

Primera Fecha 2025 - 14/05 Tema 1

- Programación Concurrente ATIC y Redictado - Parcial Práctico de MC - Primera Fecha - 14/05/2025**
1. Resolver con **SEMÁFOROS** el siguiente problema. Se tiene un vector A de 1.000.000 de números, del cual se debe obtener el promedio de sus valores utilizando 10 procesos Worker. Al terminar todos los procesos deben imprimir el resultado. Nota: maximizar concurrencia; únicamente se pueden usar los 10 procesos *Worker*.
 2. Resolver con **SEMÁFOROS** la siguiente situación. A una acopiadora de cereales llegan 20 camiones que deben descargar su cereal. Los camiones se descargan de a uno por vez, y de acuerdo con orden de llegada. Una vez que el camión llegó, espera a que llegue su turno para comenzar a descargar su cereal. Nota: sólo se pueden usar procesos que representen a los camiones; cada camión descarga sólo una vez; la descarga del camión se simula por medio de la función *DESCARGAR()* llamada por el camión.
 3. Resolver con **MONITORES** el siguiente problema. En una cátedra hay un profesor, un auxiliar y 100 alumnos. Cada alumno continuamente hace consultas que pueden ser de dos tipos: **TEÓRICAS** o **PRÁCTICAS**; cada vez que tiene una consulta para hacer, se la envía por mail al docente correspondiente, y espera a que este le envíe la respuesta. El profesor sólo atiende consultas TEÓRICAS y el auxiliar sólo consultas PRÁCTICAS; cada uno resuelve sus consultas de acuerdo con orden de llegada. Nota: maximizar concurrencia; el alumno sabe de qué tipo es cada consulta; los procesos NO deben terminar.

1.

```
int vector [1000000];
int cantDivididos = 1000000 DIV 10;
int sumaTotal = 0;
sem mutex = 1;
sem barrera = 1;
sem espero = 0;
```

Process Worker [id: 0..9] {

```
    int i;
    int ini = id * cantDivididos;
    int fin = cantDivididos + ini - 1;
    int sumaAux = 0;
    for i: ini .. fin {
        sumaAux += vector[i];
    }
    P (mutex);
```

```

sumaTotal += sumaAux;
V (mutex);
P (barrera);
cant++;
if cant == 10 {
    for i: 0..9 V (espero);
}
V (barrera);
P (espero);
writeln (sumaTotal / 10);
}

```

2.

```

sem mutex = 1;
sem espera [20] = ([20], 0);
boolean libre = true;
Cola c;

```

Process Camion [id: 0..19] {

```

int idAux;
P (mutex);
if not libre {
    c.push(id);
    V (mutex);
    P (espera[id]);
} else {
    libre = false;
    V (mutex);
}
Descargar();
P (mutex);
if c.isEmpty() {
    libre = true;
} else {
    c.pop(idAux);
    V (espera[idAux]);
}

```

```

    }
    v (mutex);
}

```

3.

```

Process Alumno [id: 0..99] {
    text respuesta;
    while (true) {
        text c = generarConsulta();
        if c.equals ("Teorica") idDocente = 0; // Profesor
        else idDocente = 1; // Auxiliar
        Docente[idDocente].enviarConsulta(id, c, respuesta);
    }
}

```

```

Process Profesor {
    text sec, res;
    int aux;
    while (true) {
        Docente[0].sig(aux, sec);
        res = ResponderConsulta(sec); // Responder la consulta
        Docente[0].resultado(aux, res);
    }
}

```

```

Process Auxiliar {
    text sec, res;
    int aux;
    while (true) {
        Docente[1].sig(aux, sec);
        res = ResponderConsulta(sec);
        Docente[1].resultado(aux, res);
    }
}

```

```

Monitor Docente [idD: 0..1] {

```

```
cond espera, hayPedido;  
text consultas[N];  
Cola c;
```

```
Procedure enviarConsulta(id: in int, consulta: in text, respuesta: out text) {  
    cola.push(id, consulta);  
    signal (hayPedido);  
    wait (espera);  
    respuesta = consultas[id];  
}
```

```
Procedure sig (id: out int, consulta: out text) {  
    if cola.isEmpty() wait (hayPedido);  
    c.pop(id, consulta);  
}
```

```
Procedure resultado(id: in int, res: out text) {  
    consultas[id] = res;  
    signal (espera);  
}  
}
```

Primera Fecha 2025 - 14/05 Tema 2

Programación Concurrente ATIC y Redictado - Parcial Práctico de MC - Primera Fecha - 14/05/2025

1. Resolver con **SEMÁFOROS** el siguiente problema. Se tiene un vector A de 1.000.000 de caracteres, del cual se debe obtener la cantidad de veces que aparecen los caracteres “F” y “C”, utilizando **4 procesos Worker**. Al terminar todos los procesos deben imprimir la cantidad de veces que aparece cada una de esas dos letras (F y C). **Nota:** maximizar concurrencia; únicamente se pueden usar los 4 procesos *Worker*.
2. Resolver con **SEMÁFOROS** el siguiente problema. Existen **B barcos** areneros que deben descargar su contenido en una playa. Los barcos se descargan de a uno por vez, y de acuerdo con orden de llegada. Una vez que el barco llegó, espera a que le llegue su turno para comenzar a descargar su contenido, y luego se retira. **Nota:** sólo se pueden usar los procesos que representen a los barcos; cada barco descarga sólo una vez; suponga que existe la función *DESCARGAR()* que simula que el barco está descargando su contenido en la playa.
3. Resolver con **MONITORES** el siguiente problema. En una acopiadora de cereales hay **2 empleados**, uno para atender a los camiones de maíz y otro para los de girasol. Hay 30 camiones que llegan para descargar su carga (15 de maíz y 15 de girasol), cuando el camión llega espera hasta que el empleado correspondiente le avise que puede descargar el cereal. Cada empleado hace descargar los camiones que le corresponden de a uno a la vez y de acuerdo con orden de llegada. **Nota:** maximizar concurrencia; el camión sabe qué tipo de cereal lleva; todos los procesos deben terminar

1.

```
char vector [1000000];
int cantDivididos = 1000000 DIV 4;
int sumaF = 0;
int sumaC = 0;
sem mutexF = 1;
sem mutexC = 1;
sem barrera = 1;
sem espero = 0;
```

Process Worker [id: 0..3] {

```
    int i;
    int ini = id * cantDivididos;
    int fin = cantDivididos + ini - 1;
    int sumaAuxF = 0;
    int sumaAuxC = 0;
    for i: ini .. fin {
        if vector[id] == 'F' {
            sumaF++;
        } else if vector[id] == 'C' {
            sumaC++;
        }
    }
    P (mutexF);
    sumaF += sumaAuxF;
```

```

V (mutexF);
P (mutexC);
sumaC = sumaAuxC;
V (mutexC);
P (barrera);
cant++;
if cant == 4 {
    for i: 0..3 V (espero);
}
V (barrera);
P (espero);
writeln (sumaF);
writeln (sumaC);
}

```

2.

```

sem mutex = 1;
sem espera [B] = ([B], 0);
boolean libre = true;
Cola c;

```

```

Process Barco [id: 0..B-1] {
    int idAux;
    P (mutex);
    if not libre {
        c.push(id);
        V (mutex);
        P (espera[id]);
    } else {
        libre = false;
        V (mutex);
    }
    Descargar();
    P (mutex);
    if c.isEmpty() {
        libre = true;
    }
}

```

```

} else {
    c.pop(idAux);
    v(espera[idAux]);
}
v(mutex);

}

3.

Procedure Empleado [id: 0..1] {
    for i: 0 .. 14 {
        Gestion[id].siguiente();
    }
}

Proceso Camion {
    int tipoCereal = ...;
    Gestion[tipoCereal].llegue();
    // Descargo
    Gestion[tipoCereal].salir();
}

Monitor Gestión [id: 0..1] {
    cond aviso;
    cond sig;
    cond descargue;
    int esperando = 0;

    Procedure llegue() {
        esperando++;
        signal(aviso);
        wait(sig);
    }

    Procedure salir() {
        signal(descargue);
    }
}

```

}

```
Procedure siguiente() {
    if esperando == 0 {
        wait (aviso);
    }
    esperando--;
    signal(sig);
    wait (descargue);
}
```

}

Primera Fecha 2025 - 14/05 Tema 3

- Programación Concurrente ATIC y Redictado - Parcial Práctico de MC - Primera Fecha - 14/05/2025**
- ✓ Resolver con **SEMÁFOROS** el siguiente problema. En un laboratorio de genética trabajan **20 empleados** que deben usar un pirosecuenciador de a uno a la vez, de acuerdo con el orden de llegada. **Nota:** sólo se pueden usar los procesos que representen a los empleados; cada empleado usa sólo una vez el pirosecuenciador; suponga que existe la función **USAR()** que simula el uso de este instrumento por parte del empleado.
2. Resolver con **SEMÁFOROS** el siguiente problema. Se tiene un vector A de 1.000.000 de números, del cual se debe obtener la suma de todos los valores utilizando **5 procesos Worker**. Al terminar todos los procesos deben imprimir el resultado final. **Nota:** maximizar concurrencia; únicamente se pueden usar los 5 procesos **Worker**.
 3. Resolver con **MONITORES** el siguiente problema. En una oficina hay **un supervisor, un empleado y 50 personas** que solicitan por mail una operación. El empleado sólo atiende CONSULTAS, mientras que el supervisor sólo atiende TRÁMITES; cada uno atiende sus pedidos de acuerdo con el orden de llegada. Cada persona envía UNA solicitud que puede ser para un TRÁMITE o para una CONSULTA, y luego espera a que le envíen el resultado. **Nota:** maximizar concurrencia; la persona sabe de qué tipo es la consulta; el empleado y el supervisor NO deben terminar su ejecución.

1.

```
sem mutex = 1;
sem espera [20] = ([20], 0);
boolean libre = true;
Cola c;
```

```
Process Empleado [id: 0..19] {
```

```
    int idAux;
    P (mutex);
    if not libre {
        c.push(id);
        V (mutex);
        P (espera[id]);
    } else {
        libre = false;
        V (mutex);
    }
    Usar();
    P (mutex);
    if c.isEmpty() {
        libre = true;
    } else {
        c.pop(idAux);
        V (espera[idAux]);
    }
}
```

```
v (mutex);
}
```

2.

```
int vector [1000000];
int cantDivididos = 1000000 DIV 5;
int sumaTotal = 0;
sem mutex = 1;
sem barrera = 1;
sem espero = 0;
```

Process Worker [id: 0..4] {

```
    int i;
    int ini = id * cantDivididos;
    int fin = cantDivididos + ini - 1;
    int sumaAux = 0;
    for i: ini .. fin {
        sumaAux += vector[i];
    }
    P (mutex);
    sumaTotal += sumaAux;
    V (mutex);
    P (barrera);
    cant++;
    if cant == 5 {
        for i: 0..4 V (espero);
    }
    V (barrera);
    P (espero);
    writeln (sumaTotal);
}
```

3.

