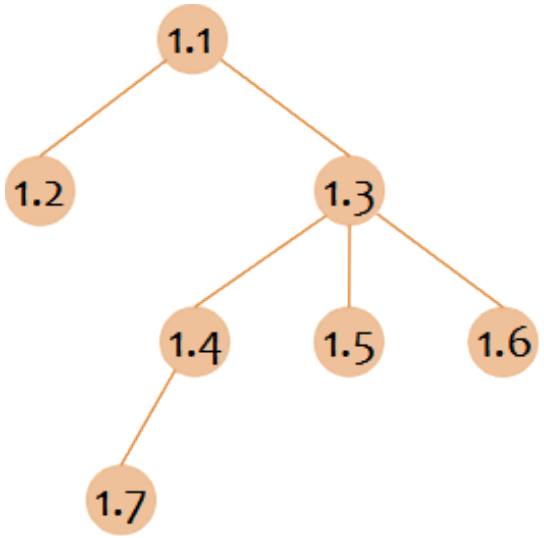
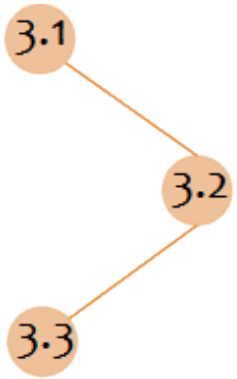
Tema 2. Árboles I

1. Conceptos básicos
2. Árboles binarios

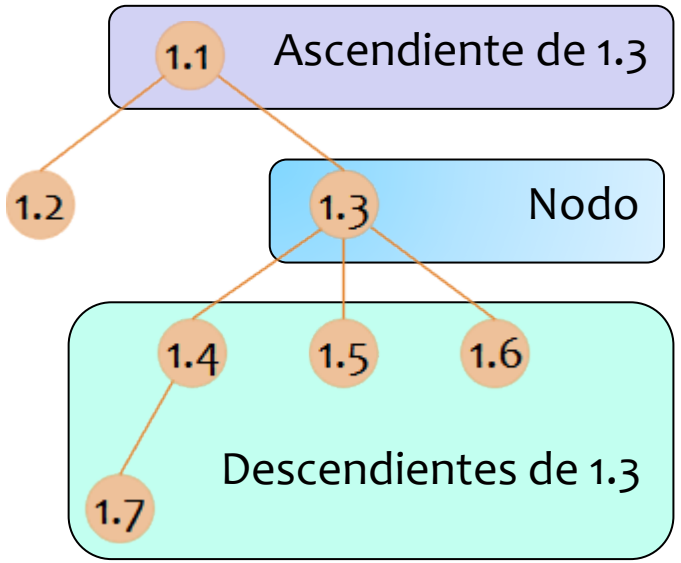
Índice

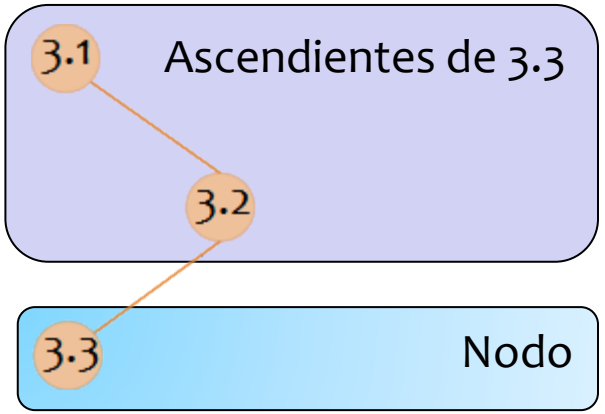
- **Definición:**
  - Estructura de datos no lineal, con una organización jerárquica y elementos homogéneos donde cada elemento puede tener varios elementos sucesores, pero un único elemento antecesor
  - No puede haber rutas circulares en las conexiones de un árbol, de tal forma que sólo puede existir una ruta única desde cada nodo hasta la raíz





- **Conceptos:**
  - **Nodo:** Cada uno de los componentes o elementos del árbol
  - **Ascendiente/antecesor:** Nodo en un nivel superior dentro de la estructura jerárquica del árbol, accesible por ascenso repetido de hijo a padre.
  - **Descendiente/sucesor:** Nodo en un nivel inferior dentro de la estructura jerárquica del árbol, accesible por descenso repetido de padre a hijo.

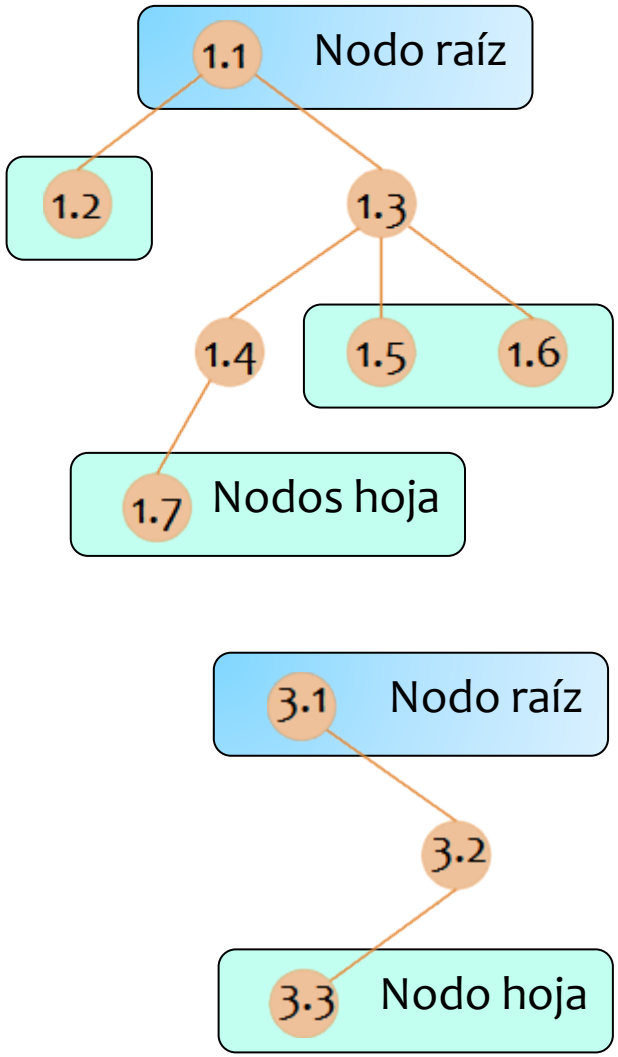




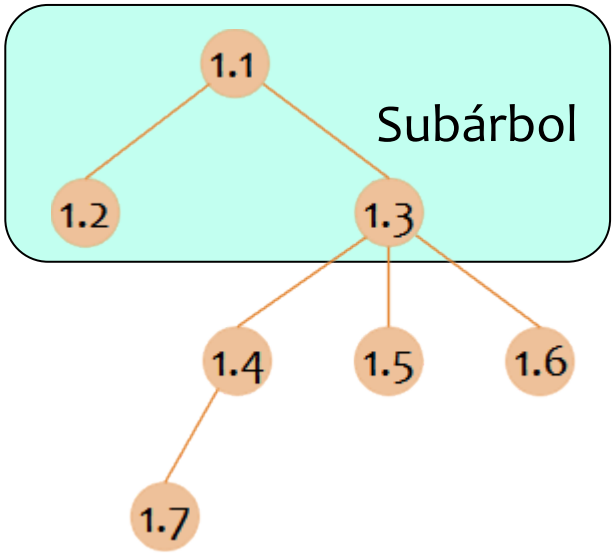
- **Conceptos:**

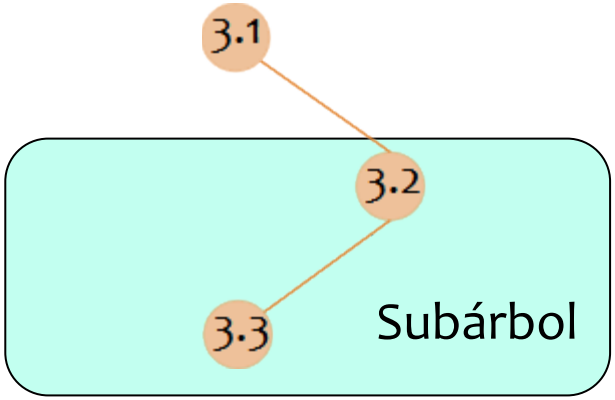
  - **Nodo Raíz:** Nodo inicial del árbol. Se encuentra en el nivel superior de la estructura jerárquica del árbol (nivel 1). **Es el único que no tiene padre.**

  - **Nodo Hoja:** Nodo terminal del árbol. Son todos aquellos nodos que no tienen descendientes y se encuentran en los niveles más bajos de la estructura jerárquica del árbol, aunque pueden estar en distintos niveles del mismo.

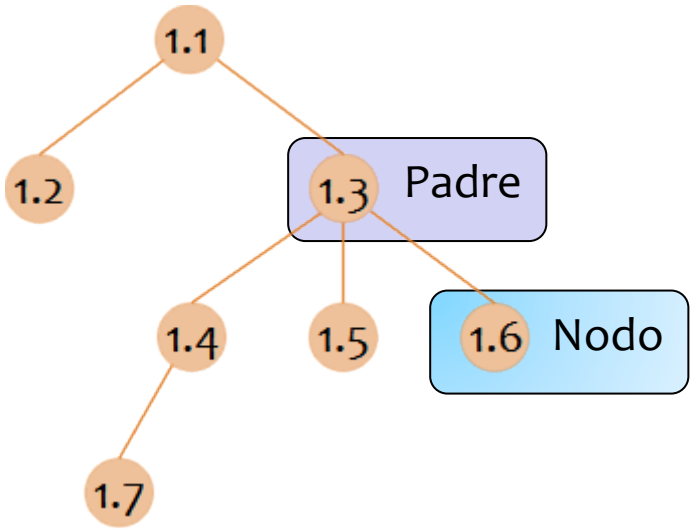


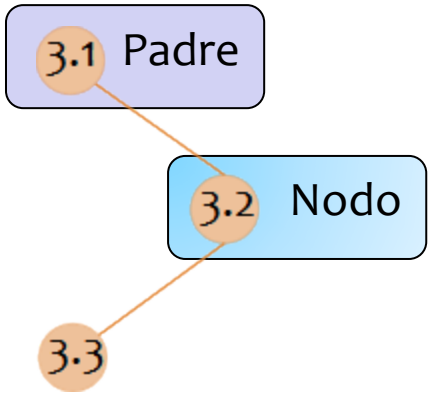
- **Conceptos:**
  - **Subárbol:** Subconjunto de nodos de un árbol que a su vez tienen estructura de árbol.



