



# Taller de Programación



# AGENDA

Estructuras de Datos vistas: arreglos - listas

Concepto de Ordenación



# ARREGLOS - Características



Un **arreglo** es una estructura de datos compuesta que permite acceder a cada componente por una variable índice.

Dicho índice da la posición del componente dentro de la estructura de datos.

La estructura arreglo se almacena en posiciones contiguas de memoria

## CARACTERISTICAS

Homogénea

Estática

Acceso directo

Indexada

Lineal

Dimensión física

Dimensión lógica



# ARREGLOS - Características

Type

elem	elem	elem	elem		
------	------	------	------	--	--

`arreglo = array [rango] of tipo;`

Var

`v:arreglo;`

## OPERACIONES

- Cargar la estructura
- Agregar un elemento
- Insertar un elemento
- Eliminar un elemento
- Recorrer la estructura
- Buscar un elemento
- Ordenar la estructura**



# LISTAS - Características



Una **lista** es una estructura de datos lineal compuesta por nodos.

Cada nodo de la lista posee el dato que almacena la lista y la dirección del siguiente nodo.

Toda lista puede recorrerse a partir de su primer elemento.

Los elementos no necesariamente están en posiciones contiguas de memoria.

Para generar nuevos elementos en la lista, o eliminar alguno se deben utilizar las operaciones de new y dispose respectivamente.

## CARACTERISTICAS

Homogénea

Dinámica

Acceso secuencial

Lineal



# LISTAS - Características



Type

L

ABCD

ADDD

```
lista= ^nodo;
```

```
nodo = record
  dato:tipo;
  sig: lista;
end;
```

```
Var
  L:lista;
```

## OPERACIONES

- Crear una lista vacía
- Agregar un elemento adelante
- Agregar un elemento atrás
- Insertar un elemento
- Eliminar un elemento
- Recorrer la estructura
- Buscar un elemento
- Ordenar la estructura**



# ARREGLOS - Ordenación

**Cuál sería el beneficio de tener una estructura ordenada?**

23	1	100	4		
1	4	23	100		



Un **algoritmo de ordenación** es un proceso por el cual un conjunto de elementos puede ser ordenado.

Existe una gran variedad de algoritmos para ordenar vectores cada uno con características diferentes (facilidad de escritura, memoria utilizada, tiempo de ejecución)



# ARREGLOS - Ordenación

ALGORITMO	ORDEN de EJECUCION
Selección	$O(N^2)$
Intercambio	$O(N^2)$
Inserción	$O(N^2)$
Heapsort	$O(N(\log N))$
Mergesort	$O(N(\log N))$
Quicksort	$O(N(\log N))$

No solo debe considerarse el tiempo de ejecución de los algoritmos.

También influye:

- la facilidad para la escritura del mismo.
- la cantidad de memoria utilizada.
- la complejidad de las estructuras auxiliares que necesite
- que ocurre si los datos se encuentran ordenados, ordenados en orden inverso, o desordenados.

## Selección    Inserción





# Taller de Programación



# AGENDA



Método de ordenación: selección



# ARREGLOS – Ordenación - SELECCION

Dado un arreglo A y una dimensión lógica (dimL), el algoritmo consiste en buscar (hasta finalizar) en cada vuelta en que posición está ubicado el elemento mínimo y al finalizar la vuelta intercambiar el elemento mínimo con el primero que no ha sido ordenado.

Es decir, en la **primera vuelta** se busca en que posición está ubicado el mínimo, esa búsqueda se realiza desde la primera posición hasta el final del arreglo (dimension lógica), luego se intercambia el elemento de la primera posición con el elemento ubicado en la posición que se encontró el mínimo.

En la **segunda vuelta**, se busca cual es la posición donde está ubicado el elemento mínimo, esa búsqueda se realiza a partir de la segunda posición (ya que el elemento de la primera posición ya fue ubicado como mínimo) y hasta el final del arreglo (dimension lógica), luego se intercambia el elemento de la segunda posición con el elemento ubicado en la posición que se encontró el mínimo.

El algoritmo repite (dimension lógica - 1) veces.

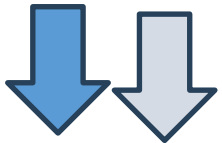


# ARREGLOS – Ordenación - SELECCION

23	1	100	4	7	
----	---	-----	---	---	--

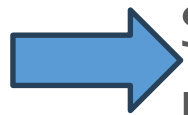
dimF = 6  
dimL = 5

## VUELTA 1



23	1	100	4	7	
----	---	-----	---	---	--

mínimo = 1  
pos = 2



Se intercambia el elemento de la posición 1 (23) con el de la posición 2 (1).

1	23	100	4	7	
---	----	-----	---	---	--

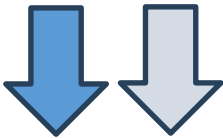


# ARREGLOS – Ordenación - SELECCION

1	23	100	4	7	
---	----	-----	---	---	--

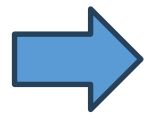
dimF = 6  
dimL = 5

VUELTA 2



1	23	100	4	7	
---	----	-----	---	---	--

mínimo = 4  
pos = 4



Se intercambia el elemento de la posición 2 (23) con el de la posición 4 (4).

1	4	100	23	7	
---	---	-----	----	---	--

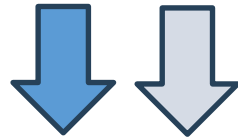


# ARREGLOS – Ordenación - SELECCION

1	4	100	23	7	
---	---	-----	----	---	--

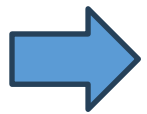
dimF = 6  
dimL = 5

VUELTA 3



1	4	100	23	7	
---	---	-----	----	---	--

mínimo = 7  
pos = 5



Se intercambia el elemento de la posición 3(100) con el de la posición 5 (7).

1	4	7	23	100	
---	---	---	----	-----	--

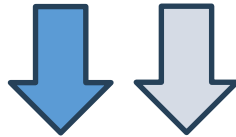


# ARREGLOS – Ordenación - SELECCION

1	4	7	23	100	
---	---	---	----	-----	--

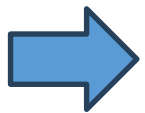
dimF = 6  
dimL = 5

VUELTA 4



1	4	7	23	100	
---	---	---	----	-----	--

mínimo = 23  
pos = 4



Se intercambia el elemento de la posición 4(23) con el de la posición 4 (23).

1	4	7	23	100	
---	---	---	----	-----	--



# ARREGLOS – Ordenación - SELECCION

**Program** ordenar;

**Const** dimF =... *{máxima longitud del arreglo}*

**Type**

TipoElem = ... *{ tipo de datos del vector }*

Indice = 0.. dimF;

Tvector = **array** [ 1..dimF] **of** TipoElem;

**Var**

a:Tvector;

dimL:integer;

**Begin**

cargarVector (a, dimL);

seleccion (a, dimL);

**End.**





# ARREGLOS – Ordenación - SELECCION

```
Procedure seleccion ( var v: tVector; dimLog: indice );  
  
var i, j, pos: indice; item : tipoElem;  
  
begin  
  for i:=1 to dimLog-1 do begin {busca el mínimo y guarda en pos la posición}  
    pos := i;  
    for j := i+1 to dimLog do  
      if v[ j ] < v[ pos ] then pos:=j;  
  
      {intercambia v[i] y v[p]}  
      item := v[ pos ];  
      v[ pos ] := v[ i ];  
      v[ i ] := item;  
    end;  
  end;  
end;
```



# ARREGLOS – Ordenación - SELECCION

## QUE NECESITAMOS CONOCER?

- Dimensión lógica del arreglo.
- Posición donde va el elemento ordenado.
- Posición desde la que vamos a buscar el mínimo.
- Posición del elemento mínimo.

## CARACTERISTICAS

- Fácil de implementar.
- El tiempo de ejecución es de orden  $N^2$ .
- Posición del elemento mínimo.



# Taller de Programación



# AGENDA



## Método de ordenación: inserción



# ARREGLOS – Ordenación - INSERCIÓN

Dado un arreglo A y una dimensión lógica (dimL), el algoritmo consiste en ordenar en cada vuelta un elemento a un subconjunto de elementos ya ordenados.

Es decir, en la **primera vuelta** se toma el subconjunto que contiene el primer elemento del arreglo y obviamente se considera ordenado.

En la **segunda vuelta**, se toma el segundo elemento del arreglo y se lo inserta de manera que quede ordenado con respecto al subconjunto que ya está ordenado (el primer elemento).

En la **tercera vuelta**, se toma el tercer elemento del arreglo y se lo inserta de manera que quede ordenado con respecto al subconjunto que ya está ordenado (el primer y segundo elemento).

En la **k-ésima vuelta**, se toma el  $k+1$  elemento del arreglo y se lo inserta de manera que quede ordenado con respecto al subconjunto de  $k$  elementos que ya está ordenados.

En cada vuelta puede ser necesario realizar corrimientos para insertar de manera ordenada cada nuevo elemento.

Dado que en la primer vuelta se considera solo el primer elemento y por lo tanto no se debe realizar ningún ordenamiento, en realidad la primera vuelta del algoritmo comienza tomando los dos primeros elementos.

El algoritmo repite (dimension lógica - 1) veces.



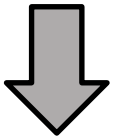
# ARREGLOS – Ordenación - INSERCIÓN

5	3	2	1	4	6	
---	---	---	---	---	---	--

**VUELTA 1**



Tomo el elemento ubicado en la posición 2 (3) y se compara desde la posición 1 hasta la 1 para ver en qué posición debe insertarse.



5	3	2	1	4	6	
---	---	---	---	---	---	--

Debe insertarse  
en la posición  
1



Se encuentra que el valor debe insertarse en la posición 1, por lo tanto, se realiza un corrimiento desde la posición 1 hasta la 2 para “hacer” lugar en el vector.

5	3	2	1	4	6	
---	---	---	---	---	---	--

3



# ARREGLOS – Ordenación - INSERCIÓN

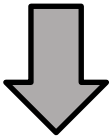
3	5	2	1	4	6	
---	---	---	---	---	---	--

$\text{dimF} = 7$   
 $\text{dimL} = 6$

**VUELTA 2**

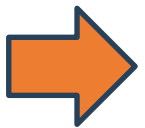


Tomo el elemento ubicado en la posición 3 (2) y se compara desde la posición 1 hasta la 2 para ver en qué posición debe insertarse.



3	5	2	1	4	6	
---	---	---	---	---	---	--

Debe insertarse  
en la posición  
1



Se encuentra que el valor debe insertarse en la posición 1, por lo tanto, se realiza un corrimiento desde la posición 1 hasta la 3 para “hacer” lugar en el vector.

3	5	2	1	4	6	
---	---	---	---	---	---	--

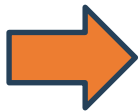


# ARREGLOS – Ordenación - INSERCIÓN

2	3	5	1	4	6	
---	---	---	---	---	---	--

$\text{dimF} = 7$   
 $\text{dimL} = 6$

**VUELTA 3**



Tomo el elemento ubicado en la posición 4 (1) y se compara desde la posición 1 hasta la 3 para ver en qué posición debe insertarse.



2	3	5	1	4	6	
---	---	---	---	---	---	--

Debe insertarse  
en la posición  
1



Se encuentra que el valor debe insertarse en la posición 1, por lo tanto, se realiza un corrimiento desde la posición 1 hasta la 3 para “hacer” lugar en el vector.

2	3	5	1	4	6	
---	---	---	---	---	---	--





# ARREGLOS – Ordenación - INSERCIÓN

1	2	3	5	4	6	
---	---	---	---	---	---	--

`dimF = 7`  
`dimL = 6`

**VUELTA 4** 

Tomo el elemento ubicado en la posición 5 (4) y se compara desde la posición 1 hasta la 4 para ver en qué posición debe insertarse.



1	2	3	5	4	6	
---	---	---	---	---	---	--

Debe insertarse  
en la posición  
4

 Se encuentra que el valor debe insertarse en la posición 4, por lo tanto, se realiza un corrimiento desde la posición 4 hasta la 5 para “hacer” lugar en el vector.

1	2	3	5	4	6	
---	---	---	---	---	---	--



# ARREGLOS – Ordenación - INSERCIÓN

1	2	3	4	5	6	
---	---	---	---	---	---	--

`dimF = 7`  
`dimL = 6`

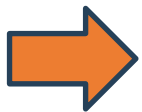
**VUELTA 5** 

Tomo el elemento ubicado en la posición 6 (6) y se compara desde la posición 1 hasta la 5 para ver en qué posición debe insertarse.



1	2	3	4	5	6	
---	---	---	---	---	---	--

Debe insertarse  
en la posición  
6



Se encuentra que el valor debe insertarse en la posición 6, por lo tanto, se realiza un corrimiento desde la posición 6 hasta la 6 para “hacer” lugar en el vector.

1	2	3	4	5	6	
---	---	---	---	---	---	--



# ARREGLOS – Ordenación - INSERCION

**Program** ordenar;

**Const** dimF =... *{máxima longitud del arreglo}*

**Type**

TipoElem = ... *{ tipo de datos del vector }*

Indice = 0.. dimF;

Tvector = **array** [ 1..dimF] **of** TipoElem;

**Var**

a:Tvector;

dimL:integer;

**Begin**

cargarVector (a, dimL);

insercion (a, dimL);

**End.**



# ARREGLOS – Ordenación - INSERCIÓN

```
Procedure insercion ( var v: tVector; dimLog: indice );  
var i, j: indice; actual: tipoElem;
```

```
begin
```

```
  for i:= 2 to dimLog do begin
```

```
    actual:= v[i];
```

```
    j:= i-1;
```

```
    while (j > 0) y (v[j] > actual) do
```

```
      begin
```

```
        v[j+1]:= v[j];
```

```
        j:= j - 1;
```

```
      end;
```

```
    v[j+1]:= actual;
```

```
  end;
```

```
end;
```



# ARREGLOS – Ordenación - insercion

## QUE NECESITAMOS CONOCER?

Dimensión lógica del arreglo.

