

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

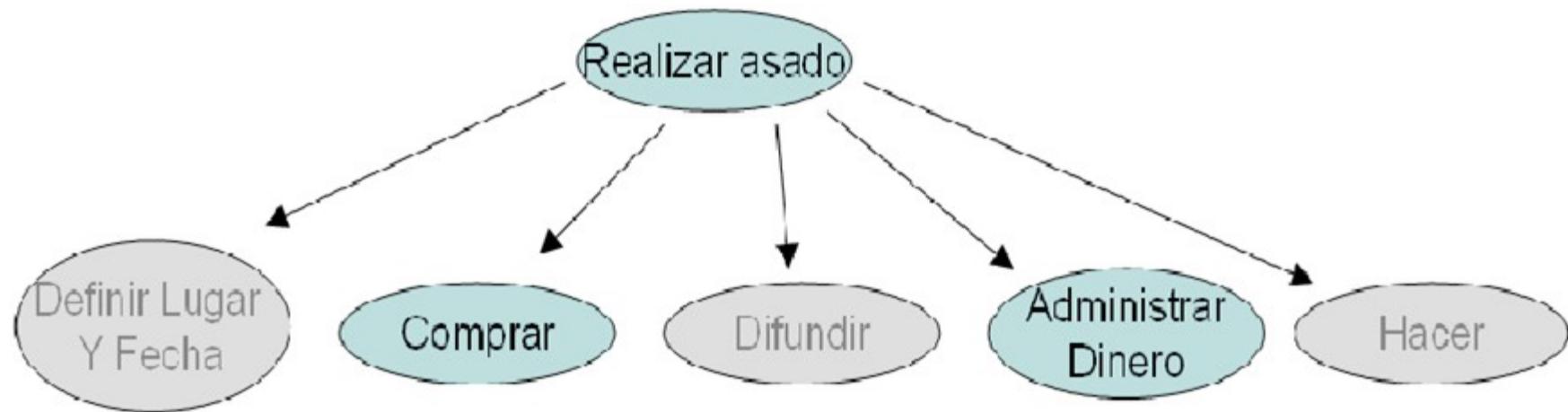
Estrategia: "Divide y Conquista",

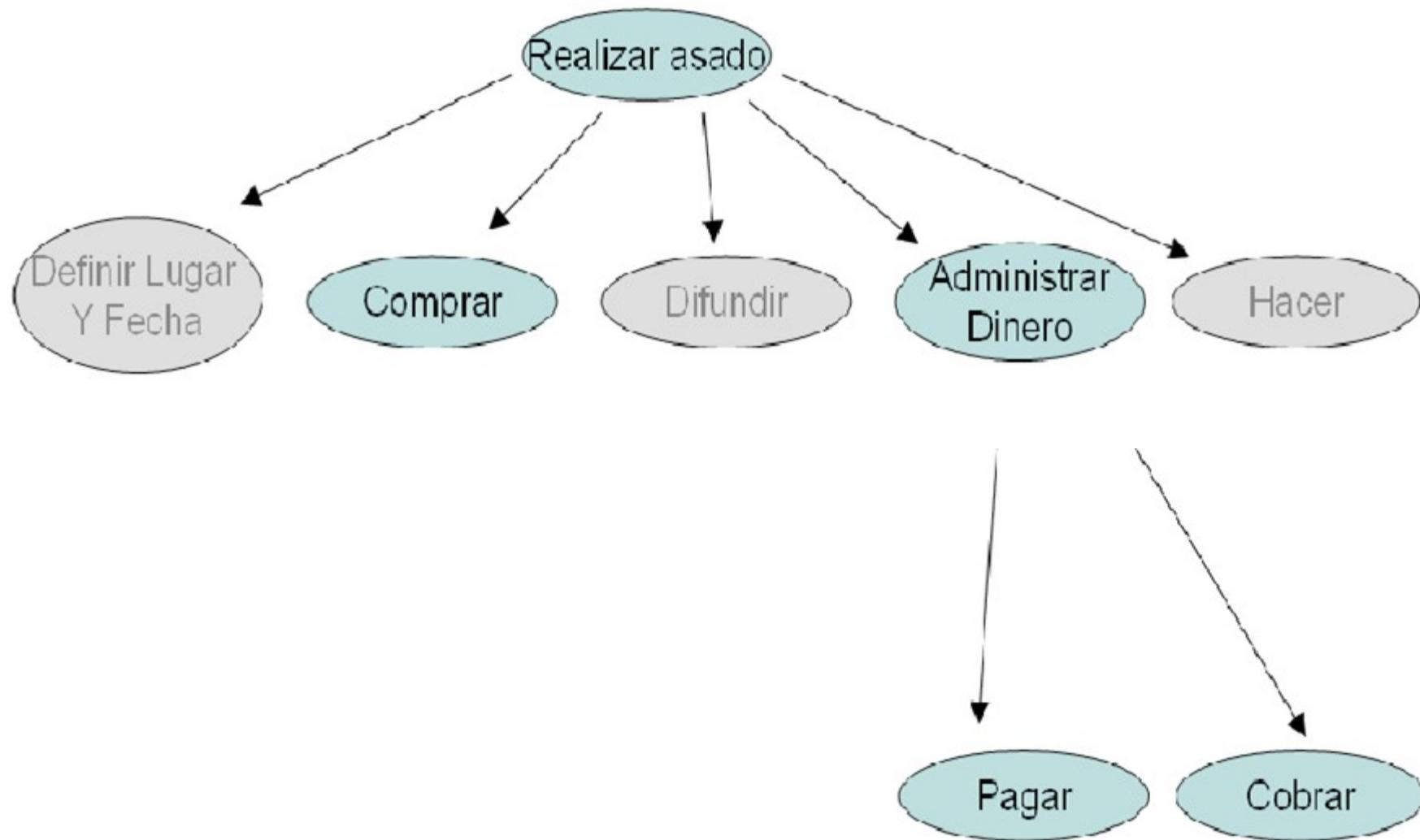
Una de las estrategias más útiles en la resolución de problemas con computadora es la **descomposición de un problema en subproblemas más simples**: “**Divide y Conquista**“.

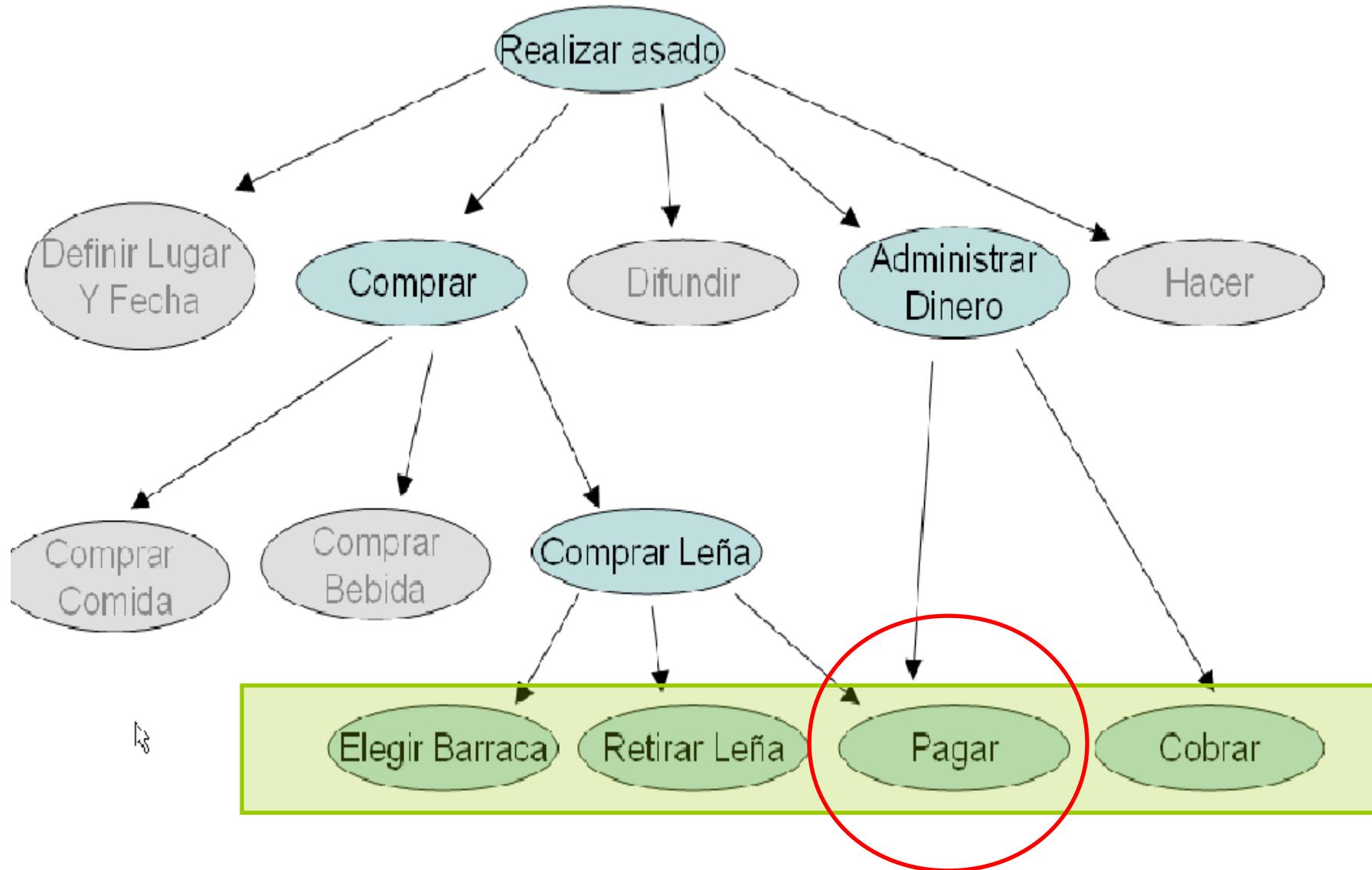
- Cada **problema es dividido en un número de subproblemas más pequeños**, cada uno de los cuales a su vez, puede dividirse en un conjunto de subproblemas más pequeños aún, y así siguiendo.
- Cada uno de estos subproblemas debiera resultar entonces más simple de resolver.
- Una metodología de resolución con estas características se conoce como diseño Top -Down.

