Práctica 3

1. Se dispone de un puente por el cual puede pasar un solo auto a la vez. Un auto pide permiso
   para pasar por el puente, cruza por el mismo y luego sigue su camino.

```
Monitor  Puente
   cond cola;
   int cant= 0;

   Procedure entrarPuente ()
       while ( cant > 0) wait (cola);
       cant = cant + 1;
   end;

   Procedure salirPuente ()
      cant = cant – 1;
      signal(cola);
   end;
End Monitor;

Process Auto [a:1..M]
  Puente. entrarPuente (a);
  "el auto cruza el puente"
  Puente. salirPuente(a);
End Process;
```

---

a. ¿El código funciona correctamente?
Justifique su respuesta.

Si

b. ¿Se podría simplificar el programa? ¿Sin monitor? ¿Menos procedimientos? ¿Sin
variable condition? En caso afirmativo, rescriba el código.

Si, se puede simplificar donde el monitor sea solo para pasar por el puente sin encolarse a nada

c. ¿La solución original respeta el orden de llegada de los vehículos? Si rescribió el código
en el punto b), ¿esa solución respeta el orden de llegada?

No, no respeta el orden de llegada.

2. Existen N procesos que deben leer información de una base de datos, la cual es administrada
   por un motor que admite una cantidad limitada de consultas simultáneas.
   a) Analice el problema y defina qué procesos, recursos y monitores/sincronizaciones
   serán necesarios/convenientes para resolverlo.
   b) Implemente el acceso a la base por parte de los procesos, sabiendo que el motor de
   base de datos puede atender a lo sumo 5 consultas de lectura simultáneas.

```
Monitor MotorBD {
    cond vc;
    int cant = 0;
    int dormido=0

    Procedure acceder(){
        if (cant == 5){
             dormido+=1;
             wait(vc);
            }
        cant+=1
    }

    Procedure salir(){
        if (dormido>=1)
            signal(vc)
        else
            cant-=1
    }
```

```
}

process procesos[id: 0.. N-1]{
    MotorBD.acceder()
    //leer BD
    MotorBD.salir()
}
```

```
Monitor MotorBD {
    cond vc;
    int cant = 0;
    Procedure acceder(){
        while (cant == 5)
             wait(vc);
        cant+=1
    }

    Procedure salir(){
        cant-=1
        signal(vc)i


}

process procesos[id: 0.. N-1]{
    MotorBD.acceder()
    //leer BD
    MotorBD.salir()
}
```

3. Existen N personas que deben fotocopiar un documento. La fotocopiadora sólo puede ser
   usada por una persona a la vez. Analice el problema y defina qué procesos,

recursos y
monitores serán necesarios/convenientes, además de las posibles sincronizaciones requeridas
para resolver el problema. Luego, resuelva considerando las siguientes situaciones:
a) Implemente una solución suponiendo no importa el orden de uso. Existe una función
Fotocopiar() que simula el uso de la fotocopiadora.

```
process Persona [id: 0..N-1]{
    fotocopiadora.Fotocopiar()
}


Monitor fotocopiadora{

    procedure Fotocopiar();
}
```

b) Modifique la solución de (a) para el caso en que se deba respetar el orden de llegada.

```
Monitor fotocopiadora {
    cond cola;
    boolean libre:=true;
    int dormidos:= 0;

    procedure solicitar(){
        if ( !libre ) {
             dormidos+=1;
             wait (cola);
        }
        libre:= false;
```

```
    }

    procedure salir(){
        if (dormidos > 0){
            dormidos -=1;
            signal(cola);
        }
        else
            libre:= true;
    }

}

process Persona [id: 0..N-1]{
    fotocopiadora.solicitar();
    id.Fotocopiar();
    fotocopiadora.salir();
}
```

