ASIGNATURA		CURSO	CALIFICACIÓN
ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN		**2007 / 2008**	
TITULACIÓN	GRUPO	CONVOCATORIA	
		EXTRAORDINARIA – SEP	
APELLIDOS		NOMBRE	

EJERCICIO 1 (3 PUNTOS)

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

1. Se trata de modelizar el servicio de petición de obras en una biblioteca. Las peticiones se clasifican a la entrada según el lugar de búsqueda de la obra a la que hacen referencia. Existen 3 lugares de búsqueda: en la propia universidad, en España, en el extranjero. Las peticiones se atienden por cercanía (primero las de la propia universidad, después las del territorio nacional y finalmente, las búsquedas en el extranjero) y, dentro del mismo nivel de búsqueda, por orden de entrada, ¿Qué estructura de datos debería utilizarse para gestionar la petición de obras?

2. Una empresa de revelado fotográfico ofrece un servicio de almacenamiento de fotos limitado a 100 fotos por cliente. Cada cliente podrá almacenar sus fotos y visualizarlas via web de manera secuencial hacia delante y hacia atrás. ¿Qué estructura de datos se debe utilizar para almacenar estas fotos de tal manera que los recorridos sean eficientes?

3. Una gran cadena de centros comerciales quiere mantener toda la información referente a sus empleados organizada en una estructura en la que se distinguen los gerentes provinciales, que existe uno por cada provincia, cada uno de ellos tiene a su cargo varios jefes de sección, que a su vez se encargan de organizar el trabajo de otros empleados. ¿Qué estructura de datos se debe utilizar para gestionar la información de los empleados de esta cadena de centros comerciales si se desea conocer las dependencias entre empleados (cuantos jefes de sección tiene a cargo cada gerente, de cuantos empleados se encarga cada jefe de sección,…)?

4. En una conocida oposición multitudinaria se cuenta con dos bombos de bolas, el primero destinado a los DNI de las personas que se presentan a la oposición y el segundo a los temas que constituyen el temario oficial de dicha oposición. Para almacenar las parejas de DNI y tema correspondiente se busca una estructura de datos que permita encontrar de forma eficiente el tema que le ha sido asignado a un DNI en concreto. ¿Qué estructura de datos se debe utilizar para gestionar estas búsquedas?

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

1. Un montículo es un árbol binario completamente lleno, excepto el último nivel que se llena de derecha a izquierda.

2. Una cola circular es una estructura de datos no lineal.

3. En una lista doblemente enlazada implementada dinámicamente es más simple la operación **`Anterior`** con respecto a una implementación dinámica simple, y es más compleja la operación **`Último`** con respecto a una implementación dinámica circular.

4. En una estructura AVL, el factor de equilibrio debe recalcularse en todos los ascendientes del nodo que se ha borrado.

