```
#|

 Copyright (C) 1994 by Yuan Yu.  All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Yuan Yu PROVIDES ABSOLUTELY NO WARRANTY.  THE EVENT SCRIPT IS PROVIDED "AS IS"
 WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT
 LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
 SCRIPT IS WITH YOU.  SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF
 ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

 IN NO EVENT WILL Yuan Yu BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS,
 LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT
 OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS
 OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD
 PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR
 FOR ANY CLAIM BY ANY OTHER PARTY.

|#
```

EVENT: Start with the library "`strlen`" using the compiled version.

```
;          Proof of the Correctness of the STRNCMP Function
#|
This is part of our effort to verify the Berkeley string library.  The
Berkeley string library is widely used as part of the Berkeley Unix OS.

This is the source code of strncmp function in the Berkeley string library.

/* compare at most char s1[] to char s2[] */
int
strncmp(s1, s2, n)
register const char *s1, *s2;
register size_t n;
{

if (n == 0)
return (0);
```

```
do {
if (*s1 != *s2++)
return (*(unsigned char *)s1 - *(unsigned char *)--s2);
if (*s1++ == 0)
break;
} while (--n != 0);
return (0);
}
```

The MC68020 assembly code of the C function strncmp on SUN-3 is given as follows. This binary is generated by "gcc -O".

```
0x2608 <strncmp>:       linkw fp,#0
0x260c <strncmp+4>:     moveml d2-d3,sp@-
0x2610 <strncmp+8>:     moveal fp@(8),a0
0x2614 <strncmp+12>:    moveal fp@(12),a1
0x2618 <strncmp+16>:    movel fp@(16),d0
0x261c <strncmp+20>:    beq 0x2642 <strncmp+58>
0x261e <strncmp+22>:    andil #255,d1
0x2624 <strncmp+28>:    andil #255,d2
0x262a <strncmp+34>:    moveb a0@,d3
0x262c <strncmp+36>:    cmpb a1@+,d3
0x262e <strncmp+38>:    beq 0x263a <strncmp+50>
0x2630 <strncmp+40>:    moveb a0@,d1
0x2632 <strncmp+42>:    moveb a1@-,d2
0x2634 <strncmp+44>:    movel d1,d0
0x2636 <strncmp+46>:    subl d2,d0
0x2638 <strncmp+48>:    bra 0x2644 <strncmp+60>
0x263a <strncmp+50>:    tstb a0@+
0x263c <strncmp+52>:    beq 0x2642 <strncmp+58>
0x263e <strncmp+54>:    subl #1,d0
0x2640 <strncmp+56>:    bne 0x262a <strncmp+34>
0x2642 <strncmp+58>:    clrl d0
0x2644 <strncmp+60>:    moveml fp@(-8),d2-d3
0x264a <strncmp+66>:    unlk fp
0x264c <strncmp+68>:    rts
```

The machine code of the above program is:

```
<strncmp>:     0x4e56  0x0000  0x48e7  0x3000  0x206e  0x0008  0x226e  0x000c
<strncmp+16>:  0x202e  0x0010  0x6724  0x0281  0x0000  0x00ff  0x0282  0x0000
<strncmp+32>:  0x00ff  0x1610  0xb619  0x670a  0x1210  0x1421  0x2001  0x9082
<strncmp+48>:  0x600a  0x4a18  0x6704  0x5380  0x66e8  0x4280  0x4cee  0x000c
<strncmp+64>:  0xfff8  0x4e5e  0x4e75
```

```
'(78      86      0       0       72      231     48      0
  32      110     0       8       34      110     0       12
  32      46      0       16      103     36      2       129
  0       0       0       255     2       130     0       0
  0       255     22      16      182     25      103     10
  18      16      20      33      32      1       144     130
  96      10      74      24      103     4       83      128
  102     232     66      128     76      238     0       12
  255     248     78      94      78      117)
|#

; in the logic, the above program is defined by (strncmp-code).
```

