PROBLEM 1  (approx. 30 %)

Three processes $P_A$, $P_B$, and $P_C$ execute three operations $A$, $B$, and $C$ respectively. The operations are to be synchronized, which is accomplished by means of semaphores:

```plaintext
var $SA, SBA, SBC, SC : \text{semaphore}$;

$SA := 0; \quad SBA := 0; \quad SBC := 0; \quad SC := 0$;

process $P_A$:
    repeat
        $A$;
        $\text{V}(SBA)$;
        $\text{P}(SA)$;
    forever

process $P_B$:
    repeat
        $B$;
        $\text{V}(SA)$;
        $\text{P}(SBA)$;
        $\text{V}(SC)$;
        $\text{P}(SBC)$;
    forever

process $P_C$:
    repeat
        $\text{P}(SC)$;
        $C$;
    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 should be represented by 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.

**Question 1.3:**

The operations are now to be executed by three sequential CSP-processes $P_1$, $P_2$, and $P_3$ respectively:

```plaintext
process $P_1$:
    repeat
        $A$;
    forever

process $P_2$:
    repeat
        $B$;
    forever

process $P_3$:
    repeat
        $C$;
    forever
```

Show how the processes may exchange void messages using CSP’s synchronous communication so that $A$, $B$, and $C$ are synchronized in the same way as in the above, semaphore-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:

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

(a) For each of the two processes, draw a transition diagram showing its atomic actions. You may assume left-to-right evaluation of expressions.

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

Question 2.2:

Consider the concurrent program:

```
var x, y : integer := 0;
co
  repeat a_1: \{ x \leq 1 \land y \geq x \rightarrow (x, y) := (x + y, y + 1) \} forever
  || repeat a_2: \{ y = 1 \rightarrow x := 1 \} forever
  || repeat a_3: \{ y \leq 3 \rightarrow x := y;  y := 0 \} forever
oc
```

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

\[
I \triangleq 0 \leq y \leq x + 1 \land 0 \leq x \leq 3
\]

(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 \(I \land y \leq 3\) 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:

\[
F \triangleq \square\Diamond(y = 1) \quad H \triangleq \Diamond\Box(\neg(x = 1 \land y = 0) \Rightarrow \Diamond(x = 2))
\]

\[
G \triangleq \square\Diamond(y = 2(x - 1)) \quad J \triangleq y = 1 \Rightarrow x \geq 2
\]

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 ModCount is shown. It
comprises an integer counter which may be incremented by the operation incr(). Processes may
call the operation pass() in order to wait for the counter to reach a multiple of a given constant
$K_0 \geq 2$. [Internally, the counter is just counted modulo $K_0$.]

```plaintext
module ModCount
  op incr();
  op pass();
body

process Control;
  var count : integer := 0;
  k : integer := $K_0$;
  repeat
    in incr() \rightarrow count := (count + 1) mod k
    [] pass() and count = 0 \rightarrow skip
    ni:
      if count = 0 then for i in 1..?pass do in pass() \rightarrow skip ni
    forever
end ModCount;
```

Question 3.1:

(a) Explain which effect is obtained by the if-statement.

(b) Show how to extend the module ModCount with an operation set($l : integer$) which (for
$l \geq 2$) sets $l$ as the new value of $k$, but not before the counter has reached a multiple of
the current $k$. Until then, the call of set($l$) must block.

Question 3.2:

(a) Show how $n$ processes, $P_1, P_2, \ldots, P_n$ ($n \geq 2$), can use the given module ModCount
to establish a one-time barrier (i.e. a synchronization point, which is to be used only once).
The constant $K_0$ may be defined to an appropriate value.

(b) Explain why the solution proposed for (a) cannot be used as a normal barrier (i.e. be used
for repeated synchronization among the $n$ processes) and show how the processes may use
two instances the module, ModCount$_1$ and ModCount$_2$, to achieve the effect of a normal
barrier.

Question 3.3:

The given module ModCount 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 pass() do not wait unnecessarily and argue
briefly that $I$ is an invariant of the monitor.

(c) Discuss whether your solution to (a) would be robust towards spurious wakeups. If not,
write a version of the monitor that is so.

[If needed, you may use the function length($c$) which returns the actual number of processes
waiting on a condition queue $c$.]