**PROBLEMA: REALIZAR UN ASADO
PARA LOS ALUMNOS DE
INTRODUCCIÓN A LA PROGRAMACIÓN**

Realizar asado







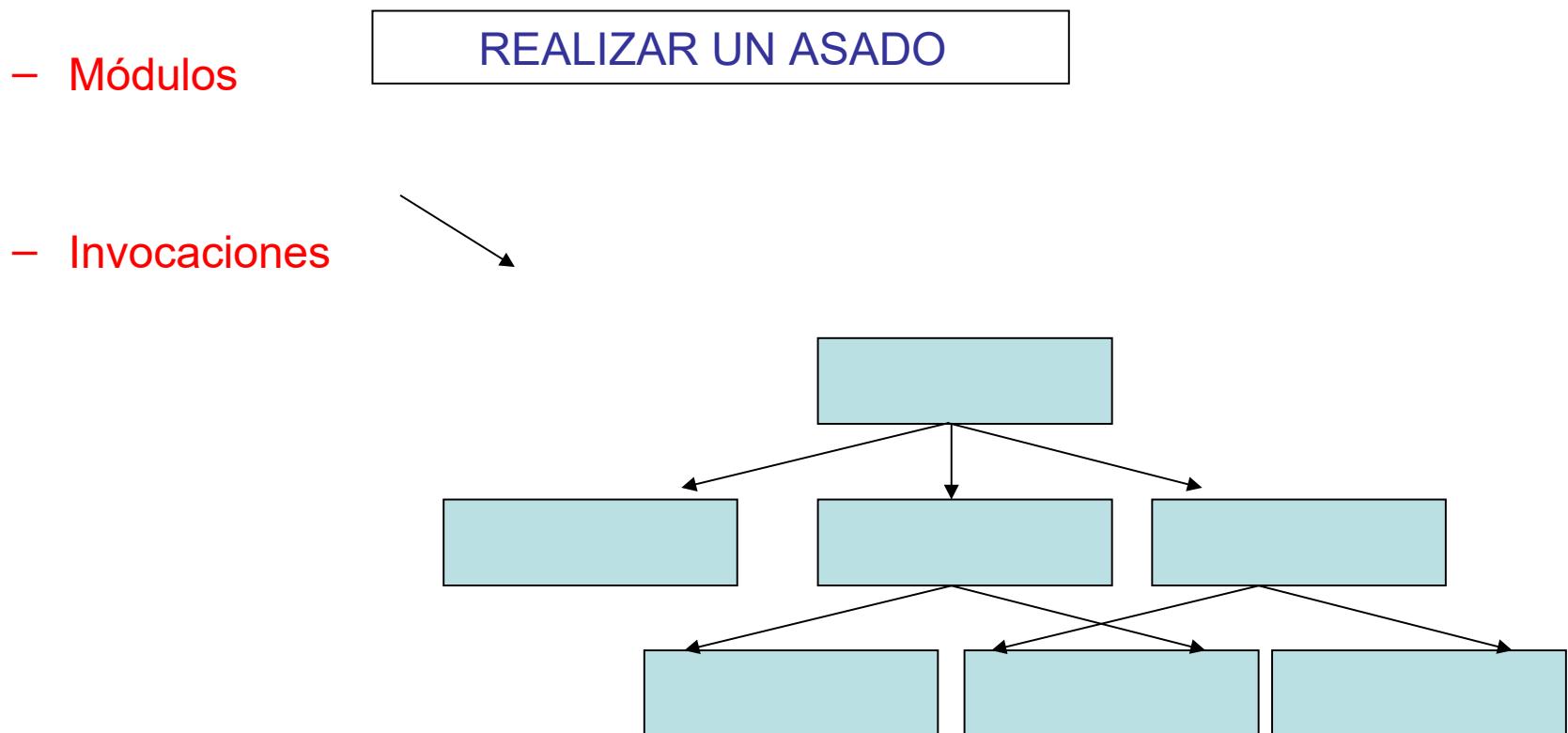
USO DE LA ESTRATEGIA **DIVIDE Y CONQUISTA** EN RESOLUCION DE PROBLEMAS CON COMPUTARDORA

- 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:



REALIZAR UN ASADO

Módulo invocador

invocación

Módulo invocado

DEFINIR LUGAR
Y FECHA

COMPRAR

.....

.....