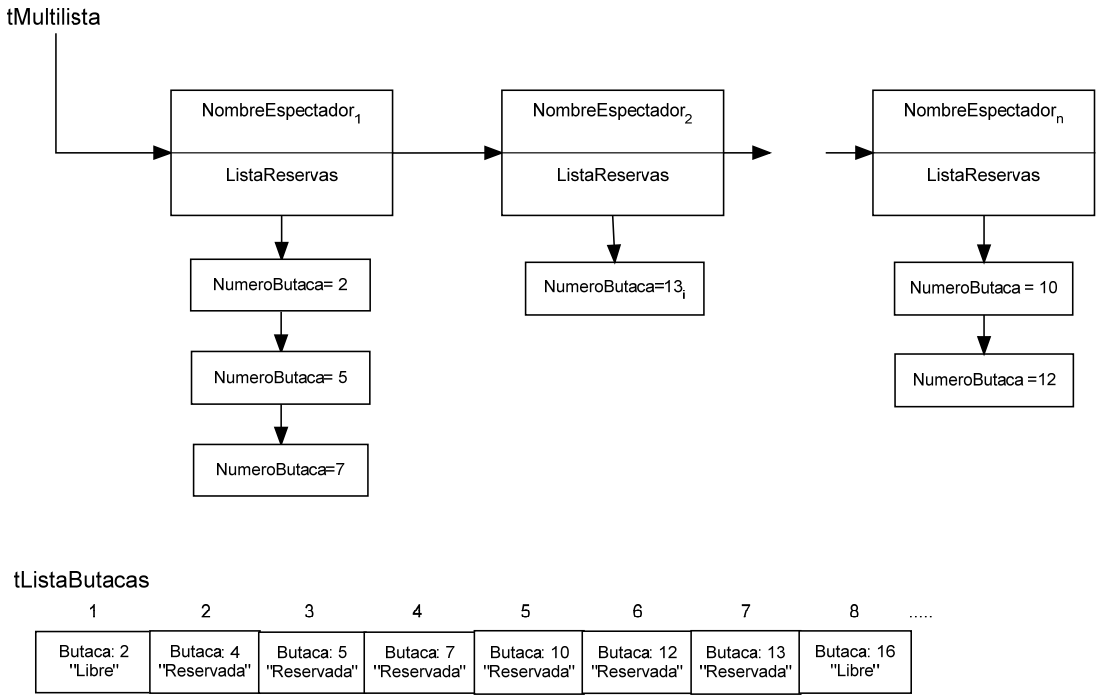
C) El procedimiento EliminarPosicion es una operación perteneciente al **TAD Lista** implementado mediante una **lista dinámica circular**. Su objetivo es eliminar de la lista un nodo con una posición *p*. Explica los errores que presenta el código siguiente respecto a su funcionamiento y/o el cumplimiento de la especificación de la operación (objetivo, entradas, salidas, precondiciones) y muestra el código corregido.

```
Procedure EliminarPosicion(L : tLista; var p:tPos);
{ Objetivo: Eliminar de la lista un nodo con cierta posición
  Entrada : Una lista L; una posición p
  PreCD   : La lista está inicializada;
            p es una posición válida de la lista }
var
   ant : tPos;

begin
   ant:= L;
   while (ant^.sig <> p) do
      ant:= ant^.sig;
   if p=L then
      if L= L^.sig then
         L:= NULO
      else
         L:= p^.sig;
   else
      ant^.sig:= p^.sig;
end;
```

EJERCICIO 2 (4 PUNTOS)

En la práctica 2 se solucionaba el problema de la gestión de butacas y entradas de espectáculos a través de una Multilista compuesta de listas dinámicas de espectadores y sus reservas, y otra lista de butacas. Partiendo del mismo problema planteado en dicha práctica, en este ejercicio se proponen una serie de modificaciones que afectan a estas estructuras y que se reflejan en la figura siguiente:



A) El primer cambio consiste en que el *TAD tListaButacas* se pasa a implementar de manera estática con un máximo de 3000 butacas. Además, deberá estar **ORDENADA de modo creciente** por *número* de butaca. Esta lista solo mantiene las butacas libres o reservadas, ya que se eliminan las que ya han sido pagadas. Se pide como ejercicio realizar la nueva definición de tipos de datos de esta lista y la implementación de operaciones de acuerdo a las siguientes especificaciones:

- NULOB: constante con el valor nulo correspondiente
- tListaB: Tipo de la lista.
- tInfoB: información guardada en cada elemento de la lista: número de butaca (tNumB) y estado (tEstadoB con los valores *libre* o *reservada*).
- tPosB: Posición de un elemento de la lista de reservas.
- **ListaButacasVacia (tListaB) → tListaB**

  Crea una lista vacía.

- **BuscarButaca (tListaB, tNumB) → tPosB**

  Busca en la lista de butacas el elemento que se corresponde con el tNumB especificado y devuelve su posición en la lista, o nulo si no existe.