Posición que se debe  
comparar.

Cuántos elementos ya están  
ordenados.

## CARACTERISTICAS

No tan fácil de implementar.

El tiempo de ejecución es de orden  $N^2$

Si los datos están ordenados de menor a mayor el algoritmo solo hace comparaciones, por lo tanto, es de orden (n).

Si los datos están ordenados de mayor a menor el algoritmo hace todas las comparaciones y todos los intercambios, por lo tanto es de orden ( $N^2$ ).

comparaciones.

¿Qué ocurre con las listas ?



# Taller de Programación



# AGENDA



## Recursión

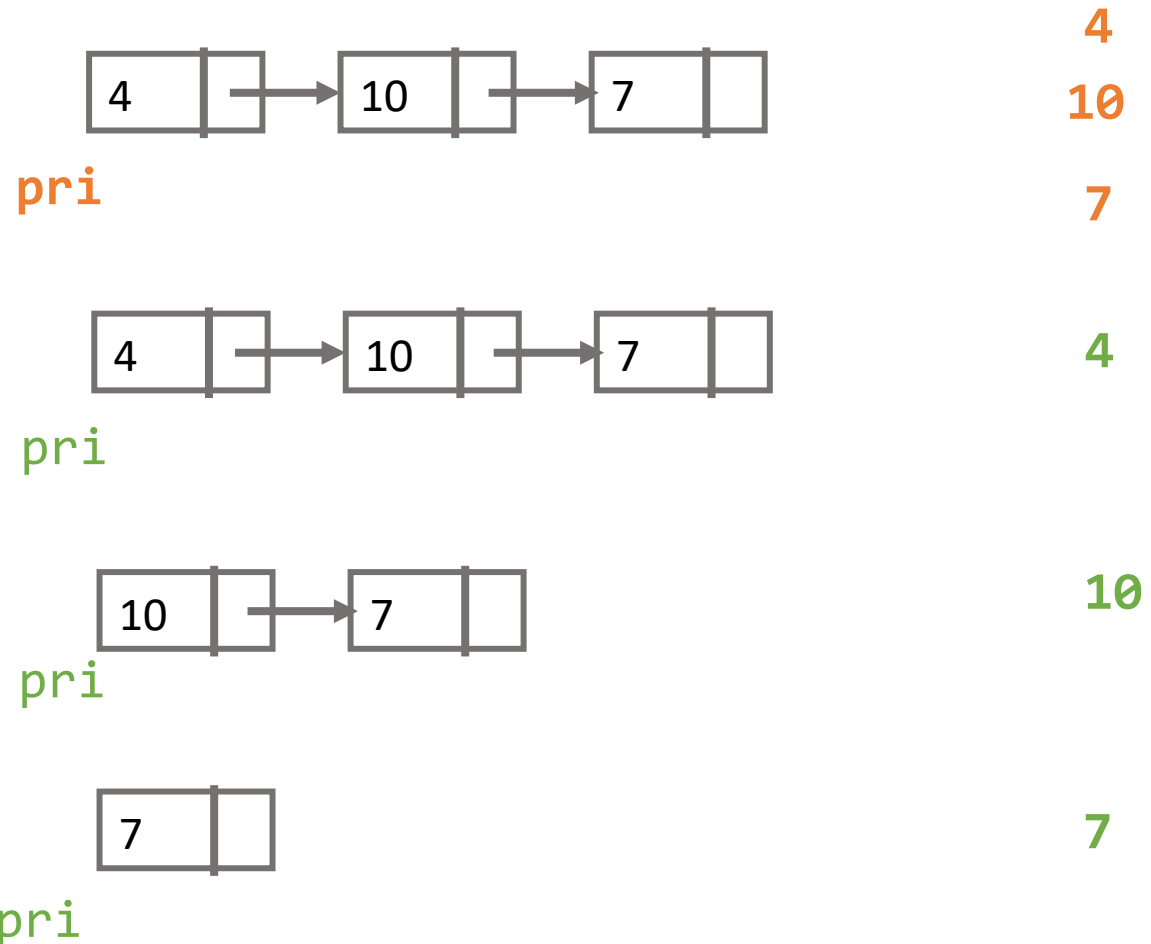


# Recursión - MOTIVACION

Suponga que debe realizar un módulo que imprima una los elementos de una lista de enteros.

```
Procedure imprimir (pri:lista);
Begin
  while (pri <> nil) do
    begin
      write(pri^.dato);
      pri:= pri^.sig;
    end;
  End;
```

Se presenta el mismo problema cada vez más chico hasta llegar a una instancia que no se debe resolver nada







# Recursión - MOTIVACION

Suponga que debe realizar un módulo que retorne el factorial de un número entero recibido.  $n = n * (n-1) * \dots * 1$  veces

$$5 = 5 * 4 * 3 * 2 * 1 = 120$$

```
Procedure factorial (num:integer; var fac:integer);  
Var  
  i:integer;
```

```
Begin  
  fac:= 1;  
  for i:= num downto 1 do  
    begin  
      fac:= fac * i;  
    end;  
End;
```

Se presenta el mismo problema cada vez más chico hasta llegar a una instancia que se resuelve de manera directa

$$\text{factorial (5)} = 5 * \text{factorial(4)}$$

$$\text{factorial (4)} = 4 * \text{factorial(3)}$$

$$\text{factorial (3)} = 3 * \text{factorial(2)}$$

$$\text{factorial (2)} = 2 * \text{factorial(1)}$$

$$\text{factorial (1)} = 1$$

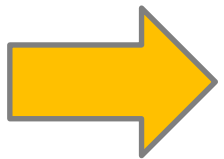


# RECURSIÓN - DEFINICION

Existen un conjunto de problemas que pueden resolverse siempre de la misma manera con la característica que el problema debe ir “achicandose” en cada instancia a resolver, hasta que en alguna instancia la solución es “trivial”.



La **recursividad** es una técnica de resolución de problemas que consiste en dividir un problema en instancias más pequeñas del mismo problema (también llamados subproblemas) hasta que obtengamos un subproblema lo suficientemente pequeño que tenga una solución trivial o directa.



La recursividad consiste en resolver un problema por medio de un módulo (procedimientos o funciones) que se llama a sí mismo, evitando el uso de bucles y otros iteradores.

Cuando el problema se va achicando llega a un punto que no puede achicarse más, esa instancia se denomina **caso base**.

Hay problemas en los cuales debe realizarse alguna tarea cuando se alcanza el caso base y otros que no.

Hay problemas que pueden tener más de un caso base.



# RECURSIÓN - EJEMPLOS

Suponga que debe realizar un módulo que imprima los elementos de una lista de enteros que recibe como parámetro.

## SOLUCIÓN ITERATIVA

```
Procedure imprimir (pri:lista);
Begin
  while (pri <> nil) do
    begin
      write (pri^.dato);
      pri:= pri^.sig;
    end;
  End;
```

Cómo achico  
el problema?

Hasta  
cuando  
achico el  
problema?

Qué hago  
cuando llego al  
caso base?

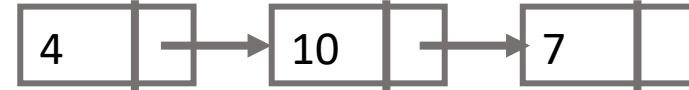
## SOLUCIÓN RECURSIVA

```
Procedure imprimir (pri:lista);
Begin
  if (pri <> nil) do
    begin
      write (pri^.dato);
      pri:= pri^.sig;
      imprimir (pri);
    end;
  End;
```

Cómo  
funciona?



# RECURSIÓN – EJEMPLOS – Cómo funciona?



```
Procedure imprimir (pri:lista);  
Begin  
  if (pri <> nil) then  
    begin  
      write (pri^.dato);  
      pri:= pri^.sig;  
      imprimir (pri);  
    end;  
  End;
```

Cuál es la  
diferencia con la  
solución  
secuencial?

Procedimiento imprimir	pri= 4	4	3
Procedimiento imprimir	pri= 10	10	3
Procedimiento imprimir	pri= 7	7	3
Procedimiento imprimir	pri= nil	En este caso no se hace nada	
Variables del programa Programa principal			



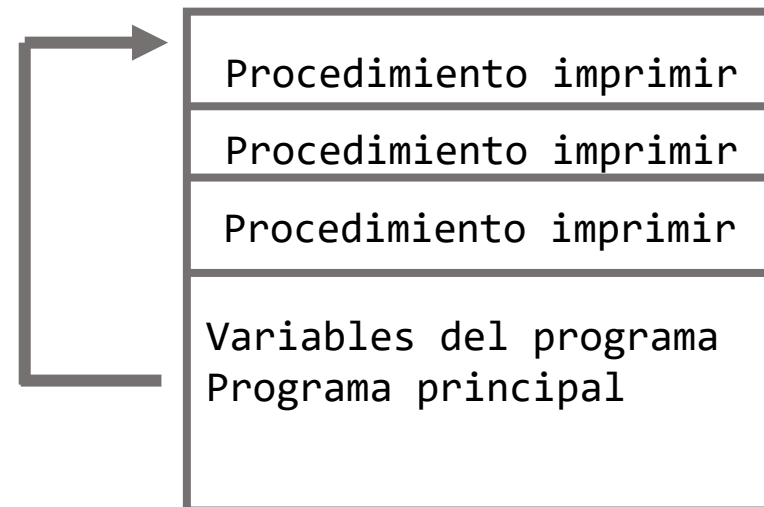
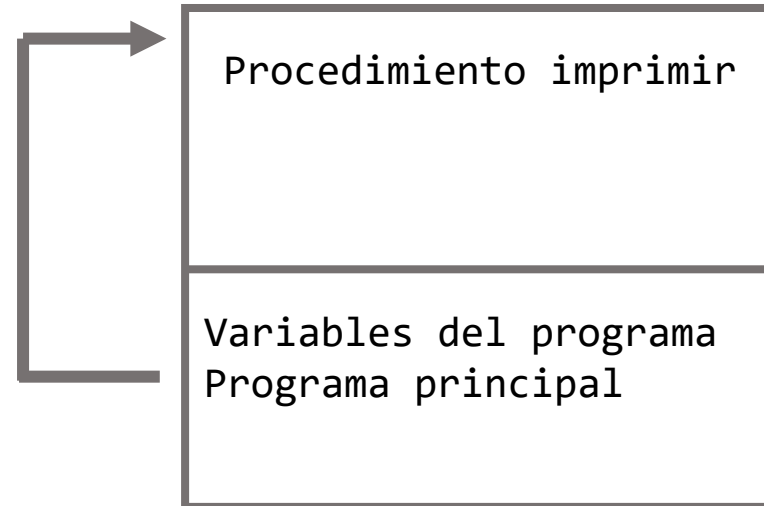
# RECURSIÓN – EJEMPLOS – Cómo funcionan?

## SOLUCIÓN ITERATIVA

```
Procedure imprimir (pri:lista);  
Begin  
  while (pri <> nil) do  
    begin  
      write (pri^.dato);  
      pri:= pri^.sig;  
    end;  
  End;
```

## SOLUCIÓN RECURSIVA

```
Procedure imprimir (pri:lista);  
Begin  
  IF (pri <> nil) then  
    begin  
      write (pri^.dato);  
      pri:= pri^.sig;  
      imprimir (pri);  
    end;  
  End;
```



Cuál cree que es más eficiente en cuanto al uso de la memoria?

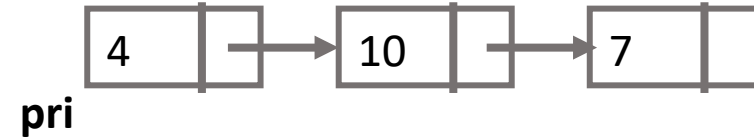
Qué pasa con los parámetros?




# RECURSIÓN – EJEMPLOS – Cómo funciona?

## SOLUCIÓN RECURSIVA

```
Procedure imprimir (pri:lista);  
Begin  
  if (pri <> nil) then  
    begin  
      write (pri^.dato);  
      pri:= pri^.sig;  
      imprimir (pri);  
    end;  
End;
```



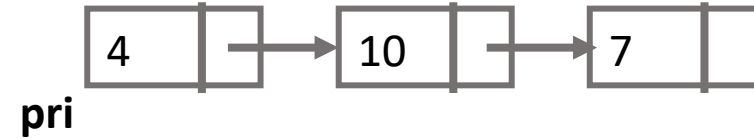
	Procedimiento imprimir	<b>pri= 4</b>	4	3
	Procedimiento imprimir	<b>pri= 10</b>	10	3
	Procedimiento imprimir	<b>pri= 7</b>	7	3
	Procedimiento imprimir	<b>pri= nil</b>	En este caso no se hace nada	
	Variables del programa Programa principal <b>pri=4</b>			



# RECURSIÓN – EJEMPLOS – Cómo funciona?

## SOLUCIÓN RECURSIVA

```
Procedure imprimir (VAR pri:lista);
Begin
  IF (pri <> nil) then
    begin
      write (pri^.dato);
      pri:= pri^.sig;
      imprimir (pri);
    end;
End;
```



Procedimiento imprimir	<b>pri= 4</b>	4	3
Procedimiento imprimir	<b>pri= 10</b>	10	3
Procedimiento imprimir	<b>pri= 7</b>	7	3
Procedimiento imprimir	<b>pri= nil</b>	En este caso no se hace nada	
Variables del programa Programa principal <b>pri = nil</b>			



# RECURSIÓN - EJEMPLOS

Suponga que debe realizar un módulo que calcular la potencia de un número  $x$  a la  $n$ , que es  $= x^n = x * x * x$  ( $n$  veces).

## SOLUCIÓN ITERATIVA

```
Procedure potencia (x,n:integer;  
                  var pot:integer);  
  
Var  
  i:integer;  
Begin  
  if (n = 0) then pot:= 1  
  else if (n = 1) then pot:= x  
  else begin  
    pot:= 1;  
    for i:= 1 to n do  
      pot:= pot * x;  
    end;  
  End;
```

Con una  
función?

Cómo lo  
pienso  
recursivo?

