```
#|

 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.

|#

;          Proof of the correctness of the AMAX program
;
```

EVENT: Start with the library "mc20-2" using the compiled version.

```
#|

This program finds the maximum value of an integer array.  The way I
wrote this program is to test the ability to handle the while statement.

/* find the maximum in an integer array  */
amax(a, n)
int a[], n;
{
  int max;

  max = a[--n];
  while (n > 0)
    if (a[--n] > max)
```

```
      max = a[n];
  return (max);
}
```

Here is the MC68020 assembly code of the above AMAX program. The code is generated by "gcc -O".

```
0x2342 <amax>:          linkw a6,#0
0x2346 <amax+4>:        moveal a6@(8),a0
0x234a <amax+8>:        movel a6@(12),d0
0x234e <amax+12>:       subl #1,d0
0x2350 <amax+14>:       bra 0x235a <amax+24>
0x2352 <amax+16>:       subl #1,d0
0x2354 <amax+18>:       cmpl 0(a0)[d0.l*4],d1
0x2358 <amax+22>:       bge 0x235e <amax+28>
0x235a <amax+24>:       movel 0(a0)[d0.l*4],d1
0x235e <amax+28>:       tstl d0
0x2360 <amax+30>:       bgt 0x2352 <amax+16>
0x2362 <amax+32>:       movel d1,d0
0x2364 <amax+34>:       unlk a6
0x2366 <amax+36>:       rts
```

The machine code of the above program is:

```
<amax>:       0x4e56  0x0000  0x206e  0x0008  0x202e  0x000c  0x5380  0x6008
<amax+16>:    0x5380  0xb2b0  0x0c00  0x6c04  0x2230  0x0c00  0x4a80  0x6ef0
<amax+32>:    0x2001  0x4e5e  0x4e75

'(78      86      0       0       32      110     0       8
  32      46      0       12      83      128     96      8
  83      128     178     176     12      0       108     4
  34      48      12      0       74      128     110     240
  32      1       78      94      78      117)
|#

; now we start to prove the correctness of this AMAX program, defined by
; (amax-code).
```

DEFINITION:
AMAX-CODE
= '(78 86 0 0 32 110 0 8 32 46 0 12 83 128 96 8 83 128
178 176 12 0 108 4 34 48 12 0 74 128 110 240 32 1 78
94 78 117)

```
; the maximum value in a list.
```

DEFINITION:
amax1 ($imax$, n, lst)
= **if** $n \simeq$ `0` **then** $imax$
elseif ilessp ($imax$, get-nth ($n - 1$, lst))
then amax1 (get-nth ($n - 1$, lst), $n - 1$, lst)
else amax1 ($imax$, $n - 1$, lst) **endif**

DEFINITION: amax (n, lst) = amax1 (get-nth ($n - 1$, lst), $n - 1$, lst)

```
; the clock of AMAX program.
```

DEFINITION:
amax1-t ($imax$, i, lst)
= **if** $i \simeq$ `0` **then** `5`
elseif ilessp ($imax$, get-nth ($i - 1$, lst))
then splus (`6`, amax1-t (get-nth ($i - 1$, lst), $i - 1$, lst))
else splus (`5`, amax1-t ($imax$, $i - 1$, lst)) **endif**

DEFINITION:
amax-t (n, lst) = splus (`6`, amax1-t (get-nth ($n - 1$, lst), $n - 1$, lst))

```
; the induction hint for the loop.
```

DEFINITION:
amax-induct ($imax$, n, lst, s)
= **if** $n \simeq$ `0` **then t**
elseif ilessp ($imax$, get-nth ($n - 1$, lst))
then amax-induct (get-nth ($n - 1$, lst), $n - 1$, lst, stepn (s, `6`))
else amax-induct ($imax$, $n - 1$, lst, stepn (s, `5`)) **endif**

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

DEFINITION:
amax-statep (s, a, n, lst)
= ((mc-status (s) = `'running`)
∧ evenp (mc-pc (s))
∧ rom-addrp (mc-pc (s), mc-mem (s), `38`)
∧ mcode-addrp (mc-pc (s), mc-mem (s), AMAX-CODE)
∧ ram-addrp (sub (`32`, `4`, read-sp (s)), mc-mem (s), `16`)
∧ mem-ilst (`4`, a, mc-mem (s), n, lst)
∧ ram-addrp (a, mc-mem (s), `4` $*$ n)
∧ disjoint (a, `4` $*$ n, sub (`32`, `4`, read-sp (s)), `16`)
∧ (a = read-mem (add (`32`, read-sp (s), `4`), mc-mem (s), `4`))
∧ (n = iread-mem (add (`32`, read-sp (s), `8`), mc-mem (s), `4`))
∧ (`0` < n))

; the conditions of the intermediate state.