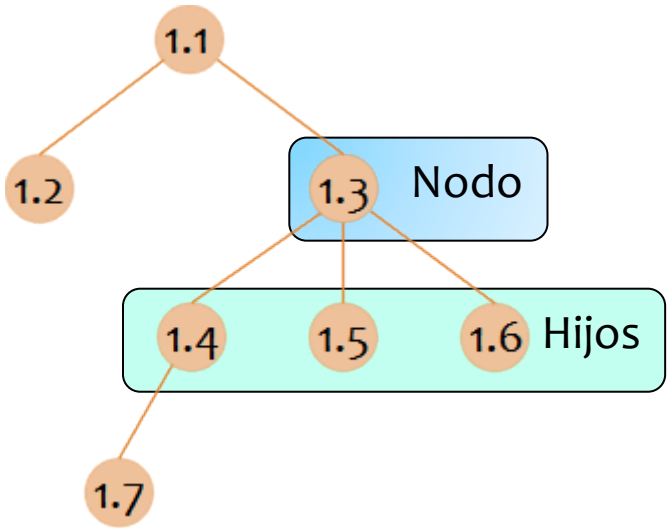


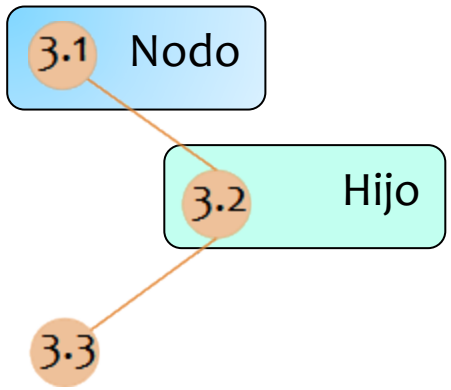
- **Conceptos:**
  - **Subárbol:** Subconjunto de nodos de un árbol que a su vez tienen estructura de árbol.
  - **Nodo padre:** Es el nodo ascendente directo de sus subárboles. En un árbol cada nodo sólo puede tener un padre, excepto el nodo raíz, que es el único que no tiene padre.



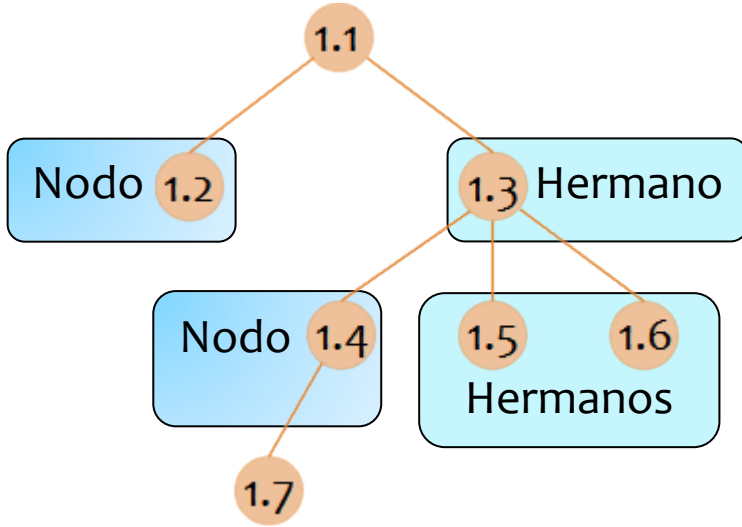


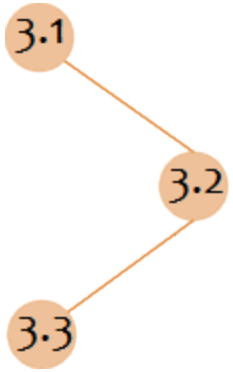
- **Conceptos:**
  - **Subárbol:** Subconjunto de nodos de un árbol que a su vez tienen estructura de árbol.
  - **Nodo padre:** Es el nodo ascendente directo de sus subárboles. En un árbol cada nodo sólo puede tener un padre, excepto el nodo raíz, que es el único que no tiene padre.
  - **Nodo hijo:** Es el nodo descendiente directo de otro nodo. En un árbol cada nodo puede tener varios hijos.



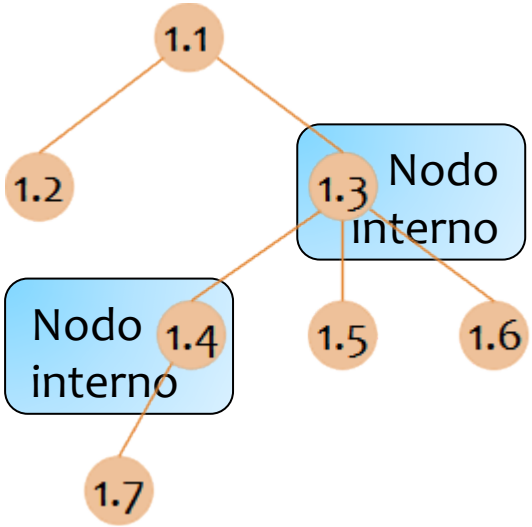


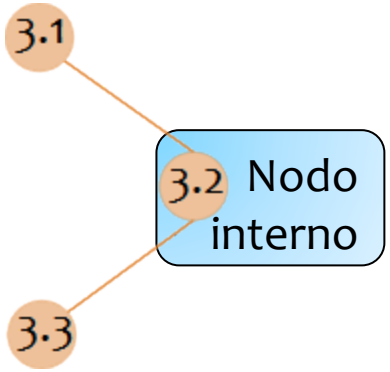
- **Conceptos:**
  - **Subárbol:** Subconjunto de nodos de un árbol que a su vez tienen estructura de árbol.
  - **Nodo padre:** Es el nodo ascendente directo de sus subárboles. En un árbol cada nodo sólo puede tener un padre, excepto el nodo raíz, que es el único que no tiene padre.
  - **Nodo hijo:** Es el nodo descendiente directo de otro nodo. En un árbol cada nodo puede tener varios hijos.
  - **Nodos hermanos**: Dos nodos son hermanos si tienen el mismo padre.





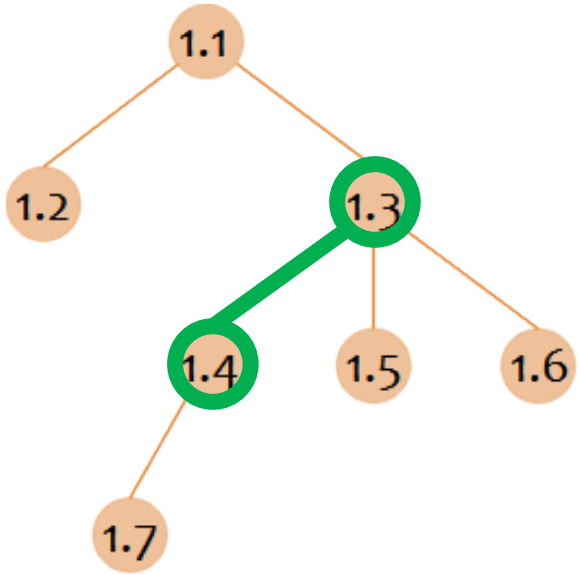
- **Conceptos:**
  - **Subárbol:** Subconjunto de nodos de un árbol que a su vez tienen estructura de árbol.
  - **Nodo padre:** Es el nodo ascendente directo de sus subárboles. En un árbol cada nodo sólo puede tener un padre, excepto el nodo raíz, que es el único que no tiene padre.
  - **Nodo hijo:** Es el nodo descendiente directo de otro nodo. En un árbol cada nodo puede tener varios hijos.
  - **Nodos hermanos**: Dos nodos son hermanos si tienen el mismo padre.
  - **Nodo interno**: Es todo nodo que no es ni nodo raíz ni nodo hoja.





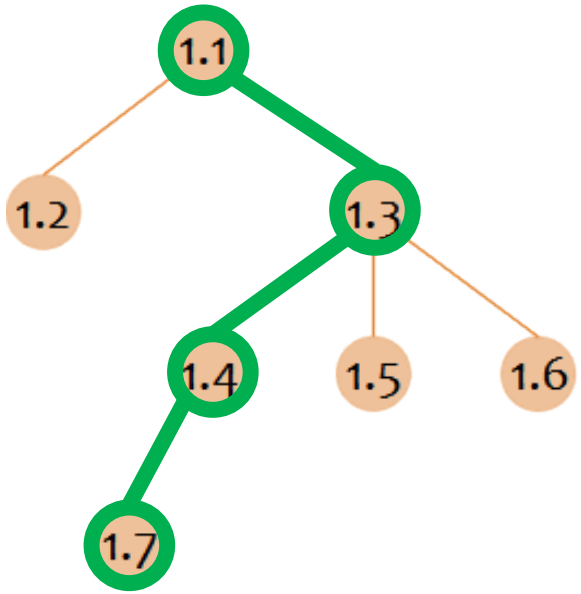
- **Conceptos:**
  - **Camino:** Secuencia de nodos donde cualquier par de nodos consecutivos son padre e hijo.
  - **Longitud del camino:** nº de nodos que forman el camino menos 1.

Longitud del camino = 1

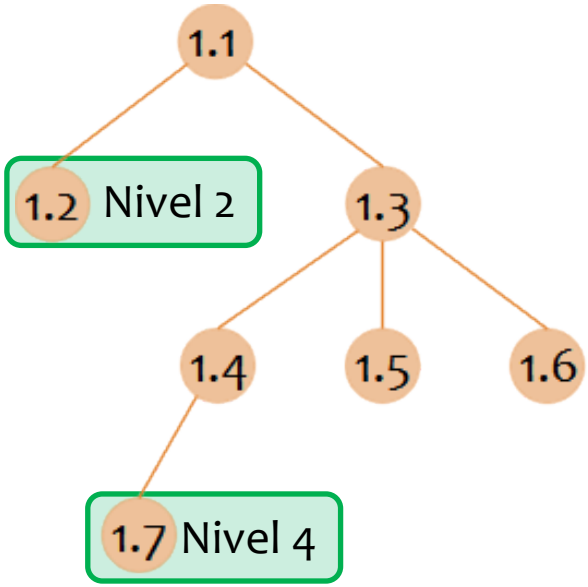


- **Conceptos:**
  - **Camino:** Secuencia de nodos donde cualquier par de nodos consecutivos son padre e hijo.
  - **Longitud del camino:** nº de nodos que forman el camino menos 1.
  - **Rama:** Cualquier camino desde el nodo raíz del árbol hasta un nodo hoja.