Process Supervisor {
 text consulta, res;
 int id;

```

while (true) {
    Consulta[0].sig (id, consulta);
    res = resolverConsulta(consulta);
    Consulta[0].res (id, res);
}

```

```

Process Empleado {
    text consulta, res;
    int id;
    while (true) {
        Consulta[1].sig (id, consulta);
        res = resolverConsulta(consulta);
        Consulta[1].res (id, res);
    }
}

```

```

Process Persona [id: 0..49] {
    text res;
    int tipoConsulta = obtenerTipoConsulta();
    text consulta = obtenerConsulta();
    Consulta[tipoConsulta].atender(id, consulta, res);
}

```

```

Monitor Consulta [id: 0..1] {
    Cola c;
    int idAux;
    signal hayPedido, hayRespuesta;
    text resultados [N];
}

```

```

Procedure atender (idP: in int, consulta: in text, res: out text) {
    c.push(idP);
    signal (hayPedido);
    wait (hayRespuesta);
    res = resultados [idP];
}

```

```
Procedure sig (idP: out int, consulta: out text) {  
    if c.isEmpty() wait (hayPedido);  
    c.pop(idP, consulta);  
}
```

```
Procedure res (idP: in int, res: in text) {  
    resultados [idP] = res;  
    signal (hayRespuesta);  
}  
}
```

Primera Fecha 7-10-24

1. Se debe simular el uso de un sistema virtual de venta de entradas para un evento musical. El sistema cuenta con C cajeros virtuales que atienden indefinidamente. Sin embargo, como la venta de entradas comienza a una hora determinada, sólo atienden a partir del aviso de un Timer. Una vez que reciben dicho aviso, los cajeros atienden de acuerdo con el orden de llegada de los compradores. La atención consiste en recibir la solicitud del comprador (datos para el pago) y responderle si pudo comprar (o no) junto al comprobante de la operación. Para este evento se cuenta con E entradas y N compradores, donde cada comprador puede solicitar a lo sumo una entrada. Resuelva usando **SEMÁFOROS**.

2. Existen N personas que desean acceder a un mirador al borde del lago Nahuel Huapi en Bariloche. Como el mirador es angosto, sólo puede ser usado por una persona a la vez. Resuelva con **MONITORES** los dos casos siguientes:
 - a. El acceso al mirador es por orden de llegada.

 - b. El acceso al mirador es por orden de llegada, pero dando prioridad a los mayores de 60 años.

1.

```

sem hayCompra;
sem excEntradas;
sem vendedor [C] = ([C], 0);
sem esperaCompra[N] = ([N], 0);
text comprobantes[N];
boolean compras[N];
int cantEntradas = E;
Cola c;
```

Process Timer {

```

  for int i: 0..C-1 {
    V (vendedor[i]);
  }
}
```

Process Comprador [id: 0..N-1] {

```

  text solicitud, comprobante;
  boolean pudeComprar;
  P (mutexCola);
  c.push(id);
  V (mutexCola);
  V (hayCompra);
  P (esperaCompra[id]);
```

```

comprobante = comprobantes[id];
pudeComprar = compras[id];
}

```

```

Process Cajero [id: 0..C-1] {
    int idAux;
    P (vendedor[i]);
    while (true) {
        P(hayCompra);
        P (mutexCola);
        c.pop(idAux);
        V (mutexCola);
        boolean hayEntrada;
        P (excEntradas);
        if cantEntradas > 0 {
            cantEntradas--;
            hayEntrada = true;
        } else {
            hayEntrada = false;
        }
        V (excEntradas);
        text comprobante = generarComprobante(hayEntrada);
        comprobantes[idAux] = comprobante;
        compras[idAux] = hayEntrada;
        V (esperaCompras[idAux]);
    }
}

```

2. a.

```

Procedure Persona [id: 0..N] {
    Mirador.usar();
    // Cruza el mirador
    Mirador.salir();
}

```

```

Monitor Mirador {

```

```

boolean libre = true;
int esperando = 0;
cond cola;

Procedure usar() {
    if not libre {
        esperando++;
        wait (cola);
    } else {
        libre = false;
    }
}

```

```

Procedure liberar() {
    if esperando > 0 {
        esperando--;
        signal (cola);
    } else {
        libre = true;
    }
}

```

b.

```

Procedure Persona [id: 0..N] {
    int edad = ...;
    Mirador.usar(id, edad);
    // Cruza el mirador
    Mirador.salir();
}

```

```

Monitor Mirador {
    boolean libre = true;
    int id;
    int esperando = 0;
    cond cola[N];
}

```

```
colaOrdenada fila;

Procedure usar(id: in int, edad: in int) {
    if not libre {
        fila.pushOrdenado(id, edad); // Deja primeros a las personas
mayores a 60..
        esperando++;
        wait (cola[id]);
    } else {
        libre = false;
    }
}

Procedure liberar() {
    if esperando > 0 {
        esperando--;
        fila.pop(id);
        signal (cola[id]);
    } else {
        libre = true;
    }
}
```

Segunda fecha 25-11-24

Ejercicios planeados.

1. Resolver con **SEMÁFOROS** el siguiente problema. La Clave Única de Identificación Tributaria (CUIT) es una clave que se utiliza en el sistema tributario de la República Argentina para poder identificar correctamente a las personas físicas o jurídicas. Consta de un total de once (11) cifras numéricas, siendo la última un dígito verificador (del 0 al 9). Una empresa cuenta con una lista de CUITs que debe procesar, debiendo informar la cantidad de CUITs por dígito verificador. Para ello, dispone de un software que emplea 5 workers, los cuales trabajan colaborativamente procesando de a una CUIT por vez cada uno. Al finalizar el procesamiento, el último worker en terminar debe informar los resultados del procesamiento. Notas: la función *obtenerDV(CUIT)* retorna el dígito verificador para la CUIT recibida como entrada. La lista de CUITs se almacena como una cola global y la solución debe maximizar la concurrencia.

2. Resolver con **MONITORES** la siguiente situación. En un negocio hay UN empleado que diseña tarjetas digitales. El empleado debe atender los pedidos de C clientes, de acuerdo con el orden en que se hacen los pedidos. El cliente envía las indicaciones, y el empleado en base a eso diseña la tarjeta y se la envía al cliente. Notas: maximizar la concurrencia; existe una función *HacerTarjeta(indicaciones)* que simula el armado de la tarjeta por parte del empleado; todos los procesos deben terminar su ejecución.

1.

```
sem mutexCola = 1;
sem mutexBarrera = 1;
sem vectorMutex[10] = ([10], 1);
int vectorContador [10] = ([10], 0);
Cola lista;
```

```
Process Worker [id: 0..4] {
    P (mutexCola);
    int cuit;
    while (not lista.isEmpty()) {
        c.pop(cuit);
        V (mutexCola);
        int digito = obtenerDV (cuit);
        P (vectorMutex[digito]);
        vectorContador [digito]++;
        V (vectorMutex[digito]);
        P (mutexCola);
    }
    V (mutexCola);
    P (mutexBarrera);
    cant++;
    if cant == 5 {
        for int i: 0 .. 9 { writeln(vectorContador[i]); }
    }
}
```

```

    }
    v (mutexBarrera);
}

```

2.

```

Process Cliente [id: 0..C-1] {
    text instrucciones, tarjeta;
    Negocio.enviarInstrucciones(id, instrucciones, tarjeta);
}

```

```

Process Empleado {
    text instrucciones, res;
    ind idCliente;
    for i: 0 .. C {
        Negocio.recibirInstrucciones(idCliente, instrucciones);
        res = HacerTarjeta(instrucciones);
        Negocio.enviarResultados(idCliente, res);
    }
}

```

```

Monitor Negocio {
    cond cola, hayPedido;
    Cola c;
    text [C] tarjetas;
}

```

```

Procedure enviarInstrucciones (id: in int, inst: in int, tarjeta: out text) {
    c.push (id, inst);
    signal (hayPedido);
    wait (cola);
    tarjeta = tarjetas[id];
}

```

```

Procedure recibirInstrucciones (id: out int, inst: out text) {
    if c.isEmpty() wait (hayPedido);
    c.pop(id, inst);
}

```

```
Procedure enviarResultados (id: in int, res: in text) {  
    tarjetas [id] = res;  
    signal (cola);  
}  
}
```

Tercera Fecha 9-12-24

1. Existe una sala de cine 3D, a las que asisten N personas a ver una película. Antes de entrar a la salsa, los asistentes deben retirar los anteojos 3D en la máquina repartidora que se encuentra en la entrada. Se debe simular el uso de la máquina repartidora de anteojos 3D, con capacidad para A anteojos ($A < N$). Además, existen un repositor encargado de reponer los anteojos en la máquina cuando se agotan. Los usuarios usan la máquina según el orden de llegada. Cuando les toca usarla, sacan un par de anteojos y luego se dirigen a la sala. En caso de que la máquina se quede sin anteojos, entonces le debe avisar al repositor para que cargue nuevamente la máquina en forma completa. Luego de la recarga, saca un par de anteojos y se retira. Implemente un programa que permita resolver el problema anterior usando **SEMÁFOROS**. **Nota:** maximizar la concurrencia; la reposición de anteojos no debe impedir que otros asistentes puedan agregarse a la fila.

2. En el Registro de la Propiedad se pueden realizar 4 trámites administrativos diferentes. Para cada trámite, hay un puesto de atención específico. Existen 100 personas que se dirigen a la oficina para resolver un trámite particular. La persona deja su trámite en el puesto correspondiente y espera a que le entreguen el resultado. El puesto atiende a las personas que le corresponden de acuerdo con el orden de llegada. Implemente un programa que permita resolver el problema anterior usando **MONITORES**. **Notas:** maximizar la concurrencia; todos los procesos deben terminar; la función *obtenerPuesto()* retorna el número de puesto al que la persona debe dirigirse para su trámite; la función *obtenerTrámite()* retorna el trámite a realizar; la función *procesarTrámite(t)* procesa el trámite recibido como entrada y retorna su resultado.

1.

```
Cola c;
sem tengoQueReponer = 0;
sem hayAnteojos = 0;
sem mutex = 1;
sem espera [N] = ([N], 0);
boolean libre = true;
int cantAnteojos = A;
```

```
Process Asistente [id: 0 .. N-1] {
    int idAux;
    P (mutex);
    if not libre {
        c.push(id);
        V (mutex);
        P (espera[id]);
    } else {
        libre = false;
        V (mutex);
    }
    if cantAnteojos == 0 {
        V (tengoQueReponer);
```

```

P (hayAnteojos);
}
cantAnteojos--;
P (mutex);
if c.isEmpty() libre = true;
else {
    c.pop (idAux);
    V (espera
}
V (mutex);
}

```

```

Process Repositor {
    while (true) {
        P (tengoQueReponer);
        cantAnteojos = A;
        V (hayAnteojos);
    }
}

```

2.

```

Process Persona [id: 0..99] {
    text trámite, res;
    trámite = obtenerTrámite();
    Puesto[obtenerPuesto()].Pedir (id, trámite, res);
}

