Quick Reference

Common lisp

Bert Burgemeister

Common Lisp Quick Reference Revision 144 [2015-05-22]

LaTeX source: http://clqr.boundp.org

Contents

1 Numbers **3**
1.1 Predicates 3
1.2 Numeric Functns . 3
1.3 Logic Functions . 5
1.4 Integer Functions . 6
1.5 Implementation-Dependent 6

2 Characters **7**

3 Strings **8**

4 Conses **8**
4.1 Predicates 8
4.2 Lists 9
4.3 Association Lists . 10
4.4 Trees 10
4.5 Sets 11

5 Arrays **11**
5.1 Predicates 11
5.2 Array Functions . 11
5.3 Vector Functions . 12

6 Sequences **12**
6.1 Seq. Predicates . . 12
6.2 Seq. Functions . . 13

7 Hash Tables **15**

8 Structures **16**

9 Control Structure **16**
9.1 Predicates 16
9.2 Variables 17
9.3 Functions 18
9.4 Macros 19
9.5 Control Flow . . . 21
9.6 Iteration 22
9.7 Loop Facility . . . 22

10 CLOS **25**
10.1 Classes 25
10.2 Generic Functns . 26
10.3 Method Combination Types . . . 28

11 Conditions and Errors **28**

12 Types and Classes **31**

13 Input/Output **33**
13.1 Predicates 33
13.2 Reader 33
13.3 Character Syntax . 35
13.4 Printer 36
13.5 Format 38
13.6 Streams 40
13.7 Paths and Files . . 42

14 Packages and Symbols **43**
14.1 Predicates 43
14.2 Packages 44
14.3 Symbols 45
14.4 Std Packages . . 46

15 Compiler **46**
15.1 Predicates 46
15.2 Compilation . . . 46
15.3 REPL & Debug . 47
15.4 Declarations . . . 48

16 External Environment **49**

Typographic Conventions

name; $_f$**name**; $_g$**name**; $_m$**name**; $_s$**name**; $_v\textbf{*name*}$; $_c$**name**
▷ Symbol defined in Common Lisp; esp. function, generic function, macro, special operator, variable, constant.

them ▷ Placeholder for actual code.

`me` ▷ Literal text.

$[foo_{\boxed{\texttt{bar}}}]$ ▷ Either one *foo* or nothing; defaults to `bar`.

foo^*; $\{foo\}^*$ ▷ Zero or more *foo*s.

foo^+; $\{foo\}^+$ ▷ One or more *foo*s.

foos ▷ English plural denotes a list argument.

$\{foo \big| bar \big| baz\}$; $\begin{cases} foo \\ bar \\ baz \end{cases}$ ▷ Either *foo*, or *bar*, or *baz*.

$\left\{\begin{array}{|l} foo \\ bar \\ baz \end{array}\right.$ ▷ Anything from none to each of *foo*, *bar*, and *baz*.

$\widehat{foo}$ ▷ Argument *foo* is not evaluated.

$\widetilde{bar}$ ▷ Argument *bar* is possibly modified.

$foo^{\overset{\mathrm{P}}{*}}$ ▷ foo^* is evaluated as in $_s$**progn**; see page 21.

$\underline{foo}$; $\underset{2}{\underline{bar}}$; $\underset{n}{\underline{baz}}$ ▷ Primary, secondary, and nth return value.

`T`; `NIL` ▷ **t**, or truth in general; and **nil** or **()**.

1 Numbers

1.1 Predicates

($_f$**=** *number*$^+$)
($_f$**/=** *number*$^+$)

▷ $\underline{\texttt{T}}$ if all *number*s, or none, respectively, are equal in value.

($_f$**>** *number*$^+$)
($_f$**>=** *number*$^+$)
($_f$**<** *number*$^+$)
($_f$**<=** *number*$^+$)

▷ Return $\underline{\texttt{T}}$ if *number*s are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively.

($_f$**minusp** *a*)
($_f$**zerop** *a*)
($_f$**plusp** *a*)

▷ $\underline{\texttt{T}}$ if $a < 0$, $a = 0$, or $a > 0$, respectively.

($_f$**evenp** *int*)
($_f$**oddp** *int*)

▷ $\underline{\texttt{T}}$ if *int* is even or odd, respectively.

($_f$**numberp** *foo*)
($_f$**realp** *foo*)
($_f$**rationalp** *foo*)
($_f$**floatp** *foo*)
($_f$**integerp** *foo*)
($_f$**complexp** *foo*)
($_f$**random-state-p** *foo*)

▷ $\underline{\texttt{T}}$ if *foo* is of indicated type.

1.2 Numeric Functions

($_f$**+** $a_{\boxed{0}}{}^*$)
($_f$***** $a_{\boxed{1}}{}^*$)

▷ Return $\underline{\sum a}$ or $\underline{\prod a}$, respectively.

($_f$**−** *a* b^*)
($_f$**/** *a* b^*)

▷ Return $\underline{a - \sum b}$ or $\underline{a/\prod b}$, respectively. Without any *b*s, return $\underline{-a}$ or $\underline{1/a}$, respectively.

($_f$**1+** *a*)
($_f$**1−** *a*)

▷ Return $\underline{a + 1}$ or $\underline{a - 1}$, respectively.

($\begin{Bmatrix} {}_m\textbf{incf} \\ {}_m\textbf{decf} \end{Bmatrix}$ $\widetilde{place}$ [$delta_{\boxed{1}}$])

▷ Increment or decrement the value of *place* by *delta*. Return $\underline{\text{new value}}$.

($_f$**exp** *p*)
($_f$**expt** *b p*)

▷ Return $\underline{e^p}$ or $\underline{b^p}$, respectively.

($_f$**log** *a* [$b_{\boxed{e}}$])

▷ Return $\underline{\log_b a}$ or, without *b*, $\underline{\ln a}$.

($_f$**sqrt** *n*)
($_f$**isqrt** *n*)

▷ $\underline{\sqrt{n}}$ in complex numbers/natural numbers.

($_f$**lcm** $integer^*{}_{\boxed{1}}$)
($_f$**gcd** $integer^*$)

▷ $\underline{\text{Least common multiple}}$ or $\underline{\text{greatest common denominator}}$, respectively, of *integer*s. (**gcd**) returns $\underline{0}$.

$_c$**pi**

▷ **long-float** approximation of π, Ludolph's number.

($_f$**sin** *a*)
($_f$**cos** *a*)
($_f$**tan** *a*)

▷ $\underline{\sin a}$, $\underline{\cos a}$, or $\underline{\tan a}$, respectively. (*a* in radians.)

($_f$**asin** *a*)
($_f$**acos** *a*)

▷ $\underline{\arcsin a}$ or $\underline{\arccos a}$, respectively, in radians.

($_f$**atan** *a* [$b_{\boxed{1}}$])

▷ $\underline{\arctan \frac{a}{b}}$ in radians.

($_f$**sinh** a)
($_f$**cosh** a) ▷ $\underline{\sinh a}$, $\underline{\cosh a}$, or $\underline{\tanh a}$, respectively.
($_f$**tanh** a)

($_f$**asinh** a)
($_f$**acosh** a) ▷ $\underline{\operatorname{asinh} a}$, $\underline{\operatorname{acosh} a}$, or $\underline{\operatorname{atanh} a}$, respectively.
($_f$**atanh** a)

($_f$**cis** a) ▷ Return $\underline{e^{i\,a}} = \underline{\cos a + i \sin a}$.

($_f$**conjugate** a) ▷ Return complex $\underline{\text{conjugate of } a}$.

($_f$**max** num^+)
($_f$**min** num^+) ▷ $\underline{\text{Greatest}}$ or $\underline{\text{least}}$, respectively, of *nums*.

$$\left(\left\{\begin{array}{l}\{{}_f\textbf{round}|{}_f\textbf{fround}\}\\ \{{}_f\textbf{floor}|{}_f\textbf{ffloor}\}\\ \{{}_f\textbf{ceiling}|{}_f\textbf{fceiling}\}\\ \{{}_f\textbf{truncate}|{}_f\textbf{ftruncate}\}\end{array}\right\}\ n\ [d_{\boxed{1}}]\right)$$

▷ Return as **integer** or **float**, respectively, $\underline{n/d}$ rounded, or rounded towards $-\infty$, $+\infty$, or 0, respectively; and $\underset{2}{\underline{\text{remainder}}}$.

$$\left(\left\{\begin{array}{l}{}_f\textbf{mod}\\ {}_f\textbf{rem}\end{array}\right\}\ n\ d\right)$$

▷ Same as $_f$**floor** or $_f$**truncate**, respectively, but return $\underline{\text{remainder}}$ only.

($_f$**random** *limit* [$\widetilde{state}_{\boxed{{}_v\textbf{*random-state*}}}$])
▷ Return non-negative $\underline{\text{random number}}$ less than *limit*, and of the same type.

($_f$**make-random-state** $\left[\{state|\texttt{NIL}|\texttt{T}\}_{\boxed{\texttt{NIL}}}\right]$)
▷ $\underline{\text{Copy}}$ of **random-state** object *state* or of the current random state; or a randomly initialized fresh $\underline{\text{random state}}$.

$_v$***random-state*** ▷ Current random state.

($_f$**float-sign** *num-a* [$num\text{-}b_{\boxed{1}}$]) ▷ $\underline{num\text{-}b}$ with *num-a*'s sign.

($_f$**signum** n)
▷ $\underline{\text{Number}}$ of magnitude 1 representing sign or phase of n.

($_f$**numerator** *rational*)
($_f$**denominator** *rational*)
▷ $\underline{\text{Numerator}}$ or $\underline{\text{denominator}}$, respectively, of *rational*'s canonical form.

($_f$**realpart** *number*)
($_f$**imagpart** *number*)
▷ $\underline{\text{Real part}}$ or $\underline{\text{imaginary part}}$, respectively, of *number*.

($_f$**complex** *real* [$imag_{\boxed{0}}$]) ▷ Make a $\underline{\text{complex number}}$.

($_f$**phase** *num*) ▷ $\underline{\text{Angle}}$ of *num*'s polar representation.

($_f$**abs** n) ▷ Return $\underline{|n|}$.

($_f$**rational** *real*)
($_f$**rationalize** *real*)
▷ Convert *real* to $\underline{\text{rational}}$. Assume complete/limited accuracy for *real*.

($_f$**float** *real* [$prototype_{\boxed{\texttt{0.0F0}}}$])
▷ Convert *real* into $\underline{\text{float}}$ with type of *prototype*.

USER-HOMEDIR-PATHNAME 42
USING 24

V 40
VALUES 18, 33
VALUES-LIST 18
VARIABLE 45
VECTOR 12, 32
VECTOR-POP 12
VECTOR-PUSH 12
VECTOR-PUSH-EXTEND 12
VECTORP 11
WARN 29
WARNING 32
WHEN 21, 24
WHILE 24
WILD-PATHNAME-P 33
WITH 22
WITH-ACCESSORS 25
WITH-COMPILATION-UNIT 47
WITH-CONDITION-RESTARTS 30
WITH-HASH-TABLE-ITERATOR 15
WITH-INPUT-FROM-STRING 41
WITH-OPEN-FILE 41
WITH-OPEN-STREAM 41
WITH-OUTPUT-TO-STRING 42
WITH-PACKAGE-ITERATOR 45
WITH-SIMPLE-RESTART 30
WITH-SLOTS 25
WITH-STANDARD-IO-SYNTAX 33
WRITE 37
WRITE-BYTE 36
WRITE-CHAR 36
WRITE-LINE 36
WRITE-SEQUENCE 36
WRITE-STRING 36
WRITE-TO-STRING 37

Y-OR-N-P 33
YES-OR-NO-P 33

ZEROP 3

NAME-CHAR 7
NAMED 22
NAMESTRING 43
NBUTLAST 9
NCONC 10, 24, 28
NCONCING 24
NEVER 25
NEWLINE 7
NEXT-METHOD-P 26
NIL 2, 46
NINTERSECTION 11
NINTH 9
NO-APPLICABLE-METHOD 27
NO-NEXT-METHOD 27
NOT 16, 33, 36
NOTANY 12
NOTEVERY 12
NOTINLINE 48
NRECONC 10
NREVERSE 13
NSET-DIFFERENCE 11
NSET-EXCLUSIVE-OR 11
NSTRING-CAPITALIZE 8
NSTRING-DOWNCASE 8
NSTRING-UPCASE 8
NSUBLIS 11
NSUBST 10
NSUBST-IF 10
NSUBST-IF-NOT 10
NSUBSTITUTE 14
NSUBSTITUTE-IF 14
NSUBSTITUTE-IF-NOT 14
NTH 9
NTH-VALUE 18
NTHCDR 9
NULL 8, 32
NUMBER 32
NUMBERP 3
NUMERATOR 4
NUNION 11

ODDP 3
OF 24
OF-TYPE 22
ON 22
OPEN 40
OPEN-STREAM-P 33
OPTIMIZE 49
OR 21, 28, 33, 36
OTHERWISE 21, 31
OUTPUT-STREAM-P 33

PACKAGE 32
PACKAGE-ERROR 32
PACKAGE-ERROR-PACKAGE 31
PACKAGE-NAME 44
PACKAGE-NICKNAMES 44
PACKAGE-SHADOWING-SYMBOLS 45
PACKAGE-USE-LIST 44
PACKAGE-USED-BY-LIST 44
PACKAGEP 43
PAIRLIS 10
PARSE-ERROR 32
PARSE-INTEGER 8
PARSE-NAMESTRING 42
PATHNAME 32, 43
PATHNAME-DEVICE 42
PATHNAME-DIRECTORY 42
PATHNAME-HOST 42
PATHNAME-MATCH-P 33
PATHNAME-NAME 42
PATHNAME-TYPE 42
PATHNAME-VERSION 42
PATHNAMEP 33
PEEK-CHAR 34
PHASE 4
PI 3
PLUSP 3
POP 9
POSITION 13
POSITION-IF 14
POSITION-IF-NOT 14
PPRINT 36
PPRINT-DISPATCH 38
PPRINT-EXIT-IF-LIST-EXHAUSTED 37
PPRINT-FILL 37
PPRINT-INDENT 37
PPRINT-LINEAR 37
PPRINT-LOGICAL-BLOCK 37
PPRINT-NEWLINE 37
PPRINT-POP 37
PPRINT-TAB 37
PPRINT-TABULAR 37
PRESENT-SYMBOL 24
PRESENT-SYMBOLS 24
PRIN1 36
PRIN1-TO-STRING 36
PRINC 36
PRINC-TO-STRING 36
PRINT 36
PRINT-NOT-READABLE 32
PRINT-NOT-READABLE-OBJECT 31
PRINT-OBJECT 36
PRINT-UNREADABLE-OBJECT 36
PROBE-FILE 43
PROCLAIM 48
PROG 21
PROG1 21
PROG2 21
PROG∗ 21
PROGN 21, 28
PROGRAM-ERROR 32
PROGV 17
PROVIDE 45
PSETF 17
PSETQ 17
PUSH 10
PUSHNEW 10

QUOTE 35, 47

RANDOM 4
RANDOM-STATE 32
RANDOM-STATE-P 3
RASSOC 10
RASSOC-IF 10
RASSOC-IF-NOT 10
RATIO 32, 35
RATIONAL 4, 32
RATIONALIZE 4
RATIONALP 3
READ 33
READ-BYTE 34
READ-CHAR 34
READ-CHAR-NO-HANG 34
READ-DELIMITED-LIST 34
READ-FROM-STRING 33
READ-LINE 34
READ-PRESERVING-WHITESPACE 33
READ-SEQUENCE 34
READER-ERROR 32
READTABLE 32
READTABLE-CASE 34
READTABLEP 33
REAL 32
REALP 3
REALPART 4
REDUCE 15
REINITIALIZE-INSTANCE 25
REM 4
REMF 17
REMHASH 15
REMOVE 14
REMOVE-DUPLICATES 14
REMOVE-IF 14
REMOVE-IF-NOT 14
REMOVE-METHOD 27
REMPROP 17
RENAME-FILE 43
RENAME-PACKAGE 44
REPEAT 24
REPLACE 14
REQUIRE 45
REST 9
RESTART 32
RESTART-BIND 30
RESTART-CASE 30
RESTART-NAME 30
RETURN 21, 24
RETURN-FROM 21
REVAPPEND 10
REVERSE 13
ROOM 48
ROTATEF 17
ROUND 4
ROW-MAJOR-AREF 11
RPLACA 9
RPLACD 9

SAFETY 49
SATISFIES 33
SBIT 12
SCALE-FLOAT 6
SCHAR 8
SEARCH 14
SECOND 9
SEQUENCE 32
SERIOUS-CONDITION 32
SET 17
SET-DIFFERENCE 11
SET-DISPATCH-MACRO-CHARACTER 35
SET-EXCLUSIVE-OR 11
SET-MACRO-CHARACTER 34
SET-PPRINT-DISPATCH 38
SET-SYNTAX-FROM-CHAR 34
SETF 17, 45
SETQ 17
SEVENTH 9
SHADOW 44
SHADOWING-IMPORT 44
SHARED-INITIALIZE 26
SHIFTF 17
SHORT-FLOAT 32, 35
SHORT-FLOAT-EPSILON 6
SHORT-FLOAT-NEGATIVE-EPSILON 6
SHORT-SITE-NAME 49
SIGNAL 29
SIGNED-BYTE 32
SIGNUM 4
SIMPLE-ARRAY 32
SIMPLE-BASE-STRING 32
SIMPLE-BIT-VECTOR 32
SIMPLE-BIT-VECTOR-P 11
SIMPLE-CONDITION 32
SIMPLE-CONDITION-FORMAT-ARGUMENTS 31
SIMPLE-CONDITION-FORMAT-CONTROL 31
SIMPLE-ERROR 32
SIMPLE-STRING 32
SIMPLE-STRING-P 8
SIMPLE-TYPE-ERROR 32
SIMPLE-VECTOR 32
SIMPLE-VECTOR-P 11
SIMPLE-WARNING 32
SIN 3
SINGLE-FLOAT 32, 35
SINGLE-FLOAT-EPSILON 6
SINGLE-FLOAT-NEGATIVE-EPSILON 6
SINH 4
SIXTH 9
SLEEP 22
SLOT-BOUNDP 25
SLOT-EXISTS-P 25
SLOT-MAKUNBOUND 25
SLOT-MISSING 26
SLOT-UNBOUND 26
SLOT-VALUE 25
SOFTWARE-TYPE 49
SOFTWARE-VERSION 49
SOME 12
SORT 13
SPACE 7, 49
SPECIAL 49
SPECIAL-OPERATOR-P 46
SPEED 49
SQRT 3
STABLE-SORT 13
STANDARD 28
STANDARD-CHAR 7, 32
STANDARD-CHAR-P 7
STANDARD-CLASS 32
STANDARD-GENERIC-FUNCTION 32
STANDARD-METHOD 32
STANDARD-OBJECT 32
STEP 48
STORAGE-CONDITION 32
STORE-VALUE 30
STREAM 32
STREAM-ELEMENT-TYPE 31
STREAM-ERROR 32
STREAM-ERROR-STREAM 31
STREAM-EXTERNAL-FORMAT 42
STREAMP 33
STRING 8, 32
STRING-CAPITALIZE 8
STRING-DOWNCASE 8
STRING-EQUAL 8
STRING-GREATERP 8
STRING-LEFT-TRIM 8
STRING-LESSP 8
STRING-NOT-EQUAL 8
STRING-NOT-GREATERP 8
STRING-NOT-LESSP 8
STRING-RIGHT-TRIM 8
STRING-STREAM 32
STRING-TRIM 8
STRING-UPCASE 8
STRING/= 8
STRING< 8
STRING<= 8
STRING= 8
STRING> 8
STRING>= 8
STRINGP 8
STRUCTURE 45
STRUCTURE-CLASS 32
STRUCTURE-OBJECT 32
STYLE-WARNING 32
SUBLIS 11
SUBSEQ 13
SUBSETP 9
SUBST 10
SUBST-IF 10
SUBST-IF-NOT 10
SUBSTITUTE 14
SUBSTITUTE-IF 14
SUBSTITUTE-IF-NOT 14
SUBTYPEP 31
SUM 24
SUMMING 24
SVREF 12
SXHASH 15
SYMBOL 24, 32, 45
SYMBOL-FUNCTION 45
SYMBOL-MACROLET 20
SYMBOL-NAME 45
SYMBOL-PACKAGE 45
SYMBOL-PLIST 45
SYMBOL-VALUE 45
SYMBOLP 43
SYMBOLS 24
SYNONYM-STREAM 32
SYNONYM-STREAM-SYMBOL 41

T 2, 32, 46
TAGBODY 21
TAILP 9
TAN 3
TANH 4
TENTH 9
TERPRI 36
THE 24, 31
THEN 24
THEREIS 25
THIRD 9
THROW 22
TIME 48
TO 22
TRACE 48
TRANSLATE-LOGICAL-PATHNAME 43
TRANSLATE-PATHNAME 43
TREE-EQUAL 10
TRUENAME 43
TRUNCATE 4
TWO-WAY-STREAM 32
TWO-WAY-STREAM-INPUT-STREAM 41
TWO-WAY-STREAM-OUTPUT-STREAM 41
TYPE 45, 48
TYPE-ERROR 32
TYPE-ERROR-DATUM 31
TYPE-ERROR-EXPECTED-TYPE 31
TYPE-OF 31
TYPECASE 31
TYPEP 31

UNBOUND-SLOT 32
UNBOUND-SLOT-INSTANCE 31
UNBOUND-VARIABLE 32
UNDEFINED-FUNCTION 32
UNEXPORT 45
UNINTERN 44
UNION 11
UNLESS 21, 24
UNREAD-CHAR 34
UNSIGNED-BYTE 32
UNTIL 24
UNTRACE 48
UNUSE-PACKAGE 44
UNWIND-PROTECT 21
UPDATE-INSTANCE-FOR-DIFFERENT-CLASS 26
UPDATE-INSTANCE-FOR-REDEFINED-CLASS 26
UPFROM 22
UPGRADED-ARRAY-ELEMENT-TYPE 31
UPGRADED-COMPLEX-PART-TYPE 6
UPPER-CASE-P 7
UPTO 22
USE-PACKAGE 44
USE-VALUE 30

1.3 Logic Functions

Negative integers are used in two's complement representation.

($_f$**boole** *operation int-a int-b*)
▷ Return value of bitwise logical *operation*. *operation*s are

$_c$**boole-1**	▷ *int-a*.
$_c$**boole-2**	▷ *int-b*.
$_c$**boole-c1**	▷ $\neg$*int-a*.
$_c$**boole-c2**	▷ $\neg$*int-b*.
$_c$**boole-set**	▷ All bits set.
$_c$**boole-clr**	▷ All bits zero.
$_c$**boole-eqv**	▷ $\textit{int-a} \equiv \textit{int-b}$.
$_c$**boole-and**	▷ $\textit{int-a} \wedge \textit{int-b}$.
$_c$**boole-andc1**	▷ $\neg\textit{int-a} \wedge \textit{int-b}$.
$_c$**boole-andc2**	▷ $\textit{int-a} \wedge \neg\textit{int-b}$.
$_c$**boole-nand**	▷ $\neg(\textit{int-a} \wedge \textit{int-b})$.
$_c$**boole-ior**	▷ $\textit{int-a} \vee \textit{int-b}$.
$_c$**boole-orc1**	▷ $\neg\textit{int-a} \vee \textit{int-b}$.
$_c$**boole-orc2**	▷ $\textit{int-a} \vee \neg\textit{int-b}$.
$_c$**boole-xor**	▷ $\neg(\textit{int-a} \equiv \textit{int-b})$.
$_c$**boole-nor**	▷ $\neg(\textit{int-a} \vee \textit{int-b})$.

($_f$**lognot** *integer*) ▷ $\neg$*integer*.

($_f$**logeqv** *integer**)
($_f$**logand** *integer**)
▷ Return value of exclusive-nored or anded *integer*s, respectively. Without any *integer*, return -1.

($_f$**logandc1** *int-a int-b*) ▷ $\neg\textit{int-a} \wedge \textit{int-b}$.

($_f$**logandc2** *int-a int-b*) ▷ $\textit{int-a} \wedge \neg\textit{int-b}$.

($_f$**lognand** *int-a int-b*) ▷ $\neg(\textit{int-a} \wedge \textit{int-b})$.

($_f$**logxor** *integer**)
($_f$**logior** *integer**)
▷ Return value of exclusive-ored or ored *integer*s, respectively. Without any *integer*, return 0.

($_f$**logorc1** *int-a int-b*) ▷ $\neg\textit{int-a} \vee \textit{int-b}$.

($_f$**logorc2** *int-a int-b*) ▷ $\textit{int-a} \vee \neg\textit{int-b}$.

($_f$**lognor** *int-a int-b*) ▷ $\neg(\textit{int-a} \vee \textit{int-b})$.

($_f$**logbitp** *i int*) ▷ T if zero-indexed *i*th bit of *int* is set.

($_f$**logtest** *int-a int-b*)
▷ Return T if there is any bit set in *int-a* which is set in *int-b* as well.

($_f$**logcount** *int*)
▷ Number of 1 bits in $\textit{int} \geq 0$, number of 0 bits in $\textit{int} < 0$.

1.4 Integer Functions

($_f$**integer-length** *integer*)
▷ Number of bits necessary to represent *integer*.

($_f$**ldb-test** *byte-spec integer*)
▷ Return `T` if any bit specified by *byte-spec* in *integer* is set.

($_f$**ash** *integer count*)
▷ Return copy of *integer* arithmetically shifted left by *count* adding zeros at the right, or, for *count* < 0, shifted right discarding bits.

($_f$**ldb** *byte-spec integer*)
▷ Extract byte denoted by *byte-spec* from *integer*. **setf**able.

$$\left(\begin{Bmatrix} {}_f\textbf{deposit-field} \\ {}_f\textbf{dpb} \end{Bmatrix} \textit{int-a byte-spec int-b}\right)$$

▷ Return *int-b* with bits denoted by *byte-spec* replaced by corresponding bits of *int-a*, or by the low ($_f$**byte-size** *byte-spec*) bits of *int-a*, respectively.

($_f$**mask-field** *byte-spec integer*)
▷ Return copy of *integer* with all bits unset but those denoted by *byte-spec*. **setf**able.

($_f$**byte** *size position*)
▷ Byte specifier for a byte of *size* bits starting at a weight of $2^{position}$.

