- **Esercizio 1**
  Un conto corrente bancario è condiviso da diverse persone (processi). Ciascuno può depositare o prelevare soldi dal conto. Il saldo non deve mai diventare negativo.

  a) Scrivere un monitor per risolvere questo problema. Il monitor deve contenere due procedure entry *deposito(cifra)* e *prelievo(cifra)* (n.b. assumere *cifra* sempre positivo).

  b) Modificare il monitor del punto a) in modo che i prelievi vengano eseguiti con politica FIFO. Assumere che esista una funzione magica *quanto(sosp)* che ritorna la cifra richiesta dal primo processo in attesa sulla condition *sosp* (cioè in attesa del prelievo).

  c) Suppore che la funzione magica *quanto(sosp)* non esista. Modificare il monitor ottenuto in b) per simulare tale funzione nella soluzione.

  Soluzione

  a)

```
type contos = Monitor;
var saldo: integer;
    sosp: condition;
    n_sosp: integer;
Procedure entry deposito(cifra: integer)
begin
    saldo := saldo + cifra;
    if n_sosp > 0 then begin
        n_sosp := n_sosp − 1;
        sosp.signal;
    end;
end;
Procedure entry prelievo(cifra: integer)
var presi: boolean;
begin
    presi := false;
    repeat
        if saldo > cifra then begin
            saldo := saldo − cifra;
            presi := true;
        end;
        else begin
            n_sosp := n_sosp + 1;
            sosp.wait;
            if n_sosp > 0 then begin
                n_sosp := n_sosp − 1;
                sosp.signal;
            end;
        end;
    until presi;
end;
begin
    saldo := 0;
    n_sosp := 0;
end;
```

b)

```
type contom1 = Monitor;
var saldo: integer;
    sosp: condition;
    n_sosp: integer;
Procedure entry deposito(cifra: integer)
begin
    saldo := saldo + cifra;
    if (n_sosp > 0) and (quanto(sosp) < saldo) then begin
        n_sosp := n_sosp − 1;
        sosp.signal;
    end;
end;
Procedure entry prelievo(cifra: integer)
begin
    if (n_sosp > 0) or (cifra > saldo) then begin
        n_sosp := n_sosp + 1;
        sosp.wait;
    end
    saldo := saldo − cifra;
    if (n_sosp > 0) and (quanto(sosp) < saldo) then begin
        n_sosp := n_sosp − 1;
        sosp.signal;
    end;
end;
begin
    saldo := 0;
    n_sosp := 0;
end;
```

c)

```
type contom2 = Monitor;
var saldo: integer;
    sosp: condition;
    n_sosp: integer;
    c: coda;
```

Si suppone di usare la variabile *c* di tipo *coda*; *coda* è un tipo di dato astratto su cui sono definite le procedure *insert(coda,valore)* e *remove(coda)* che, rispettivamente, inseriscono e tolgono elementi dalla coda e una funzione *first(coda)* che restituisce il valore del primo elemento che verrà estratto dalla coda. Infine, la coda viene inizializzata per mezzo della funzione *crea(coda)*.

```
Procedure entry deposito(cifra: integer)
begin
    saldo := saldo + cifra;
    if n_sosp > 0 then begin
        if first(c) < saldo then begin
            n_sosp := n_sosp − 1;
            remove(c);
            sosp.signal;
        end;
    end;
end;
Procedure entry prelievo(cifra: integer)
begin
    if n_sosp > 0 then begin
        n_sosp := n_sosp + 1;
        insert(c,cifra);
        sosp.wait;
    end;
    else if cifra > saldo then begin
        n_sosp := n_sosp + 1;
        insert(c,cifra);
        sosp.wait;
        if first(c) < saldo then begin
            n_sosp := n_sosp − 1;
            remove(c);
            sosp.signal;
        end;
    end;
    else saldo := saldo − cifra;
end;
begin
    saldo := 0;
    n_sosp := 0;
    crea(c);
end;
```

- **Esercizio 2**
  Lungo una strada a due corsie e doppio senso di marcia, in direzione nord–sud, vi è un ponte ad una sola corsia, percorribile a senso unico alternato. Il ponte ha la capacità di N automobili. Le A_i dirette verso nord (sud) possono attraversare il ponte solo se non è attraversato da nessuna A_j diretta verso sud (nord). Scrivere il programma relativo ai processi "A_i *diretta a nord*" e "A_j *diretta a sud*" utilizzando i semafori.

  Soluzione
  Per risolvere il problema occorre:

  1. implementare due liste di attesa, una per le automobili provenineti da nord e l'altra per le automobili provenienti da sud, ed implementarle sfruttando i semafori privati (ciascuna automobile deve essere dotata di un semaforo privato);
  2. realizzare la mutua esclusione per l'accesso alle variabili che governano il ponte e le liste di attesa.

