

Quick Reference

lisp

Common

lisp

Bert Burgemeister

Contents

| | | | | |
|----------|------------------------------------|-----------|---|-----------|
| 1 | Numbers | 3 | 9.5 Control Flow | 21 |
| 1.1 | Predicates | 3 | 9.6 Iteration | 22 |
| 1.2 | Numeric Functions | 3 | 9.7 Loop Facility | 22 |
| 1.3 | Logic Functions | 5 | 10 CLOS | 25 |
| 1.4 | Integer Functions | 6 | 10.1 Classes | 25 |
| 1.5 | Implementation-Dependent | 6 | 10.2 Generic Functions | 26 |
| | | | 10.3 Method Combination Types | 28 |
| 2 | Characters | 7 | 11 Conditions and Errors | 29 |
| 3 | Strings | 8 | 12 Types and Classes | 31 |
| 4 | Conses | 8 | 13 Input/Output | 33 |
| 4.1 | Predicates | 8 | 13.1 Predicates | 33 |
| 4.2 | Lists | 9 | 13.2 Reader | 34 |
| 4.3 | Association Lists | 10 | 13.3 Character Syntax | 35 |
| 4.4 | Trees | 10 | 13.4 Printer | 36 |
| 4.5 | Sets | 11 | 13.5 Format | 38 |
| | | | 13.6 Streams | 41 |
| | | | 13.7 Pathnames and Files | 42 |
| 5 | Arrays | 11 | 14 Packages and Symbols | 44 |
| 5.1 | Predicates | 11 | 14.1 Predicates | 44 |
| 5.2 | Array Functions | 11 | 14.2 Packages | 44 |
| 5.3 | Vector Functions | 12 | 14.3 Symbols | 45 |
| | | | 14.4 Standard Packages | 46 |
| 6 | Sequences | 12 | 15 Compiler | 46 |
| 6.1 | Sequence Predicates | 12 | 15.1 Predicates | 46 |
| 6.2 | Sequence Functions | 13 | 15.2 Compilation | 46 |
| | | | 15.3 REPL and Debugging | 48 |
| | | | 15.4 Declarations | 49 |
| 7 | Hash Tables | 15 | 16 External Environment | 49 |
| 8 | Structures | 16 | | |
| 9 | Control Structure | 16 | | |
| 9.1 | Predicates | 16 | | |
| 9.2 | Variables | 17 | | |
| 9.3 | Functions | 18 | | |
| 9.4 | Macros | 19 | | |

Typographic Conventions

name; *f***name**; *g***name**; *m***name**; *s***name**; *v****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**bar*] ▷ Either one *foo* or nothing; defaults to *bar*.

*foo**; {*foo*}* ▷ Zero or more *foos*.

foo+; {*foo*}+ ▷ One or more *foos*.

foos ▷ English plural denotes a list argument.

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

$\begin{cases} \textit{foo} \\ \textit{bar} \\ \textit{baz} \end{cases}$ ▷ Anything from none to each of *foo*, *bar*, and *baz*.

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

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

foo^{**P**} ▷ *foo** is evaluated as in **sprogn**; see page 21.

foo; *bar*; *baz*_{*n*} ▷ Primary, secondary, and *n*th return value.

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

1 Numbers

1.1 Predicates

- $(f = number^+)$
 $(f \neq number^+)$
- ▷ T if all *numbers*, or none, respectively, are equal in value.
- $(f > number^+)$
 $(f \geq number^+)$
 $(f < number^+)$
 $(f \leq number^+)$
- ▷ Return T if *numbers* are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively.
- $(f \text{minusp } a)$
 $(f \text{zerop } a)$ ▷ T if $a < 0$, $a = 0$, or $a > 0$, respectively.
 $(f \text{plusp } a)$
- $(f \text{evenp } int)$ ▷ T if *int* is even or odd, respectively.
 $(f \text{oddp } int)$
- $(f \text{numberp } foo)$
 $(f \text{realp } foo)$
 $(f \text{rationalp } foo)$
 $(f \text{floatp } foo)$ ▷ T if *foo* is of indicated type.
 $(f \text{integerp } foo)$
 $(f \text{complexp } foo)$
 $(f \text{random-state-p } foo)$

1.2 Numeric Functions

- $(f + a_{\square}^*)$ ▷ Return $\sum a$ or $\prod a$, respectively.
 $(f * a_{\square}^*)$
- $(f - a b^*)$
 $(f / a b^*)$
- ▷ Return $a - \sum b$ or $a / \prod b$, respectively. Without any *bs*, return $-a$ or $1/a$, respectively.
- $(f 1+ a)$ ▷ Return $a + 1$ or $a - 1$, respectively.
 $(f 1- a)$
- $(\left\{ \begin{matrix} m \text{incf} \\ m \text{decf} \end{matrix} \right\} \widetilde{place} [delta_{\square}])$
- ▷ Increment or decrement the value of *place* by *delta*. Return new value.
- $(f \text{exp } p)$ ▷ Return e^p or b^p , respectively.
 $(f \text{expt } b p)$
- $(f \text{log } a [b_{\square}])$ ▷ Return $\log_b a$ or, without *b*, $\ln a$.
- $(f \text{sqr t } n)$ ▷ \sqrt{n} in complex numbers/natural numbers.
 $(f \text{isqr t } n)$
- $(f \text{lcm } integer^*_{\square})$
 $(f \text{gcd } integer^*)$
- ▷ Least common multiple or greatest common denominator, respectively, of *integers*. (**gcd**) returns 0.
- `cpi` ▷ **long-float** approximation of π , Ludolph's number.
- $(f \text{sin } a)$
 $(f \text{cos } a)$ ▷ $\sin a$, $\cos a$, or $\tan a$, respectively. (*a* in radians.)
 $(f \text{tan } a)$
- $(f \text{asin } a)$ ▷ $\arcsin a$ or $\arccos a$, respectively, in radians.
 $(f \text{acos } a)$

(*f*atan *a* [*b*]) ▷ $\arctan \frac{a}{b}$ in radians.

(*f*sinh *a*)
 (*f*cosh *a*) ▷ sinh *a*, cosh *a*, or tanh *a*, respectively.
 (*f*tanh *a*)

(*f*asinh *a*)
 (*f*acosh *a*) ▷ asinh *a*, acosh *a*, or atanh *a*, respectively.
 (*f*atanh *a*)

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

(*f*conjugate *a*) ▷ Return complex conjugate of *a*.

(*f*max *num*⁺)
 (*f*min *num*⁺) ▷ Greatest or least, respectively, of *nums*.

($\left. \begin{array}{l} \{ \textit{fround} | \textit{fround} \} \\ \{ \textit{ffloor} | \textit{ffloor} \} \\ \{ \textit{fceil} | \textit{fceil} \} \\ \{ \textit{ftruncate} | \textit{ftruncate} \} \end{array} \right\} n$ [*d*])
 ▷ Return as **integer** or **float**, respectively, n/d rounded, or rounded towards $-\infty$, $+\infty$, or 0, respectively; and remainder.

($\left. \begin{array}{l} \{ \textit{fmod} \} \\ \{ \textit{frem} \} \end{array} \right\} n$ *d*)
 ▷ Same as **f**floor or **f**truncate, respectively, but return remainder only.

(*f*random *limit* [*state* [**random-state**]])
 ▷ Return non-negative random number less than *limit*, and of the same type.

(*f*make-random-state [*state* [NIL|T] [NIL]])
 ▷ Copy of **random-state** object *state* or of the current random state; or a randomly initialized fresh random state.

random-state ▷ Current random state.

(*f*float-sign *num-a* [*num-b*]) ▷ num-b with *num-a*'s sign.

(*f*signum *n*)
 ▷ Number of magnitude 1 representing sign or phase of *n*.

(*f*numerator *rational*)
 (*f*denominator *rational*)
 ▷ Numerator or denominator, respectively, of *rational*'s canonical form.

(*f*realpart *number*)
 (*f*imagpart *number*)
 ▷ Real part or imaginary part, respectively, of *number*.

(*f*complex *real* [*imag*]) ▷ Make a complex number.

(*f*phase *num*) ▷ Angle of *num*'s polar representation.

(*f*abs *n*) ▷ Return |*n*|.

(*f*rational *real*)
 (*f*rationalize *real*)
 ▷ Convert *real* to rational. Assume complete/limited accuracy for *real*.

(*f*float *real* [*prototype* [*type*]])
 ▷ Convert *real* into float with type of *prototype*.

| | | | |
|--------------------------------|-------------------------------|--------------------------------|--------------------|
| WHEN 21, 24 | TABLE-ITERATOR 15 | WITH- SIMPLE-RESTART 30 | WRITE-STRING 36 |
| WHILE 25 | WITH-INPUT- FROM-STRING 42 | WITH-SLOTS 26 | WRITE-TO-STRING 37 |
| WILD-PATHNAME-P 33 | WITH-OPEN-FILE 42 | WITH-STANDARD- IO-SYNTAX 34 | |
| WITH 22 | WITH-OPEN-STREAM 42 | WRITE 37 | Y-OR-N-P 34 |
| WITH-ACCESSORS 26 | WITH-OUTPUT- TO-STRING 42 | WRITE-BYTE 36 | YES-OR-NO-P 34 |
| WITH-COMPILED- UNIT 47 | WITH-PACKAGE- ITERATOR 45 | WRITE-CHAR 36 | |
| WITH-CONDITION- RESTARTS 31 | | WRITE-LINE 36 | |
| WITH-HASH- | | WRITE-SEQUENCE 37 | ZEROP 3 |

NINTH 9
 NO-APPLICABLE-METHOD 27
 NO-NEXT-METHOD 27
 NOT 17, 33, 36
 NOTANY 13
 NOTEVERY 12
 NOTINLINE 49
 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 19
 NTHCDR 9
 NULL 8, 32
 NUMBER 32
 NUMBERP 3
 NUMERATOR 4
 NUNION 11

 ODDP 3
 OF 24
 OF-TYPE 22
 ON 24
 OPEN 41
 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 44
 PAIRLIS 10
 PARSE-ERROR 32
 PARSE-INTEGERS 8
 PARSE-NAMESTRING 43
 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 14
 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 49
 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 34
 READ-BYTE 34
 READ-CHAR 34
 READ-CHAR-NO-HANG 34
 READ-DELIMITED-LIST 34
 READ-FROM-STRING 34
 READ-LINE 34
 READ-PRESERVING-WHITESPACE 34
 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 25
 REPLACE 15
 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 35
 SET-PPRINT-DISPATCH 38
 SET-SYNTAX-FROM-CHAR 34
 SETF 17, 46
 SETQ 17
 SEVENTH 9
 SHADOW 45
 SHADOWING-IMPORT 45
 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-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-BOUND 25
 SLOT-EXISTS-P 25
 SLOT-MAKUNBOUND 25
 SLOT-MISSING 26
 SLOT-UNBOUND 26
 SLOT-VALUE 25
 SOFTWARE-TYPE 49
 SOFTWARE-VERSION 49
 SOME 13
 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 33
 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 46
 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 16
 SYMBOL 24, 32, 45
 SYMBOL-FUNCTION 46
 SYMBOL-MACROLET 20
 SYMBOL-NAME 46
 SYMBOL-PACKAGE 46
 SYMBOL-PLIST 46
 SYMBOL-VALUE 46
 SYMBOLP 44
 SYMBOLS 24
 SYNONYM-STREAM 32
 SYNONYM-STREAM-SYMBOL 41

 T 2, 32, 46
 TAGBODY 22
 TAILP 9
 TAN 3
 TANH 4
 TENTH 9
 TERPRI 36
 THE 24, 31
 THEN 24
 THEREIS 25
 THIRD 9
 THROW 22
 TIME 48
 TO 24
 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 46, 49
 TYPE-ERROR 32
 TYPE-ERROR-DATUM 31
 TYPE-ERROR-EXPECTED-TYPE 31
 TYPE-OF 33
 TYPECASE 31
 TYPEP 31

 UNBOUND-SLOT 32
 UNBOUND-SLOT-INSTANCE 31
 UNBOUND-VARIABLE 32
 UNDEFINED-FUNCTION 32
 UNEXPORT 45
 UNINTERN 45
 UNION 11
 UNLESS 21, 24
 UNREAD-CHAR 34
 UNSIGNED-BYTE 32
 UNTIL 25
 UNTRACE 48
 UNUSE-PACKAGE 44
 UNWIND-PROTECT 21
 UPDATE-INSTANCE-FOR-DIFFERENCE-CLASS 26
 UPDATE-INSTANCE-FOR-REDEFINED-CLASS 26
 UPFROM 24
 UPGRADED-ARRAY-ELEMENT-TYPE 33
 UPGRADED-COMPLEX-PART-TYPE 6
 UPPER-CASE-P 7
 UPTO 24
 USE-PACKAGE 44
 USE-VALUE 30
 USER-HOMEDIR-PATHNAME 43
 USING 24

 V 40
 VALUES 18, 33
 VALUES-LIST 18
 VARIABLE 46
 VECTOR 12, 32
 VECTOR-POP 12
 VECTOR-PUSH 12
 VECTOR-PUSH-EXTEND 12
 VECTORP 11

 WARN 29
 WARNING 32

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. operations are

cboole-1 ▷ *int-a*.

cboole-2 ▷ *int-b*.

cboole-c1 ▷ \neg *int-a*.

cboole-c2 ▷ \neg *int-b*.

cboole-set ▷ All bits set.

cboole-clr ▷ All bits zero.

cboole-eqv ▷ $int-a \equiv int-b$.

cboole-and ▷ $int-a \wedge int-b$.

cboole-andc1 ▷ $\neg int-a \wedge int-b$.

cboole-andc2 ▷ $int-a \wedge \neg int-b$.

cboole-nand ▷ $\neg(int-a \wedge int-b)$.

cboole-ior ▷ $int-a \vee int-b$.

cboole-orc1 ▷ $\neg int-a \vee int-b$.

cboole-orc2 ▷ $int-a \vee \neg int-b$.

cboole-xor ▷ $\neg(int-a \equiv int-b)$.

cboole-nor ▷ $\neg(int-a \vee int-b)$.

(*f* **lognot** integer) ▷ \neg integer.

(*f* **logeqv** integer*)

(*f* **logand** integer*)

▷ Return value of exclusive-nored or anded integers, respectively. Without any integer, return -1.

(*f* **logandc1** *int-a int-b*) ▷ $\neg int-a \wedge int-b$.

(*f* **logandc2** *int-a int-b*) ▷ $int-a \wedge \neg int-b$.

(*f* **lognand** *int-a int-b*) ▷ $\neg(int-a \wedge int-b)$.

(*f* **logxor** integer*)

(*f* **logior** integer*)

▷ Return value of exclusive-ored or ored integers, respectively. Without any integer, return 0.

(*f* **logorc1** *int-a int-b*) ▷ $\neg int-a \vee int-b$.

(*f* **logorc2** *int-a int-b*) ▷ $int-a \vee \neg int-b$.

(*f* **lognor** *int-a int-b*) ▷ $\neg(int-a \vee int-b)$.

(*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 *int* ≥ 0 , number of 0 bits in *int* < 0 .

1.4 Integer Functions

(*finteger-length integer*)

▷ Number of bits necessary to represent *integer*.

