

File created: 19-Oct-93 10:48:08 {Pele:mv:envos}<LispCore>Sources>CLTL2>CMLFLOATARRAY.;1

previous date: 11-Jun-90 14:41:02 {Pele:mv:envos}<LispCore>Sources>CMLFLOATARRAY.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

;;  
;; Copyright (c) 1985, 1986, 1987, 1990, 1993 by Venue & Xerox Corporation. All rights reserved.

### (RPAQQ CMLFLOATARRAYCOMS

```
[ (DECLARE%: DONTCOPY DOEVAL@COMPILE (FILES (SYSLOAD FROM VALUEOF DIRECTORIES)
                                             UNBOXEDOPS FLOAT-ARRAY-SUPPORT))
```

;; MAPARRAY fns and macros

```
(FNS MAP-ARRAY)
(FUNCTIONS MAP-ARRAY-1 MAP-ARRAY-2)
(FUNCTIONS REDUCE-ARRAY EVALUATE-POLYNOMIAL FIND-ARRAY-ELEMENT-INDEX)
(FUNCTIONS FLATTEN-ARG MAX-ABS MIN-ABS)
(FUNCTIONS %%MAP-FLOAT-ARRAY-ABS %%MAP-FLOAT-ARRAY-FLOAT %%MAP-FLOAT-ARRAY-MINUS
           %%MAP-FLOAT-ARRAY-NEGATE %%MAP-FLOAT-ARRAY-PLUS %%MAP-FLOAT-ARRAY-QUOTIENT
           %%MAP-FLOAT-ARRAY-TIMES %%MAP-FLOAT-ARRAY-TRUNCATE %%REDUCE-FLOAT-ARRAY-MAX
           %%REDUCE-FLOAT-ARRAY-MAX-ABS %%REDUCE-FLOAT-ARRAY-MIN %%REDUCE-FLOAT-ARRAY-MIN-ABS
           %%REDUCE-FLOAT-ARRAY-PLUS %%REDUCE-FLOAT-ARRAY-TIMES)
```

;; For convenience

```
(PROP FILETYPE CMLFLOATARRAY)
(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY (LOCALVARS . T))
(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVERS (ADDVARS (NLAMA)
                                                                    (NLAML)
                                                                    (LAMA MAP-ARRAY]))
```

```
(DECLARE%: DONTCOPY DOEVAL@COMPILE
```

```
(FILESLOAD (SYSLOAD FROM VALUEOF DIRECTORIES)
            UNBOXEDOPS FLOAT-ARRAY-SUPPORT)
)
```

;; MAPARRAY fns and macros

```
(DEFINEQ
```

### (MAP-ARRAY

```
[LAMBDA ARGS
```

; Edited 9-Apr-87 16:22 by jop

;; First arg, RESULT, may either be an array of the correct type, or a symbol indicating the element-type of the result, or NIL if the map is for effect.  
;; Second arg is the mapping functions. Other args are arrays, all of which must have the same number of elements, or non-arrays which will be  
;; treated as scalars

```
(CL:IF (< ARGS 3)
      (CL:ERROR "MAPARRAY takes at least three args"))
(LET ((RESULT (ARG ARGS 1))
      (MAPFN (ARG ARGS 2))
      (ARRAY1 (ARG ARGS 3))
      (FIRST-ARRAY))
```

;; Arg checking. First-array is the first array map argument

```
(CL:IF (NOT (TYPEP MAPFN 'CL:FUNCTION))
      (CL:ERROR "Not a function: ~S" MAPFN))
(CL:DO ((I 3 (CL:1+ I))
      (MAP-ARG)
      (> I ARGS))
      (SETQ MAP-ARG (ARG ARGS I))
      (CL:WHEN (CL:ARRAYP MAP-ARG)
        (CL:IF FIRST-ARRAY
          (CL:IF (NOT (EQUAL-DIMENSIONS-P MAP-ARG FIRST-ARRAY))
                (CL:ERROR "Dimensions mismatch" MAP-ARG))
          (SETQ FIRST-ARRAY MAP-ARG))))
```

