```
#|

 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 "mc20-2" using the compiled version.

```
;          Proof of the Correctness of the MEMCHR 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 memchr function in the Berkeley string library.

void *
memchr(s, c, n)
const void *s;
register unsigned char c;
register size_t n;
{
if (n != 0) {
register const unsigned char *p = s;
```

```
do {
if (*p++ == c)
return ((void *)(p - 1));
} while (--n != 0);
}
return (NULL);
}

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

0x27e8 <memchr>:        linkw fp,#0
0x27ec <memchr+4>:      moveb fp@(15),d1
0x27f0 <memchr+8>:      movel fp@(16),d0
0x27f4 <memchr+12>:     beq 0x2808 <memchr+32>
0x27f6 <memchr+14>:     moveal fp@(8),a0
0x27fa <memchr+18>:     cmpb a0@+,d1
0x27fc <memchr+20>:     bne 0x2804 <memchr+28>
0x27fe <memchr+22>:     movel a0,d0
0x2800 <memchr+24>:     subl #1,d0
0x2802 <memchr+26>:     bra 0x280a <memchr+34>
0x2804 <memchr+28>:     subl #1,d0
0x2806 <memchr+30>:     bne 0x27fa <memchr+18>
0x2808 <memchr+32>:     clrl d0
0x280a <memchr+34>:     unlk fp
0x280c <memchr+36>:     rts

The machine code of the above program is:

<memchr>:        0x4e56  0x0000  0x122e  0x000f  0x202e  0x0010  0x6712  0x206e
<memchr+16>:     0x0008  0xb218  0x6606  0x2008  0x5380  0x6006  0x5380  0x66f2
<memchr+32>:     0x4280  0x4e5e  0x4e75

'(78     86      0       0       18      46      0       15
  32     46      0       16      103     18      32      110
  0      8       178     24      102     6       32      8
  83     128     96      6       83      128     102     242
  66     128     78      94      78      117)
|#

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

DEFINITION:
MEMCHR-CODE

$=$ `'(78 86 0 0 18 46 0 15 32 46 0 16 103 18 32 110 0 8`
`178 24 102 6 32 8 83 128 96 6 83 128 102 242 66 128`
`78 94 78 117)`

`; the computation time of the program.`

DEFINITION:
memchr-t1 (i, n, lst, ch)
$=$ **if** get-nth (i, lst) $=$ ch **then** 7
elseif $(n - 1) = 0$ **then** 7
else splus (4, memchr-t1 ($1 + i$, $n - 1$, lst, ch)) **endif**

DEFINITION:
memchr-t (n, lst, ch)
$=$ **if** $n = 0$ **then** 7
else splus (5, memchr-t1 (0, n, lst, ch)) **endif**

`; an induction hint.`

DEFINITION:
memchr-induct (s, $i*$, i, n, lst, ch)
$=$ **if** get-nth (i, lst) $=$ ch **then t**
elseif $(n - 1) = 0$ **then t**
else memchr-induct (stepn (s, 4), add (32, $i*$, 1), $1 + i$, $n - 1$, lst, ch) **endif**

`; the preconditions of the initial state.`

DEFINITION:
memchr-statep (s, str, n, lst, ch)
$=$ ((mc-status (s) $=$ `'running`)
$\land$ evenp (mc-pc (s))
$\land$ rom-addrp (mc-pc (s), mc-mem (s), 38)
$\land$ mcode-addrp (mc-pc (s), mc-mem (s), MEMCHR-CODE)
$\land$ ram-addrp (sub (32, 4, read-sp (s)), mc-mem (s), 20)
$\land$ ram-addrp (str, mc-mem (s), n)
$\land$ mem-lst (1, str, mc-mem (s), n, lst)
$\land$ disjoint (sub (32, 4, read-sp (s)), 20, str, n)
$\land$ (str $=$ read-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
$\land$ (ch $=$ uread-mem (add (32, read-sp (s), 11), mc-mem (s), 1))
$\land$ (n $=$ uread-mem (add (32, read-sp (s), 12), mc-mem (s), 4))
$\land$ (nat-to-uint (str) $\neq$ 0)
$\land$ uint-rangep (nat-to-uint (str) $+$ n, 32))