## SOLUCIÓN ITERATIVA

```
Function potencia (x,n:integer):integer;  
Var  
  i,pot:integer;  
Begin  
  if (n = 0) then pot:= 1  
  else if (n = 1) then pot:= x  
  else begin  
    pot:= 1;  
    for i:= 1 to n do  
      pot:= pot * x;  
    end;  
  potencia:=pot;  
End;
```





# RECURSIÓN - EJEMPLOS

Suponga que debe realizar un módulo que calcular la potencia de un número  $x$  a la  $n$ , que es  $= x^n = x * x * x$  ( $n$  veces).

## SOLUCIÓN RECURSIVA

```
Procedure potencia (x,n:integer;  
                  var pot:integer);
```

```
Var
```

```
  i:integer;
```

```
Begin
```

```
  if (n = 0) then pot:= 1
```

```
  else if (n = 1) then pot:= x
```

```
  else
```

```
    begin
```

```
      potencia (x, (n-1), pot);
```

```
      pot:= pot * n;
```

```
    end;
```

```
End;
```

Con una  
función?

Cuántos caso  
base hay?

## SOLUCIÓN RECURSIVA

```
Function potencia (x,n:integer):integer;
```

```
Begin
```

```
  if (n = 0) then potencia:= 1
```

```
  else if (n = 1) then potencia:= x
```

```
  else
```

```
    potencia:= x * potencia(x, n-1));
```

```
  end;
```

```
End;
```

Cómo  
funciona?



# RECURSIÓN - Características

Supongamos  $x = 4$   $n=3$



potencia  $x= 4, n=3$

$4 * \text{potencia}(4, 2)$

~~164~~

potencia  $x= 4, n=2$

$4 * \text{potencia}(4, 1)$

potencia  $x= 4, n=1$

4

## SOLUCIÓN RECURSIVA

```
Function potencia (x,n:integer);
```

```
Begin
```

```
  if (n = 0) then potencia:= 1
```

```
  else if (n = 1) then potencia:= x
```

```
  else
```

```
    potencia:= x * potencia(x, n-1));
```

```
  end;
```

```
End;
```

Alguna vez entrará  
por el caso  $(n=0)$ ?



# Taller de Programación



# AGENDA



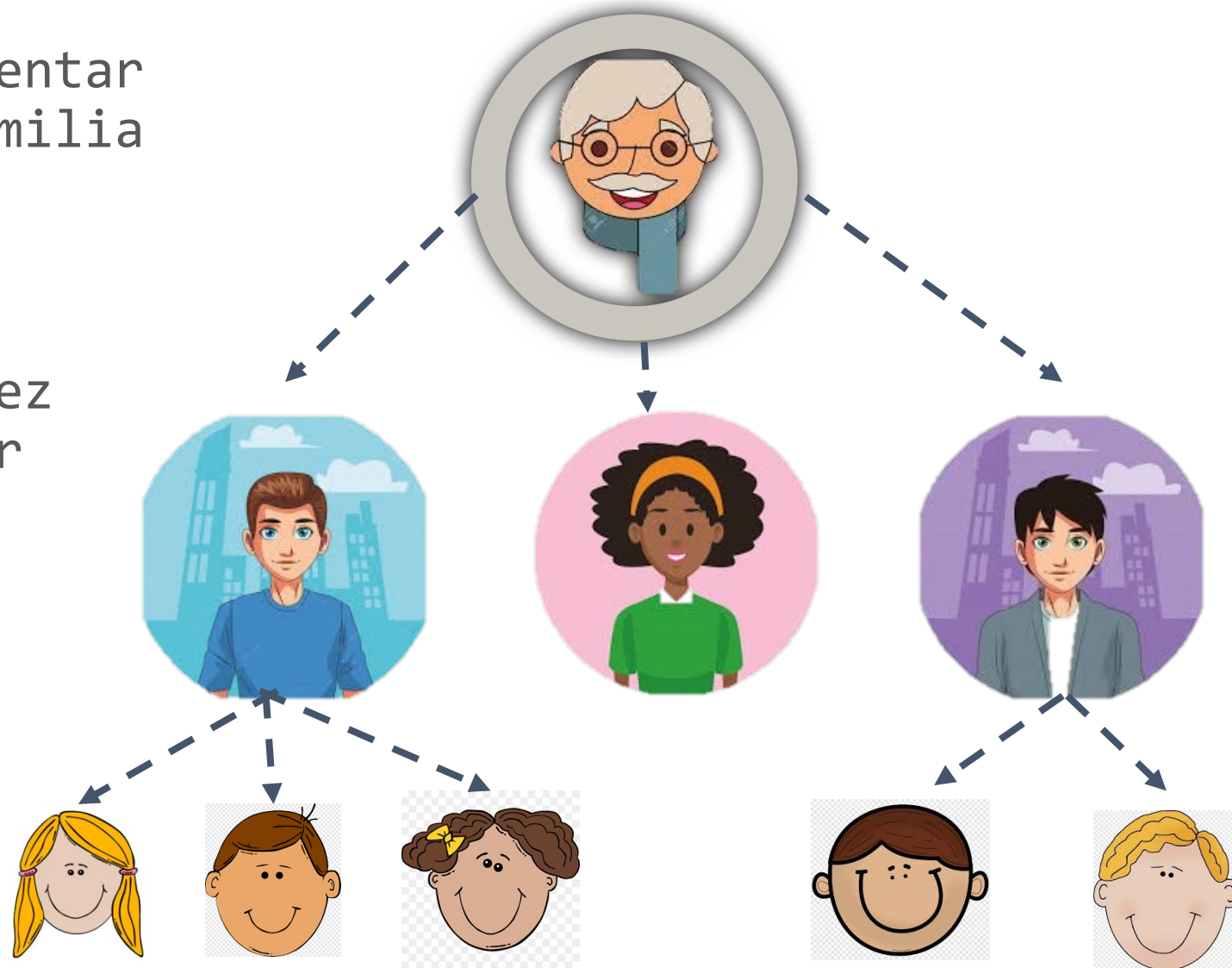
## Esturctura de datos arbol



# ESTRUCTURA DE DATOS ARBOL

Supongamos que queremos representar el árbol genealógico de una familia a partir de un integrante (por ejemplo un abuelo).

El abuelo tiene hijos y a su vez esos hijos también pueden tener hijos (nietos del abuelo)



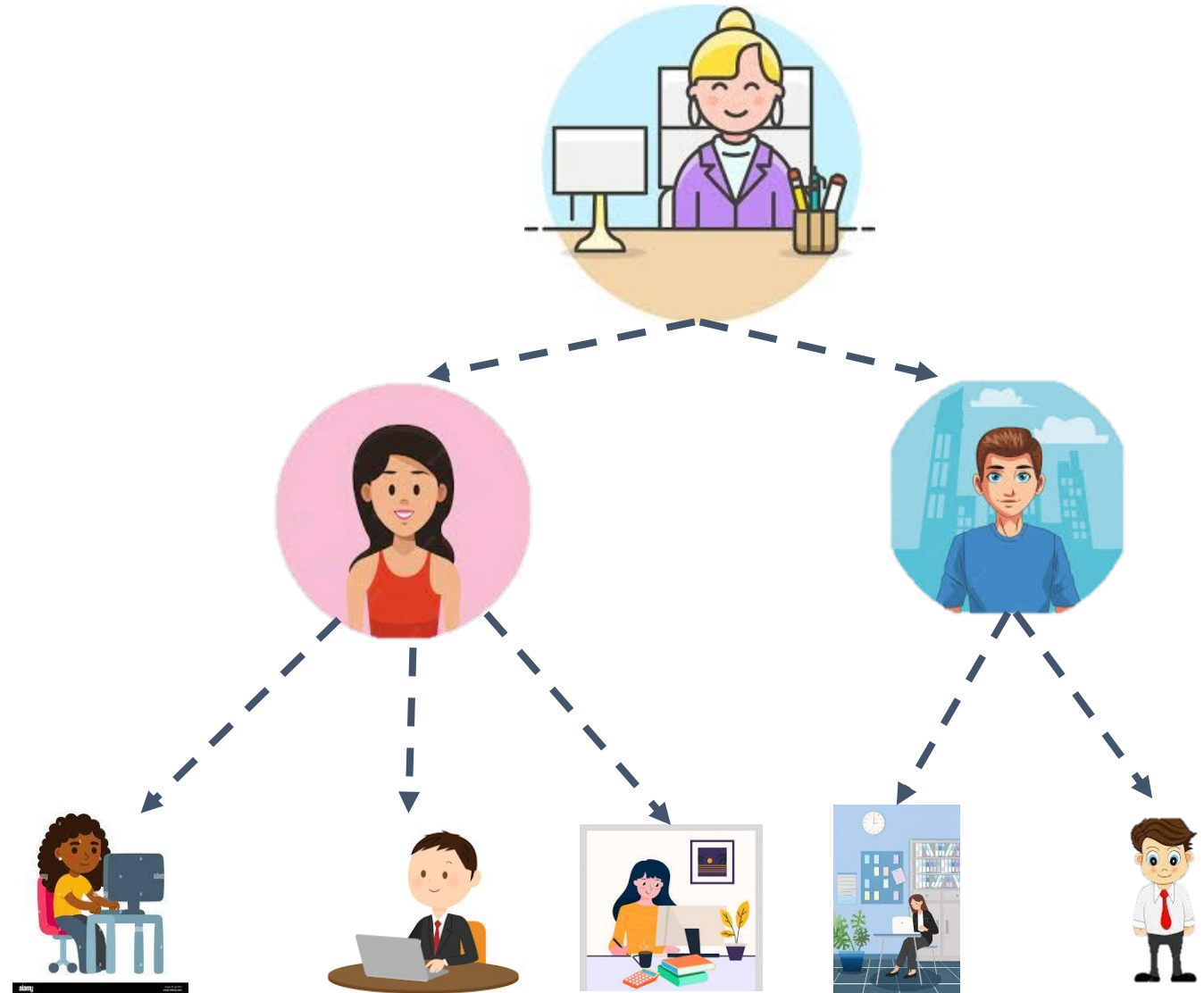


# ESTRUCTURA DE DATOS ARBOL

Supongamos que queremos representar la organización de una empresa.

La empresa tiene un area generencial (con una jefa), la cual está compuesta por varias areas de trabajo (area de personal, area contable, etc con diferentes personas a cargo).

Cada una de estas areas a su vez también podría estar compuesta por alguna/s subarea (con varias personas a cargo).





# ESTRUCTURA DE DATOS ARBOL

Por todo lo mencionado es importante notar que existen un montón de problemas que necesitan expresarse de una manera jerárquica característica que no permiten las estructuras vistas hasta el momento (arreglos y listas).

Es una estructura de datos jerárquica.

Está formada por nodos, donde cada nodo tiene a lo sumo hijos. E

El nodo principal del árbol se denomina raíz y los nodos que no tienen hijos se denominan hojas del árbol.



ARBOL

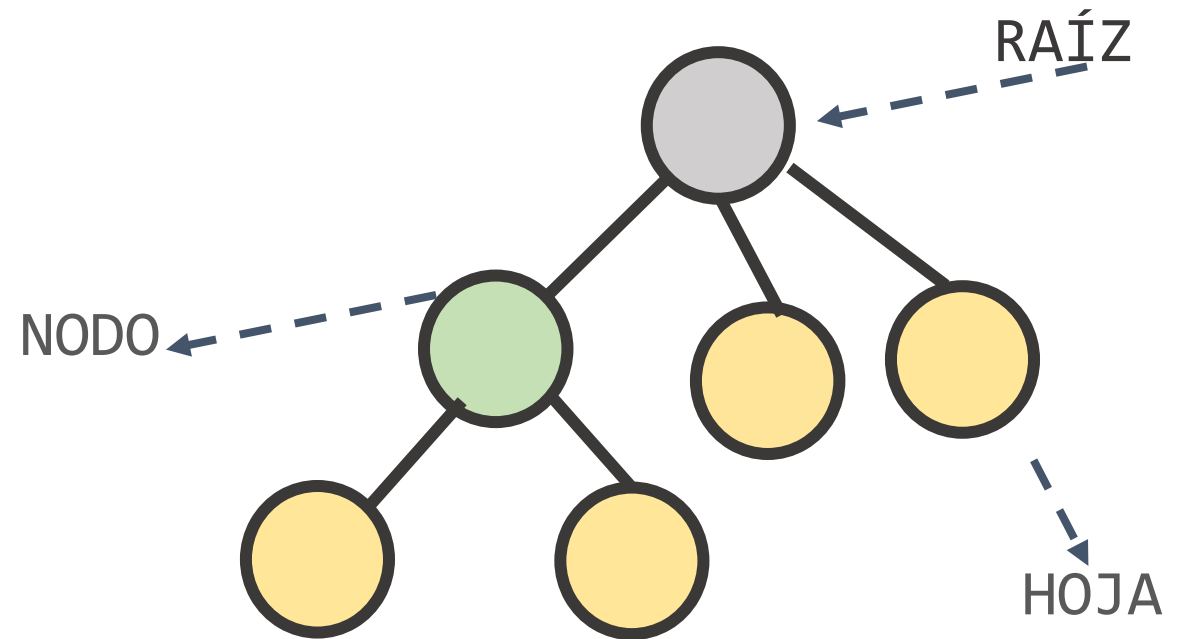
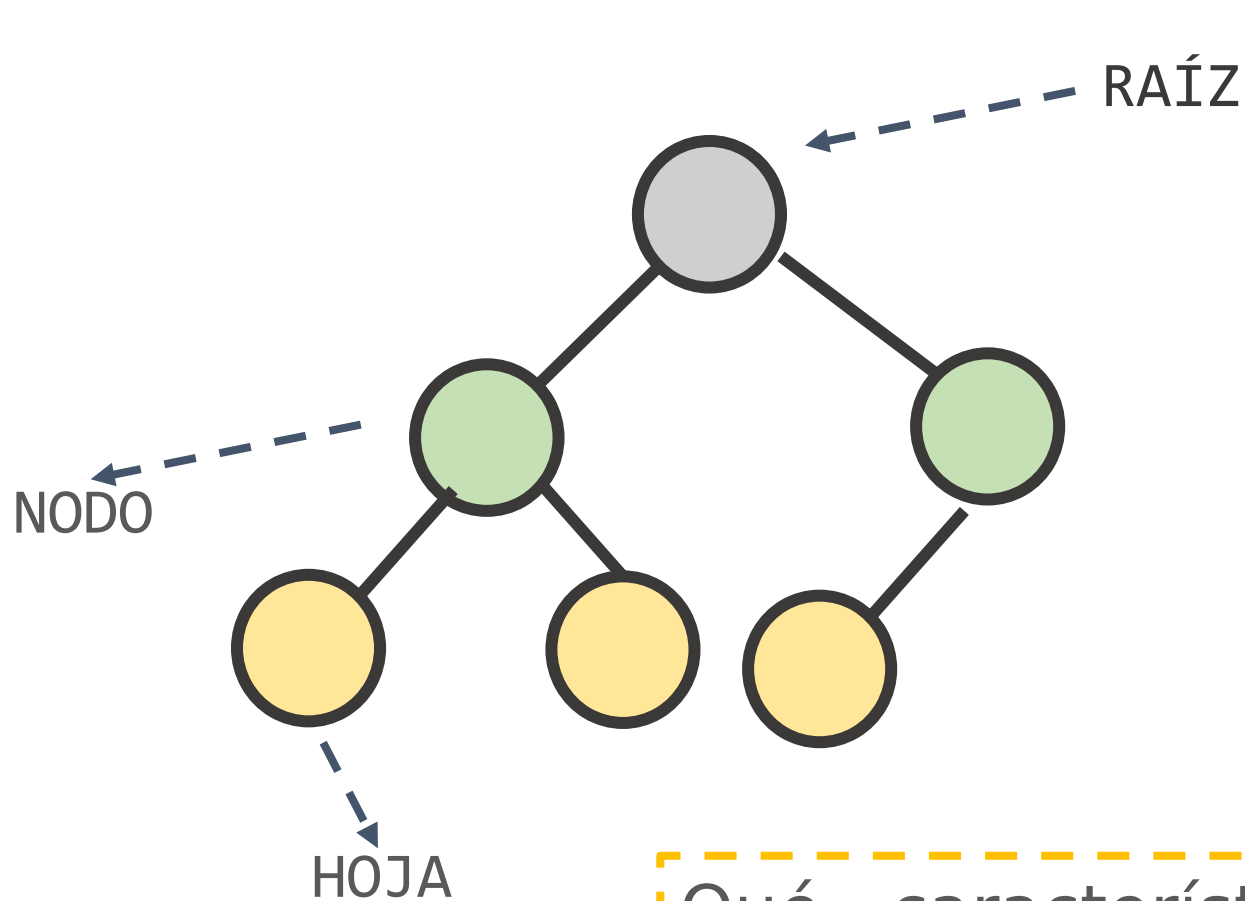
Homogénea

