# TEMAS DE INTRO I

- Pilas (datos/estructuras de control)
- Filas (datos/estructuras de control)
- **Modularización y Parámetros**
- Variables
- Funciones y Método de Desarrollo
- Arreglos
- Matrices

# **Estrategia:**"**Divide y Conquista**",

Una de las estrategias más útiles en la resolución de problemas con computadora es la descomposición de un problema en subproblemas más simples: "**Divide y Conquista**".

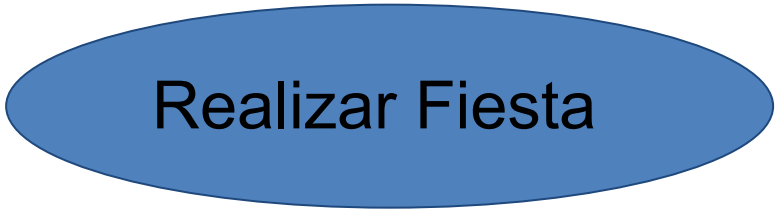
- Cada problema es dividido en un número de subproblemas más pequeños, cada uno de los cuales a su vez, puede dividirse en un conjunto de subproblemas más pequeños aún, y así siguiendo.

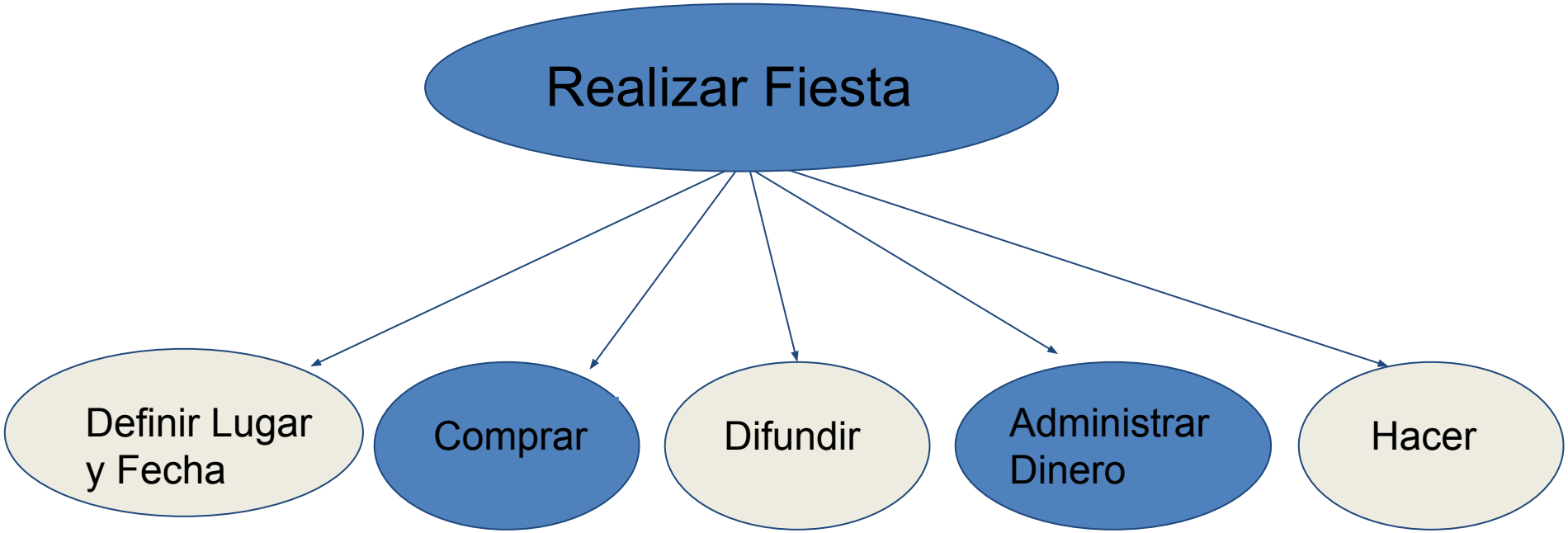
- Cada uno de estos subproblemas debiera resultar entonces más simple de resolver.

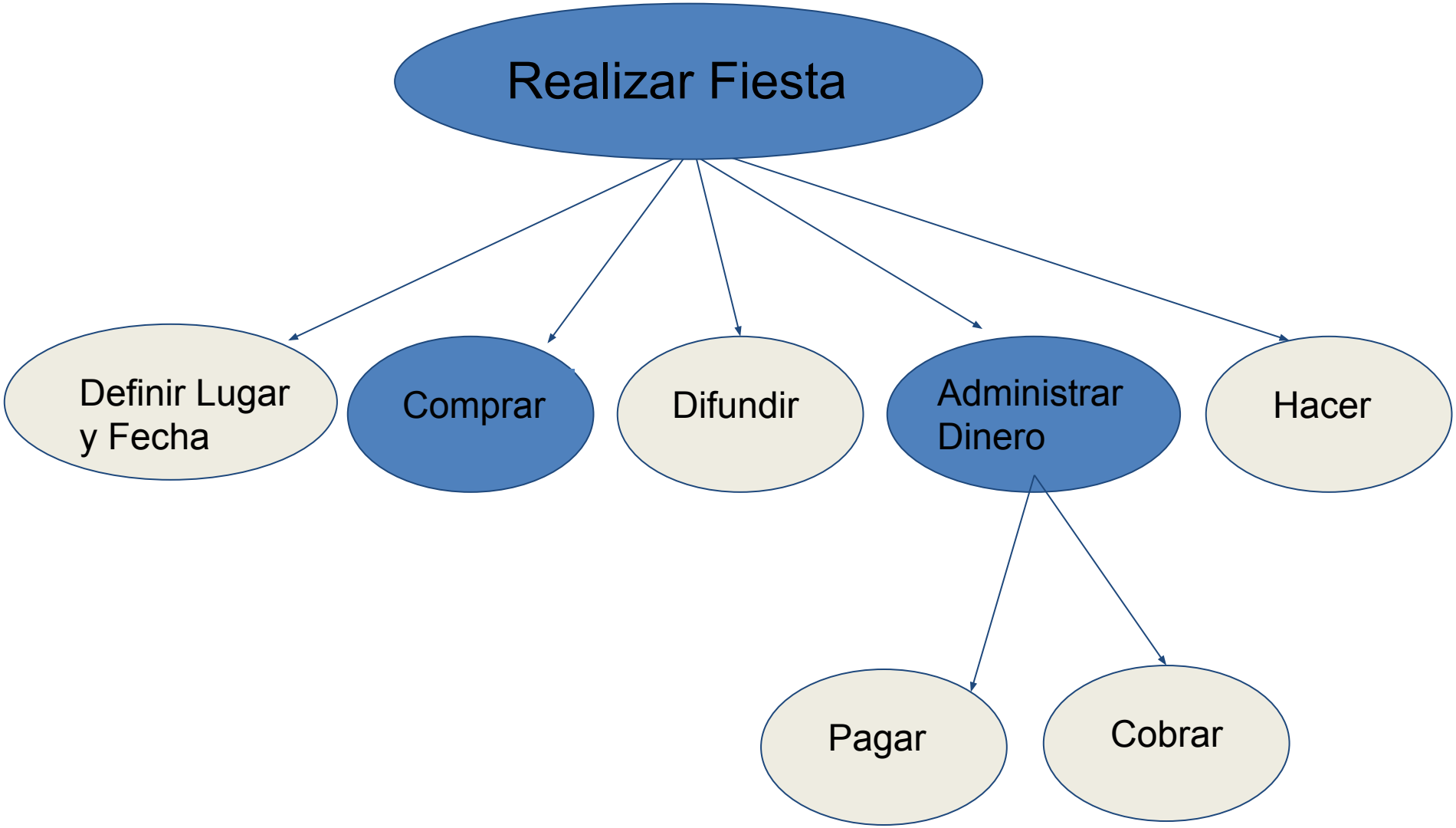
- Una metodología de resolución con estas características se conoce como diseño Top -Down.

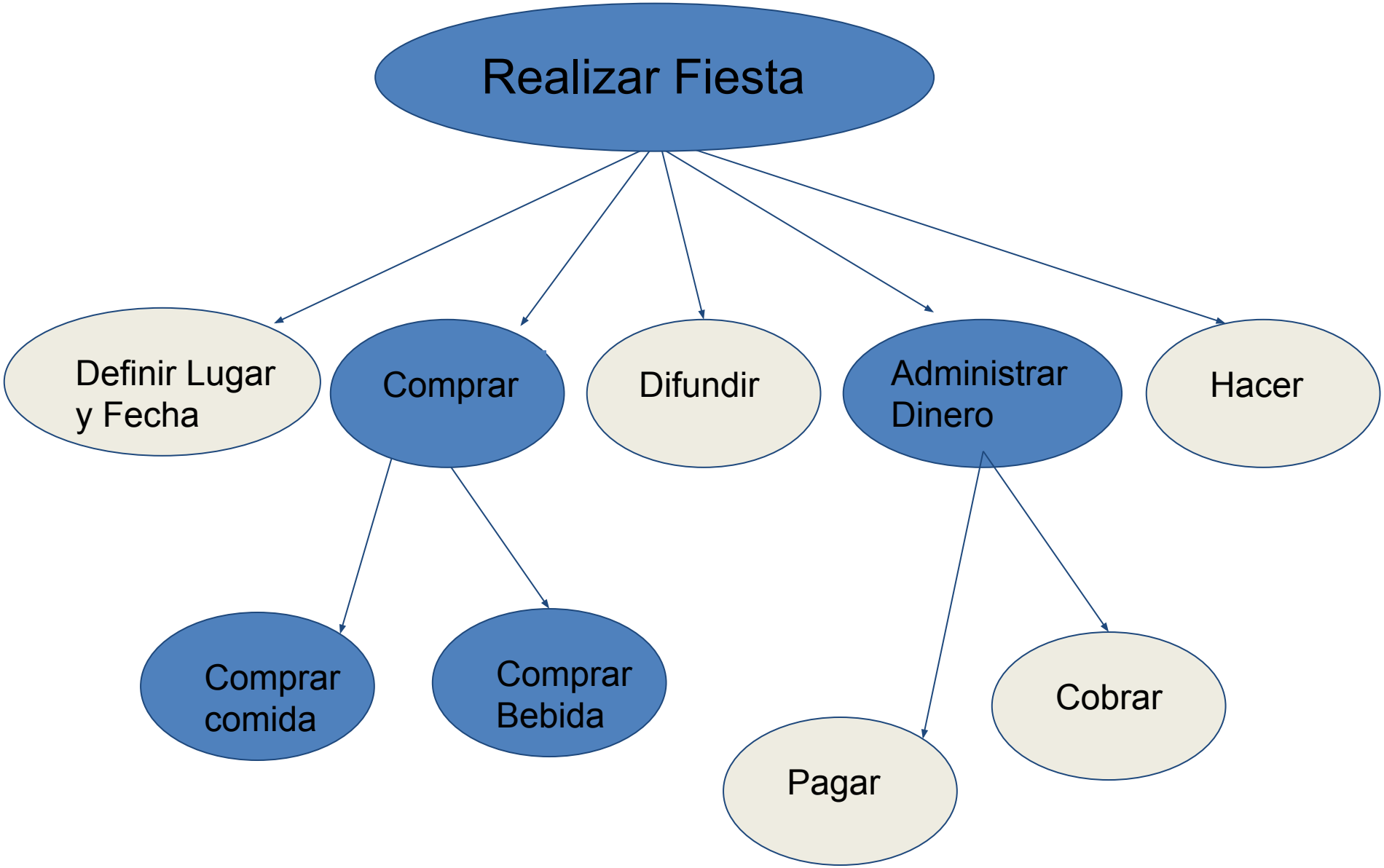
PROBLEMA:

