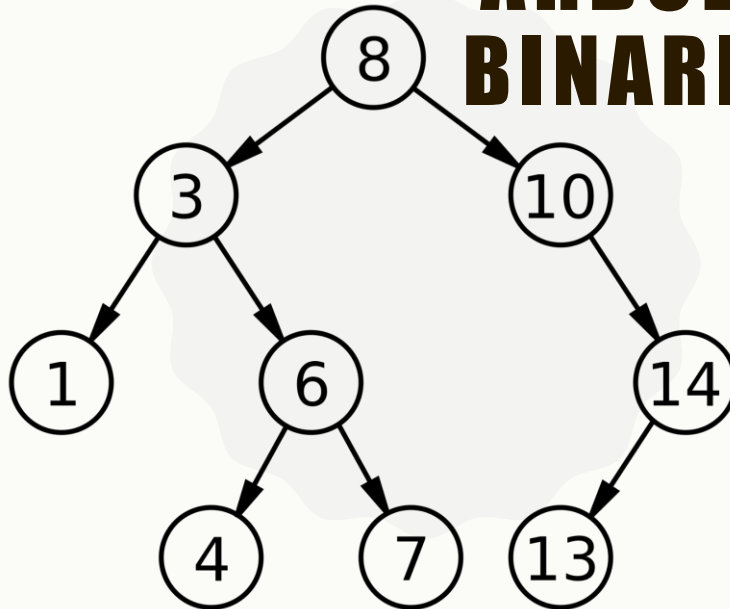


# ÁRBOLES BINARIOS

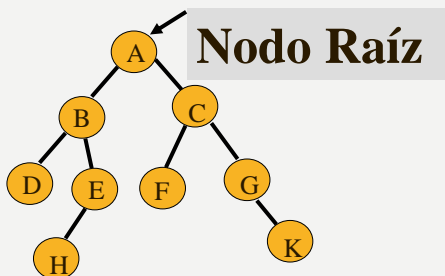


1

## GENERALIDADES

Nodo: Cada elemento en un árbol.

Nodo Raíz: Primer elemento agregado al árbol.



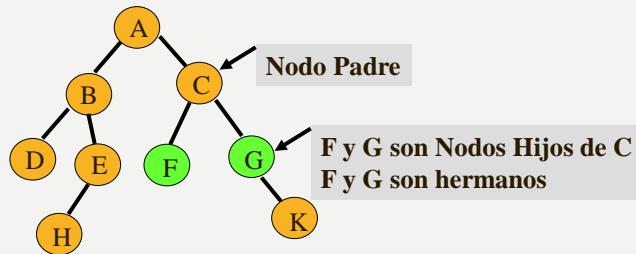
2

# GENERALIDADES

Nodo Padre: Se le llama así al nodo predecesor de un elemento.

Nodo Hijo: Es el nodo sucesor de un elemento.

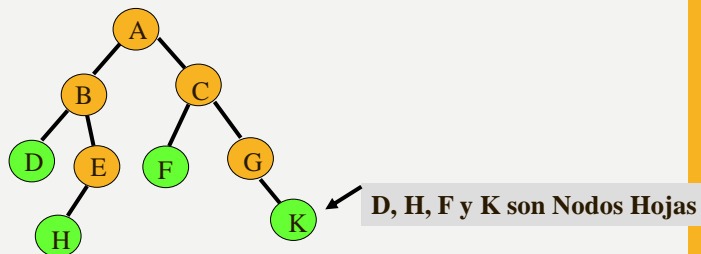
Hermanos: Nodos que tienen el mismo nodo padre.



3

# GENERALIDADES

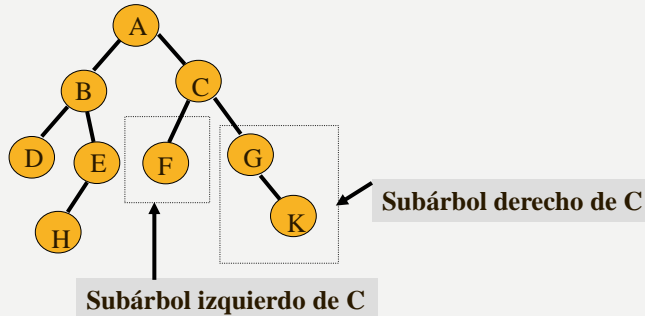
• Nodo Hoja: Aquel nodo que no tiene hijos.