DEFINITION:
STRNCMP-CODE
= `'(78 86 0 0 72 231 48 0 32 110 0 8 34 110 0 12 32 46 0 16 103 36 2 129 0 0 0 255 2 130 0 0 0 255 22 16 182 25 103 10 18 16 20 33 32 1 144 130 96 10 74 24 103 4 83 128 102 232 66 128 76 238 0 12 255 248 78 94 78 117)`

CONSERVATIVE AXIOM: strncmp-load
strncmp-loadp (s)
= (evenp (STRNCMP-ADDR)
  $\wedge$ (STRNCMP-ADDR $\in$ **N**)
  $\wedge$ nat-rangep (STRNCMP-ADDR, 32)
  $\wedge$ rom-addrp (STRNCMP-ADDR, mc-mem (s), 70)
  $\wedge$ mcode-addrp (STRNCMP-ADDR, mc-mem (s), STRNCMP-CODE))

Simultaneously, we introduce the new function symbols *strncmp-loadp* and *strncmp-addr*.

THEOREM: stepn-strncmp-loadp
strncmp-loadp (stepn (s, n)) = strncmp-loadp (s)

```
; the computation time of the program.
```

DEFINITION:
strncmp1-t (i, n, *lst1*, *lst2*)
= **if** get-nth (i, *lst1*) = get-nth (i, *lst2*)
  then if get-nth (i, *lst1*) = 0 **then** 9
    elseif ($n - 1$) = 0 **then** 11
    else splus (7, strncmp1-t ($1 + i$, $n - 1$, *lst1*, *lst2*)) **endif**
  else 11 **endif**

DEFINITION:
strncmp-t (n, *lst1*, *lst2*)
= **if** $n \simeq$ 0 **then** 10
  else splus (8, strncmp1-t (0, n, *lst1*, *lst2*)) **endif**

```
; an induction hint.
```

DEFINITION:
strncmp-induct (s, i*, i, n, *lst1*, *lst2*)
= **if** get-nth (i, *lst1*) = get-nth (i, *lst2*)
  then if get-nth (i, *lst1*) = 0 **then t**
    elseif ($n - 1$) = 0 **then t**
    else strncmp-induct (stepn (s, 7),
                add (32, i*, 1),
                $1 + i$,
                $n - 1$,
                lst1,
                lst2) **endif**
  else t endif

```
; the preconditions of the initial state.
```

DEFINITION:
strncmp-statep (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, n)
= ((mc-status (s) = 'running)
  ∧ evenp (mc-pc (s))
  ∧ rom-addrp (mc-pc (s), mc-mem (s), 70)
  ∧ mcode-addrp (mc-pc (s), mc-mem (s), STRNCMP-CODE)
  ∧ ram-addrp (sub (32, 12, read-sp (s)), mc-mem (s), 28)
  ∧ ram-addrp (*str1*, mc-mem (s), *n1*)
  ∧ mem-lst (1, *str1*, mc-mem (s), *n1*, *lst1*)
  ∧ ram-addrp (*str2*, mc-mem (s), *n2*)
  ∧ mem-lst (1, *str2*, mc-mem (s), *n2*, *lst2*)
  ∧ (slen (0, n, *lst1*) < *n1*)
  ∧ (slen (0, n, *lst2*) < *n2*)
  ∧ disjoint (sub (32, 12, read-sp (s)), 28, *str1*, *n1*)
  ∧ disjoint (sub (32, 12, read-sp (s)), 28, *str2*, *n2*)
  ∧ (*str1* = read-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
  ∧ (*str2* = read-mem (add (32, read-sp (s), 8), mc-mem (s), 4))
  ∧ (n = uread-mem (add (32, read-sp (s), 12), mc-mem (s), 4))
  ∧ (*n1* ∈ **N**)
  ∧ (*n2* ∈ **N**))

```
; an intermediate state.
```

DEFINITION:

strncmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, n, $n_$)

$=$ ((mc-status (s) $=$ `'running`)

$\wedge$ evenp (mc-pc (s))

