ADA - **CONSIDERACIONES PARA RESOLVER LOS EJERCICIOS**

1. NO SE PUEDE USAR VARIABLES COMPARTIDAS
2. Declaración de tareas

* Especificación de tareas sin ENTRY’s (nadie le puede hacer llamados).  
  TASK Nombre;  
  TASK TYPE Nombre;
* Especificación de tareas con ENTRY’s (le puede hacer llamados). Los entry’s  
  funcionan de manera semejante los procedimientos: solo pueden recibir o  
  enviar información por medio de los parámetros del entry. NO RETORNAN  
  VALORES COMO LAS FUNCIONES  
  TASK [TYPE] Nombre IS  
  ENTRY e1;  
  ENTRY e2 (p1: IN integer; p2: OUT char; p3: IN OUT float);  
  END Nombre;
* Cuerpo de las tareas.  
  TASK BODY Nombre IS  
  Codigo que realiza la Tarea;  
  END Nombre;

1. Sincronización y comunicación entre tareas

* **Entry call** para enviar información (o avisar algún evento).  
  NombreTarea.NombreEntry (parametros);
* **Accept** para atender un pedido de entry call sin cuerpo (sólo para recibir el  
  aviso de un evento para sincronización).  
  ACCEPT NombreEntry (p1: IN integer; p3: IN OUT float);
* **Accept** para atender un pedido de entry call con cuerpo.  
  ACCEPT NombreEntry (p1: IN integer; p3: IN OUT float) do  
   Cuerpo del accept donde se puede acceder a p1 y p3.  
   Fuera del entry estos parámetros no se pueden usar.  
  END NombreEntry;
* El accept se puede hacer en el cuerpo de la tarea que ha declarado el entry en su especificación. Los entry call se pueden hacer en cualquier tarea o en el programa principal.
* Tanto el entry call como el accept son bloqueantes, ambas tareas continúan  
  trabajando cuando el cuerpo del accept ha terminado su ejecución.

1. Select para ENTRY CALL.

* **Select ...OR DELAY**: espera a lo sumo x tiempo a que la tarea correspondiente haga el accept del entry call realizado. Si pasó el tiempo entonces realiza el código opcional.  
  SELECT  
  NombreTarea.NombreEntry(Parametros);  
  Sentencias;  
  OR DELAY x  
  Código opcional;  
  END SELECT;
* **Select ...ELSE**: si la tarea correspondiente no puede realizar el accept inmediatamente (en el momento que el procesador está ejecutando esa línea de código) entonces se ejecuta el código opcional.  
  SELECT  
  NombreTarea.NombreEntry(Parametros);  
  Sentencias;  
  ELSE  
  Código opcional;  
  END SELECT;
* En los select para entry call sólo puede ponerse un entry call y una única opción (OR DELAY o ELSE);

1. Select para ACCEPT.

* En los select para los accept puede haber más de una alternatica de accept, pero no se puede haber alternativas de entry call (no se puede mezclar accept con entries). Cada alternativas de ACCEPT puede ser o no condicional (WHEN).  
  SELECT  
  ACCEPT e1 (parámetros);  
  Sentencias1;  
    
  OR ACCEPT e2 (parámetros) IS cuerpo; END e2;  
    
  OR WHEN (condición) => ACCEPT e3 (parámetros) IS cuerpo; END e3;  
   Sentencias3  
  END SELECT;

**Funcionamiento**: Se evalúa la condición booleana del WHEN de cada alternativa (si  
no lo tiene se considera TRUE). Si todas son FALSAS se sale del select. Sino, de las  
alternativas cuyo condición es verdadera se elige en forma no determinística una que  
pueda ejecutarse inmediatamente (es decir que tiene un entry call pendiente). Si  
ninguna de ellas se puede ejecutar inmediatamente el select se bloquea hasta que haya  
un entry call para alguna alternativa cuya condición sea TRUE.

* Se puede poner una opción OR DELAY o ELSE.
* Dentro de la condición booleana de una alternativa (en el WHEN) se puede preguntar por la cantidad de entry call pendientes de cualquier entry de la tarea.  
  NombreEntry’count
* Después de escribir una condición por medio de un WHEN siempre se debe escribir un accept.

