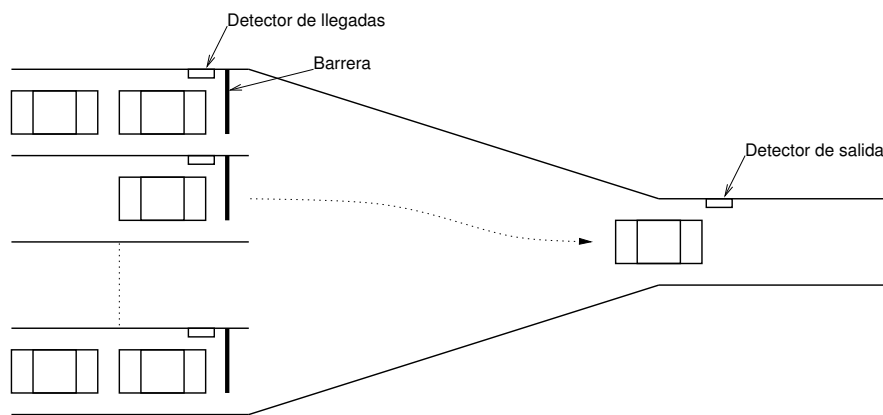


Este examen consta de tres apartados: un planteamiento de un problema con una solución incompleta, un cuestionario y una pregunta de implementación. La duración total es de **tres horas**. El cuestionario debe contestarse **en la misma hoja** en que se os entrega (consultad con un profesor si necesitáis otra hoja) y se podrá devolver hasta **una hora y media** después de recibirlo. Al hacerlo se os dará la pregunta de implementación.

La puntuación total del examen es de **10 puntos**. Las soluciones se proporcionarán antes de la revisión. Las calificaciones se darán a conocer el **14 de febrero**. La revisión del examen tendrá lugar el **16 de febrero**.

1. Zona de estacionamiento

A una zona de estacionamiento para la visita de un importante monumento llegan varios carriles que terminan en barreras que controlan la entrada de los coches. Tras dichas barreras, al aparcamiento se accede por un único carril. La siguiente figura es un esquema del acceso descrito:



El sistema de control de acceso realiza la apertura simultánea de varias barreras y el efecto que provoca es un embotellamiento cuando los coches llegan al final del *embudo*. Para evitar este problema se ha instalado un detector de salida y se va a rediseñar el programa concurrente que controla el acceso de tal forma que:

- Nunca haya más de un coche en el embudo.
- En el caso en el que haya varias barreras que puedan ser abiertas, las aperturas deben realizarse en el orden en el que se detectaron los coches.

Para el control de las barreras y de los detectores se dispone del paquete Barreras:

package Barreras **is**

N_Barreras : **constant** Natural := N;

subtype Barrera **is** Natural **range** 0 .. N_Barreras - 1;

— El proceso que la invoca queda bloqueado hasta que un coche
— llega a la barrera B

procedure Detectar_Llegada (B : **in** Barrera);

— Abre la barrera B, espera el paso de un coche y vuelve a cerrarla.

procedure Abrir (B : **in** Barrera);

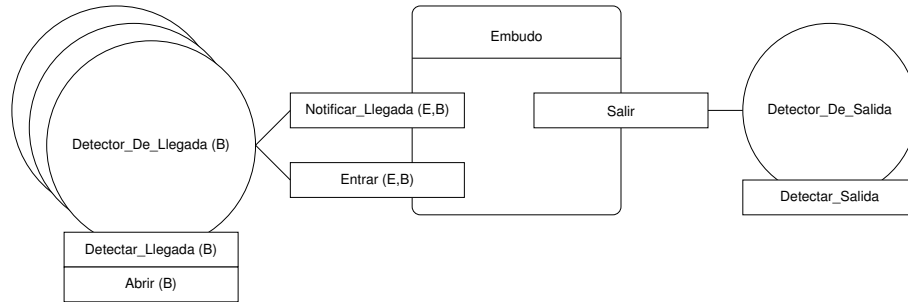
— El proceso que la invoca queda bloqueado hasta que en la salida
— del embudo se detecta el paso de un coche.

procedure Detectar_Salida;

end Barreras;

Diseño

A continuación se presenta un diseño en el que hay un proceso controlando cada barrera y otro controlando la salida del embudo. Dichos procesos se comunican mediante un recurso compartido tal y como indica el siguiente grafo de procesos y recursos:



El código de los procesos es el siguiente:

— *Recurso compartido*
 E : Embudo;