Dinámica

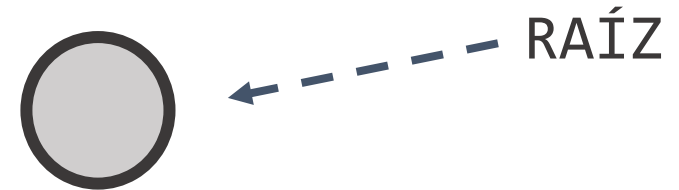
NO lineal



# ESTRUCTURA DE DATOS ARBOL



Qué característica cumple cualquier tipo de árbol ?







# ÁRBOLES - Características

Todo árbol es una estructura jerárquica

Todo árbol es una estructura dinámica

Todo árbol es una estructura homogénea

Para crear un árbol siempre se empieza por la raíz

Un árbol vacío se representa con el valor nil

Un nuevo dato siempre se inserta como una hoja

Con qué tipo de árbol vamos a trabajar ? | Cómo se declara?



# ÁRBOLES BINARIOS DE BÚSQUEDA- Declaración

```
Programa arboles;
```

```
Type
```

```
    arbol = ^nodo;
```

```
    tipo = ...;
```

```
    nodo = record
```

```
        dato: tipo;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol;
```

```
Begin
```

```
    ...
```

```
End.
```

```
Programa arbolesEnteros;
```

```
Type
```

```
    arbol = ^nodo;
```

```
    nodo = record
```

```
        dato: integer;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol;
```

```
Begin
```

```
    ...
```

```
End.
```

Qué  
característica  
tiene un ABB?

```
Programa arbolesPersonas;
```

```
Type
```

```
    persona = record
```

```
        nombre:string;
```

```
        dni:integer;
```

```
    end;
```

```
    arbol = ^nodo;
```

```
    nodo = record
```

```
        dato: persona;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol;
```

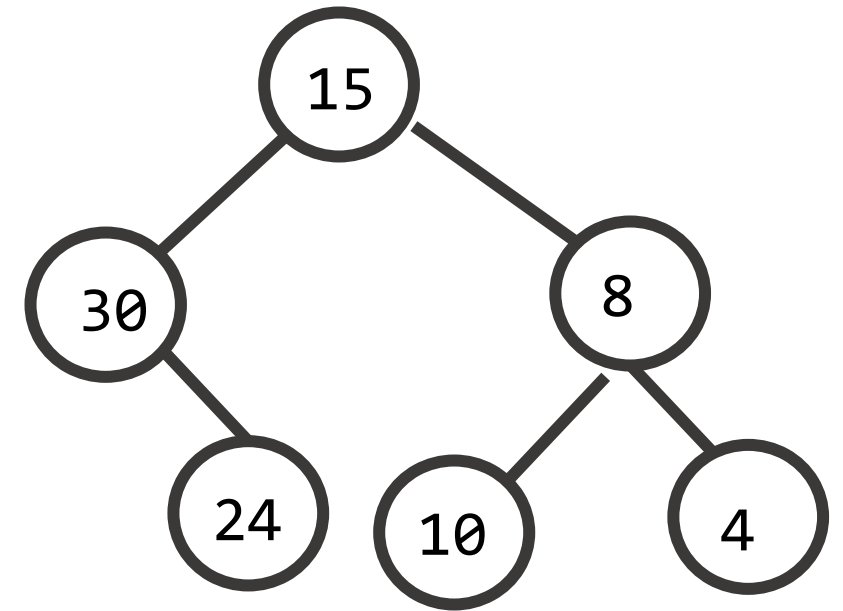
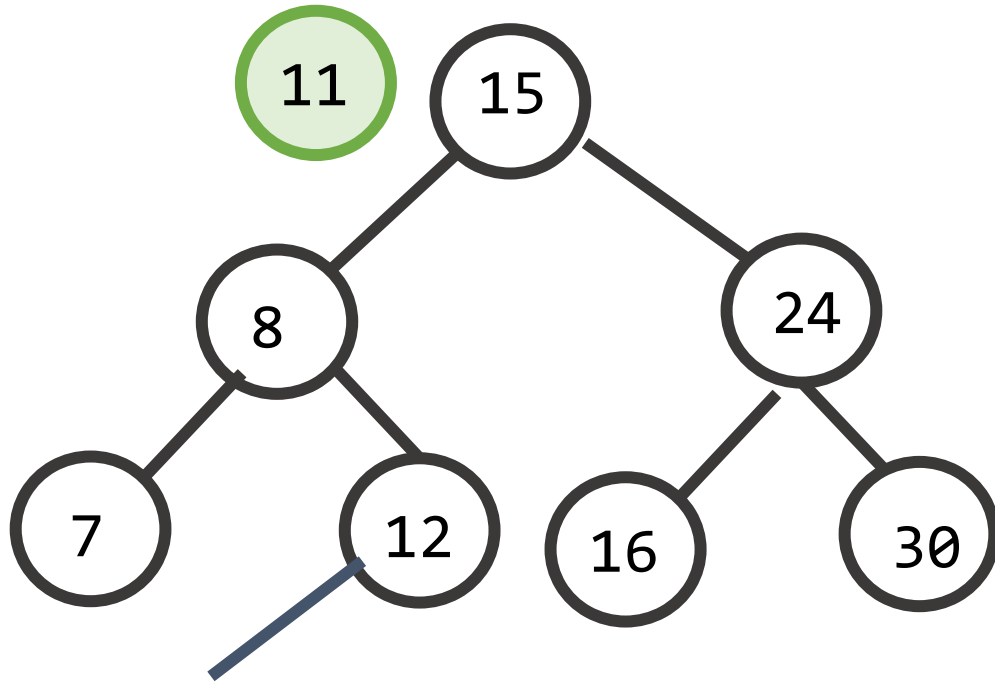
```
Begin
```

```
    ...
```

```
End.
```



# ÁRBOLES BINARIOS DE BÚSQUEDA- Característica



Un **árbol binario de búsqueda (ABB)** agrega los elementos por sus hojas. Dichos elementos quedan ordenados (todos por el mismo criterio). Esta operación lleva un tiempo de ejecución de  $O(\log n)$ .



# Taller de Programación



# AGENDA



Esturctura de datos arbol

Operaciones - CREACION

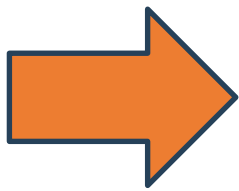


# ESTRUCTURA DE DATOS ARBOL

Es una estructura de datos jerárquica (no lineal), homogénea y dinámica.

Está formada por nodos, donde cada nodo tiene a lo sumo hijos.

El nodo principal del árbol se denomina raíz y los nodos que no tienen hijos se denominan hojas del árbol.



ABB

Los nodos del arbol respetan todos el mismo criterio (los hijos ubicados a la izquierda son menores al nodo padre o al revés)

Cómo creamos  
un ABB?



# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

Programa arboles;

Type

```
arbol = ^nodo;  
nodo = record  
    dato: integer;  
    HI: arbol;  
    HD: arbol;  
end;
```

Var

```
a:arbol;  
num:integer;
```

Begin

...

End.

Begin

```
a:= nil; //indico que el árbol está vacío
```

```
read (num); //leo un valor  
while (num <> 50) do
```

```
begin
```

```
    agregar (a,num); //agrego el valor al arbol
```

```
    read (num);
```

```
end;
```

End..

Suponga que se leen los siguientes valores y se quieren ir agregando en a (9, 18, 22, 19, 7,50). Cómo quedarán guardados?



# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

Suponga que se leen los siguientes valores y se quieren ir agregando en un ABB  
(9, 18, 22, 19, 7, 50)

Programa arboles;

Type

```
arbol = ^nodo;  
nodo = record  
    dato: integer;  
    HI: arbol;  
    HD: arbol;  
end;
```

Var

```
a:arbol; num:integer;
```

Begin

```
a:= nil;  
read (num);  
while (num <> 50) do  
    begin  
        agregar (a,num);  
        read (num);  
    end;
```

End.

Clase 3-2 – Módulo Imperativo

Cómo quedarán  
guardados los  
valores en el ABB?

**a = nil**

Se **lee el valor 9 (num)** y se invoca  
al procedimiento agregar (a,num)

Como a = nil, el primer valor leído será la  
raíz del arbol.

Para agregarlo al ser una estructura dinámica  
debe reservarse memoria.

Luego se asigna en el campo dato el valor de  
num, y como por ahora este nodo no tiene  
hijos, en los campos HI e HD debe asignarse  
nil

El procedimiento agregar termina y vuelve al  
programa principal en donde **a** ahora apunta a  
un nodo con valor 9 y sus hijos en nil.







# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

Suponga que se leen los siguientes valores y se quieren ir agregando en un ABB (9, 18, 22, 19, 7, 50)

Programa arboles;

Type

```
arbol = ^nodo;  
nodo = record  
    dato: integer;  
    HI: arbol;  
    HD: arbol;  
end;
```

Var

```
a:arbol; num:integer;
```

Begin

```
a:= nil;  
read (num);  
while (num <> 50) do  
    begin  
        agregar (a,num);  
        read (num);  
    end;
```

End.

Clase 3-2 – Módulo Imperativo

a = 9



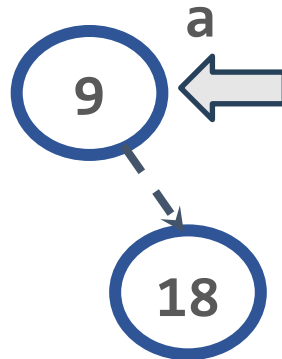
<sup>a</sup> Se lee el valor 18 (num) y se invoca al procedimiento agregar (a,num)

Como el árbol NO es vacío, tengo que recorrer desde la raíz hasta el lugar correspondiente respetando el orden. Siempre se inserta en una hoja.

Comparo num (18) con lo que está apuntado a (9), como  $18 > 9$  se determina que hay que agregarlo a la derecha de 9.

Como HD de 9 es =nil, ya se encontró el lugar (hoja), reservo memoria dinámica y asigno los valores correspondientes

El procedimiento agregar termina y vuelve al programa principal en donde a ahora apunta a un nodo con valor 9 y su HI= nil y HD = 18.

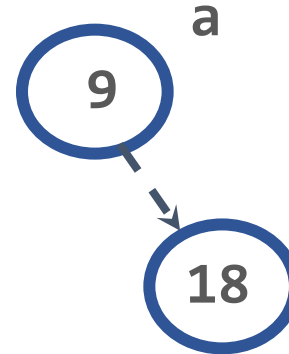




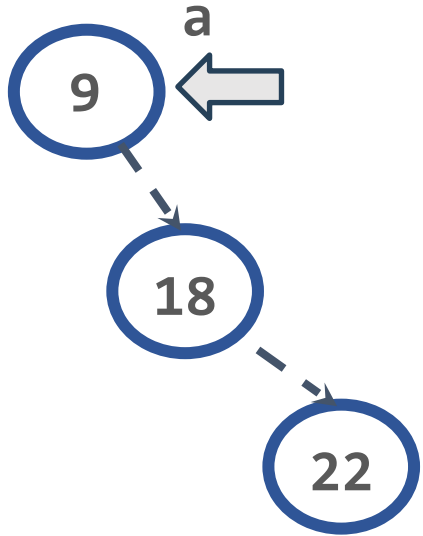
# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

Suponga que se leen los siguientes valores y se quieren ir agregando en un ABB (9, 18, 22, 19, 7, 50)

$a = 9$



Se lee el valor 22 (num) y se invoca al procedimiento agregar (a,num)



Como el árbol  $a \neq \text{nil}$ , tengo que recorrer desde la raíz hasta el lugar correspondiente respetando el orden. Siempre se inserta en una hoja.

