I-Scheme Interpreter User Manuarl

目次

1 Introduction

I-Scheme Interpreter は手軽に関数型プログラミング言語の実験を行なえるように作成したものである．基本的には，

Abelson, Sussman, "*Structure and Interpretation of Computer Programs*", MIT Press.

のプログラムが実行できることを目指してプログラミング言語 Scheme を基本にして設計されている．但し，プリミティブの名前は Common Lisp に準拠している．また，Scheme の仕様を完全に満たしているわけではない．主な問題点は次のとおりである．

1. let, let$*$, and, or など，幾つかの関数で，末尾再帰性 (Tail Recursion) がサポートされていない．
2. コンティニュエーションがインプリメントされていない
3. force, delay がインプリメントされていない．ただし，マクロ定義機能を用いて実現することが可能である．マクロによる force, delay のもっとも簡単な定義は次のようなものである．

```
(defmacro (delay s) `(lambda () ,s))
(defmacro (force s) `(,s))
```

4. 配列がインプリメントされていない．

インプリメントにあたっては Will o'Lisp [1] を，環境構成法やエバリュエータを書き換えるなど全面的に改造して利用した．

2 Start up of Interpreter

I-Scheme は起動すると最初にカレントディレクトリから "scheme.cnf" を探し，見つかればその内容に従って interpreter のメモリー空間の大きさを決定する．"scheme.cnf" の一例を示す．

```
CELLSIZ = #x3000
ATOMSIZ = #x1000
STRSIZ = #x1000
NUMSIZ = #x1000
STACKSIZ = #x1000
BIGSIZ = #x1000
LINESIZ = 256
```

上の指定ではそれぞれ，cell 領域，atom 領域，string 領域，number 領域，stack 領域，bignum 領域の大きさ，及び I-scheme がファイルや標準入力からの入力を一旦貯えるバッファの大きさを指定している．`#x` は後に続く数字が 16 進数である事を示している．指定がなければ 10 進数として扱われる．領域の大きさの限度は I-Scheme が動作している計算機の C 言語が *unsigned int* で表現できる数字の上限である．

こらの指定が省略された場合はデフォルトの値がそれぞれの領域の大きさとして用いられる．

メモリー空間の大きさが決定され初期化が行われた後，I-Scheme はカレントディレクトリから "initial.sch" を読み込み，評価する．

3 Command Loop

I-Scheme は起動すると最初にライブラリー・ロードパスから 'Minitial.sch" を検索してロードしようとする．それから I-Scheme はプロンプト "%" を画面に表示する．これは I-Scheme が S 式がタイプされるのを待っていることを示している．

完全な S 式 が入力されると I-Scheme はその式を評価しようとする．もし式の評価が成功したら I-Scheme は評価結果をプリントし，別の式がタイプされるのを待つためにプロンプトを表示する．

[1] ASCII disk album No.15

4 Data Type

I-Scheme ではプログラマが利用できるいくつかの異なったデータタイプがある.

- リスト (list) 任意のデータタイプを子に持つ 2 分木
- シンボル (symbol) アトム (atom) とも呼ばれる. 内部化 (intern) されている
- 文字列 (string) 内部化されていないアトムの印字名
- 整数 (integer) I-Scheme 動作している計算機の C 言語の long int で表現される数
- 多倍長精度整数 (Big number)
- 実数 (float) I-Scheme が動作している計算機の C 言語の double で表現される数

5 Evaluator

エバリュエータは S 式を種類に応じて処理し, 新たな S 式を作り出す. これを S 式の評価 (eval) という. その評価のプロセスは S 式の種類に応じて次の様になる.

- 整数, 多倍長精度整数, 実数, 文字列は評価すると自分自身をかえす.
- シンボルは現在の束縛で対応付けられている値をかえす.
- リストはまずリストの最初の要素を調べる.
  - もし最初の要素が `lambda` または `nlambda` であれば関数閉包 (car が lambda 式で cdr が環境リストであるリスト) を作り, `procedure` を関数閉包に cons したものを値としてかえす.
  - そうでなければリストの最初の要素を評価する.
    - もし評価した値が引き数を評価する関数であればリストの残りの要素が評価される.
    - そうでなければリストの残りの要素を (評価せずに) 引き数とする.
    - apply がよばれる.

6 Lexical Convention

I-Scheme プログラムを入力する際は, 次の書式に従う必要がある.

- コメントはセミコロン ";" で始まり, 行末まで続く.
- シンボル名は次のキャラクタ以外の任意の空白でない列で構成される.