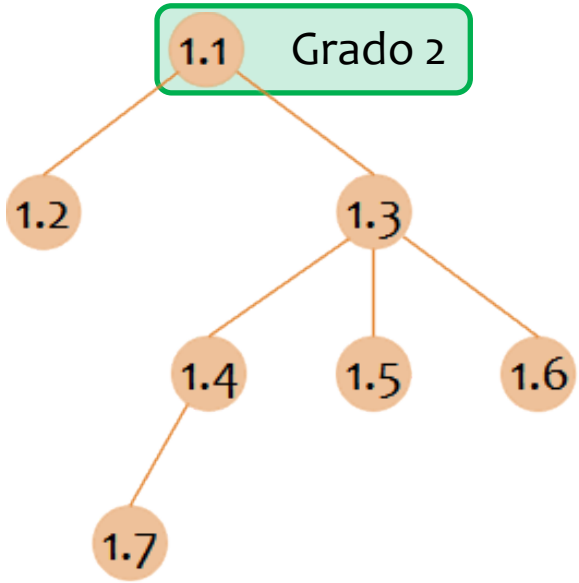
Rama=camino de longitud 3



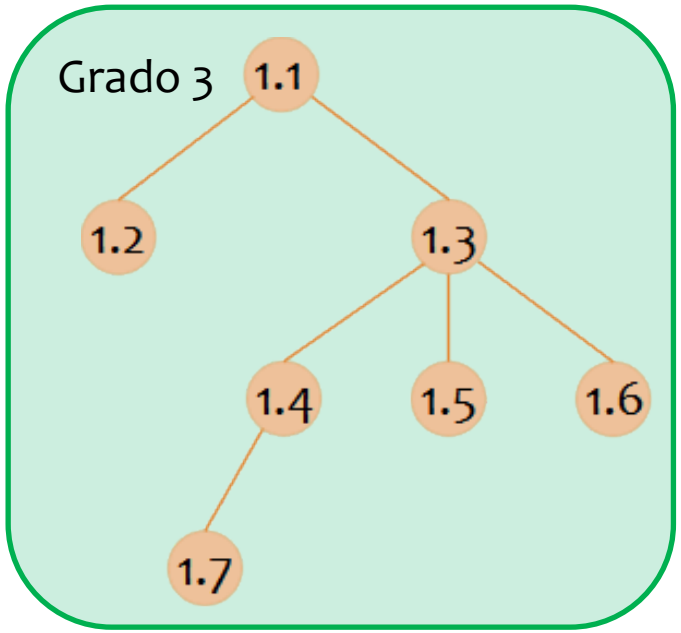
- **Conceptos:**
  - **Camino:** Secuencia de nodos donde cualquier par de nodos consecutivos son padre e hijo.
  - **Longitud del camino:** nº de nodos que forman el camino menos 1.
  - **Rama:** Cualquier camino desde el nodo raíz del árbol hasta un nodo hoja.
  - **Nivel (profundidad) de un nodo**: Nº de nodos que tiene el camino desde la raíz a dicho nodo.



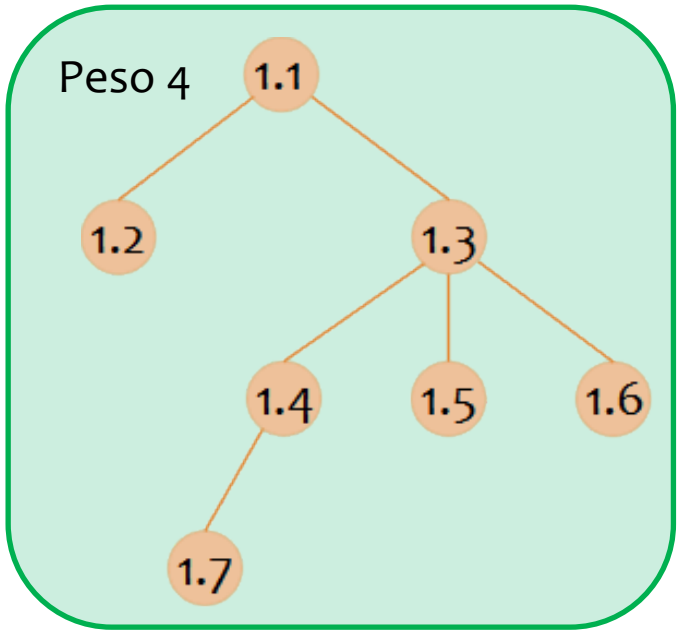
- **Conceptos:**
  - **Camino:** Secuencia de nodos donde cualquier par de nodos consecutivos son padre e hijo.
  - **Longitud del camino:** nº de nodos que forman el camino menos 1.
  - **Rama:** Cualquier camino desde el nodo raíz del árbol hasta un nodo hoja.
  - **Nivel (profundidad) de un nodo**: Nº de nodos que tiene el camino desde la raíz a dicho nodo.
  - **Grado:**
    - **De un nodo:** nº de descendientes directos de dicho nodo (nº de hijos).



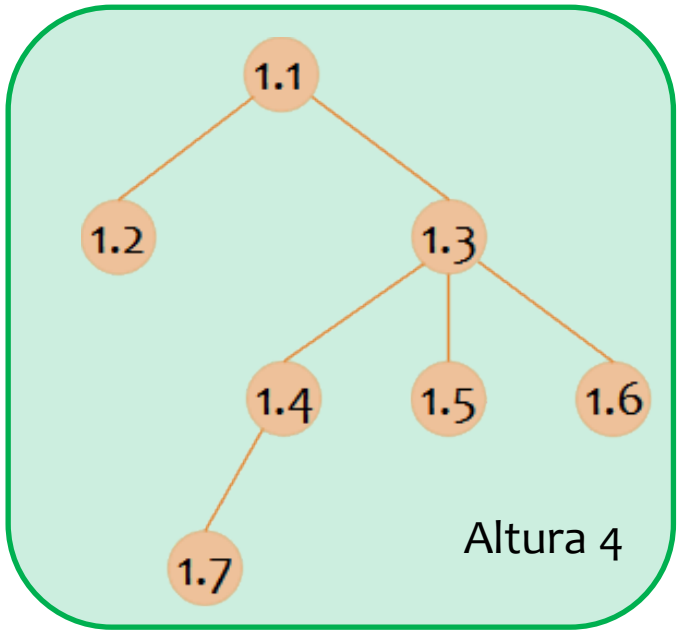
- **Conceptos:**
  - **Camino:** Secuencia de nodos donde cualquier par de nodos consecutivos son padre e hijo.
  - **Longitud del camino:** nº de nodos que forman el camino menos 1.
  - **Rama:** Cualquier camino desde el nodo raíz del árbol hasta un nodo hoja.
  - **Nivel (profundidad) de un nodo**: Nº de nodos que tiene el camino desde la raíz a dicho nodo.
  - **Grado:**
    - **De un nodo:** nº de descendientes directos de dicho nodo (nº de hijos).
    - **De un árbol:** máximo grado de sus nodos.



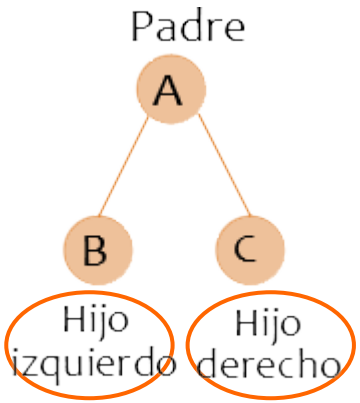
- **Conceptos:**
  - **Camino:** Secuencia de nodos donde cualquier par de nodos consecutivos son padre e hijo.
  - **Longitud del camino:** nº de nodos que forman el camino menos 1.
  - **Rama:** Cualquier camino desde el nodo raíz del árbol hasta un nodo hoja.
  - **Nivel (profundidad) de un nodo**: Nº de nodos que tiene el camino desde la raíz a dicho nodo.
  - **Grado:**
    - **De un nodo:** nº de descendientes directos de dicho nodo (nº de hijos).
    - **De un árbol:** máximo grado de sus nodos.
  - **Peso:** Nº de nodos terminales u hojas del árbol.

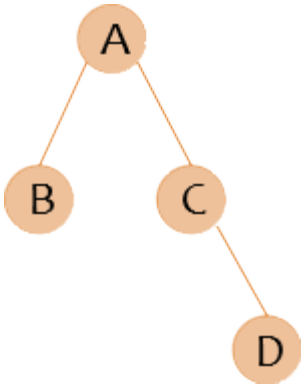


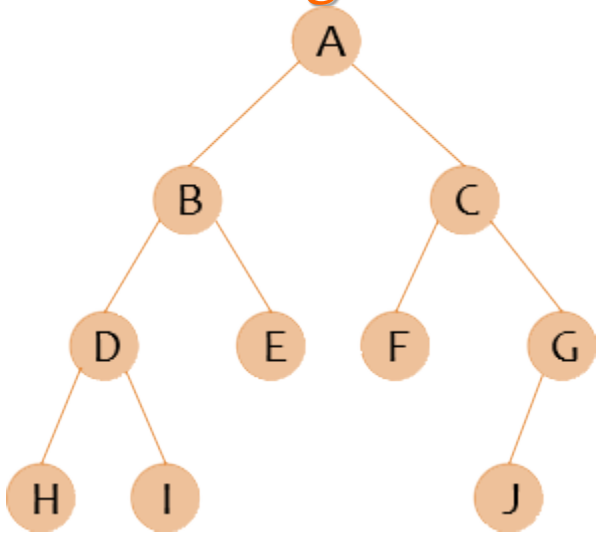
- **Conceptos:**
  - **Camino:** Secuencia de nodos donde cualquier par de nodos consecutivos son padre e hijo.
  - **Longitud del camino:** nº de nodos que forman el camino menos 1.
  - **Rama:** Cualquier camino desde el nodo raíz del árbol hasta un nodo hoja.
  - **Nivel (profundidad) de un nodo**: Nº de nodos que tiene el camino desde la raíz a dicho nodo.
  - **Grado:**
    - **De un nodo:** nº de descendientes directos de dicho nodo (nº de hijos).
    - **De un árbol:** máximo grado de sus nodos.
  - **Peso:** Nº de nodos terminales u hojas del árbol.
  - **Altura:** nivel más alto del árbol (nº de niveles del árbol). La altura es igual al nº de nodos que tiene la rama más larga del árbol.



