Solutions for Exercise Class 4

1.

```
process Merge;
  var x : integer;
  do A?x → C!x
  [] B?x → C!x
  od
```

2.

```
process Sum;
  var x, y : integer;
  do true →
    if A?x → B?y
    [] B?y → A?x
    fi;
    C!x + y
  od
```

3. In order to meet, all processes must synchronize pairwise by CSP-communications. Care must be taken to avoid deadlock.

```
process P1;          process P2;          process P3;
  repeat               repeat               repeat
    P2!();               P1?();               P2?();
    P3?();               P3!();               P1!();
    ⋮                    ⋮                    ⋮
  forever              forever              forever
```

4. Solution for Andrews Ex. 8.14

```
module Account
  op deposit(amount : posinteger);
  op withdraw(amount : posinteger);
body
  var bal : integer := 0;

  process AccountServer;
    repeat
      in deposit(amount)                          → bal := bal + amount
      [] withdraw(amount) and amount ≤ bal        → bal := bal − amount
      ni
    forever;

end Account;
```

5. Solution for Andrews Ex. 8.15

(a)

```
module ABmeeting
  op MeetA();
  op MeetB();
body

  process MeetingServer;
    repeat
      in MeetA() →
        in MeetB() → skip ni;
        in MeetB() → skip ni;
      ni
    forever;

end ABmeeting;
```

(b)

```
module ABmeeting
  op MeetA();
  op MeetB();
body

  process MeetingServer;
    repeat
      in MeetA() →
        in MeetB() →
          in MeetB() → skip ni
        ni
      ni
    forever;

end ABmeeting;
```

6. Solution for Rendez.1

(a)

```
module Event
  op Pass();
  op Clear(var r : integer);
  op Release(v : posinteger);
body
  var S : integer;
  process EventServer;
    repeat
      in Pass() and S > 0 → skip
      [] Clear(var r)     → r := S; S := 0
      [] Release(v)       → S := S + v;
                            for i in 1..?Pass do
                              in Pass() → skip ni
      ni
    forever;
end Event;
```

[The loop in the *Release* branch ensures that all current calls of *Pass* are processed before a possible call of *Clear* as in the monitor version.]

(b) The semaphore operation P(s) is implemented by:

```
var l : integer;
repeat
  e.Pass;
  e.Clear(l)
until l > 0;
if l > 1 then e.Release(l − 1)
```

[The call of *Pass* ensures that the semaphore value is not tested (with *Clear*) until it is known to have been positive. Hereby a busy wait is reduced to a semi-busy one being much less resource demanding.]