Internet READLINE implemented in C & Lisp

Gene Michael Stover

created Thursday, 2006 January 26
updated Sunday, 2006 February 19

Contents

1 What is this?

Often, you're writing a program that needs to read lines transmitted from another program. Lines of text on many Internet-based protocols are terminated

with an ASCII (carriage return, newline) sequence – or rahter, they should be. Your program has to stand alone, so you don't want to obtain the line-reading algorithm as part of a big library. What you really need is a simple function that's ready for you to copy-&-paste into your program.

At least, that's what I often need.

So here it is, implemented in C & Lisp.[1] It's licensed to be useful pretty much anywhere.

2 License

Here is a copy of the license agremeent that is on the source code. The same agreement appears in each source code file. It's the "X11 license" with my name replacing "the X Consortium".

> Copyright (c) 2006 Gene Michael Stover. All rights reserved.
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL GENE MICHAEL STOVER BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
>
> Except as contained in this notice, the name of Gene Michael Stover shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from Gene Michael Stover.

3 Requirements

Many protocols built on or for the Internet, such as FTP & HTTP, use a convention in which lines of text are transmitted as ASCII terminted with a carriage

[1] Maybe someday, I'll implement it in Delphi, too. It all depends on need.

return (CR, $0D_{16}$) & then a line feed (LF, $0A_{16}$). Programs implementing such protocols must correctly parse lines which end with CR LF, & they are encouraged to parse lines ending with just LF.

The command channel for FTP uses this convention, as do the request & reply headers of HTTP. Telnet & MUDs of every genre depend on it.[2]

We want a function which will read such lines from a suitable source & return strings with the end-of-line sequence removed.

1. The function assumes that the input is a stream of ASCII character.[3]
2. The function recognizes lines which end with a proper end-of-line sequence.
   (a) A two-character sequence of CR LF is always a proper end-of-line sequence.
   (b) A single LF is a proper end-of-line when in *lenient* mode, but it is unacceptable in *strict* mode.
   (c) An end-of-input condition is acceptable as an end-of-line sequence in *lenient* more, but it is unacceptable in *strict* mode.
   (d) The caller determines whether we're in *strict* or *lenient* mode.
3. The function protects the program from buffer overflows & other erroneous or malicious inputs.

Here are some secondary requirements, derived from the first set, plus some results of how I plan to implement the function:

1. The function never accepts a CR or LF inside the string. In other words, there is no way to escape the CR or LF so that it will be considered part of the line.
2. A CR that does not preceed an LF is always unacceptable as an end-of-line sequence.
3. A LF CR sequence is treated as a lone LF followed by a CR, & the CR will be an error unless it is followed by an LF.
4. To protect the program from buffer overflows, the caller may specify the maximum number of characters it is willing to accept in a string.
   (a) If an input line contains fewer than the maxmum number of characters, followed by a proper end-of-line sequence, the function stores all the characters, consumes & discards the end-of-line sequence, & reports success.

[2] MS-DOS uses this convention for its text files, but don't mistake a non-re-entrant program loader as a communication protocol for the Internet.

[3] I say ASCII, & I mean ASCII. I don't mean the native character set, though some implementations of the function may assume that the native character set is ASCII.

(b) If an input line contains exactly the maximum number of characters, followed by an acceptable end-of-line sequence, the function stores the entire line, consumes & discards the end-of-line sequence, then reports success.

(c) If an input line exceeds the maximum number of characters, the function may consume at least the maximum number of characters, discard none of them, & report an error.

5. The function will signal errors or end-of-input with a return value, not by throwing an exception.

6. The input source must be a file handle, socket, or "stream" following the convention of the implementation language. In other words, on unix-like systems & on most/all(?) modern Lisps, the function may be used to read from files, other programs, or connection-oriented network sources.

7. On an incorrect end-of-line sequence, characters may be consumed & discarded.[4]

4 Examples

4.1 Empty input

When reading from an input that is already at its end, the function indicates end-of-input.

4.2 Correct input in strict mode

Assume the input contains "abc", followed by a proper end-of-line sequence, followed by "de", followed by an end-of-line sequence. Like this:

```
$ hexdump -C $HOME/tmp/a.txt
00000000  61 62 63 0d 0a 64 65 0d  0a                        |abc..de..|
00000009
$
```

The first time we call *internet-read-line* on that input stream (assuming the maximum line length is greater than three), the function gives us "abc". The next time, it returns "de". The third time, it indicates end-of-input.