4

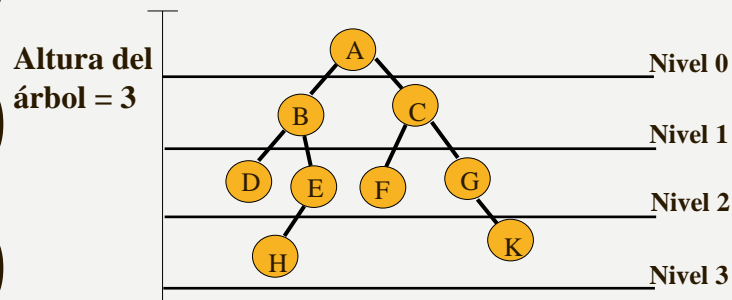
# GENERALIDADES

- Subárbol: Todos los nodos descendientes por la izquierda o derecha de un nodo.



5

# ALTURA Y NIVELES

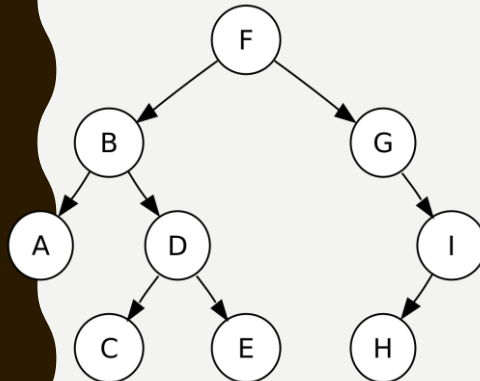


La Altura es el valor del nivel máximo.

6

Un árbol dirigido es una estructura:

- **Jerárquica** porque los componentes están a distinto nivel.
- **Organizada** porque importa la forma en que esté dispuesto el contenido.



- **Dinámica** porque su forma, tamaño y contenido pueden variar durante la ejecución.

Un árbol puede ser:

- Vacío,
- Una raíz
- Una raíz + subárboles.

7

7

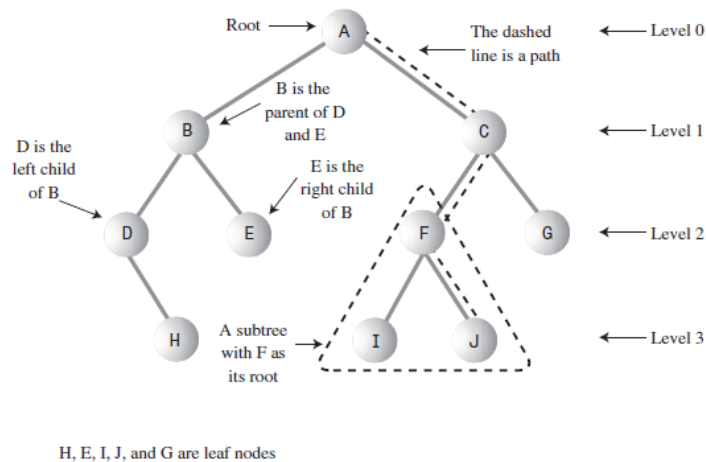
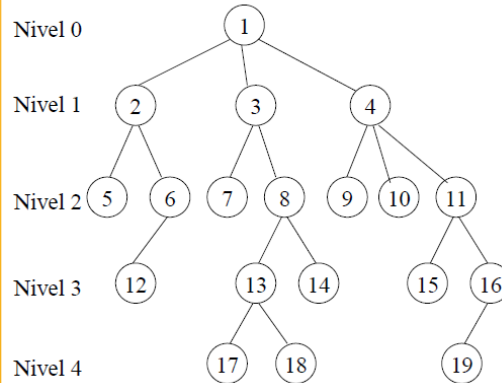


FIGURE 8.2 Tree terms.