;; Coerce RESULT into an array or NIL

```
(CL:TYPECASE RESULT
  (CL:ARRAY (CL:IF [NOT (OR (EQUAL-DIMENSIONS-P RESULT FIRST-ARRAY)
                          (AND (NULL FIRST-ARRAY)
                               (EQ 0 (CL:ARRAY-RANK RESULT))
                               (CL:ERROR "Dimensions mismatch: ~S" RESULT)))]
                ((OR CL:SYMBOL CONS) (SETQ RESULT (CL:IF FIRST-ARRAY
                                                         (CL:MAKE-ARRAY (CL:ARRAY-DIMENSIONS FIRST-ARRAY)
                                                         :ELEMENT-TYPE RESULT)
                                                         (CL:MAKE-ARRAY NIL :ELEMENT-TYPE RESULT))))
              (T (OR (NULL RESULT)
                    (CL:ERROR "RESULT must be an array, an element type, or NIL: ~S" RESULT))))
  (CL:IF FIRST-ARRAY
    (CL:IF (AND RESULT (< ARGS 5))
      (CL:ECASE ARGS
```

```

(3 ; Note: in this case (EQ ARRAY1 FIRST-ARRAY)
  (MAP-ARRAY-1 RESULT MAPFN ARRAY1))
(4 (MAP-ARRAY-2 RESULT MAPFN ARRAY1 (ARG ARGS 4)))
[LET* ((FLATTENED-RESULT (FLATTEN-ARG RESULT))
      (SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
      [FLATTENED-ARRAYS (for I from 3 to ARGS collect (FLATTEN-ARG (ARG ARGS I]
      (ELT-SLICE (CL:COPY-LIST FLATTENED-ARRAYS))
      VALUE)
      (CL:DOTIMES (INDEX SIZE RESULT)
        [SETQ VALUE (CL:APPLY MAPFN (CL:DO ((%SUBSLICE ELT-SLICE (CDR %SUBSLICE))
      (%SUBARRAYS FLATTENED-ARRAYS (CDR %SUBARRAYS)))
      ((NULL %SUBARRAYS)
      (ELT-SLICE)
      (AND (CL:ARRAYP (CAR %SUBARRAYS))
      (RPLACA %SUBSLICE (CL:AREF (CAR %SUBARRAYS)
      INDEX))))])
      (CL:IF RESULT
      (CL:SETF (CL:AREF FLATTENED-RESULT INDEX)
      VALUE)))]
      (CL:IF RESULT
      [CL:SETF (CL:AREF RESULT)
      (CL:APPLY MAPFN (for I from 3 to ARGS collect (ARG ARGS I]
      (CL:APPLY MAPFN (for I from 3 to ARGS collect (ARG ARGS I)))))]
      )

```

```

(CL:DEFUN MAP-ARRAY-1 (RESULT MAPFN ARRAY)
  ;; Does something fast for MAPFNS - abs truncate float and EXPONENT. ARRAY is always an array.
  [LET [(RESULT-FLOAT-P (EQ (CL:ARRAY-ELEMENT-TYPE RESULT)
      'CL:SINGLE-FLOAT))
      (ARRAY-FLOAT-P (EQ (CL:ARRAY-ELEMENT-TYPE ARRAY)
      'CL:SINGLE-FLOAT))
      (SETQ MAPFN (CL:TYPECASE MAPFN
      (CL:SYMBOL (CASE MAPFN
      (MINUS '-)
      (FIX 'CL:TRUNCATE)
      (T MAPFN)))
      (COMPILED-CLOSURE (COND
      ((OR (CL::%EQCODEP MAPFN '-)
      (CL::%EQCODEP MAPFN 'MINUS))
      '-)
      ((CL::%EQCODEP MAPFN 'ABS)
      'ABS)
      ((OR (CL::%EQCODEP MAPFN 'FIX)
      (CL::%EQCODEP MAPFN 'CL:TRUNCATE))
      'CL:TRUNCATE)
      ((CL::%EQCODEP MAPFN 'FLOAT)
      'FLOAT)
      (T MAPFN)))
      (T MAPFN)))
      (COND
      ((AND (EQ MAPFN '-)
      RESULT-FLOAT-P ARRAY-FLOAT-P)
      (%MAP-FLOAT-ARRAY-NEGATE RESULT ARRAY))
      ((AND (EQ MAPFN 'ABS)
      RESULT-FLOAT-P ARRAY-FLOAT-P)
      (%MAP-FLOAT-ARRAY-ABS RESULT ARRAY))
      ((AND (EQ MAPFN 'CL:TRUNCATE)
      ARRAY-FLOAT-P)
      (%MAP-FLOAT-ARRAY-TRUNCATE RESULT ARRAY))
      ((AND (EQ MAPFN 'FLOAT)
      RESULT-FLOAT-P)
      (%MAP-FLOAT-ARRAY-FLOAT RESULT ARRAY))
      (T (LET ((FLATTENED-RESULT (FLATTEN-ARG RESULT))
      (FLATTENED-ARRAY (FLATTEN-ARG ARRAY)))
      (CL:DOTIMES (INDEX (CL:ARRAY-TOTAL-SIZE RESULT)
      RESULT)
      (CL:SETF (CL:AREF FLATTENED-RESULT INDEX)
      (CL:FUNCCALL MAPFN (CL:AREF FLATTENED-ARRAY INDEX)))))]
      )

