JFP **13** (1): 193–198, January 2003. © 2003 Cambridge University Press
DOI: 10.1017/S0956796803002119 Printed in the United Kingdom

Chapter 19

Character Utilities

```
module Char (
    isAscii, isLatin1, isControl, isPrint, isSpace, isUpper, isLower,
    isAlpha, isDigit, isOctDigit, isHexDigit, isAlphaNum,
    digitToInt, intToDigit,
    toUpper, toLower,
    ord, chr,
    readLitChar, showLitChar, lexLitChar,
        -- ...and what the Prelude exports
    Char, String
    ) where

isAscii, isLatin1, isControl, isPrint, isSpace, isUpper, isLower,
 isAlpha, isDigit, isOctDigit, isHexDigit, isAlphaNum :: Char -> Bool

toUpper, toLower        :: Char -> Char

digitToInt :: Char -> Int
intToDigit :: Int -> Char

ord        :: Char -> Int
chr        :: Int  -> Char

lexLitChar  :: ReadS String
readLitChar :: ReadS Char
showLitChar :: Char -> ShowS
```

This library provides a limited set of operations on the Unicode character set. The first 128 entries of this character set are identical to the ASCII set; with the next 128 entries comes the remainder of

the Latin-1 character set. This module offers only a limited view of the full Unicode character set; the full set of Unicode character attributes is not accessible in this library.

Unicode characters may be divided into five general categories: non-printing, lower case alphabetic, other alphabetic, numeric digits, and other printable characters. For the purposes of Haskell, any alphabetic character which is not lower case is treated as upper case (Unicode actually has three cases: upper, lower, and title). Numeric digits may be part of identifiers but digits outside the ASCII range are not used by the reader to represent numbers.

For each sort of Unicode character, here are the predicates which return `True`:

Character Type	Predicates
Lower Case Alphabetic	`isPrint` `isAlphaNum` `isAlpha` `isLower`
Other Alphabetic	`isPrint` `isAlphaNum` `isAlpha` `isUpper`
Digits	`isPrint` `isAlphaNum`
Other Printable	`isPrint`
Non-printing	

The `isDigit`, `isOctDigit`, and `isHexDigit` functions select only ASCII characters. `intToDigit` and `digitToInt` convert between a single digit `Char` and the corresponding `Int`. `digitToInt` operates fails unless its argument satisfies `isHexDigit`, but recognises both upper and lower-case hexadecimal digits (i.e. `'0'..'9'`, `'a'..'f'`, `'A'..'F'`). `intToDigit` fails unless its argument is in the range `0..15`, and generates lower-case hexadecimal digits.

The `isSpace` function recognizes only white characters in the Latin-1 range.

The function `showLitChar` converts a character to a string using only printable characters, using Haskell source-language escape conventions. The function `lexLitChar` does the reverse, returning the sequence of characters that encode the character. The function `readLitChar` does the same, but in addition converts the to the character that it encodes. For example:

```
showLitChar '\n' s       =  "\\n" ++ s
lexLitChar  "\\nHello"   =  [("\\n", "Hello")]
readLitChar "\\nHello"   =  [('\n', "Hello")]
```

Function `toUpper` converts a letter to the corresponding upper-case letter, leaving any other character unchanged. Any Unicode letter which has an upper-case equivalent is transformed. Similarly, `toLower` converts a letter to the corresponding lower-case letter, leaving any other character unchanged.

The `ord` and `chr` functions are `fromEnum` and `toEnum` restricted to the type `Char`.

https://doi.org/10.1017/S0956796803002119 Published online by Cambridge University Press

19.1 Library Char

```
module Char (
    isAscii, isLatin1, isControl, isPrint, isSpace, isUpper, isLower,
    isAlpha, isDigit, isOctDigit, isHexDigit, isAlphaNum,
    digitToInt, intToDigit,
    toUpper, toLower,
    ord, chr,
    readLitChar, showLitChar, lexLitChar,
        -- ...and what the Prelude exports
    Char, String
    ) where

import Array         -- Used for character name table.
import Numeric (readDec, readOct, lexDigits, readHex)
import UnicodePrims  -- Source of primitive Unicode functions.

-- Character-testing operations
isAscii, isLatin1, isControl, isPrint, isSpace, isUpper, isLower,
 isAlpha, isDigit, isOctDigit, isHexDigit, isAlphaNum :: Char -> Bool

isAscii c               =  c < '\x80'

isLatin1 c              =  c <= '\xff'

isControl c             =  c < ' ' || c >= '\DEL' && c <= '\x9f'

isPrint                 =  primUnicodeIsPrint

isSpace c               =  c `elem` " \t\n\r\f\v\xA0"
        -- Only Latin-1 spaces recognized

isUpper                 =  primUnicodeIsUpper  -- 'A'..'Z'

isLower                 =  primUnicodeIsLower  -- 'a'..'z'

isAlpha c               =  isUpper c || isLower c

isDigit c               =  c >= '0' && c <= '9'

isOctDigit c            =  c >= '0' && c <= '7'

isHexDigit c            =  isDigit c || c >= 'A' && c <= 'F' ||
                                        c >= 'a' && c <= 'f'

isAlphaNum              =  primUnicodeIsAlphaNum

-- Digit conversion operations
digitToInt :: Char -> Int
digitToInt c
  | isDigit c           =  fromEnum c - fromEnum '0'
  | c >= 'a' && c <= 'f' =  fromEnum c - fromEnum 'a' + 10
  | c >= 'A' && c <= 'F' =  fromEnum c - fromEnum 'A' + 10
  | otherwise           =  error "Char.digitToInt: not a digit"
```

