
**Departamento de Informática**
**Universidad de Valladolid**
**Campus de Segovia**

# TEMA 1: RECURSIÓN

# ÍNDICE

- Definición
- Conceptos básicos
- Ejemplos recursivos
- Recursión mútua
- Recursión e iteración

# DEFINICIÓN

- Técnica de programación que describe los cálculos o las acciones de una manera **autoalusiva**.

- Se puede considerar como un caso particular de programación en el que se plantea la resolución en términos de subproblemas más sencillos.

  – Por esta razón en Pascal se emplean subprogramas para su resolución (subprogramas autoalusivos)

# EJEMPLO DE PROBLEMA RECURSIVO

- FACTORIAL DE UN NÚMERO ENTERO POSITIVO

$$n! = n\,(n-1)\,(n-2).....1$$

$$(n-1)! = (n-1)\,(n-2).....1$$

de donde:

Ley de recurrencia:

$$n! = n\,(n-1)! \quad \text{si } n>0$$

$$0! = 1 \quad \text{si } n=0$$

Fac(4)=4*Fac(3)

Fac(4)=4*(3*Fac(2))

Fac(4)=4*(3*(2*Fac(1)))

Fac(4)=4*(3*(2*(1*Fac(0))))

Fac(4)=4*(3*(2*(1*1)))

Fac(4)=4*(3*(2*1))

Fac(4)=4*(3*2)

Fac(4)=4*6=24

# IMPLEMENTACIÓN EN PASCAL MEDIANTE SUBPROGRAMAS

```
FUNCTION Fac(numero:integer):integer;
{Prec. numero≥0}
{Dev. Numero!}
Begin
    if numero = 0 then
            Fac := 1
    else
            Fac := numero * Fac(numero-1)
End; {Fac}
```

- El identificador de una función puede aparecer dentro de su cuerpo para poder permitir la recursión

- Fac(numero-1) crea una replica del subprograma con el nuevo parámetro.

# CONCEPTOS BÁSICOS

- Los subprogramas recursivos se caracterizan por que se invocan a sí mismos.

- Para poder definir recursivamente un problema es necesario que exista:

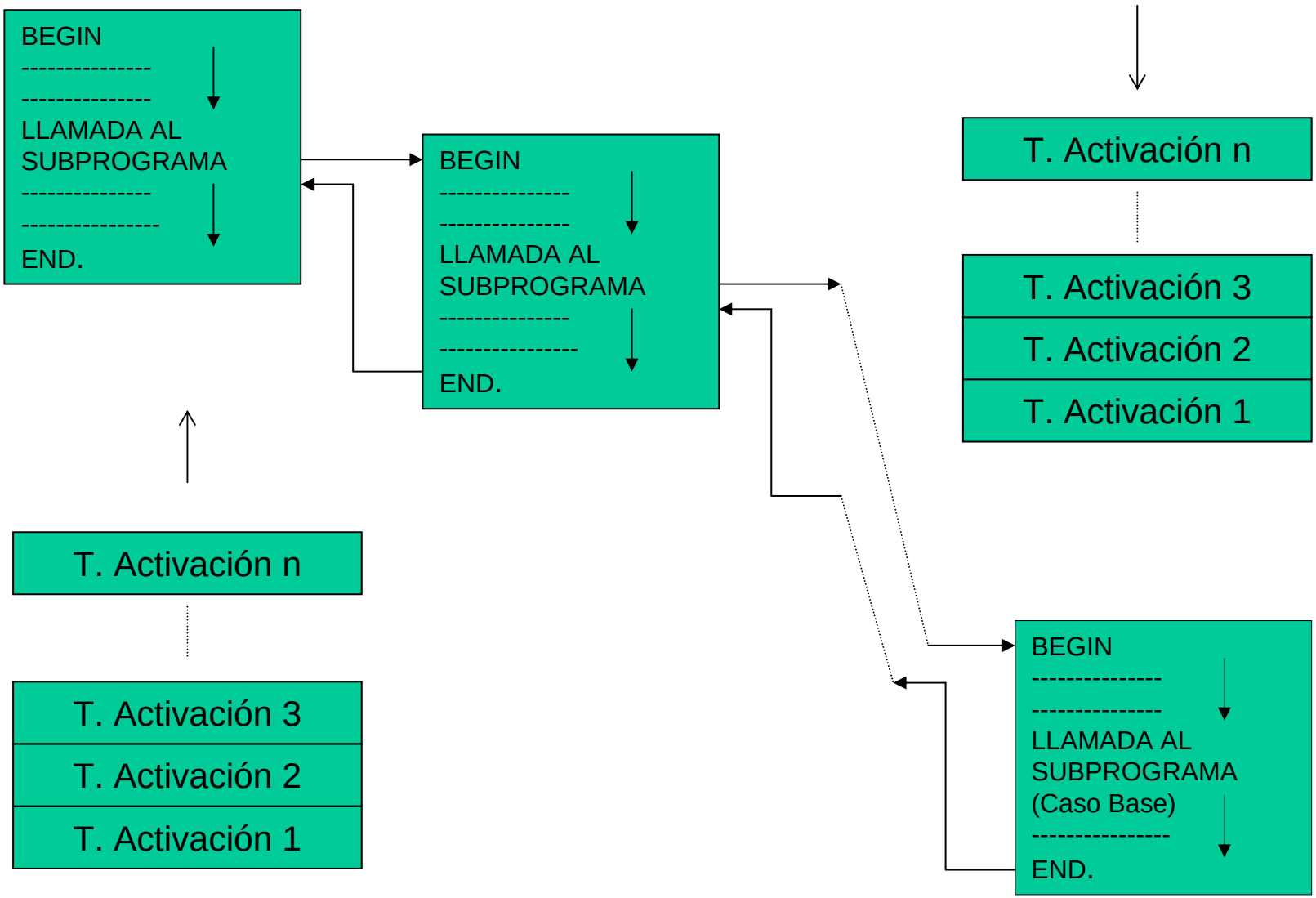
  – Al menos un valor del parámetro sobre el que se hace la recursión que no provoca un nuevo cálculo recursivo: **caso base.**

  – Un conjunto de casos llamados **recurrentes** que son los que si producen un nuevo cálculo recursivo.

