1. Sea una sucesión en la que cada término es la suma de los dos anteriores. Para calcular las sumas parciales de los términos de esta sucesión tenemos dos procesos: `sucesión` y `sumar`; el primero calcula los términos de la sucesión y el segundo calcula la suma parciales de los términos impares. Implementar el algoritmo concurrente anterior incluyendo únicamente regiones críticas condicionales.

```
i: integer := 0
j: integer := 0
sumas: integer := 0
buffer: ArrayOfIntegers(0…infinity)

Procedure sucesión
begin
        buffer[0] := 0
        buffer[1] := 1
        i = 2
        loop
                buffer[i] := buffer[i-1] + buffer[i-2]
                i := i + 1
        forever
end

Procedure sumar
begin
        loop
                Region R when i > j do
                        if(j mod 2 = 0) then
                                sumas := sumas + buffer[j]
                        j := j +1
        forever
end
```

2. En un hotel hay diez vehículos automáticos pequeños y otros diez grandes. Todos ellos están controlados por un programa con procesos concurrentes (uno por vehículo). Estamos interesados en la parte del programa que controla la entrada en un montacargas, en el que caben cuatro vehículos pequeños d dos pequeños y uno grande. Modelar el sistema utilizando regiones críticas condicionales.

```
nVPequeños: Integer := 0
nVGrandes: Integer := 0

Procedure v_pequeño
begin
        Region R when (nVPequeños < 4 and nVGrandes = 0) or (nVPequeños < 2 and nVGrandes = 1)
                nVPequeños := nVPequeños + 1
                /*Esta en el montacarga*/
        Region R
                nVPequeños := nVPequeños – 1
end

Procedure v_grande
begin
        Region R when (nVgrandes = 0)
                nVGrandes := nVGrandes + 1
                /*Esta en el montacarga*/
        Region R
                nVGrandes := nVGrandes - 1
end
```

3. En un ascensor en el que caben cuatro personas, se atienden las llamadas que se hacen desde varios pisos. En estos pisos entran personas que quieren subir o bajar a otros pisos. Se pide modelar el sistema con regiones críticas condicionales, suponiendo que existe un proceso ascensor y varios procesos persona.

```
nPersonas : Integer := 0
PisoActual := 1
SubeBaja := 1

Procedure Ascensor
begin
        loop
                PisoActual := PisoActual + SubeBaja
                if(pisoActual = N-1) SubeBaja := -1
                if(pisoActual = 0) SubeNaja := 1
        forever
end


/*Cada persona tendrá un Origen y un Destino*/
Procedure Persona
begin
        Region R when pisoActual = Origen and nPersonas < 4
                nPersonas := nPersonas + 1
        Region R when pisoActual = Destino
                nPersonas := nPersonas – 1
end
```

4. Se dispone de un sistema con t terminales e i impresoras; en cada terminal se ejecuta un proceso que desea imprimir en una de las impresoras. Programar el acceso a las impresoras utilizando regiones críticas condicionales, de manera que el acceso a las mismas sea en exclusión mutua, pero permitiendo que procesos diferentes puedan utilizar impresoras diferentes al mismo tiempo.

Impresoras: ArrayOfBoolean(0…n-1) //true esta usada false no está usada

```
Procedure terminali
begin
        impresoraUsada: integer := -1
        i: integer := 0
        Region R
                while(i < n and impresoraUsada = -1)
                        if(not impresoras[i])
                                impresoraUsada = i
                        i++
                if(impresoraUsada  != -1)
                        impresoras[impresoraUsada] = True

        if(impresoraUsada  != -1)
                imprimir(impresoraUsada)
                Region R
                        impresoras[impresoraUsada] = False
end
```

5. Implementar utilizando regiones críticas condicionales un algoritmo de planificación que asigne un recurso bajo la premisa de «el trabajo más corto primero». Suponemos que cuando un proceso solicita una asignación, específica del tiempo máximo durante el que tiene intención de utilizar el recurso.

Necesito explicación :(

6. Determinar si es correcta la siguiente propuesta para implementar una región crítica condicional utilizando semáforos. Indicar la función de cada uno de los semáforos y variables. Inicialmente, `mutex=1`, `d=0` y `nd=0`.

Mutex hace que le acceso a la región sea en e.m.

Mientras que no se cumpla la condición de la región nd++, se desbloquea la región y se bloquea el proceso que acababa de entrar con wait(d). Posteriormente a este falta un wait(mutex) para que no se produzcan inconsistencias en la resta y lo que queda es innecesario debido a que ahora debería ejecutar la sección critica y despertar a otro si hiciese falta (falta un signal(mutex) en la parte de nd > 0)

```
wait(mutex);
while(not B) do
begin
nd:=nd+1;
signal(mutex);
wait(d);
nd:=nd-1;
if nd=0 then signal(mutex)
else if nd>0 then signal(d);
end;
S;
if nd=0 then signal(mutex)
else if nd>0 then signal(d);
```

7. Un una cuenta de ahorros es compartida entre distintas personas; cada persona puede sacar o depositar dinero, siendo el balance de la cuenta la suma de los depósitos menos la suma de los reintegros. El balance no puede ser negativo. Utilizar regiones críticas condicionales para:

   - Implementar las operaciones `depositar()` y `extraer()`.
   - Modificar la implementación anterior suponiendo que las extracciones son resueltas por orden de llegada.

```
total: Integer := 0

Procedure depositar //recibe cantidad (cantidad a depositar)
begin
        Region R
                total := total + cantidad
end


Procedure extraer //recibe cantidad (cantidad a extraer)
begin
        Region R (cantidad <= total)
                        total := total - cantidad
end


Procedure extraerEnOrden //recibe cantidad (cantidad a extraer)
begin
        NOENTIENDO :(((

end
```

8. Proporcionar una solución al problema de los filósofos utilizando reacciones críticas condicionales.

Buffer : Array(0…n-1)

```
Procedure filosofoi //recibe i (si hay n palillos
begin
        loop
                pensar()
                region R when palillos[i] and palillos[(i+1) mod n]
                        palillos[i] = False
                        palillos[(i+1) mod n] = False
                comer()
                Region R
                        palillos[i] = True
                        palillos[(i+1) mod n] = True
        forever
end
```