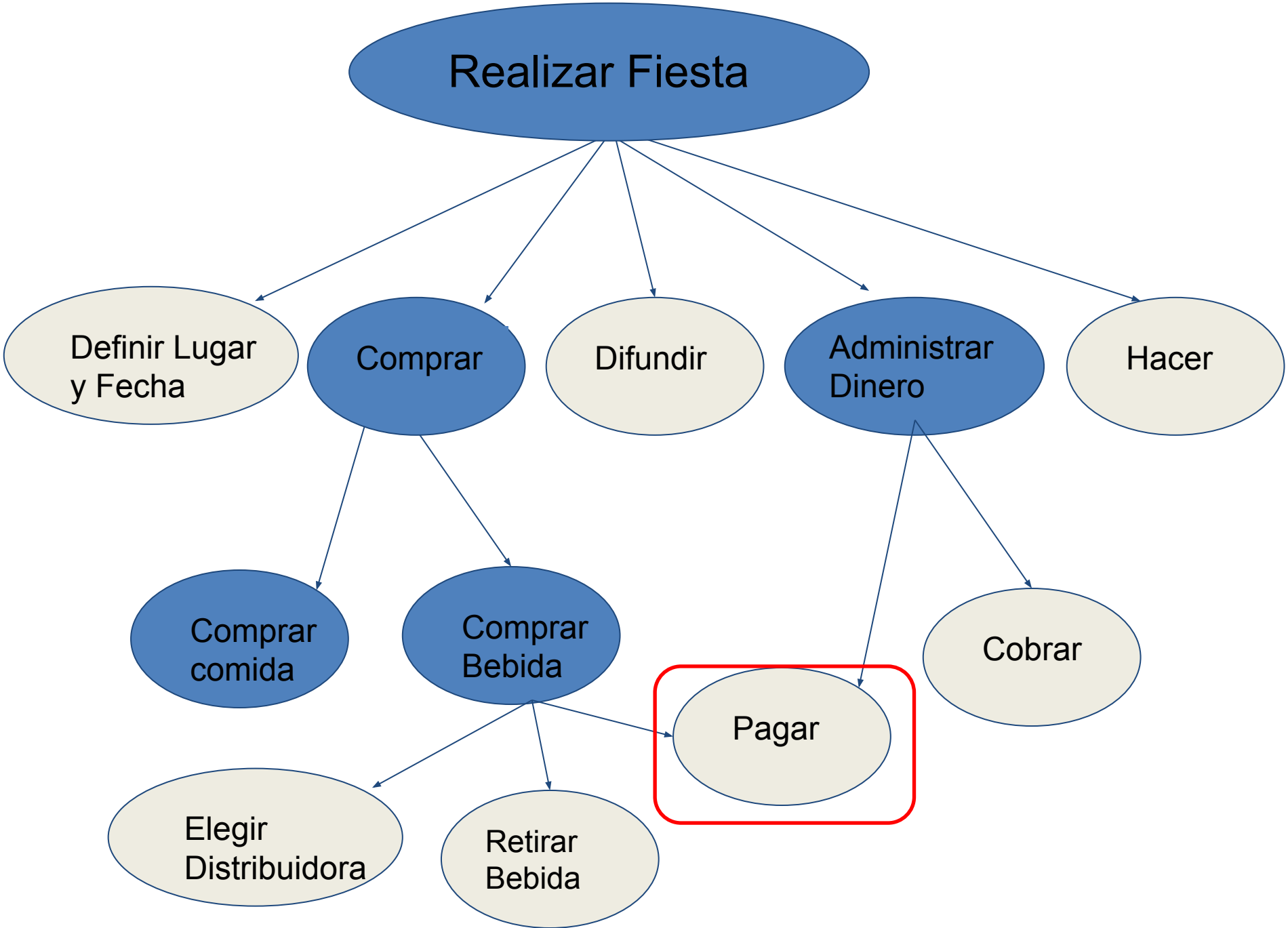
REALIZAR UNA FIESTA PARA 200 PERSONAS


Realizar Fiesta


Realizar Fiesta
Definir Lugar y Fecha
Comprar
Difundir
Administrar Dinero
Hacer


Realizar Fiesta
Definir Lugar y Fecha
Comprar
Difundir
Administrar Dinero
Hacer
Pagar
Cobrar


Realizar Fiesta
Definir Lugar y Fecha
Comprar
Difundir
Administrar Dinero
Hacer
Comprar comida
Comprar Bebida
Pagar
Cobrar


Realizar Fiesta
Definir Lugar y Fecha
Comprar
Difundir
Administrar Dinero
Hacer
Comprar comida
Comprar Bebida
Cobrar
Pagar
Elegir Distribuidora
Retirar Bebida

USO DE LA ESTRATEGIA **DIVIDE Y CONQUISTA** EN RESOLUCION DE PROBLEMAS CON COMPUTARDORA

1) Pensar en la descomposición del problema en subproblemas
   (¡no hay una única forma!)

2) FORMALIZAR la descomposición:

**DIAGRAMA DE ESTRUCTURA (DE)**

# DIAGRAMA DE ESTRUCTURA

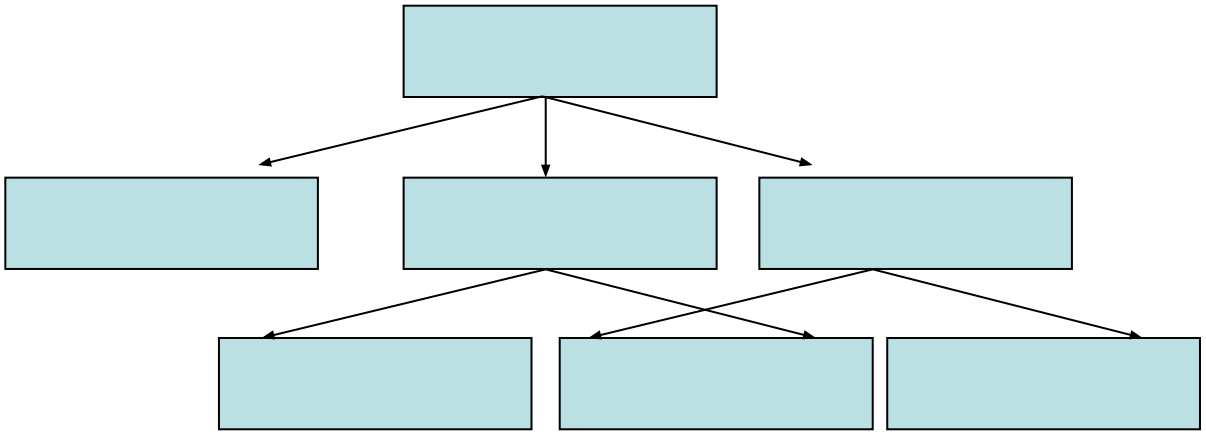
- Un diagrama de estructura(DE) permite modelar un programa como una jerarquía de módulos.

- Cada nivel de la jerarquía representa una descomposición más detallada del módulo del nivel superior. La notación usada se compone básicamente de tres símbolos:

  – Módulos

  REALIZAR UNA FIESTA

  – Invocaciones



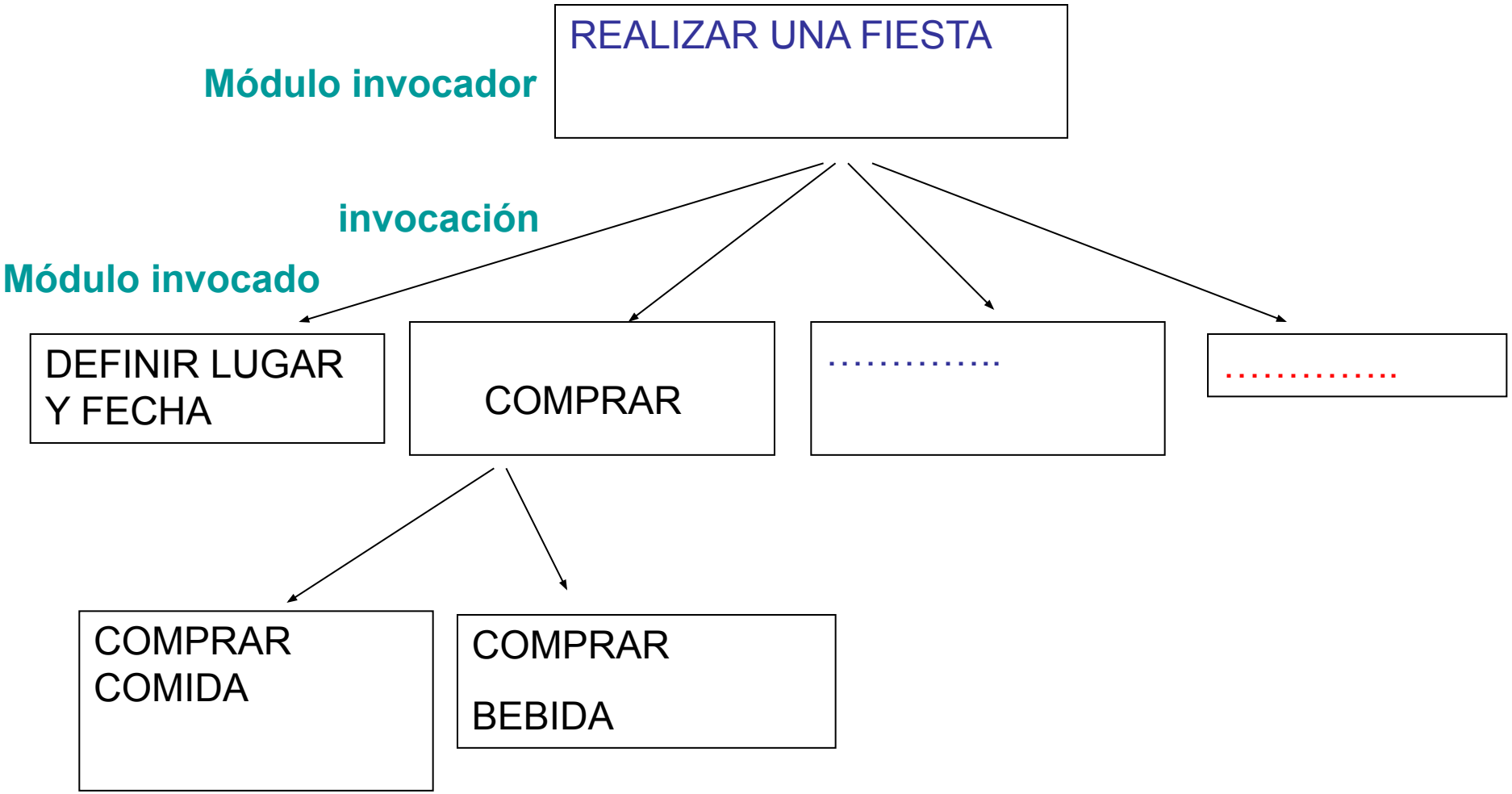
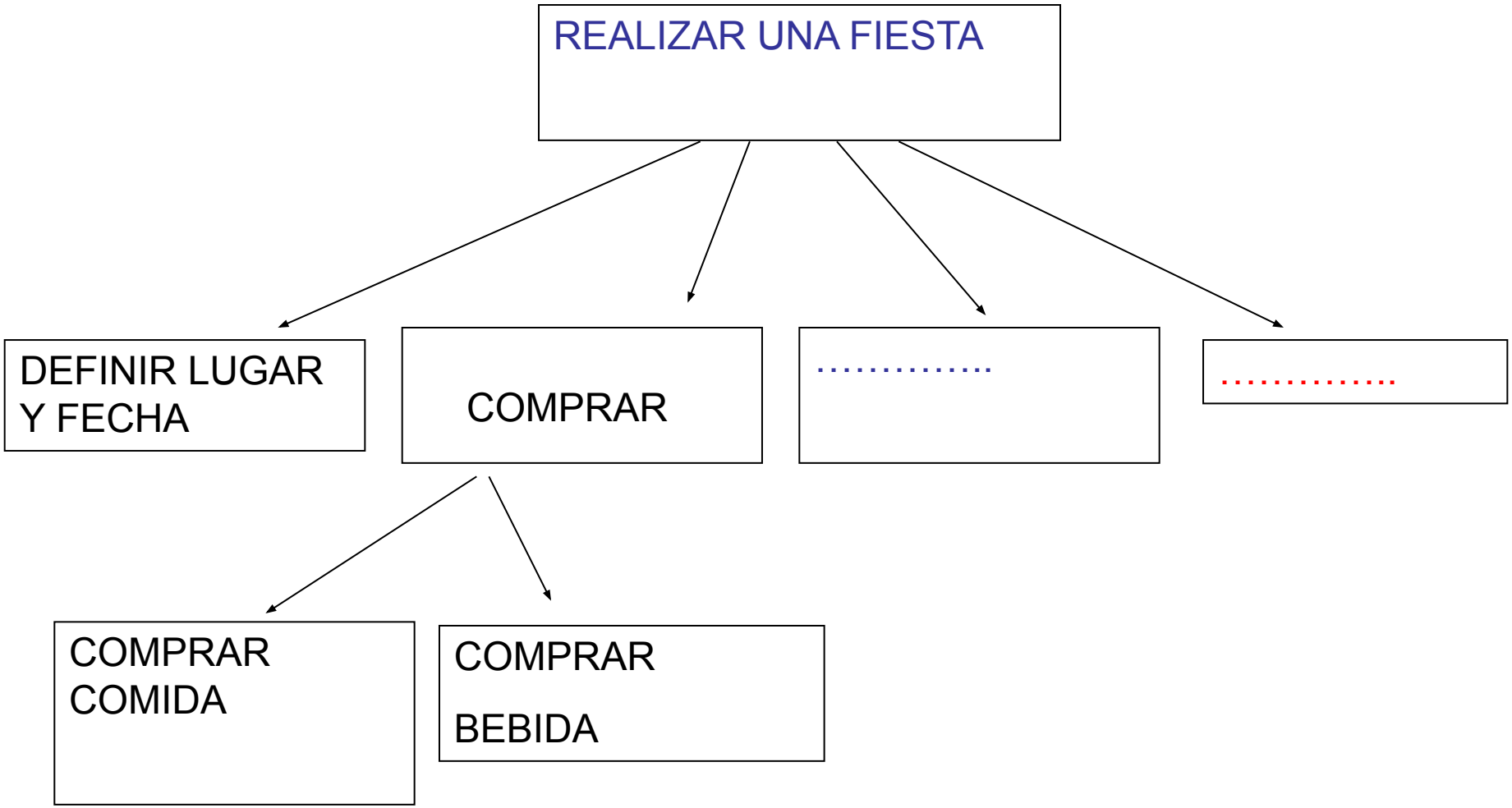
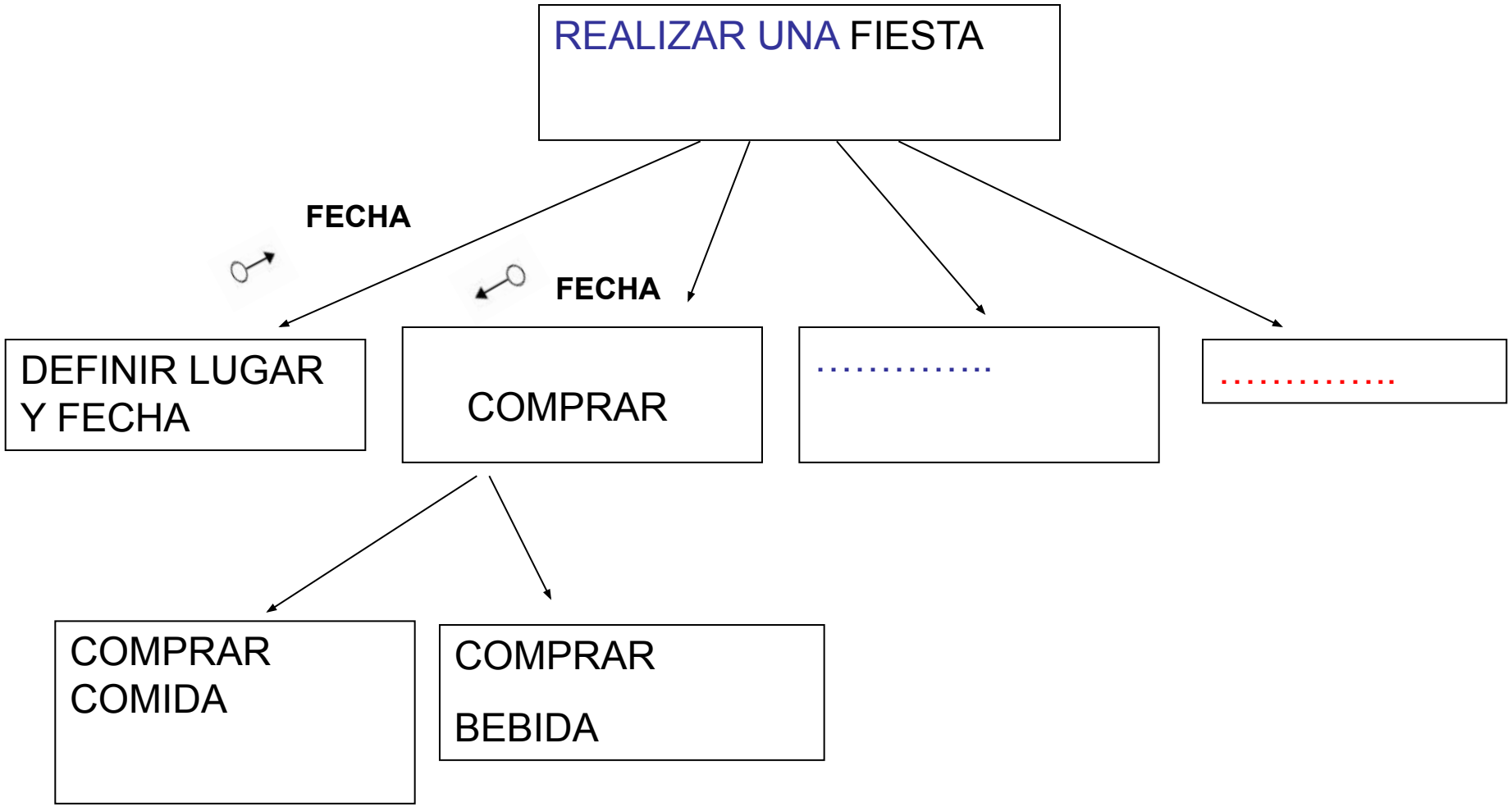