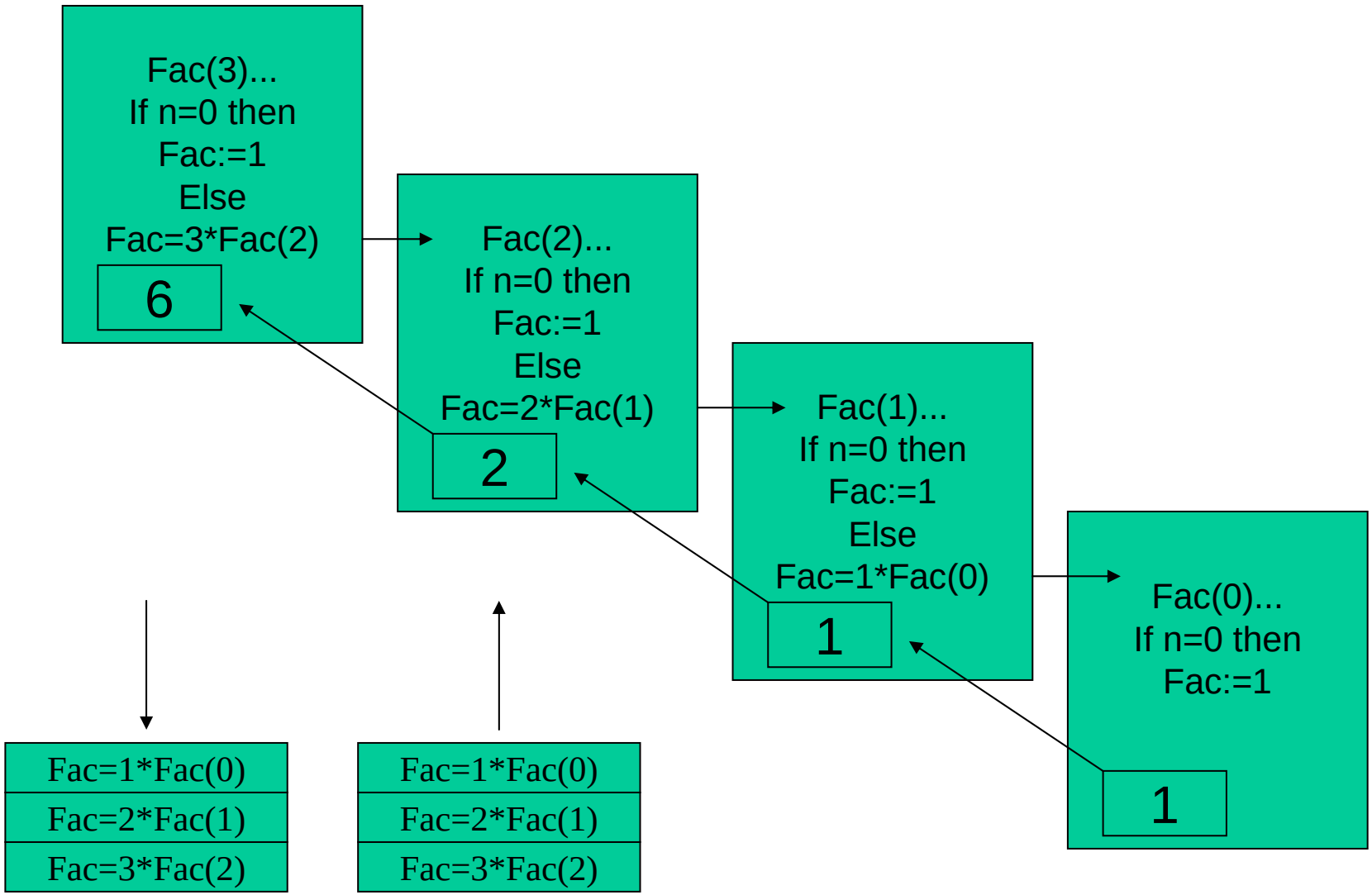
# PROCESO DE EJECUCIÓN. PILA RECURSIVA

- **Ejecución del programa**:
  - Bajo una llamada recursiva  el sistema reserva espacio (**tabla de activación**) donde almacenar una copia de los objetos locales y parámetros del subprograma en ese momento.
  - La tabla de activación se amontona sobre las llamadas recursivas anteriores formando lo que se conoce como **pila recursiva**.
  - Este proceso termina cuando un nuevo valor del parámetro no produce una nueva llamada recursiva (**se alcanza el caso base**).
  - Una vez alcanzada esta situación el sistema va liberando el espacio reservado conforme los subprogramas se van ejecutando sobre su tabla de activación.
  - Este proceso finaliza con la llamada inicial.

