File created: 17-May-90 16:15:41 **{DSK}<usr>local>lde>lispcore>sources>WALKER.;2**

changes to: (IL:VARS IL:WALKERCOMS)

previous date: 13-Jul-88 17:37:52 {DSK}<usr>local>lde>lispcore>sources>WALKER.;1

Read Table: XCL

Package: XEROX-COMMON-LISP

Format: XCCS


```
(IL:RPAQQ IL:WALKERCOMS
          (;; A simple code walker.
           (IL:VARIABLES *DECLARATIONS* *ENVIRONMENT* *LEXICAL-VARIABLES* *WALK-FORM* *WALK-FUNCTION*
                  *WALK-COPY*)
           (IL:FUNCTIONS WALK-FORM WALK-FORM-INTERNAL WALK-TEMPLATE)
           (IL:COMS (IL:FUNCTIONS VARIABLE-GLOBALLY-SPECIAL-P VARIABLE-LEXICAL-P VARIABLE-LEXICALLY-BOUNDP
                            VARIABLE-SPECIAL-P)
                  (IL:DECLARE\: IL:DOCOPY IL:DONTEVAL@LOAD (IL:P (IL:MOVD 'VARIABLE-LEXICAL-P
                                                                         'IL:VARIABLE-LEXICAL-P)
                                                                 (IL:MOVD 'VARIABLE-SPECIAL-P
                                                                         'IL:VARIABLE-SPECIAL-P))))
           (IL:FUNCTIONS WALK-TEMPLATE-HANDLE-REPEAT WALK-TEMPLATE-HANDLE-REPEAT-1)
           (IL:FUNCTIONS RECONS RELIST* RELIST*-INTERNAL)
           (IL:FUNCTIONS WALK-ARGLIST WALK-BINDINGS-1 WALK-BINDINGS-2 WALK-COMPILER-LET WALK-DECLARATIONS
                  WALK-DO WALK-DO* WALK-DO/DO* WALK-FLET/LABELS WALK-LAMBDA WALK-LET WALK-LET* WALK-LET/LET*
                  WALK-MACROLET WALK-MULTIPLE-VALUE-BIND WALK-PROG WALK-PROG* WALK-TAGBODY WALK-TAGBODY-1
                  WALK-UNEXPECTED-DECLARE WITH-NEW-CONTOUR)
           (IL:FUNCTIONS MAKE-LEXICAL-ENVIRONMENT ADD-MACROLET-ENVIRONMENT ADD-LABELS/FLET-ENVIRONMENT
                  NOTE-DECLARATION NOTE-LEXICAL-BINDING)
           (IL:COMS (IL:DEFINE-TYPES WALKER-TEMPLATES)
                  (IL:FUNCTIONS DEFINE-WALKER-TEMPLATE GET-WALKER-TEMPLATE GET-WALKER-TEMPLATE-INTERNAL))
           ;; Templates for special forms
           (WALKER-TEMPLATES AND BLOCK CATCH COMPILER-LET COND DECLARE DO DO* EVAL-WHEN FLET FUNCTION GO IF
                  LABELS LAMBDA LET LET* MACROLET MULTIPLE-VALUE-BIND MULTIPLE-VALUE-CALL MULTIPLE-VALUE-PROG1
                  MULTIPLE-VALUE-SETQ OR PROG PROG* PROGN PROGV QUOTE RETURN-FROM SETQ TAGBODY THE THROW
                  UNWIND-PROTECT)
           ;; For Interlisp. Do not remove the template for IL:SETQ or the loadup may break.
           (WALKER-TEMPLATES IL:LOAD-TIME-EVAL IL:SETQ IL:RPAQ? IL:RPAQ IL:XNLSETQ IL:ERSETQ IL:NLSETQ
                  IL:RESETVARS)
           (IL:PROP (IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
                  IL:WALKER)))
```

;; A simple code walker.

(DEFVAR ***DECLARATIONS***

;; *declarations* is a list of the declarations currently in effect.

)

(DEFVAR ***ENVIRONMENT***

;; An environment of the kind that macroexpand-1 gets as its second argument. in fact that is exactly where it comes from. For more info see:
;; MAKE-LEXICAL-ENVIRONMENT

)

(DEFVAR ***LEXICAL-VARIABLES***

;; *lexical-variables* is a list of the variables bound in the current contour. In *lexical-variables* the cons whose car is the variable is meaningful in
;; the sense that the cons whose car is the variable can be used to keep track of which contour the variable is bound in.

)

(DEFVAR ***WALK-FORM***

;; *walk-form* is used by the IF template. When the first argument to the if template is a list it will be evaluated with *walk-form* bound to the form
;; currently being walked.

)

(DEFVAR ***WALK-FUNCTION***

;; *walk-function* is the function being called on each sub-form as we walk. Normally it is supplied using the :walk-function keyword argument to
;; walk-form, but it is OK to bind it around a call to walk-form-internal.

