**PROBLEM 1** (approx. 30 %)

Three processes $P_1$, $P_2$, and $P_3$ execute three operations $A$, $B$, and $C$ respectively. The operations are to be synchronized which is accomplished by exchanging void messages using CSP’s synchronous communication:

```csp
process $P_1$;
repeat
  $A$;
  $P_2!()$
forever

process $P_2$;
repeat
  $B$;
  if $P_1?() \rightarrow P_3?()$
    $P_3?() \rightarrow P_1?()$
fi
forever

process $P_3$;
repeat
  $C$;
  $P_2!()$
forever
```

**Question 1.1:**

Draw a Petri Net in which the three operations $A$, $B$, and $C$ are synchronized in the same way as in the above program. In the net, the operations must appear as transitions.

**Question 1.2:**

Let the number of times the operations $A$ and $C$ have been executed be denoted by $a$ and $c$ respectively. Define a predicate $I$ which characterizes the reachable combinations of $a$ and $c$ in the above program.

$[I$ should thus be an invariant of the program, but you need not show this.$]$

**Question 1.3:**

The operations are now to be executed by three sequential processes $P_A$, $P_B$, and $P_C$:

```csp
process $P_A$;
repeat
  $A$
forever

process $P_B$;
repeat
  $B$
forever

process $P_C$;
repeat
  $C$
forever
```

Show how semaphores can be used to synchronize the three processes so that $A$, $B$, and $C$ become synchronized in the same way as in the above, CSP-based program.
PROBLEM 2  (approx. 30 %)

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

Question 2.1:
A concurrent program is given by:

```plaintext
var x, y : integer := 0;
co y := y + 4 || x := x + 1; || y := x + 2 oc
```

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

(b) Determine all possible final values of $y$ for the program.

Question 2.2:
Consider the concurrent program:

```plaintext
var x, y : integer := 0;
co
    repeat a_1: \{ x \leq 2 \rightarrow x := x + 2; y := 2 \}; b_1: \{ y := 0 \} forever
||
    repeat a_2: \{ y = 0 \rightarrow x := (x + 1) \mod 4 \} forever
oc
```

(a) Prove inductively that following predicate $I$ is an invariant of the program:

$$I \triangleq 0 \leq y \leq x \leq 4$$

(b) Draw the (reachable part of) the transition graph for the program. Only the $(x, y)$ part of the state has to be shown.

(c) Determine whether the predicate $I \land y \leq 2$ is a characteristic invariant of the program (i.e. exactly describes the set of reachable $(x, y)$ states).

(d) Consider the following temporal logic properties:

$$F \triangleq y > 0 \rightarrow x = 1$$
$$G \triangleq x = 3 \rightarrow x = 0$$
$$H \triangleq \Box \Diamond (x = 2)$$
$$J \triangleq \Box \Diamond (y > 0)$$

Determine for each of $F$, $G$, $H$, and $J$ whether the property holds for the program under the assumption of weak fairness. Do the same under the assumption of strong fairness.
PROBLEM 3  (approx. 40 %)

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

Below, a server-based implementation of a synchronization mechanism Balance is shown. The balance has an integer tilt which may be changed by operations left($k$) and right($k$) ($k > 0$). Processes may call the operation level() in order to wait for the tilt being zero.

```plaintext
module Balance
    op level();
    op left($k$ : posinteger);
    op right($k$ : posinteger);

body

    process Control;
    var tilt : integer := 0;
    repeat
        in level() and tilt = 0 → skip
        〈 left($k$ : posinteger) → tilt := tilt + $k$
        〈 right($k$ : posinteger) → tilt := tilt − $k$
        ni
    forever

end Balance;
```

where posinteger is the type of positive integers.

**Question 3.1:**

(a) Specify the effect [as seen by the clients] of the operations level(), left($k$), and right($k$) in the form of conditional atomic actions acting upon the state variable tilt.

(b) Discuss whether the given server implementation ensures weak fairness respectively strong fairness for the level() operation.

(c) Modify the server implementation so that all pending calls of level() are served whenever tilt becomes zero.

**Question 3.2:**

A number of processes are going to use the Balance module for barrier synchronization.

(a) Show how two processes, $P_1$ and $P_2$, can use the given module Balance to establish a one-time barrier (i.e. a synchronization point, which is to be used only once).

(b) Show how $n$ processes, $P_1, P_2, \ldots, P_n$ ($n \geq 2$), can use the module to establish a one-time barrier.

(c) Discuss whether the solution proposed for (b) can be used as a normal barrier [i.e. be used for repeated synchronization among the $n$ processes].

**Question 3.3:**

The given module Balance is to be replaced with a monitor which provides the same operations and behaves in the same way.

(a) Write such a monitor.

(b) Define a predicate $I$ expressing that calls of level() do not wait unnecessarily and argue briefly that $I$ is an invariant of the monitor.

(c) Assume that the usual signal_all($c$) and empty($c$) condition queue operations are not available. Implement the monitor under this restriction.