# REPRESENTACIÓN GRÁFICA DEL PROCESO DE EJECUCIÓN DE UN PROGRAMA RECURSIVO



# PROCESO DE EJECUCIÓN DE UN SUBPROGRAMA RECURSIVO

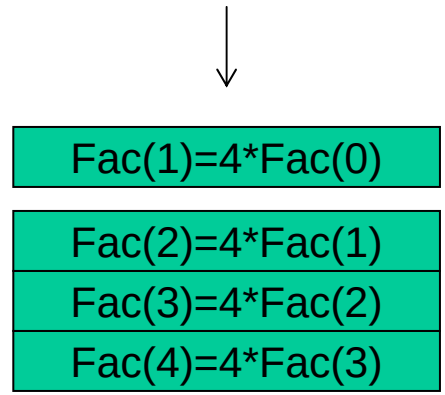


Fac(4)=4*Fac(3)

Fac(4)=4*(3*Fac(2))

Fac(4)=4*(3*(2*Fac(1)))

Fac(4)=4*(3*(2*(1*Fac(0))))



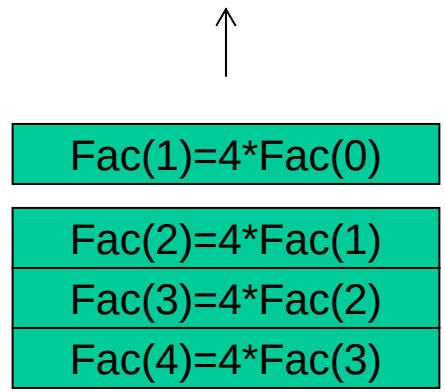

- Para obtener la solución final se deshacen las llamadas anteriores siguiendo el orden inverso (pila recursiva).

Fac(4)=4*(3*(2*(1*1)))

Fac(4)=4*(3*(2*1))

Fac(4)=4*(3*2)

Fac(4)=4*6=24



# EJEMPLOS RECURSIVOS

- La función de Fibonacci:

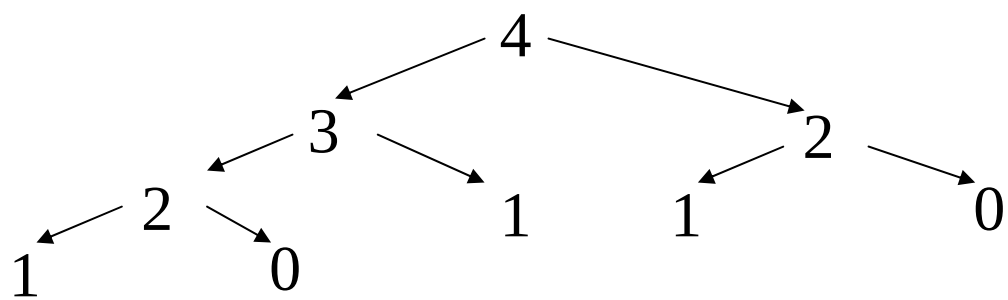
  - f(n)= 1 si n=1,0
  - f(n)= f(n-1)+f(n-2) si n>1

- Implementación en Pascal:

```
FUNCTION fib(numero:integer):integer;
{Prec. numero≥0}
{Dev.  Fibonacci de un número}
Begin
    if (numero=0) or (numero=1) then
            fib:=1
    else
            fib:=fib(numero-1)+fib(numero-2)
End; {fibonacci}
```

# TABULACIÓN DE SUBPROGRAMAS RECURSIVOS

- En ocasiones una llamada recursiva genera por diferentes vías llamadas repetidas.
- Función de Fibonacci: Fib(4)



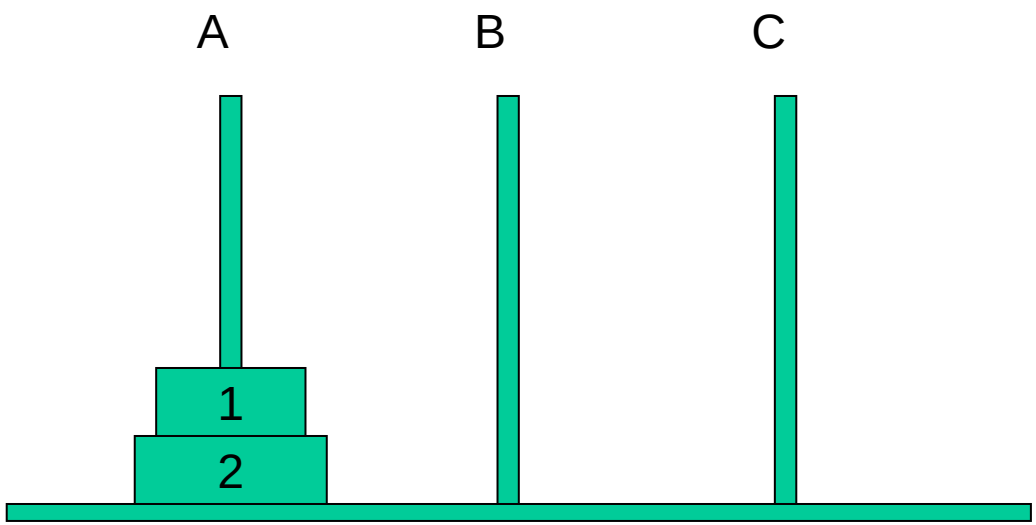

- La solución consiste en dotar a la función de memoria para recordar los valores que ya ha calculado (tabla).