($_f$**byte-size** *byte-spec*)
($_f$**byte-position** *byte-spec*)
▷ Size or position, respectively, of *byte-spec*.

1.5 Implementation-Dependent

$$\left.\begin{matrix} {}_c\textbf{short-float} \\ {}_c\textbf{single-float} \\ {}_c\textbf{double-float} \\ {}_c\textbf{long-float} \end{matrix}\right\}\text{-}\left\{\begin{matrix} \textbf{epsilon} \\ \textbf{negative-epsilon} \end{matrix}\right.$$

▷ Smallest possible number making a difference when added or subtracted, respectively.

$$\left.\begin{matrix} {}_c\textbf{least-negative} \\ {}_c\textbf{least-negative-normalized} \\ {}_c\textbf{least-positive} \\ {}_c\textbf{least-positive-normalized} \end{matrix}\right\}\text{-}\left\{\begin{matrix} \textbf{short-float} \\ \textbf{single-float} \\ \textbf{double-float} \\ \textbf{long-float} \end{matrix}\right.$$

▷ Available numbers closest to -0 or $+0$, respectively.

$$\left.\begin{matrix} {}_c\textbf{most-negative} \\ {}_c\textbf{most-positive} \end{matrix}\right\}\text{-}\left\{\begin{matrix} \textbf{short-float} \\ \textbf{single-float} \\ \textbf{double-float} \\ \textbf{long-float} \\ \textbf{fixnum} \end{matrix}\right.$$

▷ Available numbers closest to $-\infty$ or $+\infty$, respectively.

($_f$**decode-float** *n*)
($_f$**integer-decode-float** *n*)
▷ Return significand, $\underset{2}{\underline{\text{exponent}}}$, and $\underset{3}{\underline{\text{sign}}}$ of **float** *n*.

($_f$**scale-float** *n* [*i*]) ▷ With *n*'s radix b, return nb^i.

($_f$**float-radix** *n*)
($_f$**float-digits** *n*)
($_f$**float-precision** *n*)
▷ Radix, number of digits in that radix, or precision in that radix, respectively, of float *n*.

($_f$**upgraded-complex-part-type** *foo* [*environment*$_{\boxed{\texttt{NIL}}}$])
▷ Type of most specialized **complex** number able to hold parts of type *foo*.

DIGIT-CHAR-P 7
DIRECTORY 43
DIRECTORY-NAMESTRING 43
DISASSEMBLE 48
DIVISION-BY-ZERO 32
DO 22, 24
DO-ALL-SYMBOLS 45
DO-EXTERNAL-SYMBOLS 45
DO-SYMBOLS 45
DO∗ 22
DOCUMENTATION 45
DOING 24
DOLIST 22
DOTIMES 22
DOUBLE-FLOAT 32, 35
DOUBLE-FLOAT-EPSILON 6
DOUBLE-FLOAT-NEGATIVE-EPSILON 6
DOWNFROM 22
DOWNTO 22
DPB 6
DRIBBLE 47
DYNAMIC-EXTENT 48

EACH 24
ECASE 21
ECHO-STREAM 32
ECHO-STREAM-INPUT-STREAM 41
ECHO-STREAM-OUTPUT-STREAM 41
ED 47
EIGHTH 9
ELSE 24
ELT 13
ENCODE-UNIVERSAL-TIME 49
END 24
END-OF-FILE 32
ENDP 8
ENOUGH-NAMESTRING 43
ENSURE-DIRECTORIES-EXIST 43
ENSURE-GENERIC-FUNCTION 26
EQ 16
EQL 16, 33
EQUAL 16
EQUALP 16
ERROR 29, 32
ETYPECASE 31
EVAL 47
EVAL-WHEN 47
EVENP 3
EVERY 12
EXP 3
EXPORT 45
EXPT 3
EXTENDED-CHAR 32
EXTERNAL-SYMBOL 24
EXTERNAL-SYMBOLS 24

FBOUNDP 17
FCEILING 4
FDEFINITION 19
FFLOOR 4
FIFTH 9
FILE-AUTHOR 43
FILE-ERROR 32
FILE-ERROR-PATHNAME 31
FILE-LENGTH 43
FILE-NAMESTRING 43
FILE-POSITION 41
FILE-STREAM 32
FILE-STRING-LENGTH 41
FILE-WRITE-DATE 43
FILL 13
FILL-POINTER 12
FINALLY 24
FIND 13
FIND-ALL-SYMBOLS 44
FIND-CLASS 25
FIND-IF 14
FIND-IF-NOT 14
FIND-METHOD 27
FIND-PACKAGE 44
FIND-RESTART 30
FIND-SYMBOL 44
FINISH-OUTPUT 41
FIRST 9
FIXNUM 32
FLET 18
FLOAT 4, 32
FLOAT-DIGITS 6
FLOAT-PRECISION 6
FLOAT-RADIX 6
FLOAT-SIGN 4
FLOATING-POINT-INEXACT 32
FLOATING-POINT-INVALID-OPERATION 32
FLOATING-POINT-OVERFLOW 32
FLOATING-POINT-UNDERFLOW 32
FLOATP 3
FLOOR 4
FMAKUNBOUND 19
FOR 22
FORCE-OUTPUT 41
FORMAT 38
FORMATTER 38
FOURTH 9
FRESH-LINE 36
FROM 22
FROUND 4
FTRUNCATE 4
FTYPE 48
FUNCALL 18
FUNCTION 18, 32, 35, 45
FUNCTION-KEYWORDS 27
FUNCTION-LAMBDA-EXPRESSION 19
FUNCTIONP 17

GCD 3
GENERIC-FUNCTION 32
GENSYM 45
GENTEMP 45
GET 17
GET-DECODED-TIME 49
GET-DISPATCH-MACRO-CHARACTER 35
GET-INTERNAL-REAL-TIME 49
GET-INTERNAL-RUN-TIME 49
GET-MACRO-CHARACTER 34
GET-OUTPUT-STREAM-STRING 41
GET-PROPERTIES 17
GET-SETF-EXPANSION 20
GET-UNIVERSAL-TIME 49
GETF 17
GETHASH 15
GO 21
GRAPHIC-CHAR-P 7

HANDLER-BIND 29
HANDLER-CASE 29
HASH-KEY 24
HASH-KEYS 24
HASH-TABLE 32
HASH-TABLE-COUNT 15
HASH-TABLE-P 15
HASH-TABLE-REHASH-SIZE 15
HASH-TABLE-REHASH-THRESHOLD 15
HASH-TABLE-SIZE 15
HASH-TABLE-TEST 15
HASH-VALUE 24
HASH-VALUES 24
HOST-NAMESTRING 43

IDENTITY 19
IF 21, 24
IGNORABLE 48
IGNORE 48
IGNORE-ERRORS 29
IMAGPART 4
IMPORT 44
IN 22, 24
IN-PACKAGE 44
INCF 3
INITIALIZE-INSTANCE 26
INITIALLY 24
INLINE 48
INPUT-STREAM-P 33
INSPECT 48
INTEGER 32
INTEGER-DECODE-FLOAT 6
INTEGER-LENGTH 6
INTEGERP 3
INTERACTIVE-STREAM-P 33
INTERN 44
INTERNAL-TIME-UNITS-PER-SECOND 49
INTERSECTION 11
INTO 24
INVALID-METHOD-ERROR 27
INVOKE-DEBUGGER 29
INVOKE-RESTART 30
INVOKE-RESTART-INTERACTIVELY 30
ISQRT 3
IT 24

KEYWORD 32, 44, 46
KEYWORDP 43

LABELS 18
LAMBDA 18
LAMBDA-LIST-KEYWORDS 20
LAMBDA-PARAMETERS-LIMIT 19
LAST 9
LCM 3
LDB 6
LDB-TEST 6
LDIFF 9
LEAST-NEGATIVE-DOUBLE-FLOAT 6
LEAST-NEGATIVE-LONG-FLOAT 6
LEAST-NEGATIVE-NORMALIZED-DOUBLE-FLOAT 6
LEAST-NEGATIVE-NORMALIZED-LONG-FLOAT 6
LEAST-NEGATIVE-NORMALIZED-SHORT-FLOAT 6
LEAST-NEGATIVE-NORMALIZED-SINGLE-FLOAT 6
LEAST-NEGATIVE-SHORT-FLOAT 6
LEAST-NEGATIVE-SINGLE-FLOAT 6
LEAST-POSITIVE-DOUBLE-FLOAT 6
LEAST-POSITIVE-LONG-FLOAT 6
LEAST-POSITIVE-NORMALIZED-DOUBLE-FLOAT 6
LEAST-POSITIVE-NORMALIZED-LONG-FLOAT 6
LEAST-POSITIVE-NORMALIZED-SHORT-FLOAT 6
LEAST-POSITIVE-NORMALIZED-SINGLE-FLOAT 6
LEAST-POSITIVE-SHORT-FLOAT 6
LEAST-POSITIVE-SINGLE-FLOAT 6
LENGTH 13
LET 17
LET∗ 17
LISP-IMPLEMENTATION-TYPE 49
LISP-IMPLEMENTATION-VERSION 49
LIST 9, 28, 32
LIST-ALL-PACKAGES 44
LIST-LENGTH 9
LIST∗ 9
LISTEN 41
LISTP 8
LOAD 46
LOAD-LOGICAL-PATHNAME-TRANSLATIONS 43
LOAD-TIME-VALUE 47
LOCALLY 47
LOG 3
LOGAND 5
LOGANDC1 5
LOGANDC2 5
LOGBITP 5
LOGCOUNT 5
LOGEQV 5
LOGICAL-PATHNAME 32, 43
LOGICAL-PATHNAME-TRANSLATIONS 43
LOGIOR 5
LOGNAND 5
LOGNOR 5
LOGNOT 5
LOGORC1 5
LOGORC2 5
LOGTEST 5
LOGXOR 5
LONG-FLOAT 32, 35
LONG-FLOAT-EPSILON 6
LONG-FLOAT-NEGATIVE-EPSILON 6
LONG-SITE-NAME 49
LOOP 22
LOOP-FINISH 25
LOWER-CASE-P 7
MACHINE-INSTANCE 49
MACHINE-TYPE 49
MACHINE-VERSION 49
MACRO-FUNCTION 47
MACROEXPAND 47
MACROEXPAND-1 47
MACROLET 19
MAKE-ARRAY 11
MAKE-BROADCAST-STREAM 41
MAKE-CONCATENATED-STREAM 41
MAKE-CONDITION 29
MAKE-DISPATCH-MACRO-CHARACTER 34
MAKE-ECHO-STREAM 41
MAKE-HASH-TABLE 15
MAKE-INSTANCE 25
MAKE-INSTANCES-OBSOLETE 26
MAKE-LIST 9
MAKE-LOAD-FORM 47
MAKE-LOAD-FORM-SAVING-SLOTS 47
MAKE-METHOD 28
MAKE-PACKAGE 44
MAKE-PATHNAME 42
MAKE-RANDOM-STATE 4
MAKE-SEQUENCE 13
MAKE-STRING 8
MAKE-STRING-INPUT-STREAM 41
MAKE-STRING-OUTPUT-STREAM 41
MAKE-SYMBOL 45
MAKE-SYNONYM-STREAM 41
MAKE-TWO-WAY-STREAM 41
MAKUNBOUND 17
MAP 14
MAP-INTO 15
MAPC 10
MAPCAN 10
MAPCAR 10
MAPCON 10
MAPHASH 15
MAPL 10
MAPLIST 10
MASK-FIELD 6
MAX 4, 28
MAXIMIZE 24
MAXIMIZING 24
MEMBER 9, 33
MEMBER-IF 9
MEMBER-IF-NOT 9
MERGE 13
MERGE-PATHNAMES 42
METHOD 32
METHOD-COMBINATION 32, 45
METHOD-COMBINATION-ERROR 27
METHOD-QUALIFIERS 27
MIN 4, 28
MINIMIZE 24
MINIMIZING 24
MINUSP 3
MISMATCH 13
MOD 4, 33
MOST-NEGATIVE-DOUBLE-FLOAT 6
MOST-NEGATIVE-FIXNUM 6
MOST-NEGATIVE-LONG-FLOAT 6
MOST-NEGATIVE-SHORT-FLOAT 6
MOST-NEGATIVE-SINGLE-FLOAT 6
MOST-POSITIVE-DOUBLE-FLOAT 6
MOST-POSITIVE-FIXNUM 6
MOST-POSITIVE-LONG-FLOAT 6
MOST-POSITIVE-SHORT-FLOAT 6
MOST-POSITIVE-SINGLE-FLOAT 6
MUFFLE-WARNING 30
MULTIPLE-VALUE-BIND 18
MULTIPLE-VALUE-CALL 18
MULTIPLE-VALUE-LIST 18
MULTIPLE-VALUE-PROG1 21
MULTIPLE-VALUE-SETQ 17
MULTIPLE-VALUES-LIMIT 19

Index

" 35
' 35
(35
() 46
) 35
* 3, 32, 33, 43, 47
** 43, 47
*** 47
BREAK-ON-SIGNALS 31
COMPILE-FILE-PATHNAME 46
COMPILE-FILE-TRUENAME 46
COMPILE-PRINT 46
COMPILE-VERBOSE 46
DEBUG-IO 42
DEBUGGER-HOOK 31
DEFAULT-PATHNAME-DEFAULTS 42
ERROR-OUTPUT 42
FEATURES 36
GENSYM-COUNTER 45
LOAD-PATHNAME 46
LOAD-PRINT 46
LOAD-TRUENAME 46
LOAD-VERBOSE 46
MACROEXPAND-HOOK 48
MODULES 45
PACKAGE 44
PRINT-ARRAY 37
PRINT-BASE 37
PRINT-CASE 38
PRINT-CIRCLE 38
PRINT-ESCAPE 38
PRINT-GENSYM 38
PRINT-LENGTH 38
PRINT-LEVEL 38
PRINT-LINES 38
PRINT-MISER-WIDTH 38
PRINT-PPRINT-DISPATCH 38
PRINT-PRETTY 38
PRINT-RADIX 38
PRINT-READABLY 38
PRINT-RIGHT-MARGIN 38
QUERY-IO 42
RANDOM-STATE 4
READ-BASE 34
READ-DEFAULT-FLOAT-FORMAT 34
READ-EVAL 35
READ-SUPPRESS 34
READTABLE 34
STANDARD-INPUT 42
STANDARD-OUTPUT 42
TERMINAL-IO 42
TRACE-OUTPUT 48
+ 3, 28, 47
++ 47
+++ 47
, 35
,. 35
,@ 35
− 3, 47
. 35
/ 3, 35, 47
// 47
/// 47
/= 3
: 44
:: 44
:ALLOW-OTHER-KEYS 20
; 35
< 3
<= 3
= 3, 22, 24
> 3
>= 3
\ 36
40
#\ 35
#' 35
#(35
#* 35
#+ 36
#− 36
#. 35
#: 35
#< 35
#= 35
#A 35
#B 35
#C(35
#O 35
#P 35
#R 35
#S(35
#X 35
35
#| |# 35
&ALLOW-OTHER-KEYS 20
&AUX 20
&BODY 20
&ENVIRONMENT 20
&KEY 20
&OPTIONAL 20
&REST 20
&WHOLE 20
∼(∼) 39
∼* 40
∼/ / 40
∼< ∼:> 39
∼< ∼> 39
∼? 40
∼A 38
∼B 39
∼C 39
∼D 39
∼E 39
∼F 39
∼G 39
∼I 40
∼O 39
∼P 39
∼R 38
∼S 38
∼T 40
∼W 40
∼X 39
∼[∼] 40
∼$ 39
∼% 39
∼& 39
∼^ 40
∼_ 39
∼| 39
∼{ ∼} 40
∼∼ 39
∼↩ 39
` 35
| | 36
1+ 3
1− 3

ABORT 30
ABOVE 22
ABS 4
ACONS 10
ACOS 3
ACOSH 4
ACROSS 24
ADD-METHOD 27
ADJOIN 9
ADJUST-ARRAY 11
ADJUSTABLE-ARRAY-P 11
ALLOCATE-INSTANCE 26
ALPHA-CHAR-P 7
ALPHANUMERICP 7
ALWAYS 25
AND 21, 22, 24, 28, 33, 36
APPEND 10, 24, 28
APPENDING 24
APPLY 18
APROPOS 47
APROPOS-LIST 47
AREF 11
ARITHMETIC-ERROR 32
ARITHMETIC-ERROR-OPERANDS 30
ARITHMETIC-ERROR-OPERATION 30
ARRAY 32
ARRAY-DIMENSION 11
ARRAY-DIMENSION-LIMIT 12
ARRAY-DIMENSIONS 11
ARRAY-DISPLACEMENT 11
ARRAY-ELEMENT-TYPE 31
ARRAY-HAS-FILL-POINTER-P 11
ARRAY-IN-BOUNDS-P 11
ARRAY-RANK 11
ARRAY-RANK-LIMIT 12
ARRAY-ROW-MAJOR-INDEX 11
ARRAY-TOTAL-SIZE 11
ARRAY-TOTAL-SIZE-LIMIT 12
ARRAYP 11
AS 22
ASH 6
ASIN 3
ASINH 4
ASSERT 29
ASSOC 10
ASSOC-IF 10
ASSOC-IF-NOT 10
ATAN 3
ATANH 4
ATOM 9, 32

BASE-CHAR 32
BASE-STRING 32
BEING 24
BELOW 22
BIGNUM 32
BIT 12, 32
BIT-AND 12
BIT-ANDC1 12
BIT-ANDC2 12
BIT-EQV 12
BIT-IOR 12
BIT-NAND 12
BIT-NOR 12
BIT-NOT 12
BIT-ORC1 12
BIT-ORC2 12
BIT-VECTOR 32
BIT-VECTOR-P 11
BIT-XOR 12
BLOCK 21
BOOLE 5
BOOLE-1 5
BOOLE-2 5
BOOLE-AND 5
BOOLE-ANDC1 5
BOOLE-ANDC2 5
BOOLE-C1 5
BOOLE-C2 5
BOOLE-CLR 5
BOOLE-EQV 5
BOOLE-IOR 5
BOOLE-NAND 5
BOOLE-NOR 5
BOOLE-ORC1 5
BOOLE-ORC2 5
BOOLE-SET 5
BOOLE-XOR 5
BOOLEAN 32
BOTH-CASE-P 7
BOUNDP 16
BREAK 48
BROADCAST-STREAM 32
BROADCAST-STREAM-STREAMS 41
BUILT-IN-CLASS 32
BUTLAST 9
BY 22
BYTE 6
BYTE-POSITION 6
BYTE-SIZE 6

CAAR 9
CADR 9
CALL-ARGUMENTS-LIMIT 19
CALL-METHOD 28
CALL-NEXT-METHOD 27
CAR 9
CASE 21
CATCH 22
CCASE 21
CDAR 9
CDDR 9
CDR 9
CEILING 4
CELL-ERROR 32
CELL-ERROR-NAME 31
CERROR 29
CHANGE-CLASS 26
CHAR 8
CHAR-CODE 7
CHAR-CODE-LIMIT 7
CHAR-DOWNCASE 7
CHAR-EQUAL 7
CHAR-GREATERP 7
CHAR-INT 7
CHAR-LESSP 7
CHAR-NAME 7
CHAR-NOT-EQUAL 7
CHAR-NOT-GREATERP 7
CHAR-NOT-LESSP 7
CHAR-UPCASE 7
CHAR/= 7
CHAR< 7
CHAR<= 7
CHAR= 7
CHAR> 7
CHAR>= 7
CHARACTER 7, 32, 35
CHARACTERP 7
CHECK-TYPE 31
CIS 4
CL 46
CL-USER 46
CLASS 32
CLASS-NAME 25
CLASS-OF 25
CLEAR-INPUT 41
CLEAR-OUTPUT 41
CLOSE 41
CLQR 1
CLRHASH 15
CODE-CHAR 7
COERCE 31
COLLECT 24
COLLECTING 24
COMMON-LISP 46
COMMON-LISP-USER 46
COMPILATION-SPEED 49
COMPILE 46
COMPILE-FILE 46
COMPILE-FILE-PATHNAME 46
COMPILED-FUNCTION 32
COMPILED-FUNCTION-P 46
COMPILER-MACRO 45
COMPILER-MACRO-FUNCTION 47
COMPLEMENT 18
COMPLEX 4, 32, 35
COMPLEXP 3
COMPUTE-APPLICABLE-METHODS 27
COMPUTE-RESTARTS 30
CONCATENATE 13
CONCATENATED-STREAM 32
CONCATENATED-STREAM-STREAMS 41
COND 21
CONDITION 32
CONJUGATE 4
CONS 9, 32
CONSP 8
CONSTANTLY 19
CONSTANTP 17
CONTINUE 30
CONTROL-ERROR 32
COPY-ALIST 10
COPY-LIST 10
COPY-PPRINT-DISPATCH 38
COPY-READTABLE 34
COPY-SEQ 15
COPY-STRUCTURE 16
COPY-SYMBOL 45
COPY-TREE 11
COS 3
COSH 4
COUNT 13, 24
COUNT-IF 13
COUNT-IF-NOT 13
COUNTING 24
CTYPECASE 31

DEBUG 49
DECF 3
DECLAIM 48
DECLARATION 48
DECLARE 48
DECODE-FLOAT 6
DECODE-UNIVERSAL-TIME 49
DEFCLASS 25
DEFCONSTANT 17
DEFGENERIC 26
DEFINE-COMPILER-MACRO 19
DEFINE-CONDITION 29
DEFINE-METHOD-COMBINATION 28
DEFINE-MODIFY-MACRO 20
DEFINE-SETF-EXPANDER 20
DEFINE-SYMBOL-MACRO 19
DEFMACRO 19
DEFMETHOD 27
DEFPACKAGE 44
DEFPARAMETER 17
DEFSETF 20
DEFSTRUCT 16
DEFTYPE 33
DEFUN 18
DEFVAR 17
DELETE 14
DELETE-DUPLICATES 14
DELETE-FILE 43
DELETE-IF 14
DELETE-IF-NOT 14
DELETE-PACKAGE 44
DENOMINATOR 4
DEPOSIT-FIELD 6
DESCRIBE 48
DESCRIBE-OBJECT 48
DESTRUCTURING-BIND 18
DIGIT-CHAR 7

2 Characters

The **standard-char** type comprises `a-z`, `A-Z`, `0-9`, `Newline`, `Space`, and `!?$"'‘.:,;*+-/|\~_^<=>#%@&()[]{}`.

($_f$**characterp** *foo*)
($_f$**standard-char-p** *char*)
▷ T if argument is of indicated type.

($_f$**graphic-char-p** *character*)
($_f$**alpha-char-p** *character*)
($_f$**alphanumericp** *character*)
▷ T if *character* is visible, alphabetic, or alphanumeric, respectively.

($_f$**upper-case-p** *character*)
($_f$**lower-case-p** *character*)
($_f$**both-case-p** *character*)
▷ Return T if *character* is uppercase, lowercase, or able to be in another case, respectively.

($_f$**digit-char-p** *character* [*radix*$_{\boxed{10}}$])
▷ Return its weight if *character* is a digit, or NIL otherwise.

($_f$**char=** *character*$^+$)
($_f$**char/=** *character*$^+$)
▷ Return T if all *character*s, or none, respectively, are equal.

($_f$**char-equal** *character*$^+$)
($_f$**char-not-equal** *character*$^+$)
▷ Return T if all *character*s, or none, respectively, are equal ignoring case.

($_f$**char>** *character*$^+$)
($_f$**char>=** *character*$^+$)
($_f$**char<** *character*$^+$)
($_f$**char<=** *character*$^+$)
▷ Return T if *character*s are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively.

($_f$**char-greaterp** *character*$^+$)
($_f$**char-not-lessp** *character*$^+$)
($_f$**char-lessp** *character*$^+$)
($_f$**char-not-greaterp** *character*$^+$)
▷ Return T if *character*s are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively, ignoring case.

($_f$**char-upcase** *character*)
($_f$**char-downcase** *character*)
▷ Return corresponding uppercase/lowercase character, respectively.

($_f$**digit-char** *i* [*radix*$_{\boxed{10}}$]) ▷ Character representing digit *i*.

($_f$**char-name** *char*) ▷ *char*'s name if any, or NIL.

($_f$**name-char** *foo*) ▷ Character named *foo* if any, or NIL.

($_f$**char-int** *character*)
($_f$**char-code** *character*)
▷ Code of *character*.

($_f$**code-char** *code*) ▷ Character with *code*.

$_c$**char-code-limit** ▷ Upper bound of ($_f$**char-code** *char*); ≥ 96.

($_f$**character** *c*) ▷ Return #\c.

3 Strings

Strings can as well be manipulated by array and sequence functions; see pages 11 and 12.

$({}_f\textbf{stringp}\ \textit{foo})$
$({}_f\textbf{simple-string-p}\ \textit{foo})$
▷ $\underline{\texttt{T}}$ if *foo* is of indicated type.

$$\left(\begin{Bmatrix} {}_f\textbf{string=} \\ {}_f\textbf{string-equal} \end{Bmatrix} \textit{foo bar} \left\{\begin{vmatrix} \textbf{:start1}\ \textit{start-foo}_{\boxed{0}} \\ \textbf{:start2}\ \textit{start-bar}_{\boxed{0}} \\ \textbf{:end1}\ \textit{end-foo}_{\boxed{\texttt{NIL}}} \\ \textbf{:end2}\ \textit{end-bar}_{\boxed{\texttt{NIL}}} \end{vmatrix}\right\}\right)$$

▷ Return $\underline{\texttt{T}}$ if subsequences of *foo* and *bar* are equal. Obey/ignore, respectively, case.

$$\left(\begin{Bmatrix} {}_f\textbf{string}\{/= \mid \textbf{-not-equal}\} \\ {}_f\textbf{string}\{> \mid \textbf{-greaterp}\} \\ {}_f\textbf{string}\{>= \mid \textbf{-not-lessp}\} \\ {}_f\textbf{string}\{< \mid \textbf{-lessp}\} \\ {}_f\textbf{string}\{<= \mid \textbf{-not-greaterp}\} \end{Bmatrix} \textit{foo bar} \left\{\begin{vmatrix} \textbf{:start1}\ \textit{start-foo}_{\boxed{0}} \\ \textbf{:start2}\ \textit{start-bar}_{\boxed{0}} \\ \textbf{:end1}\ \textit{end-foo}_{\boxed{\texttt{NIL}}} \\ \textbf{:end2}\ \textit{end-bar}_{\boxed{\texttt{NIL}}} \end{vmatrix}\right\}\right)$$

▷ If *foo* is lexicographically not equal, greater, not less, less, or not greater, respectively, then return $\underline{\text{position}}$ of first mismatching character in *foo*. Otherwise return $\underline{\texttt{NIL}}$. Obey/ignore, respectively, case.

$$\left({}_f\textbf{make-string}\ \textit{size} \left\{\begin{vmatrix} \textbf{:initial-element}\ \textit{char} \\ \textbf{:element-type}\ \textit{type}_{\boxed{\textsf{character}}} \end{vmatrix}\right\}\right)$$

▷ Return $\underline{\text{string}}$ of length *size*.