c) Modifique la solución de (b) para el caso en que se deba dar prioridad de acuerdo con la
edad de cada persona (cuando la fotocopiadora está libre la debe usar la persona de mayor
edad entre las que estén esperando para usarla)

```
Monitor fotocopiadora {
    cond cola[N];
    boolean libre:=true;
    cola fila;

    procedure solicitar(id, edad: in int){
        if ( !libre ) {
             insertarOrdenado(fila, edad, id);
             wait (cola[id]);
```

```
        }
        libre:= false;
    }

    procedure salir(){
        fila.pop()
        if ( !fila.vacia() ){
            obtenerSiguiente(id)
            signal(cola[id]);
        }
        else
            libre:= true;
    }

}

process Persona [id: 0..N-1]{
    fotocopiadora.solicitar(id, edad);
    id.Fotocopiar();
    fotocopiadora.salir();
}
```

d) Modifique la solución de (a) para el caso en que se deba respetar estrictamente el orden
dado por el identificador del proceso (la persona X no puede usar la fotocopiadora hasta
que no haya terminado de usarla la persona X-1)

```
process Persona [id: 0..N-1]{
    fotocopiadora.realizarFotocopia(id);
    fotocopiadora.Fotocopiar();
    fotocopiadora.terminar();
}


Monitor fotocopiadora{
```

```
    cond procesos[N];
    int sig=0;

    procedure Fotocopiar();


procedure realizarFotocopia(id){
    if (id !=sig) -> wait(procesos[id];
}

procedure terminar(){
    sig+=1
    signal(procesos[sig]);
}
```

e) Modifique la solución de (b) para el caso en que además haya un Empleado que le indica
a cada persona cuando debe usar la fotocopiadora

```
Monitor fotocopiadora {
    cond cola, vcE, turno;
    boolean libre:= true;
    int dormidos:= 0;

    procedure solicitar(){
        dormidos+=1;
        signal(vcE);
        wait(cola);
        libre:= false;
    }

    procedure salir(){
        signal(vcE);
        libre:= true;
    }
```

```
    procedure esperarPersona(){
        if (dormidos==0) -> wait(vcE);
        if(!libre) -> wait(turno);
        dormidos-=1;
        signal(cola);


}

process Persona [id: 0..N-1]{
    fotocopiadora.solicitar();
    id.Fotocopiar();
    fotocopiadora.salir();
}

process Empleado(){
    for i:= 1 to N do
        fotocopiadora.esperarPersona();
```

f) Modificar la solución (e) para el caso en que sean 10 fotocopiadoras. El empleado le indica
a la persona cuál fotocopiadora usar y cuándo hacerlo.

```
Monitor Fotocopiadora {

    cond cola[N], esperar_empleado, avisar_empleado;
    vector fotocopiadora[N];
    cola fila;
    int dormidos= 0;
    cola idLibre;

    procedure entrar(id: int in, idFotocopiadora: int out){
        dormidos+=1
        signal(esperar_empleado)
        fila.push(id);
        wait(cola[id]);
        idFotocopiadora:= fotocopiadora[id]
```

```
    }

    procedure salir(id: int in){
        idLibre.push(id);
        signal(avisar_empleado);
    }

    procedure esperar(){
        int id;
        if (dormidos == 0) ->wait(esperarEmpleado);
        if (idLibre.empty()) -> wait(avisar_Empleado);
        id=fila.pop();
        fotocopiadora[id]= idLibre.pop;
        signal(cola);
        dormidos -=1;
    }
}

process Persona[id: 0.. N-1]{
    int idFotocopiadora;
    Fotocopiadora.usar(id, idFotocopiadora);
    //usar fotocopiadora con el idFotocopiadora;
    Fotocopiadora.salir(idFotocopiadora);
}

process Empleado(){
    for i:= 1 to N
        Fotocopiadora.esperar();
```

4. Existen N vehículos que deben pasar por un puente de acuerdo con el orden de llegada. Considere que el puente no soporta más de 50000kg y que cada vehículo cuenta con su propio peso (ningún vehículo supera el peso soportado por el puente).

```
Monitor puente{
    int peso= 50000;
    cond cola;
```

```
    cola fila;



    procedure entrar(pesoVehículo: in real){
        if ( (!fila.vacia()) or (peso - pesoVehículo< 0)){
            insertar(fila, pesoVehículo)
            wait(cola);
        }
        else
            peso-= pesoVehículo
    }


    procedure salir(pesoVehículo: in real){
         int sig;
         peso = peso + pesoVehículo;
         if (!fila.vacia()){
             sig= fila.top();
             while ((!fila.vacia) and (peso-sig >=0)){
                    fila.pop();
                    peso-=sig;
                    signal(cola);
                    if (!fila.vacia)
                         sig= fila.top();
                    }
             }
        }

}

process Auto[id: 0.. N-1]{

    puente.entrar(peso);
    //pasa por el puente
    puente.salir(peso);
```