DEFINITION:
amax1-statep (s, *imax*, i, n, *lst*)
= ((mc-status (s) = 'running)
∧ evenp (mc-pc (s))
∧ rom-addrp (sub (32, 28, mc-pc (s)), mc-mem (s), 38)
∧ mcode-addrp (sub (32, 28, mc-pc (s)), mc-mem (s), AMAX-CODE)
∧ ram-addrp (read-an (32, 6, s), mc-mem (s), 16)
∧ mem-ilst (4, read-an (32, 0, s), mc-mem (s), n, *lst*)
∧ ram-addrp (read-an (32, 0, s), mc-mem (s), $4 * n$)
∧ disjoint (read-an (32, 0, s), $4 * n$, read-an (32, 6, s), 16)
∧ (i = nat-to-int (read-dn (32, 0, s), 32))
∧ (*imax* = nat-to-int (read-dn (32, 1, s), 32))
∧ ($0 < n$)
∧ ($i \in \mathbf{N}$)
∧ ($i < n$))

EVENT: Enable iplus.

EVENT: Disable ilessp.

; the initial segment of the program.

THEOREM: amax->amax1-statep
amax-statep (s, a, n, *lst*)
→ amax1-statep (stepn (s, 6), get-nth ($n - 1$, *lst*), $n - 1$, n, *lst*)

THEOREM: amax-initial
amax-statep (s, a, n, *lst*)
→ ((linked-rts-addr (stepn (s, 6)) = rts-addr (s))
∧ (linked-a6 (stepn (s, 6)) = read-an (32, 6, s))
∧ (read-rn (32, 14, mc-rfile (stepn (s, 6)))
= sub (32, 4, read-sp (s)))))

THEOREM: amax-initial-rfile
(amax-statep (s, a, n, *lst*) ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 6)))
= read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: amax-initial-mem
(amax-statep (s, a, n, *lst*) ∧ disjoint (x, k, sub (32, 4, read-sp (s)), 16))
→ (read-mem (x, mc-mem (stepn (s, 6)), k) = read-mem (x, mc-mem (s), k))

; base case.

THEOREM: amax-base-case
(amax1-statep (*s*, *imax*, *i*, *n*, *lst*) ∧ (*i* ≃ 0))
→ ((mc-status (stepn (*s*, 5)) = 'running)
∧ (mc-pc (stepn (*s*, 5)) = linked-rts-addr (*s*))
∧ (iread-dn (32, 0, stepn (*s*, 5)) = *imax*)
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 5))) = linked-a6 (*s*))
∧ (read-rn (32, 15, mc-rfile (stepn (*s*, 5)))
= add (32, read-an (32, 6, *s*), 8))
∧ (read-mem (*x*, mc-mem (stepn (*s*, 5)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: amax-base-rfile
(amax1-statep (*s*, *imax*, *i*, *n*, *lst*) ∧ d2-7a2-5p (*rn*) ∧ (*i* ≃ 0))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 5)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; induction case.
```

THEOREM: amax-induct-case1
(amax1-statep (*s*, *imax*, *i*, *n*, *lst*)
∧ (*i* ≠ 0)
∧ ilessp (*imax*, get-nth (*i* − 1, *lst*)))
→ (amax1-statep (stepn (*s*, 6), get-nth (*i* − 1, *lst*), *i* − 1, *n*, *lst*)
∧ (read-rn (*oplen*, 14, mc-rfile (stepn (*s*, 6)))
= read-rn (*oplen*, 14, mc-rfile (*s*)))
∧ (linked-a6 (stepn (*s*, 6)) = linked-a6 (*s*))
∧ (linked-rts-addr (stepn (*s*, 6)) = linked-rts-addr (*s*))
∧ (read-mem (*x*, mc-mem (stepn (*s*, 6)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: amax-induct-case2
(amax1-statep (*s*, *imax*, *i*, *n*, *lst*)
∧ (*i* ≠ 0)
∧ (¬ ilessp (*imax*, get-nth (*i* − 1, *lst*))))
→ (amax1-statep (stepn (*s*, 5), *imax*, *i* − 1, *n*, *lst*)
∧ (read-rn (*oplen*, 14, mc-rfile (stepn (*s*, 5)))
= read-rn (*oplen*, 14, mc-rfile (*s*)))
∧ (linked-a6 (stepn (*s*, 5)) = linked-a6 (*s*))
∧ (linked-rts-addr (stepn (*s*, 5)) = linked-rts-addr (*s*))
∧ (read-mem (*x*, mc-mem (stepn (*s*, 5)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: amax-induct-rfile1
(amax1-statep (*s*, *imax*, *i*, *n*, *lst*)
∧ (*i* ≠ 0)

$\wedge$ ilessp (*imax*, get-nth (*i* − 1, *lst*))
$\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 6)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: amax-induct-rfile2
(amax1-statep (*s*, *imax*, *i*, *n*, *lst*)
$\wedge$ (*i* ≠ 0)
$\wedge$ (¬ ilessp (*imax*, get-nth (*i* − 1, *lst*)))
$\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 5)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

EVENT: Disable amax-statep.

EVENT: Disable amax1-statep.