$({}_f\textbf{string}\ x)$

$$\left(\begin{Bmatrix} {}_f\textbf{string-capitalize} \\ {}_f\textbf{string-upcase} \\ {}_f\textbf{string-downcase} \end{Bmatrix} x \left\{\begin{vmatrix} \textbf{:start}\ \textit{start}_{\boxed{0}} \\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}} \end{vmatrix}\right\}\right)$$

▷ Convert x (**symbol**, **string**, or **character**) into a $\underline{\text{string}}$, a $\underline{\text{string with capitalized words}}$, an $\underline{\text{all-uppercase string}}$, or an $\underline{\text{all-lowercase string}}$, respectively.

$$\left(\begin{Bmatrix} {}_f\textbf{nstring-capitalize} \\ {}_f\textbf{nstring-upcase} \\ {}_f\textbf{nstring-downcase} \end{Bmatrix} \widetilde{\textit{string}} \left\{\begin{vmatrix} \textbf{:start}\ \textit{start}_{\boxed{0}} \\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}} \end{vmatrix}\right\}\right)$$

▷ Convert *string* into a $\underline{\text{string with capitalized words}}$, an $\underline{\text{all-uppercase string}}$, or an $\underline{\text{all-lowercase string}}$, respectively.

$$\left(\begin{Bmatrix} {}_f\textbf{string-trim} \\ {}_f\textbf{string-left-trim} \\ {}_f\textbf{string-right-trim} \end{Bmatrix} \textit{char-bag string}\right)$$

▷ Return $\underline{\textit{string}}$ with all characters in sequence *char-bag* removed from both ends, from the beginning, or from the end, respectively.

$({}_f\textbf{char}\ \textit{string}\ i)$
$({}_f\textbf{schar}\ \textit{string}\ i)$
▷ Return zero-indexed $\underline{i\text{th character}}$ of string ignoring/obeying, respectively, fill pointer. **setf**able.

$$\left({}_f\textbf{parse-integer}\ \textit{string} \left\{\begin{vmatrix} \textbf{:start}\ \textit{start}_{\boxed{0}} \\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}} \\ \textbf{:radix}\ \textit{int}_{\boxed{10}} \\ \textbf{:junk-allowed}\ \textit{bool}_{\boxed{\texttt{NIL}}} \end{vmatrix}\right\}\right)$$

▷ Return $\underline{\text{integer}}$ parsed from *string* and $\underset{2}{\underline{\text{index}}}$ of parse end.

4 Conses

4.1 Predicates

$({}_f\textbf{consp}\ \textit{foo})$
$({}_f\textbf{listp}\ \textit{foo})$
▷ Return $\underline{\texttt{T}}$ if *foo* is of indicated type.

$({}_f\textbf{endp}\ \textit{list})$
$({}_f\textbf{null}\ \textit{foo})$
▷ Return $\underline{\texttt{T}}$ if *list*/*foo* is `NIL`.

$$\left(\textbf{optimize} \left\{\begin{vmatrix} \textbf{compilation-speed} \mid (\textbf{compilation-speed}\ n_{\boxed{3}}) \\ \textbf{debug} \mid (\textbf{debug}\ n_{\boxed{3}}) \\ \textbf{safety} \mid (\textbf{safety}\ n_{\boxed{3}}) \\ \textbf{space} \mid (\textbf{space}\ n_{\boxed{3}}) \\ \textbf{speed} \mid (\textbf{speed}\ n_{\boxed{3}}) \end{vmatrix}\right\}\right)$$

▷ Tell compiler how to optimize. $n = 0$ means unimportant, $n = 1$ is neutral, $n = 3$ means important.

$(\textbf{special}\ \textit{var}^*)$ ▷ Declare *var*s to be dynamic.

16 External Environment

$({}_f\textbf{get-internal-real-time})$
$({}_f\textbf{get-internal-run-time})$
▷ $\underline{\text{Current time}}$, or $\underline{\text{computing time}}$, respectively, in clock ticks.

${}_c\textbf{internal-time-units-per-second}$
▷ Number of clock ticks per second.

$({}_f\textbf{encode-universal-time}\ \textit{sec min hour date month year}\ [\textit{zone}_{\boxed{\text{curr}}}])$
$({}_f\textbf{get-universal-time})$
▷ $\underline{\text{Seconds from 1900-01-01, 00:00}}$, ignoring leap seconds.

$({}_f\textbf{decode-universal-time}\ \textit{universal-time}\ [\textit{time-zone}_{\boxed{\text{current}}}])$
$({}_f\textbf{get-decoded-time})$
▷ Return $\underline{\text{second}}$, $\underset{2}{\underline{\text{minute}}}$, $\underset{3}{\underline{\text{hour}}}$, $\underset{4}{\underline{\text{date}}}$, $\underset{5}{\underline{\text{month}}}$, $\underset{6}{\underline{\text{year}}}$, $\underset{7}{\underline{\text{day}}}$, $\underset{8}{\underline{\text{daylight-p}}}$, and $\underset{9}{\underline{\text{zone}}}$.

$({}_f\textbf{short-site-name})$
$({}_f\textbf{long-site-name})$
▷ $\underline{\text{String}}$ representing physical location of computer.

$$\left(\begin{Bmatrix} {}_f\textbf{lisp-implementation} \\ {}_f\textbf{software} \\ {}_f\textbf{machine} \end{Bmatrix}\text{-}\begin{Bmatrix} \textbf{type} \\ \textbf{version} \end{Bmatrix}\right)$$

▷ $\underline{\text{Name}}$ or $\underline{\text{version}}$ of implementation, operating system, or hardware, respectively.

$({}_f\textbf{machine-instance})$ ▷ $\underline{\text{Computer name}}$.

$_v$***macroexpand-hook***
▷ Function of arguments expansion function, macro form, and environment called by $_f$**macroexpand-1** to generate macro expansions.

($_m$**trace** $\left\{\begin{array}{l}\textit{function}\\(\textbf{setf}\ \textit{function})\end{array}\right\}^*$)
▷ Cause *function*s to be traced. With no arguments, return list of traced functions.

($_m$**untrace** $\left\{\begin{array}{l}\textit{function}\\(\textbf{setf}\ \textit{function})\end{array}\right\}^*$)
▷ Stop *function*s, or each currently traced function, from being traced.

$_v$***trace-output***
▷ Output stream $_m$**trace** and $_m$**time** send their output to.

($_m$**step** *form*)
▷ Step through evaluation of *form*. Return values of *form*.

($_f$**break** [*control arg**])
▷ Jump directly into debugger; return `NIL`. See page 38, $_f$**format**, for *control* and *arg*s.

($_m$**time** *form*)
▷ Evaluate *form*s and print timing information to $_v$***trace-output***. Return values of *form*.

($_f$**inspect** *foo*) ▷ Interactively give information about *foo*.

($_f$**describe** *foo* [$\widetilde{stream}_{\boxed{v\textbf{*standard-output*}}}$])
▷ Send information about *foo* to *stream*.

($_g$**describe-object** *foo* [$\widetilde{stream}$])
▷ Send information about *foo* to *stream*. Called by $_f$**describe**.

($_f$**disassemble** *function*)
▷ Send disassembled representation of *function* to $_v$***standard-output***. Return `NIL`.

($_f$**room** [{`NIL`|**:default**|`T`}$_{\boxed{\textbf{:default}}}$])
▷ Print information about internal storage management to ***standard-output***.

15.4 Declarations

($_f$**proclaim** *decl*)
($_m$**declaim** $\widehat{decl}^*$)
▷ Globally make declaration(s) *decl*. *decl* can be: **declaration**, **type**, **ftype**, **inline**, **notinline**, **optimize**, or **special**. See below.

(**declare** $\widehat{decl}^*$)
▷ Inside certain forms, locally make declarations *decl**. *decl* can be: **dynamic-extent**, **type**, **ftype**, **ignorable**, **ignore**, **inline**, **notinline**, **optimize**, or **special**. See below.

(**declaration** foo*)
▷ Make *foo*s names of declarations.

(**dynamic-extent** *variable** (**function** *function*)*)
▷ Declare lifetime of *variable*s and/or *function*s to end when control leaves enclosing block.

([**type**] *type variable**)
(**ftype** *type function**)
▷ Declare *variable*s or *function*s to be of *type*.

($\left\{\begin{array}{l}\textbf{ignorable}\\\textbf{ignore}\end{array}\right\}$ $\left\{\begin{array}{l}\textit{var}\\(\textbf{function}\ \textit{function})\end{array}\right\}^*$)
▷ Suppress warnings about used/unused bindings.

(**inline** *function**)
(**notinline** *function**)
▷ Tell compiler to integrate/not to integrate, respectively, called *function*s into the calling routine.

($_f$**atom** *foo*) ▷ Return `T` if *foo* is not a **cons**.

($_f$**tailp** *foo list*) ▷ Return `T` if *foo* is a tail of *list*.

($_f$**member** *foo list* $\left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\textbf{\#'eql}}}\\\textbf{:test-not}\ \textit{function}\end{array}\right\}\\\textbf{:key}\ \textit{function}\end{array}\right.\right\}$)
▷ Return tail of *list* starting with its first element matching *foo*. Return `NIL` if there is no such element.

($\left\{\begin{array}{l}{}_f\textbf{member-if}\\{}_f\textbf{member-if-not}\end{array}\right\}$ *test list* [**:key** *function*])
▷ Return tail of *list* starting with its first element satisfying *test*. Return `NIL` if there is no such element.

($_f$**subsetp** *list-a list-b* $\left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\textbf{\#'eql}}}\\\textbf{:test-not}\ \textit{function}\end{array}\right\}\\\textbf{:key}\ \textit{function}\end{array}\right.\right\}$)
▷ Return `T` if *list-a* is a subset of *list-b*.

4.2 Lists

($_f$**cons** *foo bar*) ▷ Return new cons (*foo* . *bar*).

($_f$**list** *foo**) ▷ Return list of *foo*s.

($_f$**list*** foo^+)
▷ Return list of *foo*s with last *foo* becoming cdr of last cons. Return *foo* if only one *foo* given.

($_f$**make-list** *num* [**:initial-element** $foo_{\boxed{\texttt{NIL}}}$])
▷ New list with *num* elements set to *foo*.

($_f$**list-length** *list*) ▷ Length of *list*; `NIL` for circular *list*.

($_f$**car** *list*) ▷ Car of *list* or `NIL` if *list* is `NIL`. **setf**able.

($_f$**cdr** *list*)
($_f$**rest** *list*) ▷ Cdr of *list* or `NIL` if *list* is `NIL`. **setf**able.

($_f$**nthcdr** *n list*) ▷ Return tail of *list* after calling $_f$**cdr** *n* times.

({$_f$**first**|$_f$**second**|$_f$**third**|$_f$**fourth**|$_f$**fifth**|$_f$**sixth**|...|$_f$**ninth**|$_f$**tenth**} *list*)
▷ Return nth element of *list* if any, or `NIL` otherwise. **setf**able.

($_f$**nth** *n list*) ▷ Zero-indexed *n*th element of *list*. **setf**able.

($_f$**c***X***r** *list*)
▷ With *X* being one to four **a**s and **d**s representing $_f$**car**s and $_f$**cdr**s, e.g. ($_f$**cadr** *bar*) is equivalent to ($_f$**car** ($_f$**cdr** *bar*)). **setf**able.

($_f$**last** *list* [$num_{\boxed{1}}$]) ▷ Return list of last *num* conses of *list*.

($\left\{\begin{array}{l}{}_f\textbf{butlast}\ \textit{list}\\{}_f\textbf{nbutlast}\ \widetilde{\textit{list}}\end{array}\right\}$ [$num_{\boxed{1}}$]) ▷ *list* excluding last *num* conses.

($\left\{\begin{array}{l}{}_f\textbf{rplaca}\\{}_f\textbf{rplacd}\end{array}\right\}$ $\widetilde{cons}$ *object*)
▷ Replace car, or cdr, respectively, of *cons* with *object*.

($_f$**ldiff** *list foo*)
▷ If *foo* is a tail of *list*, return preceding part of *list*. Otherwise return *list*.

($_f$**adjoin** *foo list* $\left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\textbf{\#'eql}}}\\\textbf{:test-not}\ \textit{function}\end{array}\right\}\\\textbf{:key}\ \textit{function}\end{array}\right.\right\}$)
▷ Return *list* if *foo* is already member of *list*. If not, return ($_f$**cons** *foo list*).

($_m$**pop** $\widetilde{place}$)
▷ Set *place* to ($_f$**cdr** *place*), return ($_f$**car** *place*).

($_m$**push** *foo* $\widetilde{\textit{place}}$) ▷ Set *place* to ($_f$**cons** *foo place*).

($_m$**pushnew** *foo* $\widetilde{\textit{place}}$ $\left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\#'\textbf{eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:key } \textit{function}\end{array}\right.\right\}$)
▷ Set *place* to ($_f$**adjoin** *foo place*).

($_f$**append** [*proper-list** $\textit{foo}_{\boxed{\texttt{NIL}}}$])
($_f$**nconc** [$\widetilde{\textit{non-circular-list}}$* $\textit{foo}_{\boxed{\texttt{NIL}}}$])
▷ Return concatenated list or, with only one argument, *foo*. *foo* can be of any type.

($_f$**revappend** *list foo*)
($_f$**nreconc** $\widetilde{\textit{list}}$ *foo*)
▷ Return concatenated list after reversing order in *list*.

($\left\{\begin{array}{l}{}_f\textbf{mapcar}\\ {}_f\textbf{maplist}\end{array}\right\}$ *function* $\textit{list}^{+}$)
▷ Return list of return values of *function* successively invoked with corresponding arguments, either cars or cdrs, respectively, from each *list*.

($\left\{\begin{array}{l}{}_f\textbf{mapcan}\\ {}_f\textbf{mapcon}\end{array}\right\}$ *function* $\widetilde{\textit{list}}^{+}$)
▷ Return list of concatenated return values of *function* successively invoked with corresponding arguments, either cars or cdrs, respectively, from each *list*. *function* should return a list.

($\left\{\begin{array}{l}{}_f\textbf{mapc}\\ {}_f\textbf{mapl}\end{array}\right\}$ *function* $\textit{list}^{+}$)
▷ Return first *list* after successively applying *function* to corresponding arguments, either cars or cdrs, respectively, from each *list*. *function* should have some side effects.

($_f$**copy-list** *list*) ▷ Return copy of *list* with shared elements.

4.3 Association Lists

($_f$**pairlis** *keys values* [$\textit{alist}_{\boxed{\texttt{NIL}}}$])
▷ Prepend to *alist* an association list made from lists *keys* and *values*.

($_f$**acons** *key value alist*)
▷ Return *alist* with a (*key* **.** *value*) pair added.

($\left\{\begin{array}{l}{}_f\textbf{assoc}\\ {}_f\textbf{rassoc}\end{array}\right\}$ *foo alist* $\left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test } \textit{test}_{\boxed{\#'\textbf{eql}}}\\ \textbf{:test-not } \textit{test}\end{array}\right.\\ \textbf{:key } \textit{function}\end{array}\right.\right\}$)
($\left\{\begin{array}{l}{}_f\textbf{assoc-if[-not]}\\ {}_f\textbf{rassoc-if[-not]}\end{array}\right\}$ *test alist* [**:key** *function*])
▷ First cons whose car, or cdr, respectively, satisfies *test*.

($_f$**copy-alist** *alist*) ▷ Return copy of *alist*.

4.4 Trees

($_f$**tree-equal** *foo bar* $\left\{\begin{array}{l}\textbf{:test } \textit{test}_{\boxed{\#'\textbf{eql}}}\\ \textbf{:test-not } \textit{test}\end{array}\right\}$)
▷ Return T if trees *foo* and *bar* have same shape and leaves satisfying *test*.

($\left\{\begin{array}{l}{}_f\textbf{subst } \textit{new old tree}\\ {}_f\textbf{nsubst } \textit{new old } \widetilde{\textit{tree}}\end{array}\right\}$ $\left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\#'\textbf{eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:key } \textit{function}\end{array}\right.\right\}$)
▷ Make copy of *tree* with each subtree or leaf matching *old* replaced by *new*.

($\left\{\begin{array}{l}{}_f\textbf{subst-if[-not] } \textit{new test tree}\\ {}_f\textbf{nsubst-if[-not] } \textit{new test } \widetilde{\textit{tree}}\end{array}\right\}$ [**:key** *function*])
▷ Make copy of *tree* with each subtree or leaf satisfying *test* replaced by *new*.

($_s$**eval-when** ($\left\{\left|\begin{array}{l}\{\textbf{:compile-toplevel}|\textbf{compile}\}\\ \{\textbf{:load-toplevel}|\textbf{load}\}\\ \{\textbf{:execute}|\textbf{eval}\}\end{array}\right.\right\}$) $\textit{form}^{P_*}$)
▷ Return values of *forms* if $_s$**eval-when** is in the top-level of a file being compiled, in the top-level of a compiled file being loaded, or anywhere, respectively. Return `NIL` if *forms* are not evaluated. (**compile**, **load** and **eval** deprecated.)

($_s$**locally** (**declare** $\widehat{\textit{decl}}$*)* $\textit{form}^{P_*}$)
▷ Evaluate *forms* in a lexical environment with declarations *decl* in effect. Return values of *forms*.

($_m$**with-compilation-unit** ([**:override** $\textit{bool}_{\boxed{\texttt{NIL}}}$]) $\textit{form}^{P_*}$)
▷ Return values of *forms*. Warnings deferred by the compiler until end of compilation are deferred until the end of evaluation of *forms*.

($_s$**load-time-value** *form* [$\widehat{\textit{read-only}}_{\boxed{\texttt{NIL}}}$])
▷ Evaluate *form* at compile time and treat its value as literal at run time.

($_s$**quote** $\widehat{\textit{foo}}$) ▷ Return unevaluated *foo*.

($_g$**make-load-form** *foo* [*environment*])
▷ Its methods are to return a creation form which on evaluation at $_f$**load** time returns an object equivalent to *foo*, and an optional $\underset{2}{\text{initialization form}}$ which on evaluation performs some initialization of the object.

($_f$**make-load-form-saving-slots** *foo* $\left\{\left|\begin{array}{l}\textbf{:slot-names } \textit{slots}_{\boxed{\text{all local slots}}}\\ \textbf{:environment } \textit{environment}\end{array}\right.\right\}$)
▷ Return a creation form and an $\underset{2}{\text{initialization form}}$ which on evaluation construct an object equivalent to *foo* with *slots* initialized with the corresponding values from *foo*.

($_f$**macro-function** *symbol* [*environment*])
($_f$**compiler-macro-function** $\left\{\begin{array}{l}\textit{name}\\ (\textbf{setf } \textit{name})\end{array}\right\}$ [*environment*])
▷ Return specified macro function, or compiler macro function, respectively, if any. Return `NIL` otherwise. **setf**able.

($_f$**eval** *arg*)
▷ Return values of value of *arg* evaluated in global environment.

15.3 REPL and Debugging

$_v$**+**|$_v$**++**|$_v$**+++**
$_v$***** | $_v$****** | $_v$*******
$_v$**/** | $_v$**//** | $_v$**///**
▷ Last, penultimate, or antepenultimate form evaluated in the REPL, or their respective primary value, or a list of their respective values.

$_v$**–** ▷ Form currently being evaluated by the REPL.

($_f$**apropos** *string* [$\textit{package}_{\boxed{\texttt{NIL}}}$])
▷ Print interned symbols containing *string*.

($_f$**apropos-list** *string* [$\textit{package}_{\boxed{\texttt{NIL}}}$])
▷ List of interned symbols containing *string*.

($_f$**dribble** [*path*])
▷ Save a record of interactive session to file at *path*. Without *path*, close that file.

($_f$**ed** [$\textit{file-or-function}_{\boxed{\texttt{NIL}}}$]) ▷ Invoke editor if possible.

($\left\{\begin{array}{l}{}_f\textbf{macroexpand-1}\\ {}_f\textbf{macroexpand}\end{array}\right\}$ *form* [$\textit{environment}_{\boxed{\texttt{NIL}}}$])
▷ Return macro expansion, once or entirely, respectively, of *form* and $\underset{2}{\texttt{T}}$ if *form* was a macro form. Return *form* and $\underset{2}{\texttt{NIL}}$ otherwise.

$_c$**t**

▷ Truth; the supertype of every type including **t**; the superclass of every class except **t**; $_v$***terminal-io***.

$_c$**nil**|$_c$**()**

▷ Falsity; the empty list; the empty type, subtype of every type; $_v$***standard-input***; $_v$***standard-output***; the global environment.

14.4 Standard Packages

common-lisp|**cl**

▷ Exports the defined names of Common Lisp except for those in the **keyword** package.

common-lisp-user|**cl-user**

▷ Current package after startup; uses package **common-lisp**.

keyword

▷ Contains symbols which are defined to be of type **keyword**.

15 Compiler

15.1 Predicates

($_f$**special-operator-p** *foo*) ▷ $\underline{\texttt{T}}$ if *foo* is a special operator.

($_f$**compiled-function-p** *foo*)

▷ $\underline{\texttt{T}}$ if *foo* is of type **compiled-function**.

15.2 Compilation

$$(_f\textbf{compile} \left\{\begin{array}{l}\texttt{NIL}\ \textit{definition}\\ \left\{\begin{array}{l}\textit{name}\\ (\textbf{setf}\ \textit{name})\end{array}\right\}\ [\textit{definition}]\end{array}\right\})$$

▷ Return $\underline{\text{compiled function}}$ or replace $\underline{\textit{name}}$'s function definition with the compiled function. Return $\underset{2}{\underline{\texttt{T}}}$ in case of **warning**s or **error**s, and $\underset{3}{\underline{\texttt{T}}}$ in case of **warning**s or **error**s excluding **style-warning**s.

$$(_f\textbf{compile-file}\ \textit{file} \left\{\begin{array}{|l}\textbf{:output-file}\ \textit{out-path}\\ \textbf{:verbose}\ \textit{bool}_{\boxed{_v\textbf{*compile-verbose*}}}\\ \textbf{:print}\ \textit{bool}_{\boxed{_v\textbf{*compile-print*}}}\\ \textbf{:external-format}\ \textit{file-format}_{\boxed{\textbf{:default}}}\end{array}\right\})$$

▷ Write compiled contents of *file* to *out-path*. Return $\underline{\text{true output path}}$ or $\underline{\texttt{NIL}}$, $\underset{2}{\underline{\texttt{T}}}$ in case of **warning**s or **error**s, $\underset{3}{\underline{\texttt{T}}}$ in case of **warning**s or **error**s excluding **style-warning**s.

($_f$**compile-file-pathname** *file* [**:output-file** *path*] [*other-keyargs*])

▷ $\underline{\text{Pathname}}$ $_f$**compile-file** writes to if invoked with the same arguments.

$$(_f\textbf{load}\ \textit{path} \left\{\begin{array}{|l}\textbf{:verbose}\ \textit{bool}_{\boxed{_v\textbf{*load-verbose*}}}\\ \textbf{:print}\ \textit{bool}_{\boxed{_v\textbf{*load-print*}}}\\ \textbf{:if-does-not-exist}\ \textit{bool}_{\boxed{\texttt{T}}}\\ \textbf{:external-format}\ \textit{file-format}_{\boxed{\textbf{:default}}}\end{array}\right\})$$

▷ Load source file or compiled file into Lisp environment. Return $\underline{\texttt{T}}$ if successful.

$$\left.\begin{array}{l}_v\textbf{*compile-file}\\ _v\textbf{*load}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{pathname*}_{\boxed{\texttt{NIL}}}\\ \textbf{truename*}_{\boxed{\texttt{NIL}}}\end{array}\right.$$

▷ Input file used by $_f$**compile-file**/by $_f$**load**.

$$\left.\begin{array}{l}_v\textbf{*compile}\\ _v\textbf{*load}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{print*}\\ \textbf{verbose*}\end{array}\right.$$

▷ Defaults used by $_f$**compile-file**/by $_f$**load**.

$$(\left\{\begin{array}{l}_f\textbf{sublis}\ \textit{association-list tree}\\ _f\textbf{nsublis}\ \textit{association-list}\ \widetilde{\textit{tree}}\end{array}\right\} \left\{\begin{array}{|l}\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\textbf{\#'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right\}\\ \textbf{:key}\ \textit{function}\end{array}\right\})$$

▷ Make $\underline{\text{copy of } \textit{tree}}$ with each subtree or leaf matching a key in *association-list* replaced by that key's value.

($_f$**copy-tree** *tree*) ▷ $\underline{\text{Copy of } \textit{tree}}$ with same shape and leaves.

4.5 Sets

$$(\left\{\begin{array}{l}\left.\begin{array}{l}_f\textbf{intersection}\\ _f\textbf{set-difference}\\ _f\textbf{union}\\ _f\textbf{set-exclusive-or}\end{array}\right\}\ a\ b\\ \left.\begin{array}{l}_f\textbf{nintersection}\\ _f\textbf{nset-difference}\end{array}\right\}\ \widetilde{a}\ b\\ \left.\begin{array}{l}_f\textbf{nunion}\\ _f\textbf{nset-exclusive-or}\end{array}\right\}\ \widetilde{a}\ \widetilde{b}\end{array}\right\} \left\{\begin{array}{|l}\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\textbf{\#'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right\}\\ \textbf{:key}\ \textit{function}\end{array}\right\})$$

▷ Return $\underline{a \cap b}$, $\underline{a \setminus b}$, $\underline{a \cup b}$, or $\underline{a \triangle b}$, respectively, of lists a and b.

