# 1.

a) Explique la semántica de un semáforo. ¿Que diferencia hay entre los semáforos generales y los binarios¿. ¿Como se representan las operaciones de cada uno con await?
b) Indique los posibles valores finales de *x* en el siguiente programa (**justifique claramente su respuesta**):

```
int x = 4; sem s1 = 1, s2 = 0;
co P(s1); x = x * x ; V(s1);
   // P(s2); P(s1); x = x * 3; V(s1):
   // P(s1); x = x - 2; V(s2); V(s1);
oc
```

## a)

La semántica de un semáforo utiliza dos métodos

- `V` -> Señala la ocurrencia de un evento
- `P` -> Se usa para demorar un proceso hasta que ocurra un evento
  Para declarar un semáforo se hace de la siguiente manera

```
sem NOMBRE = VALOR;
```

### Semáforos generales

Sus operaciones se definen de la siguiente manera

```
P(s) : <await (s>0) s = s-1;>
V(s) : <s = s+1;>
```

En estos semáforos, el contador del semáforos es "ilimitado", no hay una cota superior
Además, el método `V` no es bloqueante, a diferencia de los semáforos binarios donde sí lo es

### Semáforos binarios

Sus operaciones se definen de la siguiente manera

```
P(b) : <await (b>0) b = b-1;>
V(b) : <await (b<1) b = b+1;>
```

En estos semáforos, el contador está limitado a dos valores: `0` y `1`, limitando la cota superior
En este semáforo, el método `V` es bloqueante

## b)

Variables:

```
int x = 4;
sem s1 = 1;
```

```
sem s2 = 0;
```

Procesos:

**P1**

```
P(s1);
x = x * x;
V(s1);
```

**P2**

```
P(s2)
P(s1);
x = x * 3;
V(s1);
```

**P3**

```
P(s1);
x = x - 2;
V(s2);
V(s1);
```

Posibles caminos de ejecución

1. Si entra primero el proceso 1, x toma el valor 16. El proceso 2 no puede entrar porque requiere de que se haga un V sobre s2. Entra entonces el proceso 3, dejando x en 14. Y luego se ejecuta el proceso 2, dejando x en **42**
2. Si entra primero P2, se va a quedar esperando, luego puede ejecutar P3, dejando x en el valor 2, luego podría ejecutarse P2, dejando x en el valor 6 y finalmente se ejecuta P1, dejando x en **36**
3. Si entra primero P2, se va a quedar esperando, luego puede ejecutar P3, dejando x en el valor 2, luego podría ejecutarse P1, dejando x en el valor 4 y finalmente se ejecuta P2, dejando x en **12**

# 2.

Desarrolle utilizando semáforos una solución centralizada al problema de los filósofos, con un administrador único de los tenedores, y posiciones libres para los filósofos (es decir, cada filósofo puede comer en cualquier posición siempre que tenga los dos tenedores correspondientes).

```
sem pedidos          = 0;
sem mutex            = 1;
bool tenedores[1..9] = [(9) true];
cola colaF;
sem mutexCola        = 1;
sem comer[1..5] = [(5) 0];

Process Administrador {
    while(true) {
        P(pedidos);
```

```
        P(mutexCola);
        int idFilosofo = pop(colaF);
        V(mutexCola);

        P(mutex);
        otorgarTenedores(tenedores, idFilosofo) // Función que pone en false los
tenedores ocupados por el filósofo
        V(mutex);

        V(comer[idFilosofo]);
    }
}

Process Filosofo[id = 1 .. 5] {
    while(true){
        P(mutexCola);
        push(colaF, id);
        V(mutexCola);

        V(pedidos);

        P(comer[id]);

        P(mutex);
        devolverTenedores(tenedores, id);
        V(mutex);
    }
}
```

# 3.

Describa la técnica de *Passing the Baton*. ¿Cuál es su utilidad en la resolución de problemas mediante semáforos?