)

```
(DEFVAR *WALK-COPY*)

(DEFUN WALK-FORM (FORM &KEY ((:DECLARATIONS *DECLARATIONS*)
                             NIL)
                            ((:LEXICAL-VARIABLES *LEXICAL-VARIABLES*)
                             NIL)
                            ((:ENVIRONMENT *ENVIRONMENT*)
                             NIL)
                            ((:COPY *WALK-COPY*)
                             T)
                            ((:WALK-FUNCTION *WALK-FUNCTION*)
                             #'(LAMBDA (X Y)
                                       (DECLARE (IGNORE Y))
                                       X)))
   ;; The main entry-point is walk-form, calls back in should use walk-form-internal.

   ;; If :COPY is true (default), will return the expansion

   (LET ((RESULT (WALK-FORM-INTERNAL FORM ':EVAL)))
        (AND *WALK-COPY* RESULT)))

(DEFUN WALK-FORM-INTERNAL (FORM CONTEXT &AUX NEWFORM NEWNEWFORM WALK-NO-MORE-P MACROP FN TEMPLATE)
   ;; WALK-FORM-INTERNAL is the main driving function for the code walker. It takes a form and the current context and walks the form calling itself
   ;; or the appropriate template recursively.

   ;; It is recommended that a program-analyzing-program process a form  that is a list whose car is a symbol as follows:

   ;;  1. If the program has particular knowledge about the symbol,  process the form using special-purpose code.  All of the  standard special forms
   ;; should fall into this category.

   ;;  2. Otherwise, if macro-function is true of the symbol apply  either macroexpand or macroexpand-1 and start over.

   ;; 3. Otherwise, assume it is a function call.

   ;;  First apply the *walk-function* to perform whatever translation  the user wants to to this form.  If the second value returned  by *walk-function* is
   ;; T then we don't recurse...

   (MULTIPLE-VALUE-SETQ (NEWFORM WALK-NO-MORE-P)
          (FUNCALL *WALK-FUNCTION* FORM CONTEXT))
   (COND
      (WALK-NO-MORE-P NEWFORM)
      ((NOT (EQ FORM NEWFORM))
       (WALK-FORM-INTERNAL NEWFORM CONTEXT))
      ((NOT (CONSP NEWFORM))
       NEWFORM)
      ((SETQ TEMPLATE (GET-WALKER-TEMPLATE (SETQ FN (CAR NEWFORM))))
       (IF (SYMBOLP TEMPLATE)
           (FUNCALL TEMPLATE NEWFORM CONTEXT)
           (WALK-TEMPLATE NEWFORM TEMPLATE CONTEXT)))
      ((PROGN (MULTIPLE-VALUE-SETQ (NEWNEWFORM MACROP)
                    (MACROEXPAND-1 NEWFORM *ENVIRONMENT*))
              MACROP)
       (WALK-FORM-INTERNAL NEWNEWFORM CONTEXT))
      ((AND (SYMBOLP FN)
            (NOT (FBOUNDP FN))
            (SPECIAL-FORM-P FN))
       (ERROR "~S is a special form, not defined in the CommonLisp manual. Please define a template for this
              special form and try again." FN))
      (T ;; Otherwise, walk the form as if its just a standard function call using a template for standard function call.
         (WALK-TEMPLATE NEWFORM '(:CALL :REPEAT (:EVAL))
                CONTEXT))))

(DEFUN WALK-TEMPLATE (FORM TEMPLATE CONTEXT)
   (DECLARE (IL:GLOBALVARS IL:LAMBDASPLST))
   (IF (ATOM TEMPLATE)
       (ECASE TEMPLATE
           ((:QUOTE NIL) FORM)
           ((:EVAL :FUNCTION :TEST :EFFECT :RETURN) (WALK-FORM-INTERNAL FORM :EVAL))
           (:SET (WALK-FORM-INTERNAL FORM :SET))
           ((:LAMBDA :CALL) (COND
                               ((ATOM FORM)
                                FORM)
                               ((NOT (MEMBER (CAR FORM)
                                             IL:LAMBDASPLST :TEST 'EQ))
                                ;; Don't descend into things that aren't known LAMBDA-like forms.
                                FORM)
                               (T (WALK-LAMBDA FORM CONTEXT)))))
       (CASE (CAR TEMPLATE)
           (:IF
              ;; Conditional template
              (LET ((*WALK-FORM* FORM))
                   (WALK-TEMPLATE FORM (IF (IF (LISTP (SECOND TEMPLATE))
                                               (EVAL (SECOND TEMPLATE))
```

```
                                                (FUNCALL (SECOND TEMPLATE)
                                                         FORM))
                                               (THIRD TEMPLATE)
                                               (FOURTH TEMPLATE))
                                    CONTEXT)))
          (:REPEAT (WALK-TEMPLATE-HANDLE-REPEAT FORM (CDR TEMPLATE)
                                                ;; For the case where nothing happens after the repeat optimize out the call to length.