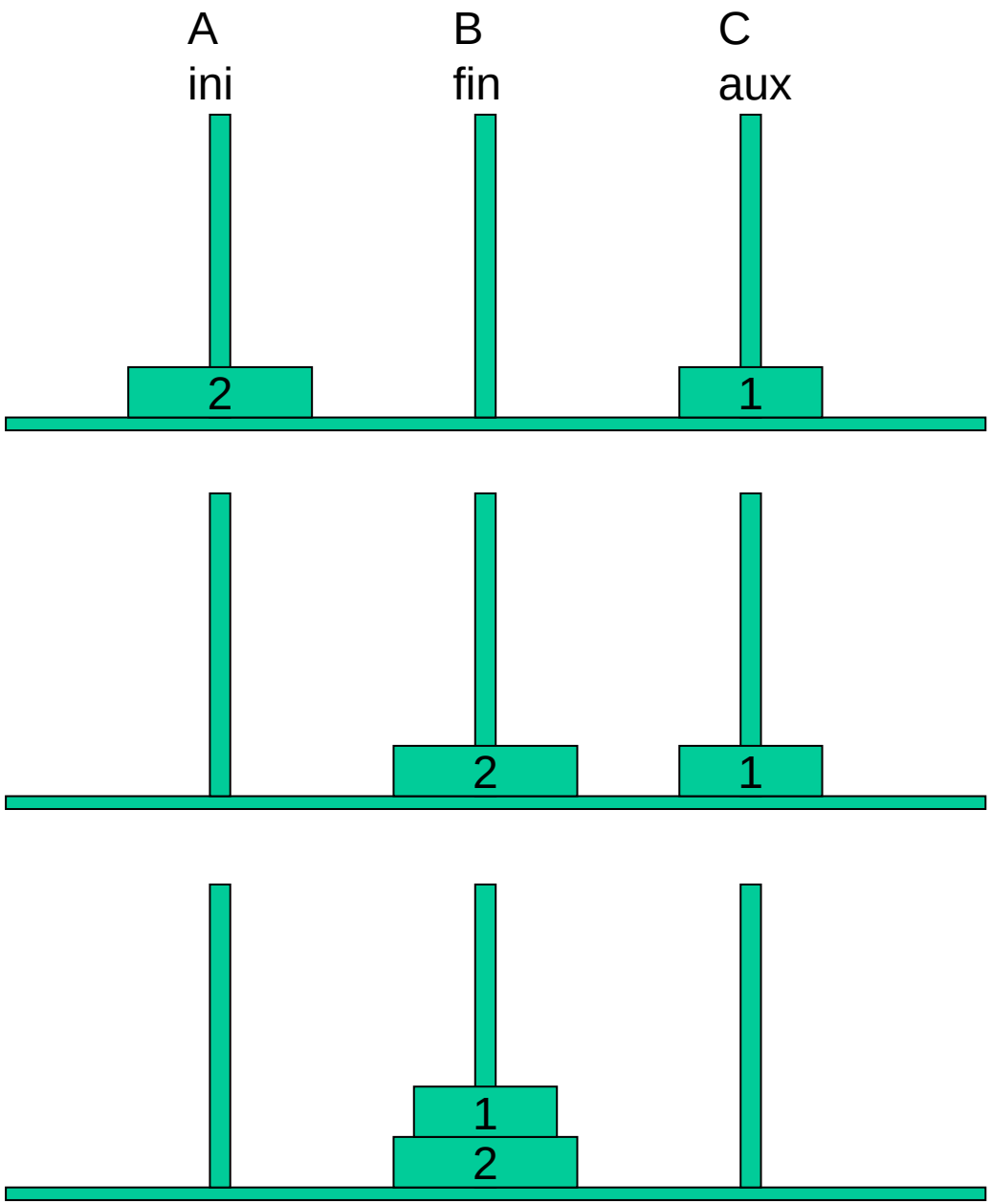
# IMPLEMENTACIÓN DE LA TABLA PARA FIBONACCI

```
type
  tDominio =0..20
  tRegistro= record
            definido: boolean;
             resultado: integer
          End;
  tTabla=array [tDominio] of tRegistro;
 var
 tablafib:tTabla; i:tDominio;
 Function Fib(n: tDominio; var
     t:tTabla):integer;
  Begin
      if not t[n].definido then Begin
          t[n].definido:=true;
          t[n].resultado:=Fib(n-1,t)+Fib(n-2,t);

       End {if};
    Fib:=t[n].resultado
 End; {Fibonacci}


  Begin
  tablafib[0].definido:=true;
      tablafib[0].resultado:=1;
  tablafib[1].definido:=true;
      tablafib[1].resultado:=1;
  for i:=2 to 20 do
      tablafib[i].definido:=false;
.......
```

# OTROS EJEMPLOS RECURSIVOS

- **El problema de las torres de Hanoi:**
  - Pasar discos de la torre A a la B, moviendo un único disco cada vez y apilando siempre un disco más pequeños sobre uno más grandes.