(*fldb-test byte-spec integer*)

▷ Return T if any bit specified by *byte-spec* in *integer* is set.

(*fash integer count*)

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

(*fldb byte-spec integer*)

▷ Extract byte denoted by *byte-spec* from *integer*. **setfable**.

{*fdeposit-field* } *int-a byte-spec int-b*
{*fdpb* }

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

(*fmask-field byte-spec integer*)

▷ Return copy of *integer* with all bits unset but those denoted by *byte-spec*. **setfable**.

(*rbyte size position*)

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

(*rbyte-size byte-spec*)

(*rbyte-position byte-spec*)

▷ Size or position, respectively, of *byte-spec*.

1.5 Implementation-Dependent

cshort-float }
csingle-float } { epsilon
cdouble-float } { negative-epsilon
clong-float }

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

cleast-negative }
cleast-negative-normalized } { short-float
cleast-positive } { single-float
cleast-positive-normalized } { double-float
 } { long-float

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

cmost-negative }
cmost-positive } { short-float
 } { single-float
 } { double-float
 } { long-float
 } { fixnum

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

(*fdecode-float n*)

(*finteger-decode-float n*)

▷ Return significant, exponent, and sign of **float** *n*.

(*rscale-float n i*)

▷ With *n*'s radix *b*, return nb^i .

(*rfloat-radix n*)

(*rfloat-digits n*)

(*rfloat-precision n*)

▷ Radix, number of digits in that radix, or precision in that radix, respectively, of float *n*.

(*rupgraded-complex-part-type foo [environment_{NTT}]*)

▷ Type of most specialized **complex** number able to hold parts of type *foo*.

DISASSEMBLE 48
DIVISION-BY-ZERO 32
DO 22, 24
DO-ALL-SYMBOLS 45
DO-EXTERNAL-SYMBOLS 45
DO-SYMBOLS 45
DO* 22
DOCUMENTATION 46
DOING 24
DOLIST 22
DOTIMES 22
DOUBLE-FLOAT 32, 35
DOUBLE-FLOAT-EPSILON 6
DOUBLE-FLOAT-NEGATIVE-EPSILON 6
DOWNFROM 24
DOWNTO 24
DPB 6
DRIBBLE 48
DYNAMIC-EXTENT 49

EACH 24
ECASE 21
ECHO-STREAM 32
ECHO-STREAM-INPUT-STREAM 41
ECHO-STREAM-OUTPUT-STREAM 41
ED 48
EIGHTH 9
ELSE 24
ELT 13
ENCODE-UNIVERSAL-TIME 49
END 24
END-OF-FILE 32
ENDP 8
ENOUGH-NAMESTRING 43
ENSURE-DIRECTORIES-EXIST 44
ENSURE-GENERIC-FUNCTION 27
EQ 16
EQL 16, 33
EQUAL 16
EQUALP 16
ERROR 29, 32
ETYPESCASE 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 25
FIND 14
FIND-ALL-SYMBOLS 45
FIND-CLASS 25
FIND-IF 14
FIND-IF-NOT 14
FIND-METHOD 27
FIND-PACKAGE 45
FIND-RESTART 30
FIND-SYMBOL 45
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 24
FROUND 4
FTRUNCATE 4
FTYPE 49
FUNCALL 18
FUNCTION 18, 32, 35, 46
FUNCTION-KEYWORDS 28
FUNCTION-LAMBDA-EXPRESSION 19
FUNCTIONP 17

GCD 3
GENERIC-FUNCTION 32
GENSYM 46
GENTEMP 46
GET 17
GET-DECODED-TIME 49
GET-DISPATCH-MACRO-CHARACTER 35
GET-INTERNAL-REAL-TIME 49
GET-INTERNAL-RUN-TIME 49
GET-MACRO-CHARACTER 35
GET-OUTPUT-STREAM-STRING 41
GET-PROPERTIES 17
GET-SETF-EXPANSION 20
GET-UNIVERSAL-TIME 49
GETF 17
GETHASH 15
GO 22
GRAPHIC-CHAR-P 7

HANDLER-BIND 30
HANDLER-CASE 30
HASH-KEY 24
HASH-KEYS 24
HASH-TABLE 32
HASH-TABLE-COUNT 15
HASH-TABLE-P 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 49
IGNORE 49
IGNORE-ERRORS 29
IMAGPART 4
IMPORT 45
IN 24
IN-PACKAGE 44
INCF 3
INITIALIZE-INSTANCE 26
INITIALLY 25
INLINE 49
INPUT-STREAM-P 33
INSPECT 48
INTEGER 32
INTEGER-DECODE-FLOAT 6
INTEGER-LENGTH 6
INTEGERP 3
INTERACTIVE-STREAM-P 33
INTERN 45
INTERNAL-TIME-UNITS-PER-SECOND 49
INTERSECTION 11
INTO 24
INVALID-METHOD-ERROR 27
INVOKE-DEBUGGER 29
INVOKE-RESTART 30
INVOKE-RESTART-INTERACTIVELY 30
ISORT 3
IT 24

KEYWORD 32, 44, 46
KEYWORDP 44

LABELS 18

LAMBDA 18
LAMBDA-CONCATENATED-STREAM 41
LAMBDA-MAKE-CONDITION 29
LAMBDA-MAKE-DISPATCH-MACRO-CHARACTER 35
LAMBDA-MAKE-ECHO-STREAM 41
LAMBDA-LDB 6
LAMBDA-LDB-TEST 6
LAMBDA-LDIFF 9
LAMBDA-LEAST-NEGATIVE-DOUBLE-FLOAT 6
LAMBDA-LEAST-NEGATIVE-LONG-FLOAT 6
LAMBDA-LEAST-NEGATIVE-NORMALIZED-DOUBLE-FLOAT 6
LAMBDA-LEAST-NEGATIVE-NORMALIZED-LONG-FLOAT 6
LAMBDA-LEAST-NEGATIVE-NORMALIZED-SHORT-FLOAT 6
LAMBDA-LEAST-NEGATIVE-NORMALIZED-DOUBLE-FLOAT 6
LAMBDA-LEAST-NEGATIVE-NORMALIZED-SHORT-FLOAT 6
LAMBDA-LEAST-POSITIVE-DOUBLE-FLOAT 6
LAMBDA-LEAST-POSITIVE-LONG-FLOAT 6
LAMBDA-LEAST-POSITIVE-NORMALIZED-DOUBLE-FLOAT 6
LAMBDA-LEAST-POSITIVE-NORMALIZED-LONG-FLOAT 6
LAMBDA-LEAST-POSITIVE-NORMALIZED-SHORT-FLOAT 6
LAMBDA-LEAST-POSITIVE-SINGLE-FLOAT 6
LAMBDA-LENGTH 13
LAMBDA-LET 18
LAMBDA-LET* 18
LAMBDA-LISP-IMPLEMENTATION-TYPE 49
LAMBDA-LISP-IMPLEMENTATION-VERSION 49
LAMBDA-LIST 9, 28, 32
LAMBDA-LIST-ALL-PACKAGES 44
LAMBDA-LIST-LENGTH 9
LAMBDA-LIST* 9
LAMBDA-LISTEN 41
LAMBDA-LISTP 8
LAMBDA-LOAD 47
LAMBDA-LOAD-LOGICAL-PATHNAME-TRANSLATIONS 43
LAMBDA-LOAD-TIME-VALUE 47
LAMBDA-LOCALLY 47
LAMBDA-LOG 3
LAMBDA-LOGAND 5
LAMBDA-LOGANDC1 5
LAMBDA-LOGANDC2 5
LAMBDA-LOGBITP 5
LAMBDA-LOGCOUNT 5
LAMBDA-LOGEQV 5
LAMBDA-LOGICAL-PATHNAME 32, 43
LAMBDA-LOGICAL-PATHNAME-TRANSLATIONS 43
LAMBDA-LOGIOR 5
LAMBDA-LOGNAND 5
LAMBDA-LOGNOR 5
LAMBDA-LOGNOT 5
LAMBDA-LOGORC1 5
LAMBDA-LOGORC2 5
LAMBDA-LOGTEST 5
LAMBDA-LOGXOR 5
LAMBDA-LONG-FLOAT 32, 35
LAMBDA-LONG-FLOAT-EPSILON 6
LAMBDA-LONG-FLOAT-NEGATIVE-EPSILON 6
LAMBDA-LONG-SITE-NAME 49
LAMBDA-LOOP 22
LAMBDA-LOOP-FINISH 25
LAMBDA-LOWER-CASE-P 7

MACHINE-INSTANCE 49
MACHINE-TYPE 49
MACHINE-VERSION 49
MACRO-FUNCTION 47
MACROEXPAND 48
MACROEXPAND-1 48
MACROLET 20
MAKE-ARRAY 11
MAKE-BROADCAST-STREAM 41

MAKE-CONCATENATED-STREAM 41
MAKE-CONDITION 29
MAKE-DISPATCH-MACRO-CHARACTER 35
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 15
MAP-INTO 15
MAPC 10
MAPCAN 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 43
METHOD 32
METHOD-COMBINATION 32, 46
METHOD-ERROR 27
METHOD-QUALIFIERS 28
MIN 4, 28
MINIMIZE 24
MINIMIZING 24
MINUS 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