                                                (IF (NULL (CDDR TEMPLATE))
                                                    NIL
                                                    (NTHCDR (- (LENGTH FORM)
                                                               (LENGTH (CDDR TEMPLATE)))
                                                            FORM))
                                                CONTEXT))
          (:REMOTE (WALK-TEMPLATE FORM (CADR TEMPLATE)
                                  CONTEXT))
          (OTHERWISE (IF (ATOM FORM)
                         FORM
                         (RECONS FORM (WALK-TEMPLATE (CAR FORM)
                                                     (CAR TEMPLATE)
                                                     CONTEXT)
                                      (WALK-TEMPLATE (CDR FORM)
                                                     (CDR TEMPLATE)
                                                     CONTEXT)))))))
```

```
(DEFUN VARIABLE-GLOBALLY-SPECIAL-P (SYMBOL)
   ;; VARIABLE-GLOBALLY-SPECIAL-P is used to ask if a variable has been declared globally special. Any particular CommonLisp implementation
   ;; should customize this function accordingly and send their customization back. The default version of variable-globally-special-p is probably pretty
   ;; slow, so it uses *globally-special-variables* as a cache to remember variables that it has already figured out are globally special. This would need
   ;; to be reworked if an unspecial declaration got added to Common Lisp. Common Lisp nit: variable-globally-special-p should be defined in Common
   ;; Lisp.
   (IL:VARIABLE-GLOBALLY-SPECIAL-P SYMBOL))


(DEFUN VARIABLE-LEXICAL-P (VAR)
   (IF (NOT (BOUNDP '*WALK-FUNCTION*))
       :UNSURE
       (AND (NOT (EQ (VARIABLE-SPECIAL-P VAR)
                     'T))
            (MEMBER VAR *LEXICAL-VARIABLES* :TEST #'EQ))))


(DEFUN VARIABLE-LEXICALLY-BOUNDP (VAR)
   (IF (NOT (BOUNDP '*WALK-FUNCTION*))
       :UNSURE
       (VALUES (MEMBER VAR *LEXICAL-VARIABLES* :TEST #'EQ)
               (VARIABLE-SPECIAL-P VAR)
               'T)))


(DEFUN VARIABLE-SPECIAL-P (VAR)
   (IF (NOT (BOUNDP '*WALK-FUNCTION*))
       (OR (VARIABLE-GLOBALLY-SPECIAL-P VAR)
           :UNSURE)
       (OR (DOLIST (DECL *DECLARATIONS*)
             (AND (EQ (CAR DECL)
                      'SPECIAL)
                  (MEMBER VAR (CDR DECL)
                          :TEST
                          #'EQ)
                  (RETURN T)))
           (VARIABLE-GLOBALLY-SPECIAL-P VAR))))

(IL:DECLARE\: IL:DOCOPY IL:DONTEVAL@LOAD

(IL:MOVD 'VARIABLE-LEXICAL-P 'IL:VARIABLE-LEXICAL-P)

(IL:MOVD 'VARIABLE-SPECIAL-P 'IL:VARIABLE-SPECIAL-P)
)


(DEFUN WALK-TEMPLATE-HANDLE-REPEAT (FORM TEMPLATE STOP-FORM CONTEXT)
   (IF (EQ FORM STOP-FORM)
       (WALK-TEMPLATE FORM (CDR TEMPLATE)
              CONTEXT)
       (WALK-TEMPLATE-HANDLE-REPEAT-1 FORM TEMPLATE (CAR TEMPLATE)
              STOP-FORM CONTEXT)))


(DEFUN WALK-TEMPLATE-HANDLE-REPEAT-1 (FORM TEMPLATE REPEAT-TEMPLATE STOP-FORM CONTEXT)
   (COND
      ((NULL FORM)
       NIL)
      ((EQ FORM STOP-FORM)
       (IF (NULL REPEAT-TEMPLATE)
```

```
            (WALK-TEMPLATE STOP-FORM (CDR TEMPLATE)
                  CONTEXT)
            (ERROR "While handling repeat: Ran into stop while still in repeat template.")))
       ((NULL REPEAT-TEMPLATE)
        (WALK-TEMPLATE-HANDLE-REPEAT-1 FORM TEMPLATE (CAR TEMPLATE)
            STOP-FORM CONTEXT))
       (T (RECONS FORM (WALK-TEMPLATE (CAR FORM)
                              (CAR REPEAT-TEMPLATE)
                              CONTEXT)
                  (WALK-TEMPLATE-HANDLE-REPEAT-1 (CDR FORM)
                         TEMPLATE
                         (CDR REPEAT-TEMPLATE)
                         STOP-FORM CONTEXT)))))


(DEFUN RECONS (X CAR CDR)
   (IF *WALK-COPY*
       (IF (OR (NOT (EQ (CAR X)
                        CAR))
               (NOT (EQ (CDR X)
                        CDR)))
           (CONS CAR CDR)
           X)))


(DEFUN RELIST* (X &REST ARGS)
   (IF *WALK-COPY* (RELIST*-INTERNAL X ARGS)))


(DEFUN RELIST*-INTERNAL (X ARGS)
   (IF (NULL (CDR ARGS))
       (CAR ARGS)
       (RECONS X (CAR ARGS)
              (RELIST*-INTERNAL (CDR X)
                     (CDR ARGS)))))


(DEFUN WALK-ARGLIST (ARGLIST CONTEXT &OPTIONAL (DESTRUCTURINGP NIL)
                             &AUX ARG)
   (COND
      ((NULL ARGLIST)
       NIL)
      ((SYMBOLP (SETQ ARG (CAR ARGLIST)))
       (OR (MEMBER ARG LAMBDA-LIST-KEYWORDS :TEST #'EQ)
