*File created:* 14-Jun-2024 23:09:54 **{DSK}<home>matt>Interlisp>medley>sources>XCL-LOOP.;4**

*edit by:* mth

*changes to:* (IL:FUNCTIONS DEFAULT-TYPE DEFAULT-VALUE)

*previous date:* 8-Apr-2024 19:38:27 {DSK}<home>matt>Interlisp>medley>sources>XCL-LOOP.;2

*Read Table:* XCL

*Package:* LOOP

*Format:* XCCS

```
(IL:RPAQQ IL:XCL-LOOPCOMS
          ((FILE-ENVIRONMENTS IL:LOOP)
           (IL:STRUCTURES SIMPLE-PROGRAM-ERROR)
           (IL:VARIABLES *ACCUMULATORS* *ANONYMOUS-ACCUMULATOR* *BOOLEAN-TERMINATOR* *CURRENT-CLAUSE*
                  *CURRENT-KEYWORD* *ENVIRONMENT* *FOR-AS-COMPONENTS* *FOR-AS-SUBCLAUSES* *HASH-GROUP*
                  *FOR-AS-PREPOSITIONS* *IGNORABLE* *IT-SYMBOL* *IT-VISIBLE-P* *LIST-END-TEST* *LOOP-CLAUSES*
                  *LOOP-COMPONENTS* *LOOP-NAME* *LOOP-TOKENS* *MESSAGE-PREFIX* *SYMBOL-GROUP* *TEMPORARIES*)
           (IL:FUNCTIONS %KEYWORD %LIST ACCUMULATE-IN-LIST ACCUMULATION-CLAUSE ACCUMULATOR-KIND ACCUMULATOR-SPEC
                  ALONG-WITH ALWAYS-NEVER-THEREIS-CLAUSE AMBIGUOUS-LOOP-RESULT-ERROR APPEND-CONTEXT APPENDF
                  BINDINGS BOUND-VARIABLES BY-STEP-FUN CAR-TYPE CDR-TYPE CHECK-MULTIPLE-BINDINGS CL-EXTERNAL-P
                  CLAUSE* CLAUSE1 COMPOUND-FORMS* COMPOUND-FORMS+ CONDITIONAL-CLAUSE CONSTANT-BINDINGS
                  CONSTANT-FUNCTION-P CONSTANT-VECTOR CONSTANT-VECTOR-P D-VAR-SPEC-P D-VAR-SPEC1 D-VAR-TYPE-SPEC
                  DECLARATIONS DEFAULT-BINDING DEFAULT-BINDINGS DEFAULT-TYPE DEFAULT-VALUE
                  DESTRUCTURING-MULTIPLE-VALUE-BIND DESTRUCTURING-MULTIPLE-VALUE-SETQ DISPATCH-FOR-AS-SUBCLAUSE
                  DO-CLAUSE EMPTY-P ENUMERATE EXTENDED-LOOP FILL-IN FINALLY-CLAUSE FOR FOR-AS-ACROSS-SUBCLAUSE
                  FOR-AS-ARITHMETIC-POSSIBLE-PREPOSITIONS FOR-AS-ARITHMETIC-STEP-AND-TEST-FUNCTIONS
                  FOR-AS-ARITHMETIC-SUBCLAUSE FOR-AS-BEING-SUBCLAUSE FOR-AS-CLAUSE FOR-AS-EQUALS-THEN-SUBCLAUSE
                  FOR-AS-FILL-IN FOR-AS-HASH-SUBCLAUSE FOR-AS-IN-LIST-SUBCLAUSE FOR-AS-ON-LIST-SUBCLAUSE
                  FOR-AS-PACKAGE-SUBCLAUSE FOR-AS-PARALLEL-P FORM-OR-IT FORM1 GENSYM-IGNORABLE
                  GLOBALLY-SPECIAL-P HASH-D-VAR-SPEC INITIALLY-CLAUSE INVALID-ACCUMULATOR-COMBINATION-ERROR
                  KEYWORD1 KEYWORD? LET-FORM LOOP-ERROR LOOP-FINISH-TEST-FORMS LOOP-WARN LP MAIN-CLAUSE*
                  MAPAPPEND MULTIPLE-VALUE-LIST-ARGUMENT-FORM MULTIPLE-VALUE-LIST-FORM-P NAME-CLAUSE? ONE
                  ORDINARY-BINDINGS PREPOSITION1 PREPOSITION? PSETQ-FORMS QUOTED-FORM-P QUOTED-OBJECT
                  REDUCE-REDUNDANT-CODE REPEAT-CLAUSE RETURN-CLAUSE SELECTABLE-CLAUSE SIMPLE-LOOP SIMPLE-VAR-P
                  SIMPLE-VAR1 STRAY-OF-TYPE-ERROR CL::SYMBOL-MACROLET TYPE-SPEC? UNTIL-CLAUSE USING-OTHER-VAR
                  VARIABLE-CLAUSE* WHILE-CLAUSE WITH WITH-ACCUMULATORS WITH-BINDING-FORMS WITH-CLAUSE
                  WITH-ITERATOR-FORMS WITH-LIST-ACCUMULATOR WITH-LOOP-CONTEXT WITH-NUMERIC-ACCUMULATOR
                  WITH-TEMPORARIES ZERO)
           (IL:FUNCTIONS LOOP)
           (IL:PROP (IL:FILETYPE IL:MAKEFILE-ENVIRONMENT IL:COPYRIGHT IL:LICENSE)
                  IL:XCL-LOOP)))

(DEFINE-FILE-ENVIRONMENT IL:LOOP :PACKAGE (DEFPACKAGE "LOOP" (:USE "LISP" "XCL"))
   :READTABLE "XCL")

(DEFINE-CONDITION SIMPLE-PROGRAM-ERROR (SIMPLE-CONDITION PROGRAM-ERROR)
   NIL)

(DEFVAR *ACCUMULATORS* NIL)

(DEFVAR *ANONYMOUS-ACCUMULATOR* NIL)

(DEFVAR *BOOLEAN-TERMINATOR* NIL)

(DEFVAR *CURRENT-CLAUSE* NIL)

(DEFVAR *CURRENT-KEYWORD* NIL)

(DEFVAR *ENVIRONMENT*)

(DEFVAR *FOR-AS-COMPONENTS*)

(DEFVAR *FOR-AS-SUBCLAUSES*
   (LET ((TABLE (MAKE-HASH-TABLE)))
        (MAPC #'(LAMBDA (SPEC)
                        (DESTRUCTURING-BIND (SUBCLAUSE-NAME . KEYWORDS)
                               SPEC
                               (DOLIST (KEY KEYWORDS)
                                   (SETF (GETHASH KEY TABLE)
                                         SUBCLAUSE-NAME))))
              '((FOR-AS-ARITHMETIC-SUBCLAUSE :FROM :DOWNFROM :UPFROM :TO :DOWNTO :UPTO :BELOW :ABOVE :BY)
                (FOR-AS-IN-LIST-SUBCLAUSE :IN)
                (FOR-AS-ON-LIST-SUBCLAUSE :ON)
                (FOR-AS-EQUALS-THEN-SUBCLAUSE :=)
```

```
                (FOR-AS-ACROSS-SUBCLAUSE :ACROSS)
                (FOR-AS-BEING-SUBCLAUSE :BEING)))
        TABLE)
   "A table mapping for-as prepositions to their processor function-designator.")


(DEFVAR *HASH-GROUP* '(:HASH-KEY :HASH-KEYS :HASH-VALUE :HASH-VALUES))


(DEFVAR *FOR-AS-PREPOSITIONS*
   (LET ((PREPOSITIONS NIL))
        (MAPHASH #'(LAMBDA (KEY VALUE)
                           (DECLARE (IGNORE VALUE))
                           (PUSH KEY PREPOSITIONS))
                   *FOR-AS-SUBCLAUSES*)
        PREPOSITIONS))


(DEFVAR *IGNORABLE* NIL
   "Ignorable temporary variables in *temporaries*.")


(DEFVAR *IT-SYMBOL* NIL)


(DEFVAR *IT-VISIBLE-P* NIL)


(DEFVAR *LIST-END-TEST* 'ATOM)


(DEFVAR *LOOP-CLAUSES*
   (LET ((TABLE (MAKE-HASH-TABLE)))
        (MAPC #'(LAMBDA (SPEC)
                        (DESTRUCTURING-BIND (CLAUSE-NAME . KEYWORDS)
                               SPEC
                               (DOLIST (KEY KEYWORDS)
                                   (SETF (GETHASH KEY TABLE)
                                         CLAUSE-NAME))))
              '((FOR-AS-CLAUSE :FOR :AS)
                (WITH-CLAUSE :WITH)
                (DO-CLAUSE :DO :DOING)
                (RETURN-CLAUSE :RETURN)
                (INITIALLY-CLAUSE :INITIALLY)
                (FINALLY-CLAUSE :FINALLY)
                (ACCUMULATION-CLAUSE :COLLECT :COLLECTING :APPEND :APPENDING :NCONC :NCONCING :COUNT :COUNTING
                       :SUM :SUMMING :MAXIMIZE :MAXIMIZING :MINIMIZE :MINIMIZING)
                (CONDITIONAL-CLAUSE :IF :WHEN :UNLESS)
                (REPEAT-CLAUSE :REPEAT)
                (ALWAYS-NEVER-THEREIS-CLAUSE :ALWAYS :NEVER :THEREIS)
                (WHILE-CLAUSE :WHILE)
                (UNTIL-CLAUSE :UNTIL)))
        TABLE)
   "A table mapping loop keywords to their processor function-designator.")


(DEFVAR *LOOP-COMPONENTS* NIL)


(DEFVAR *LOOP-NAME* NIL)


(DEFVAR *LOOP-TOKENS*)


(DEFVAR *MESSAGE-PREFIX* "")


(DEFVAR *SYMBOL-GROUP* '(:SYMBOL :SYMBOLS :PRESENT-SYMBOL :PRESENT-SYMBOLS :EXTERNAL-SYMBOL
                                 :EXTERNAL-SYMBOLS))


(DEFVAR *TEMPORARIES* NIL
   "Temporary variables used in with-clauses and for-as-clauses.")


(DEFUN %KEYWORD (DESIGNATOR)
   (INTERN (STRING DESIGNATOR)
          "KEYWORD"))


(DEFUN %LIST (DESIGNATOR)                                    ; Edited 14-Mar-2024 11:46 by lmm
   (IF (LISTP DESIGNATOR)
       DESIGNATOR
       (LIST DESIGNATOR)))
```

```
(DEFUN ACCUMULATE-IN-LIST (FORM ACCUMULATOR-SPEC)
   (DESTRUCTURING-BIND (NAME &KEY VAR SPLICE &ALLOW-OTHER-KEYS)
          ACCUMULATOR-SPEC
          (DECLARE (IGNORE NAME))
          (LET* ((COPY-F (ECASE *CURRENT-KEYWORD*
                             ((:COLLECT :COLLECTING) 'LIST)
                             ((:APPEND :APPENDING) 'COPY-LIST)
                             ((:NCONC :NCONCING) 'IDENTITY)))
                 (COLLECTING-P (MEMBER *CURRENT-KEYWORD* '(:COLLECT :COLLECTING)))
                 (LAST-F (IF COLLECTING-P
                             'CDR
                             'LAST))
                 (SPLICING-FORM (IF COLLECTING-P
                                    `(RPLACD ,SPLICE (SETQ ,SPLICE (LIST ,FORM)))
                                    `(SETF (CDR ,SPLICE)
                                           (,COPY-F ,FORM)