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

Index

" 35
 ' 35
 (35
) 46
 * 3, 32, 33, 43, 48
 ** 43, 48
 *** 48
 #BREAK-ON-SIGNALS* 31
 #COMPILE-FILE-PATHNAME* 47
 #COMPILE-FILE-TRUENAME* 47
 #COMPILE-PRINT* 47
 #COMPILE-VERBOSE* 47
 #DEBUG-IO* 42
 #DEBUGGER-HOOK* 31
 #DEFAULT-PATHNAME-DEFAULTS* 43
 #ERROR-OUTPUT* 42
 #FEATURES* 36
 #GENSYM-COUNTER* 46
 #LOAD-PATHNAME* 47
 #LOAD-PRINT* 47
 #LOAD-TRUENAME* 47
 #LOAD-VERBOSE* 47
 #MACROEXPAND-HOOK* 48
 #MODULES* 45
 #PACKAGE* 44
 #PRINT-ARRAY* 38
 #PRINT-BASE* 38
 #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* 35
 #READ-DEFAULT-FLOAT-FORMAT* 35
 #READ-EVAL* 36
 #READ-SUPPRESS* 35
 #READTABLE* 34
 #STANDARD-INPUT* 42
 #STANDARD-OUTPUT* 42
 #TERMINAL-IO* 42
 #TRACE-OUTPUT* 48
 + 3, 28, 48
 ++ 48
 +++ 48
 . 35
 .. 35
 @ 35
 - 3, 48
 / 3, 35, 48
 // 48
 /// 48
 / = 3
 : 44
 :: 44
 :ALLOW-OTHER-KEYS 21
 : 35
 < 3
 < = 3
 = 3, 22, 24
 > 3
 > = 3
 \ 36
 # 40
 #\ 35
 #' 35
 # (35
 #* 36
 #+ 36
 #- 36
 #. 36
 #< 36
 #= 36
 #A 35
 #B 35
 #C (35
 #O 35
 #P 36
 #R 35
 #S (36
 #X 35
 ## 36
 #| # 35
 &ALLOW-OTHER-KEYS 21
 &AUX 21
 &BODY 20
 &ENVIRONMENT 21
 &KEY 20
 &OPTIONAL 20
 &REST 20
 &WHOLE 20
 ~ (~) 39
 ~* 40
 ~ / / 40
 ~ < ~ : > 40
 ~ < ~ > 39
 ~ ? 40
 ~ A 38
 ~ B 39
 ~ C 39
 ~ D 39
 ~ E 39
 ~ F 39
 ~ G 39
 ~ I 40
 ~ O 39
 ~ P 39
 ~ R 39
 ~ 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 24
 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 48
 APROPOS-LIST 48
 AREF 11
 ARITHMETIC-ERROR 22
 ARITHMETIC-ERROR-OPERANDS 31
 ARITHMETIC-ERROR-OPERATION 31
 ARRAY 32
 ARRAY-DIMENSION 11
 ARRAY-DIMENSION-LIMIT 12
 ARRAY-DIMENSIONS 11
 ARRAY-DISPLACEMENT 12
 ARRAY-ELEMENT-TYPE 33
 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
 ARRAYS-P 11
 AS 22
 ASH 6
 ASIN 3
 ASINH 4
 ASSERT 29
 ASSOC 10
 ASSOC-IF 10
 ASSOC-IF-NOT 10
 ATAN 4
 ATANH 4
 ATOM 9, 32
 BASE-CHAR 32
 BASE-STRING 32
 BEING 24
 BELOW 24
 BIGNUM 32
 BIT 12, 32
 BIT-AND 12
 BIT-AND1 12
 BIT-ANDC2 12
 BIT-ANDI 12
 BIT-ANDJ 12
 BIT-ANDK 12
 BIT-ANDL 12
 BIT-ANDM 12
 BIT-ANDN 12
 BIT-ANDO 12
 BIT-ANDP 12
 BIT-ANDQ 12
 BIT-ANDR 12
 BIT-ANDS 12
 BIT-ANDT 12
 BIT-ANDU 12
 BIT-ANDV 12
 BIT-ANDW 12
 BIT-ANDX 12
 BIT-ANDY 12
 BIT-ANDZ 12
 BIT-AND11 12
 BIT-AND12 12
 BIT-AND13 12
 BIT-AND14 12
 BIT-AND15 12
 BIT-AND16 12
 BIT-AND17 12
 BIT-AND18 12
 BIT-AND19 12
 BIT-AND20 12
 BIT-AND21 12
 BIT-AND22 12
 BIT-AND23 12
 BIT-AND24 12
 BIT-AND25 12
 BIT-AND26 12
 BIT-AND27 12
 BIT-AND28 12
 BIT-AND29 12
 BIT-AND30 12
 BIT-AND31 12
 BIT-AND32 12
 BIT-AND33 12
 BIT-AND34 12
 BIT-AND35 12
 BIT-AND36 12
 BIT-AND37 12
 BIT-AND38 12
 BIT-AND39 12
 BIT-AND40 12
 BIT-AND41 12
 BIT-AND42 12
 BIT-AND43 12
 BIT-AND44 12
 BIT-AND45 12
 BIT-AND46 12
 BIT-AND47 12
 BIT-AND48 12
 BIT-AND49 12
 BIT-AND50 12
 BIT-AND51 12
 BIT-AND52 12
 BIT-AND53 12
 BIT-AND54 12
 BIT-AND55 12
 BIT-AND56 12
 BIT-AND57 12
 BIT-AND58 12
 BIT-AND59 12
 BIT-AND60 12
 BIT-AND61 12
 BIT-AND62 12
 BIT-AND63 12
 BIT-AND64 12
 BIT-AND65 12
 BIT-AND66 12
 BIT-AND67 12
 BIT-AND68 12
 BIT-AND69 12
 BIT-AND70 12
 BIT-AND71 12
 BIT-AND72 12
 BIT-AND73 12
 BIT-AND74 12
 BIT-AND75 12
 BIT-AND76 12
 BIT-AND77 12
 BIT-AND78 12
 BIT-AND79 12
 BIT-AND80 12
 BIT-AND81 12
 BIT-AND82 12
 BIT-AND83 12
 BIT-AND84 12
 BIT-AND85 12
 BIT-AND86 12
 BIT-AND87 12
 BIT-AND88 12
 BIT-AND89 12
 BIT-AND90 12
 BIT-AND91 12
 BIT-AND92 12
 BIT-AND93 12
 BIT-AND94 12
 BIT-AND95 12
 BIT-AND96 12
 BIT-AND97 12
 BIT-AND98 12
 BIT-AND99 12
 BIT-AND100 12
 BIT-AND101 12
 BIT-AND102 12
 BIT-AND103 12
 BIT-AND104 12
 BIT-AND105 12
 BIT-AND106 12
 BIT-AND107 12
 BIT-AND108 12
 BIT-AND109 12
 BIT-AND110 12
 BIT-AND111 12
 BIT-AND112 12
 BIT-AND113 12
 BIT-AND114 12
 BIT-AND115 12
 BIT-AND116 12
 BIT-AND117 12
 BIT-AND118 12
 BIT-AND119 12
 BIT-AND120 12
 BIT-AND121 12
 BIT-AND122 12
 BIT-AND123 12
 BIT-AND124 12
 BIT-AND125 12
 BIT-AND126 12
 BIT-AND127 12
 BIT-AND128 12
 BIT-AND129 12
 BIT-AND130 12
 BIT-AND131 12
 BIT-AND132 12
 BIT-AND133 12
 BIT-AND134 12
 BIT-AND135 12
 BIT-AND136 12
 BIT-AND137 12
 BIT-AND138 12
 BIT-AND139 12
 BIT-AND140 12
 BIT-AND141 12
 BIT-AND142 12
 BIT-AND143 12
 BIT-AND144 12
 BIT-AND145 12
 BIT-AND146 12
 BIT-AND147 12
 BIT-AND148 12
 BIT-AND149 12
 BIT-AND150 12
 BIT-AND151 12
 BIT-AND152 12
 BIT-AND153 12
 BIT-AND154 12
 BIT-AND155 12
 BIT-AND156 12
 BIT-AND157 12
 BIT-AND158 12
 BIT-AND159 12
 BIT-AND160 12
 BIT-AND161 12
 BIT-AND162 12
 BIT-AND163 12
 BIT-AND164 12
 BIT-AND165 12
 BIT-AND166 12
 BIT-AND167 12
 BIT-AND168 12
 BIT-AND169 12
 BIT-AND170 12
 BIT-AND171 12
 BIT-AND172 12
 BIT-AND173 12
 BIT-AND174 12
 BIT-AND175 12
 BIT-AND176 12
 BIT-AND177 12
 BIT-AND178 12
 BIT-AND179 12
 BIT-AND180 12
 BIT-AND181 12
 BIT-AND182 12
 BIT-AND183 12
 BIT-AND184 12
 BIT-AND185 12
 BIT-AND186 12
 BIT-AND187 12
 BIT-AND188 12
 BIT-AND189 12
 BIT-AND190 12
 BIT-AND191 12
 BIT-AND192 12
 BIT-AND193 12
 BIT-AND194 12
 BIT-AND195 12
 BIT-AND196 12
 BIT-AND197 12
 BIT-AND198 12
 BIT-AND199 12
 BIT-AND200 12
 BIT-AND201 12
 BIT-AND202 12
 BIT-AND203 12
 BIT-AND204 12
 BIT-AND205 12
 BIT-AND206 12
 BIT-AND207 12
 BIT-AND208 12
 BIT-AND209 12
 BIT-AND210 12
 BIT-AND211 12
 BIT-AND212 12
 BIT-AND213 12
 BIT-AND214 12
 BIT-AND215 12
 BIT-AND216 12
 BIT-AND217 12
 BIT-AND218 12
 BIT-AND219 12
 BIT-AND220 12
 BIT-AND221 12
 BIT-AND222 12
 BIT-AND223 12
 BIT-AND224 12
 BIT-AND225 12
 BIT-AND226 12
 BIT-AND227 12
 BIT-AND228 12
 BIT-AND229 12
 BIT-AND230 12
 BIT-AND231 12
 BIT-AND232 12
 BIT-AND233 12
 BIT-AND234 12
 BIT-AND235 12
 BIT-AND236 12
 BIT-AND237 12
 BIT-AND238 12
 BIT-AND239 12
 BIT-AND240 12
 BIT-AND241 12
 BIT-AND242 12
 BIT-AND243 12
 BIT-AND244 12
 BIT-AND245 12
 BIT-AND246 12
 BIT-AND247 12
 BIT-AND248 12
 BIT-AND249 12
 BIT-AND250 12
 BIT-AND251 12
 BIT-AND252 12
 BIT-AND253 12
 BIT-AND254 12
 BIT-AND255 12
 BIT-AND256 12
 BIT-AND257 12
 BIT-AND258 12
 BIT-AND259 12
 BIT-AND260 12
 BIT-AND261 12
 BIT-AND262 12
 BIT-AND263 12
 BIT-AND264 12
 BIT-AND265 12
 BIT-AND266 12
 BIT-AND267 12
 BIT-AND268 12
 BIT-AND269 12
 BIT-AND270 12
 BIT-AND271 12
 BIT-AND272 12
 BIT-AND273 12
 BIT-AND274 12
 BIT-AND275 12
 BIT-AND276 12
 BIT-AND277 12
 BIT-AND278 12
 BIT-AND279 12
 BIT-AND280 12
 BIT-AND281 12
 BIT-AND282 12
 BIT-AND283 12
 BIT-AND284 12
 BIT-AND285 12
 BIT-AND286 12
 BIT-AND287 12
 BIT-AND288 12
 BIT-AND289 12
 BIT-AND290 12
 BIT-AND291 12
 BIT-AND292 12
 BIT-AND293 12
 BIT-AND294 12
 BIT-AND295 12
 BIT-AND296 12
 BIT-AND297 12
 BIT-AND298 12
 BIT-AND299 12
 BIT-AND300 12
 BIT-AND301 12
 BIT-AND302 12
 BIT-AND303 12
 BIT-AND304 12
 BIT-AND305 12
 BIT-AND306 12
 BIT-AND307 12
 BIT-AND308 12
 BIT-AND309 12
 BIT-AND310 12
 BIT-AND311 12
 BIT-AND312 12
 BIT-AND313 12
 BIT-AND314 12
 BIT-AND315 12
 BIT-AND316 12
 BIT-AND317 12
 BIT-AND318 12
 BIT-AND319 12
 BIT-AND320 12
 BIT-AND321 12
 BIT-AND322 12
 BIT-AND323 12
 BIT-AND324 12
 BIT-AND325 12
 BIT-AND326 12
 BIT-AND327 12
 BIT-AND328 12
 BIT-AND329 12
 BIT-AND330 12
 BIT-AND331 12
 BIT-AND332 12
 BIT-AND333 12
 BIT-AND334 12
 BIT-AND335 12
 BIT-AND336 12
 BIT-AND337 12
 BIT-AND338 12
 BIT-AND339 12
 BIT-AND340 12
 BIT-AND341 12
 BIT-AND342 12
 BIT-AND343 12
 BIT-AND344 12
 BIT-AND345 12
 BIT-AND346 12
 BIT-AND347 12
 BIT-AND348 12
 BIT-AND349 12
 BIT-AND350 12
 BIT-AND351 12
 BIT-AND352 12
 BIT-AND353 12
 BIT-AND354 12
 BIT-AND355 12
 BIT-AND356 12
 BIT-AND357 12
 BIT-AND358 12
 BIT-AND359 12
 BIT-AND360 12
 BIT-AND361 12
 BIT-AND362 12
 BIT-AND363 12
 BIT-AND364 12
 BIT-AND365 12
 BIT-AND366 12
 BIT-AND367 12
 BIT-AND368 12
 BIT-AND369 12
 BIT-AND370 12
 BIT-AND371 12
 BIT-AND372 12
 BIT-AND373 12
 BIT-AND374 12
 BIT-AND375 12
 BIT-AND376 12
 BIT-AND377 12
 BIT-AND378 12
 BIT-AND379 12
 BIT-AND380 12
 BIT-AND381 12
 BIT-AND382 12
 BIT-AND383 12
 BIT-AND384 12
 BIT-AND385 12
 BIT-AND386 12
 BIT-AND387 12
 BIT-AND388 12
 BIT-AND389 12
 BIT-AND390 12
 BIT-AND391 12
 BIT-AND392 12
 BIT-AND393 12
 BIT-AND394 12
 BIT-AND395 12
 BIT-AND396 12
 BIT-AND397 12
 BIT-AND398 12
 BIT-AND399 12
 BIT-AND400 12
 BIT-AND401 12
 BIT-AND402 12
 BIT-AND403 12
 BIT-AND404 12
 BIT-AND405 12
 BIT-AND406 12
 BIT-AND407 12
 BIT-AND408 12
 BIT-AND409 12
 BIT-AND410 12
 BIT-AND411 12
 BIT-AND412 12
 BIT-AND413 12
 BIT-AND414 12
 BIT-AND415 12
 BIT-AND416 12
 BIT-AND417 12
 BIT-AND418 12
 BIT-AND419 12
 BIT-AND420 12
 BIT-AND421 12
 BIT-AND422 12
 BIT-AND423 12
 BIT-AND424 12
 BIT-AND425 12
 BIT-AND426 12
 BIT-AND427 12
 BIT-AND428 12
 BIT-AND429 12
 BIT-AND430 12
 BIT-AND431 12
 BIT-AND432 12
 BIT-AND433 12
 BIT-AND434 12
 BIT-AND435 12
 BIT-AND436 12
 BIT-AND437 12
 BIT-AND438 12
 BIT-AND439 12
 BIT-AND440 12
 BIT-AND441 12
 BIT-AND442 12
 BIT-AND443 12
 BIT-AND444 12
 BIT-AND445 12
 BIT-AND446 12
 BIT-AND447 12
 BIT-AND448 12
 BIT-AND449 12
 BIT-AND450 12
 BIT-AND451 12
 BIT-AND452 12
 BIT-AND453 12
 BIT-AND454 12
 BIT-AND455 12
 BIT-AND456 12
 BIT-AND457 12
 BIT-AND458 12
 BIT-AND459 12
 BIT-AND460 12
 BIT-AND461 12
 BIT-AND462 12
 BIT-AND463 12
 BIT-AND464 12
 BIT-AND465 12
 BIT-AND466 12
 BIT-AND467 12
 BIT-AND468 12
 BIT-AND469 12
 BIT-AND470 12
 BIT-AND471 12
 BIT-AND472 12
 BIT-AND473 12
 BIT-AND474 12
 BIT-AND475 12
 BIT-AND476 12
 BIT-AND477 12
 BIT-AND478 12
 BIT-AND479 12
 BIT-AND480 12
 BIT-AND481 12
 BIT-AND482 12
 BIT-AND483 12
 BIT-AND484 12
 BIT-AND485 12
 BIT-AND486 12
 BIT-AND487 12
 BIT-AND488 12
 BIT-AND489 12
 BIT-AND490 12
 BIT-AND491 12
 BIT-AND492 12
 BIT-AND493 12
 BIT-AND494 12
 BIT-AND495 12
 BIT-AND496 12
 BIT-AND497 12
 BIT-AND498 12
 BIT-AND499 12
 BIT-AND500 12
 BIT-AND501 12
 BIT-AND502 12
 BIT-AND503 12
 BIT-AND504 12
 BIT-AND505 12
 BIT-AND506 12
 BIT-AND507 12
 BIT-AND508 12
 BIT-AND509 12
 BIT-AND510 12
 BIT-AND511 12
 BIT-AND512 12
 BIT-AND513 12
 BIT-AND514 12
 BIT-AND515 12
 BIT-AND516 12
 BIT-AND517 12
 BIT-AND518 12
 BIT-AND519 12
 BIT-AND520 12
 BIT-AND521 12
 BIT-AND522 12
 BIT-AND523 12
 BIT-AND524 12
 BIT-AND525 12
 BIT-AND526 12
 BIT-AND527 12
 BIT-AND528 12
 BIT-AND529 12
 BIT-AND530 12
 BIT-AND531 12
 BIT-AND532 12
 BIT-AND533 12
 BIT-AND534 12
 BIT-AND535 12
 BIT-AND536 12
 BIT-AND537 12
 BIT-AND538 12
 BIT-AND539 12
 BIT-AND540 12
 BIT-AND541 12
 BIT-AND542 12
 BIT-AND543 12
 BIT-AND544 12
 BIT-AND545 12
 BIT-AND546 12
 BIT-AND547 12
 BIT-AND548 12
 BIT-AND549 12
 BIT-AND550 12
 BIT-AND551 12
 BIT-AND552 12
 BIT-AND553 12
 BIT-AND554 12
 BIT-AND555 12
 BIT-AND556 12
 BIT-AND557 12
 BIT-AND558 12
 BIT-AND559 12
 BIT-AND560 12
 BIT-AND561 12
 BIT-AND562 12
 BIT-AND563 12
 BIT-AND564 12
 BIT-AND565 12
 BIT-AND566 12
 BIT-AND567 12
 BIT-AND568 12
 BIT-AND569 12
 BIT-AND570 12
 BIT-AND571 12
 BIT-AND572 12
 BIT-AND573 12
 BIT-AND574 12
 BIT-AND575 12
 BIT-AND576 12
 BIT-AND577 12
 BIT-AND578 12
 BIT-AND579 12
 BIT-AND580 12
 BIT-AND581 12
 BIT-AND582 12
 BIT-AND583 12
 BIT-AND584 12
 BIT-AND585 12
 BIT-AND586 12
 BIT-AND587 12
 BIT-AND588 12
 BIT-AND589 12
 BIT-AND590 12
 BIT-AND591 12
 BIT-AND592 12
 BIT-AND593 12
 BIT-AND594 12
 BIT-AND595 12
 BIT-AND596 12
 BIT-AND597 12
 BIT-AND598 12
 BIT-AND599 12
 BIT-AND600 12
 BIT-AND601 12
 BIT-AND602 12
 BIT-AND603 12
 BIT-AND604 12
 BIT-AND605 12
 BIT-AND606 12
 BIT-AND607 12
 BIT-AND608 12
 BIT-AND609 12
 BIT-AND610 12
 BIT-AND611 12
 BIT-AND612 12
 BIT-AND613 12
 BIT-AND614 12
 BIT-AND615 12
 BIT-AND616 12
 BIT-AND617 12
 BIT-AND618 12
 BIT-AND619 12
 BIT-AND620 12
 BIT-AND621 12
 BIT-AND622 12
 BIT-AND623 12
 BIT-AND624 12
 BIT-AND625 12
 BIT-AND626 12
 BIT-AND627 12
 BIT-AND628 12
 BIT-AND629 12
 BIT-AND630 12
 BIT-AND631 12
 BIT-AND632 12
 BIT-AND633 12
 BIT-AND634 12
 BIT-AND635 12
 BIT-AND636 12
 BIT-AND637 12
 BIT-AND638 12
 BIT-AND639 12
 BIT-AND640 12
 BIT-AND641 12
 BIT-AND642 12
 BIT-AND643 12
 BIT-AND644 12
 BIT-AND645 12
 BIT-AND646 12
 BIT-AND647 12
 BIT-AND648 12
 BIT-AND649 12
 BIT-AND650 12
 BIT-AND651 12
 BIT-AND652 12
 BIT-AND653 12
 BIT-AND654 12
 BIT-AND655 12
 BIT-AND656 12
 BIT-AND657 12
 BIT-AND658 12
 BIT-AND659 12
 BIT-AND660 12
 BIT-AND661 12
 BIT-AND662 12
 BIT-AND663 12
 BIT-AND664 12
 BIT-AND665 12
 BIT-AND666 12
 BIT-AND667 12
 BIT-AND668 12
 BIT-AND669 12
 BIT-AND670 12
 BIT-AND671 12
 BIT-AND672 12
 BIT-AND673 12
 BIT-AND674 12
 BIT-AND675 12
 BIT-AND676 12
 BIT-AND677 12
 BIT-AND678 12
 BIT-AND679 12
 BIT-AND680 12
 BIT-AND681 12
 BIT-AND682 12
 BIT-AND683 12
 BIT-AND684 12
 BIT-AND685 12
 BIT-AND686 12
 BIT-AND687 12
 BIT-AND688 12
 BIT-AND689 12
 BIT-AND690 12
 BIT-AND691 12
 BIT-AND692 12
 BIT-AND693 12
 BIT-AND694 12
 BIT-AND695 12
 BIT-AND696 12
 BIT-AND697 12
 BIT-AND698 12
 BIT-AND699 12
 BIT-AND700 12
 BIT-AND701 12
 BIT-AND702 12
 BIT-AND703 12
 BIT-AND704 12
 BIT-AND705 12
 BIT-AND706 12
 BIT-AND707 12
 BIT-AND708 12
 BIT-AND709 12
 BIT-AND710 12
 BIT-AND711 12
 BIT-AND712 12
 BIT-AND713 12
 BIT-AND714 12
 BIT-AND715 12
 BIT-AND716 12
 BIT-AND717 12
 BIT-AND718 12
 BIT-AND719 12
 BIT-AND720 12
 BIT-AND721 12
 BIT-AND722 12
 BIT-AND723 12
 BIT-AND724 12
 BIT-AND725 12
 BIT-AND726 12
 BIT-AND727 12
 BIT-AND728 12
 BIT-AND729 12
 BIT-AND730 12
 BIT-AND731 12
 BIT-AND732 12
 BIT-AND733 12
 BIT-AND734 12
 BIT-AND735 12
 BIT-AND736 12
 BIT-AND737 12
 BIT-AND738 12
 BIT-AND739 12
 BIT-AND740 12
 BIT-AND741 12
 BIT-AND742 12
 BIT-AND743 12
 BIT-AND744 12
 BIT-AND745 12
 BIT-AND746 12
 BIT-AND747 12
 BIT-AND748 12
 BIT-AND749 12
 BIT-AND750 12
 BIT-AND751 12
 BIT-AND752 12
 BIT-AND753 12
 BIT-AND754 12
 BIT-AND755 12
 BIT-AND756 12
 BIT-AND757 12
 BIT-AND758 12
 BIT-AND759 12
 BIT-AND760 12
 BIT-AND761 12
 BIT-AND762 12
 BIT-AND763 12
 BIT-AND764 12
 BIT-AND765 12
 BIT-AND766 12
 BIT-AND767 12
 BIT-AND768 12
 BIT-AND769 12
 BIT-AND770 12
 BIT-AND771 12
 BIT-AND772 12
 BIT-AND773 12
 BIT-AND774 12
 BIT-AND775 12
 BIT-AND776 12
 BIT-AND777 12
 BIT-AND778 12
 BIT-AND779 12
 BIT-AND780 12
 BIT-AND781 12
 BIT-AND782 1