Comparo num (22) con lo que está apuntado a (9),  $22 > 9$  se determina que hay que agregarlo a la derecha de 9. Como HD de 9  $\neq \text{nil}$ , sigo recorriendo hacia la derecha. Luego se compara y  $22 > 18$  y como HD 18 = nil se encontró el lugar donde agregar el 22.

Reservo memoria dinámica y asigno los valores correspondientes

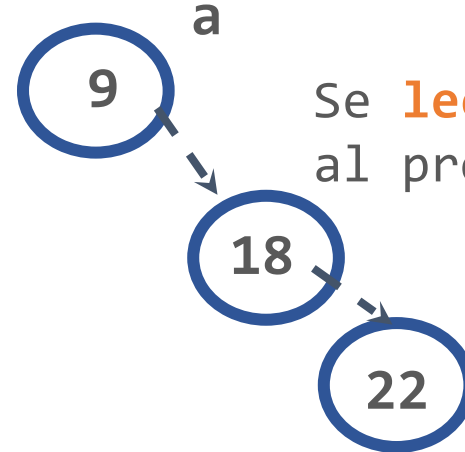
El procedimiento agregar termina y vuelve al programa principal en donde a ahora apunta **a** un nodo con valor 9 y su HI= nil y HD = 18 y 18 con su **HD= 22**.



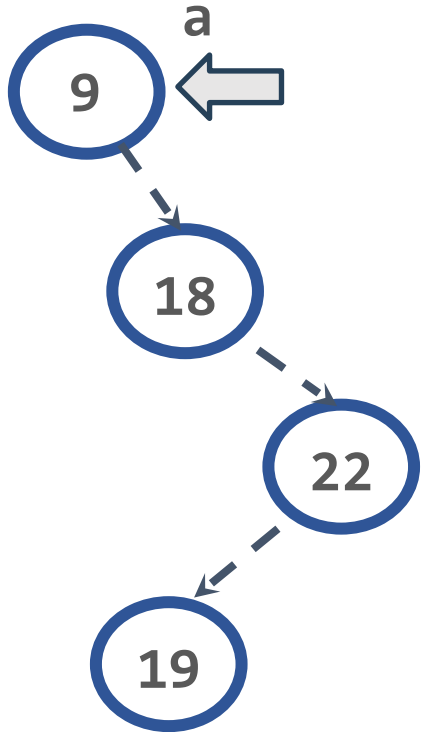
# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

Suponga que se leen los siguientes valores y se quieren ir agregando en un ABB (9, 18, 22, 19, 7, 50)

$a = 9$



Se lee el valor 19 (num) y se invoca al procedimiento agregar (a,num)



Como  $a \neq \text{nil}$ , tengo que recorrer desde la raíz hasta el lugar correspondiente respetando el orden. Siempre se inserta en una hoja.

Comparo num (19) con lo que está apunta a (9) y como  $18 > 9$ , hay que agregarlo a la derecha 9. Luego  $18 \leq 19$  entonces hay que agregarlo a la derecha. Luego  $19 \leq 22$ , hay que agregarlo a la izquierda de 22. Como es nil, se encontró el lugar.

Reservo memoria dinámica y asigno los valores correspondientes

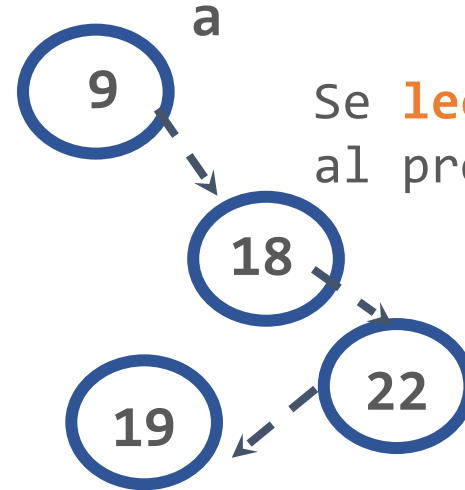
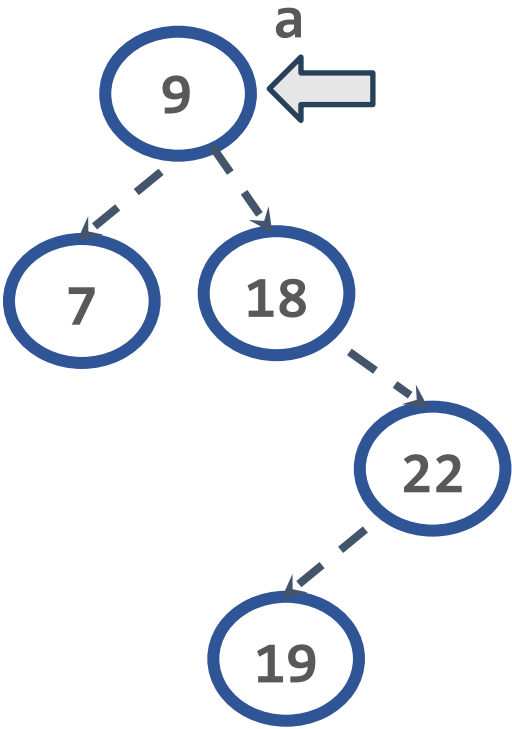
El procedimiento agregar termina y vuelve al programa principal en donde  $a = 9$ , su HD =18, a su vez su HD=22 y el HI de 22 = 19.



# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

Suponga que se leen los siguientes valores y se quieren ir agregando en un ABB (9, 18, 22, 19, 7, 50)

$a = 9$



Se lee el valor 7 (num) y se invoca al procedimiento agregar (a,num)

Como  $a \neq \text{nil}$ , tengo que recorrer desde la raíz hasta el lugar correspondiente respetando el orden. Siempre se inserta en una hoja.

Comparo num (7) con lo que está apunta a (9) y como es  $7 \leq 9$  se determina que hay que agregarlo a la izquierda de 9, que como 9 no tiene HI se encontró el lugar.

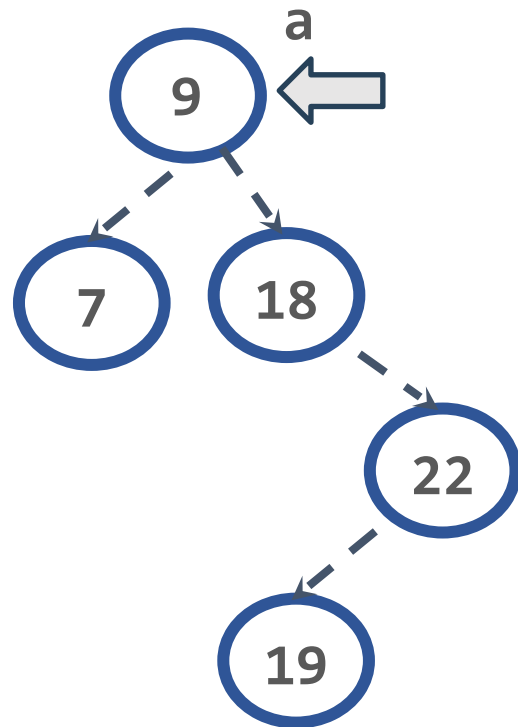
Reservo memoria dinámica y asigno los valores correspondientes

El procedimiento agregar termina y vuelve al programa principal en donde  $a = 9$ , su HD =18, y su **HI=7**.

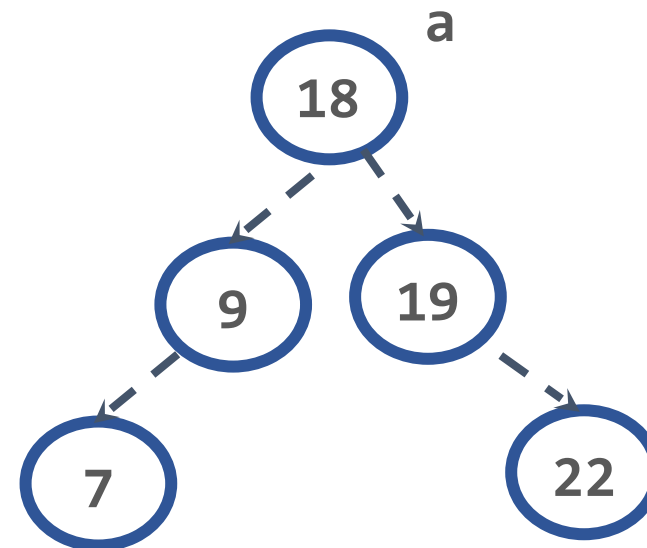


# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

Suponga que se leen los siguientes valores y se quieren ir agregando en un ABB (9, 18, 22, 19, 7, 50)



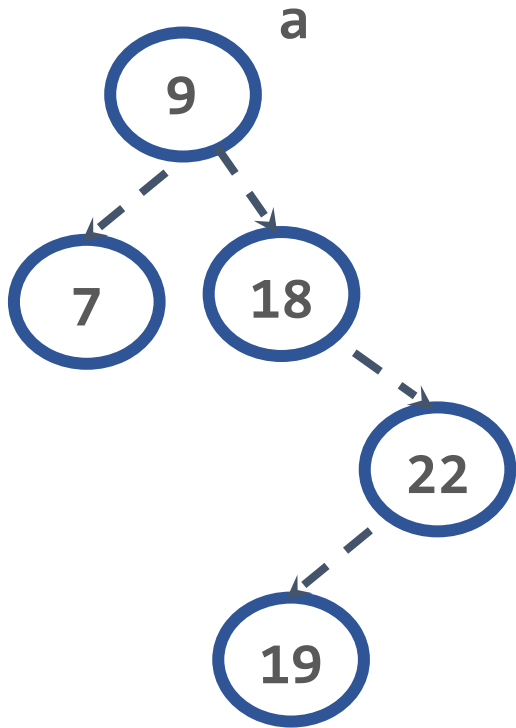
Si se leyeran los mismos valores pero en otro orden quedaría formado el mismo arbol? (18, 9, 7, 19, 22, 50)



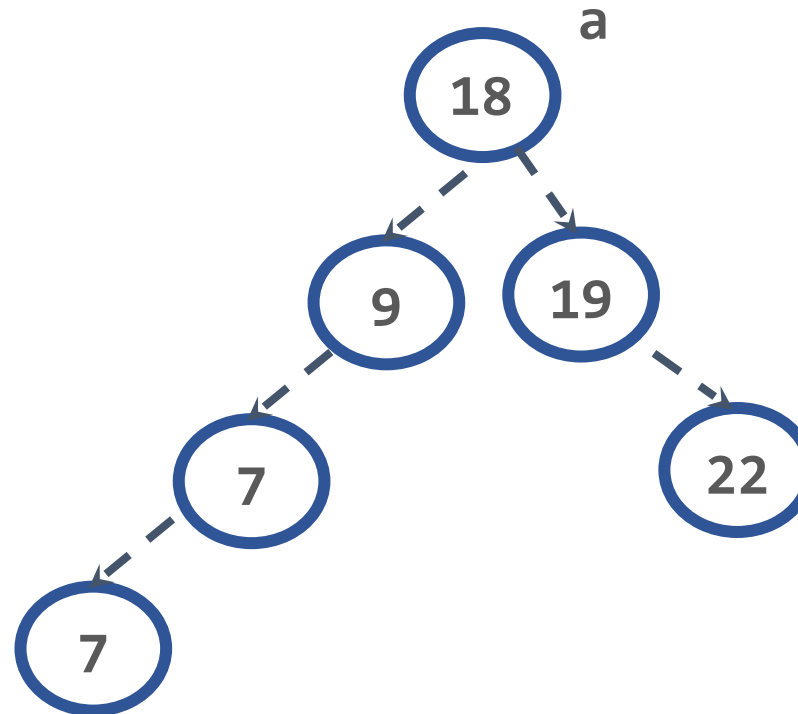


# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

Suponga que se leen los siguientes valores y se quieren ir agregando en un ABB (9, 18, 22, 19, 7, 50)



Qué ocurre si se leen valores repetidos?  
(18, 9, 7, 7, 19, 22, 50)



Cómo lo implementamos?  
Cuál sería el caso base?



# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

Programa arboles;

Type

arbol = ^nodo;

nodo = record

dato: integer;

HI: arbol;

HD: arbol;

end;

Var

abb:arbol; x:integer;

Begin

abb:=nil;

read (x);

while (x<>50)do

begin

AGREGAR(abb,x);

read(x);

end;

End.