Pasar un disco de A a C

ini-aux

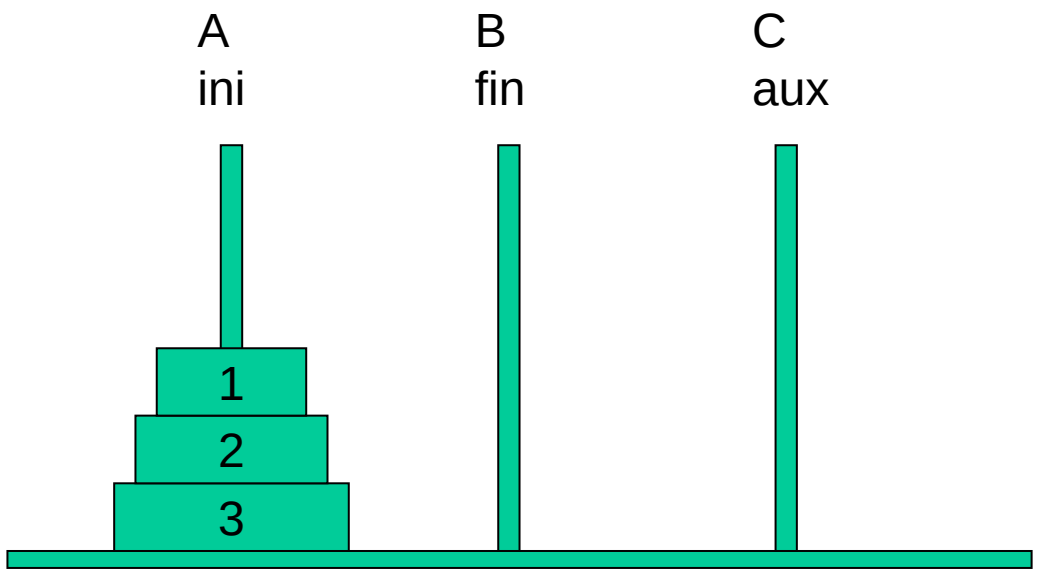
Pasar un disco de A a B

ini-fin

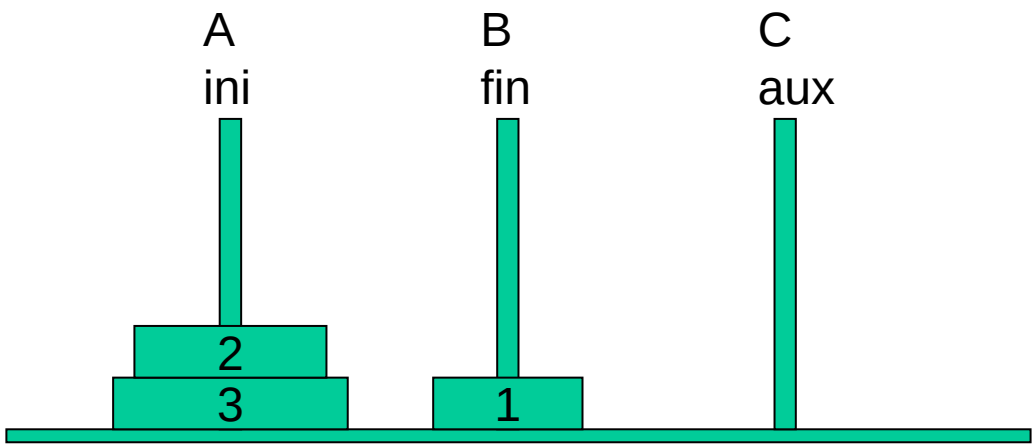
Pasar un disco de C a B

aux-fin

- **El problema de las torres de Hanoi con tres discos:**

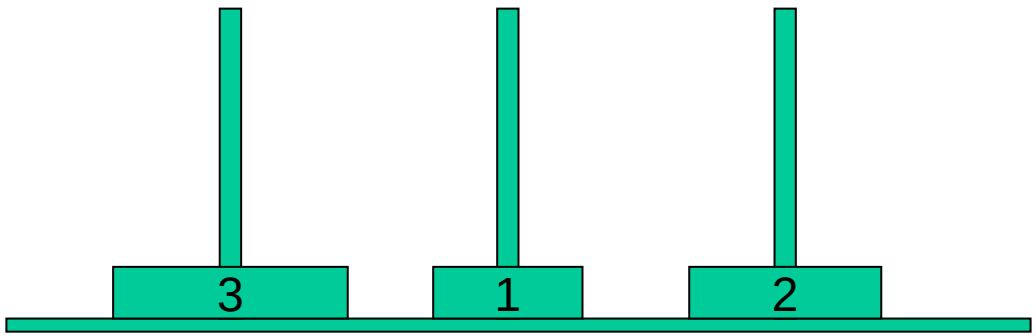



Argumentos del procedimiento:
(n, ini, fin, aux)