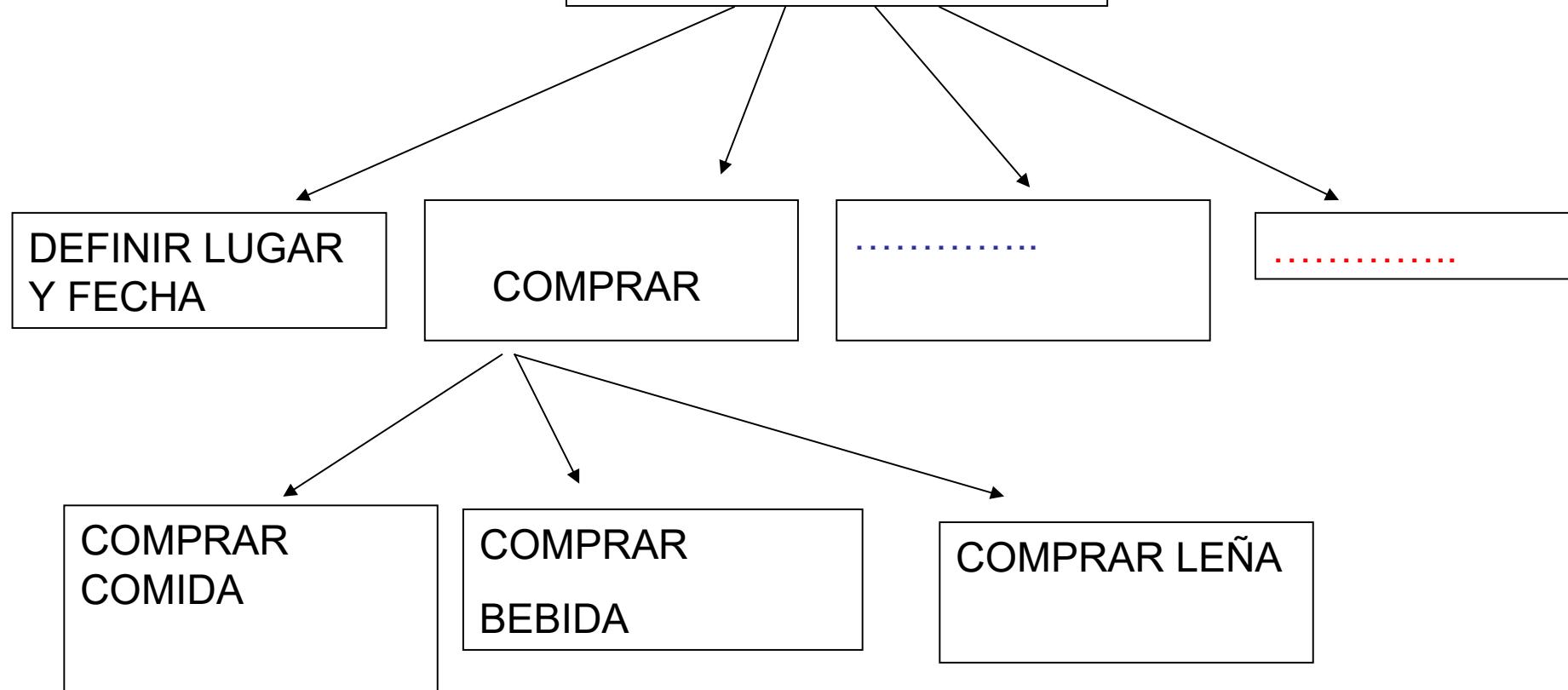
COMPRAR
COMIDA

COMPRAR
BEBIDA

COMPRAR LEÑA

DIAGRAMA de ESTRUCTURA (DE) INCOMPLETO DE “REALIZAR UN ASADO”

REALIZAR UN ASADO



¿El módulo **COMPRAR** necesita saber cuando será el asado?

REALIZAR UN ASADO

FECHA

FECHA

DEFINIR LUGAR
Y FECHA

COMPRAR

.....

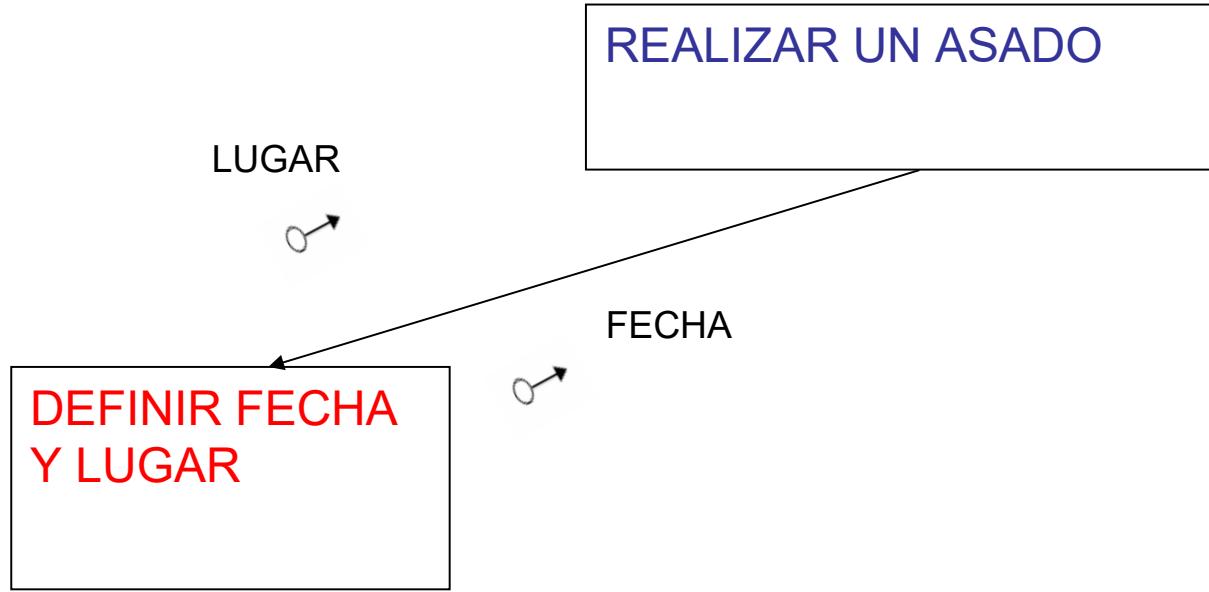
.....

COMPRAR
COMIDA

COMPRAR
BEBIDA

COMPRAR LEÑA

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

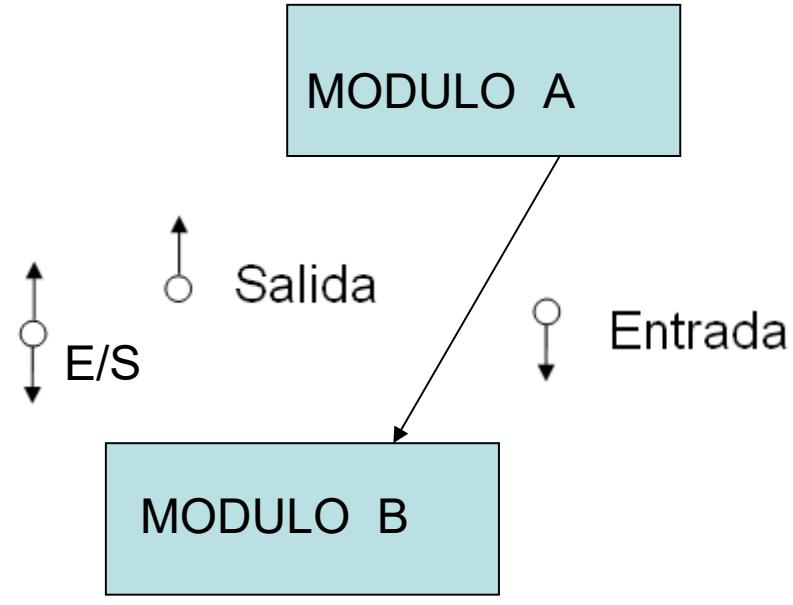
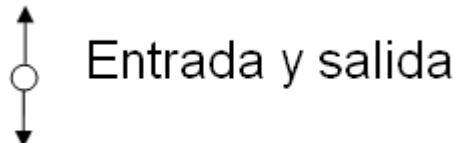
- Módulos



