This proposal is the joint work of a collaboration with the following members.

Katsuhiro Yuara (Hitachi, Ltd.)
Mitsuki Kato (Fujitsu Laboratories Ltd.)
Yukiko Hashimoto (NEC Corp.)
Kazusaku Kawagome (SORD Computer Corp.)
Susumu Kawai (Hitachi Digital Equipment Corp.)
Shigeki Harada (Sharp Corp.)
Yoichi Yamamura (Nippon UNIVAC Kaisha Ltd.)
Takashi Shinzui (Fujixerox Co., Ltd.)
Takashi Hamada (Matsushita Electric Industrial Co., Ltd.)
Haruyuki Kawabe (Nippon UNIVAC Kaisha Ltd.)
Atsuko Tanaka (Toshiba Corp.)
Nobuyuki Saji (NEC Corp.)
Masayuki Ida (Aoyama Gakuin Univ.)

1. Goals of Common Lisp/Core

(1) To set up the standard specification for small/sized personal computers and hand-held computers.

(2) To set up basic level of Common Lisp.

Common Lisp should have two levels: one is full set which is growing up and the other is subset which is fixed.

2. Discussions for Common Lisp/Core

I. (1) IUP members of subset WG in Jeda Common Lisp Committee, the authors have discussed as to the following subjects since Dec. 1985.

(a) A review of Ida's personal proposal for a subset.
(b) An examination of the necessities and the difficulties to implement each function.
(c) A decision on the basic issues for Common Lisp/Core.
(d) A choice of the functions based on the vote by the members.

(2) The authors had an open meeting for the announcement and the decision of Common Lisp/Core on Jul. 8 '86. About twenty IUP and programming language researchers and about forty IUP users met and commented as follows.

(i) From the implementor's point of view, it is supposed. the scale of Common Lisp/Core is not so much smaller than that of the full set.

(ii) From the user's point of view, useful functions are selected all over sections and it is expected that more functions are selected from packages, streams or declarations.

3. Basic issues and decisions

(1) Arms and legs of Common Lisp are kept, because it is important to transfer programs in the subset to those in the full set easily.

(2) It is aimed that the number of functions in Common Lisp/Core is about a half in the full set.

(3) The following features of Common Lisp are kept:

(i) scope and extent rules including lexical closure.
(ii) keyword parameters.
(iii) the principles of type hierarchy and generic function.
(iv) functional richness over Unix (Tokyo Univ. '81: one of the most famous Lisp in Japan) or FranzLisp.
(v) some useful on characteristic data types: bignum, ratio, structure and readable

(4) The choice of functions is based on the following rules in order.

(i) Functions related to the features (3).
(ii) Functions having high necessities.
(iii) Functions not having as many difficulties to implement.

The following issues are also discussed, but these policies are not adopted.

(a) To keep the "language" oriented features and to leave out the "system" oriented features.

(b) To keep the kernel part that users for themselves can not "define" or "defframe".

Some members of this WG opposed to the issues (3) and (4), argued for the Common Lisp/Core approach (b), and then they did not join this proposal.

4. Summary of Common Lisp/Core

Major deleted items are: most of system parameter constants, complex numbers, most of package features, local functions, iradible arrays, hash-tables, and pathnames.
<table>
<thead>
<tr>
<th>intro</th>
<th>Page 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. Program Structure</td>
<td>x defparameter 4/0 5/2</td>
</tr>
<tr>
<td>6. Predicates</td>
<td>x equisp. complexp 26/2 32/2</td>
</tr>
<tr>
<td>7. Control Structure</td>
<td>x progb. progb. prog2 49/0 67/2</td>
</tr>
<tr>
<td>x flit. labels. macrolet o tagbody. go</td>
<td></td>
</tr>
<tr>
<td>8. Macro</td>
<td>x all declaration specifiers 4/0 4/1</td>
</tr>
<tr>
<td>9. Declarations</td>
<td>x proclaim. locally 2/- 4/-</td>
</tr>
<tr>
<td>10. Symbols</td>
<td>x getf. remf 9/- 13/-</td>
</tr>
<tr>
<td>11. Packages</td>
<td>x almost all 4/0 26/2</td>
</tr>
<tr>
<td>x intern. unintern. find-symbol. do-all-symbols</td>
<td></td>
</tr>
<tr>
<td>12. Numbers</td>
<td>x complex numbers 49/1 96/44</td>
</tr>
<tr>
<td>o ratio. pi</td>
<td></td>
</tr>
<tr>
<td>13. Characters</td>
<td>x bits. font attributes 28/0 36/7</td>
</tr>
<tr>
<td>14. Sequences</td>
<td>x xxx-if. -if-not 22/- 44/-</td>
</tr>
<tr>
<td>15. Lists</td>
<td>x c....r. xxx-if. -if-not 57/- 94/-</td>
</tr>
<tr>
<td>16. Hash Tables</td>
<td>x all 0/- 8/-</td>
</tr>
<tr>
<td>17. Arrays</td>
<td>Only simple-arrays are available 7/0 31/3</td>
</tr>
<tr>
<td>x adjustable array. fill-pointer</td>
<td></td>
</tr>
<tr>
<td>18. Strings</td>
<td>o almost all 25/- 25/-</td>
</tr>
<tr>
<td>19. Structures</td>
<td>o defstruct</td>
</tr>
<tr>
<td>20. The Evaluator</td>
<td>x hooks 1/- 1/-</td>
</tr>
<tr>
<td>21. Streams</td>
<td>x make-xx-stream functions 2/4 4/12</td>
</tr>
<tr>
<td>22. Input/Output</td>
<td>x :aborn of &quot;close&quot; 5/7 16/7</td>
</tr>
<tr>
<td>23. File System</td>
<td>Pathname is a string 28/4 41/14</td>
</tr>
<tr>
<td>24. Interface</td>
<td></td>
</tr>
<tr>
<td>25. Miscellaneous</td>
<td>o error. warn. break 3/0 10/1</td>
</tr>
<tr>
<td>32. Features</td>
<td>19/2 32/2</td>
</tr>
<tr>
<td>Total</td>
<td>356/20 622/101</td>
</tr>
</tbody>
</table>

5. Difference from Ida's personal proposal

(1) added features in Common Lisp/Core

classification parameters. readtables. tagbody. go.
character predicates ignoring case. string functions and so on
(2) deleted features in Common Lisp/Core
list functions. file i/o functions and so on
(3) number of functions. variables and constant
Common Lisp/Core 376 (356,17,3)
Ida's personal proposal 341 (332, 7,2)

The authors will appreciate opinions from all the persons through network.
2. Data Types

2.1 Numbers
   NUMBER
   INTEGER
   FIXNUM
   BIGNUM
   RATIONAL
   RATIO
   FLOAT (SHORT-/-SINGLE-/-DOUBLE-/-LONG-)
   -- Deleted Item  COMPLEX

2.2 Characters
   CHARACTER
   STANDARD-CHARACTER
   STRING-CHARACTER
   -- Comment  -> bits-attribute and font-attribute may be zero.