### 1.**Se requiere modelar un puente de un solo sentido, el puente solo soporta el peso de 5 unidades de peso. Cada auto pesa 1 unidad, cada camioneta pesa 2 unidades y cada camión 3 unidades. Suponga que hay una cantidad innumerable de vehículos (A autos, B camionetas y C camiones).**

#### **a. Realice la solución suponiendo que todos los vehículos tienen la misma prioridad. CORREGIDO**

**Procedure Ejercicio1()**

##### **TASK TYPE** Auto

end Auto;

**TASK BODY** Auto **IS**

Puente.entrarAuto();

delay(random);

Puente.salirAuto();

end Auto;

##### **TASK TYPE** Camion

end Camion;

**TASK BODY** Camión **IS**

Puente.entrarCamion();

delay(random);

Puente.salirCamion();

end Camion;

##### 

##### **TASK TYPE** Camioneta

end Camioneta;

**TASK BODY** Camioneta **IS**

Puente.entrarCamioneta();

delay(random);

Puente.salirCamioneta();

end Camioneta;

##### TASK Puente IS // Se pone asi porque es un SOLO PUENTE, si fueran muchos, se pone TASK TYPE

entry entrarAuto;

entry salirAuto;

entry entrarCamion;

entry salirCamion;

entry entrarCamioneta;;

entry salirCamioneta;;

end Puente;

**TASK BODY** Puente **IS**

cantAuto, cantCamioneta, cantCamion, peso:= 0;

**loop**

**select**

//COMO MUCHO PUEDEN ENTRAR 5 AUTOS

when **peso + 1** <= 5

=> accept entrarAuto

peso ++;

end entrarAuto;

//COMO MUCHO PUEDEN ENTRAR 2 CAMIONETAS

**or**

when **peso + 2** <= 5

=> accept entrarCamioneta

peso:= peso + 2;

end entrarCamioneta;

//COMO MUCHO PUEDE ENTRAR 1 CAMION

**or**

when **peso + 3** <= 5

=> accept entrarCamion

peso + 3;

end entrarCamion;

**or**

=> accept salirAuto

peso := peso - 1;

end salirAuto;

**or**

=> accept salirCamioneta

peso:= peso - 2;

end salirCamioneta;

**or**

=> accept salirCamion

peso:= peso - 3;

end salirCamion;

end select;

end loop

end Puente;

**VAR**

array of auto [1..A] autos;

array of camion [1..B] camiones;

array of camionetas [1..C] camionetas;

**BEGIN**

**END.**

### 

### 

### 

### 

### 

### b. Modifique la solución para que tengan mayor prioridad los camiones que el resto de los vehículos. CORREGIDO

##### 

##### **TASK BODY** Puente **IS**

int cantAuto, cantCamioneta, cantCamion, peso:= 0;

**loop**

**select**

//COMO MUCHO PUEDEN ENTRAR 5 AUTOS

when peso + 1 <= 5 & **entrarCamion’count() = 0**

=> accept entrarAuto

peso:= peso + 1;

end entrarAuto;

//COMO MUCHO PUEDEN ENTRAR 2 CAMIONETAS

**or**

when peso + 2 <= 5 & **entrarCamion’count() = 0**

=> accept entrarCamioneta

peso:= peso + 2;

end entrarCamioneta;

//COMO MUCHO PUEDE ENTRAR 1 CAMION

**or**

when peso + 3 <= 5

=> accept entrarCamion

peso:= peso + 3;

end entrarCamion;

**or**

=> accept salirAuto

peso := peso - 1;

end salirAuto;

**or**

=> accept salirCamioneta

peso:= peso - 2;

end salirCamioneta;

**or**

=> accept salirCamion

peso:= peso - 3;

end salirCamion;

end select

end loop;

end Puente;

**VAR**

array of auto [1..A] autos;

array of camion [1..B] camiones;

array of camionetas [1..C] camionetas;

**BEGIN**

**END**.

### 2.**Se quiere modelar la cola de un banco que atiende un solo empleado, los clientes llegan y si esperan más de 10 minutos se retiran. CORREGIDO**

**Procedure Ejercicio2()**

##### TASK Empleado IS

entry llegada

end Empleado;

**TASK BODY Empleado IS**

**loop**

accept llegada();

//atiende a la persona

end llegada;

end loop;

end Empleado;

##### 

##### **TASK TYPE Persona IS**

end Persona;

**TASK BODY Persona IS**

select

Empleado.llegada();

or delay 10

//se retira

end select;

end Persona;

**VAR**

array of persona [1..N] personas;