           (NOTE-LEXICAL-BINDING ARG))
       (RECONS ARGLIST ARG (WALK-ARGLIST (CDR ARGLIST)
                                  CONTEXT
                                  (AND DESTRUCTURINGP (NOT (MEMBER ARG LAMBDA-LIST-KEYWORDS :TEST #'EQ))))))
      ((CONSP ARG)
       (PROG1 (IF DESTRUCTURINGP
                  (WALK-ARGLIST ARG CONTEXT DESTRUCTURINGP)
                  (RECONS ARGLIST (RELIST* ARG (CAR ARG)
                                           (WALK-FORM-INTERNAL (CADR ARG)
                                                  ':EVAL)
                                           (CDDR ARG))
                          (WALK-ARGLIST (CDR ARGLIST)
                                 CONTEXT NIL)))
           (IF (SYMBOLP (CAR ARG))
               (NOTE-LEXICAL-BINDING (CAR ARG))
               (NOTE-LEXICAL-BINDING (CADAR ARG)))
           (OR (NULL (CDDR ARG))
               (NOT (SYMBOLP (CADDR ARG)))
               (NOTE-LEXICAL-BINDING ARG))))
      (T (ERROR "Can't understand something in the arglist ~S" ARGLIST))))


(DEFUN WALK-BINDINGS-1 (BINDINGS OLD-DECLARATIONS OLD-LEXICAL-VARIABLES CONTEXT SEQUENTIALP)
   (AND BINDINGS (LET ((BINDING (CAR BINDINGS)))
                      (RECONS BINDINGS (IF (SYMBOLP BINDING)
                                           (PROG1 BINDING (NOTE-LEXICAL-BINDING BINDING))
                                           (PROG1 (LET ((*DECLARATIONS* OLD-DECLARATIONS)
                                                        (*LEXICAL-VARIABLES* (IF SEQUENTIALP
                                                                                 *LEXICAL-VARIABLES*
                                                                                 OLD-LEXICAL-VARIABLES)))
                                                    (RELIST* BINDING (CAR BINDING)
                                                           (WALK-FORM-INTERNAL (CADR BINDING)
                                                                  CONTEXT)
                                                           (CDDR BINDING)))
                                                  ; save cddr for DO/DO* it is the next value; form. Don't walk it
                                                  ; now though.
                                                  (NOTE-LEXICAL-BINDING (CAR BINDING))))
                              (WALK-BINDINGS-1 (CDR BINDINGS)
                                     OLD-DECLARATIONS OLD-LEXICAL-VARIABLES CONTEXT SEQUENTIALP)))))


(DEFUN WALK-BINDINGS-2 (BINDINGS WALKED-BINDINGS CONTEXT)
   (AND BINDINGS (LET ((BINDING (CAR BINDINGS))
```

```
                    (WALKED-BINDING (CAR WALKED-BINDINGS)))
                   (RECONS BINDINGS (IF (SYMBOLP BINDING)
                                        BINDING
                                        (RELIST* BINDING (CAR WALKED-BINDING)
                                                 (CADR WALKED-BINDING)
                                                 (WALK-TEMPLATE (CDDR BINDING)
                                                                '(:EVAL)
                                                                CONTEXT)))
                           (WALK-BINDINGS-2 (CDR BINDINGS)
                                  (CDR WALKED-BINDINGS)
                                  CONTEXT)))))
```

```
(DEFUN WALK-COMPILER-LET (FORM CONTEXT)
   (WITH-NEW-CONTOUR (LET ((VARS NIL)
                           (VALS NIL))
                       (DOLIST (BINDING (CADR FORM))
                           (COND
                              ((SYMBOLP BINDING)
                               (PUSH BINDING VARS)
                               (PUSH NIL VALS))
                              (T (PUSH (CAR BINDING)
                                       VARS)
                                 (PUSH (EVAL (CADR BINDING))
                                       VALS))))
                       (RELIST* FORM (CAR FORM)
                              (CADR FORM)
                              (PROGV VARS VALS
                                  (NOTE-DECLARATION (CONS 'SPECIAL VARS))
                                  (WALK-TEMPLATE (CDDR FORM)
                                         '(:REPEAT (:EVAL))
                                         CONTEXT))))))
```

```
(DEFUN WALK-DECLARATIONS (BODY FN &OPTIONAL DOC-STRING-P DECLARATIONS OLD-BODY &AUX (FORM (CAR BODY))
                                  MACROP NEW-FORM)
   (COND
      ((AND (STRINGP FORM)                                          ; might be a doc string
