ASIGNATURA	CURSO	GRUPO	CONVOCATORIA
PROGRAMACIÓN II		**1/2/3/4/5**	**PRIMERA OPORTUNIDAD (JUNIO)**

APELLIDOS	NOMBRE	CALIFICACIÓN

OBSERVACIONES

EJERCICIO 1

Dados los siguientes supuestos prácticos decidir: 1) cuál es la MEJOR estructura de datos, y 2) su MEJOR implementación para resolver el problema (para ser considerada correcta, la respuesta deberá estar justificada):

1. Una empresa quiere montar un servicio de cafetería en el que se sirva preferentemente café aunque también dispondrá de té, bebidas gaseadas y zumos naturales. Para mantener esta política, los camareros servirán primero a los clientes que hayan pedido café, según vayan llegando, y después a los clientes que hayan pedido te, bebidas gaseadas y zumo, en este orden. ¿Qué estructura sería la más adecuada para gestionar las peticiones de los clientes?

   SOLUCION: Una cola de prioridad, donde la prioridad vendría definida por el tipo de bebida pedida: por propio funcionamiento de la estructura se atenderían antes los de mayor prioridad, y dentro de la misma prioridad, primero los más antiguos. Al no haber número de clientes fijo la implementación más adecuada sería una lista estática de 4 elementos (los tipos de bebidas) de colas dinámicas, para los clientes que piden cada una de las bebidas.

2. Un conocido centro comercial ha decidido que a la hora de cancelar el alquiler de sus locales lo hará de aquellos que lleven menos tiempo ocupados. ¿Qué estructura sería la más adecuada para determinar qué local debe ser desalojado en cada momento?

   SOLUCION: Una pila ya que de esta forma el local que menos tiempo lleve ocupado será el primero cuyo alquiler se cancele. Su implementación será estática ya que normalmente el número de locales del centro comercial es fijo.

3. Un diseñador de complementos decide ofrecer la posibilidad a sus clientas de comprar zapatos y bolso de forma combinada. Para ello decide contratar el diseño de una aplicación que permita visualizar todos sus modelos de zapatos y para cada par de zapatos, la lista de bolsos con los que mejor combina. La estructura deberá permitir que las clientas avancen o retrocedan tanto en la lista de bolsos como en cada lista de zapatos. ¿Qué estructura sería la más adecuada para diseñar esta aplicación?

   SOLUCIÓN: Una multilista doblemente enlazada. El primer nivel de la multilista almacenaría la información de los zapatos, y el segundo nivel la información de los bolsos a juego. El doble enlace en ambas listas nos permitirá

acceder al elemento anterior y al siguiente tal y como requiere el enunciado. La implementación de ambas listas sería dinámica ya que no conocemos el número de elementos que van a contener.

EJERCICIO 2

Contestar Verdadero o Falso y explicar el porqué a las siguientes preguntas (para ser considerada correcta, la respuesta deberá estar justificada):

1. En la especificación de una operación, las poscondiciones indican lo que ocurre si no se cumplen las precondiciones.

   FALSO. Las poscondiciones completan la información del objetivo, entradas y salidas de una operación. De hecho, indican lo que ocurre si la operación se lleva a cabo correctamente.

2. Una cola de prioridad se comporta en ocasiones como una cola estándar.

   VERDADERO. En caso de que todos los elementos tengan la misma prioridad.

3. Una multilista es una lista que debe permitir el recorrido ordenado de los elementos atendiendo a más de un criterio.

   FALSO. Eso sería una lista multienlazada o multiordenada. La multilista es una lista en la que de cada elemento pende una nueva lista.

4. En un AVL, una eliminación puede obligar a realizar una rotación como máximo.

   FALSO. Tras el proceso de eliminación, la reestructuración podría alcanzar la raíz del árbol y por tanto realizar más de una rotación. Es en la inserción en dónde después de realizar la primera rotación, el árbol queda equilibrado.