```

```

Process Empleado [id: 0..3] {
    int idPersona;
    text trámite, res;
    for i: 0 .. 24 {
        Puesto[id].Siguiente (idPersona, trámite);
        res = procesarTrámite (trámite);
        Puesto[id].Res (idPersona, res);
    }
}

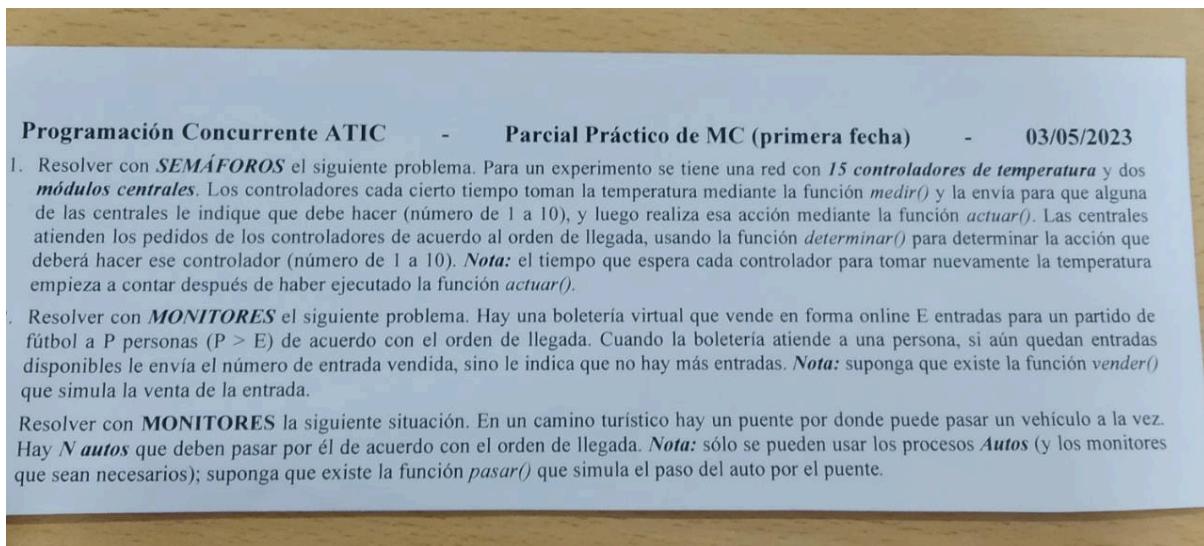
```

```
Monitor Puesto [id: 0..3] {
    Cola c;
    cond hayPedido, pedidoListo;
    text [N] tramites;

    Procedure Pedir (id: in int, trámite: in text, res: out text) {
        c.push (id, trámite);
        signal (hayPedido);
        wait (pedidoListo);
        res = tramites[id];
    }

    Procedure Siguiente (idP: out int, trámite: out text) {
        if c.isEmpty() wait (hayPedido);
        c.pop (idP, trámite);
    }

    Procedure Res (idP: in int, res: in text) {
        tramites[idP] = res;
        signal (pedidoListo);
    }
}
```

Primera Fecha 03/05/2023 ATIC

1.

Cola c;

```
sem mutex = 1;
sem hayPedido = 0;
sem espera[15] = ([15], 0);
int vecAcciones [15];
```

```
Process Controlador [id: 0..14] {
    while (true) {
        double temp = medir();
        P (mutex);
        c.push(id, temp);
        V (mutex);
        V (hayPedido);
        P (espera[id]);
        actuar (vecAcciones[id]); // El controlador espera para otra temp
    }
}
```

```
Process Modulos [id: 0..1] {
    int idAux;
    double temp;
    while (true) {
```

```

P (hayPedido);
P (mutex);
c.pop (idAux, temp);
V (mutex);
vecAcciones[idAux] = determinar(temp);
V (espera[idAux]);
}
}

```

2.

```

Process Cliente [id: 0.. P-1] {
    int numEntrada;
    Admin.pedido (id, numEntrada);
}

```

```

Process Empleado {
    int numEntrada, aux;
    int cantEntradas = E;
    for i: 0 .. P-1 {
        Admin.sig (aux);
        if (cantEntradas > 0) {
            numEntrada = vender();
        } else {
            numEntrada = -1;
        }
        Admin.res (aux, numEntrada);
    }
}

```

```

Monitor Admin {
    Cola C;
    cond espera;
    cond hayPedido;
    int res [N];

```

```

Procedure pedido (id: in int, numEntrada: out int) {

```

```

C.push(id);
signal (hayPedido);
wait (espera);
numEntrada = res[id];
}

Procedure sig (aux: out int) {
    if C.isEmpty wait (hayPedido);
    c.pop(aux);
}

Procedure res (aux: in int, entrada: in int) {
    res[aux] = entrada;
    signal (espera);
}

```

3.

```

Process Autos [0..N-1] {
    Puente.ingresar();
    pasar();
    Puente.salir();
}

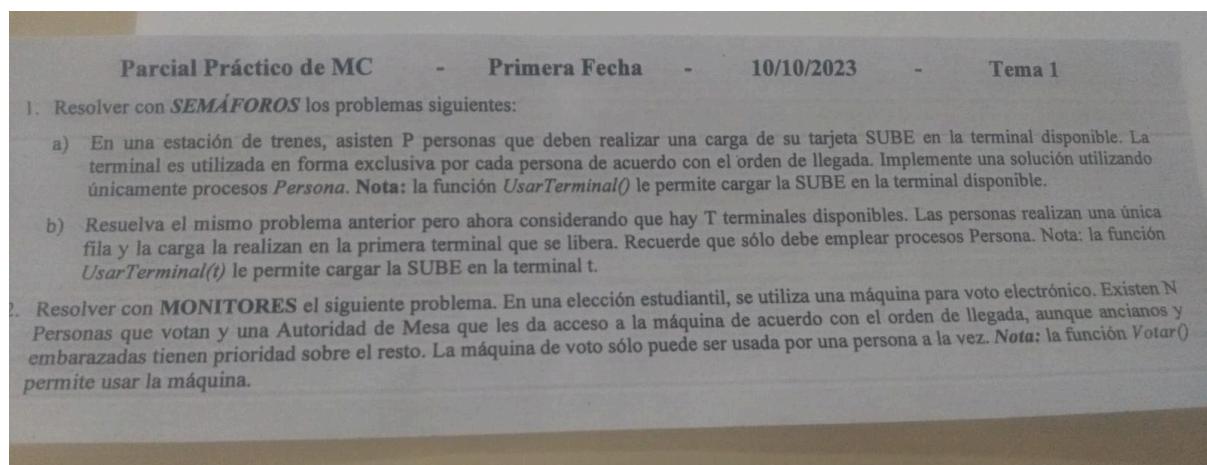
Monitor Puente {
    bool libre = true;
    int esperando = 0;
    cond cola;

    Procedure ingresar() {
        if not libre {
            esperando++;
            wait (cola);
        } else {
            libre = false;
        }
    }
}

```

```
Procedure salir() {
    if esperando > 0 {
        esperando--;
        signal (cola);
    } else {
        libre = true;
    }
}
```

Primera Fecha 10/10/2023 Tema 1



1.

a)

Cola c;

```
boolean libre = true;
sem mutex = 1;
sem espera[P] = ([P], 0);
```

Process Persona [id: 0..P-1] {

```
    int idAux;
    P (mutex);
    if not libre {
        c.push(id);
        V (mutex);
        P (espera[id]);
    } else {
        libre = false;
        V (mutex);
    }
    UsarTerminal();
    P (mutex);
    if c.isEmpty() {
        libre = true;
    } else {
        c.pop(idAux);
        V (espera[idAux]);
    }
}
```

```
v (mutex);
}
```

b)

```
Cola terminalesLibres;
Cola esperaTerminal;
sem mutexLibres = 1;
sem mutexEspera = 1;
sem espera[P] = ([P], 0);
int vecTerminal [P];
```

```
Process Persona [id: 0..P-1] {
    int terminalLibre, siguiente;
    P (mutexLibres);
    if not terminalesLibres.isEmpty(){
        terminalesLibre.pop (terminalLibre);
        vecTerminal [id] = terminalLibre;
        v (mutexLibres);
    } else {
        v (mutexLibres);
        P (mutexEspera);
        c.push (id);
        v (mutexEspera);
        P (espera[id]);
    }
    UsarTerminal (vecTerminal[id]);
    P (mutexLibres);
    P (mutexEspera);
    if esperaTerminal.isEmpty() {
        v (mutexEspera);
        terminalesLibres.push(vecTerminal[id]);
        v (mutexLibres);
    } else {
        v (mutexLibres);
        c.pop (idAux);
        v (mutexEspera);
    }
}
```

```

    vecTerminal [idAux] = vecTerminal[id];
    v (espera[idAux]);
}
}

```

2.

```

Process Persona [id: 0..N-1] {
    int edad = ...;
    Maquina.llegar(id, edad);
    Votar();
    Maquina.liberar();
}

Process Autoridad {
    for i: 0 .. N-1 {
        Maquina.darPermiso();
    }
}

```

```

Monitor Maquina {
    ColaOrdenada c;
    cond espera[N];
    cond hayPedido, esperaUso;

    Procedure llegar (id: in int, edad: in int) {
        c.pushOrdenado(id, edad);
        signal (hayPedido);
        wait (espera[id]);
    }
}

```

```

Procedure darPermiso() {
    if c.isEmpty() wait (hayPedido);
    c.pop(id);
    signal (espera[id]);
    wait (esperaUso);
}

```

```
Procedure liberar() {  
    signal (esperaUso);  
}  
}
```

Primera Fecha 11-10-22

1. Resolver con *SEMÁFOROS* el siguiente problema. En una planta verificadora de vehículos, existen 7 *estaciones* donde se dirigen 150 vehículos para ser verificados. Cuando un vehículo llega a la planta, el *coordinador* de la planta le indica a qué estación debe dirigirse. El coordinador selecciona la estación que tenga menos vehículos asignados en ese momento. Una vez que el vehículo sabe qué estación le fue asignada, se dirige a la misma y espera a que lo llamen para verificar. Luego de la revisión, la estación le entrega un comprobante que indica si pasó la revisión o no. Más allá del resultado, el vehículo se retira de la planta. *Nota:* maximizar la concurrencia.

2. Resolver con *MONITORES* el siguiente problema. En un sistema operativo se ejecutan 20 *procesos* que periódicamente realizan cierto cómputo mediante la función *Procesar()*. Los resultados de dicha función son persistidos en un archivo, para lo que se requiere de acceso al subsistema de E/S. Sólo un proceso a la vez puede hacer uso del subsistema de E/S, y el acceso al mismo se define por la prioridad del proceso (menor valor indica mayor prioridad).

1.

Cola c;

```
Cola colaEsperaAtencion [7];
int vecContador [7] = ([7], 0);
int estacionAsignada [150];
boolean pasaRevision [150];
sem esperandoEstacion [150] = ([150], 0);
sem hayPedidoEstacion = 0;
sem mutexColaEspera = 1;
sem mutexMinimo [7] = ([7], 1);
sem mutexAtencion [7] = ([7], 1);
sem hayPedidoAtencion [7] = ([7], 0);
sem esperandoAtencion [150] = ([150], 0);
sem sali [7] = ([7], 0);
```

Process Coordinador {

```
int id;
for int i: 0..149 {
    P (hayPedidoEstacion);
    int estacionMin = min (vecContador);
    P (mutexMinimo[estacionMin]);
    vecContador [estacionMin]++;
    V (mutexMinimo[estacionMin]);
    P (mutexColaEspera);
    c.pop (id);
    V (mutexColaEspera);
    estacionAsignada [id] = estacionMin;
```

```

    V (esperandoEstacion [id]);
}
}

Process Vehiculo [id: 0..149] {
    P (mutexColaEspera);
    c.push (id);
    V (mutexColaEspera);
    V (hayPedidoEstacion);
    P (esperandoEstacion [id]);
    int estAsignada = estacionAsignada[id]; // Estacion asignada para ir
    P (mutexAtencion[estAsignada]);
    colaEsperaAtencion[estAsignada].push(id);
    V (mutexAtencion[estAsignada]);
    V (hayPedidoAtencion [estAsignada]);
    P (esperandoAtencion [id]);
    P (esperandoAtencion [id]);
    boolean res = pasaRevision [id];
    V (sali[estAsignada]);
}

Process Estacion [id: 0..6] {
    int idAux;
    while (true) {
        P (hayPedidoAtencion [id]);
        P (mutexAtencion [id]);
        colaEsperaAtencion[id].pop (idAux);
        V (mutexAtencion [id]);
        P (mutexMinimo[id]);
        vecContador[id]--;
        V (mutexMinimo[id]);
        V (esperandoAtencion [idAux]); // Le aviso que puede pasar
        pasaRevision[idAux] = evaluarSituacion();
        V (esperandoAtencion [idAux]); // Le avisa que se puede retirar
        P (sali[id]); // Espera a que se retire
    }
}