- Invocaciones



- Cuplas



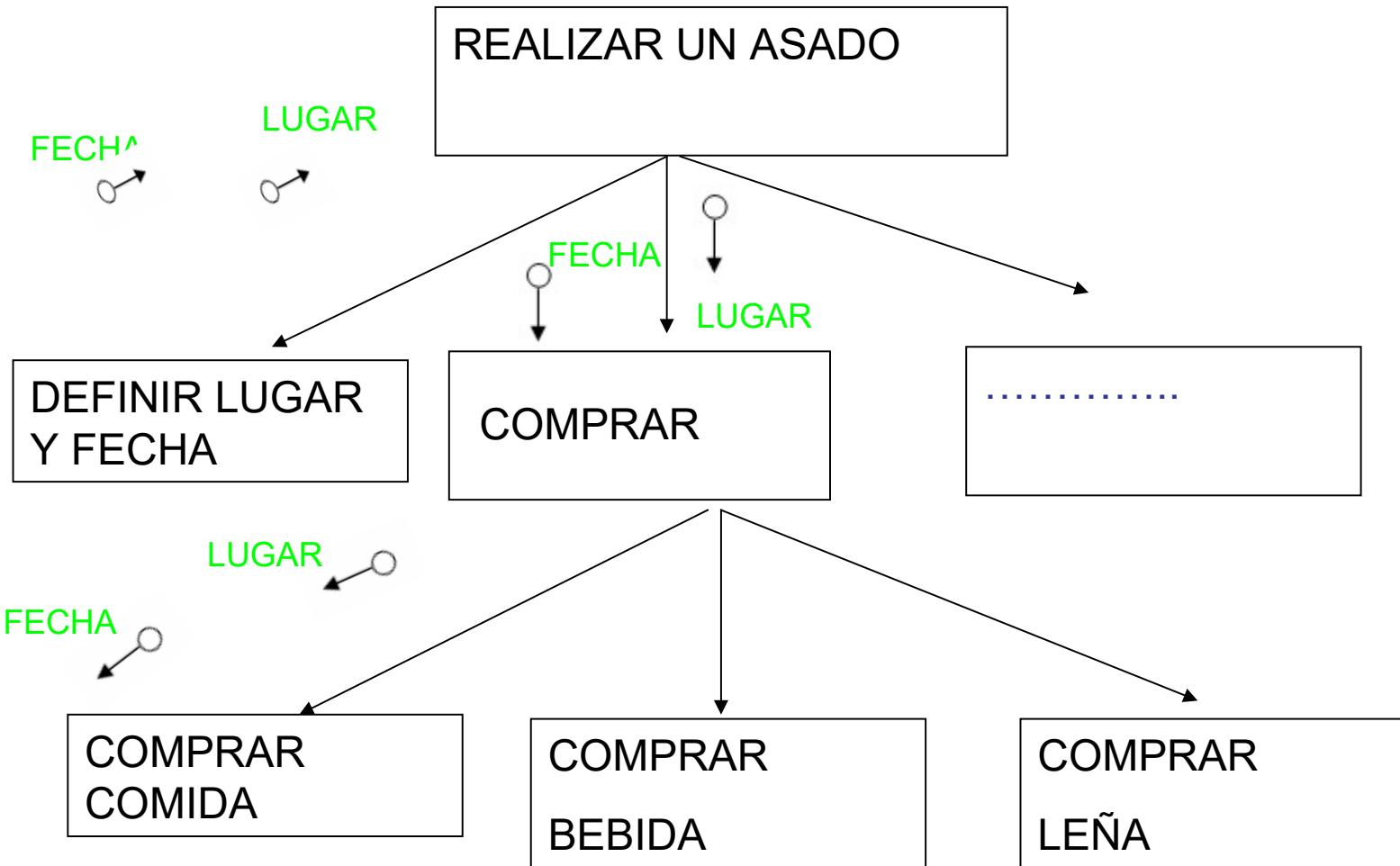


DIAGRAMA de ESTRUCTURA INCOMPLETO DE REALIZAR UN ASADO

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.

```

{ Concatena dos pilas de manera que la pila de abajo queda abajo, si son iguales, Pilas dos queda abajo.
- El orden de los elementos se debe respetar}

uses estructur;
var
    uno, dos, resultado, aux1, aux2: Pila;

begin
    inicPila(uno, '3 4 5');
    inicPila(dos, '6 7 8 9 10 ');
    inicPila (Resultado, ' ');
    inicPila (aux1, ' ');
    inicPila (aux2, ' ');

    While (not PilaVacia(uno) and not PilaVacia(dos)) do
        begin
            apilar(Aux1,desapilar(uno));
            apilar(Aux2, desapilar(dos));
        end;

    If not PilaVacia(uno) then
        begin
            { termino de pasar los datos a aux1 para luego pasarlo a resultado en orden y que la pila Uno quede abajo}
            While (not PilaVacia(uno)) do
                apilar(Aux1, desapilar(uno));
            While (not PilaVacia(Aux1)) do
                apilar(Resultado, desapilar(aux1));
            { termino de pasar los datos a aux2 para luego pasarlo a resultado en orden y que la pila Dos quede arriba}
            While (not PilaVacia(dos)) do
                apilar(Aux2, desapilar(dos));
            While (not PilaVacia(Aux2)) do
                apilar(Resultado, desapilar(aux2));
        end;
end.

```

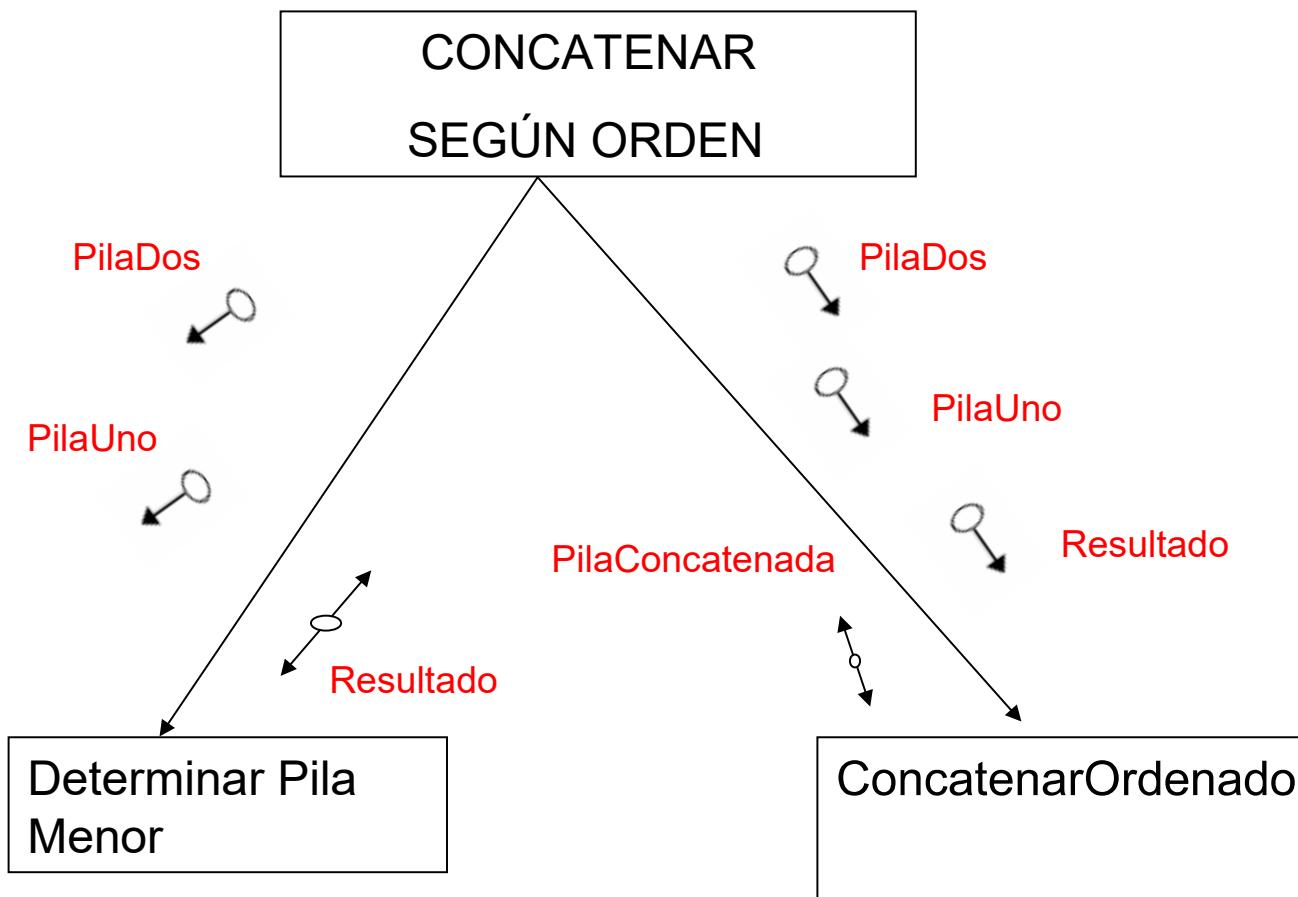
Fragmento de una posible solución al ejercicio

- 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.

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

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)

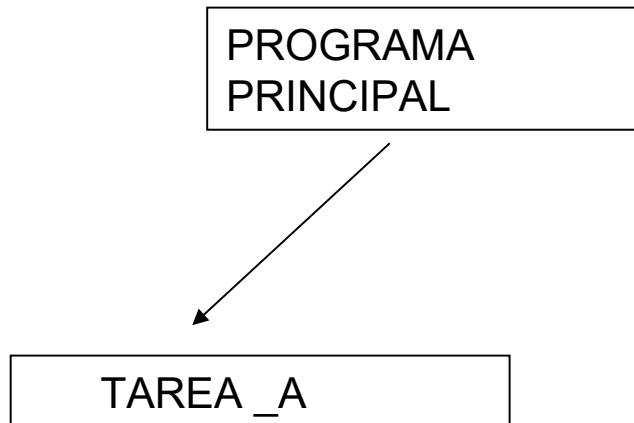
- 3) Volcar el DE en un programa escrito en Pascal

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**.