EJERCICIO 3

Dado el siguiente código, indicar cuál es el error que se produce en su ejecución y porqué:

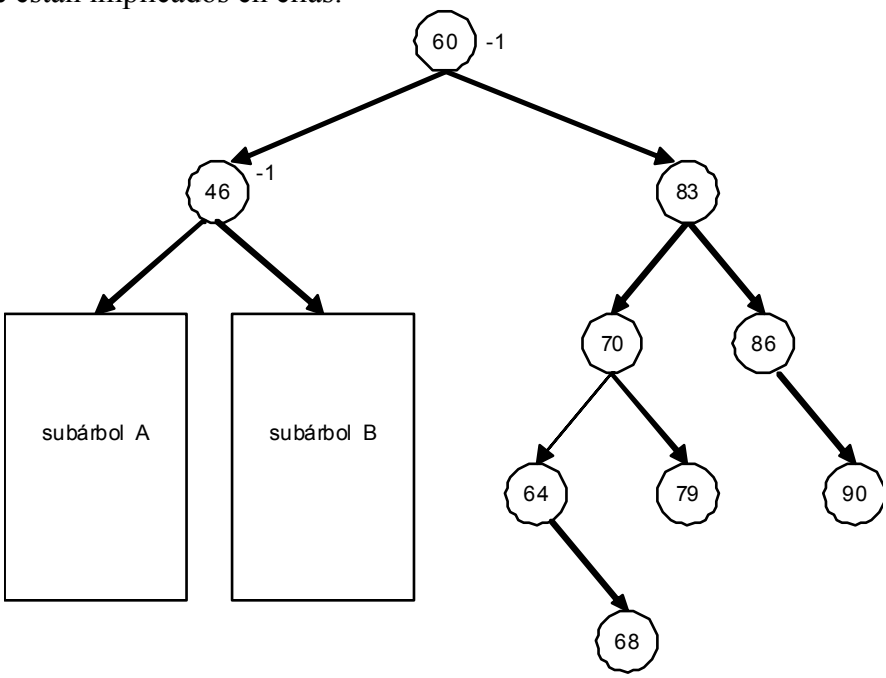
```
Program Error;
type tPEntero= ^integer;

var P,Q: tPEntero;

procedure ProcesarEntero(var M: tPEntero);
   begin
      M^:= M^ * 3 + 8;
      writeln(M^);
      dispose(M);
      M:= nil;
   end;

begin
   new(P);
```

```
   P^:= 4;
   Q:= P;
   ProcesarEntero(P);
   ProcesarEntero(Q);
end.
```

SOLUCIÓN: El procedimiento `ProcesarEntero` *hace el* `dispose` *sobre el puntero que se le envía, destruyendo la variable dinámica a la que apunta. Como P y Q apuntan a la misma variable, una vez que se hace la llamada a* `ProcesarEntero(P)` *la siguiente llamada* `ProcesarEntero(Q)` *daría un error de ejecución al no existir ya la variable* `Q^` *y tratar de acceder a ella.*

EJERCICIO 4

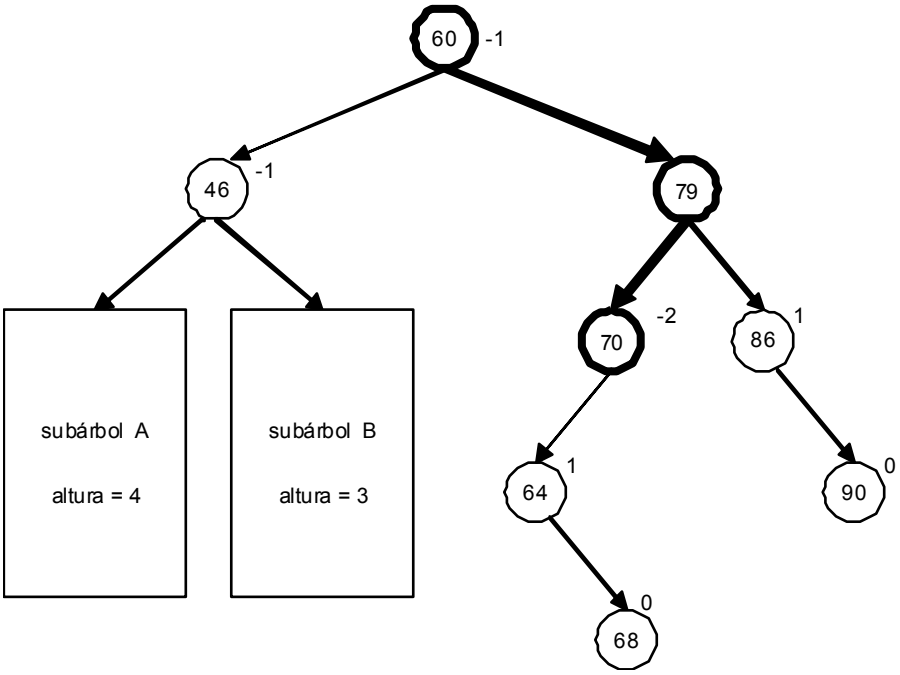
Dado el árbol AVL siguiente, calcular la altura de los subárboles A y B en función de los dos factores de equilibrio indicados. A continuación, eliminar la clave 83 y dibujar los sucesivos estados que atraviesa el árbol hasta llegar al estado final, indicando los factores de equilibrio en cada fase, las rotaciones que sea necesario realizar y los 2 ó 3 nodos que están implicados en ellas.