```

```
}
```

2.

```
Process Proceso [id: 0..19] {
    int prioridad;
    Subsistema.usar(id, prioridad);
    // Guarda los datos
    Subsistema.salir();
}
```

```
Monitor Subsistema {
```

```
    boolean libre = true;
    int esperando = 0;
    cond vecEspera[N];
```

```
Procedure usar (id: in int, prioridad: in int) {
```

```
    if not libre {
        cola.pushOrdenado(id, prioridad);
        esperando++;
        wait (vecEspera[id]);
    } else {
        libre = false;
    }
}
```

```
Procedure salir() {
```

```
    int id;
    if esperando > 0 {
        esperando--;
        cola.pop (id);
        signal(vecEspera[id]);
    } else {
        libre = true;
    }
}
```

Primera Fecha 24-06-2020**Parcial Práctico de MC - Primera Fecha - 24/06/2020**

1. Resolver con **SEMÁFOROS** el siguiente problema. En una empresa hay **UN Coordinador** y **30 Empleados** que formarán 3 grupos de 10 empleados cada uno. Cada grupo trabaja en una sección diferente y debe realizar 345 unidades de un producto. Cada **empleado** al llegar se dirige al **coordinador** para que le indique el número de grupo al que pertenece y una vez que conoce este dato comienza a trabajar hasta que se han terminado de hacer las 345 unidades correspondientes al grupo (cada unidad es hecha por un único empleado). Al terminar de hacer las 345 unidades los 10 empleados del grupo se deben juntar para retirarse todos juntos. El coordinador debe atender a los empleados de acuerdo al orden de llegada para darle el número de grupo (*a los 10 primeros que lleguen se le asigna el grupo 1, a los 10 del medio el 2, y a los 10 últimos el 3*). Cuando todos los grupos terminaron de trabajar el coordinador debe informar (imprimir en pantalla) el empleado que más unidades ha realizado (si hubiese más de uno con la misma cantidad máxima debe informarlos a todos ellos). **Nota:** maximizar la concurrencia; suponga que existe una función **Generar** que simula la elaboración de una unidad de un producto.

2. Resolver con **MONITORES** la siguiente situación. En la guardia de traumatología de un hospital trabajan **5 médicos** y **una enfermera**. A la guardia acuden **P Pacientes** que al llegar se dirigen a la enfermera para que le indique a qué médico se debe dirigir y cuál es su gravedad (entero entre 1 y 10); cuando tiene estos datos se dirige al médico correspondiente y espera hasta que lo termine de atender para retirarse. Cada médico atiende a sus pacientes en orden de acuerdo a la gravedad de cada uno. **Nota:** maximizar la concurrencia.

1.

```

int numGrupo [30];
int cantUnidadesGrupo [3] = ([3], 0);
int barreraGrupo [3] = ([3], 0);
int productosEmpleado [30] = ([30], 0);
sem colaAsigGrupo = 1;
sem llegadaEmpleado = 0;
sem finProduccion = 0;
sem mutexCantGrupo [3] = ([3], 1);
sem mutexContadorGrupo [3] = ([3], 1);
sem barreraGrupo [3] = ([3], 0);
sem esperaNumGrupo[30] = ([30], 0);
Cola colaEsperaGrupo;

```

Procedure Empleado [id: 0..29] {

```

P(colaAsigGrupo);
colaEsperaGrupo.push(id);
V(colaAsigGrupo);

```

```

V(llegadaEmpleado);
P(esperaNumGrupo[id]);
int numGrupo = numGrupo[id];
P(mutexCantGrupo[numGrupo]);
while (cantUnidadesGrupo[numGrupo] < 345) {
    cantUnidadesGrupo[numGrupo]++;
    V(mutexCantGrupo[numGrupo]);
    // Hace la unidad
    productosEmpleado[id]++;
    P(mutexCantGrupo[numGrupo]);
}
V(mutexCantGrupo[numGrupo]);
P(mutexContadorGrupo[numGrupo]);
cantUnidadesGrupo[numGrupo]++;
if cantUnidadesGrupo[numGrupo] == 10 {
    for int i: 0..9 {
        V(barreraGrupo[numGrupo]);
    }
    V(finProduccion);
}
V(mutexContadorGrupo[numGrupo]);
P(barreraGrupo[numGrupo]);
}

```

```

Process Coordinador {
    int i, j, idAux;
    for i: 0 .. 2 {
        for j: 0..9 {
            P(llegadaEmpleado);
            P(colaAsigGrupo);
            colaEsperaGrupo.pop(idAux);
            V(colaAsigGrupo);
            numGrupo[idAux] = i;
            V(esperaNumGrupo[idAux]);
        }
    }
}

```

```

for i: 0..2 {
    P (finProduccion);
}
Cola aux = obtenerMaximos(productosEmpleado); // Obtiene los id max
while (not aux.isEmpty()) {
    writeln(c.pop(id));
}
}

```

2.

```

Process Paciente [id: 0..P-1] {
    int medico;
    MostradorEnfermera.llegar (id, gravedad, medico);
    Consultorio[medico].atender(id, gravedad);
    // Se atiende
    Consultorio[medico].salir();
}

```

```

Process Medico [id: 0..4] {
    while (true) {
        Consultorio[id].siguiente();
    }
}

```

```

Process Enfermera {
    for i: 0 .. P-1 {
        MostradorEnfermera.asignarTurno();
    }
}

```

```

Monitor MostradorEnfermera {
    int esperando = 0;
    cond hayPedido, esperaDatos;
    int gravedad, medico;
    int vecDiml [5] = ([5], 0);
}

```

```

Procedure llegar (grav: out int, med: out int) {
    esperando++;
    signal (hayPedido);
    wait (esperaDatos);
    grav = gravedad;
    med = medico;
}

```

```

Procedure asignarTurno() {
    if esperando == 0 wait (hayPedido);
    esperando--;
    gravedad = ...;
    medico = min (vecDiml);
    vecDiml[medico]++;
    signal (esperaDatos);
}

```

```

Procedure termine (id: in int) {
    vecDiml[id]--;
}

```

```

Monitor Consultorio [id: 0..4] {
    ColaOrdenada c;
    cond hayPedido, esperaUso;
    cond espera[P];
}

```

```

Procedure atender (id: in int, gravedad: in int) {
    c.pushOrdenado(id, gravedad);
    signal (hayPedido);
    wait (espera[id]);
}

```

```

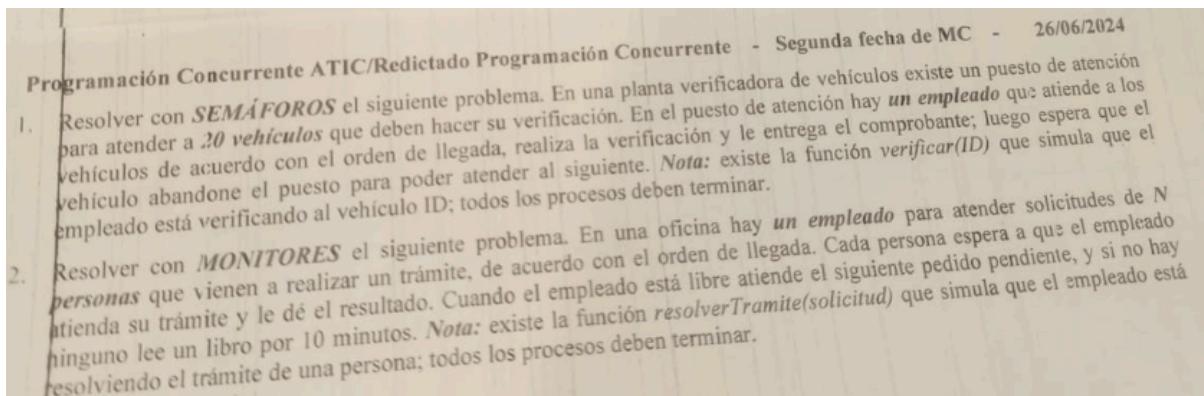
Procedure siguiente () {
    if c.isEmpty() wait (hayPedido);
    c.pop(id);
}

```

```
    signal (espera[id]);
    wait (esperaUso);
    MostradorEnfermera.termino (id);
}

Procedure salir () {
    signal (esperaUso);
}

}
```

Recuperatorios**Segunda Fecha MC 26/06/2024**

1.

Cola c;

```
sem llegue = 0;
sem salida = 0;
sem mutexCola = 1;
sem esperaAtencion[20] = ([20], 0);
text vecComprobantes [20];
```

Process Empleado {

```
int id;
for int i: 0..19 {
    P (llegue);
    P (mutexCola);
    c.pop (id);
    V (mutexCola);
    V (esperaAtencion[id]); // Llama al vehículo
    text comprobante = verificar (id);
    vecComprobantes [id] = comprobante;
    V (esperaAtencion[id]); // Le avisa que se puede retirar
    P (salida); // Espera a que se retire
}
```

Process Vehiculo [id: 0..19] {

```
P (mutexCola);
c.push (id);
```

```

    v (mutexCola);
    v (llegue);
    P (esperaAtencion[id]); // Espera llamado
    P (esperaAtencion[id]); // Espera asignación de comprobante
    text comprobante = vecComprobantes[id];
    v (salida); // Se retira
}

```

2.

```

Process Empleado {
    text solicitud;
    int id;
    int cant = 0;
    while (cant < N) {
        Oficina.sig(id, solicitud);
        if (solicitud = null) sleep (10min);
        else {
            text res = resolverTramite (solicitud);
            Oficina.res (id, res);
            cant++;
        }
    }
}

```

```

Process Persona [id: 0 .. N-1] {
    text solicitud, res;
    Oficina.llegar (id, solicitud, res);
}

```

```

Monitor Oficina {
    Cola c;
    cond hayResultado;
    text resultados [N];

```

```

Procedure llegar (id: in int, sol: in text, res: out text) {
    c.push(id, sol);
}

```

```
    wait (hayResultado);
    res = resultados [id];
}

Procedure sig (id: out int, res: out text) {
    if c.isEmpty() res = null;
    else c.pop (id, res);
}

Procedure res (id: in int, res: in text) {
    resultados [id] = res;
    signal (hayResultado);
}
}
```

Segunda Fecha 4-12-23

1. Un sistema debe validar un conjunto de 10000 transacciones que se encuentran disponibles en una estructura de datos. Para ello, el sistema dispone de 7 workers, los cuales trabajan colaborativamente validando de a 1 transacción por vez cada uno. Cada validación puede tomar un tiempo diferente y para realizarla los workers disponen de la función Validar(t), la cual retorna como resultado un número entero entre 0 al 9. Al finalizar el procesamiento, el último worker en terminar debe informar la cantidad de transacciones por cada resultado de la función de validación. Nota: maximizar la concurrencia.

- 2) En una empresa trabajan 20 vendedores ambulantes que forman 5 equipos de 4 personas cada uno (cada vendedor conoce previamente a que equipo pertenece). Cada equipo se encarga de vender un producto diferente. Las personas de un equipo se deben juntar antes de comenzar a trabajar. Luego cada integrante del equipo trabaja independientemente del resto vendiendo ejemplares del producto correspondiente. Al terminar cada integrante del grupo debe conocer la cantidad de ejemplares vendidos por el grupo. Nota: maximizar la concurrencia.