5 Arrays

5.1 Predicates

($_f$**arrayp** *foo*)
($_f$**vectorp** *foo*)
($_f$**simple-vector-p** *foo*) ▷ $\underline{\texttt{T}}$ if *foo* is of indicated type.
($_f$**bit-vector-p** *foo*)
($_f$**simple-bit-vector-p** *foo*)

($_f$**adjustable-array-p** *array*)
($_f$**array-has-fill-pointer-p** *array*)

▷ $\underline{\texttt{T}}$ if *array* is adjustable/has a fill pointer, respectively.

($_f$**array-in-bounds-p** *array* [*subscripts*])

▷ Return $\underline{\texttt{T}}$ if *subscripts* are in *array*'s bounds.

5.2 Array Functions

$$(\left\{\begin{array}{l}_f\textbf{make-array}\ \textit{dimension-sizes}\ \big[\textbf{:adjustable}\ \textit{bool}_{\boxed{\texttt{NIL}}}\big]\\ _f\textbf{adjust-array}\ \widetilde{\textit{array}}\ \textit{dimension-sizes}\end{array}\right\}$$
$$\left\{\begin{array}{|l}\textbf{:element-type}\ \textit{type}_{\boxed{\texttt{T}}}\\ \textbf{:fill-pointer}\ \{\textit{num}|\textit{bool}\}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:initial-element}\ \textit{obj}\\ \textbf{:initial-contents}\ \textit{tree-or-array}\\ \textbf{:displaced-to}\ \textit{array}_{\boxed{\texttt{NIL}}}\ [\textbf{:displaced-index-offset}\ i_{\boxed{0}}]\end{array}\right.\end{array}\right\})$$

▷ Return fresh, or readjust, respectively, $\underline{\text{vector}}$ or $\underline{\text{array}}$.

($_f$**aref** *array* [*subscripts*])

▷ Return $\underline{\text{array element}}$ pointed to by *subscripts*. **setf**able.

($_f$**row-major-aref** *array* *i*)

▷ Return $\underline{i\text{th element}}$ of *array* in row-major order. **setf**able.

($_f$**array-row-major-index** *array* [*subscripts*])

▷ $\underline{\text{Index}}$ in row-major order of the element denoted by *subscripts*.

($_f$**array-dimensions** *array*)

▷ $\underline{\text{List}}$ containing the lengths of *array*'s dimensions.

($_f$**array-dimension** *array* *i*)

▷ $\underline{\text{Length of } i\text{th dimension}}$ of *array*.

($_f$**array-total-size** *array*) ▷ $\underline{\text{Number of elements}}$ in *array*.

($_f$**array-rank** *array*) ▷ $\underline{\text{Number of dimensions}}$ of *array*.

($_f$**array-displacement** *array*) ▷ $\underline{\text{Target array}}$ and $\underset{2}{\underline{\text{offset}}}$.

($_f$**bit** *bit-array* [*subscripts*])
($_f$**sbit** *simple-bit-array* [*subscripts*])

> ▷ Return element of *bit-array* or of *simple-bit-array*. **setf**able.

($_f$**bit-not** $\widetilde{\textit{bit-array}}$ [$\widetilde{\textit{result-bit-array}}_{\boxed{\texttt{NIL}}}$])

> ▷ Return result of bitwise negation of *bit-array*. If *result-bit-array* is `T`, put result in *bit-array*; if it is `NIL`, make a new array for result.

$\left(\left\{\begin{array}{l} {}_f\textbf{bit-eqv} \\ {}_f\textbf{bit-and} \\ {}_f\textbf{bit-andc1} \\ {}_f\textbf{bit-andc2} \\ {}_f\textbf{bit-nand} \\ {}_f\textbf{bit-ior} \\ {}_f\textbf{bit-orc1} \\ {}_f\textbf{bit-orc2} \\ {}_f\textbf{bit-xor} \\ {}_f\textbf{bit-nor} \end{array}\right\} \widetilde{\textit{bit-array-a}}\ \textit{bit-array-b}\ [\widetilde{\textit{result-bit-array}}_{\boxed{\texttt{NIL}}}]\right)$

> ▷ Return result of bitwise logical operations (cf. operations of $_f$**boole**, page 5) on *bit-array-a* and *bit-array-b*. If *result-bit-array* is `T`, put result in *bit-array-a*; if it is `NIL`, make a new array for result.

$_c$**array-rank-limit** ▷ Upper bound of array rank; ≥ 8.

$_c$**array-dimension-limit**

> ▷ Upper bound of an array dimension; ≥ 1024.

$_c$**array-total-size-limit** ▷ Upper bound of array size; ≥ 1024.

5.3 Vector Functions

Vectors can as well be manipulated by sequence functions; see section 6.

($_f$**vector** *foo**) ▷ Return fresh simple vector of *foo*s.

($_f$**svref** *vector i*) ▷ Element *i* of simple *vector*. **setf**able.

($_f$**vector-push** *foo* $\widetilde{\textit{vector}}$)

> ▷ Return `NIL` if *vector*'s fill pointer equals size of *vector*. Otherwise replace element of *vector* pointed to by fill pointer with *foo*; then increment fill pointer.

($_f$**vector-push-extend** *foo* $\widetilde{\textit{vector}}$ [*num*])

> ▷ Replace element of *vector* pointed to by fill pointer with *foo*, then increment fill pointer. Extend *vector*'s size by ≥ *num* if necessary.

($_f$**vector-pop** $\widetilde{\textit{vector}}$)

> ▷ Return element of *vector* its fillpointer points to after decrementation.

($_f$**fill-pointer** *vector*) ▷ Fill pointer of *vector*. **setf**able.

6 Sequences

6.1 Sequence Predicates

$\left(\left\{\begin{array}{l} {}_f\textbf{every} \\ {}_f\textbf{notevery} \end{array}\right\} \textit{test sequence}^+\right)$

> ▷ Return `NIL` or `T`, respectively, as soon as *test* on any set of corresponding elements of *sequence*s returns `NIL`.

$\left(\left\{\begin{array}{l} {}_f\textbf{some} \\ {}_f\textbf{notany} \end{array}\right\} \textit{test sequence}^+\right)$

> ▷ Return value of *test* or `NIL`, respectively, as soon as *test* on any set of corresponding elements of *sequence*s returns non-`NIL`.

($_f$**package-shadowing-symbols** *package*)

> ▷ List of symbols of *package* that shadow any otherwise accessible, equally named symbols from other packages.

($_f$**export** *symbols* [*package*$_{\boxed{{}_v\textbf{*package*}}}$])

> ▷ Make *symbols* external to *package*. Return `T`.

($_f$**unexport** *symbols* [*package*$_{\boxed{{}_v\textbf{*package*}}}$])

> ▷ Revert *symbols* to internal status. Return `T`.

$\left(\left\{\begin{array}{l} \left.\begin{array}{l} {}_m\textbf{do-symbols} \\ {}_m\textbf{do-external-symbols} \end{array}\right\} (\widehat{\textit{var}}\ [\textit{package}_{\boxed{{}_v\textbf{*package*}}}\ [\textit{result}_{\boxed{\texttt{NIL}}}]]) \\ {}_m\textbf{do-all-symbols}\ (\textit{var}\ [\textit{result}_{\boxed{\texttt{NIL}}}]) \end{array}\right\}\right.$
$(\textbf{declare}\ \widehat{\textit{decl}^*})^*\ \left\{\begin{array}{l} \widehat{\textit{tag}} \\ \textit{form} \end{array}\right\}^*\Big)$

> ▷ Evaluate $_s$**tagbody**-like body with *var* successively bound to every symbol from *package*, to every external symbol from *package*, or to every symbol from all registered packages, respectively. Return values of *result*. Implicitly, the whole form is a $_s$**block** named `NIL`.

($_m$**with-package-iterator** (*foo packages* [**:internal**|**:external**|**:inherited**])
(**declare** $\widehat{\textit{decl}^*}$)* $\textit{form}^{P_*}$)

> ▷ Return values of *forms*. In *forms*, successive invocations of (*foo*) return: `T` if a symbol is returned; a symbol from *packages*; accessibility (**:internal**, **:external**, or **:inherited**); and the package the symbol belongs to.

($_f$**require** *module* [*paths*$_{\boxed{\texttt{NIL}}}$])

> ▷ If not in $_v$***modules***, try *paths* to load *module* from. Signal **error** if unsuccessful. Deprecated.

($_f$**provide** *module*)

> ▷ If not already there, add *module* to $_v$***modules***. Deprecated.

$_v$***modules*** ▷ List of names of loaded modules.

14.3 Symbols

A **symbol** has the attributes *name*, home **package**, property list, and optionally value (of global constant or variable *name*) and function (**function**, macro, or special operator *name*).

($_f$**make-symbol** *name*)

> ▷ Make fresh, uninterned symbol *name*.

($_f$**gensym** [$s_{\boxed{\text{G}}}$])

> ▷ Return fresh, uninterned symbol **#:**sn with n from $_v$***gensym-counter***. Increment $_v$***gensym-counter***.

($_f$**gentemp** [*prefix*$_{\boxed{\text{T}}}$ [*package*$_{\boxed{{}_v\textbf{*package*}}}$]])

> ▷ Intern fresh symbol in package. Deprecated.

($_f$**copy-symbol** *symbol* [*props*$_{\boxed{\texttt{NIL}}}$])

> ▷ Return uninterned copy of *symbol*. If *props* is `T`, give copy the same value, function and property list.

($_f$**symbol-name** *symbol*)
($_f$**symbol-package** *symbol*)
($_f$**symbol-plist** *symbol*)
($_f$**symbol-value** *symbol*)
($_f$**symbol-function** *symbol*)

> ▷ Name, package, property list, value, or function, respectively, of *symbol*. **setf**able.

$\left(\left\{\begin{array}{l} {}_g\textbf{documentation} \\ (\textbf{setf}\ {}_g\textbf{documentation})\ \textit{new-doc} \end{array}\right\} \textit{foo} \left\{\begin{array}{l} \textbf{'variable}|\textbf{'function} \\ \textbf{'compiler-macro} \\ \textbf{'method-combination} \\ \textbf{'structure}|\textbf{'type}|\textbf{'setf}|\texttt{T} \end{array}\right\}\right)$

> ▷ Get/set documentation string of *foo* of given type.

14.2 Packages

:*bar* | **keyword:***bar* ▷ Keyword, evaluates to :*bar*.

package:*symbol* ▷ Exported *symbol* of *package*.

package::*symbol* ▷ Possibly unexported *symbol* of *package*.

($_m$**defpackage** *foo* {|(**:nicknames** *nick**)* | (**:documentation** *string*) | (**:intern** *interned-symbol**)* | (**:use** *used-package**)* | (**:import-from** *pkg imported-symbol**)* | (**:shadowing-import-from** *pkg shd-symbol**)* | (**:shadow** *shd-symbol**)* | (**:export** *exported-symbol**)* | (**:size** *int*)})

▷ Create or modify package *foo* with *interned-symbols*, symbols from *used-packages*, *imported-symbols*, and *shd-symbols*. Add *shd-symbols* to *foo*'s shadowing list.

($_f$**make-package** *foo* {|**:nicknames** (*nick**)$_{\boxed{\texttt{NIL}}}$ | **:use** (*used-package**)})

▷ Create package *foo*.

($_f$**rename-package** *package new-name* [*new-nicknames*$_{\boxed{\texttt{NIL}}}$])

▷ Rename *package*. Return renamed package.

($_m$**in-package** $\widehat{foo}$) ▷ Make package *foo* current.

({$_f$**use-package** | $_f$**unuse-package**} *other-packages* [*package*$_{\boxed{_v\textbf{*package*}}}$])

▷ Make exported symbols of *other-packages* available in *package*, or remove them from *package*, respectively. Return T.

($_f$**package-use-list** *package*)
($_f$**package-used-by-list** *package*)

▷ List of other packages used by/using *package*.

($_f$**delete-package** $\widetilde{package}$)

▷ Delete *package*. Return T if successful.

$_v$***package***$_{\boxed{\textbf{common-lisp-user}}}$ ▷ The current package.

($_f$**list-all-packages**) ▷ List of registered packages.

($_f$**package-name** *package*) ▷ Name of *package*.

($_f$**package-nicknames** *package*) ▷ Nicknames of *package*.

($_f$**find-package** *name*) ▷ Package with *name* (case-sensitive).

($_f$**find-all-symbols** *foo*)

▷ List of symbols *foo* from all registered packages.

({$_f$**intern** | $_f$**find-symbol**} *foo* [*package*$_{\boxed{_v\textbf{*package*}}}$])

▷ Intern or find, respectively, symbol *foo* in *package*. Second return value is one of $\underset{2}{\textbf{:internal}}$, $\underset{2}{\textbf{:external}}$, or $\underset{2}{\textbf{:inherited}}$ (or $\underset{2}{\texttt{NIL}}$ if $_f$**intern** has created a fresh symbol).

($_f$**unintern** *symbol* [*package*$_{\boxed{_v\textbf{*package*}}}$])

▷ Remove *symbol* from *package*, return T on success.

({$_f$**import** | $_f$**shadowing-import**} *symbols* [*package*$_{\boxed{_v\textbf{*package*}}}$])

▷ Make *symbols* internal to *package*. Return T. In case of a name conflict signal correctable **package-error** or shadow the old symbol, respectively.

($_f$**shadow** *symbols* [*package*$_{\boxed{_v\textbf{*package*}}}$])

▷ Make *symbols* of *package* shadow any otherwise accessible, equally named symbols from other packages. Return T.

($_f$**mismatch** *sequence-a sequence-b* {|**:from-end** *bool*$_{\boxed{\texttt{NIL}}}$ | {**:test** *function*$_{\boxed{\textbf{\#'eql}}}$ | **:test-not** *function*} | **:start1** *start-a*$_{\boxed{0}}$ | **:start2** *start-b*$_{\boxed{0}}$ | **:end1** *end-a*$_{\boxed{\texttt{NIL}}}$ | **:end2** *end-b*$_{\boxed{\texttt{NIL}}}$ | **:key** *function*})

▷ Return position in *sequence-a* where *sequence-a* and *sequence-b* begin to mismatch. Return NIL if they match entirely.

6.2 Sequence Functions

($_f$**make-sequence** *sequence-type size* [**:initial-element** *foo*])

▷ Make sequence of *sequence-type* with *size* elements.

($_f$**concatenate** *type sequence**)

▷ Return concatenated sequence of *type*.

($_f$**merge** *type* $\widetilde{sequence}$*-a* $\widetilde{sequence}$*-b test* [**:key** *function*$_{\boxed{\texttt{NIL}}}$])

▷ Return interleaved sequence of *type*. Merged sequence will be sorted if both *sequence-a* and *sequence-b* are sorted.

($_f$**fill** $\widetilde{sequence}$ *foo* {|**:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\texttt{NIL}}}$})

▷ Return *sequence* after setting elements between *start* and *end* to *foo*.

($_f$**length** *sequence*)

▷ Return length of *sequence* (being value of fill pointer if applicable).

($_f$**count** *foo sequence* {|**:from-end** *bool*$_{\boxed{\texttt{NIL}}}$ | {**:test** *function*$_{\boxed{\textbf{\#'eql}}}$ | **:test-not** *function*} | **:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\texttt{NIL}}}$ | **:key** *function*})

▷ Return number of elements in *sequence* which match *foo*.

({$_f$**count-if** | $_f$**count-if-not**} *test sequence* {|**:from-end** *bool*$_{\boxed{\texttt{NIL}}}$ | **:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\texttt{NIL}}}$ | **:key** *function*})

▷ Return number of elements in *sequence* which satisfy *test*.

($_f$**elt** *sequence index*)

▷ Return element of *sequence* pointed to by zero-indexed *index*. **setf**able.

($_f$**subseq** *sequence start* [*end*$_{\boxed{\texttt{NIL}}}$])

▷ Return subsequence of *sequence* between *start* and *end*. **setf**able.

({$_f$**sort** | $_f$**stable-sort**} $\widetilde{sequence}$ *test* [**:key** *function*])

▷ Return *sequence* sorted. Order of elements considered equal is not guaranteed/retained, respectively.

($_f$**reverse** *sequence*)
($_f$**nreverse** $\widetilde{sequence}$) ▷ Return *sequence* in reverse order.

({$_f$**find** | $_f$**position**} *foo sequence* {|**:from-end** *bool*$_{\boxed{\texttt{NIL}}}$ | {**:test** *function*$_{\boxed{\textbf{\#'eql}}}$ | **:test-not** *test*} | **:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\texttt{NIL}}}$ | **:key** *function*})

▷ Return first element in *sequence* which matches *foo*, or its position relative to the begin of *sequence*, respectively.

$$\left(\left\{\begin{array}{l} {}_f\textbf{find-if}\\ {}_f\textbf{find-if-not}\\ {}_f\textbf{position-if}\\ {}_f\textbf{position-if-not}\end{array}\right\} \textit{test sequence} \left\{\begin{array}{|l} \textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\end{array}\right\}\right)$$

▷ Return first element in *sequence* which satisfies *test*, or its position relative to the begin of *sequence*, respectively.

$$\left({}_f\textbf{search } \textit{sequence-a sequence-b} \left\{\begin{array}{|l} \textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:start1 } \textit{start-a}_{\boxed{0}}\\ \textbf{:start2 } \textit{start-b}_{\boxed{0}}\\ \textbf{:end1 } \textit{end-a}_{\boxed{\texttt{NIL}}}\\ \textbf{:end2 } \textit{end-b}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\end{array}\right\}\right)$$

▷ Search *sequence-b* for a subsequence matching *sequence-a*. Return position in *sequence-b*, or `NIL`.

$$\left(\left\{\begin{array}{l} {}_f\textbf{remove } \textit{foo sequence}\\ {}_f\textbf{delete } \textit{foo } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\begin{array}{|l} \textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\\ \textbf{:count } \textit{count}_{\boxed{\texttt{NIL}}}\end{array}\right\}\right)$$

▷ Make copy of *sequence* without elements matching *foo*.

$$\left(\left\{\begin{array}{l} \left.\begin{array}{l}{}_f\textbf{remove-if}\\ {}_f\textbf{remove-if-not}\end{array}\right\} \textit{test sequence}\\ \left.\begin{array}{l}{}_f\textbf{delete-if}\\ {}_f\textbf{delete-if-not}\end{array}\right\} \textit{test } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\begin{array}{|l} \textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\\ \textbf{:count } \textit{count}_{\boxed{\texttt{NIL}}}\end{array}\right\}\right)$$

▷ Make copy of *sequence* with all (or *count*) elements satisfying *test* removed.

$$\left(\left\{\begin{array}{l} {}_f\textbf{remove-duplicates } \textit{sequence}\\ {}_f\textbf{delete-duplicates } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\begin{array}{|l} \textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\end{array}\right\}\right)$$

▷ Make copy of *sequence* without duplicates.

$$\left(\left\{\begin{array}{l} {}_f\textbf{substitute } \textit{new old sequence}\\ {}_f\textbf{nsubstitute } \textit{new old } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\begin{array}{|l} \textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\\ \textbf{:count } \textit{count}_{\boxed{\texttt{NIL}}}\end{array}\right\}\right)$$

▷ Make copy of *sequence* with all (or *count*) *old*s replaced by *new*.

$$\left(\left\{\begin{array}{l} \left.\begin{array}{l}{}_f\textbf{substitute-if}\\ {}_f\textbf{substitute-if-not}\end{array}\right\} \textit{new test sequence}\\ \left.\begin{array}{l}{}_f\textbf{nsubstitute-if}\\ {}_f\textbf{nsubstitute-if-not}\end{array}\right\} \textit{new test } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\begin{array}{|l} \textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\\ \textbf{:count } \textit{count}_{\boxed{\texttt{NIL}}}\end{array}\right\}\right)$$

▷ Make copy of *sequence* with all (or *count*) elements satisfying *test* replaced by *new*.

$$\left({}_f\textbf{replace } \widetilde{\textit{sequence}}\textit{-a sequence-b} \left\{\begin{array}{|l} \textbf{:start1 } \textit{start-a}_{\boxed{0}}\\ \textbf{:start2 } \textit{start-b}_{\boxed{0}}\\ \textbf{:end1 } \textit{end-a}_{\boxed{\texttt{NIL}}}\\ \textbf{:end2 } \textit{end-b}_{\boxed{\texttt{NIL}}}\end{array}\right\}\right)$$

▷ Replace elements of *sequence-a* with elements of *sequence-b*.

(${}_f$**map** *type function sequence*$^+$)

▷ Apply *function* successively to corresponding elements of the *sequence*s. Return values as a sequence of *type*. If *type* is `NIL`, return `NIL`.

(${}_f$**enough-namestring** *path-or-stream* [*root-path*$_{\boxed{{}_v\textbf{*default-pathname-defaults*}}}$])

▷ Return minimal path string that sufficiently describes the path of *path-or-stream* relative to *root-path*.

(${}_f$**namestring** *path-or-stream*)
(${}_f$**file-namestring** *path-or-stream*)
(${}_f$**directory-namestring** *path-or-stream*)
(${}_f$**host-namestring** *path-or-stream*)

▷ Return string representing full pathname; name, type, and version; directory name; or host name, respectively, of *path-or-stream*.

(${}_f$**translate-pathname** *path-or-stream wildcard-path-a wildcard-path-b*)

▷ Translate the path of *path-or-stream* from *wildcard-path-a* into *wildcard-path-b*. Return new path.

(${}_f$**pathname** *path-or-stream*) ▷ Pathname of *path-or-stream*.

(${}_f$**logical-pathname** *logical-path-or-stream*)

▷ Logical pathname of *logical-path-or-stream*. Logical pathnames are represented as all-uppercase

$$\texttt{"}[\textit{host}\texttt{:}][\texttt{;}]\left\{\left\{\begin{array}{l}\{\textit{dir}|\texttt{*}\}^+\\ \texttt{**}\end{array}\right\}\texttt{;}\right\}^*\{\textit{name}|\texttt{*}\}^*\left[\texttt{.}\left\{\begin{array}{l}\{\textit{type}|\texttt{*}\}^+\\ \texttt{LISP}\end{array}\right\}\right.$$
$$\left.[\texttt{.}\{\textit{version}|\texttt{*}|\texttt{newest}|\texttt{NEWEST}\}]\right]\texttt{"}.$$

(${}_f$**logical-pathname-translations** *logical-host*)

▷ List of (*from-wildcard to-wildcard*) translations for *logical-host*. **setf**able.

(${}_f$**load-logical-pathname-translations** *logical-host*)

▷ Load *logical-host*'s translations. Return `NIL` if already loaded; return `T` if successful.

(${}_f$**translate-logical-pathname** *path-or-stream*)

▷ Physical pathname corresponding to (possibly logical) pathname of *path-or-stream*.

(${}_f$**probe-file** *file*)
(${}_f$**truename** *file*)

▷ Canonical name of *file*. If *file* does not exist, return `NIL`/signal **file-error**, respectively.

(${}_f$**file-write-date** *file*) ▷ Time at which *file* was last written.

(${}_f$**file-author** *file*) ▷ Return name of *file* owner.

(${}_f$**file-length** *stream*) ▷ Return length of *stream*.

(${}_f$**rename-file** *foo bar*)

▷ Rename file *foo* to *bar*. Unspecified components of path *bar* default to those of *foo*. Return new pathname, $\underset{2}{\text{old physical file name}}$, and $\underset{3}{\text{new physical file name}}$.

(${}_f$**delete-file** *file*) ▷ Delete *file*. Return `T`.

(${}_f$**directory** *path*) ▷ List of pathnames matching *path*.

(${}_f$**ensure-directories-exist** *path* [**:verbose** *bool*])

▷ Create parts of *path* if necessary. Second return value is $\underset{2}{\texttt{T}}$ if something has been created.

14 Packages and Symbols

The Loop Facility provides additional means of symbol handling; see **loop**, page 22.

14.1 Predicates

(${}_f$**symbolp** *foo*)
(${}_f$**packagep** *foo*) ▷ `T` if *foo* is of indicated type.
(${}_f$**keywordp** *foo*)

(${}_m$**with-output-to-string** (*foo* [$\widetilde{string}_{\boxed{\texttt{NIL}}}$ [**:element-type** $type_{\boxed{\textsf{character}}}$]])
(**declare** $\widehat{decl}^*$)* $form^{P_*}$)
▷ Evaluate *forms* with *foo* locally bound to an output **string-stream**. Append output to *string* and return values of forms if *string* is given. Return string containing output otherwise.

(${}_f$**stream-external-format** *stream*)
▷ External file format designator.

${}_v$***terminal-io*** ▷ Bidirectional stream to user terminal.

${}_v$***standard-input***
${}_v$***standard-output***
${}_v$***error-output***
▷ Standard input stream, standard output stream, or standard error output stream, respectively.

${}_v$***debug-io***
${}_v$***query-io***
▷ Bidirectional streams for debugging and user interaction.

13.7 Pathnames and Files

(${}_f$**make-pathname**

$$\left\{\begin{array}{|l}
\textbf{:host } \{host|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:device } \{device|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:directory } \left\{\begin{array}{l} \{directory|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\} \\ \left(\left\{\begin{array}{l}\textbf{:absolute}\\ \textbf{:relative}\end{array}\right\} \left\{\begin{array}{l} directory\\ \textbf{:wild}\\ \textbf{:wild-inferiors}\\ \textbf{:up}\\ \textbf{:back}\end{array}\right\}^* \right)\end{array}\right. \\
\textbf{:name } \{file\text{-}name|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:type } \{file\text{-}type|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:version } \{\textbf{:newest}|version|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:defaults } path_{\boxed{\text{host from } {}_v\textbf{*default-pathname-defaults*}}} \\
\textbf{:case } \{\textbf{:local}|\textbf{:common}\}_{\boxed{\textbf{:local}}}
\end{array}\right\})$$

▷ Construct a logical pathname if there is a logical pathname translation for *host*, otherwise construct a physical pathname. For **:case :local**, leave case of components unchanged. For **:case :common**, leave mixed-case components unchanged; convert all-uppercase components into local customary case; do the opposite with all-lowercase components.

$$\left(\left\{\begin{array}{l} {}_f\textbf{pathname-host} \\ {}_f\textbf{pathname-device} \\ {}_f\textbf{pathname-directory} \\ {}_f\textbf{pathname-name} \\ {}_f\textbf{pathname-type} \end{array}\right\} path\text{-}or\text{-}stream\ \left[\textbf{:case} \left\{\begin{array}{l}\textbf{:local}\\ \textbf{:common}\end{array}\right\}_{\boxed{\textbf{:local}}}\right]\right)$$

(${}_f$**pathname-version** *path-or-stream*)
▷ Return pathname component.