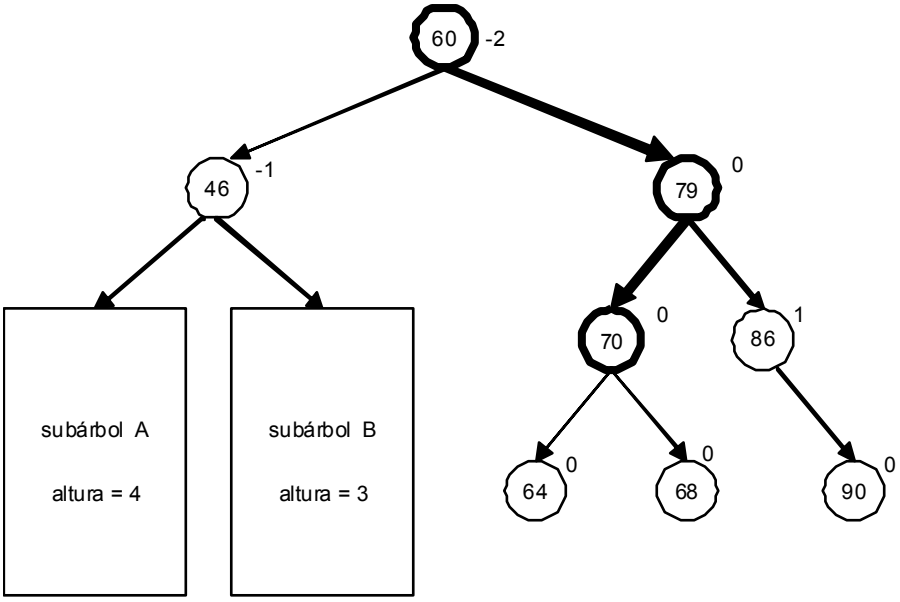
SOLUCION:
Determinamos en primer lugar la altura de los subárboles indicados:

1) *según el FE del nodo 60 la altura del nodo 46 = 5*
2) *según el FE del nodo 46 la altura del subárbol A = 4 y del subárbol B=3*

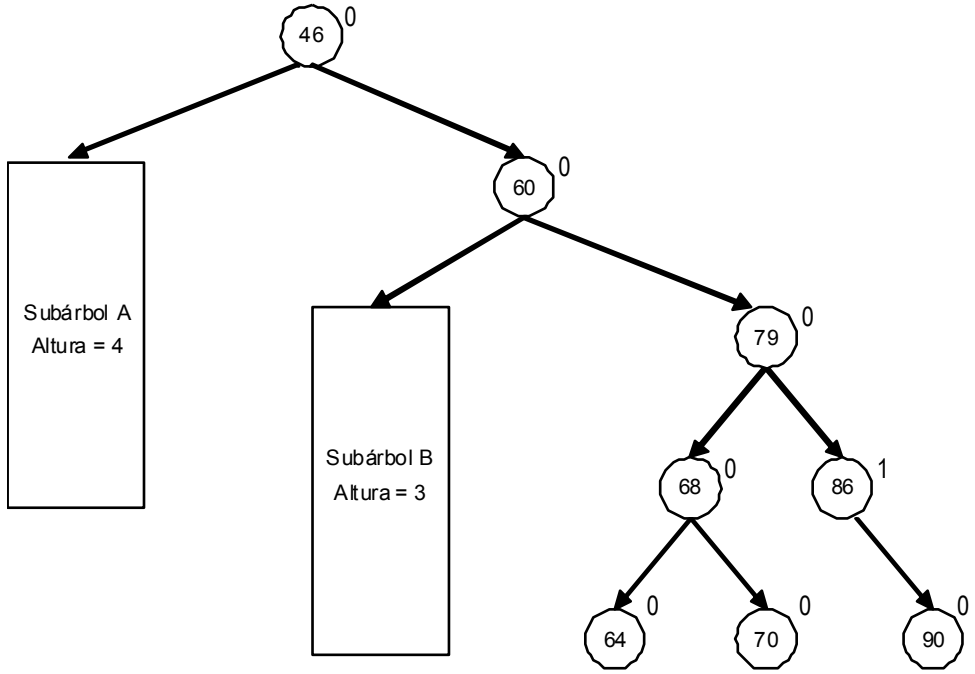
sustituimos la clave 83 por el 79 y eliminamos ese nodo:



Regresamos hacia atrás por el camino de búsqueda (marcado). Rotación ID: implica a los nodos 70-64-68.

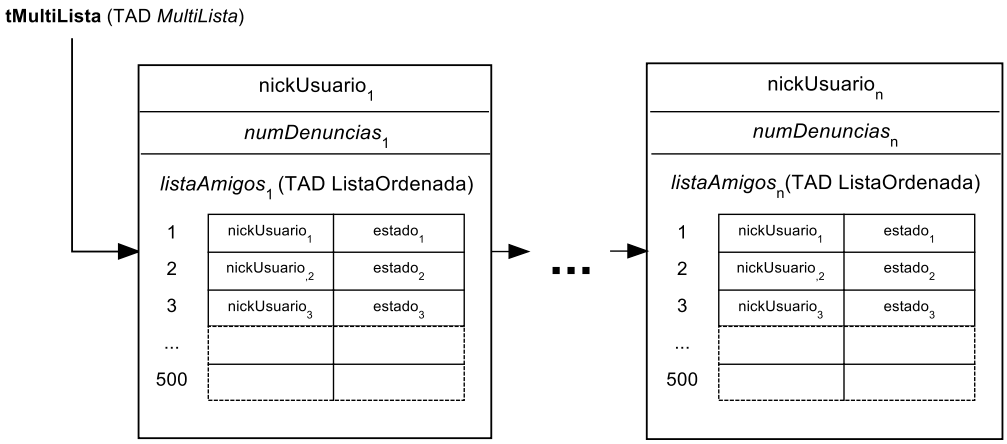


Necesaria una segunda rotación II: implica a los nodos 60-46. Resultado:



EJERCICIO 5

En la práctica 2 se implementaba la gestión de usuarios de una red social utilizando una lista simplemente enlazada y ordenada para almacenar las amistades para un usuario dado (TAD ListaOrdenada) y una multilista ordenada (TAD MultiLista) para almacenar los datos de los usuarios de la red social y su lista de amistades asociada. Partiendo del mismo problema, en este ejercicio se proponen una serie de modificaciones, tal como se explicará a continuación.



A)

El primer cambio afecta al TAD Lista Ordenada de amistades ahora de implementación estática. Se pide como ejercicio realizar la nueva definición de tipos de datos de la lista:

- tListaOrd — Representa una lista ordenada alfabéticamente por nick de usuario.
- tNickUsuario: — Nick de usuario (string[9]).
- tEstado: — Estado de la amistad (enumerado): aceptada, enespera.

- tDatoL: Dato de un elemento de la lista, compuesto por los campos *NickUsuario* y *estado* que contienen el nickname del usuario que ha solicitado/tiene su amistad y el estado de dicha relación de amistad, respectivamente.
- tPosL: Posición de un elemento de la lista.
- NULOL: Constante que expresa una posición nula.
- MAXL: Tamaño máximo de la lista (500 amistades).

Además, partiendo de la nueva definición de tipos de datos, realizar la implementación de las siguientes operaciones:

EliminarPosicion (tListaOrd, tPosL) → tListaOrd
Objetivo: Borra de la lista el elemento que está en la posición indicada
Entrada: Lista a modificar y posición del elemento que queremos borrar
Salida: La lista sin el elemento indicado
Precondición: La posición es una posición válida de la lista.