5. En un corralón de materiales se deben atender a N clientes de acuerdo con el orden de llegada. Cuando un cliente es llamado para ser atendido, entrega una lista con los productos que comprará, y espera a que alguno de los empleados le entregue el comprobante de la compra realizada.
   a) Resuelva considerando que el corralón tiene un único empleado

```
Monitor Corralon {
    cond cola;
    int dormidos=0;
    cond cola, esperar;

    procedure esperarAtencion(){
        dormidos +=1;
        signal(esperar)
        wait(cola);
        dormidos-=1;
    }


    procedure esperarOAvisarCliente(){
        if (dormidos == 0) -> wait (esperar);
        signal(cola);
    }

}


    Monitor Escritorio {
        elem productos, comprobante;
        cond avisarEmpleado, comprobante, lista;

    procedure entregarProductos (lista: elem in){
        productos:= lista;
        signal(lista);
```

```
    }

    procedure esperarComprobante(compr: elem out){
        wait(comprobante);
        compr:= comprobante
        signal(comprobante);
    }

    procedure generarComprobante(){
        wait(lista);
        comprobante:=hacerComprobante(lista);
        signal(comprobante);
        wait(comprobante);
    }
}

process Persona [id: 0.. N-1]{
    Corralon.esperarAtencion();
    Escritorio.entregarProductos(lista);
    Escritorio.esperarComprobante(comprobante);

}

process Empleado(){
    for i:= N -1
        Corralon.esperarOAvisarCliente();
        Escritorio.generarComprobante();
}
```

b) Resuelva considerando que el corralón tiene E empleados (E > 1). Los empleados no
deben terminar su ejecución

```
Monitor corralón{
        cola eLibres;
        cond esperaC;
        int esperando=0 , cantELibres=0;
```

```
        Procedure llegada(idE: out int){
            if (cantELibres == 0){
                esperando+=1;
                wait(esperaC);
            }
            else
                cantELibres-=1;
            pop(eLibres, idE); //si no entré al if o me despe
        }

        Procedure proximo(id: in int){
            push(eLibres); //encolo el id del empleado libre
            if (esperando>0){
                esperando-=1;
                signal(esperaC);
            }
            else
                cantLibres+=1; //no hay nadie en la cola por
        }
}




process Cliente [id: 0.. N-1]{
    int idE;
    text lista, comprobante;

    corralon.llegada(idE);
    Escritorio[idE].atencion(lista, comprobante);

}


Monitor Escritorio [id: 0.. E-1]{
    cond colaCliente, colaEmpleado;
```

```
    boolean listo=false;
    text lista, comprobante;


    procedure atencion(listaP: in text, comprobanteListo: out
        lista:= listaP;
        listo:= true;
        signal(colaEmpleado);
        wait(colaCliente);
        comprobanteListo:= comprobante;
        signal(colaEmpleado);

    procedure esperarLista(listaP: out text){
        if (!listo)
            wait(colaEmpleado);
        listaP:= lista;
    }

    procedure enviarComprobante(comprob: int text){
        comprob:= comprobante;
        signal(colaCliente);
        wait(colaEmpleado);
        listo:= false;
}


process Empleado[id: 0.. E-1]{
    text lista, comprobante;
    while (true){
        corralon.proximo(id);
        Escritorio[id].esperarLista(lista);
        comprobante= resolver solicitud en base a los product
        Escritorio[id].enviarComprobante(comprobante);
    }
}
```