(${}_f$**parse-namestring** *foo* [*host*
[$default\text{-}pathname_{\boxed{{}_v\textbf{*default-pathname-defaults*}}}$

$$\left\{\begin{array}{|l} \textbf{:start } start_{\boxed{0}} \\ \textbf{:end } end_{\boxed{\texttt{NIL}}} \\ \textbf{:junk-allowed } bool_{\boxed{\texttt{NIL}}} \end{array}\right\}]])$$

▷ Return pathname converted from string, pathname, or stream *foo*; and $\underset{2}{\text{position}}$ where parsing stopped.

(${}_f$**merge-pathnames** *path-or-stream*
[$default\text{-}path\text{-}or\text{-}stream_{\boxed{{}_v\textbf{*default-pathname-defaults*}}}$
[$default\text{-}version_{\boxed{\textbf{:newest}}}$]])
▷ Return pathname made by filling in components missing in *path-or-stream* from *default-path-or-stream*.

${}_v$***default-pathname-defaults***
▷ Pathname to use if one is needed and none supplied.

(${}_f$**user-homedir-pathname** [*host*]) ▷ User's home directory.

(${}_f$**map-into** $\widetilde{result\text{-}sequence}$ *function* *sequence**)
▷ Store into *result-sequence* successively values of *function* applied to corresponding elements of the *sequences*.

$$({}_f\textbf{reduce } function\ sequence \left\{\begin{array}{|l} \textbf{:initial-value } foo_{\boxed{\texttt{NIL}}} \\ \textbf{:from-end } bool_{\boxed{\texttt{NIL}}} \\ \textbf{:start } start_{\boxed{0}} \\ \textbf{:end } end_{\boxed{\texttt{NIL}}} \\ \textbf{:key } function \end{array}\right\})$$

▷ Starting with the first two elements of *sequence*, apply *function* successively to its last return value together with the next element of *sequence*. Return last value of function.

(${}_f$**copy-seq** *sequence*)
▷ Copy of *sequence* with shared elements.

7 Hash Tables

The Loop Facility provides additional hash table-related functionality; see **loop**, page 22.

Key-value storage similar to hash tables can as well be achieved using association lists and property lists; see pages 10 and 17.

(${}_f$**hash-table-p** *foo*) ▷ Return `T` if *foo* is of type **hash-table**.

$$({}_f\textbf{make-hash-table} \left\{\begin{array}{|l} \textbf{:test } \{{}_f\textbf{eq}|{}_f\textbf{eql}|{}_f\textbf{equal}|{}_f\textbf{equalp}\}_{\boxed{\textbf{\#'eql}}} \\ \textbf{:size } int \\ \textbf{:rehash-size } num \\ \textbf{:rehash-threshold } num \end{array}\right\})$$

▷ Make a hash table.

(${}_f$**gethash** *key* *hash-table* [$default_{\boxed{\texttt{NIL}}}$])
▷ Return object with *key* if any or *default* otherwise; and $\underset{2}{\texttt{T}}$ if found, $\underset{2}{\texttt{NIL}}$ otherwise. **setf**able.

(${}_f$**hash-table-count** *hash-table*)
▷ Number of entries in *hash-table*.

(${}_f$**remhash** *key* $\widetilde{hash\text{-}table}$)
▷ Remove from *hash-table* entry with *key* and return `T` if it existed. Return `NIL` otherwise.

(${}_f$**clrhash** $\widetilde{hash\text{-}table}$) ▷ Empty *hash-table*.

(${}_f$**maphash** *function* *hash-table*)
▷ Iterate over *hash-table* calling *function* on key and value. Return `NIL`.

(${}_m$**with-hash-table-iterator** (*foo* *hash-table*) (**declare** $\widehat{decl}^*$)* $form^{P_*}$)
▷ Return values of forms. In *forms*, invocations of (*foo*) return: `T` if an entry is returned; its key; its value.

(${}_f$**hash-table-test** *hash-table*)
▷ Test function used in *hash-table*.

(${}_f$**hash-table-size** *hash-table*)
(${}_f$**hash-table-rehash-size** *hash-table*)
(${}_f$**hash-table-rehash-threshold** *hash-table*)
▷ Current size, rehash-size, or rehash-threshold, respectively, as used in ${}_f$**make-hash-table**.

(${}_f$**sxhash** *foo*)
▷ Hash code unique for any argument ${}_f$**equal** *foo*.

8 Structures

$$(_m\textbf{defstruct}\ \left\{\begin{array}{l} foo \\ (foo \left\{\begin{array}{|l} \left\{\begin{array}{l}\textbf{:conc-name}\\ (\textbf{:conc-name}\ [\widehat{slot\text{-}prefix}_{\boxed{foo\text{-}}}])\end{array}\right. \\ \left\{\begin{array}{l}\textbf{:constructor}\\ (\textbf{:constructor}\ [\widehat{maker}_{\boxed{\texttt{MAKE-}foo}}\ [(\widehat{ord\text{-}\lambda}^*)]])\end{array}\right\}^* \\ \left\{\begin{array}{l}\textbf{:copier}\\ (\textbf{:copier}\ [\widehat{copier}_{\boxed{\texttt{COPY-}foo}}])\end{array}\right. \\ (\textbf{:include}\ \widehat{struct} \left\{\begin{array}{l}\widehat{slot}\\ (\widehat{slot}\ [init \left\{\begin{array}{|l}\textbf{:type}\ \widehat{sl\text{-}type}\\ \textbf{:read-only}\ \widehat{b}\end{array}\right\}])\end{array}\right\}^*) \\ \left\{\begin{array}{l}(\textbf{:type} \left\{\begin{array}{l}\textbf{list}\\ \textbf{vector}\\ (\textbf{vector}\ \widehat{type})\end{array}\right\}) \left\{\begin{array}{|l}\textbf{:named}\\ (\textbf{:initial-offset}\ \widehat{n})\end{array}\right. \\ \left\{\begin{array}{l}(\textbf{:print-object}\ [\widehat{o\text{-}printer}])\\ (\textbf{:print-function}\ [\widehat{f\text{-}printer}])\end{array}\right.\end{array}\right. \\ \left\{\begin{array}{l}\textbf{:predicate}\\ (\textbf{:predicate}\ [\widehat{p\text{-}name}_{\boxed{foo\text{-}\texttt{P}}}])\end{array}\right. \end{array}\right\})\end{array}\right\}$$

$$[\widehat{doc}] \left\{\begin{array}{l}slot\\ (slot\ [init \left\{\begin{array}{|l}\textbf{:type}\ \widehat{slot\text{-}type}\\ \textbf{:read-only}\ \widehat{bool}\end{array}\right\}])\end{array}\right\}^*)$$

▷ Define structure foo together with functions `MAKE-`*foo*, `COPY-`*foo* and *foo*`-P`; and **setf**able accessors *foo*`-`*slot*. Instances are of class *foo* or, if **defstruct** option **:type** is given, of the specified type. They can be created by (`MAKE-`*foo* $\{$:*slot value*$\}^*$) or, if *ord-λ* (see page 18) is given, by (*maker* arg^* $\{$:*key value*$\}^*$). In the latter case, *arg*s and **:***key*s correspond to the positional and keyword parameters defined in *ord-λ* whose *var*s in turn correspond to *slot*s. **:print-object**/**:print-function** generate a $_g$**print-object** method for an instance *bar* of *foo* calling (*o-printer bar stream*) or (*f-printer bar stream print-level*), respectively. If **:type** without **:named** is given, no *foo*`-P` is created.

($_f$**copy-structure** *structure*)
▷ Return copy of *structure* with shared slot values.

9 Control Structure

9.1 Predicates

($_f$**eq** *foo bar*) ▷ `T` if *foo* and *bar* are identical.

($_f$**eql** *foo bar*)
▷ `T` if *foo* and *bar* are identical, or the same **character**, or **number**s of the same type and value.

($_f$**equal** *foo bar*)
▷ `T` if *foo* and *bar* are $_f$**eql**, or are equivalent **pathname**s, or are **cons**es with $_f$**equal** cars and cdrs, or are **string**s or **bit-vector**s with $_f$**eql** elements below their fill pointers.

($_f$**equalp** *foo bar*)
▷ `T` if *foo* and *bar* are identical; or are the same **character** ignoring case; or are **number**s of the same value ignoring type; or are equivalent **pathname**s; or are **cons**es or **array**s of the same shape with $_f$**equalp** elements; or are structures of the same type with $_f$**equalp** elements; or are **hash-table**s of the same size with the same **:test** function, the same keys in terms of **:test** function, and $_f$**equalp** elements.

($_f$**not** *foo*) ▷ `T` if *foo* is `NIL`; `NIL` otherwise.

($_f$**boundp** *symbol*) ▷ `T` if *symbol* is a special variable.

($_f$**make-concatenated-stream** $input\text{-}stream^*$)
($_f$**make-broadcast-stream** $output\text{-}stream^*$)
($_f$**make-two-way-stream** *input-stream-part output-stream-part*)
($_f$**make-echo-stream** *from-input-stream to-output-stream*)
($_f$**make-synonym-stream** *variable-bound-to-stream*)
▷ Return stream of indicated type.

($_f$**make-string-input-stream** *string* $[start_{\boxed{0}}\ [end_{\boxed{\texttt{NIL}}}]]$)
▷ Return a **string-stream** supplying the characters from *string*.

($_f$**make-string-output-stream** [**:element-type** $type_{\boxed{\textbf{character}}}$])
▷ Return a **string-stream** accepting characters (available via $_f$**get-output-stream-string**).

($_f$**concatenated-stream-streams** *concatenated-stream*)
($_f$**broadcast-stream-streams** *broadcast-stream*)
▷ Return list of streams *concatenated-stream* still has to read from/*broadcast-stream* is broadcasting to.

($_f$**two-way-stream-input-stream** *two-way-stream*)
($_f$**two-way-stream-output-stream** *two-way-stream*)
($_f$**echo-stream-input-stream** *echo-stream*)
($_f$**echo-stream-output-stream** *echo-stream*)
▷ Return source stream or sink stream of *two-way-stream*/*echo-stream*, respectively.

($_f$**synonym-stream-symbol** *synonym-stream*)
▷ Return symbol of *synonym-stream*.

($_f$**get-output-stream-string** $\widetilde{string\text{-}stream}$)
▷ Clear and return as a string characters on *string-stream*.

$(_f\textbf{file-position}\ stream\ [\left\{\begin{array}{l}\textbf{:start}\\ \textbf{:end}\\ position\end{array}\right\}])$
▷ Return position within stream, or set it to *position* and return `T` on success.

($_f$**file-string-length** *stream foo*)
▷ Length *foo* would have in *stream*.

($_f$**listen** $[stream_{\boxed{_v\textbf{*standard-input*}}}]$)
▷ `T` if there is a character in input *stream*.

($_f$**clear-input** $[\widetilde{stream}_{\boxed{_v\textbf{*standard-input*}}}]$)
▷ Clear input from *stream*, return `NIL`.

$(\left\{\begin{array}{l}_f\textbf{clear-output}\\ _f\textbf{force-output}\\ _f\textbf{finish-output}\end{array}\right\}\ [\widetilde{stream}_{\boxed{_v\textbf{*standard-output*}}}])$
▷ End output to *stream* and return `NIL` immediately, after initiating flushing of buffers, or after flushing of buffers, respectively.

($_f$**close** $\widetilde{stream}$ [**:abort** $bool_{\boxed{\texttt{NIL}}}$])
▷ Close *stream*. Return `T` if *stream* had been open. If **:abort** is `T`, delete associated file.

($_m$**with-open-file** ($stream\ path\ open\text{-}arg^*$) (**declare** $\widehat{decl}^*$)* $form^{P_*}$)
▷ Use $_f$**open** with *open-arg*s to temporarily create *stream* to *path*; return values of *form*s.

($_m$**with-open-stream** (*foo* $\widetilde{stream}$) (**declare** $\widehat{decl}^*$)* $form^{P_*}$)
▷ Evaluate *form*s with *foo* locally bound to *stream*. Return values of *forms*.

$(_m\textbf{with-input-from-string}\ (foo\ string \left\{\begin{array}{|l}\textbf{:index}\ \widetilde{index}\\ \textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}}\end{array}\right\})\ (\textbf{declare}$
$\widehat{decl}^*$)* $form^{P_*}$)
▷ Evaluate *forms* with *foo* locally bound to input **string-stream** from *string*. Return values of *forms*; store next reading position into *index*.

{~ [*n*$_{\boxed{0}}$] **i**|~ [*n*$_{\boxed{0}}$] **:i**}

▷ **Indent.** Set indentation to *n* relative to leftmost/to current position.

~ [*c*$_{\boxed{1}}$] [,*i*$_{\boxed{1}}$] [**:**] [**@**] **T**

▷ **Tabulate.** Move cursor forward to column number $c + ki$, $k \geq 0$ being as small as possible. With **:**, calculate column numbers relative to the immediately enclosing section. With **@**, move to column number $c_0 + c + ki$ where c_0 is the current position.

{~ [*m*$_{\boxed{1}}$] *****|~ [*m*$_{\boxed{1}}$] **:***|~ [*n*$_{\boxed{0}}$] **@***}

▷ **Go-To.** Jump *m* arguments forward, or backward, or to argument *n*.

~ [*limit*] [**:**] [**@**] **{** *text* **~}**

▷ **Iteration.** Use *text* repeatedly, up to *limit*, as control string for the elements of the list argument or (with **@**) for the remaining arguments. With **:** or **@:**, list elements or remaining arguments should be lists of which a new one is used at each iteration step.

~ [*x* [,*y* [,*z*]]] **^**

▷ **Escape Upward.** Leave immediately **~< ~>**, **~< ~:>**, **~{ ~}**, **~?**, or the entire $_f$**format** operation. With one to three prefixes, act only if $x = 0$, $x = y$, or $x \leq y \leq z$, respectively.

~ [*i*] [**:**] [**@**] **[** [{*text* **~;**}* *text*] [**~:;** *default*] **~]**

▷ **Conditional Expression.** Use the zero-indexed argumenth (or *i*th if given) *text* as a $_f$**format** control subclause. With **:**, use the first *text* if the argument value is `NIL`, or the second *text* if it is `T`. With **@**, do nothing for an argument value of `NIL`. Use the only *text* and leave the argument to be read again if it is `T`.

{**~?**|**~@?**}

▷ **Recursive Processing.** Process two arguments as control string and argument list, or take one argument as control string and use then the rest of the original arguments.

~ [*prefix* {,*prefix*}*] [**:**] [**@**] **/**[*package* [**:**]**:**$_{\boxed{\texttt{cl-user:}}}$]*function***/**

▷ **Call Function.** Call all-uppercase *package***::***function* with the arguments stream, format-argument, colon-p, at-sign-p and *prefix*es for printing format-argument.

~ [**:**] [**@**] **W**

▷ **Write.** Print argument of any type obeying every printer control variable. With **:**, pretty-print. With **@**, print without limits on length or depth.

{**V**|**#**}

▷ In place of the comma-separated prefix parameters: use next argument or number of remaining unprocessed arguments, respectively.

13.6 Streams

$$(_f\textbf{open}\ path \left\{\begin{array}{|l}
\textbf{:direction} \left\{\begin{array}{l}\textbf{:input}\\ \textbf{:output}\\ \textbf{:io}\\ \textbf{:probe}\end{array}\right\}_{\boxed{\textbf{:input}}} \\
\textbf{:element-type} \left\{\begin{array}{l}type\\ \textbf{:default}\end{array}\right\}_{\boxed{\textbf{character}}} \\
\textbf{:if-exists} \left\{\begin{array}{l}\textbf{:new-version}\\ \textbf{:error}\\ \textbf{:rename}\\ \textbf{:rename-and-delete}\\ \textbf{:overwrite}\\ \textbf{:append}\\ \textbf{:supersede}\\ \texttt{NIL}\end{array}\right\}_{\boxed{\begin{array}{l}\textbf{:new-version}\text{ if }path\\ \text{specifies }\textbf{:newest};\\ \texttt{NIL}\text{ otherwise}\end{array}}} \\
\textbf{:if-does-not-exist} \left\{\begin{array}{l}\textbf{:error}\\ \textbf{:create}\\ \texttt{NIL}\end{array}\right\}_{\boxed{\begin{array}{l}\texttt{NIL}\text{ for }\textbf{:direction :probe};\\ \{\textbf{:create}|\textbf{:error}\}\text{ otherwise}\end{array}}} \\
\textbf{:external-format}\ format_{\boxed{\textbf{:default}}}
\end{array}\right\})$$

▷ Open **file-stream** to *path*.

($_f$**constantp** *foo* [*environment*$_{\boxed{\texttt{NIL}}}$])

▷ `T` if *foo* is a constant form.

($_f$**functionp** *foo*) ▷ `T` if *foo* is of type **function**.

($_f$**fboundp** $\left\{\begin{array}{l}foo\\ (\textbf{setf}\ foo)\end{array}\right\}$) ▷ `T` if *foo* is a global function or macro.

9.2 Variables

($\left\{\begin{array}{l}_m\textbf{defconstant}\\ _m\textbf{defparameter}\end{array}\right\}$ $\widehat{foo}$ *form* [$\widehat{doc}$])

▷ Assign value of *form* to global constant/dynamic variable *foo*.

($_m$**defvar** $\widehat{foo}$ [*form* [$\widehat{doc}$]])

▷ Unless bound already, assign value of *form* to dynamic variable *foo*.

($\left\{\begin{array}{l}_m\textbf{setf}\\ _m\textbf{psetf}\end{array}\right\}$ {*place form*}*)

▷ Set *place*s to primary values of *form*s. Return values of last *form*/`NIL`; work sequentially/in parallel, respectively.

($\left\{\begin{array}{l}_s\textbf{setq}\\ _m\textbf{psetq}\end{array}\right\}$ {*symbol form*}*)

▷ Set *symbol*s to primary values of *form*s. Return value of last *form*/`NIL`; work sequentially/in parallel, respectively.

($_f$**set** $\widetilde{symbol}$ *foo*) ▷ Set *symbol*'s value cell to *foo*. Deprecated.

($_m$**multiple-value-setq** *vars form*)

▷ Set elements of *vars* to the values of *form*. Return *form*'s primary value.

($_m$**shiftf** $\widetilde{place}^{+}$ *foo*)

▷ Store value of *foo* in rightmost *place* shifting values of *place*s left, returning first *place*.

($_m$**rotatef** $\widetilde{place}^{*}$)

▷ Rotate values of *place*s left, old first becoming new last *place*'s value. Return `NIL`.

($_f$**makunbound** $\widetilde{foo}$) ▷ Delete special variable *foo* if any.

($_f$**get** *symbol key* [*default*$_{\boxed{\texttt{NIL}}}$])
($_f$**getf** *place key* [*default*$_{\boxed{\texttt{NIL}}}$])

▷ First entry *key* from property list stored in *symbol*/in *place*, respectively, or *default* if there is no *key*. **setf**able.

($_f$**get-properties** *property-list keys*)

▷ Return key and $\underset{2}{\text{value}}$ of first entry from *property-list* matching a key from *keys*, and $\overset{3}{\text{tail of } property\text{-}list}$ starting with that key. Return `NIL`, $\underset{2}{\texttt{NIL}}$, and $\underset{3}{\texttt{NIL}}$ if there was no matching key in *property-list*.

($_f$**remprop** $\widetilde{symbol}$ *key*)
($_m$**remf** $\widetilde{place}$ *key*)

▷ Remove first entry *key* from property list stored in *symbol*/in *place*, respectively. Return `T` if *key* was there, or `NIL` otherwise.

($_s$**progv** *symbols values* $form^{P_*}$)

▷ Evaluate *form*s with locally established dynamic bindings of *symbols* to *values* or `NIL`. Return values of *forms*.

($\left\{\begin{array}{l}_s\textbf{let}\\ _s\textbf{let*}\end{array}\right\}$ ($\left\{\begin{array}{|l}name\\ (name\ [value_{\boxed{\texttt{NIL}}}])\end{array}\right\}^{*}$) (**declare** $\widehat{decl}^{*}$)* $form^{P_*}$)

▷ Evaluate *forms* with *name*s lexically bound (in parallel or sequentially, respectively) to *value*s. Return values of *forms*.

$({}_m$**multiple-value-bind** $(\widehat{var^*})$ *values-form* (**declare** $\widehat{decl^*})^*$ *body-form*$^{\overset{P}{*}}$)

▷ Evaluate *body-forms* with *var*s lexically bound to the return values of *values-form*. Return values of *body-forms*.

$({}_m$**destructuring-bind** *destruct-λ bar* (**declare** $\widehat{decl^*})^*$ *form*$^{\overset{P}{*}}$)

▷ Evaluate *forms* with variables from tree *destruct-λ* bound to corresponding elements of tree *bar*, and return their values. *destruct-λ* resembles *macro-λ* (section 9.4), but without any **&environment** clause.

9.3 Functions

Below, ordinary lambda list (*ord-λ**) has the form

$$(var^*\ \left[\textbf{\&optional} \begin{Bmatrix} var \\ (var\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{Bmatrix}^*\right]\ [\textbf{\&rest}\ var]$$

$$\left[\textbf{\&key} \begin{Bmatrix} var \\ \left(\begin{Bmatrix} var \\ (:key\ var) \end{Bmatrix}\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]\right) \end{Bmatrix}^*\ [\textbf{\&allow-other-keys}]\right]$$

$$\left[\textbf{\&aux} \begin{Bmatrix} var \\ (var\ [init_{\boxed{\texttt{NIL}}}]) \end{Bmatrix}^*\right]).$$

supplied-p is `T` if there is a corresponding argument. *init* forms can refer to any *init* and *supplied-p* to their left.

$$\left(\begin{Bmatrix} {}_m\textbf{defun} \begin{Bmatrix} foo\ (ord\text{-}\lambda^*) \\ (\textbf{setf}\ foo)\ (new\text{-}value\ ord\text{-}\lambda^*) \end{Bmatrix} \\ {}_m\textbf{lambda}\ (ord\text{-}\lambda^*) \end{Bmatrix} (\textbf{declare}\ \widehat{decl^*})^*\ [\widehat{doc}]\right.$$

form$^{\overset{P}{*}}$)

▷ Define a function named *foo* or (**setf** *foo*), or an anonymous function, respectively, which applies *forms* to *ord-λ*s. For ${}_m$**defun**, *forms* are enclosed in an implicit ${}_s$**block** named *foo*.

$$\left(\begin{Bmatrix} {}_s\textbf{flet} \\ {}_s\textbf{labels} \end{Bmatrix} \left(\left(\begin{Bmatrix} foo\ (ord\text{-}\lambda^*) \\ (\textbf{setf}\ foo)\ (new\text{-}value\ ord\text{-}\lambda^*) \end{Bmatrix} (\textbf{declare}\ \widehat{local\text{-}decl^*})^*\right.\right.\right.$$

$[\widehat{doc}]$ *local-form*$^{\overset{P}{*}}$)*) (**declare** $\widehat{decl^*})^*$ *form*$^{\overset{P}{*}}$)

▷ Evaluate *forms* with locally defined functions *foo*. Globally defined functions of the same name are shadowed. Each *foo* is also the name of an implicit ${}_s$**block** around its corresponding *local-form**. Only for ${}_s$**labels**, functions *foo* are visible inside *local-forms*. Return values of *forms*.

$$\left({}_s\textbf{function} \begin{Bmatrix} foo \\ ({}_m\textbf{lambda}\ form^*) \end{Bmatrix}\right)$$

▷ Return lexically innermost function named *foo* or a lexical closure of the ${}_m$**lambda** expression.

$$\left({}_f\textbf{apply} \begin{Bmatrix} function \\ (\textbf{setf}\ function) \end{Bmatrix}\ arg^*\ args\right)$$

▷ Values of *function* called with *args* and the list elements of *args*. **setf**able if *function* is one of ${}_f$**aref**, ${}_f$**bit**, and ${}_f$**sbit**.

$({}_f$**funcall** *function* arg*) ▷ Values of *function* called with *args*.

$({}_s$**multiple-value-call** *function form**)

▷ Call *function* with all the values of each *form* as its arguments. Return values returned by *function*.

$({}_f$**values-list** *list*) ▷ Return elements of *list*.

$({}_f$**values** *foo**)

▷ Return as multiple values the primary values of the *foo*s. **setf**able.

$({}_f$**multiple-value-list** *form*) ▷ List of the values of *form*.

$({}_m$**nth-value** *n form*)

▷ Zero-indexed *n*th return value of *form*.

$({}_f$**complement** *function*)

▷ Return new function with same arguments and same side effects as *function*, but with complementary truth value.

`{~R|~:R|~@R|~@:R}`

▷ **Roman.** Take argument as number and print it as English cardinal number, as English ordinal number, as Roman numeral, or as old Roman numeral, respectively.

$\sim [width]\ \big[,[\text{'}pad\text{-}char_{\boxed{␣}}]\ \big[,[\text{'}comma\text{-}char_{\boxed{,}}]\ \big[,comma\text{-}interval_{\boxed{3}}\big]\big]\big]$ [:] [@] {**D**|**B**|**O**|**X**}

▷ **Decimal/Binary/Octal/Hexadecimal.** Print integer argument as number. With **:**, group digits *comma-interval* each; with **@**, always prepend a sign.

$\sim [width]\ \big[,[dec\text{-}digits]\ \big[,[shift_{\boxed{0}}]\ \big[,[\text{'}overflow\text{-}char]\ \big[,\text{'}pad\text{-}char_{\boxed{␣}}\big]\big]\big]\big]$ [@] **F**

▷ **Fixed-Format Floating-Point.** With **@**, always prepend a sign.

$\sim [width]\ \big[,[dec\text{-}digits]\ \big[,[exp\text{-}digits]\ \big[,[scale\text{-}factor_{\boxed{1}}]\ \big[,[\text{'}overflow\text{-}char]\ \big[,[\text{'}pad\text{-}char_{\boxed{␣}}]\ \big[,\text{'}exp\text{-}char\big]\big]\big]\big]\big]\big]$ [@] {**E**|**G**}

▷ **Exponential/General Floating-Point.** Print argument as floating-point number with *dec-digits* after decimal point and *exp-digits* in the signed exponent. With **~G**, choose either **~E** or **~F**. With **@**, always prepend a sign.

