ASIGNATURA		CURSO	CALIFICACIÓN
ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN		**2007 / 2008**	
TITULACIÓN	GRUPO	CONVOCATORIA	
		ORDINARIA – JUNIO	
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. La universidad está organizando unos cursos de verano de gran demanda. Para gestionar las peticiones de inscripción, establece que se considerarán primero aquellas que provengan de personal de la propia universidad, después las que provengan de alumnos, y por último las de personas que no pertenezcan a la comunidad universitaria. Dentro de cada grupo, se atenderán las solicitudes por estricto orden de llegada. ¿Qué estructura de datos debería utilizarse para gestionar las peticiones de inscripción?

2. Una editorial con un amplísimo catálogo de obras se propone abrir una tienda en Internet. Se pretende que los usuarios, con sólo introducir el título de una obra, accedan a toda la información de la misma. ¿Qué estructura de datos se debe utilizar para gestionar de forma eficiente estas consultas?

3. El Consejo de Desorganización Universitaria pretende centralizar toda la oferta de titulaciones de las universidades del país. Para ello precisa almacenar los nombres de las universidades, los de los centros que las componen y los nombres de las titulaciones que se imparten en cada centro. ¿Qué estructura de datos se debe utilizar para almacenar esta información, de tal manera que se evite la repetición de datos y que sea eficiente obtener la lista de las titulaciones que se imparten en un determinado centro de una determinada universidad?

4. El departamento de Contabilidad Creativa de una sospechosa empresa ha determinado que para sus oscuros intereses lo más adecuado es establecer una política de gestión de inventario en la cual se considera que los productos que se sirven a sus clientes se corresponden con aquellos que han entrado más recientemente en el almacén. De este modo se minimiza el valor de las existencias en almacén, ya que los productos más antiguos han sido adquiridos a un menor precio. ¿Qué estructura de datos se debe utilizar para gestionar las entradas y salidas del inventario?

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

1. En una cola de prioridades el primer dato que entra es el primero que sale, ya que se sigue una disciplina FIFO.

2. El orden en que se han insertado los datos en un árbol binario de búsqueda afectará a la eficiencia de las operaciones de búsqueda que se realicen sobre él.

3. Los árboles AVL basan su eficiencia en el hecho de que tienen la forma de un árbol completo.

4. La realización de modificaciones en la definición de los operadores de un TAD puede conllevar la realización de un cambio en la implementación del TAD.

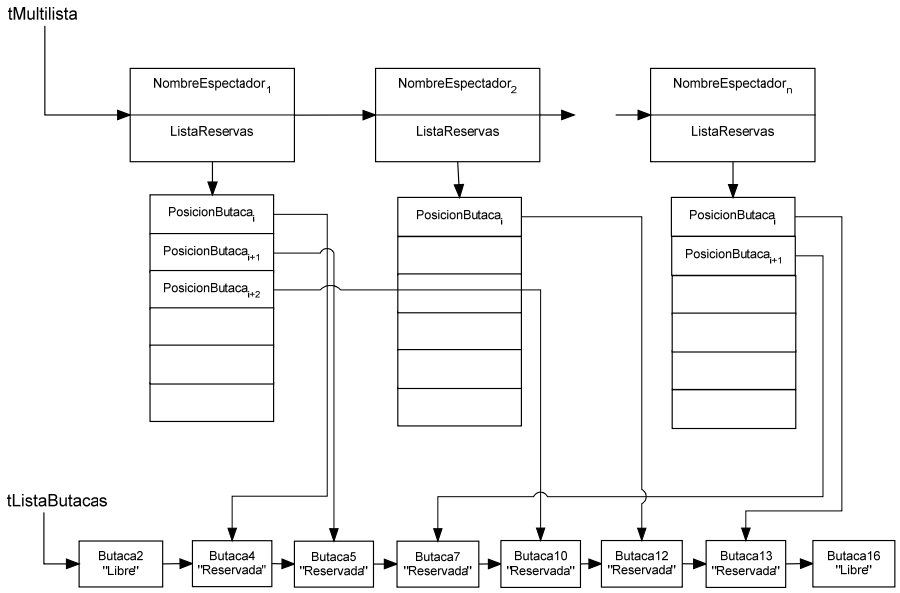
C) La función anterior es una operación perteneciente al **TAD Lista** implementado mediante una **lista dinámica enlazada simple**. Su objetivo es obtener la posición anterior a *p*. Explica los errores que presenta el código siguiente respecto a su funcionamiento y/o el cumplimiento de la especificación de la operación (objetivo, entradas, salidas, precondiciones) y muestra el código corregido.

```
function anterior(L : tLista; p:tPos):tPos;
{ Objetivo: Obtener la posición anterior a p}
  Entrada : Una lista L; una posición p}
  Salida  : La posición anterior a p}
  PreCD   : La lista está inicializada;
            La lista no está vacía;
            p es una posición válida de la lista
}
var
   q,r : tPos;

begin
   new(q);
   q := L^.sig;
   while (q <> p) do begin
      r := q;
      q := q^.sig;
   end; {while}
   anterior := r;
end; { anterior }
```