— *Tarea para controlar la salida del embudo*
task type Detector_De_Salida;

task body Detector_De_Salida **is**
begin
 loop
 Detectar_Salida;
 Salir (E);
end loop;
end Detector_De_Salida;

— *Tarea para controlar la barrera B*
task type Detector_De_Llegada
 (B : Barrera);

task body Detector_De_Llegada **is**
begin
 loop
 Detectar_Llegada (B);
 Notificar_Llegada (E, B);
 Entrar (E, B);
 Abrir (B);
end loop;
end Detector_De_Llegada;

La especificación (incompleta) del recurso compartido es la siguiente:

— *CTAD Embudo*
 — *OPERACIONES*
 — ACCION Notificar_Llegada : Embudo[es] × Barrera[e]
 — ACCION Entrar : Embudo[es] × Barrera[e]
 — ACCION Salir : Embudo[es]
 — *SEMÁNTICA*
 — *DOMINIO*
 — TIPO Embudo = (Ocupado : Boolean × Llegadas : Secuencia (Barrera))
 — INV: $\forall e \in \text{Embudo}. \forall i, j \in 1 \dots \text{Longitud}(e.\text{Llegadas}).$
 $e.\text{Llegadas}(i) = e.\text{Llegadas}(j) \Rightarrow i = j$
 — INV (informal): no hay entradas repetidas en la secuencia
 — INICIAL(e): $\neg e.\text{Ocupado} \wedge e.\text{Llegadas} = \langle \rangle$
type Embudo **is private**;

— CPRE: Cierta
 — POST: INICIAL(resultado)
function Crear_Embudo **return** Embudo;

— CPRE: Cierta
 — POST: $E^{sal}.\text{Ocupado} = E^{ent}.\text{Ocupado} \wedge E^{sal}.\text{Llegadas} = E^{ent}.\text{Llegadas} + \langle B \rangle$
procedure Notificar_Llegada (E : **in out** Embudo; B : **in** Barrera);

— *Especificación de Entrar a completar en apartado 2*
procedure Entrar (E : **in out** Embudo; B : **in** Barrera);

— CPRE: Cierta
 — POST: $\neg E^{sal}.\text{Ocupado} \wedge E^{sal}.\text{Llegadas} = E^{ent}.\text{Llegadas}$
procedure Salir (E : **in out** Embudo);

Apellidos: _____

Nombre: _____ Número de matrícula: _____

2. Cuestionario

[5 puntos]

Importante: sólo hay una respuesta válida por pregunta excepto para la pregunta 2.1. Para el resto, toda pregunta en que se marque más de una respuesta se considerará incorrectamente contestada. Toda pregunta incorrectamente contestada restará del examen una cantidad de puntos igual a la puntuación de la pregunta dividido por el número de alternativas ofrecidas.

2.1. Zona de estacionamiento

[1 punto]

Especificar la operación Entrar del recurso compartido Embudo de forma que se complete el diseño exigido en el enunciado *Zona de estacionamiento* del apartado 1.

CPRE:	
CPRE informal:	
POST:	
POST informal:	

2.2. Más canales

[1 punto]

Las implementaciones basadas en objetos protegidos necesitan, en general, almacenar las peticiones que dependen de parámetros de entrada en una tabla cuya clave debe poder servir de índice de un *array*. Las basadas en paso de mensajes ofrecen mayor libertad: las peticiones pueden almacenarse en cualquier estructura de datos y los clientes se desbloquean cuando reciben un mensaje del servidor. ¿Por qué no unir ambas ideas y realizar una implementación basada en objetos protegidos, pero cuyo desbloqueo se realiza con un canal explícito? La idea se muestra en el siguiente fragmento de código, donde suponemos, como se vino haciendo durante el curso, que tanto **Send** como **Receive** trabajan sobre canales síncronos:

Recurso con objetos protegidos

— CPRE: $P(\text{Estado}, \text{Arg1})$
 — $E(\text{Estado}, \text{Arg1}, \text{Arg2})$
 — POST: $\text{Arg2}^{\text{sal}} = Q(\text{Estado}, \text{Arg1}) \wedge \dots$

— Implementación:

entry E (Arg1: **in** T1; Can: **in out** Channel)

when True **is begin**

 — *T es una estructura de datos cualquiera que nos venga bien*

 T := Almacenar (T, (Arg1, Can));

enqueue E1;

end E;

entry E1 (....)