$\sim [dec\text{-}digits_{\boxed{2}}]\ \big[,[int\text{-}digits_{\boxed{1}}]\ \big[,[width_{\boxed{0}}]\ \big[,\text{'}pad\text{-}char_{\boxed{␣}}\big]\big]\big]$ [:] [@] **$**

▷ **Monetary Floating-Point.** Print argument as fixed-format floating-point number. With **:**, put sign before any padding; with **@**, always prepend a sign.

`{~C|~:C|~@C|~@:C}`

▷ **Character.** Print, spell out, print in `#\` syntax, or tell how to type, respectively, argument as (possibly non-printing) character.

{**~(** *text* **~)**|**~:(** *text* **~)**|**~@(** *text* **~)**|**~@:(** *text* **~)**}

▷ **Case-Conversion.** Convert *text* to lowercase, convert first letter of each word to uppercase, capitalize first word and convert the rest to lowercase, or convert to uppercase, respectively.

`{~P|~:P |~@P|~@:P}`

▷ **Plural.** If argument **eql** `1` print nothing, otherwise print `s`; do the same for the previous argument; if argument **eql** `1` print `y`, otherwise print `ies`; do the same for the previous argument, respectively.

$\sim [n_{\boxed{1}}]$ **%** ▷ **Newline.** Print *n* newlines.

$\sim [n_{\boxed{1}}]$ **&**

▷ **Fresh-Line.** Print *n* − 1 newlines if output stream is at the beginning of a line, or *n* newlines otherwise.

`{~_|~:_|~@_|~@:_}`

▷ **Conditional Newline.** Print a newline like **pprint-newline** with argument **:linear**, **:fill**, **:miser**, or **:mandatory**, respectively.

`{~:↩|~@↩|~↩}`

▷ **Ignored Newline.** Ignore newline, or whitespace following newline, or both, respectively.

$\sim [n_{\boxed{1}}]$ **|** ▷ **Page.** Print *n* page separators.

$\sim [n_{\boxed{1}}]$ **~** ▷ **Tilde.** Print *n* tildes.

$\sim [min\text{-}col_{\boxed{0}}]\ \big[,[col\text{-}inc_{\boxed{1}}]\ \big[,[min\text{-}pad_{\boxed{0}}]\ \big[,\text{'}pad\text{-}char_{\boxed{␣}}\big]\big]\big]$ [:] [@] **<** $\big[nl\text{-}text$ **~**$[spare_{\boxed{0}}\ [,width]]$**:;**$\big]$ {*text* **~;**}* *text* **~>**

▷ **Justification.** Justify text produced by *texts* in a field of at least *min-col* columns. With **:**, right justify; with **@**, left justify. If this would leave less than *spare* characters on the current line, output *nl-text* first.

~ [:] [@] **<** $\big\{[prefix_{\boxed{""}}$ **~;**$]\big|[per\text{-}line\text{-}prefix$ **~@;**$]\big\}$ *body* $\big[$**~;** $suffix_{\boxed{""}}\big]$ **~:** [@] **>**

▷ **Logical Block.** Act like **pprint-logical-block** using *body* as ${}_f$**format** control string on the elements of the list argument or, with **@**, on the remaining arguments, which are extracted by **pprint-pop**. With **:**, *prefix* and *suffix* default to `(` and `)`. When closed by **~@:>**, spaces in *body* are replaced with conditional newlines.

$_v$***print-case***$_{\boxed{\texttt{:upcase}}}$

▷ Print symbol names all uppercase (**:upcase**), all lowercase (**:downcase**), capitalized (**:capitalize**).

$_v$***print-circle***$_{\boxed{\texttt{NIL}}}$

▷ If `T`, avoid indefinite recursion while printing circular structure.

$_v$***print-escape***$_{\boxed{\texttt{T}}}$

▷ If `NIL`, do not print escape characters and package prefixes.

$_v$***print-gensym***$_{\boxed{\texttt{T}}}$ ▷ If `T`, print **#:** before uninterned symbols.

$_v$***print-length***$_{\boxed{\texttt{NIL}}}$
$_v$***print-level***$_{\boxed{\texttt{NIL}}}$
$_v$***print-lines***$_{\boxed{\texttt{NIL}}}$

▷ If integer, restrict printing of objects to that number of elements per level/to that depth/to that number of lines.

$_v$***print-miser-width***

▷ If integer and greater than the width available for printing a substructure, switch to the more compact miser style.

$_v$***print-pretty*** ▷ If `T`, print prettily.

$_v$***print-radix***$_{\boxed{\texttt{NIL}}}$ ▷ If `T`, print rationals with a radix indicator.

$_v$***print-readably***$_{\boxed{\texttt{NIL}}}$

▷ If `T`, print $_f$**read**ably or signal error **print-not-readable**.

$_v$***print-right-margin***$_{\boxed{\texttt{NIL}}}$

▷ Right margin width in ems while pretty-printing.

($_f$**set-pprint-dispatch** *type function* [*priority*$_{\boxed{0}}$ [*table*$_{\boxed{_v\textbf{*print-pprint-dispatch*}}}$]])

▷ Install entry comprising *function* of arguments stream and object to print; and *priority* as *type* into *table*. If *function* is `NIL`, remove *type* from *table*. Return `NIL`.

($_f$**pprint-dispatch** *foo* [*table*$_{\boxed{_v\textbf{*print-pprint-dispatch*}}}$])

▷ Return highest priority *function* associated with type of *foo* and $\underset{2}{\underline{\texttt{T}}}$ if there was a matching type specifier in *table*.

($_f$**copy-pprint-dispatch** [*table*$_{\boxed{_v\textbf{*print-pprint-dispatch*}}}$])

▷ Return copy of *table* or, if *table* is `NIL`, initial value of $_v$***print-pprint-dispatch***.

$_v$***print-pprint-dispatch*** ▷ Current pretty print dispatch table.

13.5 Format

($_m$**formatter** $\widehat{control}$)

▷ Return function of *stream* and *arg** applying $_f$**format** to *stream*, *control*, and *arg** returning `NIL` or any excess *arg*s.

($_f$**format** {`T`|`NIL`|*out-string*|*out-stream*} *control arg**)

▷ Output string *control* which may contain ~ directives possibly taking some *arg*s. Alternatively, *control* can be a function returned by $_m$**formatter** which is then applied to *out-stream* and *arg**. Output to *out-string*, *out-stream* or, if first argument is `T`, to $_v$***standard-output***. Return `NIL`. If first argument is `NIL`, return formatted output.

~ [*min-col*$_{\boxed{0}}$] [,[*col-inc*$_{\boxed{1}}$] [,[*min-pad*$_{\boxed{0}}$] [,'*pad-char*$_{\boxed{\llcorner\lrcorner}}$]]]
[**:**] [**@**] {**A**|**S**}

▷ **Aesthetic/Standard.** Print argument of any type for consumption by humans/by the reader, respectively. With **:**, print `NIL` as **()** rather than **nil**; with **@**, add *pad-char*s on the left rather than on the right.

~ [*radix*$_{\boxed{10}}$] [,[*width*] [,['*pad-char*$_{\boxed{\llcorner\lrcorner}}$] [,['*comma-char*$_{\boxed{,}}$]
[,*comma-interval*$_{\boxed{3}}$]]]] [**:**] [**@**] **R**

▷ **Radix.** (With one or more prefix arguments.) Print argument as number; with **:**, group digits *comma-interval* each; with **@**, always prepend a sign.

($_f$**constantly** *foo*)

▷ Function of any number of arguments returning *foo*.

($_f$**identity** *foo*) ▷ Return *foo*.

($_f$**function-lambda-expression** *function*)

▷ If available, return lambda expression of *function*, $\underset{2}{\underline{\texttt{NIL}}}$ if *function* was defined in an environment without bindings, and $\underset{3}{\underline{\text{name}}}$ of *function*.

$$(_f\textbf{fdefinition} \left\{\begin{matrix} foo \\ (\textbf{setf}\ foo) \end{matrix}\right\})$$

▷ Definition of global function *foo*. **setf**able.

($_f$**fmakunbound** *foo*)

▷ Remove global function or macro definition *foo*.

$_c$**call-arguments-limit**
$_c$**lambda-parameters-limit**

▷ Upper bound of the number of function arguments or lambda list parameters, respectively; ≥ 50.

$_c$**multiple-values-limit**

▷ Upper bound of the number of values a multiple value can have; ≥ 20.

9.4 Macros

Below, macro lambda list (*macro-λ**) has the form of either

$$([\textbf{\&whole}\ var]\ [E] \left\{\begin{matrix} var \\ (macro\text{-}\lambda^*) \end{matrix}\right\}^* [E]$$

$$[\textbf{\&optional} \left\{\begin{matrix} var \\ (\left\{\begin{matrix} var \\ (macro\text{-}\lambda^*) \end{matrix}\right\} [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{matrix}\right\}^*]\ [E]$$

$$[\left\{\begin{matrix} \textbf{\&rest} \\ \textbf{\&body} \end{matrix}\right\} \left\{\begin{matrix} rest\text{-}var \\ (macro\text{-}\lambda^*) \end{matrix}\right\}]\ [E]$$

$$[\textbf{\&key} \left\{\begin{matrix} var \\ (\left\{\begin{matrix} var \\ (\textbf{:}key \left\{\begin{matrix} var \\ (macro\text{-}\lambda^*) \end{matrix}\right\}) \end{matrix}\right\} [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{matrix}\right\}^* [E]$$

$$[\textbf{\&allow-other-keys}]]\ [\textbf{\&aux} \left\{\begin{matrix} var \\ (var\ [init_{\boxed{\texttt{NIL}}}]) \end{matrix}\right\}^*]\ [E])$$

or

$$([\textbf{\&whole}\ var]\ [E] \left\{\begin{matrix} var \\ (macro\text{-}\lambda^*) \end{matrix}\right\}^* [E]\ [\textbf{\&optional}$$

$$\left\{\begin{matrix} var \\ (\left\{\begin{matrix} var \\ (macro\text{-}\lambda^*) \end{matrix}\right\} [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{matrix}\right\}^*]\ [E]\ \textbf{.}\ rest\text{-}var).$$

One toplevel [*E*] may be replaced by **&environment** *var*. *supplied-p* is `T` if there is a corresponding argument. *init* forms can refer to any *init* and *supplied-p* to their left.

$$(\left\{\begin{matrix} _m\textbf{defmacro} \\ _f\textbf{define-compiler-macro} \end{matrix}\right\} \left\{\begin{matrix} foo \\ (\textbf{setf}\ foo) \end{matrix}\right\} (macro\text{-}\lambda^*)\ (\textbf{declare}\ \widehat{decl}^*)^*$$
$$[\widehat{doc}]\ form^{\text{P}*})$$

▷ Define macro *foo* which on evaluation as (*foo tree*) applies expanded *forms* to arguments from *tree*, which corresponds to *tree*-shaped *macro-λ*s. *forms* are enclosed in an implicit $_s$**block** named *foo*.

($_m$**define-symbol-macro** *foo form*)

▷ Define symbol macro *foo* which on evaluation evaluates expanded *form*.

($_s$**macrolet** ((*foo* (*macro-λ**) (**declare** $\widehat{local\text{-}decl}^*$)* [$\widehat{doc}$] *macro-form*$^{\text{P}*}$)*) (**declare** $\widehat{decl}^*$)* *form*$^{\text{P}*}$)

▷ Evaluate *forms* with locally defined mutually invisible macros *foo* which are enclosed in implicit $_s$**block**s of the same name.

($_s$**symbol-macrolet** ((*foo expansion-form*)*) (**declare** $\widehat{decl}^*$)* $form^{P_*}$)

▷ Evaluate $\underline{forms}$ with locally defined symbol macros *foo*.

($_m$**defsetf** $\widehat{function}$

$$\left\{\begin{array}{l} \widehat{updater}\ [\widehat{doc}] \\ (setf\text{-}\lambda^*)\ (s\text{-}var^*)\ (\textbf{declare}\ \widehat{decl}^*)^*\ [\widehat{doc}]\ form^{P_*} \end{array}\right\})$$

where defsetf lambda list (*setf-λ**) has the form

$$(var^*\ \left[\textbf{\&optional} \left\{\begin{array}{l} var \\ (var\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{array}\right\}^*\right]$$

$$[\textbf{\&rest}\ var]\ \left[\textbf{\&key} \left\{\begin{array}{l} var \\ (\left\{\begin{array}{l} var \\ (\textbf{:}key\ var) \end{array}\right\}\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{array}\right\}^*\right.$$

$$\left.[\textbf{\&allow-other-keys}]\right]\ [\textbf{\&environment}\ var])$$

▷ Specify how to **setf** a place accessed by $\underline{function}$. **Short form:** (**setf** (*function arg**) *value-form*) is replaced by (*updater arg** *value-form*); the latter must return *value-form*. **Long form:** on invocation of (**setf** (*function arg**) *value-form*), *forms* must expand into code that sets the place accessed where *setf-λ* and *s-var** describe the arguments of *function* and the value(s) to be stored, respectively; and that returns the value(s) of *s-var**. *forms* are enclosed in an implicit $_s$**block** named *function*.

($_m$**define-setf-expander** *function* (*macro-λ**) (**declare** $\widehat{decl}^*$)* [$\widehat{doc}$] $form^{P_*}$)

▷ Specify how to **setf** a place accessed by $\underline{function}$. On invocation of (**setf** (*function arg**) *value-form*), *form** must expand into code returning *arg-vars*, *args*, *newval-vars*, *set-form*, and *get-form* as described with $_f$**get-setf-expansion** where the elements of macro lambda list *macro-λ** are bound to corresponding *args*. *forms* are enclosed in an implicit $_s$**block** named *function*.

($_f$**get-setf-expansion** *place* [$environment_{\boxed{\texttt{NIL}}}$])

▷ Return lists of temporary variables $\underline{arg\text{-}vars}$ and of corresponding $\underline{args}_2$ as given with *place*, list $\underline{newval\text{-}vars}_3$ with temporary variables corresponding to the new values, and $\underline{set\text{-}form}_4$ and $\underline{get\text{-}form}_5$ specifying in terms of *arg-vars* and *newval-vars* how to **setf** and how to read *place*.

($_m$**define-modify-macro** *foo* ([**&optional**

$$\left\{\begin{array}{l} var \\ (var\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{array}\right\}^*]\ [\textbf{\&rest}\ var])\ function\ [\widehat{doc}])$$

▷ Define macro $\underline{foo}$ able to modify a place. On invocation of (*foo place arg**), the value of *function* applied to *place* and *args* will be stored into *place* and returned.

$_c$**lambda-list-keywords**

▷ List of macro lambda list keywords. These are at least:

&whole *var*

▷ Bind *var* to the entire macro call form.

&optional *var**

▷ Bind *vars* to corresponding arguments if any.

{**&rest**|**&body**} *var*

▷ Bind *var* to a list of remaining arguments.

&key *var**

▷ Bind *vars* to corresponding keyword arguments.

&allow-other-keys

▷ Suppress keyword argument checking. Callers can do so using **:allow-other-keys** `T`.

&environment *var*

▷ Bind *var* to the lexical compilation environment.

&aux *var** ▷ Bind *vars* as in $_s$**let***.

$$(\left\{\begin{array}{l} {}_f\textbf{write} \\ {}_f\textbf{write-to-string} \end{array}\right\}\ foo\ \left\{\left|\begin{array}{l} \textbf{:array}\ bool \\ \textbf{:base}\ radix \\ \textbf{:case} \left\{\begin{array}{l} \textbf{:upcase} \\ \textbf{:downcase} \\ \textbf{:capitalize} \end{array}\right. \\ \textbf{:circle}\ bool \\ \textbf{:escape}\ bool \\ \textbf{:gensym}\ bool \\ \textbf{:length}\ \{int|\texttt{NIL}\} \\ \textbf{:level}\ \{int|\texttt{NIL}\} \\ \textbf{:lines}\ \{int|\texttt{NIL}\} \\ \textbf{:miser-width}\ \{int|\texttt{NIL}\} \\ \textbf{:pprint-dispatch}\ dispatch\text{-}table \\ \textbf{:pretty}\ bool \\ \textbf{:radix}\ bool \\ \textbf{:readably}\ bool \\ \textbf{:right-margin}\ \{int|\texttt{NIL}\} \\ \textbf{:stream}\ \widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}} \end{array}\right|\right\})$$

▷ Print *foo* to *stream* and return $\underline{foo}$, or print *foo* into $\underline{string}$, respectively, after dynamically setting printer variables corresponding to keyword parameters (***print-***bar****** becoming **:***bar*). (**:stream** keyword with $_f$**write** only.)

($_f$**pprint-fill** $\widetilde{stream}$ *foo* [$parenthesis_{\boxed{\texttt{T}}}$ [*noop*]])

($_f$**pprint-tabular** $\widetilde{stream}$ *foo* [$parenthesis_{\boxed{\texttt{T}}}$ [*noop* [$n_{\boxed{16}}$]]])

($_f$**pprint-linear** $\widetilde{stream}$ *foo* [$parenthesis_{\boxed{\texttt{T}}}$ [*noop*]])

▷ Print *foo* to *stream*. If *foo* is a list, print as many elements per line as possible; do the same in a table with a column width of *n* ems; or print either all elements on one line or each on its own line, respectively. Return $\underline{\texttt{NIL}}$. Usable with $_f$**format** directive ~//.

$$({}_m\textbf{pprint-logical-block}\ (\widetilde{stream}\ list\ \left\{\left|\begin{array}{l} \left\{\begin{array}{l} \textbf{:prefix}\ string \\ \textbf{:per-line-prefix}\ string \end{array}\right\} \\ \textbf{:suffix}\ string_{\boxed{\texttt{""}}} \end{array}\right|\right\})$$

(**declare** $\widehat{decl}^*$)* $form^{P_*}$)

▷ Evaluate *forms*, which should print *list*, with *stream* locally bound to a pretty printing stream which outputs to the original *stream*. If *list* is in fact not a list, it is printed by $_f$**write**. Return $\underline{\texttt{NIL}}$.

($_m$**pprint-pop**)

▷ Take $\underline{\text{next element}}$ off *list*. If there is no remaining tail of *list*, or $_v$***print-length*** or $_v$***print-circle*** indicate printing should end, send element together with an appropriate indicator to *stream*.

$$({}_f\textbf{pprint-tab} \left\{\begin{array}{l} \textbf{:line} \\ \textbf{:line-relative} \\ \textbf{:section} \\ \textbf{:section-relative} \end{array}\right\}\ c\ i\ [\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}])$$

▷ Move cursor forward to column number $c + ki$, $k \geq 0$ being as small as possible.

$$({}_f\textbf{pprint-indent} \left\{\begin{array}{l} \textbf{:block} \\ \textbf{:current} \end{array}\right\}\ n\ [\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}])$$

▷ Specify indentation for innermost logical block relative to leftmost position/to current position. Return $\underline{\texttt{NIL}}$.

($_m$**pprint-exit-if-list-exhausted**)

▷ If *list* is empty, terminate logical block. Return $\underline{\texttt{NIL}}$ otherwise.

$$({}_f\textbf{pprint-newline} \left\{\begin{array}{l} \textbf{:linear} \\ \textbf{:fill} \\ \textbf{:miser} \\ \textbf{:mandatory} \end{array}\right\}\ [\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}])$$

▷ Print a conditional newline if *stream* is a pretty printing stream. Return $\underline{\texttt{NIL}}$.

$_v$***print-array*** ▷ If `T`, print arrays $_f$**read**ably.

$_v$***print-base***$_{\boxed{10}}$ ▷ Radix for printing rationals, from 2 to 36.

#+*feature when-feature*
#−*feature unless-feature*

▷ Means *when-feature* if *feature* is `T`; means *unless-feature* if *feature* is `NIL`. *feature* is a symbol from ${}_v$***features***, or ({**and**|**or**} *feature**), or (**not** *feature*).

${}_v$***features***

▷ List of symbols denoting implementation-dependent features.

|c^*|; \\c

▷ Treat arbitrary character(s) c as alphabetic preserving case.

13.4 Printer

$(\left\{\begin{array}{l} {}_f\textbf{prin1} \\ {}_f\textbf{print} \\ {}_f\textbf{pprint} \\ {}_f\textbf{princ} \end{array}\right\} \textit{foo}\ [\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-output*}}}])$

▷ Print *foo* to *stream* ${}_f$**read**ably, ${}_f$**read**ably between a newline and a space, ${}_f$**read**ably after a newline, or human-readably without any extra characters, respectively. ${}_f$**prin1**, ${}_f$**print** and ${}_f$**princ** return *foo*.

(${}_f$**prin1-to-string** *foo*)
(${}_f$**princ-to-string** *foo*)

▷ Print *foo* to *string* ${}_f$**read**ably or human-readably, respectively.

$({}_g\textbf{print-object}\ \textit{object}\ \widetilde{\textit{stream}})$

▷ Print *object* to *stream*. Called by the Lisp printer.

$({}_m\textbf{print-unreadable-object}\ (\textit{foo}\ \widetilde{\textit{stream}}\ \left\{\begin{array}{l} |\textbf{:type}\ \textit{bool}_{\boxed{\texttt{NIL}}} \\ |\textbf{:identity}\ \textit{bool}_{\boxed{\texttt{NIL}}} \end{array}\right\})\ \textit{form}^{P*})$

▷ Enclosed in #< and >, print *foo* by means of *form*s to *stream*. Return `NIL`.

$({}_f\textbf{terpri}\ [\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-output*}}}])$

▷ Output a newline to *stream*. Return `NIL`.

$({}_f\textbf{fresh-line}\ [\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-output*}}}])$

▷ Output a newline to *stream* and return `T` unless *stream* is already at the start of a line.

$({}_f\textbf{write-char}\ \textit{char}\ [\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-output*}}}])$

▷ Output *char* to *stream*.

$(\left\{\begin{array}{l} {}_f\textbf{write-string} \\ {}_f\textbf{write-line} \end{array}\right\}\ \textit{string}\ [\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-output*}}}\ [\left\{\begin{array}{l} |\textbf{:start}\ \textit{start}_{\boxed{0}} \\ |\textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}} \end{array}\right\}]])$

▷ Write *string* to *stream* without/with a trailing newline.

$({}_f\textbf{write-byte}\ \textit{byte}\ \widetilde{\textit{stream}})$ ▷ Write *byte* to binary *stream*.

$({}_f\textbf{write-sequence}\ \textit{sequence}\ \widetilde{\textit{stream}}\ \left\{\begin{array}{l} |\textbf{:start}\ \textit{start}_{\boxed{0}} \\ |\textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}} \end{array}\right\})$

▷ Write elements of *sequence* to binary or character *stream*.

9.5 Control Flow

$({}_s\textbf{if}\ \textit{test}\ \textit{then}\ [\textit{else}_{\boxed{\texttt{NIL}}}])$

▷ Return values of *then* if *test* returns `T`; return values of *else* otherwise.

$({}_m\textbf{cond}\ (\textit{test}\ \textit{then}^{P*}{}_{\boxed{\textit{test}}})^*)$

▷ Return the values of the first *then** whose *test* returns `T`; return `NIL` if all *test*s return `NIL`.

$(\left\{\begin{array}{l} {}_m\textbf{when} \\ {}_m\textbf{unless} \end{array}\right\}\ \textit{test}\ \textit{foo}^{P*})$

▷ Evaluate *foo*s and return their values if *test* returns `T` or `NIL`, respectively. Return `NIL` otherwise.

$({}_m\textbf{case}\ \textit{test}\ (\left\{\begin{array}{l} (\widehat{\textit{key}}^*) \\ \widehat{\textit{key}} \end{array}\right\}\ \textit{foo}^{P*})^*\ [(\left\{\begin{array}{l} \textbf{otherwise} \\ \texttt{T} \end{array}\right\}\ \textit{bar}^{P*})_{\boxed{\texttt{NIL}}}])$

▷ Return the values of the first *foo** one of whose *key*s is **eql** *test*. Return values of *bar*s if there is no matching *key*.

$(\left\{\begin{array}{l} {}_m\textbf{ecase} \\ {}_m\textbf{ccase} \end{array}\right\}\ \textit{test}\ (\left\{\begin{array}{l} (\widehat{\textit{key}}^*) \\ \widehat{\textit{key}} \end{array}\right\}\ \textit{foo}^{P*})^*)$

▷ Return the values of the first *foo** one of whose *key*s is **eql** *test*. Signal non-correctable/correctable **type-error** if there is no matching *key*.

$({}_m\textbf{and}\ \textit{form}^*{}_{\boxed{\texttt{T}}})$

▷ Evaluate *form*s from left to right. Immediately return `NIL` if one *form*'s value is `NIL`. Return values of last *form* otherwise.

$({}_m\textbf{or}\ \textit{form}^*{}_{\boxed{\texttt{NIL}}})$

▷ Evaluate *form*s from left to right. Immediately return primary value of first non-`NIL`-evaluating form, or all values if last *form* is reached. Return `NIL` if no *form* returns `T`.

$({}_s\textbf{progn}\ \textit{form}^*{}_{\boxed{\texttt{NIL}}})$

▷ Evaluate *form*s sequentially. Return values of last *form*.

(${}_s$**multiple-value-prog1** *form-r form**)
(${}_m$**prog1** *form-r form**)
(${}_m$**prog2** *form-a form-r form**)

▷ Evaluate forms in order. Return values/primary value, respectively, of *form-r*.

$(\left\{\begin{array}{l} {}_m\textbf{prog} \\ {}_m\textbf{prog*} \end{array}\right\}\ (\left\{\begin{array}{l} |\textit{name} \\ |(\textit{name}\ [\textit{value}_{\boxed{\texttt{NIL}}}]) \end{array}\right\}^*)\ (\textbf{declare}\ \widehat{\textit{decl}}^*)^*\ \left\{\begin{array}{l} \widehat{\textit{tag}} \\ \textit{form} \end{array}\right\}^*)$

▷ Evaluate ${}_s$**tagbody**-like body with *name*s lexically bound (in parallel or sequentially, respectively) to *value*s. Return `NIL` or explicitly ${}_m$**return**ed values. Implicitly, the whole form is a ${}_s$**block** named `NIL`.

(${}_s$**unwind-protect** *protected cleanup**)

▷ Evaluate *protected* and then, no matter how control leaves *protected*, *cleanup*s. Return values of *protected*.