EJERCICIO 2 (4 PUNTOS)

En la práctica 2 se solucionaba el problema de la gestión de butacas y entradas de espectáculos a través de unas estructuras que se relacionaban de la manera que indica la figura siguiente:



Partiendo del mismo problema planteado en dicha práctica, en este ejercicio se proponen una serie de modificaciones. Para ello **supondremos implementados** los siguientes TADs de los cuales sólo se conoce la parte de la interfaz siguiente:

TAD ListaButacas: Mantiene el mapa de la ocupación del local

Tipos de datos

tListaB: Representa a una lista ordenada por número de butaca.
tNumB: Número de la butaca.
tEstadoB: Estado de la butaca—libre o reservada (tipo enumerado).
tInfoB: Dato de un elemento de la lista, compuesto por los campos **NumB** y **EstadoB** que contienen el número de la butaca y su estado.
tPosB: Posición de un elemento de la lista de butacas.
NULO: Constante utilizada para representar posiciones nulas.

Operaciones

```
procedure ObtenerDatoButaca (L: tListaB; P: tPosB; var NumButaca: tNumB; var estado: tEstadoB)
Devuelve el dato situado en la posición indicada de la lista.
PreCD: La posición tiene que ser válida.
```

procedure ActualizaDatoButaca (var L: tListaB; P: tPosB; estado: tEstadoB)
Actualiza el dato situado en la posición indicada con estado tEstadoB.
PreCD: La posición tiene que ser válida.

Function PrimeroButaca (L: tListaB): tPosB
Devuelve la posición del primer elemento de la lista (o NULO si la lista está vacía).

Function SiguienteButaca (L: tListaB; P: tPosB): tPosB
Devuelve la posición en la lista del siguiente elemento a la posición indicada (o NULO si la posición no tiene siguiente).
PreCD: La posición tiene que ser válida.

TAD Multilista

Tipos de datos

tMultilista: Representa a una multilista ordenada de simple enlace.
tNomE: Nombre del espectador, criterio de ordenación de la lista.
tListaR: Lista de reservas.
tInfoM: Dato de un elemento de la multilista, compuesto por los campos *NombreEspectador* y *ListaReservas*, que contienen el nombre del espectador y la lista de sus reservas respectivamente.
tPosM: Posición de un elemento de la multilista.

Operaciones

procedure InsertarOrdenadoM (var L: tMultilista; nombre: tNomE; Reservas: tListaR);
Inserta en la multilista un nuevo espectador con nombre *tNomE* y su lista de reservas correspondiente.
La multilista quedará ordenada con respecto a este campo.
PreCD: se supone memoria suficiente para realizar la operación.

procedure ObtenerDatoM (L: tMultilista; P: tPosM; var nombre: tNomE; var reservas: tListaR);
Devuelve el dato situado en la posición indicada de la multilista.
PreCD: La posición tiene que ser válida.

procedure ActualizaDatoM (var L: tMultilista; P: tPosM; nombre: tNomE; reservas: tListaR)
Actualiza el dato situado en la posición indicada con nombre *tNomE* y lista de reservas *tListaR*.
PreCD: La posición tiene que ser válida.