$\wedge$ rom-addrp (sub (32, 34, mc-pc (s)), mc-mem (s), 70)

$\wedge$ mcode-addrp (sub (32, 34, mc-pc (s)), mc-mem (s), STRNCMP-CODE)

$\wedge$ ram-addrp (sub (32, 8, read-an (32, 6, s)), mc-mem (s), 28)

$\wedge$ ram-addrp (*str1*, mc-mem (s), *n1*)

$\wedge$ mem-lst (1, *str1*, mc-mem (s), *n1*, *lst1*)

$\wedge$ ram-addrp (*str2*, mc-mem (s), *n2*)

$\wedge$ mem-lst (1, *str2*, mc-mem (s), *n2*, *lst2*)

$\wedge$ (slen (i, $n_$, *lst1*) $<$ *n1*)

$\wedge$ (slen (i, $n_$, *lst2*) $<$ *n2*)

$\wedge$ disjoint (sub (32, 8, read-an (32, 6, s)), 28, *str1*, *n1*)

$\wedge$ disjoint (sub (32, 8, read-an (32, 6, s)), 28, *str2*, *n2*)

$\wedge$ equal* (read-an (32, 0, s), add (32, *str1*, i^*))

$\wedge$ equal* (read-an (32, 1, s), add (32, *str2*, i^*))

$\wedge$ nat-rangep (read-rn (32, 1, mc-rfile (s)), 8)

$\wedge$ nat-rangep (read-rn (32, 2, mc-rfile (s)), 8)

$\wedge$ (n $=$ nat-to-uint (read-dn (32, 0, s)))

$\wedge$ ($i^* \in \mathbf{N}$)

$\wedge$ nat-rangep (i^*, 32)

$\wedge$ (i $=$ nat-to-uint (i^*))

$\wedge$ ($n_ \in \mathbf{N}$)

$\wedge$ (($i + n$) $\leq$ $n_$)

$\wedge$ ($n \neq 0$)

$\wedge$ uint-rangep ($n_$, 32))

```
; from the initial state s to exit: s --> sn, when n = 0.
```

THEOREM: strncmp-s-sn

(strncmp-statep (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, n) $\wedge$ ($n \simeq 0$))

$\rightarrow$ ((mc-status (stepn (s, 10)) $=$ `'running`)

$\wedge$ (mc-pc (stepn (s, 10)) $=$ rts-addr (s))

$\wedge$ (iread-dn (32, 0, stepn (s, 10)) $=$ 0)

$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, 10)))

$=$ add (32, read-an (32, 7, s), 4))

$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 10))) $=$ read-an (32, 6, s)))

THEOREM: strncmp-s-sn-rfile

(strncmp-statep (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, n)

$\wedge$ ($n \simeq 0$)

$\wedge$ (*oplen* $\leq$ 32)

$\wedge$ d2-7a2-5p (rn))

$\rightarrow$ (read-rn (*oplen*, rn, mc-rfile (stepn (s, 10)))

$= \quad$ read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: strncmp-s-sn-mem
(strncmp-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, n)
$\wedge$ ($n \simeq$ 0)
$\wedge$ disjoint (x, k, sub (32, 12, read-sp (s)), 28))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 10)), k) = read-mem (x, mc-mem (s), k))

```
; from the initial state s to s0: s --> s0.
```

THEOREM: strncmp-s-s0
(strncmp-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, n) $\wedge$ ($n \not\simeq$ 0))
$\rightarrow$ strncmp-s0p (stepn (s, 8), 0, 0, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, n, n)

THEOREM: strncmp-s-s0-else
(strncmp-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, n) $\wedge$ ($n \not\simeq$ 0))
$\rightarrow$ ((linked-rts-addr (stepn (s, 8)) = rts-addr (s))
$\wedge$ (linked-a6 (stepn (s, 8)) = read-an (32, 6, s))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 8)))
$=$ sub (32, 4, read-sp (s)))
$\wedge$ (movem-saved (stepn (s, 8), 4, 8, 2)
$=$ readm-rn (32, '(2 3), mc-rfile (s))))

