


```
(FUNCTION (LAMBDA (I)
  (COND
    ((CDR I)
     (CONS (CHARCODE.DECODE (CAAR I))
           (CDAR I)))
    (T (CAR I])))
```

```
(PUTPROPS ALPHACHARP MACRO ((CHAR)
  ([LAMBDA (UCHAR)
    (DECLARE (LOCALVARS UCHAR))
    (AND (IGEQ UCHAR (CHARCODE A))
         (ILEQ UCHAR (CHARCODE Z])
         (LOGAND CHAR 95)))))
```

```
(PUTPROPS DIGITCHARP MACRO [LAMBDA (CHAR)
  (AND (IGEQ CHAR (CHARCODE 0))
        (ILEQ CHAR (CHARCODE 9]))
```

```
(PUTPROPS UCASECODE MACRO (OPENLAMBDA (CHAR)
  (COND
    ((AND (IGEQ CHAR (CHARCODE a))
          (ILEQ CHAR (CHARCODE z))))
    (LOGAND CHAR 95))
    (T CHAR))))
```

```
(DEFOPTIMIZER CHARCODE (C)
  (KWOTE (CHARCODE.DECODE C T)))
```

```
(ADDTOVAR DWIMEQUIVLST (SELCHARQ . SELECTQ))
```

```
(ADDTOVAR PRETTYEQUIVLST (SELCHARQ . SELECTQ))
```

:: Common Lisp CHARACTER type

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

```
(DECLARE%: EVAL@COMPILE
```

```
(ACCESSFNS CHARACTER [(CODE (\LOLOC (\DTEST DATUM 'CHARACTER]
  (CREATE (\VAG2 \CHARHI CODE)))
)
)
```

```
(CL:DEFCONSTANT \CHARHI 7)
```

```
(CL:DEFCONSTANT CL:CHAR-BITS-LIMIT 1)
```

```
(CL:DEFCONSTANT CL:CHAR-CODE-LIMIT 65536)
```

```
(CL:DEFCONSTANT CL:CHAR-CONTROL-BIT 0)
```

```
(CL:DEFCONSTANT CL:CHAR-FONT-LIMIT 1)
```

```
(CL:DEFCONSTANT CL:CHAR-HYPER-BIT 0)
```

```
(CL:DEFCONSTANT CL:CHAR-META-BIT 0)
```

```
(CL:DEFCONSTANT CL:CHAR-SUPER-BIT 0)
```

:: Basic character fns

```
(DEFINEQ
```

```
(CL:CHAR-CODE
  [LAMBDA (CHAR)
    (\LOLOC (\DTEST CHAR 'CHARACTER])
```

(* jop%: "25-Aug-86 17:30")

```
(CL:CHAR-INT
  [LAMBDA (CHAR)
    (CL:CHAR-CODE CHAR)])
```

```
(CL:INT-CHAR
  [LAMBDA (INTEGER)
    (CL:CODE-CHAR INTEGER)])
```

(* Imm " 7-Jul-85 16:50")

)

(CL:DEFUN **CL:CODE-CHAR** (CODE &OPTIONAL (BITS 0) (FONT 0))

(CL:IF (AND (EQ BITS 0) (EQ FONT 0))
;; This checks for smallposp
(EQ (\HILOC CODE) \SmallPosHi))
(%%CODE-CHAR CODE))

(DEFOPTIMIZER **CL:CHAR-CODE** (CHAR)
[LET [(CONSTANT-CHAR (AND (CL:CONSTANTP CHAR) (CL:EVAL CHAR)
(CL:IF (**CL:CHARACTERP** CONSTANT-CHAR)
(\LOLOC CONSTANT-CHAR)
'(\LOLOC (\DTEST ,CHAR 'CHARACTER)))]]

(DEFOPTIMIZER **CL:CHAR-INT** (CHAR)
\(**CL:CHAR-CODE** ,CHAR))

(DEFOPTIMIZER **CL:CODE-CHAR** (CODE &OPTIONAL (BITS 0) (FONT 0))
(CL:IF (AND (EQ BITS 0) (EQ FONT 0))
[LET [(CONSTANT-CODE (AND (CL:CONSTANTP CODE) (CL:EVAL CODE)
(CL:IF (EQ (\HILOC CONSTANT-CODE) \SmallPosHi)
(%%CODE-CHAR CONSTANT-CODE)
\ (LET ((%CODE ,CODE))
(AND (EQ (\HILOC %CODE) \SmallPosHi)
(%%CODE-CHAR %CODE)))]])
'COMPILER:PASS))

(DEFOPTIMIZER **CL:INT-CHAR** (INTEGER)
\(**CL:CODE-CHAR** ,INTEGER))

;; I/O; Some is here, the rest is in LLREAD.