- **InsertarButaca (tListaB, tNumB, tEstadoB) → tListaB, Boolean**

  Añade de manera ordenada y creciente por tNumB (número de butaca a reservar) un nuevo elemento en la lista de butacas. Devuelve falso si no hay memoria suficiente para realizar la operación o la butaca ya existe en la lista.

B) El segundo cambio afecta al TAD tListaReservas que pasa a guardar números de butacas (en lugar de almacenar posiciones como ocurría en la 2ª práctica de la asignatura). Se pide como ejercicio realizar la nueva definición de tipos de datos de esta lista y la implementación de una operación de acuerdo a las especificaciones:

- NULOR: constante con el valor nulo correspondiente
- tListaR: Tipo de la lista.
- tInfoR: Dato de un elemento de la lista.
- tPosR: Posición de un elemento de la lista de reservas.
- **BorrarReserva (tListaR, tPosR) → tListaR**

  Borra de la lista el elemento que está en la posición indicada.
  PreCD: La posición es válida.

C) Utilizando los TADS tMultilista, tListaReservas y tListaButacas implementar la operación:

CancelarReserva (tMultilista, tListaB, tNomE, tNumB) → tMultilista, tListaB, boolean

Objetivo: Cancelar la reserva del espectador *tNomE* correspondiente a la butaca *tNumB*

Salidas: *tMultilista* con la nueva lista de reservas para el espectador de nombre TNomE. Si su lista de reservas queda vacía, el espectador se borra.
tListaB con el nuevo estado de la butaca (libre)
Booleano indicando si se ha podido o no realizar la cancelación (es necesario comprobar que los datos tNomE y tNumB de la reserva son correctos) .

PreCD: El tNumB es válido de acuerdo al aforo del teatro (3000 butacas).

Para hacer este ejercicio, además de las operaciones de los apartados previos, **supondremos implementado** lo siguiente:

TAD ListaReservas: Mantiene las reservas de un espectador
Operaciones
function EsListaReservasVacia (L: tListaR): boolean {Objetivo: Indica si la lista está o no vacía} **function BuscarReserva (L: tListaR; NumButaca: tNumB): tPosR** {Objetivo: Busca en la lista de reservas el elemento correspondiente al número de butaca tNumB y devuelve su posición en tListaR, o nulo si no existe}

TAD ListaButacas: Mantiene la lista de butacas libres o reservadas pendientes de pago

Operaciones

procedure ObtenerDatoButaca (L: tListaB; P: tPosB; var NumButaca: tNumB; var estado: tEstadoB)
{Objetivo: Devuelve los datos almacenados en la posición indicada de la lista.
PreCD: La posición tiene que ser válida}

procedure ActualizaDatoButaca(var L: tListaB; P: tPosB; estado: tEstadoB)
{Objetivo: Cambia el estado del elemento de la lista que está en la posición P.
PreCD: La posición tiene que ser válida}

TAD Multilista: Mantiene el listado de reservas pendientes de pago

Tipos de datos

NULOM: constante con el valor nulo correspondiente
tMultilista: Representa a una multilista ordenada de simple enlace.
tNomE: Nombre del espectador, criterio de ordenación de la lista.
tListaR: Lista de reservas.
tInfoM: Contenido de un elemento de la multilista, compuesto por dos campos que contienen el nombre del espectador (tNomE) y la lista de sus reservas (tListaR).
tPosM: Posición de un elemento de la multilista.

Operaciones

procedure BorrarM (var L: tMultilista; P: tPosM);
{Objetivo: Borra de la lista el elemento que está en la posición indicada.
PreCD: La posición es válida}

procedure ObtenerDatoM (L: tMultilista; P: tPosM; var nombre: tNomE; var reservas: tListaR);
{Objetivo: Devuelve los datos almacenados en la posición P de la multilista.
PreCD: La posición tiene que ser válida}

procedure ActualizaDatoM(var L: tMultilista; P: tPosM; nombre: tNomE; reservas: tListaR)
{Objetivo: Cambia los datos almacenados (tNomE y tListaR) en la posición P.
PreCD: La posición tiene que ser válida}

function BuscarM (L: tMultilista; nombre: tNomE): tPosM
{Objetivo: Devuelve la posición del elemento con nombre tNomE (o NULO si el elemento no existe)}