THEOREM: strncmp-s-s0-rfile
(strncmp-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, n) $\wedge$ ($n \not\simeq$ 0) $\wedge$ d4-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 8)))
$=$ read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: strncmp-s-s0-mem
(strncmp-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, n)
$\wedge$ ($n \not\simeq$ 0)
$\wedge$ disjoint (x, k, sub (32, 12, read-sp (s)), 28))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 8)), k) = read-mem (x, mc-mem (s), k))

```
; from s0 to exit: s0 --> sn.
; base case 1:  s0 --> sn, when lst1[i] =\= lst2[i].
```

THEOREM: strncmp-s0-sn-base1
(strncmp-s0p (s, i*, i, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, n, $n_$)
$\wedge$ (get-nth (i, $lst1$) $\neq$ get-nth (i, $lst2$)))
$\rightarrow$ ((mc-status (stepn (s, 11)) = 'running)
$\wedge$ (mc-pc (stepn (s, 11)) = linked-rts-addr (s))
$\wedge$ (iread-dn (32, 0, stepn (s, 11))
$=$ idifference (get-nth (i, $lst1$), get-nth (i, $lst2$)))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 11))) = linked-a6 (s))

$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, 11)))
= add (32, read-an (32, 6, s), 8))
$\wedge$ (read-mem (x, mc-mem (stepn (s, 11)), k)
= read-mem (x, mc-mem (s), k)))

THEOREM: strncmp-s0-sn-rfile-base1
(strncmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, n, $n_$)
$\wedge$ (get-nth (i, *lst1*) $\neq$ get-nth (i, *lst2*))
$\wedge$ (*oplen* $\leq$ 32)
$\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 11)))
= **if** d4-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (s))
else get-vlst (*oplen*, 0, *rn*, '(2 3), movem-saved (s, 4, 8, 2)) **endif**)

```
; base case 2: s0 --> sn, when lst[i] = lst2[i] and lst[i] = 0.
```

THEOREM: strncmp-s0-sn-base2
(strncmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, n, $n_$)
$\wedge$ (get-nth (i, *lst1*) = get-nth (i, *lst2*))
$\wedge$ (get-nth (i, *lst1*) = 0))
$\rightarrow$ ((mc-status (stepn (s, 9)) = 'running)
$\wedge$ (mc-pc (stepn (s, 9)) = linked-rts-addr (s))
$\wedge$ (iread-dn (32, 0, stepn (s, 9)) = 0)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 9))) = linked-a6 (s))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, 9)))
= add (32, read-an (32, 6, s), 8))
$\wedge$ (read-mem (x, mc-mem (stepn (s, 9)), k)
= read-mem (x, mc-mem (s), k)))

THEOREM: strncmp-s0-sn-rfile-base2
(strncmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, n, $n_$)
$\wedge$ (get-nth (i, *lst1*) = get-nth (i, *lst2*))
$\wedge$ (get-nth (i, *lst1*) = 0)
$\wedge$ (*oplen* $\leq$ 32)
$\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 9)))
= **if** d4-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (s))
else get-vlst (*oplen*, 0, *rn*, '(2 3), movem-saved (s, 4, 8, 2)) **endif**)

```
; base case 3: s0 --> sn, when lst[i] = lst2[i], lst[i] =\= 0, and n-1 = 0.
```

THEOREM: strncmp-s0-sn-base3
(strncmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, n, $n_$)
$\wedge$ (get-nth (i, *lst1*) = get-nth (i, *lst2*))
$\wedge$ (get-nth (i, *lst1*) $\neq$ 0)

```
 ∧   ((n − 1) = 0))
 →   ((mc-status (stepn (s, 11)) = 'running)
      ∧   (mc-pc (stepn (s, 11)) = linked-rts-addr (s))
      ∧   (iread-dn (32, 0, stepn (s, 11)) = 0)
      ∧   (read-rn (32, 14, mc-rfile (stepn (s, 11))) = linked-a6 (s))
      ∧   (read-rn (32, 15, mc-rfile (stepn (s, 11)))
          =   add (32, read-an (32, 6, s), 8))
      ∧   (read-mem (x, mc-mem (stepn (s, 11)), k)
          =   read-mem (x, mc-mem (s), k)))
```