```

```

(CL:DEFUN MAP-ARRAY-2 (RESULT MAPFN ARRAY-1 ARRAY-2)
  ;; Does something fast for MAPFNS + - * /. At least one of ARRAY-1 and ARRAY-2 is an array
  [LET [(ARRAYS-FLOAT-P (AND (EQ (CL:ARRAY-ELEMENT-TYPE RESULT)
      'CL:SINGLE-FLOAT)
      [OR (TYPEP ARRAY-1 ' (CL:ARRAY CL:SINGLE-FLOAT))
      (TYPEP ARRAY-1 ' (OR FLOAT CL:RATIONAL])
      (OR (TYPEP ARRAY-2 ' (CL:ARRAY CL:SINGLE-FLOAT))
      (TYPEP ARRAY-2 ' (OR FLOAT CL:RATIONAL])
      )
      )
      (SETQ MAPFN (CL:TYPECASE MAPFN
      (CL:SYMBOL (CASE MAPFN
      (PLUS '+)
      (MINUS '-)

```

```

(TIMES 'CL:*)
(QUOTIENT '/')
(T MAPFN))
(COMPILED-CLOSURE (COND
  ((OR (CL::%EQCODEP MAPFN '+)
        (CL::%EQCODEP MAPFN 'PLUS))
    '+)
  ((OR (CL::%EQCODEP MAPFN '-')
        (CL::%EQCODEP MAPFN 'MINUS))
    '-)
  ((OR (CL::%EQCODEP MAPFN 'CL:*)
        (CL::%EQCODEP MAPFN 'TIMES))
    'CL:*)
  ((OR (CL::%EQCODEP MAPFN '/')
        (CL::%EQCODEP MAPFN 'QUOTIENT))
    '/)
  (T MAPFN)))
(COND
  ((AND (EQ MAPFN '+)
        ARRAYS-FLOAT-P)
    (%%MAP-FLOAT-ARRAY-PLUS RESULT ARRAY-1 ARRAY-2))
  ((AND (EQ MAPFN '-')
        ARRAYS-FLOAT-P)
    (%%MAP-FLOAT-ARRAY-MINUS RESULT ARRAY-1 ARRAY-2))
  ((AND (EQ MAPFN 'CL:*)
        ARRAYS-FLOAT-P)
    (%%MAP-FLOAT-ARRAY-TIMES RESULT ARRAY-1 ARRAY-2))
  ((AND (EQ MAPFN '/')
        ARRAYS-FLOAT-P)
    (%%MAP-FLOAT-ARRAY-QUOTIENT RESULT ARRAY-1 ARRAY-2))
  (T (LET ((FLATTENED-RESULT (FLATTEN-ARG RESULT))
           (FLATTENED-ARRAY-1 (FLATTEN-ARG ARRAY-1))
           (FLATTENED-ARRAY-2 (FLATTEN-ARG ARRAY-2)))
        (CL:IF (CL:ARRAYP ARRAY-1)
              (CL:IF (CL:ARRAYP ARRAY-2)
                    (CL:DOTIMES (INDEX (CL:ARRAY-TOTAL-SIZE RESULT)
                                       RESULT)
                                (CL:SETF (CL:AREF FLATTENED-RESULT INDEX)
                                           (CL:FUNCALL MAPFN (CL:AREF FLATTENED-ARRAY-1 INDEX)
                                                             (CL:AREF FLATTENED-ARRAY-2 INDEX))))
                  (CL:DOTIMES (INDEX (CL:ARRAY-TOTAL-SIZE RESULT)
                                       RESULT)
                                (CL:SETF (CL:AREF FLATTENED-RESULT INDEX)
                                           (CL:FUNCALL MAPFN (CL:AREF FLATTENED-ARRAY-1 INDEX)
                                                             FLATTENED-ARRAY-2))))
              (CL:DOTIMES (INDEX (CL:ARRAY-TOTAL-SIZE RESULT)
                                 RESULT)
                          (CL:SETF (CL:AREF FLATTENED-RESULT INDEX)
                                    (CL:FUNCALL MAPFN FLATTENED-ARRAY-1 (CL:AREF FLATTENED-ARRAY-2 INDEX)))))))))

