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:

```
var SA, SBA, SBC, SC : semaphore;
SA := 0;  SBA := 0;  SBC := 0;  SC := 0;
```

```
process $P_A$:
  repeat
    A;
    V(SBA);
    P(SA);
  forever

process $P_B$:
  repeat
    B;
    V(SC);
    V(SA);
    P(SBA);
  forever

process $P_C$:
  repeat
    P(SC);
    C;
    V(SBC);
  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:**

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

```
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.

**Question 1.3:**

Determine with a brief argument whether in the semaphore-based program, the two semaphores $SBA$ and $SBC$ may be replaced by a common semaphore $SB$ (also initialized to 0 and such that $V(SBA)$ is substituted by $V(SB)$ etc.).
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 x := x + y + 1 || y := y + 3 || x := y oc
```

(a) For each of the three processes, draw a transition diagram showing its atomic actions.
(b) Determine all possible final values of $x$ for the program.

Question 2.2:
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 $c$:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Definition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$x + y &gt; 0$</td>
<td>$a$: $\langle x := x + 1 \rangle$</td>
</tr>
<tr>
<td>$Q$</td>
<td>$x &lt; 0 \lor y \geq 0$</td>
<td>$b$: $\langle y \leq 0 \rightarrow y := y + x \rangle$</td>
</tr>
<tr>
<td>$R$</td>
<td>$y &lt; x$</td>
<td>$c$: $\langle y := 5 \rangle$</td>
</tr>
</tbody>
</table>

Question 2.3:
Let $x$ and $y$ be integer variables and let the temporal logic formula $F$ be defined by:

$$F \triangleq \square(x \geq 0 \land \diamond x > 0) \land (y > 0 \Rightarrow y = 0) \land \square y \geq x$$

(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 (part of) which is repeated forever.
(b) Assume that $F$ is valid for a program. State the strongest program invariant which is implied by $F$.

Now, consider each of the following actions within the program:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$:</td>
<td>$\langle \text{await } x = 0 \rangle$</td>
</tr>
<tr>
<td>$a_2$:</td>
<td>$\langle \text{await } x = 1 \rangle$</td>
</tr>
<tr>
<td>$a_3$:</td>
<td>$\langle \text{await } y &gt; 0 \lor x = 0 \rangle$</td>
</tr>
<tr>
<td>$a_4$:</td>
<td>$\langle \text{await } x = y \rangle$</td>
</tr>
</tbody>
</table>

Assume that control has reached the particular action.
(c) Determine which of the actions will be eventually executed assuming weak fairness.
(d) Determine which of the actions will be eventually executed assuming strong fairness.
**PROBLEM 3  (approx. 40 %)**

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

In a system, there are a number of donor processes and a number of donee processes. In the system, donations are modelled by the type posinteger of positive integers. Transfer of donations takes place through a module Hub as shown below. The donors call donate(d) to offer a donation d. The call of donate() does not return until the donation (together with other donations) is received by a donee. A donee receives a bundle of donations from donors by calling receive(). The size of a bundle is determined by an internal state variable k initialized to the constant \( K_0 \geq 1 \). A call of receive() will await that there are k active calls of donate() before receiving their total donations.

```
module Hub
    op donate(c : posinteger);
    op receive() returns posinteger;

body
    process Control;
    var total : integer;
        k : posinteger := K_0;
    repeat
        in receive() and ?donate \geq k \rightarrow
            total := 0;
        for j in 1..k do
            in donate(d : posinteger) \rightarrow total := total + d ni;
        return total
    ni
    forever;
end Hub;
```

**Question 3.1:**

(a) Show how to extend the given module Hub with an operation set(b : posinteger) which should change the bundle size k to b.

(b) Characterize the behaviour of Hub, when k is set to 1.

**Question 3.2:**

The given module Hub (without set()) is to be replaced with a monitor which provides the same operations and behaves in the same way.

(a) Assuming that only one donee process may call receive(), write such a monitor.

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

(b) State a monitor invariant expressing that calls of donate() do not wait unnecessarily.

(c) Describe how to modify your monitor such that it may work for several donee processes calling receive() concurrently.

**Question 3.3:**

Show how n processes, \( P_1, P_2, \ldots, P_n \) (\( n > 1 \)), may use the given module Hub for barrier synchronization. [The solution may define the constant \( K_0 \) to an appropriate value.]