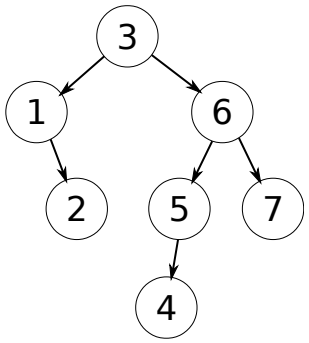
Ejercicios de árboles binarios

OBJETIVOS

El objetivo de estos ejercicios es repasar los conceptos sobre el TAD árbol binario.

En la página de UAMX de la asignatura dispones del esqueleto del código para poder trabajar con estos ejercicios. El examen estará formado por dos ejercicios similares a estos.

En la explicación de las funciones se utiliza como ejemplo el árbol de la siguiente figura:



Ejercicio RecorrerEnProfundidadPreOrden

Implementar una función denominada RecorrerEnProfundidadPreOrden que reciba como argumento un árbol. Esta función debe recorrer en pre orden el árbol e incluir en la cola resultado los elementos.

```
int RecorrerEnProfundidadPreOrden (Arbol ab, Cola *resultado);
```

Ejemplo de ejecución: Tras llamar a la función con el árbol de la figura, la función debe incluir en la cola los números 3 1 2 6 5 4 7.

Ejercicio RecorrerEnProfundidadEnOrden

Implementar una función denominada RecorrerEnProfundidadEnOrden que reciba como argumento un árbol. Esta función debe recorrer en orden el árbol e incluir en la pila resultado los elementos.

```
int RecorrerEnProfundidadEnOrden (Arbol ab, Cola *resultado);
```

Ejemplo de ejecución: Tras llamar a la función con el árbol de la figura, la función debe incluir en la pila los números 1 2 3 4 5 6 7.

Ejercicio RecorrerEnProfundidadPostOrden

Crear una función RecorrerEnProfundidadPostOrden que reciba como argumento un árbol. Esta función debe recorrer en profundidad post orden el árbol e incluir en la pila resultado los elementos.

```
int RecorrerEnProfundidadPostOrden (Arbol ab, Cola *resultado);
```

Ejemplo de ejecución: Tras llamar a la función con el árbol de la figura, la función debe incluir en la pila los números 2 1 4 5 7 6 3.

Ejercicio CalcularNodosInternos

Crear una función CalcularNodosInternos que reciba como argumento un árbol. Debe indicar el número de nodos internos, no terminales, que contiene.

```
int CalcularNodosInternos (Arbol ab);
```

Ejemplo de ejecución: Tras llamar al programa con el árbol de la figura debe devolver el número 4.

Ejercicio CalcularAltura

Crear una función CalcularAltura que reciba como argumento un árbol. Debe devolver la altura del árbol. La función debe ser recursiva.

```
int CalcularAltura(Arbol ab);
```

Ejemplo de ejecución: Tras llamar al programa con el árbol de la figura debe devolver el número 3.

Ejercicio CalcularNElementos

Crear una función CalcularNElementos que reciba como argumento un árbol. Debe devolver el n° de elementos del árbol.

```
int CalcularNElementos (Arbol ab);
```

Ejemplo de ejecución: Tras llamar al programa con el árbol de la figura debe devolver el número 7.

Ejercicio CalcularEquilibrio

Crear una función CalcularEquilibrio que reciba como argumento un árbol. Debe devolver el factor de equilibrio del árbol recorriendo la rama izquierda y a continuación la derecha.

```
int CalcularEquilibrio (Arbol ab);
```

Ejemplo de ejecución: Tras llamar al programa con el árbol de la figura debe devolver el número 1.

Ejercicio CalcularAltura

Crear una función CalcularAltura que reciba como argumento un árbol. Debe devolver la altura del árbol recorriendo la rama izquierda y a continuación la derecha.

```
int CalcularAltura(Arbol ab);
```

Ejemplo de ejecución: Tras llamar al programa con el árbol de la figura debe devolver el número 3.

Ejercicio CalcularAlturaMinima

Crear una función CalcularAlturaMinima que reciba como argumento un árbol. Debe calcular, recorriendo la rama izquierda y a continuación la derecha, la altura de los nodos a las hojas y devolver el valor mínimo.

```
int CalcularAlturaMinima (Arbol ab);
```

Ejemplo de ejecución: Tras llamar al programa con el árbol de la figura debe devolver el número 2.