3 Strings

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

(*fstringp* *foo*)
(*fstring-p* *foo*) ▷ T if *foo* is of indicated type.

(*fstring=* *foo bar*)
(*fstring-equal* *foo bar*)

$$\left. \begin{array}{l} \text{:start1 } start\text{-}foo_{\boxed{0}} \\ \text{:start2 } start\text{-}bar_{\boxed{0}} \\ \text{:end1 } end\text{-}foo_{\boxed{NIL}} \\ \text{:end2 } end\text{-}bar_{\boxed{NIL}} \end{array} \right\}$$

▷ Return T if subsequences of *foo* and *bar* are equal. Obey/ignore, respectively, case.

(*fstring{/= |> |>= |< |<= |<= |>= |>= |<= |<=}* *foo bar*)

$$\left. \begin{array}{l} \text{:start1 } start\text{-}foo_{\boxed{0}} \\ \text{:start2 } start\text{-}bar_{\boxed{0}} \\ \text{:end1 } end\text{-}foo_{\boxed{NIL}} \\ \text{:end2 } end\text{-}bar_{\boxed{NIL}} \end{array} \right\}$$

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

(*fmake-string* *size* *initial-element* *char* *element-type* *type* *character*)

▷ Return string of length *size*.

(*fstring* *x*)
(*fstring-capitalize* *x*)
(*fstring-upcase* *x*)
(*fstring-downcase* *x*)

$$\left. \begin{array}{l} \text{:start } start_{\boxed{0}} \\ \text{:end } end_{\boxed{NIL}} \end{array} \right\}$$

▷ Convert *x* (**symbol**, **string**, or **character**) into a string, a string with capitalized words, an all-uppercase string, or an all-lowercase string, respectively.

(*fstring-capitalize* *string*)
(*fstring-upcase* *string*)
(*fstring-downcase* *string*)

$$\left. \begin{array}{l} \text{:start } start_{\boxed{0}} \\ \text{:end } end_{\boxed{NIL}} \end{array} \right\}$$

▷ Convert *string* into a string with capitalized words, an all-uppercase string, or an all-lowercase string, respectively.

(*fstring-trim* *char-bag string*)
(*fstring-left-trim* *char-bag string*)
(*fstring-right-trim* *char-bag string*)

▷ Return string with all characters in sequence *char-bag* removed from both ends, from the beginning, or from the end, respectively.

(*fchar* *string* *i*)
(*fchar* *string* *i*)

▷ Return zero-indexed *i*th character of string ignoring/obeying, respectively, fill pointer. **setfable**.

(*fparse-integer* *string* *start* *end* *radix* *int* *junk-allowed* *bool*)

$$\left. \begin{array}{l} \text{:start } start_{\boxed{0}} \\ \text{:end } end_{\boxed{NIL}} \\ \text{:radix } int_{\boxed{10}} \\ \text{:junk-allowed } bool_{\boxed{NIL}} \end{array} \right\}$$

▷ Return integer parsed from *string* and index of parse end.

4 Conses

4.1 Predicates

(*fconsp* *foo*)
(*flistp* *foo*) ▷ Return T if *foo* is of indicated type.

(*fendp* *list*)
(*fnull* *foo*) ▷ Return T if *list/foo* is NIL.

15.4 Declarations

(*fproclaim* *decl*)
(*mdeclaim* *decl**)

▷ Globally make declaration(s) *decl*. *decl* can be: **declaration**, **type**, **ftype**, **inline**, **notinline**, **optimize**, or **special**. See below.

(*declare* *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 *foos* names of declarations.

(**dynamic-extent** *variable** (**function** *function*)*)

▷ Declare lifetime of *variables* and/or *functions* to end when control leaves enclosing block.

(**[type]** *type variable**)

(**ftype** *type function**)

▷ Declare *variables* or *functions* to be of *type*.

(**ignorable** *var*)
(**ignore** *var* *function**)

▷ Suppress warnings about used/unused bindings.

(**inline** *function**)

(**notinline** *function**)

▷ Tell compiler to integrate/not to integrate, respectively, called *functions* into the calling routine.

(**optimize** *compilation-speed* (*compilation-speed* *n*)
debug (*debug* *n*)
safety (*safety* *n*)
space (*space* *n*)
speed (*speed* *n*)

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

(**special** *var**) ▷ Declare *vars* to be dynamic.

16 External Environment

(*fget-internal-real-time*)

(*fget-internal-run-time*)

▷ Current time, or computing time, respectively, in clock ticks.

internal-time-units-per-second

▷ Number of clock ticks per second.

(*fencode-universal-time* *sec* *min* *hour* *date* *month* *year* [*zone* *current*])

(*fget-universal-time*)

▷ Seconds from 1900-01-01, 00:00, ignoring leap seconds.

(*fdecode-universal-time* *universal-time* [*time-zone* *current*])

(*fget-decoded-time*)

▷ Return second, minute, hour, date, month, year, day, daylight-p, and zone.

(*fshort-site-name*)

(*flong-site-name*)

▷ String representing physical location of computer.

(*f* *lisp-implementation*)
(*f* *software*)
(*f* *machine*)

$$\left. \begin{array}{l} \text{:type} \\ \text{:version} \end{array} \right\}$$

▷ Name or version of implementation, operating system, or hardware, respectively.

(*f* *machine-instance*) ▷ Computer name.

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* [*package* NTI])
 ▷ Print interned symbols containing *string*.

(f apropos-list *string* [*package* NTI])
 ▷ 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 [*file-or-function* NTI]) ▷ Invoke editor if possible.

($\left\{ \begin{array}{l} f\text{macroexpand-1} \\ f\text{macroexpand} \end{array} \right\}$ *form* [*environment* NTI])
 ▷ Return macro expansion, once or entirely, respectively, of *form* and T if *form* was a macro form. Return form and NIL otherwise.

$v*\text{macroexpand-hook*}$
 ▷ Function of arguments expansion function, macro form, and environment called by $f\text{macroexpand-1}$ to generate macro expansions.

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

(m untrace $\left\{ \begin{array}{l} \text{function} \\ \text{(setf function)} \end{array} \right\}^*$)
 ▷ Stop *functions*, or each currently traced function, from being traced.

$v*\text{trace-output*}$
 ▷ Output stream $m\text{trace}$ and $m\text{time}$ 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, $f\text{format}$, for *control* and *args*.

(m time *form*)
 ▷ Evaluate *forms* and print timing information to $v*\text{trace-output*}$. Return values of form.

(f inspect *foo*) ▷ Interactively give information about *foo*.

(f describe *foo* [*stream* NTI])
 ▷ Send information about *foo* to *stream*.

(g describe-object *foo* [*stream*])
 ▷ Send information about *foo* to *stream*. Called by $f\text{describe}$.

(f disassemble *function*)
 ▷ Send disassembled representation of *function* to $v*\text{standard-output*}$. Return NIL.

(f room [NIL | default | T] [default])
 ▷ Print information about internal storage management to $*\text{standard-output*}$.

(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* $\left\{ \begin{array}{l} \text{:test function} \\ \text{:test-not function} \\ \text{:key function} \end{array} \right\}$)
 ▷ Return tail of list starting with its first element matching *foo*. Return NIL if there is no such element.

($\left\{ \begin{array}{l} f\text{member-if} \\ f\text{member-if-not} \end{array} \right\}$ *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* $\left\{ \begin{array}{l} \text{:test function} \\ \text{:test-not function} \\ \text{:key function} \end{array} \right\}$)
 ▷ 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 foos.

(f list* *foo+*)
 ▷ Return list of foos with last *foo* becoming cdr of last cons. Return *foo* if only one *foo* given.

(f make-list *num* [*:initial-element* *foo* NTI])
 ▷ 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. **setfable**.

(f cdr *list*) ▷ Cdr of list or NIL if *list* is NIL. **setfable**.
 (f rest *list*)

(f nthcdr *n list*) ▷ Return tail of list after calling $f\text{cdr}$ *n* times.

($\left\{ \begin{array}{l} f\text{first} \\ f\text{second} \\ f\text{third} \\ f\text{fourth} \\ f\text{fifth} \\ f\text{sixth} \\ \dots \\ f\text{ninth} \\ f\text{tenth} \end{array} \right\}$ *list*)
 ▷ Return nth element of list if any, or NIL otherwise. **setfable**.

(f nth *n list*) ▷ Zero-indexed nth element of *list*. **setfable**.

(f cXr *list*)
 ▷ With *X* being one to four **as** and **ds** representing $f\text{cars}$ and $f\text{cdrs}$, e.g. ($f\text{cadr}$ *bar*) is equivalent to ($f\text{car}$ ($f\text{cdr}$ *bar*)). **setfable**.

(f last *list* [*num* NTI]) ▷ Return list of last num conses of *list*.

($\left\{ \begin{array}{l} f\text{butlast} \\ f\text{nbutlast} \end{array} \right\}$ *list* [*num* NTI]) ▷ list excluding last *num* conses.

($\left\{ \begin{array}{l} f\text{rplaca} \\ f\text{rplacd} \end{array} \right\}$ $\widetilde{\text{cons}}$ *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* $\left\{ \begin{array}{l} \text{:test function} \\ \text{:test-not function} \\ \text{:key function} \end{array} \right\}$)
 ▷ Return list if *foo* is already member of *list*. If not, return ($f\text{cons}$ *foo list*).

(m pop $\widetilde{\text{place}}$) ▷ Set *place* to ($f\text{cdr}$ *place*), return ($f\text{car}$ *place*).

(*m*push *foo* *place*) ▷ Set *place* to (*f*cons *foo* *place*).

(*m*pushnew *foo* *place* {[:test *function*_{#'eq}]
[:test-not *function*]
[:key *function*])

▷ Set *place* to (*f*adjoin *foo* *place*).

(*f*append [*proper-list** *foo*_{NIL}])

(*f*nconc [*non-circular-list** *foo*_{NIL}])

▷ Return concatenated list or, with only one argument, *foo*. *foo* can be of any type.

(*f*revappend *list* *foo*)

(*f*nreconc *list* *foo*)

▷ Return concatenated list after reversing order in *list*.

{(*f*mapcar)
(*f*maplist)} *function* *list*⁺)

▷ Return list of return values of *function* successively invoked with corresponding arguments, either cars or cdrs, respectively, from each *list*.

{(*f*mapcan)
(*f*mapcon)} *function* *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.

{(*f*mapc)
(*f*mapl)} *function* *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* [*alist*_{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.

{(*f*assoc)
(*f*rassoc)} *foo* *alist* {[:test *test*_{#'eq}]
[:test-not *test*]
[:key *function*]}

{(*f*assoc-if[-not])
(*f*rassoc-if[-not])} *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* {[:test *test*_{#'eq}]
[:test-not *test*])

▷ Return T if trees *foo* and *bar* have same shape and leaves satisfying *test*.

{(*f*subst *new* *old* *tree*)
(*f*nsubst *new* *old* *tree*)} {[:test *function*_{#'eq}]
[:test-not *function*]
[:key *function*]}

▷ Make copy of *tree* with each subtree or leaf matching *old* replaced by *new*.

{(*f*subst-if[-not] *new* *test* *tree*)
(*f*nsubst-if[-not] *new* *test* *tree*)} [:key *function*])

▷ Make copy of *tree* with each subtree or leaf satisfying *test* replaced by *new*.

(*f*compile-file *file* {[:output-file *out-path*
:verbose *bool*_{v*compile-verbose*}
:print *bool*_{v*compile-print*}
:external-format *file-format*_{default}])

▷ Write compiled contents of *file* to *out-path*. Return true output path or NIL, T in case of warnings or errors, T in case of warnings or errors excluding style-warnings.

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

▷ Pathname *f*compile-file writes to if invoked with the same arguments.