EJERCICIO 3 (3 PUNTOS)

A) Dado el tipo abstracto de datos (TAD) **tListaOrdenada** que sirve para representar listas de **enteros ordenadas de modo creciente, con elementos repetidos,** y del que sólo se conoce la siguiente parte de la interfaz

```
type    tInfo= integer;   tPos= ...;  tListaOrdenada = …;

function EsListaVacia (L: tListaOrdenada): boolean;
{Objetivo: Determina si la lista contiene o no elementos}

function primerDato(L:tListaOrdenada): tinfo;
{Objetivo: Devuelve el dato almacenado en la primera posición de L
PreCD: L no vacía}

function ultimoDato(L:tListaOrdenada): tinfo;
{Objetivo: Devuelve el dato almacenado en la última posición de L
PreCD: L no vacía}

function restoDeLista (L:tListaOrdenada): tListaOrdenada
{Objetivo: Devuelve una lista resultado de eliminar el primer
elemento de L
Entrada: Una lista
Salida: Una nueva lista
PreCD: L no vacía}

function ExisteElemento (d: tInfo; L: tListaOrdenada): boolean;
{Objetivo: Determina si la lista contiene o no el elemento d }

procedure Anadir (x:tinfo; var L: tLista);
{Objetivo: Añade un elemento al final de L
PreCD: se supone memoria suficiente }

Procedure Concatenar (var O: tLista; D: tLista);
{Objetivo: Añade al final de la lista O los elementos de la lista D
eliminando los elementos repetidos
PreCD: O y D no vacías y se supone memoria suficiente }
```

Se pide definir una operación **recursiva** que dadas dos listas L1 y L2 devuelva una tercera lista ordenada L3 resultado de la unión de ambas (Unión (L1, L2) → L3) **que no contenga elementos repetidos**. **Precondición**: Se supone memoria suficiente para realizar la operación y L3 es una lista vacía.

B) Dado el tipo abstracto de datos (TAD) **tArbolBin** que sirve para representar árboles binarios, del que **SÓLO** se conoce la siguiente parte de la interfaz:

```
type   tArbolBin = ...
function  EsArbolVacio (a: tArbolBin): boolean;
function  Raiz         (a:tArbolBin): tInfo;     // PreCD: A no vacío
function  HijoIzdo     (a:tArbolBin): tArbolBin; // PreCD: A no vacío
function  HijoDcho     (a:tArbolBin): tArbolBin; // PreCD: A no vacío
```

Implementar una función que cuente el número de nodos del árbol que tengan por lo menos 1 descendiente por la izquierda.

Soluciones

EJERCICIO 1 (3 PUNTOS)

A)

1. SOLUCIÓN: cola de prioridad (con tres prioridades) implementada como lista estática de colas dinámicas
2. SOLUCIÓN: Si hablamos de todos los clientes una lista dinámica enlazada simple donde cada nodo tiene a su vez una lista estática para almacenar las fotos.
3. SOLUCIÓN: Multilista dinámica
4. SOLUCIÓN: AVL dinámico

B)

1. FALSO, el montículo es un caso particular de árbol binario completo, que se define como un árbol lleno hasta el penúltimo nivel y en donde los nodos del último nivel están lo más a la izquierda posible.

2. FALSO. Una cola circular es un tipo particular de implementación del TAD Cola que es una estructura lineal.

3. CIERTO, gracias al doble enlace podemos acceder directamente a la posición anterior a una dada sin tener que recorrer toda la lista hasta dicha posición. Sin embargo, en esta implementación, la operación último requiere recorrer toda la lista mientras que para el caso de una lista circular el último elemento es fácilmente accesible mediante el enlace anterior de la propia lista (cabeza de la lista).