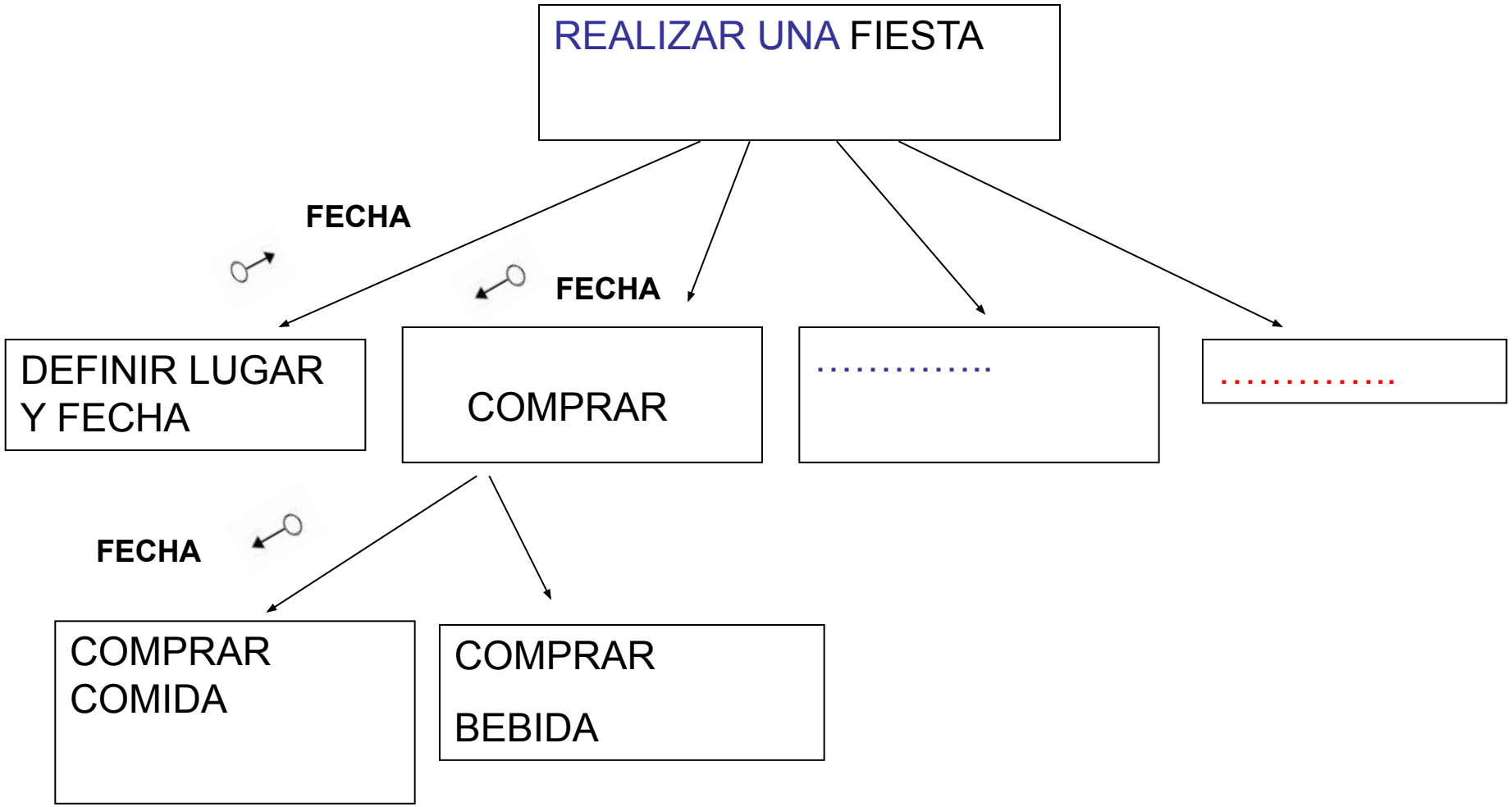

**DIAGRAMA de ESTRUCTURA (DE) INCOMPLETO DE "REALIZAR UNA FIESTA"**



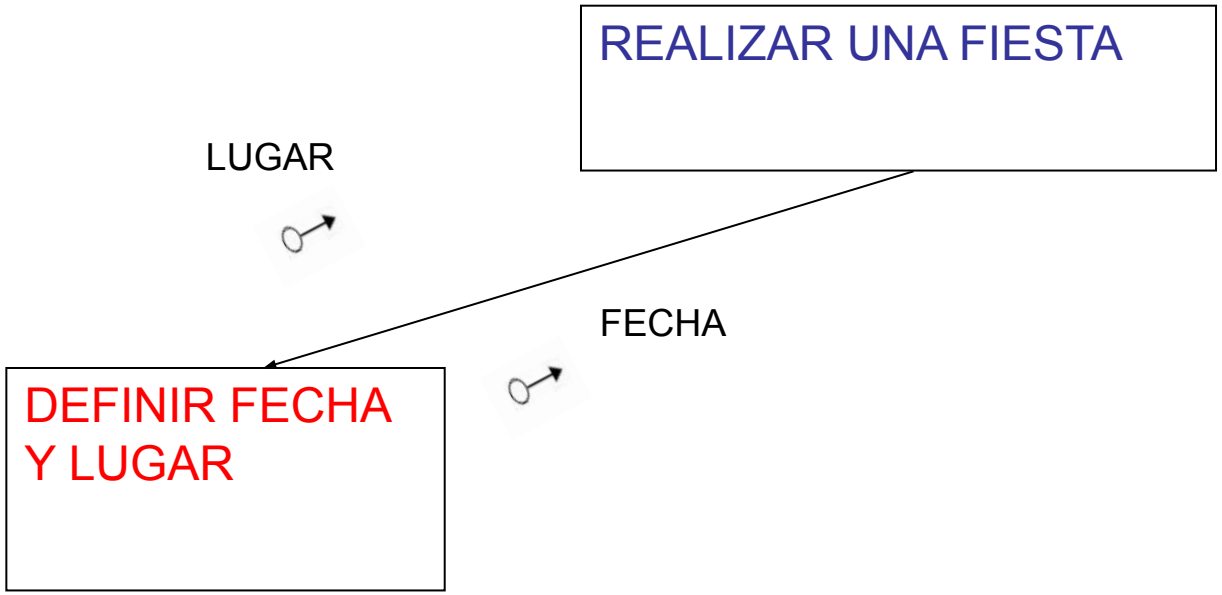

**¿El módulo COMPRAR necesita saber cuando será la fiesta?**




**LOS MODULOS DEBEN ENVIARSE INFORMACION**



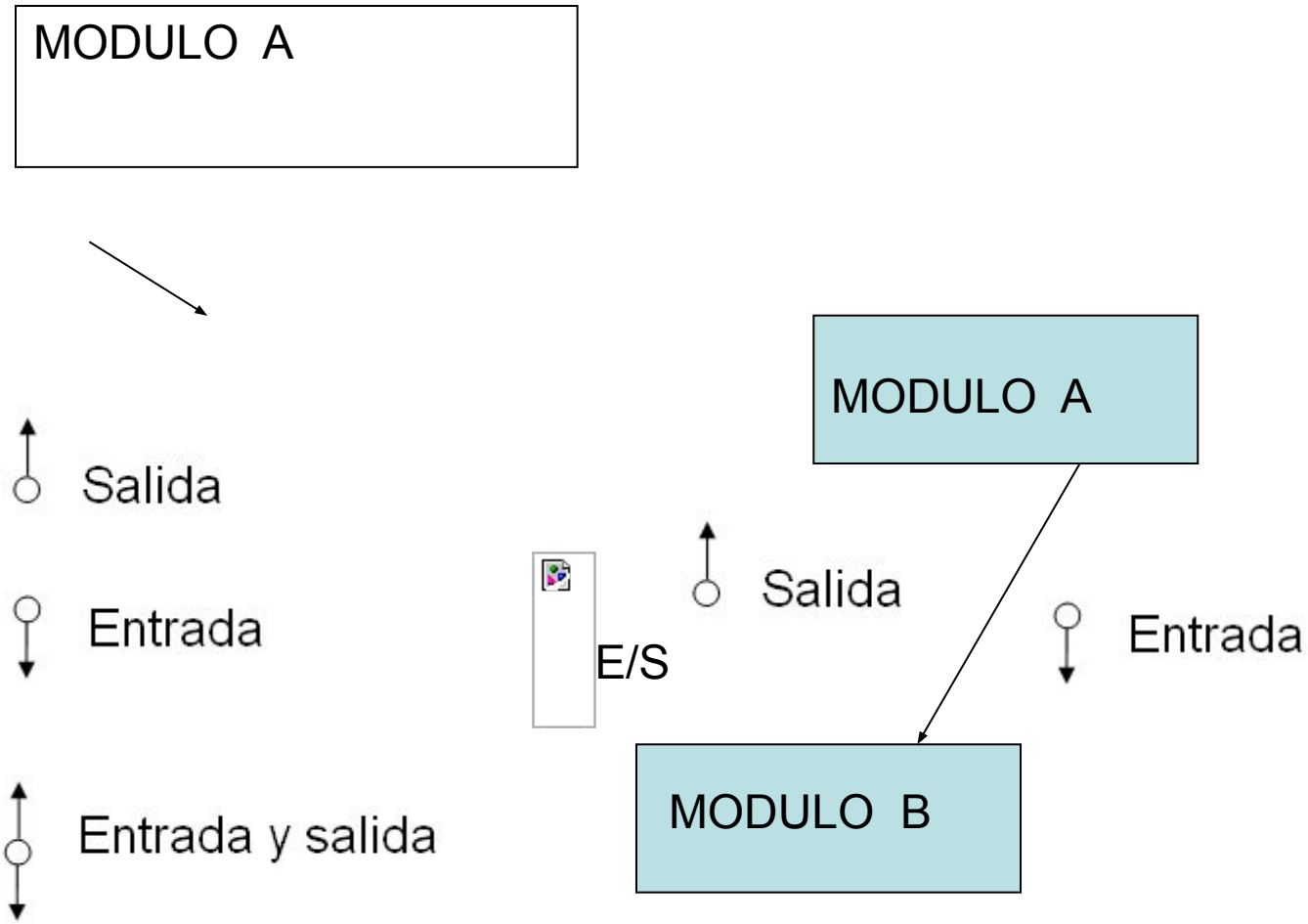

**LOS MODULOS DEBEN ENVIARSE INFORMACION**




**CUPLAS**: es el mecanismo mediante el cual los módulos de un Diagrama de Estructuras se envían **la información**

# DIAGRAMA DE ESTRUCTURA

- Módulos
- Invocaciones
- Cuplas



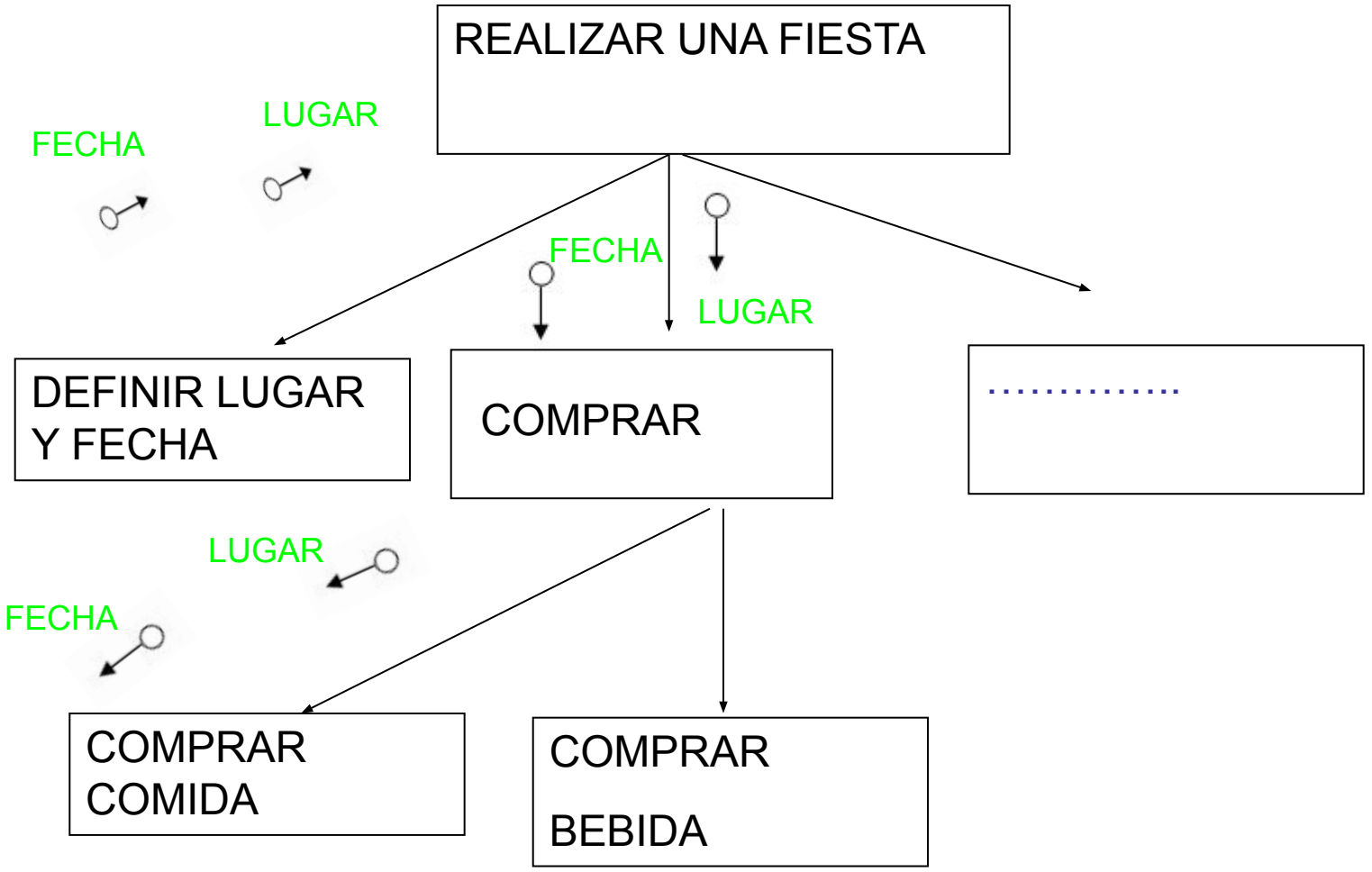



**DIAGRAMA de ESTRUCTURA INCOMPLETO DE REALIZAR UNA FIESTA**

- Ejercicio 11 del Pco. 2 :

- Concatenar dos pilas de modo que la que posee menos elementos quede abajo; si ambas tienen la misma cantidad de elementos, cualquiera puede quedar abajo.

