PROBLEM 1  (approx. 30 %)

The questions in this problem can be solved independently of each other.

Question 1.1:

A concurrent program is given by:

\[
\begin{align*}
\text{var} & \quad x, y : \text{integer} := 0; \\
\text{co} & \quad x := 3; \langle y := y + 1 \rangle \parallel y := x + 2; \ x := 1 \ o c
\end{align*}
\]

(a) For each of the two processes, draw a transition diagram showing its atomic actions.

(b) Determine all possible final states \((x, y)\) of the program.

Question 1.2:

Let \(x\) and \(y\) be integer variables. Determine for each of the predicates \(P, Q,\) and \(R\) whether it is preserved by each of the actions \(a, b,\) and \(c:\)

\[
\begin{align*}
P & \triangleq x + y \neq 0 & a: & \langle y := x + 1 \rangle \\
Q & \triangleq 0 \leq x < y & b: & \langle x := 0 \rangle \\
R & \triangleq x > 0 \lor y > 0 & c: & \langle x = y \rightarrow y := 0 \rangle
\end{align*}
\]

Question 1.3:

Let \(x\) and \(y\) be integer variables and let the temporal logic formula \(F\) be defined by:

\[
F \triangleq \Box (x \geq 0 \land y \geq 0) \land (\Diamond x \neq y \land \Diamond y = 0) \land (x > y \rightarrow x = 0)
\]

(a) Let states be given by pairs \((x, y)\). Give an example of an execution for which \(F\) holds. The execution should be given as a short sequence of states which is repeated forever.

(b) Now, consider the following temporal formulas:

\[
\begin{align*}
G & \triangleq \Box \Diamond x = 0 \\
H & \triangleq \Box \Diamond x = y \\
I & \triangleq x = 0 \leadsto x > 0 \\
J & \triangleq (\Box x = 0) \leadsto y > 0
\end{align*}
\]

For each of the formulas \(G, H, I,\) and \(J,\) determine with a brief argument whether the formula follows from \(F\) by temporal reasoning.
PROBLEM 2 (approx. 35 %)

The questions in this problem can be solved independently of each other.

In a system a module TokBuf is supposed to buffer tokens (represented by their numbers). Tokens are delivered to the module by the operation put(n) where n is a number of tokens. Tokens are consumed by the operation get(t) which awaits that at least a threshold of t tokens are available and then takes them all.

A server-based implementation of such a module is given by:

```
module TokBuf
  op put(n : nat);
  op get(t : nat) returns nat;
body
  process Control;
    var count : nat := C_0;
    r : nat;
    repeat
      in put(n : nat) → count := count + n
      [] get(t : nat) and t ≤ count → r := count; count := 0; return r
    ni
  forever;
end TokBuf;
```

where nat is the type of non-negative integers and $C_0 \in \text{nat}$ is a given constant representing an initial number of tokens.

**Question 2.1:**

The module TokBuf may be used as a semaphore (with initial value $C_0$). Show how the semaphore operations P and V can be implemented using put and get.

**Question 2.2:**

The module TokBuf may be replaced with a monitor which provides the same operations and behaves in the same way.

(a) Write such a monitor. Calls of get may be served in any feasible order.

(b) State informally (i.e. in words) a monitor invariant which expresses that calls of get do not wait unnecessarily.

(c) Show how to optimize the monitor so that no calls of get are woken up in situations where none of the waiting calls can be served.
   [If this is already the case for your solution to (a), just refer to that.]

(d) Determine whether your solution to (c) would still work if spurious wakeups could occur.

**Question 2.3:**

A system with $N \,(N > 0)$ reader processes and $M \,(M > 0)$ writer processes are to be synchronized using the given module TokBuf.

Show how this may be accomplished by stating the initial token count $C_0$ as well as the pre and post protocols for reading and writing.
PROBLEM 3  (approx. 35 %)

The questions in this problem can be solved independently of each other.

In a system, a number of operations $A_1, A_2, \ldots, A_n$ ($n \geq 2$) and corresponding operations $B_1, B_2, \ldots, B_n$ are to be executed the following way:

(*) Initially, $A_1, A_2, \ldots, A_n$ are executed concurrently. The operation $B_i$ may be executed as soon as all $A_j$ for which $j \leq i$ have finished. When all the operations $B_1, B_2, \ldots, B_n$ have finished, the execution starts all over again.

**Question 3.1:**

For a system with $n = 3$, draw a Petri Net in which the six operations $A_1, A_2, A_3, B_1, B_2, B_3$ are synchronized as described by (*). In the net, the operations should be represented by transitions.

**Question 3.2:**

The operations are to be executed by $n$ sequential processes $P_1, P_2, \ldots, P_n$ of the form:

```
process $P[i : 1..n]$;
    repeat
        $A_i$;
        $B_i$
    forever
```

Show how to synchronize these processes using semaphores so that the operations $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ become synchronized as described by (*).

**Question 3.3:**

The processes $P_1, P_2, \ldots, P_n$ are now to be synchronized using a module $Synch$ instead. The module has two operations $doneA(i)$ and $doneB()$ to be called by the processes as shown below:

```
module $Synch$
    op $doneA(i : integer)$;
    op $doneB()$;
end:

process $P[i : 1..n]$;
    repeat
        $A_i$;
        $Synch.doneA(i)$;
        $B_i$;
        $Synch.doneB()$
    forever
```

Write a server process for the module $Synch$ so that the operations $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ become synchronized as described by (*).

[You may assume that rendezvous guards which use the expression $?op$ for the number of pending invocations of $op$ are reevaluated whenever this number changes.]