object")
                        ((consp x) "cell")
                        (t "unknown"))) {100@100})
        "object"

defun   (defun A L X1 X2 ... Xn)
        (defun A lambda L X1 X2 ... Xn)
        (defun A nlambda L X1 X2 ... Xn)
  関数の定義をします。Aは関数名，Lは引数リスト，X1, X2, ... , Xn
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
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
ばEXPR関数の定義とみなされます。
  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
定義です。その後，その関数をチェックしています。
        > (defun objectp lambda (x)
                (cond
                        ((null x) nil)
                        ((numberp x) nil)
                        ((stringp x) nil)
                        ((symbolp x) nil)
                        ((atom x) t)
                        (t nil)))
        objectp
        > (objectp nil)
        nil
        > (objectp 123)
        nil
        > (objectp 123.456)
```

```
        nil
        > (objectp 123.456d)
        nil
        > (objectp 'symbol)
        nil
        > (objectp "string")
        nil
        > (objectp {100@100})
        t
        > (objectp '(1 2 3))
        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。

```
        > (defun fractionp lambda (x)
                (cond
                        ((numberp x)
                                (cond
                                        ((integerp x) nil)
                                        ((floatp x) nil)
                                        ((doublep x) nil)
                                        (t t)))
                        (t nil)))
        fractionp
        > (fractionp nil)
        nil
        > (fractionp 123)
        nil
        > (fractionp 123.456)
        nil
        > (fractionp 123.456d)
        nil
        > (fractionp (/ 3 4))
        t
        > (fractionp 'symbol)
        nil
        > (fractionp "string")
        nil
        > (fractionp {100@100})
        nil
        > (fractionp '(1 2 3))
        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。

```
        > (defun de nlambda (x)
                (eval (cons
                                'defun
                                (cons
                                        (car x)
                                        (cons 'lambda (cdr x))))))
        de
        > (de plusExpr (x y) (+ x y))
        plusExpr
        > (plusExpr 3 (+ 5 4))
        12
```

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を作成し，実験しています。

```
        > (defun df nlambda (x)
                (eval (cons
                                'defun
                                (cons
                                        (car x)
                                        (cons 'nlambda (cdr x))))))
        df
        > (df plusFexpr (x)
                (do
                        (y z)
                        (setq y x)
                        (setq z 0)
                        (while
                                (not (null y))
                                do
                                (setq z (+ z (eval (car y))))
                                (setq y (cdr y)))
                        z))
        plusFexpr
        > (plusFexpr 3 (+ 4 5) 5)
        17
```

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。

```
        > ((nlambda (x) (print x) nil) 1 (+ 2 3) 4)
        (1 (+ 2 3) 4)
        nil
        > ((lambda (x y) (print x) (print y) nil) 1 (+ 2 3))
        1
        5
        nil
```

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，皆さんの好みのLispにしてみて下さい。

do　(do L X1 X2 ... Xn)

リストLは局所変数リストで，X1，X2，... ，Xnを順次実行します。Xnの評価結果を答えます。

```
        > (do
                (x y)
                (setq x 100)
                (setq y 200)
                (do
                        (x y) % local variables
                        (setq x 10000)
                        (setq y 20000)
                        (print x)
                        (print y))
                (print x)
                (print y)
                'end)
        10000
        20000
```

```
        100
        200
        end

if   (if X then X1 X2 ... Xn else Y1 Y2 ... Yn)
  Xがnil以外ならば，X1，X2，... ，Xnを順次実行し，Xnの評価結果を
答えます。Xがnilならば，Y1，Y2，... ，Ynを順次実行し，Ynの評価結
果を答えます。else以降を省略することもできます。
        > (progn
                (if
                        (= (send {Dialog} `confirm: "Are you happy?")
{true})
                        then
                        (print "yes")
                        else
                        (print "no"))
                'end)
        yes
        end

list   (list X1 X2 ... Xn)
  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。
        > (list 1 '(2 3) '(4 (5) 6))
        (1 (2 3) (4 (5) 6))

progn   (progn X1 X2 ... Xn)
  X1，X2，... ，Xnを順次実行します。Xnの評価結果を答えます。
        > (progn
                (print 1)
                (print 2)
                (print 3)
                'end)
        1
        2
        3
        end

quote   (quote X)
  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
くこともできますし，バッククォートを使って「`X」のように書くこと
もできます。
        > (quote x)
        x
        > 'x
        x
        > `x
        x
        > x
        *** Error: x is unbound atom
        nil

read   (read X Y)
  入力をします。Xはその際のメッセージ，Yはデフォルトの値です。X
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。
        > (do
```