$({}_s\textbf{block}\ \textit{name}\ \textit{form}^{P*})$

▷ Evaluate *form*s in a lexical environment, and return their values unless interrupted by ${}_s$**return-from**.

$({}_s\textbf{return-from}\ \textit{foo}\ [\textit{result}_{\boxed{\texttt{NIL}}}])$
$({}_m\textbf{return}\ [\textit{result}_{\boxed{\texttt{NIL}}}])$

▷ Have nearest enclosing ${}_s$**block** named *foo*/named `NIL`, respectively, return with values of *result*.

$({}_s\textbf{tagbody}\ \{\widehat{\textit{tag}}|\textit{form}\}^*)$

▷ Evaluate *form*s in a lexical environment. *tag*s (symbols or integers) have lexical scope and dynamic extent, and are targets for ${}_s$**go**. Return `NIL`.

$({}_s\textbf{go}\ \widehat{\textit{tag}})$

▷ Within the innermost possible enclosing ${}_s$**tagbody**, jump to a tag ${}_f$**eql** *tag*.

(${}_s$**catch** *tag form*$\overset{P}{*}$)
▷ Evaluate *forms* and return $\underline{\text{their values}}$ unless interrupted by ${}_s$**throw**.

(${}_s$**throw** *tag form*)
▷ Have the nearest dynamically enclosing ${}_s$**catch** with a tag ${}_f$**eq** *tag* return with the values of *form*.

(${}_f$**sleep** *n*) ▷ Wait *n* seconds; return $\underline{\texttt{NIL}}$.

9.6 Iteration

$\left(\begin{Bmatrix}{}_m\textbf{do}\\{}_m\textbf{do*}\end{Bmatrix}\ \left(\begin{Bmatrix}var\\(var\ [start\ [step]])\end{Bmatrix}^*\right)\ (stop\ result\overset{P}{*})\ (\textbf{declare}\ \widehat{decl}^*)^*\ \begin{Bmatrix}\widehat{tag}\\form\end{Bmatrix}^*\right)$

▷ Evaluate ${}_s$**tagbody**-like body with *vars* successively bound according to the values of the corresponding *start* and *step* forms. *vars* are bound in parallel/sequentially, respectively. Stop iteration when *stop* is `T`. Return $\underline{\text{values of } result^*}$. Implicitly, the whole form is a ${}_s$**block** named `NIL`.

$({}_m\textbf{dotimes}\ (var\ i\ [result_{\boxed{\texttt{NIL}}}])\ (\textbf{declare}\ \widehat{decl}^*)^*\ \{\widehat{tag}|form\}^*)$
▷ Evaluate ${}_s$**tagbody**-like body with *var* successively bound to integers from 0 to $i-1$. Upon evaluation of $\underline{result}$, *var* is i. Implicitly, the whole form is a ${}_s$**block** named `NIL`.

$({}_m\textbf{dolist}\ (var\ list\ [result_{\boxed{\texttt{NIL}}}])\ (\textbf{declare}\ \widehat{decl}^*)^*\ \{\widehat{tag}|form\}^*)$
▷ Evaluate ${}_s$**tagbody**-like body with *var* successively bound to the elements of *list*. Upon evaluation of $\underline{result}$, *var* is `NIL`. Implicitly, the whole form is a ${}_s$**block** named `NIL`.

9.7 Loop Facility

(${}_m$**loop** *form**)
▷ **Simple Loop.** If *forms* do not contain any atomic Loop Facility keywords, evaluate them forever in an implicit ${}_s$**block** named `NIL`.

(${}_m$**loop** *clause**)
▷ **Loop Facility.** For Loop Facility keywords see below and Figure 1.

named $n_{\boxed{\texttt{NIL}}}$ ▷ Give ${}_m$**loop**'s implicit ${}_s$**block** a name.

$\{\textbf{with}\ \begin{Bmatrix}var\text{-}s\\(var\text{-}s^*)\end{Bmatrix}\ [d\text{-}type]\ [=foo]\}^+$
$\{\textbf{and}\ \begin{Bmatrix}var\text{-}p\\(var\text{-}p^*)\end{Bmatrix}\ [d\text{-}type]\ [=bar]\}^*$

where destructuring type specifier *d-type* has the form

$\left\{\textbf{fixnum}|\textbf{float}|\texttt{T}|\texttt{NIL}|\left\{\textbf{of-type}\ \begin{Bmatrix}type\\(type^*)\end{Bmatrix}\right\}\right\}$

▷ Initialize (possibly trees of) local variables *var-s* sequentially and *var-p* in parallel.

$\{\{\textbf{for}|\textbf{as}\}\ \begin{Bmatrix}var\text{-}s\\(var\text{-}s^*)\end{Bmatrix}\ [d\text{-}type]\}^+\ \{\textbf{and}\ \begin{Bmatrix}var\text{-}p\\(var\text{-}p^*)\end{Bmatrix}\ [d\text{-}type]\}^*$

▷ Begin of iteration control clauses. Initialize and step (possibly trees of) local variables *var-s* sequentially and *var-p* in parallel. Destructuring type specifier *d-type* as with **with**.

{**upfrom**|**from**|**downfrom**} *start*
▷ Start stepping with *start*

{**upto**|**downto**|**to**|**below**|**above**} *form*
▷ Specify *form* as the end value for stepping.

{**in**|**on**} *list*
▷ Bind *var* to successive elements/tails, respectively, of *list*.

by $\{step_{\boxed{1}}|function_{\boxed{\texttt{\#'cdr}}}\}$
▷ Specify the (positive) decrement or increment or the *function* of one argument returning the next part of the list.

(${}_f$**set-dispatch-macro-character** *char sub-char function* $[\widetilde{rt}_{\boxed{{}_v\textbf{*readtable*}}}]$)
▷ Make *function* of stream, *n*, *sub-char* a dispatch function of *char* followed by *n*, followed by *sub-char*. Return $\underline{\texttt{T}}$.

(${}_f$**get-dispatch-macro-character** *char sub-char* $[rt_{\boxed{{}_v\textbf{*readtable*}}}]$)
▷ $\underline{\text{Dispatch function}}$ associated with *char* followed by *sub-char*.

13.3 Character Syntax

#| *multi-line-comment** **|#**
; *one-line-comment**
▷ Comments. There are stylistic conventions:

;;;; *title*	▷ Short title for a block of code.
;;; *intro*	▷ Description before a block of code.
;; *state*	▷ State of program or of following code.
;*explanation* ; *continuation*	▷ Regarding line on which it appears.

(*foo**[**.** $bar_{\boxed{\texttt{NIL}}}$]**)** ▷ List of *foo*s with the terminating cdr *bar*.

" ▷ Begin and end of a string.

'*foo* ▷ (${}_s$**quote** *foo*); *foo* unevaluated.

\`([*foo*] [**,***bar*] [**,@***baz*] [**,.**$\widetilde{quux}$] [*bing*])
▷ Backquote. ${}_s$**quote** *foo* and *bing*; evaluate *bar* and splice the lists *baz* and *quux* into their elements. When nested, outermost commas inside the innermost backquote expression belong to this backquote.

#*c* ▷ (${}_f$**character** "*c*"), the character *c*.

#B*n*; **#O***n*; *n***.**; **#X***n*; **#***r***R***n*
▷ Integer of radix 2, 8, 10, 16, or *r*; $2 \le r \le 36$.

*n***/***d* ▷ The **ratio** $\frac{n}{d}$.

$\left\{[m].n\left[\{\textbf{S}|\textbf{F}|\textbf{D}|\textbf{L}|\textbf{E}\}x_{\boxed{\textbf{E0}}}\right]\middle|m\left[.[n]\right]\{\textbf{S}|\textbf{F}|\textbf{D}|\textbf{L}|\textbf{E}\}x\right\}$
▷ $m.n\cdot 10^x$ as **short-float**, **single-float**, **double-float**, **long-float**, or the type from ***read-default-float-format***.

#C(*a b***)** ▷ (${}_f$**complex** *a b*), the complex number $a + bi$.

#'*foo* ▷ (${}_s$**function** *foo*); the function named *foo*.

#*n***A***sequence* ▷ *n*-dimensional array.

#[*n*]**(***foo****)**
▷ Vector of some (or *n*) *foo*s filled with last *foo* if necessary.

#[*n*]******b**
▷ Bit vector of some (or *n*) *b*s filled with last *b* if necessary.

#S(*type* {*slot value*}***)** ▷ Structure of *type*.

#P*string* ▷ A pathname.

#:*foo* ▷ Uninterned symbol *foo*.

#.*form* ▷ Read-time value of *form*.

${}_v$***read-eval***$_{\boxed{\texttt{T}}}$ ▷ If `NIL`, a **reader-error** is signalled at **#.**.

#*integer***=** *foo* ▷ Give *foo* the label *integer*.

#*integer***#** ▷ Object labelled *integer*.

#< ▷ Have the reader signal **reader-error**.

(${}_f$**read-delimited-list** *char* [$\widetilde{stream}_{\boxed{{}_v\textbf{*standard-input*}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]])
▷ Continue reading until encountering *char*. Return $\underline{\text{list}}$ of objects read. Signal error if no *char* is found in stream.

(${}_f$**read-char** [$\widetilde{stream}_{\boxed{{}_v\textbf{*standard-input*}}}$ [$eof\text{-}err_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]]]])
▷ Return $\underline{\text{next character}}$ from *stream*.

(${}_f$**read-char-no-hang** [$\widetilde{stream}_{\boxed{{}_v\textbf{*standard-input*}}}$ [$eof\text{-}error_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]]]])
▷ $\underline{\text{Next character}}$ from *stream* or $\underline{\texttt{NIL}}$ if none is available.

(${}_f$**peek-char** [$mode_{\boxed{\texttt{NIL}}}$ [$\widetilde{stream}_{\boxed{{}_v\textbf{*standard-input*}}}$ [$eof\text{-}error_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]]]]])
▷ Next, or if *mode* is `T`, next non-whitespace $\underline{\text{character}}$, or if *mode* is a character, $\underline{\text{next instance}}$ of it, from *stream* without removing it there.

(${}_f$**unread-char** *character* [$\widetilde{stream}_{\boxed{{}_v\textbf{*standard-input*}}}$])
▷ Put last ${}_f$**read-char**ed *character* back into *stream*; return $\underline{\texttt{NIL}}$.

(${}_f$**read-byte** $\widetilde{stream}$ [$eof\text{-}err_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$]])
▷ Read $\underline{\text{next byte}}$ from binary *stream*.

(${}_f$**read-line** [$\widetilde{stream}_{\boxed{{}_v\textbf{*standard-input*}}}$ [$eof\text{-}err_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]]]])
▷ Return a $\underline{\text{line of text}}$ from *stream* and $\underset{2}{\underline{\texttt{T}}}$ if line has been ended by end of file.

(${}_f$**read-sequence** $\widetilde{sequence}$ $\widetilde{stream}$ [**:start** $start_{\boxed{0}}$][**:end** $end_{\boxed{\texttt{NIL}}}$])
▷ Replace elements of *sequence* between *start* and *end* with elements from binary or character *stream*. Return $\underline{\text{index}}$ of *sequence*'s first unmodified element.

(${}_f$**readtable-case** *readtable*)$_{\boxed{\textbf{:upcase}}}$
▷ $\underline{\text{Case sensitivity attribute}}$ (one of **:upcase**, **:downcase**, **:preserve**, **:invert**) of *readtable*. **setf**able.

(${}_f$**copy-readtable** [$from\text{-}readtable_{\boxed{{}_v\textbf{*readtable*}}}$ [$\widetilde{to\text{-}readtable}_{\boxed{\texttt{NIL}}}$]])
▷ Return $\underline{\text{copy of } \textit{from-readtable}}$.

(${}_f$**set-syntax-from-char** *to-char from-char* [$\widetilde{to\text{-}readtable}_{\boxed{{}_v\textbf{*readtable*}}}$ [$from\text{-}readtable_{\boxed{\text{standard readtable}}}$]])
▷ Copy syntax of *from-char* to *to-readtable*. Return $\underline{\texttt{T}}$.

${}_v$***readtable*** ▷ Current readtable.

${}_v$***read-base***$_{\boxed{10}}$ ▷ Radix for reading **integer**s and **ratio**s.

${}_v$***read-default-float-format***$_{\boxed{\textbf{single-float}}}$
▷ Floating point format to use when not indicated in the number read.

${}_v$***read-suppress***$_{\boxed{\texttt{NIL}}}$
▷ If `T`, reader is syntactically more tolerant.

(${}_f$**set-macro-character** *char function* [$non\text{-}term\text{-}p_{\boxed{\texttt{NIL}}}$ [$\widetilde{rt}_{\boxed{{}_v\textbf{*readtable*}}}$]])
▷ Make *char* a macro character associated with *function* of stream and *char*. Return $\underline{\texttt{T}}$.

(${}_f$**get-macro-character** *char* [$rt_{\boxed{{}_v\textbf{*readtable*}}}$])
▷ $\underline{\text{Reader macro function}}$ associated with *char*, and $\underset{2}{\underline{\texttt{T}}}$ if *char* is a non-terminating macro character.

(${}_f$**make-dispatch-macro-character** *char* [$non\text{-}term\text{-}p_{\boxed{\texttt{NIL}}}$ [$rt_{\boxed{{}_v\textbf{*readtable*}}}$]])
▷ Make *char* a dispatching macro character. Return $\underline{\texttt{T}}$.

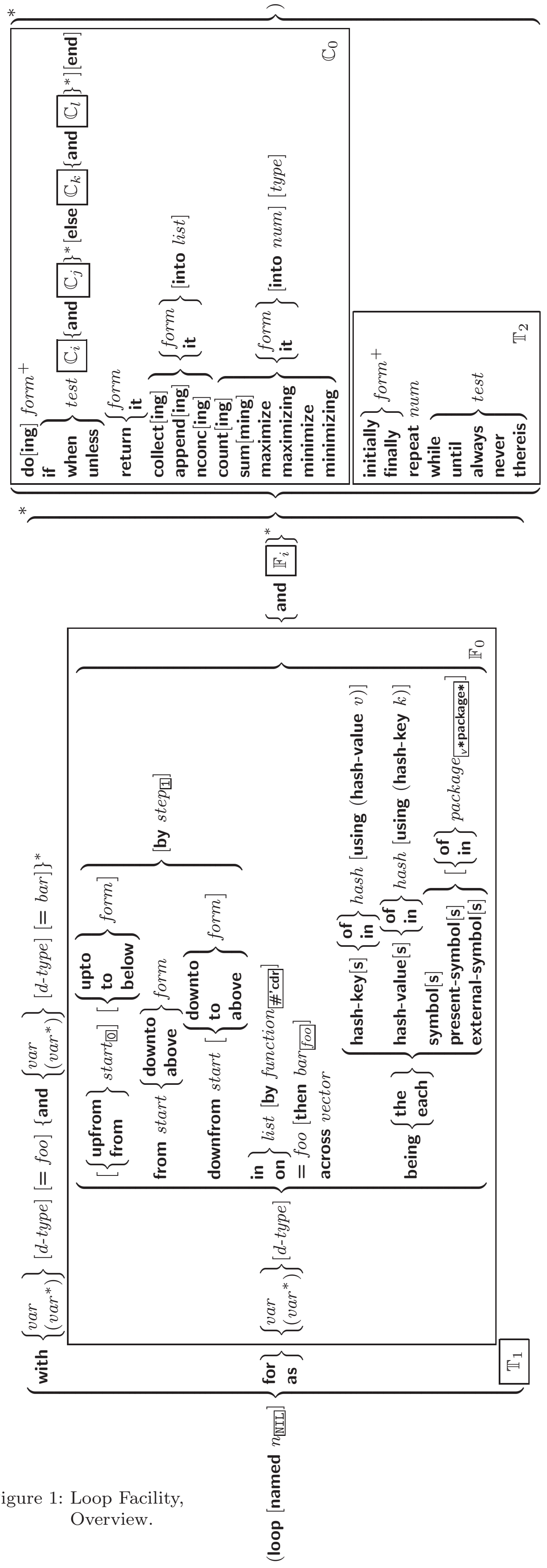
(loop [named n NIL]
with {var | (var*)} [d-type] [= foo] {and {var | (var*)} [d-type] [= bar]}*
for | as {var | (var*)} [d-type]
[{upfrom | from} start 0] [{upto | to | below} form]
from start {downto | above} form
downfrom start [{downto | to | above} form]
[by step 1]
in | on list [by function #'cdr]
= foo [then bar foo]
across vector
being {the | each} hash-key[s] {of | in} hash [using (hash-value v)]
hash-value[s] {of | in} hash [using (hash-key k)]
symbol[s] | present-symbol[s] | external-symbol[s] [{of | in} package v*package*]
F0
T1
{and F_i}*
do[ing] form+
if | when | unless test C_i {and C_j}* [else C_k {and C_l}*] [end]
return {form | it}
collect[ing] | append[ing] | nconc[ing] {form | it} [into list]
count[ing] | sum[ming] | maximize | maximizing | minimize | minimizing {form | it} [into num] [type]
C0
initially | finally form+
repeat num
while | until | always | never | thereis test
T2

Figure 1: Loop Facility, Overview.

= *foo* [**then** $bar_{\boxed{foo}}$]
▷ Bind *var* initially to *foo* and later to *bar*.

across *vector*
▷ Bind *var* to successive elements of *vector*.

being {**the**|**each**}
▷ Iterate over a hash table or a package.

{**hash-key**|**hash-keys**} {**of**|**in**} *hash-table* [**using** (**hash-value** *value*)]
▷ Bind *var* successively to the keys of *hash-table*; bind *value* to corresponding values.

{**hash-value**|**hash-values**} {**of**|**in**} *hash-table* [**using** (**hash-key** *key*)]
▷ Bind *var* successively to the values of *hash-table*; bind *key* to corresponding keys.

{**symbol**|**symbols**|**present-symbol**|**present-symbols**| **external-symbol**|**external-symbols**} [{**of**|**in**} $package_{\boxed{_v\text{*package*}}}$]
▷ Bind *var* successively to the accessible symbols, or the present symbols, or the external symbols respectively, of *package*.

{**do**|**doing**} $form^+$
▷ Evaluate *form*s in every iteration.

{**if**|**when**|**unless**} *test i-clause* {**and** *j-clause*}* [**else** *k-clause* {**and** *l-clause*}*] [**end**]
▷ If *test* returns T, T, or NIL, respectively, evaluate *i-clause* and *j-clause*s; otherwise, evaluate *k-clause* and *l-clause*s.

it ▷ Inside *i-clause* or *k-clause*: value of *test*.

return {*form*|**it**}
▷ Return immediately, skipping any **finally** parts, with values of *form* or **it**.

{**collect**|**collecting**} {*form*|**it**} [**into** *list*]
▷ Collect values of *form* or **it** into *list*. If no *list* is given, collect into an anonymous list which is returned after termination.

{**append**|**appending**|**nconc**|**nconcing**} {*form*|**it**} [**into** *list*]
▷ Concatenate values of *form* or **it**, which should be lists, into *list* by the means of $_f$**append** or $_f$**nconc**, respectively. If no *list* is given, collect into an anonymous list which is returned after termination.

{**count**|**counting**} {*form*|**it**} [**into** *n*] [*type*]
▷ Count the number of times the value of *form* or of **it** is T. If no *n* is given, count into an anonymous variable which is returned after termination.

{**sum**|**summing**} {*form*|**it**} [**into** *sum*] [*type*]
▷ Calculate the sum of the primary values of *form* or of **it**. If no *sum* is given, sum into an anonymous variable which is returned after termination.

{**maximize**|**maximizing**|**minimize**|**minimizing**} {*form*|**it**} [**into** *max-min*] [*type*]
▷ Determine the maximum or minimum, respectively, of the primary values of *form* or of **it**. If no *max-min* is given, use an anonymous variable which is returned after termination.

{**initially**|**finally**} $form^+$
▷ Evaluate *form*s before begin, or after end, respectively, of iterations.

repeat *num*
▷ Terminate $_m$**loop** after *num* iterations; *num* is evaluated once.

{**while**|**until**} *test*
▷ Continue iteration until *test* returns NIL or T, respectively.

($_m$**deftype** *foo* (*macro*-λ^*) (**declare** $\widehat{decl^*}$)* [$\widehat{doc}$] $form^{P*}$)
▷ Define type *foo* which when referenced as (*foo* $\widehat{arg^*}$) (or as *foo* if *macro-λ* doesn't contain any required parameters) applies expanded *form*s to *arg*s returning the new type. For (*macro*-λ^*) see page 19 but with default value of * instead of NIL. *form*s are enclosed in an implicit $_s$**block** named *foo*.

(**eql** *foo*)
(**member** *foo**) ▷ Specifier for a type comprising *foo* or *foo*s.

(**satisfies** *predicate*)
▷ Type specifier for all objects satisfying *predicate*.

(**mod** *n*) ▷ Type specifier for all non-negative integers < *n*.

(**not** *type*) ▷ Complement of type.

(**and** $type^*_{\boxed{T}}$) ▷ Type specifier for intersection of *type*s.

(**or** $type^*_{\boxed{NIL}}$) ▷ Type specifier for union of *type*s.

(**values** *type** [**&optional** *type** [**&rest** *other-args*]])
▷ Type specifier for multiple values.

* ▷ As a type argument (cf. Figure 2): no restriction.

13 Input/Output

13.1 Predicates

($_f$**streamp** *foo*)
($_f$**pathnamep** *foo*) ▷ T if *foo* is of indicated type.
($_f$**readtablep** *foo*)

($_f$**input-stream-p** *stream*)
($_f$**output-stream-p** *stream*)
($_f$**interactive-stream-p** *stream*)
($_f$**open-stream-p** *stream*)
▷ Return T if *stream* is for input, for output, interactive, or open, respectively.

($_f$**pathname-match-p** *path wildcard*)
▷ T if *path* matches *wildcard*.

($_f$**wild-pathname-p** *path* [{**:host**|**:device**|**:directory**|**:name**|**:type**| **:version**|NIL}])
▷ Return T if indicated component in *path* is wildcard. (NIL indicates any component.)

13.2 Reader

({$_f$**y-or-n-p** | $_f$**yes-or-no-p**} [*control arg**])
▷ Ask user a question and return T or NIL depending on their answer. See page 38, $_f$**format**, for *control* and *arg*s.

($_m$**with-standard-io-syntax** $form^{P*}$)
▷ Evaluate *form*s with standard behaviour of reader and printer. Return values of *forms*.

({$_f$**read** | $_f$**read-preserving-whitespace**} [$\widetilde{stream}_{\boxed{_v\text{*standard-input*}}}$ [$eof\text{-}err_{\boxed{T}}$ [$eof\text{-}val_{\boxed{NIL}}$ [$recursive_{\boxed{NIL}}$]]]])
▷ Read printed representation of object.

($_f$**read-from-string** *string* [$eof\text{-}error_{\boxed{T}}$ [$eof\text{-}val_{\boxed{NIL}}$ [{|**:start** $start_{\boxed{0}}$ | **:end** $end_{\boxed{NIL}}$ | **:preserve-whitespace** $bool_{\boxed{NIL}}$|}]]])
▷ Return object read from string and zero-indexed $\underline{position}_2$ of next character.

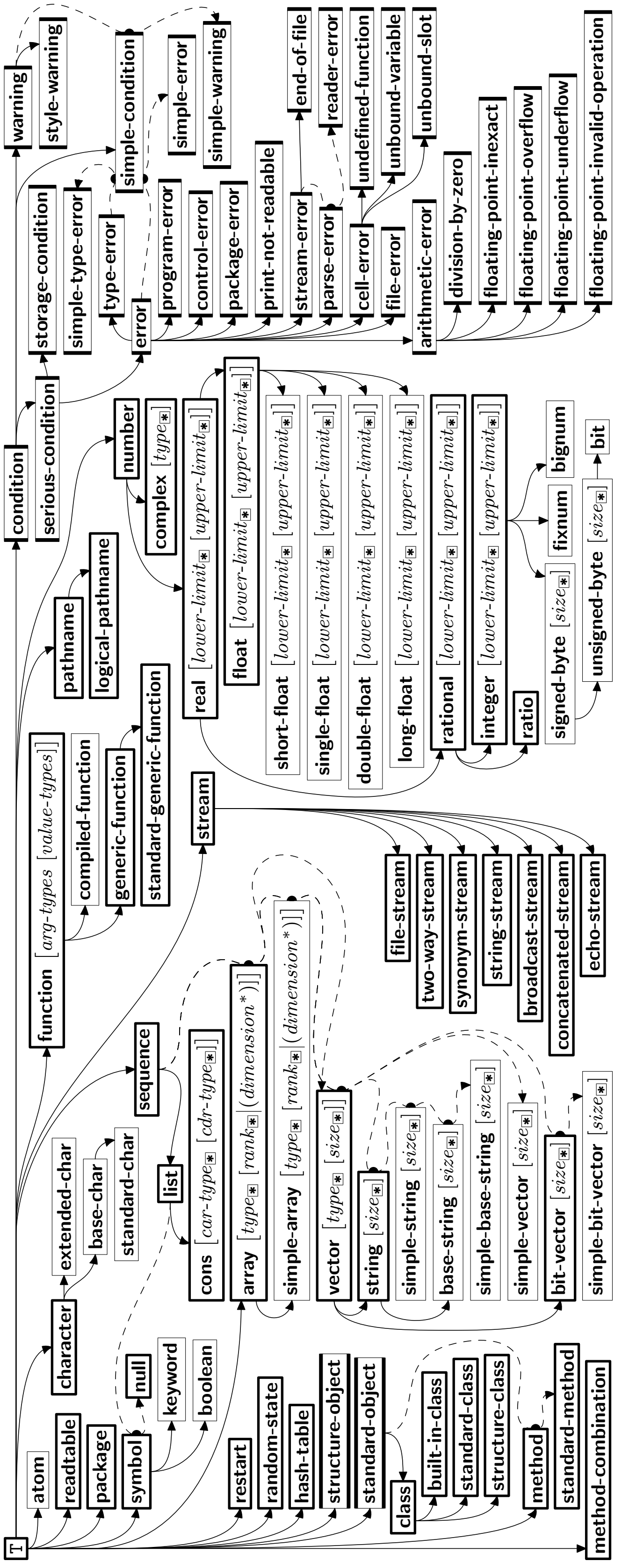

Figure 2: Precedence Order of System Classes (▭), Classes (▬), Types (▭), and Condition Types (▭).
Every type is also a supertype of NIL, the empty type.