(DEFINEQ

(**CHARACTER.PRINT**

[LAMBDA (CHAR STREAM) ; Edited 10-Sep-87 16:29 by amd
[COND
[*PRINT-ESCAPE* ; Name that can be read back
(LET ((PNAME (**CL:CHAR-NAME** CHAR)))
[.SPACECHECK. STREAM (+ 2 (COND
(PNAME (CL:LENGTH PNAME))
(T 1) ; Print as #\ followed by charcter name
(\OUTCHAR STREAM (fetch (READTABLEP HASHMACROCHAR) of *READTABLE*))
(\OUTCHAR STREAM (**CHARCODE** "\"))
(COND
(PNAME (WRITE-STRING* PNAME STREAM))
(T (\OUTCHAR STREAM (**CL:CHAR-CODE** CHAR) ; Character as character
))]
)]])

)

(DECLARE%: DONTEVAL@LOAD DOCOPY

(SETTOPVAL (\TYPEGLOBALVARIABLE 'CHARACTER T)
(NTYPX (**CL:CODE-CHAR** 0 0 0)))

(DEFPRIINT 'CHARACTER 'CHARACTER.PRINT)
)

;; Common lisp character functions

(DEFINEQ

(**CL:CHAR-BIT**

[LAMBDA (CHAR NAME) ; (* jop%: "26-Aug-86 15:01")
(CL:ERROR "Bit ~A not supported" NAME)]

(**CL:CHAR-BITS**

[LAMBDA (CHAR) (* jop%: "25-Aug-86 17:35")
(AND (CL:CHARACTERP CHAR)
0)]

(CL:CHAR-DOWNCASE

[LAMBDA (CHAR) (* jop%: "25-Aug-86 18:01")
(%%CODE-CHAR (%%CHAR-DOWNCASE-CODE (CL:CHAR-CODE CHAR)])

(CL:CHAR-FONT

[LAMBDA (CHAR) (* jop%: "25-Aug-86 17:35")
(AND (CL:CHARACTERP CHAR)
0)]

(CL:CHAR-NAME

[LAMBDA (CHAR) ; Edited 30-Oct-2023 17:57 by mth
; Edited 19-Mar-87 15:49 by bvm:
(DECLARE (GLOBALVARS CHARACTERNAMES CHARACTERSETNAMES))
(COND
((EQ CHAR #\Space) ; Space is special because it is graphic but has a name
"Space")
((CL:GRAPHIC-CHAR-P CHAR) ; graphics have no special names
NIL)
(T (LET ((CODE (CL:CHAR-CODE CHAR))
CSET)
(COND
[(for X in CHARACTERNAMES when (EQ (CADR X)
CODE)
do ;; This assumes that (CAR X) is SYMBOL or STRING!!
;; (Should this be enforced? I.e., error if not?)
(RETURN (STRING (CAR X)
(T (SETQ CSET (LRSH CODE 8))
(SETQ CODE (LOGAND CODE 255))
(COND
[(AND (EQ CSET 0)
(<= CODE (CHARCODE "^Z"))] ; represent ascii control chars nicely
(CONCAT "^" (CL:CODE-CHAR (LOGOR CODE (- (CHARCODE "A")
(CHARCODE "^A"))
; Else charset-charcode
(T (CONCAT (for X in CHARACTERSETNAMES when (EQ (CADR X)
CSET)
do (RETURN (CAR X)) finally (RETURN (OCTALSTRING CSET)))
"_"
(OCTALSTRING CODE)]

(CL:CHAR-UPCASE

[LAMBDA (CHAR) (* jop%: "25-Aug-86 18:01")
(%%CODE-CHAR (%%CHAR-UPCASE-CODE (CL:CHAR-CODE CHAR)])