8

8

**Ejemplo:** Considere el árbol A siguiente:



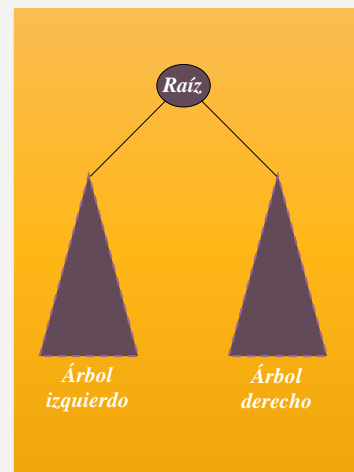
Se pide determinar:

- Su altura
- Recorridos InOrden, PreOrden, PostOrden, en Anchura
- Tabla de grados para los nodos internos.<

9

## ÁRBOLES BINARIOS

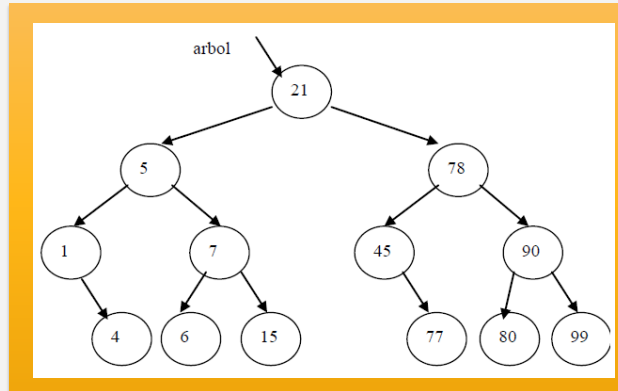
- Recursivamente un árbol binario puede definirse como:
  - un árbol vacío, o
  - un nodo raíz con un subárbol izquierdo y un subárbol derecho.



10

# ÁRBOLES BINARIOS

- Un nodo puede tener 0, 1 ó 2 hijos.
- Los dos hijos de cada nodo en un árbol binario son llamados hijo izquierdo y derecho, respectivamente, según su posición relativa al padre.
- Si tiene 0 hijos, es llamado hoja.



11

11

# IMPLEMENTACIÓN

```

typedef struct NodoArbol *Arbol;

struct NodoArbol {
    TipoDato dato;
    struct NodoArbol *izq;
    struct NodoArbol *der;
};
  
```



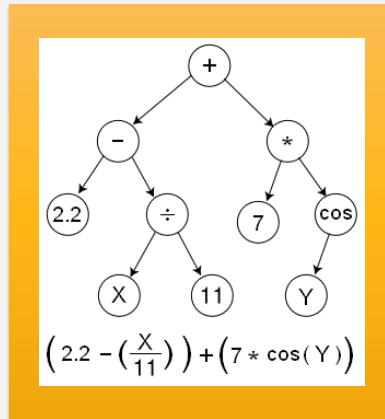
- Cada nodo del árbol consiste en:
  - Un dato (cualquier cosa)
  - Un puntero al hijo izquierdo
  - Un puntero al hijo derecho
- Inicialmente el nodo raíz apunta a NULL.
- En las hojas del árbol, los apuntadores hacia los hijos izquierdo y derecho son NULL.

12

# TIPOS DE ÁRBOLES

## Un árbol ordenado:

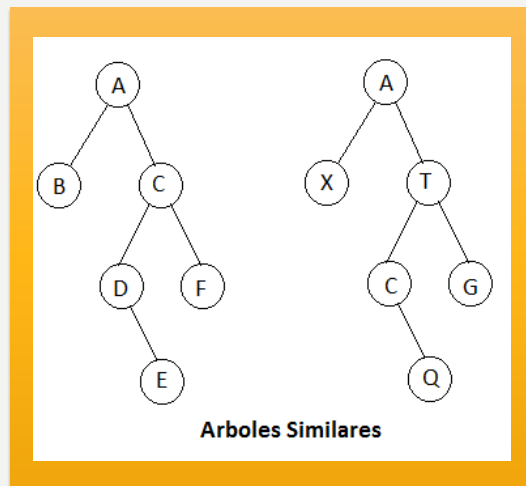
Es aquel en el que las ramas de los nodos están ordenadas con algún criterio.



13

# ÁRBOLES SIMILARES

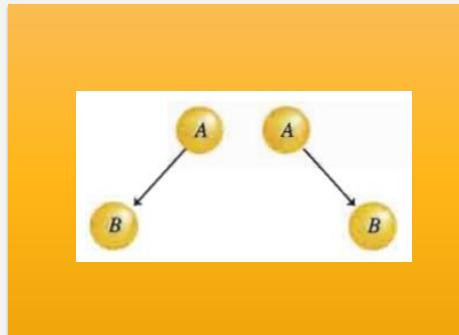
Dos árboles son similares, cuando tienen la misma estructura (forma), pero contenido diferente.