c) Modifique la solución (b) considerando que los empleados deben terminar su ejecución
cuando se hayan atendido todos los clientes.

```
Monitor corralón{
        cola eLibres;
        cond esperaC;
        int esperando=0 , cantELibres=0;
        int atendidos=0;


        Procedure llegada(idE: out int){
            if (cantELibres == 0){
                esperando+=1;
                wait(esperaC);
            }
            else
                cantELibres-=1;
            pop(eLibres, idE); //si no entré al if o me despe
        }

        Procedure proximo(id: in int){
            push(eLibres); //encolo el id del empleado libre
            if (esperando>0){
                esperando-=1;
                signal(esperaC);
            }
            else
                cantLibres+=1; //no hay nadie en la cola por
        }

        procedure incrementarAtendidos(atendidos: in out int)
            totAtendidos +=1;
            atendidos:= totAtendidos;
}
```

```
process Cliente [id: 0.. N-1]{
    int idE;
    text lista, comprobante;

    corralon.llegada(idE);
    Escritorio[idE].atencion(lista, comprobante);

}


Monitor Escritorio [id: 0.. E-1]{
    cond colaCliente, colaEmpleado;
    boolean listo=false;
    text lista, comprobante;


    procedure atencion(listaP: in text, comprobanteListo: out
        lista:= listaP;
        listo:= true;
        signal(colaEmpleado);
        wait(colaCliente);
        comprobanteListo:= comprobante;
        signal(colaEmpleado);

    procedure esperarLista(listaP: out text){
        if (!listo)
            wait(colaEmpleado);
        listaP:= lista;
    }


    procedure enviarComprobante(comprob: int text, totalAtend
        comprob:= comprobante;
        signal(colaCliente);
        wait(colaEmpleado);
```

```
        listo:= false;
        corralon.incrementarAtendidos(totalAtendidos);

    }


process Empleado[id: 0.. E-1]{
    text lista, comprobante;
    int atendidos:=0
    corralon.proximo(id);
    Escritorio[id].esperarLista(lista);
    comprobante= resolver solicitud en base a los productos;
    Escritorio[id].enviarComprobante(comprobante, atendidos);
    while (atendidos!= N){
        corralon.proximo(id);
        Escritorio[id].esperarLista(lista);
        comprobante= resolver solicitud en base a los product
        Escritorio[id].enviarComprobante(comprobante, atendid
    }
}
```

6. Existe una comisión de 50 alumnos que deben realizar tareas de a pares, las cuales son
corregidas por un JTP. Cuando los alumnos llegan, forman una fila. Una vez que están todos
en fila, el JTP les asigna un número de grupo a cada uno. Para ello, suponga que existe una
función AsignarNroGrupo() que retorna un número “aleatorio” del 1 al 25. Cuando un alumno ha recibido su número de grupo, comienza a realizar su tarea. Al terminarla, el alumno le avisa
al JTP y espera por su nota. Cuando los dos alumnos del grupo completaron la tarea, el JTP
les asigna un puntaje (el primer grupo en terminar tendrá como nota 25, el segundo 24, y así
sucesivamente hasta el último que tendrá nota 1). Nota: el JTP no guarda el

número de grupo
que le asigna a cada alumno.

```
Monitor Tarea {
    cond avisar, barrera;
    int alumnos=0;
    vector idGrupo[50];
    cola fila;
    vector grupos [25]; // posee un campo cantidad y un campo
    cond esperaPuntaje [25];
    int finalizados=0;
    int puntaje=25;
    cola filaAlumnos;

    procedure obtenerNro(id: int in, nro: int out){
        alumnos+=1;
        fila.push(id)
        if (alumnos == 50 ){
            signal(avisar);
        }
        wait(barrera);
        nro:= idGrupo[id];
    }

    procedure esperarAlumno(){
        if (alumnos < 50 ) -> wait (avisar);
        for i:= 1 to 50
            idGrupo[fila.pop]= AsignarNroGrupo;
        signal_all(barrera);
    }

    procedure avisar (grupo: int in, puntaje: int out){
        grupos[id].cantidad +=1;
        filaAlumnos.push(id);
        signal (jtp);
```

```
        wait(esperaPuntaje[id]);
        puntaje:= grupos[id].puntaje
    }


    procedure asignarPuntaje(){
        if (fila.vacia()) -> wait (jtp);
        id:= filaAlumnos.pop();
        if (grupos [id] == 2 ){
            finalizados +=1;
            grupos[id].puntaje := puntaje;
            puntaje-=1;
            signal_all (esperaPuntaje [id]);
        }
    }
}



process Alumno [id: 0.. 49] {
    int grupo, puntaje;
    Tarea.obtenerNro(id, grupo);
    //hacer tarea
    Tarea.avisar(grupo, puntaje);
}

process JTP (){
    Tarea.esperarAlumno();
    for i:= 1 to 50
        Tarrea.asignarPuntaje();

}
```