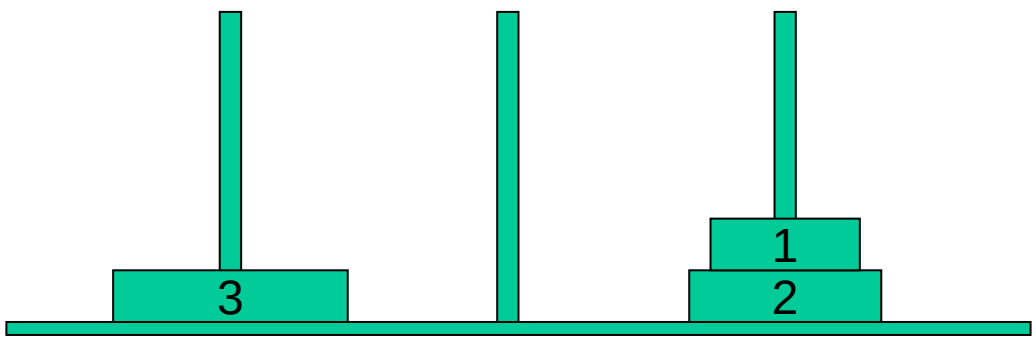



Pasar los dos discos superiores de A a C

Ini-fin



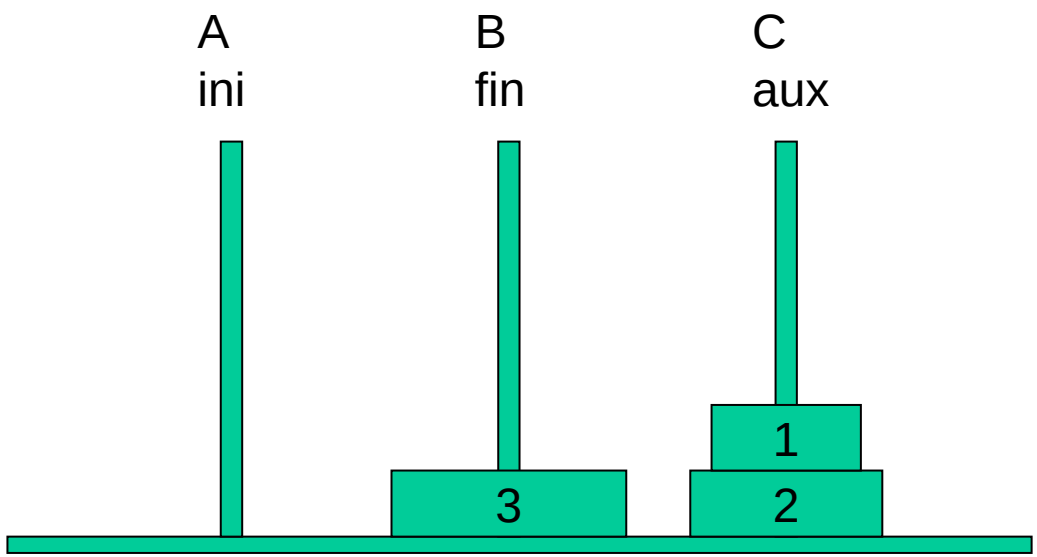

Ini-aux



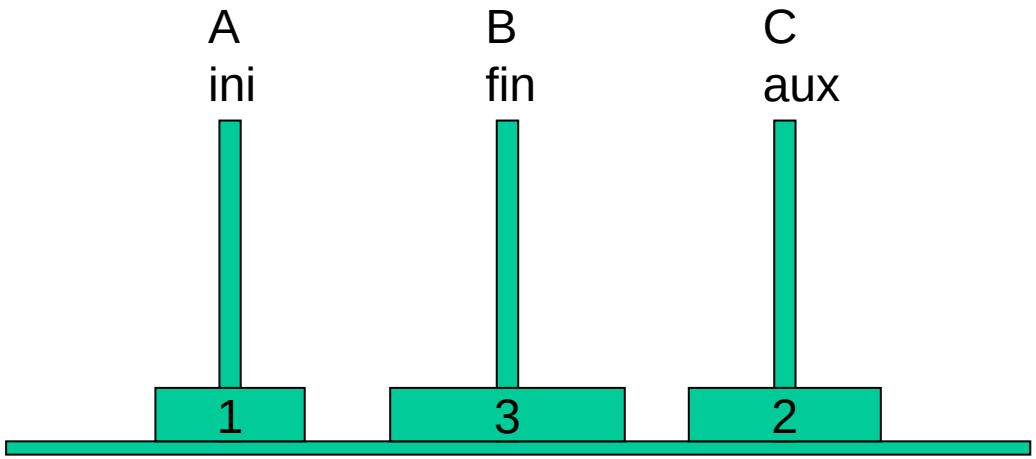

fin-aux

(n-1,ini, aux, fin)

# Pasar el tercer disco de A a B

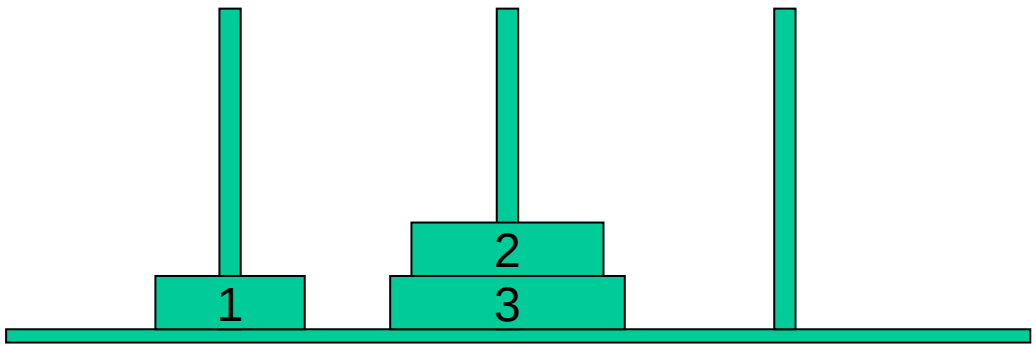



Ini-fin

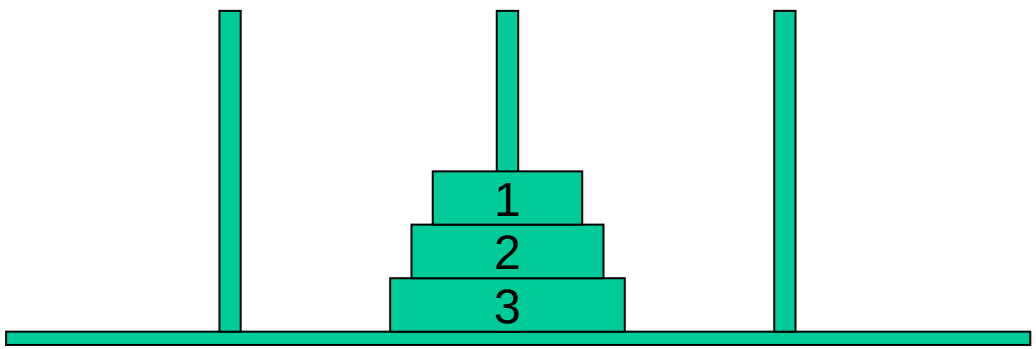



Pasar los dos discos superiores de C a B

aux-ini




aux-fin




ini-fin

(n-1,aux, fin, ini)