                                           ,SPLICE
                                           (,LAST-F ,SPLICE)))))
                (IF (GLOBALLY-SPECIAL-P VAR)
                    (LP :DO `(IF ,SPLICE
                                 ,SPLICING-FORM
                                 (SETQ ,SPLICE (,LAST-F (SETQ ,VAR (,COPY-F ,FORM))))))
                    (LP :DO SPLICING-FORM)))))


(DEFUN ACCUMULATION-CLAUSE ()
   (LET* ((FORM (FORM-OR-IT))
          (NAME (IF (PREPOSITION? :INTO)
                    (SIMPLE-VAR1)
                    (PROGN (SETQ *ANONYMOUS-ACCUMULATOR* *CURRENT-KEYWORD*)
                           (WHEN *BOOLEAN-TERMINATOR* (AMBIGUOUS-LOOP-RESULT-ERROR))
                           NIL)))
          (ACCUMULATOR-SPEC (ACCUMULATOR-SPEC NAME)))
         (DESTRUCTURING-BIND (NAME &REST PLIST &KEY VAR &ALLOW-OTHER-KEYS)
                ACCUMULATOR-SPEC
                (DECLARE (IGNORE NAME))
                (ECASE *CURRENT-KEYWORD*
                    ((:COLLECT :COLLECTING :APPEND :APPENDING :NCONC :NCONCING) (ACCUMULATE-IN-LIST FORM
                                                                                                    ACCUMULATOR-SPEC))
                    ((:COUNT :COUNTING) (LP :IF FORM :DO `(INCF ,VAR)))
                    ((:SUM :SUMMING) (LP :DO `(INCF ,VAR ,FORM)))
                    ((:MAXIMIZE :MAXIMIZING :MINIMIZE :MINIMIZING)
                       (LET ((FIRST-P (GETF PLIST :FIRST-P))
                             (FUN (IF (MEMBER *CURRENT-KEYWORD* '(:MAXIMIZE :MAXIMIZING))
                                      '<
                                      '>)))
                            (LP :DO `(LET ((VALUE ,FORM))
                                          (COND
                                             (,FIRST-P (SETQ ,FIRST-P NIL ,VAR VALUE))
                                             ((,FUN ,VAR VALUE)
                                              (SETQ ,VAR VALUE)))))))))))


(DEFUN ACCUMULATOR-KIND (KEY)
   (ECASE KEY
       ((:COLLECT :COLLECTING :APPEND :APPENDING :NCONC :NCONCING) :LIST)
       ((:SUM :SUMMING :COUNT :COUNTING) :TOTAL)
       ((:MAXIMIZE :MAXIMIZING :MINIMIZE :MINIMIZING) :LIMIT)))


(DEFUN ACCUMULATOR-SPEC (NAME)
   (LET* ((KIND (ACCUMULATOR-KIND *CURRENT-KEYWORD*))
          (SPEC (ASSOC NAME *ACCUMULATORS*))
          (PLIST (CDR SPEC)))
         (IF SPEC
             (IF (NOT (EQ KIND (GETF PLIST :KIND)))
                 (INVALID-ACCUMULATOR-COMBINATION-ERROR (REVERSE (GETF PLIST :KEYS)))
                 (PROGN (PUSHNEW *CURRENT-KEYWORD* (GETF PLIST :KEYS))
                        (WHEN (MEMBER KIND '(:TOTAL :LIMIT))
                            (MULTIPLE-VALUE-BIND (TYPE SUPPLIED-P)
                                (TYPE-SPEC?)
                              (WHEN SUPPLIED-P
                                 (PUSH TYPE (GETF PLIST :TYPES)))))))
             (LET ((VAR (OR NAME (GENSYM "ACCUMULATOR-"))))
                  (SETQ PLIST `(:VAR ,VAR :KIND ,KIND :KEYS (,*CURRENT-KEYWORD*)))
                  (ECASE KIND
                      (:LIST
                         (SETF (GETF PLIST :SPLICE)
                               (GENSYM "SPLICE-"))
                         (UNLESS NAME
                             (FILL-IN :RESULTS `((CDR ,VAR)))))
                      ((:TOTAL :LIMIT)
                         (MULTIPLE-VALUE-BIND (TYPE SUPPLIED-P)
                             (TYPE-SPEC?)
                           (WHEN SUPPLIED-P
                              (PUSH TYPE (GETF PLIST :TYPES))))
                         (WHEN (EQ KIND :LIMIT)
```

```
                              (LET ((FIRST-P (GENSYM "FIRST-P-")))
                                   (SETF (GETF PLIST :FIRST-P)
                                         FIRST-P)
                                   (WITH FIRST-P T := T)))
                             (UNLESS NAME
                                 (FILL-IN :RESULTS `(,VAR)))))
                       (PUSH (SETQ SPEC `(,NAME ,@PLIST))
                             *ACCUMULATORS*)))
          SPEC))

(DEFUN ALONG-WITH (VAR TYPE &KEY EQUALS (THEN EQUALS))
   (FOR-AS-FILL-IN :BINDINGS (APPLY #'BINDINGS TYPE VAR (WHEN (QUOTED-FORM-P EQUALS)
                                                              `(,EQUALS))))
   (UNLESS (QUOTED-FORM-P EQUALS)
       (FOR-AS-FILL-IN :AFTER-HEAD `((SETQ ,@(MAPAPPEND #'CDR (BINDINGS TYPE VAR EQUALS))))))
   (FOR-AS-FILL-IN :AFTER-TAIL `((SETQ ,@(MAPAPPEND #'CDR (BINDINGS TYPE VAR THEN))))))


(DEFUN ALWAYS-NEVER-THEREIS-CLAUSE ()
   (SETQ *BOOLEAN-TERMINATOR* *CURRENT-KEYWORD*)
   (WHEN *ANONYMOUS-ACCUMULATOR* (AMBIGUOUS-LOOP-RESULT-ERROR))
   (ECASE *CURRENT-KEYWORD*
       (:ALWAYS
          (LP :UNLESS (FORM1)
              :RETURN NIL :END)
          (FILL-IN :RESULTS '(T)))
       (:NEVER (LP :ALWAYS `(NOT ,(FORM1))))
       (:THEREIS
          (LP :IF (FORM1)
              :RETURN :IT :END)
          (FILL-IN :RESULTS '(NIL)))))


(DEFUN AMBIGUOUS-LOOP-RESULT-ERROR ()
   (ERROR 'SIMPLE-PROGRAM-ERROR :FORMAT-CONTROL (APPEND-CONTEXT "~S cannot be used without `into' preposition
                                                                with ~S")
          :FORMAT-ARGUMENTS
          `(,*ANONYMOUS-ACCUMULATOR* ,*BOOLEAN-TERMINATOR*)))


(DEFUN APPEND-CONTEXT (MESSAGE)
   (CONCATENATE 'STRING MESSAGE (LET ((CLAUSE (LDIFF *CURRENT-CLAUSE* *LOOP-TOKENS*)))
                                     (FORMAT NIL "~%Current LOOP context:~{ ~S~}" CLAUSE))))


(DEFINE-MODIFY-MACRO APPENDF (&REST ARGS) APPEND
   "Append onto list")


(DEFUN BINDINGS (D-TYPE-SPEC D-VAR-SPEC &OPTIONAL (VALUE-FORM "NEVER USED" VALUE-FORM-P))
   (COND
      ((NULL VALUE-FORM-P)
       (DEFAULT-BINDINGS D-TYPE-SPEC D-VAR-SPEC))
      ((QUOTED-FORM-P VALUE-FORM)
       (CONSTANT-BINDINGS D-TYPE-SPEC D-VAR-SPEC (QUOTED-OBJECT VALUE-FORM)))
      (T (ORDINARY-BINDINGS D-TYPE-SPEC D-VAR-SPEC VALUE-FORM))))


(DEFUN BOUND-VARIABLES (BINDING-FORM)
   (LET ((OPERATOR (FIRST BINDING-FORM))
         (SECOND (SECOND BINDING-FORM)))
        (ECASE OPERATOR
            ((LET LET* SYMBOL-MACROLET) (MAPCAR #'FIRST SECOND))
            (T SECOND)
            ((WITH-PACKAGE-ITERATOR WITH-HASH-TABLE-ITERATOR) `(,(FIRST SECOND))))))


(DEFUN BY-STEP-FUN ()
   (IF (PREPOSITION? :BY)
       (FORM1)
       '#'CDR))


(DEFUN CAR-TYPE (D-TYPE-SPEC)
   (IF (CONSP D-TYPE-SPEC)
       (CAR D-TYPE-SPEC)
       D-TYPE-SPEC))


(DEFUN CDR-TYPE (D-TYPE-SPEC)
   (IF (CONSP D-TYPE-SPEC)
       (CDR D-TYPE-SPEC)
       D-TYPE-SPEC))


(DEFUN CHECK-MULTIPLE-BINDINGS (VARIABLES)
```

```
   (MAPL #'(LAMBDA (VARS)
                (WHEN (MEMBER (FIRST VARS)
                              (REST VARS))
                   (LOOP-ERROR 'SIMPLE-PROGRAM-ERROR :FORMAT-CONTROL "Variable ~S is bound more than once."
                          :FORMAT-ARGUMENTS (LIST (FIRST VARS)))))
         VARIABLES))


(DEFUN CL-EXTERNAL-P (SYMBOL)
   (MULTIPLE-VALUE-BIND (CL-SYMBOL STATUS)
       (FIND-SYMBOL (SYMBOL-NAME SYMBOL)
            "CL")
     (AND (EQ SYMBOL CL-SYMBOL)
          (EQ STATUS :EXTERNAL))))


(DEFUN CLAUSE* ()
   (LOOP (LET ((KEY (KEYWORD?)))
              (UNLESS KEY (RETURN))
              (CLAUSE1)))))


(DEFUN CLAUSE1 ()
   (MULTIPLE-VALUE-BIND (CLAUSE-FUNCTION-DESIGNATOR PRESENT-P)
       (GETHASH *CURRENT-KEYWORD* *LOOP-CLAUSES*)
     (UNLESS PRESENT-P
         (LOOP-ERROR "Unknown loop keyword ~S encountered." (CAR *CURRENT-CLAUSE*)))
     (LET ((*MESSAGE-PREFIX* (FORMAT NIL "LOOP ~A clause: " *CURRENT-KEYWORD*)))
          (FUNCALL CLAUSE-FUNCTION-DESIGNATOR))))


