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

Procs ≜ 1 .. N

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

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

 receive and process operations, one by one
macro Update(v)begin
  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]) ;   clear processed operation
  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
        ∧ ops = [j ∈ Procs ↦ ⟨⟩]
        Process Counter
        ∧ count = [self ∈ Procs ↦ 0]
        ∧ pc = [self ∈ ProcSet ↦ "Main"]

Main(self) ≜ ∧ pc[self] = "Main"
             ∧ IF Len(ops[self]) > 0
                  THEN ∧ IF Head(ops[self]) = "I"
                            THEN ∧ count' = [count EXCEPT ![self] = count[self] + 1]
                            ELSE ∧ IF Head(ops[self]) = "D"
                                      THEN ∧ count' = [count EXCEPT ![self] = count[self] + 1]
                                      ELSE ∧ TRUE
                                           ∧ count' = count
                       ∧ ops' = [ops EXCEPT ![self] = Tail(ops[self])]
                  ELSE ∧ TRUE
                       ∧ UNCHANGED ⟨ops, count⟩
             ∧ ∨ ∧ pc' = [pc EXCEPT ![self] = "Increment"]
               ∨ ∧ pc' = [pc EXCEPT ![self] = "Decrement"]

Increment(self) ≜ ∧ pc[self] = "Increment"
                  ∧ ops' = [j ∈ Procs ↦ Append(ops[j], "I")]
                  ∧ pc' = [pc EXCEPT ![self] = "Main"]
                  ∧ count' = count

Decrement(self) ≜ ∧ pc[self] = "Decrement"
                  ∧ ops' = [j ∈ Procs ↦ Append(ops[j], "D")]
                  ∧ pc' = [pc EXCEPT ![self] = "Main"]
                  ∧ count' = count

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

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

Spec ≜ Init ∧ □[Next]_vars

 END TRANSLATION

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

└──────────────────────────────────────────────┘
\* Modification History
\* Last modified Wed Dec 12 17:53:13 PST 2018 by ocosta
\* Created Sat Dec 01 16:58:15 PST 2018 by ocosta
```