  Le modalità relizzative sono le seguenti:

  - Ciascuna nuova macchina che arriva in prossimità del ponte deve controllare tre condizioni:
    1. se ci sono macchine che stanno attraversando il ponte nell'altro senso;
    2. se ci sono N automobili provenienti dallo stesso verso che stanno già attraversando il ponte;
    3. se ci sono automobili che stanno attraversando il ponte nello stesso senso e ci sono altre automobili che sono in attesa dall'altra parte del ponte.

    In tutti e tre i casi l'automobile deve sostare ad un semaforo privato.
  - Quando un'automobile A_i sta attraversando il ponte, all'uscita da questo deve controllare se è l'ultima: in tal caso, se ci sono automobili in attesa in senso opposto, la prima di queste dovrà iniziare l'attraversamento (per evitare l'attesa indefinita); altrimenti il ponte verrà attraversato da un'altra macchina proveniente dallo stesso senso di marcia. Viceversa, se A_i non è l'ultima automobile ad uscire dal ponte, si deve controllare che non ci siano automobili in attesa alla fine del ponte, nel qual caso si devono risvegliare k=N−r automobili provenienti dallo stesso senso (essendo N la portata del ponte, ed r il numero di auto rimanenti sul ponte una volta che A_i ne è uscita).

```
#define boolean int
#define false 0
#define true !false

boolean SulPonteDaSud = false;
int AutomobiliSulPonte = 0;
/****************** DA NORD ******************/
void AccessoAutomobiliProvenientiDaNord()
    {
       int i;
       i = DeterminaProprioIndice();
       wait(mutex); /* Ottiene la mutua esclusione */
       if (SulPonteDaSud || !Vuota(ListaAttesaDaSud) || AutomobiliSulPonte == N)
       {
            Push(ListaAttesaDaNord,i); /* L'automobile va messa in lista d'attesa */
       }
       else
       {
           SulPonteDaNord = true;
           AutomobiliSulPonte++;
           signal(V(i)); /* NON vogliamo che l'automobile si addormenti ... zzz */
       }
       signal(mutex); /* Rilascia la mutua esclusione */
       wait(V(i)); /* Addormenta l'automobile */
    }
```

```
void UscitaAutomobiliProvenientiDaNord()
    {
        int j;
        wait(mutex); /* Ottiene la mutua esclusione */
        AutomobiliSulPonte– –;
        if (AutomobiliSulPonte == 0) /* Se è l'ultima automobile... */
        {
             /* ...allora si deve controllare che ce non ne siano dall'altra parte */
            if (Vuota(ListaAttesaDaSud))
            {
                 /* Sveglia al più N automobili in attesa da nord */
                while (!Vuota(ListaAttesaDaNord) && AutomobiliSulPonte < N)
                {
                    AutomobiliSulPonte++;
                    j = Pop(ListaAttesaDaNord);
                    signal(V(j));
                }
            }
            else
            {
                /* Sveglia al più N automobili in attesa da sud */
                SulPonteDaNord = false;
                SulPonteDaSud = true;
                while (!Vuota(ListaAttesaDaSud) && AutomobiliSulPonte < N)
                {
                    AutomobiliSulPonte++;
                    j = Pop(ListaAttesaDaSud);
                    signal(V(j));
                }
            }
        }
        else
        {
            if (Vuota(ListaAttesaDaSud) && !Vuota(ListaAttesaDaNord))
            {
                AutomobiliSulPonte++;
                j = Pop(ListaAttesaDaNord);
                signal(V(j));
            }
            else
            {
                SulPonteDaNord = false;
            }
        }
        signal(mutex); /* Rilascia la mutua esclusione */
    }
```

```
/****************** DA SUD ******************/
void AccessoAutomobiliProvenientiDaSud()
    {
        int i;
        i = DeterminaProprioIndice();
        wait(mutex); /* Ottiene la mutua esclusione */
        if (SulPonteDaNord) || !Vuota(ListaAttesaDaNord) || AutomobiliSulPonte == N)
        {
            Push(ListaAttesaDaSud,i); /* L'automobile va messa in lista d'attesa */
        }
        else
        {
            SulPonteDaSud = true;
            AutomobiliSulPonte++;
            signal(V(i)); /* NON vogliamo che l'automobile si addormenti ... zzz */
        }
        signal(mutex); /* Rilascia la mutua esclusione */
        wait(V(i)); /* Addormenta l'automobile */
    }
```

```
void UscitaAutomobiliProvenientiDaSud()
    {
        int j;
        wait(mutex); /* Ottiene la mutua esclusione */
        AutomobiliSulPonte– –;
        if (AutomobiliSulPonte == 0) /* Se è l'ultima automobile... */
        {
            /* ...allora si deve controllare che ce non ne siano dall'altra parte */
            if (Vuota(ListaAttesaDaNord))
            {
                /* sveglia al più N automobili in attesa da sud */
                while (!Vuota(ListaAttesaDaSud) && AutomobiliSulPonte < N)
                {
                    AutomobiliSulPonte++;
                    j = Pop(ListaAttesaDaSud);
                    signal(V(j));
                }
            }
            else
            {
                /* Sveglia al più N automobili in attesa da nord */
                SulPonteDaSud = false;
                SulPonteDaNord = true;
                while (!Vuota(ListaAttesaDaNord) && AutomobiliSulPonte < N)
                {
                    AutomobiliSulPonte++;
                    j = Pop(ListaAttesaDaNord);
                    signal(V(j));
                }
            }
        }
        else
        {
            if (Vuota(ListaAttesaDaNord) && !Vuota(ListaAttesaDaSud))
            {
                AutomobiliSulPonte++;
                j = Pop(ListaAttesaDaSud);
                signal(V(j));
            }
            else
            {
                SulPonteDaSud = false;
            }
        }
        signal(mutex); /* Rilascia la mutua esclusione */
    }
```