```

(CL:DEFUN **REDUCE-ARRAY** (REDUCTION-FN ARRAY &OPTIONAL (INITIAL-VALUE NIL INITIAL-VALUE-P))

```

  (SETQ REDUCTION-FN (CL:TYPECASE REDUCTION-FN
    (CL:SYMBOL (CASE REDUCTION-FN
      (PLUS '+)
      (TIMES 'CL:*)
      (T REDUCTION-FN)))
    (COMPILED-CLOSURE (COND
      ((OR (CL::%EQCODEP REDUCTION-FN '+)
            (CL::%EQCODEP REDUCTION-FN 'PLUS))
        '+)
      ((OR (CL::%EQCODEP REDUCTION-FN 'CL:*)
            (CL::%EQCODEP REDUCTION-FN 'TIMES))
        'CL:*)
      ((CL::%EQCODEP REDUCTION-FN 'MIN)
        'MIN)
      ((CL::%EQCODEP REDUCTION-FN 'MAX)
        'MAX)
      ((CL::%EQCODEP REDUCTION-FN 'MIN-ABS)
        'MIN-ABS)
      ((CL::%EQCODEP REDUCTION-FN 'MAX-ABS)
        'MAX-ABS)
      (T REDUCTION-FN)))
    (T REDUCTION-FN)))
  (CL:IF (NOT (CL:ARRAYP ARRAY))
    (CL:IF INITIAL-VALUE-P
      (CL:FUNCALL REDUCTION-FN INITIAL-VALUE ARRAY)
      ARRAY)
    [LET [(SIZE (CL:ARRAY-TOTAL-SIZE ARRAY))
          (ARRAY-FLOAT-P (EQ (CL:ARRAY-ELEMENT-TYPE ARRAY)
                             'CL:SINGLE-FLOAT))]
      (CASE SIZE
        (0 (CL:IF INITIAL-VALUE-P
                  INITIAL-VALUE
                  (CL:FUNCALL REDUCTION-FN)))
        (1 (CL:IF INITIAL-VALUE-P

```

```

(CL:FUNCALL REDUCTION-FN INITIAL-VALUE (CL:AREF (FLATTEN-ARG ARRAY)
0))
(CL:AREF (FLATTEN-ARG ARRAY)
0)))
(T [COND
((AND (EQ REDUCTION-FN '+)
ARRAY-FLOAT-P)
(%%REDUCE-FLOAT-ARRAY-PLUS ARRAY INITIAL-VALUE))
((AND (EQ REDUCTION-FN 'CL:*)
ARRAY-FLOAT-P)
(%%REDUCE-FLOAT-ARRAY-TIMES ARRAY INITIAL-VALUE))
((AND (EQ REDUCTION-FN 'MIN)
ARRAY-FLOAT-P)
(%%REDUCE-FLOAT-ARRAY-MIN ARRAY INITIAL-VALUE))
((AND (EQ REDUCTION-FN 'MAX)
ARRAY-FLOAT-P)
(%%REDUCE-FLOAT-ARRAY-MAX ARRAY INITIAL-VALUE))
((AND (EQ REDUCTION-FN 'MIN-ABS)
ARRAY-FLOAT-P)
(%%REDUCE-FLOAT-ARRAY-MIN-ABS ARRAY INITIAL-VALUE))
((AND (EQ REDUCTION-FN 'MAX-ABS)
ARRAY-FLOAT-P)
(%%REDUCE-FLOAT-ARRAY-MAX-ABS ARRAY INITIAL-VALUE))
(T (CL:DO* ((FLATTENED-ARRAY (FLATTEN-ARG ARRAY))
(ACCUMULATOR (CL:IF INITIAL-VALUE-P
INITIAL-VALUE
(CL:AREF FLATTENED-ARRAY 0)))
(INDEX (CL:IF INITIAL-VALUE-P
0
1)
(CL:1+ INDEX)))
((EQ INDEX SIZE)
ACCUMULATOR)
(SETQ ACCUMULATOR (CL:FUNCALL REDUCTION-FN ACCUMULATOR (CL:AREF FLATTENED-ARRAY
INDEX)))))))]))

```

```

(CL:DEFUN EVALUATE-POLYNOMIAL (X COEFFICIENTS)
(CL:IF (NOT (CL:ARRAYP COEFFICIENTS))
(CL:ERROR "Not an array: ~S" COEFFICIENTS)
(CL:IF (EQ (CL:ARRAY-ELEMENT-TYPE COEFFICIENTS)
'CL:SINGLE-FLOAT)
(%%POLY-EVAL (FLOAT X)
(%%GET-FLOAT-ARRAY-BASE COEFFICIENTS)
(CL:1- (CL:ARRAY-TOTAL-SIZE COEFFICIENTS)))
(CL:DO ((FLATTENED-ARRAY (FLATTEN-ARG COEFFICIENTS))
(INDEX 1 (CL:1+ INDEX))
(SIZE (CL:ARRAY-TOTAL-SIZE COEFFICIENTS))
(PRODUCT (CL:AREF COEFFICIENTS 0)))
(EQ INDEX SIZE)
PRODUCT)
(SETQ PRODUCT (+ (CL:* X PRODUCT)
(CL:AREF COEFFICIENTS INDEX))))))