4. CIERTO. La operación de borrado, una vez finalizada, requiere rehacer el camino inverso desde el nodo eliminado recalculando los factores de equilibrio ya que esa es la rama que modifica su altura y puede alterar el factor de equilibrio de los nodos ascendientes. Además es necesario recorrer todo el camino hasta la raíz ya que con el borrado puede producirse más de una rotación.

C)

1. La lista debe ser pasada por referencia
2. La línea **`ant^.sig:=p^.sig;`** no puede ir dentro de un else, hay que hacerla siempre
3. **`L:=p^.sig`** debe modificarse por **`L:=ant`** para que L siga apuntando al último elemento si el que se borra es el último.
4. Falta liberar la posición eliminada (**`dispose p`**)

```
Procedure EliminarPosicion(var L : tLista; var p:tPos);
{ Objetivo: Eliminar de la lista un nodo con cierta posición
  Entrada : Una lista L; una posición p
  PreCD   : La lista está inicializada;
            p es una posición válida de la lista }
var
   ant : tPos;

begin
   ant:=L;
```

```
         while (ant^.sig <> p) do
            ant:=ant^.sig;
         if p=L then
            if L=L^.sig then
               L:=NULO
            else
               L:=ant;
         ant^.sig:=p^.sig;
         dispose(p);
      end;
```

SOLUCIÓN EJERCICIO 2 (4 PUNTOS)

A)

```
const
      NULO = 0;
      MAX=3000;
type
      tNumB = integer;
      tEstadoB = (libre, reservada);
      tInfoB = record
                  numero: tNumB;
                  estado: tEstadoB;
            end;
      tPosB = NULO..MAX;
      tListaB = record
                  info: array [1..MAX] of tInfoB;
                  ultimo: tPosB;
            end;

procedure ListaButacasVacia (var L: tListaB);
begin
      L.ultimo:=0;
end;

function ButacasLibres(L:tListaB):integer;
var cont:integer;
      i: tPosB;
begin
      cont:=1;
      for i:=1 to L.ultimo do
            if L.info[i].estado=libre
            then cont:=cont+1;
      ButacasLibres:=cont;
end;


function BuscarButaca(L:tListaB;nButaca:tNumB):tPosB;
var i:tPosB;
begin
   if L.ultimo=NULO
   then BuscarButaca:= NULO
   else begin
        i:=1;
        while (i<L.ultimo) and (L.info[i].numero < nButaca) do
```

```
            i:=i+1;
        if (L.info[i].numero = nButaca)
        then       BuscarButaca:=i
        else
            BuscarButaca:=NULO;
  end
end;

function EstadoButaca (L:tListaB;nButaca:tNumB): string;
var p:tPosB;
begin
      p:=BuscarButaca(L,nButaca);
      case L.info[p].estado of
            libre: EstadoButaca:='libre';
            reservada: EstadoButaca:='reservada';
            else EstadoButaca:='vendida';
      end;
end;


function InsertarButaca (var L:tListaB; nButaca:tNumB;
                                          estado: tEstadoB):boolean;
var
      pos,pos2:tPosB;
begin
   if L.ultimo<MAX  {no ha alcanzado el maximo de reservas}
   then begin
      {busca la posicion p donde tiene que insertar la butaca}
      L.ultimo:=L.ultimo+1;
      if L.ultimo=0 {vacia}
      then   begin
            L.info[L.ultimo].numero:=nButaca;
            L.info[L.ultimo].estado:=estado;
      end
      else begin
            pos:=1;
            while ((pos <> L.ultimo-1) and (L.info[pos].numero<nButaca)) do
                  pos:=pos+1;
            if (L.info[pos].numero = nButaca)
            then InsertarButaca:=false
            else  if (L.info[pos].numero > nButaca) {insertar en el medio}
                        then begin
                              for pos2:=L.ultimo downto pos+1 do
                                    L.info[pos2]:=L.info[pos2-1];
                              L.info[pos].numero:=nButaca;
                              L.info[pos].estado:=estado;
                        end else begin {insertar al final}
                              L.info[L.ultimo].numero:=nButaca;
                              L.info[L.ultimo].estado:=estado;

                        end
      end;
   end
      else InsertarButaca:=false;{L.ultimo>=MAX}
end;
```