[4] My opinions about syntactically incorrect input in this case & others is that, after such an error, all bets are off, so it doesn't matter what the parser did to the characters in the immediate vicinity of the error.

4.3 Lines that are too long

Assume that the maximum line length we'll accept is three & the input contains "abcde", followed by a proper end-of-line sequence, followed by "fghi", followed by an end-of-line sequence. Like this:

```
$ hexdump -C $HOME/tmp/a.txt
00000000  61 62 63 64 65 0d 0a 66  67 68 69 0d 0a           |abcde..fghi..|
0000000d
$
```

Notice that the input contains two lines, but both are longer than the maximum we wil accept.

The first time we call *internet-read-line* on that input stream (assuming the maximum line length is three), indicates an error.

The function probably consumes the "abc" & the "d"; it is permitted to consume anything because all bets are off after an erroneous input. Strictly speaking, the caller should not call *internet-readline* on that input source again, but if it did, *internet-readline* might return "e". If the caller used *internet-readline* on the input again, the function would indicate an error because "fghi" is too long.

This example assumed that the maximum line length accepted by the caller is three. If the caller had used a maximum line length of five or more, there would have been no errors. With a maximum line length of five or more, *internet-readline* would have returned "abcde" on the fist call & "fghi" on the second call. It would have indicated end-of-input for the third call.

5 Pseudocode

Inputs to the function are

strm The input source. We can read octets or characters (as appropriate to the language) from it, one at a time.

max A positive integer. It is the maximum number of characters the caller is willing to accept on a line.

is-strict A Boolean. When true, the function requires a CR LF sequence to end a string. Otherwise, the function accepts CR LF, but it also accepts a single LF as an end-of-line sentinel.

The returned value is a new, dynamically allocated string. On error or end-of-input, the return value is `nil`. The function does not toss any exceptions.

This algorithm assumes "short circuit" evaluation.

Here is the algorithm:

1. If *strm* is at its end, there is no more input. Return `nil`.
2. Create an empty string into which we may accumulate characters.

3. Until the next character is CR or LF, or end-of-input, or $string'slength \geq max$, read & accumulate characters into the string.

4. If the $string'slength \geq max$, the string is full, so we can't attempt to read the terminator. So return the string.

5. Let C0 be the next input character. (Consume the character.)

6. If (not *is-strict*) & C0 = LF, we're good. Return the string.

7. If (not *is-strict*) & C0 indicates end-of-input, we're good. Return the string.

8. If C0 = CR & next character = LF, we're good. Return the string.

9. Otherwise, the terminator is not acceptable. Return `nil`.

6 Lisp implementations

This Lisp implementation works whether or not the computer's native character set is ASCII. If you're reading from a stream whose *element-type* is character, it assumes the internal character set is ASCII. If you're reading from a stream with another *element-type* (namely `(unsigned-byte 8)`), it does not assume the internal character set is ASCII.

The *is-strict* & *max* arguments are keywords with defaults.

The function & its test programs are in a file called `nrdl.lisp`. That file is in Appendix A. The INTERNET-READLINE & INTERNET-READALL functions only use symbols from Common Lisp, but the test programs require my Lisp unit test framework [1] and my permutation utilities for Lisp [2].

There is an INTERNET-READLINE function which reads one line each time you call it. There is also a INTERNET-READALL function which reads & returns a list of all the lines from the input.