  ( ) ' ` , " ;

- 整数は数字の列から構成され, 先頭に + または − がつくことがある. 整数の取りうる範囲は I-Scheme が走っている計算機上の C 言語の long 型によって表現できる範囲である.
- 多倍長精度整数は数字の列から構成され, 先頭に + または − がつくことがある. C 言語の long 型の精度を越える整数は自動的に多倍長精度整数として扱われる.
- 実数は数字の列から構成され, 先頭に + または − がつくことがあり, 小数点が間に入る. 実数の取りうる範囲は I-Scheme が走っている計算機上の C 言語の double 型が表現できる範囲である.
- 文字列はダブルクォート " で囲まれた文字の列である. クォートされた文字列の中では印字できない文字が含まれることが許される.

  \\ は文字 '\' を意味する

  | 2 つの '|' で囲まれた特殊文字は英字としての扱いを受ける.

- 次のリードマクロが用意されている．

'<*expr*>	==	(quote <*expr*>)
#x<*hex*>	==	16 進数
#\ <*char*>	==	文字のアスキーコード
#:<*symbol*>	==	オブリスト (oblist) に登録されないアトム
`<*expr*>	==	(backquote <*expr*>)
,<*expr*>	==	(comma <*expr*>)
,@<*expr*>	==	(comma-at <*expr*>)

7 Special Atoms

i-scheme ではシステムが既に定義しているシンボルアトムがあり，それらは予約語として扱われ，トップレベルで再定義することはできない．それは，システムによって提供される関数を示すシンボルアトムと，その他のアトムである．関数を示すアトムは次章で説明されるので，ここではその他のアトムに付いて説明する．

t　条件判定，cond 節などでの真の値を示す

nil　空のリスト () であり，条件判定での偽も示す

EOF　ファイルの終了を示す

$c1, $c2, $c3　これまでに入力した式を新しい順に記憶しているアトム

$d1, $d2, $d3　これまでの計算結果を新しい順に記憶しているアトム

%　標準のプロンプト

procedure　関数閉包を示すために内部で使われるアトム

8 Summary of Functions

以下の関数の説明では (function-name <*arg1*> [*arg2*]) の形で関数を示している．ここで function-name が関数名，< > で囲まれた引数 *arg1* は省略できない引数，[ ] で囲まれた *arg2* は省略可能な引数を示している．また，引数等の繰り返しを "..." によって示すことにする．

8.1 Evaluation functions

(eval <*expr*> [*env*])　式を評価する

expr　評価されるクオートされた式

env　*expr* を評価する環境

戻り値　式を評価した結果

(apply <*fun*> <*args*> [*env*])　引き数のリストに関数を適用する

fun　クオートされた，適用すべき関数 (または関数シンボル)

args　引き数リスト

env　引数を評価する環境

戻り値　関数を引き数に適用した結果

8.2 Symbol functions

(set <*sym*> <*expr*>)　シンボルの値をセットする

sym　値をセットするシンボル

expr　新しい値

戻り値　セットされた値

(setq [<*sym*> <*expr*>] ···)　シンボルの値をセットする

sym　値をセットする (引用された) シンボル

expr　新しい値

戻り値　セットされた値

(psetq [<*sym*> <*expr*>] ···)