https://doi.org/10.1017/S0956796803002119 Published online by Cambridge University Press

```
intToDigit :: Int -> Char
intToDigit i
  | i >= 0  && i <=  9   =  toEnum (fromEnum '0' + i)
  | i >= 10 && i <= 15   =  toEnum (fromEnum 'a' + i - 10)
  | otherwise            =  error "Char.intToDigit: not a digit"

-- Case-changing operations
toUpper :: Char -> Char
toUpper =  primUnicodeToUpper

toLower :: Char -> Char
toLower =  primUnicodeToLower

-- Character code functions
ord  :: Char -> Int
ord  =  fromEnum

chr  :: Int  -> Char
chr  =  toEnum

-- Text functions
readLitChar          :: ReadS Char
readLitChar ('\\':s) =  readEsc s
readLitChar (c:s)    =  [(c,s)]

readEsc          :: ReadS Char
readEsc ('a':s)  = [('\a',s)]
readEsc ('b':s)  = [('\b',s)]
readEsc ('f':s)  = [('\f',s)]
readEsc ('n':s)  = [('\n',s)]
readEsc ('r':s)  = [('\r',s)]
readEsc ('t':s)  = [('\t',s)]
readEsc ('v':s)  = [('\v',s)]
readEsc ('\\':s) = [('\\',s)]
readEsc ('"':s)  = [('"',s)]
readEsc ('\'':s) = [('\'',s)]
readEsc ('^':c:s) | c >= '@' && c <= '_'
                 = [(chr (ord c - ord '@'), s)]
readEsc s@(d:_) | isDigit d
                 = [(chr n, t) | (n,t) <- readDec s]
readEsc ('o':s)  = [(chr n, t) | (n,t) <- readOct s]
readEsc ('x':s)  = [(chr n, t) | (n,t) <- readHex s]
readEsc s@(c:_) | isUpper c
                 = let table = ('\DEL', "DEL") : assocs asciiTab
                   in case [(c,s') | (c, mne) <- table,
                                     ([],s') <- [match mne s]]
                      of (pr:_) -> [pr]
                         []     -> []
readEsc _        = []

match                        :: (Eq a) => [a] -> [a] -> ([a],[a])
match (x:xs) (y:ys) | x == y  =  match xs ys
match xs     ys               =  (xs,ys)
```

https://doi.org/10.1017/S0956796803002119 Published online by Cambridge University Press

```
showLitChar                :: Char -> ShowS
showLitChar c | c > '\DEL' =  showChar '\\' .
                              protectEsc isDigit (shows (ord c))
showLitChar '\DEL'         =  showString "\\DEL"
showLitChar '\\'           =  showString "\\\\"
showLitChar c | c >= ' '   =  showChar c
showLitChar '\a'           =  showString "\\a"
showLitChar '\b'           =  showString "\\b"
showLitChar '\f'           =  showString "\\f"
showLitChar '\n'           =  showString "\\n"
showLitChar '\r'           =  showString "\\r"
showLitChar '\t'           =  showString "\\t"
showLitChar '\v'           =  showString "\\v"
showLitChar '\SO'          =  protectEsc (== 'H') (showString "\\SO")
showLitChar c              =  showString ('\\' : asciiTab!c)

protectEsc p f             = f . cont
                             where cont s@(c:_) | p c = "\\&" ++ s
                                   cont s             = s
asciiTab = listArray ('\NUL', ' ')
           ["NUL", "SOH", "STX", "ETX", "EOT", "ENQ", "ACK", "BEL",
            "BS",  "HT",  "LF",  "VT",  "FF",  "CR",  "SO",  "SI",
            "DLE", "DC1", "DC2", "DC3", "DC4", "NAK", "SYN", "ETB",
            "CAN", "EM",  "SUB", "ESC", "FS",  "GS",  "RS",  "US",
            "SP"]

lexLitChar          :: ReadS String
lexLitChar ('\\':s) =  map (prefix '\\') (lexEsc s)
       where
         lexEsc (c:s)     | c `elem` "abfnrtv\\\"'" = [([c],s)]
         lexEsc ('^':c:s) | c >= '@' && c <= '_'    = [(['^',c],s)]
         -- Numeric escapes
         lexEsc ('o':s)              = [prefix 'o' (span isOctDigit s)]
         lexEsc ('x':s)              = [prefix 'x' (span isHexDigit s)]
         lexEsc s@(d:_)   | isDigit d = [span isDigit s]
         -- Very crude approximation to \XYZ.
         lexEsc s@(c:_)   | isUpper c = [span isCharName s]
         lexEsc _                     = []
         isCharName c   = isUpper c || isDigit c
         prefix c (t,s) = (c:t, s)

lexLitChar (c:s)    =  [([c],s)]
lexLitChar ""       =  []
```

https://doi.org/10.1017/S0956796803002119 Published online by Cambridge University Press