Nota: para simplificar, en la nueva pila no vamos a mantener el orden de los elementos en cada Pila

```
Program ConcatenarSegúnOrden
{ este programa …..}
{$INCLUDE /IntroProg/Estructu}
var             {programa principal}
    PilaUno, pilaDos, PilaConcat: Pila;
 begin
    readpila(PilaUno);
    readpila(PilaDos);
    inicpila(Resultado, ' ' );
    inicpila(PilaConcat, ' ' );
    while not  pilavacia(PilaUno) and  not pilavacia(PilaDos)do
    begin
         apilar (uno, desapilar(PilaUno));
         apilar (dos, desapilar(PilaDos))
   end;
   if pilavacia(PilaUno)  then   { pilaUno es menor o igual, con lo cual va abajo}
     begin
        while not pilavacia(Pila1) do
          apilar(PilaConcat, desapilar(Pila1));
        while not pilavacia(Pila2) do
         apilar(PilaConcat, desapilar(Pila2));
    end
  else …….. { pilaDos es menor o igual, con lo cual va abajo}
```

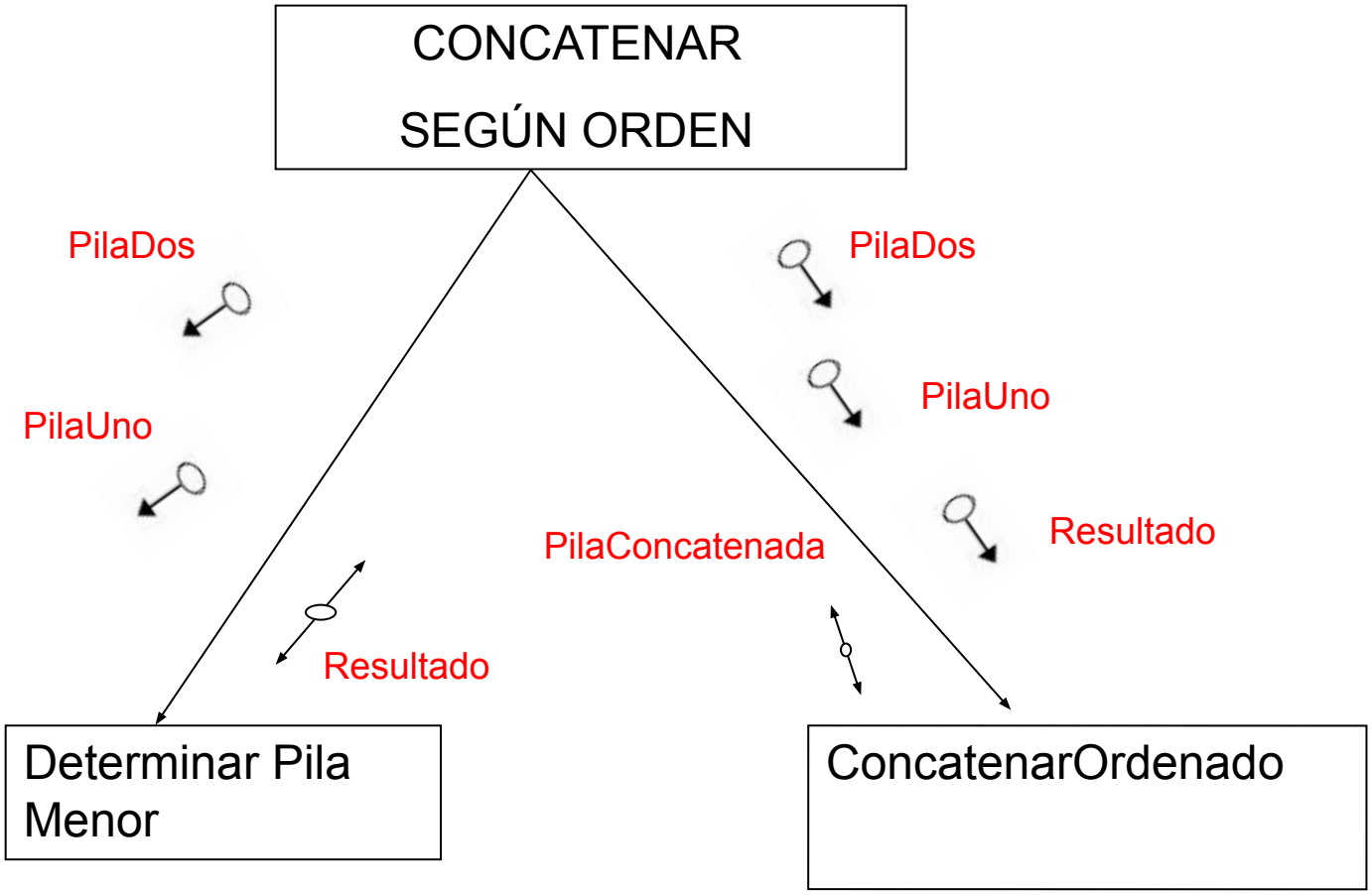
- Ejercicio 11 del Pco. 2 :  Concatenar dos pilas de modo que la que posee menos elementos quede abajo; si ambas tienen la misma cantidad de elementos, cualquiera puede quedar abajo.Nota: para simplificar, en la nueva pila no vamos a mantener el orden de los elementos en cada Pila

**Este problema se puede dividir en dos subproblemas**

- Detectar Pila Menor

- Concatenar primero la Pila  Menor

## Un posible DE para el problema Pco.2 ej. 11




**Nombres SIGNIFICATIVOS para Módulos y cuplas.**

¿Estos nombres son significativos?

USO DE LA ESTRATEGIA DIVIDE Y CONQUISTA EN RESOLUCION DE PROBLEMAS CON COMPUTARDORA

1) Pensar en la descomposición del problema en subproblemas
(¡no hay una única forma!)

2) FORMALIZAR la descomposición:

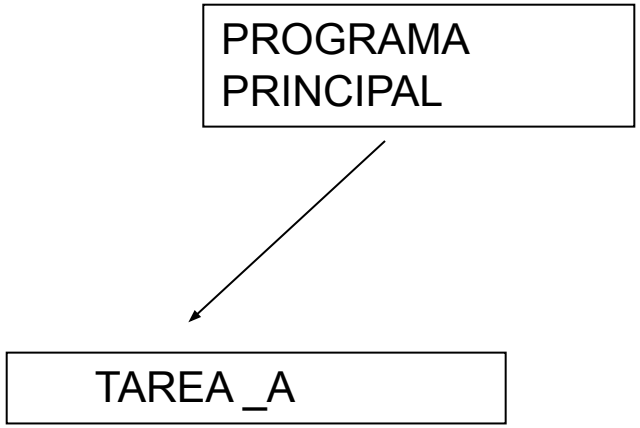
**DIAGRAMA DE ESTRUCTURA (DE)**

3) Volcar el DE en un programa escrito en Pascal

# Procedimientos

Un **procedimiento** es un conjunto de instrucciones Pascal que ejecutan una tarea
Es una herramienta que brinda Pascal para implementar los **módulos**.

**PROGRAM** PROGRAMA PRINCIPAL
{ Comentario del programa principal}




**Var {del programa principal}**
**Begin {del programa principal}**
….
….

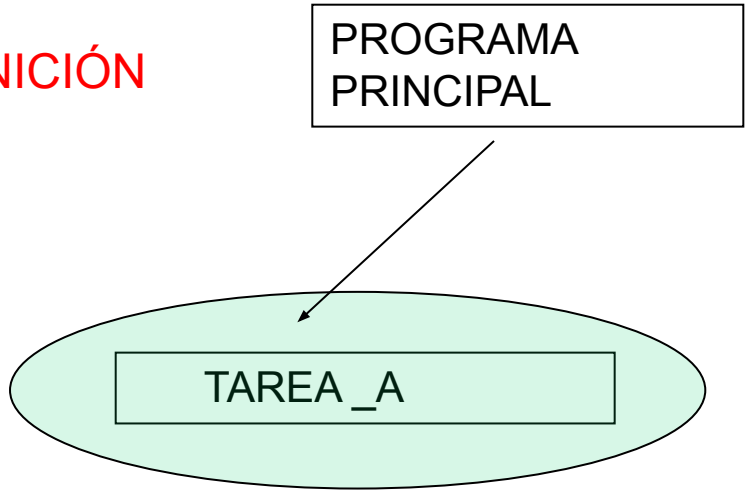
**End. {del programa principal}**

# Procedimientos

Un **procedimiento** es un conjunto de instrucciones Pascal que ejecutan una tarea
Es una herramienta que brinda Pascal para implementar los **módulos**.

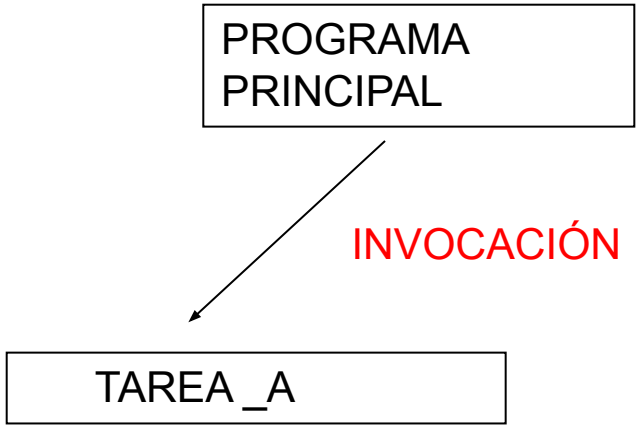
```
PROGRAM  PROGRAMAPRINCIPAL
{ Comentario del programa principal}

Procedure TAREA_A (…);
Begin
….
End;

….

Var {del programa principal}
Begin {del programa principal}

End.
```

DEFINICIÓN



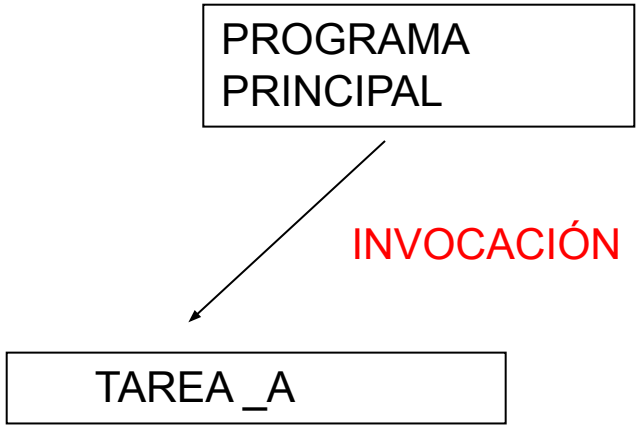
# Procedimientos

```
PROGRAM  PROGRAMAPRINCIPAL
{ Comentario del programa principal}


Procedure TAREA_A (…);
Begin
….
End;

….

Var {del programa principal}
Begin {del programa principal}
……..
TAREA_A (…);
……..

End.
```



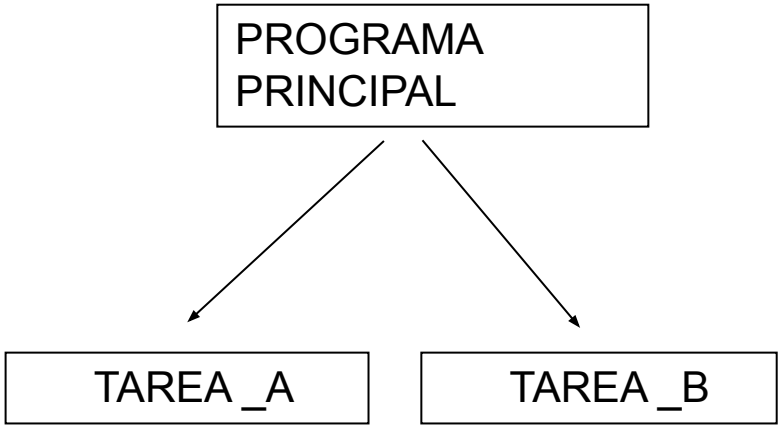
# Procedimientos

```
PROGRAM  PROGRAMAPRINCIPAL
{ Comentario del programa principal}


Procedure TAREA_A (…);
Begin
….
End;

….

Var {del programa principal}
Begin {del programa principal}
……..
TAREA_A (…);
……..
TAREA_A (…);
End.
```