(DEFUN COMPOUND-FORMS* ()
   (WHEN (AND *LOOP-TOKENS* (CONSP (CAR *LOOP-TOKENS*)))
       (CONS (POP *LOOP-TOKENS*)
             (COMPOUND-FORMS*))))


(DEFUN COMPOUND-FORMS+ ()
   (OR (COMPOUND-FORMS*)
       (LOOP-ERROR "At least one compound form is needed.")))


(DEFUN CONDITIONAL-CLAUSE ()
   (LET* ((*IT-SYMBOL* NIL)
          (MIDDLE (GENSYM "MIDDLE-"))
          (BOTTOM (GENSYM "BOTTOM-"))
          (TEST-FORM (IF (EQ *CURRENT-KEYWORD* :UNLESS)
                         `(NOT ,(FORM1))
                         (FORM1)))
          (CONDITION-FORM `(UNLESS ,TEST-FORM
                                   (GO ,MIDDLE))))
         (LP :DO CONDITION-FORM)
         (LET ((*IT-VISIBLE-P* T))
              (SELECTABLE-CLAUSE))
         (LOOP (UNLESS (PREPOSITION? :AND)
                       (RETURN))
               (SELECTABLE-CLAUSE))
         (COND
            ((PREPOSITION? :ELSE)
             (LP :DO `(GO ,BOTTOM))
             (FILL-IN :BODY `(,MIDDLE))
             (LET ((*IT-VISIBLE-P* T))
                  (SELECTABLE-CLAUSE))
             (LOOP (UNLESS (PREPOSITION? :AND)
                           (RETURN))
                   (SELECTABLE-CLAUSE))
             (FILL-IN :BODY `(,BOTTOM)))
            (T (FILL-IN :BODY `(,MIDDLE))))
         (PREPOSITION? :END)
         (WHEN *IT-SYMBOL*
             (WITH *IT-SYMBOL*)
             (SETF (SECOND CONDITION-FORM)
                   `(SETQ ,*IT-SYMBOL* ,(SECOND CONDITION-FORM))))))


(DEFUN CONSTANT-BINDINGS (D-TYPE-SPEC D-VAR-SPEC VALUE)
   (LET ((BINDINGS NIL))
        (LABELS ((DIG (TYPE VAR VALUE)
                      (COND
                         ((NULL VAR)
                          NIL)
                         ((SIMPLE-VAR-P VAR)
                          (APPENDF BINDINGS `((,TYPE ,VAR ',VALUE))))
                         (T (DIG (CAR-TYPE TYPE)
                                 (CAR VAR)
                                 (CAR VALUE))
                            (DIG (CDR-TYPE TYPE)
```

```
                                (CDR VAR)
                                (CDR VALUE))))))
               (DIG D-TYPE-SPEC D-VAR-SPEC VALUE)
               BINDINGS)))

(DEFUN CONSTANT-FUNCTION-P (FORM)
   (LET ((EXPANSION (MACROEXPAND FORM *ENVIRONMENT*)))
        (AND (CONSP EXPANSION)
             (EQ (FIRST EXPANSION)
                 'FUNCTION)
             (SYMBOLP (SECOND EXPANSION))
             (LET ((SYMBOL (SECOND EXPANSION)))
                  (AND (CL-EXTERNAL-P SYMBOL)
                       (FBOUNDP SYMBOL))))))

(DEFUN CONSTANT-VECTOR (FORM)
   (COND
      ((QUOTED-FORM-P FORM)
       (QUOTED-OBJECT FORM))
      ((VECTORP FORM)
       FORM)
      (T (ERROR "~S is not a vector form." FORM))))

(DEFUN CONSTANT-VECTOR-P (FORM)
   (OR (QUOTED-FORM-P FORM)
       (VECTORP FORM)))

(DEFUN D-VAR-SPEC-P (SPEC)
   (OR (SIMPLE-VAR-P SPEC)
       (NULL SPEC)
       (AND (CONSP SPEC)
            (D-VAR-SPEC-P (CAR SPEC))
            (D-VAR-SPEC-P (CDR SPEC)))))

(DEFUN D-VAR-SPEC1 ()
   (UNLESS (AND *LOOP-TOKENS* (D-VAR-SPEC-P (CAR *LOOP-TOKENS*)))
           (LOOP-ERROR "A destructured-variable-spec is missing."))
   (LET ((D-VAR-SPEC (POP *LOOP-TOKENS*)))
        D-VAR-SPEC))

(DEFUN D-VAR-TYPE-SPEC ()
   (LET ((VAR (D-VAR-SPEC1))
         (TYPE (TYPE-SPEC?)))
        (WHEN (EMPTY-P VAR)
            (UNLESS (MEMBER TYPE '(NIL T))
                    (LOOP-WARN "Type spec ~S is ignored." TYPE))
            (SETQ VAR (GENSYM)
                  TYPE T))
        (VALUES VAR TYPE)))

(DEFUN DECLARATIONS (BINDINGS)
   (LET ((DECLARATIONS (MAPCAN #'(LAMBDA (BINDING)
                                         (DESTRUCTURING-BIND (TYPE VAR . REST)
                                                BINDING
                                                (DECLARE (IGNORE REST))
                                                (UNLESS (EQ TYPE 'T)
                                                    `((TYPE ,TYPE ,VAR)))))
                               BINDINGS)))
        (WHEN DECLARATIONS
            `((DECLARE ,@DECLARATIONS)))))

(DEFUN DEFAULT-BINDING (TYPE VAR)
   `(,(DEFAULT-TYPE TYPE)
     ,VAR
     ,(DEFAULT-VALUE TYPE)))

(DEFUN DEFAULT-BINDINGS (D-TYPE-SPEC D-VAR-SPEC)
   (LET ((BINDINGS NIL))
        (LABELS ((DIG (TYPE VAR)
                      (COND
                         ((NULL VAR)
                          NIL)
                         ((SIMPLE-VAR-P VAR)
                          (APPENDF BINDINGS `(,(DEFAULT-BINDING TYPE VAR))))
                         (T (DIG (CAR-TYPE TYPE)
                                 (CAR VAR))
                            (DIG (CDR-TYPE TYPE)
                                 (CDR VAR))))))
```

```
              (DIG D-TYPE-SPEC D-VAR-SPEC)
              BINDINGS)))
```

```
(DEFUN DEFAULT-TYPE (TYPE)                                  ; Edited 13-Jun-2024 20:05 by mth
   ;; Probably shouldn't ever happen, but if TYPE is NIL
   (IF (OR (NULL TYPE)
           (EQ TYPE T))
       T
       (LET ((VALUE (DEFAULT-VALUE TYPE)))
            (IF (TYPEP VALUE TYPE)
                TYPE
                (LET ((DEFAULT-TYPE (TYPE-OF VALUE)))
                     (IF (SUBTYPEP TYPE DEFAULT-TYPE)
                         DEFAULT-TYPE
                         (IF (NULL VALUE)
                             `(OR NULL ,TYPE)
                             `(OR ,DEFAULT-TYPE ,TYPE)))))))))
```

```
(DEFUN DEFAULT-VALUE (TYPE)                                 ; Edited 13-Jun-2024 20:31 by mth
   (COND
      ((NULL TYPE)
       ;; giving NIL specifically as the VAR type probably shouldn't happen, but seems to be "legal", so handle it
       NIL)
      ((SUBTYPEP TYPE 'BIGNUM)
       (1+ MOST-POSITIVE-FIXNUM))
      ((SUBTYPEP TYPE 'INTEGER)
       0)
      ((SUBTYPEP TYPE 'RATIO)
       1/10)
      ((SUBTYPEP TYPE 'FLOAT)
       0.0)
      ((SUBTYPEP TYPE 'NUMBER)
       0)
      ((SUBTYPEP TYPE 'CHARACTER)
       #\Space)
      ((SUBTYPEP TYPE 'STRING)
       "")
      ((SUBTYPEP TYPE 'BIT-VECTOR)
       #*0)
      ((SUBTYPEP TYPE 'VECTOR)
       #())
      ((SUBTYPEP TYPE 'PACKAGE)
       *PACKAGE*)
      (T NIL)))
```

```
(DEFUN DESTRUCTURING-MULTIPLE-VALUE-BIND (D-TYPE-SPEC D-VAR-SPEC VALUE-FORM)
   (LET ((MV-BINDINGS NIL)
         (D-BINDINGS NIL)
         (PADDING-TEMPS NIL)
         TEMP)
        (DO ((VARS D-VAR-SPEC (CDR VARS))
             (TYPES D-TYPE-SPEC (CDR-TYPE TYPES)))
            ((ENDP VARS))
          (IF (LISTP (CAR VARS))
              (PROGN (SETQ TEMP (GENSYM))
                     (APPENDF MV-BINDINGS `((T ,TEMP)))
                     (APPENDF D-BINDINGS `((,(CAR-TYPE TYPES)
                                            ,(CAR VARS)
                                            ,TEMP)))
                     (WHEN (EMPTY-P (CAR VARS))
                           (PUSH TEMP PADDING-TEMPS)))
              (APPENDF MV-BINDINGS `((,(CAR-TYPE TYPES)
                                      ,(CAR VARS))))))
        (FILL-IN :BINDING-FORMS `((MULTIPLE-VALUE-BIND ,(MAPCAR #'SECOND MV-BINDINGS)
                                   ,(MULTIPLE-VALUE-LIST-ARGUMENT-FORM VALUE-FORM)
                                   ,@(DECLARATIONS MV-BINDINGS)
                                   ,@(WHEN PADDING-TEMPS
                                           `((DECLARE (IGNORE ,@PADDING-TEMPS)))))))
        (LET ((BINDINGS (MAPAPPEND #'(LAMBDA (D-BINDING)
                                       (APPLY #'BINDINGS D-BINDING))
                                   D-BINDINGS)))
             (WHEN BINDINGS
                   (FILL-IN :BINDING-FORMS `(,(LET-FORM BINDINGS))))))))
