PRÁCTICA 2: PUENTE DE AMBITO

1.] Desarrolla en papel la solución y escribe el invariante.

MONITOR PUENTE

```
n_north: int=0
n_south: int=0
n_ped: int=0
```
Número de coches (norte o sur) y peatones que están cruzando el puente.

```
waiting_N: int=0
waiting_S: int=0
waiting_P: int=0
```
Número de coches (norte o sur) y peatones que están esperando a cruzar el puente.

```
pass_N: VC
pass_S: VC
pass_P: VC
```
Variables condición que permiten o no el paso de coches o peatones para cruzar.

```
wait_N: VC
wait_S: VC
wait_P: VC
```
Variables condición que permiten o no que coches y peatones estén esperando a cruzar.

$$INV \equiv n\_north \geq 0 \wedge n\_south \geq 0 \wedge n\_ped \geq 0 \wedge$$
$$waiting\_N \geq 0 \wedge waiting\_S \geq 0 \wedge waiting\_P \geq 0 \wedge$$
$$n\_north > 0 \longrightarrow n\_south = 0 \wedge n\_ped = 0 \wedge$$
$$n\_south > 0 \longrightarrow n\_north = 0 \wedge n\_ped = 0 \wedge$$
$$n\_ped > 0 \longrightarrow n\_north = 0 \wedge n\_south = 0$$

```
wants_enter_car_N():
    INV
    wait_N.wait(n_north==0 v (waiting_S==0 ∧ waiting_P==0))
    waiting_N++ (=> waiting_N > 0)
    INV (∧ waiting_N > 0)
    pass_N.wait(n_south==0 ∧ n_ped==0)
    INV(∧ waiting_N > 0 ∧ n_south==0 ∧ n_ped==0)
    waiting_N--
    INV(n_south==0 ∧ n_ped==0)
    n_north++ (=> n_north > 0)
    INV(∧ n_south==0 ∧ n_ped==0 ∧ n_north > 0)
```

```
    wait_S.notify
    wait_P.notify
leaves_car_N():
    INV(∧ n_south==0 ∧ n_ped==0 ∧ n_north>0)
    n_north--1(⇒n_north≥0)
    INV(∧ n_south==0 ∧ n_ped==0)
    if n_north==0:
        pass_P.notify
        pass_S.notify
        wait_N.notify
    INV
wants_enter_car_S():
    INV
    wait_S.wait(n_south==0 ∨(waiting_N==0∧waiting_P==0))
    waiting_S++1(⇒waiting_S>0)
    INV(∧ waiting_S>0)
    pass_S.wait(n_north==0∧ n_ped==0)
    INV(∧waiting_S>0 ∧ n_north==0 ∧n_ped==0)
    waiting_S --1(⇒waiting_S≥0)
    INV(∧ n_north==0∧n_ped==0)
    n_south++1(⇒n_south>0)
    INV(∧ n_north==0∧n_ped==0 ∧ n_south>0)
    wait_P.notify
    wait_N.notify
leaves_car_S():
    INV(∧ n_north==0∧n_ped==0∧ n_south>0)
    n_south--1(⇒n_south≥0)
    INV(∧ n_south==0∧ n_ped==0)
    if n_south==0:
        pass_N.notify
        pass_P.notify
        wait_S.notify
    INV
```

```
wants_enter_pedestrian():
    INV
    wait_P.wait(n_ped == 0 v (waiting_N == 0 ∧ waiting_S == 0))
    waiting_P += 1 (=> waiting_P > 0)
    INV (∧ waiting_P > 0)
    pass_P.wait(n_north == 0 ∧ n_south == 0)
    INV (∧ waiting_P > 0 ∧ n_north == 0 ∧ n_south == 0)
    waiting_P -= 1 (=> waiting_P ≥ 0)
    INV (∧ n_north == 0 ∧ n_south == 0)
    n_ped += 1 (=> n_ped > 0)
    INV (∧ n_north == 0 ∧ n_south == 0 ∧ n_ped > 0)
    wait_N.notify
    wait_S.notify
leaves_pedestrian():
    INV (∧ n_north == 0 ∧ n_south == 0 ∧ n_ped > 0)
    n_ped -= 1 (=> n_ped ≥ 0)
    INV (∧ n_north == 0 ∧ n_south == 0)
    if n_ped == 0:
        pass_S.notify
        pass_N.notify
        wait_P.notify
    INV
```

Tenemos un proceso para cada dirección de los coches:

```
•) car_N():
    LOOP:
        monitor.wants_enter_car_N()
        operaciones_cruzar
        monitor.leaves_car_N()
```

```
•) car_S():
    LOOP:
        monitor.wants_enter_car_S()
        operaciones_cruzar
        monitor.leaves_car_S()
```

Y un proceso para los peatones:

```
•) pedestrian():
    LOOP:
        monitor.wants_enter_pedestrian()
        operaciones_cruzar
        monitor.leaves_pedestrian()
```

2.) Demuestre que el puente es seguro.