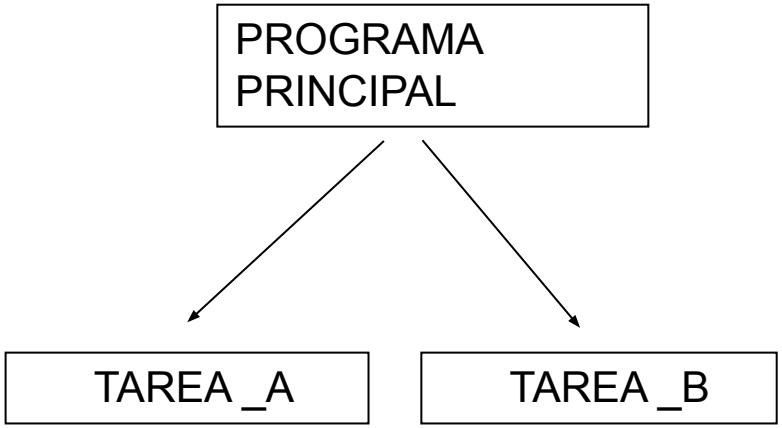
# Procedimientos

```
PROGRAM  PROGRAMAPRINCIPAL
{ Comentario del programa principal}
……
Procedure TAREA_A (…);
Begin
….
End;

Procedure TAREA_B (…);
Begin
….
End;
….

Var      {del programa principal}
Begin  {del programa principal}
……….
TAREA_A (…);
TAREA_B (…..);
……….
End.
```

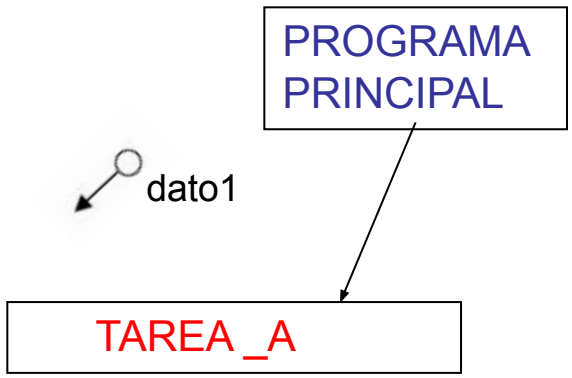


# Procedimientos

```
PROGRAM  PROGRAMAPRINCIPAL
{ Comentario del programa principal}

Procedure TAREA_A (…);
Begin
….
End;

Procedure TAREA_B (…);
Begin
….
End;
….

Var      {del programa principal}
Begin  {del programa principal}
……….
IF   (….)
     THEN
          TAREA_A (…)
     ELSE
          TAREA_B (…..);
……….
End.
```

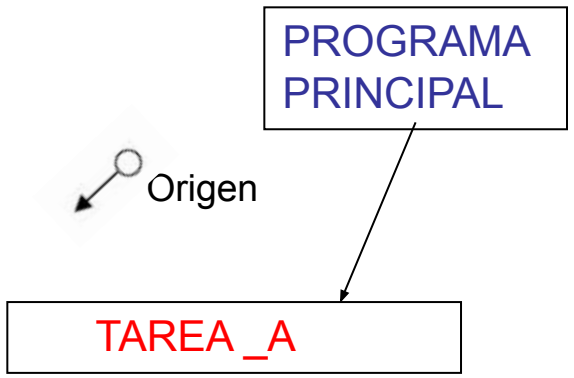


# CUPLAS del DE → PARÁMETROS de los procedimientos

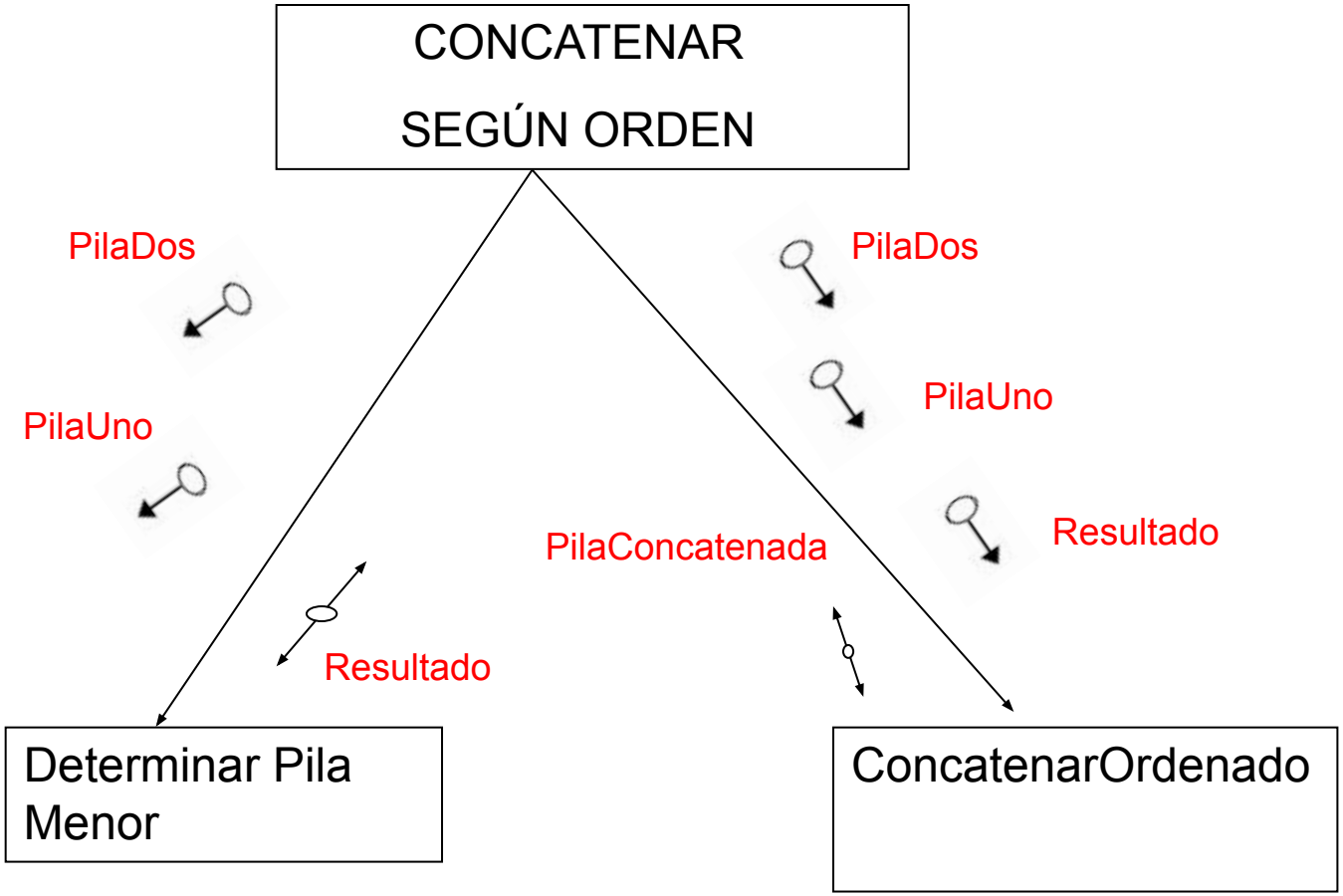



```
PROGRAM PROGRAMA PRINCIPAL
{ Comentario del programa principal}

Procedure TAREA _A  (dato1: tipoPar1);
Begin
….
End;


….

Var {del programa principal}
Begin {del programa principal}
….
….
  TAREA_A (dato1);
…
End.
```

# CUPLAS del DE → PARÁMETROS entre los procedimientos




```
PROGRAM PROGRAMA PRINCIPAL
{ Comentario del programa principal}

Procedure TAREA_A  (origen:Pila);
Begin
….
End;


….

Var {del programa principal}
   Origen: Pila;

Begin {del programa principal}
  readPila(Origen);
  ….
 TAREA_A  (Origen);
   …
end.
```




**Un <u>posible</u> DE para el problema Pco.2 ej. 11**

```
Program ConcatenarSegúnOrden
{ este programa …..}
{$INCLUDE /IntroProg/Estructu}
Procedure DeterminarPilaMenor ( PilaUno, PilaDos: Pila , var Resultado: Pila);
……..

 Procedure Concatenar( PilaUno, PilaDos, Resultado: Pila , var PilaConcat: Pila );
……..

var                          {programa principal}
   PilaUno, pilaDos, Resultado, PilaConcat: Pila;

begin
   readpila(PilaUno);
   readpila(PilaDos);
   inicpila(Resultado, ' ' );
   inicpila(PilaConcat, ' ' );
   DeterminarPilaMenor (PilaUno, PilaDos, Resultado);
   Concatenar(PilaUno, PilaDos, Resultado,PilaConcat );
   writePila (PilaConcat)
end.
```

```
Program ConcatenarSegúnOrden
{ este programa …..}
{$INCLUDE /IntroProg/Estructu}

Procedure DeterminarPilaMenor ( PilaUno, PilaDos: Pila , var Resultado: Pila);
……..

 Procedure Concatenar( PilaUno, PilaDos, Resultado: Pila , var PilaConcat: Pila );
……..

var                         {programa principal}
   PilaUno, pilaDos, Resultado, PilaConcat: Pila;

begin
   readpila(PilaUno);
   readpila(PilaDos);
   inicpila(Resultado, ' ' );
   inicpila(PilaConcat, ' ' );
   DeterminarPilaMenor (PilaUno, PilaDos, Resultado);
   Concatenar(PilaUno, PilaDos, Resultado,PilaConcat );
   writePila (PilaConcat)
end.
```

```
{ definición de procedimientos}


Procedure DeterminarPilaMenor( PilaUno,Pilados:Pila; var Resultado: Pila);
 {Resultado tendra un 1, si PilaUno es menor o igual, un 2 si Pila 2 es menor}
 var
    uno, dos:Pila;
 begin
 while not  pilavacia(PilaUno) and  not pilavacia(PilaDos)do
    begin
         apilar (uno, desapilar(PilaUno));
         apilar (dos, desapilar(PilaDos))
   end;
  if pilavacia(PIlaUno)  then
       inicpila(Resultado, '1')
   else
       inicpila(Resultado, '2')
end;
```

.....

