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

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. Un grupo de individuos con mochila decide competir con Spotify e implementar un buscador de música, de tal forma que a partir del nombre artístico de un grupo o solista, nos proporcione toda su información. ¿Qué estructura sería la más adecuada para buscar eficientemente esta información?

   SOLUCIÓN: Un árbol binario de búsqueda AVL con clave el nombre artístico, implementado de forma dinámica. Un AVL ya que es una estructura sobre la cual se pueden realizar eficientemente las operaciones de búsqueda. Su implementación será dinámica ya que desconocemos el número de grupos o artistas existentes.

2. Una editorial desea editar una guía de los distintos vinos españoles, organizada por denominación de origen (Rioja, ribera del Duero, Rías Baixas, etc.). En cada categoría aparecerá una lista de los vinos que poseen dicha denominación. ¿Cuál es la estructura más adecuada para almacenar esta información?

   SOLUCION: Una multilista (lista de listas). El primer nivel de la multilista contendría un elemento por denominación de origen y el segundo nivel un elemento por vino. Si suponemos que hay una cantidad limitada de denominaciones de origen, la lista del primer nivel sería implementada de forma estática (dinámica en caso contrario) y la lista del segundo nivel sería implementada de forma dinámica ya que no conocemos el número de vinos de cada denominación.

3. La salida al mercado de la autobiografía de Belén Esteban ha dado lugar a que una conocida librería tenga un número exagerado de reservas del mismo. ¿Qué estructura sería la más adecuada para almacenar las reservas de los clientes de la librería si se quieren atender por orden de petición?

   SOLUCIÓN: Una cola para servir las reservas en orden de petición, implementada dinámicamente ya que desconocemos el número de reservas que tiene la librería.

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 **implementación** de una operación se debe incluir código que controle el cumplimiento de las precondiciones de su especificación.

   FALSO. Las precondiciones se suponen ciertas cuando se realiza la implementación de una operación.

2. Una cola de prioridad implementada con una única lista ordenada por prioridad siempre tiene que ser implementada de forma estática.

   FALSO. La implementación de la estructura viene determinada por las especificaciones del problema que se quiere resolver utilizando dicha estructura.

3. El TAD Lista puede funcionar como un TAD Pila.

   VERDADERO. Siempre y cuando las inserciones y eliminaciones se realicen por el mismo extremo.

4. Un árbol binario completo con tres niveles contiene como máximo 6 nodos.

   FALSO. Un árbol binario completo con tres niveles contiene entre 4 y 7 nodos. En el caso de tener 7 nodos, también es un árbol binario lleno.

EJERCICIO 3

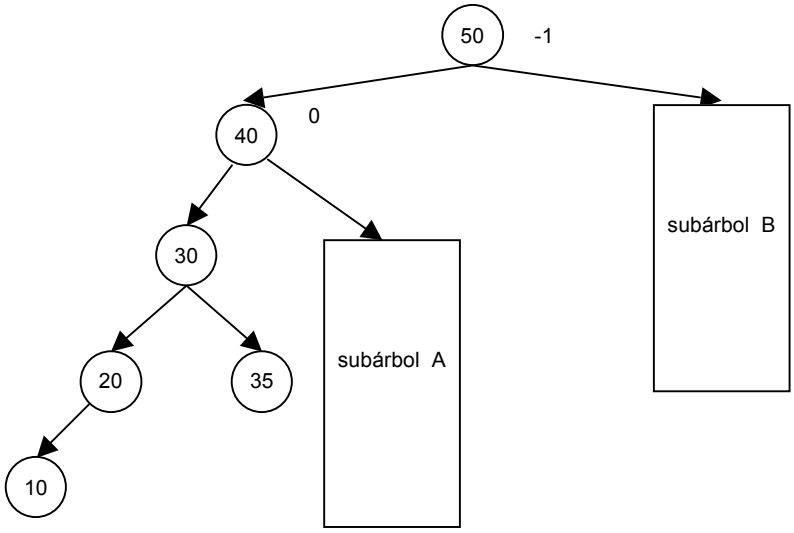
Dado el siguiente código, indicar qué error básico en el manejo de punteros se produce y porqué:

```
Program Error;

type tDato= integer;
     tPos = ^tNodo;
     tNodo= record
        dato: tDato;
         sig: tPos;
     end;
     tColeccion= tPos;

var L: tColeccion;
    P: tPos;
    i: integer;

begin
   new(L);
   L^.dato:=0;
   for i:= 1 to 20 do begin
       new(P);
       P^.dato:= L^.dato+1;
      L^.sig:=P;
   end;
end.
```