(CL:CHARACTER

[LAMBDA (OBJECT) (* jop%: "14-Nov-86 16:22")
(COND
((TYPEP OBJECT 'CL:CHARACTER)
OBJECT)
((TYPEP OBJECT 'CL:FIXNUM)
(CL:INT-CHAR OBJECT))
([AND (OR (TYPEP OBJECT 'STRING)
(TYPEP OBJECT 'CL:SYMBOL))
(EQL 1 (CL:LENGTH (SETQ OBJECT (STRING OBJECT)
(CL:CHAR OBJECT 0))
(T (CL:ERROR "Object cannot be coerced to a character: ~S" OBJECT))

(CL:NAME-CHAR

[LAMBDA (NAME) ; Edited 18-Feb-87 22:05 by bvm:
(LET ((CODE (CHARCODE.DECODE (STRING NAME)
T)))
(AND CODE (CL:CODE-CHAR CODE)]

(CL:SET-CHAR-BIT

[LAMBDA (CHAR NAME NEWVALUE) (* jop%: "26-Aug-86 15:02")
(CL:ERROR "Bit ~A not supported" NAME)]

(CL:DEFUN CL:DIGIT-CHAR (WEIGHT &OPTIONAL (RADIX 10)

(FONT 0))
[AND (EQ FONT 0)
(< -1 WEIGHT RADIX 37)
(CL:IF (< WEIGHT 10)

```
(%%CODE-CHAR (+ (CONSTANT (CL:CHAR-CODE #\0))
                 WEIGHT))
(%%CODE-CHAR (+ (CONSTANT (CL:CHAR-CODE #\A))
                 (- WEIGHT 10))))]
```

```
(CL:DEFUN CL:MAKE-CHAR (CHAR &OPTIONAL (BITS 0)
                        (FONT 0))
  (CL:IF (AND (EQL BITS 0)
              (EQL FONT 0))
    CHAR))
```

```
(DEFOPTIMIZER CL:CHAR-UPCASE (CHAR)
  '[%%CODE-CHAR (%%CHAR-UPCASE-CODE (CL:CHAR-CODE ,CHAR))
```

```
(DEFOPTIMIZER CL:CHAR-DOWNCASE (CHAR)
  '[%%CODE-CHAR (%%CHAR-DOWNCASE-CODE (CL:CHAR-CODE ,CHAR))
```

```
(DEFOPTIMIZER CL:MAKE-CHAR (CHAR &OPTIONAL BITS FONT)
  (CL:IF (AND (OR (NULL BITS)
                  (EQL BITS 0))
              (OR (NULL FONT)
                  (EQL FONT 0)))
    CHAR
    'COMPILER:PASS))
```

:: Predicates

```
(DEFINEQ
```

(CL:ALPHA-CHAR-P

```
[LAMBDA (CHAR)
  (LET ((CODE (CL:CHAR-CODE CHAR)))
    (OR (<= (CONSTANT (CL:CHAR-CODE #\A))
         CODE
         (CONSTANT (CL:CHAR-CODE #\Z)))
        (<= (CONSTANT (CL:CHAR-CODE #\a))
         CODE
         (CONSTANT (CL:CHAR-CODE #\z))
```

(* raf "23-Oct-85 15:03")
; Might want to make this true for Greek char sets, etc.

(CL:ALPHANUMERICP

```
[LAMBDA (CHAR)
  (OR (CL:ALPHA-CHAR-P CHAR)
      (NOT (NULL (CL:DIGIT-CHAR-P CHAR))
```

(* Imm "28-Oct-85 20:40")

(CL:BOTH-CASE-P

```
[LAMBDA (CHAR)
  (OR (CL:UPPER-CASE-P CHAR)
      (CL:LOWER-CASE-P CHAR))
```

(CL:CHARACTERP

```
[LAMBDA (OBJECT)
  (TYPENAMEP OBJECT 'CHARACTER)]
```

(* Imm " 1-Aug-85 22:45")

(CL:GRAPHIC-CHAR-P

```
[LAMBDA (CHAR)
```

(* bvm%: "14-May-86 16:19")