2.3 Symbols
   SYMBOL

2.4 Lists and Conses
   LIST
   CONS
   NULL

2.5 Arrays
   ARRAY
   SIMPLE-ARRAY
   VECTOR
   SIMPLE-VECTOR
   STRING
   SIMPLE-STRING
   -- Deleted Item  BIT-VECTOR
   -- Comment  -> ARRAY means SIMPLE-ARRAY, and rank is restricted to max 3.

2.6 Hash Tables
   -- Deleted Item  HASHTABLES

2.7 Readables
   READABLE

2.8 Packages
   PACKAGE
   -- Comment  -> Package is restricted to LISP package.

2.9 Pathnames
   -- Deleted Item  PATHNAME

2.10 Streams
   STREAM

2.11 Random-States
   -- Deleted Item  RANDOM-STATES

2.12 Structures
   STRUCTURE

2.13 Functions
   FUNCTION
   COMPILED-FUNCTION
   LAMBDA-EXPRESSION
   SYMBOL

3. Scope and Extent
   Same as CLTL
   -- Comment  -> Modern Lisp must have the compiler. The compatibility of compiler and interpreter is one of the major goals of CL. We evaluate the effort to make the semantics of Lisp clearer.

4. Type Specifiers

4.1 Type Specifier Symbols
   The type symbols are the same as data types.

4.2 Type Specifier Lists
   Type specifier lists are allowed.

4.3 Predicating Type Specifiers
   -- Deleted Item  SATISFIES predicate-name

4.4 Type Specifiers That Combine
   Type specifier combination is omitted.

4.5 Type Specifier That Specialize
   Type specifier specializations are omitted.

4.6 Type Specifiers That Abbreviate
   Type specifier abbreviations are omitted.

4.7 Defining New Type Specifiers
subset.long

101  -- Deleted item --
102  DEFTYPE name lambda-list (declaration|doc-string)* (form)*  [Macro]
103  ________________________________
104  4.8 Type Conversion Function
105  _____ COREFACT object result-type  [Function]
106  ________________________________
107  4.9 Determining the Type of an Object
108  TYPEOF object  [Function]
109  ________________________________
110  5. Program Structure
111  ________________________________
112  5.1 Forms
113  Same as CLL except absence of some special forms.
114  -- Deleted items ------------------
115  Following special forms:
116  PROG
117  COMPILE-LET
118  FLET
119  LABELS
120  MACROLET
121  ________________________________
122  5.2 Functions
123  -- Deleted items ------------------
124  LAMBDA-LIST-KEYWORDS  [Constant]
125  LAMBDA-PARAMETERS-LIST  [Constant]
126  --> to keep the system compact.
127  ________________________________
128  5.3 Top-level Forms
129  DEFUN name lambda-list (declaration | doc-string)* (form)*  [Macro]
130  DEFPAR name (initial-value [documentation])  [Macro]
131  EVAL-WHEN ((situation)*) (form)*  [Special Form]
132  -- Deleted items ------------------
133  DEFPARAMETER name initial-value [documentation]  [Macro]
134  --> It is redundant
135  ________________________________
136  6. Predicates
137  ________________________________
138  6.1. Logical Values
139  NIL  [Constant]
140  T  [Constant]
141  ________________________________
142  6.2. Data Type Predicates
143  ________________________________
144  6.2.1. General Type Predicates
145  TYPEP object type  [Function]
146  SUBTYPEP typep type2  [Function]
147  ________________________________
148  6.2.2. Specific Data Type Predicates
149  NULL object  [Function]
150  SYMBOLOP object  [Function]
151  ATOM object  [Function]
152  CONSP object  [Function]
153  LISTP object  [Function]
154  NUMBERP object  [Function]
155  INTEGERP object  [Function]
156  RATIONALP object  [Function]
157  FLOATP object  [Function]
158  STRINGP object  [Function]
159  VECTORP object  [Function]
160  SIMPLE-VECTOR-P object  [Function]
161  SIMPLE-STRING-P object  [Function]
162  ARRAYP object  [Function]
163  FUNCTIONP object  [Function]
164  COMPILED-FUNCTION-P object  [Function]
165  STREAMP object  [Function]
166  COMMONP object  [Function]
167  -- see also ...  [Function]
168  STANDARD-CHAR-P, STRING-CHAR-P, READTABLEP
169  -- Comment ----------------------
170  Data type predicates' existence is depending on data type existence
171  itself.
172  Data type hierarchy remains.
173  There is a recommend with using following predicates.
174  VECTORP, STRINGP
175  Use SIMPLE-xxx functions of these is much better, because data type ARRAY
176  has no fill-pointer. is not adjustable. is not able to be displaced
177  from another array.
178  ________________________________
179  6.3. Equality Predicates
180  EQ x y  [Function]
181  EQL x y  [Function]
182  EQUAL x y  [Function]
183  -- Deleted item ------------------
184  EQUALP x y  [Function]
185  --> Other function can replace this. Use type specific function to compare
186  rougher or more exactly.
187  ________________________________
188  6.4. Logical Operators
189  NOT x  [Function]
190  AND (form)*  [Macro]
191  OR (form)*  [Macro]
7 Control Structure

7.1 Constants and Variables

7.1.1 Reference

QUOTE object

FUNCTION fn

SYMBOL-VALUE symbol

SYMBOL-FUNCTION symbol

BOUND symbol

GROUND symbol

SPECIAL-FILE symbol

SETQ (var form)*

PICTQ (var form)*

SET symbol value

MAKEBOUND symbol

MAKEBOUND symbol

-- Comments

-> These are primitive.