7. Se debe simular una maratón con C corredores donde en la llegada hay UNA máquina
   expendedoras de agua con capacidad para 20 botellas. Además, existe un

repositor encargado de reponer las botellas de la máquina. Cuando los C corredores han llegado al inicio comienza la carrera. Cuando un corredor termina la carrera se dirigen a la máquina expendedora, espera su turno (respetando el orden de llegada), saca una botella y se retira. Si encuentra la máquina sin botellas, le avisa al repositor para que cargue nuevamente la máquina con 20 botellas; espera a que se haga la recarga; saca una botella y se retira. Nota: mientras se reponen las botellas se debe permitir que otros corredores se encolen.

```
Monitor carrera {

        cond corredor, colaBotellas;
        int cantC=0;
        int C;
        int colaCant=0;
        boolean libre=true;


        procedure llegarInicio(){
            cantC+=1;
            if (cantC== C)
                signal_all(corredor);
            else
                wait(corredor);

        }

        procedure usarMaquina(){
            if (!libre ){
                colaCant +=1
                wait (colaBotellas);
            }
        }

        procedure salir(){
            if (colaCant > 0) -> signal (colaBote
            else
```

```
                        libre= true;
                }


        }



        Monitor Maquina {

            vector botellas[20];
            int cantBotellas = 20;
            cond repositor, carga;

             process sacarBotella (){
                      if (cantBotellas ==  0){
                            signal(repositor);
                            wait (carga);
                        }
                            cantBotellas-=1;
                            botella = botellas[cantBotellas];
                }

                        procedure esperarACargar(){
                            wait (repositor);
                            for i := 1 to 20
                                botellas[i].cargar();
                            signal(carga);
                        }

        }

        process Corredor[id: 0.. C-1]{
            elem botella
            carrera.llegarInicio();
            //correr
            Carrera.usarMaquina();
            Maquina.sacarBotella(botella);
```

```
            carrera.salir();

          }

          process Repositor(){

              for i:=  1 to C
               Maquina.esperarACargar();
            }
```

8. En un entrenamiento de fútbol hay 20 jugadores que forman 4 equipos (cada jugador conoce
el equipo al cual pertenece llamando a la función DarEquipo()). Cuando un equipo está listo
(han llegado los 5 jugadores que lo componen), debe enfrentarse a otro equipo que también
esté listo (los dos primeros equipos en juntarse juegan en la cancha 1, y los otros dos equipos
juegan en la cancha 2). Una vez que el equipo conoce la cancha en la que juega, sus jugadores se dirigen a ella. Cuando los 10 jugadores del partido llegaron a la cancha comienza el partido, juegan durante 50 minutos, y al terminar todos los jugadores del partido se retiran (no es necesario que se esperen para salir).

```
Monitor Equipo[0..3]{
    int llegaron=0;
    cond espera;
    int cancha;

    procedure llegar(nroCancha: int out){
        llegaron+=1;
        if(llegaron == 5){
            Administrador.pedirCancha(cancha);
            signal_all(espera);
        }
        else
```

```
            wait(espera);
        nroCancha:= cancha;
    }

}

Monitor Administrador{
    int llegaron=0;

    procedure pedirCancha(cnacha: int out){
        llegaron+=1;
        if(llegaron <= 2)
            cancha:=0;
        else
            cancha:=1;
    }
}

Monitor Cancha[id:0..1]{
    int llegaron=0;
    cond esperar, fin;

    procedure iniciar(){
        llegaron+=1;
        if( llegaron == 10 )
            signal(esperar)
        wait(fin);
  }

  procedure jugar(){
      if(llegaron < 10) -> wait (esperar);
    }

    procedure terminar(){
        signal_All(fin);
    }
}
```

```
process Jugador[id: 0.. 19]{
    int equipo:= darEquipo()
    int cancha;
    Equipo[equipo].llegar(cancha);
    Cancha[cancha].iniciar();
    //retirarse
}

process Partido[id:0..1]{
    Cancha[id].jugar();
    delay(50);
    Cancha[id]terminar();
}
```

