Some notes on Dijkstra's mutual exclusion algorithm (CACM 8(9), Sept 1965, p. 569).

Here's Dijkstra's mutual exclusion algorithm using different notation. (The interested array is the negation of Dijkstra's b array and the passed array is the negation of Dijkstra's c array.)

```
Shared variables:
interested[1..N] % Boolean entries indicate which processes are interested in using the resource
passed[1..N] % Boolean entries indicate which processes have passed Phase 1
k % integer is the name of a process allowed to try for the resource next

Local variable:
done % Boolean variable that indicates when the process can enter the critical section

interested[i] ← true
done ← false
loop
    loop % Entry Phase 1
        exit when k = i
        if not interested[k] then k ← i % try to go next
    end loop
    passed[i] ← true % begin Entry Phase 2
    done ← true
    for j ← 1..N except i % check if anyone else has passed Phase 1
        if passed[j] then passed[i] ← false; done ← false % go back to Phase 1
    end for
    exit when done
end loop
CRITICAL SECTION
passed[i] ← false
interested[i] ← false
REMAINDER SECTION
```

Exclusion property: No two processes are ever in the critical section simultaneously. The argument for this property is based entirely on the passed array. (It has nothing to do with the interested array or k.) To derive a contradiction, suppose two agents i and j are in the critical section simultaneously. Then, consider the last time each of them set their entry of passed to true prior to entering the critical seciton. Without loss of generality, suppose i did this after j did. Then, when i reads passed[j] during Phase 2, it will see that passed[j] is true, and it will go back to Phase 1. This contradicts the fact that i gets into the critical section.

Progress property: If anybody wants to enter the critical section, eventually somebody does. Suppose some processes are trying to get to the critical section (*i.e.,* in the big loop) and no process is in the critical section. We must show that some process eventually enters the critical section. Eventually k will get set to the name of one of the processes that is trying to enter the critical section. The value of k may change a few times (for example if several processes are about to write their own ids into k) but, eventually, k will stop changing until some progress is made because interested[k] will be true. Once k has settled down to a particular value, that process must just wait until all other processes get kicked out of phase 2. Those processes won't be able to re-enter phase 2. So the process whose id is in k will eventually get into the critical section.

Notice that this algorithm allows starvation: one process may be left out of the critical section forever while another enters it infinitely many times.