FUNCTION is very important, because it returns "lexical closure'.

7.2 Generalized Variables

SETF (place newvalue)*

-> This is very important macro on CL.

-- Deleted Items

-> This macro can be defined easily using SETF macro.

DEFINE-MODIFY-MACRO name lambda-list function [doc-string] [Macro]

DEFSETF access-fn (update-fn [doc-string] [Macro]

lambda-list [store-variable])

😀 (declaration [doc-string] (form)*)

DEFINE-SETF-METHOD access-fn lambda-list

(declaration [doc-string] (form)* [Macro]

GET-SETF-METHOD

GET-SETF-METHOD-MULTIPLE-VALUE form

-> These macros and functions are provided to define macros

like a SETF and to modify SETF.

-- Forms of Place

1) The name of a variable.

2) Functions.

ANGQ NTH

REST FIRST SECOND

THIRD FOURTH FIFTH

SIXTH SEVENTH EIGHTH

NINTH TENTH CAR

CHAR C..R

SRELP GET SYMBOL-PLIST

SYMBOL-VALUE SYMBOL-ROUTE MACRO-FUNCTION

3) Selector function constructed by DEFSTRUCT.

4) Functions.

-> CHAR SCHEL SUBSEQ

5) A THE type declaration form.

6) A call to APPLY where the first argument form is of the form #'name.

7) A macro call.

-- Deleted Forms of Place

1) Functions.

C...R

GETF

GETHASH

BIT

LDB

2) Any form for which a DEFSETF or DEFINE-SETF-METHOD declaration

has been made.

--> These functions are deleted on CL Core.

7.3 Function Invocation

APPLY function arg more-args [Function]

FUNCALL fn arg argument [Function]

-> These functions are very primitive in LISP language.

CALL-ARGUMENT-LIMIT [Constant]

7.4 Simple Sequencing

PROGN (form)*

PROG1 first (form)*

-> PROG2 is provided mostly for historical compatibility.

-> PROG2 is provided mostly for historical compatibility.

7.5 Establishing New Variable Bindings

LET ((var | (var value)) (declaration) (form)* [Special Form]

LET ((var | (var value)) (declaration) (form)* [Special Form]

-> These are very important to establish new variable binding.

-> These are very important to establish new variable binding.

COMPILE-LET ((var | (var value)) (form)* [Special Form]

PROG symbols values (form)*

-> These are not important to ordinary users.

FLET ((name lambda-list (declaration | doc-string) (form)* [Special Form]

LABELS ((name lambda-list (declaration | doc-string) (form)* [Special Form]

MACROLET ((name varlist (declaration | doc-string) (form)* [Special Form]

-> In ordinary, local named functions and macros are not necessary.

7.6 Conditionals

IF test then else

WHEN test (form)*

UNLESS test (form)*

COND ((test (form))*
CASE keyform (((key)* | key)(form)*)
-- Deleted items ------------------------------- [Macro]
303  TYPECASE keyform ((type (form)*)
304  -- Comments ------------------------------- [Macro]
305  -> '#IF is the primitive of the CL
306  -> #COND is one of the originator of lisp.
307  it is constructive but not harmful.
308  ---------------------------------------------------------------
309  7.7 Blocks and Exits
310  BLOCK name (form)*
311  RETURN-FROM name [result]
312  RETURN [result]
313  -- Comments ------------------------------- [Macro]
314  -> These features are quite necessary for constructive programming.
315  ---------------------------------------------------------------
316  7.8 Iteration
317  7.8.1 Infinite Iteration
318  LOOP (form)*
319  -- Comments ------------------------------- [Macro]
320  7.8.2 Infinite Iteration
321  DO (((var [(init [step])])]*) (end-test (result)*)
322  (declaration)* (tag statement)*
323  DO* (((var [(init [step])])]*) (end-test (result)*)
324  (declaration)* (tag statement)*
325  7.8.3 Simple Iteration Constructs
326  DOLIST (var listform [resultform]) (declaration)* (tag statement)*
327  -- Comments ------------------------------- [Macro]
328  DOTIMES (var countform [resultform]) (declaration)* (tag statement)*
329  -- Comments ------------------------------- [Macro]
330  7.8.4 Mapping
331  MAPCAR function list $rest more-lists
332  MAPLIST function list $rest more-lists
333  MAP function list $rest more-lists
334  MAP function list $rest more-lists
335  MAPCNT function list $rest more-lists
336  -- Comments ------------------------------- [Function]
337  -> These are very primitive in LISP language.
338  7.8.5 The "Program Feature"
339  TAGBODY (tag statement)*
340  -- Comments ------------------------------- [Special Form]
341  7.9 Multiple Values
342  VALUES $rest args
343  MULTIPLE-VALUE-LIST form
344  MULTIPLE-VALUE-CALL function (form)*
345  MULTIPLE-VALUE-PROG1 form (form)*
346  MULTIPLE-VALUE-SETO variables form
347  -- Comments ------------------------------- [Function]
348  -> Multiple-values.
349  MULTIPLE-VALUES-LIMIT
350  VALUES-LIST list
351  -- Comments ------------------------------- [Constant]
352  7.10 Dynamic Non-local Exits
353  CATCH tag (form)*
354  UNWIND-PROTECT protected-form (cleanup-form)*
355  THROW tag result
356  -- Comments ------------------------------- [Function]
357  -> These features are quite necessary for constructive programming.
358  8. Macro
359  8.1 Macro Definition
360  MACRO-FUNCTION symbol
361  DEFMACRO name lambda-list (declaration(doc-string)*
362  (form)*)
363  -- Comments ------------------------------- [Function]
364  -> DEFMACRO var-list keywords: &optional, &rest, &key,
365  &allow-other-keys and &aux are allowed, and &body,
366  &whole and &environment are not allowed.
367  -> &key, &allow-other-keys and &aux don't have so high necessity,
368  but it leads to understand the specification clearly to have
369  equality to DEFUN lambda-list keywords.
370  8.2 Macro Expansion
371  MACROEXPAND form
372  MACROEXPAND-1 form
373  -- Comments ------------------------------- [Macro]
374  -> MACROEXPAND,MACROEXPAND-1: optional parameter env
375  is not allowed.
376  9. Declarations
377  9.1 Declaration Syntax
378  DECLARE (decl-spec)*
379  -- Deleted items ----------------------------- [Special form]
380  LOCALLY (declaration)* (form)*
381  PROCLAIM decl-spec
382  -- Comments ------------------------------- [Function]
383  -> PROCLAIM is equivalent to 'DEFVAR', because the declaration
384  specifiers except 'special' are omitted.
385  9.2 Declaration Specifiers
386  special
387  -- Deleted items -----------------------------
TYPE, FTYPE, FUNCTION, INLINE, NOTINLINE, IGNORE, OPTIMIZE, DECLARATION