1.

```
sem mutexCola = 1;
sem mutexBarrera = 1;
sem contador [10] = ([10], 1);
int vectorContador [10] = ([10], 0);
int cant = 0;
```

```
Process Worker [id: 0..6] {
    text transaccion;
    int i;
    P (mutexCola);
    while not c.isEmpty() {
        c.pop(transaccion);
        V (mutexCola);
        int res = Validar(transaccion);
        P (contador[res]);
        vectorContador[res]++;
        V (contador[res]);
        P (mutexCola);
    }
    V (mutexCola);
    P (mutexBarrera);
    cant++;
    if (cant == 7) {
        for i: 0..9 {
            writeln(i, ' ', vectorContador[i]);
    }
}
```

```

        }
    }
    v (mutexBarrera);
}

2)
Process Vendedor [id: 0..19] {
    int equipo = ...;
    int cant = 0;
    int cantGrupo;
    Equipo[equipo].llegar();
    cant = VenderProductos();
    Equipo[equipo].fin(cant, cantGrupo);
}

Monitor Equipo [id: 0..4] {
    cond espera;
    int total = 0;
    int cant = 0;

    Procedure llegar() {
        cant++;
        if (cant = 4) {
            signal_all(espera);
            cant = 0;
        } else {
            wait (espera);
        }
    }

    Procedure fin (cantProd: in int, cantGrupo: out int) {
        total += cantProd;
        cant++;
        if (cant = 4) {
            signal_all(espera);
        } else {
    
```

```
    wait(espera);
}
cantGrupo = total;
}
}
```

Tercera Fecha 18-12-23*ejercicios planteados.*

1. Resolver con **SEMÁFOROS** el siguiente problema. Se debe simular el uso de una máquina expendedora de gaseosas con capacidad para 100 latas por parte de U usuarios. Además existe un repositor encargado de reponer las latas de la máquina. Los usuarios usan la máquina según el orden de llegada. Cuando les toca usarla, sacan una lata y luego se retiran. En el caso de que la máquina se quede sin latas, entonces le debe avisar al repositor para que cargue nuevamente la máquina en forma completa. Luego de la recarga, saca una lata y se retira. **Nota:** maximizar la concurrencia; mientras se reponen las latas se debe permitir que otros usuarios puedan agregarse a la fila.

- 2) Resolver el siguiente problema con **MONITORES**. En una montaña hay 30 escaladores que en una parte de la subida deben utilizar un único paso de a uno a la vez y de acuerdo al orden de llegada al mismo. **Nota:** sólo se pueden utilizar procesos que representen a los escaladores; cada escalador usa sólo una vez el paso.

1.

```
sem mutex = 1;
sem hayQueReponer = 0;
sem hayLatas = 0;
sem espera[U] = ([U], 0);
Cola c;
int cantLatas = 100;
boolean libre = true;
```

```
Process Usuario [id: 0..U-1] {
    int idAux;
    P (mutex);
    if not libre {
        c.push(id);
        V (mutex);
        P (espera[id]);
    } else {
        libre = false;
        V (mutex);
    }
    if cantLatas == 0 {
        V (hayQueReponer);
        P (hayLatas);
    }
    cantLatas--;
    P (mutex);}
```

```

if c.isEmpty() {
    libre = true;
} else {
    c.pop(idAux);
    v (espera[idAux]);
}
v (mutex);
}

```

```

Process Repositor {
    while (true) {
        P (hayQueReponer);
        cantLatas = 100;
        v (hayLatas);
    }
}

```

2)

```

Process Escalador [id: 0..29] {
    Montaña.llegar();
    // Cruza la montaña
    Montaña.salir();
}

```

```

Monitor Montaña {
    cond espera;
    boolean libre = true;
    int esperando = 0;

    Procedure llegar() {
        if not libre {
            esperando++;
            wait (esperando);
        } else {
            libre = false;
        }
    }
}

```

}

```
Procedure salir() {
    if esperando > 0 {
        esperando--;
        signal (esperando);
    } else {
        libre = true;
    }
}
```

Parcial 13-12-22

1. Resolver con **SEMÁFOROS** el siguiente problema. En una fábrica de muebles trabajan 50 *empleados*. A llegar, los empleados forman 10 grupos de 5 personas cada uno, de acuerdo al orden de llegada (los 5 primeros en llegar forman el primer grupo, los 5 siguientes el segundo grupo, y así sucesivamente). Cuando un grupo se ha terminado de formar, todos sus integrantes se ponen a trabajar. Cada grupo debe armar M muebles (cada mueble es armado por un solo empleado); mientras haya muebles por armar en el grupo los empleados los irán resolviendo (cada mueble es armado por un solo empleado). *Nota:* Cada empleado puede tardar distinto tiempo en armar un mueble. Sólo se pueden usar los procesos “*Empleado*”, y todos deben terminar su ejecución. Maximizar la concurrencia.
2. Resolver con **MONITORES** el siguiente problema. En un comedor estudiantil hay un horno microondas que debe ser usado por E *estudiantes* de acuerdo con el orden de llegada. Cuando el estudiante accede al horno, lo usa y luego se retira para dejar al siguiente. *Nota:* cada Estudiante una sólo una vez el horno; los únicos procesos que se pueden usar son los “*estudiantes*”.

1.

```
sem llegue = 1;
sem esperaGrupo [10] = ([10], 0);
sem mutexGrupo [10] = ([10], 1);
int cant = 0;
int grupo = 0;
int cantMuebles [10] = ([10], 0);
```

```
Process Empleado [id: 0..49] {
    int grupoAsignado;
    P (llegue);
    grupoAsignado = grupo;
    cant++;
    if cant MOD 5 == 0 {
        for int i: 0 .. 4 {
            V (esperaGrupo[grupoAsignado]);
        }
        grupo++;
    }
    V (llegue);
    P (esperaGrupo[grupoAsignado]);
    P (mutexGrupo[grupoAsignado]);
    while (cantMuebles[grupoAsignado] < M) {
        cantMuebles[grupoAsignado]++;
        V (mutexGrupo[grupoAsignado]);
        // Se arma el mueble -> Tiempo random
        P (mutexGrupo[grupoAsignado]);
    }
}
```

```
v (mutexGrupo[grupoAsignado]);
}
```

2.

```
Process Estudiante [id: 0..E-1] {
    Microondas.usar();
    // Calienta la comida
    Microondas.salir();
}
```

```
Monitor Microondas {
```

```
    int esperando = 0;
    boolean libre = true;
    cond cola;
```

```
    Procedure usar() {
        if not libre {
            esperando++;
            wait (cola);
        } else {
            libre = false;
        }
    }
```

```
    Procedure salir() {
        if esperando > 0 {
            esperando--;
            signal (cola);
        } else {
            libre = true;
        }
    }
```

```
}
```

Segunda Fecha del Parcial - 1/7/2021 ATIC

1. Resolver el siguiente problema con MONITORES. 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 cuando los 45 han llegado (es el mismo enunciado para todos). La profesora se encarga de ir corrigiendo los exámenes de acuerdo al 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 dé la nota. **Nota:** maximizar la concurrencia; todos los procesos deben terminar su ejecución; suponga que la profesora tienen una función *corregirExamen* que recibe un examen y devuelve un entero con la nota.

```
Process Alumno [id: 0..44] {
```

```
    int nota;
```

```
    text examen;
```

```
    Aula.Llegada (examen);
```

```
    // Rendir examen
```

```
    Aula.Entrega (id, examen, nota);
```

```
}
```

```
Process Profesora {
```

```
    int i, idAlumno, nota;
```

```
    text examen;
```

```
    for i: 1 .. 45 {
```

```
        Aula.Examen (idAlumno, examen);
```

```
        nota = corregirExamen(examen);
```

```
        Aula.Corregido (idAlumno, nota);
```

```
}
```

```
}
```

```
Process Preceptor {
```

```
    text enunciado = ...;
```

```
    Aula.DarEnunciado (enunciado);
```

```
}
```

```
Monitor Aula {
```

```
    int cant = 0, notas [45];
```

```
    text enunciado;
```

```
    cola C;
```

```
    cond elnicio, ePreceptor, eProfesora, eNota;
```

```

Procedure Llegada (E: out text) {
    cant++;
    if (cant == 45) signal (ePreceptor);
    wait (eInicio);
    E = enunciado;
}

Procedure DarEnunciado (E: in text) {
    if (cant < 45) wait (ePreceptor);
    enunciado = E;
    signal_all (eInicio);
}

Procedure Entrega (id: in int, e: in text, N: out int) {
    C.push(id, e);
    signal (eProfesora);
    wait (eNota);
    N = notas [id];
}

Procedure Examen (idAlumno: out int, E: out text) {
    if c.isEmpty() wait (eProfesora);
    c.pop (idAlumno, E);
}

Procedure Corregido (id: in int, nota: in int) {
    notas[id] = nota;
    signal (eNota);
}

```

Tercera Fecha del Parcial - 15/7/2021 ATIC

1. Resolver el siguiente problema con SEMÁFOROS. En un vacunatorio hay un empleado de salud para vacunar a 50 personas. El empleado de salud atiende a las personas de acuerdo al orden de llegada y de a 5 personas a la vez, es decir que cuando está libre debe esperar a que haya al menos 5 personas esperando, luego vacuna a las 5 primeras personas, y al terminar las deja ir para esperar por otras 5. Cuando ha atendido a las 50 personas el empleado de salud se retira. *Nota:* todos los procesos deben terminar su ejecución; asegurarse de no realizar *Busy Waiting*, suponga que el empleado tienen una función *VacunarPersona()* que simula que el empleado está vacunando a UNA persona.

1.

```
sem mutex = 1, atender = 0, listo = 0;
sem espera [50] = ([50], 0), irse [50] = ([50], 0);
cola c;
int cant = 0;
```

```
Process Persona [id: 0..49] {
    P (mutex);
    c.push (id);
    cant++;
    if (cant MOD 5 == 0) {
        V (atender);
    }
    V (mutex);
    P (espera [id]); // Espera a ser llamada
    V (listo); // Listo para vacunacion
    P (espera [id]); // Espera a que terminen de vacunarlo
    P (irse [id]); // Se retira
}
```

```
Process Empleado {
    int i, j;
    int actuales [5];
    for i: 0..9 {
        P (atender); // Espera al grupo de 5 personas
        P (mutex);
        for j: 0..4 {
            c.pop(actuales[j]);
        }
    }
}
```

```
v (mutex);
for j: 0..4 {
    v (espera[actuales[j]]); // Llama a un paciente
    P (listo); // Espera confirmacion
    VacunarPersona (actuales[j]);
    v (espera[actuales[j]]); // Libera al vacunado
}
for j: 0..4 {
    v (irse[actuales[j]]); // Espera a que terminen 5 personas
}
}
```