- **Esercizio 3**
  Un processo mittente P ed N processi riceventi (R_1, ..., R_N) comunicano tramite un buffer circolare di capacità B messaggi. P deposita i messaggi nel buffer e gli R_i li leggono. Ciascun messaggio deve essere letto da tutti gli R_i e inoltre ciascun R_i deve leggere i messaggi nell'ordine in cui questi sono stati spediti. Tuttavia gli R_i possono leggere uno stesso messaggio in tempi diversi: in un dato istante R_k può aver letto fino a B messaggi più di R_j. Scrivere un monitor per implementare questo modello di comunicazione.

  Soluzione

```
type slot = record
      dati: msg; quanti: 0..N;
      end;
      buffer = array[0..B − 1] of slot;
      atmu = monitor;
var buff: buffer;
     disp: 0..B;
     in: 0..B − 1;
     adest: 0..N;
     dove: array[1..N] of 0..B − 1;
     mitt, dest: condition;
procedure entry send (M: msg)
begin
   if disp = 0 then mitt.wait;
   disp := disp − 1;
   buff[in].dati := M;
   buff[in].quanti := 0;
   in := (in + 1) mod B;
   while adest > 0 do
      begin
         adest := adest − 1;
         dest.signal
      end;
end;
procedure entry receive (i: 1..N; var M: msg)
begin
   if (disp = B or dove[i] = in) then
      begin
         adest := adest + 1;
         dest.wait
      end;
   M := buff[dove[i]].dati;
   buff[dove[i]].quanti := buff[dove[i]].quanti + 1;
   if buff[dove[i]].quanti = N then
      begin
         disp := disp + 1;
         mitt.signal
      end;
   dove[i] := (dove[i] + 1) mod B;
end;
begin {inizializzazione}
   disp := B − 1;
   in := 0;
   for adest := 1 to N do dove[adest] := 0;
   adest := 0;
end;
```

Esercizi da svolgere

1. N processi P_1, P_2, P_3, ..., P_N condividono 3 stampanti; prima di usare una stampante un processo P_i invoca una *request(stamp)*, che ritorna l'identificatore della stampante libera da usare. Dopo avere utilizzato la stampante, il processo la rilascia con una *release(stamp)*.

   (a) Sviluppare un monitor che implementa *request* e *release*.

   (b) Assumendo che i processi abbiano associato delle priorità che vengono passate come parametro addizionale alla *request*, modificare le procedure in modo che le stampanti vengano assegnate ai processi in attesa con priorità maggiore (**NB**: le strutture dati già definite nel monitor non vanno modificate).

2. Consideriamo il *processo produttore* che produce pagine di stampa e le deposita nel buffer di una stampante laser. Il *processo consumatore* corrispondente procede al prelevamento dell'informazione dal buffer ed alla successiva fase di stampa.

   (a) Il produttore non può inserire una nuova pagina se il buffer della stampante è pieno;

   (b) Il consumatore non può prelevare una pagina per stamparla se il buffer è vuoto.

   Se N è il numero massimo di pagine che possono essere contenute nel buffer, d è il numero di pagine attualmente depositate dal produttore ed e il numero di pagine estratte dal consumatore, deve essere sempre verificata la condizione:

   $$0 \leq d - e \leq N.$$

   Descrivere la soluzione del problema, utilizzando due semafori, "*spazio–disponibile*" e "*pagina–disponibile*", i cui valori iniziali sono rispettivamente N e 0.