Here is the source code for the functions, embedded in a great big LABELS form:

```
(labels
    ((nrdl (read-code to-char &key (is-strict nil) (max 1024))
       (declare (type function read-code to-char) (type integer max))
       (assert (plusp max))
       (labels ((xread-code () (funcall read-code))
(xto-char (code)
                  (declare (type integer code))
                  (funcall to-char code))
(is-end (string code)
  (declare (type string string))
                  (or (>= (length string) max)
      (member code '(#x0D #x0A nil))))
(is-good-terminator (code)
  (or
```

```
   (and (not is-strict) (member code '(nil #x0A)))
   (and (eql code #x0D) (eql (xread-code) #x0A)))))
 (let ((first-code (xread-code)))
   (and first-code
(do ((x (make-array max :element-type 'character
    :adjustable nil :fill-pointer 0))
     (code first-code (xread-code)))
    ((is-end x code)
     (and (is-good-terminator code) x))
    (vector-push (xto-char code) x))))))
     (ascii-char (ascii)
       ;; If the ASCII value can't be converted to a native
       ;; character, return that value unchanged. Otherwise,
       ;; return a character which corresonds to the ASCII value.
       ;; This is analogous to CODE-CHAR."
       (case ascii
 ;; (0 #\Nul)
 ;; (1 #\Soh)
 ;; (2 #\Stx)
 ;; (3 #\Etx)
 ;; (4 #\Eot)
 ;; (5 #\Enq)
 ;; (6 #\Ack)
 ;; (7 #\Bel)
 (8 #\Backspace)
 (9 #\Tab)
 ;; (10 #\Newline)
 ;; (11 #\Vt)
 ;; (12 #\Page)
 ;; (13 #\Return)
 ;; (14 #\So)
 ;; (15 #\Si)
 ;; (16 #\Dle)
 ;; (17 #\Dc1)
 ;; (18 #\Dc2)
 ;; (19 #\Dc3)
 ;; (20 #\Dc4)
 ;; (21 #\Nak)
 ;; (22 #\Syn)
 ;; (23 #\Etb)
 ;; (24 #\Can)
 ;; (25 #\Em)
 ;; (26 #\Sub)
 ;; (27 #\Esc)
 ;; (28 #\Fs)
 ;; (29 #\Gs)
 ;; (30 #\Rs)
 ;; (31 #\Us)
 (32 #\Space)
 (33 #\!)
```

```
(34 #\")
(35 #\#)
(36 #\$)
(37 #\%)
(38 #\&)
(39 #\')
(40 #\()
(41 #\))
(42 #\*)
(43 #\+)
(44 #\,)
(45 #\-)
(46 #\.)
(47 #\/)
(48 #\0)
(49 #\1)
(50 #\2)
(51 #\3)
(52 #\4)
(53 #\5)
(54 #\6)
(55 #\7)
(56 #\8)
(57 #\9)
(58 #\:)
(59 #\;)
(60 #\<)
(61 #\=)
(62 #\>)
(63 #\?)
(64 #\@)
(65 #\A)
(66 #\B)
(67 #\C)
(68 #\D)
(69 #\E)
(70 #\F)
(71 #\G)
(72 #\H)
(73 #\I)
(74 #\J)
(75 #\K)
(76 #\L)
(77 #\M)
(78 #\N)
(79 #\O)
(80 #\P)
(81 #\Q)
(82 #\R)
(83 #\S)
```

```
 (84 #\T)
 (85 #\U)
 (86 #\V)
 (87 #\W)
 (88 #\X)
 (89 #\Y)
 (90 #\Z)
 (91 #\[)
 (92 #\\)
 (93 #\])
 (94 #\^)
 (95 #\_)
 (96 #\`)
 (97 #\a)
 (98 #\b)
 (99 #\c)
 (100 #\d)
 (101 #\e)
 (102 #\f)
 (103 #\g)
 (104 #\h)
 (105 #\i)
 (106 #\j)
 (107 #\k)
 (108 #\l)
 (109 #\m)
 (110 #\n)
 (111 #\o)
 (112 #\p)
 (113 #\q)
 (114 #\r)
 (115 #\s)
 (116 #\t)
 (117 #\u)
 (118 #\v)
 (119 #\w)
 (120 #\x)
 (121 #\y)
 (122 #\z)
 (123 #\{)
 (124 #\|)
 (125 #\})
 (126 #\~)
 ;; (127 #\Rubout)
 ;; Finally, make a valiant attempt to convert to a native
 ;; character.  If that doesn't work, return ASCII (which
 ;; is almost sure to cause problems because it is not a
 ;; character).
 (otherwise (or (ignore-errors (code-char ascii))
ascii)))))
```

```
  (defun internet-readline (strm &key (is-strict nil) (max 1024))
    (case (stream-element-type strm)
      (character (nrdl #'(lambda ()
   (let ((x (read-char strm nil)))
     (and x (char-code x))))
       #'code-char
       :is-strict is-strict
       :max max))
      (otherwise (nrdl #'(lambda () (read-byte strm nil))
       #'ascii-char
       :is-strict is-strict
       :max max))))

  (defun internet-readall (pn &key (is-strict nil) (max 1024)
      (element-type 'character))
    (with-open-file (strm pn :element-type element-type)
      (labels
  ((next () (internet-readline strm :is-strict is-strict :max max)))
(do ((lst nil (cons line lst))
     (line (next) (next)))
    ((null line) (nreverse lst)))))))
```

To use it in your own programs, you could LOAD the `nrdl.lisp` file, or you could copy-&-paste the whole LABELS form into your program. Either way should work.

