ASIGNATURA		CURSO	CALIFICACIÓN
ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN		**2002/2003**	
TITULACIÓN	GRUPO	CONVOCATORIA	
		JUNIO	
APELLIDOS		NOMBRE	

EJERCICIO 1 (3 PUNTOS)

Dado el tipo abstracto de datos (TAD) tArbolBin que sirve para representar árboles binarios de enteros (no árboles de búsqueda), del que sólo se conoce la parte del interfaz

```
type tArbolBin = ...
function EsArbolVacio (a:tArbolBin):boolean;
function Raiz (a:tArbolBin):integer;
function RamaIzqda (a:tArbolBin):tArbolBin;
function RamaDcha (a:tArbolBin):tArbolBin;
function Altura (a:tArbolBin):integer;
```

SE PIDE:

A) Definir una función que devuelva el menor entero almacenado en un árbol no vacío:

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

B) Definir una función que nos indique si un árbol binario de enteros es o no un "ÁrbolMin":

```
function EsArbolMin (a:tArbolBin):boolean:
```

Nota: un árbol binario de enteros es un "ÁrbolMin" si:
- bien es vacío;
- bien es un árbol lleno (todas las hojas al mismo nivel) tal que la raíz contiene el menor valor almacenado en el árbol y sus subárboles izquierdo y derecho son "ÁrbolMin".

EJERCICIO 2 (3 PUNTOS)

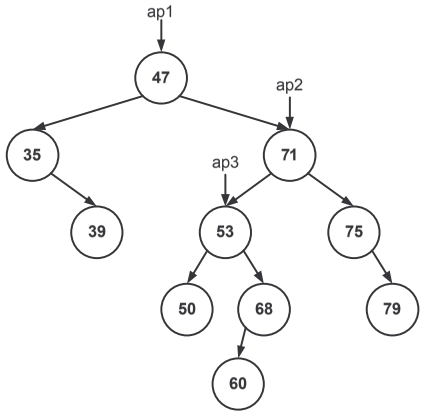
A) En el siguientes montículo (árbol en montón) *max*, realizar las operaciones de inserción y eliminación pedidas, **detallando los pasos intermedios** y **mostrando el resultado final en forma de array**. Las posiciones con un guión no forman parte del montículo.

25	15	12	13	10	7	11	8	6	-	-	-

Después de borrar la clave 25

A partir del montículo inicial, después de insertar la clave 19

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



Se pide:

Utilizando los apuntadores **`ap1`**, **`ap2`** y **`ap3`**, indicar una secuencia de asignaciones que reequilibre el árbol, dejando todos los nodos correctamente enlazados y el apuntador **`ap1`** apuntando a la raíz. Los campos del tipo nodo para acceder a los subárboles se llaman **`iz`** y **`de`**.

EJERCICIO 3 (4 PUNTOS)

Se trata de modelizar una unidad de urgencias. Los pacientes se clasifican a la entrada según su gravedad (1:muy grave, 2:grave, 3:leve). Existen 3 salas de atención, una para cada nivel de gravedad. Los pacientes se atienden por orden de gravedad y, dentro del mismo nivel de gravedad, por orden de entrada, para ello a cada paciente se le asocia un tiempo de entrada (suponer un entero).

Periódicamente, se revisan las colas para cada sala y los pacientes que llevan mucho tiempo esperando (el tiempo actual supera al de entrada más una constante *Tmax*, *Tentrada*+*Tmax*<*Tactual*) en su cola, pasan a la cola de mayor gravedad, asignándoles como tiempo de entrada el tiempo actual.

Se pide:

A. Definición de tipos y especificación de operaciones de un TAD para representar la cola correspondiente a un nivel de gravedad. Se deberán implementar las operaciones
   a. añadir un paciente a la cola
   b. extraer un paciente de la cola.

B. Utilizando el TAD del paso anterior, definición de tipos de un TAD para representar el sistema. Se deberán especificar e implementar las operaciones
   c. añadir un paciente al sistema
   d. extraer un paciente del sistema
   e. reasignar pacientes en colas

Las especificaciones serán breves: nombre operación, parámetros, objetivo y precondiciones. El número de pacientes es ilimitado, pero el de colas es limitado (3 niveles de gravedad) por lo que puede optarse por una representación del sistema totalmente dinámica o sólo parcialmente. Indica la opción elegida y justifica su uso.