```

```

(CL:DEFUN FIND-ARRAY-ELEMENT-INDEX (ELEMENT ARRAY)
(CL:IF (NOT (CL:ARRAYP ARRAY))
(CL:ERROR "Not an array: ~S" ARRAY)
(CL:IF (EQ (CL:ARRAY-ELEMENT-TYPE ARRAY)
'CL:SINGLE-FLOAT)
(CL:DO ((BASE (%%GET-FLOAT-ARRAY-BASE ARRAY)
(\ADDBASE BASE 2))
(INDEX 0 (CL:1+ INDEX))
(F-ELEMENT (FLOAT ELEMENT))
(SIZE (CL:ARRAY-TOTAL-SIZE ARRAY)))
(EQ INDEX SIZE)
NIL)
(DECLARE (TYPE FLOAT F-ELEMENT))
(CL:IF (UFEQP F-ELEMENT (\GETBASEFLOATP BASE 0))
(RETURN INDEX)))
(CL:DO ((FLATTENED-ARRAY (FLATTEN-ARG ARRAY))
(INDEX 0 (CL:1+ INDEX))
(SIZE (CL:ARRAY-TOTAL-SIZE ARRAY)))
(EQ INDEX SIZE)
NIL)
(CL:IF (EQL ELEMENT (CL:AREF FLATTENED-ARRAY INDEX))
(RETURN INDEX))))))

```

```

(CL:DEFUN FLATTEN-ARG (ARG)
(CL:IF (OR (NOT (CL:ARRAYP ARG))
(EQ 1 (CL:ARRAY-RANK ARG)))
ARG
(CL:MAKE-ARRAY (CL:ARRAY-TOTAL-SIZE ARG)
:ELEMENT-TYPE
(CL:ARRAY-ELEMENT-TYPE ARG)
:DISPLACED-TO ARG))

