1. The Byzantine Generals problem discussed in class is a form of a fault-tolerant consensus problem in which (1) all non-faulty officers come to a consensus, and (2) if the general is non-faulty then this consensus value is the same as the value set by the general. The problem discussed in class deals with consensus on a single bit (advance or retreat).

Consider a generalization of the algorithm to two bits. Consider only the case of non-forgable (encrypted) messages. The general sets two bits (one of the values 00, 01, 10, 11) instead of the single (attack/retreat) bit. For example, the first bit represents artillery and the second cavalry. The same officers command both artillery and cavalry units. For example, the general issues the command 1,0 to indicate that all units of artillery should attack and all units of cavalry should retreat.

The obvious solution to solve a 2-bit problem is to solve two separate 1-bit problems: Run two completely separate Byzantine agreement problems, one for the artillery and one for the cavalry. Somebody proposes an algorithm to reduce the number of messages. The proposed algorithm with two bits is identical to the one-bit algorithm discussed in class and the text except that messages have two bits. Prove that the algorithm works or give a counterexample. The kind of proof or counterexample used in the class handout is adequate.

The next paragraph describes the algorithm in more detail. Officers (including the general) send messages with two bits. (Recall that a faulty officer also has the option of sending no messages at all.) Each officer sends a message \( x[0], x[1] \) where \( x[j] = 1 \) to indicate that this officer is committed to set bit \( j \) to 1. It sets \( x[j] = 0 \) to indicate that it has not yet committed to set bit \( j \) to 1. For example, an officer sends 10 to indicate that the officer’s artillery unit is committed to attack, and the officer hasn’t yet committed her cavalry unit to attack or not.

2. Consider the Byzantine Generals problem and algorithm discussed in class with unforgeable messages. Consider a problem with a very large number of officers, say 1000002 of officers (including the general), of which at most 2 can be faulty. We showed in class that the problem can be solved in 3 synchronous rounds. Will exactly the same algorithm work in the case where messages can be forged? Prove or give a counterexample.

Note: In the non-encrypted case, a faulty agent \( i \) can send a message to another agent \( j \) of what \( i \) claims to be a copy of a message from a third agent \( k \) even though \( k \) sent no such message.

3. This problem will help you in understanding the proof of Paxos algorithm and in exploring its variants. You will have to write down the Paxos algorithm in understandable pseudo-code or
as in the text (UNITY). Your answer should have separate code for a proposer, an acceptor and a learner.

(a) Messages in the Algorithm

Write down the message types used in the algorithm, i.e., the names of the messages and the types of the arguments in the messages. Write down which type of agent sends each message and which type of agent receives each message.

For example, a message type is \textit{prepare}.

Message type \textit{prepare} has a single argument \textit{n} which is a value in a total order. In this code \textit{n} is a pair \((t, id)\) where \(t\) is a non-negative integers and \(id\) is the static id of a proposer, and comparisons in the total order are made lexicographically.

Message type \textit{prepare} is sent by a proposer and received by an acceptor.

(b) Local Variables of Agents

Write down the local variables and their types used by each agent type: proposer, acceptor and learner.

For example, for agent proposer: a local variable of proposer is \textit{last_n} which as a value in the total order given earlier. \textit{last_n} is the last value of \textit{n} sent by this proposer in a \textit{prepare(n)} message.

(c) Algorithm for each Agent

Write down the algorithm for each type of agent using pseudo-code.

In UNITY the code for each type of agent would have guards and action where the guard would be “receive message \textit{m}” or an external trigger and the action would be a set of assignments possibly coupled with sending messages.

For example, the code for the proposer should be of the form:

\begin{itemize}
  \item \textbf{if} triggered \textbf{then} send \textit{prepare(n)} message to a majority set of acceptors.
  \item \textbf{if} receive \textit{granted}(n, latest_accepted, latest_accepted_n) \textbf{then} ...... 
\end{itemize}

You don’t need to specify what “triggered” is for the proposer because we didn’t discuss this in class. (We will discuss in later class that the proposer sends prepare messages periodically until the proposer gets a message from a learner stating a proposal has been chosen, and we didn’t discuss that aspect in class. In other words, the proposer is “triggered” periodically until it gets a stop message.)

You don’t have to identify the acceptors or define formally what a majority set is.