BuscarDato (tNickUsuario, tListaOrd) → tPosL
Objetivo: Busca el primer elemento con cierto contenido en la lista
Entrada: Contenido del elemento buscado y lista donde realizar la búsqueda
Salida: Posición del elemento encontrado o nulo si no se encuentra.

En el ANEXO a este examen se adjunta la interfaz del TAD Lista Ordenada, la misma usada en la realización de las prácticas, y que incluye todas las operaciones disponibles para su manejo.

```
const
    NULOL=0;
    MAXL=500;
type
    tNickUsuario= string[9];
    tEstado= (aceptada, enespera);
    tDatoL= record
        nickUsuario: tNickUsuario;
        Estado: tEstado;
    end;
    tPosL: 0..MAXL;
    tListaOrd= record
        datos: array[1..MAXL] of tDatoL;
        fin: tPosL;
    end;


procedure EliminarPosicion (var L: tListaOrd; p: tPosL);
var
    i: tPosL;
begin
    L.fin:= L.fin - 1;
    for i:= p to L.fin do
        {desplaza a la izquierda}
        L.datos[i]:= L.datos [i+1];
end;

function BuscarDato (nick: tNickUsuario; L: tListaOrd): tPosL;
var
    p: tPosL;
begin
```

```
    if EsListaVacia(L) or
                (L.datos[Ultima(L)].nickUsuario < nick) then
        BuscarDato:= NULOL
    else begin
        p:= Primera(L);
        while (p < L.fin) and (L.datos[p].nickUsuario < nick) do
            p:= Siguiente(Lista,p);
        if (L.datos[p].nickUsuario = nick) then
            BuscarDato:= p
        else
            BuscarDato:= NULOL
    end;
end;
```

B)

Supongamos que los gestores de la plataforma quieren premiar a los usuarios más activos, es decir, aquellos que han confirmado todas las solicitudes de amistad (el usuario sin amistades no se considera activo). Implementar una operación que, utilizando el TAD Multilista y el TAD ListaOrdenada, efectúe un listado de estos usuarios. Como precondición se establece que la multilista no esté vacía.

ImprimirUsuariosActivos (tMultiLista) → Ø

En el ANEXO a este examen se adjunta la interfaz de ambos TADs, la misma usada en la realización de las prácticas, y que incluye todas las operaciones disponibles para su manejo.

```
procedure ImprimirUsuariosActivos (M: tMultiLista);
var
    posM: tPosM;
    datoM: tDatoM;
    posL: tPosL;
    activo: boolean;
begin
    posM:= PrimeraM(M);
    while (posM <> NULOM) do begin
        datoM:= ObtenerDatoM (M, posM);
        activo:= true;
        if not EsListaVacia(datoM.listaAmigos) then
           posL:= Primera (datoM.listaAmigos);
        else
           activo:= false;
        while activo and (posL<>NULOL) do begin
            datoL:= ObtenerDato (datoM.listaAmigos, posL);
            if datoL.estado = enespera then
                activo:= false
            else posL:= Siguiente (datoM.listaAmigos, posL);
        end;
        if activo then writeln (datoM.nickUsuario);
        posM:= SiguienteM (M, posM);
    end;
end;
```

ANEXO

TAD ListaAmistades: Mantiene las amistades de un usuario ordenadas por nick de usuario

Tipos de datos

tListaOrd	Representa una lista ordenada por nick de usuario
tNickUsuario:	Nick de usuario (string[9])
tEstado:	Estado de la amistad (tipo enumerado: {aceptada, enespera})
tDatoL:	Dato de un elemento de la lista, compuesto por los campos NickUsuario y estado que contienen el nickname del usuario que ha solicitado/tiene su amistad y el estado de dicha relación de amistad, respectivamente.
tPosL:	Posición de un elemento de la lista.
NULOL:	Constante que expresa una posición nula.
MAXL:	Tamaño máximo de la lista (500 amistades).

Operaciones

ListaVacia (tListaOrd) → tListaOrd
Crea una lista vacía.

esListaVacia (tListaOrd) → Boolean
Determina si la lista está vacía.

Primera (tListaOrd) → tPosL
Devuelve la posición del primer elemento de la lista.
Precondición: La lista no está vacía.