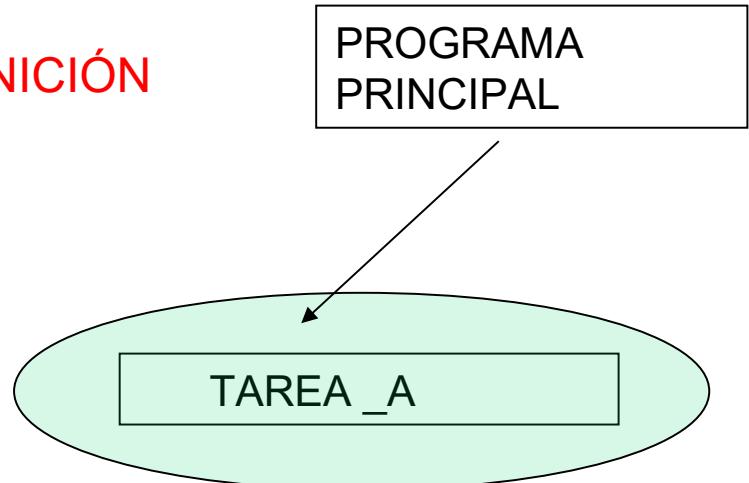
```
PROGRAM PROGRAMAPRINCIPAL  
{ Comentario del programa principal}
```

```
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.
```

PROGRAMA
PRINCIPAL

INVOCACIÓN

TAREA_A

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}

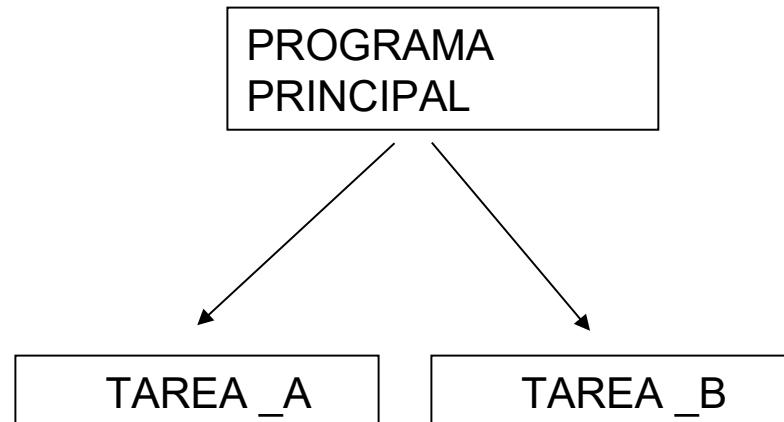
.....

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

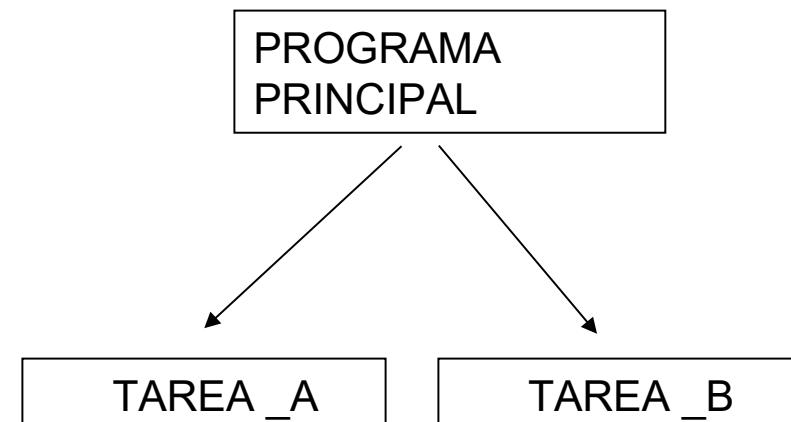
 TAREA_A (...)

ELSE

 TAREA_B (.....);

.....

End.



CUPLAS del DE

PARÁMETROS

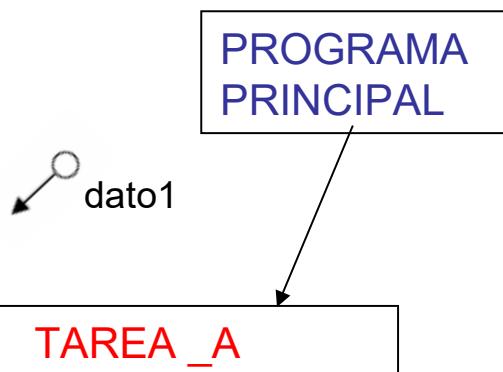
entre los procedimientos y PP

PROGRAMA 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

PROGRAMA PRINCIPAL
{ Comentario del programa principal}

Procedure TAREA_A (origen:Pilas);
Begin

....

End;

....

Var {del programa principal}
Origen: Pila;

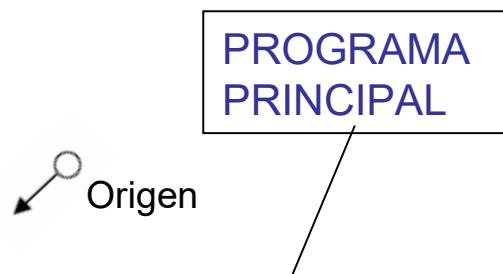
Begin {del programa principal}
readPila(Origen);

....

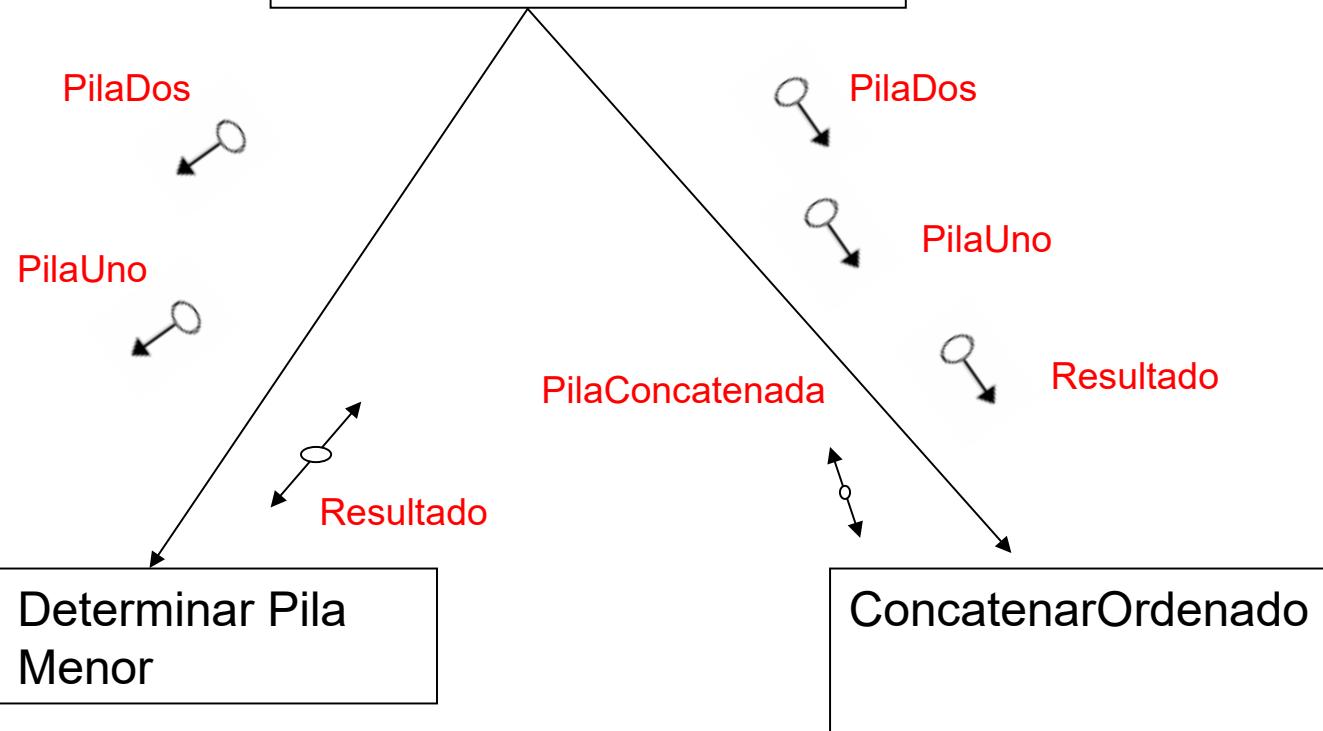
TAREA_A (Origen);

...

end.