(*f*load *path* {[:verbose *bool*_{v*load-verbose*}
:print *bool*_{v*load-print*}
:if-does-not-exist *bool*_T
:external-format *file-format*_{default}])

▷ Load source file or compiled file into Lisp environment. Return T if successful.

v*compile-file {pathname*_{NIL}
v*load {truename*_{NIL}

▷ Input file used by *f*compile-file/by *f*load.

v*compile {print*
v*load {verbose*

▷ Defaults used by *f*compile-file/by *f*load.

(*s*eval-when ({[:compile-toplevel|compile]
{[:load-toplevel|load]}
{[:execute|eval]})) *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 *decl*^{*})* *form*^{P*})

▷ Evaluate *forms* in a lexical environment with declarations *decl* in effect. Return values of *forms*.

(*m*with-compilation-unit ([:override *bool*_{NIL}]) *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* [*read-only*_{NIL}])

▷ Evaluate *form* at compile time and treat its value as literal at run time.

(*s*quote *foo*) ▷ Return unevaluated *foo*.

(*g*make-load-form *foo* [*environment*])

▷ Its methods are to return a creation form which on evaluation at load time returns an object equivalent to *foo*, and an optional initialization form which on evaluation performs some initialization of the object.

(*f*make-load-form-saving-slots *foo* {[:slot-names *slots*_{all local slots}]
[:environment *environment*])

▷ Return a creation form and an 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 {*name*
(setf *name*)} [*environment*])

▷ Return specified macro function, or compiler macro function, respectively, if any. Return NIL otherwise. setfable.

(*f*eval *arg*)

▷ Return values of value of *arg* evaluated in global environment.

(*fgensym* [*s*])
 ▷ Return fresh, uninterned symbol `#:sn` with *n* from `*gensym-counter*`. Increment `*gensym-counter*`.

(*fgentemp* [*prefix*] [*package*])
 ▷ Intern fresh symbol in package. Deprecated.

(*fcopy-symbol* *symbol* [*props*])
 ▷ 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*. **setfable**.

(*g*documentation
 (setf *g*documentation) *new-doc*) *foo* {
 'variable'|'function
 'compiler-macro
 'method-combination
 'structure'|'type'|setf T
 }

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

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

*nil*_c()
 ▷ Falsity; the empty list; the empty type, subtype of every type; `*standard-input*`; `*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*) ▷ T if *foo* is a special operator.

(*f*compiled-function-p *foo*) ▷ T if *foo* is of type **compiled-function**.

15.2 Compilation

(*f*compile {NIL *definition*
 {*name*
 (setf *name*) } [*definition*])
 }
 ▷ Return compiled function or replace *name*'s function definition with the compiled function. Return T in case of **warnings** or **errors**, and T in case of **warnings** or **errors** excluding **style-warnings**.

{*f*sublis *association-list tree*
*f*nsublis *association-list tree* } {
 :test *function* [≠=eq]
 :test-not *function*
 :key *function*
 }

▷ Make copy of *tree* with each subtree or leaf matching a key in *association-list* replaced by that key's value.

(*f*copy-tree *tree*) ▷ Copy of *tree* with same shape and leaves.

4.5 Sets

{*f*intersection
*f*set-difference
*f*union
*f*set-exclusive-or
*f*nintersection
*f*nset-difference
*f*nunion
*f*nset-exclusive-or } *a b* } {
 :test *function* [≠=eq]
 :test-not *function*
 :key *function*
 }

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

5 Arrays

5.1 Predicates

(*f*arrayp *foo*)
 (*f*vectorp *foo*)
 (*f*simple-vector-p *foo*) ▷ 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*)
 ▷ T if *array* is adjustable/has a fill pointer, respectively.

(*f*array-in-bounds-p *array* [*subscripts*])
 ▷ Return T if *subscripts* are in *array*'s bounds.

5.2 Array Functions

{*f*make-array *dimension-sizes* [:adjustable *bool*] _{NTD}
*f*adjust-array *array* *dimension-sizes* }
 {
 :element-type *type*_{NTD}
 :fill-pointer {*num*} *bool* _{NTD}
 :initial-element *obj*
 :initial-contents *tree-or-array*
 :displaced-to *array* _{NTD} [:displaced-index-offset *i*] _{NTD}
 }

▷ Return fresh, or readjust, respectively, vector or array.

(*f*aref *array* [*subscripts*])
 ▷ Return array element pointed to by *subscripts*. **setfable**.

(*f*row-major-aref *array* *i*)
 ▷ Return *i*th element of *array* in row-major order. **setfable**.

(*f*array-row-major-index *array* [*subscripts*])
 ▷ Index in row-major order of the element denoted by *subscripts*.

(*f*array-dimensions *array*)
 ▷ List containing the lengths of *array*'s dimensions.

(*f*array-dimension *array* *i*) ▷ Length of *i*th dimension of *array*.

(*f*array-total-size *array*) ▷ Number of elements in *array*.

(*f*array-rank *array*) ▷ Number of dimensions of *array*.

(*f*array-displacement *array*) ▷ Target array and offset.

(*f*bit *bit-array* [*subscripts*])
 (*f*sbit *simple-bit-array* [*subscripts*])
 ▷ Return element of *bit-array* or of *simple-bit-array*. **setfable**.

(*f*bit-not *bit-array* [*result-bit-array*])
 ▷ 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.

(*f*bit-eqv
*f*bit-and
*f*bit-andc1
*f*bit-andc2
*f*bit-nand
*f*bit-ior
*f*bit-orc1
*f*bit-orc2
*f*bit-xor
*f*bit-nor

bit-array-a bit-array-b [*result-bit-array*])
 ▷ 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 *foos*.

(*f*svref *vector* *i*) ▷ Element *i* of simple *vector*. **setfable**.

(*f*vector-push *foo* *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* *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 *vector*)
 ▷ Return element of vector its fillpointer points to after decrementation.

(*f*fill-pointer *vector*) ▷ Fill pointer of *vector*. **setfable**.

6 Sequences

6.1 Sequence Predicates

(*f*every
*f*notevery) *test sequence*⁺
 ▷ Return NIL or T, respectively, as soon as *test* on any set of corresponding elements of *sequences* returns NIL.

(*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* *v.*package**])
 ▷ Intern or find, respectively, symbol *foo* in *package*. Second return value is one of **:internal**, **:external**, or **:inherited** (or NIL if *f*intern has created a fresh symbol).

(*f*unintern *symbol* [*package* *v.*package**])
 ▷ Remove *symbol* from *package*, return T on success.

(*f*import
*f*shadowing-import) *symbols* [*package* *v.*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* *v.*package**])
 ▷ Make *symbols* of *package* shadow any otherwise accessible, equally named symbols from other packages. Return T.

(*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* *v.*package**])
 ▷ Make *symbols* external to *package*. Return T.

(*f*unexport *symbols* [*package* *v.*package**])
 ▷ Revert *symbols* to internal status. Return T.

(*m*do-symbols
*m*do-external-symbols) (*var* [*package* *v.*package**] [*result*])
*m*do-all-symbols (*var* [*result*])
 (declare *decl**)* (*tag* *form*)*
 ▷ 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 *decl**)* *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*])
 ▷ 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*directory *path*) ▷ List of pathnames matching *path*.

(*f*ensure-directories-exist *path* [:verbose *bool*])
▷ Create parts of path if necessary. Second return value is 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*)

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**)NIL)
 (:use (*used-package**)*)
})

▷ Create package *foo*.

(*f*rename-package *package* *new-name* [*new-nicknames*NIL])
▷ Rename *package*. Return renamed package.

(*m*in-package *foo*) ▷ Make package *foo* current.

{
 (*f*use-package
 other-packages [*package**packages*])
 (*f*unuse-package
 other-packages [*package**packages*])
}
▷ 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 *package*)
▷ Delete *package*. Return T if successful.

*v**package*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*some
 (*f*notany)
 }
 test sequence+)
▷ Return value of test or NIL, respectively, as soon as *test* on any set of corresponding elements of *sequences* returns non-NIL.

(*f*mismatch *sequence-a* *sequence-b* {
 (:from-end *bool*NIL)
 (:test *function*#'eq)
 (:test-not *function*)
 (:start1 *start-a*0)
 (:start2 *start-b*0)
 (:end1 *end-a*NIL)
 (:end2 *end-b*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* *sequence-a* *sequence-b* *test* [:key *function*NIL])
▷ Return interleaved sequence of *type*. Merged sequence will be sorted if both *sequence-a* and *sequence-b* are sorted.

(*f*fill *sequence* *foo* {
 (:start *start*0)
 (:end *end*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*NIL)
 (:test *function*#'eq)
 (:test-not *function*)
 (:start *start*0)
 (:end *end*NIL)
 (:key *function*)
})
▷ Return number of elements in *sequence* which match *foo*.

{
 (*f*count-if
 (*f*count-if-not)
 }
 test sequence {
 (:from-end *bool*NIL)
 (:start *start*0)
 (:end *end*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*. setfable.

(*f*subseq *sequence* *start* [*end*NIL])
▷ Return subsequence of sequence between *start* and *end*. setfable.

{
 (*f*sort
 (*f*stable-sort)
 }
 sequence *test* [:key *function*])
}
▷ Return sequence sorted. Order of elements considered equal is not guaranteed/retained, respectively.

(*f*reverse *sequence*)
(*f*nreverse *sequence*) ▷ Return sequence in reverse order.

(*f*close *stream* [:abort *bool*_{NIL}])
 ▷ Close *stream*. Return T if *stream* had been open. If **:abort** is T, delete associated file.

(*m*with-open-file (*stream path open-arg**) (declare *decl**)* *form*^P*)
 ▷ Use *f*open with *open-args* to temporarily create *stream* to *path*; return values of forms.

(*m*with-open-stream (*foo stream*) (declare *decl**)* *form*^P*)
 ▷ Evaluate *forms* with *foo* locally bound to *stream*. Return values of forms.

(*m*with-input-from-string (*foo string* {*:index index*
*:start start*₀
*:end end*_{NIL}}) (declare *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*.

(*m*with-output-to-string (*foo* [*string*_{NIL} [:element-type *type*_{character}]])
 (declare *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 {*:host* {*host*_{NIL}[:unspecific]}
:device {*device*_{NIL}[:unspecific]}
:directory {*directory* {*:wild*_{NIL}[:unspecific]}
 {*:absolute* {*directory*
:wild
:wild-inferiors}*
:relative {*:up*
:back}
 }
:name {*file-name*[:wild_{NIL}[:unspecific]}
:type {*file-type*[:wild_{NIL}[:unspecific]}
:version {*:newest* *version*[:wild_{NIL}[:unspecific]}
:defaults *path*_{{host from *v**default-pathname-defaults*}
:case {*:local*[:common]}_{local} }

▷ 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.

{*f*pathname-host
*f*pathname-device
*f*pathname-directory
*f*pathname-name
*f*pathname-type
*f*pathname-version *path-or-stream* [:case {*:local*
:common}_{local}])

▷ Return pathname component.

(*f*replace *sequence-a sequence-b* {*:start1 start-a*₀
*:start2 start-b*₀
*:end1 end-a*_{NIL}
*:end2 end-b*_{NIL}})

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

(*f*map *type function sequence*⁺)

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

(*f*map-into *result-sequence function sequence*⁺)

▷ Store into result-sequence successively values of *function* applied to corresponding elements of the sequences.

(*f*reduce *function sequence* {*:initial-value* *foo*_{NIL}
:from-end *bool*_{NIL}
:start *start*₀
:end *end*_{NIL}
:key function }

▷ 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*make-hash-table {*:test* {*f*eq|*f*eqi|*f*equal|*f*equalp}|*#*='*eq*})
:size int
:rehash-size num
:rehash-threshold num }

▷ Make a hash table.

(*f*gethash *key hash-table* [*default*_{NIL}])

▷ Return object with *key* if any or default otherwise; and T if found, NIL otherwise. **setfable**.

(*f*hash-table-count *hash-table*)

▷ Number of entries in hash-table.

(*f*remhash *key hash-table*)

▷ Remove from hash-table entry with *key* and return T if it existed. Return NIL otherwise.

(*f*clrhash *hash-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 *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.

~ [:] [C] < { [prefix_{⌈n}] ~; } [per-line-prefix ~C;] } body [-; suffix_{⌈n}] ~; [C] >
 ▷ **Logical Block.** Act like **pprint-logical-block** using *body* as *f*format control string on the elements of the list argument or, with **C**, on the remaining arguments, which are extracted by **pprint-pop**. With **:**, *prefix* and *suffix* default to (and). When closed by ~C>, spaces in *body* are replaced with conditional newlines.

{~ [n_⌈] i |~ [n_⌈] :i}
 ▷ **Indent.** Set indentation to *n* relative to leftmost/to current position.

~ [c_⌈] [,i_⌈] [:] [C] T
 ▷ **Tabulate.** Move cursor forward to column number *c + ki*, *k* ≥ 0 being as small as possible. With **:**, calculate column numbers relative to the immediately enclosing section. With **C**, move to column number *c₀ + c + ki* where *c₀* is the current position.

{~ [m_⌈] * |~ [m_⌈] :* |~ [n_⌈] C*}
 ▷ **Go-To.** Jump *m* arguments forward, or backward, or to argument *n*.

~ [limit] [:] [C] { text ~}
 ▷ **Iteration.** Use *text* repeatedly, up to *limit*, as control string for the elements of the list argument or (with **C**) for the remaining arguments. With **:** or **C:**, 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* ≤ *y* ≤ *z*, respectively.

~ [i] [:] [C] [[text ~;]* text] [~::; default] ~
 ▷ **Conditional Expression.** Use the zero-indexed argument (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 **C**, do nothing for an argument value of NIL. Use the only *text* and leave the argument to be read again if it is T.

{~?|~C?}
 ▷ **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}*] [:] [C] / [package [:] :class] function /
 ▷ **Call Function.** Call all-uppercase *package::function* with the arguments stream, format-argument, colon-p, at-sign-p and *prefixes* for printing format-argument.

~ [:] [C] W
 ▷ **Write.** Print argument of any type obeying every printer control variable. With **:**, pretty-print. With **C**, 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.

(*f*not *foo*) ▷ T if *foo* is NIL; NIL otherwise.

(*f*boundp *symbol*) ▷ T if *symbol* is a special variable.

(*f*constantp *foo* [environment_{⌈n}])
 ▷ T if *foo* is a constant form.

(*f*functionp *foo*) ▷ T if *foo* is of type **function**.

(*f*boundp {*foo* (setf *foo*)}) ▷ T if *foo* is a global function or macro.

9.2 Variables

{*m*defconstant } *foo* *form* [*doc*]
 ▷ Assign value of *form* to global constant/dynamic variable *foo*.

(*m*defvar *foo* [*form* [*doc*]])
 ▷ Unless bound already, assign value of *form* to dynamic variable *foo*.

{*m*setf } {*place* *form*}*
 ▷ Set *places* to primary values of *forms*. Return values of last *form*/NIL; work sequentially/in parallel, respectively.

{*s*setq } {*symbol* *form*}*
 ▷ Set *symbols* to primary values of *forms*. Return value of last *form*/NIL; work sequentially/in parallel, respectively.

(*f*set *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 *place*⁺ *foo*)
 ▷ Store value of *foo* in rightmost *place* shifting values of *places* left, returning first *place*.