{**always**|**never**} *test*

▷ Terminate $_m$**loop** returning NIL and skipping any **finally** parts as soon as *test* is NIL or T, respectively. Otherwise continue $_m$**loop** with its default return value set to T.

thereis *test*

▷ Terminate $_m$**loop** when *test* is T and return value of *test*, skipping any **finally** parts. Otherwise continue $_m$**loop** with its default return value set to NIL.

($_m$**loop-finish**)

▷ Terminate $_m$**loop** immediately executing any **finally** clauses and returning any accumulated results.

10 CLOS

10.1 Classes

($_f$**slot-exists-p** *foo bar*) ▷ T if *foo* has a slot *bar*.

($_f$**slot-boundp** *instance slot*) ▷ T if *slot* in *instance* is bound.

($_m$**defclass** *foo* (*superclass*$^*_{\boxed{\text{standard-object}}}$)

$$\left(\left\{\begin{array}{l} slot \\ (slot \left\{\begin{array}{l} \{\textbf{:reader}\ reader\}^* \\ \{\textbf{:writer} \left\{\begin{array}{l} writer \\ (\textbf{setf}\ writer)\end{array}\right\}\}^* \\ \{\textbf{:accessor}\ accessor\}^* \\ \textbf{:allocation} \left\{\begin{array}{l}\textbf{:instance}\\ \textbf{:class}\end{array}\right\}_{\boxed{\text{:instance}}} \\ \{\textbf{:initarg}\ \textbf{:}initarg\text{-}name\}^* \\ \textbf{:initform}\ form \\ \textbf{:type}\ type \\ \textbf{:documentation}\ slot\text{-}doc \end{array}\right\}) \end{array}\right\}^*\right)$$

$$\left\{\begin{array}{l} (\textbf{:default-initargs}\ \{name\ value\}^*) \\ (\textbf{:documentation}\ class\text{-}doc) \\ (\textbf{:metaclass}\ name_{\boxed{\text{standard-class}}}) \end{array}\right\})$$

▷ Define or modify class *foo* as a subclass of *superclass*es. Transform existing instances, if any, by $_g$**make-instances-obsolete**. In a new instance *i* of *foo*, a *slot*'s value defaults to *form* unless set via **:***initarg-name*; it is readable via (*reader i*) or (*accessor i*), and writable via (*writer value i*) or (**setf** (*accessor i*) *value*). *slot*s with **:allocation :class** are shared by all instances of class *foo*.

($_f$**find-class** *symbol* [*errorp*$_{\boxed{T}}$ [*environment*]])

▷ Return class named *symbol*. **setf**able.

($_g$**make-instance** *class* {**:***initarg value*}* *other-keyarg**)

▷ Make new instance of *class*.

($_g$**reinitialize-instance** *instance* {**:***initarg value*}* *other-keyarg**)

▷ Change local slots of *instance* according to *initarg*s by means of $_g$**shared-initialize**.

($_f$**slot-value** *foo slot*) ▷ Return value of *slot* in *foo*. **setf**able.

($_f$**slot-makunbound** *instance slot*)

▷ Make *slot* in *instance* unbound.

$$\left(\left\{\begin{array}{l} {}_m\textbf{with-slots}\ (\{\widehat{slot}|(\widehat{var}\ \widehat{slot})\}^*) \\ {}_m\textbf{with-accessors}\ ((\widehat{var}\ \widehat{accessor})^*) \end{array}\right\}\ instance\ (\textbf{declare}\ \widehat{decl}^*)^*\ form^{P*}\right)$$

▷ Return values of *forms* after evaluating them in a lexical environment with slots of *instance* visible as **setf**able *slot*s or *var*s/with *accessor*s of *instance* visible as **setf**able *var*s.

($_g$**class-name** *class*)
((**setf** $_g$**class-name**) *new-name class*) ▷ Get/set name of *class*.

($_f$**class-of** *foo*) ▷ Class *foo* is a direct instance of.

($_g$**change-class** $\widehat{instance}$ *new-class* {**:***initarg value*}* *other-keyarg**)
▷ Change class of instance to *new-class*. Retain the status of any slots that are common between *instance*'s original class and *new-class*. Initialize any newly added slots with the *value*s of the corresponding *initarg*s if any, or with the values of their **:initform** forms if not.

($_g$**make-instances-obsolete** *class*)
▷ Update all existing instances of *class* using $_g$**update-instance-for-redefined-class**.

$\left(\left\{\begin{array}{l} {}_g\textbf{initialize-instance}\ \textit{instance} \\ {}_g\textbf{update-instance-for-different-class}\ \textit{previous current} \end{array}\right\}\right.$
{**:***initarg value*}* *other-keyarg**)
▷ Set slots on behalf of $_g$**make-instance**/of $_g$**change-class** by means of $_g$**shared-initialize**.

($_g$**update-instance-for-redefined-class** *new-instance added-slots discarded-slots discarded-slots-property-list* {**:***initarg value*}* *other-keyarg**)
▷ On behalf of $_g$**make-instances-obsolete** and by means of $_g$**shared-initialize**, set any *initarg* slots to their corresponding *value*s; set any remaining *added-slots* to the values of their **:initform** forms. Not to be called by user.

($_g$**allocate-instance** *class* {**:***initarg value*}* *other-keyarg**)
▷ Return uninitialized instance of *class*. Called by $_g$**make-instance**.

($_g$**shared-initialize** *instance* $\left\{\begin{array}{l} \textit{initform-slots} \\ \texttt{T} \end{array}\right\}$ {**:***initarg-slot value*}* *other-keyarg**)
▷ Fill the *initarg-slots* of *instance* with the corresponding *value*s, and fill those *initform-slots* that are not *initarg-slots* with the values of their **:initform** forms.

($_g$**slot-missing** *class instance slot* $\left\{\begin{array}{l} \textbf{setf} \\ \textbf{slot-boundp} \\ \textbf{slot-makunbound} \\ \textbf{slot-value} \end{array}\right\}$ [*value*])
($_g$**slot-unbound** *class instance slot*)
▷ Called on attempted access to non-existing or unbound *slot*. Default methods signal **error**/**unbound-slot**, respectively. Not to be called by user.

10.2 Generic Functions

($_f$**next-method-p**) ▷ T if enclosing method has a next method.

($_m$**defgeneric** $\left\{\begin{array}{l} \textit{foo} \\ (\textbf{setf}\ \textit{foo}) \end{array}\right\}$ (*required-var** [**&optional** $\left\{\begin{array}{l} \textit{var} \\ (\textit{var}) \end{array}\right\}^*$]
[**&rest** *var*] [**&key** $\left\{\begin{array}{l} \textit{var} \\ (\textit{var} \mid (\textbf{:}\textit{key var})) \end{array}\right\}^*$ [**&allow-other-keys**]])
$\left\{\begin{array}{l} \mid (\textbf{:argument-precedence-order}\ \textit{required-var}^+) \\ \mid (\textbf{declare}\ (\textbf{optimize}\ \textit{method-selection-optimization})^+) \\ \mid (\textbf{:documentation}\ \widehat{\textit{string}}) \\ \mid (\textbf{:generic-function-class}\ \textit{gf-class}_{\boxed{\textbf{standard-generic-function}}}) \\ \mid (\textbf{:method-class}\ \textit{method-class}_{\boxed{\textbf{standard-method}}}) \\ \mid (\textbf{:method-combination}\ \textit{c-type}_{\boxed{\textbf{standard}}}\ \textit{c-arg}^*) \\ \mid (\textbf{:method}\ \textit{defmethod-args})^* \end{array}\right\}$)
▷ Define or modify generic function *foo*. Remove any methods previously defined by defgeneric. *gf-class* and the lambda paramters *required-var** and *var** must be compatible with existing methods. *defmethod-args* resemble those of $_m$**defmethod**. For *c-type* see section 10.3.

($_f$**ensure-generic-function** $\left\{\begin{array}{l} \textit{foo} \\ (\textbf{setf}\ \textit{foo}) \end{array}\right\}$

($_f$**cell-error-name** *condition*)
▷ Name of cell which caused *condition*.

($_f$**unbound-slot-instance** *condition*)
▷ Instance with unbound slot which caused *condition*.

($_f$**print-not-readable-object** *condition*)
▷ The object not readably printable under *condition*.

($_f$**package-error-package** *condition*)
($_f$**file-error-pathname** *condition*)
($_f$**stream-error-stream** *condition*)
▷ Package, path, or stream, respectively, which caused the *condition* of indicated type.

($_f$**type-error-datum** *condition*)
($_f$**type-error-expected-type** *condition*)
▷ Object which caused *condition* of type **type-error**, or its expected type, respectively.

($_f$**simple-condition-format-control** *condition*)
($_f$**simple-condition-format-arguments** *condition*)
▷ Return $_f$**format** control or list of $_f$**format** arguments, respectively, of *condition*.

$_v$***break-on-signals***$_{\boxed{\texttt{NIL}}}$
▷ Condition type debugger is to be invoked on.

$_v$***debugger-hook***$_{\boxed{\texttt{NIL}}}$
▷ Function of condition and function itself. Called before debugger.

12 Types and Classes

For any class, there is always a corresponding type of the same name.

($_f$**typep** *foo type* [*environment*$_{\boxed{\texttt{NIL}}}$]) ▷ T if *foo* is of *type*.

($_f$**subtypep** *type-a type-b* [*environment*])
▷ Return T if *type-a* is a recognizable subtype of *type-b*, and $\underset{2}{\texttt{NIL}}$ if the relationship could not be determined.

($_s$**the** $\widehat{type}$ *form*) ▷ Declare values of *form* to be of *type*.

($_f$**coerce** *object type*) ▷ Coerce object into *type*.

($_m$**typecase** *foo* ($\widehat{type}$ *a-form*^{P*})* [($\left\{\begin{array}{l} \textbf{otherwise} \\ \texttt{T} \end{array}\right\}$ *b-form*$_{\boxed{\texttt{NIL}}}^{P*}$)])
▷ Return values of the first *a-form** whose *type* is *foo* of. Return values of *b-forms* if no *type* matches.

$\left(\left\{\begin{array}{l} {}_m\textbf{etypecase} \\ {}_m\textbf{ctypecase} \end{array}\right\}\right.$ *foo* ($\widehat{type}$ *form*^{P*})*)
▷ Return values of the first *form** whose *type* is *foo* of. Signal non-correctable/correctable **type-error** if no *type* matches.

($_f$**type-of** *foo*) ▷ Type of *foo*.

($_m$**check-type** *place type* [*string*$_{\boxed{\{\texttt{a}\mid\texttt{an}\}\ \textit{type}}}$])
▷ Signal correctable **type-error** if *place* is not of *type*. Return NIL.

($_f$**stream-element-type** *stream*) ▷ Type of *stream* objects.

($_f$**array-element-type** *array*) ▷ Element type *array* can hold.

($_f$**upgraded-array-element-type** *type* [*environment*$_{\boxed{\texttt{NIL}}}$])
▷ Element type of most specialized array capable of holding elements of *type*.

($_m$**with-simple-restart** ($\begin{Bmatrix}\textit{restart}\\ \texttt{NIL}\end{Bmatrix}$ *control arg**) *form*$^{\text{P}*}$)

▷ Return $\underline{\text{values of } \textit{forms}}$ unless *restart* is called during their evaluation. In this case, describe *restart* using $_f$**format** *control* and *arg*s (see page 38) and return $\underline{\texttt{NIL}}$ and $\underset{2}{\underline{\texttt{T}}}$.

($_m$**restart-case** *form* (*restart* (*ord-λ**) $\left\{\begin{vmatrix}\textbf{:interactive}\ \textit{arg-function}\\ \textbf{:report}\ \begin{Bmatrix}\textit{report-function}\\ \textit{string}_{\boxed{\texttt{"}\textit{restart}\texttt{"}}}\end{Bmatrix}\\ \textbf{:test}\ \textit{test-function}_{\boxed{\texttt{T}}}\end{vmatrix}\right\}$

(**declare** $\widehat{\textit{decl}^*}$)* *restart-form*$^{\text{P}*}$)*)

▷ Return $\underline{\text{values of } \textit{form}}$ or, if during evaluation of *form* one of the dynamically established *restart*s is called, the $\underline{\text{values of its } \textit{restart-forms}}$. A *restart* is visible under *condition* if (**funcall #'***test-function condition*) returns `T`. If presented in the debugger, *restart*s are described by *string* or by **#'***report-function* (of a stream). A *restart* can be called by (**invoke-restart** *restart arg**), where *arg*s match *ord-λ**, or by (**invoke-restart-interactively** *restart*) where a list of the respective *arg*s is supplied by **#'***arg-function*. See page 18 for *ord-λ**.

($_m$**restart-bind** (($\widehat{\begin{Bmatrix}\textit{restart}\\ \texttt{NIL}\end{Bmatrix}}$ *restart-function*

$\left\{\begin{vmatrix}\textbf{:interactive-function}\ \textit{arg-function}\\ \textbf{:report-function}\ \textit{report-function}\\ \textbf{:test-function}\ \textit{test-function}\end{vmatrix}\right\}$)*) *form*$^{\text{P}*}$)

▷ Return $\underline{\text{values of } \textit{forms}}$ evaluated with dynamically established *restarts* whose *restart-function*s should perform a non-local transfer of control. A restart is visible under *condition* if (*test-function condition*) returns `T`. If presented in the debugger, *restart*s are described by *restart-function* (of a stream). A *restart* can be called by (**invoke-restart** *restart arg**), where *arg*s must be suitable for the corresponding *restart-function*, or by (**invoke-restart-interactively** *restart*) where a list of the respective *arg*s is supplied by *arg-function*.

($_f$**invoke-restart** *restart arg**)
($_f$**invoke-restart-interactively** *restart*)

▷ Call function associated with *restart* with arguments given or prompted for, respectively. If *restart* function returns, return $\underline{\text{its values}}$.

($\begin{Bmatrix}{}_f\textbf{find-restart}\\ {}_f\textbf{compute-restarts}\ \textit{name}\end{Bmatrix}$ [*condition*])

▷ Return innermost $\underline{\text{restart}}$ *name*, or a $\underline{\text{list of all restarts}}$, respectively, out of those either associated with *condition* or un-associated at all; or, without *condition*, out of all restarts. Return $\underline{\texttt{NIL}}$ if search is unsuccessful.

($_f$**restart-name** *restart*) ▷ $\underline{\text{Name of } \textit{restart}}$.

($\begin{Bmatrix}{}_f\textbf{abort}\\ {}_f\textbf{muffle-warning}\\ {}_f\textbf{continue}\\ {}_f\textbf{store-value}\ \textit{value}\\ {}_f\textbf{use-value}\ \textit{value}\end{Bmatrix}$ [*condition*$_{\boxed{\texttt{NIL}}}$])

▷ Transfer control to innermost applicable restart with same name (i.e. **abort**, ..., **continue** ...) out of those either associated with *condition* or un-associated at all; or, without *condition*, out of all restarts. If no restart is found, signal **control-error** for $_f$**abort** and $_f$**muffle-warning**, or return $\underline{\texttt{NIL}}$ for the rest.

($_m$**with-condition-restarts** *condition restarts form*$^{\text{P}*}$)

▷ Evaluate *forms* with *restarts* dynamically associated with *condition*. Return $\underline{\text{values of } \textit{forms}}$.

($_f$**arithmetic-error-operation** *condition*)
($_f$**arithmetic-error-operands** *condition*)

▷ $\underline{\text{List of function}}$ or $\underline{\text{of its operands}}$ respectively, used in the operation which caused *condition*.

$\left\{\begin{vmatrix}\textbf{:argument-precedence-order}\ \textit{required-var}^+\\ \textbf{:declare}\ (\textbf{optimize}\ \textit{method-selection-optimization})\\ \textbf{:documentation}\ \textit{string}\\ \textbf{:generic-function-class}\ \textit{gf-class}\\ \textbf{:method-class}\ \textit{method-class}\\ \textbf{:method-combination}\ \textit{c-type}\ \textit{c-arg}^*\\ \textbf{:lambda-list}\ \textit{lambda-list}\\ \textbf{:environment}\ \textit{environment}\end{vmatrix}\right\}$)

▷ Define or modify $\underline{\text{generic function}}$ *foo*. *gf-class* and *lambda-list* must be compatible with a pre-existing generic function or with existing methods, respectively. Changes to *method-class* do not propagate to existing methods. For *c-type* see section 10.3.

($_m$**defmethod** $\begin{Bmatrix}\textit{foo}\\ (\textbf{setf}\ \textit{foo})\end{Bmatrix}$ [$\begin{Bmatrix}\textbf{:before}\\ \textbf{:after}\\ \textbf{:around}\\ \textit{qualifier}^*\end{Bmatrix}_{\boxed{\text{primary method}}}$]

($\begin{Bmatrix}\textit{var}\\ (\textit{spec-var}\ \begin{Bmatrix}\textit{class}\\ (\textbf{eql}\ \textit{bar})\end{Bmatrix})\end{Bmatrix}^*$ [**&optional**

$\begin{Bmatrix}\textit{var}\\ (\textit{var}\ [\textit{init}\ [\textit{supplied-p}]])\end{Bmatrix}^*$] [**&rest** *var*] [**&key**

$\begin{Bmatrix}\textit{var}\\ (\begin{Bmatrix}\textit{var}\\ (\textit{:key}\ \textit{var})\end{Bmatrix}[\textit{init}\ [\textit{supplied-p}]])\end{Bmatrix}^*$ [**&allow-other-keys**]]

[**&aux** $\begin{Bmatrix}\textit{var}\\ (\textit{var}\ [\textit{init}])\end{Bmatrix}^*$]) $\left\{\begin{vmatrix}(\textbf{declare}\ \widehat{\textit{decl}^*})^*\\ \widehat{\textit{doc}}\end{vmatrix}\right\}$ *form*$^{\text{P}*}$)

▷ Define $\underline{\text{new method}}$ for generic function *foo*. *spec-var*s specialize to either being of *class* or being **eql** *bar*, respectively. On invocation, *var*s and *spec-var*s of the $\underline{\text{new method}}$ act like parameters of a function with body *form**. *forms* are enclosed in an implicit $_s$**block** *foo*. Applicable *qualifier*s depend on the **method-combination** type; see section 10.3.

($\begin{Bmatrix}{}_g\textbf{add-method}\\ {}_g\textbf{remove-method}\end{Bmatrix}$ *generic-function method*)

▷ Add (if necessary) or remove (if any) *method* to/from $\underline{\textit{generic-function}}$.

($_g$**find-method** *generic-function qualifiers specializers* [*error*$_{\boxed{\texttt{T}}}$])

▷ Return suitable $\underline{\text{method}}$, or signal **error**.

($_g$**compute-applicable-methods** *generic-function args*)

▷ $\underline{\text{List of methods}}$ suitable for *args*, most specific first.

($_f$**call-next-method** *arg*$^*_{\boxed{\text{current args}}}$)

▷ From within a method, call next method with *args*; return $\underline{\text{its values}}$.

($_g$**no-applicable-method** *generic-function arg**)

▷ Called on invocation of *generic-function* on *args* if there is no applicable method. Default method signals **error**. Not to be called by user.

($\begin{Bmatrix}{}_f\textbf{invalid-method-error}\ \textit{method}\\ {}_f\textbf{method-combination-error}\end{Bmatrix}$ *control arg**)

▷ Signal **error** on applicable method with invalid qualifiers, or on method combination. For *control* and *args* see **format**, page 38.

($_g$**no-next-method** *generic-function method arg**)

▷ Called on invocation of **call-next-method** when there is no next method. Default method signals **error**. Not to be called by user.

($_g$**function-keywords** *method*)

▷ Return list of $\underline{\text{keyword parameters}}$ of *method* and $\underset{2}{\underline{\texttt{T}}}$ if other keys are allowed.

($_g$**method-qualifiers** *method*) ▷ $\underline{\text{List of qualifiers}}$ of *method*.

10.3 Method Combination Types

standard

▷ Evaluate most specific **:around** method supplying the values of the generic function. From within this method, $_f$**call-next-method** can call less specific **:around** methods if there are any. If not, or if there are no **:around** methods at all, call all **:before** methods, most specific first, and the most specific primary method which supplies the values of the calling $_f$**call-next-method** if any, or of the generic function; and which can call less specific primary methods via $_f$**call-next-method**. After its return, call all **:after** methods, least specific first.

and|**or**|**append**|**list**|**nconc**|**progn**|**max**|**min**|**+**

▷ Simple built-in **method-combination** types; have the same usage as the *c-type*s defined by the short form of $_m$**define-method-combination**.

($_m$**define-method-combination** *c-type*
{ | **:documentation** $\widehat{string}$
| **:identity-with-one-argument** *bool*$_{\boxed{\text{NIL}}}$
| **:operator** *operator*$_{\boxed{c\text{-}type}}$ })

▷ **Short Form.** Define new **method-combination** *c-type*. In a generic function using *c-type*, evaluate most specific **:around** method supplying the values of the generic function. From within this method, $_f$**call-next-method** can call less specific **:around** methods if there are any. If not, or if there are no **:around** methods at all, return from the calling **call-next-method** or from the generic function, respectively, the values of (*operator* (*primary-method gen-arg**)*), *gen-arg** being the arguments of the generic function. The *primary-methods* are ordered [{**:most-specific-first** / **:most-specific-last**}$_{\boxed{\text{:most-specific-first}}}$] (specified as *c-arg* in $_m$**defgeneric**). Using *c-type* as the *qualifier* in $_m$**defmethod** makes the method primary.

($_m$**define-method-combination** *c-type* (*ord-λ**) ((*group*
{ ***** / (*qualifier** [*****]) / *predicate* }
{ | **:description** *control*
| **:order** {**:most-specific-first** / **:most-specific-last**}$_{\boxed{\text{:most-specific-first}}}$
| **:required** *bool* })*)
{ | (**:arguments** *method-combination-λ**)
| (**:generic-function** *symbol*)
| (**declare** $\widehat{decl}$*)*
| $\widehat{doc}$ } *body*^{P}*)

▷ **Long Form.** Define new **method-combination** *c-type*. A call to a generic function using *c-type* will be equivalent to a call to the forms returned by *body** with *ord-λ** bound to *c-arg** (cf. $_m$**defgeneric**), with *symbol* bound to the generic function, with *method-combination-λ** bound to the arguments of the generic function, and with *group*s bound to lists of methods. An applicable method becomes a member of the leftmost *group* whose *predicate* or *qualifier*s match. Methods can be called via $_m$**call-method**. Lambda lists (*ord-λ**) and (*method-combination-λ**) according to *ord-λ* on page 18, the latter enhanced by an optional **&whole** argument.

($_m$**call-method**
{ $\widehat{method}$ / ($_m$**make-method** $\widehat{form}$) } [({ $\widehat{next\text{-}method}$ / ($_m$**make-method** $\widehat{form}$) }*)])

▷ From within an effective method form, call *method* with the arguments of the generic function and with information about its *next-methods*; return its values.

11 Conditions and Errors

For standardized condition types cf. Figure 2 on page 32.

($_m$**define-condition** *foo* (*parent-type**$_{\boxed{\text{condition}}}$)
({ *slot*
| (*slot* { | {**:reader** *reader*}*
| {**:writer** {*writer* / (**setf** *writer*)}}*
| {**:accessor** *accessor*}*
| **:allocation** {**:instance** / **:class**}$_{\boxed{\text{:instance}}}$
| {**:initarg** **:***initarg-name*}*
| **:initform** *form*
| **:type** *type*
| **:documentation** *slot-doc* }) }*)
{ | (**:default-initargs** {*name value*}*)
| (**:documentation** *condition-doc*)
| (**:report** {*string* / *report-function*}) })

▷ Define, as a subtype of *parent-type*s, condition type *foo*. In a new condition, a *slot*'s value defaults to *form* unless set via **:***initarg-name*; it is readable via (*reader i*) or (*accessor i*), and writable via (*writer value i*) or (**setf** (*accessor i*) *value*). With **:allocation :class**, *slot* is shared by all conditions of type *foo*. A condition is reported by *string* or by *report-function* of arguments condition and stream.

($_f$**make-condition** *condition-type* {**:***initarg-name value*}*)

▷ Return new instance of *condition-type*.

({ $_f$**signal** / $_f$**warn** / $_f$**error** } { *condition* / *condition-type* {**:***initarg-name value*}* / *control arg** })

▷ Unless handled, signal as **condition**, **warning** or **error**, respectively, *condition* or a new instance of *condition-type* or, with $_f$**format** *control* and *args* (see page 38), **simple-condition**, **simple-warning**, or **simple-error**, respectively. From $_f$**signal** and $_f$**warn**, return NIL.

($_f$**cerror** *continue-control* { *condition continue-arg** / *condition-type* {**:***initarg-name value*}* / *control arg** })

▷ Unless handled, signal as correctable **error** *condition* or a new instance of *condition-type* or, with $_f$**format** *control* and *args* (see page 38), **simple-error**. In the debugger, use $_f$**format** arguments *continue-control* and *continue-args* to tag the continue option. Return NIL.

($_m$**ignore-errors** *form*^{P}*)

▷ Return values of *forms* or, in case of **error**s, NIL and the condition.

($_f$**invoke-debugger** *condition*)

▷ Invoke debugger with *condition*.

($_m$**assert** *test* [(*place**) [{ *condition continue-arg** / *condition-type* {**:***initarg-name value*}* / *control arg** }]])

▷ If *test*, which may depend on *place*s, returns NIL, signal as correctable **error** *condition* or a new instance of *condition-type* or, with $_f$**format** *control* and *args* (see page 38), **error**. When using the debugger's continue option, *places* can be altered before re-evaluation of *test*. Return NIL.

($_m$**handler-case** *foo* (*type* ([*var*]) (**declare** $\widehat{decl}$*)* *condition-form*^{P}*)*
[(**:no-error** (*ord-λ**) (**declare** $\widehat{decl}$*)* *form*^{P}*)])

▷ If, on evaluation of *foo*, a condition of *type* is signalled, evaluate matching *condition-forms* with *var* bound to the condition, and return their values. Without a condition, bind *ord-λ*s to values of *foo* and return values of *forms* or, without a **:no-error** clause, return values of *foo*. See page 18 for (*ord-λ**).

($_m$**handler-bind** ((*condition-type handler-function*)*) *form*^{P}*)

▷ Return values of *forms* after evaluating them with *condition-types* dynamically bound to their respective *handler-functions* of argument condition.