THEOREM: strncmp-s0-sn-rfile-base3

```
 (strncmp-s0p (s, i*, i, str1, n1, lst1, str2, n2, lst2, n, n_)
 ∧   (get-nth (i, lst1) = get-nth (i, lst2))
 ∧   (get-nth (i, lst1) ≠ 0)
 ∧   ((n − 1) = 0)
 ∧   (oplen ≤ 32)
 ∧   d2-7a2-5p (rn))
 →   (read-rn (oplen, rn, mc-rfile (stepn (s, 11)))
      =   if d4-7a2-5p (rn)  then read-rn (oplen, rn, mc-rfile (s))
          else get-vlst (oplen, 0, rn, '(2 3), movem-saved (s, 4, 8, 2)) endif)
```

```
; induction case: s0 --> s0, lst[i] = lst2[i], lst[i] =\= 0 and n-1 =\= 0.
```

THEOREM: strncmp-s0-s0

```
 (strncmp-s0p (s, i*, i, str1, n1, lst1, str2, n2, lst2, n, n_)
 ∧   (get-nth (i, lst1) = get-nth (i, lst2))
 ∧   (get-nth (i, lst1) ≠ 0)
 ∧   ((n − 1) ≠ 0))
 →   (strncmp-s0p (stepn (s, 7),
                   add (32, i*, 1),
                   1 + i,
                   str1,
                   n1,
                   lst1,
                   str2,
                   n2,
                   lst2,
                   n − 1,
                   n_)
      ∧   (read-rn (32, 14, mc-rfile (stepn (s, 7)))
          =   read-rn (32, 14, mc-rfile (s)))
      ∧   (linked-a6 (stepn (s, 7)) = linked-a6 (s))
      ∧   (linked-rts-addr (stepn (s, 7)) = linked-rts-addr (s))
      ∧   (movem-saved (stepn (s, 7), 4, 8, 2) = movem-saved (s, 4, 8, 2))
      ∧   (read-mem (x, mc-mem (stepn (s, 7)), k)
```

```
        =   read-mem (x, mc-mem (s), k)))

THEOREM: strncmp-s0-s0-rfile
 (strncmp-s0p (s, i*, i, str1, n1, lst1, str2, n2, lst2, n, n_)
  ∧   (get-nth (i, lst1) = get-nth (i, lst2))
  ∧   (get-nth (i, lst1) ≠ 0)
  ∧   ((n − 1) ≠ 0)
  ∧   d4-7a2-5p (rn))
 →   (read-rn (oplen, rn, mc-rfile (stepn (s, 7)))
       =   read-rn (oplen, rn, mc-rfile (s)))

; put together. s0 --> exit.

THEOREM: strncmp-s0-sn
 let sn  be  stepn (s, strncmp1-t (i, n, lst1, lst2))
 in
 strncmp-s0p (s, i*, i, str1, n1, lst1, str2, n2, lst2, n, n_)
 →   ((mc-status (sn) = 'running)
       ∧   (mc-pc (sn) = linked-rts-addr (s))
       ∧   (iread-dn (32, 0, sn) = strncmp1 (i, n, lst1, lst2))
       ∧   (read-rn (32, 14, mc-rfile (sn)) = linked-a6 (s))
       ∧   (read-rn (32, 15, mc-rfile (sn))
            =   add (32, read-an (32, 6, s), 8))
       ∧   (read-mem (x, mc-mem (sn), k) = read-mem (x, mc-mem (s), k))) endlet

THEOREM: strncmp-s0-sn-rfile
 let sn  be  stepn (s, strncmp1-t (i, n, lst1, lst2))
 in
 (strncmp-s0p (s, i*, i, str1, n1, lst1, str2, n2, lst2, n, n_)
  ∧   d2-7a2-5p (rn)
  ∧   (oplen ≤ 32))
 →   (read-rn (oplen, rn, mc-rfile (sn))
       =   if d4-7a2-5p (rn)  then read-rn (oplen, rn, mc-rfile (s))
           else get-vlst (oplen,
                          0,
                          rn,
                          '(2 3),
                          movem-saved (s, 4, 8, 2)) endif) endlet

; the correctness of strncmp.

THEOREM: strncmp-correctness
 let sn  be  stepn (s, strncmp-t (n, lst1, lst2))
 in
 strncmp-statep (s, str1, n1, lst1, str2, n2, lst2, n)
```

