ASIGNATURA		CURSO	CALIFICACIÓN
ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN		**2002 / 2003**	
TITULACIÓN	GRUPO	CONVOCATORIA	
		EXTRAORDINARIA – SEP.	
APELLIDOS		NOMBRE	

EJERCICIO 1 (3 PUNTOS)

Dado el tipo abstracto de datos (TAD) tArbolBin que sirve para representar árboles binarios de enteros, del que sólo se conoce la parte de la interfaz, con las siguientes funciones disponibles

```
type
    tArbolBin = . . .

    function EsArbolVacio (a: tArbolBin): boolean;
    function Raiz (a: tArbolBin): integer;
    function RamaIzqda (a: tArbolBin): tArbolBin;
    function RamaDrcha (a: tArbolBin): tArbolBin;
    function Altura (a:tArbolBin):integer;
```

Se pide:

A) Definir una función que devuelva el número de nodos internos (nodos no hoja) de un árbol:

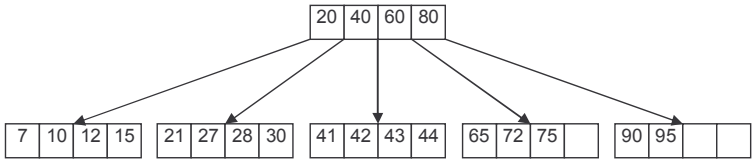
```
function numNodosInternos (a: tArbolBin): integer;
```

B) Definir una función que dado un árbol binario de búsqueda nos diga si cumple las condiciones de un árbol balanceado AVL:

```
function esAVL (a: tArbolBin): boolean;
```

EJERCICIO 2 (3 PUNTOS)

A) Sea el árbol B de la figura siguiente, con número máximo de claves por página (nodo) igual a 4:

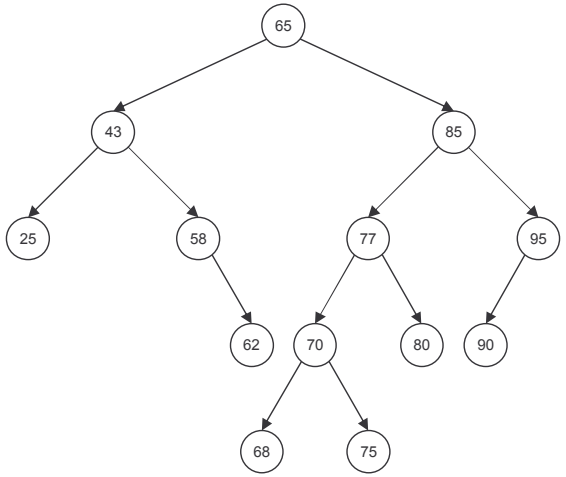


Se pide:

- Realizar la operación de inserción del elemento 8 del árbol
- Realizar la operación de borrado del elemento 90 del árbol original (no del resultante de insertar el elemento 8)

Se deberá dibujar el estado del árbol después de cada operación e indicar detalladamente los pasos que se han llevado a cabo.

B) Sea el árbol balanceado (AVL) de la figura siguiente:



Se pide:

- Realizar la operación de borrado del elemento 25 del árbol

Se deberá dibujar el estado del árbol después de cada operación e indicar el Factor de Equilibrio (FE) de cada nodo. Reflejar las reestructuraciones que sean necesarias (dibujando los estados intermedios) e indicar el tipo de rotación aplicado cuando proceda.

EJERCICIO 3 **(4 PUNTOS)**

Sea una implementación del tipo abstracto de datos (TAD) *tLista*, a través de listas simplemente enlazadas donde cada nodo guarda una cadena, que corresponde al nombre de un producto, y una cantidad numérica, que especifica el número de unidades disponibles para ese producto.

La interfaz del TAD *tLista* incluye las operaciones que se detallan a continuación, en donde *tPos* representa una posición en la lista, *tProducto* representa el tipo nombre de producto, y *tCantidad* el tipo cantidad numérica.