```
                (m d)
                (setq m "What's your name?")
                (setq d "default")
                (read m d))
        Aoki

repeat   (repeat X1 X2 ... Xn until Y)
  Yがnil以外に評価される間，X1，X2，... ，Xnを順次実行します。Y
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。
        > (do
                (count)
                (setq count 1)
                (repeat
                        (print count)
                        (setq count (+ count 1))
                        until
                        (<= count 10))
                'end)
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
        end
        > (do
                (count)
                (setq count 1)
                (repeat
                        (print count)
                        (setq count (+ count 1))
                        until
                        nil)
                'end)
        1
        end

send   (send X A Y1 Y2 ... Yn)
  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。
メッセージの送信結果が，この関数の値となります。
        > (do
                (image message canvas point)
                (setq image {Image fromUser})
                (setq message `displayOn:at:)
                (setq canvas
                        {ScheduledControllers activeController
                                view graphicsContext})
                (setq point {30@50})
                (send (send image message canvas point) `inspect))
        {Depth8Image(extent: 57@57 depth: 8)}
```

```
setq   (setq X1 Y1 X2 Y2 ... Xn Yn)
  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
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
ことに注意されたし。）

while   (while X do Y1 Y2 ... Yn)
  Xがnil以外に評価される間，Y1，Y2，... ，Ynを順次実行します。X
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。
        > (do
                (count)
                (setq count 1)
                (while
                        (<= count 10)
                        do
                        (print count)
                        (setq count (+ count 1)))
                'end)
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
        end
        > (do
                (count)
                (setq count 1)
                (while
                        nil
                        do
                        (print count)
                        (setq count (+ count 1)))
                'end)
        end

¥¥   (¥¥ X1 X2 ... Xn)
  X1，X2，... ，Xnを順次モジュロ演算した結果を答えます。

EXPRシステム関数

  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
ています。引数を評価します。実際の関数は，以下の式を評価して求め
ることができます。
(getprop 'assoc 'expr)

++   (++ N)
  Nに1を加えます。

--   (-- N)
  Nから1を減じます。
```

```
assoc   (assoc X Alist)
  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。
ないならばnilを答えます。
        > (do
                (alist)
                (setq agelist '((father 38) (mother 38) (son 10)))
                (assoc 'son agelist))
        (son 10)

copy   (copy L)
  リストLの複製を答えます。

mapc   (mapc Fn L)
  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。
        > (do
                (alist)
                (setq agelist '((father 38) (mother 38) (son 10)))
                (mapc 'print agelist))
        (father 38)
        (mother 38)
        (son 10)
        nil
        > (do
                (alist)
                (setq agelist '((father 38) (mother 38) (son 10)))
                (mapc '(lambda (x) (print x)) agelist))
        (father 38)
        (mother 38)
        (son 10)
        nil

mapcar   (mapcar Fn L)
  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
トを答えます。
        > (do
                (alist)
                (setq agelist '((father 38) (mother 38) (son 10)))
                (mapcar 'cdr agelist))
        ((38) (38) (10))
        > (do
                (alist)
                (setq agelist '((father 38) (mother 38) (son 10)))
                (mapcar '(lambda (x) (car (cdr x))) agelist))
        (38 38 10)

FEXPRシステム関数

  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
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
求めることができます。
(getprop 'and 'fexpr)

and   (and X1 X2 ... Xn)
  X1, X2, ... , Xnを順次評価し，nilのなるものがあればnilを答えま
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
```

数の評価を止めます。

