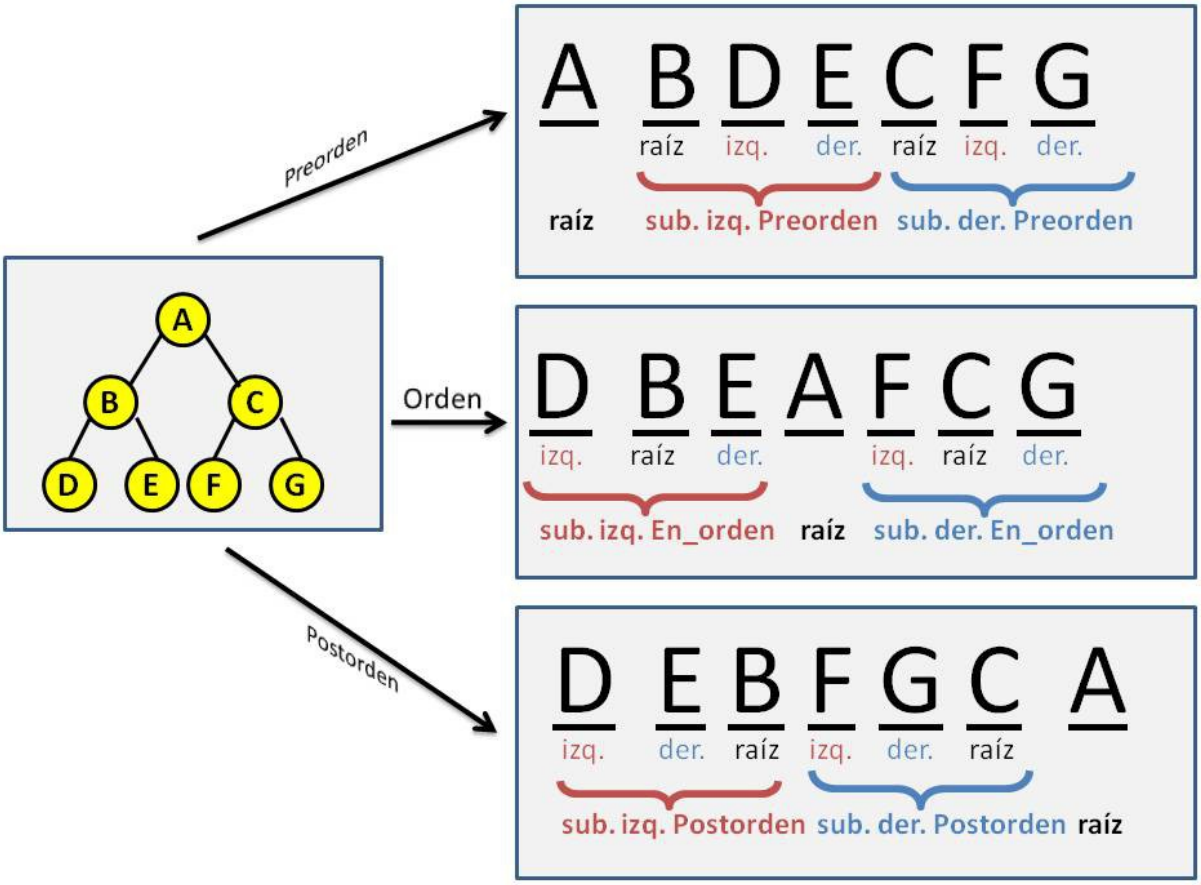
Prácticas de Árboles Binarios

1. Recorridos en Árboles Binarios

Utilizando la representación de árboles binarios mediante punteros implementar los **recorridos en profundidad y en amplitud.** El recorrido en amplitud debe utilizar el TAD Cola implementado en la práctica anterior, ajustándolo de forma que los elementos de la cola sean punteros a nodos de árboles binarios.

Ayuda para la realización de la práctica

Para comprobar el correcto funcionamiento de las funciones implementadas en la práctica es necesario generar un árbol binario al cual aplicarle las diferentes operaciones. Para ello puede utilizarse el fichero **ejercicio1.c** que construye el árbol ejemplo de las transparencias de recorridos y que se muestra en la siguientes figura.