```

```
(CL:DEFUN MAX-ABS (X Y)
  (CL:IF (> (ABS X)
           (ABS Y))
    X
    Y))
```

```
(CL:DEFUN MIN-ABS (X Y)
  (CL:IF (< (ABS X)
           (ABS Y))
    X
    Y))
```

```
(CL:DEFUN %%MAP-FLOAT-ARRAY-ABS (RESULT ARRAY)
  (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
          (RESULT-BASE (%GET-FLOAT-ARRAY-BASE RESULT)
                       (\ADDBASE RESULT-BASE 2))
          (ARRAY-BASE (%GET-FLOAT-ARRAY-BASE ARRAY)
                       (\ADDBASE ARRAY-BASE 2))
          (INDEX 0 (CL:1+ INDEX)))
    ((EQ INDEX SIZE)
     RESULT)
    (\PUTBASEFLOATP RESULT-BASE 0 (UFABS (\GETBASEFLOATP ARRAY-BASE 0)))))
```

```
(CL:DEFUN %%MAP-FLOAT-ARRAY-FLOAT (RESULT ARRAY)
  (LET ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT)))
    (CL:IF (EQUAL (CL:ARRAY-ELEMENT-TYPE ARRAY)
                  (CL:UNSIGNED-BYTE 16))
      (%BLKSMALLP2FLOAT (%GET-FLOAT-ARRAY-BASE ARRAY)
                         (%GET-FLOAT-ARRAY-BASE RESULT)
                         SIZE)
      (CL:DO ((RESULT-BASE (%GET-FLOAT-ARRAY-BASE RESULT)
                           (\ADDBASE RESULT-BASE 2))
              (INDEX 0 (CL:1+ INDEX)))
        ((EQ INDEX SIZE)
         (\PUTBASEFLOATP RESULT-BASE 0 (FLOAT (CL:AREF ARRAY INDEX))))))
    RESULT))
```

```
(CL:DEFUN %%MAP-FLOAT-ARRAY-MINUS (RESULT ARRAY-1 ARRAY-2)
  (CL:IF (CL:ARRAYP ARRAY-1)
    (CL:IF (CL:ARRAYP ARRAY-2)
      (%BLKFDIFF (%GET-FLOAT-ARRAY-BASE ARRAY-1)
                 (%GET-FLOAT-ARRAY-BASE ARRAY-2)
                 (%GET-FLOAT-ARRAY-BASE RESULT)
                 (CL:ARRAY-TOTAL-SIZE RESULT))
      (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
              (RESULT-BASE (%GET-FLOAT-ARRAY-BASE RESULT)
                           (\ADDBASE RESULT-BASE 2))
              (ARRAY-1-BASE (%GET-FLOAT-ARRAY-BASE ARRAY-1)
                             (\ADDBASE ARRAY-1-BASE 2))
              (SCALAR (FLOAT ARRAY-2))
              (INDEX 0 (CL:1+ INDEX)))
        ((EQ INDEX SIZE)
         (DECLARE (TYPE FLOATP SCALAR)
          (\PUTBASEFLOATP RESULT-BASE 0 (FDIFFERENCE (\GETBASEFLOATP ARRAY-1-BASE 0)
                                                       SCALAR))))
      (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
              (RESULT-BASE (%GET-FLOAT-ARRAY-BASE RESULT)
                           (\ADDBASE RESULT-BASE 2))
              (SCALAR (FLOAT ARRAY-1))
              (ARRAY-2-BASE (%GET-FLOAT-ARRAY-BASE ARRAY-2)
                             (\ADDBASE ARRAY-2-BASE 2))
              (INDEX 0 (CL:1+ INDEX)))
        ((EQ INDEX SIZE)
         (DECLARE (TYPE FLOATP SCALAR)
          (\PUTBASEFLOATP RESULT-BASE 0 (FDIFFERENCE SCALAR (\GETBASEFLOATP ARRAY-2-BASE 0)))))
    RESULT)
```

```
(CL:DEFUN %%MAP-FLOAT-ARRAY-NEGATE (RESULT ARRAY)
  (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
          (RESULT-BASE (%GET-FLOAT-ARRAY-BASE RESULT)
                       (\ADDBASE RESULT-BASE 2))
          (ARRAY-BASE (%GET-FLOAT-ARRAY-BASE ARRAY)
                       (\ADDBASE ARRAY-BASE 2))
          (INDEX 0 (CL:1+ INDEX)))
    ((EQ INDEX SIZE)
     RESULT)
    (\PUTBASEFLOATP RESULT-BASE 0 (UFMINUS (\GETBASEFLOATP ARRAY-BASE 0)))))
```

```
(CL:DEFUN %%MAP-FLOAT-ARRAY-PLUS (RESULT ARRAY-1 ARRAY-2)
```

```
(CL:IF (NOT (CL:ARRAYP ARRAY-1))
  (CL:ROTATEF ARRAY-1 ARRAY-2)) ; addition is commutative