::: True if CHAR represents a graphic (printing) character. Definition follows NS character standard

```
(LET* ((CODE (CL:CHAR-CODE CHAR))
       (CSET (LRSH CODE 8)))
  (AND [PROGN
        (OR (EQ CSET 0)
            (AND (> (SETQ CSET (LOGAND CSET 127))
                   32)
              (NOT (EQ CSET 127))
            (PROGN
              (OR (EQ (SETQ CODE (LOGAND CODE 255))
                     (CONSTANT (CL:CHAR-CODE #\Space)))
                  (AND (> (SETQ CODE (LOGAND CODE 127))
                         32)
                    (NOT (EQ CODE 127))
```

; Graphic charsets are zero, 41 thru 176, 241 thru 276

; Printing chars within a character set are SPACE thru 176 and
; 241 thru 276

(CL:LOWER-CASE-P

```
[LAMBDA (CHAR)
  (<= (CONSTANT (CL:CHAR-CODE #\a))
      (CL:CHAR-CODE CHAR)
      (CONSTANT (CL:CHAR-CODE #\z))
```

(CL:STANDARD-CHAR-P

; Edited 7-Jan-87 11:42 by jop

```
[LAMBDA (CHAR)
  (AND (CL:MEMBER CHAR
        '(#\! #\" #\$ #\% #\& #\' #( #\ ) #* #+ #\, #- #\. #\/ #\0 #\1 #\2 #\3 #\4 #\5 #\6 #\7
          #\8
          #\9 #: #\; #\< #\= #\> #\? #\@ #\A #\B #\C #\D #\E #\F #\G #\H #\I #\J #\K #\L #\M #\N
          #\O
          #\P #\Q #\R #\S #\T #\U #\V #\W #\X #\Y #\Z #\[ #\] #\^ #\_ #\` #\a #\b #\c #\d #\e
          #\f
          #\g #\h #\i #\j #\k #\l #\m #\n #\o #\p #\q #\r #\s #\t #\u #\v #\w #\x #\y #\z #\{ #\|
          #\}
          #\~ #\Space #\Newline))
  T])
```

(CL:STRING-CHAR-P

```
[LAMBDA (CHAR)
  (\DTEST CHAR 'CHARACTER)]
```

(CL:UPPER-CASE-P

```
[LAMBDA (CHAR)
  (<= (CONSTANT (CL:CHAR-CODE #\A))
       (CL:CHAR-CODE CHAR)
       (CONSTANT (CL:CHAR-CODE #\Z]))
```

)

(DEFINEQ

(CL:CHAR-EQUAL

(* jop%: "25-Aug-86 16:03")

```
[LAMBDA N
  (CL:IF (< N 1)
         (CL:ERROR "CHAR-EQUAL takes at least one arg"))
  (CL:DO ((TEST (CL:CHAR-UPCASE (ARG N 1)))
          (I 2 (CL:1+ I)))
        (> I N)
        T)
  (CL:IF [NOT (EQ TEST (CL:CHAR-UPCASE (ARG N I)
                                         (RETURN NIL))))])
```

(CL:CHAR-GREATERP

(* jop%: "25-Aug-86 17:15")

```
[LAMBDA N
  (CL:IF (< N 1)
         (CL:ERROR "CHAR-LESSP takes at least one arg"))
  (CL:DO ([LAST (%%CHAR-UPCASE-CODE (CL:CHAR-CODE (ARG N 1]
                                     NEXT
                                     (I 2 (CL:1+ I)))
          (> I N)
          T)
  [SETQ NEXT (%%CHAR-UPCASE-CODE (CL:CHAR-CODE (ARG N I]
  (CL:IF (NOT (> LAST NEXT))
          (RETURN NIL)
          (SETQ LAST NEXT))))])
```

(CL:CHAR-LESSP

(* jop%: "25-Aug-86 17:17")

```
[LAMBDA N
  (CL:IF (< N 1)
         (CL:ERROR "CHAR-LESSP takes at least one arg"))
  (CL:DO ([LAST (%%CHAR-UPCASE-CODE (CL:CHAR-CODE (ARG N 1]
                                     NEXT
                                     (I 2 (CL:1+ I)))
          (> I N)
          T)
  [SETQ NEXT (%%CHAR-UPCASE-CODE (CL:CHAR-CODE (ARG N I]
  (CL:IF (NOT (< LAST NEXT))
          (RETURN NIL)
          (SETQ LAST NEXT))))])
```