setq と同様に *expr* の値を *sym* にセットするが，その操作に先立ってすべての *expr* の値が評価される．

(lambda (<*fargs*>) <*expr*> ···)　関数閉包を返す

(nlambda (<*fargs*>) <*fexpr*> ···)　fexpr 型の関数閉包を返す

fargs　(クォートされた) 仮引き数のリスト

expr　関数の本体を構成する式

戻り値　関数閉包

(define <*sym*> <*expr*>)　変数を定義する

sym　定義される (クォートされた) シンボル

expr　評価される式

戻り値　定義されたシンボル

(define (<*sym*> [*fargs*] ··· [. <*rarg*>]) <*expr*> ···)　関数を定義する

sym　定義される (クォートされた) シンボル

farg　(クォートされた) 仮引き数．

rarg　(クォートされた) 仮引き数．*farg* と対応付けた残りの引数のリストと一致する．

expr　関数 (マクロ) の本体を構成する式

戻り値　関数のシンボル

(df (<*sym*> [*farg*] ···)　<*fexpr*> ···) fexpr 型の関数を定義する

(defmacro (<*sym*> [*farg*] ···) <*expr*> ···)　マクロを定義する

sym　定義される (クォートされた) シンボル

farg　(クォートされた) 仮引き数

expr　関数 (マクロ) の本体を構成する式

戻り値　関数 (マクロ) のシンボル

(getd <*atm*>)　シンボルアトム *atm* から関数定義をとりだす

(getdef <*atm*>)　シンボルアトム *atm* から，関数 read で読み込める形で関数定義をとりだす

戻り値　関数定義

(gensym)　印字名 g000, g001, ··· のオブリストに登録されていないシンボルを生成する

戻り値　新しいシンボル

(intern <*atm1*> ···)　オブリストに登録されていないシンボルアトム *atm1* ··· をオブリストに登録する

戻り値　シンボルアトム

(oblist)　オブリストのコピーを作ってかえす

戻り値　オブリストのコピー

(user)　ユーザーが定義した関数，マクロの名前を返す

(remob <*atm1*> ···)　オブリストに登録されているアトム *atm1* ··· をオブリストから削除する

8.3 Propaty list functions

(get <*sym*> <*prop*>)　属性値を得る

sym　シンボル

prop　属性名のシンボル

戻り値　対応する属性があれば属性値を，無ければ nil を返す

(putprop [<*sym*> <*prop*> <*val*>] ···)　属性リストに属性を登録する．

sym　シンボル

prop　属性名のシンボル

val　属性値

戻り値　登録された属性値

(remprop <*sym*> <*prop*>)　属性を取り除く

sym　シンボル

prop　属性名のシンボル

戻り値　nil

8.4 List functions

(car <*expr*>)　リストの先頭の要素をかえす

expr　リスト

戻り値　リストの先頭の要素

(cdr <*expr*>)　リストから先頭の要素を除いたリストをかえす

expr　リスト

戻り値　リストから先頭の要素を除いた残りのリストをかえす

(cxxr <*expr*>)　すべての cxxr の組み合わせ

(cxxxr <*expr*>)　すべての cxxxr の組み合わせ

(cxxxxr <*expr*>)　すべての cxxxxr の組み合わせ

(first <*expr*>)　リストの最初の要素を返す

(second <*expr*>)　リストの 2 番目の要素を返す

(third <*expr*>)　リストの 3 番目の要素を返す

(fourth <*expr*>)　リストの 4 番目の要素を返す

(nth <*num*> <*list*>)　リストの *num* 番目の要素を返す．リストの先頭の要素を 1 とする

(nthcdr <*num*> <*list*>)　リストの *num* 番目の cdr を返す．リスト全体は 1 とする

(tailn <*num*> <*expr*>)　nthcdr の別名

(last <*expr*>)　リストの最後の要素を返す

(cons <*expr1*> <*expr2*>)　*expr1* と *expr2* を結合した新しいリストを構成する

expr1　新しいリストの先頭の要素

expr2　新しいリストの残りの要素

戻り値　新しいリスト

(list <*expr*> ···)　引き数のリストを作成する

expr　リストに結合される式

戻り値　新しいリスト

(append <*expr*> ···)　リストを連接する

expr　連接するリスト

戻り値　新しいリスト

(reverse <*expr*>)　リストを逆転する

expr　逆転するリスト

戻り値　逆転された新しいリスト

(member <*expr*> <*list*>)　リストから式を見つける

expr　見つけたい式

list　探索するリスト

戻り値　　見つかった場合　　*expr* を先頭とする *list* の部分リスト
　　見つからない場合　nil

(assoc <*expr*> <*alist*>)　連想リスト中に式を見つける

expr　見つけたい式

alist　連想リスト

戻り値　連想リストの対応するエントリーまたは nil

(assoc1 <*expr*> <*alist*>)　連想リスト中に equal を用いて式を見つける

expr　見つけたい式

alist　連想リスト

戻り値　連想リストの対応するエントリーまたは nil

(assoc2 <*expr*> <*alist*>)　連想リストの値として *expr* を持つ式を equal を用いて見つける

expr　見つけたい値

alist　連想リスト

戻り値　連想リストの対応するエントリーまたは nil

(length <*expr*>)　リストまたは文字列の長さをかえす

expr　リストまたは文字列

戻り値　リストまたは文字列の長さ

(firstn <*num*> <*expr*>)　リストの最初から *expr* 個の要素を取り出して作ったリストを返す

(flatten <*expr*>)　リストを要素とするリストを引数としてうけとり，全ての要素を append で結合した結果のリストを返す

(change-nth *num expr list*)　*list* の先頭から *num* 番目の要素を *expr* で入れ換えたリストを返す

(remove <*expr*> <*list*>)　*list* から *expr* と equal である要素を全て取り除いたリストを返す

(remove1 <*expr*> <*list*>)　*list* から *expr* と equal である最初の要素を取り除いたリストを返す

(mapcar <*fnc*> <*list*>)　1 引き数関数 *fnc* をリスト *list* の各要素に順に適用し，その結果を cons でつないだものをかえす

fnc　1 引き数関数

list　リスト

戻り値　関数の戻り値のリスト

(mapcan <*fnc*> <*list*>)　1 引き数関数 *fnc* をリスト *list* の各要素に順に適用し，その結果を nconc でつないだものをかえす