(CL:IF (CL:ARRAYP ARRAY-2)
  (%%BLKFPLUS (%%GET-FLOAT-ARRAY-BASE ARRAY-1)
    (%%GET-FLOAT-ARRAY-BASE ARRAY-2)
    (%%GET-FLOAT-ARRAY-BASE RESULT)
    (CL:ARRAY-TOTAL-SIZE RESULT))
  (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
    (RESULT-BASE (%%GET-FLOAT-ARRAY-BASE RESULT)
      (\ADDBASE RESULT-BASE 2))
    (ARRAY-1-BASE (%%GET-FLOAT-ARRAY-BASE ARRAY-1)
      (\ADDBASE ARRAY-1-BASE 2))
    (SCALAR (FLOAT ARRAY-2))
    (INDEX 0 (CL:1+ INDEX)))
    (EQ INDEX SIZE)
    (DECLARE (TYPE FLOATP SCALAR))
    (\PUTBASEFLOATP RESULT-BASE 0 (FPLUS (\GETBASEFLOATP ARRAY-1-BASE 0)
      SCALAR))))
RESULT)
```

```
(CL:DEFUN %%MAP-FLOAT-ARRAY-QUOTIENT (RESULT ARRAY-1 ARRAY-2)
  (CL:IF (CL:ARRAYP ARRAY-1)
    (CL:IF (CL:ARRAYP ARRAY-2)
      (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
        (RESULT-BASE (%%GET-FLOAT-ARRAY-BASE RESULT)
          (\ADDBASE RESULT-BASE 2))
        (ARRAY-1-BASE (%%GET-FLOAT-ARRAY-BASE ARRAY-1)
          (\ADDBASE ARRAY-1-BASE 2))
        (ARRAY-2-BASE (%%GET-FLOAT-ARRAY-BASE ARRAY-2)
          (\ADDBASE ARRAY-1-BASE 2))
        (INDEX 0 (CL:1+ INDEX)))
        (EQ INDEX SIZE)
        (\PUTBASEFLOATP RESULT-BASE 0 (FQUOTIENT (\GETBASEFLOATP ARRAY-1-BASE 0)
          (\GETBASEFLOATP ARRAY-2-BASE 0))))
      (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
        (RESULT-BASE (%%GET-FLOAT-ARRAY-BASE RESULT)
          (\ADDBASE RESULT-BASE 2))
        (ARRAY-1-BASE (%%GET-FLOAT-ARRAY-BASE ARRAY-1)
          (\ADDBASE ARRAY-1-BASE 2))
        (SCALAR (FLOAT ARRAY-2))
        (INDEX 0 (CL:1+ INDEX)))
        (EQ INDEX SIZE)
        (DECLARE (TYPE FLOATP SCALAR))
        (\PUTBASEFLOATP RESULT-BASE 0 (FQUOTIENT (\GETBASEFLOATP ARRAY-1-BASE 0)
          SCALAR))))
      (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
        (RESULT-BASE (%%GET-FLOAT-ARRAY-BASE RESULT)
          (\ADDBASE RESULT-BASE 2))
        (SCALAR (FLOAT ARRAY-1))
        (ARRAY-2-BASE (%%GET-FLOAT-ARRAY-BASE ARRAY-2)
          (\ADDBASE ARRAY-2-BASE 2))
        (INDEX 0 (CL:1+ INDEX)))
        (EQ INDEX SIZE)
        (DECLARE (TYPE FLOATP SCALAR))
        (\PUTBASEFLOATP RESULT-BASE 0 (FQUOTIENT SCALAR (\GETBASEFLOATP ARRAY-2-BASE 0))))
    )
  )
RESULT)
```

```
(CL:DEFUN %%MAP-FLOAT-ARRAY-TIMES (RESULT ARRAY-1 ARRAY-2)
  (CL:IF (NOT (CL:ARRAYP ARRAY-1))
    (CL:ROTATEF ARRAY-1 ARRAY-2)) ; Multiplication is commutative
(CL:IF (CL:ARRAYP ARRAY-2)
  (%%BLKFTIMES (%%GET-FLOAT-ARRAY-BASE ARRAY-1)
    (%%GET-FLOAT-ARRAY-BASE ARRAY-2)
    (%%GET-FLOAT-ARRAY-BASE RESULT)
    (CL:ARRAY-TOTAL-SIZE RESULT))
  (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
    (RESULT-BASE (%%GET-FLOAT-ARRAY-BASE RESULT)
      (\ADDBASE RESULT-BASE 2))
    (ARRAY-1-BASE (%%GET-FLOAT-ARRAY-BASE ARRAY-1)
      (\ADDBASE ARRAY-1-BASE 2))
    (SCALAR (FLOAT ARRAY-2))
    (INDEX 0 (CL:1+ INDEX)))
    (EQ INDEX SIZE)
    (DECLARE (TYPE FLOATP SCALAR))
    (\PUTBASEFLOATP RESULT-BASE 0 (FTIMES (\GETBASEFLOATP ARRAY-1-BASE 0)
      SCALAR))))
RESULT)
```

```
(CL:DEFUN %%MAP-FLOAT-ARRAY-TRUNCATE (RESULT ARRAY)
  (CL:DO ((SIZE (CL:ARRAY-TOTAL-SIZE RESULT))
    (ARRAY-BASE (%%GET-FLOAT-ARRAY-BASE ARRAY)
      (\ADDBASE ARRAY-BASE 2))
    (INDEX 0 (CL:1+ INDEX)))
    (EQ INDEX SIZE)
```

```

RESULT)
(CL:SETF (CL:AREF RESULT INDEX)
  (UFIX (\GETBASEFLOATP ARRAY-BASE 0))))