**BEGIN**

**END.**

### 3.**Se debe modelar un buscador para contar la cantidad de veces que aparece un número dentro de un vector distribuido entre las N tareas contador. Además existe un administrador que decide el número que se desea buscar y se lo envía a los N contadores para que lo busquen en la parte del vector que poseen. CORREGIDO**

**Procedure Ejercicio3() {**

##### TASK TYPE Contador IS

end Contador;

**TASK BODY Contador IS**

array [1..N] arreglo; int sum = 0;

Administrador.darNumero(num);

for i = 1 to N

if (arreglo[i] = num)

sum:= sum + 1;

Administrador.conteoRealizado(sum);

end Contador;

##### TASK TYPE Administrador IS

entry darNumero (out num);

entry conteoRealizado(in suma);

end Administrador;

**TASK BODY Administrador IS**

int total, cant = 0;

while (cant < N) {

select

accept darNumero(out num);

num:= random(1..N);

end darNumero;

or when darNumero’count = 0

accept conteoRealizado(in suma)

total:= total + suma;

cant ++;

end conteoRealizado;

end select;

}

end Administrador;

**VAR**

array of contador [1..C] contadores;

**BEGIN**

**END**;

### 4.**Se debe controlar el acceso a una base de datos. Existen A procesos de Tipo 1, B procesos de Tipo 2 y C procesos de Tipo 3 que trabajan indefinidamente de la siguiente manera:**

### **Proceso Tipo 1: intenta escribir, si no lo logro en 2 minutos, espera 5 minutos y vuelve a intentarlo.**

### **Proceso Tipo 2: intenta escribir, si no lo logra en 5 minutos, intenta leer, si no lo logra en 5 minutos vuelve a comenzar.**

### **Proceso Tipo 3: intenta leer, si no puede inmediatamente entonces espera hasta poder escribir.**

### **Un proceso que quiera escribir podrá acceder si no hay ningún otro proceso en la base de datos, al acceder escribe y avisa que termino de escribir.**

### **Un proceso que quiera leer podrá acceder si no hay procesos que escriban, al acceder lee y avisa que termino de leer.**

### **Siempre se le debe dar prioridad al pedido de acceso para escribir sobre el pedido de acceso para leer. CORREGIDO**

**Procedure Ejercicio4(){**

##### TASK TYPE Proceso1 IS

end Proceso1;

**TASK BODY Proceso1 IS**

**loop**

**select**

Administrador.escribir();

delay(random); //ESCRIBE

Administrador.liberarEscribir();

**or delay 2**

delay 5

end select

end loop

end Proceso1;

##### **TASK TYPE Proceso2 IS**

end Proceso2;

**TASK BODY Proceso2 IS**

**loop**

**select**

Administrador.escribir();

delay(random);

Administrador.liberarEscribir();

**or delay 5**

**select**

Administrador.leer();

delay(random);

Administrador.liberarLeer();

**or delay 5**

end select;

end select;

end loop;

end Proceso2;

##### **TASK TYPE Proceso3 IS**

end Proceso3;

**TASK BODY Proceso3 IS**

**loop**

**select**

Administrador.leer();

delay(random); //lee

Administrador.liberarLeer();

**else**

Administrador.escribir();

delay(random); //escribe

Administrador.liberarEscribir();

end select

end loop

end Proceso3;

##### **TASK Administrador IS**

entry leer;

entry escribir;

entry liberarLeer;

entry liberarEscribir;

end Administrador;

**TASK BODY Administrador IS**

int cantLectores, cantEscritores = 0;

**loop**

**select**

when (cantEscritores = 0 & cantLectores = 0)

accept **escribir**() do

cantEscritores ++;

end escribir;

or (when escribir’count = 0 & cantEscritores = 0)

accept **leer**() do

cantLectores ++;

end leer;

or accept liberarEscribir();

cantEscritores --;

or accept liberarLeer():

cantLectores --;

end select;

end loop;

end Administrador;

//////**/ otra opción //////**

select

when (cantLectores = 0)

accept escribir();

accept liberarEscribir();

or (when escribir’count = 0)

accept leer() do

cantLectores ++;

end leer;

or accept liberarLeer():

cantLectores --;

end liberarLeer();

end select

**/////////////////////////////**

**VAR**

array of proceso1 [1..A] procesos1;

array of proceso2 [1..B] procesos2;

array of proceso3 [1..C] procesos3;