Procedure agregar (**var** a:árbol; num:integer);

Begin

if (a = nil) then

begin

new(A);

a^.dato:= num; a^.HI:= nil; a^.HD:= nil;

end

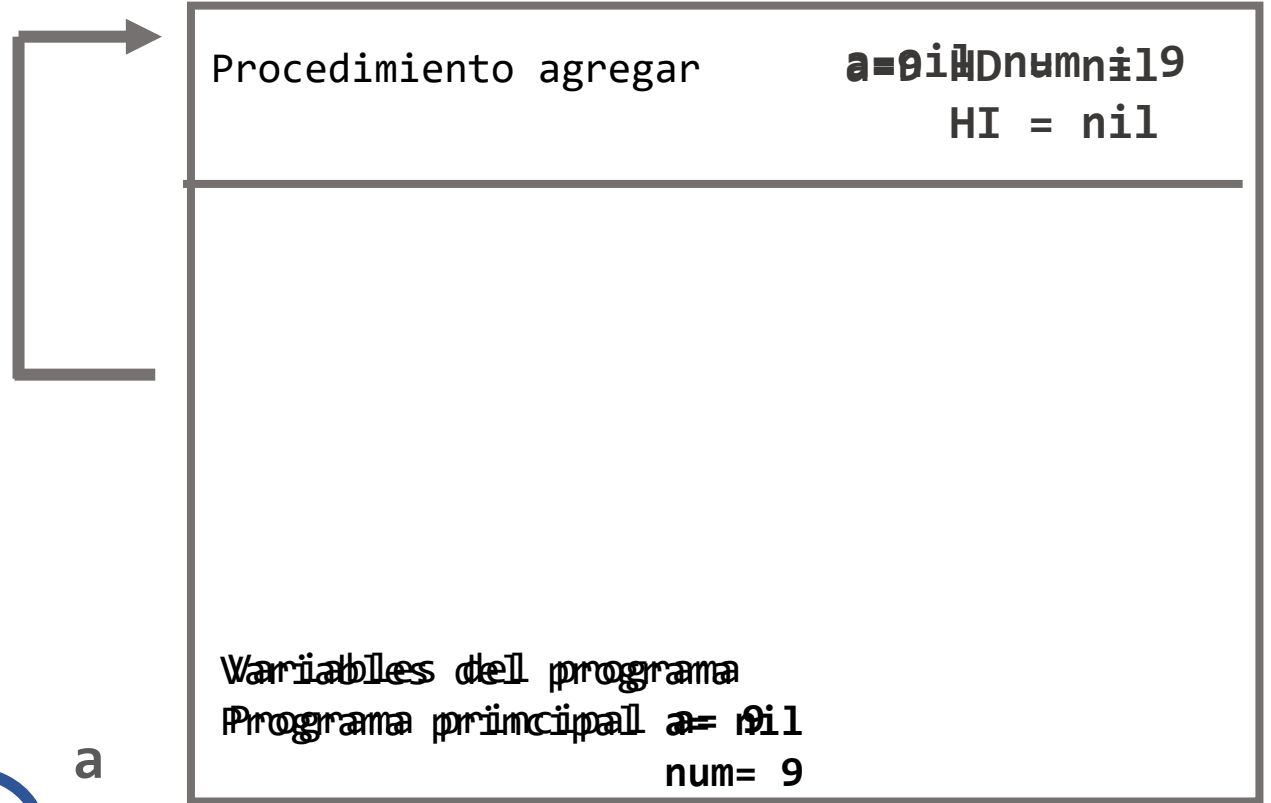
else

if (num <= A^.dato) then **agregar**(a^.HI,num)

else **agregar** (a^.HD,num)

End;

Cómo  
funciona?



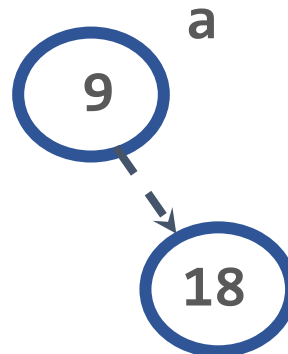
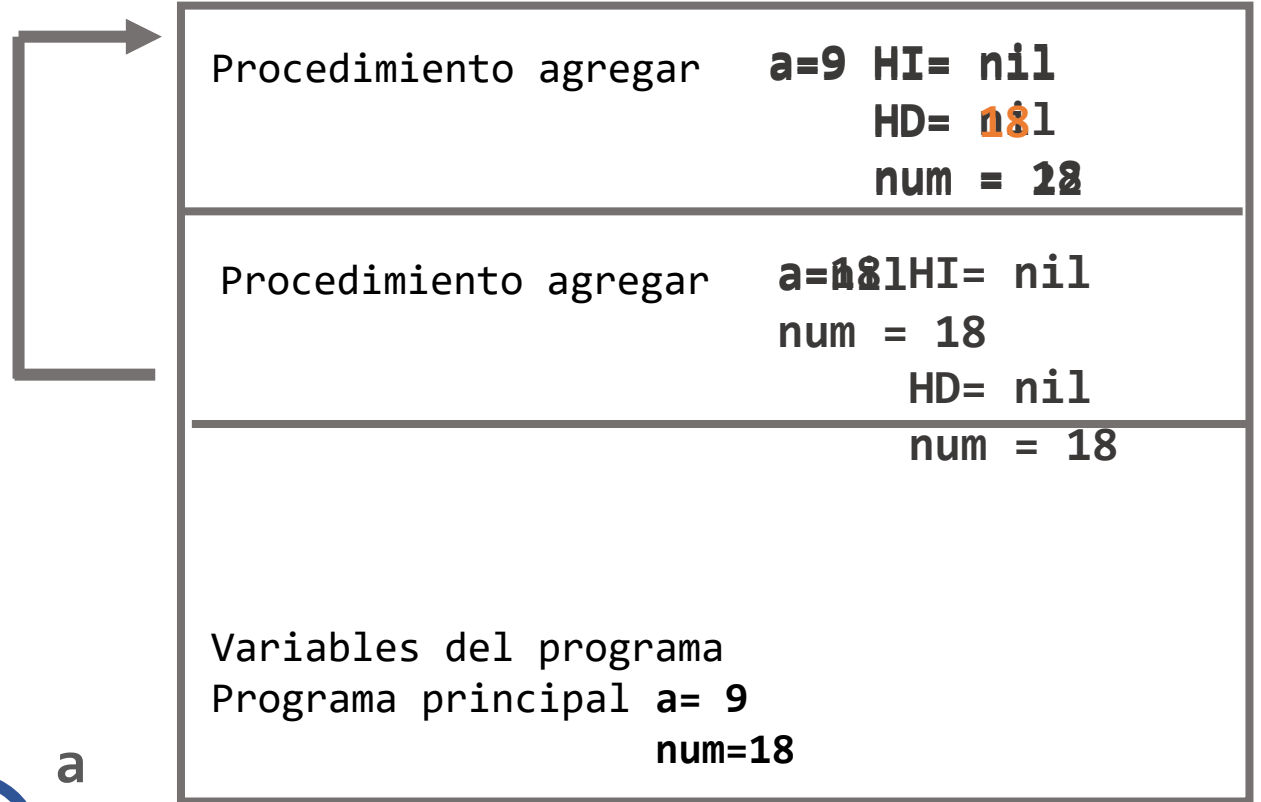




# ÁRBOLES BINARIOS DE BÚSQUEDA- CREACION

(9, 18, 22, 7)

```
Procedure agregar (var a:arbol; num:integer);
Begin
  if (a = nil) then
    begin
      new(a);
      a^.dato:= num; a^.HI:= nil; a^.HD:= nil;
    end
  else
    if (num <= a^.dato) then agregar(a^.HI,num)
    else agregar(a^.HD,num)
  End;
```





## Clase 3 -1– Módulo Imperativo





# Taller de Programación



# AGENDA



Esturctura de datos arbol

Operaciones - RECORRIDO

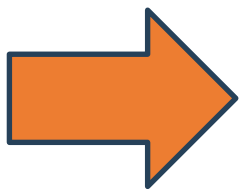


# ESTRUCTURA DE DATOS ARBOL - RECORRIDO

Es una estructura de datos jerárquica (no lineal), homogénea y dinámica.

Está formada por nodos, donde cada nodo tiene a lo sumo hijos.

El nodo principal del árbol se denomina raíz y los nodos que no tienen hijos se denominan hojas del árbol.



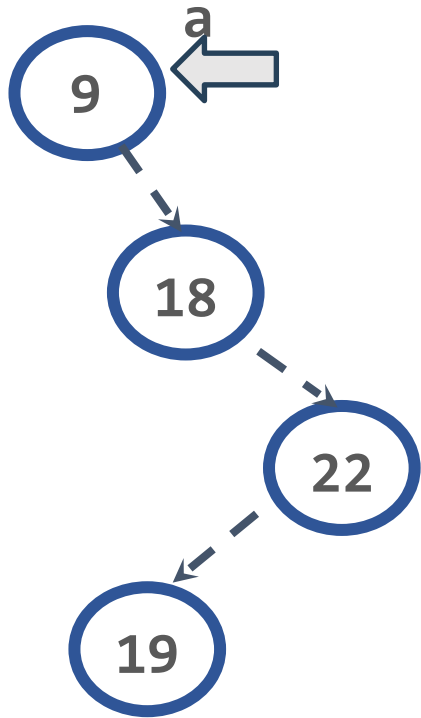
ABB

Los nodos del arbol respetan todos el mismo criterio (los hijos ubicados a la izquierda son menores al nodo padre o al revés)

Cómo  
recorremos un  
ABB?



# ESTRUCTURA DE DATOS ARBOL - RECORRIDO



Para poder recorrer un ABB siempre debe comenzarse el recorrido desde la raíz.

Una vez que se esta parado en la raíz debe hacerse la acción que se quiera con el valor (imprimir, agregar en otro arbol, agregarlo en una lista, modificarlo, etc).

Luego debe tomarse uno de sus hijos y realizar la misma acción que para el nodo padre y luego el otro de sus hijos.

Cuál es el caso base?

Cuántos llamados recursivos se hacen en cada nodo?

Cómo se implementa?



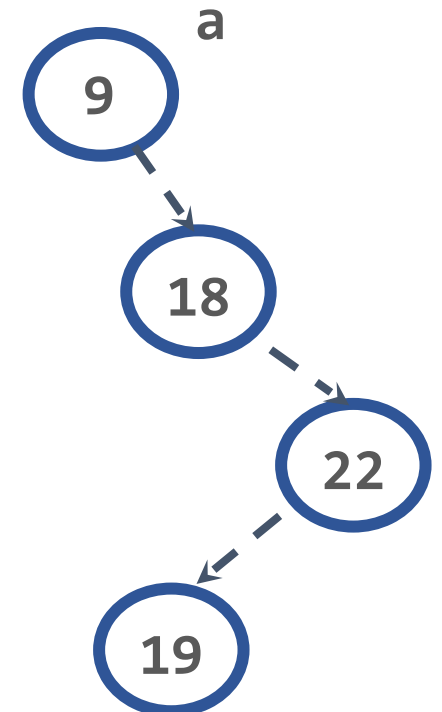
# ESTRUCTURA DE DATOS ARBOL - RECORRIDO

```
Programa arboles;  
Type  
  arbol = ^nodo;  
  nodo = record  
    dato: integer;  
    HI: arbol;  
    HD: arbol;  
  end;  
  
Var  
  a:arbol;  num:integer;  
  
Begin  
  a:= nil;  
  read (num);  
  while (num <> 50) do  
    begin  
      agregar (a,num);  
      read (num);  
    end;  
    recorrido_enOrden(a);  
  End.
```

```
Procedure enOrden ( a : arbol );  
begin  
  if ( a <> nil ) then begin  
    enOrden (a^.HI);  
    write (a^.dato); //o cualquier otra acción  
    enOrden (a^.HD);  
  end;  
end;
```

Es lo mismo  
pasar **a** por  
referencia?

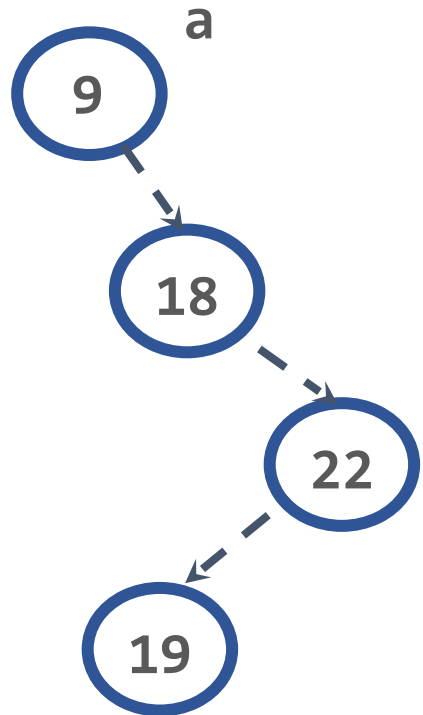
Cómo funciona?