            (CDR BODY)                                              ; isn't the returned value
            (NULL DOC-STRING-P)                                     ; no doc string yet
            (NULL DECLARATIONS))                                    ; no declarations yet
       (RECONS BODY FORM (WALK-DECLARATIONS (CDR BODY)
                                 FN T)))
      ((AND (LISTP FORM)
            (EQ (CAR FORM)
                'DECLARE))
       ;; Got ourselves a real live declaration.  Record it, look for more.
       (DOLIST (DECLARATION (CDR FORM))
           (NOTE-DECLARATION DECLARATION)
           (PUSH DECLARATION DECLARATIONS))
       (RECONS BODY FORM (WALK-DECLARATIONS (CDR BODY)
                                 FN DOC-STRING-P DECLARATIONS)))
      ((AND FORM (LISTP FORM)
            (NULL (GET-WALKER-TEMPLATE (CAR FORM)))
            (PROGN (MULTIPLE-VALUE-SETQ (NEW-FORM MACROP)
                         (MACROEXPAND-1 (CAR FORM)
                                 *ENVIRONMENT*))
                   MACROP))
       ;; This form was a call to a macro.  Maybe it expanded into a declare?  Recurse to find out.
       (WALK-DECLARATIONS (RECONS BODY NEW-FORM (CDR BODY))
              FN DOC-STRING-P DECLARATIONS (OR OLD-BODY BODY)))
      (T ;; Now that we have walked and recorded the declarations, call the function our caller provided to expand the body.  We call that function
         ;; rather than passing the real-body back, because we are RECONSING up the new body.
         (FUNCALL FN (OR OLD-BODY BODY)))))
```

```
(DEFUN WALK-DO (FORM CONTEXT)
   (WALK-DO/DO* FORM CONTEXT NIL))
```

```
(DEFUN WALK-DO* (FORM CONTEXT)
   (WALK-DO/DO* FORM CONTEXT T))
```

```
(DEFUN WALK-DO/DO* (FORM CONTEXT SEQUENTIALP)
   (LET ((OLD-DECLARATIONS *DECLARATIONS*)
         (OLD-LEXICAL-VARIABLES *LEXICAL-VARIABLES*))
       (WITH-NEW-CONTOUR (LET* ((DO/DO* (CAR FORM))
                                (BINDINGS (CADR FORM))
                                (END-TEST (CADDR FORM))
                                (BODY (CDDDR FORM))
                                WALKED-BINDINGS
                                (WALKED-BODY (WALK-DECLARATIONS BODY
                                                      #'(LAMBDA (REAL-BODY)
```

```
                                                    (SETQ WALKED-BINDINGS
                                                          (WALK-BINDINGS-1 BINDINGS OLD-DECLARATIONS
                                                                 OLD-LEXICAL-VARIABLES CONTEXT
                                                                 SEQUENTIALP))
                                                    (WALK-TEMPLATE REAL-BODY '(:REPEAT (:EVAL))
                                                           CONTEXT)))))
                                  (RELIST* FORM DO/DO* (WALK-BINDINGS-2 BINDINGS WALKED-BINDINGS CONTEXT)
                                         (WALK-TEMPLATE END-TEST '(:TEST :REPEAT (:EVAL))
                                                CONTEXT)
                                         WALKED-BODY)))))


(DEFUN WALK-FLET/LABELS (FORM CONTEXT)
   (WITH-NEW-CONTOUR (LABELS ((WALK-DEFINITIONS (DEFINITIONS)
                                (IF (NULL DEFINITIONS)
                                    NIL
                                    (RECONS DEFINITIONS (WALK-LAMBDA (CAR DEFINITIONS)
                                                               CONTEXT)
                                           (WALK-DEFINITIONS (CDR DEFINITIONS)))))
                                                    ;
                              (UPDATE-ENVIRONMENT NIL (SETQ *ENVIRONMENT* (MAKE-LEXICAL-ENVIRONMENT FORM
                                                                    *ENVIRONMENT*))))
                             (RELIST* FORM (CAR FORM)
                                    (ECASE (CAR FORM)
                                        (FLET (PROG1 (WALK-DEFINITIONS (CADR FORM))
                                                     (UPDATE-ENVIRONMENT)))
                                        (LABELS
                                           (UPDATE-ENVIRONMENT)
                                           (WALK-DEFINITIONS (CADR FORM))))
                                        (WALK-DECLARATIONS (CDDR FORM)
                                               #'(LAMBDA (REAL-BODY)
                                                         (WALK-TEMPLATE REAL-BODY '(:REPEAT (:EVAL))