- **Definición:**
  - Árbol donde cada nodo tiene como máximo grado 2





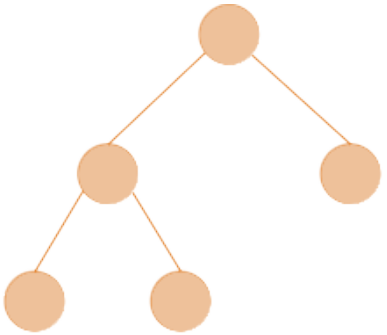


Definición recursiva: *Un árbol binario es un árbol vacío, o bien un nodo raíz con subárboles formados por árboles binarios a la derecha y a la izquierda.*

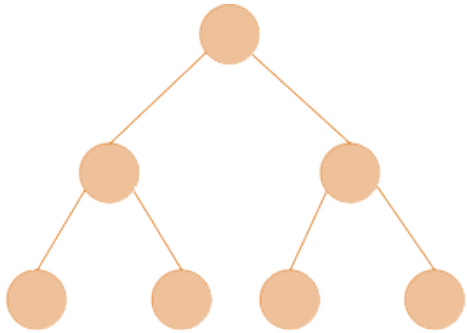
Ejemplos:

1. Representar la solución a un problema con dos posibles alternativas
2. Historia de un campeonato (clasificaciones, cuartos, semis, final, pero al revés…).
3. Expresiones algebraicas (con operadores dinámicos)

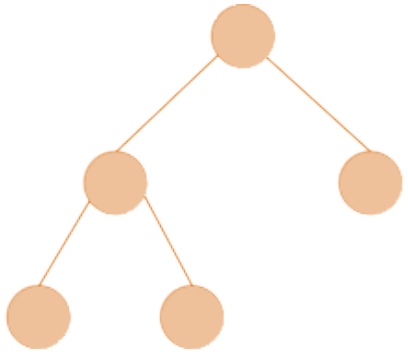
- **Conceptos:**
  - **Árbol binario equilibrado:** Cuando la diferencia de altura entre los subárboles de cualquier nodo es como máximo una unidad.
    - Si los subárboles de todos los nodos tienen la misma altura se dice que está perfectamente o totalmente equilibrado.
  - **Árbol binario lleno:** Un árbol binario de altura h es lleno si tiene todas sus hojas a nivel h y todos los nodos que están en un nivel menor que h tiene cada uno dos hijos.
    - Un árbol binario lleno es totalmente equilibrado.
  - **Árbol binario completo:** Un árbol binario de altura h es completo si está relleno a partir del nivel h-1, con el nivel h relleno de izquierda a derecha.
    - SI UN ÁRBOL BINARIO ES LLENO, ES NECESARIAMENTE COMPLETO.
    - UN ÁRBOL BINARIO COMPLETO ES EQUILIBRADO



Árbol binario equilibrado de altura 3



Árbol binario lleno de altura 3



Árbol binario completo de altura 3

2.1. Especificación del TAD Árbol Binario

ESPECIFICACIÓN= *definición del **tipo de dato e** y las **operaciones** que sirven para manipularlo o hacer preguntas sobre él, ya que trabajaremos con **TIPOS DE DATOS OPACOS** (la implementación está oculta al usuario).*

abin=TAD con operaciones **crear, es_vacio, izq, der, leer, insizq, insder, supizq, supder, modificar.**

DESCRIPCIÓN: Los valores del TAD **abin** son árboles binarios donde cada nodo contiene un dato del tipo **tipoelem**. Los árboles son **mutables**: **insizq, insder, supizq, supder** y **modificar** añaden, eliminan y modifican respectivamente elementos de un árbol.

OPERACIONES de creación/destrucción* del árbol:

`crear()` devuelve **`(abin)`**

- efecto: Devuelve el árbol binario vacío.

2.1. Especificación del TAD Árbol Binario

OPERACIONES de información, ya que no puedo acceder al contenido del tipo de datos por ser OPACO:

`es_vacio(A:abin)` devuelve **`(booleano)`**

- efecto: Devuelve **cierto** si **A** es el árbol vacío, y **falso** en caso contrario.

`izq/der(A:abin)` devuelve **`(abin)`**

- requisitos: El árbol **A** es no vacío.
- efecto: Devuelve la posición del nodo hijo izquierdo/derecho del subárbol **A**. Si el nodo **A** no tiene hijo izquierdo/derecho, devuelve **nulo**.

`leer(A:abin)` devuelve **`(tipoelem)`**

- requisitos: El árbol **A** es no vacío.
- efecto: Devuelve el contenido del nodo **A** en el árbol.

2.1. Especificación del TAD Árbol Binario

OPERACIONES de modificación, ya que no puedo acceder al contenido del tipo de datos por ser OPACO:

`insizq/insder(A:abin; E:tipoelem)`

- requisitos: El árbol **A** es no vacío y **`esVacio(izq(A)/der(A))=cierto`**.
- modifica: **A**.
- efecto: Añade un nodo, con contenido **E**, como hijo izquierdo/derecho del nodo **A**.

`supizq/supder(A:abin)`

- requisitos: El subárbol **A** es no vacío
- modifica: **A**.
- efecto: Suprime el hijo izquierdo/derecho del subárbol **A** y todos sus descendientes.

`modificar(A:abin; E:tipoelem)`

- requisitos: El árbol **A** es no vacío.
- modifica: **A**.
- efecto: Modifica el contenido del nodo **A**, cambiándolo por el nuevo contenido **E**.

2.2. Implementación del TAD Árbol Binario

- **Implementación del TAD:**
  - **Selección** de un lenguaje de programación para pasar de la especificación a la implementación: lenguaje C
  - Implementación de un TAD en lenguaje C:
    - **Interfaz de usuario**: `abin.h`
    - **Implementación**: `abin.c` (oculta al usuario)

OPCIÓN A: menos repetición, menos opacidad

abin.h

- Se define `abin` como un puntero a `struct` (se pierde opacidad):
  ```
  typedef struct celda *abin;
  ```
- Se define `tipoelem`, el tipo de dato de cada nodo del árbol binario:
  ```
  typedef int tipoelem;
  ```

abin.c

- Se incluye `abin.h`, lo que evita la repetición de la definición de `tipoelem`:
  ```
  #include "abin.h"
  ```

OPCIÓN B: más repetición, pero más opacidad

abin.h

- Se define `abin` como un puntero a `void` (se gana opacidad):
  ```
  typedef void *abin;
  ```
- Se define `tipoelem`, el tipo de dato de cada nodo del árbol binario:
  ```
  typedef int tipoelem;
  ```

abin.c

- Se repite la definición de `tipoelem`, el tipo de dato de cada nodo del árbol:
  ```
  typedef int tipoelem;
  ```

2.2. Implementación del TAD Árbol Binario

- **abin.h:**

abin.h

```
1   #ifndef ABIN_H
2   #define ABIN_H
3
4   //contenido de cada nodo del árbol: números enteros
5   typedef int tipoelem;
6   //definición del tipo opaco
7   typedef void *abin;
8   ////////////////////////////// FUNCIONES
9   void crear(abin *A);
10  void destruir(abin *A);
11  unsigned es_vacio(abin A);
12  abin izq(abin A);
13  abin der(abin A);
14  void leer(abin A, tipoelem *E);
15  void insizq(abin *A, tipoelem E);
16  void insder(abin *A, tipoelem E);
17  void supizq(abin *A);
18  void supder(abin *A);
19  void modificar(abin A, tipoelem E);
20
21  #endif  // ABIN_H
```

GUARDAS DEL COMPILADOR

Evitan la redefinición de tipos y funciones

2.2. Implementación del TAD Árbol Binario

- **abin.c:**

abin.c

```c
#include <stdio.h>
#include <stdlib.h>

//////////////////////REPETIMOS LA DEFINICIÓN DE LOS TIPOS DE DATOS
//abin es un puntero a struct celda
typedef struct celda *abin;
//contenido de cada nodo del árbol
//es un árbol de números enteros
typedef int tipoelem;

//////////////////////ESTRUCTURAS DE DATOS
struct celda{
    tipoelem info;
    struct celda *izq, *der;
};

//////////////////////////FUNCIONES
```

Detalles de la implementación (ocultos al usuario)

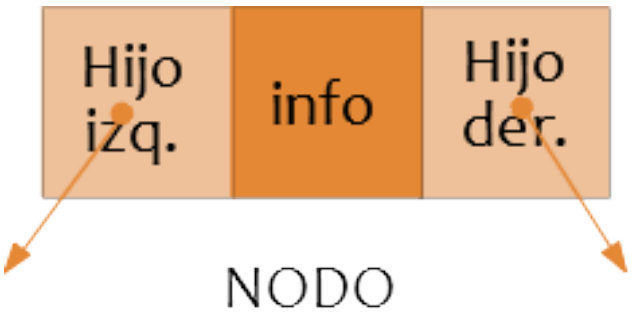
SE REPITE LA DEFINICIÓN HECHA EN abin.h

Detalles de la implementación (ocultos al usuario)

1. Memoria dinámica (punteros)
2. Memoria estática (vectores) (simulación de memoria dinámica, análoga al caso de listas)

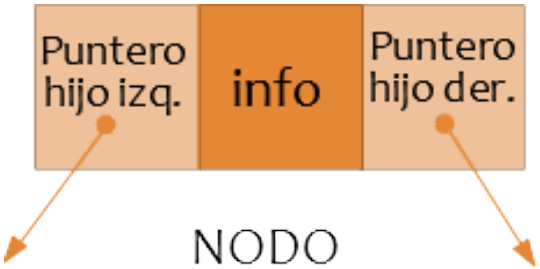
En cualquier caso, en cada nodo del árbol:

- **Información** con los datos necesarios.
- **Enlace hacia el hijo izquierdo** o raíz del subárbol izquierdo.
- **Enlace hacia el hijo derecho** o raíz del subárbol derecho.