---

9.3 Type Declaration for Forms

THE value-type form

-- Comments [Special form]

\> The syntax of 'THE' must be accepted, but its function may not be interpreted.

---

10.1 The Property List

GET symbol, indicator optional default

REMPROP symbol indicator [Function]

SYMBOL-PLIST symbol [Function]

-- Deleted items [Macro]

GETF place indicator optional default

REMPF place indicator [Macro]

\> Not necessary

GET-PROPERTIES place indicator-list

\> This is not so necessary and can be implemented by SYMBOL-PLIST very easily.

10.2 The Print Name

SYMBOL-NAME sym [Function]

---

10.3 Creating Symbols

MAKE-SYMBOL print-name [Function]

COPY-SYMBOL sym optional copy-progs

GENSYM optional sym x [Function]

SYMBOL-PACKAGE sym [Function]

KEYWORPD object

-- Deleted item [Function]

GENTEMP optional prefix package

-- Comments [Special form]

\> GET and REMPROP are very important package functions on plist.

\> SYMBOL-xxxx are the primitive functions on symbols.

\> GENSYM is, of course, necessary but GENTEMP is not.

\> (Someone said, however, that GENTEMP is more useful than GENSYM.)

11. Packages

---

11.1. Consistency Rules

-- Read-read consistency

-- Print-read consistency

---

11.2. Package Names

-- #: is a separator of package name and symbol name.

---

11.3. Translating Strings to Symbols

\> :Bar is an external symbol in package KEYWORD package.

-- Bar is an internal symbol but accessible from current package.

-- Deleted Items [Function]

Symbol "package" contains current package. "Package" is deleted.

Foop::bar is a external symbol BAR that accessible from package FOO.

Foo::bar is a internal or external symbol BAR.

that accessible form package FOO.

System locally binds "package" to access and the symbol BAR.

---

11.4. Exporting and Importing Symbols

\> Function INTERN makes a symbol in current package.

-- Deleted Items [Function]

Package using and symbol importing.

-- Comment [Function]

The following two features are considered:

The exporting symbols are considered:

using packages

In exporting symbols in a package, it is necessary to searched all symbol tables to find the conflict. It causes the system inefficient, especially in personal computers. Consequently, symbol exporting and package using packages are eliminated. As a natural course of event.

Furthermore, the 'colon' notation is difficult to use correctly for beginners. For example, 'USER::CAR' represents the symbol in package USER but it does not mean the symbol locally defined in the package USER. This is because package USER uses package LISP, so the symbol is specified as the inherited one from package LISP (the above notation represents the symbol which is ACCESSIBLE from package USER by the CLOS definition). Symbol-shadowing feature is the only way to define CAR locally in the package USER.

---

11.5. Name Conflicts

-- Comment [Special form]

The symbol shadowing are deleted. because we have no other package except LISP and KEYWORD package.

---

11.6. Built-in Packages

-- Lisp system initially must have following packages.

LISP

-- Deleted Items [Function]

Next packages are deleted.

user

system

And we can't make new packages.

---

11.7. Package System Functions and Variables

INTERN string OPTIONAL package [Function]

FIND-SYMBOL string OPTIONAL package [Function]

UNINTER symbol OPTIONAL package [Function]

DO-ALL-SYMBOLS (var (result-form)) [Macro]

(declaration) [Macro]

(tag | statement) [Macro]
Function INTERNAL returns two values, the first value is the symbol itself and the second value is one of following.

1. internal ... When the symbol is already there.
2. nil ... When the symbol is newly created.
3. Function FIND-SYMBOL's second value is one of followings.
4. internal ... When the symbol is already accessible.
5. nil ... There is no symbol with that name.

Deleted items

*PACKAGE* [Variable]
MAKE-PACKAGE package-name KEY :nicknames :use
IN-PACKAGE package-name KEY :nicknames :use
FIND-PACKAGE name
PACKAGE-NAME package
PACKAGE-NICKNAMES package
RENAME-PACKAGE package new-name OPTIMAL new-nicknames
PACKAGE-USE-LIST package
PACKAGE-USED-BY-LIST package
PACKAGE-SHAOWING-SYMBOLS package
LIST-ALL-PACKAGES
EXPORT symbols OPTIMAL package
UNEXPORT symbols OPTIMAL package
IMPORT symbols OPTIMAL package
SHADOWING-IMPORT symbols OPTIMAL package
SHADOW symbols OPTIMAL package
USE-PACKAGE package-to-use OPTIMAL package
UNUSE-PACKAGE package-to-unused OPTIMAL package
FIND-ALL-SYMBOLS string-or-symbol
DO-SYMBOLS (var [package [result-form]]) [Macro]
(declaration) [tag statement]
DO-EXTERNAL-SYMBOLS (var [package [result]]) [Macro]
(declaration) [tag statement]

11.8 Modules

Deleted items

*MODULES* [Variable]
PROVIDE module-name
REQUIRE module-name OPTIMAL pathname
Comment [Function]
We think it is very poor without package creation.

12. Numbers

All features on complex numbers are omitted, since they are too inefficient on PC environment.