(mapcon <*fnc*> <*list*>)　1 引き数関数 *fnc* をリスト *list* を先頭から順に cdr を取ったものに適用し，その結果を nconc でつないだものをかえす

(map <*fnc*> <*list*> ···)　任意個の引数をとる関数 *fnc* にリスト *list* ··· の要素を順に適用し，その結果を cons でつないだものを返す

func　関数

list ···　*func* に与えられる引数

戻り値　*list* ··· として (a1, a2, a3) (b1, b2, b3) が与えられたとすると，(list (func a1 b1) (func a2 b2) (func a3 b3)) を返す

(mix <*fnc*> <*list*>)　リストを要素とするリスト *list* を，関数 *fnc* に適用し，その結果を cons でつないだものを返す．適用の方法は map と同様である

(sort <*fnc*> <*list*>)　2 引き数の比較関数 *fnc* をリスト *list* を先頭から順に適用し，リストの連続する要素間で *fnc* が満足される様に並べ変えられたリストを返す

8.5 Destructive list functions

(rplaca <*list*> <*expr*>)　リストの先頭を置き換える

list　先頭が置き換えられるリスト

expr　リストの先頭に置かれる値

戻り値　置き換えが行なわれた結果のリスト

(rplacd <*list*> <*expr*>)　リストの残りを置き換える

list　置き換えられるリスト

expr　リストの cdr に置かれる値

戻り値　置き換えが行なわれた結果のリスト

(nconc <*list*> ···)　リストを破壊的に結合する

list　結合されるリスト

戻り値　結合されたリスト

8.6 Predicate functions

(atom <*expr*>)　*expr* がアトムなら t, それ以外なら nil をかえす

(symbolp <*expr*>)　*expr* がシンボルなら t, それ以外なら nil をかえす

(numberp <*expr*>)　*expr* が数なら t, それ以外なら nil をかえす

(integerp <*expr*>)　*expr* が整数なら t, それ以外なら nil をかえす

(bignump <*expr*>)　*expr* が多倍長精度整数なら t, それ以外なら nil をかえす

(floatp <*expr*>)　*expr* が実数なら t, それ以外なら nil をかえす

(stringp <*expr*>)　*expr* が文字列なら t, それ以外なら nil をかえす

(functionp <*expr*>)　*expr* が関数なら t, それ以外なら nil をかえす

(null <*expr*>)　*expr* が空リストなら t, それ以外なら nil をかえす

(not <*expr*>)　*expr* が nil なら t, それ以外なら nil をかえす

(listp <*expr*>)　*expr* がリストなら t, それ以外なら nil をかえす

(consp <*expr*>)　*expr* が空でないリストなら t, それ以外なら nil をかえす

(zerop <*expr*>)　*expr* が 0 なら t, それ以外なら nil をかえす

(plusp <*expr*>)　*expr* が正なら t, それ以外なら nil をかえす

(minusp <*expr*>)　*expr* が負なら t, それ以外なら nil をかえす

(oddp <*expr*>)　*expr* が奇数なら t, それ以外なら nil をかえす

(evenp <*expr*>)　*expr* が偶数なら t, それ以外なら nil をかえす

(eq <*expr1*> <*expr2*>)　式は同一のものか?

expr1　第一の式

expr2　第二の式

戻り値　値が同じものを示しているなら t, それ以外なら nil をかえす

(eql <*expr1*> <*expr2*>)　式は同一のものか (数と文字列に対しては値を比較する)?

expr1　第一の式

expr2　第二の式

戻り値　値が同じものを示しているなら t, それ以外なら nil をかえす

(equal <*expr1*> <*expr2*>)　式は等価か？

expr1　第一の式

expr2　第二の式

戻り値　値が等価なら t, それ以外なら nil をかえす

(equalp <*expr1*> <*expr2*>)　式は等価か？ (文字列の大文字小文字を無視する)

expr1　第一の式

expr2　第二の式

戻り値　等価なら t, それ以外なら nil をかえす

8.7 Control functions

(cond (<*ppair*> ···))　条件的に評価する

pair　(*pred* [*expr*] ···) から構成される対．ここで

pred　は述語表現

expr　は *pred* が nil でなければ評価される

戻り値　*pred* が nil でない最初の *expr* の値

(and <*expr*> ···)　式のリストの論理積

expr　論理積を取るべき式

戻り値　式を順に評価して nil であれば nil, そうでなければ最後に評価した式の値 (評価して nil となった最初の式の後で評価は止る)