2.2. Implementación del TAD Árbol Binario

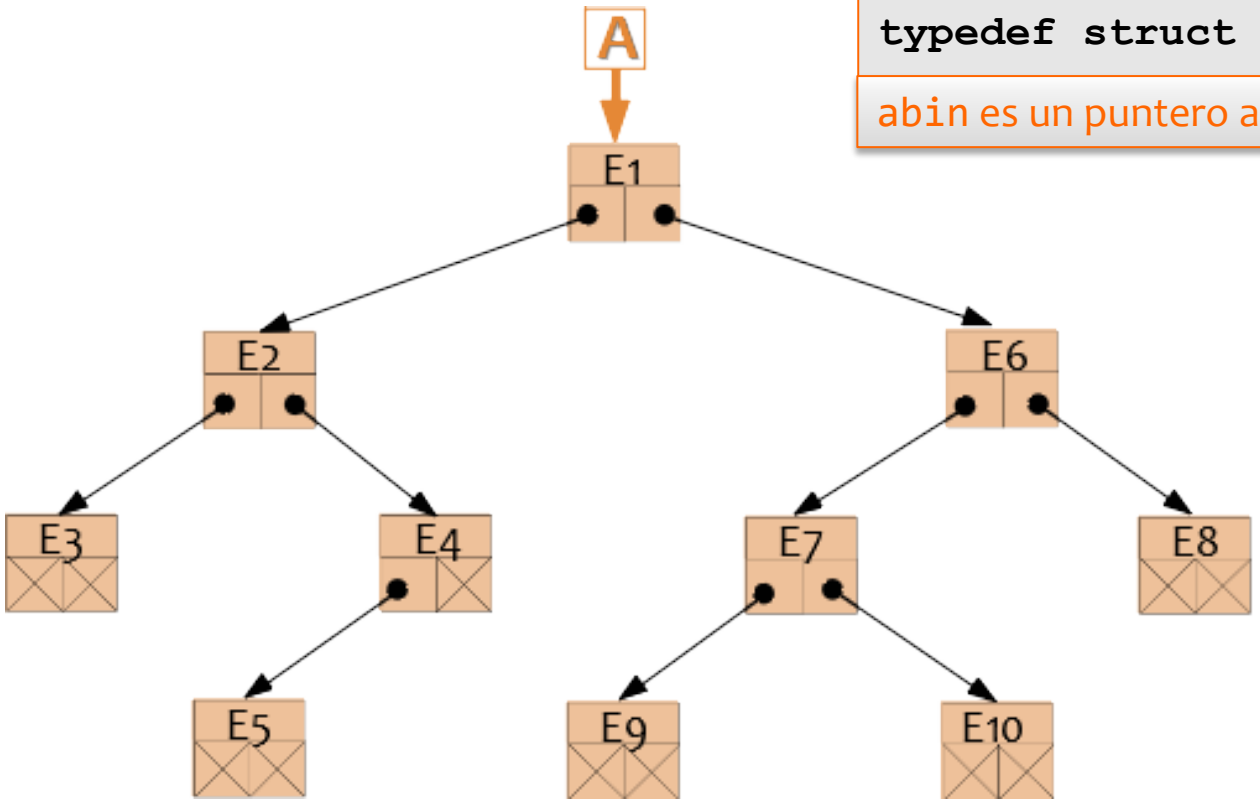
- Realización mediante memoria dinámica




```
struct celda{
     tipoelem info;
     struct celda *izq, *der;
};

typedef struct celda *abin;
```

abin es un puntero a una celda del árbol



- Realización mediante memoria dinámica
  - Creación/Destrucción de abin

```
void crear(abin *A){
    *A=NULL;
}
```

```
void supizq(abin *A);
void supder(abin *A);
unsigned esVacio(abin A);

void destruir(abin *A){
    abin aux;
    aux=*A;
    if(!esVacio(aux)){
      supizq(&aux);
      supder(&aux);
      free(aux);
      *A=NULL;
    }
}
```

- Realización mediante memoria dinámica
  - Funciones de información

```
unsigned es_vacio(abin A){
    return (A==NULL);
}
```

```
abin izq(abin A){
    return A->izq;
}
```

```
abin der(abin A){
    return A->der;
}
```

```
void leer(abin A, tipoelem *E){
    *E=A->info;
}
```

- Realización mediante memoria dinámica
  - Funciones de modificación

```
void insizq/insder(abin *A, tipoelem E){
   abin aux;
   aux=(abin)malloc(sizeof(struct celda));
   aux->info=E;
   aux->izq=NULL;
   aux->der=NULL;
   if(esVacio(*A))
      *A=aux;
   else
      (*A)->izq=aux;
      //(*A)->der=aux;
}
```

```
void supizq/supder(abin *A){
     abin aux;
     aux=izq(*A); //aux=der(*A);

     if(!esVacio(aux)){
       supizq(&aux);
       supder(&aux);
       (*A)->izq=NULL; //(*A)->der=NULL;
       free(aux);
     }
}
```

```
void modificar(abin A, tipoelem E){
     A->info=E;
}
```

2.3. Recorrido de un árbol binario

- Tipos de recorrido:
  - **Recorrido en anchura:** Por niveles y, dentro de cada nivel, de izquierda a derecha
  - **Recorrido en profundidad**
    - Inorden: Izquierda – Raíz – Derecha

      IRD, orden normal de expresiones, orden simétrico u orden central

      Recorro subárbol izquierdo - Visito nodo raíz - Recorro subárbol derecho
    - Preorden: Raíz – Izquierda – Derecha

      RID, notación prefija o notación polaca

      Visito nodo raíz - Recorro subárbol izquierdo-Recorro subárbol derecho
    - Postorden: Izquierda – Derecha – Raíz

      IDR o notación postfija de expresiones o notación polaca inversa

      Recorro subárbol izquierdo - Recorro subárbol derecho - Visito nodo raíz

- Simulación recorrido en profundidad:

Hay muchas páginas en las que se realizan estos recorridos:

- Binary Tree Visualizer (J Melezinek, TFG Faculty of Information Technology, Czech Technical University in Prague)

  http://btv.melezinek.cz/binary-search-tree.html

- Data Structure Visualizations (D Galles, Computer Science, University of San Francisco) , sólo se realiza el recorrido inorden:

  https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

2.3. Recorrido de un árbol binario

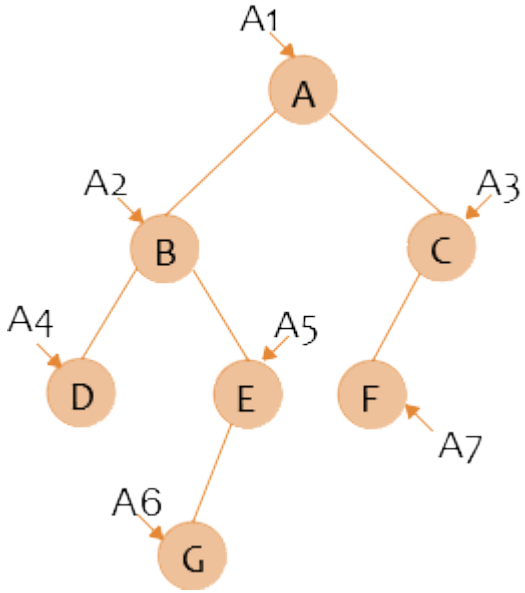
- Recorrido inorden en profundidad recursivo:

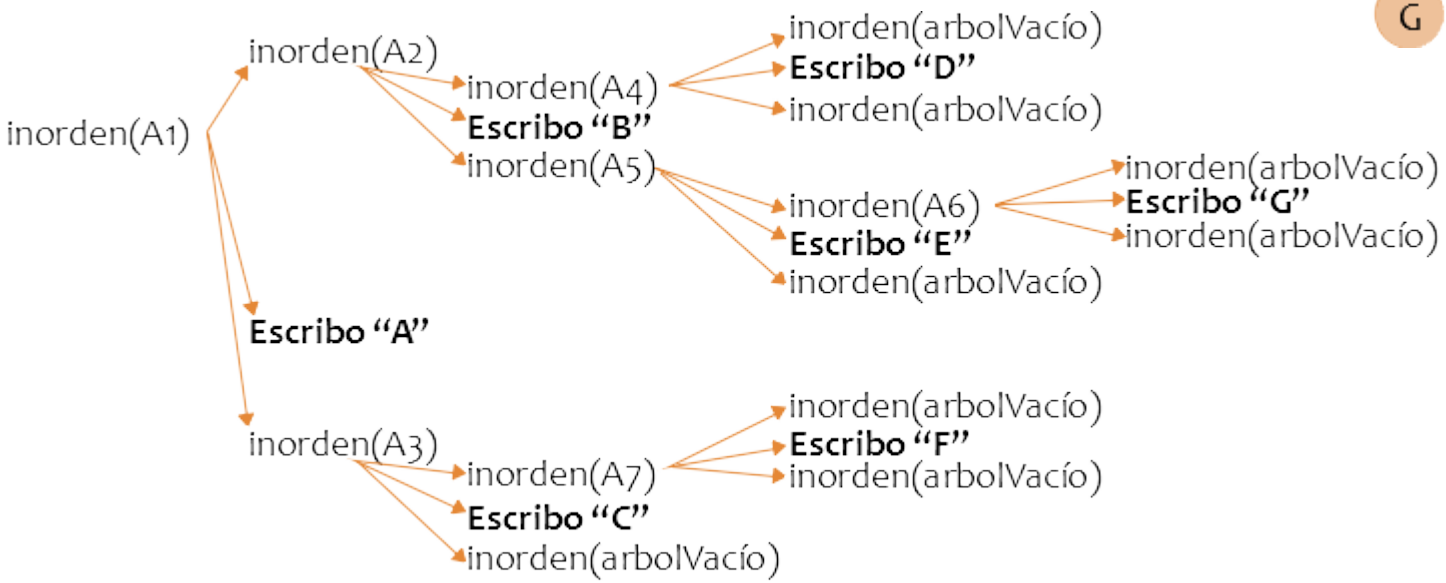
```
void inorden(abin A){
     tipoelem E;
     if(!esVacio(A)){
       inorden(izq(A));
       leer(A,&E);
       printf("%d\n",E);
       inorden(der(A));
     }
}
```

Izquierda

Raíz

Derecha





D

D-B

D-B-G

D-B-G-E

D-B-G-E-A

D-B-G-E-A-F

D-B-G-E-A-F-C

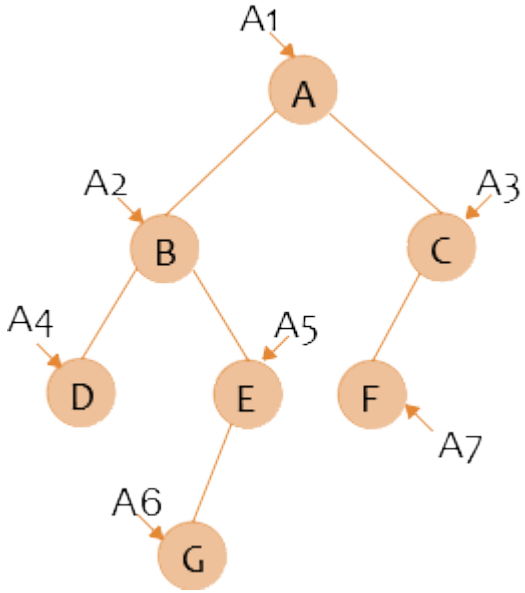
- Recorrido preorden en profundidad recursivo:

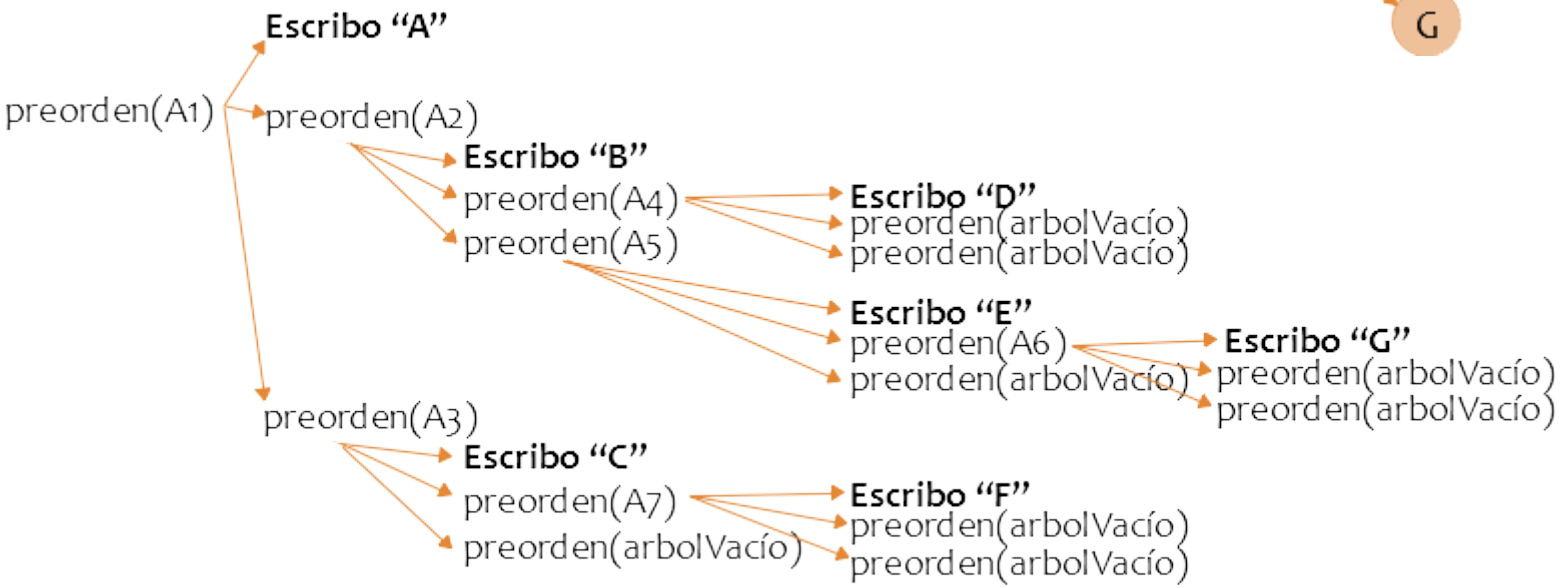
```
void preorden(abin A){
     tipoelem E;
     if(!esVacio(A)){
       leer(A,&E);
       printf("%d\n",E);
       preorden(izq(A));
       preorden(der(A));
     }
}
```

Raíz

Izquierda

Derecha





A

A-B

A-B-D

A-B-D-E

A-B-D-E-G

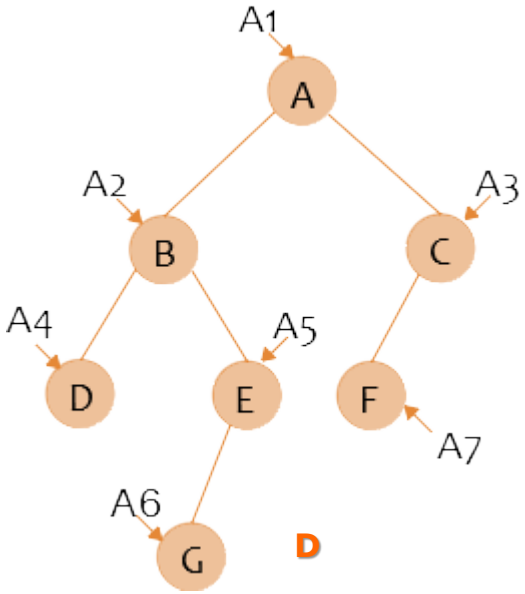
A-B-D-E-G-C

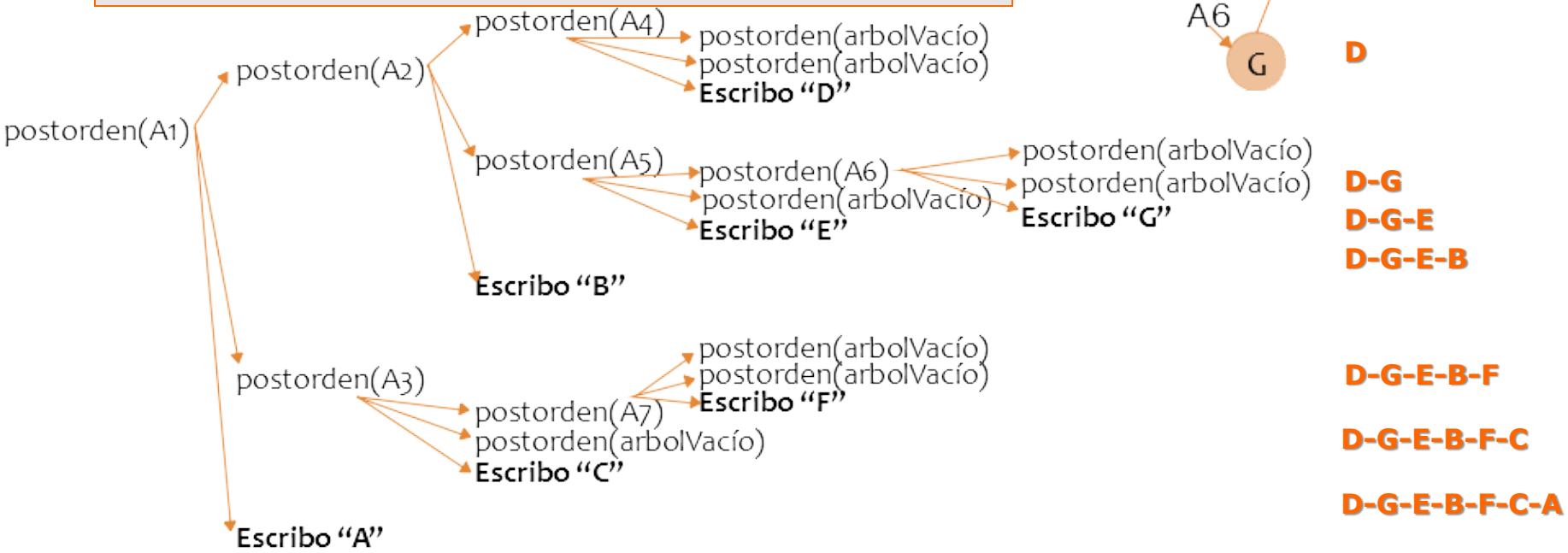
A-B-D-E-G-C-F

2.3. Recorrido de un árbol binario

- Recorrido postorden en profundidad recursivo:

```
void postorden(abin A){
     tipoelem E;
     if(!esVacio(A)){
       postorden(izq(A)); Izquierda
       postorden(der(A)); Derecha
       leer(A,&E);
       printf("%d\n",E);  Raíz
     }
}
```





2.3. Recorrido de un árbol binario

- **Recorridos no recursivos:** usan estructuras auxiliares para almacenar punteros a los nodos del árbol.
  - **Recorrido en profundidad inorden no recursivo:**

    Usa una **PILA** para almacenar los punteros a los nodos del árbol.

    1. Guardar en la pila punteros a hijos izquierdos comenzando por raíz
    2. Si pila no vacía: Desapilar el tope, imprimir y hacer que nueva raíz=derecha del tope.
    3. Repetir pasos 1-2 hasta que pila vacía y árbol recorrido.

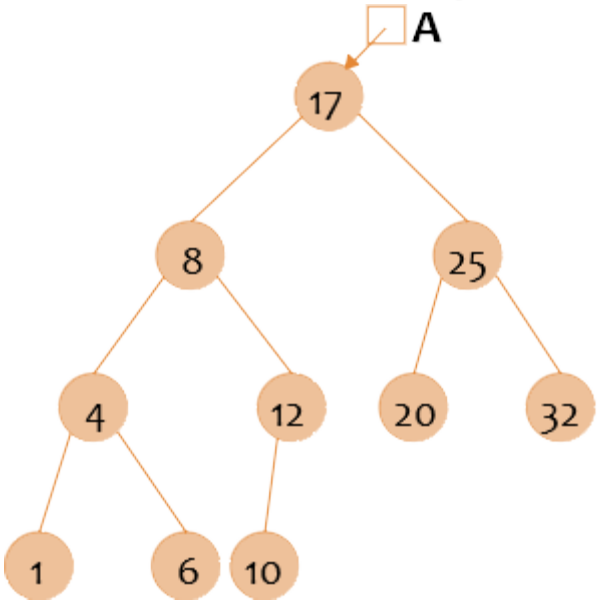
  - **Recorrido en anchura no recursivo:**

    Usa una **COLA** auxiliar para visitar los nodos por niveles.

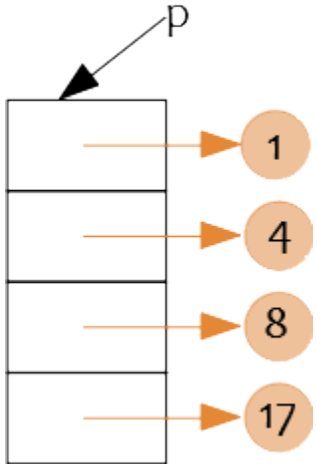
    1. Tomar puntero raíz y ponerlo en la cola.
    2. A continuación, quitar primer elemento de la cola, mostrar el contenido de dicho nodo y almacenar los punteros correspondientes a sus hijos izquierdo y derecho.
    3. Repetir paso 2 hasta que cola vacía y árbol recorrido.

2.3. Recorrido de un árbol binario

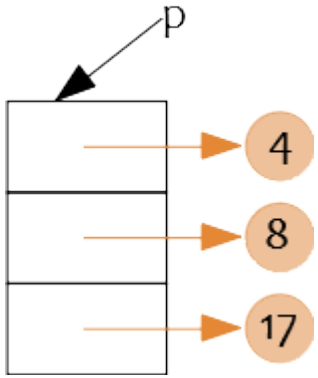
- Recorrido en profundidad inorden no recursivo: usa una pila



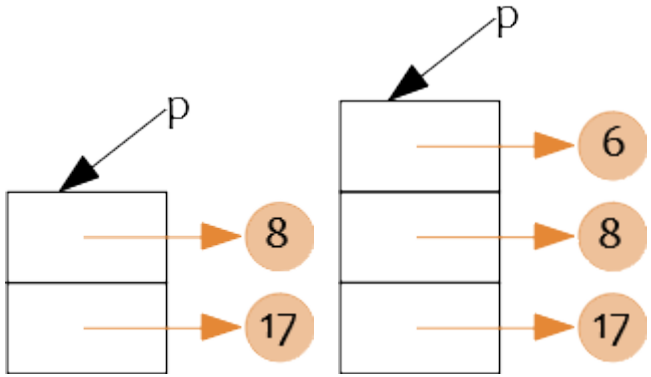
PASO 1: Se van colocando en la pila p punteros a la raíz y los sucesivos hijos izquierdos de cada nodo:
17-8-4-1



PASO 2: Recupera de la pila y escribe 1.
Raíz=derecha (1)=(NULL)
PASO 1: raíz= NULL
PASO 2: recupera de la pila y escribe 4.
Raíz=derecha(4)→(6).
PASO 1: Pone en la pila el puntero al 6 y toda su rama izquierda (vacía).



