Esercizio di Sincronizzazione tra Processi:
Ponte a Senso Unico Alternato con Capacità Limitata

Supponiamo sempre di avere un ponte stretto che permette il passaggio delle auto solo in un verso per volta, a senso unico alternato. Supponiamo che la resistenza del ponte sia limitata, e quindi la sua capacità di carico sia limitata ad un numero massimo di auto CAPAC contemporaneamente presenti sul ponte.

1° Soluzione con l'uso del costrutto monitor

```
program PonteASensoUnicoConCapacitaLimitata;

const CAPAC = ...;
type dir  = ( su, giu );

{Notiamo che la struttura dei processi auto non cambia rispetto a quella del ponte semplice }
type auto  = process ( d : dir );
begin       Ponte. IN (d);
            < transita >
       Ponte .OUT (d);
end;

{Con l'introduzione del vincolo della capacità del ponte, si devono introdurre nel monitor:
- una variabile condition per l'attesa di accesso al ponte - attesa_dir - in cui sono sospese le auto quando il ponte non ne permette l'accesso a causa della direzione;
- una coda di sospensione per raggiunta capacita' massima del ponte : attesa_cap; per questa invece basta una sola direzione
- contatori opportuni, di auto in passaggio sul ponte: nautoponte, numero totale di auto sul ponte, nauto, di auto che hanno gia' eseguito la in e non hanno ancora fatto la out.
}

type bridge = monitor ;

var  direz : dir ; nauto, nautoponte: integer;
    attesa_cap : condition;
    attesa_dir: condition;

procedure entry IN ( d : dir);
begin
    if (d <> direz and nauto <> 0 )
       then  attesa_dir. wait ;
       { c'e' sospensione se la direzione non è quella corretta }
    nauto := nauto +1;
    direz := d ;
```

```
    {in questo momento non mi considero ancora sul ponte devo prima controllare la capacità}
     if nautoponte = CAPAC then attesa_cap.wait;
     nautoponte := nautoponte + 1;
end;

procedure entry OUT;
begin
     nautoponte := nautoponte - 1 ;
    { segnalazione delle auto eventualmente sospese per vincoli di capacita' del ponte }
     attesa_cap.signal;
    { la signal non ha effetto se la coda è vuota }
     nauto := nauto - 1;

    { segnalazione delle auto nella direzione opposta, se ce ne sono, o di auto nella stessa direzione}
     if ( nauto = 0 ) then
        begin
              while attesa_dir.queue do  attesa_dir.signal;
end;

var   Ponte : bridge;
      auto1, auto2,.... : auto(su);
      auto100, auto101,.... : auto(giu);

begin end.
```

2° Soluzione con l'uso del costrutto monitor

{soluzione che fa uso dello stesso tipo di coda diverse (ma comunque una per ogni direzione) per il problema della direzione e per il problema della capacità. In effetti la prima soluzione di prima è un po' ridondante }

```
program PonteASensoUnicoConCapacitaLimitata;

const CAPAC = ...;
type dir  = ( su, giu );

{la struttura dei processi auto non cambia rispetto alla prima soluzione }
type auto = process ( d : dir );
begin      Ponte. IN (d);
           < transita >
      Ponte .OUT (d);
```

```
end;
```

{Questa volta usiamo come variabili del monitor:

- due variabili condition per l'attesa di accesso al ponte - attesa_dir [dir] - in cui sono sospese le auto quando il ponte non ne permette l'accesso a causa della direzione o della capacità. Questa volta servono due variabili condition perché a causa della capacità ci potrebbero essere auto sospese contemporaneamente in entrambe le direzioni sullo stesso tipo di coda,
- contatori nauto, numero totale di auto sul ponte

}

```
type bridge = monitor ;

var   direz : dir;
      nauto : integer;
      attesa_dir: array [dir] of condition;

procedure entry IN ( d : dir);
begin
    if (d <> direz and nauto <> 0 )   or  (nauto+1>CAPAC)
       then  attesa_dir[d]. wait ;
       { c'e' sospensione se la direzione non è quella corretta oppure se si superasse la capacità
       massima}
    nauto := nauto +1;
    direz := d ;
end;

procedure entry OUT (d : dir);
{qui d serve perché i risvegliati cambiano direz}
begin
      nauto := nauto - 1;
      { segnalazione delle auto nella direzione opposta, se ce ne sono, o di auto nella stessa
direzione}
      if ( nauto = 0 ) then
      begin
      while ( attesa_dir [ other (d) ]. queue and nauto<CAPAC)
do
            begin
                  attesa_dir [other(d) ].signal;
            end;
      end;
```

{ le segnalazioni vengono fatte solo fino a quando si è sicuri di andare a svegliare solo le macchine che hanno il diritto di entrare. Devo controllare tutte le condizioni }

```
        else if attesa_dir[d].queue  then
        begin
        while ( attesa_dir[d].queue and nauto<CAPAC ) do
                begin
                        attesa_dir [d].signal
                end;
       end;

end;

var   Ponte : bridge;
        auto1, auto2,.... : auto(su);
        auto100, auto101,.... : auto(giu);

begin end.
```

Con l'uso delle regioni critiche condizionali:

```
type d = (su,giu);

VAR ponte : shared record
    nauto : integer;        { contatore degli utenti che hanno acquisito il ponte }
    dir  :  d;              { direzione corrente del ponte}
                end record;

procedure IN (miadir : d);
region ponte
when ((((miadir = dir) and (nauto < CAPAC)) or ((miadir <> dir) and (nauto=0))))
{si entra se o nella direzione corretta e non raggiungo la capacità o la direzione è sbagliata ma non ci sono auto }
do
  dir := miadir; nauto + := 1;
end;
end IN;

procedure OUT (miadir : d);
region ponte
do
   nauto - := 1;
end region;
end OUT;


process type utente;
var dir : d;
begin   loop   <scegli dir>
                    IN (dir);
                    <passa sul ponte>
                    OUT (dir);
          end loop;
end utente;

var u1, ..., un : utente;
```