(or <*expr*> ···)　式のリストの論理和

expr　論理和を取るべき式

戻り値　すべての式を評価して nil であれば nil, そうでなければ最初の nil でない式の値 (評価して nil 以外の値をかえす最初の式の後で評価は止る)

(if <*texpr*> <*expr1*> [*expr2*])　条件的に式を評価する

texpr　条件式

expr1　*texpr* が nil でない場合に評価される式

expr2　*texpr* が nil の場合に評価される式，*expr2* の指定がない場合は常に nil をかえす

戻り値　選択された式の値

(when <*texpr*> [*expr*] ···)　条件的に式を評価する

texpr　条件式

expr　*texpr* が nil でない場合に評価される式

戻り値　選択された式の値

(unless <*texpr*> [*expr*] ···)　条件的に式を評価する

texpr　条件式

expr　*texpr* が nil の場合に評価される式

戻り値　選択された式の値

(let ([*binding*] ···) <*expr*> ···)　局所的な束縛を作る
(let* ([*binding*] ··· <*expr*> ···)　逐次的に局所的な束縛を作る

binding　局所的な束縛．その各々は
1. シンボル (nil に初期化される)
2. car がシンボルで cadr が初期化される式

expr　束縛が作られた後で評価される式

戻り値　最後の式の値

8.8 Prog formats

(progn [*expr*] ···)　式を順番に評価する

expr　評価する式

戻り値　最後の式の値 (または nil)

8.9 Debug functions

(trace <*atm*> ···)　引き数として与えられたシンボルアトムが関数として呼ばれる時にトレースされるよう，シンボルアトムに印を付ける

(untrace [*atm*] ···)　引き数として与えられたシンボルアトムが，これ以後関数として呼ばれる時にトレースされないようにシンボルアトムから印をはずす．引数が省略された場合，全てのシンボルアトムから印をはずす．