— *PodemosServir* recorre T y determina si hay alguna petición que podamos servir

when PodemosServir (T) **is begin**

 (Arg1, Canal) := Recuperar (T);

— *Alguna petición que podamos atender*

Send (Canal, Q (Estado, Arg1));

end E1;

Código del cliente

loop — Cliente

 ObProt.E (MiArg1, Canal);

Receive (Canal, MiArg2);

end loop;

Se pide: marcar cuál de las siguientes afirmaciones sobre el esquema anterior es verdadera:

- | | |
|--|---|
| <input type="checkbox"/> Es perfectamente válido. | <input type="checkbox"/> No compila porque en el estado privado del objeto protegido sólo se pueden usar vectores, y no “una estructura de datos cualquiera”. |
| <input type="checkbox"/> Se bloquea siempre. | |
| <input type="checkbox"/> Podría bloquearse en algunos casos. | |

2.3. ¿Exclusión mutua?

[1 punto]

El código que aparece inmediatamente debajo está destinado a asegurar la exclusión mutua de dos procesos, P1 y P2, cuando intenten acceder a una sección crítica en el punto marcado con un comentario. Las variables c1 y c2 son compartidas por los dos procesos. Asumir que la sentencia en cada línea del código se ejecuta de modo atómico.

```
c1, c2: Integer := 1;
```

```
loop — Proceso P1
  loop
    c1 := 1 - c2;
    exit when c2 /= 0;
  end loop;
  ... — Sección crítica
  c1 := 1;
end loop;
```

```
loop — Proceso P2
  loop
    c2 := 1 - c1;
    exit when c1 /= 0;
  end loop;
  ... — Sección crítica
  c2 := 1;
end loop;
```

Se pide contestar, marcando una de las opciones mostradas más abajo, a la siguiente pregunta: el código anterior, ¿realmente implementa exclusión mutua sobre la región marcada? Nótese que no estamos interesados en posibles problemas de vivacidad, etc., sino **únicamente** en saber si se garantiza o no la exclusión mutua de ambos procesos.

☐ **Sí** implementa exclusión mutua.

☐ **No** implementa exclusión mutua.

2.4. Semaforo ternario

[1 punto]

Queremos establecer una sección con exclusión parcial con la siguiente propiedad: como mucho dos procesos deben poder acceder simultáneamente a la sección protegida. Podría hacerse con un semáforo general inicializado a 2, pero sólo disponemos de semáforos binarios, así que se propone el siguiente código, a ser utilizado por todos los procesos que desean acceder a la sección crítica:

Variables compartidas e inicialización de las mismas

```
BinMutex1: Bin_Semaphore;
BinMutex2: Bin_Semaphore;
...
Init (BinMutex1, 1);
Init (BinMutex2, 1);
```

Código a utilizar por los procesos que desean entrar en la sección protegida

```
— Antes de sección protegida
Wait (BinMutex1 + BinMutex2);
— Dentro de sección protegida
Signal (BinMutex1 + BinMutex2);
— Después de sección protegida
```

La idea es la siguiente: al hacer un **Wait** sobre BinMutex1 + BinMutex2 se forzará a que la suma de ambos disminuya (si no es ya cero), de manera que uno de los semáforos se decrementará. Lo contrario sucederá con el **Signal**. Elegir una de las siguientes aseveraciones:

☐ Sólo va a dejar entrar a un proceso en cada momento, porque se decrementarán ambos semáforos binarios a la vez.

☐ No funcionará correctamente porque **Signal** no sabe qué variable decrementó **Wait**, así que puede intentar incrementar un semáforo binario que ya tiene el valor 1.

☐ No hace lo que queremos porque, al utilizar dos semáforos binarios, la expresión BinMutex1 + BinMutex2 va a tener cuatro estados en lugar de los tres necesarios.

☐ Ninguna de las anteriores respuestas es válida porque el código mostrado no tiene ningún sentido.

2.5. Encapsular o no encapsular

[1 punto]

Suponiendo que estamos usando objetos protegidos para implementar recursos compartidos, señalar cuál de las siguientes afirmaciones es correcta:

☐ El recurso se debe encapsular siempre dentro de un O.P.

☐ Si el recurso se encapsula dentro de un O.P., sólo lo puede usar un proceso a la vez.

☐ Aunque se encapsule el recurso dentro de un O.P., varios procesos pueden usarlo a la vez.

☐ Siempre debe definirse fuera del O.P. y usar el O.P. para gestionar los permisos de acceso.

Departamento de Lenguajes, Sistemas Informáticos e Ingeniería del Software

Este examen consta de tres apartados: un planteamiento de un problema con una solución incompleta, un cuestionario y una pregunta de implementación. La duración total es de **tres horas**. El cuestionario debe contestarse **en la misma hoja** en que se os entrega (consultad con un profesor si necesitáis otra hoja) y se podrá devolver hasta **una hora y media** después de recibirlo. Al hacerlo se os dará la pregunta de implementación.