7 C implementation

The C `internet_readline` function assumes the native character set is ASCII.

It uses Standard C/89 functions with two exceptions. Instead of `malloc` & `free`, it uses XMALLOC & XFREE. I picked up the habit of XMALLOC & XFREE from some Gnu source code. If your program calls XMALLOC & XFREE, you are free to change the memory allocation system easily. I often implement `xmalloc` as a wrapper about the Boehm collector's `GC_alloc`. Also, I often implement `xmalloc` to abort if we're out of memory. You could implement `xmalloc` as a faster allocator or to align memory on more restrictive boundaries or whatever.

Anyway, it uses Standard C functions, plus `xmalloc` & `xfree`.

The function & its tes programs (which I never got around to writing) are in a file called `nrdl0.c`. That file is in Appendix B.

The `internet_readline` function reads one line each time you call it. I didn't write a C version of `internet_readall`, as I did for Lisp.

Here is the source code for the C function:

```
/*
 */
static Boolean
is_good_terminator (FILE *fp, int c, int is_strict, size_t i,
```

```
                    size_t max)
{
  return (!is_strict && (c == -1 || c == 0x0A)) ||
    (c == 0x0D && fgetc (fp) == 0x0A);
}

/*
 */
char *
internet_readline (FILE *fp, Boolean is_strict, size_t max)
{
  char *x = NULL;
  int c, i = 0;

  c = fgetc (fp);
  if (c == -1) {
    /*
     * When the input source is already at end-of-file,
     * we return NULL.
     */
    assert (x == NULL);
  } else {
    /*
     * The input is not already empty, so we allocate space &
     * accumulate characters.
     */
    x = (char *) xmalloc (max + 1);
    while (c != -1 && c != 0x0D && c != 0x0A && i < max) {
      x[i++] = c;
      c = fgetc (fp);
    }
    if (is_good_terminator (fp, c, is_strict, i, max)) {
      x[i] = '\0';
    } else {
      /*
       * There was an error, so we ditch the memory we've
       * allocated.  We'll return NULL.
       */
      x = (char *) xfree (x);
    }
  }
  return x;
}
```

To use it in your own programs, copy-&-paste these two functions into your own program. You may need to prototype them in a header file, too.

8 Delphi implementation

fixme: Is this worth the time? If so, it must work whether or not I/O exceptions are enabled at compile time. And no, I don't need two versions of the function for those two situations. One function can do both without duplicating code.