**BEGIN**

**END.**

### **En una clínica existe un médico de guardia que recibe continuamente peticiones de atención de las E enfermeras que trabajan en su piso y de las P personas que llegan a la clínica ser atendidos. Cuando una persona necesita que la atiendan *espera a lo sumo 5 minutos a que el médico lo haga, si pasado ese tiempo no lo hace, espera 10 minutos y vuelve a requerir la atención del médico*. Si no es atendida *tres veces, se enoja y se retira de la clínica*. Cuando una enfermera requiere la atención del médico, si este no lo atiende inmediatamente le hace una nota y se la deja en el consultorio para que esta resuelva su pedido en el momento que pueda (el pedido puede ser que el médico le firme algún papel). Cuando la petición ha sido recibida por el médico o la nota ha sido dejada en el escritorio, continúa trabajando y haciendo más peticiones. El médico atiende los pedidos dándoles prioridad a los pacientes que llegan para ser atendidos. Cuando atiende un pedido, recibe la solicitud y la procesa durante un cierto tiempo. Cuando está libre aprovecha a procesar las notas dejadas por las enfermeras. CORREGIDO**

**Procedure Ejercicio5(){**

##### TASK Medico IS

entry atender;

entry atencionEnfermera;

end Medico;

**TASK BODY Medico IS**

**loop**

**select**

accept atender();

delay(random); // atiende al paciente

end atender;

**or** (when atender’count = 0)

accept atencionEnfermera();

delay(random); //atiende a la enfermera

end atencionEnfermera;

**ELSE**

**select**

**Consultorio.tomarNota(nota);**

**else**

**null // en el caso de que no haya notas**

end loop;

end Medico;

##### TASK TYPE Enfermera IS

end Enfermera;

**TASK BODY Enfermera IS**

**loop**

**select**

Medico.atencionEnfermera();

**else**

nota = hacerNota();

Consultorio.dejarNota(nota);

end select;

end loop;

end Enfermera;

##### **TASK Consultorio IS**

entry dejarNota(in string nota);

entry tomarNota(out string nota);

end consultorio;

**TASK BODY Consultorio IS**

**loop**

**select**

=> accept dejarNota(in string nota)

cola.push(nota);

end accept dejarNota;

**or**

when (not empty(cola))

=> accept tomarNota(nota)

nota= cola.pop();

end select;

end loop;

end Consultorio;

##### **TASK TYPE Persona IS**

end Persona;

**TASK BODY Persona IS**

int cantIntentos = 0;

while (cantIntentos < 3) {

**select**

Medico.atender(); // me estan atendiendo

**or delay 5**

delay 10;

cantIntentos ++;

}

end Persona;

**VAR**

array of persona [1..A] personas;

array of enfermera [1..A] enfermeras;

**BEGIN**

**END.**

### **Hay un sistema web para reservas de pasajes de micro donde existen N Clientes que solicitan un pasaje para un cierto destino; espera hasta que el servidor le indique el número de asiento reservado (-1 si no hubiese disponibles); y luego (si había asiento disponible) el cliente imprime su pasaje. El servidor atiende los pedidos de acuerdo al orden de llegada, cuando un cliente le solicita un pasaje a un cierto destino, busca un asiento disponible para ese destino y luego le indica a ese cliente el asiento reservado (o -1 si no hubiese ninguno disponible).**

**Procedure Ejercicio6()**

##### TASK TYPE Cliente IS

endCliente;

**TASK BODY Cliente IS**

**select**

Servidor.solicitarPasaje(numero);

if (numero != -1) {

//imprimir pasaje

}

end select;

endCliente;

##### TASK Servidor IS

entry solicitarPasaje(out número);

end Servidor;

**TASK BODY Servidor IS**

**loop**

accept solicitarPasaje(número);

número = buscarPasaje();

end solicitarPasaje;

end loop;

end Servidor;

**VAR**

array of cliente [1..N] clientes;

**BEGIN**

**END;**

### **Hay una empresa de servicios que tiene un Recepcionista, N Clientes que hacen reclamos y E Empleados que los resuelven. Cada vez que un cliente debe hacer un reclamo, espera a que el Recepcionista lo atienda y le dé un Número de Reclamo, y luego continúa trabajando. Los empleados cuando están libres le piden un reclamo para resolver al Recepcionista y lo resuelven. El Recepcionista recibe los reclamos de los clientes y les entrega el Número de Reclamo, o bien atiende los pedidos de más trabajo de los Empleados (cuando hay reclamos sin resolver le entrega uno al empleado para que trabaje); siempre le debe dar prioridad a los pedidos de los Empleados. Nota: los clientes, empleados y recepcionista trabajan infinitamente. CORREGIDO**