Primera Fecha del Parcial - 8/2021

1. Resolver con **SEMAFOROS** el siguiente problema. Simular un examen técnico para concursos Nodocentes en la Facultad, en el mismo participan **100 personas** distribuidas en **4 concursos** (25 personas en cada concurso) con un coordinador en cada una de ellos. Cada persona ya conoce en qué concurso participa. El coordinador de cada concurso espera hasta que lleguen las 25 personas correspondientes al mismo, les entrega el examen a resolver (el mismo para todos los de ese concurso), y luego corrige los exámenes de esas 25 personas de acuerdo al orden en que van entregando. Cada persona al llegar debe esperar a que su coordinador (el que corresponde a su concurso) le dé el examen, lo resuelve, lo entrega para que su coordinador lo evalúe y espera hasta que le deje la nota para luego retirarse. **Nota:** maximizar la concurrencia; sólo usar los procesos que representen a las personas y a los coordinadores; todos los procesos deben terminar.

```
sem mutexConcurso [4] = ([4], 1);
sem mutexEntrega [4] = ([4], 1);
sem entregarExamen [4] = ([4], 0);
sem esperarExamen [4] = ([4], 0);
sem examenEntregado [4] = ([4], 0);
sem esperaNota [100] = ([100], 0);
int vectorContador [4] = ([4, 0]);
int vecNotas [100] = ([100], 0);
Cola colaEspera [4];
```

```
Process Persona [id: 0.99] {
    int concurso = ...;
    P (mutexConcurso[concurso]);
    vectorContador [concurso]++;
    if (vectorContador[concurso] == 25) {
        V (entregarExamen[concurso]);
    }
    V (mutexConcurso[concurso]);
    P (esperarExamen[concurso]);
    // Hacer examen
    P (mutexEntrega[concurso]);
    colaEspera[concurso].push(id);
    V (mutexEntrega[concurso]);
    V (examenEntregado[concurso]);
    P (esperaNota[id]);
    int nota = vecNotas[id];
}
```

```
Process Coordinador [id: 0..3] {
    int i, idAux;
    P (entregarExamen[id]);
    for i: 0..24 {
        V (esperarExamen[id]);
    }
    for i: 0..24 {
        P (examenEntregado[id]);
        P (mutexEntrega[id]);
        colaEspera[id].pop(idAux);
        V (mutexEntrega[id]);
        vecNotas[idAux] = CorregirExamen();
        V (esperaNota[idAux]);
    }
}
```

Segunda Fecha del Parcial - 25/11/2021

1. Resolver con **SEMÁFOROS** el siguiente problema. Se debe simular el funcionamiento de una mesa de votación en una elección donde hay 2 listas candidatas (A y B). A la misma acuden **300 personas** para votar en el cuarto oscuro (se dispone de una función *obtenerVoto()* que retorna la lista a votar: A o B). Además está el **Presidente de la Mesa** que es quien habilita a las personas a pasar al cuarto oscuro de a una a la vez de acuerdo al orden de llegada; y cuando todas las personas han votado determina cual es la lista ganadora (A o B). **Nota:** sólo se deben usar los procesos que representen a las personas y al Presidente de Mesa.
2. Resolver con **MONITORES** el siguiente problema. Se debe modelar el funcionamiento de un Complejo de Canchas de Paddle que posee 10 canchas, adonde acuden **40 personas** para jugar. Cuando una persona llega busca el número de cancha a la cual debe ir, y se dirige a ella. Cuando a una cancha han llegado las 4 personas correspondientes, se juega el partido durante 60 minutos, y al terminar el mismo las 4 personas se retiran. El número de cancha se asigna a los jugadores de acuerdo al orden de llegada (los 4 primeros a la cancha 1, los siguientes 4 a la 2 y así sucesivamente). **Nota:** maximizar la concurrencia.

1.

```
sem mutex = 1;
sem hayEspera = 0;
sem sali = 0;
sem espera [300] = ([300], 0);
int vecContador['A'..'B'] = (['A'..'B'], 0);
Cola c;
```

```
Process Persona [id: 0.299] {
    P (mutex);
    c.push (id);
    V (mutex);
    V (hayEspera);
    P (espera[id]); // Espera a ser llamado
    // Vota
    vectorContador[obtenerVoto()]++;
    V (sali); // Avisa que salio
}
```

```
Process Presidente {
    int id;
    for int i: 0..299 {
        P (hayEspera);
        P (mutex);
        c.pop (id);
        V (mutex);
        V (espera[id]); // Avisa que puede pasar
```

```

    P (sali); // Espera salida
}
if vectorContador['A'] > vectorContador['B'] {
    writeln('A gano');
} else if vectorContador['A'] < vectorContador['B'] {
    writeln('B gano');
} else {
    writeln('Empate');
}
}

```

2.

```

Process Persona [id: 0..39] {
    int numCancha;
    Administracion.llegada(numCancha);
    Cancha[id].jugar();
}

```

```

Process Partido [id: 0..9] {
    Cancha[id].iniciar();
    delay (60min);
    Cancha[id].fin();
}

```

```

Monitor Administracion {
    int cant = 0;
    Procedure llegada (numCancha: out int) {
        numCancha = cant DIV 4;
        cant++;
    }
}

```

```

Monitor Cancha [id: 0..9] {
    int cant = 0;
    cond inicioPartido, finPartido;
}

```

```
Procedure iniciar() {  
    if cant < 4 wait (inicioPartido);  
}  
  
Procedure iniciar() {  
    cant++;  
    if cant == 4 signal (inicioPartido);  
    wait (finPartido);  
}  
  
Procedure fin() {  
    signal_all (finPartido);  
}  
}
```

Más parciales**Semáforos:**

1.

Resolver con **SEMAFOROS** el siguiente problema. En un examen final hay P alumnos y 3 profesores. Cuando todos los alumnos y los profesores han llegado comienza el examen. Para esto uno de los profesores (el primero en llegar) le da el examen a cada alumno. Cada alumno resuelve su examen, lo entrega y espera a que alguno de los profesores lo corrija y le indique la nota. Los profesores corrigen los exámenes respetando el orden en que los alumnos van entregando. **Nota:** maximizar la concurrencia.

```

sem mutexContador = 1;
sem mutexCola = 1;
sem profesor = 0;
sem hayExamen = 0;
sem esperaExamen [P] = ([P], 0);
sem esperaNota[P] = ([P], 0);
Cola c;
int primero = -1;
int cant = 0;
int vecNotas[P];
text examenes [P];

Process Alumno [id: 0..P-1]{
    P (mutexContador);
    cant++;
    if (cant == P+3) {
        V (profesor);
    }
    V (mutexContador);
    P (esperaExamen[id]);
    text examenResuelto = HacerExamen(examenes[id]);
    P (mutexCola);
    c.push(id, examenResuelto);
}

```

```

    V (mutexCola);
    V (hayExamen);
    P (esperaNota[id]);
    int nota = vecNotas[id];
}

```

```

Process Profesor [id: 0..2] {
    int idAux;
    P (mutexContador);
    cant++;
    if (cant == P+3) {
        V (profesor);
    }
    if (primero == -1) {
        primero = id;
    }
    V (mutexContador);
    if (primero == id) {
        P (profesor);
        for int i: 0..P-1 {
            examenes[i] = EntregarExamen();
            V (esperaExamen[i]);
        }
    }
    while (true) {
        P (hayExamen);
        P (mutexCola);
        c.pop(idAux, examen);
        V (mutexCola);
        int nota = CorregirExamen(examen);
        vecNotas[idAux] = nota;
        V (esperandoNota[idAux]);
    }
}

```

2.

Resolver el siguiente problema con **SEMAFOROS**. En una competencia culinaria se juntan **1 jurado** y **C concursantes**. Una vez que todos los concursantes llegaron, el jurado les asigna la receta que deben realizar. A continuación, los concursantes cocinan el plato pedido (a cada uno le lleva un tiempo variable) y lo exhiben ante el jurado en el orden en que van terminando. El jurado asigna un puntaje a cada concursante, el cual lo guarda para su CV.

```
sem mutexCola = 1;
sem barrera = 0;
sem llegue = 0;
sem espera[C] = ([C], 0);
sem esperaPuntaje[C] = ([C], 0);
int cant = 0;
text receta[N];
double puntajes[N];
Cola c;
```

```
Process Jurado {
    int i, id;
    for i: 0 .. C-1 {
        P (barrera);
    }
    for i: 0 .. C-1 {
        receta[id] = DarReceta();
        V (espera[id]);
    }
    for i: 0 .. C-1 {
        P (llegue);
        P (mutexCola);
        c.pop(id);
        V (mutexCola);
        puntajes[id] = EvaluarReceta();
        V (esperaPuntaje[id]);
    }
}
```

```
    }  
}  
  
Process Concursante [id: 0..C-1] {  
    V (barrera);  
    P (espera[id]);  
    CocinarReceta(receta[id]);  
    P (mutexCola);  
    c.push(id);  
    V (mutexCola);  
    V (llegue);  
    P (esperaPuntaje[id]);  
    double puntaje = puntajes[id];  
}
```

3.

Resolver con **SEMÁFOROS** el siguiente problema. Se debe simular el uso de UNA máquina expendedora de agua con capacidad para 20 botellas por parte de U usuarios. Además existe **un repositor** encargado de reponer las botellas de la máquina. Cuando un usuario llega 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:** maximizar la concurrencia; mientras se reponen las botellas se debe permitir que otros usuarios se encolen.

```
sem mutex = 1;
sem hayBotellas = 0;
sem sinBotellas = 0;
sem espera [U] = ([U], 0);
boolean libre = true;
int cantBotellas = 20;
Cola c;
```

```
Process Usuario [id: 0 .. U-1] {
```

```
    int idAux;
    P (mutex);
    if not libre {
        c.push(id);
        V (mutex);
        P (espera[id]);
    } else {
        libre = false;
        V (mutex);
    }
    if cantBotellas == 0 {
        V (sinBotellas);
        P (hayBotellas);
    }
}
```

```
cantBotellas--;
P (mutex);
if c.isEmpty() {
    libre = true;
} else {
    c.pop(idAux);
    V (espera[idAux]);
}
V (mutex);
}
```

```
Process Repositor {
    while (true) {
        P (sinBotellas);
        cantBotellas = 20;
        V (hayBotellas);
    }
}
```

4.

Resolver con **Semáforos** el siguiente problema. En una clínica hay un médico que debe atender a 20 pacientes de acuerdo al turno de cada uno de ellos (*no puede atender al paciente con turno $i+1$ si aún no atendió al que tiene turno i*). Cada paciente ya conocen su turno al comenzar (*valor entero entre 0 y 19, o entre 1 y 20, como les resulte más cómodo*), al llegar espera hasta que el médico lo llame para ser atendido, se dirige al consultorio y luego espera hasta que el médico lo termine de atender para retirarse. **Nota:** los únicos procesos que se pueden usar son los que representen a los pacientes y al médico; se debe asegurar que nunca haya más de un paciente en el consultorio; no se puede usar el ID del proceso como turno.

```
sem espera [20] = ([20], 0);
sem adentro, finConsulta, afuera = 0;
```

```
Process Paciente [id: 0..19] {
    int turno = ...;
    P (espera[miTurno]); // Espera llamado
    V (adentro); // Entrá a consultorio
    P (finConsulta); // Esperá a que lo atiendan
    V (afuera); // Se retira
}
```

```
Process Medico {
    for int i: 0 .. 19 {
        V (espera[miTurno]); // Llama al paciente
        P (adentro); // Esperá que entre
        AtenderPaciente();
        V (finConsulta); // Avisa que se puede retirar
        P (afuera); // Esperá a que el paciente salga
    }
}
```

5.