CONCATENAR SEGÚN ORDEN



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

Program ConcatenarSegúnOrden
{ este programa

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);

inicipila(Resultado, ' ');

inicipila(PilaConcat, ' ');

DeterminarPilaMenor (PilaUno, PilaDos, Resultado),

Concatenar(PilaUno, PilaDos, Resultado,PilaConcat);

writePila (PilaConcat)

end.

Program ConcatenarSegúnOrden
{ este programa

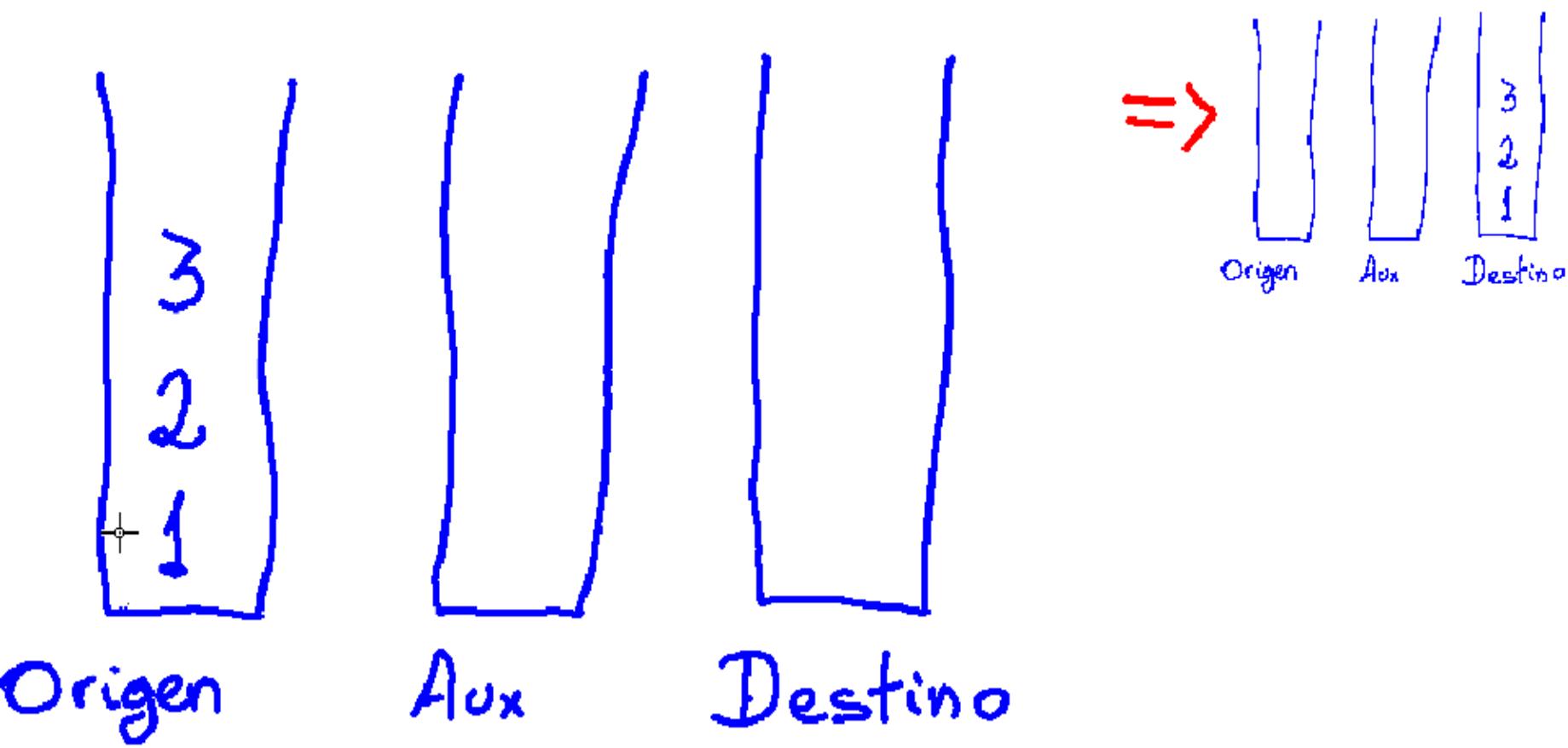
Procedure DeterminarPilaMenor (PilaUno, PilaDos: Pila , var Resultado: Pila);
.....

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

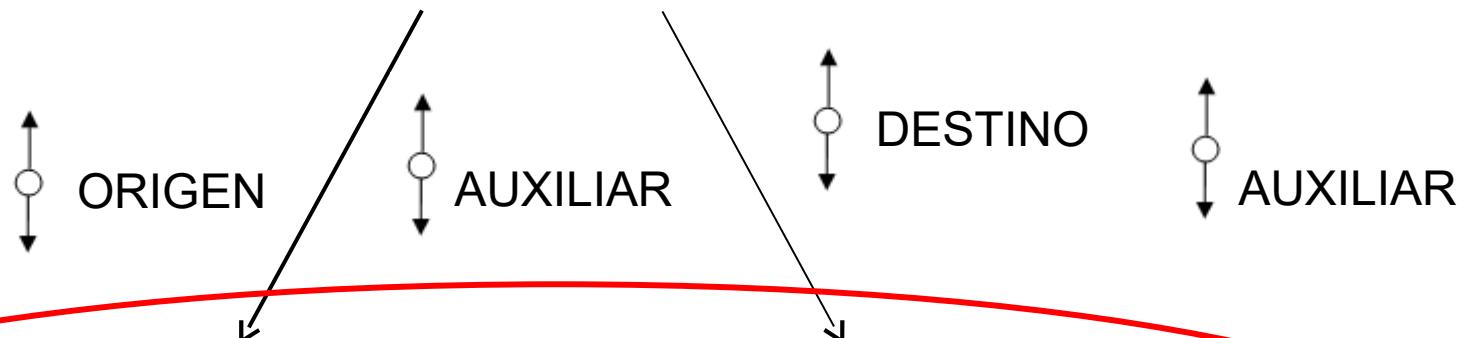
var {programa principal}
PilaUno, pilaDos, Resultado, PilaConcat: Pila;

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

Problema: Pasar los elementos de la pila origen a destino y que queden en el mismo orden



PASA DATOS ORDENADOS



PASA PILA ORIGEN A
AUXILIAR

PASA PILA AUXILIAR
A DESTINO

Program PasaDatosOrdenados;
{la pila Origen es pasada a la pila destino quedando los elementos en el mismo orden}
uses Estructu;

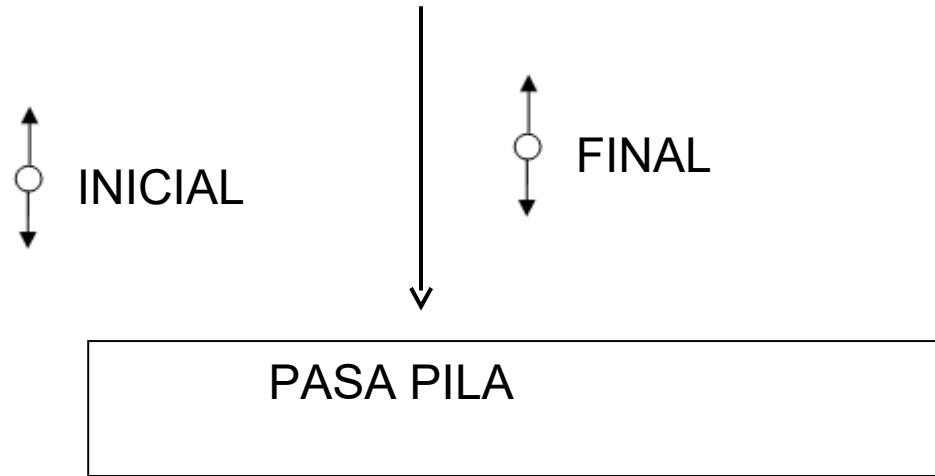
Procedure PasaOrigenAuxiliar(var Origen: Pila; var Aux:Pila);
begin
while not PilaVacia(Origen) do
Apilar (Aux, Desapilar(Origen))
End;

Procedure PasaAuxiliarDestino(var Aux: Pila; var destino:Pila);
begin
while not PilaVacia(Aux) do
Apilar (Destino, Desapilar(Aux));
End;

var
Origen, Destino, Aux: Pila;

Begin
ReadPila(Origen);
InicPila(Destino, ' ');
InicPila(Aux, ' ');
PasaOrigenAuxiliar(Origen, Aux);
PasaAuxiliarDestino(Aux, destino);
WritePila(Origen);
WritePila(Destino);
Readln();
end.

