Un parcheggio offre 30 posti auto, non tutti uguali:

- 10 posti di tipo "S": possono ospitare solo vetture di tipo "S";
- 10 posti di tipo "M": possono ospitare vetture "S" oppure "M";
- 10 posti di tipo "L": possono ospitare vetture "S" oppure "M" oppure "L".

("S","M", "L" rappresentano le tre possibili dimensioni di vetture/posti auto).

Le vetture che non possono accedere al parcheggio vengono bloccate all'ingresso. In questo caso, quando queste vetture verranno sbloccate potranno accedere solamente a parcheggi del tipo corrispondente (vetture "S" in posti "S", vetture "M" in posti "M", vetture "L" in posti "L").

Una vettura "S" che arriva all'ingresso puo' accedere ad un posto "M" solo se non ci sono posti liberi "S" e almeno 3 posti "M" sono occupati da vetture "M".

Una vettura "S" che arriva all'ingresso puo' accedere ad un posto "L" solo se non ci sono posti liberi "S" e la vettura non puo' accedere a posti "M";

Una vettura "M" che arriva all'ingresso puo' accedere ad un posto "L" solo se non ci sono posti liberi "M" e le vetture "M" gia' presenti nel parcheggio sono meno di 15.

Programmare l'ingresso e l'uscita delle vetture di ogni tipo usando i semafori con la semantica tradizionale.

variabili:

busyL = busyM = busyS = 0; // number of busy slots, always between 0 and 10

inM = 0; // number of M cars in the parking (slots M or L)

inMM = 0; // number of M cars in slots M

wL = wM = wS =0; // number of cars waiting at gate

semafori:

mutex = 1; // mutual exclusion, value always between 0 and 1

semL = semM = semS = 0 // semaphores to block cars, always value 0

```
boolean canSS(){return busyS<10;}

boolean canMM(){return busyM<10;}

boolean canLL(){return busyL<10;}

boolean canSM(){return (busyM<10 & inMM>=3);}   // condition
busyS>=10 not checked

boolean canSL(){return (busyL<10 & ! canSM());} // condition
busyS>=10 not checked

boolean canML(){return (busyL<10 & inM<15);}    // condition busyM>0
not checked




void inSS(){busyS++;} // car S enters slot S
void inSM(){busyM++;} // car S enters slot M
void inSL(){busyL++;} // car S enters slot L
void inMM(){busyM++; inM++; inMM++;} // car M enters slot M
void inML(){busyL++; inM++;}         // car M enters slot L
void inLL(){busyL++;}                // car L enters slot L

void outS(){if(wS>0){wS--; signal(semS);} else{busyS--;}} // car S
exits from slot S

void outM(){if(wM>0){wM--; inM++; inMM++; signal(semM);}
else{busyM--;}}//car S or M exits from slot M

void outL(){if(wL>0){wL--; signal(semL);} else{busyL--;}} // car S
or M or L exits from slot L

void carS(){
```

```
  wait(mutex);
  if(canSS){busyS++; signal(mutex); <park>; wait(mutex); outS();
signal(mutex);}
  else{
    if(canSM){busyM++; signal(mutex); <park>; wait(mutex); outM();
signal(mutex);}
    else{
       if(canSL){busyL++; signal(mutex); <park>; wait(mutex);
outL(); signal(mutex);}
       else{
          wS++; signal(mutex); wait(semS); <park>; wait(mutex);
outS(); signal(mutex);}
       }
    }
  }
}

void carM(){
  wait(mutex);
  if(canMM){busyM++; inM++; inMM++; signal(mutex); <park>;
             wait(mutex); inM--; inMM--; outM(); signal(mutex);}
    else{
       if(canML){busyL++; inM++; signal(mutex); <park>;
                  wait(mutex); inM--; outL(); signal(mutex);}
       else{
          wM++; signal(mutex); wait(semM); <park>;
          wait(mutex);  outM(); signal(mutex);}
       }
    }
}

void carL(){
  wait(mutex);
  if(canLL){busyL++; signal(mutex); <park>; wait(mutex); outL();
signal(mutex);}
  else{
    wL++; signal(mutex); wait(semL); <park>; wait(mutex); outL();
signal(mutex);
  }
}
```