```

```
(DEFUN DESTRUCTURING-MULTIPLE-VALUE-SETQ (D-VAR-SPEC VALUE-FORM &KEY ITERATOR-P)
   (LET (D-BINDINGS MV-VARS TEMP)
        (DO ((VARS D-VAR-SPEC (CDR VARS)))
            ((ENDP VARS))
          (IF (LISTP (CAR VARS))
              (PROGN (SETQ TEMP (OR (POP *TEMPORARIES*)
                                    (GENSYM-IGNORABLE)))
```

```
                    (APPENDF MV-VARS `(,TEMP))
                    (APPENDF D-BINDINGS `((T ,(CAR VARS)
                                             ,TEMP))))
             (APPENDF MV-VARS `(,(CAR VARS)))))
       (LET ((MV-SETQ-FORM `(MULTIPLE-VALUE-SETQ ,MV-VARS ,VALUE-FORM))
             (BINDINGS NIL))
            (DO ((D-BINDINGS D-BINDINGS (CDR D-BINDINGS)))
                ((ENDP D-BINDINGS))
              (DESTRUCTURING-BIND (TYPE VAR TEMP)
                     (CAR D-BINDINGS)
                     (DECLARE (IGNORE TYPE VAR))
                     (PUSH TEMP *TEMPORARIES*)
                     (APPENDF BINDINGS (APPLY #'BINDINGS (CAR D-BINDINGS)))))
            (WHEN ITERATOR-P
                (SETQ MV-SETQ-FORM `(UNLESS ,MV-SETQ-FORM (LOOP-FINISH))))
            (IF BINDINGS
                `(PROGN ,MV-SETQ-FORM (SETQ ,@(MAPAPPEND #'CDR BINDINGS)))
                MV-SETQ-FORM))))

(DEFUN DISPATCH-FOR-AS-SUBCLAUSE (VAR TYPE)
   (UNLESS *LOOP-TOKENS* (LOOP-ERROR "A preposition is missing."))
   (LET ((PREPOSITION (PREPOSITION1 *FOR-AS-PREPOSITIONS*)))
        (MULTIPLE-VALUE-BIND (SUBCLAUSE-FUNCTION-DESIGNATOR PRESENT-P)
            (GETHASH PREPOSITION *FOR-AS-SUBCLAUSES*)
          (UNLESS PRESENT-P (LOOP-ERROR "Unknown preposition ~S is supplied." PREPOSITION))
          (PUSH PREPOSITION *LOOP-TOKENS*)
          (FUNCALL SUBCLAUSE-FUNCTION-DESIGNATOR VAR TYPE))))

(DEFUN DO-CLAUSE ()
   (FILL-IN :BODY (COMPOUND-FORMS+)))

(DEFUN EMPTY-P (D-VAR-SPEC)
   (OR (NULL D-VAR-SPEC)
       (AND (CONSP D-VAR-SPEC)
            (EMPTY-P (CAR D-VAR-SPEC))
            (EMPTY-P (CDR D-VAR-SPEC)))))

(DEFUN ENUMERATE (ITEMS)
   (CASE (LENGTH ITEMS)
       (1 (FORMAT NIL "~S" (FIRST ITEMS)))
       (2 (FORMAT NIL "~S and ~S" (FIRST ITEMS)
                  (SECOND ITEMS)))
       (T (FORMAT NIL "~{~S, ~}and ~S" (BUTLAST ITEMS)
                  (FIRST (LAST ITEMS))))))

(DEFMACRO EXTENDED-LOOP (&REST TOKENS &ENVIRONMENT ENVIRONMENT)
   (LET
    ((*ENVIRONMENT* ENVIRONMENT))
    (WITH-LOOP-CONTEXT
     TOKENS
     (LET
      ((BODY-TAG (GENSYM "LOOP-BODY-"))
       (EPILOGUE-TAG (GENSYM "LOOP-EPILOGUE-")))
      (NAME-CLAUSE?)
      (VARIABLE-CLAUSE*)
      (MAIN-CLAUSE*)
      (WHEN *LOOP-TOKENS* (ERROR "Loop form tail ~S remained unprocessed." *LOOP-TOKENS*))
      (REDUCE-REDUNDANT-CODE)
      (DESTRUCTURING-BIND
       (&KEY BINDING-FORMS ITERATOR-FORMS INITIALLY HEAD NECK BODY TAIL FINALLY RESULTS)
       *LOOP-COMPONENTS*
       (CHECK-MULTIPLE-BINDINGS (APPEND *TEMPORARIES* (MAPAPPEND #'BOUND-VARIABLES BINDING-FORMS)
                                        (MAPCAR #'(LAMBDA (SPEC)
                                                          (GETF (CDR SPEC)
                                                                :VAR))
                                                *ACCUMULATORS*)))
       `(BLOCK ,*LOOP-NAME*
            ,(WITH-TEMPORARIES
              `(,*TEMPORARIES* :IGNORABLE ,*IGNORABLE*)
              (WITH-ACCUMULATORS
               *ACCUMULATORS*
               (WITH-BINDING-FORMS BINDING-FORMS
                      (WITH-ITERATOR-FORMS
                       ITERATOR-FORMS
                       `(MACROLET ((LOOP-FINISH NIL '(GO ,EPILOGUE-TAG)))
                                  (TAGBODY ,@HEAD ,@INITIALLY ,BODY-TAG ,@NECK ,@BODY ,@TAIL (GO ,BODY-TAG)
                                         ,EPILOGUE-TAG
                                         ,@FINALLY
                                         ,@(WHEN RESULTS
                                                 `((RETURN-FROM ,*LOOP-NAME* ,(CAR RESULTS))))))))))))))))
```

```
(DEFUN FILL-IN (&REST ARGS)
   (WHEN ARGS
       (APPENDF (GETF *LOOP-COMPONENTS* (FIRST ARGS))
           (SECOND ARGS))
       (APPLY #'FILL-IN (CDDR ARGS))))


(DEFUN FINALLY-CLAUSE ()
   (FILL-IN :FINALLY (COMPOUND-FORMS+)))


(DEFUN FOR (VAR TYPE &REST REST)
   (LET ((*LOOP-TOKENS* REST))
        (DISPATCH-FOR-AS-SUBCLAUSE VAR TYPE)))


(DEFUN FOR-AS-ACROSS-SUBCLAUSE (VAR TYPE)
   (PREPOSITION1 :ACROSS)
   (LET* ((FORM (FORM1))
          (VECTOR (IF (CONSTANT-VECTOR-P FORM)
                      FORM
                      (GENSYM "VECTOR-")))
          (LENGTH (IF (CONSTANT-VECTOR-P FORM)
                      (LENGTH (CONSTANT-VECTOR FORM))
                      (GENSYM "LENGTH-")))
          (I (GENSYM "INDEX-"))
          (AT-LEAST-ONE-ITERATION-P (AND (CONSTANT-VECTOR-P FORM)
                                         (PLUSP LENGTH))))
         (UNLESS (CONSTANT-VECTOR-P FORM)
             (FOR-AS-FILL-IN :BINDINGS `((T ,VECTOR ,FORM))
                    :BINDINGS2
                    `((FIXNUM ,LENGTH (LENGTH ,VECTOR)))))
         (FOR-AS-FILL-IN :BINDINGS `((FIXNUM ,I 0))
                :HEAD-TESTS
                (UNLESS AT-LEAST-ONE-ITERATION-P
                    `((= ,I ,LENGTH)))
                :TAIL-PSETQ
                `(,I (1+ ,I))
                :TAIL-TESTS
                `((= ,I ,LENGTH)))
         (ALONG-WITH VAR TYPE :EQUALS (IF AT-LEAST-ONE-ITERATION-P
                                          `',(AREF (CONSTANT-VECTOR FORM)
                                                   0)
                                          `(AREF ,VECTOR ,I))
                :THEN
                `(AREF ,VECTOR ,I))))


(DEFUN FOR-AS-ARITHMETIC-POSSIBLE-PREPOSITIONS (USED-PREPOSITIONS)
   (APPEND (COND
              ((INTERSECTION '(:FROM :DOWNFROM :UPFROM)
                      USED-PREPOSITIONS)
               NIL)
              ((INTERSECTION '(:DOWNTO :ABOVE)
                      USED-PREPOSITIONS)
               '(:FROM :DOWNFROM))
              ((INTERSECTION '(:UPTO :BELOW)
                      USED-PREPOSITIONS)
               '(:FROM :UPFROM))
              (T '(:FROM :DOWNFROM :UPFROM)))
          (COND
             ((INTERSECTION '(:TO :DOWNTO :UPTO :BELOW :ABOVE)
                     USED-PREPOSITIONS)
              NIL)
             ((FIND :UPFROM USED-PREPOSITIONS)
              '(:TO :UPTO :BELOW))
             ((FIND :DOWNFROM USED-PREPOSITIONS)
              '(:TO :DOWNTO :ABOVE))
             (T '(:TO :DOWNTO :UPTO :BELOW :ABOVE)))
          (UNLESS (FIND :BY USED-PREPOSITIONS)
              '(:BY))))


(DEFUN FOR-AS-ARITHMETIC-STEP-AND-TEST-FUNCTIONS (USED-PREPOSITIONS)
   (LET ((UP-P (SUBSETP USED-PREPOSITIONS '(:BELOW :UPTO :UPFROM :FROM :TO :BY))))
        (VALUES (IF UP-P
                    '+
                    '-)
                (COND
                   ((MEMBER :TO USED-PREPOSITIONS)
                    (IF UP-P
                        '>
                        '<))
                   ((MEMBER :UPTO USED-PREPOSITIONS)
                    '>)
                   ((MEMBER :BELOW USED-PREPOSITIONS)
                    '>=)
```

```
                ((MEMBER :DOWNTO USED-PREPOSITIONS)
                 '<)
                ((MEMBER :ABOVE USED-PREPOSITIONS)
                 '<=)
                (T NIL)))))

(DEFUN FOR-AS-ARITHMETIC-SUBCLAUSE (VAR TYPE)
   (UNLESS (SIMPLE-VAR-P VAR)
           (LOOP-ERROR "Destructuring on a number is invalid."))
   (MULTIPLE-VALUE-BIND (SUBTYPE-P VALID-P)
       (SUBTYPEP TYPE 'REAL)
     (WHEN (AND (NOT SUBTYPE-P)
                VALID-P)
           (SETQ TYPE 'REAL)))