A nrdl.lisp

```
;;; -*- Mode: Lisp -*-
;;;
;;; $Header: /home/gene/library/website/docsrc/nrdl/RCS/nrdl.tex,v 395.1 2008/04/20 17:25:50 gene Exp
;;;
;;; Copyright (c) 2006 Gene Michael Stover.  All rights reserved.
;;;
;;; Permission is hereby granted, free of charge, to any person obtaining
;;; a copy of this software and associated documentation files (the
;;; "Software"), to deal in the Software without restriction, including
;;; without limitation the rights to use, copy, modify, merge, publish,
;;; distribute, sublicense, and/or sell copies of the Software, and to
;;; permit persons to whom the Software is furnished to do so, subject to
;;; the following conditions:
;;;
;;; The above copyright notice and this permission notice shall be
;;; included in all copies or substantial portions of the Software.
;;;
;;; THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
;;; EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
;;; MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
;;; NONINFRINGEMENT. IN NO EVENT SHALL GENE MICHAEL STOVER BE LIABLE FOR
;;; ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
;;; CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
;;; WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
;;;
;;; Except as contained in this notice, the name of Gene Michael Stover
;;; shall not be used in advertising or otherwise to promote the sale, use
;;; or other dealings in this Software without prior written authorization
;;; from Gene Michael Stover.
;;;

(defpackage "COM.CYBERTIGGYR.NRDL"
  (:use "COMMON-LISP")
  (:import-from "CYBERTIGGYR-TEST" check deftest)
  (:import-from "COM.CYBERTIGGYR.PRM" make-permutator maperm))
(in-package "COM.CYBERTIGGYR.NRDL")

;;;
;;; To use INTERNET-READLINE in another program, you could load
;;; this whole file, "nrdl.lisp", or you could copy the following
;;; LABELS form into your program.  The LABELS form is the self-
```

```
;;; contained INTERNET-READLINE function, though the DEFUN form
;;; which actually creates INTERNET-READLINE isn't.
;;;

(labels
    ((nrdl (read-code to-char &key (is-strict nil) (max 1024))
       (declare (type function read-code to-char) (type integer max))
       (assert (plusp max))
       (labels ((xread-code () (funcall read-code))
(xto-char (code)
                  (declare (type integer code))
                  (funcall to-char code))
(is-end (string code)
  (declare (type string string))
                  (or (>= (length string) max)
      (member code '(#x0D #x0A nil))))
(is-good-terminator (code)
  (or
   (and (not is-strict) (member code '(nil #x0A)))
   (and (eql code #x0D) (eql (xread-code) #x0A)))))
 (let ((first-code (xread-code)))
   (and first-code
(do ((x (make-array max :element-type 'character
    :adjustable nil :fill-pointer 0))
     (code first-code (xread-code)))
    ((is-end x code)
     (and (is-good-terminator code) x))
    (vector-push (xto-char code) x))))))
     (ascii-char (ascii)
       ;; If the ASCII value can't be converted to a native
       ;; character, return that value unchanged. Otherwise,
       ;; return a character which corresonds to the ASCII value.
       ;; This is analogous to CODE-CHAR."