(*m*rotatef *place**)
 ▷ Rotate values of *places* left, old first becoming new last *place*'s value. Return NIL.

(*f*makunbound *foo*) ▷ Delete special variable *foo* if any.

(*f*get *symbol* *key* [default_{⌈n}])
 (*f*getf *place* *key* [default_{⌈n}])
 ▷ First entry *key* from property list stored in *symbol*/in *place*, respectively, or *default* if there is no *key*. **setfable**.

(*f*get-properties *property-list* *keys*)
 ▷ Return *key* and *value* of first entry from *property-list* matching a key from *keys*, and tail of *property-list* starting with that key. Return NIL, NIL, and NIL if there was no matching key in *property-list*.

(*f*remprop *symbol* *key*)
 (*m*remf *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_k})
 ▷ Evaluate *forms* with locally established dynamic bindings of *symbols* to *values* or NIL. Return values of *forms*.

$\left\{ \begin{array}{l} \text{slet} \\ \text{slet*} \end{array} \right\} \left(\left\{ \begin{array}{l} \text{name} \\ \text{(name [value_{NTL}])} \end{array} \right\}^* \right) (\text{declare } \widehat{\text{decl}}^*)^* \text{form}^{\text{Pk}}$
 ▷ Evaluate *forms* with *names* lexically bound (in parallel or sequentially, respectively) to *values*. Return values of forms.

$(\text{multiple-value-bind } (\widehat{\text{var}}^*) \text{values-form } (\text{declare } \widehat{\text{decl}}^*)^* \text{body-form}^{\text{Pk}})$
 ▷ Evaluate *body-forms* with *vars* lexically bound to the return values of *values-form*. Return values of body-forms.

$(\text{destructuring-bind } \text{destruct-}\lambda \text{ bar } (\text{declare } \widehat{\text{decl}}^*)^* \text{form}^{\text{Pk}})$
 ▷ 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

$(\text{var}^* \text{ \&optional } \left\{ \begin{array}{l} \text{var} \\ \text{(var [init_{NTL}] [supplied-p])} \end{array} \right\}^* \text{ \&rest var}$
 $\left\{ \begin{array}{l} \text{\&key} \\ \text{\&aux} \end{array} \right\} \left\{ \begin{array}{l} \text{var} \\ \text{(:(key var)} \\ \text{(var [init_{NTL}])} \end{array} \right\}^* \left\{ \begin{array}{l} \text{[init_{NTL}] [supplied-p]} \\ \text{[allow-other-keys]} \end{array} \right\}$

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

$\left(\begin{array}{l} \text{mdefun} \\ \text{mlambda} \end{array} \left\{ \begin{array}{l} \text{foo (ord-}\lambda^*) \\ \text{(setf foo) (new-value ord-}\lambda^*) \end{array} \right\} \left\{ \begin{array}{l} \text{(declare } \widehat{\text{decl}}^*)^* \\ \text{doc} \end{array} \right\} \right)$
 form^{Pk}
 ▷ Define a function named *foo* or (**setf** *foo*), or an anonymous **function**, respectively, which applies *forms* to *ord-λs*. For **mdefun**, *forms* are enclosed in an implicit **block** named *foo*.

$\left\{ \begin{array}{l} \text{sfllet} \\ \text{slabels} \end{array} \right\} \left(\left\{ \begin{array}{l} \text{foo (ord-}\lambda^*) \\ \text{(setf foo) (new-value ord-}\lambda^*) \end{array} \right\} \left\{ \begin{array}{l} \text{(declare } \widehat{\text{local-decl}}^*)^* \\ \text{doc} \end{array} \right\} \right)$
 $\text{local-form}^{\text{Pk}})^* (\text{declare } \widehat{\text{decl}}^*)^* \text{form}^{\text{Pk}}$
 ▷ 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 **block** around its corresponding *local-form**. Only for **slabels**, functions *foo* are visible inside *local-forms*. Return values of forms.

$(\text{sfunction } \left\{ \begin{array}{l} \text{foo} \\ \text{(mlambda form}^*) \end{array} \right\})$
 ▷ Return lexically innermost **function** named *foo* or a lexical closure of the **mlambda** expression.

$(\text{fapply } \left\{ \begin{array}{l} \text{function} \\ \text{(setf function)} \end{array} \right\} \text{arg}^* \text{args})$
 ▷ Values of *function* called with *args* and the list elements of *args*. **setfable** if *function* is one of **faref**, **fbit**, and **fsbit**.

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

$(\text{multiple-value-call } \text{function } \text{form}^*)$
 ▷ Call *function* with all the values of each *form* as its arguments. Return values returned by function.

$(\text{values-list } \text{list})$ ▷ Return elements of list.

$(\text{values } \text{foo}^*)$
 ▷ Return as multiple values the primary values of the *foos*. **setfable**.

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

$\sim [\text{radix}] [\text{width}] [\text{'pad-char}] [\text{'comma-char}] [\text{'comma-interval}] [:] [\text{\&R}]$
 ▷ **Radix**. (With one or more prefix arguments.) Print argument as number; with **:**, group digits *comma-interval* each; with **\&**, always prepend a sign.

$\{\sim\text{R}|\sim\text{R}|\sim\text{\&R}|\sim\text{\&:R}\}$
 ▷ **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.

$\sim [\text{width}] [\text{'pad-char}] [\text{'comma-char}] [\text{'comma-interval}] [:] [\text{\&}] \{\text{D}|\text{B}|\text{O}|\text{X}\}$
 ▷ **Decimal/Binary/Octal/Hexadecimal**. Print integer argument as number. With **:**, group digits *comma-interval* each; with **\&**, always prepend a sign.

$\sim [\text{width}] [\text{dec-digits}] [\text{shift}] [\text{'overflow-char}] [\text{'pad-char}] [\text{\&}] [\text{\&F}]$
 ▷ **Fixed-Format Floating-Point**. With **\&**, always prepend a sign.

$\sim [\text{width}] [\text{dec-digits}] [\text{exp-digits}] [\text{scale-factor}] [\text{'overflow-char}] [\text{'pad-char}] [\text{'exp-char}] [\text{\&}] \{\text{E}|\text{G}\}$
 ▷ **Exponential/General Floating-Point**. Print argument as floating-point number with *dec-digits* after decimal point and *exp-digits* in the signed exponent. With **\&**, choose either **\&E** or **\&F**. With **\&**, always prepend a sign.

$\sim [\text{dec-digits}] [\text{int-digits}] [\text{width}] [\text{'pad-char}] [:] [\text{\&}] [\text{\&S}]$
 ▷ **Monetary Floating-Point**. Print argument as fixed-format floating-point number. With **:**, put sign before any padding; with **\&**, always prepend a sign.

$\{\sim\text{C}|\sim\text{C}|\sim\text{\&C}|\sim\text{\&:C}\}$
 ▷ **Character**. Print, spell out, print in **\#** syntax, or tell how to type, respectively, argument as (possibly non-printing) character.

$\{-(\text{text } \sim)|\sim:(\text{text } \sim)|\sim\text{\&}(\text{text } \sim)|\sim\text{\&:}(\text{text } \sim)\}$
 ▷ **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.

$\{\sim\text{P}|\sim\text{P}|\sim\text{\&P}|\sim\text{\&:P}\}$
 ▷ **Plural**. If argument **eq1** print nothing, otherwise print **s**; do the same for the previous argument; if argument **eq1** print **y**, otherwise print **ies**; do the same for the previous argument, respectively.

$\sim [n] \%$ ▷ **Newline**. Print *n* newlines.

$\sim [n] \&$
 ▷ **Fresh-Line**. Print *n* – 1 newlines if output stream is at the beginning of a line, or *n* newlines otherwise.

$\{\sim\text{.}|\sim\text{.}|\sim\text{\&.}|\sim\text{\&:}\}$
 ▷ **Conditional Newline**. Print a newline like **pprint-newline** with argument **:linear**, **:fill**, **:miser**, or **:mandatory**, respectively.

$\{\sim\text{;}|\sim\text{;}|\sim\text{\&;}|\sim\text{\&:}\}$
 ▷ **Ignored Newline**. Ignore newline, or whitespace following newline, or both, respectively.

$\sim [n] |$ ▷ **Page**. Print *n* page separators.

$\sim [n] \sim$ ▷ **Tilde**. Print *n* tildes.

$\sim [\text{min-col}] [\text{col-inc}] [\text{min-pad}] [\text{'pad-char}] [:] [\text{\&} < [\text{nl-text } \sim\text{[spare]} [\text{width}]]:] \{\text{text } \sim\text{;}^* \text{text } \sim\}$
 ▷ **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.

- `v*print-array*` ▷ If T, print arrays *f* readably.
- `v*print-base*`_[10] ▷ Radix for printing rationals, from 2 to 36.
- `v*print-case*`_[upcase]
 - ▷ Print symbol names all uppercase (:upcase), all lowercase (:downcase), capitalized (:capitalize).
- `v*print-circle*`_[NIL]
 - ▷ If T, avoid indefinite recursion while printing circular structure.
- `v*print-escape*`_[NIL]
 - ▷ If NIL, do not print escape characters and package prefixes.
- `v*print-gensym*`_[NIL] ▷ If T, print #: before uninterned symbols.
- `v*print-length*`_[NIL]
- `v*print-level*`_[NIL]
- `v*print-lines*`_[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*`_[NIL] ▷ If T, print rationals with a radix indicator.
- `v*print-readably*`_[NIL]
 - ▷ If T, print *f* readably or signal error **print-not-readable**.
- `v*print-right-margin*`_[NIL]
 - ▷ Right margin width in ems while pretty-printing.
- `(fset-pprint-dispatch type function [priority[table[v*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.
- `(fpprint-dispatch foo [table[v*print-pprint-dispatch*]]]`
 - ▷ Return highest priority *function* associated with type of *foo* and T if there was a matching type specifier in *table*.
- `(fcopy-pprint-dispatch [table[v*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

- `(mformatter control)`
 - ▷ Return function of *stream* and *arg** applying *f*format to *stream*, *control*, and *arg** returning NIL or any excess *args*.
- `(fformat {T|NIL|out-string|out-stream} control arg*)`
 - ▷ Output string *control* which may contain ~ directives possibly taking some *args*. Alternatively, *control* can be a function returned by *mformatter* 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*_[0] [, [*col-inc*_[1] [, [*min-pad*_[0] [, [*pad-char*<sub>[]]]]]]
 - [:] @ {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-chars* on the left rather than on the right.</sub>

- `(mnth-value n form)`
 - ▷ Zero-indexed *n*th return value of *form*.
- `(fcomplement function)`
 - ▷ Return new function with same arguments and same side effects as *function*, but with complementary truth value.
- `(fconstantly foo)`
 - ▷ Function of any number of arguments returning *foo*.
- `(fidentity foo)` ▷ Return *foo*.
- `(ffunction-lambda-expression function)`
 - ▷ If available, return lambda expression of *function*, NIL if *function* was defined in an environment without bindings, and name of *function*.
- `(fdefinition {foo (setf foo)})`
 - ▷ Definition of global function *foo*. **setfable**.
- `(fmkunbound foo)`
 - ▷ Remove global function or macro definition *foo*.
- `ccall-arguments-limit`
- `clambda-parameters-limit`
 - ▷ Upper bound of the number of function arguments or lambda list parameters, respectively; ≥ 50.
- `cmultiple-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

$$([\&whole \textit{var}] [E] \left\{ \begin{array}{l} \textit{var} \\ (\textit{macro-}\lambda^*) \end{array} \right\}^* [E])$$

$$[\&optional \left\{ \left(\left\{ \begin{array}{l} \textit{var} \\ (\textit{macro-}\lambda^*) \end{array} \right\} [init_{[NIL]} [supplied-p]] \right) \right\}] [E]$$

$$[\&\{ \begin{array}{l} \&rest \\ \&body \end{array} \} \left\{ \begin{array}{l} \textit{rest-var} \\ (\textit{macro-}\lambda^*) \end{array} \right\}] [E]$$

$$[\&key \left\{ \left\{ \begin{array}{l} \textit{var} \\ (:key \left\{ \begin{array}{l} \textit{var} \\ (\textit{macro-}\lambda^*) \end{array} \right\}) \end{array} \right\} [init_{[NIL]} [supplied-p]] \right\} \right\}] [E]$$

$$[\&allow-other-keys] [\&aux \left\{ \begin{array}{l} \textit{var} \\ (\textit{var} [init_{[NIL]}]) \end{array} \right\}] [E]$$

or

$$([\&whole \textit{var}] [E] \left\{ \begin{array}{l} \textit{var} \\ (\textit{macro-}\lambda^*) \end{array} \right\}^* [E])$$

$$[\&optional \left\{ \left(\left\{ \begin{array}{l} \textit{var} \\ (\textit{macro-}\lambda^*) \end{array} \right\} [init_{[NIL]} [supplied-p]] \right) \right\}] [E] . \textit{rest-var}.$$

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.

- `(mdefmacro {fdefine-compiler-macro} {foo (setf foo)} (macro-λ*)`
 - $\left\{ \left(\begin{array}{l} \textit{declare} \widehat{\textit{decl}}^* \\ \textit{doc} \end{array} \right)^* \textit{form}^{\text{P}_k} \right\}$
 - ▷ 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 **sblock** named *foo*.

(*m*define-symbol-macro *foo* *form*)

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

(*s*macrolet ((*foo* (*macro-λ**) $\left\{ \begin{array}{l} \text{(declare } \widehat{\text{local-decl}}^* \text{)} \\ \text{/doc} \end{array} \right\}$ *macro-form*^{P_k}*)

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

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

(*s*symbol-macrolet ((*foo* *expansion-form**) (*declare* $\widehat{\text{decl}}^*$)* *form*^{P_k})

▷ Evaluate *forms* with locally defined symbol macros *foo*.

(*m*defsetf *function* $\left\{ \begin{array}{l} \widehat{\text{updater}} \text{ } [\widehat{\text{doc}}] \\ \text{(setf-}\lambda^* \text{)} (s\text{-var}^*) \left\{ \begin{array}{l} \text{(declare } \widehat{\text{decl}}^* \text{)} \\ \text{/doc} \end{array} \right\} \text{ } \text{form}^{\text{P}_k} \end{array} \right\}$

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

(*var** [*&optional* $\left\{ \begin{array}{l} \text{var} \\ \text{(var } [\text{init}_{\text{NIL}}] [\text{supplied-p}]) \end{array} \right\}$] [*&rest* *var*]

[*&key* $\left\{ \begin{array}{l} \text{var} \\ \text{(key var)} \end{array} \right\}$] [*init*_{NIL}] [*supplied-p*])

[*&allow-other-keys*] [*&environment* *var*]

▷ Specify how to **setf** a place accessed by *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**) $\left\{ \begin{array}{l} \text{(declare } \widehat{\text{decl}}^* \text{)} \\ \text{/doc} \end{array} \right\}$

form^{P_k})

▷ Specify how to **setf** a place accessed by *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*_{NIL}])

▷ Return lists of temporary variables *arg-vars* and of corresponding *args* as given with *place*, list *newval-vars* with temporary variables corresponding to the new values, and *set-form* and *get-form* specifying in terms of *arg-vars* and *newval-vars* how to **setf** and how to read *place*.