   (LET (FROM TO BY PREPOSITION USED CANDIDATES BINDINGS)
        (LOOP (SETQ CANDIDATES (OR (FOR-AS-ARITHMETIC-POSSIBLE-PREPOSITIONS USED)
                                   (RETURN)))
              (PUSH (OR (SETQ PREPOSITION (PREPOSITION? CANDIDATES))
                        (RETURN))
                    USED)
              (LET ((VALUE-FORM (FORM1)))
                   (IF (MEMBER PREPOSITION '(:FROM :DOWNFROM :UPFROM))
                       (PROGN (SETQ FROM VALUE-FORM)
                              (APPENDF BINDINGS `((,TYPE ,VAR ,FROM))))
                       (PROGN (WHEN (NOT (CONSTANTP VALUE-FORM *ENVIRONMENT*))
                                    (LET ((TEMP (GENSYM)))
                                         (APPENDF BINDINGS `((NUMBER ,TEMP ,VALUE-FORM)))
                                         (SETQ VALUE-FORM TEMP)))
                              (ECASE PREPOSITION
                                  ((:TO :DOWNTO :UPTO :BELOW :ABOVE) (SETQ TO VALUE-FORM))
                                  (:BY (SETQ BY VALUE-FORM)))))))
        (UNLESS (INTERSECTION USED '(:FROM :DOWNFROM :UPFROM))
            (APPENDF BINDINGS `((,TYPE ,VAR ,(ZERO TYPE)))))
        (MULTIPLE-VALUE-BIND (STEP TEST)
            (FOR-AS-ARITHMETIC-STEP-AND-TEST-FUNCTIONS USED)
          (LET ((TESTS (WHEN TEST
                             `((,TEST ,VAR ,TO)))))
               (FOR-AS-FILL-IN :BINDINGS BINDINGS :HEAD-TESTS TESTS :TAIL-PSETQ
                      `(,VAR (,STEP ,VAR ,(OR BY (ONE TYPE))))
                      :TAIL-TESTS TESTS))))))

(DEFUN FOR-AS-BEING-SUBCLAUSE (VAR TYPE)
   (PREPOSITION1 :BEING)
   (PREPOSITION1 '(:EACH :THE))
   (LET* ((KIND (PREPOSITION1 (APPEND *HASH-GROUP* *SYMBOL-GROUP*))))
         (COND
            ((FIND KIND *HASH-GROUP*)
             (FOR-AS-HASH-SUBCLAUSE VAR TYPE KIND))
            ((FIND KIND *SYMBOL-GROUP*)
             (FOR-AS-PACKAGE-SUBCLAUSE VAR TYPE KIND))
            (T (LOOP-ERROR "Internal logic error")))))

(DEFUN FOR-AS-CLAUSE ()
   (LET ((*FOR-AS-COMPONENTS* NIL))
        (LOOP (MULTIPLE-VALUE-BIND (VAR TYPE)
                  (D-VAR-TYPE-SPEC)
                (DISPATCH-FOR-AS-SUBCLAUSE VAR TYPE))
              (UNLESS (PREPOSITION? :AND)
                      (RETURN)))
        (DESTRUCTURING-BIND (&KEY BINDINGS BINDINGS2 BEFORE-HEAD HEAD-PSETQ HEAD-TESTS AFTER-HEAD BEFORE-TAIL
                                  TAIL-PSETQ TAIL-TESTS AFTER-TAIL)
              *FOR-AS-COMPONENTS*
              (FILL-IN :BINDING-FORMS `(,@(WHEN BINDINGS
                                                `(,(LET-FORM BINDINGS)))
                                        ,@(WHEN BINDINGS2
                                                `(,(LET-FORM BINDINGS2))))
                       :HEAD
                       `(,@BEFORE-HEAD ,@(PSETQ-FORMS HEAD-PSETQ)
                               ,@(LOOP-FINISH-TEST-FORMS HEAD-TESTS)
                               ,@AFTER-HEAD)
                       :TAIL
                       `(,@BEFORE-TAIL ,@(PSETQ-FORMS TAIL-PSETQ)
                               ,@(LOOP-FINISH-TEST-FORMS TAIL-TESTS)
                               ,@AFTER-TAIL))))))

(DEFUN FOR-AS-EQUALS-THEN-SUBCLAUSE (VAR TYPE)
   (PREPOSITION1 :=)
   (LET* ((FIRST (FORM1))
          (THEN (IF (PREPOSITION? :THEN)
                    (FORM1)
                    FIRST))
          (PARALLEL-P (FOR-AS-PARALLEL-P)))
         (FOR-AS-FILL-IN :BINDINGS (APPLY #'BINDINGS TYPE VAR (WHEN (QUOTED-FORM-P FIRST)
```

```
                                                                    `(,FIRST))))
        (IF (AND (NOT PARALLEL-P)
                 (CONSP VAR)
                 (MULTIPLE-VALUE-LIST-FORM-P FIRST))
            (FOR-AS-FILL-IN :BEFORE-HEAD `(,(DESTRUCTURING-MULTIPLE-VALUE-SETQ VAR (
                                                                    MULTIPLE-VALUE-LIST-ARGUMENT-FORM
                                                                        FIRST))))
            (UNLESS (QUOTED-FORM-P FIRST)
                (FOR-AS-FILL-IN :HEAD-PSETQ (MAPAPPEND #'CDR (BINDINGS TYPE VAR FIRST)))))
        (IF (AND (NOT PARALLEL-P)
                 (CONSP VAR)
                 (MULTIPLE-VALUE-LIST-FORM-P THEN))
            (FOR-AS-FILL-IN :BEFORE-TAIL `(,(DESTRUCTURING-MULTIPLE-VALUE-SETQ VAR (
                                                                    MULTIPLE-VALUE-LIST-ARGUMENT-FORM
                                                                        THEN))))
            (FOR-AS-FILL-IN :TAIL-PSETQ (MAPAPPEND #'CDR (BINDINGS TYPE VAR THEN))))))


(DEFUN FOR-AS-FILL-IN (&REST KEY-LIST-PAIRS)
   (WHEN KEY-LIST-PAIRS
       (DESTRUCTURING-BIND (KEY LIST . REST)
             KEY-LIST-PAIRS
             (APPENDF (GETF *FOR-AS-COMPONENTS* KEY)
                    LIST)
             (APPLY #'FOR-AS-FILL-IN REST))))


(DEFUN FOR-AS-HASH-SUBCLAUSE (VAR TYPE KIND)
   (LET* ((HASH-TABLE (PROGN (PREPOSITION1 '(:IN :OF))
                             (FORM1)))
          (OTHER-VAR (USING-OTHER-VAR KIND))
          (FOR-AS-PARALLEL-P (FOR-AS-PARALLEL-P))
          (RETURNED-P (OR (POP *TEMPORARIES*)
                          (GENSYM-IGNORABLE)))
          (ITERATOR (GENSYM))
         NARROW-TYPED-VAR NARROW-TYPE)
        (WHEN (AND (SIMPLE-VAR-P VAR)
                   (NOT (TYPEP 'NIL TYPE)))
            (SETQ NARROW-TYPED-VAR VAR NARROW-TYPE TYPE)
            (SETQ VAR (GENSYM)
                  TYPE
                  `(OR NULL ,TYPE))
            (FOR-AS-FILL-IN :BINDINGS `(,(DEFAULT-BINDING NARROW-TYPE NARROW-TYPED-VAR))))
        (FLET ((ITERATOR-FORM NIL `(WITH-HASH-TABLE-ITERATOR (,ITERATOR ,HASH-TABLE))))
          (IF FOR-AS-PARALLEL-P
              (PROGN (UNLESS (CONSTANTP HASH-TABLE *ENVIRONMENT*)
                         (LET ((TEMP (GENSYM "HASH-TABLE-")))
                              (FOR-AS-FILL-IN :BINDINGS `((T ,TEMP ,HASH-TABLE)))
                              (SETQ HASH-TABLE TEMP)))
                     (FILL-IN :ITERATOR-FORMS `(,(ITERATOR-FORM))))
              (FILL-IN :BINDING-FORMS `(,(ITERATOR-FORM)))))
        (LET* ((D-VAR-SPEC (HASH-D-VAR-SPEC RETURNED-P VAR OTHER-VAR KIND))
               (D-MV-SETQ (DESTRUCTURING-MULTIPLE-VALUE-SETQ D-VAR-SPEC `(,ITERATOR)
                                 :ITERATOR-P T))
               (SETTERS `(,D-MV-SETQ ,@(WHEN NARROW-TYPED-VAR
                                         `((SETQ ,NARROW-TYPED-VAR ,VAR))))))
              (PUSH RETURNED-P *TEMPORARIES*)
              (FOR-AS-FILL-IN :BINDINGS `(,@(BINDINGS TYPE VAR)
                                          ,@(WHEN OTHER-VAR (BINDINGS T OTHER-VAR)))
                    :AFTER-HEAD SETTERS :AFTER-TAIL SETTERS))))


(DEFUN FOR-AS-IN-LIST-SUBCLAUSE (VAR TYPE)
   (PREPOSITION1 :IN)
   (LET ((*LIST-END-TEST* 'ENDP))
        (FOR `(,VAR)
             `(,TYPE)
             :ON
             (FORM1)
             :BY
             (BY-STEP-FUN))))


(DEFUN FOR-AS-ON-LIST-SUBCLAUSE (VAR TYPE)
   (PREPOSITION1 :ON)
   (LET* ((FORM (FORM1))
          (BY-STEP-FUN (BY-STEP-FUN))
          (TEST *LIST-END-TEST*)
          (LIST-VAR (IF (SIMPLE-VAR-P VAR)
                        VAR
                        (GENSYM "LIST-")))
          (LIST-TYPE (IF (SIMPLE-VAR-P VAR)
                         TYPE
                         T))
          (AT-LEAST-ONE-ITERATION-P (AND (QUOTED-FORM-P FORM)
                                         (NOT (FUNCALL TEST (QUOTED-OBJECT FORM))))))
        (FOR-AS-FILL-IN :BINDINGS `((,LIST-TYPE ,LIST-VAR ,FORM)
```

```
                                   ,@(UNLESS (CONSTANT-FUNCTION-P BY-STEP-FUN)