```
        > (and
                (print 100)
                (print 200)
                (print 300))
        100
        200
        300
        t
        > (and
                (print 100)
                (progn (print 200) nil)
                (print 300))
        100
        200
        nil
```

or　(or X1 X2 ... Xn)
　X1, X2, ... , Xnを順次評価して，すべてがnilのならばnilを答えます。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，引数の評価を止めます。

```
        > (or
                (progn (print 100) nil)
                (progn (print 200) nil)
                (progn (print 300) nil))
        100
        200
        300
        nil
        > (or
                (progn (print 100) nil)
                (progn (print 200))
                (progn (print 300) nil))
        100
        200
        t
```

システム関数一覧

```
        *   (FSUBR)
        +   (FSUBR)
        ++   (EXPR)
        -   (FSUBR)
        --   (EXPR)
        /   (FSUBR)
        //   (FSUBR)
        <   (SUBR)
        <=   (SUBR)
        =   (SUBR)
        ==   (SUBR)
        >   (SUBR)
        >=   (SUBR)
        and   (FEXPR)
        append   (SUBR)
        assoc   (SUBR)
```

```
        atom   (SUBR)
        car   (SUBR)
        cdr   (SUBR)
        clear   (SUBR)
        cond   (FSUBR)
        cons   (SUBR)
        consp   (SUBR)
        copy   (SUBR)
        defun   (FSUBR)
        do   (FSUBR)
        doublep   (SUBR)
        dtpr   (SUBR)
        eq   (SUBR)
        equal   (SUBR)
        eval   (SUBR)
        exprs   (SUBR)
        fexprs   (SUBR)
        floatp   (SUBR)
        fsubrs   (SUBR)
        gc   (SUBR)
        gensym   (SUBR)
        getprop   (SUBR)
        if   (FSUBR)
        integerp   (SUBR)
        last   (SUBR)
        length   (SUBR)
        list   (FSUBR)
        listp   (SUBR)
        mapc   (FEXPR)
        mapcar   (FEXPR)
        member   (SUBR)
        memq   (SUBR)
        nconc   (SUBR)
        neq   (SUBR)
        nequal   (SUBR)
        nospy   (SUBR)
        not   (SUBR)
        notrace   (SUBR)
        nth   (SUBR)
        null   (SUBR)
        numberp   (SUBR)
        oblist   (SUBR)
        or   (FEXPR)
        pp   (SUBR)
        princ   (SUBR)
        print   (SUBR)
        progn   (FSUBR)
        putprop   (SUBR)
        quote   (FSUBR)
        read   (FSUBR)
        remprop   (SUBR)
        repeat   (FSUBR)
        reverse   (SUBR)
        rplaca   (SUBR)
        rplacd   (SUBR)
```

```
        send   (FSUBR)
        setq   (FSUBR)
        spy   (SUBR)
        stringp   (SUBR)
        subrs   (SUBR)
        symbolp   (SUBR)
        terpri   (SUBR)
        trace   (SUBR)
        while   (FSUBR)
        ¥¥   (FSUBR)
        ~=   (SUBR)
        ~~   (SUBR)
```

SUBRシステム関数一覧

```
        <
        <=
        =
        ==
        >
        >=
        append
        atom
        car
        cdr
        clear
        cons
        consp
        doublep
        dtpr
        eq
        equal
        eval
        exprs
        fexprs
        floatp
        fsubrs
        gc
        gensym
        getprop
        integerp
        last
        length
        listp
        member
        memq
        nconc
        neq
        nequal
        nospy
        not
        notrace
        nth
        null
        numberp
        oblist
```

pp
princ
print
putprop
remprop
reverse
rplaca
rplacd
spy
stringp
subrs
symbolp
terpri
trace
~=
~~

FSUBRシステム関数一覧

*
+
-
/
//
cond
defun
do
if
list
progn
quote
read
repeat
send
setq
while
¥¥

EXPRシステム関数一覧

++
--
assoc
copy
mapc
mapcar

FEXPRシステム関数一覧

and
or