*12.1 Precision, Contagion, and Coercion

Same as CLtL except absence of features on complex numbers

12.2 Predicates on Numbers

ZEROP number [Function]
PUBLP number [Function]
MINUSP number [Function]
ODDP number [Function]
EVENP number [Function]

12.3 Comparisons on Numbers

number &rest more-numbers
/ \ number &rest more-numbers
\ number &rest more-numbers
/ \ number &rest more-numbers
\ number &rest more-numbers
number &rest more-numbers
number &rest more-numbers

12.4 Arithmetic Operations

+ &rest numbers [Function]
- number &rest more-numbers
* &rest numbers [Function]
/ number &rest more-numbers
1+ number [Function]
1- number [Function]
INCF place OPTIMAL delta [Macro]
DECF place OPTIMAL delta [Macro]
GCD &rest integers [Function]
LCM integer &rest more-integers [Function]

12.5 Irrational and Transcendental Functions

EXP number [Function]
EXPT base-number power-number [Function]
LOG number OPTIMAL base [Function]
SQRT number [Function]
ABS number [Function]
SIGNUM number [Function]
SIN number [Function]
COS number [Function]
TAN number [Function]
ATAN y OPTIMAL x [Function]
PI [Constant]

Comments [Function]
The function 'SQRT' must signal error, when a negative argument is passed.
The function 'LOG' must signal error, when a negative argument is passed.
Deleted items

ISEQF integer [Function]
PHASE number [Function]
ASIN number [Function]
ACOS number [Function]
SIN number [Function]
12.6 Type Conversions and Component Extractions on Numbers

- Rational numbers
- Denominator rational
- Floored number optional divisor
- Ceiling number optional divisor
- Truncated number optional divisor
- Rounding number optional divisor
- Mod number divisor
- Rem number divisor

-- Deleted items ---

- Rationalize number
- Fploor number optional divisor
- Fceiling number optional divisor
- Ftrunc number optional divisor
- Fround number optional divisor
- Decode-float
- Scale-float
- Float-radix
- Float-sign
- Float-digits
- Float-precision
- Integer-decode-float
- Complex realpart optional imagpart
- Realpart number
- Imagpart number

--> They are meaningless.

12.7 Logical Operations on Numbers

- Logior integer
- Lognor integer
- Logand integer
- Lognot integer
- Logbitp index integer
- Ash integer count

-- Deleted items ---

- Logeq
- Logand integer1 integer2
- Lognot integer1
- Logand1 integer1 integer2
- Logand2 integer1 integer2
- Logor1 integer1 integer2
- Logor2 integer1 integer2
- Boole op integer1 integer2
- Boo1
- Boo1-1
- Boo1-2
- Boo1-3
- Boo1-c1
- Boo1-c2
- Boo1-and
- Boo1-ior
- Boo1-xor
- Boo1-eqv
- Boo1-nand
- Boo1-nor
- Boo1-ocl
- Boo1-and1
- Boo1-and2
- Boo1-or1
- Boo1-or2
- Logeq integer1 integer2
- Logcount integer
- Integer-length integer

12.8 Byte Manipulation Functions

- Byte size position
- Byte-size bytespec
- Byte-position bytespec
- Ldb bytespec integer
- Lds-byte bytespec integer
- Mask-field bytespec integer
- Dpb newbyte bytespec integer
- Deposit-field newbyte bytespec integer

12.9 Random Numbers

- Random state

-- Deleted items ---

- Random state
- Make-random-state optional state
- Random-state-p object

12.10 Implementation Parameters

- All implementation parameters on number are omitted to keep the system compact.

-- Deleted items ---

- Most-positive-fixnum
- Most-negative-fixnum
- Most-positive-short-float
- Least-positive-short-float
- Least-negative-short-float
13 Character

-- Comments
-> The CL Core's character doesn't have the font and bits attributes.
-> If necessary, the CL Core consider the font and bits attributes

as zero.

13.1 Character Attributes

-- Deleted Items

CHAR-CODE-LIMIT  [Constant]
CHAR-POINT-LIMIT  [Constant]
CHAR-BITS-LIMIT  (Constant)

-> System constants should be omitted.

13.2 Predicates on Characters

STANDARD-CHAR-P char  [Function]
GRAPHIC-CHAR-P char  [Function]
STRING-CHAR-P char  [Function]
ALPHA-CHAR-P char  [Function]
UPPER-CASE-P char  [Function]
LOWER-CASE-P char  [Function]
BOTH-CHAR-P char  [Function]
DIGIT-CHAR-P char optional (radix 10)  [Function]
CHARP character &rest more-characters  [Function]
CHAR= character &rest more-characters  [Function]
CHAR character &rest more-characters  [Function]
CHARP character &rest more-characters  [Function]
CHARP character &rest more-characters  [Function]
-> These are primitive predicates for treating characters.
CHAR-EQUAL character &rest more-characters  [Function]
CHAR-NOT-EQUAL character &rest more-characters  [Function]
CHAR-LESSP character &rest more-characters  [Function]
CHAR-GREATERTP character &rest more-characters  [Function]
CHAR-NOT-GREATERTP character &rest more-characters  [Function]
CHAR-NOT-LESSP character &rest more-characters  [Function]

-- Deleted Items

ALPHANUMERIC-P char  [Function]
-> ALPHANUMERIC-P is equal to (OR (ALPHA-CHAR-P char) (DIGIT-CHAR-P char))

13.3 Character Construction and Selection

CHAR-CODE char  [Function]
CODE-CHAR code  [Function]

-- Comment
-> CODE-CHAR's optional parameter (font, bits) is omitted.

-- Deleted Items

CHAR-BITS char  [Function]
CHAR-POINT char  [Function]
MAKE-CHAR char optional (bits 0) (font 0)  [Function]
-> When the font and bits attributes are zero, MAKE-CHAR returns char

13.4 Character Conversions

CHARACTER object  [Function]
CHAR-UPCASE char  [Function]
CHAR-LOWERCASE char  [Function]
DIGIT-CHAR weight optional (radix 10)  [Function]
CHAR-NAME char  [Function]
NAME-CHAR char  [Function]

-- Comment
-> DIGIT-CHAR's optional font parameter is omitted.

-- Deleted Items

CHAR-INT char  [Function]

CHAR-INT integer  [Function]
-> CHAR-INT returns the same integer CHAR-CODE would, because
the font and bits attributes are zero.
-> And thus INT-CHAR is equal to CODE-CHAR.

