```
MODULE op_counter
EXTENDS Integers, Sequences, TLC
CONSTANTS N

Procs ≜ 1 .. N

--algorithm op_counter
variables
  ops = [m ∈ Procs ↦ ⟨⟩] ;   utilized to broadcast the operations

macro Broadcast(o)begin
  ops := [m ∈ Procs ↦ Append(ops[m], o)] ;   send the operation to all
end macro ;

macro Update(v)begin   receive the operation
  if Len(ops[self]) > 0 then
    if Head(ops[self]) = "I" then
      v := v + 1 ;
    elsif Head(ops[self]) = "D" then
      v := v + 1 ;
    end if ;
    ops[self] := Tail(ops[self]) ;   clean received msg
  end if ;
end macro ;

process Counter ∈ Procs
variables
  count = 0 ;   local counter
begin Main:
  while TRUE do
    Update(count) ;
    either Increment:
      Broadcast("I") ;
    or Decrement:
      Broadcast("D") ;
    end either ;
  end while ;
end process ;

end algorithm ;
 BEGIN TRANSLATION
VARIABLES ops, pc, count

vars ≜ ⟨ops, pc, count⟩

ProcSet ≜ (Procs)

Init ≜  Global variables
```

```
51          ∧ ops = [m ∈ Procs ↦ ⟨⟩]
52          Process Counter
53          ∧ count = [self ∈ Procs ↦ 0]
54          ∧ pc = [self ∈ ProcSet ↦ "Main"]

56  Main(self) ≜  ∧ pc[self] = "Main"
57                ∧ IF Len(ops[self]) > 0
58                     THEN ∧ IF Head(ops[self]) = "I"
59                                THEN ∧ count' = [count EXCEPT ![self] = count[self] + 1]
60                                ELSE ∧ IF Head(ops[self]) = "D"
61                                           THEN ∧ count' = [count EXCEPT ![self] = count[self] + 1]
62                                           ELSE ∧ TRUE
63                                                ∧ count' = count
64                          ∧ ops' = [ops EXCEPT ![self] = Tail(ops[self])]
65                     ELSE ∧ TRUE
66                          ∧ UNCHANGED ⟨ops, count⟩
67                ∧ ∨ ∧ pc' = [pc EXCEPT ![self] = "Increment"]
68                  ∨ ∧ pc' = [pc EXCEPT ![self] = "Decrement"]

70  Increment(self) ≜  ∧ pc[self] = "Increment"
71                     ∧ ops' = [m ∈ Procs ↦ Append(ops[m], "I")]
72                     ∧ pc' = [pc EXCEPT ![self] = "Main"]
73                     ∧ count' = count

75  Decrement(self) ≜  ∧ pc[self] = "Decrement"
76                     ∧ ops' = [m ∈ Procs ↦ Append(ops[m], "D")]
77                     ∧ pc' = [pc EXCEPT ![self] = "Main"]
78                     ∧ count' = count

80  Counter(self) ≜ Main(self) ∨ Increment(self) ∨ Decrement(self)

82  Next ≜ (∃ self ∈ Procs : Counter(self))

84  Spec ≜ Init ∧ □[Next]_vars

86  END TRANSLATION

88  Eventual Convergence:
89  Safety: i, j : C(xi) = C(xj) implies that the abstract states of i and j are equivalent.
90  Safety ≜ (∀ i, j ∈ Procs : count[i] = count[j])
91  Liveness: i, j : f C(xi) implies that, eventually, f C(xj).
92  Liveness ≜ ◇(∀ i, j ∈ Procs : count[i] = count[j])

94
```

* Modification History
* Last modified *Wed Dec* 12 15:22:32 *PST* 2018 by *ocosta*
* Created Sat *Dec* 01 16:58:15 *PST* 2018 by *ocosta*