Práctica 1

1) - Verdadero
   - Verdadero
   - Verdadero

2) int contador = 0; int N = … ; int M = … ;
   process encontrar (i = 0 to M-1)
      if (array[i]==N){
         <contador++>
      }

3) sd
   - Puede que un productor comience a producir un elemento, comenzando por incrementar la variable cant (de manera atómica), pero antes de poder guardar el elemento venga un consumidor que decrementa la variable y busca en el buffer el elemento que todavía no pudo cargar el productor, por lo que genera un error.

     ```
     int cant = 0; int pri_ocupada = 0; int pri_vacia = 0; int buffer[N];
     Process Productor:: {
        while (true) {
           //produce elemento
           <await (cant < N); cant++;
           buffer[pri_vacia] = elemento;>
           pri_vacia = (pri_vacia + 1) mod N;
        }
     }
     Process Consumidor:: {
        while (true) {
           <await (cant > 0); cant--;
           elemento = buffer[pri_ocupada];>
           pri_ocupada = (pri_ocupada + 1) mod N;
           //consume elemento
      }
     }
     ```

   - ```
     int cant = 0; int pri_ocupada = 0; int pri_vacia = 0; int buffer[N];
     Process Productor[id: 0 ... P-1] {
        while (true) {
           //produce elemento
           <await (cant < N); cant++;
           buffer[pri_vacia] = elemento;
           pri_vacia = (pri_vacia + 1) mod N;>
        }
     }
     ```
     ```

```
Process Consumidor[id: 0 ... C-1] {
   while (true) {
      <await (cant > 0); cant--;
      elemento = buffer[pri_ocupada];
      pri_ocupada = (pri_ocupada + 1) mod N;>
      //consume elemento
 }
}
```

4)
```
ColaRecursos cola;
int cant = 5;
Process proceso [id: 0 ... N-1] {
   <await (cant > 0); recurso = cola.pop();
   cant--;>
   //usa el recurso
   <cola.push(recurso); cant++;>
}
```

5)

- ```
  Process Persona [id: 0 … N-1]{
     Documento documento;
     <Imprimir(documento);>
  }
  ```

- ```
  ColaPersonas cola;
  int siguiente = -1;
  Process Persona [id: 0 … N-1]{
     Documento documento;
     <if (siguiente == -1) siguiente = id;
     else cola.push(id);>
     <await (siguiente == id); >
     Imprimir(documento);
     <if Empty(cola) siguiente = -1;
     else siguiente = cola.pop(id);>
  }
  ```

- ```
  ColaPersonas cola; //asumo función pushOrdenado().
  int min = -1;
  Process Persona [id: 0 … N-1]{
     Documento documento;
     <if (min == -1) min = id;
     else cola.pushOrdenado(id);>
     <await (min == id);>
     Imprimir(documento);
     <if Empty(cola) min = -1;
     else min = cola.pop();>
  }
  ```
  ```

- ColaPersonas cola; //asumo función pushOrdenado().
  int min = -1;
  Process Persona [id: 0 … N-1]{
  Documento documento;
  <if (min == -1) min = id;
  else cola.pushOrdenado(id);>
  <await (min == id);
  Imprimir(documento);>
  <if Empty(cola) min = -1;
  else min = cola.pop();>
  }

- ColaPersonas cola; //asumo función pushOrdenado().
  int siguiente = -1;
  boolean impresoraOcupada = false;
  Process Persona [id: 0 … N-1]{
  Documento documento;
  <cola.push(id);>
  <await (siguiente == id);>
  Imprimir(documento);
  impresoraOcupada = false;
  }
  Process Coordinador {
  while true {
  <await (!Empty(cola)) && (!impresoraOcupada);>
  impresoraOcupada = true;
  siguiente = cola.pop();
  }
  }

6) ColaEntregas cola;
   int contador = 0;
   boolean comenzar = false;

   Process Profesor [id: 1 … 3]{
   <await contador == N;>
   comenzar = true;
   while (true){
   <await (!Empty(cola)); cola.pop(id);>
   }
   }

   Process Alumno [id: 0 … N-1]{
   <contador++;>
   <await (comenzar);>
   //realiza el examen;
   <cola.push(id);>
   }

7)

- Exclusión Mutua: a lo sumo un proceso va a estar en la sección crítica, ya que el otro loopea hasta que termina el anterior.
- Ausencia de DeadLock: un proceso siempre va a loopear hasta que el otro cambie de estado y el otro siempre va a hacerlo, por lo que no puede pasar que se esperen mutuamente indefinidamente.
- Ausencia de demora innecesaria: no ocurre ya que cambia su turno al salir de la sección crítica, sin hacer nada adicional.
- Eventual entrada: la variable turno comienza en uno y va modificándose siempre que sale el proceso de su sección crítica, por lo que ambos procesos pueden ejecutarse por turnos.

8)

```
bool peticiones[N]; int actual = -1;
Process Trabajador [id: 0 … N-1]{
    while (true){
        peticiones[i] = true;
        while (actual != id) skip;
        //SC
        actual = -1;
    }
}

Process Coordinador {
    while (true){
        for (i = 1 to N){
            if (peticiones[i]){
                peticiones[i] = false;
                actual = i;
                while (actual != -1) skip;
            }
        }
    }
}
```