```
; the correctness of the loop.
```

THEOREM: amax1-correctness
amax1-statep (*s*, *imax*, *i*, *n*, *lst*)
$\rightarrow$ ((mc-status (stepn (*s*, amax1-t (*imax*, *i*, *lst*))) = 'running)
$\wedge$ (mc-pc (stepn (*s*, amax1-t (*imax*, *i*, *lst*))) = linked-rts-addr (*s*))
$\wedge$ (iread-dn (32, 0, stepn (*s*, amax1-t (*imax*, *i*, *lst*)))
= amax1 (*imax*, *i*, *lst*))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (*s*, amax1-t (*imax*, *i*, *lst*))))
= linked-a6 (*s*))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (*s*, amax1-t (*imax*, *i*, *lst*))))
= add (32, read-an (32, 6, *s*), 8))
$\wedge$ (read-mem (*x*, mc-mem (stepn (*s*, amax1-t (*imax*, *i*, *lst*))), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: amax1-rfile-correctness
(amax1-statep (*s*, *imax*, *i*, *n*, *lst*) $\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, amax1-t (*imax*, *i*, *lst*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; the correctness of the AMAX program.
```

THEOREM: amax-correctness
amax-statep (*s*, *a*, *n*, *lst*)
$\rightarrow$ ((mc-status (stepn (*s*, amax-t (*n*, *lst*))) = 'running)
$\wedge$ (mc-pc (stepn (*s*, amax-t (*n*, *lst*))) = rts-addr (*s*))
$\wedge$ (iread-dn (32, 0, stepn (*s*, amax-t (*n*, *lst*))) = amax (*n*, *lst*))

$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, amax-t (n, lst))))
= read-rn (32, 14, mc-rfile (s)))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, amax-t (n, lst))))
= add (32, read-rn (32, 15, mc-rfile (s)), 4)))

THEOREM: amax-rfile-correctness
(amax-statep (s, a, n, lst) $\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, amax-t (n, lst))))
= read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: amax-mem-correctness
(amax-statep (s, a, n, lst) $\wedge$ disjoint (x, k, sub (32, 4, read-sp (s)), 16))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, amax-t (n, lst))), k)
= read-mem (x, mc-mem (s), k))

EVENT: Disable amax-t.

```
; amax does find the maximum of an integer list.  We need to prove two
; things:
;      1. the output is no less than any element of the list.
;      2. the output is in the list.
```

THEOREM: ilessp-transitivity
(ilessp (x, y) $\wedge$ ileq (y, z)) $\rightarrow$ ilessp (x, z)

THEOREM: amax1-mono
($\neg$ ilessp ($imax$, $imax1$)) $\rightarrow$ ($\neg$ ilessp (amax1 ($imax$, n, lst), $imax1$))

THEOREM: amax1-max
($i < n$) $\rightarrow$ ($\neg$ ilessp (amax1 ($imax$, n, lst), get-nth (i, lst)))

THEOREM: amax-max
($i < n$) $\rightarrow$ ($\neg$ ilessp (amax (n, lst), get-nth (i, lst)))

THEOREM: get-nth-memberp
($n <$ len (lst)) $\rightarrow$ (get-nth (n, lst) $\in$ lst)

THEOREM: amax1-closed
(($n \leq$ len (lst)) $\wedge$ ($imax \in lst$)) $\rightarrow$ (amax1 ($imax$, n, lst) $\in$ lst)

THEOREM: amax-closed
(($n \leq$ len (lst)) $\wedge$ ($n \not\simeq$ 0)) $\rightarrow$ (amax (n, lst) $\in$ lst)

Index