13.5 Character Control-Bits

-> Deleted Items

CHAR-CONTROL-BIT  [Constant]
CHAR-META-BIT  [Constant]
CHAR-SUPER-BIT  [Constant]
CHAR-HYPHEN-BIT  [Constant]
CHAR-BIT char name  [Function]
SET-CHAR-BIT char name new-value  [Function]
-> Because the font and bits attributes are not implemented.
all character control-bit functions are omitted.

14 Sequences
14.1 Simple Sequence Function

14.2 Concatenating, Mapping, and Reducing Sequences

14.3 Modifying Sequences

14.4 Searching Sequences for Items

14.5 Sorting and Merging

15. Lists
subset.long

901  CDAAR list [Function]
902  CDAAR list [Function]
903  CDAAR list [Function]
904  CDDDR list [Function]
905  CONS y [Function]
906  --- Deleted items ---------------------------------------------
907  CAAARR list [Function]
908  CAAARR list [Function]
909  CAADAR list [Function]
910  CAADAR list [Function]
911  CAADAR list [Function]
912  CADADD list [Function]
913  CADARR list [Function]
914  CADARR list [Function]
915  CDAAAR list [Function]
916  CDAAAR list [Function]
917  CDADAR list [Function]
918  CDADAR list [Function]
919  CDAAAR list [Function]
920  CDADD list [Function]
921  CDADD list [Function]
922  CDDDR list [Function]
923  TREE-EQUAL x y [Function]
924  REST list [Function]
925  NTHCAR n list [Function]
926  LAST list [Function]
927  LIST &rest args [Function]
928  LIST &rest others [Function]
929  APPEND &rest lists [Function]
930  COPY-TREE object [Function]
931  WARNCONC &rest lists [Function]
932  PUSH item place [Function]
933  POP place [Function]
934  LDIFF list sublist [Function]
935  MARK-LIST size &key :initial-element [Function]
936  COPY-LIST list [Function]
937  COPY-ALIST list [Function]
938  REAPPEND x y [Function]
939  NRCONC x y [Function]
940  PUSHNEW item place &key :test :test-not :key [Function]
941  BUTLAST list &optional n [Function]
942  BUTLAST list &optional n [Function]
943  REPLCA x y [Function]
944  REPLCD x y [Function]
945  ---\-Subst, NSUBST: lambda keyword :test-not and :key are not allowed.
946  )-IF, IF-IFNOT and IF-TEST-not. :key are deleted as Sequences.
947  )-Comments -------------------------------------------------
948  ---\-Deleting new old tree &key :test
949  SUBST new old tree &key :test [Function]
950  SUBST new old tree &key :test [Function]
951  SUBST-IF new old tree &key :test [Function]
952  SUBST-IF-NOT new old tree &key :test [Function]
953  -Comments -------------------------------------------------
954  )-Using Lists as Sets
955  MEMBER item list &key :test [Function]
956  ADJOIN item list [Function]
957  UNION list1 list2 &key :test [Function]
958  MUNION list1 list2 &key :test [Function]
959  INTERSECTION list1 list2 &key :test [Function]
960  UNION-DIFFERENCE list1 list2 &key :test [Function]
961  INTERSECTION list1 list2 &key :test [Function]
962  SET-DIFFERENCE list1 list2 &key :test [Function]
963  SET-DIFFERENCE list1 list2 &key :test [Function]
964  SET-DIFFERENCE list1 list2 &key :test [Function]
965  SET-DIFFERENCE list1 list2 &key :test [Function]
966  SET-EXCLUSIVE-OR list1 list2 &key :test [Function]
967  SET-EXCLUSIVE-OR list1 list2 &key :test [Function]
968  SET-EXCLUSIVE-OR list1 list2 &key :test [Function]
969  SET-EXCLUSIVE-OR list1 list2 &key :test [Function]
970  SET-EXCLUSIVE-OR list1 list2 &key :test [Function]
971  SET-EXCLUSIVE-OR list1 list2 &key :test [Function]
972  )-Comments -------------------------------------------------
973  )-Using Lists as Sets non-destructively are very useful and using
974  lists as sets destructively are useful and primitive.
15.6 Association lists

ACONS key datum a-list
ASSOC item a-list $key$ test
RASSOC item a-list $key$ test

-- Deleted items
PAIRLIST keys data Optional a-list
ASSOC-IF item a-list $key$ test
RASSOC-IF item a-list $key$ test
ASSOC-IF-NOT item a-list $key$ test
RASSOC-IF-NOT item a-list $key$ test

-- Comment
-\rightarrow ASSOC,RASSOC: lambda keyword :test=not and $key$ are
not allowed.
-\rightarrow association lists don't have so many necessities now.

15 Hash Tables

-- Deleted items
MAKE-HASH-TABLE $key$ :test :size :rehash-size

-- Comments
-\rightarrow the hash table is too complicated. and the necessity is low.

17. Arrays

Only simple arrays are available. Bit-arrays are omitted.

-- Comments
Simple arrays are enough on PCs.
-\rightarrow Bit-arrays are not necessary in usual applications.

17.1 Array Creation
MAKE-ARRAY dimensions $key$ :initial-element

-- Deleted items
:element-type :adjustable :fill-pointer
:displaced-to :displaced-index-offset

-- Restricted
The maximum array rank available can be 3 (not 7).

-- Comments
-\rightarrow Element-type "t" is enough on PCs.
-\rightarrow Restricted array element feature is for efficiency and is useful
with foreign languages. These extensions should be left for vendors.
-\rightarrow Adjustables are complexed feature and is not necessary on PCs.
-\rightarrow Fill-pointer may be useful but is not an essential feature on vectors.
-\rightarrow VECTOR vector objects
-\rightarrow VECTOR helps users write clearer code using sequences though
-\rightarrow it can be substituted by MAKE-ARRAY.

-- Deleted items

17.2 Array Access
AREF array vector subscript
SVREF simple-vector index
-\rightarrow SVREF helps users write clearer code accessing vectors though
-\rightarrow it can be substituted by AREF.

17.3 Array Information
ARRAY-RANK array
ARRAY-DIMENSION array axis-number
ARRAY-IN-BOUNDS-P array vector subscript

-- Comments
-\rightarrow These are the very primitive functions on arrays.