**Procedure Ejercicio7()**

##### **TASK Recepcionista IS**

entry pedirNumero(out num, in reclamo); //cliente

entry pedirReclamo(out reclamo); //empleado

**end Recepcionista;**

**TASK BODY Recepcionista IS**

queue reclamos;

**loop**

**select**

when(pedirReclamo’count = 0)

=> accept pedirNumero(num, reclamo); // el cliente deja un reclamo y el recepcionista le da un número de seguimiento.

num = darNumero();

cola.push(reclamo);

end pedirNumero;

**or** accept pedirReclamo(reclamo) //EMPLEADO

if (not empty(cola))

reclamo = cola.pop();

else // si no dejaron ningun reclamo el recepcionista le envía -1 al empleado indicando que no hay reclamos.

reclamo = -1;

end pedirReclamo;

end select;

end loop;

end Recepcionista;

##### TASK TYPE Cliente IS

end Cliente;

**TASK BODY Cliente IS**

**loop**

Recepcionista.pedirNumero(num, reclamo);

//aca bloqueo por el accept

end loop;

end Cliente;

##### TASK TYPE Empleado IS

end Empleado;

**TASK BODY Empleado IS**

**loop**

**select**

Recepcionista.pedirReclamo(reclamo);

if ( reclamo != -1) { //si la cola de reclamos estaba vacia, el recepcionista me envió -1

// atiendo el reclamo

else

// trabajando

end select;

end loop;

end Empleado;

**VAR**

array of cliente[1..N] clientes;

array of empleado[1..E] empleados;

**BEGIN END.**

### **En una empresa hay 5 controladores de temperatura y una Central. Cada controlador toma la temperatura del ambiente cada 10 minutos y se la envía a una central para que analice el dato y le indique que hacer. Cuando la central recibe una temperatura que es mayor de 40 grados, detiene a ese controlador durante 1 hora. CORREGIDO**

##### TASK TYPE Controlador IS

end Controlador;

**TASK BODY Controlador IS**

int temp, boolean detener;

**loop**

**select**

delay (10); **//ES LO MISMO PONERLO ACA O ABAJO DEL IF**

temp = calcularTemperatura();

Central.enviarTemperatura(temp,detener);

if (detener) {

delay(1); //1 hora

}

end select;

end loop;

end Controlador;

##### TASK Central IS

entry enviarTemperatura(temp in, detener out);

end central;

**TASK BODY Central IS**

**loop**

**select**

=> accept enviarTemperatura(temp, detener);

if (temp > 40) {

detener = true;

} else {

detener = false;

}

end enviarTemperatura;

end select;

end loop;

end Central;

**VAR**

array of controlador [1..C] controladores;

**BEGIN**

**END.**

### **Una empresa de limpieza se encarga de recolectar residuos en una ciudad. Hay P personas que continuamente hacen reclamos para que pasen por su casa. La empresa tiene 2 camiones, que cuando alguno de ellos está libre lo envía a recolectar los residuos de la persona que más reclamos ha hecho.**

### 

##### **TASK TYPE Persona IS**

end Persona;

**TASK BODY Persona IS**

boolean atendido = false;

**while not atendido {**

Buffer.dejarReclamo(id, atendido);

delay(random);

}

end Persona;

##### TASK TYPE Buffer IS

entry dejarReclamo(id in, out atendido);

entry pedirReclamo(id out); //id persona máxima

end Buffer;

**TASK BODY Buffer IS**

array of int [1..P] personas;

array of boolean [1..P] atendidos = [false];

**loop**

**select**

=> accept dejarReclamo(in int id, out bool atendido); //cliente atendido = atendidos[id];

if(!atendido){

personas[id] ++;

}

end dejarReclamo;

**or**

=> accept pedirReclamo(out idMax); //camión

idMax = obtenerIndiceMaximo(personas);

if(idMax <> -1){

atendidos[idMax] = true;

personas[idMax] = -1;

}

end pedirReclamo;

end select;

end loop;

end Buffer;

##### 

##### 

##### 

##### 

##### 

##### 

##### TASK TYPE Camión IS

end Camión;

**TASK BODY Camión IS**

**loop**

Buffer.pedirReclamo(id);

if (id != -1) {

// atiendo a ese cliente

} else {

// hago otra cosa

}

end loop;

end Camión;

**VAR**

array of persona [1..P] personas;

array of camión [1..2] camiones;

**BEGIN**

**END.**