9. En un examen de la secundaria hay un preceptor y una profesora que deben tomar un examen
escrito a 45 alumnos. El preceptor se encarga de darle el enunciado del examen a los alumnos
cundo los 45 han llegado (es el mismo enunciado para todos). La profesora se encarga de ir
corrigiendo los exámenes de acuerdo con el orden en que los alumnos van entregando. Cada
alumno al llegar espera a que le den el enunciado, resuelve el examen, y al terminar lo deja
para que la profesora lo corrija y le envíe la nota. Nota: maximizar la concurrencia; todos los
procesos deben terminar su ejecución; suponga que la profesora tiene una función
corregirExamen que recibe un examen y devuelve un entero con la nota.

```
Monitor Preceptor {
    int llegaron=0;
    cond barrera, todos;
    elem enunciado;
```

```
    procedure llegar(enun: elem out){
        llegaron +=1;
        if (llegaron == 45) -> signal(todos);
        wait(barrera);
        enun:= enunciado
    }

    procedure esperarAlumnos(){
        if (llegaron < 45) -> wait(todos);
        enunciado= //enunciado
        signal_all(barrera);
    }
}

Monitor Profesora{

    cond esperaNota, llegue;
    int nota;
    cola fila;
    vector nota[45];

    procedure esperarNota(examen: elem in, not: int out){
        fila.push(examen, id);
        signal(llegue);
        wait(esperaNota);
        not:= nota[id];
  }

  procedure corregir(){
      int id;
      elem examen;
      if (fila.vacia()) -> wait(llegue);
      fila.pop(examen, id);
      nota[id]:= corregirExamen(examen;
    }


}
```

```
process Alumno[id: 0.. 44] {
    elem enunciado;
    Preceptor.llegar(enunciado);
    //hacer examen
    Profesora.esperarNota(examen, nota);

}

process Preceptor(){
    Preceptor.esperarAlumnos();
}

process Profesora(){
    for i:= 1 to 45
        Profesora.corregir();
}
```

10.En un parque hay un juego para ser usada por N personas de a una a la vez y de acuerdo al
orden en que llegan para solicitar su uso. Además, hay un empleado encargado de desinfectar el
juego durante 10 minutos antes de que una persona lo use. Cada persona al llegar espera hasta
que el empleado le avisa que puede usar el juego, lo usa por un tiempo y luego lo devuelve.
Nota: suponga que la persona tiene una función Usar_juego que simula el uso del juego; y el
empleado una función Desinfectar_Juego que simula su trabajo. Todos los procesos deben
terminar su ejecución.

Este ejercicio es el único que no está corregido

```
Monitor Juego {
```

```
        cond cola, empleados, libre;
        int dormidos=0

    procedure jugar(){
        dormidos +=1;
        if (dormidos> 1){
            wait(cola);
        }
        dormidos -=1

    }

    procedure salir (){
        signal(empleado);
    }

    procedure esperar(){
        wait(empleado);
    }

    procedure avisar(){
        if (dormidos >0 ) signal(cola);

}


process Persona[id:0 .. N-1]{
    Juego.jugar();
    //jugar
    Juego.salir();
}

process Empleado(){
    for i:= 1 to N {
        Juego.esperar();
        delay(10);
        Juego.avisar();
```

```
    }
}
```

Resolver el siguiente problema con ***SEMÁFOROS***. Simular la atención de una Planta Verificadora de Vehículos, donde se revisan cuestiones de seguridad de vehículos de a uno por vez. Hay ***N vehículos*** que deben ser verificados, donde algunos son autos y otros ambulancias. Antes de ser verificados, los autos deben hacer el pago correspondiente en la Caja de la Planta, donde le entregarán un recibo de pago. Las ambulancias no pagan. Los vehículos son atendidos de acuerdo al orden de llegada pero siempre dando prioridad a las ambulancias.