`; an intermediate state.`

DEFINITION:
memchr-s0p (s, $i*$, i, str, n, lst, ch, $n\_$)
= ((mc-status (s) = `'running`)
∧ evenp (mc-pc (s))
∧ rom-addrp (sub (32, 18, mc-pc (s)), mc-mem (s), 38)
∧ mcode-addrp (sub (32, 18, mc-pc (s)), mc-mem (s), MEMCHR-CODE)
∧ ram-addrp (read-an (32, 6, s), mc-mem (s), 20)
∧ ram-addrp (str, mc-mem (s), $n\_$)
∧ mem-lst (1, str, mc-mem (s), $n\_$, lst)
∧ disjoint (read-an (32, 6, s), 20, str, $n\_$)
∧ equal* (read-an (32, 0, s), add (32, str, $i*$))
∧ (ch = nat-to-uint (read-dn (8, 1, s)))
∧ (n = nat-to-uint (read-dn (32, 0, s)))
∧ (i = nat-to-uint ($i*$))
∧ (($i + n$) $\leq$ $n\_$)
∧ ($n \neq 0$)
∧ ($i* \in \mathbf{N}$)
∧ ($n\_ \in \mathbf{N}$)
∧ nat-rangep ($i*$, 32)
∧ uint-rangep ($n\_$, 32))

```
; from the intial state s to exit: s --> sn.
```

THEOREM: memchr-s-sn
(memchr-statep (s, str, n, lst, ch) ∧ ($n = 0$))
→ ((mc-status (stepn (s, 7)) = `'running`)
∧ (mc-pc (stepn (s, 7)) = rts-addr (s))
∧ (read-dn (32, 0, stepn (s, 7)) = 0)
∧ (read-rn (32, 15, mc-rfile (stepn (s, 7)))
= add (32, read-an (32, 7, s), 4))
∧ (read-rn (32, 14, mc-rfile (stepn (s, 7))) = read-an (32, 6, s)))

THEOREM: memchr-s-sn-rfile
(memchr-statep (s, str, n, lst, ch) ∧ ($n = 0$) ∧ d2-7a2-5p (rn))
→ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 7)))
= read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: memchr-s-sn-mem
(memchr-statep (s, str, n, lst, ch)
∧ ($n = 0$)
∧ disjoint (x, k, sub (32, 4, read-sp (s)), 20))
→ (read-mem (x, mc-mem (stepn (s, 7)), k) = read-mem (x, mc-mem (s), k))

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

THEOREM: memchr-s-s0
(memchr-statep $(s,\ str,\ n,\ lst,\ ch) \wedge (n \neq \mathtt{0})$)
$\rightarrow$ memchr-s0p (stepn $(s,\ \mathtt{5}),\ \mathtt{0},\ \mathtt{0},\ str,\ n,\ lst,\ ch,\ n$)

THEOREM: memchr-s-s0-else
(memchr-statep $(s,\ str,\ n,\ lst,\ ch) \wedge (n \neq \mathtt{0})$)
$\rightarrow$ ((linked-rts-addr (stepn $(s,\ \mathtt{5})$) = rts-addr (s))
  $\wedge$ (linked-a6 (stepn $(s,\ \mathtt{5})$) = read-an $(\mathtt{32},\ \mathtt{6},\ s)$)
  $\wedge$ (read-rn ($\mathtt{32},\ \mathtt{14}$, mc-rfile (stepn $(s,\ \mathtt{5})$))
    = sub ($\mathtt{32},\ \mathtt{4}$, read-sp (s)))))

THEOREM: memchr-s-s0-rfile
(memchr-statep $(s,\ str,\ n,\ lst,\ ch) \wedge (n \neq \mathtt{0}) \wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen,\ rn$, mc-rfile (stepn $(s,\ \mathtt{5})$)))
  = read-rn ($oplen,\ rn$, mc-rfile (s))))

THEOREM: memchr-s-s0-mem
(memchr-statep $(s,\ str,\ n,\ lst,\ ch)$
$\wedge$ $(n \neq \mathtt{0})$
$\wedge$ disjoint ($x,\ k$, sub ($\mathtt{32},\ \mathtt{4}$, read-sp (s)), $\mathtt{20}$))
$\rightarrow$ (read-mem (x, mc-mem (stepn $(s,\ \mathtt{5})$), k) = read-mem (x, mc-mem (s), k))

