1. Exercise 6.1　Implement Consensus using total-order broadcast

```
Implements: Consensus, instance c.
Uses: TotalOrderBroadcast, instance tob.

# 初始化事件处理
upon event < c, init > do
    decided := false;  # 决定状态标志位
    proposal := ⊥;     # 提议值

# 当有提议值时的事件处理
upon event < c, Propose | v > such that proposal = ⊥ do
    proposal := v;  # 设置提议值
    trigger < tob, Broadcast | proposal >;  # 通过总序广播发送提议值

# 当总序广播交付消息时的事件处理
upon event < tob, Deliver | p, m > do
    if decided = FALSE then  # 如果还未做出决定
        decided := TRUE;     # 设置决定状态为真
        trigger < c, Decide | m >;  # 触发决定事件，决定值为m
```

这段代码实现了Consensus（共识）模块，其中使用了TotalOrderBroadcast（总序广播）来传播提议值。在收到提议时，它通过总序广播发送这个值。当收到总序广播交付的消息时，如果还未做出决定，它将触发决定事件，并以收到的消息作为决定值。这保证了所有参与共识的进程都会以相同的顺序接收并处理提议，从而达成一致的决定。

2. Exercise 6.2 Explain why any consensus algorithm that uses　P actually solves uniform consensus

这段文字解释了为什么任何使用了最终完美故障检测器（ P）的共识算法实际上也解决了均匀共识问题。

假设存在一个使用 P的共识算法，但它没有实现均匀共识。这意味着在某个执行过程（我们称之为ex）中，两个进程p和q做出了不同的决定，其中一个进程在过程中崩溃了。我们不妨假设进程p在执行过程ex中崩溃了。

使用 P的特性，可能存在另一个执行过程（我们称之为ex'），在这个过程中，进程p实际上并没有崩溃，但是被所有其他进程错误地怀疑崩溃了。在这种情况下，进程q在ex'中的决定将与它在ex中的决定相同。但是，这样一来，算法甚至违反了非均匀一致性（即使进程未崩溃，它们也可能作出不同的决定）。

因此，这种假设（即使用 P的共识算法不实现均匀共识）会导致矛盾，因为即使在进程未崩溃的情况下，算法也无法保证所有正确的进程做出相同的决定。所以，我们可以得出结论，任何使用了 P的共识算法实际上也解决了均匀共识问题。