IniciarLista (tLista) → tLista
Objetivo: Inicializa una lista a vacío

EsListaVacia (tLista) → boolean
Objetivo: Determina si una lista está vacía
Precond: Lista debe estar inicializada

Primero (tLista) → tPos
Objetivo: Halla la posición del primer elemento de la Lista
Precond: La lista no puede estar vacía

EsFinLista (tLista, tPos) → boolean
Objetivo: Determina si la posición corresponde al último elemento de la lista
Precond: La lista no puede estar vacía

Siguiente (tLista, tPos) → tPos
Objetivo: Devuelve la posición que sigue en la lista a la posición indicada.
Si la posición indicada es la última devolverá nulo.
Precond: La lista no puede estar vacía y tPos es una posición válida de la lista

DevuelveValor (tLista, tPos) → tCantidad, tProducto
Objetivo: Devuelve la cantidad almacenada en el elemento de la posición indicada.
Precond: La lista no puede estar vacía y tPos es una posición válida de tLista

InsertarPrimero (tLista, tProducto, tCantidad) → tLista
Objetivo: Inserta en la cabeza de la lista, el elemento definido por
los datos de entrada tProducto y tCantidad
Precond: La lista debe estar inicializada

Insertar (tLista, tPos, tProducto, tCantidad) → tLista
Objetivo: Inserta después de la posición indicada de la lista, el elemento definido por
los datos de entrada tProducto y tCantidad
Precond: La lista no puede estar vacía y tPos es una posición válida de tLista

Se pide:

A) Realizar en lenguaje Pascal la definición de tipos del *TAD tLista* y la implementación de todas las operaciones descritas.

B) Implementar en Pascal una operación que a partir de dos listas NO VACÍAS, ordenadas por el campo Cadena, genere una lista resultado, en la cual exista una sola ocurrencia para cada cadena (en caso de que existan dos cadenas iguales en ambas listas, será necesario sumar las cantidades) El acceso a las listas se realizará utilizando exclusivamente la interfaz proporcionada para el *TAD tLista*.

Mezcla (tLista, tLista) → tLista

Soluciones
Estructura de Datos y de la Información.
Curso 2001/2002. Convocatoria de Septiembre.

EJERCICIO 1

A)

```
function numNodosInternos (a:tArbolBin):integer;
begin
  if esArbolVacio(A)
     then numNodosInternos := 0
  else if esArbolVacio(ramaIzqda(A)) AND esArbolVacio(ramaDcha(A))
     then numNodosInternos := 0
  else numNodosInternos := 1 + numNodosInternos(ramaIzqda(A))
                             + numNodosInternos(ramaDcha(A))
end;
```

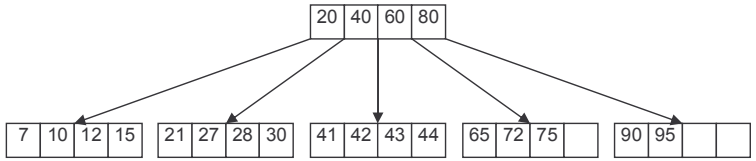
B)

```
function esAVL (a:tArbolBin):integer;
var fe:integer;
begin
  if esArbolVacio(A)
     then EsAVL := TRUE
  else begin
    fe := altura(ramaIzqda(A)) - altura(ramaDcha(A));
    esAVL := (fe >= -1) AND (fe <= 1) AND
             esAVL(ramaIzqda(A)) AND esAVL(ramaDcha(A))
end;
```

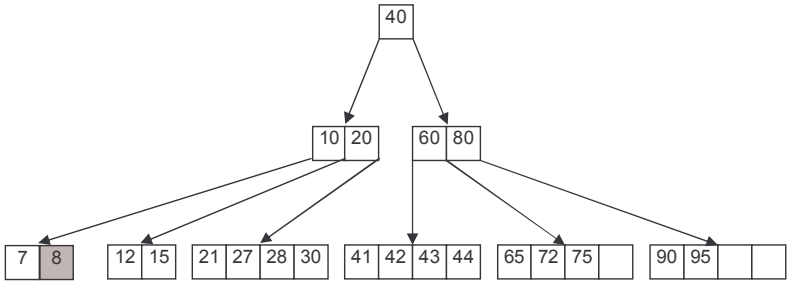
EJERCICIO 2 (A)

SOLUCION:

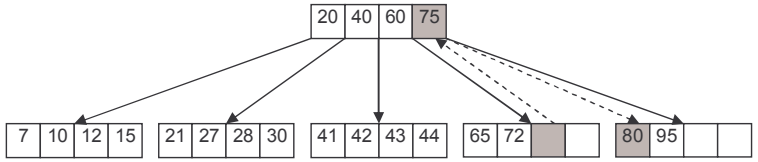
Árbol original



Añadir 8 al árbol original

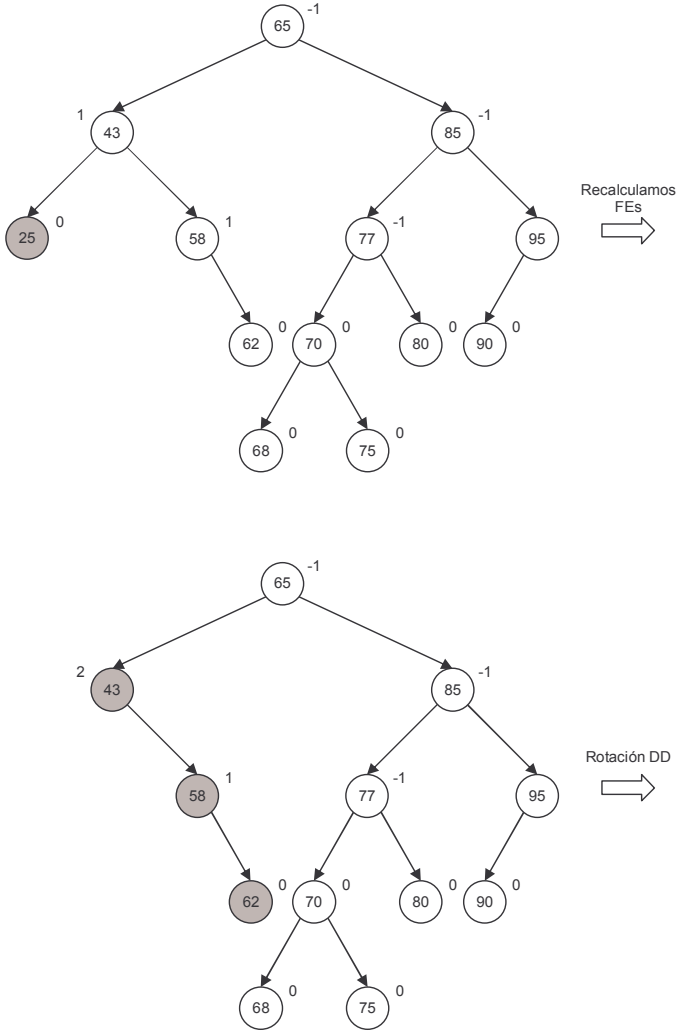


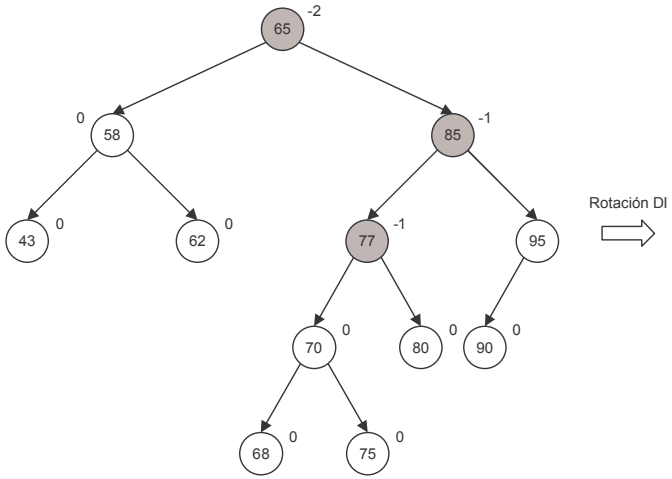
Borrar 90 al árbol original

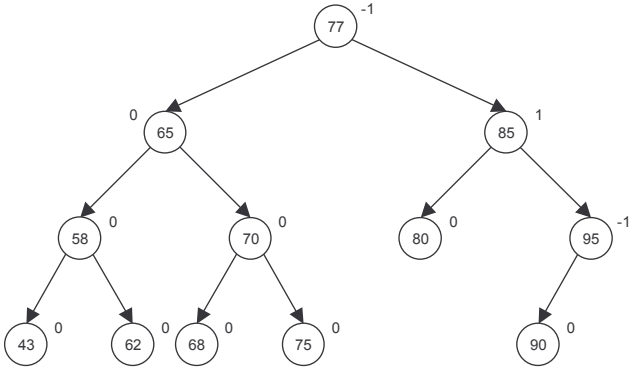


EJERCICIO 2 (B)