                                         (LET ((TEMP (GENSYM "STEPPER-")))
                                              (PROG1 `((T ,TEMP ,BY-STEP-FUN))
                                                     (SETQ BY-STEP-FUN TEMP)))))
                :HEAD-TESTS
                (UNLESS AT-LEAST-ONE-ITERATION-P
                    `((,TEST ,LIST-VAR)))
                :TAIL-PSETQ
                `(,LIST-VAR (FUNCALL ,BY-STEP-FUN ,LIST-VAR))
                :TAIL-TESTS
                `((,TEST ,LIST-VAR)))
          (UNLESS (SIMPLE-VAR-P VAR)
              (ALONG-WITH VAR TYPE :EQUALS (IF AT-LEAST-ONE-ITERATION-P
                                               FORM
                                               LIST-VAR)
                    :THEN LIST-VAR))))

(DEFUN FOR-AS-PACKAGE-SUBCLAUSE (VAR TYPE KIND)
   (LET* ((PACKAGE (IF (PREPOSITION? '(:IN :OF))
                       (FORM1)
                       '*PACKAGE*))
          (FOR-AS-PARALLEL-P (FOR-AS-PARALLEL-P))
          (RETURNED-P (OR (POP *TEMPORARIES*)
                          (GENSYM-IGNORABLE)))
          (ITERATOR (GENSYM))
          (KINDS (ECASE KIND
                     ((:SYMBOL :SYMBOLS) '(:INTERNAL :EXTERNAL :INHERITED))
                     ((:PRESENT-SYMBOL :PRESENT-SYMBOLS) '(:INTERNAL :EXTERNAL))
                     ((:EXTERNAL-SYMBOL :EXTERNAL-SYMBOLS) '(:EXTERNAL)))))
          (UNLESS (TYPEP 'NIL TYPE)
              (SETQ TYPE `(OR NULL ,TYPE)))
          (FLET ((ITERATOR-FORM NIL `(WITH-PACKAGE-ITERATOR (,ITERATOR ,PACKAGE ,@KINDS))))
                (IF FOR-AS-PARALLEL-P
                    (PROGN (UNLESS (CONSTANTP PACKAGE *ENVIRONMENT*)
                                (LET ((TEMP (GENSYM "PACKAGE-")))
                                     (FOR-AS-FILL-IN :BINDINGS `((T ,TEMP ,PACKAGE)))
                                     (SETQ PACKAGE TEMP)))
                           (FILL-IN :ITERATOR-FORMS `(,(ITERATOR-FORM))))
                    (FILL-IN :BINDING-FORMS `(,(ITERATOR-FORM)))))
          (LET* ((D-VAR-SPEC `(,RETURNED-P ,VAR))
                 (D-MV-SETQ (DESTRUCTURING-MULTIPLE-VALUE-SETQ D-VAR-SPEC `(,ITERATOR)
                                   :ITERATOR-P T)))
                (PUSH RETURNED-P *TEMPORARIES*)
                (FOR-AS-FILL-IN :BINDINGS (BINDINGS TYPE VAR)
                       :AFTER-HEAD
                       `(,D-MV-SETQ)
                       :AFTER-TAIL
                       `(,D-MV-SETQ)))))

(DEFUN FOR-AS-PARALLEL-P ()
   (OR *FOR-AS-COMPONENTS* (AND *LOOP-TOKENS* (SYMBOLP (CAR *LOOP-TOKENS*))
                                (STRING= (SYMBOL-NAME (CAR *LOOP-TOKENS*))
                                         "AND"))))

(DEFUN FORM-OR-IT ()
   (IF (AND *IT-VISIBLE-P* (PREPOSITION? :IT))
       (OR *IT-SYMBOL* (SETQ *IT-SYMBOL* (GENSYM)))
       (FORM1)))

(DEFUN FORM1 ()
   (UNLESS *LOOP-TOKENS* (LOOP-ERROR "A normal lisp form is missing."))
   (POP *LOOP-TOKENS*))

(DEFUN GENSYM-IGNORABLE ()
   (LET ((VAR (GENSYM)))
        (PUSH VAR *IGNORABLE*)
        VAR))

(DEFUN GLOBALLY-SPECIAL-P (SYMBOL)
   (ASSERT (SYMBOLP SYMBOL))
   (IL:VARIABLE-GLOBALLY-SPECIAL-P SYMBOL))

(DEFUN HASH-D-VAR-SPEC (RETURNED-P VAR OTHER-VAR KIND)
   (IF (FIND KIND '(:HASH-KEY :HASH-KEYS))
       `(,RETURNED-P ,VAR ,OTHER-VAR)
       `(,RETURNED-P ,OTHER-VAR ,VAR)))

(DEFUN INITIALLY-CLAUSE ()
   (FILL-IN :INITIALLY (COMPOUND-FORMS+)))
```

```
(DEFUN INVALID-ACCUMULATOR-COMBINATION-ERROR (KEYS)
   (LOOP-ERROR "Accumulator ~S cannot be mixed with ~S." *CURRENT-KEYWORD* (ENUMERATE KEYS)))


(DEFUN KEYWORD1 (KEYWORD-LIST-DESIGNATOR &KEY PREPOSITIONP)
   (LET ((KEYWORDS (%LIST KEYWORD-LIST-DESIGNATOR)))
        (OR (KEYWORD? KEYWORDS)
            (LET ((LENGTH (LENGTH KEYWORDS))
                  (KIND (IF PREPOSITIONP
                            "preposition"
                            "keyword")))
                 (CASE LENGTH
                     (0 (LOOP-ERROR "A loop ~A is missing." KIND))
                     (1 (LOOP-ERROR "Loop ~A ~S is missing." KIND (CAR KEYWORDS)))
                     (T (LOOP-ERROR "One of the loop ~As ~S must be supplied." KIND KEYWORDS)))))))


(DEFUN KEYWORD? (&OPTIONAL KEYWORD-LIST-DESIGNATOR)
   (AND *LOOP-TOKENS* (SYMBOLP (CAR *LOOP-TOKENS*))
        (LET ((KEYWORD-LIST (%LIST KEYWORD-LIST-DESIGNATOR))
              (KEYWORD (%KEYWORD (CAR *LOOP-TOKENS*))))
             (AND (OR (NULL KEYWORD-LIST)
                      (FIND KEYWORD KEYWORD-LIST))
                  (SETQ *CURRENT-CLAUSE* *LOOP-TOKENS* *LOOP-TOKENS* (REST *LOOP-TOKENS*)
                        *CURRENT-KEYWORD* KEYWORD))))))


(DEFUN LET-FORM (BINDINGS)
   `(LET ,(MAPCAR #'CDR BINDINGS)
         ,@(DECLARATIONS BINDINGS)))


(DEFUN LOOP-ERROR (DATUM &REST ARGUMENTS)
   (WHEN (STRINGP DATUM)
       (SETQ DATUM (APPEND-CONTEXT DATUM)))
   (APPLY #'ERROR DATUM ARGUMENTS))


(DEFUN LOOP-FINISH-TEST-FORMS (TESTS)
   (CASE (LENGTH TESTS)
       (0 NIL)
       (1 `((WHEN ,@TESTS (LOOP-FINISH))))
       (T `((WHEN (OR ,@TESTS)
                  (LOOP-FINISH))))))


(DEFUN LOOP-WARN (DATUM &REST ARGUMENTS)
   (WHEN (STRINGP DATUM)
       (SETQ DATUM (APPEND-CONTEXT DATUM)))
   (APPLY #'WARN DATUM ARGUMENTS))


(DEFUN LP (&REST TOKENS)
   (LET ((*LOOP-TOKENS* TOKENS)
         *CURRENT-KEYWORD* *CURRENT-CLAUSE*)
        (CLAUSE*)
        (WHEN *LOOP-TOKENS* (ERROR "~S remained after lp." *LOOP-TOKENS*))))


(DEFUN MAIN-CLAUSE* ()
   (LOOP (IF (KEYWORD? '(:DO :DOING :RETURN :IF :WHEN :UNLESS :INITIALLY :FINALLY :WHILE :UNTIL :REPEAT
                              :ALWAYS :NEVER :THEREIS :COLLECT :COLLECTING :APPEND :APPENDING :NCONC :NCONCING
                              :COUNT :COUNTING :SUM :SUMMING :MAXIMIZE :MAXIMIZING :MINIMIZE :MINIMIZING))
             (CLAUSE1)
             (RETURN))))


(DEFUN MAPAPPEND (FUNCTION &REST LISTS)
   (APPLY #'APPEND (APPLY #'MAPCAR FUNCTION LISTS)))


(DEFUN MULTIPLE-VALUE-LIST-ARGUMENT-FORM (FORM)
   (LET ((EXPANSION FORM)
         (EXPANDED-P NIL))
        (LOOP (WHEN (AND (CONSP EXPANSION)
                         (EQ (FIRST EXPANSION)
                             'MULTIPLE-VALUE-LIST))
                    (RETURN (SECOND EXPANSION)))
              (MULTIPLE-VALUE-SETQ (EXPANSION EXPANDED-P)
                     (MACROEXPAND-1 EXPANSION *ENVIRONMENT*))
              (UNLESS EXPANDED-P (ERROR "~S is not expanded into a multiple-value-list form." FORM)))))


(DEFUN MULTIPLE-VALUE-LIST-FORM-P (FORM)
   (LET (EXPANDED-P)
```

```
        (LOOP (WHEN (AND (CONSP FORM)
                         (EQ (FIRST FORM)
                             'MULTIPLE-VALUE-LIST))
                    (RETURN T))
              (MULTIPLE-VALUE-SETQ (FORM EXPANDED-P)
                     (MACROEXPAND-1 FORM *ENVIRONMENT*))
              (UNLESS EXPANDED-P (RETURN NIL)))))


(DEFUN NAME-CLAUSE? ()
   (WHEN (KEYWORD? :NAMED)
       (UNLESS *LOOP-TOKENS* (LOOP-ERROR "A loop name is missing."))
       (LET ((NAME (POP *LOOP-TOKENS*)))
            (UNLESS (SYMBOLP NAME)
                    (LOOP-ERROR "~S cannot be a loop name which must be a symbol." NAME))
            (SETQ *LOOP-NAME* NAME))))


(DEFUN ONE (TYPE)
   (COND
      ((SUBTYPEP TYPE 'SHORT-FLOAT)
       1.0)
      ((SUBTYPEP TYPE 'SINGLE-FLOAT)
       1.0)
      ((SUBTYPEP TYPE 'DOUBLE-FLOAT)
       1.0)
      ((SUBTYPEP TYPE 'LONG-FLOAT)
       1.0)
      ((SUBTYPEP TYPE 'FLOAT)
       1.0)
      (T 1)))


(DEFUN ORDINARY-BINDINGS (D-TYPE-SPEC D-VAR-SPEC VALUE-FORM)
   (LET ((TEMPORARIES *TEMPORARIES*)
         (BINDINGS NIL))