-- Deleted items

17.4 Functions on Arrays of Bits

-- Deleted items

17.5 Fill Pointers

-- Deleted items
1101 ARRAY-HAS-FILL-POINTER-P array
1102 FILL-POINTER vector
1103 VECTOR-PUSH new-element vector
1104 VECTOR-PUSH-EXTEND new-element vector &optional extension
1105 VECTOR-POP vector
1106 => Only simple-arrays are available.
1107 -- Comments -------------------------------
1108 => Fill-pointer might be useful in some applications
1109 but seems to be unnatural feature on the vector.
1110 17.6 Changing the Dimensions of an Array
1111 -- Deleted item -----------------------------
1112 ADJUST-ARRAY array new-dimensions &key
1113 [:element-type]
1114 [:initial-element]
1115 [:initial-contents]
1116 [:fill-pointer]
1117 [:displaced-to]
1118 :[:displaced-index-offset]
1119 => Only simple-arrays are available.
1120 17.6 Changing the Dimensions of an Array
1121 -- Deleted item -----------------------------
1122 ADJUST-ARRAY array new-dimensions &key
1123 [:element-type]
1124 [:initial-element]
1125 -------------------------------
1126 18. Strings
1127 -- Comment ---
1128 Strings' partial manipulation is handled by functions that treat sequence
1129 data type like SUBSEQ.
1130 18.1 String Access
1131 CHAR string index
1132 SCHAR simple-string index
1133 [Function]
1134 [Function]
1135 -- Comment -------------------------------
1136 There is a recommend of these two functions. Use SCHAR
1137 because of transporting program to Common Lisp full set. SCHAR is faster
1138 than CHAR.
1139 18.2. String Comparison
1140 STRING= string1 string2 &KEY :start1 :end1 :start2 :end2
1141 STRING-EQUAL string1 string2 &KEY :start1 :end1 :start2 :end2
1142 STRING> string1 string2 &KEY :start1 :end1 :start2 :end2
1143 STRING< string1 string2 &KEY :start1 :end1 :start2 :end2
1144 STRING>= string1 string2 &KEY :start1 :end1 :start2 :end2
1145 STRING<= string1 string2 &KEY :start1 :end1 :start2 :end2
1146 STRING-LESSP string1 string2 &KEY :start1 :end1 :start2 :end2
1147 STRING-GREATERTP string1 string2 &KEY :start1 :end1 :start2 :end2
1148 STRING-NOT-GREATERTP string1 string2 &KEY :start1 :end1 :start2 :end2
1149 STRING-NOT-LESSP string1 string2 &KEY :start1 :end1 :start2 :end2
1150 STRING-NOT-EQUAL string1 string2 &KEY :start1 :end1 :start2 :end2
1151 STRING N string1 string2 &KEY :start1 :end1 :start2 :end2
1152 STRING comparisons remain with character comparisons.
1153 -- Comment -------------------------------
1154 We need MAKE-STRING to prepare the buffer of large string. For
1155 example, prepare a buffer for screen of screen editor.
1156 -- Restrictions -----------------------------
1157 Cut keyword parameters :START and :END.
1158 Usually case conversion function will be applied to whole string.
1159 Please use SUBSEQ to modify the part of string.
1160 19. Structures
1161 DEFSTRUCT name [doc-string] (slot-description)+
1162 Legal syntax for the slot-descriptions:
1163 (slot-name [default-init])
1164 or
1165 -- Deleted Items -----------------------------
1166 -- slot options:
1167 :type
1168 :read-only
1169 :definition-options:
1170 :conc-name
1171 :constructor
1172 :coercer
1173 :predicate
1174 :include
1175 :print-function
1176 :type
1177 :named
subset-long

20 The Evaluator
20.1 Run-Time Evaluation of Forms
   EVAL form
   \( \rightarrow \) EVAL is a very important primitive function.
   \( \text{CONSTANTP} \) object
   \( \rightarrow \) \text{CONSTANTP} is the only function to judge if object is constant.

-- Deleted Items --
   *EVALHOOK*
   **APPLYHOOK**

-- Top-Level Loop --
   *
   *
   *
   /\ At least these are necessary for standard user interaction.

-- Deleted Items --
   ***
   ***
   ///

21 Streams
21.1 Standard Streams
   *standard-input*
   *standard-output*
   *error-output*
   *query-lo*
   *debug-lo*
   *terminal-lo*
   *trace-output*

21.2 Creating New Streams

-- Deleted Items --
   MAKE-SYNONYM-STREAM
   MAKE-BROADCAST-STREAM
   MAKE-CONCATENATED-STREAM
   MAKE-TWO-WAY-STREAM
   MAKE-ECHO-STREAM
   MAKE-STRING-INPUT-STREAM
   MAKE-STRING-OUTPUT-STREAM
   MAKE-STRING-INPUT-STREAM
   MAKE-STRING-OUTPUT-STREAM
   MAKE-OUTPUT-TO-STREAM
   MAKE-INPUT-TO-STRING

-- Comments --
   It is very useful and important to abstract I/O.

21.3 Operations on Streams
   STREAM object
   INPUT-STREAM-P
   OUTPUT-STREAM-P
   STREAM-ELEMENT-TYPE stream

-- Restricted --
   Keyword parameter, :abort is omitted from CLOSE.

CLOSE stream

22. Input/Output
22.1 Printed Representation of Lisp Objects

-- Deleted Items --
   *READABLE*
   *PRINT-LEVEL*
   *PRINT-CASE*
   ***

-- Standard dispatch macro character syntax

-- Deleted Items --
   ***
22.2 Input Functions

READ optional input-stream eof-error-p eof-value

READ-LINE optional input-stream eof-error-p eof-value

READ-CHAR optional input-stream eof-error-p eof-value

UNREAD-CHAR character optional input-stream

LISKEN optional input-stream

READ-CHAR-NO-HANG optional in-stream eof-error-p

eof-value recursive-p

CLEAR-INPUT optional input-stream

READ-STRING string optional eof-error-p eof-value

READ-BYTE binary-input-stream optional eof-error-p

-- Deleted items

READ-DEFAULT-FLOAT-FORMAT

READ-PRESERVING-WHITESPACE optional in-stream eof-error-p

eof-value recursive-p

READ-DELIMITED-LIST char optional input-stream

PEEK-CHAR optional peek-type input-stream eof-error-p

eof-value recursive-p

PARSE-INTEGER string key :start :end :radix :junk-allowed

-- Comments

\( \rightarrow \) READ-FROM-STRING keyword parameter :preserve-white

\( \rightarrow \) string i/o is useful for the lisp internal editor and binary i/o

\( \rightarrow \) is necessary for handling alien structures produced by other

language.

