Problema dello **sleeping barber**.

In un barber shop lavora un solo barbiere, vi è una sola sedia adibita al taglio, e vi sono N sedie per i clienti in attesa. Assumiamo N=20.

Comportamento del barbiere:

- all'apertura del negozio si mette a dormire nella sedia adibita al taglio, in attesa che un cliente entri e lo svegli;
- quando ci sono clienti in attesa, il barbiere li chiama e li serve uno alla volta;
- quando non ci sono clienti in attesa, il barbiere si rimette a dormire nella sedia adibita al taglio.

Comportamento del cliente:

- quando entra nel negozio, se non ci sono sedie libere va a cercarsi un altro barbiere;
- quando entra nel negozio, se c'è almeno una sedia libera ne occupa una, svegliando il barbiere se sta dormendo, ed attendendo di essere chiamato dal barbiere per il taglio.

Programmare il barbiere ed il singolo cliente.

```
semaphore clients_in_shop=0; barber_free=0; mutex=1;
int waiting_clients = 0; /* variabile condivisa */
final int CHAIRS = 20;
```

```
barbiere( ){
  while(true){
    wait(clients_in_shop);
    wait(mutex);
    waiting_clients --;
    signal(barber_free);
    signal(mutex);
    cut_hair( );
  }
}
```

```
cliente( ){
  wait(mutex);
  if(waiting_clients < CHAIRS){
    waiting_clients ++;
    signal(clients_in_shop);
    signal(mutex);
    wait(barber_free);
    receive_cut( );
  }
  else{
    signal(mutex);
  }
}
```

Un vecchio ponte consente di attraversare un fiume nelle direzioni nord→sud e sud→nord, con i seguenti vincoli:

- per ragioni di peso, in ogni istante al più un veicolo può passare sul ponte;
- se un veicolo trova il ponte occupato, attende che si liberi (non è previsto che il veicolo decida di rinunciare ad attraversare il ponte);
- dopo che un veicolo ha attraversato il ponte in una direzione, se ci sono veicoli in attesa su entrambi i lati allora deve passare per primo un veicolo che viaggia nella direzione opposta.

Programmare il veicolo che viaggia in senso nord→sud (programma **goingToSouth**) ed il veicolo che viaggia in senso sud→nord (programma **goingToNorth**)

```
semaphore mutex=1; toNorth=0; toSouth=0;
boolean bridgeFree=true; int bookToNorth=0; int bookToSouth=0;

enteringNorth( ){
  wait(mutex);
  if(bridgeFree){
     bridgeFree=false;
     signal(mutex);
  else{
     bookToSouth++;
     signal(mutex);
     wait(toSouth);
  }
}

exitingSouth( ){
   wait(mutex);
   if(bookToNorth>0){
      bookToNorth--;
      signal(toNorth);
   }
   else{
     if(bookToSouth>0){
        bookToSouth--;
        signal(toSouth);
     }
     else{
        bridgeFree=true;
     }
   }
  signal(mutex);
}

goingToSouth( ){
  enteringNorth( );
  crossingToSouth( );
  exitingSouth( );
}

goingToNorth( ){
  enteringSouth( );
  crossingToNorth( );
  exitingNorth( );
}
```

Supponiamo che il ponte non abbia più problemi di peso, e possa reggere più veicoli, ma abbia una sola corsia:

- essendoci una sola corsia, in ogni istante tutti i veicoli sul ponte devono andare nella medesima direzione;
- un veicolo può entrare sul ponte nei seguenti casi:

  - il ponte è libero,

  - sul ponte ci sono veicoli che stanno andando nella sua medesima direzione e dall'altro lato del ponte non ci sono veicoli in attesa;

```
semaphore mutex=1; toNorth=0; toSouth=0;
int bookToNorth=0;  int bookToSouth=0;
int goingToNorth=0; int goingToSouth=0;

enteringNorth( ){
  wait(mutex);
  if(goingToNorth=0 & bookToNorth=0){
      goingToSouth++;
      signal(mutex);
  else{
      bookToSouth++;
      signal(mutex);
      wait(toSouth);
  }
}
```

```
exitingSouth( ){
    wait(mutex);
    goingToSouth--;
    if(goingToSouth=0){
      while(bookToNorth>0){
        goingToNorth++
        bookToNorth--;
        signal(toNorth);
      }
    }
    signal(mutex);
}
```

Per smaltire il traffico con maggior efficienza, supponiamo ora che quando un veicolo è in attesa ad un lato del ponte perché altri veicoli stanno andando verso il lato in cui si trova, sia consentito ad ulteriori 10 veicoli di entrare dal lato opposto.

```
semaphore mutex=1; toNorth=0; toSouth=0; boolean bridgeFree=true;
int bookToNorth=0; int bookToSouth=0; int goingToNorth=0;
int goingToSouth=0; extraToNorth=0; extraToSouth=0;

enteringNorth( ){
  wait(mutex);
  if(goingToNorth=0 & bookToNorth=0){
    goingToSouth++;
    signal(mutex);
  }
  else{
    if(goingToSouth>0 & extraToSouth<10{
      extraToSouth++;  goingToSouth++;
      signal(mutex);
    }
    else{
       bookToSouth++;
       signal(mutex);
       wait(toSouth);
     }
  }
}
```

```
exitingSouth( ){
    wait(mutex);
    goingToSouth--;
    if(goingToSouth=0){
      if(extraToSouth>0){
        extraToSouth=0;
      }
      while(bookToNorth>0){
        goingToNorth++;
        bookToNorth--;
        signal(toNorth);
       }
    }
    signal(mutex);
}
```

Un parcheggio ha 30 posti, due ingressi con sbarra, A e B, ed un'uscita.

Quando un veicolo si presenta ad uno dei due ingressi, se c'è almeno un posto libero entra, parcheggia ed esce dal parcheggio, altrimenti prenota l'ingresso ed attende di poter entrare.

Se ci sono veicoli in attesa ad entrambi gli ingressi, vengono fatti entrare quando altri veicoli escono dal parcheggio, aprendo le due sbarre alternativamente.

Quando un veicolo esce dal parcheggio, se ci sono veicoli in attesa ad almeno uno dei due ingressi, ne fa entrare uno.

```
semaphore mutex=1; gateA=0; gateB=0;
int bookA=0; bookB=0; freeSlots=30; turn=0;
```

```
enteringA( ){
  wait(mutex);
  if(freeSlots>0){
     freeSlots--;
     signal(mutex);
  }
  else{
     bookA++;
     signal(mutex);
     wait(gateA);
  }
```

```
enteringB( ){
  wait(mutex);
  if(freeSlots>0){
     freeSlots--;
     signal(mutex);
  }
  else{
     bookB++;
     signal(mutex);
     wait(gateB);
  }
```

```
inA( ){enteringA( ); park( ); exiting( );}
inB( ){enteringB( ); park( ); exiting( );}
```

```
exiting( ){
   wait(mutex);
   if(freeSlots=0){
      if((turn=0 |
bookB==0) & bookA>0){
         bookA--;
         turn=1;
         signal(gateA);
      }
      else{
         if(bookB>0){
            bookB--;
            turn=0;
            signal(gateB);
         }
         else{ freeSlots++;
         }
   }
   else{
      freeSlots++;
   }
   signal(mutex);
}
```