14

## ÁRBOLES DISTINTOS

- Dos árboles con diferente estructura son distintos, aunque almacenen los mismos datos.

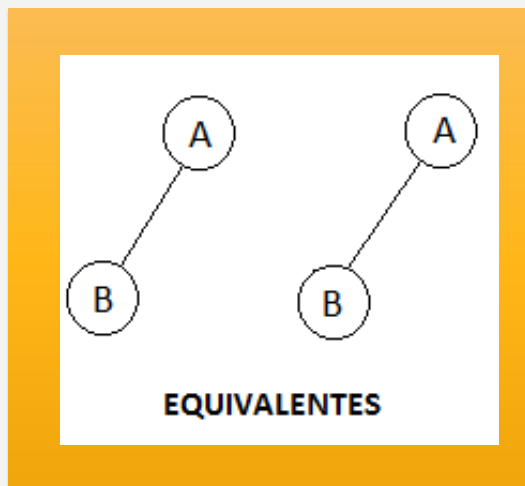


15

15

## ÁRBOLES EQUIVALENTES

Dos árboles son equivalentes cuando son similares (tienen la misma estructura o forma), y el mismo contenido.

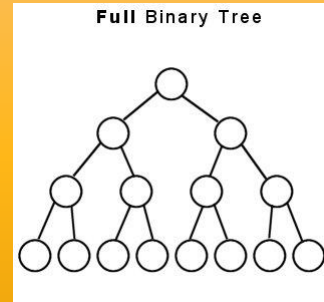


16



## ÁRBOL BINARIO LLENO

- Un árbol de altura  $h$ , está lleno si:
  - Todas sus hojas están en el nivel  $h$
  - Los nodos de altura menor a  $h$  tienen siempre 2 hijos
- Recursiva
  - Si  $T$  esta vacío,
    - Entonces  $T$  es un árbol binario lleno de altura 0
  - Si no esta vacío, y tiene  $h > 0$ 
    - Está lleno si los subárboles de la raíz, son ambos árboles binarios llenos de altura  $h-1$

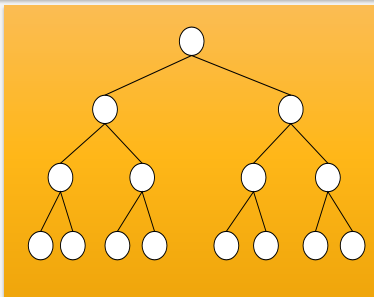


17

## TIPOS DE ÁRBOLES

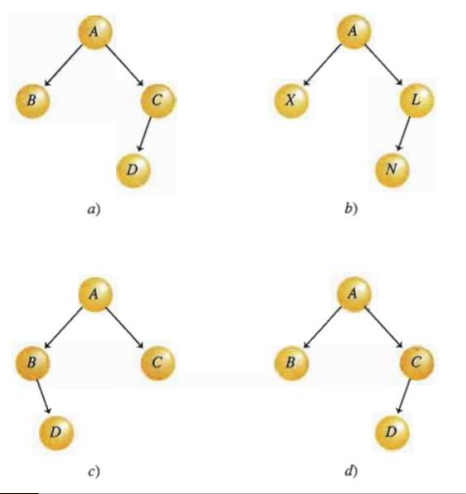
### Árbol binario lleno:

- Número de nodos en un árbol binario completo =  $2^{h+1} - 1$   
(en el ejemplo  $h = 3$ , Total =  $2^{3+1} - 1 = 15$  nodos).
- Esto ayuda a calcular el tamaño de memoria necesario para almacenar los datos de una aplicación en un árbol binario.



18

18

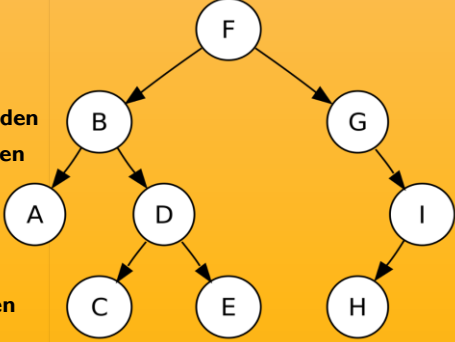


- El árbol c) es **distinto** de los árboles a), b) y d)
- Los árboles a), b) y d) son **similares**.
- Los árboles a) y d) son **equivalentes**.