                                                                CONTEXT)))))))


(DEFUN WALK-LAMBDA (FORM CONTEXT)
   (WITH-NEW-CONTOUR (LET* ((ARGLIST (CADR FORM))
                            (BODY (CDDR FORM))
                            (WALKED-ARGLIST NIL)
                            (WALKED-BODY (WALK-DECLARATIONS BODY #'(LAMBDA (REAL-BODY)
                                                                     (SETQ WALKED-ARGLIST
                                                                           (WALK-ARGLIST ARGLIST
                                                                                  CONTEXT))
                                                                     (WALK-TEMPLATE
                                                                      REAL-BODY
                                                                      '(:REPEAT (:EVAL))
                                                                      CONTEXT)))))
                           (RELIST* FORM (CAR FORM)
                                  WALKED-ARGLIST WALKED-BODY))))


(DEFUN WALK-LET (FORM CONTEXT)
   (WALK-LET/LET* FORM CONTEXT NIL))


(DEFUN WALK-LET* (FORM CONTEXT)
   (WALK-LET/LET* FORM CONTEXT T))


(DEFUN WALK-LET/LET* (FORM CONTEXT SEQUENTIALP)
   (LET ((OLD-DECLARATIONS *DECLARATIONS*)
         (OLD-LEXICAL-VARIABLES *LEXICAL-VARIABLES*))
        (WITH-NEW-CONTOUR (LET* ((LET/LET* (CAR FORM))
                                 (BINDINGS (CADR FORM))
                                 (BODY (CDDR FORM))
                                 WALKED-BINDINGS
                                 (WALKED-BODY (WALK-DECLARATIONS BODY
                                                    #'(LAMBDA (REAL-BODY)
                                                              (SETQ WALKED-BINDINGS
                                                                    (WALK-BINDINGS-1 BINDINGS OLD-DECLARATIONS
                                                                           OLD-LEXICAL-VARIABLES CONTEXT
                                                                           SEQUENTIALP))
                                                              (WALK-TEMPLATE REAL-BODY '(:REPEAT (:EVAL))
                                                                     CONTEXT)))))
                                (RELIST* FORM LET/LET* WALKED-BINDINGS WALKED-BODY)))))


(DEFUN WALK-MACROLET (FORM CONTEXT)
   (LABELS ((WALK-DEFINITIONS
             (DEFINITIONS)
             (AND (NOT (NULL DEFINITIONS))
                  (LET ((DEFINITION (CAR DEFINITIONS)))
                       (RECONS DEFINITIONS (WITH-NEW-CONTOUR
                                            (RELIST* DEFINITION (CAR DEFINITION)
                                                   (WALK-ARGLIST (CADR DEFINITION)
                                                          CONTEXT T)
                                                   (WALK-DECLARATIONS
```

```
                                                    (CDDR DEFINITION)
                                                    #'(LAMBDA (REAL-BODY)
                                                              (WALK-TEMPLATE REAL-BODY '(:REPEAT (:EVAL))
                                                                    CONTEXT)))))
                                  (WALK-DEFINITIONS (CDR DEFINITIONS)))))))
          (WITH-NEW-CONTOUR (RELIST* FORM (CAR FORM)
                                     (WALK-DEFINITIONS (CADR FORM))
                                     (PROGN (SETQ *ENVIRONMENT* (MAKE-LEXICAL-ENVIRONMENT FORM *ENVIRONMENT*))
                                            (WALK-DECLARATIONS (CDDR FORM)
                                                   #'(LAMBDA (REAL-BODY)
                                                             (WALK-TEMPLATE REAL-BODY '(:REPEAT (:EVAL))
                                                                   CONTEXT)))))))))


(DEFUN WALK-MULTIPLE-VALUE-BIND (FORM CONTEXT)
   (LET ((OLD-DECLARATIONS *DECLARATIONS*)
         (OLD-LEXICAL-VARIABLES *LEXICAL-VARIABLES*))
        (WITH-NEW-CONTOUR (LET* ((MVB (CAR FORM))
                                 (BINDINGS (CADR FORM))
                                 (MV-FORM (WALK-TEMPLATE (CADDR FORM)
                                                         ':EVAL CONTEXT))
                                 (BODY (CDDDR FORM))
                                 WALKED-BINDINGS
                                 (WALKED-BODY (WALK-DECLARATIONS BODY