Borrar 25







EJERCICIO 3

A) Definición de tipos

```
const NULO = nil;
type
  tProducto = string;
  tCantidad = integer;
  tInfo = record
              producto : tProducto;
              cantidad : tCantidad;
            end;
  tPos = ^tNodo;
  tNodo = record
            info : tInfo;
            sig : tPos;
          end;
  tLista = tPos;


...
```

B) Función Mezcla

```
/*funciones auxiliares*/


function Mover_siguiente(L:tLista; var Pos: tPos; var cantidad: tcantidad;
                         var producto: tproducto; var final: boolean);

{Se mueve al siguiente nodo de la lista y devuelve esa posición y su contenido
En caso de que ya no exista un nodo siguiente devuelve true en la variable final.
Con esto se evita tener que tratar aparte el último nodo de las listas}

begin
      if not EsFinLista(L, Pos) then begin
            Pos := Siguiente(L,Pos);
            DevuelveValor(L, Pos, cantidad, producto);
      end
      else final:=true;
end;


/************************/

procedure Añadir (A:tLista; PosA: tPos; var Copia:tLista; finCopia: tPos);
{Añade al final de Copia los nodos de A situados a partir de Pos
precond: A no vacía}

var
      ICopia: tInfo;
      final: boolean;

begin
   final:= false;
   repeat
      DevuelveValor(A, PosA, ICopia.cantidad, ICopia.producto);
      Insertar (Copia, finCopia, ICopia.producto, ICopia.cantidad);
      finCopia:= Siguiente(finCopia);
      if not EsFinLista(A, PosA)
      then Pos := Siguiente(L,Pos);
      else final:=true;
   until final;
end; /*copiar*/

/************************/
```

```
procedure Calcular_info (A,B: tLista; var PosA, PosB: tPos; var Aproducto, Bproducto,
Rproducto: tProducto; var Acantidad, Bcantidad, Rcantidad: tCantidad; var final: boolean);

{A partir de las dos listas originales calcula la información que deberá insertarse en la copia
además de dejar las listas originales en el estado adecuado para tratar el siguiente nodo}

begin
  if Aproducto < Bproducto then begin
        IR.producto := Aproducto;
        IR.cantidad:=Acantidad;
        Mover_siguiente(A, PosA, Acantidad, Aproducto, final);
  end
  else if Aproducto = Bproducto
        then begin
               IR.producto := Aproducto;
               IR.cantidad := Acantidad + Bcantidad;
               Mover_siguiente(A, PosA, Acantidad, Aproducto, final);
               Mover_siguiente(B, PosB, Bcantidad, Bproducto, final);
               end
        else begin
               IR.producto := Bproducto;
               IR.cantidad:=Bcantidad;
               Mover_siguiente(B, PosB, Bcantidad, Bproducto, final);
        end
end;




/***********************/

procedure Mezcla (A,B:tLista; var Resultado: tLista);
var
        IA, IB, IR: tInfo;
        PosR, PosA, PosB: tPos;
        final: boolean;

begin
        IniciarLista(Resultado);
        PosA:=Primero(A);
        PosB:=Primero(B);
        final:=false;

        /*realizamos primero la inserción del primer nodo*/
        DevuelveValor(A, PosA, IA.cantidad, IA.producto);
        DevuelveValor(B, PosB, IB.cantidad, IB.producto);
        Calcular_info (A,B,PosA, PosB, IA.producto, IB.producto, IR.producto, IA.cantidad,
        IB.cantidad, IR.cantidad, final);
        InsertarPrimero (Resultado, IR.producto, IR.cantidad);
        posR:=primero(Resultado);

        while not final do
        begin
               Calcular_info (A,B,PosA, PosB, IA.producto, IB.producto, IR.producto,
               IA.cantidad, IB.cantidad, IR.cantidad, final);
               Insertar (Resultado, PosR, IR.producto, IR.cantidad);
              PosR := Siguiente(Resultado,PosR);
        end; /*while*/

        if EsFinLista(A) then
               if not EsFinLista(B)
               then añadir(B, PosB, Resultado, PosR)
        else añadir(A, PosA, Resultado, PosR)
end.
```