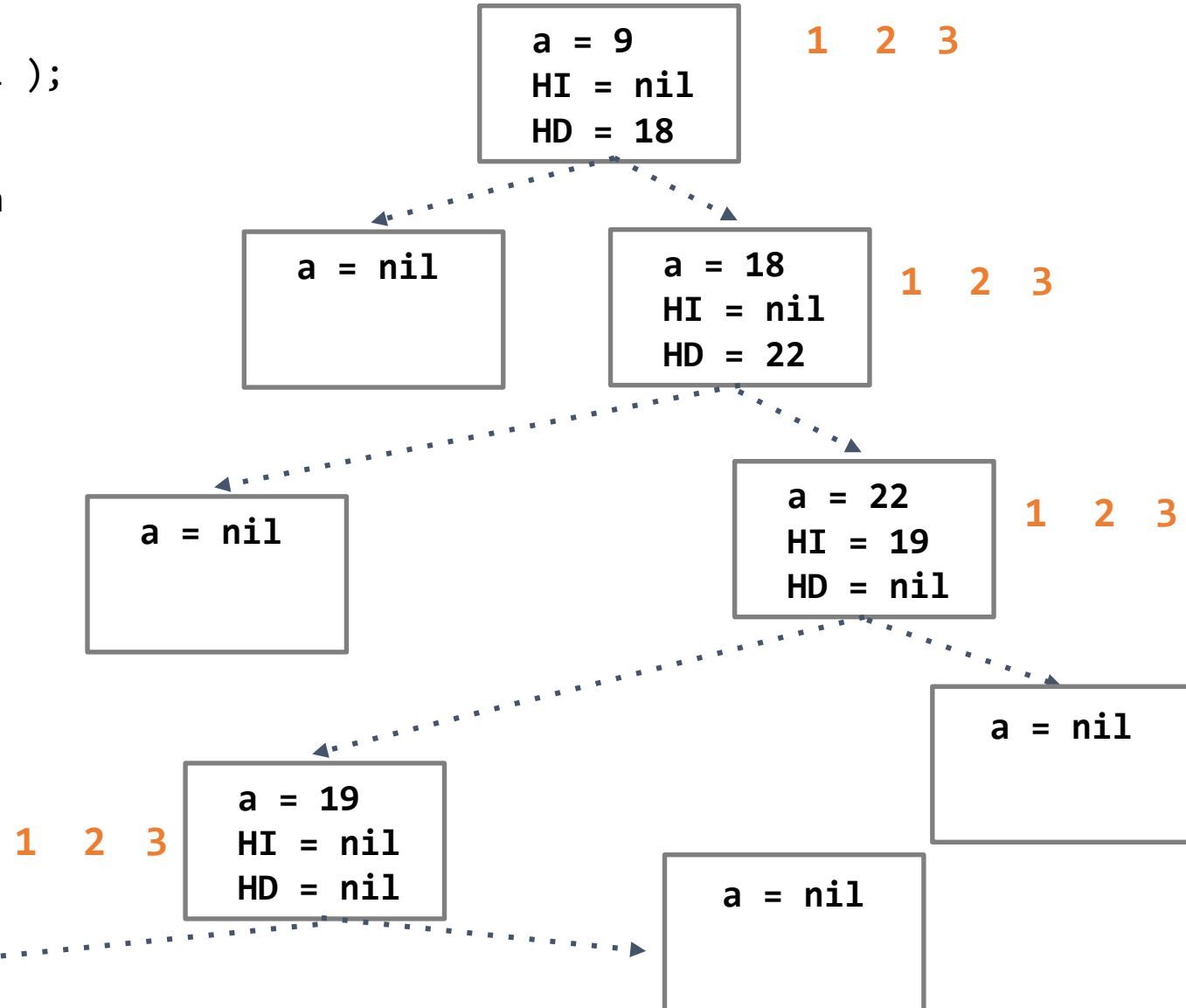


# ESTRUCTURA DE DATOS ARBOL - RECORRIDO



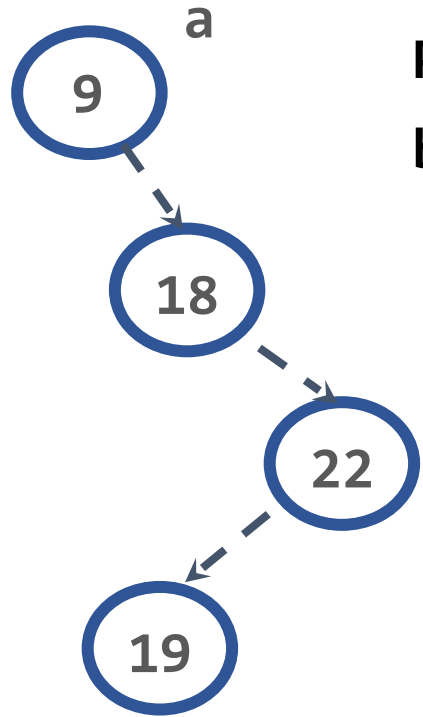
```
Procedure enOrden ( a : arbol );  
begin  
  if ( a <> nil ) then begin  
    1 enOrden (a^.HI);  
    2 write (a^.dato);  
    3 enOrden (a^.HD);  
  end;  
end;
```

Pantalla 9  
Pantalla 18  
Pantalla 19  
Pantalla 22





# ESTRUCTURA DE DATOS ARBOL - RECORRIDO



```
Procedure preOrden (a:arbol);  
begin  
  if ( a <> nil ) then  
    begin  
      write (a^.dato);  
      preOrden (a^.HI);  
      preOrden (a^.HD);  
    end;  
end;
```

```
Procedure postOrden (a:arbol);  
begin  
  if ( a <> nil ) then  
    begin  
      postOrden (a^.HI);  
      postOrden (a^.HD);  
      write (a^.dato);  
    end;  
end;
```

Qué imprimen?

Si **a** se pasa por referencia que imprime cada uno?



# Taller de Programación



# AGENDA



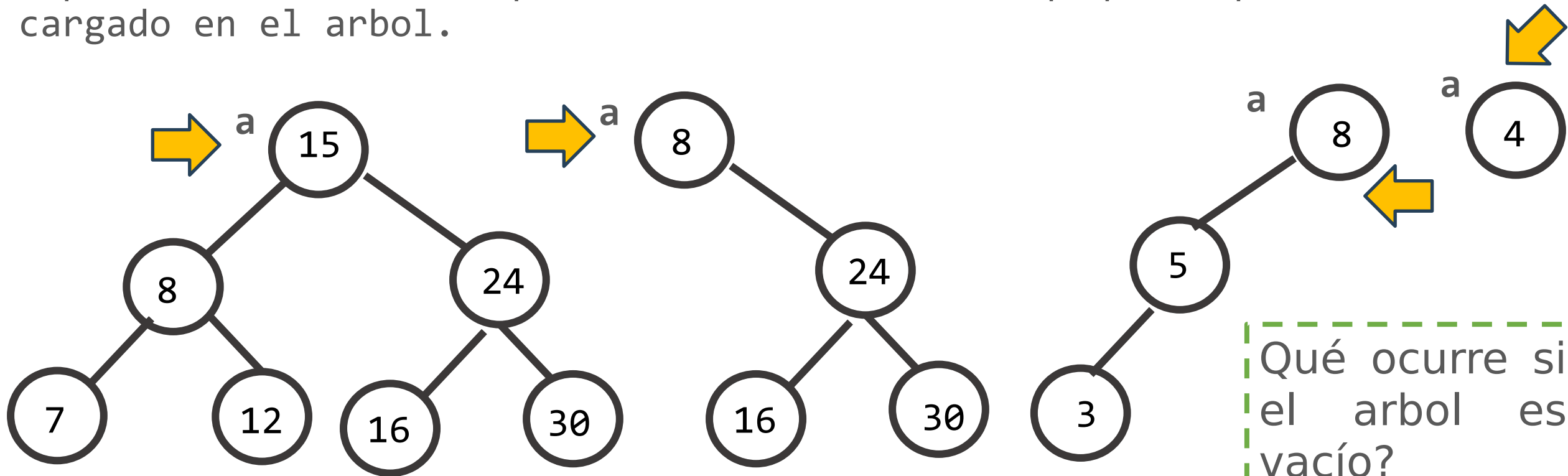
Esturctura de datos ABB – MAYOR ELEMENTO

Esturctura de datos ABB – MENOR ELEMENTO



# ARBOLES BINARIOS DE BUSQUEDA - MENOR

Supongamos que disponemos de un árbol binario de búsqueda, y queremos implementar un módulo que retorne el valor más pequeño que está cargado en el árbol.



El valor más pequeño de cualquier ABB es el valor ubicado "mas " a la izquierda

Qué ocurre si el árbol es vacío?

Cómo lo implemento?



# ARBOLES BINARIOS DE BUSQUEDA - MENOR

## DEVUELVE EL VALOR MINIMO

Programa arbolesEnteros;

Type

arbol = ^nodo;

nodo = record

dato: integer;

HI: arbol;

HD: arbol;

end;

Var

**a:arbol; min:integer;**

Begin

cargarArbol(a);

if (a <> nil) then

min:= minimo(a);

write (min);

End.

## DEVUELVE EL NODO QUE CONTIENE EL MINIMO

Programa arbolesEnteros;

Type

arbol = ^nodo;

nodo = record

dato: integer;

HI: arbol;

HD: arbol;

end;

Var

**a:arbol; min:arbol;**

Begin

cargarArbol(a);

min:= minimoNodo(a);

if (min <> nil) then write (min^.dato);

End.



# ARBOLES BINARIOS DE BUSQUEDA - MENOR

## DEVUELVE EL VALOR MINIMO

```
Programa arbolesEnteros;
```

```
Type
```

```
    arbol = ^nodo;
```

```
    nodo = record
```

```
        dato: integer;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol; min:integer;
```

```
Begin
```

```
    cargarArbol(a);
```

```
    if (a <> nil) then
```

```
        min:= minimo(a);
```

```
        write (min);
```

```
End.
```

```
function minimo (a:arbol): integer;
```

```
begin
```

```
    if (a^.HI = nil) then
```

```
        minimo:= a^.dato
```

```
    else mínimo:= mínimo (a^.HI);
```

```
end;
```

```
function minimo (a:arbol): integer;
```

```
begin
```

```
    while (a^.HI <> nil) do
```

```
        a:= a^.HI;
```

```
        minimo:= a^.dato
```

```
end;
```

**Puede  
implementarse  
de manera  
iterativa?**



# ARBOLES BINARIOS DE BUSQUEDA - MENOR

## DEVUELVE EL NODO QUE CONTIENE EL MINIMO

```
Programa arbolesEnteros;
```

```
Type
```

```
    arbol = ^nodo;
```

```
    nodo = record
```

```
        dato: integer;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol; min:arbol;
```

```
Begin
```

```
    cargarArbol(a);
```

```
    min:= minimoNodo(a);
```

```
    if (min <> nil) then write (min^.dato);
```

```
End.
```

```
function minimoNodo (a:arbol): arbol;
```

```
Begin
```

```
    if (a = nil) then minimoNodo:= nil
```

```
    else if (a^.HI = nil) then
```

```
        minimoNodo:= a;
```

```
    else mínimoNodo:= mínimoNodo (a^.HI);
```

```
end;
```

```
function minimoNodo (a:arbol): arbol;
```

```
Begin
```

```
    if (a = nil) then minimoNodo:= nil
```

```
    else
```

```
        while (a^.HI <> nil) do
```

```
            a:= a^.HI;
```

```
        minimoNodo:= a
```

```
end;
```

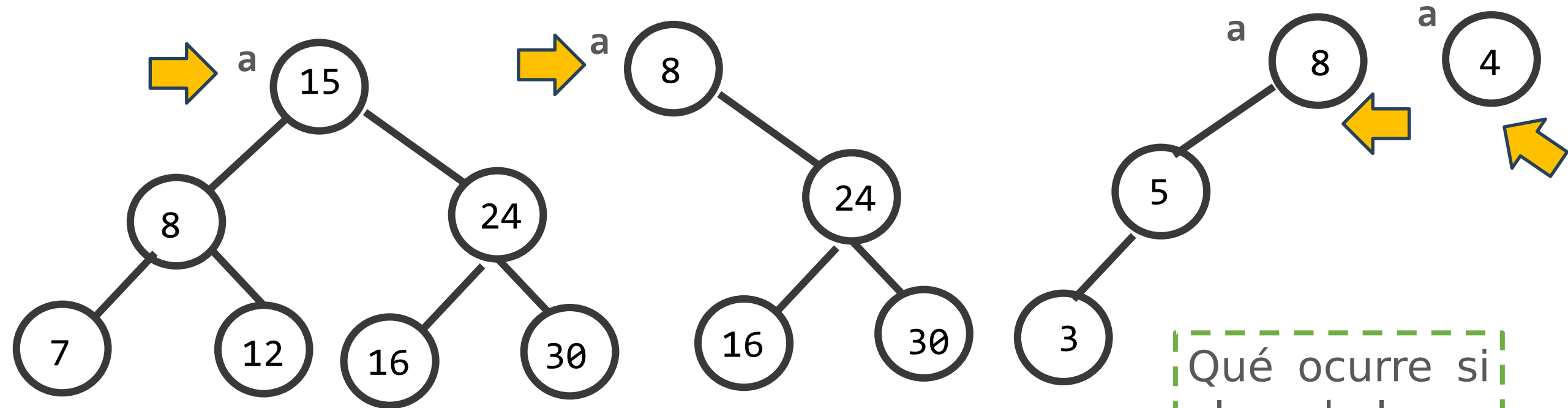
Puede  
implementarse  
de manera  
iterativa?





# ARBOLES BINARIOS DE BUSQUEDA - MAXIMO

Supongamos que disponemos de un árbol binario de búsqueda, y queremos implementar un modulo que retorne el valor más grande que está cargado en el árbol.



El valor más grande de cualquier ABB es el valor ubicado "mas " a la derecha

Qué ocurre si el árbol es vacío?

Cómo implementarlo?



# ARBOLES BINARIOS DE BUSQUEDA - MAXIMO

## DEVUELVE EL VALOR MAXIMO

```
Programa arbolesEnteros;
```

```
Type
```

```
    arbol = ^nodo;
```

```
    nodo = record
```

```
        dato: integer;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol; max:integer;
```

```
Begin
```

```
    cargarArbol(a);
```

```
    if (a <> nil) then
```

```
        max:= maximo(a);
```

```
        write (max);
```

```
End.
```

## DEVUELVE EL NODO QUE CONTIENE EL MAXIMO

```
Programa arbolesEnteros;
```

```
Type
```

```
    arbol = ^nodo;
```

```
    nodo = record
```

```
        dato: integer;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol; max:arbol;
```

```
Begin
```

```
    cargarArbol(a);
```

```
    max:= maximoNodo(a);
```

```
    if (max <> nil) then    write (max^.dato);
```

```
End.
```



# ARBOLES BINARIOS DE BUSQUEDA - MAXIMO

## DEVUELVE EL VALOR MAXIMO

```
Programa arbolesEnteros;
```

```
Type
```

```
    arbol = ^nodo;
```

```
    nodo = record
```

```
        dato: integer;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol; max:integer;
```

```
Begin
```

```
    cargarArbol(a);
```

```
    if (a <> nil) then
```

```
        max:= maximo(a);
```

```
        write (max);
```

```
End.
```

```
function maximo (a:arbol): integer;
```

```
begin
```

```
    if (a^.HD = nil) then
```

```
        maximo:= a^.dato
```

```
    else maximo:= maximo (a^.HD);
```

```
end;
```

```
function maximo (a:arbol): integer;
```

```
begin
```

```
    while (a^.HD <> nil) do
```

```
        a:= a^.HD;
```

```
        maximo:= a^.dato;
```

```
end;
```