Fuente: "Estructuras de Dtos" Cairó & Guardati.

19

## RECORRIDOS ESTÁNDAR

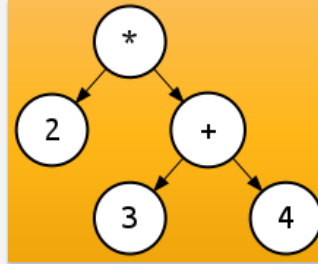


- Preorden:**
  - Procesar nodo actual
  - Procesar árbol izquierdo en PreOrden
  - Procesar árbol derecho en PreOrden
- Inorden:**
  - Procesar árbol izquierdo en Inorden
  - Procesar nodo actual
  - Procesar árbol derecho en Inorden
- Postorden:**
  - Procesar árbol izquierdo en Postorden
  - Procesar árbol derecho en Postorden
  - Procesar nodo actual

- Preorden:** F, B, A, D, C, E, G, I, H
- Inorden:** A, B, C, D, E, F, G, H, I
- Postorden:** A, C, E, D, B, H, I, G, F
- Amplitud:** F, B, G, A, D, I, C, E, H

20

## EJEMPLO:

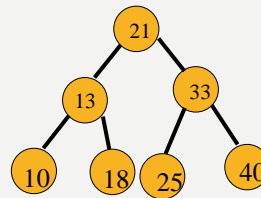


- **Preorden:** \*2+34
- **Inorden:** 2\*3+4
- **Postorden:** 234+\*
- **Amplitud:** \*, 2, +, 3, 4

21

## RECORRIDO PREORDEN

- **Proceso:**
  - Visita el nodo raíz del árbol.
  - Recorre el preorden el subárbol izquierdo del nodo raíz.
  - Recorre el preorden el subárbol derecho del nodo raíz.
- **Aplicación:** Generar una réplica del árbol.



### Recorrido en Preorden

21, 13, 10, 18, 33, 24, 40

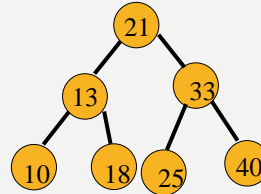
22

# RECORRIDO INORDEN

- **Proceso:**

- Recorre en inorden el subárbol izquierdo.
- Visita la raíz del árbol.
- Recorre en inorden el subárbol derecho.

- **Aplicación:** Desplegar en orden creciente los elementos del árbol si este es un ABB.



## Recorrido en Inorden

10, 13, 18, 21, 25, 33, 40

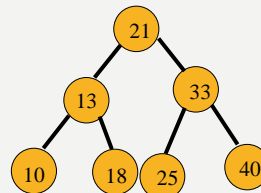
23

# RECORRIDO POSTORDEN

- **Proceso:**

- Recorre en postorden el subárbol izquierdo.
- Recorre en postorden el subárbol derecho.
- Visita la raíz del árbol.

- **Aplicación:** Liberar los nodos de un árbol.

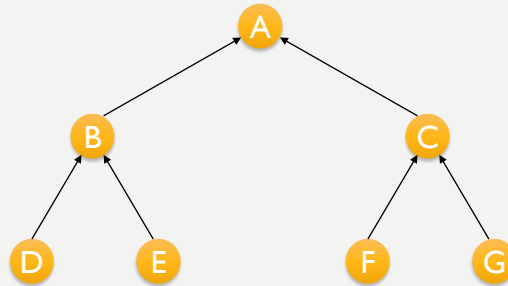


## Recorrido en Postorden

10, 18, 13, 25, 40, 33, 21

24

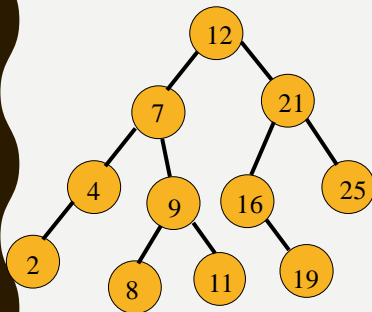
## EJEMPLO DE RECORRIDOS



PREORDEN:  
INORDEN:  
POSTORDEN:  
AMPLITUD:

25

## EJEMPLO....



### Recorrido en Preorden

12, 7, 4, 2, 9, 8, 11, 21, 16, 19, 25

### Recorrido en Inorden

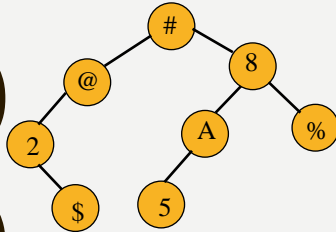
2, 4, 7, 8, 9, 11, 12, 16, 19, 21, 25

### Recorrido en Postorden

2, 4, 8, 11, 9, 7, 19, 16, 25, 21, 12

26

## EJEMPLO....



### Recorrido en Preorden

#, @, 2, \$, 8, A, 5, %

### Recorrido en Inorden

2, \$, @, #, 5, A, 8, %

### Recorrido en Postorden

\$, 2, @, 5, A, %, 8, #

27

## DADOS 2 RECORRIDOS, CONSTRUIR EL ÁRBOL BINARIO ORIGINAL:

Dado los siguientes recorridos:

### Recorrido en Preorden

\$, %, A, &, #

### Recorrido en Inorden

A, %, &, \$, #

El **Preorden** indica que la raíz es: \$

El **Inorden** indica quién está a la izquierda y quién a la derecha

El **Preorden** también indica cuál es el siguiente valor a procesar.

Paso 1

\$, %, A, &, #

A, %, & \$ #

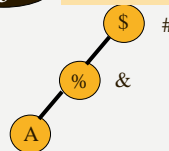
Paso 2

\$, %, A, &, #



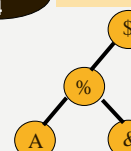
Paso 3

\$, %, A, &, #



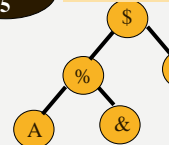
Paso 4

\$, %, A, &, #



Paso 5

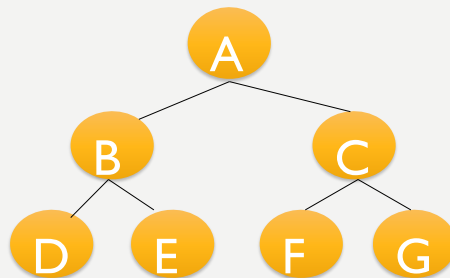
\$, %, A, &, #



28

## RECORRIDO PREORDEN

Si se hace el recorrido en preorden del árbol de la figura 1 las visitas serían en el orden siguiente:



29

## RECORRIDO EN PREORDEN

- Consiste en visitar el nodo actual (visitar puede ser simplemente mostrar la clave del nodo por pantalla), y después visitar el subárbol izquierdo y una vez visitado, visitar el subárbol derecho.
- Es un proceso recursivo por naturaleza.

30

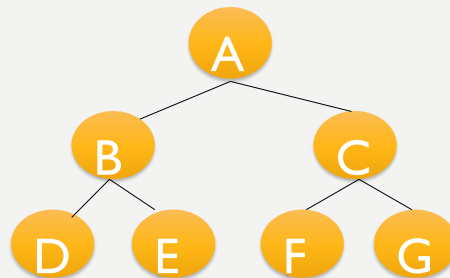
## RECORRIDO EN PREORDEN.

```
void preorden(tarbol *a)
{
  if (a != NULL)
  {
    visitar(a);
    preorden(a->izq);
    preorden(a->der);
  }
}
```

31

## RECORRIDO EN INORDEN U ORDEN CENTRAL:

- Se visita el subárbol izquierdo, el nodo actual, y después se visita el subárbol derecho:.