La puntuación total del examen es de **10 puntos**. Las soluciones se proporcionarán antes de la revisión. Las calificaciones se darán a conocer el **14 de febrero**. La revisión del examen tendrá lugar el **16 de febrero**.

3. Implementación

[5 puntos]

La especificación completa del recurso compartido Embudo sería:

with Barreras; **use** Barreras;

package Embudos **is**

```
— CTAD Embudo
— OPERACIONES
— ACCION Notificar_Llegada : Embudo[es] × Barrera[e]
— ACCION Entrar : Embudo[es] × Barrera[e]
— ACCION Salir : Embudo[es]
— SEMÁNTICA
— DOMINIO
— TIPO Embudo = (Ocupado : Boolean × Llegadas : Secuencia (Barrera))
— INV:  $\forall e \in \text{Embudo}. \forall i, j \in 1 \dots \text{Longitud}(e.\text{Llegadas}).$ 
—  $e.\text{Llegadas}(i) = e.\text{Llegadas}(j) \Rightarrow i = j$ 
— INV (informal): no hay entradas repetidas en la secuencia
— INICIAL(e):  $\neg e.\text{Ocupado} \wedge e.\text{Llegadas} = \langle \rangle$ 
type Embudo is private;

— CPRE: Cierto
— POST: INICIAL(resultado)
function Crear_Embudo return Embudo;

— CPRE: Cierto
— POST:  $E^{sal}.\text{Ocupado} = E^{ent}.\text{Ocupado} \wedge E^{sal}.\text{Llegadas} = E^{ent}.\text{Llegadas} + \langle B \rangle$ 
procedure Notificar_Llegada (E : in out Embudo; B : in Barrera);

— CPRE:  $\neg E.\text{Ocupado} \wedge E.\text{Llegadas}(1) = B$ 
— POST:  $E^{sal}.\text{Ocupado} \wedge E^{sal}.\text{Llegadas} = E^{ent}.\text{Llegadas}(2 \dots \text{Longitud}(E^{ent}.\text{Llegadas}))$ 
procedure Entrar (E : in out Embudo; B : in Barrera);

— CPRE: Cierto
— POST:  $\neg E^{sal}.\text{Ocupado} \wedge E^{sal}.\text{Llegadas} = E^{ent}.\text{Llegadas}$ 
procedure Salir (E : in out Embudo);

private
type Embudo_Impl;
type Embudo is access Embudo_Impl;
end Embudos;
```

Para cada uno de los apartados siguientes ha de entregarse lo que se pide en ellos. Recordad **entregar la respuesta a cada apartado en un juego de hojas separadas**.

3.1. Objetos protegidos

[2.5 puntos]

Completar toda la implementación del recurso compartido Embudo utilizando objetos protegidos como mecanismo de concurrencia, aplicando la metodología propia de la asignatura y siguiendo el esquema que se ofrece en el apartado 3.3.

3.2. Rendez-Vous / Paso de mensajes

[2.5 puntos]

Completar toda la implementación del recurso compartido Embudo utilizando *rendez-vous* y paso de mensajes como mecanismos de concurrencia, aplicando la metodología propia de la asignatura y siguiendo el esquema que se ofrece en el apartado 3.3.

3.3. Esquema para completar la implementación

— *Importación de todos los paquetes necesarios*
 — **A completar**

package body Embudos **is**

— *Definición de tipos auxiliares e instanciación de paquetes genéricos*
 — *(si fuera necesario)*
 — **A completar**

— *Declaración del tipo como objeto protegido o como servidor*
protected ó **task type** Embudo_Impl **is**
 — **A completar**
end Embudo_Impl;

— *Implementación del tipo como objeto protegido o como servidor*
protected ó **task body** Embudo_Impl **is**
 — **A completar**
end Embudo_Impl;

function Crear_Embudo **return** Embudo **is**
 — **A completar**
begin
 — **A completar**
end Crear_Embudo;

procedure Notificar_Llegada (E : **in out** Embudo; B : **in** Barrera) **is**
 — **A completar**
begin
 — **A completar**
end Notificar_Llegada;

procedure Entrar (E : **in out** Embudo; B : **in** Barrera) **is**
 — **A completar**
begin
 — **A completar**
end Entrar;

procedure Salir **is**
 — **A completar**
begin
 — **A completar**
end Salir;

end Embudos;