       (case ascii
 ;; (0 #\Nul)
 ;; (1 #\Soh)
 ;; (2 #\Stx)
 ;; (3 #\Etx)
 ;; (4 #\Eot)
 ;; (5 #\Enq)
 ;; (6 #\Ack)
 ;; (7 #\Bel)
 (8 #\Backspace)
 (9 #\Tab)
 ;; (10 #\Newline)
 ;; (11 #\Vt)
 ;; (12 #\Page)
 ;; (13 #\Return)
 ;; (14 #\So)
 ;; (15 #\Si)
```

```
;; (16 #\Dle)
;; (17 #\Dc1)
;; (18 #\Dc2)
;; (19 #\Dc3)
;; (20 #\Dc4)
;; (21 #\Nak)
;; (22 #\Syn)
;; (23 #\Etb)
;; (24 #\Can)
;; (25 #\Em)
;; (26 #\Sub)
;; (27 #\Esc)
;; (28 #\Fs)
;; (29 #\Gs)
;; (30 #\Rs)
;; (31 #\Us)
(32 #\Space)
(33 #\!)
(34 #\")
(35 #\#)
(36 #\$)
(37 #\%)
(38 #\&)
(39 #\')
(40 #\()
(41 #\))
(42 #\*)
(43 #\+)
(44 #\,)
(45 #\-)
(46 #\.)
(47 #\/)
(48 #\0)
(49 #\1)
(50 #\2)
(51 #\3)
(52 #\4)
(53 #\5)
(54 #\6)
(55 #\7)
(56 #\8)
(57 #\9)
(58 #\:)
(59 #\;)
(60 #\<)
(61 #\=)
(62 #\>)
(63 #\?)
(64 #\@)
(65 #\A)
```

```
(66 #\B)
(67 #\C)
(68 #\D)
(69 #\E)
(70 #\F)
(71 #\G)
(72 #\H)
(73 #\I)
(74 #\J)
(75 #\K)
(76 #\L)
(77 #\M)
(78 #\N)
(79 #\O)
(80 #\P)
(81 #\Q)
(82 #\R)
(83 #\S)
(84 #\T)
(85 #\U)
(86 #\V)
(87 #\W)
(88 #\X)
(89 #\Y)
(90 #\Z)
(91 #\[)
(92 #\\)
(93 #\])
(94 #\^)
(95 #\_)
(96 #\`)
(97 #\a)
(98 #\b)
(99 #\c)
(100 #\d)
(101 #\e)
(102 #\f)
(103 #\g)
(104 #\h)
(105 #\i)
(106 #\j)
(107 #\k)
(108 #\l)
(109 #\m)
(110 #\n)
(111 #\o)
(112 #\p)
(113 #\q)
(114 #\r)
(115 #\s)
```

```
 (116 #\t)
 (117 #\u)
 (118 #\v)
 (119 #\w)
 (120 #\x)
 (121 #\y)
 (122 #\z)
 (123 #\{)
 (124 #\|)
 (125 #\})
 (126 #\~)
 ;; (127 #\Rubout)
 ;; Finally, make a valiant attempt to convert to a native
 ;; character.  If that doesn't work, return ASCII (which
 ;; is almost sure to cause problems because it is not a
 ;; character).
 (otherwise (or (ignore-errors (code-char ascii))
ascii)))))

  (defun internet-readline (strm &key (is-strict nil) (max 1024))
    (case (stream-element-type strm)
      (character (nrdl #'(lambda ()
   (let ((x (read-char strm nil)))
     (and x (char-code x))))
       #'code-char
       :is-strict is-strict
       :max max))
      (otherwise (nrdl #'(lambda () (read-byte strm nil))
       #'ascii-char
       :is-strict is-strict
       :max max))))

  (defun internet-readall (pn &key (is-strict nil) (max 1024)
      (element-type 'character))
    (with-open-file (strm pn :element-type element-type)
      (labels
  ((next () (internet-readline strm :is-strict is-strict :max max)))
(do ((lst nil (cons line lst))
     (line (next) (next)))
    ((null line) (nreverse lst)))))))

;;;
;;; TESTS
;;;

(deftest test0000 ()
  "Null test.  Always succeeds."
  'test0000)

;;;
```

```
;;; These tests require support functions.
;;; Support functions for the rest of the tests.
;;; Normally, I believe tests are best if they are
;;; simple & self-contained because it makes them
;;; less succeptable to bugs.  (Tests are for
;;; detecting bugs in another program.  If the
;;; possibility of bugs in the tests themselves is
;;; high...)  I'm breaking that rule this time.
;;;

(defun basic-test (is-strict max element-type octets expected)
  (let ((pn (make-pathname :directory '(:relative)
   :name "test" :type "txt")))
    ;; Create the input file
    (with-open-file (strm pn :element-type '(unsigned-byte 8)
  :direction :output :if-exists :rename-and-delete)