2. TAD Árbol Binario

A continuación se definen una serie de operaciones sobre árboles binarios. Deben implementarse todas ellas mediante algoritmos **recursivos** ajustándose a los tipos y prototipos que se adjuntan en el fichero cabecera (arbol.h).

altura(a): calcula y devuelve la altura de un árbol binario con raíz *a*, es decir, el número de aristas en el camino más largo desde el nodo raíz hasta una hoja, teniendo en cuenta que los nodos hoja tienen altura 0.

numNodos(a): calcula y devuelve el número de nodos del árbol binario con raíz *a*.

anula(a): convierte el árbol *a* en un árbol vacío (si inicialmente no lo era), liberando la memoria que ocupan sus nodos.

sustituye(a,x,y): busca en el árbol *a* los nodos que contienen el valor ***x*** en el campo de información, sustituye este valor por ***y*** **y** devuelve el número de sustituciones realizadas.

numNodosHoja(a): calcula y devuelve el número de nodos hoja del árbol binario *a*.

numNodosInternos(a): calcula y devuelve el número de nodos internos del árbol binario *a*.

numHijoÚnico(a): calcula y devuelve el número de nodos del árbol *a* que tienen un sólo hijo.

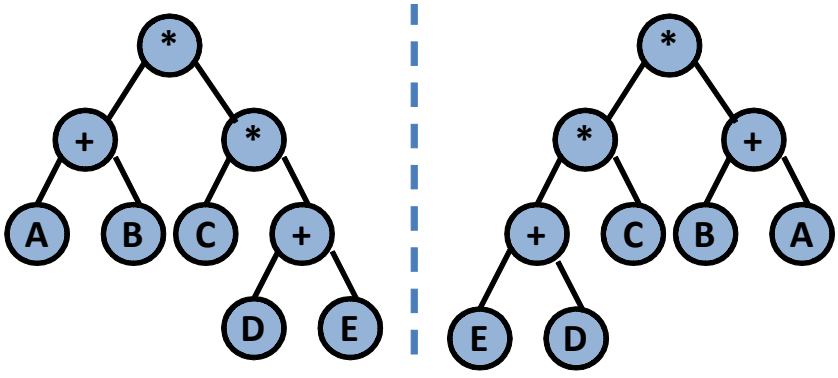
buscarMax(a)/buscarMin(a): busca y devuelve el nodo del árbol binario *a* que contiene el valor máximo/mínimo en su campo de información.

similares(a_1,a_2): comprueba si los árboles a_1 y a_2 son similares devolviendo verdadero si lo son y falso en caso contrario. Dos árboles son similares si tienen la misma estructura.

equilvalentes(a_1,a_2): comprueba si los árboles a_1 y a_2 son equivalentes devolviendo verdadero si lo son y falso en caso contrario. Dos árboles son equivalentes si tienen la misma estructura y el mismo contenido en sus nodos.