(error <*string*>) *string* を名称とするエラーを発生する

8.10 Arithmetic functions

(+ <*expr*> ···)　数のリストを加える

(− <*expr*> ···)　数のリストを引き算する，または 1 つの数の符号を変える

(∗ <*expr*> ···)　数のリストを掛る

(sqr <*expr*>)　数の 2 乗を返す

(/ <*expr*> ···)　数のリストを割る

(1+ <*expr*>)　数に 1 を加える

(1− <*expr*>)　数から 1 を引く

(rem <*expr*> ···)　数のリストの余りを求める

(divide <*expr*> ···)　割り算を行ない，その商と余りをリストとしたものをかえす

(sin <*expr*>)　数の正弦を計算する

(cos <*expr*>) 数の余弦を計算する

(tan <*expr*>) 数の正接を計算する

(atan2 <*expr1*> <*expr2*>) *expr1* / *expr2* の逆正接を計算する

(expt <*x-expr*> <*y-expr*>) x の y 乗を計算する

(exp <*x-exp*>) e の x 乗を計算する

(sqrt <*expr*>) 平方根を求める

(log <*expr*>) 自然対数を求める

(log10 <*expr*>) 10 を底とする対数を求める

(abs <*expr*>) 絶対値を求める

(pi) 円周率を返す

(signum <*expr*>) 数値が正ならば 1 を，負ならば -1 を返す

(asin <*expr*>) 数の逆正弦を計算する

(acos <*expr*>) 数の逆余弦を計算する

(sinh <*expr*>) 数の双曲線正弦を計算する

(cosh <*expr*>) 数の双曲線余弦を計算する

(tanh <*expr*>) 数の双曲線正接を計算する

(ceil <*expr*>) 数を切り上げて整数にする

(floor <*expr*>) 数を切り下げて整数にする

(round <*expr*>) 数を四捨五入して整数にする

(random <*expr*>) 乱数を返す

(float <*expr*>) 実数に変換する

(max <*expr*> ···) 引数の中で最大のものを返す

(min <*expr*> ···) 引数の中で最小のものを返す

8.10.1 Bit manipulation functions

固定長精度の整数を bit 列とみなして，論理和，論理積などの bit 演算を行う関数を提供している．

(logior <*num*> ···) 引数を順に論理和で結合した結果を返す

(logxor <*num*> ···) 引数を順に排他的論理和で結合した結果を返す

(logior <*num*> ···) 引数を順に論理積で結合した結果を返す

(lognot <*num*>) 引数を bit 毎に反転した結果を返す

(logbitp <*index*> <*num*>) *num* の *index* 番目の bit が 1 の場合 t を， 0 の場合 nil を返す

8.11 Relational functions

関係演算関数は整数・実数・多倍長精度整数の比較に用いられる

(< <*expr1*> <*expr2*>) 小さいか

expr1 右の式

expr2 左の式

戻り値 比較の結果が真ならば t, 偽ならば nil をかえす

(= <*expr1*> <*expr2*>) 等しいか

expr1 右の式

expr2 左の式

戻り値 比較の結果が真ならば t, 偽ならば nil をかえす

(> <*expr1*> <*expr2*>) 大きいか

expr1 右の式

expr2 左の式

戻り値 比較の結果が真ならば t, 偽ならば nil をかえす

(<= <*expr1*> <*expr2*>) 以下か

expr1 右の式

expr2 左の式

戻り値 比較の結果が真ならば t, 偽ならば nil をかえす

(>= <*expr1*> <*expr2*>) 以上か

expr1 右の式

expr2 左の式

戻り値 比較の結果が真ならば t, 偽ならば nil をかえす

(/ = <*expr1*> <*expr2*>) 等しくないか

expr1 右の式

expr2 左の式

戻り値 比較の結果が真ならば t, 偽ならば nil をかえす

8.12 String functions

(ascii <*expr*>) expr がシンボルアトムなら，その印字名の先頭文字の文字コードを， *expr* が数値ならその数値にあたる文字コードのの文字を印字名とするシンボルアトムをかえす

(alen <*string*>) シンボルアトムの印字名の長さをかえす

string シンボルアトム

戻り値 印字名の長さ

(implode <*list*>) *list* はシンボルアトムを要素とするリスト． *list* の各要素の印字名をつなぎ合わせたシンボルアトムをかえす

(explode <*atm*>) シンボルアトム *atm* の印字名を 1 文字ごとに分解しそれをリストにしてかえす

(itoa <*expr*>) 整数を文字列に変換する．

8.13 I/O functions

入出力関数のほとんどはファイルディスクリプタを引き数に持つことができる．ファイルディスクリプタは 0 以上の整数で，オープンされたファイルにつけられた固有の番号である．ファイルディスクリプタを入出力関数に渡すことで，入出力をそのファイルから行なうことができる．0 から 2 までのファイルディスクリプタは予め

0: 標準入力

1: 標準出力

2: 標準エラー出力

と割り当てられている．

(read [<*fd-num*>]) ファイルディスクリプタ *fd-num* が指定されればそのファイルから， *fd-num* が省略されれば現在の入力先から S 式を読み込む．

戻り値 読み込んだ S 式

(print <*expr*> [*fd*]) *expr* を改行付きで出力する

expr S 式

fd ファイルディスクリプタ (デフォルトは標準出力)

戻り値 *expr*

(prin1 <*expr*> [*fd*]) *expr* を改行を付けずに出力する

expr S 式

fd ファイルディスクリプタ (デフォルトは標準出力)

戻り値 *expr*

(princ <*expr*> [*fd*]) *expr* をクォートせず，改行を付けずに出力する

expr S 式

fd ファイルディスクリプタ (デフォルトは標準出力)

戻り値 *expr*

(terpri [*fd*]) 改行のみを行なう

fd ファイルディスクリプタ (デフォルトは標準出力)

戻り値 nil

(format <*fd*> <*ctrl-atm*> <*expr*> ···) 書式指定付き出力を行なう

fd ファイルディスクリプタ (nil は標準出力)

ctrl-atm 出力の書式を指定するアトム．この印字名の中に ~ で始まり次の表に示すいくつかの変換文字で終わる変換指定を含めることで，その後の引き数 *expr* ··· を変換指定に対応した形で展開することができる．a オプションでシンボルアトムを出力する場合，d,o,x,e,f,g オプションで数値を出力する場合は，その出力にフィールド指定を行なうことができる．

1. チルダと変換文字の間に 10 進数 n をはさむことで， n 桁分確保される
2. n の前に 0 を置くと余分な空白の代わりに 0 が展開される
3. 不動小数点数の出力の場合は n の後ろに “,m” をつけて，小数点以下の桁数を指定することができる

変換文字の直前に : を置くことにより，その前の変換に使った引き数を再度使用することができる．第 1 引き数以前にこのオプションが使われた場合は， nil が引き数になる．

a	対応する引き数を princ 出力する．引き数は任意の S 式
s	対応する引き数を prin1 出力する．引き数は任意の S 式
c	引き数は整数．引き数に相当する文字コードを持つ文字が展開される
d	引き数は整数．引き数を 10 進数 で出力する
o	引き数は整数．引き数を 8 進数 で出力する
x	引き数は整数．引き数を 16 進数 で出力する
e	引き数は実数．引き数を [−]n.nnnnnnE[−]mm の形に変換 小数点以下はデフォルトでは 6 桁
f	引き数は実数．引き数を [−]n.nnnnnn の形に変換 小数点以下はデフォルトでは 6 桁
g	引き数は実数．引き数を e, f オプションのうち印字の短いほうに変換する 小数点以下はデフォルトでは 6 桁
p	引き数は数値．引き数が整数の 1 に等しければ何も出力されず， そうでなければ s が出力される
@p	引き数は数値．引き数が整数の 1 に等しければ y が出力され， そうでなければ s が出力される
n	改行コード (cr+lf) が出力される
t	タブが出力される
r	cr のみが出力される
b	バックスペースが出力される