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

Se pide:

A) El *TAD ListaReservas* se pasa a implementar de manera estática con un máximo de 6 reservas por espectador. Además, esta lista deberá estar **ORDENADA** por *número* de butaca.

1. Realizar la nueva definición de tipos de datos de esta lista donde:
   - tListaR: Tipo de la lista.
   - tInfoR: Dato de un elemento de la lista, que continúa siendo la posición de una butaca de la lista de butacas (tPosB).
   - tPosR: Posición de un elemento de la lista de reservas.
   - NULO: Constante utilizada para representar posiciones nulas.
2. Implementar las siguientes operaciones:
   - **ListaReservasVacia (tListaR) → tListaR**

     Crea una lista vacía.
   - **InsertarReserva(tListaR, tNumB, tListaB) → tListaR, Boolean**

     Añade de manera ordenada y creciente por *tNumB* (número de butaca a reservar) un nuevo dato de tipo *tInfoR* en la lista. Devuelve un valor falso si no hay memoria suficiente para realizar la operación o la butaca no está libre.
     PreCD: *tNumB* es válido.
   - **BorrarReserva(tListaR, tPosR) → tListaR**

     Borra de la lista el elemento que está en la posición indicada.
     PreCD: La posición tiene que ser válida.
   - **BuscarReserva (tListaR, tNumB, tListaB) → tPosR**

     Busca en la lista de reservas el elemento que se corresponde con el *tNumB* especificado y devuelve su posición en la lista de reservas, o nulo si no existe.
     PreCD: *tNumB* es válido.
   - **NumeroReservas (tListaR) → integer**

     Devuelve el número de elementos de la lista de reservas.

B) Utilizando los TADs anteriores realizar el nuevo procedimiento de reserva:

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

Objetivo: Insertar en la lista de reservas del espectador *tNomE* la butaca correspondiente al número *tNumB* siempre que se cumpla que esta butaca está libre y el espectador no tenga ya reservadas más de 6 butacas.
Salidas: *tMultilista* que incluirá los datos del espectador y su nueva reserva.
tListaB con el nuevo estado de la butaca (reservado)
Booleano indicando si se ha podido o no realizar la reserva.
PreCD: El número de butaca es válido

EJERCICIO 3 (3 PUNTOS)

A) Supongamos que ha sido definido un TAD tTren para almacenar una secuencia de datos. Cada uno de estos datos de almacena en lo que se denomina un "vagón automotor" (este TAD es por tanto una metáfora de un tren con vagones cargados de mercancía). Algunos de los operadores que lo conforman son los siguientes:

```
type
     tTren = ...

function    esTrenVacio (t: tTren): boolean;
{Devuelve cierto si el tren T no contienen ningún vagón}

function     Locomotora (t: tTren): tDato;
{Devuelve el dato almacenado en el primer vagón automotor,
 llamado convencionalmente "locomotora"}

function    RestoDeTren (t: tTren): tTren;
{Elimina el primer vagón automotor del tren T, y devuelve
 lo que resta del tren}
```

Supongamos que tDato es de tipo **real**. Utilizando los operadores mostrados más arriba, defina una función **recursiva** que devuelva como resultado la multiplicación del contenido de todos los vagones de un tren.

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
     tInfo= integer;
     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 sume los valores almacenados en las hojas de un árbol binario.