```

```

(CL:DEFUN %%REDUCE-FLOAT-ARRAY-MAX (ARRAY INITIAL-VALUE)
  (LET [(RESULT (CL:AREF ARRAY (%BLKFMAX (%GET-FLOAT-ARRAY-BASE ARRAY)
    0
    (CL:ARRAY-TOTAL-SIZE ARRAY)
    (CL:IF INITIAL-VALUE
      (MAX INITIAL-VALUE RESULT)
      RESULT)))

```

```

(CL:DEFUN %%REDUCE-FLOAT-ARRAY-MAX-ABS (ARRAY INITIAL-VALUE)
  (LET [(RESULT (CL:AREF ARRAY (%BLKFABS MAX (%GET-FLOAT-ARRAY-BASE ARRAY)
    0
    (CL:ARRAY-TOTAL-SIZE ARRAY)
    (CL:IF INITIAL-VALUE
      (MAX-ABS INITIAL-VALUE RESULT)
      RESULT)))

```

```

(CL:DEFUN %%REDUCE-FLOAT-ARRAY-MIN (ARRAY INITIAL-VALUE)
  (LET [(RESULT (CL:AREF ARRAY (%BLKFMIN (%GET-FLOAT-ARRAY-BASE ARRAY)
    0
    (CL:ARRAY-TOTAL-SIZE ARRAY)
    (CL:IF INITIAL-VALUE
      (MIN INITIAL-VALUE RESULT)
      RESULT)))

```

```

(CL:DEFUN %%REDUCE-FLOAT-ARRAY-MIN-ABS (ARRAY INITIAL-VALUE)
  (LET [(RESULT (CL:AREF ARRAY (%BLKFABS MIN (%GET-FLOAT-ARRAY-BASE ARRAY)
    0
    (CL:ARRAY-TOTAL-SIZE ARRAY)
    (CL:IF INITIAL-VALUE
      (MIN-ABS INITIAL-VALUE RESULT)
      RESULT)))

```

```

(CL:DEFUN %%REDUCE-FLOAT-ARRAY-PLUS (ARRAY INITIAL-VALUE)
  (LET [(RESULT (%POLY-EVAL 1.0 (%GET-FLOAT-ARRAY-BASE ARRAY)
    (CL:1- (CL:ARRAY-TOTAL-SIZE ARRAY)
    (CL:IF INITIAL-VALUE
      (+ INITIAL-VALUE RESULT)
      RESULT)))

```

```

(CL:DEFUN %%REDUCE-FLOAT-ARRAY-TIMES (ARRAY INITIAL-VALUE)
  (LET ((TOTAL 1.0)
    (DECLARE (TYPE FLOAT TOTAL))
    (CL:DO ((I 0 (CL:1+ I))
      (BASE (%GET-FLOAT-ARRAY-BASE ARRAY)
        (\ADDBASE BASE 2))
      (SIZE (CL:ARRAY-TOTAL-SIZE ARRAY)))
      (EQ I SIZE)
      TOTAL)
    (SETQ TOTAL (CL:* TOTAL (\GETBASEFLOATP BASE 0))))
  (CL:IF INITIAL-VALUE
    (CL:* INITIAL-VALUE TOTAL)
    TOTAL))

```

:: For convenience

```
(PUTPROPS CMLFLOATARRAY FILETYPE CL:COMPILE-FILE)
```

```
(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY
```

```
(DECLARE%: DOEVAL@COMPILE DONTCOPY
```

```
(LOCALVARS . T)
)
)
```

```
(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVERS
```

```
(ADDTOVAR NLAMA )
```

```
(ADDTOVAR NLAML )
```

```
(ADDTOVAR LAMA MAP-ARRAY)
)
```

```
(PUTPROPS CMLFLOATARRAY COPYRIGHT ("Venue & Xerox Corporation" 1985 1986 1987 1990 1993))
```

---

**FUNCTION INDEX**

%%MAP-FLOAT-ARRAY-ABS .....	5	%%REDUCE-FLOAT-ARRAY-MAX .....	7	FLATTEN-ARG .....	4
%%MAP-FLOAT-ARRAY-FLOAT .....	5	%%REDUCE-FLOAT-ARRAY-MAX-ABS .....	7	MAP-ARRAY .....	1
%%MAP-FLOAT-ARRAY-MINUS .....	5	%%REDUCE-FLOAT-ARRAY-MIN .....	7	MAP-ARRAY-1 .....	2
%%MAP-FLOAT-ARRAY-NEGATE .....	5	%%REDUCE-FLOAT-ARRAY-MIN-ABS .....	7	MAP-ARRAY-2 .....	2
%%MAP-FLOAT-ARRAY-PLUS .....	5	%%REDUCE-FLOAT-ARRAY-PLUS .....	7	MAX-ABS .....	5
%%MAP-FLOAT-ARRAY-QUOTIENT .....	6	%%REDUCE-FLOAT-ARRAY-TIMES .....	7	MIN-ABS .....	5
%%MAP-FLOAT-ARRAY-TIMES .....	6	EVALUATE-POLYNOMIAL .....	4	REDUCE-ARRAY .....	3
%%MAP-FLOAT-ARRAY-TRUNCATE .....	6	FIND-ARRAY-ELEMENT-INDEX .....	4		

---

**PROPERTY INDEX**

CMLFLOATARRAY .....	7
---------------------	---

---