      (dolist (ascii octets) (write-byte ascii strm)))
    ;; Read from the input file.
    (let ((x (internet-readall pn :is-strict is-strict :max max
       :element-type element-type)))
      (if (equal x expected)
  (delete-file pn)
;; Else, Retain the file for the programmer to inspect
;; later.  Also, print an error message.
(progn
  (format t "~&~A returned ~S." 'internet-readall x)
  (format t "~&Expected ~S." expected)))
      (equal x expected))))

(deftest test0100 ()
  "Read a single, short line terminated with CR LF.  Strict.
Element type is character."
  (basic-test t 3 'character '(#x42 #x0D #x0A) '("B")))

(deftest test0103 ()
  "Read a single, long (but legal) line terminated with CR LF.  Strict.
Element type is character."
  (basic-test t 3 'character '(#x41 #x42 #x43 #x0D #x0A) '("ABC")))

(deftest test0106 ()
  "Read a three lines, each terminated with CR LF.  Strict.
Element type is character."
  (basic-test t 3 'character
      '(#x41 #x0D #x0A          ; "A" CR LF
                #x42 #x43 #x0D #x0A     ; "BC" CR LF
#x44 #x45 #x46 #x0D #x0A) ; "DEF" CR LF
      '("A" "BC" "DEF")))

(deftest test0110 ()
  "Read a single, short line terminated with CR LF.  Strict.
```

```
Element type is octet."
  (basic-test t 3 '(unsigned-byte 8) '(#x42 #x0D #x0A) '("B")))

(deftest test0113 ()
  "Read a single, long (but legal) line terminated with CR LF.  Strict.
Element type is octet."
  (basic-test t 3 '(unsigned-byte 8) '(#x41 #x42 #x43 #x0D #x0A) '("ABC")))

(deftest test0116 ()
  "Read a three lines, each terminated with CR LF.  Strict.
Element type is octet."
  (basic-test t 3 '(unsigned-byte 8)
      '(#x41 #x0D #x0A          ; "A" CR LF
                #x42 #x43 #x0D #x0A     ; "BC" CR LF
#x44 #x45 #x46 #x0D #x0A) ; "DEF" CR LF
      '("A" "BC" "DEF")))

(defun make-lisp-test (eol len is-strict element-type)
  (let ((testname (read-from-string
   (remove #\: (format nil "test-~A-~A-~A-~A"
       eol len
       (if is-strict 'strict 'lenient)
       (if (eq element-type 'character)
   'character 'octet))))))
    `(deftest ,testname ()
       (basic-test ,is-strict
   3                    ; max
   ',element-type
   ',(append (ecase len
       (:len-zero nil)
       (:len-short '(#x41))
       (:len-max '(#x41 #x42 #x43))
       (:len-longer '(#x41 #x42 #x43 #x44)))
     (ecase eol
       (:eol-nil nil)
       (:eol-cr '(#x0D))
       (:eol-lf '(#x0A))
       (:eol-crlf '(#x0D #x0A))))
   ',(cond ((and is-strict (not (eq eol :eol-crlf))) nil)
   ((eq eol :eol-cr) nil)
   ((eq len :len-longer) nil)
   ((and (eq eol :eol-nil) (eq len :len-zero)) nil)
   (t
    (ecase len
      (:len-zero '(""))
      (:len-short '("A"))
      (:len-max '("ABC"))
      (:len-longer nil)))))))

(defmacro def-lisp-tests ()
```

```
  (cons
   'progn
   (let ((prm (make-permutator '(:eol-crlf :eol-nil :eol-lf :eol-cr)
      '(:len-zero :len-short :len-max :len-longer)
      '(t nil)
      '(character (unsigned-byte 8)))))
     (declare (type function prm))
     (do ((x (funcall prm) (funcall prm))
  (lst nil (cons (apply #'make-lisp-test x) lst)))
 ((null x) (nreverse lst))))))


;;; --- end of file ---
```