(str-format <*ctrl-atm*> <*expr*> ···)　書式指定付に従って文字列を作成する． *ctrl-atm* で利用できる書式は format の表のうち a から g までである．また，書式 a および s で出力できるのはシンボルアトムのみである．

(flatsiz <*expr*>)　prin1 で *expr* を印字するために必要な文字数

expr　S 式

戻り値　文字数

(prompt <*atm*>)　プロンプトをシンボルアトム *atm* の印字名に切り替える． *atm* が t の時，プロンプトは “%” になる．

(pprint <*expr*> [*len*])　*expr* を適当に字下げして出力する

expr　S 式

len　一行の最大文字数の指定 (デフォルトは 45 字)

戻り値　*expr*

8.14 File I/O functions

(open <*fname*> [*mode*])　指定されたモードでファイルをオープンする

fname　ファイル名

mode　モード (デフォルトは r)

mode	モード
r	リードテキスト (デフォルト)
w	ライトテキスト
a	アペンドテキスト
rb	リードバイナリ
wb	ライトバイナリ
ab	アペンドバイナリ

戻り値　ファイルディスクリプタ

(close <*fd*>)　ファイルを閉じる

fd　ファイルディスクリプタ

戻り値　nil

(fmode <*fd*>)　ファイルディスクリプタで示されるファイルのモードをかえす

fd　ファイルディスクリプタ

戻り値　ファイルのモード

(readch [*source*])　1 文字入力

source　ファイルディスクリプタ (デフォルトは標準入力)

戻り値　読み込んだ文字を印字名に持つシンボルアトム

(dirin <*fd*>)　入力先を *fd* にリダイレクトする

fd　ファイルディスクリプタ

(dirout <*fd*>)　出力先を *fd* にリダイレクトする

fd　ファイルディスクリプタ

(curin)　現在の入力先のファイルディスクリプタをかえす

戻り値　ファイルディスクリプタ

(curout)　現在の出力先のファイルディスクリプタをかえす

戻り値　ファイルディスクリプタ

8.15 System functions

(load <*fname*> ···)　指定したすべてのファイルから S 式を読み込み評価する．ファイルはコンパイル時に指定されたライブラリ・ロードパス中のディレクトリから順に検索される

fname　ファイル名

戻り値　t

(reclaim)　ガーベジコレクタを強制起動する

(verbos [*expr*])　ガーベジコレクタの起動時のメッセージの on/off を行なう

expr　*expr* が nil ならメッセージは表示せず，nil 以外ならメッセージを表示する

(quit)　I-Scheme を終了する

8.16 Unix functions

ここで説明される関数は UNIX の下でのみ提供されている.

(time <*expr*> [*env*]) *expr* を評価するのに要した，ユーザー時間，システム時間，ユーザー時間のうちでガーベジコレクトに費された時間を出力する.

expr 評価されるクオートされた式

env *expr* を評価する環境

戻り値 *expr* を評価した結果の値

(system <*atom*>) シンボルアトムで与えられたコマンドを実行する.

atom コマンド名

戻り値 t

(unix-pipe <*expr*> <*list*> [*num*]) シンボルアトムで与えられたコマンドを実行し，その結果をリストとして読み込む.

expr コマンド名のシンボルアトム．または，コマンド名とコマンドに渡す引数のシンボルアトムからなるリスト．実行されるコマンドは，標準入力からデータを読み込み標準出力に計算結果を出力するフィルターとして動作するものでなければならない.

list コマンドに対する入力．リストの要素を順に，コマンドの標準入力に出力する.

num リストの要素を出力する際， *num* 個ずつ改行で区切って出力する．デフォルトは 1 .

戻り値 コマンドの出力を read し，リストに結合して返す.

(unix-pipe-f <*expr*> <*func*>) シンボルアトムで与えられたコマンドを実行し，その結果をリストとして読み込む.

expr コマンド名のシンボルアトム．または，コマンド名とコマンドに渡す引数のシンボルアトムからなるリスト．実行されるコマンドは，標準入力からデータを読み込み標準出力に計算結果を出力するフィルターとして動作するものでなければならない.

func コマンドに対する入力を構成するための関数．標準出力へ出力を行う関数を渡す.