(CL:CHAR-NOT-EQUAL

(* jop%: "25-Aug-86 16:02")

```
[LAMBDA N
  (CL:IF (< N 1)
         (CL:ERROR "CHAR-NOT-EQUAL takes at least one arg"))
  (CL:DO ((I 1 (CL:1+ I))
          (TEST)
          (> I N)
          T)
  (SETQ TEST (CL:CHAR-UPCASE (ARG N I)))
  (CL:IF (CL:DO ((J (CL:1+ I)
                  (CL:1+ J))
                (> J N)
                NIL)
```

```

      (CL:IF (EQ TEST (CL:CHAR-UPCASE (ARG N J)))
              (RETURN T)))
    (RETURN NIL)))]

```

(CL:CHAR-NOT-GREATERP

(* jop%: "25-Aug-86 17:18")

```

[LAMBDA N
  (CL:IF (< N 1)
          (CL:ERROR "CHAR-LESSP takes at least one arg"))
  (CL:DO ((LAST (%%CHAR-UPCASE-CODE (CL:CHAR-CODE (ARG N 1]
    NEXT
      (I 2 (CL:1+ I)))
      (> I N)
      T)
    [SETQ NEXT (%%CHAR-UPCASE-CODE (CL:CHAR-CODE (ARG N I]
      (CL:IF (NOT (<= LAST NEXT))
              (RETURN NIL)
              (SETQ LAST NEXT))))))

```

(CL:CHAR-NOT-LESSP

(* jop%: "25-Aug-86 17:19")

```

[LAMBDA N
  (CL:IF (< N 1)
          (CL:ERROR "CHAR-LESSP takes at least one arg"))
  (CL:DO ((LAST (%%CHAR-UPCASE-CODE (CL:CHAR-CODE (ARG N 1]
    NEXT
      (I 2 (CL:1+ I)))
      (> I N)
      T)
    [SETQ NEXT (%%CHAR-UPCASE-CODE (CL:CHAR-CODE (ARG N I]
      (CL:IF (NOT (>= LAST NEXT))
              (RETURN NIL)
              (SETQ LAST NEXT))))))

```

(CL:CHAR/=

(* jop%: "25-Aug-86 17:07")

```

[LAMBDA N
  (CL:IF (< N 1)
          (CL:ERROR "CHAR/= takes at least one arg"))
  (CL:DO ((I 1 (CL:1+ I))
          (TEST)
          (> I N)
          T)
    (SETQ TEST (CL:CHAR-CODE (ARG N I)))
    (CL:IF (CL:DO ((J (CL:1+ I)
                  (CL:1+ J))
                  (> J N)
                  NIL)
            (CL:IF (EQ TEST (CL:CHAR-CODE (ARG N J)))
                    (RETURN T))))
    (RETURN NIL))))

```

(CL:CHAR<

(* jop%: "25-Aug-86 14:29")

```

[LAMBDA N
  (CL:IF (< N 1)
          (CL:ERROR "CHAR< takes at least one arg"))
  (CL:DO ((LAST (CL:CHAR-CODE (ARG N 1)))
          NEXT
            (I 2 (CL:1+ I)))
          (> I N)
          T)
    (SETQ NEXT (CL:CHAR-CODE (ARG N I)))
    (CL:IF (NOT (< LAST NEXT))
            (RETURN NIL)
            (SETQ LAST NEXT))))

```

(CL:CHAR<=

(* jop%: "25-Aug-86 14:38")

```

[LAMBDA N
  (CL:IF (< N 1)
          (CL:ERROR "CHAR< takes at least one arg"))
  (CL:DO ((LAST (CL:CHAR-CODE (ARG N 1)))
          NEXT
            (I 2 (CL:1+ I)))
          (> I N)
          T)
    (SETQ NEXT (CL:CHAR-CODE (ARG N I)))
    (CL:IF (NOT (<= LAST NEXT))
            (RETURN NIL)
            (SETQ LAST NEXT))))

```

(CL:CHAR=

(* jop%: "25-Aug-86 17:05")

```

[LAMBDA N
  (CL:IF (< N 1)
          (CL:ERROR "CHAR= takes at least one arg"))

```

```
(CL:DO ((TEST (CL:CHAR-CODE (ARG N 1)))
        (I 2 (CL:1+ I)))
      (> I N)
      T)
  (CL:IF [NOT (EQ TEST (CL:CHAR-CODE (ARG N I)
    (RETURN NIL)))]])
```