Ejercicio CrearListaConHermanos

Crear una función CrearListaConHermanos que reciba como argumento un árbol y una lista. Debe incluir en la lista resultado los hermanos del árbol. Debe devolver 1 si el proceso ha concluido con éxito y 0 en caso contrario.

```
int CrearListaConHermanos(Arbol ab, Lista resultado);
```

Ejemplo de ejecución: Tras llamar al programa con el árbol de la figura debe devolver la lista: 1, 6, 2, 5, 7, 4.

Ejercicio CrearListaConHojas

Crear una función CrearListaConHojass que reciba como argumento un árbol y una lista. Debe incluir en la lista resultado las hojas del árbol. Debe devolver 1 si el proceso ha concluido con éxito y 0 en caso contrario.

```
int CrearListaConHojas(Arbol ab, Lista resultado);
```

Ejemplo de ejecución: Tras llamar al programa con el árbol de la figura debe devolver la lista (2, 4, 7).

Ejercicio CrearListaConInternos

Crear una función CrearListaConPadres que reciba como argumento un árbol y una lista. Debe incluir en la lista resultado los padres del árbol. Debe devolver 1 si el proceso ha concluido con éxito y 0 en caso contrario.

```
int CrearListaConInternos(Arbol ab, Lista resultado);
```

Ejemplo de ejecución: Tras llamar al programa con el árbol de la figura debe devolver la lista (3, 1, 6, 5).

Ejercicio RecorrerEnAnchura

Implementar una función denominada RecorrerEnAnchura que reciba como argumento un árbol. Esta función debe recorrer en anchura el árbol e incluir en la cola resultado los elementos.

```
int RecorrerArbolEnAnchura (Arbol ab, Cola *resultado);
```

Ejemplo de ejecución: Tras llamar la función con el árbol de la figura, la función incluir en la cola los números 3 1 6 2 5 7 4.

Ejercicios del TAD árboles binarios de búsqueda

Estos ejercicios se deben implementar dentro del TAD de árboles binarios de búsqueda en el fichero abdb.c.

Ejercicio borrarArbol

Crear una función del TAD abdb para liberar la memoria reservada por todos los nodos del árbol. Se debe implementar en el fichero abdb.c de forma recursiva:

```
void borrarArbol(Arbol ab);
```

Ejercicio encontrar_min

Crear una función borrar que reciba como argumento un árbol y que devuelva un puntero al nodo con el valor más pequeño dentro del árbol. Esta función es necesaria para implementar borrar (ver siguiente ejercicio)

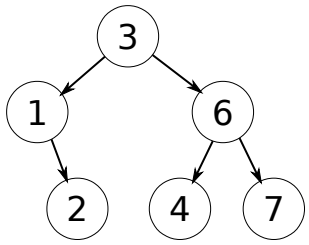
```
PNodoa encontrar_min(Arbol A);
```

Ejercicio borrar

Crear una función borrar que reciba como argumento un número entero y un árbol binario de búsqueda. Debe buscar el elemento y borrarlo del árbol. La función devolverá el árbol con el elemento eliminado. Esta función se debe implementar en el TAD abdb.

```
Arbol borrar (int X, Arbol ab);
```

Ejemplo de ejecución: Tras llamar al programa borrando el elemento 5 en el árbol de la figura debe devolver el árbol



Ejercicios de árboles AVL

Ejercicio rotacionSI

Crear una función rotacionSI que reciba como argumento un árbol AVL. Debe devolver el árbol realizando la rotación simple izquierda de sus nodos. La función devolverá el árbol rotado.

```
static PNodo rotacionSI(PNodo R);
```

Ejercicio rotacionDD

Crear una función rotacionDD que reciba como argumento un árbol AVL. Debe devolver el árbol realizando la rotación doble derecha de sus nodos. La función devolverá el árbol rotado.

```
static PNodo rotacionDD (PNodo R);
```

Ejercicio rotacionDI

Crear una función rotacionDI que reciba como argumento un árbol AVL. Debe devolver el árbol realizando la rotación doble izquierda de sus nodos. La función devolverá el árbol rotado.

```
static PNodo rotacionDI (PNodo R);
```