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

THEOREM: memchr-s0-sn-base1
(memchr-s0p $(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ n\_) \wedge$ (get-nth $(i,\ lst) = ch$))
$\rightarrow$ ((mc-status (stepn $(s,\ \mathtt{7})$) = `'running`)
  $\wedge$ (mc-pc (stepn $(s,\ \mathtt{7})$) = linked-rts-addr (s))
  $\wedge$ (read-dn ($\mathtt{32},\ \mathtt{0}$, stepn $(s,\ \mathtt{7})$) = add $(\mathtt{32},\ str,\ i^*)$)
  $\wedge$ (read-rn ($\mathtt{32},\ \mathtt{14}$, mc-rfile (stepn $(s,\ \mathtt{7})$)) = linked-a6 (s))
  $\wedge$ (read-rn ($\mathtt{32},\ \mathtt{15}$, mc-rfile (stepn $(s,\ \mathtt{7})$))
    = add ($\mathtt{32}$, read-an $(\mathtt{32},\ \mathtt{6},\ s)$, $\mathtt{8}$))
  $\wedge$ (read-mem (x, mc-mem (stepn $(s,\ \mathtt{7})$), k)
    = read-mem (x, mc-mem (s), k)))

THEOREM: memchr-s0-sn-rfile-base1
(memchr-s0p $(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ n\_)$
$\wedge$ (get-nth $(i,\ lst) = ch$)
$\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen,\ rn$, mc-rfile (stepn $(s,\ \mathtt{7})$)))
  = read-rn ($oplen,\ rn$, mc-rfile (s))))

```
; base case 2: s0 --> sn, when lst[i] =\= ch, n-1 = 0.
```

THEOREM: memchr-s0-sn-base2

$$
\begin{array}{l}
(\text{memchr-s0p}\,(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ n\_) \\
\ \wedge\quad (\text{get-nth}\,(i,\ lst) \neq ch) \\
\ \wedge\quad ((n - 1) = \mathtt{0})) \\
\rightarrow\quad ((\text{mc-status}\,(\text{stepn}\,(s,\ \mathtt{7})) = \mathtt{'running}) \\
\qquad\wedge\quad (\text{mc-pc}\,(\text{stepn}\,(s,\ \mathtt{7})) = \text{linked-rts-addr}\,(s)) \\
\qquad\wedge\quad (\text{read-dn}\,(\mathtt{32},\ \mathtt{0},\ \text{stepn}\,(s,\ \mathtt{7})) = \mathtt{0}) \\
\qquad\wedge\quad (\text{read-rn}\,(\mathtt{32},\ \mathtt{14},\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \mathtt{7}))) = \text{linked-a6}\,(s)) \\
\qquad\wedge\quad (\text{read-rn}\,(\mathtt{32},\ \mathtt{15},\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \mathtt{7}))) \\
\qquad\qquad = \quad \text{add}\,(\mathtt{32},\ \text{read-an}\,(\mathtt{32},\ \mathtt{6},\ s),\ \mathtt{8})) \\
\qquad\wedge\quad (\text{read-mem}\,(x,\ \text{mc-mem}\,(\text{stepn}\,(s,\ \mathtt{7})),\ k) \\
\qquad\qquad = \quad \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k)))
\end{array}
$$

THEOREM: memchr-s0-sn-rfile-base2

$$
\begin{array}{l}
(\text{memchr-s0p}\,(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ n\_) \\
\ \wedge\quad (\text{get-nth}\,(i,\ lst) \neq ch) \\
\ \wedge\quad ((n - 1) = \mathtt{0}) \\
\ \wedge\quad \text{d2-7a2-5p}\,(rn)) \\
\rightarrow\quad (\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \mathtt{7}))) \\
\qquad = \quad \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s)))
\end{array}
$$

```
; induction case: s0 --> s0.
```

THEOREM: memchr-s0-s0

$$
\begin{array}{l}
(\text{memchr-s0p}\,(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ n\_) \\
\ \wedge\quad (\text{get-nth}\,(i,\ lst) \neq ch) \\
\ \wedge\quad ((n - 1) \neq \mathtt{0})) \\
\rightarrow\quad (\text{memchr-s0p}\,(\text{stepn}\,(s,\ \mathtt{4}),\ \text{add}\,(\mathtt{32},\ i^*,\ \mathtt{1}),\ 1 + i,\ str,\ n - 1,\ lst,\ ch,\ n\_) \\
\qquad\wedge\quad (\text{read-rn}\,(\mathtt{32},\ \mathtt{14},\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \mathtt{4}))) \\
\qquad\qquad = \quad \text{read-rn}\,(\mathtt{32},\ \mathtt{14},\ \text{mc-rfile}\,(s))) \\
\qquad\wedge\quad (\text{linked-a6}\,(\text{stepn}\,(s,\ \mathtt{4})) = \text{linked-a6}\,(s)) \\
\qquad\wedge\quad (\text{linked-rts-addr}\,(\text{stepn}\,(s,\ \mathtt{4})) = \text{linked-rts-addr}\,(s)) \\
\qquad\wedge\quad (\text{read-mem}\,(x,\ \text{mc-mem}\,(\text{stepn}\,(s,\ \mathtt{4})),\ k) \\
\qquad\qquad = \quad \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k)))
\end{array}
$$