                                                    #'(LAMBDA (REAL-BODY)
                                                              (SETQ WALKED-BINDINGS
                                                                    (WALK-BINDINGS-1 BINDINGS OLD-DECLARATIONS
                                                                           OLD-LEXICAL-VARIABLES CONTEXT NIL))
                                                              (WALK-TEMPLATE REAL-BODY '(:REPEAT (:EVAL))
                                                                     CONTEXT)))))
                                (RELIST* FORM MVB WALKED-BINDINGS MV-FORM WALKED-BODY)))))


(DEFUN WALK-PROG (FORM CONTEXT)
   (WALK-LET/LET* FORM CONTEXT NIL))


(DEFUN WALK-PROG* (FORM CONTEXT)
   (WALK-LET/LET* FORM CONTEXT T))


(DEFUN WALK-TAGBODY (FORM CONTEXT)
   (RECONS FORM (CAR FORM)
           (WALK-TAGBODY-1 (CDR FORM)
                  CONTEXT)))


(DEFUN WALK-TAGBODY-1 (FORM CONTEXT)
   (AND FORM (RECONS FORM (WALK-FORM-INTERNAL (CAR FORM)
                                    (IF (SYMBOLP (CAR FORM))
                                        ':QUOTE
                                        CONTEXT))
                     (WALK-TAGBODY-1 (CDR FORM)
                            CONTEXT))))


(DEFUN WALK-UNEXPECTED-DECLARE (FORM CONTEXT)
   (DECLARE (IGNORE CONTEXT))
   (WARN "Encountered declare ~S in a place where a declare was not expected." FORM)
   FORM)


(DEFMACRO WITH-NEW-CONTOUR (&BODY BODY)

   ;; With new contour is used to enter a new lexical binding contour which inherits from the exisiting one. I admit that using with-new-contour is often
   ;; overkill. It would suffice for the the walker to rebind *lexical-variables* and *declarations* when walking LET and rebind *environment* and
   ;; *declarations* when walking MACROLET etc.WITH-NEW-CONTOUR is much more convenient and just as correct.

   `(LET ((*DECLARATIONS* NIL)
          (*LEXICAL-VARIABLES* *LEXICAL-VARIABLES*)
          (*ENVIRONMENT* *ENVIRONMENT*))
         IL:\\\, BODY))


(DEFUN MAKE-LEXICAL-ENVIRONMENT (MACROLET/FLET/LABELS-FORM ENVIRONMENT)

   ;; make-lexical-environemnt is kind of gross. It would be less gross if EVAL took an environment argument.

   (ECASE (CAR MACROLET/FLET/LABELS-FORM)
       (MACROLET (ADD-MACROLET-ENVIRONMENT MACROLET/FLET/LABELS-FORM ENVIRONMENT))
       ((FLET LABELS) (ADD-LABELS/FLET-ENVIRONMENT MACROLET/FLET/LABELS-FORM ENVIRONMENT))))


(DEFUN ADD-MACROLET-ENVIRONMENT (MACROLET-FORM ENV)
   (DESTRUCTURING-BIND (CAR-OF-FORM LOCAL-MACROS &REST BODY)
          MACROLET-FORM
          (COND
             ((TYPEP ENV 'COMPILER:ENV)

              ;; From the compiler
```

```
             (LET ((NEW-ENV (COMPILER::MAKE-CHILD-ENV ENV)))
                  (DOLIST (MACRO-DEFN LOCAL-MACROS)
                      (COMPILER::ENV-BIND-FUNCTION NEW-ENV (CAR MACRO-DEFN)
                             :MACRO
                             (COMPILER::CRACK-DEFMACRO (CONS 'DEFMACRO MACRO-DEFN))))
                  NEW-ENV))
            ((OR (TYPEP ENV 'IL:ENVIRONMENT)
                 (NULL ENV))
             ;; from the interpreter
             (LET ((NEW-ENV (IL:\\MAKE-CHILD-ENVIRONMENT ENV)))
                  (SETF (IL:ENVIRONMENT-FUNCTIONS NEW-ENV)
                        (NCONC (WITH-COLLECTION (DOLIST (MACRO-DEFN LOCAL-MACROS)
                                                        (COLLECT (CAR MACRO-DEFN))
                                                        (COLLECT (CONS :MACRO (COMPILER::CRACK-DEFMACRO
                                                                                (CONS 'DEFMACRO MACRO-DEFN))))))
                               (IL:ENVIRONMENT-FUNCTIONS NEW-ENV)))
                  NEW-ENV))
            (T (ERROR "Not a recognized environment type: ~s" ENV)))))

(DEFUN ADD-LABELS/FLET-ENVIRONMENT (LABELS/FLET-FORM ENV)
   (DESTRUCTURING-BIND (CAR-OF-FORM LOCAL-FNS &REST BODY)
          LABELS/FLET-FORM
          (COND
             ((TYPEP ENV 'COMPILER:ENV)
              ;; From the compiler
              (LET ((NEW-ENV (COMPILER::MAKE-CHILD-ENV ENV)))
                   (DOLIST (FN-DEFN LOCAL-FNS)
                       (COMPILER::ENV-BIND-FUNCTION NEW-ENV (CAR FN-DEFN)
                              :FUNCTION
                              (CONS 'LAMBDA (CDR FN-DEFN))))
                   NEW-ENV))
             ((OR (TYPEP ENV 'IL:ENVIRONMENT)
                  (NULL ENV))
              ;; from the interpreter