# PASAR TRES DISCOS DE A a B

1. Pasar dos discos de A a C:
    a. Mover 1 disco de A a B
    b. Mover 1 disco de A a C
    c. Mover 1 disco de B a C

2. Pasar un disco de a A a B

3. Pasar dos discos de C a B:
    a. Mover 1 disco de C a A
    b. Mover 1 disco de C a B
    c. Mover 1 disco de A a B

# LEY DE RECURRENCIA: PASAR n DISCOS DE A a B

1. Pasar n-1 discos de A a C

2. Pasar un disco de a A a B

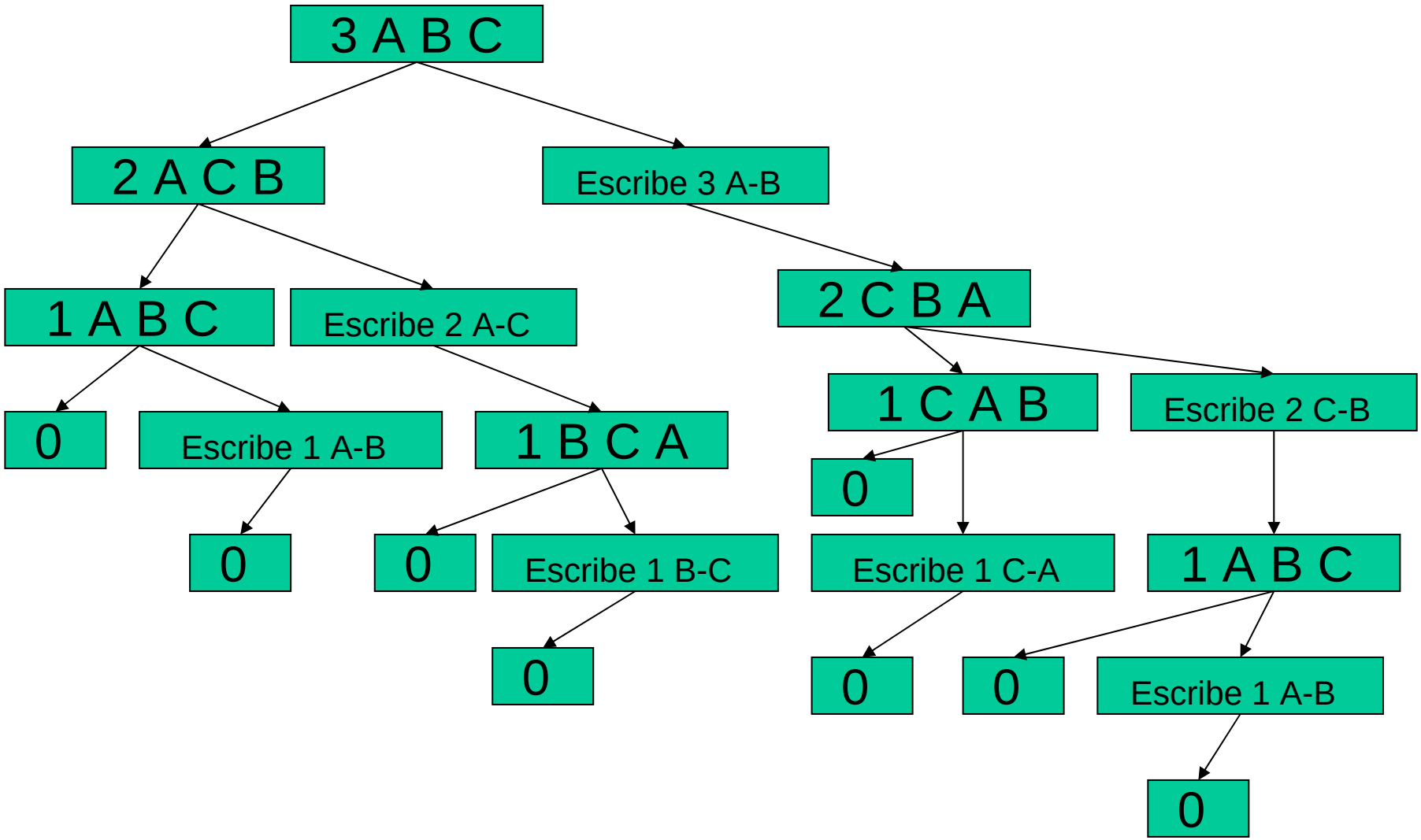
3. Pasar n-1 discos de C a B

Siendo el caso base cuando n=0

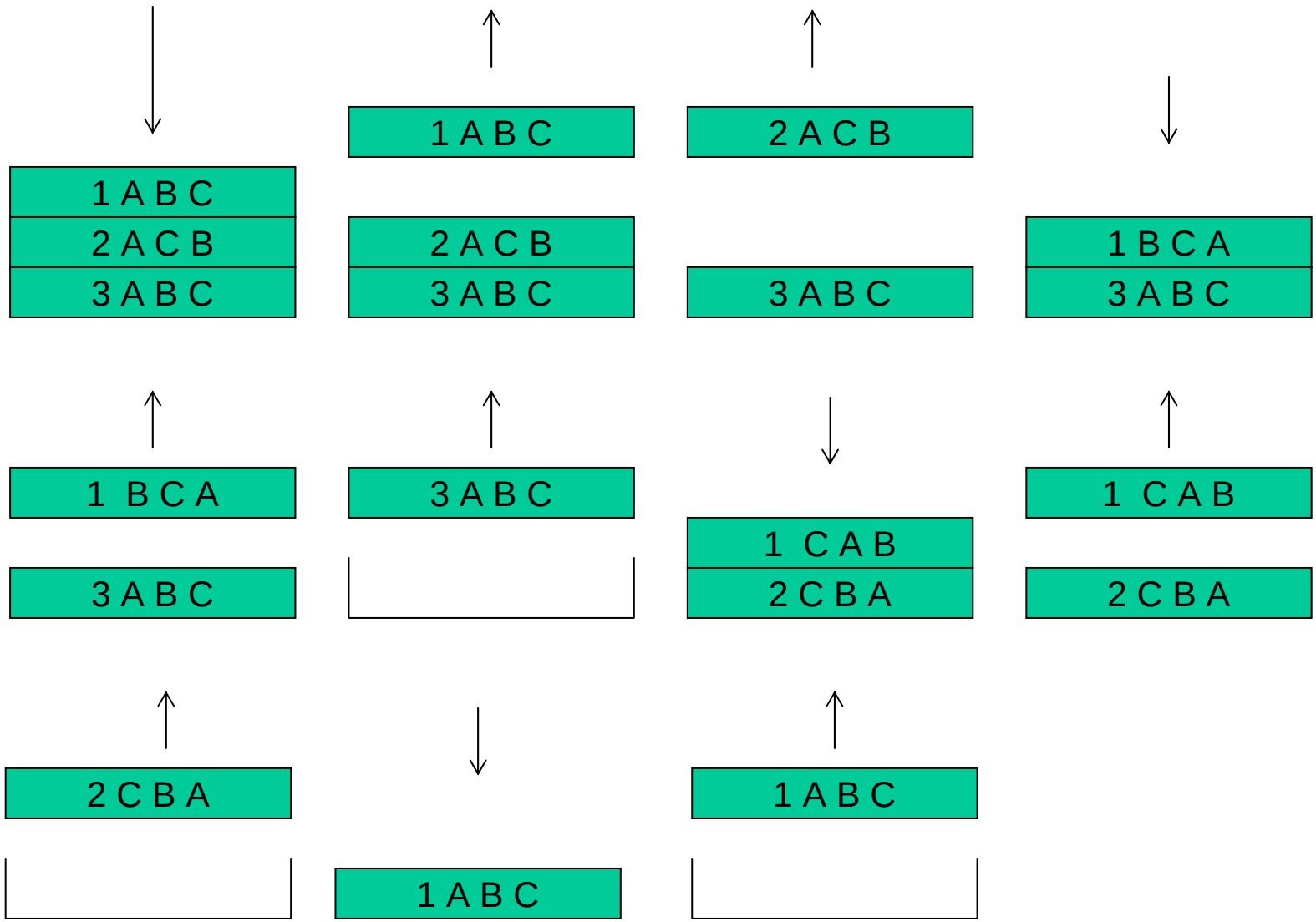
# IMPLEMENTACIÓN DEL PROBLEMA EN PASCAL

```
PROCEDURE Pasardiscos(n:integer; ini,fin,aux:char);
{Prec. n≥0}
{Efecto. Se pasan n discos de la torre inicial a la final}

Begin
    if n>0 then begin
        Pasardiscos(n-1,ini,aux,fin);
        writeln('mover el disco', n:3,'desde', ini,'a', fin);
        Pasardiscos(n-1,aux,fin,ini)
    end {if}
End; {Pasardiscos}
```

# TRAZA DEL ALGORITMO PARA n=3



# PILA RECURSIVA PARA LA TRAZA ANTERIOR



# RECURSIÓN MUTUA

- La **recursión mutua** se da cuando un programa llama a otro y éste a su vez es invocado por el primero.

- A esta recursión también se la denomina “cruzada”.

- Este problema para Pascal se resuelve mediante la palabra reservada “**forward**”. De esta forma el identificador del subprograma definido en segundo lugar esta predeclarado.

# RECURSIÓN MUTUA: DECLARACIÓN

```
PROCEDURE Segundo (parámetros); forward;
PROCEDURE Primero (parámetros);
