

I/O Functions in LISP

The following represents a sampling of the available I/O functions in LISP. For a complete list see the XLISP Reference Manual. If the file pointers outf or inf are omitted, the system defaults to CRT.

(setf outf (open "MSDOS fname" :direction :output))	Opens a disk file for output. Stores the file pointer in variable outf for further use.
(print Sex outf)	Returns the value of the Sex and sends a copy to outf.
(dribble "MSDOS fname")	Echoes all CRT I/O to the disk file
(prin1 Sex outf)	Same as (print Sex outf) except for the trailing cr/lf
(terpri outf)	Issues a cr/lf
(read inf)	Inputs the next Sex from inf (or CRT if inf omitted)
(raton1 inf)	Same as (read inf) but inputs the next atom from inf.
(readch inf)	Read 1 character from inf
(close outf)	Close file outf (or inf)

PROG

The PROG construct allows for an indefinite number of forms to be evaluated with local variables. In modern LISP it has been replaced by LET and LOOP constructs, but it is still useful.

```
(prog (var1 var2 ... varn) <forms>)
```

1. The number of forms in prog is indeterminate.
2. Prog initializes all its local variables to nil (var1 var2 ... varn).
3. <Forms> are evaluated in order except:
 - a. Atomic forms are not evaluated. They are treated as labels.
 - b. (GO label) means go back to the form that follows label & continue.
 - c. (return expr) exits prog evaluation & returns the (eval expr) as the value of the prog.
4. If prog runs out of forms, it returns nil.
5. If a cond runs out of stuff inside a prog it is treated as a NOOP.

Example:

```
(defun fact (n) (prog (i j)
  (if (< n 0) (return nil) )
  (setf i 1 j 1)
  ( if (< n 2) (return j) )
  LOOP
  (setf i (+ i 1) j (* j i) )
  (if (= i n) (return j))
  (go LOOP)
))
```

PROPERTY LISTS

The ability to associate with an atom in LISP a lisp form (i.e., the property value) under a given atomic designation (the property name).

template: (putprop atm pvalue pname)

Requires 3 arguments.

atm - must be atomic, i.e., the atom to receive the property.

pname - must also be atomic, i.e., the property name.

pvalue - the property value, which can be any sex.

```
> (putprop 'cain 'father 'adam)
```

```
> (putprop 'eve 'mother '(cain abel))
```

Putprop is not always included in some LISP dialects. If not use:

```
> (setf (get 'cain 'father) 'adam)
```

```
> (setf (get 'eve 'mother) '(cain abel))
```

```
> (setf (get 'eve 'grandmother) '(tubalcain jubal))
```

```
> (defun putprop (atm pname pvalue) (setf (get atm pname) pvalue))
```

In memory, property lists are stored as association lists or dotted pairs associated with the atom, e.g.,
(eve (mother (cain abel)) (grandmother (tubalcain jubal)))

To retrieve properties use the function (get atm pname). It returns the property value if one has been defined or nil. Therefore, the system cannot distinguish between undefined properties and those whose value has been set to nil. A good rule to follow is to always set pvalue to something non-nil

```
(get 'cain 'father) ==> adam
```

```
(get 'eve 'mother) ==> (cain abel)
```

To remove a property use (remprop atm pname). Returns nil.

Another way to view this is as an associated list of pairs (a-list for short). Associated pairs are lists of sublists where the first element of each sublist is a "key or indicator" and the cdr of the sublist is the value. For example:

```
(setq eve '( (mother (cain abel)) (grandmother (tubalcain jubal)) ))
```

The function (assoc key a-list) returns the corresponding pair that has a key value matching argument.

```
(assoc 'mother eve) ==> (mother (cain abel))
```

-or-

```
(assoc 'mother eve) ==> (cain abel) {in some systems}
```