              (LET ((NEW-ENV (IL:\\MAKE-CHILD-ENVIRONMENT ENV)))
                   (SETF (IL:ENVIRONMENT-FUNCTIONS NEW-ENV)
                         (NCONC (WITH-COLLECTION (DOLIST (FN-DEFN LOCAL-FNS)
                                                         (COLLECT (CAR FN-DEFN))
                                                         (COLLECT (CONS :FUNCTION (IL:MAKE-CLOSURE
                                                                                    :FUNCTION
                                                                                    (CONS 'LAMBDA (CDR FN-DEFN))
                                                                                    :ENVIRONMENT ENV)))))
                                (IL:ENVIRONMENT-FUNCTIONS NEW-ENV)))
                   NEW-ENV))
             (T (ERROR "Not a recognized environment type: ~s" ENV)))))

(DEFMACRO NOTE-DECLARATION (DECLARATION)
   `(PUSH ,DECLARATION *DECLARATIONS*))

(DEFMACRO NOTE-LEXICAL-BINDING (THING)
   `(PUSH ,THING *LEXICAL-VARIABLES*))

(DEF-DEFINE-TYPE WALKER-TEMPLATES "Walker templates")

(DEFDEFINER DEFINE-WALKER-TEMPLATE WALKER-TEMPLATES (NAME TEMPLATE)
   `(EVAL-WHEN (LOAD EVAL)
           (SETF (GET-WALKER-TEMPLATE-INTERNAL ',NAME)
                 ',TEMPLATE)))

(DEFUN GET-WALKER-TEMPLATE (X)
   (COND
      ((SYMBOLP X)
       (GET-WALKER-TEMPLATE-INTERNAL X))
      ((AND (LISTP X)
            (EQ (CAR X)
                'LAMBDA))
       '(:LAMBDA :REPEAT (:EVAL)))))

(DEFMACRO GET-WALKER-TEMPLATE-INTERNAL (X)
   `(GET ,X 'WALKER-TEMPLATES))

;; Templates for special forms

(DEFINE-WALKER-TEMPLATE AND (NIL :REPEAT (:EVAL)))
```

```
(DEFINE-WALKER-TEMPLATE BLOCK (NIL NIL :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE CATCH (NIL :EVAL :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE COMPILER-LET WALK-COMPILER-LET)

(DEFINE-WALKER-TEMPLATE COND (NIL :REPEAT ((:TEST :REPEAT (:EVAL)))))

(DEFINE-WALKER-TEMPLATE DECLARE WALK-UNEXPECTED-DECLARE)

(DEFINE-WALKER-TEMPLATE DO WALK-DO)

(DEFINE-WALKER-TEMPLATE DO* WALK-DO*)

(DEFINE-WALKER-TEMPLATE EVAL-WHEN (NIL :QUOTE :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE FLET WALK-FLET/LABELS)

(DEFINE-WALKER-TEMPLATE FUNCTION (NIL :CALL))

(DEFINE-WALKER-TEMPLATE GO (NIL :QUOTE))

(DEFINE-WALKER-TEMPLATE IF (NIL :TEST :RETURN :RETURN))

(DEFINE-WALKER-TEMPLATE LABELS WALK-FLET/LABELS)

(DEFINE-WALKER-TEMPLATE LAMBDA WALK-LAMBDA)

(DEFINE-WALKER-TEMPLATE LET WALK-LET)

(DEFINE-WALKER-TEMPLATE LET* WALK-LET*)

(DEFINE-WALKER-TEMPLATE MACROLET WALK-MACROLET)

(DEFINE-WALKER-TEMPLATE MULTIPLE-VALUE-BIND WALK-MULTIPLE-VALUE-BIND)

(DEFINE-WALKER-TEMPLATE MULTIPLE-VALUE-CALL (NIL :EVAL :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE MULTIPLE-VALUE-PROG1 (NIL :RETURN :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE MULTIPLE-VALUE-SETQ (NIL (:REPEAT (:SET))
                                                 :EVAL))

(DEFINE-WALKER-TEMPLATE OR (NIL :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE PROG WALK-PROG)

(DEFINE-WALKER-TEMPLATE PROG* WALK-PROG*)

(DEFINE-WALKER-TEMPLATE PROGN (NIL :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE PROGV (NIL :EVAL :EVAL :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE QUOTE (NIL :QUOTE))

(DEFINE-WALKER-TEMPLATE RETURN-FROM (NIL :QUOTE :REPEAT (:RETURN)))

(DEFINE-WALKER-TEMPLATE SETQ (NIL :REPEAT (:SET :EVAL)))
```

```
(DEFINE-WALKER-TEMPLATE TAGBODY WALK-TAGBODY)

(DEFINE-WALKER-TEMPLATE THE (NIL :QUOTE :EVAL))

(DEFINE-WALKER-TEMPLATE THROW (NIL :EVAL :EVAL))

(DEFINE-WALKER-TEMPLATE UNWIND-PROTECT (NIL :RETURN :REPEAT (:EVAL)))

;; For Interlisp. Do not remove the template for IL:SETQ or the loadup may break.

(DEFINE-WALKER-TEMPLATE IL:LOAD-TIME-EVAL (NIL :EVAL))

(DEFINE-WALKER-TEMPLATE IL:SETQ (NIL :SET :EVAL))

(DEFINE-WALKER-TEMPLATE IL:RPAQ? (NIL :SET :EVAL))

(DEFINE-WALKER-TEMPLATE IL:RPAQ (NIL :SET :EVAL))

(DEFINE-WALKER-TEMPLATE IL:XNLSETQ (NIL :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE IL:ERSETQ (NIL :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE IL:NLSETQ (NIL :REPEAT (:EVAL)))

(DEFINE-WALKER-TEMPLATE IL:RESETVARS WALK-LET)
(IL:PUTPROPS IL:WALKER IL:FILETYPE :COMPILE-FILE)
(IL:PUTPROPS IL:WALKER IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "XCL"))
(IL:PUTPROPS IL:WALKER IL:COPYRIGHT ("Venue & Xerox Corporation" 1987 1988 1990))
```

FUNCTION INDEX

WALKER-TEMPLATE INDEX

VARIABLE INDEX

MACRO INDEX

PROPERTY INDEX

DEFINER INDEX

DEFINE-TYPE INDEX