La técnica passing the baton es una técnica general para implementar sentencias await, cuando un proceso está dentro de una sección crítica, mantiene el baton, que significa permiso para ejecutar y cuando sale de la sección crítica, pasa el baton a otro proceso o, si no hay nadie esperando por el baton, lo libera para ser tomado por el próximo.
Su utilidad en cuando a semáforos, es que nos permite respetar un orden y evitar la inanición, dado que permite que todos los procesos puedan ejecutarse en algún momento

# 4.

Modifique las soluciones de Lectores-Escritores con semáforos de modo de no permitir más de 10 lectores simultáneos en la BD y además que no se admita el ingreso a más lectores cuando hay escritores esperando.

```
int numeroLectores = 0;
int numeroEscritores = 0;
int lectoresDormidos = 0;
int escritoresDormidos = 0;
```

```
sem mutex = 1;
sem leer = 0;
sem escribir = 0;

Process Lector[int id = 1 .. L] {
    while(true) {
        P(mutex);
        if( numeroEscritores > 0
            or escritoresDormidos > 0
            or numeroLectores >= 10)
        {
            lectoresDormidos++;
            V(mutex);
            P(leer);
        }
        numeroLectores++;
        if(     lectoresDormidos    > 0
            and numeroLectores      < 10
            and escritoresDormidos == 0)
        {
            lectoresDormidos--;
            V(leer);
        } else {
            V(mutex);
        }

        Lee la BD

        P(mutex);
        numeroLectores--;
        if(numeroLectores == 0 and escritoresDormidos > 0) {
            escritoresDormidos--;
            V(escribir);
        } else if (     lectoresDormidos    > 0
                    and escritoresDormidos == 0
                    and numeroLectores      < 10)
        {
            lectoresDormidos--;
            V(leer);
        } else {
            V(mutex);
        }
    }
}

Process Escritor[int id = 1 .. E] {
    while(true) {
        P(mutex);
        if(numeroLectores > 0 or numeroEscritores > 0) {
            escritoresDormidos++;
            V(mutex);
            P(escribir);
        }
```

```
        numeroEscritores++;
        V(mutex);

        Escribe la BD

        P(mutex);
        numeroEscritores--;
        if(escritoresDormidos > 0) {
            escritoresDormidos--;
            V(escribir);
        } else if (lectoresDormidos > 0 && numeroLectores < 10) {
            lectoresDormidos--;
            V(leer);
        } else {
            V(mutex);
        }
    }
}
```

# 5.

Describa el funcionamiento de los monitores como herramienta de sincronización. Como se realiza la comunicación y la sincronización por condición entre procesos en esta herramienta.

Los monitores son un mecanismo de abstracción de datos, que encapsulan las representaciones de recursos y brindan un conjunto de operaciones que son los únicos medios para manipular esos recursos.
Contiene variables que almacenan el estado del recurso y procedimientos que implementan las operaciones sobre el mismo
La exclusión mutua es implícita, asegurando que los procedimientos en un monitor, no se ejecutan concurrentemente
La sincronización por condición es explícita utilizando variables condición
La comunicación es a través de los procedures y variables permanentes del monitor

# 6.

¿Qué diferencias existen entre las disciplinas de señalización "Signal and wait" y "Signal and continue"?

## Signal and continue

El proceso que hace `signal` continúa usando el monitor y el proceso despertado pasa a competir por acceder nuevamente al monitor para continuar su ejecución en la instrucción que le sigue al `wait` ejecutado por el proceso despertado al momento de dormirse

## Signal and wait

El proceso que hace `signal` pasa a competir por acceder nuevamente al monitor, mientras que el proceso despertado pasa a ejecutar dentro del monitor a partir de la instrucción que le sigue al `wait` ejecutado por el proceso despertado al momento de dormirse

## 7. Corregir

¿En qué consiste la técnica de Passing the Condition y cuál es su utilidad en la resolución de problemas con monitores? ¿Qué relación encuentra entre passing the condition y passing the baton?

Los procesos que no pueden continuar su ejecución porque una condición no se cumple, se bloquean en una variable de condición. La técnica passing the condition introduce una regla estricta para la liberación de la exclusión mutua, de manera tal que el proceso que está saliendo del monitor, cede inmediatamente la exclusión mutua al proceso que acaba de despertar

La principal utilizada de passing the condition es la eliminación de la inanición y la mejora de la eficiencia y predictibilidad del código concurrente. Además de ofrecer un orden en los procesos que ejecutan.

Ambas técnicas, passing the condition y passing the baton, tienen un objetivo comun: garantizar una transferencia justa y ordenada del control entre procesos para evitar la inanición. La diferencia es que uno se implementa con monitores y otro con semáforos

## 8.

Desarrolle utilizando monitores una solución centralizada al problema de los filósofos, con un administrador único de los tenedores, y posiciones libres para los filósofos (es decir, cada filósofo puede comer en cualquier posición siempre que tenga los dos tenedores correspondientes).

Se duplica?