SOLUCIÓN: Trata de crear una estructura enlazada de 20 elementos, pero el Nuevo elemento creado lo enlaza siempre como segundo elemento, de tal manera que el final de la ejecución queda una estructura enlazada de 2 elementos y 18 elementos desreferenciados.

EJERCICIO 4

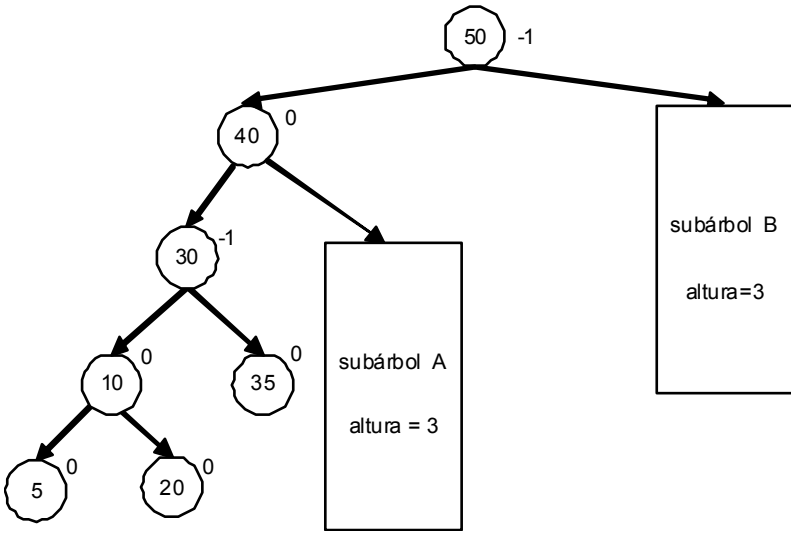
Dado el árbol AVL siguiente, calcular la altura de los subárboles A y B en función de los dos factores de equilibrio indicados. A continuación, insertar la clave 5 y después eliminar la clave 40. Dibujar los sucesivos estados que atraviesa el árbol hasta llegar al estado final, indicando los factores de equilibrio en cada fase, las rotaciones que sea necesario realizar y los 2 ó 3 nodos que están implicados en ellas.



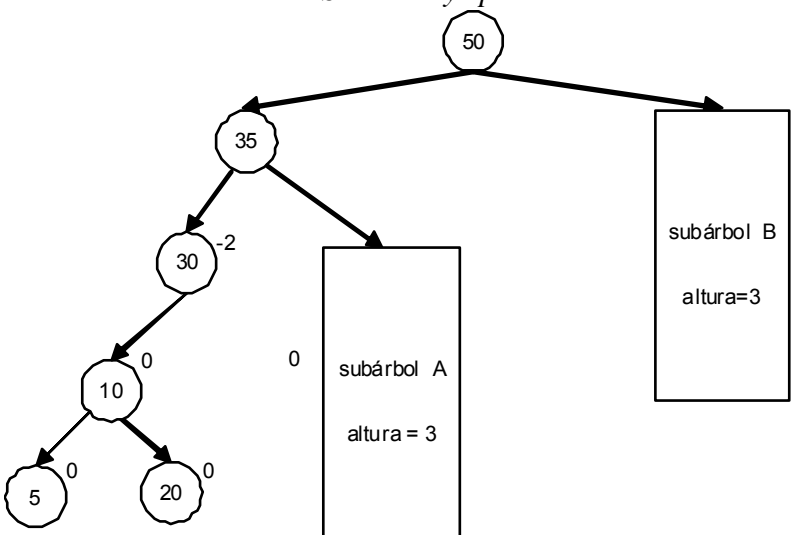
SOLUCION:
Determinamos en primer lugar la altura de los subáboles indicados:

1) *según el FE del nodo 40 la altura del subárbol A = 3*
2) *según el FE del nodo 50 la altura del subárbol B =3*

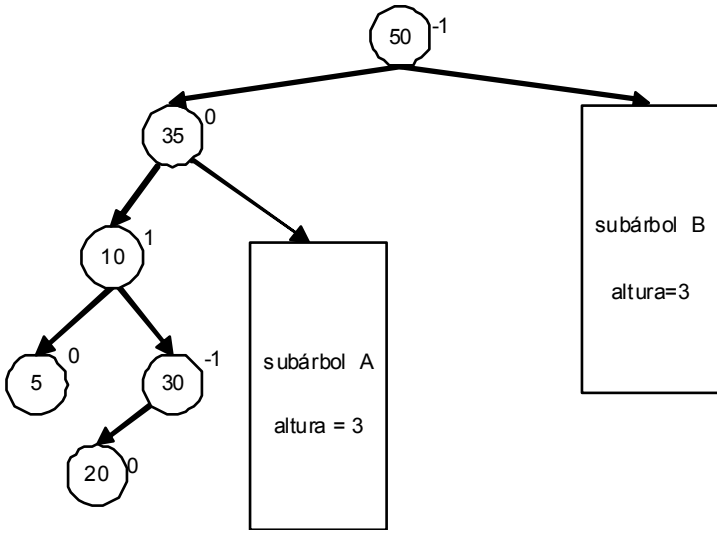
Insertamos la clave 5. Rotación II: implica a los nodos 20-10



Eliminamos la clave 40. Se sustituye por 35.

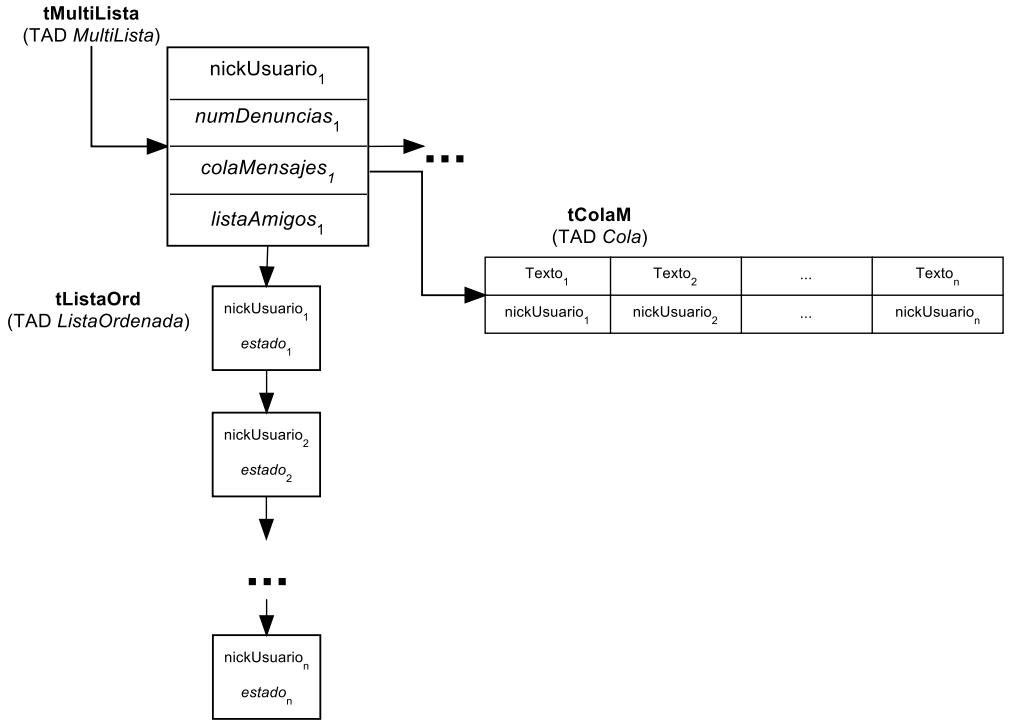


Rotación II (factores -2, 0): implica a los nodos 30-10



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)