B nrdl0.c

```
/* -*- Mode: C -*-
 *
 * $Header: /home/gene/library/website/docsrc/nrdl/RCS/nrdl.tex,v 395.1 2008/04/20 17:25:50 gene Exp $
 *
 * Copyright (c) 2006 Gene Michael Stover.  All rights reserved.
 *
 * Permission is hereby granted, free of charge, to any person obtaining
 * a copy of this software and associated documentation files (the
 * "Software"), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL GENE MICHAEL STOVER BE LIABLE FOR
 * ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 *
 * Except as contained in this notice, the name of Gene Michael Stover
 * shall not be used in advertising or otherwise to promote the sale, use
 * or other dealings in this Software without prior written authorization
 * from Gene Michael Stover.
 */

/* Standard C */
#include <assert.h>
```

```
#include <ctype.h>
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

/* This */
typedef int Boolean;
#ifndef FALSE
#define FALSE (0)
#endif
#ifndef TRUE
#define TRUE (!0)
#endif

/*
 */
void *
xmalloc (size_t count)
{
  void *p;

  p = malloc (count);
  if (p == NULL) {
    fprintf (stderr, "\n%s:%d: malloc (%u) failed", __FILE__, __LINE__,
             (unsigned) count);
    abort ();
  }
  return p;
}

/*
 */
static void *
xfree (void *p)
{
  if (p == NULL) {
    fprintf (stderr, "\n%s:%d: Can't free NULL", __FILE__, __LINE__);
    abort ();
  }
  free (p);
  return NULL;
}

/*
 * To use in your own programs, copy-&-paste everything
 * between the "cut here" comment lines.
 */
```

```
/* --- cut here --- begin --- cut here --- */
/*
 */
static Boolean
is_good_terminator (FILE *fp, int c, int is_strict, size_t i,
                    size_t max)
{
  return (!is_strict && (c == -1 || c == 0x0A)) ||
    (c == 0x0D && fgetc (fp) == 0x0A);
}

/*
 */
char *
internet_readline (FILE *fp, Boolean is_strict, size_t max)
{
  char *x = NULL;
  int c, i = 0;

  c = fgetc (fp);
  if (c == -1) {
    /*
     * When the input source is already at end-of-file,
     * we return NULL.
     */
    assert (x == NULL);
  } else {
    /*
     * The input is not already empty, so we allocate space &
     * accumulate characters.
     */
    x = (char *) xmalloc (max + 1);
    while (c != -1 && c != 0x0D && c != 0x0A && i < max) {
      x[i++] = c;
      c = fgetc (fp);
    }
    if (is_good_terminator (fp, c, is_strict, i, max)) {
      x[i] = '\0';
    } else {
      /*
       * There was an error, so we ditch the memory we've
       * allocated.  We'll return NULL.
       */
      x = (char *) xfree (x);
    }
  }
  return x;
}
/* --- cut here --- end --- cut here */
```

```
/*
 * Null test.  Always succeeds.
 */
static Boolean
S_Test0000 ()
{
  return TRUE;
}

/*
 */
static Boolean
S_BasicTest (Boolean is_strict, size_t max, Boolean is_binary,
             char octets[], size_t octets_len, char *expected[])
{
  Boolean is_good = FALSE;
  char pathname[] = "test.txt";
  FILE *fp;
  size_t i;
  char *line;

  /* Create the input file */
  fp = fopen (pathname, "wb");
  if (fp != NULL) {
    if (octets_len == 0 || fwrite (octets,
                                   octets_len * sizeof octets[0],
                                   1, fp) == 1) {
      is_good = TRUE;
    } else {
      printf ("\n");
      perror ("fwrite");
      printf ("%s:%d:", __FILE__, __LINE__);
      printf (" Did not write the list of %u octets", octets_len);
      printf (" to the temporary file.");
      is_good = FALSE;
    }
    fclose (fp);
  } else {
    printf ("\n");
    perror ("fopen");
    printf ("%s:%d:", __FILE__, __LINE__);
    printf (" Could not create the temorary file, %s.", pathname);
    is_good = FALSE;
  }
  if (is_good) {
    /* Read the input file. */
    fp = fopen (pathname, is_binary ? "rb" : "r");
    if (fp != NULL) {
      i = 0;
```

```
      line = internet_readline (fp, is_strict, max);
      while (line != NULL && expected[i] != NULL &&
             strcmp (expected[i], line) == 0) {
        line = (char *) xfree (line);
        ++i;
        line = internet_readline (fp, is_strict, max);
      }
      is_good = expected[i] == NULL && line == NULL;
      if (line != NULL) {
        line = (char *) xfree (line);
      }
      fclose (fp);
    } else {
      printf ("\n");
      perror ("fopen");
      printf ("\n%s:%d:", __FILE__, __LINE__);
      printf (" Could not open %s to read from it.", pathname);
      is_good = FALSE;
    }
  }
  if (is_good) {
    remove (pathname);
  } else {
    /*
     * There was an error, so we don't delete the temporary
     * file.  It might be useful to inspect it later.
     */
  }
  return is_good;
}

/*
 */
#define entry(test) { &test, #test }
static struct {
  int (*fn) ();
  char *name;
} S_a[] = {
  entry (S_Test0000),
  { NULL, NULL }
};
static size_t S_alen = sizeof S_a / sizeof S_a[0];

int
main ()
{
  int rc = 0;
  size_t i;

  for (i = 0; rc == 0 && S_a[i].fn != NULL; ++i) {
```

```
    printf ("\n%3d%% %s =>", i * 100 / (S_alen - 1),
            S_a[i].name);
    if ((*S_a[i].fn) ()) {
      printf (" good");
    } else {
      printf (" FAIL");
      rc = 2057;
    }
  }
  printf ("\n");
  return rc == 0 ? EXIT_SUCCESS : EXIT_FAILURE;
}

/* --- end of file --- */
```

C Other File Formats

- This document is available in multi-file HTML format at http://cybertiggyr.com/gene/nrdl/.
- This document is available in Pointless Document Format (PDF) at http://cybertiggyr.com/gene/nrdl/nrdl.pdf.

References

[1] Gene Michael Stover. My lisp unit test framework. *cybertiggyr.com/gene*, June 2005. http://cybertiggyr.com/gene/lut/.

[2] Gene Michael Stover. Permutation utilities in lisp. *cybertiggyr.com/gene*, February 2006. http://cybertiggyr.com/gene/prm/.