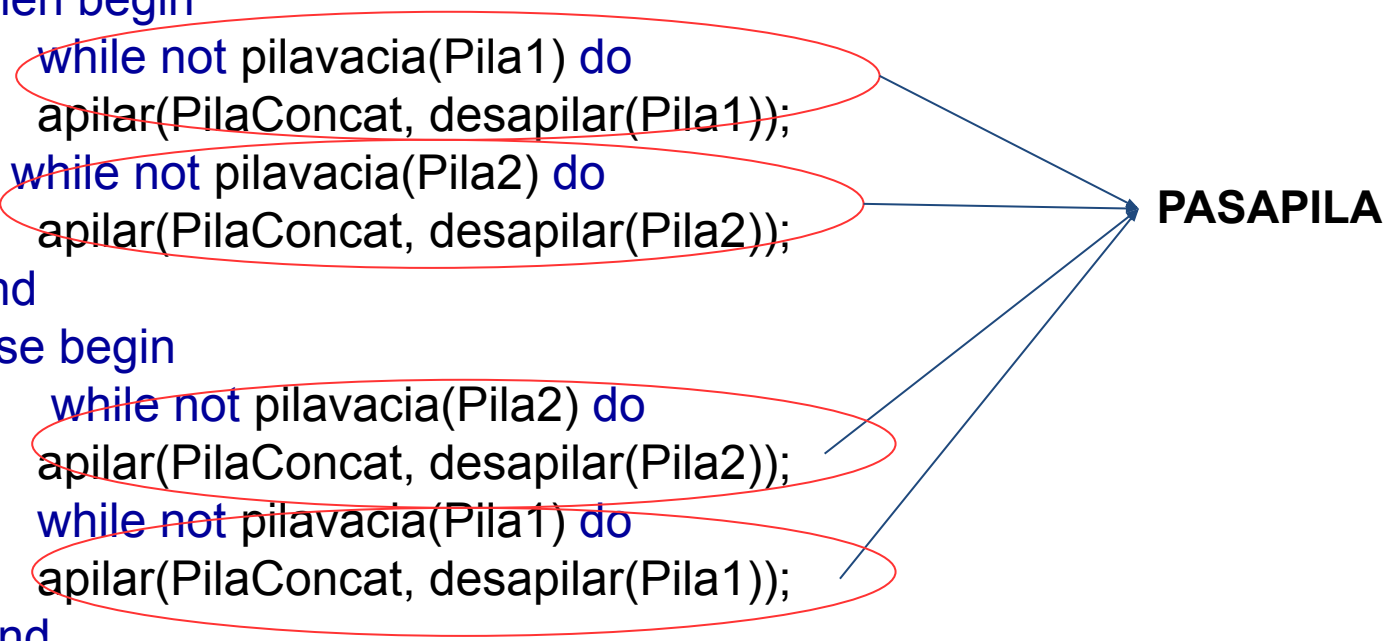
{ definición de procedimientos}

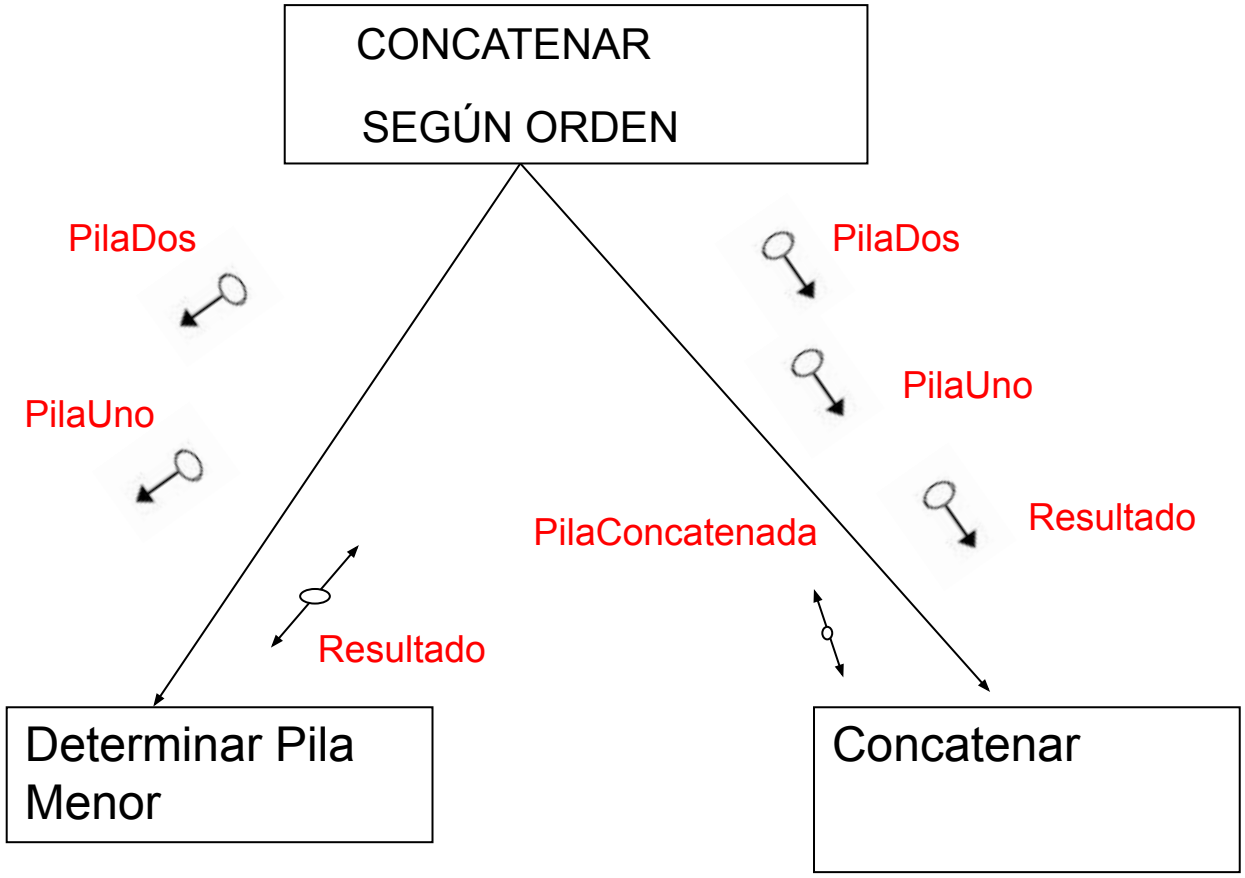
```
Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
   if tope(Resultado)=1    { La pilaUno es menor o igual, con lo cual va abajo}
    then begin
        while not pilavacia(PilaUno) do
        apilar(PilaConcat, desapilar(PilaUno));
       while not pilavacia(PilaDos) do
        apilar(PilaConcat, desapilar(PilaDos));
   end
   else begin
         while not pilavacia(PilaDos) do
         apilar(PilaConcat, desapilar(Pilados));
         while not pilavacia(PilaUno) do
         apilar(PilaConcat, desapilar(PilaUno));
    end
end;
```

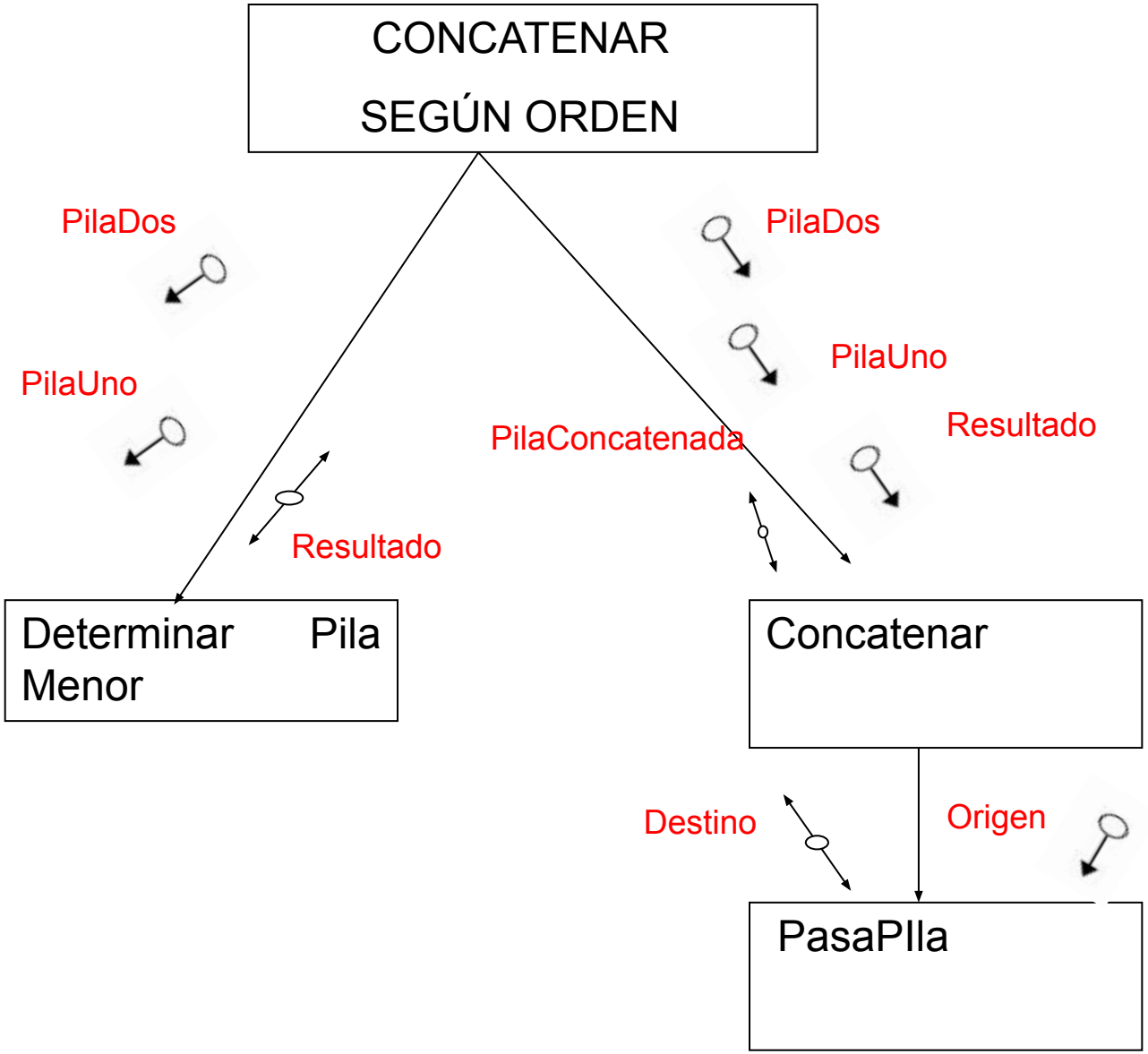
.....

{ definición de procedimientos}

```
Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
  if tope(Resultado)=1    { La pilaUno es menor o igual, con lo cual va abajo}
   then begin
        while not pilavacia(Pila1) do
        apilar(PilaConcat, desapilar(Pila1));
       while not pilavacia(Pila2) do
        apilar(PilaConcat, desapilar(Pila2));
  end
  else begin
         while not pilavacia(Pila2) do
        apilar(PilaConcat, desapilar(Pila2));
        while not pilavacia(Pila1) do
        apilar(PilaConcat, desapilar(Pila1));
   end
end;
```




CONCATENAR
SEGÚN ORDEN
PilaDos
PilaUno
Resultado
PilaDos
PilaUno
Resultado
PilaConcatenada
Determinar Pila Menor
Concatenar




**Otro <u>posible</u> DE para el problema Pco.2 ej. 11**

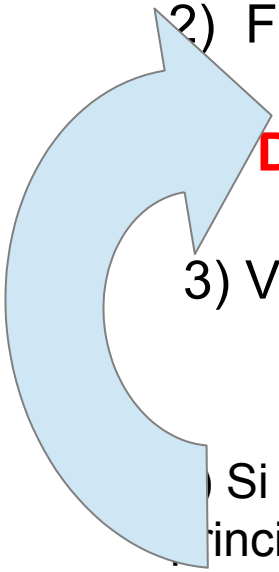
USO DE LA ESTRATEGIA DIVIDE Y CONQUISTA EN RESOLUCION DE PROBLEMAS CON COMPUTARDORA

1) Pensar en la descomposición del problema en subproblemas (¡no hay una única forma!)