        (LABELS ((DIG (TYPE VAR FORM TEMP)
                      (COND
                         ((EMPTY-P VAR)
                          NIL)
                         ((SIMPLE-VAR-P VAR)
                          (WHEN TEMP (PUSH TEMP TEMPORARIES))
                          (APPENDF BINDINGS `((,TYPE ,VAR ,FORM))))
                         ((EMPTY-P (CAR VAR))
                          (DIG (CDR-TYPE TYPE)
                               (CDR VAR)
                               `(CDR ,FORM)
                               TEMP))
                         ((EMPTY-P (CDR VAR))
                          (WHEN TEMP (PUSH TEMP TEMPORARIES))
                          (DIG (CAR-TYPE TYPE)
                               (CAR VAR)
                               `(CAR ,FORM)
                               NIL))
                         (T (UNLESS TEMP
                                    (SETQ TEMP (OR (POP TEMPORARIES)
                                                   (GENSYM))))
                            (DIG (CAR-TYPE TYPE)
                                 (CAR VAR)
                                 `(CAR (SETQ ,TEMP ,FORM))
                                 NIL)
                            (DIG (CDR-TYPE TYPE)
                                 (CDR VAR)
                                 `(CDR ,TEMP)
                                 TEMP)))))
                (DIG D-TYPE-SPEC D-VAR-SPEC VALUE-FORM NIL)
                (SETQ *TEMPORARIES* TEMPORARIES)
                BINDINGS)))


(DEFUN PREPOSITION1 (&OPTIONAL KEYWORD-LIST-DESIGNATOR)
   (LET ((*CURRENT-KEYWORD* *CURRENT-KEYWORD*)
         (*CURRENT-CLAUSE* *CURRENT-CLAUSE*))
        (KEYWORD1 KEYWORD-LIST-DESIGNATOR :PREPOSITIONP T)))


(DEFUN PREPOSITION? (&OPTIONAL KEYWORD-LIST-DESIGNATOR)
   (LET ((*CURRENT-KEYWORD* *CURRENT-KEYWORD*)
         (*CURRENT-CLAUSE* *CURRENT-CLAUSE*))
        (KEYWORD? KEYWORD-LIST-DESIGNATOR)))


(DEFUN PSETQ-FORMS (ARGS)
   (ASSERT (EVENP (LENGTH ARGS)))
   (CASE (LENGTH ARGS)
       (0 NIL)
       (2 `((SETQ ,@ARGS)))
```

```
       (T `((PSETQ ,@ARGS)))))

(DEFUN QUOTED-FORM-P (FORM)
   (LET ((EXPANSION (MACROEXPAND FORM *ENVIRONMENT*)))
        (AND (CONSP EXPANSION)
             (EQ (FIRST EXPANSION)
                 'QUOTE))))

(DEFUN QUOTED-OBJECT (FORM)
   (LET ((EXPANSION (MACROEXPAND FORM *ENVIRONMENT*)))
        (DESTRUCTURING-BIND (QUOTE-SPECIAL-OPERATOR OBJECT)
               EXPANSION
               (ASSERT (EQ QUOTE-SPECIAL-OPERATOR 'QUOTE))
               OBJECT)))

(DEFUN REDUCE-REDUNDANT-CODE ()
   (WHEN (NULL (GETF *LOOP-COMPONENTS* :INITIALLY))
       (LET ((RHEAD (REVERSE (GETF *LOOP-COMPONENTS* :HEAD)))
             (RTAIL (REVERSE (GETF *LOOP-COMPONENTS* :TAIL)))
             (NECK NIL))
           (LOOP (WHEN (OR (NULL RHEAD)
                           (NULL RTAIL)
                           (NOT (EQUAL (CAR RHEAD)
                                       (CAR RTAIL))))
                       (RETURN))
                 (PUSH (POP RHEAD)
                       NECK)
                 (POP RTAIL))
           (SETF (GETF *LOOP-COMPONENTS* :HEAD)
                 (NREVERSE RHEAD)
                 (GETF *LOOP-COMPONENTS* :NECK)
                 NECK
                 (GETF *LOOP-COMPONENTS* :TAIL)
                 (NREVERSE RTAIL)))))

(DEFUN REPEAT-CLAUSE ()                                    ; Edited 2-Apr-2024 12:55 by lmm
   (LET ((FORM (FORM1)))
        (LP :FOR (GENSYM)
            :DOWNFROM FORM :TO 1)
        (CLAUSE*)))

(DEFUN RETURN-CLAUSE ()
   (LP :DO `(RETURN-FROM ,*LOOP-NAME* ,(FORM-OR-IT))))

(DEFUN SELECTABLE-CLAUSE ()
   (LET ((*CURRENT-KEYWORD* *CURRENT-KEYWORD*)
         (*CURRENT-CLAUSE* *CURRENT-CLAUSE*))
        (UNLESS (KEYWORD? '(:IF :WHEN :UNLESS :DO :DOING :RETURN :COLLECT :COLLECTING :APPEND :APPENDING
                                :NCONC :NCONCING :COUNT :COUNTING :SUM :SUMMING :MAXIMIZE :MAXIMIZING
                                :MINIMIZE :MINIMIZING))
                (LOOP-ERROR "A selectable-clause is missing."))
        (ECASE *CURRENT-KEYWORD*
            ((:IF :WHEN :UNLESS) (CONDITIONAL-CLAUSE))
            ((:DO :DOING) (DO-CLAUSE))
            ((:RETURN) (RETURN-CLAUSE))
            ((:COLLECT :COLLECTING :APPEND :APPENDING :NCONC :NCONCING :COUNT :COUNTING :SUM :SUMMING :MAXIMIZE
                    :MAXIMIZING :MINIMIZE :MINIMIZING) (ACCUMULATION-CLAUSE)))))

(DEFMACRO SIMPLE-LOOP (&REST COMPOUND-FORMS)
   (LET ((TOP (GENSYM)))
        `(BLOCK NIL
             (TAGBODY ,TOP ,@COMPOUND-FORMS (GO ,TOP)))))

(DEFUN SIMPLE-VAR-P (VAR)
   (AND (NOT (NULL VAR))
        (SYMBOLP VAR)))

(DEFUN SIMPLE-VAR1 ()
   (UNLESS (AND *LOOP-TOKENS* (SIMPLE-VAR-P (CAR *LOOP-TOKENS*)))
          (LOOP-ERROR "A simple variable name is missing."))
   (POP *LOOP-TOKENS*))

(DEFUN STRAY-OF-TYPE-ERROR ()
   (LOOP-ERROR "OF-TYPE keyword should be followed by a type spec."))

(DEFMACRO CL::SYMBOL-MACROLET (VARDEFS &BODY BODY)         ; Edited 24-Mar-2024 21:46 by lmm
```

```
   ;;
   `(PROGN ,@(IL:SUBPAIR (CONS 'SETQ (MAPCAR VARDEFS #'CAR))
                    (CONS 'SETF (MAPCAR VARDEFS #'CADR))
                    BODY)))


(DEFUN TYPE-SPEC? ()
   (LET ((TYPE T)
         (SUPPLIED-P NIL))
        (WHEN (OR (AND (PREPOSITION? :OF-TYPE)
                       (OR *LOOP-TOKENS* (STRAY-OF-TYPE-ERROR)))
                  (AND *LOOP-TOKENS* (MEMBER (CAR *LOOP-TOKENS*)
                                             '(FIXNUM FLOAT T NIL))))
            (SETQ TYPE (POP *LOOP-TOKENS*)
                  SUPPLIED-P T))
        (VALUES TYPE SUPPLIED-P)))


(DEFUN UNTIL-CLAUSE ()
   (LP :WHILE `(NOT ,(FORM1))))


(DEFUN USING-OTHER-VAR (KIND)
   (LET ((USING-PHRASE (WHEN (PREPOSITION? :USING)
                             (POP *LOOP-TOKENS*)))
         (OTHER-KEY-NAME (IF (FIND KIND '(:HASH-KEY :HASH-KEYS))
                             "HASH-VALUE"
                             "HASH-KEY")))
        (WHEN USING-PHRASE
            (DESTRUCTURING-BIND (OTHER-KEY OTHER-VAR)
                   USING-PHRASE
                   (UNLESS (STRING= OTHER-KEY OTHER-KEY-NAME)
                           (LOOP-ERROR "Keyword ~A is missing." OTHER-KEY-NAME))
                   OTHER-VAR))))


(DEFUN VARIABLE-CLAUSE* ()
   (LOOP (LET ((KEY (KEYWORD? '(:WITH :INITIALLY :FINALLY :FOR :AS))))
              (IF KEY
                  (CLAUSE1)
                  (RETURN)))))


(DEFUN WHILE-CLAUSE ()
   (LP :UNLESS (FORM1)
       :DO
       '(LOOP-FINISH)
       :END))


(DEFUN WITH (VAR &OPTIONAL (TYPE T)
                 &KEY
                 (= (DEFAULT-VALUE TYPE)))
   (FILL-IN :BINDING-FORMS `(,(LET-FORM `((,TYPE ,VAR ,=))))))


(DEFUN WITH-ACCUMULATORS (ACCUMULATOR-SPECS FORM)
   (IF (NULL ACCUMULATOR-SPECS)
       FORM
       (DESTRUCTURING-BIND (SPEC . REST)
              ACCUMULATOR-SPECS
              (ECASE (GETF (CDR SPEC)
                           :KIND)
                  (:LIST (WITH-LIST-ACCUMULATOR SPEC (WITH-ACCUMULATORS REST FORM)))
                  ((:TOTAL :LIMIT) (WITH-NUMERIC-ACCUMULATOR SPEC (WITH-ACCUMULATORS REST FORM)))))))


(DEFUN WITH-BINDING-FORMS (BINDING-FORMS FORM)
   (IF (NULL BINDING-FORMS)
       FORM
       (DESTRUCTURING-BIND (BINDING-FORM0 . REST)
              BINDING-FORMS
              (APPEND BINDING-FORM0 (LIST (WITH-BINDING-FORMS REST FORM))))))


(DEFUN WITH-CLAUSE ()
   (LET ((D-BINDINGS NIL))
        (LOOP (MULTIPLE-VALUE-BIND (VAR TYPE)
                   (D-VAR-TYPE-SPEC)
                 (LET ((REST (WHEN (PREPOSITION? :=)
                                   `(,(FORM1)))))