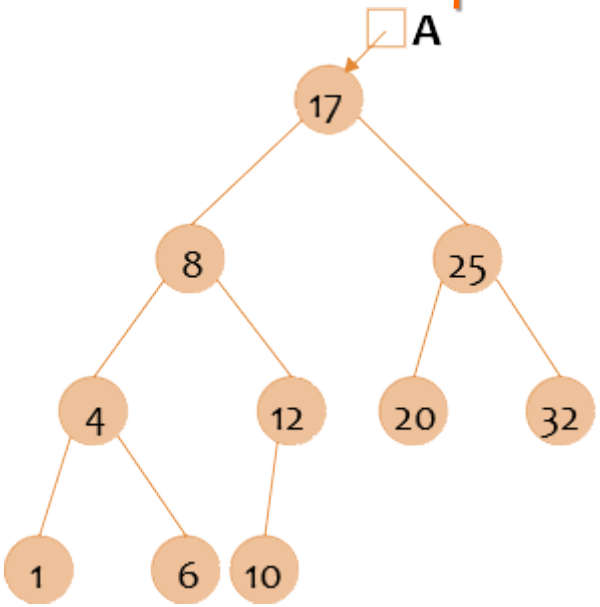
1



1 - 4

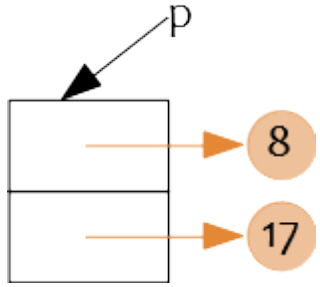
2.3. Recorrido de un árbol binario

- Recorrido en profundidad inorden no recursivo: usa una pila

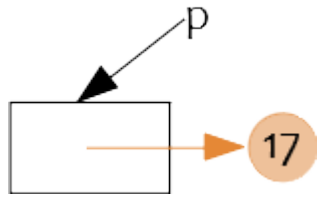


PASO 1: Pone en la pila el puntero al 12 y toda su rama izquierda (puntero al 10).
PASO 2: Recupera de la pila y escribe **10**. Raíz=derecha(10)=NULL
PASO 1: raíz=NULL
PASO 2: recupera de la pila y escribe **12**. Raíz=derecha(12)=NULL

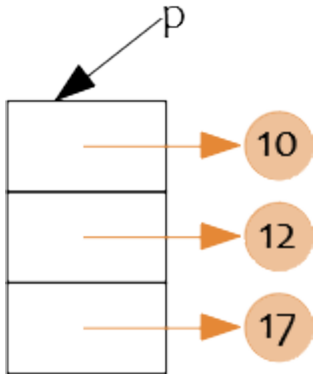
PASO 2: Recupera de la pila y escribe **6**. Raíz=derecha(6)=(NULL)
PASO 1: raíz= NULL
PASO 2: recupera de la pila y escribe **8**. Raíz=derecha(8)→(12).

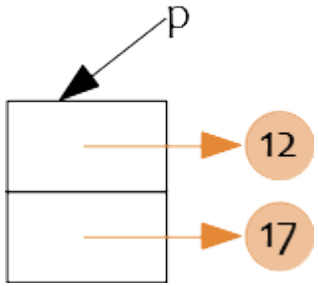


1 – 4 – 6

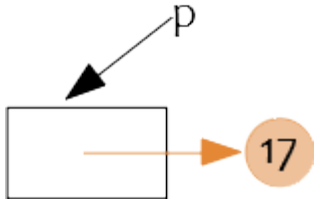


1 – 4 – 6 – 8





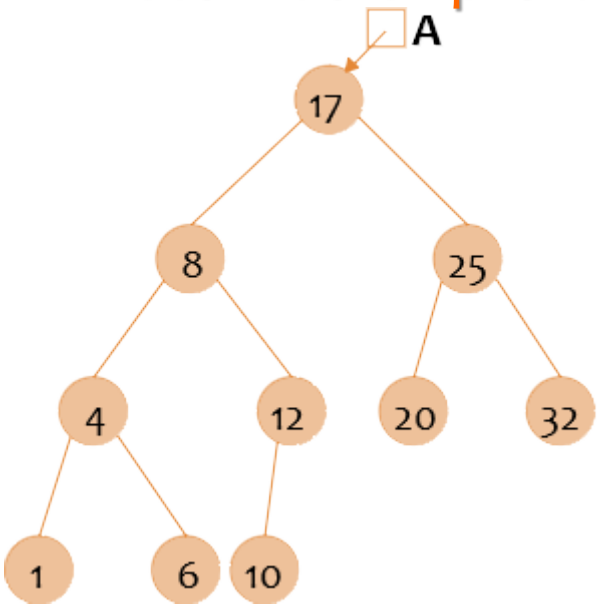
1 – 4 – 6 – 8 – 10



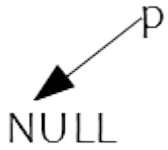
1 – 4 – 6 – 8 – 10 - 12

2.3. Recorrido de un árbol binario

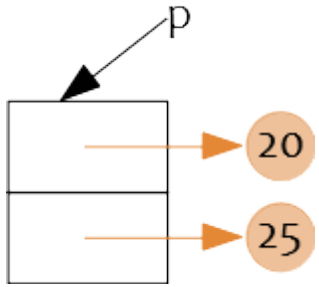
- Recorrido en profundidad inorden no recursivo: usa una pila

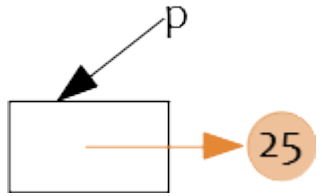


PASO 2: Recupera de la pila y escribe **17**. Raíz=derecha (17)=25
PASO 1: Pone en la pila el puntero al 25 y toda su rama izquierda (puntero al 20).
PASO 2: recupera de la pila y escribe **20**.
Raíz=derecha(20)→(NULL)



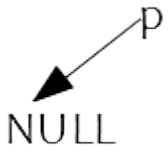
1-4-6-8-10-12-17

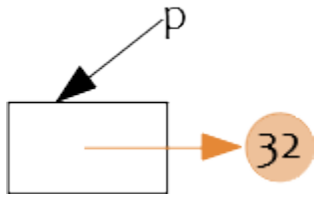


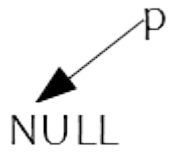


1-4-6-8-10-12-17-20

PASO 1: raíz=NULL
PASO 2: Recupera de la pila y escribe **25**. Raíz=derecha(25)=32
PASO 1: Pone en la pila el puntero al 32 y toda su rama izquierda (NULL).
PASO 2: recupera de la pila y escribe **32**. Raíz=derecha(32)=NULL







1-4-6-8-10-12-17-20-25

1-4-6-8-10-12-17-20-25-32

PILA VACÍA. TODOS LOS NODOS DEL ÁRBOL VISITADOS

2.3. Recorrido de un árbol binario

- Recorrido en profundidad inorden no recursivo: usa una pila

```
void inordenNR(abin A){
  tipoelem E;
  abin aux;
  pila p;
  aux=A;
  crearPila(&p);
  do{
    while(!esVacio(aux)){
      push(&p,aux);
      aux=izq(aux);
    }
    if(!esVaciaPila(p)){
      aux=tope(p);
      pop(&p);
      leer(aux,&E); printf("%d\n",E);
      aux=der(aux);
    }
  }while(!esVaciaPila(p) || !esVacio(aux));
  destruirPila(&p);
}
```

Mientras aux NO vacío, mete en pila aux y sus hijos izquierdos

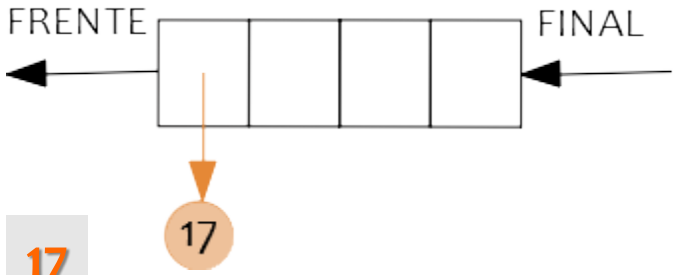
Desapila tope, lo imprime y hace aux=derecha(tope)

2.3. Recorrido de un árbol binario

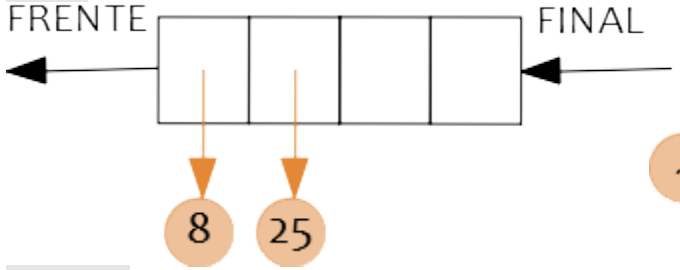
- Recorrido en anchura no recursivo: usa una cola

PASO 1: Creo la cola e inserto puntero a raíz (17) en la cola
PASO 2: Saco primer elemento (17), lo imprimo e inserto sus hijos izquierdo y derecho.
PASO 2: Saco primer elemento (8), lo imprimo e inserto sus hijos izquierdo y derecho.
…