PASA DATOS ORDENADOS



La modularización facilita el REUSO:

El módulo PASAPILA es invocado dos veces desde el Programa Principal

*C:\fpc\bin\PasaOrdenadoconProc.pas - Notepad++

Archivo Editar Buscar Ver Formato Lenguaje Configurar Macro Ejecutar TextFX Plugins Venta

EjemploArchivo.pas ArchivoBasico.pas LeerArchivoBasico.pas PasaOrdenadoconProc.pas

```
1 Program PasaDatosOrdenados;
2   (Dada la pila Origen este porgrama los pasa a la pila
3     destino quedando en el mismo orden que estaban en Origen )
4
5 uses Estructu;
6 procedure PasaPila(var Inicial:Pila; var Final:Pila);
7 begin
8   while not PilaVacia(Inicial) do
9     Apilar (Final, Desapilar(Inicial));
10 end;
11
12 var
13   Origen, Destino, Aux: Pila;
14
15 Begin
16   ReadPila(Origen);
17   InicPila(Destino, ' ');
18   InicPila(Aux, ' ');
19   PasaPila(Origen, Aux);
20   PasaPila(Aux, Destino);
21   WritePila(Origen);
22   WritePila(Destino);
23   readln();
24 end.
```

The diagram illustrates the data flow between variables in the `PasaPila` procedure. It shows five variables represented by green ovals:

- `Inicial`: Located at the top left of the oval.
- `Final`: Located at the top right of the oval.
- `Origen`: Located at the bottom left of the oval.
- `Aux`: Located at the bottom center of the oval.
- `Destino`: Located at the bottom right of the oval.

Red arrows indicate the flow of data between these variables:

- An arrow points from `Inicial` to `Final`.
- An arrow points from `Origen` to `Aux`.
- An arrow points from `Aux` to `Destino`.
- An arrow points from `Origen` to `Destino`.

*C:\fpc\bin\PasaOrdenadoconProc.pas - Notepad++

Archivo Editar Buscar Ver Formato Lenguaje Configurar Macro Ejecutar TextFX Plugins Venta

EjemploArchivo.pas ArchivoBasico.pas LeerArchivoBasico.pas PasaOrdenadoconProc.pas

```
1  Program PasaDatosOrdenados;
2  {Dada la pila Origen este porgrama los pasa a la pila
3  destino quedando en el mismo orden que estaban en Origen }
4
5  uses Estructu;
6  procedure PasaPila(var Inicial:Pila; var Final:Pila);
7  begin
8    while not PilaVacia(Inicial) do
9      Apilar (Final, Desapilar(Inicial));
10 end;
11
12 var
13   Origen, Destino, Aux: Pila;
14
15 Begin
16   ReadPila(Origen);
17   InicPila(Destino, ' ');
18   InicPila(Aux, ' ');
19   PasaPila(Origen, Aux);
20   PasaPila(Aux, Destino);
21   WritePila(Origen);
22   WritePila(Destino);
23   readln();
24 end.
```

Parámetros Formales:
(DEFINICIÓN)

Asignación posicional

Parámetros Reales (INVOCACIÓN)

Tipo de pasaje de parámetros

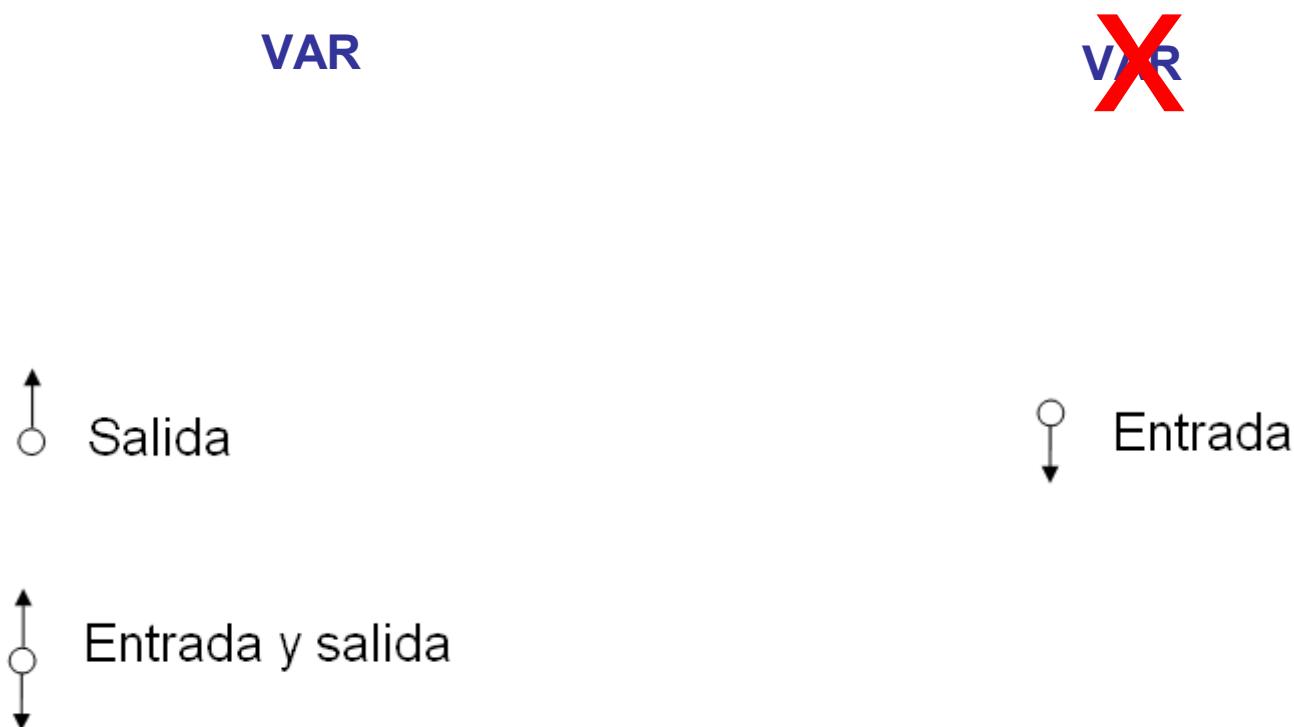
VAR

(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

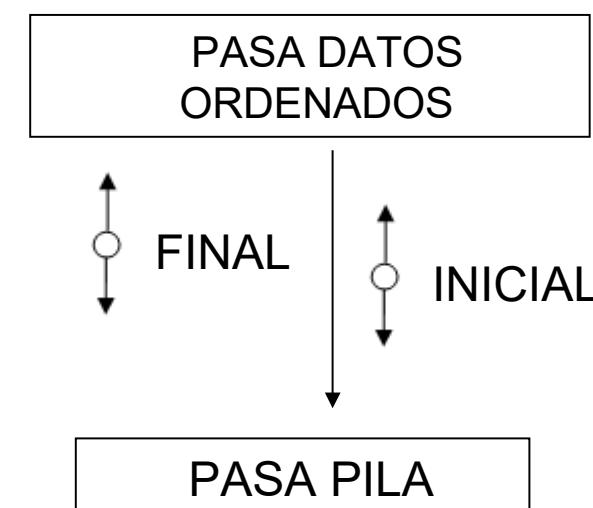


*C:\fpc\bin\PasaOrdenadoconProc.pas - Notepad++

Archivo Editar Buscar Ver Formato Lenguaje Configurar Macro Ejecutar TextFX Plugins Venta