INORDEN: D, B, E, A, F, C, G

32



## RECORRIDO INORDEN:

```
void inorden(tarbol *a)
{
  if (a != NULL)
  {
    inorden(a->izq);
    visitar(a);
    inorden(a->der);
  }
}
```

33

## ALGORITMO INORDEN CON PILA

```
1. proc inorden-it-simp(a : árbol)
2. var p : árbol, pila : pila-árboles
3.   pila := pila-vacia()
4.   p := a
5.   mientras tiene-hi?(p) hacer
6.     pila := apilar(p, pila) ; p := hijo-iz(p)
7.   fmientras
8.   pila := apilar(p, pila)
9.   mientras ¬es-vacia?(pila) hacer
10.    p := cima(pila) ; pila := desapilar(pila)
11.    visitar(raíz(p))
12.    si tiene-hd?(p) entonces
13.      p := hijo-dr(p)
14.      mientras tiene-hi?(p) hacer
15.        pila := apilar(p, pila) ; p := hijo-iz(p)
16.      fmientras
17.      pila := apilar(p, pila)
18.    fsi
19.  fmientras
20. fproc
```

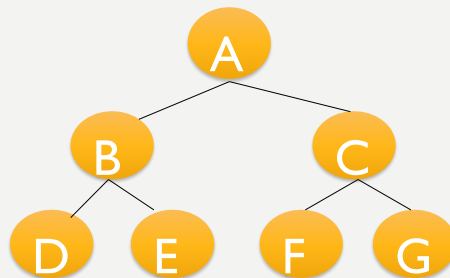
34

34

## RECORRIDO POSTORDEN

- En el ejemplo de la figura el recorrido quedaría así:

POSTORDEN: D, E, B, F, G, C, A



35

## RECORRIDO POSTORDEN:

```
void postorden(arbol *a)
{
    if (a != NULL)
    {
        postorden(a->izq);
        postorden(a->der);
        visitar(a);
    }
}
```

36

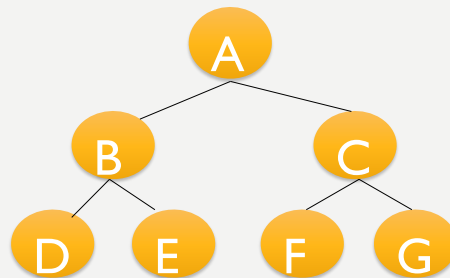
## VENTAJA DEL RECORRIDO EN POSTORDEN

- permite borrar el árbol de forma consistente.
- Si visitar se traduce por borrar el nodo actual, al ejecutar este recorrido se borrará el árbol o subárbol que se pasa como parámetro. Sin temor a borrar un nodo y desconectar sus subárboles porque al borrarlo se pueden perder los enlaces.
- Una alternativa es utilizar una variable auxiliar, pero es innecesario aplicando este recorrido.

37

## RECORRIDO EN AMPLITUD:

Si se hace el recorrido en amplitud del árbol de la figura una visitaría los nodos en este orden:



38

## Recorrido en amplitud:

- En este caso el recorrido no se realizará de forma recursiva sino iterativa, utilizando una cola como estructura de datos auxiliar.
- El procedimiento consiste en encolar (si no están vacíos) los subárboles izquierdo y derecho del nodo extraído de la cola, y seguir desencolando y encolando hasta que la cola esté vacía.

39

## IMPLEMENTACIÓN

```

1. void amplitud(arbol *a)
2. {
3.     tCola cola; /* las claves de la cola serán las del árbol binario */
4.     arbol *aux;
5.     if (a != NULL) {
6.         CrearCola(cola);
7.         encolar(cola, a);
8.         while (!colavacia(cola)) {
9.             desencolar(cola, aux);
10.            visitar(aux);
11.            if (aux->izq != NULL) encolar(cola, aux->izq);
12.            if (aux->der != NULL) encolar(cola, aux->der);
13.        }
14.    }
15. }

```

40

## EJEMPLO:

```

1. int iguales(tarbol *r1, tarbol *r2)
2. {
3.   if ((r1!=NULL)&&(r2!=NULL))
4.     if (r1->clave == r2->clave){
5.       iguales(r1->hizq, r2->hizq)*iguales(r1->hder, r2->hder);
6.     }
7.   else {
8.     if((r1== NULL)&&(r2 ==NULL))return(1);
9.     else return(0);
10.  }
11. }

```

41

41

## EJERCICIO PROPUESTO:

- Escriba una o más funciones en C que permitan determinar si un árbol binario ingresado es o no completo.
- Para el desarrollo se debe trabajar comparando el total de nodos del árbol (recorridos, contándolos uno a uno) con el total de la fórmula para un árbol binario completo ( $2^{h+1} - 1$ ).

42

42