(*m*define-modify-macro *foo* ([*&optional* $\left\{ \begin{array}{l} \text{var} \\ \text{(var } [\text{init}_{\text{NIL}}] [\text{supplied-p}]) \end{array} \right\}$])

[*&rest* *var*] *function* [*doc*])

▷ Define macro *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.

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.

(*f*write-sequence *sequence* $\widehat{\text{stream}} \left\{ \begin{array}{l} \text{:start } \text{start}_{\text{NIL}} \\ \text{:end } \text{end}_{\text{NIL}} \end{array} \right\}$)

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

($\left\{ \begin{array}{l} \text{:array } \text{bool} \\ \text{:base } \text{radix} \\ \text{:case } \left\{ \begin{array}{l} \text{:supcase} \\ \text{:downcase} \\ \text{:capitalize} \end{array} \right. \\ \text{:circle } \text{bool} \\ \text{:escape } \text{bool} \\ \text{:gensym } \text{bool} \\ \text{:length } \{ \text{int} | \text{NIL} \} \\ \text{:level } \{ \text{int} | \text{NIL} \} \\ \text{:lines } \{ \text{int} | \text{NIL} \} \\ \text{:miser-width } \{ \text{int} | \text{NIL} \} \\ \text{:pprint-dispatch } \text{dispatch-table} \\ \text{:pretty } \text{bool} \\ \text{:radix } \text{bool} \\ \text{:readably } \text{bool} \\ \text{:right-margin } \{ \text{int} | \text{NIL} \} \\ \text{:stream } \text{stream}_{\text{v,standard-output*}} \end{array} \right\}$ *foo*)

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

(*f*pprint-fill $\widehat{\text{stream}}$ *foo* [*parenthesis*_{NIL}] [*noop*])

(*f*pprint-tabular $\widehat{\text{stream}}$ *foo* [*parenthesis*_{NIL}] [*noop*] [*n*_{NIL}])

(*f*pprint-linear $\widehat{\text{stream}}$ *foo* [*parenthesis*_{NIL}] [*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 NIL. Usable with *f*format directive ~//.

(*m*pprint-logical-block ($\widehat{\text{stream}}$ *list* $\left\{ \begin{array}{l} \text{:prefix } \text{string} \\ \text{:per-line-prefix } \text{string} \\ \text{:suffix } \text{string}_{\text{NIL}} \end{array} \right\}$))

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

▷ 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 NIL.

(*m*pprint-pop)

▷ Take *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*pprint-tab $\left\{ \begin{array}{l} \text{:line} \\ \text{:line-relative} \\ \text{:section} \\ \text{:section-relative} \end{array} \right\}$ *c* *i* [$\widehat{\text{stream}}_{\text{v,standard-output*}}$])

▷ Move cursor forward to column number *c* + *ki*, *k* ≥ 0 being as small as possible.

(*f*pprint-indent $\left\{ \begin{array}{l} \text{:block} \\ \text{:current} \end{array} \right\}$ *n* [$\widehat{\text{stream}}_{\text{v,standard-output*}}$])

▷ Specify indentation for innermost logical block relative to leftmost position/to current position. Return NIL.

(*m*pprint-exit-if-list-exhausted)

▷ If *list* is empty, terminate logical block. Return NIL otherwise.

(*f*pprint-newline $\left\{ \begin{array}{l} \text{:linear} \\ \text{:fill} \\ \text{:miser} \\ \text{:mandatory} \end{array} \right\}$ [$\widehat{\text{stream}}_{\text{v,standard-output*}}$])

▷ Print a conditional newline if *stream* is a pretty printing stream. Return NIL.

- # $[n]*b^*$** ▷ Bit vector of some (or n) bs 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* \square** ▷ 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**.
- #+*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

- { f prin1
 f print
 f pprint
 f princ}** *foo* [*stream* **v^* standard-output***]
- ▷ Print *foo* to *stream* f **readably**, f **readably** between a newline and a space, f **readably** 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 **readably** or human-readably, respectively.
- (g print-object *object* *stream*)**
 ▷ Print *object* to *stream*. Called by the Lisp printer.
- (m print-unreadable-object (*foo* *stream* **{:type bool**_{NIL} **:identity bool**_{NIL}}) *form*^P***)
- ▷ Enclosed in **#<** and **>**, print *foo* by means of *forms* to *stream*. Return **NIL**.
- (f terpri [*stream* **v^* standard-output***])**
 ▷ Output a newline to *stream*. Return **NIL**.
- (f fresh-line [*stream* **v^* standard-output***])**
 ▷ Output a newline to *stream* and return **T** unless *stream* is already at the start of a line.
- (f write-char *char* [*stream* **v^* standard-output***])**
 ▷ Output *char* to *stream*.
- { f write-string
 f write-line}** *string* [*stream* **v^* standard-output*** **{:start start**_□ **:end end**_{NIL}}]])
- ▷ Write *string* to *stream* without/with a trailing newline.
- (f write-byte *byte* *stream*)** ▷ Write *byte* to binary *stream*.

- &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 **let***.

9.5 Control Flow

- (s if *test* then [*else*_{NIL}])**
 ▷ Return values of *then* if *test* returns T; return values of *else* otherwise.
- (m cond (*test* then^P* **\overline{test}** *)**)
 ▷ Return the values of the first *then** whose *test* returns T; return **NIL** if all *tests* return NIL.
- { m when
 m unless}** *test* *foo*^P*)
 ▷ Evaluate *foos* and return their values if *test* returns T or NIL, respectively. Return **NIL** otherwise.
- (m case *test* (**\overline{key}** *) *foo*^P*) [**{otherwise}** *bar*^P*)_{NIL}]**
 ▷ Return the values of the first *foo** one of whose *keys* is **eq** *test*. Return values of *bars* if there is no matching *key*.
- { m ecase
 m ccase}** *test* (**\overline{key}** *) *foo*^P*)
- ▷ Return the values of the first *foo** one of whose *keys* is **eq** *test*. Signal non-correctable/correctable **type-error** if there is no matching *key*.
- (m and *form**_□)**
 ▷ Evaluate *forms* from left to right. Immediately return **NIL** if one *form*'s value is NIL. Return values of last *form* otherwise.
- (m or *form**_{NIL})**
 ▷ Evaluate *forms* 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 progn *form**_{NIL})**
 ▷ Evaluate *forms* 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*.
- { m prog
 m prog*** (**{*name*
(*name* [*value*_{NIL}])***) (**declare *decl*****) **{*tag*
form}***)
- ▷ Evaluate **s tagbody**-like body with *names* lexically bound (in parallel or sequentially, respectively) to *values*. Return **NIL** or explicitly **m returned** 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*, *cleanups*. Return values of *protected*.
- (s block *name form*^P*)**
 ▷ Evaluate *forms* in a lexical environment, and return their values unless interrupted by **s return-from**.
- (s return-from *foo* [*result*_{NIL}])**
(m return [*result*_{NIL}])
 ▷ Have nearest enclosing **s block** named *foo*/named **NIL**, respectively, return with values of *result*.

- (**s**tagbody $\{\widehat{tag|form}^*\}$)
 ▷ Evaluate *forms* in a lexical environment. *tags* (symbols or integers) have lexical scope and dynamic extent, and are targets for **s**go. Return NIL.
- (**s**go \widehat{tag})
 ▷ Within the innermost possible enclosing **s**tagbody, jump to a tag *f*eq *tag*.
- (**s**catch *tag form*^k)
 ▷ Evaluate *forms* and return 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 NIL.

9.6 Iteration

- ($\left\{ \begin{array}{l} \text{m} \\ \text{m} \end{array} \right\} \left\{ \begin{array}{l} \text{var} \\ \text{var} \end{array} \right\} [start [step]]^* \left\{ \begin{array}{l} \text{stop} \\ \text{result} \end{array} \right\}^* (\text{declare } \widehat{decl}^*)^* \left\{ \begin{array}{l} \widehat{tag} \\ \text{form} \end{array} \right\}^*$)
 ▷ 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 values of result^{*}. Implicitly, the whole form is a **s**block named NIL.
- (*m*dotimes (*var i* [result_{NIL}]) (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 result, *var* is *i*. Implicitly, the whole form is a **s**block named NIL.
- (*m*dolist (*var list* [result_{NIL}]) (declare \widehat{decl}^*)^{*} $\{\widehat{tag|form}^*\}$)
 ▷ Evaluate **s**tagbody-like body with *var* successively bound to the elements of *list*. Upon evaluation of 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*_{NIL} ▷ Give *m*loop's implicit **s**block a name.
- $\left\{ \begin{array}{l} \text{with} \\ \text{with} \end{array} \right\} \left\{ \begin{array}{l} \text{var-s} \\ \text{var-s} \end{array} \right\} [d\text{-type}] [= \text{foo}]^+$
 $\left\{ \begin{array}{l} \text{and} \\ \text{and} \end{array} \right\} \left\{ \begin{array}{l} \text{var-p} \\ \text{var-p} \end{array} \right\} [d\text{-type}] [= \text{bar}]^*$
 where destructuring type specifier *d-type* has the form
 $\left\{ \begin{array}{l} \text{fixnum} | \text{float} | \text{T} | \text{NIL} \\ \text{of-type} \end{array} \right\} \left\{ \begin{array}{l} \text{type} \\ \text{type} \end{array} \right\}^*$
 ▷ Initialize (possibly trees of) local variables *var-s* sequentially and *var-p* in parallel.
- $\left\{ \begin{array}{l} \text{for} | \text{as} \\ \text{for} | \text{as} \end{array} \right\} \left\{ \begin{array}{l} \text{var-s} \\ \text{var-s} \end{array} \right\} [d\text{-type}]^+ \left\{ \begin{array}{l} \text{and} \\ \text{and} \end{array} \right\} \left\{ \begin{array}{l} \text{var-p} \\ \text{var-p} \end{array} \right\} [d\text{-type}]^*$
 ▷ 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**.

- *read-base***₁₀ ▷ Radix for reading **integers** and **ratios**.
- *read-default-float-format***_{single-float}
 ▷ Floating point format to use when not indicated in the number read.
- *read-suppress***_{NIL} ▷ If T, reader is syntactically more tolerant.
- (*f*set-macro-character *char function* [*non-term-p*_{NIL}] [*rt*_{*readtable*}])
 ▷ Make *char* a macro character associated with *function* of stream and *char*. Return T.
- (*f*get-macro-character *char* [*rt*_{*readtable*}])
 ▷ Reader macro function associated with *char*, and T if *char* is a non-terminating macro character.
- (*f*make-dispatch-macro-character *char* [*non-term-p*_{NIL}] [*rt*_{*readtable*}])
 ▷ Make *char* a dispatching macro character. Return T.
- (*f*set-dispatch-macro-character *char sub-char function* [*rt*_{*readtable*}])
 ▷ Make *function* of stream, *n*, *sub-char* a dispatch function of *char* followed by *n*, followed by *sub-char*. Return T.
- (*f*get-dispatch-macro-character *char sub-char* [*rt*_{*readtable*}])
 ▷ 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 ▷ Regarding line on which it appears.
; *continuation*
- (*foo** [*bar*_{NIL}]) ▷ List of *foos* with the terminating cdr *bar*.
- " ▷ Begin and end of a string.
- '*foo* ▷ (**s**quote *foo*); *foo* unevaluated.
- `([*foo*] [*bar*] [*@baz*] [*,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***Rn*
 ▷ Integer of radix 2, 8, 10, 16, or *r*; $2 \leq r \leq 36$.
- n*/*d* ▷ The **ratio** $\frac{n}{d}$.
- $\left\{ [m].n \left[\left\{ \begin{array}{l} \text{S} | \text{F} | \text{D} | \text{L} | \text{E} \end{array} \right\} x_{\text{E0}} \right] m \left[\left[\begin{array}{l} \text{S} | \text{F} | \text{D} | \text{L} | \text{E} \end{array} \right] x \right] \right\}$
 ▷ $m.n \cdot 10^x$ 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*.
- #nA***sequence* ▷ *n*-dimensional array.
- #**[*n*](*foo*^{*})
 ▷ Vector of some (or *n*) *foos* filled with last *foo* if necessary.

{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 $\{step_{\square} | function_{\#cdn}\}$
 ▷ Specify the (positive) decrement or increment or the *function* of one argument returning the next part of the list.

= *foo* **[then** *bar*_{*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*_{**package**}
 ▷ Bind *var* successively to the accessible symbols, or the present symbols, or the external symbols respectively, of *package*.

{do|doing} *form*⁺ ▷ Evaluate *forms* 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-clauses*; otherwise, evaluate *k-clause* and *l-clauses*.

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 *fappend* or *fnconc*, 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.

(f type-of foo) ▷ Type of *foo*.

(mcheck-type place type [string_{a|an} 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_{NIL}])
 ▷ Element type of most specialized array capable of holding elements of *type*.

(mdeftype foo (macro-λ*) $\left\{ \begin{array}{l} \text{(declare } \widehat{decl}^* \text{)}^* \\ \text{doc} \end{array} \right\}$ *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 *forms* to *args* returning the new type. For $(macro-λ^*)$ see page 19 but with default value of ***** instead of NIL. *forms* are enclosed in an implicit **sblock** named *foo*.

(eql foo)
(member foo*) ▷ Specifier for a type comprising *foo* or *foos*.

(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*_{\square}) ▷ Type specifier for intersection of *types*.

(or type*_{NIL}) ▷ Type specifier for union of *types*.

(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 stream-p foo)
(f pathname-p foo) ▷ T if *foo* is of indicated type.
(f readable-p 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.)

$\left\{ \begin{array}{l} \text{(mwith-slots } (\widehat{\text{slot}} (\widehat{\text{var slot}})^*) \\ \text{(mwith-accessors } (\widehat{\text{var accessor}})^*) \end{array} \right\}$ *instance* (**declare** $\widehat{\text{decl}}^*$)^{*} *form*^{P*})
 ▷ Return values of forms after evaluating them in a lexical environment with slots of *instance* visible as **setfable slots** or *vars*/with *accessors* of *instance* visible as **setfable vars**.

$\text{(gclass-name class)}$
 $\text{((setf gclass-name) new-name class)}$ ▷ Get/set name of class.

(fclass-of foo) ▷ Class foo is a direct instance of.

$\text{(gchange-class } \widehat{\text{instance new-class}} \{[:]\text{initarg value}\}^* \text{ other-keyarg}^*)$
 ▷ 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 *values* of the corresponding *initargs* if any, or with the values of their **:initform** forms if not.

$\text{(gmake-instances-obsolete class)}$
 ▷ Update all existing instances of *class* using $\text{gupdate-instance-for-redefined-class}$.

$\left\{ \begin{array}{l} \text{(ginitialize-instance } \textit{instance} \\ \text{(gupdate-instance-for-different-class } \textit{previous current} \\ \{[:]\text{initarg value}\}^* \text{ other-keyarg}^*) \end{array} \right\}$
 ▷ Set slots on behalf of gmake-instance /of gchange-class by means of $\text{gshared-initialize}$.