B)

```
const nulo=nil;
```

```
type
      tInfoR= tNumB;
      tPosR = ^tNodo;
      tNodo =  record
                  info:tInfoR;
                  sig: tPosR;
            end;
      tListaR=tPosR;



procedure BorrarReserva(var L: tListaR; p: tPosR);
{PreCD: La posición es  válida.}
var q: tPosR;
begin
      if p=L {borrar primero}
      then L:=L^.sig
      else begin
            {busca la posicion anterior}
            q:=L;
            while (q^.sig <> p) do
                  q:= q^.sig ;
            q^.sig:=p^.sig;
      end;
      dispose(P);
end;
```

C)

```
function CancelarReserva (var M:tMultilista; var LB:tListaB;
                              nombre:tNomE; nButaca:tNumB): boolean;
  var
      posEspectador:tPosM;
      aux: tNomE;
      reservas: tListaR;
      posReserva: tPosR;

  begin
      posEspectador:=BuscarM(M, nombre);
      if posEspectador=NULO
      then CancelarReserva:=false
      else begin
            ObtenerDatoM (M, posEspectador, aux, reservas);
            posReserva:=BuscarReserva (reservas, nButaca);
            if posReserva=NULO
            then CancelarReserva:=false
            else begin
                  BorrarReserva(reservas, posReserva);
                  if  EsListaReservasVacia(reservas)  {si  la  lista  queda
vacía}
                  then BorrarM (M, posEspectador)
                  else ActualizaDatoM(M, posEspectador, nombre, reservas);
                  ActualizaDatoButaca(LB,     BuscarButaca(LB,     nButaca),
libre);
            end;
      end;
  end;
```

SOLUCIÓN EJERCICIO 3 (3 PUNTOS)

A)

```
Procedure Union (L1,L2: tListaOrdenada; var L3: tListaOrdenada);
var
      dato1, dato2: tInfo;
begin
      if not EsListaVacia(L1) and not EsListaVacia(L2) then begin
            dato1:= primerDato (L1); dato2:= primerDato (L2);
            if (dato1 = dato2) then begin
                  if not ExisteElemento(dato1, L3) then
                        Anadir (dato1, L3);
                  Union (restoDeLista (L1), restoDeLista(L2), L3);
            end else if (dato1 < dato2) then begin
                  if not ExisteElemento(dato1, L3) then
                        Anadir (dato1, L3);
                  Union (restoDeLista(L1), L2, L3)
            end else if (dato1 > dato2) then begin
                  if not ExisteElemento(dato2, L3) then
                        Anadir (dato2, L3);
                  Union (L1, restoDeLista(L2), L3)
            end;
      end else if (EsListaVacia(L1) and not EsListaVacia(L2)) then
            Concatena (L3, L2)
      else if (not EsListaVacia(L1) and EsListaVacia(L2)) then
            Concatena (L3, L1);
end;
```

B)

```
Function CuentaAlmenos1Desc (A: tArbolBin): integer;
begin
      if EsArbolVacio (A) then CuentaAlmenos1Desc:= 0
      else if EsArbolVacio (HijoIzq(A)) and EsArbolVacio (HijoDer(A))
      then CuentaAlmenos1Desc:= 0
      else if EsArbolVacio (HijoIzq(A))
      then CuentaAlmenos1Desc:= CuentaAlmenos1Desc (HijoDer(A))
      else CuentaAlmenos1Desc:= 1 + CuentaAlmenos1Desc (HijoIzq(A))
                                  + CuentaAlmenos1Desc (HijoDer(A))
end;
```