22.3 Output Functions

PRINT object optional output-stream

PRINT object optional output-stream

PPRINT object optional output-stream

PRIN1 object optional output-stream

PRIN1-TO-STRING object

WRITE-CHAR object optional output-stream

TERPRI optional output-stream

FRESH-LINE optional output-stream

WRITE-BYTE integer binary-output-stream

FORMAT destination control-string arg arguments

format-directives

\( \rightarrow \) A S 0 A " X " C " P " G " A " A " \) -- <NEWLINE > T

WRITE object key :stream :escape :radix :base :circle


WRITE-TO-STRING object key :escape :radix :base :circle


WRITE-STRING string optional output-stream

key :start :end

WRITE-LINE string optional output-stream

key :start :end

FINISH-OUTPUT optional output-stream

FORC-OUTPUT optional output-stream

CLEAR-OUTPUT optional output-stream

format directives

\( \rightarrow \) \( \rightarrow \) ( " " ) ( " " ) ( " " )

-- Comments

\( \rightarrow \) PRN1, PPRINT and PRINC are more friendly than

generic WRITE.

22.4 Querying the Users

-- Deleted items

Y-OR-N-P optional format-string arg arguments

YES-OR-NO-P optional format-string arg arguments

23 File System Interface

23.1 File Names

File names can be expressed as strings.

23.1.2 Pathname Functions

-- Deleted items

TRUENAME pathname

PARSE-NAME-STRING thing optional host defaults

MERGE-PATHNAMES pathname optional default arg arg

MAKE-PATHNAME key :host :device :directory :name

PATTERNMP object

PATHNAME-HOST pathname
23.2 Opening and Closing Files

open filename key direction :element-type

--- Deleted items ------------------- [Declaration] (form)* [Macro]

--- Deleted items ------------------- [Function]

23.3 Renaming, Deleting, and Other File Operations

rename-file file new-name
delete-file file
probe-file file
file-write-date file
file-position file
file-length file
file-length file-stream
file-length file
delete-item
--- Deleted item ------------------- [Function]

23.4 Loading Files

load filename &key :verbose :print

--- Deleted items ------------------- [Function]

--- Keyword parameter ------------------ [Variable]

23.5 Accessing Directories

DIRECTORY filename &key

--- Comments -------------------------- [Function]

24. Errors

24.1 General Error-Signalling Function

ERROR formal-string gerror args
WARN format-string gerror args
BREAK optional format-string gerror args
--- Deleted items ------------------- [Function]

24.2 Specialized Error-Signalling Forms and Macros

--- Deleted items ------------------- [Macro]

CHECK-TYPE place typespec optional string

--- These are not so necessary.

ETYPENCASE keyform ((type (form)))*
SYSTEMCASE keyplace ((type (form)))*
--- These are not so necessary and can be implemented very easily

using ETYPENCASE.

ECASE keyform (((key)* | key) (form))*
CCASE keyform (((((key)* | key) (form))*
--- Same reason as ECTYPENCASE

ASSERT test-form (((place)* | string (arg)*)
--- These are not so necessary.

ETYPENCASE keyform (((type (form)))*

CASE keyplace (((type (form)))*
--- Macro

25. Miscellaneous Features
25.3. Debugging Tools

TRACE (function-name)*  [Macro]
UNTRACE (function-name)*  [Macro]
STEP form  [Macro]
TIME form  [Function]
DESCRIBE object  [Function]
ED &OPTIONAL x  [Function]
APPROPOS string &OPTIONAL package  [Function]
APPROPOS-LIST string &OPTIONAL package  [Function]

-- Comment --
We think the common Lisp is a system. So implementor have to deliver
these debugging tools with Common Lisp.
Following functions may be defined.
GC ... Performs garbage collection.
EXIT ... Exit from lisp system if it can.

-- Deleted Items --
INSPECT object  [Function]
ROOM &OPTIONAL x  [Function]
DRIBBLE &OPTIONAL pathname  [Function]

We think these functions are not so important with ordinary debugging.

25.4. Environment Inquiries

25.4.1. Time Functions
GET-DECODED-TIME  [Function]
INTERNAL-TIME-UNITS-FOR-SECOND  [Constant]
GET-INTERNAL-RUN-TIME  [Function]
GET-INTERNAL-REAL-TIME  [Function]
SLEEP seconds  [Function]

-- Comment --
RUN-TIME and REAL-TIME may be equal on some kind of the system that can't
calculate CPU time only.

-- Deleted Items --
GET-UNIVERSAL-TIME  [Function]
DECODE-UNIVERSAL-TIME universal-time &OPTIONAL time-zone  [Function]
ENCODE-UNIVERSAL-TIME second minute hour date month year  [Function]

-- Comment --
Function GET-DECODED-TIME will be enough. So the concept of
universal time is deleted.

25.4.2. Other Environment Inquiries
LISP-IMPLEMENTATION-TYPE  [Function]
LISP-IMPLEMENTATION-VERSION  [Function]
SHORT-SITE-NAME  [Function]
LONG-SITE-NAME  [Function]

*FEATURES*  [Variable]

-- Comment --
If Lisp read the return value of LISP-IMPLEMENTATION-VERSION, we want
read return number.

-- Deleted Items --
MACHINE-TYPE  [Function]
MACHINE-VERSION  [Function]
MACHINE-INSTANCE  [Function]
SOFTWARE-TYPE  [Function]
SOFTWARE-VERSION  [Function]

Function LISP-IMPLEMENTATION-TYPE and LISP-IMPLEMENTATION-VERSION
will be enough.

25.5. Identity Function

IDENTITY object  [Function]

-- Comment --
This function is the default value of :KEY keyword parameter.