Esta parte ya la hemos visto comprobando que el invariante se conserva a lo largo de toda la ejecución.

La idea es que un coche o un peatón solo puede empezar a cruzar si dentro del puente hay únicamente procesos de su mismo "tipo". Este hecho está garantizado por las variables condición pass_N, pass_S y pass_P.

Luego, nunca puede haber simultáneamente coches y peatones ni coches en distinta dirección, por lo que el puente es seguro.

3.) Demuestre la ausencia de Deadlocks.

El deadlock surgiría si, al menos, dos procesos se impidieran mutuamente cruzar el puente, bloqueando el sistema.

Este bloqueo nunca ocurre ya que un proceso siempre podrá seguir ejecutándose, de ocurrir sería al intentar entrar en alguna de las variables condición.

- Si un proceso quiere esperar solo necesita que no haya nadie de su mismo "tipo" cruzando el puente. En caso de que haya alguien de su "tipo" cruzando este terminará de cruzar y cuando se quede vacío el puente se le notifica para que pueda esperar.

- Una vez que ya esté esperando intentará cruzar, para lo que simplemente necesita que no haya nadie de los otros "tipos" cruzando. En caso de que hubiese alguno antes o dos pues terminará de cruzar y se le notificará para que intente cruzar. Podría ocurrir que se "colara"

algún proceso de otro tipo, pero eso sería un problema de inanición, no de deadlock.

Todo proceso, antes o después, espera todos los semáforos que le pudieran bloquear, independientemente de lo que hagan los demás procesos.

3.) Demuestra que no hay inanición.

La inanición se produciría si hubiese un proceso que quisiera cruzar el puente y no pudiese (no necesariamente se tendría por qué bloquear del estado.

- Si un proceso quiere espera y es bloqueado, entonces es porque están cruzando los de su mismo "tipo" y hay alguno de los otros procesos ya esperando. Puede que empiece a cruzar alguno de estos procesos pero lo importante es que, antes o después, los procesos de su "tipo" terminarán de cruzar y le notificarán para que pueda esperar.
- Si un proceso es bloqueado cuando va a entrar es porque alguno de los otros "tipos" está cruzando. Antes o después terminarán de cruzar y notificarán a todos los procesos para que crucen. Hay dos opciones:
  - Empieza a cruzar el proceso y ya no hay inanición.
  - Empieza a cruzar un proceso de otro "tipo" y se vuelve a bloquear. Entonces volvemos a la situación inicial y se vuelve a repetir. Como esto ocurre "infinitas" veces por hipótesis de justicia podemos asumir que en algún momento

cruzará y, por tanto, no hay inanición.

IMPORTANTE: Encuentro demostraciones hemos asumido que si hay unos determinados procesos cruzando el puente, antes o después cruzarán todos y el puente quedará vacío. Pero nos falta justificar que esto siempre es así:

Supongamos que están cruzando procesos de un determinado "tipo". Mientras no aparezcan procesos de los demás "tipos" pueden cruzar todos los procesos de este "tipo" que quieran. Pero esto no es un problema ni para el Deadlock ni la inanición. El problema aparecerá en cuanto aparezca un proceso de un segundo "tipo". Como no hay nadie de este segundo "tipo" esperando se parará a esperar. En cuanto se quede vacío podrá cruzar. Y el puente se va a quedar vacío porque de momento solo pueden cruzar procesos del primer "tipo". Ahora supongamos que aparece otro proceso de este "tipo". Como hay elementos de su mismo "tipo" cruzando y de este "tipo" esperando por lo que no podrá parase a esperar, ni mucho menos cruzar. Luego solo queda la opción de que en algún momento el puente se quede vacío. De ahí el motivo de introducir las variables condición y los contadores para que los procesos esperen.

6.