(CL:CHAR>

(* jop%: "25-Aug-86 14:34")

```
[LAMBDA N
  (CL:IF (< N 1)
    (CL:ERROR "CHAR< takes at least one arg"))
  (CL:DO ((LAST (CL:CHAR-CODE (ARG N 1)))
          NEXT
          (I 2 (CL:1+ I)))
        (> I N)
        T)
    (SETQ NEXT (CL:CHAR-CODE (ARG N I)))
    (CL:IF (NOT (> LAST NEXT))
      (RETURN NIL)
      (SETQ LAST NEXT))))]
```

(CL:CHAR>=

(* jop%: "25-Aug-86 14:40")

```
[LAMBDA N
  (CL:IF (< N 1)
    (CL:ERROR "CHAR< takes at least one arg"))
  (CL:DO ((LAST (CL:CHAR-CODE (ARG N 1)))
          NEXT
          (I 2 (CL:1+ I)))
        (> I N)
        T)
    (SETQ NEXT (CL:CHAR-CODE (ARG N I)))
    (CL:IF (NOT (>= LAST NEXT))
      (RETURN NIL)
      (SETQ LAST NEXT))))]
```

)

(CL:DEFUN CL:DIGIT-CHAR-P (CHAR &OPTIONAL (RADIX 10))

"Returns the weigh of CHAR in radix RADIX, or NIL if CHAR is not a digit char in that radix."

```
(LET* [(CODE (CL:CHAR-CODE CHAR))
      (VAL (COND
        [(<= (CONSTANT (CL:CHAR-CODE #\0))
              CODE
              (CONSTANT (CL:CHAR-CODE #\9)))]
        [- CODE (CONSTANT (CL:CHAR-CODE #\0))
           [(<= (CONSTANT (CL:CHAR-CODE #\A))
                 CODE
                 (CONSTANT (CL:CHAR-CODE #\Z)))]
           [+ 10 (- CODE (CONSTANT (CL:CHAR-CODE #\A))
                   [(<= (CONSTANT (CL:CHAR-CODE #\a))
                         CODE
                         (CONSTANT (CL:CHAR-CODE #\z)))]
                   [+ 10 (- CODE (CONSTANT (CL:CHAR-CODE #\a))
                               (AND VAL (< VAL RADIX)
                                   VAL))])])])])
```

(DEFOPTIMIZER CL:CHAR-EQUAL (CHAR &REST MORE-CHARS)

```
(CL:IF (EQL 1 (CL:LENGTH MORE-CHARS))
  `[EQ (%%CHAR-UPCASE-CODE (CL:CHAR-CODE ,CHAR))
        (%%CHAR-UPCASE-CODE (CL:CHAR-CODE ,(CAR MORE-CHARS))
        'COMPILER:PASS))
```

(DEFOPTIMIZER CL:CHAR-GREATERP (CHAR &REST MORE-CHARS)

```
`(> (%%CHAR-UPCASE-CODE (CL:CHAR-CODE ,CHAR))
      ,@(CL:MAPCAR [FUNCTION (CL:LAMBDA (FORM)
                          \ (%%CHAR-UPCASE-CODE
                            (CL:CHAR-CODE ,FORM)
                          MORE-CHARS))])
```

(DEFOPTIMIZER CL:CHAR-LESSP (CHAR &REST MORE-CHARS)

```
`(< (%%CHAR-UPCASE-CODE (CL:CHAR-CODE ,CHAR))
      ,@(CL:MAPCAR [FUNCTION (CL:LAMBDA (FORM)
                          \ (%%CHAR-UPCASE-CODE (CL:CHAR-CODE
                                                    ,FORM)
                          MORE-CHARS))])
```

(DEFOPTIMIZER CL:CHAR-NOT-EQUAL (CHAR &REST MORE-CHARS)

```
(CL:IF (EQL 1 (CL:LENGTH MORE-CHARS))
  `[NOT (EQ (%%CHAR-UPCASE-CODE (CL:CHAR-CODE ,CHAR))
            (%%CHAR-UPCASE-CODE (CL:CHAR-CODE ,(CAR MORE-CHARS))
            'COMPILER:PASS))
```


'COMPILER:PASS))