EjemploArchivo.pas ArchivoBasico.pas LeerArchivoBasico.pas PasaOrdenadoconProc.pas

```
1  Program PasaDatosOrdenados;
2  {Dada la pila Origen este porgrama los pasa a la pila
3  destino quedando en el mismo orden que estaban en Origen }
4
5  uses Estructu;
6  procedure PasaPila(var Inicial:Pila; var Final:Pila);
7 begin
8     while not PilaVacia(Inicial) do
9         Apilar (Final, Desapilar(Inicial));
10    end;
11
12    var
13        Origen, Destino, Aux: Pila;
14
15 Begin
16     ReadPila(Origen);
17     InicPila(Destino, ' ');
18     InicPila(Aux, ' ');
19     PasaPila(Origen, Aux);
20     PasaPila(Aux, Destino);
21     WritePila(Origen);
22     WritePila(Destino);
23     readln();
24 end.
```

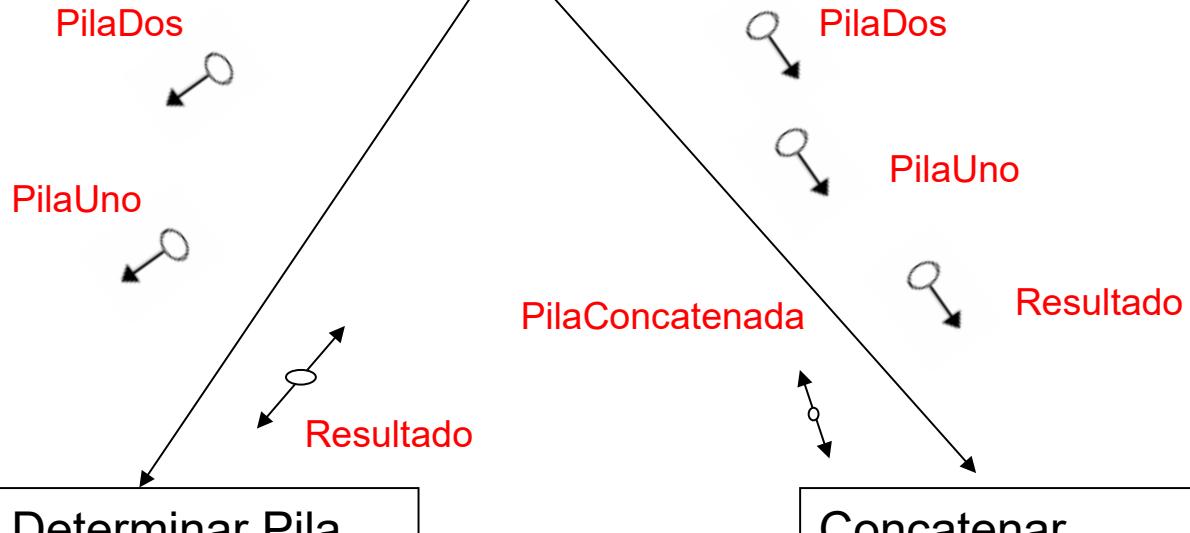


¿ Cómo queda la Pila Origen ?

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.

CONCATENAR SEGÚN ORDEN



Program ConcatenarSegúnOrden

{ este programa

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);

inicipila(Resultado, ' ');

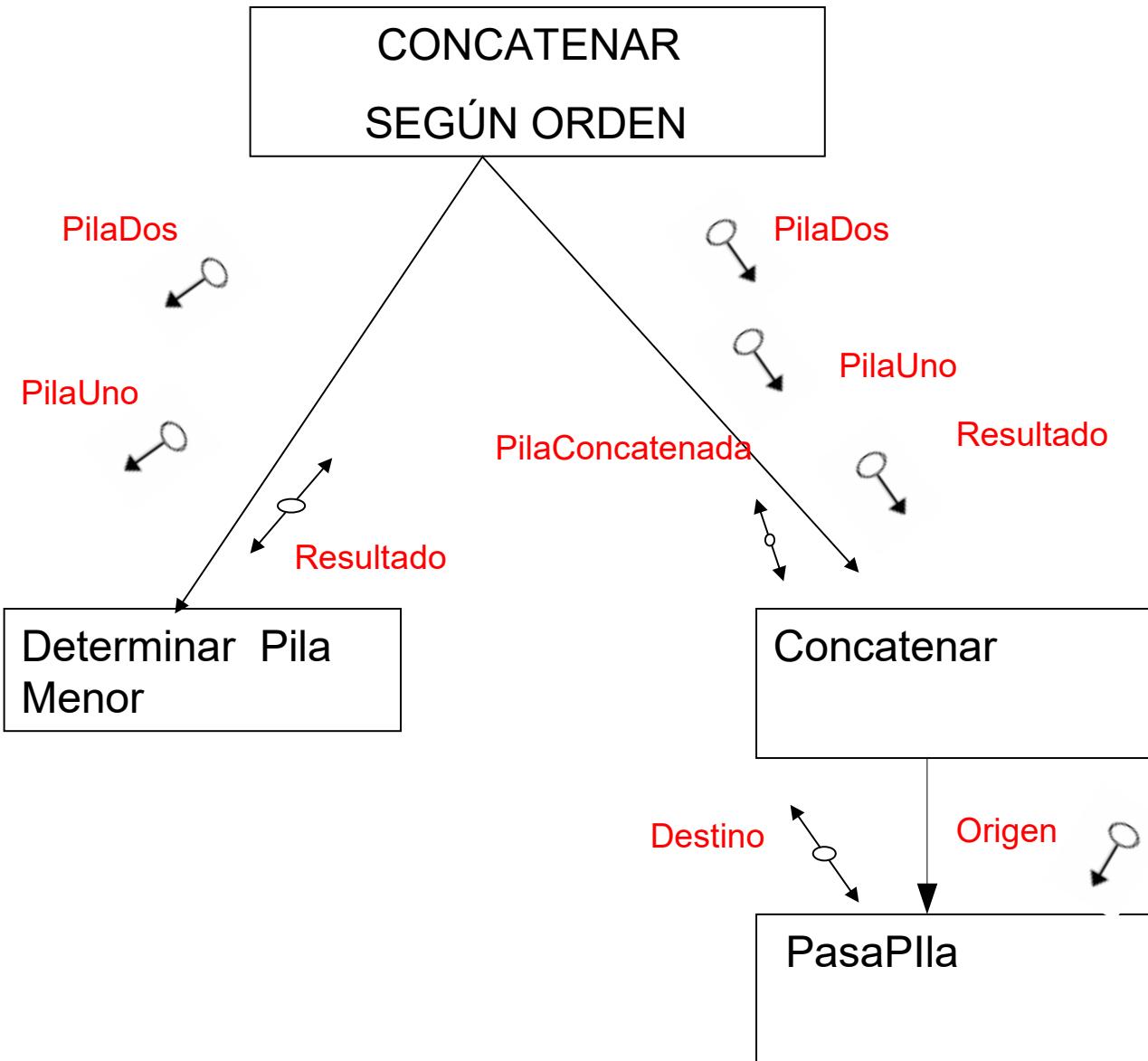
inicipila(PilaConcat, ' ');

DeterminarPilaMenor (PilaUno, PilaDos, Resultado);

Concatenar(PilaUno, PilaDos, Resultado,PilaConcat);

writePila (PilaConcat)

end.



Otro possible DE para el problema Pco.2 ej. 11

Program ConcatenarSegúnOrden
{ este programa}

.....

```
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
      pasarPila(PilaConcat, pilaUno);
      pasarPila(PilaConcat, pilaDos);
    end
  else begin
    pasarPila(PilaConcat, pilaDos);
    pasarPila(PilaConcat, pilaUno);
  end
end;
```

{continua la definición de procedimientos }

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

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, ' ');

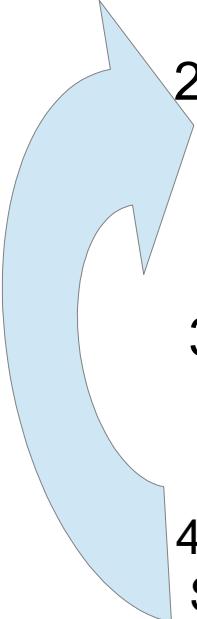
DeterminarPilaMenor (PilaUno, PilaDos, Resultado);

Concatenar(PilaUno, PilaDos, Resultado,PilaConcat);

writePila (PilaConcat)

end.

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
 - 4) Si se **agregan/borran** procedimientos en el programa principal,
SE DEBE actualizar el DE