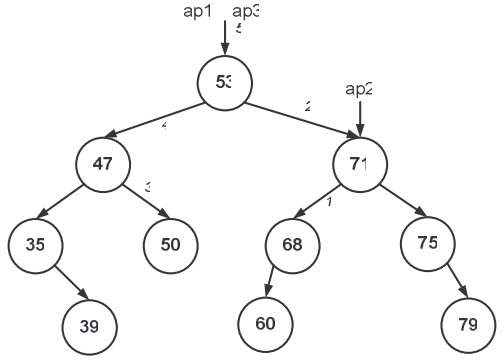
Soluciones

Ejercicio 1:

```
Function ValorMinimo (a:tArbolBin):integer;

  Function min (x,y:integer):integer;
  begin
    if x<y
       then min := x
    else min := y
  end;

begin
  if EsArbolVacio(RamaIzqda(a)) and EsArbolVacio(RamaDcha(a))
  then ValorMinimo := Raiz(a)
  else if EsArbolVacio(RamaIzqda(a))
       then ValorMinimo := min(Raiz(a), ValorMinimo(RamaDcha(a))
       else if EsArbolVacio(RamaDcha(a))
            then ValorMinimo := min(Raiz(a), ValorMinimo(RamaIzqda(a))
            else ValorMinimo := min(Raiz(a),
                  min(ValorMinimo(RamaIzqda(a)),ValorMinimo(RamaDcha(a))))
end;


Function EsArbolMin (a:tArbolBin):boolean;
begin
  if EsArbolVacio(a)
  then EsArbolMin := true
  else EsArbolMin :=     (Raiz(a) = ValorMinimo(a))
                     and (Altura(RamaIzqda(a)) = Altura(RamaDcha(a)))
                     and EsArbolMin(RamaIzqda(a))
                     and EsArbolMin(RamaDcha(a))
end;
```

Ejercicio 2:

ROTACIÓN DI



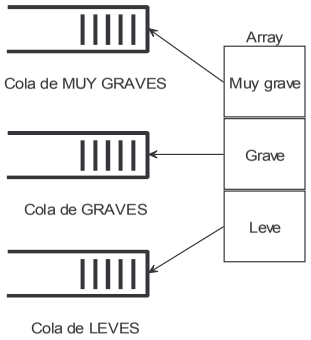

```
  1) ap2^.iz  := ap3^.de;          ap1^.de := ap3^.iz;  (3)
  2) ap3^.de  := ap2;              ap2^.iz := ap3^.de;  (1)
  3) ap1^.de  := ap3^.iz;          ap3^.iz := ap1;      (4)
  4) ap3^.iz  := ap1;              ap3^.de := ap2;      (2)
  5) ap1      := ap3;              ap1     := ap3;      (5)
```

Ejercicio 3:

Cola

Definición de tipos:

```
tTiempo = integer;
tNombre = string;
tPos = ^tNodo;
tNodo = record
 nombre: tNombre;
 Tiempo: tTiempo;
 sig: tPos;
end;

tCola = record
     ini, fin: tPos;
end;
```



Especificación de operaciones:

```
procedure IniciarC (var C: tCola);
{Objetivo : Crear una cola vacia
Entrada   : C: variable de acceso a la cola
Salida    : La cola vacia C
Precond   : C no debe tener datos
Poscond   : C esta inicializada y sin datos}
```

```
Function EsColaVacia (C: tCola): boolean;
{Objetivo : Comprobar si hay datos en la cola
Entrada   : Cola: variable de acceso a la cola
Salida    : verdadero si la cola esta vacía, falso en caso contrario.
Precond   : Cola inicializada
Poscond   : ninguna}
```

```
Procedure FrenteC (C:tCola; var paciente:tNombre; Var entrada:tTiempo);
{Objetivo : obtener la informacion del elemento al frente de la cola
Entrada   : C: variable de acceso a  la cola
Salida    : paciente: nombre del primer elemento
            Entrada: tiempo de entrada en la cola
PreCond   : La cola no debe de estar vacia }
```