戻り値 コマンドの出力を read し，リストに結合して返す.

8.17 Graphics functions

I-Scheme interpreter では， X11R3 以上の Xlib の関数を用いて簡単な図形を描くための関数を用意している．まだまだこれらの関数は製作途中にあり，利用に当たっては注意が必要である.

関数 (initgraph) によって描画用の window が準備され， (closegraph) によって close されるまで，window に直線，長方形，円，文字列などを描画することができる.

(initgraph [*width* [[*height*] [*color*]]]) graphics 用の window を表示し graphics の初期化を行う．window の大きさを引数 *width, height* によって指定することができる． *color*が 0 以外の整数であれば，カラーモードが選択される.

(closegraph) X11 server の接続を切り， graphics 利用を終了する

(cls) window を clear する

(line <*x1*> <*y1*> <*x2*> <*y2*>) (*x1, y1*) から (*x1, y2*) への直線を引く

(circle <*x*> <*y*> <*r*>) (*x, y*) を中心として半径 *r* の円を描く.

(point <*x*> <*y*>)　$(x,\ y)$ に点を描く．

(rectangle <*x*> <*y*> <*width*> <*hegiht*>)　$(x,\ y)$ を左肩として 幅 *width* 高さ *height* の長方形を描く．

(normal-mode)　描画方式を normal mode にする．

(xor-mode)　図形を xor で描画する．

(reverse-mode)　図形を白黒反転して描画する．

(paint-pattern <*number*>)　図形の塗りつぶしパターンを *number* のパターンにする．現在 0 (黒ベタ) から 11 までの指定が可能である．カラーモードの場合は，塗り潰しの色が指定される．

(fill-rectangle <*x*> <*y*> <*width*> <*hegiht*>)　$(x,\ y)$ を左肩として 幅 *width* 高さ *height* の長方形を塗りつぶす．

(fill-circle <*x*> <*y*> <*r*>)　$(x,\ y)$ を中心として半径 r の円を塗りつぶす．

(button-press)　マウスのボタンが押された座標と押されたボタンのリストを返す．

(query-pointer)　マウスポインターの座標と，ボタンと修飾キーの状態のリストを返す．ボタンと修飾キーの状態については X11 のマニュアルを参照すること．

(drawstring <*x*> <*y*> <*string*>)　$(x,\ y)$ をベースラインの原点として文字列 *string* を表示する．

(set-font <*number*>)　drawstring で用いる font を設定する．現在 75dpi の times の 10, 12, 14, 18, 24 pt を，それぞれ 0 から 5 の数字によって指定することができる．

A Function Name List

*
+
−
/
/ =
1+
1−
<
<=
=
>
>=
abs
acos
alen
and
append
apply
ascii
asin
assoc
assoc1
assoc2
atan2
atom
bignump
button-press
caaaar
caaadr
caaar
caadar
caaddr
caadr
caar
cadaar
cadadr
cadar
caddar
cadddr
caddr
cadr
car
catch
catcherror
cdaaar
cdaadr
cdaar
cdadar
cdaddr
cdadr
cdar
cddaar
cddadr
cddar
cdddar
cddddr
cdddr
cddr
cdr
ceil
change-nth
circle
close
closegraph
cls
cond
cons
consp
cos
cosh
curin
curout
define
defmacro
df
dirin
dirout
divide
drawstring
eq
eql
equal
equalp
error
eval
evenp
exp
explode
expt
fill-circle
fill-rectangle
first
firstn
flatsiz
flatten
float
floatp
floor
fmode
format
fourth
functionp
gensym
get
getd
getdef
go
if
implode
initgraph
integerp
intern
itoa
last
length
let
let*
line
list
listp
load
log
log10
logand
logbitp
logior
lognot
logxor
map
mapcan
mapcar
mapcon
max
member
min
minusp
mix
nconc
normal-mode
not
nth
nthcdr
null
numberp
oblist
oddp
open
or
paint-pattern
pi
plusp
point
pprint
prin1
princ
print
prog
progn
prompt
psetq
putd
putprop
query-pointer
quit
random
read
readch
reclaim
rectangle
rem
remob
remove
remove1
remprop
return
reverse
reverse-mode
round
rplaca
rplacd
second
set
set-font
setq
signum
sin
sinh
sort
sqr
sqrt
str-format
stringp
symbolp
system
tailn
tan
tanh
terpri
third
throw
time
trace
unix-pipe
unix-pipe-f
unless
untrace
user
verbos
when
xor-mode
zerop