Problem 1. Write an algorithm that implements a *fetch-and-increment* object using atomic registers and compare-and-swap objects.

**Reminder:** Fetch-and-increment is a shared object that maintains a single variable \( c \), initialized to 0, and provides a single operation `fetch&inc` with the following sequential specification:

```java
operation fetch&inc()
    c' := c
    c := c + 1
    return c'
end
```

A compare-and-swap object is a shared object that maintains a single variable \( v \), initialized to \( \bot \), and provides a single operation `CAS` with the following sequential specification:

```java
operation CAS(oldVal, newVal)
    v' := v
    if v = oldVal then v := newVal
    return v'
end
```
Solution

Fetch-and-increment has a consensus number of 2, while compare-and-swap (CAS) has an infinite consensus number. Therefore we will use the universal construction to implement a fetch-and-increment object from consensus objects. Then we can replace consensus objects with their implementation\(^1\) from CAS objects. The resulting algorithm is an implementation of fetch-and-increment from CAS.

Universal construction algorithm for fetch-and-increment: \(\text{Shared objects:}\)

- Array of \(n\) atomic registers \(R[1, \ldots, n]\), where \(n\) is the number of processes.
- Infinite list \(C\) of consensus objects.

\(\text{Local objects:}\)

- register \(seq\) the value of which is the number of executed operations by process \(p[i]\), initially \(seq = 0\).
- register \(k\) the value of which is the number of decided batches of requests, initially \(k = 0\).
- list \(Perf\) of performed requests.
- list \(Inv\) of requests which need to be performed.
- local copy \(f\) of fetch-and-increment.

Pseudocode for process \(p[i]::\)

\[
\text{fetch\&inc()}
\]

\[
\begin{align*}
\text{seq} &+\ \\
R[i] &:= (\text{fetch\&inc()}, i, \text{seq}) \quad // \text{inform other processes about the request} \\
\text{repeat} &
\end{align*}
\]

\[
\begin{align*}
\text{Inv} &:= \text{Inv} + R[1, \ldots, n].\text{read} \quad // \text{add new requests of other processes to the list} \\
\text{Inv} &:= \text{Inv} - \text{Perf} \quad // \text{remove performed requests from the list} \\
\text{if} &\quad \text{Inv} \neq \emptyset \quad // \text{if there are requests that were not performed} \\
&\quad k++ \\
\text{Dec} &:= C[k].\text{propose(Inv)} \quad // \text{decide on requests to be performed} \\
\text{Res} &:= f.\text{Dec} \quad // \text{perform all requests from Dec on local copy} f \\
\text{Perf} &:= \text{Perf} + \text{Dec} \quad // \text{add the performed responses to list Perf} \\
\text{if} &\quad (\text{fetch\&inc()}, i, \text{seq}) \in \text{Dec} \quad // \text{if the request by } p[i] \text{ is in} \\
&\quad \text{the list of decided responses} \\
&\quad \text{return the result of } (\text{fetch\&inc()}, i, \text{seq}) \text{ from Res} \\
&\quad \text{return the corresponding response}
\end{align*}
\]

\(^{1}\text{For the implementation of consensus from CAS see the lecture on the limitations of registers}\)