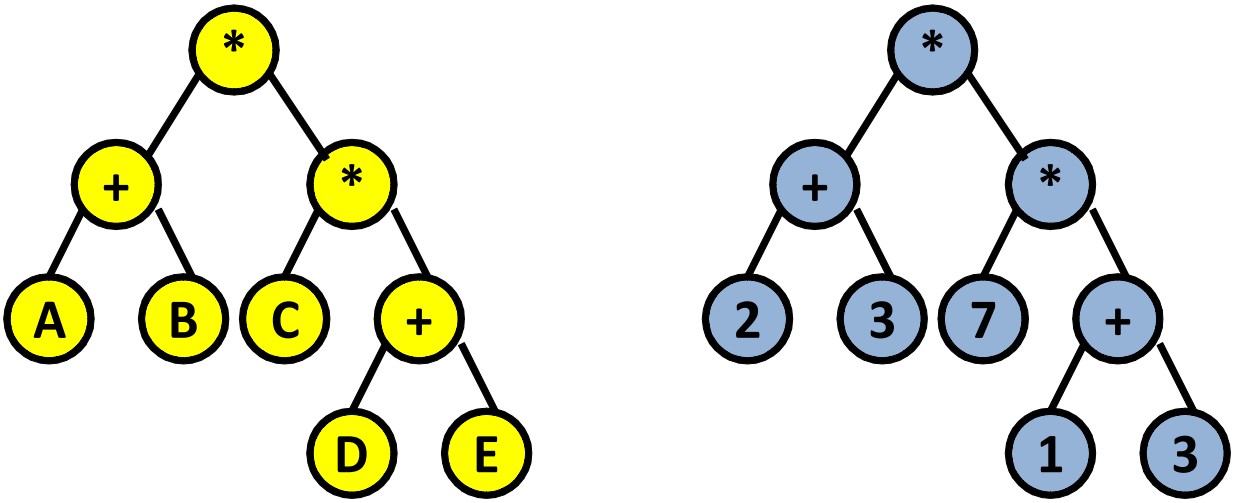
Nota: en la figura del enunciado 3 se observan dos árboles similares pero no equivalentes.

especular(a): crea un nuevo árbol binario que es la imagen especular del árbol *a*, es decir, todos los subárboles derechos se convierten en subárboles izquierdos y viceversa. En la siguiente figura se muestra un árbol a la izquierda y su imagen especular a la derecha. Estos dos árboles además no son ni similares ni equivalentes.



3. **TAD árbol algebraico**

Una de las aplicaciones de los árboles binarios es la representación de expresiones aritméticas en las cuales los nodos internos representan operadores y los nodos hoja operandos. Implementar el algoritmo que permite construir un árbol algebraico a partir de de una expresión aritmética en notación postfija correcta (algoritmo transparencia 15 tema 1).
Ejemplo: para la expresión postfija AB+CDE+** el algoritmo debe devolver el árbol de la izquierda de la siguiente figura (transparencia 14 tema 1)



En esta práctica se necesita el TAD pila implementado en la práctica anterior (EDA I) ajustándolo de forma que los elementos de la pila sean punteros a nodos de árboles binarios. Suponiendo un árbol algebraico correcto y limitando el contenido de sus nodos de forma que:

- los nodos hoja tienen en su campo de información algún carácter comprendido entre el '0' y el '9'
- los nodos internos se corresponde con alguno de los operadores algebraicos +, - ó *

Implementar en lenguaje C una función recursiva que evalúe el resultado de la expresión algebraica que representa el árbol. El prototipo de la función debe ser

```
int evaluar(Arbol a);
```

Ejemplo: si la función evalúa el árbol de la derecha de la imagen anterior devolverá el valor 140

4. **Reconstrucción de un árbol binario a partir de sus recorridos**

Suponiendo que todos los elementos de un árbol binario son distintos, implementar un algoritmo que reconstruya un árbol binario a partir de sus recorridos en preorden y en orden.

ERRORES TÍPICOS

Los errores típicos que se suelen cometer en la implementación de las funciones recursivas sobre árboles binarios se pueden resumir en cuatro:

1. No controlar el caso básico (raiz =null)
2. No devolver el valor (null o int)
3. No recoger el valor que devuelve la función
4. Hacer comprobaciones redundantes sobre posibles hijos izquierdos y derechos que están resueltas con el caso básico que debe controlarse obligatoriamente

Normas generales

En la realización de las prácticas se deben seguir los siguientes criterios:

1. Es **obligatorio** utilizar los tipos y prototipos siempre que se proporcionan en el fichero cabecera. Si no se proporcionan debe crearse el fichero cabecera correspondiente o añadir al proporcionado los prototipos de todas las funciones que se implementen.
2. La codificación de las funciones se realizará en un fichero fuente diferente al del programa principal.
3. Para visualizar el correcto funcionamiento de las operaciones implementadas se debe crear un fichero de prueba desde el cual se llamará a las funciones implementadas.
4. Se debe crear el correspondiente Makefile para la correcta creación del fichero ejecutable.