THEOREM: memchr-s0-s0-rfile

$$
\begin{array}{l}
(\text{memchr-s0p}\,(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ n\_) \\
\ \wedge\quad (\text{get-nth}\,(i,\ lst) \neq ch) \\
\ \wedge\quad ((n - 1) \neq \mathtt{0}) \\
\ \wedge\quad \text{d2-7a2-5p}\,(rn)) \\
\rightarrow\quad (\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \mathtt{4}))) \\
\qquad = \quad \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s)))
\end{array}
$$

```
; put together (s0 --> exit).
```

THEOREM: memchr-s0-sn
let *sn* **be** stepn (*s*, memchr-t1 (*i*, *n*, *lst*, *ch*))
in
memchr-s0p (*s*, *i**, *i*, *str*, *n*, *lst*, *ch*, n_)
→ ((mc-status (*sn*) = `'running`)
∧ (mc-pc (*sn*) = linked-rts-addr (*s*))
∧ (read-dn (`32`, `0`, *sn*)
= **if** memchr1 (*i*, *n*, *lst*, *ch*)
then add (`32`, *str*, memchr* (*i**, *i*, *n*, *lst*, *ch*))
else `0` **endif**)
∧ (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: memchr-s0-sn-rfile
(memchr-s0p (*s*, *i**, *i*, *str*, *n*, *lst*, *ch*, n_) ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, memchr-t1 (*i*, *n*, *lst*, *ch*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; the correctness of the MEMCHR program.
```

THEOREM: memchr-correctness
let *sn* **be** stepn (*s*, memchr-t (*n*, *lst*, *ch*))
in
memchr-statep (*s*, *str*, *n*, *lst*, *ch*)
→ ((mc-status (*sn*) = `'running`)
∧ (mc-pc (*sn*) = rts-addr (*s*))
∧ (read-rn (`32`, `14`, mc-rfile (*sn*))
= read-rn (`32`, `14`, mc-rfile (*s*)))
∧ (read-rn (`32`, `15`, mc-rfile (*sn*))
= add (`32`, read-sp (*s*), `4`))
∧ (d2-7a2-5p (*rn*)
→ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))))
∧ (disjoint (*x*, *k*, sub (`32`, `4`, read-sp (*s*)), `20`)
→ (read-mem (*x*, mc-mem (*sn*), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))
∧ (read-dn (`32`, `0`, *sn*)
= **if** memchr (*n*, *lst*, *ch*)
then add (`32`, *str*, memchr* (`0`, `0`, *n*, *lst*, *ch*))
else `0` **endif**)) **endlet**

EVENT: Disable memchr-t.

```
; memchr* --> memchr.
```

THEOREM: memchr*-memchr1
(memchr1 (i, n, lst, ch)
 $\wedge$ (i = nat-to-uint ($i*$))
 $\wedge$ nat-rangep ($i*$, 32)
 $\wedge$ uint-rangep ($i + n$, 32))
$\rightarrow$ (nat-to-uint (memchr* ($i*$, i, n, lst, ch)) = memchr1 (i, n, lst, ch))

THEOREM: memchr-non-zerop-la
let sn **be** stepn (s, memchr-t (n, lst, ch))
in
(memchr-statep (s, str, n, lst, ch)
 $\wedge$ ($n \in \mathbf{N}$)
 $\wedge$ nat-rangep (str, 32)
 $\wedge$ (nat-to-uint (str) $\neq$ 0)
 $\wedge$ uint-rangep (nat-to-uint (str) + n, 32)
 $\wedge$ memchr (n, lst, ch))
$\rightarrow$ (nat-to-uint (read-dn (32, 0, sn)) $\neq$ 0) **endlet**

THEOREM: memchr-non-zerop
let sn **be** stepn (s, memchr-t (n, lst, ch))
in
(memchr-statep (s, str, n, lst, ch) $\wedge$ memchr (n, lst, ch))
$\rightarrow$ (nat-to-uint (read-dn (32, 0, sn)) $\neq$ 0) **endlet**

EVENT: Disable memchr*.

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

Index