Resolver con **Semáforos** el siguiente problema. En una casa de ropa, hay una modista que debe atender a 15 clientes que acuden para tomarse las medidas para realizarse un traje. Los clientes son atendidos de acuerdo al turno que tiene asignado cada uno de ellos (*no se puede atender al cliente con turno $i+1$ si aún no atendió al que tiene turno i*). Cada cliente ya conocen su turno al comenzar (*valor entero entre 0 y 14, o entre 1 y 15, como les resulte más cómodo*), al llegar espera hasta que la modista lo llame para ser atendido, se dirige al vestidor y espera hasta que la modista termina de tomarle las medidas para luego retirarse. **Nota:** los únicos procesos que se pueden usar son los que representen a los clientes y la modista; se debe asegurar que nunca haya más de un cliente en el vestidor; no se puede usar el ID del proceso como turno.

```
sem espera [15] = ([15], 0);
sem adentro, fin, afuera = 0;
```

```
Process Cliente [id: 0..14] {
```

```
    int turno = ...;
    P (espera[miTurno]);
    V (adentro);
    P (fin);
    V (afuera);
```

```
}
```

```
Process Modista {
```

```
    for int i: 0 .. 14 {
        V (espera[miTurno]);
        P (adentro);
        AtenderCliente();
        V (fin);
        P (afuera);
```

```
}
```

```
}
```

6.

Resolver con **Semáforos** el siguiente problema. En una sucursal de un banco, hay un empleado que debe atender a 20 clientes que acuden para averiguar sobre los productos del banco. Los clientes son atendidos de acuerdo al turno que tiene asignado cada uno de ellos (*no se puede atender al cliente con turno $i+1$ si aún no atendió al que tiene turno i*). Cada cliente ya conocen su turno al comenzar (*valor entero entre 0 y 19, o entre 1 y 20, como les resulte más cómodo*), al llegar espera hasta que la empleado lo llame para ser atendido, se dirige al escritorio y espera hasta que empleado termina de atenderlo para luego retirarse. **Nota:** los únicos procesos que se pueden usar son los que representen a los clientes y al empleado; se debe asegurar que nunca haya más de un cliente en el escritorio; no se puede usar el ID del proceso como turno.

```
sem espera [20] = ([20], 0);
sem adentro, fin, afuera = 0;
```

```
Process Cliente [id: 0..19] {
    int turno = ...;
    P (espera[miTurno]);
    V (adentro);
    P (fin);
    V (afuera);
}
```

```
Process Empleado {
    for int i: 0 .. 19 {
        V (espera[miTurno]);
        P (adentro);
        AtenderCliente();
        V (fin);
        P (afuera);
    }
}
```

7.

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.

```

sem mutexCaja = 1;
sem hayPedidoCaja = 0;
sem esperaCaja[N] = ([N], 0);
sem mutexVerificacion = 1;
sem hayVerificacion = 0;
sem esperaVerificacion[N] = ([N], 0);
text recibos[id];
Cola caja;

Process Vehiculo [id: 0..N-1] {
    boolean auto = ...; // True si es auto
    if auto {
        P (mutexCaja);
        caja.push (id);
        V (mutexCaja);
        V (hayPedidoCaja);
        P (esperaCaja[id]);
        text recibo = recibos[id];
    }
    P (mutexVerificacion);
    verificacion.pushOrdenado(id, auto);
    V (mutexVerificacion);
    V (hayVerificacion);
    P (esperaVerificacion[id]);
}

```

```
Process Caja {
    int idAux;
    while (true) {
        P (hayPedidoCaja);
        P (mutexCaja);
        caja.pop (id);
        V (mutexCaja);
        recibos[id] = cobrar();
        V (esperaCaja[id]);
    }
}
```

```
Process AdminVerificacion {
    int idAux;
    while (true) {
        P (hayVerificacion);
        P (mutexVerificacion);
        c.pop(idAux);
        V (mutexVerificacion);
        Verificar (idAux);
        V (esperaVerificacion[idAux]);
    }
}
```

8.

Resolver con **SEMAFOROS** el siguiente problema. Una empresa de turismo posee UN micro con capacidad para 50 personas. Hay un único *vendedor* que atiende a los *C clientes* ($C > 50$) que intentan comprar un pasaje de acuerdo al orden de llegada (suponga que la atención de un cliente tarda un par de minutos), si aún hay lugares disponibles le indica el número de asiento que le tocó. Cada cliente, luego de ser atendidos por el vendedor, se dirige al micro para subir en caso de que le hayan dado un asiento, y en caso contrario se retira sin viajar. El micro espera a que los 50 pasajeros hayan subido para realizar el viaje. **Nota:** maximizar la concurrencia.

```
sem mutexCola = 1;
sem hayPedido = 0;
sem llegue = 0;
sem esperandoResultado[C] = ([C], 0);
boolean vecAsientosPide [C];
int vecAsientos[C];
int cant = 0;
int cantEntradas = 50;
```

```
Process Cliente [id: 0..C-1] {
    P(mutexCola);
    c.push(id);
    V(mutexCola);
    V(hayPedido);
    P (esperandoResultado[id]);
    if vecAsientosPide[id] {
        V(llegue);
    }
}
```

```
Process Vendedor {
    int id;
    int cantPersonas = 0;
```

```
for int i: 0 .. C-1 {  
    P (hayPedido);  
    P (mutexCola);  
    c.pop(id);  
    V (mutexCola);  
    boolean pude = false;  
    if cantPersonas < cantEntradas {  
        pude = true;  
        vecAsientos[id] = cantPersonas;  
        vecAsientosPide[id] = pude;  
        cantPersonas++;  
    }  
    V (esperandoResultado[id]);  
}  
}
```

```
Process Micro {  
    for int i: 0..49 {  
        P(llegue);  
    }  
}
```

9.

Resolver el siguiente problema con **SEMÁFOROS**. En un vacunatorio hay un empleado de salud para vacunar a 50 personas. El empleado de salud atiende a las personas de acuerdo al orden de llegada y de a 5 personas a la vez, es decir que cuando está libre debe esperar a que haya al menos 5 personas esperando, luego vacuna a las 5 primeras personas, y al terminar las deja ir para esperar por otras 5. Cuando ha atendido a las 50 personas el empleado de salud se retira. **Nota:** todos los procesos deben terminar su ejecución; asegurarse de no realizar *Busy Waiting*; suponga que el empleado tienen una función *VacunarPersona()* que simula que el empleado está vacunando a UNA persona.

```
sem mutex = 1, atender = 0, listo = 0;
sem espera [50] = ([50], 0), irse [50] = ([50], 0);
cola c;
int cant = 0;
```

```
Process Persona [id: 0.49] {
    P (mutex);
    c.push (id);
    cant++;
    if (cant MOD 5 == 0) {
        V (atender);
    }
    V (mutex);
    P (espera [id]); // Espera a ser llamada
    V (listo); // Listo para vacunacion
    P (espera [id]); // Espera a que terminen de vacunarlo
    P (irse [id]); // Se retira
}
```

```
Process Empleado {
    int i, j;
    int actuales [5];
    for i: 0..9 {
        P (atender); // Espera al grupo de 5 personas
        P (mutex);
        for j: 0..4 {
            c.pop(actuales[j]);
```

```
}

V (mutex);
for j: 0..4 {
    V (espera[actuales[j]]); // Llama a un paciente
    P (listo); // Espera confirmacion
    VacunarPersona (actuales[j]);
    V (espera[actuales[j]]); // Libera al vacunado
}
for j: 0..4 {
    V (irse[actuales[j]]); // Espera a que terminen 5 personas
}
}
```

10.

Resolver con **SEMÁFOROS** el siguiente problema. Simular un examen escrito que deben rendir **60 alumnos** repartidos en **3 aulas** (20 alumnos en cada aula) con un docente en cada una de ellas. Cada alumno ya tiene asignado el aula en la que debe rendir. El docente de cada aula espera hasta que sus 20 alumnos hayan llegado para darles el enunciado del examen (el mismo para todos los alumnos), y luego les corrige el examen de acuerdo al orden en que van entregando. Cada alumno cuando llega debe esperar a que su docente (el correspondiente a su aula) le dé el enunciado del examen, lo resuelve, lo entrega para que el mismo docente lo corrija y le deje la nota. Cuando el alumno ya tiene su nota se retira. **Nota:** maximizar la concurrencia; sólo usar los procesos que representen a los alumnos y a los docentes; todos los procesos deben terminar.

```
sem aulas[3];
vector text enunciados[60];
sem esperaEnunciados[60];
vector entregas[3]=([3],null);
sem mutexEntregas[3];
vector correcciones[60];
sem mutexCorrecciones[60];
sem esperaFin=0;
sem esperaCorrecciones[60]=([60],0);
```

```
Process Alumno[id: 0..59]{
    int aula=..;
    Correccion correccion;
    Resolucion resolucion;
    v(aulas[aula]);
    P(esperaEnunciado[id]);
    resolucion=resolver();
    P(mutexEntregas[aula]);
    entregas.push(id,resolucion );
    v(mutexEntregas[aula]);
    v(espera[aula]);
    P(esperaCorrecciones[id]);
    correccion=correcciones[id];
}
```

```
Process Docente[id:0..2]{
    int i;
    int sig;
    Correccion correccion;
```

```
for(i=0;i<20;i++){
    P(aulas[id]);
}

int desde=id*20;
int hasta=desde+20-1;
enunciado=generarEnunciado();
for(i=desde..hasta){
    enunciados[i]=enunciado;
    V(esperaEnunciado[i]);
}

for(i=desde..hasta){
    P(espera[id]);
    P(mutexEntregas[id]);
    sig, resolucion=entregas[id].pop();
    V(mutexEntregas[id]);
    correccion=corregir(resolucion);
    correcciones[sig]=correccion;
    V(esperaCorrecciones[sig]);
}

}
```

11.

Resolver con **SEMÁFOROS** el siguiente problema. Simular un examen técnico para concursos Nodocentes en la Facultad, en el mismo participan **100 personas** distribuidas en **4 concursos** (25 personas en cada concurso) con un coordinador en cada una de ellos. Cada persona ya conoce en qué concurso participa. El coordinador de cada concurso espera hasta que lleguen las 25 personas correspondientes al mismo, les entrega el examen a resolver (el mismo para todos los de ese concurso), y luego corrige los exámenes de esas 25 personas de acuerdo al orden en que van entregando. Cada persona al llegar debe esperar a que su coordinador (el que corresponde a su concurso) le dé el examen, lo resuelve, lo entrega para que su coordinador lo evalúe y espera hasta que le deje la nota para luego retirarse. **Nota:** maximizar la concurrencia; sólo usar los procesos que representen a las personas y a los coordinadores; todos los procesos deben terminar.

```
sem mutexConcurso [4] = ([4], 1);
sem mutexEntrega [4] = ([4], 1);
sem entregarExamen [4] = ([4], 0);
sem esperarExamen [4] = ([4], 0);
sem examenEntregado [4] = ([4], 0);
sem esperaNota [100] = ([100], 0);
int vectorContador [4] = ([4, 0]);
int vecNotas [100] = ([100], 0);
Cola colaEspera [4];
```

```
Process Persona [id: 0.99] {
    int concurso = ...;
    P (mutexConcurso[concurso]);
    vectorContador [concurso]++;
    if (vectorContador[concurso] == 25) {
        V (entregarExamen[concurso]);
    }
    V (mutexConcurso[concurso]);
    P (esperarExamen[concurso]);
    // Hacer examen
    P (mutexEntrega[concurso]);
    colaEspera[concurso].push(id);
    V (mutexEntrega[concurso]);
    V (examenEntregado[concurso]);
    P (esperaNota[id]);
    int nota = vecNotas[id];
}
```

```
Process Coordinador [id: 0..3] {
    int i, idAux;
    P (entregarExamen[id]);
    for i: 0..24 {
        V (esperarExamen[id]);
    }
    for i: 0..24 {
        P (examenEntregado[id]);
        P (mutexEntrega[id]);
        colaEspera[id].pop(idAux);
        V (mutexEntrega[id]);
        vecNotas[idAux] = CorregirExamen();
        V (esperaNota[idAux]);
    }
}
```

Monitores:**1.**

Resolver el siguiente problema con **MONITORES**. Simular la atención en una Salita Médica para vacunar contra el coronavirus. Hay **UNA enfermera** encargada de vacunar a **30 pacientes**, cada paciente tiene un turno asignado (valor entero entre 1..30 ya conocido por el paciente). La enfermera atiende a los pacientes de acuerdo al turno que cada uno tiene asignado. Cada paciente al llegar espera a que sea su turno y se dirige al consultorio para que la enfermera lo vacune, y luego se retira. **Nota:** suponer que existe una función *Vacunar()* que simula la atención del paciente por parte de la enfermera. Todos los procesos deben terminar.