Supongamos que la plataforma permite escribir mensajes entre los usuarios, aún cuando no estén conectados. Para ello se utilizaría un TAD Cola por cada usuario en el que se almacenarían un máximo de 10 mensajes en el mismo orden en el que le fueran enviados. En el momento en que el usuario se conectara, se le mostrarían todos los mensajes.

Se pide como ejercicio realizar la definición de tipos de datos de esta nueva estructura:

- tColaM — Representa una cola
- tNickUsuario: — Nick del usuario que envía el mensaje (string)
- tTexto: — Texto del mensaje (string)
- tDatoColaM: — Dato de un elemento de la cola, compuesto por los campos *NickUsuario* y *Texto* que contienen el nick del usuario que envía el mensaje y el texto del mensaje, respectivamente.
- tPosColaM: — Posición de un elemento de la cola.
- NULOC: — Constante que expresa una posición nula.
- MAXC: — Máximo número de mensajes a almacenar.

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

InsertarDatoColaM (tColaM, tDatoColaM)→ tColaM, Boolean
Objetivo: Inserta un elemento en la cola quedando al final
Entrada: Cola a modificar y contenido del elemento a insertar
Salida: La cola con el elemento insertado y verdadero si se ha podido insertar, falso en caso contrario

FrenteColaM (tColaM) → tDatoColaM

Objetivo: Recupera el contenido del primer elemento de la cola
Entrada: La cola donde obtener el dato
Salida: El contenido del primer elemento de la cola
Precondición: La cola no está vacía

```
const
    NULOC=0;
    MAXC=10;
type
    tNickUsuario= string[9];
    tTexto= string;
    tDatoColaM= record
        nickUsuario: tNickUsuario;
        texto: tTexto;
    end;
    tPosColaM: 0..MAXC;
    tColaM= record
        datos: array[1..MAXC] of tDatoColaM;
        ini, fin: tPosColaM;
    end;

function InsertarDatoColaM (var C:tColaM; d: tDatoColaM): boolean;
var
    p: tPosColaM;
begin
    if C.fin = MAXC then
        InsertarDatoColaM:= false
    else begin
        InsertarDatoColaM:= true;
        C.datos[C.fin+1]:= d;
        C.fin:= C.fin + 1:
    end;
end;

function FrenteColaM (C: tColaM): tDatoColaM;
begin
    FrenteColaM:= C.datos[C.ini];
end;
```

B)

Supongamos que los administradores de la plataforma desean recabar información estadística. Implementar una operación

CalcularMediaSolicitudesEnEspera (tMultiLista) → real

que utilizando el TAD Multilista y del TAD ListaOrdenada, obtenga el número medio de solicitudes en espera por parte de los usuarios. Como precondición se establece que la multilista no está vacía. 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.

```
function CalcularMediaSolicitudesEnEspera (M: tMultiLista): real;
var
   posM: tPosM;
   lista: tListaOrd;
   posL: tPosL;
   numenespera, numUsuarios: integer;
begin
   posM:= PrimeraM(M);
   numespera:= 0; numUsuarios:= 0;
   while (posM <> NULOM) do begin
       lista:= ObtenerDatoM (M, posM).listaAmigos;
```

```
      posL:= Primera (lista);
      while posL<>NULOL do begin
          if ObtenerDatoL (lista, posL).estado = enespera then
          numespera:= numespera + 1;
          posL:= Siguiente(lista, posL);
      end;
      numUsuarios:= numUsuarios + 1;
      posM:= SiguienteM (M, posM);
  end;
  CalcularMediaSolicitudesEnEspera:= numenespera/numUsuarios;
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

function ListaVacia: tListaOrd;
Crea una cola vacía.

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) → tDatoM
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).