Ultima (tListaOrd) → tPosL
Devuelve la posición del último elemento de la lista.
Precondición: La lista no está vacía.

Siguiente (tListaOrd, tPosL) → tPosL
Devuelve la posición del siguiente elemento a la posición indicada (o NULO si la posición no tiene siguiente).
Precondición: La posición tiene que ser válida.

Anterior (tListaOrd, tPosL) → tPosL
Devuelve la posición del anterior elemento a la posición indicada (o NULO si la posición no tiene anterior).
Precondición: La posición tiene que ser válida.

InsertarDatoLista(tListaOrd, tDatoL)→ tListaOrd, Boolean
Inserta ordenadamente en la lista, en base al campo nickUsuario, un nuevo amigo. Devuelve falso sólo si no hay memoria suficiente para realizar la operación.

EliminarPosicion (tListaOrd, tPosL) → tListaOrd
Borra de la lista el elemento que está en la posición indicada.
Precondición: La posición tiene que ser válida.

ObtenerDato (tListaOrd, tPosL) → tDatoL
Devuelve el dato situado en la posición indicada de la lista.
Precondición: La posición tiene que ser válida.

ActualizarDato(tListaOrd, tPosL, tDatoL) → tListaOrd
Actualiza la información asociada al elemento situado en la posición indicada.
Precondición: La posición tiene que ser válida

BuscarDato (tListaOrd, tNickUsuario) → tPosL
Devuelve la posición del elemento con nick tNickUsuario (o NULO si el elemento no existe).

TAD MultiLista: Almacena ordenadamente en base al nick los usuarios de la red social, y su lista de amistades.

Tipos de datos

tMultilista	Representa a una multilista ordenada simplemente enlazada.
tNickUsuario	Nick de usuario (string[9]).
tNumDenuncias	Número de denuncias recibidas por un usuario (integer).
tListaOrd	Lista de amistades ordenada por nick de usuario.
tDatoM	Dato de un elemento de la multilista, compuesto por los campos *nickUsuario*, *numDenuncias* y *listaAmigos*, que corresponden al nick asociado al usuario, número de denuncias recibidas y lista ordenada de amigos del usuario.
tPosM	Posición de un elemento de la multilista.
NULOM	Constante utilizada para representar posiciones nulas.

Operaciones

MultilistaVacia (tMultilista) → tMultiLista
Crea una multilista vacía.

esMultilistaVacia (tMultilista) → Boolean
Determina si la multilista está vacía.

PrimeraM (tMultilista) → tPosM
Devuelve la posición del primer elemento de la multilista.
Precondición: La multilista no está vacía.

UltimaM (tMultilista) → tPosM
Devuelve la posición del último elemento de la multilista.
Precondición: La multilista no está vacía.

SiguienteM (tMultilista, tPosM) → tPosM
Devuelve la posición del siguiente elemento a la posición indicada, en la multilista (o NULO si la posición no tiene siguiente).
Precondición: La posición tiene que ser válida.

AnteriorM (tMultilista, tPosM) → tPosM
Devuelve la posición del anterior elemento a la posición indicada, en la multilista (o NULO si la posición no tiene anterior).
Precondición: La posición tiene que ser válida.

InsertarDatoM (tMultilista, tDatoM) → tMultilista, Boolean
Inserta ordenadamente en la multilista, en base al campo nickUsuario, un nuevo elemento y su lista de amigos asociada. Devuelve falso sólo si no hay memoria suficiente para realizar la operación.

EliminarPosicionM (tMultilista, tPosM) → tMultilista
Borra de la multilista el usuario que está en la posición indicada (así como su lista de amigos asociada).
Precondición: La posición tiene que ser válida.

ObtenerDatoM (tMultilista, tPosM) → tInfoM
Devuelve el dato situado en la posición indicada de la multilista.
Precondición: La posición tiene que ser válida.

ActualizaDatoM(tMultilista, tPosM, tNumDenuncias, tListaOrd)→ tMultilista
Actualiza la información asociada al elemento situado en la posición indicada.
Precondición: La posición tiene que ser válida.

BuscarDatoM (tMultilista, tNickUsuario) → tPosM
Devuelve la posición del elemento con nick tNickUsuario (o NULO si el elemento no existe).