Implementación de operaciones:

```
Procedure AnadirC (Var C: tCola; paciente: tNombre; entrada: tTiempo);
{Objetivo : Inserta un nodo (al final de la cola)
Entrada   : paciente: nombre del paciente
            entrada: tiempo de entrada en la cola
            C: variable de acceso a la cola
Salida    : La Cola con el paciente insertado al final
PreCond   : La Cola debe estar inicializada, y existe memoria suficiente}

var
      nuevo: tPos;
begin
      new(nuevo);
      nuevo^.nombre:= paciente;
      nuevo^.tiempo:= entrada;
      nuevo^.Sig:=nulo;
      with C do
            begin
                if Es_Cola_vacia(Cola)
                then
                    Ini:= nuevo
                else
                    Fin^.sig:= nuevo;
                Fin:= nuevo;
            end
end;
```

```
Procedure ExtraerC (Var C: tCola);
{Objetivo : Elimina de la cola un paciente (siempre el primero)
Entrada   : C: variable de acceso a  la cola
Salida    : La cola C sin el primer elemento
PreCond   : La Cola no debe estar vacía}
PosCond   : Se modifica el valor del inicio de la Cola }

var
      auxt: tPos
begin
      with Cola do
      begin
            aux:= ini;
            ini:=ini^.Sig;
            if Ini = nulo
            then fin:=nulo; (* si cola se queda vacía inicializarla *)
            dispose(aux);
      end;
end;
```

Definición de tipos:

```
tGravedad = (muygrave, grave, leve); (* También tipo subrango 1..3 *)
tColaPri = array [tGravedad] of tCola;
```

Especificación e Implementación de operaciones:

```
Procedure AnadirPac (Var CP: tColaPri; paciente: tNombre;
                     gravedad: tGravedad; tiempo: tTiempo);
{Objetivo: añadir un paciente a la CP clasificándolo según gravedad
 Entradas:       Cola de Prioridad,
                 paciente: nombre del paciente
                 gravedad: Tipo de gravedad del paciente
                 tiempo: Tiempo del sistema
 Salidas: La cola de prioridad con el paciente en una de las colas
 Precondiciones: No existe ningún paciente con ese mismo nombre y la CP
                     está inicializada, así como cada una de las colas}

Begin AnadirC (CP[gravedad], nombre, Tiempo) end;
```

```
Procedure ExtraerPac (var CP: tColaPri);
{Objetivo: Extraer (atender) el paciente de mayor prioridad de la CP
 Entradas: Cola de Prioridad
 Salidas: La cola de prioridad sin el paciente de mayor prioridad
 Precondiciones: La CP no está vacía}
Var
    cola_i: tGravedad;
    fin: boolean;
begin
    fin:= false;
    cola_i:= muygrave;

    (* Sabemos que CP no está vacía *)
    while not fin do begin
          if not EsColaVacia (CP[cola_i])
          then begin
                ExtraerC (cp[cola_i]);
                fin := true;
          end else inc(cola_i);
    end; (*while*)
end;
```

```
Procedure ReasignarPac (var CP:tColaPri; tMax:tTiempo; tSistema:tTiempo);
{Objetivo: Incrementar prioridad de pacientes tiempo_espera>tMax
 Entradas:       Cola de Prioridad
                 Tiempo: Tiempo de espera máximo en una de las colas
 Salidas: La cola de prioridad con pacientes reasignados
 Precondiciones: La CP no está vacia}
Var
    cola_i: tGravedad;
    fin: boolean;
begin
    for cola_i:= grave to leve do begin
          fin:= false;
          while not EsColaVacia (CP[cola_i]) and not fin do begin
                FrenteC(CP[cola_i], nombre, tiempo);
                if (tiempo + tMax) < tSistema
                then begin
                      Sacar(CP[cola_i]);
                      AnadirC (CP[cola_i-1], nombre, tSistema);
                end
                else fin:= true;
          end;
    end; (*for*)
end;
```