**Puede  
implementarse  
de manera  
iterativa?**



# ARBOLES BINARIOS DE BUSQUEDA - MAXIMO

## DEVUELVE EL NODO QUE CONTIENE EL MAXIMO

```
Programa arbolesEnteros;
```

```
Type
```

```
    arbol = ^nodo;
```

```
    nodo = record
```

```
        dato: integer;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol; max:arbol;
```

```
Begin
```

```
    cargarArbol(a);
```

```
    max:= maximoNodo(a);
```

```
    if (max <> nil) then write (max^.dato);
```

```
End.
```

```
function maximoNodo (a:arbol): arbol;
```

```
Begin
```

```
    if (a = nil) then maximoNodo:= nil
```

```
    else if (a^.HD = nil) then
```

```
        maximoNodo:= a
```

```
    else maximoNodo:= maximoNodo (a^.HD);
```

```
end;
```

```
function maximoNodo (a:arbol): arbol;
```

```
Begin
```

```
    if (a = nil) then maximoNodo:= nil
```

```
    else
```

```
        while (a^.HD <> nil) do
```

```
            a:= a^.HD;
```

```
            maximoNodo:= a;
```

```
end;
```

**Puede  
implementarse  
de manera  
iterativa?**



# Taller de Programación



# AGENDA

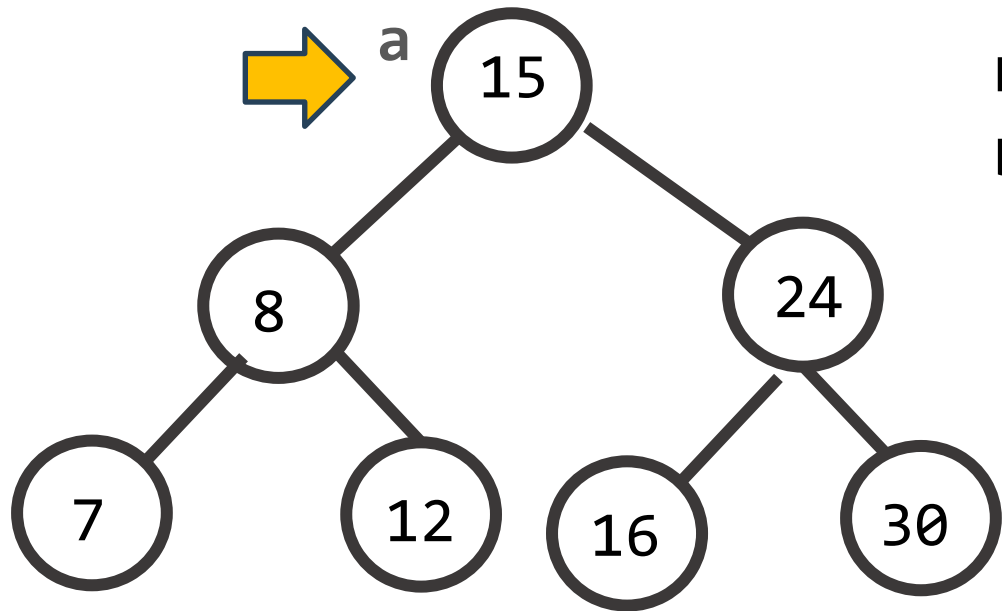


## Esturctura de datos ABB – BUSCAR UN VALOR



# ARBOLES BINARIOS DE BUSQUEDA - BUSCAR

Supongamos que disponemos de un árbol binario de búsqueda, y queremos implementar un módulo que retorne un booleano si se encuentra un valor buscado que se recibe como parámetro.



```
Procedure Buscar (a:arbol;x:integer;var ok:boolean);  
begin  
    if ( a <> nil ) then begin  
        buscar (a^.HI,x,ok);  
        if (a^.dato = x) then ok:= true;  
        buscar (a^.HD,x,ok);  
    end;  
end;
```

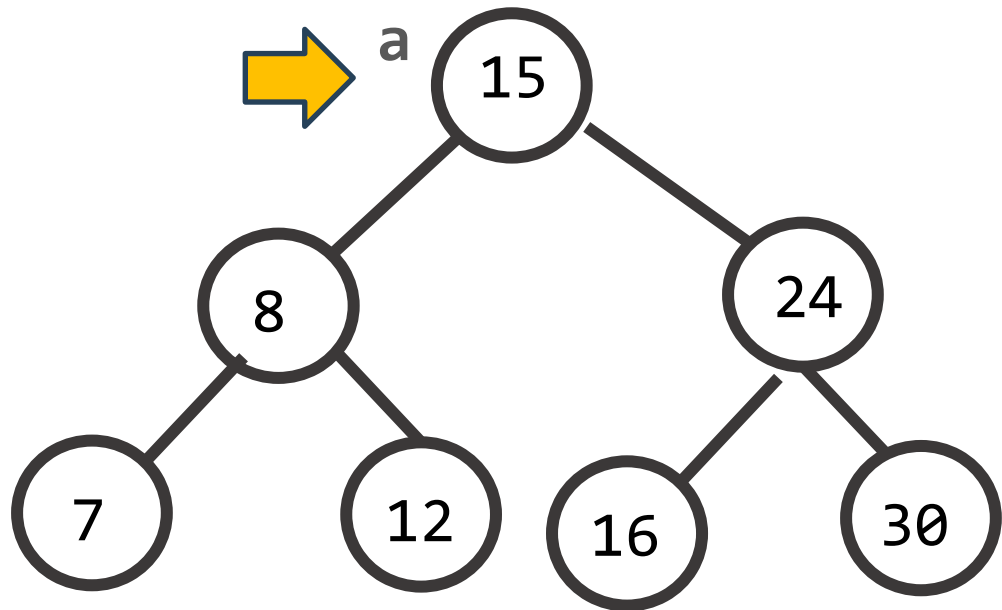


Para buscar un valor siempre se debe “aprovechar” el orden del ABB.



# ARBOLES BINARIOS DE BUSQUEDA - BUSCAR

Supongamos que disponemos de un árbol binario de búsqueda, y queremos implementar un módulo que retorne el nodo donde se encuentra un valor buscado que se recibe como parámetro.



```
Procedure Buscar (a:arbol;x:integer;var ab:arbol);  
begin  
  if ( a <> nil ) then begin  
    buscar (a^.HI,x,ab);  
    if (a^.dato = x) then ab:= a;  
    buscar (a^.HD,x,ab);  
  end;  
end;
```

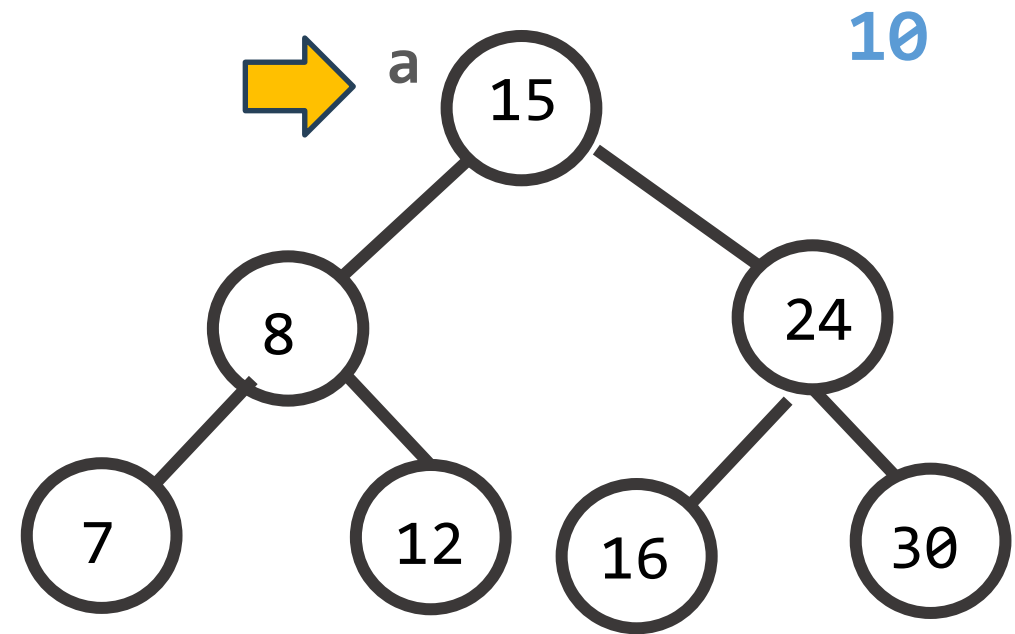
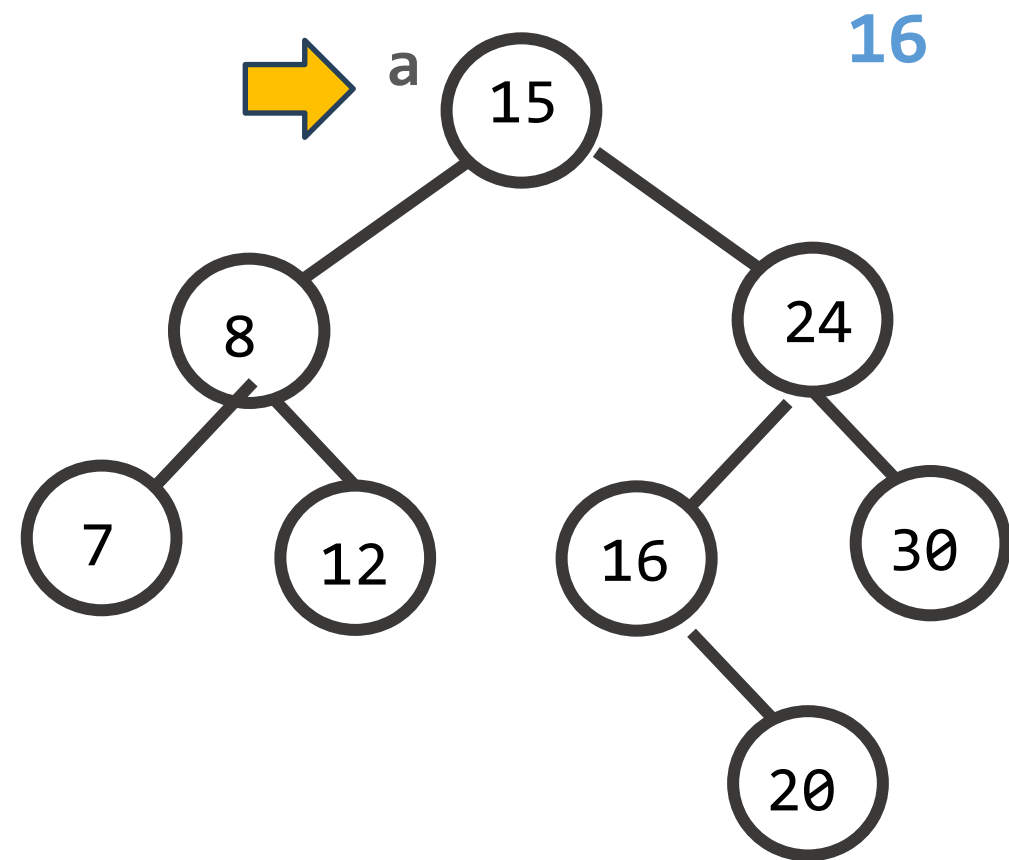






# ARBOLES BINARIOS DE BUSQUEDA - BUSCAR

Supongamos que disponemos de un árbol binario de búsqueda, y queremos implementar un módulo que retorne el nodo donde se encuentra un valor buscado que se recibe como parámetro.



Cómo  
implementarlo?



# ARBOLES BINARIOS DE BUSQUEDA - BUSCAR

## DEVUELVE EL VALOR BOOLEAN

Programa arbolesEnteros;

Type

```
arbol = ^nodo;  
nodo = record  
    dato: integer;  
    HI: arbol;  
    HD: arbol;  
end;
```

Var

**a:arbol;min:integer;ok:booleanx:integer**

Begin

```
cargarArbol(a); read(x);  
if (a <> nil) then  
    ok:= buscar(a,x);  
    write (ok);
```

End.

## DEVUELVE EL NODO QUE CONTIENE EL BUSCADO

Programa arbolesEnteros;

Type

```
arbol = ^nodo;  
nodo = record  
    dato: integer;  
    HI: arbol;  
    HD: arbol;  
end;
```

Var

**a:arbol; bus:arbol; x:integer;**

Begin

```
cargarArbol(a); read(x);  
bus:= buscarNodo(a,x);  
if (bus <> nil) then write ("encontro");
```

End.



# ARBOLES BINARIOS DE BUSQUEDA - BUSCAR

**DEVUELVE EL BOOLEAN**

```
Programa arbolesEnteros;
```

```
Type
```

```
    arbol = ^nodo;
```

```
    nodo = record
```

```
        dato: integer;
```

```
        HI: arbol;
```

```
        HD: arbol;
```

```
    end;
```

```
Var
```

```
    a:arbol; min:integer; x:integer;
```

```
Begin
```

```
    cargarArbol(a); read(x);
```

```
    if (a <> nil) then
```

```
        ok:= buscar(a,x);
```

```
        write (ok);
```

```
End.
```

```
Function buscar (a:arbol; x:integer): boolean;
```

```
begin
```

```
    if (a = nil) then buscar:= false
```

```
    else (a^.dato = x) then buscar:= true
```

```
    else if (x > a^.dato) then buscar:= buscar(a^.HD, x)
```

```
    else buscar:= buscar(a^.HI, x)
```

```
end;
```

**Puede  
implementarse de  
manera iterativa?**



# ARBOLES BINARIOS DE BUSQUEDA - BUSCAR

## DEVUELVE EL NODO BUSCADO

Programa arbolesEnteros;

Type

arbol = ^nodo;

nodo = record

dato: integer;

HI: arbol;

HD: arbol;

end;

Var

a:arbol; bus:arbol; x:integer;

Begin

cargarArbol(a); read(x);

bus:= buscarNodo(a,x);

if (bus <> nil) then write ("encontro");

End.

function buscarNodo (a:arbol; x:integer): arbol;

Begin

if (a = nil) then buscarNodo:= nil

else (a^.dato = x) then buscarNodo:= a

else if (x > a^.dato) then

buscarNodo:= buscarNodo(a^.HD, x)

else buscarNodo:= buscarNodo(a^.HI, x);

End;

**Puede implementarse  
de manera iterativa?**



# ÁRBOLES – OTRAS OPERACIONES

Imprimir los valores entre rangos

Calcular el nivel de un arbol

Calcular la altura de un arbol

Obtener un árbol espejo