$\rightarrow$ ((mc-status (sn) = `'running`)
$\wedge$ (mc-pc (sn) = rts-addr (s))
$\wedge$ (read-rn (32, 14, mc-rfile (sn))
= read-rn (32, 14, mc-rfile (s)))
$\wedge$ (read-rn (32, 15, mc-rfile (sn))
= add (32, read-an (32, 7, s), 4))
$\wedge$ ((d2-7a2-5p (rn) $\wedge$ ($oplen \leq$ 32))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (sn))
= read-rn ($oplen$, rn, mc-rfile (s))))
$\wedge$ (disjoint (x, k, sub (32, 12, read-sp (s)), 28)
$\rightarrow$ (read-mem (x, mc-mem (sn), k)
= read-mem (x, mc-mem (s), k)))
$\wedge$ (iread-dn (32, 0, sn) = strncmp (n, *lst1*, *lst2*))) **endlet**

EVENT: Disable strncmp-t.

```
; some properties of strncmp.
; see file cstring.events.
```

EVENT: Make the library "`strncmp`" and compile it.

Index

add, 4, 5, 7–10

d2-7a2-5p, 5, 7–10
d4-7a2-5p, 6–9
disjoint, 4–6, 10

equal*, 5
evenp, 3–5

get-nth, 3, 4, 6–9
get-vlst, 7–9

idifference, 6
iread-dn, 5–10

linked-a6, 6–9
linked-rts-addr, 6–9

mc-mem, 3–10
mc-pc, 4–10
mc-rfile, 5–10
mc-status, 4–10
mcode-addrp, 3–5
mem-lst, 4, 5
movem-saved, 6–9

nat-rangep, 3, 5
nat-to-uint, 5

ram-addrp, 4, 5
read-an, 5–10
read-dn, 5
read-mem, 4, 6–10
read-rn, 5–10
read-sp, 4, 6, 10
readm-rn, 6
rom-addrp, 3–5
rts-addr, 5, 6, 10

slen, 4, 5
splus, 3, 4
stepn, 3–9
stepn-strncmp-loadp, 3
strncmp, 10
strncmp-addr, 3
strncmp-code, 3–5
strncmp-correctness, 9
strncmp-induct, 4
strncmp-load, 3
strncmp-loadp, 3
strncmp-s-s0, 6
strncmp-s-s0-else, 6
strncmp-s-s0-mem, 6
strncmp-s-s0-rfile, 6
strncmp-s-sn, 5
strncmp-s-sn-mem, 6
strncmp-s-sn-rfile, 5
strncmp-s0-s0, 8
strncmp-s0-s0-rfile, 9
strncmp-s0-sn, 9
strncmp-s0-sn-base1, 6
strncmp-s0-sn-base2, 7
strncmp-s0-sn-base3, 7
strncmp-s0-sn-rfile, 9
strncmp-s0-sn-rfile-base1, 7
strncmp-s0-sn-rfile-base2, 7
strncmp-s0-sn-rfile-base3, 8
strncmp-s0p, 5–9
strncmp-statep, 4–6, 9
strncmp-t, 4, 9
strncmp1, 9
strncmp1-t, 3, 4, 9
sub, 4–6, 10

uint-rangep, 5
uread-mem, 4