                      (APPENDF D-BINDINGS `((,TYPE ,VAR ,@REST)))))
              (UNLESS (PREPOSITION? :AND)
                      (RETURN)))
        (DESTRUCTURING-BIND (D-BINDING0 . REST)
               D-BINDINGS
```

```
              (IF (AND (NULL REST)
                       (CDDR D-BINDING0)
                       (DESTRUCTURING-BIND (TYPE VAR FORM)
                              D-BINDING0
                              (DECLARE (IGNORE TYPE))
                              (AND (CONSP VAR)
                                   (MULTIPLE-VALUE-LIST-FORM-P FORM))))
                  (APPLY #'DESTRUCTURING-MULTIPLE-VALUE-BIND D-BINDING0)
                  (LET ((BINDINGS (MAPAPPEND #'(LAMBDA (D-BINDING)
                                                 (APPLY #'BINDINGS D-BINDING))
                                             D-BINDINGS)))
                       (FILL-IN :BINDING-FORMS `(,(LET-FORM BINDINGS))))))))


(DEFUN WITH-ITERATOR-FORMS (ITERATOR-FORMS FORM)
   (IF (NULL ITERATOR-FORMS)
       FORM
       (DESTRUCTURING-BIND ((ITERATOR-MACRO SPEC) . REST)
              ITERATOR-FORMS
              `(,ITERATOR-MACRO ,SPEC ,(WITH-ITERATOR-FORMS REST FORM)))))


(DEFUN WITH-LIST-ACCUMULATOR (ACCUMULATOR-SPEC FORM)     ; Edited 8-Apr-2024 19:28 by lmm
   (DESTRUCTURING-BIND (NAME &KEY VAR SPLICE &ALLOW-OTHER-KEYS)
          ACCUMULATOR-SPEC
          (LET* ((ANONYMOUS-P (NULL NAME))
                 (LIST-VAR (IF (OR ANONYMOUS-P (GLOBALLY-SPECIAL-P VAR))
                               VAR
                               (GENSYM "LIST-")))
                 (VALUE-FORM (IF (AND (NOT ANONYMOUS-P)
                                      (GLOBALLY-SPECIAL-P VAR))
                                 NIL
                                 '(LIST NIL)))
                 (FORM (IF (AND (NOT ANONYMOUS-P)
                                (NOT (GLOBALLY-SPECIAL-P VAR)))
                           `(CL::SYMBOL-MACROLET ((,VAR (CDR ,LIST-VAR)))
                                 ,FORM)
                           FORM)))
                `(LET ((,LIST-VAR ,VALUE-FORM))
                      (DECLARE (TYPE LIST ,LIST-VAR))
                      (LET ((,SPLICE ,LIST-VAR))
                           (DECLARE (TYPE LIST ,SPLICE))
                           ,FORM))))))


(DEFMACRO WITH-LOOP-CONTEXT (TOKENS &BODY BODY)
   `(LET ((*LOOP-TOKENS* ,TOKENS)
          (*LOOP-NAME* NIL)
          (*CURRENT-KEYWORD* NIL)
          (*CURRENT-CLAUSE* NIL)
          (*LOOP-COMPONENTS* NIL)
          (*TEMPORARIES* NIL)
          (*IGNORABLE* NIL)
          (*ACCUMULATORS* NIL)
          (*ANONYMOUS-ACCUMULATOR* NIL)
          (*BOOLEAN-TERMINATOR* NIL)
          (*MESSAGE-PREFIX* "LOOP: "))
         ,@BODY))


(DEFUN WITH-NUMERIC-ACCUMULATOR (ACCUMULATOR-SPEC FORM)
   (DESTRUCTURING-BIND (NAME &KEY VAR TYPES &ALLOW-OTHER-KEYS)
          ACCUMULATOR-SPEC
          (LABELS ((TYPE-EQ (A B)
                        (AND (SUBTYPEP A B)
                             (SUBTYPEP B A))))
                 (WHEN (NULL TYPES)
                     (SETQ TYPES '(NUMBER)))
                 (DESTRUCTURING-BIND (TYPE0 . REST)
                        TYPES
                        (WHEN (AND REST (NOTEVERY #'(LAMBDA (TYPE)
                                                      (TYPE-EQ TYPE0 TYPE))
                                                  TYPES))
                            (WARN "Different types ~A are declared for ~A accumulator." (ENUMERATE TYPES)
                                  (OR NAME "the anonymous")))
                        (LET ((TYPE (IF REST
                                        `(OR ,TYPE0 ,@REST)
                                        TYPE0)))
                             `(LET ((,VAR ,(ZERO TYPE)))
                                   (DECLARE (TYPE ,TYPE ,VAR))
                                   ,FORM))))))


(DEFUN WITH-TEMPORARIES (TEMPORARY-SPECS FORM)           ; Edited 21-Mar-2024 11:50 by lmm
                                                         ; Edited 16-Mar-2024 14:22 by lmm
   (DESTRUCTURING-BIND (TEMPORARIES &KEY ((:IGNORABLE IGNORABLE)))
          TEMPORARY-SPECS
```

```
          (IF TEMPORARIES
              `(LET ,TEMPORARIES ,@(WHEN IGNORABLE
                                     `((DECLARE (IGNORABLE ,@IGNORABLE))))
                    ,FORM)
              FORM)))


(DEFUN ZERO (TYPE)
   (COND
      ((SUBTYPEP TYPE 'SHORT-FLOAT)
       0.0)
      ((SUBTYPEP TYPE 'SINGLE-FLOAT)
       0.0)
      ((SUBTYPEP TYPE 'DOUBLE-FLOAT)
       0.0)
      ((SUBTYPEP TYPE 'LONG-FLOAT)
       0.0)
      ((SUBTYPEP TYPE 'FLOAT)
       0.0)
      (T 0)))


(DEFMACRO LOOP (&REST FORMS)
   (IF (EVERY #'CONSP FORMS)
       `(SIMPLE-LOOP ,@FORMS)
       `(EXTENDED-LOOP ,@FORMS)))

(IL:PUTPROPS IL:XCL-LOOP IL:FILETYPE :COMPILE-FILE)

(IL:PUTPROPS IL:XCL-LOOP IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE (DEFPACKAGE "LOOP" (:USE "LISP"
                                                                                                    "XCL"))))

(IL:PUTPROPS IL:XCL-LOOP IL:COPYRIGHT (("Interlisp.org" 2004)
                                       ("Yuji Minejima <ggb01164@nifty.ne.jp>")
                                       2002 2004 2024))

(IL:PUTPROPS IL:XCL-LOOP IL:LICENSE "See COPYRIGHT and LICENSE in the repository
                                    ;; $Id: loop.lisp,v 1.38 2005/04/16 07:34:27 yuji Exp $
                                    ;;
                                    ;; Redistribution and use in source and binary forms, with or without
                                    ;; modification, are permitted provided that the following conditions
                                    ;; are met:
                                    ;;
                                    ;;  * Redistributions of source code must retain the above copyright
                                    ;;    notice, this list of conditions and the following disclaimer.
                                    ;;  * Redistributions in binary form must reproduce the above copyright
                                    ;;    notice, this list of conditions and the following disclaimer in
                                    ;;    the documentation and/or other materials provided with the
                                    ;;    distribution.
                                    ;;
                                    ;; THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
                                    ;; 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
                                    ;; LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
                                    ;; A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
                                    ;; OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
                                    ;; SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
                                    ;; LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
                                    ;; DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
                                    ;; THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
                                    ;; (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
                                    ;; OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.")
```

## FUNCTION INDEX



## VARIABLE INDEX



## MACRO INDEX



## PROPERTY INDEX

{MEDLEY}<sources>XCL-LOOP.;1

---

**FILE-ENVIRONMENT INDEX**



---

**STRUCTURE INDEX**



---