## 9. CORREGIR

Sea la siguiente solución propuesta al problema de alocación SJN:

```
monitor SJN {
    bool libre = true;
    cond turno;

    procedure request(int tiempo) {
        if (not libre) wait(turno, tiempo);
        libre = false;
    }

    procedure release() {
        libre = true
        signal(turno);
    }
}
```

a) Funciona correctamente con disciplina de señalización Signal and Continue?
b) Funciona correctamente con disciplina de señalización Signal and Wait?
**EXPLIQUE CLARAMENTE SUS RESPUESTAS**

Al utilizar una sola variable condición, que es "general" a todos los procesos, con la disciplina "Signal and Continue", no se garantiza que el proceso despertado sea realmente el proceso al que le toque el turno
Funciona correctamente con la disciplina "Signal and Wait", ya que el proceso despertado toma directamente el monitor y pone libre en false, garantizando la ocupación del recurso

# 10.

Modifique la solución anterior para el caso de no contar con una instrucción wait con prioridad.

```
Monitor SJN {
    bool libre = true;
    cond turno[1..N];
    colaOrdenada colaTurnos;

    Procedure request(int tiempo, int id) {
        if(not libre) {
            push(colaTurnos, id, tiempo)
            wait(turno[id]);
        }
        libre = false;
    }

    Procedure release() {
        if(not empty(colaTurnos)) { debería usar contador
            int id = pop(colaTurnos);
            signal(turno[id]);
        } else {
            libre = true;
        }
    }
}
```

# 11.

Modifique utilizando monitores las soluciones de Lectores-Escritores de modo de no permitir más de 10 lectores simultáneos en la BD, y además que no se admita el ingreso a más lectores cuando hay escritores esperando.

```
monitor Controlador_RW {
    int numeroLectores = 0;
    int numeroEscritores = 0;
    int lectoresDormidos = 0;
    int escritoresDormidos = 0;
    cond ok_leer;
    cond ok_escribir;

    Procedure pedido_leer() {
        if ( numeroEscritores > 0
            or escritoresDormidos > 0
            or numeroLectores >= 10)
```

```
        {
            lectoresDormidos++;
            wait(ok_leer);
            // lectoresDormidos--;
        } else {
            numeroLectores++;
        }
    }

    Procedure libera_leer() {
        numeroLectores---;
        if(numeroLectores == 0 and escritoresDormidos > 0) {
            escritoresDormidos-;
            signal(ok_escribir);
            numeroEscritores++;
        } else if (lectoresDormidos > 0
                    and escritoresDormidos == 0
                    and numeroLectores < 10)
        {
            signal(ok_leer);
        }
    }

    Procedure pedido_escribir() {
        if(numeroLectores > 0 or numeroEscritores > 0) {
            escritoresDormidos++;
            wait(ok_escribir);
        } else {
            numeroEscritores++;
        }
    }

    Procedure libera_escribir() {
        if(escritoresDormidos > 0) {
            escritoresDormidos--;
            signal(ok_escribir);
        } else {
            numeroEscritores--;
            if(lectoresDormidos > 0) {
                numeroLectores = lectoresDormidos;
                lectoresDormidos = 0;
                signalall(ok_leer);
            }
        }
    }
}
```

# 12.

Resuelva con monitores el siguiente problema: tres clases de procesos comparten el acceso a una lista enlazada: searchers, inserters y deleters. Los searchers sólo examinan la lista, y por lo tanto pueden ejecutar concurrentemente unos con otros. Los inserters agregan nuevos ítems al final de la lista; las inserciones deben ser mutuamente exclusivas para evitar insertar dos ítems casi al mismo tiempo. Sin embargo, un insert puede hacerse en paralelo con uno o más searches. Por último, los deleters remueven ítems de cualquier lugar de la lista. A lo sumo un deleter puede acceder la lista a la vez, y el borrado también debe ser mutuamente exclusivo con searches e inserciones.

## 13.

El problema del "Puente de una sola vía" (One-Lane Bridge): autos que provienen del Norte y del Sur llegan a un puente con una sola vía. Los autos en la misma dirección pueden atravesar el puente al mismo tiempo, pero no puede haber autos en distintas direcciones sobre el puente.

a) Desarrolle una solución al problema, modelizando los autos como procesos y sincronizando con un monitor (no es necesario que la solución sea fair ni dar preferencia a ningún tipo de auto).
b) Modifique la solución para asegurar fairness (Pista: los autos podrían obtener turnos).

## a)

## b)