(DEFOPTIMIZER CL:CHAR-NOT-GREATERP (CHAR &REST MORE-CHARS)
\(<= (%%CHAR-UPCASE-CODE (CL:CHAR-CODE ,CHAR))
,@(CL:MAPCAR [FUNCTION (CL:LAMBDA (FORM)
(FORM)
\(%%CHAR-UPCASE-CODE
(CL:CHAR-CODE ,FORM]
MORE-CHARS)))

(DEFOPTIMIZER CL:CHAR-NOT-LESSP (CHAR &REST MORE-CHARS)
\(>= (%%CHAR-UPCASE-CODE (CL:CHAR-CODE ,CHAR))
,@(CL:MAPCAR [FUNCTION (CL:LAMBDA (FORM)
(FORM)
\(%%CHAR-UPCASE-CODE
(CL:CHAR-CODE ,FORM]
MORE-CHARS)))

(DEFOPTIMIZER CL:CHAR/= (CHAR &REST MORE-CHARS)
(CL:IF (CDR MORE-CHARS)
'COMPILER:PASS
\ (NEQ ,CHAR ,(CAR MORE-CHARS))))

(DEFOPTIMIZER CL:CHAR< (CHAR &REST MORE-CHARS)
\(< (CL:CHAR-CODE ,CHAR)
,@(CL:MAPCAR [FUNCTION (CL:LAMBDA (FORM)
(FORM)
\ (CL:CHAR-CODE ,FORM]
MORE-CHARS)))

(DEFOPTIMIZER CL:CHAR<= (CHAR &REST MORE-CHARS)
\(<= (CL:CHAR-CODE ,CHAR)
,@(CL:MAPCAR [FUNCTION (CL:LAMBDA (FORM)
(FORM)
\ (CL:CHAR-CODE ,FORM]
MORE-CHARS)))

(DEFOPTIMIZER CL:CHAR= (CHAR &REST MORE-CHARS)
(CL:IF (CDR MORE-CHARS)
[LET ((CH (GENSYM)))
\ (LET ((,CH ,CHAR))
(AND ,@(for X in MORE-CHARS collect \ (EQ ,CH ,X)
\ (EQ ,CHAR ,(CAR MORE-CHARS))))

(DEFOPTIMIZER CL:CHAR> (CHAR &REST MORE-CHARS)
\(> (CL:CHAR-CODE ,CHAR)
,@(CL:MAPCAR [FUNCTION (CL:LAMBDA (FORM)
(FORM)
\ (CL:CHAR-CODE ,FORM]
MORE-CHARS)))

(DEFOPTIMIZER CL:CHAR>= (CHAR &REST MORE-CHARS)
\(>= (CL:CHAR-CODE ,CHAR)
,@(CL:MAPCAR [FUNCTION (CL:LAMBDA (FORM)
(FORM)
\ (CL:CHAR-CODE ,FORM]
MORE-CHARS)))

(DEFOPTIMIZER CL:CHARACTERP (OBJECT)
\ (TYPENAMEP ,OBJECT 'CHARACTER))

(DEFOPTIMIZER CL:LOWER-CASE-P (CHAR)
\ (<= (CONSTANT (CL:CHAR-CODE #\a)
(CL:CHAR-CODE ,CHAR)
(CONSTANT (CL:CHAR-CODE #\z))))

(DEFOPTIMIZER CL:STRING-CHAR-P (CHAR)
\ (\DTEST ,CHAR 'CHARACTER))

(DEFOPTIMIZER CL:UPPER-CASE-P (CHAR)
\ (<= (CONSTANT (CL:CHAR-CODE #\A)
(CL:CHAR-CODE ,CHAR)
(CONSTANT (CL:CHAR-CODE #\Z))))

:: Internals

(DEFMACRO %%CHAR-DOWNCASE-CODE (CODE)
\ (LET ((%%CODE ,CODE))
(CL:IF (<= (CONSTANT (CL:CHAR-CODE #\A)

```

%%CODE
(CONSTANT (CL:CHAR-CODE #\Z)))
[+ %%CODE (- (CONSTANT (CL:CHAR-CODE #\a))
(CONSTANT (CL:CHAR-CODE #\A]
%%CODE))