Para que quede más clara la explicación supondremos que los turnos son de 0..29

```
Process Paciente [id: 0..29] {
    int turno = ...;
    Mostrador.llegar(turno);
    Consultorio.atenderse();
}
```

```
Process Enfermera {
    for i: 0 .. 29 {
        Mostrador.siguiente();
        Consultorio.sig();
        Vacunar();
        Consultorio.fin();
    }
}
```

```
Monitor Mostrador {
    boolean pacientes [30] = ([30], false);
    cond espera [30];
    cond hayPedido;
```

```

int sig = -1;
Procedure llegar (turno: in int) {
    pacientes[turno] = true;
    if (turno == sig) signal (hayPedido);
    wait (espera[turno]);
}

Procedure siguiente() {
    sig++;
    if (not pacientes[sig]) {
        wait (hayPedido);
    }
    signal (pacientes[sig]);
}
}

Monitor Consultorio {
    boolean llegue = false;
    cond hayConsulta, paciente;

    Procedure atenderse() {
        llegue = true;
        signal (hayConsulta);
        wait (paciente);
        signal (hayConsulta);
    }

    Procedure sig() {
        if (not llegue) {
            wait (hayConsulta);
        }
        llegue = false;
    }

    Procedure fin() {
        signal (paciente);
    }
}

```

```
    wait (hayConsulta);  
}  
}
```

2.

Resolver con **MONITORES** el siguiente problema. Simular el uso de un puente de un solo carril y sentido por donde puede pasar un vehículo a la vez. Hay **N vehículos** donde algunos son autos y otros ambulancias. Los vehículos deben pasar de acuerdo al orden de llegada pero siempre dando prioridad a las ambulancias. **Nota:** suponga que cada vehículo tarda 5 minutos en pasar por el puente.

```

Monitor Puente {
    boolean libre = true;
    int cant = 0;
    int idAux;
    ColaOrdenada c;
    cond espera[N];

    Procedure pasar(id: in int, prioridad: in boolean) {
        if (not libre) {
            cant++;
            c.pushOrdenado(id);
            wait(espera[id]);
        } else {
            libre = false;
        }
    }

    Procedure salir() {
        if cant > 0 {
            cant--;
            c.pop(idAux);
            signal(espera[idAux]);
        } else {
            libre = true;
        }
    }
}

```

```
}

Process Vehiculo [id: 0..N-1] {
    boolean prioridad = ...;
    Puente.pasar(id, prioridad);
    delay (5min);
    Puente.salir();
}
```

3.

Resolver con **MONITORES** el siguiente problema. Se debe simular una carrera a campo traviesa con C corredores donde en la mitad del recorrido hay un puente colgante que puede ser usado por una única persona a la vez. Cuando los C corredores han llegado al punto de partida comienza la carrera. Cuando un corredor llega al puente espera su turno (respetando el orden de llegada al mismo) y lo cruza (suponga que tarda un par de minutos en cruzarlo); y luego continua su carrera hasta llegar a la meta. **Nota:** cada corredor pasa sólo una vez por el puente. Sólo se pueden usar procesos que representen a los corredores.

```
Process Corredor [id: 0..C-1] {
```

```
    Carrera.llegar();
    Puente.pasar(id);
    // Cruza el puente
    Puente.salir();
```

```
}
```

```
Monitores Carrera {
```

```
    int cant = 0;
```

```
    cond espera;
```

```
    Procedure llegar() {
```

```
        cant++;
    
```

```
        if cant == C {
```

```
            signal_all (espera);
```

```
        } else {
```

```
            wait (espera);
```

```
}
```

```
}
```

```
}
```

```
Monitor Puente() {
```

```
    int esperando = 0;
```

```
cond cola;  
boolean libre = true;  
  
Procedure pasar() {  
    if not libre {  
        esperando++;  
        wait (cola);  
    } else {  
        libre = false;  
    }  
}  
  
Procedure salir() {  
    if esperando > 0 {  
        esperando-;  
        signal (cola);  
    } else {  
        libre = true;  
    }  
}
```

4.

Resolver el siguiente problema con **MONITORES**. Simular la atención en un Centro de Vacunación con 8 puestos para vacunar contra el coronavirus. Al Centro acuden **200 pacientes** para ser vacunados, cada uno de ellos ya conoce el puesto al que se debe dirigir. En cada puesto hay **UN empleado** para vacunar a los pacientes asignados a dicho puesto, y lo hace de acuerdo al orden dado por la edad de paciente (cuando está libre atiende al paciente de mayor edad que esté esperando en ese momento en ese puesto). Cada paciente al llegar al puesto que tenía asignado espera a que lo llamen y se dirige a la silla para que el empleado lo vacune, y luego se retira. **Nota:** suponer que existe una función *Vacunar()* que simula la atención del paciente. Suponer que cada puesto tiene asignado 25 pacientes. Todos los procesos deben terminar.

```
Process Paciente [id: 0.199] {
```

```
    int puesto = ...;
    int edad = ...;
    Puesto[puesto].llegar(id, edad);
    Silla[puesto].atenderse();
}
```

```
Process Empleado [id: 0..7] {
```

```
    for int i: 0 .. 24 {
        Puesto[id].sig(id);
        Sala[id].espera();
        Vacunar();
        Sala[id].fin();
    }
}
```

```
Monitor Puesto [id: 0..7] {
```

```
    ColaOrdenada c;
    cond espera[25];
    cond hayPedido;
```

```

Procedure llegar (idP: in int, edad: in int) {
    c.pushOrdenado(idP, edad);
    signal (hayPedido);
    wait (espera[idP]);
}

Procedure sig (idP: out int) {
    if c.isEmpty() wait (hayPedido);
    c.pop(idP);
    signal(espera[idP]);
}

Monitor Silla [id: 0..7] {
    boolean hayPaciente = false;
    cond llegue, vacunacion;

    Procedure espera() {
        if not hayPaciente wait (llegue);
        hayPaciente = false;
    }

    Procedure fin() {
        signal(vacunacion);
        wait(llegue);
    }

    Procedure llegar() {
        hayPaciente = true;
        signal(llegue);
        wait(vacunacion);
        signal(llegue);
    }
}

```

5.

Resolver con **MONITORES** el siguiente problema. En una sala se juntan **20 conferencistas** y **un coordinador** para una conferencia internacional. Cuando todos han llegado a la sala (los 20 conferencistas y el coordinador) el coordinador abre la sesión con una presentación de 30 minutos, y luego cada conferencista realiza su presentación de 10 minutos, de a uno a la vez y de acuerdo al orden en que llegaron a la sala. Cuando todas las presentaciones terminaron, las personas (conferencistas y coordinador) se retiran.

```
Process Conferencista [id: 0..19] {
    Conferencia.llegarConferencista();
    // Presentacion 30 mins
    Conferencia.fin();
}
```

```
Process Coordinador {
    Conferencia.llegarCoordinador();
    // Presentacion 10 mins
    Conferencia.fin();
}
```

```
Monitor Conferencia {
    cond esperalnicio;
    cond esperaFin;
    cond esperaLlamado;
    int cant = 0;
    int terminaron = 0;
```

```
Procedure llegarCoordinador() {
    if cant < 20 {
        wait (esperalnicio);
    }
}
```

```
Procedure llegarConferencista() {
    cant++;
    if (cant == 20) signal (esperalinicio);
    wait (esperallamado);
}

Procedure fin() {
    terminaron++;
    if (terminaron == 21) {
        signal_all (esperafin);
    }
    else {
        signal (esperallamado);
        wait (esperafin);
    }
}
```

6.

Resolver con **Monitores** el siguiente problema. 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.

```
Process Persona [id: 0..N-1] {
```

```
    Admin.SolicitarUso();
```

```
    Usar_Juego();
```

```
    Admin.Liberar();
```

```
}
```

```
Process Empleado {
```

```
    while (true) {
```

```
        Desinfectar_Juego();
```

```
        Admin.Listo();
```

```
}
```

```
}
```

```
Monitor Admin {
```

```
    cond ePersona, eEmpleado;
```

```
    int esperando = 0;
```

```
    bool libre = true;
```

```
Procedure SolicitarUso() {
```

```
    if (not libre) {
```

```
        esperando++;
```

```
        wait (esperaAutos);
```

```
}
```

```
    else libre = false;
```

```
}
```

```
Procedure Liberar() {
    signal (eEmpleado);
}

Procedure Listo () {
    if (esperando > 0) {
        esperando--;
        signal (ePersona);
    }
    else libre = true;
    wait (eEmpleado);
}
}
```

7.

Resolver con **Monitores** el siguiente problema. En un gimnasio para rehabilitación hay una máquina que debe ser usada por N personas, de a una a la vez y de acuerdo al orden dado por la edad (*cuando la máquina está libre la debe usar la de mayor edad que está esperando usarla*). **Nota:** suponga que la persona tiene una función *Usar_Máquina* que simula el uso de la máquina.

```
Process Persona [id: 0..N-1] {
    int edad = ...;
    AdminMaquina.usar(id, edad);
    Usar_Maquina
    AdminMaquina.salir();
}
```

```
Monitor AdminMaquina {
    cond espera[N];
    boolean libre = true;
    ColaOrdenada c;
    int idAux;
    int cant = 0;

    Procedure usar(id: in int, edad: in int) {
        if not libre {
            cant++;
            c.pushOrdenado(id, edad);
            wait(espera[id]);
        } else {
            libre = false;
        }
    }
}
```

```
Procedure salir() {
    if cant > 0 {
        cant--;
        c.pop(idAux);
        signal(espera[id]);
    } else {
```

```
    libre = true;  
}  
}  
}
```

8.

Resolver con **Monitores** el siguiente problema. Hay N personas que deben usar un teléfono público de a una a la vez y de acuerdo al orden de llegada, pero dando prioridad a las que tienen que usarlo con urgencia (*cada persona ya sabe al comenzar si es de tipo urgente o no*). **Nota:** suponga que la persona tiene una función *Usar_Teléfono* que simula el uso del teléfono.

```
Process Persona [id: 0..N-1] {
    boolean urgente = ...;
    AdminLlegada.llegar(id, urgente);
    Usar_Teléfono();
    AdminLlegada.salir();
}
```

```
Monitor AdminLlegada {
    cond espera[N];
    boolean libre = true;
    int cantidad = 0;
    int idAux;
    Cola c;
```

```
Procedure llegar (id: int int, urgente: in boolean) {
    if not libre {
        cantidad++;
        c.pushOrdenado(id, urgente);
        wait (espera[id]);
    } else {
        libre = false;
    }
}
```

```
Procedure salir() {
    if cantidad > 0 {
        cantidad--;
        c.pop(idAux);
        signal (espera[idAux]);
    } else {
```

```
    libre = true;  
}  
}  
  
}
```