2) FORMALIZAR la descomposición:

**DIAGRAMA DE ESTRUCTURA (DE)**

3) Volcar el DE en un programa escrito en Pascal



) Si se agregan/borran procedimientos en el programa rincipal,
SE DEBE actualizar el DE

```
Program ConcatenarSegúnOrden;

{ Definiciones de procedimientos}

Procedure pasarPila( var Destino: Pila; Origen:Pila);
…..................
Procedure  Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
……
Procedure  DeterminarPilaMenor( PilaUno,Pilados:Pila; var Resultado: Pila);
….
{ Programa principal}
var
   PilaUno, pilaDos, Resultado, PilaConcat: Pila;
 begin
   readpila(PilaUno);
   readpila(PilaDos);
   inicpila(Resultado, ' ' );
   inicpila(PilaConcat, ' ' );
   DeterminarPilaMenor (PilaUno, PilaDos, Resultado);
   Concatenar(PilaUno, PilaDos, Resultado,PilaConcat );
   writePila (PilaConcat)
end.
```

.....

{ definición de procedimientos}

```
Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
   if tope(Resultado)=1    { La pilaUno es menor o igual, con lo cual va abajo}
    then begin
       pasarPila(PilaConcat, pilaUno);
       pasarPila(PilaConcat, pilaDos);
    end
    else begin
        pasarPila(PilaConcat, pilaDos);
        pasarPila(PilaConcat, pilaUno);
    end
end;
```

La Modularización permite el REUSO

{continua la defínción de procedimientos }

```
procedure pasarPila( var Destino: Pila; Origen:Pila);
begin
 while not pilavacia(Origen) do
    begin
        apilar(Destino, desapilar(Origen));
    end
 end;
```

La Modularización permite el REUSO

.....

```
procedure pasarPila( var Destino: Pila  ; Origen:Pila);
begin
 while not pilavacia(Origen) do
    begin
         apilar(Destino, desapilar(Origen));
    end
 end;
```

ASIGNACIÓN POSICIONAL

```
Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
   if tope(Resultado)=1
    then begin
       pasarPila(PilaConcat , pilaUno);
       pasarPila(PilaConcat, pilaDos);
    end
    else begin
        pasarPila(PilaConcat, pilaDos);
        pasarPila(PilaConcat, pilaUno);
    end
end;
```

```
.............

procedure pasarPila( var Destino: Pila  ; Origen:Pila);
begin
 while not pilavacia(Origen) do
    begin
         apilar(Destino, desapilar(Origen));
    end
 end;


Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
   if tope(Resultado)=1
    then begin
       pasarPila(PilaConcat  , pilaUno);
       pasarPila(PilaConcat, pilaDos);
    end
    else begin
        pasarPila(PilaConcat, pilaDos);
        pasarPila(PilaConcat, pilaUno);
    end
end;
```

PARÁMETROS FORMALES (definición)

PARÁMETROS REALES (invocación)

```
.............

procedure pasarPila( var Destino: Pila ; Origen:Pila);
begin
 while not pilavacia(Origen) do
    begin
        apilar(Destino, desapilar(Origen));
    end
 end;


Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
   if tope(Resultado)=1
    then begin
       pasarPila(PilaConcat  , pilaUno);
       pasarPila(PilaConcat , pilaDos);
    end
    else begin
        pasarPila(PilaConcat, pilaDos);
        pasarPila(PilaConcat, pilaUno);
    end
end;
```

PARÁMETROS FORMALES (definición)

PARÁMETROS REALES (invocación)

```
.............

procedure pasarPila( var Destino: Pila  ; Origen:Pila);
begin
 while not pilavacia(Origen) do
    begin
         apilar(Destino, desapilar(Origen));
    end
 end;


Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
   if tope(Resultado)=1
    then begin
       pasarPila(PilaConcat  , pilaUno);
       pasarPila(PilaConcat  , pilaDos);
    end
    else begin
        pasarPila(PilaConcat, pilaDos);
        pasarPila(PilaConcat, pilaUno);
    end
end;
```

PARÁMETROS FORMALES (definición)

# Tipo de pasaje de parámetros

**VAR**

(PASAJE POR **REFERENCIA**)
(SE PASA EL OBJETO REAL)
(VUELVE MODIFICADO)

**VAR**

(PASAJE POR **COPIA**)
(SE PASA UNA COPIA DEL OBJETO REAL)
(EL OBJETO REAL NO ES AFECTADO
POR NINGUNA MODIFICACIÓN)

## Relación de parámetros de procedimientos y cuplas del DE

**VAR**

**VAR** 

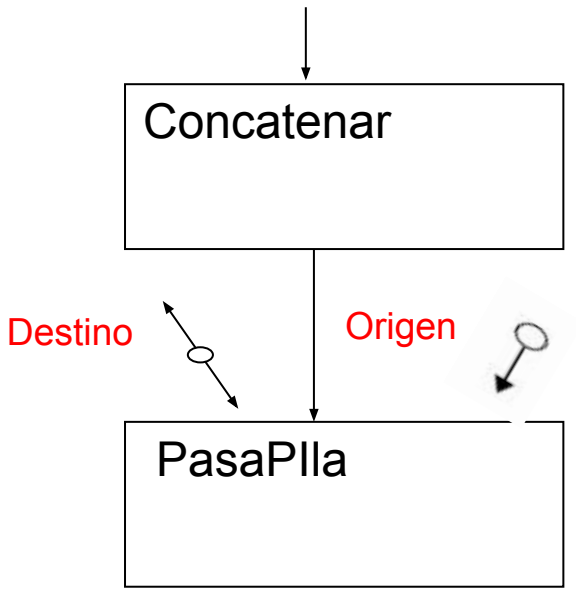
Entrada

Entrada y salida

.....

```
procedure pasarPila( var Destino: Pila  ; Origen:Pila);
begin
 while not pilavacia(Origen) do
    begin
        apilar(Destino, desapilar(Origen));
    end
 end;
```

```
Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
   if tope(Resultado)=1
    then begin
       pasarPila(PilaConcat  , pilaUno);
       pasarPila(PilaConcat  , pilaDos);
    end
    else begin
        pasarPila(PilaConcat, pilaDos);
        pasarPila(PilaConcat, pilaUno);
    end
end;
```

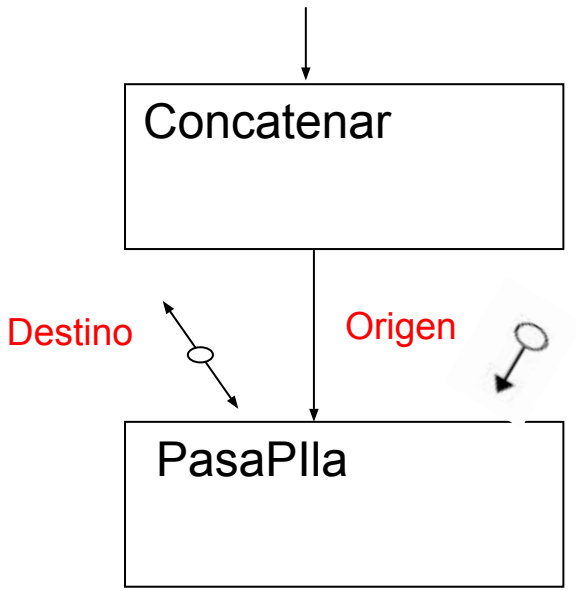



**DE  PARCIAL del  problema Pco.2 ej. 11**

.....

```
procedure pasarPila( var Destino: Pila  ; Origen:Pila);
begin
 while not pilavacia(Origen) do
    begin
        apilar(Destino, desapilar(Origen));
    end
 end;
```

```
Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
   if tope(Resultado)=1
    then begin
       pasarPila(PilaConcat  , pilaUno);
       pasarPila(PilaConcat  , pilaDos);
    end
    else begin
        pasarPila(PilaConcat, pilaDos);
        pasarPila(PilaConcat, pilaUno);
    end
end;
```



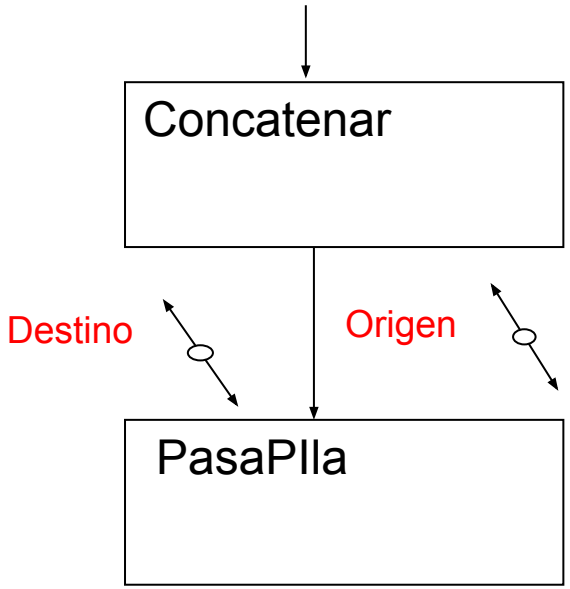

**cómo quedan PIlaUno y PIlaDos?**

**NO SE MODIFICAN**

.....

```
procedure pasarPila( var Destino: Pila  ; var Origen:Pila);
begin
 while not pilavacia(Origen) do
    begin
         apilar(Destino, desapilar(Origen));
    end
 end;
```

```
Procedure Concatenar( var PilaUno, PilaDos, Resultado:Pila; var PilaConcat: Pila);
begin
   if tope(Resultado)=1
    then begin
       pasarPila(PilaConcat  , pilaUno);
       pasarPila(PilaConcat  , pilaDos);
    end
    else begin
        pasarPila(PilaConcat, pilaDos);
        pasarPila(PilaConcat, pilaUno);
    end
end;
```

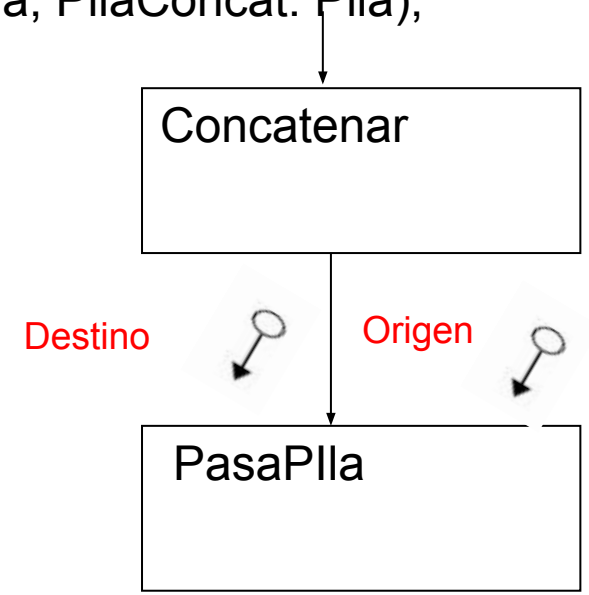



**cómo quedan PilaUno y PIlaDos?** **VACIAS**

.....

```
procedure pasarPila(  Destino: Pila  ; Origen:Pila);
begin
 while not pilavacia(Origen) do
    begin
        apilar(Destino, desapilar(Origen));
    end
 end;
```

```
Procedure Concatenar(PilaUno, PilaDos, Resultado:Pila; PilaConcat: Pila);
begin
   if tope(Resultado)=1
    then begin
       pasarPila(PilaConcat  , pilaUno);
       pasarPila(PilaConcat  , pilaDos);
    end
    else begin
        pasarPila(PilaConcat, pilaDos);
        pasarPila(PilaConcat, pilaUno);
    end
end;
```




**cómo queda PIlaCONCAT?** Queda sin modificar; esta MAL el pasaje por copia