```

```

(DEFMACRO %%CHAR-UPCASE-CODE (CODE)
  \ (LET ((%%CODE ,CODE))
    (CL:IF (<= (CONSTANT (CL:CHAR-CODE #\a))
%%CODE
(CONSTANT (CL:CHAR-CODE #\z))))
[- %%CODE (- (CONSTANT (CL:CHAR-CODE #\a))
(CONSTANT (CL:CHAR-CODE #\A]
%%CODE)))

```

```

(DEFMACRO %%CODE-CHAR (CODE)
  \ (\VAG2 \CHARHI ,CODE))

```

:: Compiler options

```

(PUTPROPS CMLCHARACTER 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 CHARCODE)
(ADDTOVAR LAMA CL:CHAR>= CL:CHAR> CL:CHAR= CL:CHAR<= CL:CHAR< CL:CHAR/= CL:CHAR-NOT-LESSP CL:CHAR-NOT-GREATERP
CL:CHAR-NOT-EQUAL CL:CHAR-LESSP CL:CHAR-GREATERP CL:CHAR-EQUAL)
)
(PUTPROPS CMLCHARACTER COPYRIGHT ("Venue & Xerox Corporation" 1985 1986 1987 1990 1995 1999 2023))

```

FUNCTION INDEX

CL:ALPHA-CHAR-P5	CL:CHAR-INT2	CL:CHAR=7	CL:DIGIT-CHAR-P8
CL:ALPHANUMERICP5	CL:CHAR-LESSP6	CL:CHAR>8	CL:GRAPHIC-CHAR-P5
CL:BOTH-CASE-P5	CL:CHAR-NAME4	CL:CHAR>=8	CL:INT-CHAR2
CL:CHAR-BIT3	CL:CHAR-NOT-EQUAL6	CL:CHARACTER4	CL:LOWER-CASE-P5
CL:CHAR-BITS3	CL:CHAR-NOT-GREATERP7	CHARACTER.PRINT3	CL:MAKE-CHAR5
CL:CHAR-CODE2	CL:CHAR-NOT-LESSP7	CL:CHARACTERP5	CL:NAME-CHAR4
CL:CHAR-DOWNCASE4	CL:CHAR-UPCASE4	CHARCODE1	CL:SET-CHAR-BIT4
CL:CHAR-EQUAL6	CL:CHAR/=7	CHARCODE.UNDECODE1	CL:STANDARD-CHAR-P6
CL:CHAR-FONT4	CL:CHAR<7	CL:CODE-CHAR3	CL:STRING-CHAR-P6
CL:CHAR-GREATERP6	CL:CHAR<=7	CL:DIGIT-CHAR4	CL:UPPER-CASE-P6

OPTIMIZER INDEX

CL:CHAR-CODE3	CL:CHAR-NOT-EQUAL8	CL:CHAR<=9	CL:CODE-CHAR3
CL:CHAR-DOWNCASE5	CL:CHAR-NOT-GREATERP9	CL:CHAR=9	CL:INT-CHAR3
CL:CHAR-EQUAL8	CL:CHAR-NOT-LESSP9	CL:CHAR>9	CL:LOWER-CASE-P9
CL:CHAR-GREATERP8	CL:CHAR-UPCASE5	CL:CHAR>=9	CL:MAKE-CHAR5
CL:CHAR-INT3	CL:CHAR/=9	CL:CHARACTERP9	CL:STRING-CHAR-P9
CL:CHAR-LESSP8	CL:CHAR<9	CHARCODE2	CL:UPPER-CASE-P9

CONSTANT INDEX

CL:CHAR-BITS-LIMIT2	CL:CHAR-CONTROL-BIT2	CL:CHAR-HYPER-BIT2	CL:CHAR-SUPER-BIT2
CL:CHAR-CODE-LIMIT2	CL:CHAR-FONT-LIMIT2	CL:CHAR-META-BIT2	\CHARHI2

MACRO INDEX

%%CHAR-DOWNCASE-CODE9	%%CODE-CHAR10	DIGITCHARP2	UCASECODE2
%%CHAR-UPCASE-CODE10	ALPHACHARP2	SELCHARQ1	

VARIABLE INDEX

DWIMEQUIVLST2	PRETTYEQUIVLST2
---------------------	-----------------------

PROPERTY INDEX

CMLCHARACTER10

RECORD INDEX

CHARACTER2