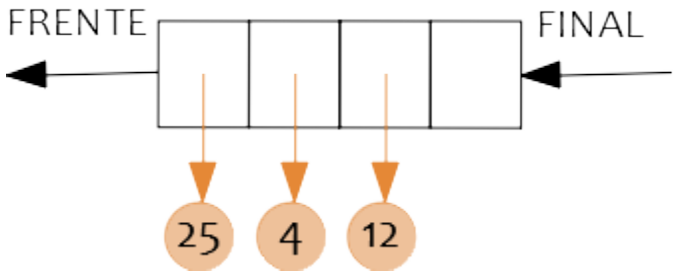
…



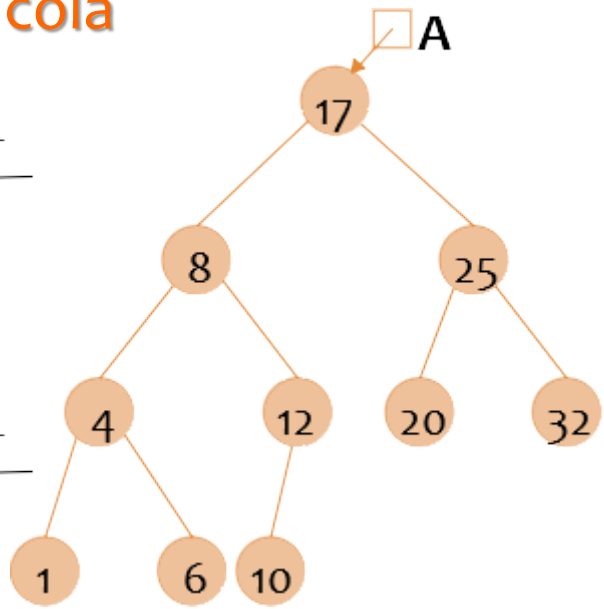
17



17-8



17-8-25-4-12-20-32-1-6-10



……

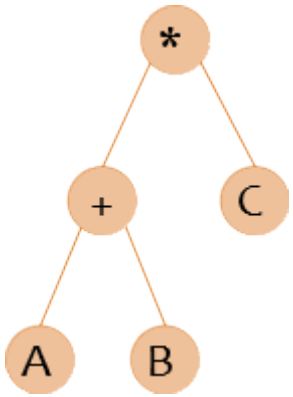
- Recorrido en anchura no recursivo: usa una cola

```
void anchura(abin A){
    tipoelem E;
    abin aux;
    cola c;
    crearCola(&c);
    if(!esVacio(A)) //Insertamos nodo raíz, primer nivel
       insertar(&c,A);

    while(!esVaciaCola(c)){
       primero(c,&aux); //Se lee elemento de la cola
       suprimir(&c); //Se elimina elemento de la cola
       leer(aux,&E); printf("%d\n",E);
       if(!esVacio(izq(aux)))
             insertar(&c,izq(aux));
       if(!esVacio(der(aux)))
             insertar(&c,der(aux));
    }
    destruirCola(&c);
}
```

2.4. Árboles de expresión

- Los árboles binarios se utilizan para representar expresiones en memoria, esencialmente en **compiladores de lenguajes de programación**.
- Si suponemos operadores *binarios*:
  - Los paréntesis no se almacenan pero están implicados en la forma del árbol
  - Todos los operandos letras están almacenados en hojas.
  - La raíz de cada subárbol es un operador



Expresión: (A + B) * C

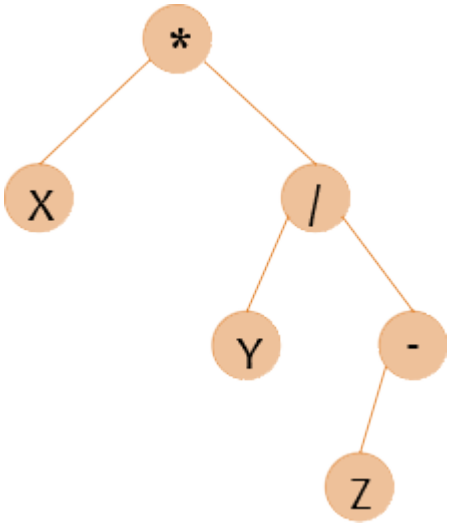
Inorden (IRD): A + B * C

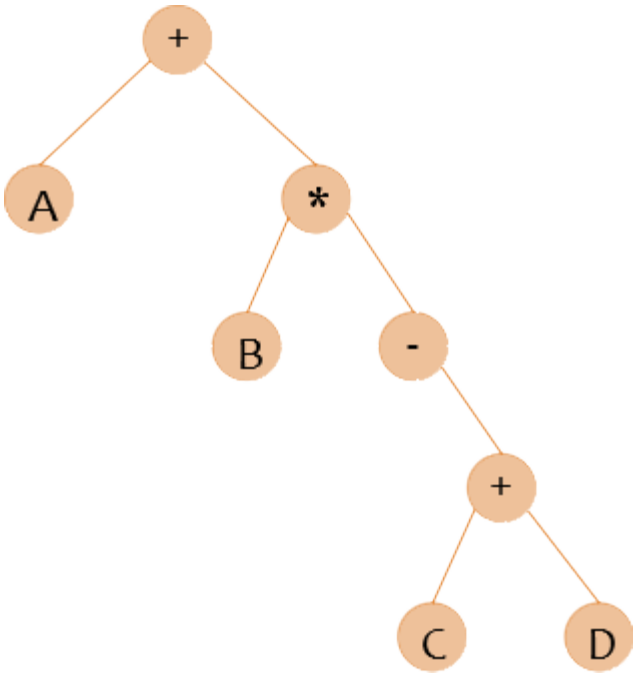
Preorden (RID): * + A B C (notación prefija o polaca)

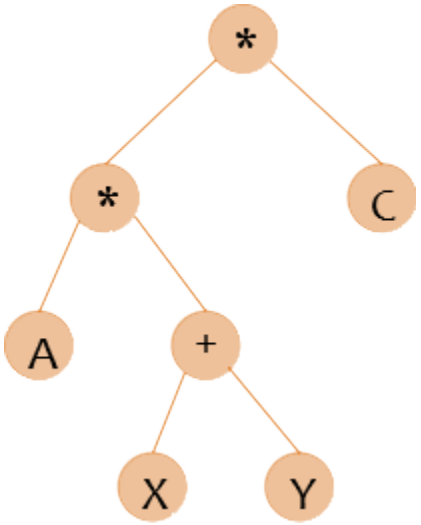
Postorden (IDR): A B + C * (notación postfija o polaca inversa)

2.4. Árboles de expresión

- Ejemplos:







Expresión: X*(Y/-Z)
Inorden (IRD): X*Y/-Z
Preorden (RID): *X/Y-Z
Postorden (IDR): XYZ-/*

Expresión: A+[B*-C+D)]
Inorden (IRD): A+B*-C+D
Preorden (RID): +A*B-+CD
Postorden (IDR): ABCD+-*+

Expresión [A*(X+Y)]*C
Inorden (IRD): A*X+Y*C
Preorden (RID): **A+XYC
Postorden (IDR): AXY+*C*

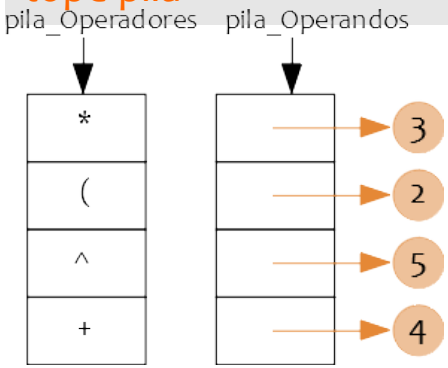
- Construcción a partir de una expresión en notación convencional (infija):
  - Estructuras de datos auxiliares:
    - **Pila** para almacenar **punteros a los nodos de un árbol** (pila de abin)
    - **Pila** para retener los **operadores** temporalmente hasta que llegue el momento de incorporarlos al árbol (**pila de caracteres**).
  - Pasos a seguir:
    - Cuando se lee un operando se crea un árbol de un nodo y se mete el puntero a él en la pila de punteros (pila de abin).
    - Cuando se lee un operador:
      - Si su *prioridad es **menor o igual** que la del **tope** de la pila de operadores, se* desapilan sucesivamente los operadores de la pila que cumplen la condición.
        - Al sacar un operador de su pila, hay que extraer dos punteros de la pila de punteros. Con estos tres elementos se forma un nuevo árbol. El puntero a este nuevo árbol se coloca en la pila de punteros.
        - **Excepción:** El paréntesis izquierdo sólo se saca cuando aparece un paréntesis derecho (que no se almacena en la pila).
      - Se introduce el operador en la pila.
    - El proceso termina cuando se acaba la entrada y, en caso de que no esté vacía, se vacíe la pila de operadores.

2.4. Árboles de expresión

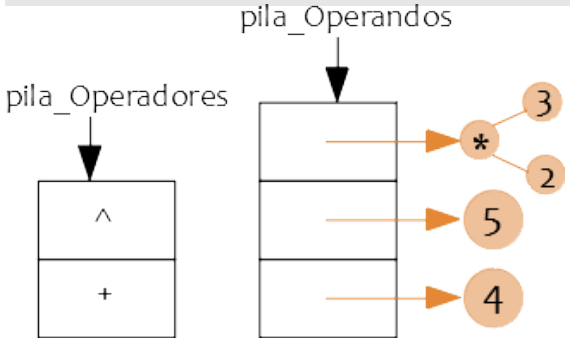
- Construcción a partir de una expresión en notación convencional (infija):

4 + 5 ^(2 * 3) + 8

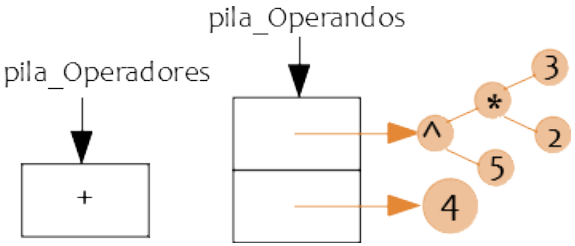
1º: apilo operadores y operandos mientras prioridad del operador que leo sea <= prioridad operador tope pila

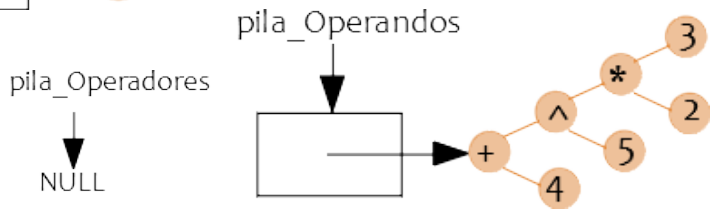


2º: El) saca los operadores de la pila hasta el (

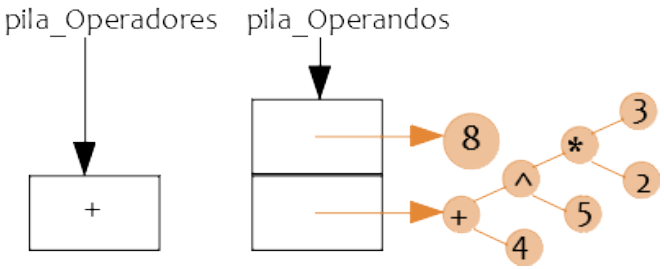


3º: Leo '+'.
Su prioridad <= prioridad(tope) => desapilo ^
Su prioridad <= prioridad(tope) => desapilo +





4º: Apilo +
Leo el 8 y lo apilo



5º: Vacío la pila de operadores