$\text{(gupdate-instance-for-redefined-class } \textit{new-instance added-slots discarded-slots discarded-slots-property-list} \{[:]\text{initarg value}\}^* \text{ other-keyarg}^*)$
 ▷ On behalf of $\text{gmake-instances-obsolete}$ and by means of $\text{gshared-initialize}$, set any *initarg* slots to their corresponding *values*; set any remaining *added-slots* to the values of their **:initform** forms. Not to be called by user.

$\text{(gallocate-instance class } \{[:]\text{initarg value}\}^* \text{ other-keyarg}^*)$
 ▷ Return uninitialized instance of *class*. Called by gmake-instance .

$\text{(gshared-initialize } \textit{instance} \left\{ \begin{array}{l} \textit{initform-slots} \\ \text{T} \end{array} \right\} \{[:]\text{initarg-slot value}\}^* \text{ other-keyarg}^*)$
 ▷ Fill the *initarg-slots* of *instance* with the corresponding *values*, and fill those *initform-slots* that are not *initarg-slots* with the values of their **:initform** forms.

$\text{(gslot-missing class } \textit{instance slot} \left\{ \begin{array}{l} \text{setf} \\ \text{slot-boundp} \\ \text{slot-makunbound} \\ \text{slot-value} \end{array} \right\} [\textit{value}]$

$\text{(gslot-unbound class } \textit{instance slot})$
 ▷ 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

(fnext-method-p) ▷ T if enclosing method has a next method.

$\text{(mdefgeneric } \left\{ \begin{array}{l} \textit{foo} \\ \text{(setf } \textit{foo}) \end{array} \right\} (\textit{required-var}^* [\&\text{optional } \left\{ \begin{array}{l} \textit{var} \\ \text{(var)} \end{array} \right\}]^* [\&\text{rest } \textit{var}] [\&\text{key } \left\{ \begin{array}{l} \textit{var} \\ \text{(var)} \end{array} \right\}^* (\text{:key } \textit{var}) \} [\&\text{allow-other-keys}]) \left\{ \begin{array}{l} (\text{:argument-precedence-order } \textit{required-var}^+) \\ (\text{declare } (\text{optimize } \textit{method-selection-optimization})^+) \\ (\text{:documentation } \textit{string}) \\ (\text{:generic-function-class } \textit{gf-class} \text{standard-generic-function}) \\ (\text{:method-class } \textit{method-class} \text{standard-method}) \\ (\text{:method-combination } \textit{c-type} \text{standard } \textit{c-arg}^*) \\ (\text{:method } \textit{defmethod-args})^* \end{array} \right\})$

▷ 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 fabort and fmuffle-warning , or return NIL for the rest.

$\text{(mwith-condition-restarts } \textit{condition restarts form}^{\text{P}})$
 ▷ Evaluate *forms* with *restarts* dynamically associated with *condition*. Return values of forms.

$\text{(f arithmetic-error-operation } \textit{condition})$
 $\text{(f arithmetic-error-operands } \textit{condition})$
 ▷ List of function or of its operands respectively, used in the operation which caused *condition*.

$\text{(fcell-error-name } \textit{condition})$
 ▷ Name of cell which caused *condition*.

$\text{(funbound-slot-instance } \textit{condition})$
 ▷ Instance with unbound slot which caused *condition*.

$\text{(fprint-not-readable-object } \textit{condition})$
 ▷ The object not readably printable under *condition*.

$\text{(fpackage-error-package } \textit{condition})$
 $\text{(f file-error-pathname } \textit{condition})$
 $\text{(f stream-error-stream } \textit{condition})$
 ▷ Package, path, or stream, respectively, which caused the *condition* of indicated type.

$\text{(f type-error-datum } \textit{condition})$
 $\text{(f type-error-expected-type } \textit{condition})$
 ▷ Object which caused *condition* of type **type-error**, or its expected type, respectively.

$\text{(fsimple-condition-format-control } \textit{condition})$
 $\text{(fsimple-condition-format-arguments } \textit{condition})$
 ▷ Return fformat control or list of fformat arguments, respectively, of *condition*.

$\text{**break-on-signals**}$ NIL
 ▷ Condition type debugger is to be invoked on.

debugger-hook 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.

$\text{(f typep } \textit{foo type} [\textit{environment} \text{NIL}])$ ▷ T if *foo* is of *type*.

$\text{(f subtypep } \textit{type-a type-b} [\textit{environment}])$
 ▷ Return T if *type-a* is a recognizable subtype of *type-b*, and NIL if the relationship could not be determined.

$\text{(sthe } \widehat{\text{type form}})$ ▷ Declare values of form to be of *type*.

$\text{(f coerce } \textit{object type})$ ▷ Coerce object into *type*.

$\text{(m typecase } \textit{foo} (\widehat{\text{type } a\text{-form}}^{\text{P}})^* [\left\{ \begin{array}{l} \text{otherwise} \\ \text{T} \end{array} \right\} b\text{-form}_{\text{NIL}}^{\text{P}}])$
 ▷ Return values of the first a-form* whose *type* is *foo* of. Return values of b-forms if no *type* matches.

$\left\{ \begin{array}{l} \text{(m etypecase)} \\ \text{(m ctypecase)} \end{array} \right\} \textit{foo} (\widehat{\text{type form}}^{\text{P}})^*$
 ▷ Return values of the first form* whose *type* is *foo* of. Signal non-correctable/correctable **type-error** if no *type* matches.

(*m*handler-case *foo* (*type* ([*var*]) (declare $\widehat{\text{decl}}^*$)^{*} *condition-form*^P*)
 [(:no-error (*ord-λ**) (declare $\widehat{\text{decl}}^*$)^{*} *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.

(*m*with-simple-restart ({*restart*
NIL } *control* *arg**) *form*^P*)
 ▷ Return values of *forms* unless *restart* is called during their evaluation. In this case, describe *restart* using *f*format *control* and *args* (see page 38) and return NIL and T.

(*m*restart-case *form* (*restart* (*ord-λ**) { :interactive *arg-function*
:report { *report-function*
string *restart*
:test *test-function* } })
 (declare $\widehat{\text{decl}}^*$)^{*} *restart-form*^P*)
 ▷ Return values of *form* or, if during evaluation of *form* one of the dynamically established *restarts* is called, the values of its *restart-forms*. A *restart* is visible under *condition* if (*funcall* #'*test-function* *condition*) returns T. If presented in the debugger, *restarts* are described by *string* or by #'*report-function* (of a stream). A *restart* can be called by (*invoke-restart* *restart* *arg**)^{*}, where *args* match *ord-λ**, or by (*invoke-restart-interactively* *restart*) where a list of the respective *args* is supplied by #'*arg-function*. See page 18 for *ord-λ*.*

(*m*restart-bind ({*restart*
NIL } *restart-function*
 { :interactive-function *arg-function*
:report-function *report-function*
:test-function *test-function* })^{*}) *form*^P*)
 ▷ Return values of *forms* evaluated with dynamically established *restarts* whose *restart-functions* 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, *restarts* are described by *restart-function* (of a stream). A *restart* can be called by (*invoke-restart* *restart* *arg**)^{*}, where *args* must be suitable for the corresponding *restart-function*, or by (*invoke-restart-interactively* *restart*) where a list of the respective *args* 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 its values.

({*f*find-restart
:compute-restarts *name* } [*condition*])
 ▷ Return innermost *restart name*, or a 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 NIL if search is unsuccessful.

(*f*restart-name *restart*) ▷ Name of restart.

({*f*abort
:muffle-warning
:continue
:store-value *value*
:use-value *value* } [*condition*])

▷ Define or modify generic function *foo*. Remove any methods previously defined by *defgeneric*. *gf-class* and the lambda parameters *required-var** and *var** must be compatible with existing methods. *defmethod-args* resemble those of *mdefmethod*. For *c-type* see section 10.3.

(*f*ensure-generic-function ({*foo*
(*setf* *foo*) }
 { :argument-precedence-order *required-var*⁺
:declare (optimize *method-selection-optimization*)
:documentation *string*
:generic-function-class *gf-class*
:method-class *method-class*
:method-combination *c-type* *c-arg**
:lambda-list *lambda-list*
:environment *environment* })

▷ Define or modify 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 ({*foo*
(*setf* *foo*) } { :before
:after
:around } [*primary method*]
 {*var*
({*spec-var* {*class*
(*eql* *bar*) } } })^{*} [*&optional*
 {*var*
(*var* [*init* [*supplied-p*]]) })^{*} [*&rest* *var*] [*&key*
 {*var*
({*key* *var* }) [*init* [*supplied-p*]]) })^{*} [*&allow-other-keys*]
 [*&aux* {*var*
(*var* [*init*]) }] }) { (declare $\widehat{\text{decl}}^*$)^{*} } *form*^P*)

▷ Define new method for generic function *foo*. *spec-vars* specialize to either being of *class* or being *eql bar*, respectively. On invocation, *vars* and *spec-vars* of the new method act like parameters of a function with body *form**. *forms* are enclosed in an implicit **block** *foo*. Applicable *qualifiers* depend on the **method-combination** type; see section 10.3.

({*g*add-method
:remove-method } *generic-function* *method*)

▷ Add (if necessary) or remove (if any) *method* to/from *generic-function*.

(*g*find-method *generic-function* *qualifiers* *specializers* [*error*])

▷ Return suitable method, or signal **error**.

(*g*compute-applicable-methods *generic-function* *args*)

▷ List of methods suitable for *args*, most specific first.

(*f*call-next-method *arg** [*current args*])

▷ From within a method, call next method with *args*; return 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.

({*f*invalid-method-error *method*
:method-combination-error } *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.

(*gfunction-keywords* *method*)

▷ Return list of keyword parameters of *method* and T if other keys are allowed.

(*gmethod-qualifiers* *method*)

▷ 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, **fcall-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 **fcall-next-method** if any, or of the generic function; and which can call less specific primary methods via **fcall-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-types* defined by the short form of **mdefine-method-combination**.

(*mdefine-method-combination* *c-type*

$$\left\{ \begin{array}{l} \text{:documentation } \textit{string} \\ \text{:identity-with-one-argument } \textit{bool} \underline{\text{NIL}} \\ \text{:operator } \textit{operator} \underline{\textit{c-type}} \end{array} \right\}$$

▷ **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, **fcall-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 $\left\{ \begin{array}{l} \text{:most-specific-first} \\ \text{:most-specific-last} \end{array} \right\}$ (specified as *c-arg* in **mdefgeneric**). Using *c-type* as the *qualifier* in **mdefmethod** makes the method primary.

(*mdefine-method-combination* *c-type* (*ord-λ**) ((*group*

$$\left\{ \begin{array}{l} \text{*} \\ \textit{qualifier}* \textit{[*]} \\ \textit{predicate} \\ \text{:description } \textit{control} \\ \text{:order } \left\{ \begin{array}{l} \text{:most-specific-first} \\ \text{:most-specific-last} \end{array} \right\} \underline{\textit{most-specific-first}} \textit{)*} \\ \text{:required } \textit{bool} \\ \left\{ \begin{array}{l} \text{:arguments } \textit{method-combination-λ*} \\ \text{:generic-function } \textit{symbol} \end{array} \right\} \textit{body}^{\text{P}} \\ \left\{ \begin{array}{l} \text{(declare } \widehat{\textit{decl}} \textit{)*} \\ \textit{doc} \end{array} \right\} \end{array} \right\}$$

▷ **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. **mdefgeneric**), with *symbol* bound to the generic function, with *method-combination-λ** bound to the arguments of the generic function, and with *groups* bound to lists of methods. An applicable method becomes a member of the left-most *group* whose *predicate* or *qualifiers* match. Methods can be called via **mcall-method**. Lambda lists (*ord-λ**) and (*method-combination-λ**) according to *ord-λ* on page 18, the latter enhanced by an optional **&whole** argument.

(*mcall-method*

$$\left\{ \begin{array}{l} \widehat{\textit{method}} \\ \text{(mmake-method } \widehat{\textit{form}} \textit{)} \end{array} \right\} \left[\left(\left\{ \begin{array}{l} \widehat{\textit{next-method}} \\ \text{(mmake-method } \widehat{\textit{form}} \textit{)} \end{array} \right\} \right)^* \right]$$

▷ 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.

(*mdefine-condition* *foo* (*parent-type* *condition**)

$$\left\{ \begin{array}{l} \textit{slot} \\ \left\{ \begin{array}{l} \text{:reader } \textit{reader} \textit{*} \\ \text{:writer } \left\{ \begin{array}{l} \textit{writer} \\ \text{(self } \textit{writer} \textit{)} \end{array} \right\} \textit{*} \\ \text{:accessor } \textit{accessor} \textit{*} \\ \text{:allocation } \left\{ \begin{array}{l} \text{:instance} \\ \text{:class} \end{array} \right\} \underline{\textit{instance}} \end{array} \right\} \\ \text{:initarg } \textit{[:initarg-name]} \textit{*} \\ \text{:initform } \textit{form} \\ \text{:type } \textit{type} \\ \text{:documentation } \textit{slot-doc} \end{array} \right\} \\ \left\{ \begin{array}{l} \text{(default-initargs } \textit{[name value]} \textit{)*} \\ \text{(documentation } \textit{condition-doc} \textit{)} \\ \text{(report } \left\{ \begin{array}{l} \textit{string} \\ \textit{report-function} \end{array} \right\} \end{array} \right\} \end{array} \right\}$$

▷ Define, as a subtype of *parent-types*, 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 (*self* (*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.

(*fmake-condition* *condition-type* {*[:initarg-name* *value*}*)

▷ Return new instance of condition-type.

$$\left\{ \begin{array}{l} \textit{fsignal} \\ \textit{fwarn} \\ \textit{ferror} \end{array} \right\} \left\{ \begin{array}{l} \textit{condition} \\ \textit{condition-type} \{ \textit{[:initarg-name value]} \textit{*} \} \\ \textit{control arg} \textit{*} \end{array} \right\}$$

▷ Unless handled, signal as **condition**, **warning** or **error**, respectively, *condition* or a new instance of *condition-type* or, with **fformat** *control* and *args* (see page 38), **simple-condition**, **simple-warning**, or **simple-error**, respectively. From **fsignal** and **fwarn**, return NIL.

$$\left(\textit{fcerror} \textit{continue-control} \left\{ \begin{array}{l} \textit{condition continue-arg} \textit{*} \\ \textit{condition-type} \{ \textit{[:initarg-name value]} \textit{*} \} \\ \textit{control arg} \textit{*} \end{array} \right\} \right)$$

▷ Unless handled, signal as correctable **error** *condition* or a new instance of *condition-type* or, with **fformat** *control* and *args* (see page 38), **simple-error**. In the debugger, use **fformat** arguments *continue-control* and *continue-args* to tag the continue option. Return NIL.

(*mignore-errors* *form*^P)

▷ Return values of forms or, in case of **errors**, NIL and the condition.

(*finvoke-debugger* *condition*)

▷ Invoke debugger with *condition*.

$$\left(\textit{massert} \textit{test} \left[\left(\textit{place} \textit{*} \right) \left\{ \begin{array}{l} \textit{condition continue-arg} \textit{*} \\ \textit{condition-type} \{ \textit{[:initarg-name value]} \textit{*} \} \\ \textit{control arg} \textit{*} \end{array} \right\} \right] \right]$$

▷ If *test*, which may depend on *places*, returns NIL, signal as correctable **error** *condition* or a new instance of *condition-type* or, with **fformat** *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.