.....
Begin {Primero}
.....
Segundo(parámetros)
.....
End; {Primero}
PROCEDURE Segundo (parámetros);
.....
Begin {Segundo}
.....
Primero(parámetros)
.....
End; {Segundo}
```

# RECURSIÓN MUTUA: EJEMPLO

- Programa que determina la paridad de un entero positivo empleando para ello dos funciones recurrentes:

  Function Espar(n:integer):boolean;

  Function Esimpar(n:integer):boolean;

  que son mutuamente recurrentes entre sí.

```
Program Paridad;
uses
    crt;
Var
    n:integer;
PROCEDURE Esimpar(n:integer); forward
PROCEDURE Espar(n:integer);
    Begin
        if n=0 then
            writeln('el número es par')
        else
            Esimpar(n-1)
    End;{Espar}

PROCEDURE Esimpar(n:integer);
    Begin
        if n=0 then
            writeln('el numero es
    impar')
        else
            Espar(n-1)
    End;{Esimpar}

Begin {Paridad}
    clrscr;
    writeln('Introduce un número entero
    positivo');
    readln(numero);
    espar(numero);
    readln {pausa}
End; {Paridad}
```

# RECURSIÓN vs ITERACIÓN

- Un programa que se invoca a sí mismo se repite un cierto número de veces.

- Por esta razón cualquier cálculo recursivo puede ser expresado como una iteración y viceversa.

- La elección entre uno y otro método vendrá dado por motivos de eficiencia y legibilidad.
  - Un problema recursivo posee una solución más legible si se resuelve mediante un algoritmo recursivo.
  - Una solución recursiva ocupa más memoria que una solución iterativa.