Soluciones

EJERCICIO 1 (3 PUNTOS)

1A)

1. SOLUCIÓN: cola de prioridad (con tres prioridades: universidad, alumnos, ajenos a la universidad) implementada como lista estática de colas dinámicas, puesto que no se limita el número inscripciones en cada categoría.
2. SOLUCIÓN: AVL dinámico. La clave de búsqueda es el título de la obra. En cada paso de la búsqueda, el conjunto de elementos se divide a la mitad—más o menos porque está equilibrado. La implementación es dinámica puesto que no se menciona para nada el tamaño del catálogo.
3. SOLUCIÓN: Multilista de tres niveles, universidad, centro, titulación. Ya que desconocemos el número de universidades, centros y titulaciones del pais—y es algo que permance abierto—usaríamos implementación estática.
4. SOLUCIÓN: Una pila. Los últimos productos que entran en almacén, son los primeros en ser servidos. La implementación es dinámica por cuanto no sabemos el tamaño del inventario.

1B)

1. FALSO, el orden de salida depende de las prioridades

2. CIERTO. El peor caso es introducir los datos ya ordenados, porque crea un árbol con descendientes sólo por la izquierda (o sólo por la derecha dependiendo del criterio que siga el ABB), en el que el número de comparaciones en el mejor caso es 1 pero en el peor es de n, siendo n el número de elementos almacenados..

3. FALSO, un árbol AVL no tiene por que ser un árbol completo.

4. CIERTO. La implementación de un TAD depende de la especificación.

1C)

ERRORES:

1. Hay un problema si p es la primera posición de la lista
2. el new(q) es incorrecto
3. la variable r es innecesaria pero:
    a. si se usa, no está bien inicializada (valor erróneo si se busca el anterior al 2º elemento de la lista)
    b. si no se usa, la asignación inicial de q es errónea y también la condición del bucle (no encontraría el anterior)

SOLUCIÓN 1

```
function anterior(L : tLista; p:tPos):tPos;
{ objetivo: obtener la posición anterior a p
  entrada : una lista L; una posición p
  salida  : la posición anterior a p
  preCD   : la lista está inicializada
            la lista no está vacía
            p es una posición válida de la lista
            p no es la primera posición de la lista
}

var
   q : tPos;
begin
   q := L;
   while (q^.sig <> p) do
      q := q^.sig;
   anterior := q;
end; { anterior }
```

SOLUCIÓN 2:

```
function anterior(L : tLista; p:tPos):tPos;
{ objetivo: obtener la posición anterior a p
  entrada : una lista L; una posición p
  salida  : la posición anterior a p o nulo si no existe
  preCD   : la lista está inicializada
            la lista no está vacía
            p es una posición válida de la lista
}

var
   q,r : tPos;
begin
   if (p=L) then anterior:= nulo
   else begin
```

```
{bucle con r}
r := L;
q := L^.sig;
while (q <> p) do begin
      r := q;
      q := q^.sig;
end; {while}
anterior := r;
```

```
{o bucle sin r}
q:=l;
while (q^.sig <> p) do
      q:= q^.sig;
anterior:= q;
```

```
end; { anterior }
```

EJERCICIO 2 (4 PUNTOS)

A)

```
const
   MAX = 6;
   NULO = 0;


type
   tPosR = NULO..MAX;
   tInfoR = tPosB;
   tListaR = record
     datos: array [1..MAX] of tInfoR;
     ultimo:tPosR;
   end;


procedure ListaReservasVacia (var L: tListaR);
begin
      L.ultimo:=0;
end;


function EsListaReservasVacia (L: tListaR): boolean;
begin
      EsListaReservasVacia:= (L.ultimo=0);
end;


function Primero(L:tListaR): tPosR;
begin
      primero:= 1;
end;


function NumeroReservas (ListaR: tListaR): integer;
begin
   NumeroReservas:= ListaR.ultimo;
end;


procedure BorrarReserva (var lista: tListaR; pos: tPosR);
var q:tPosR;
begin
   lista.ultimo:=lista.ultimo-1;
   for q:= pos to lista.ultimo do begin
      lista.datos[q].dato:= lista.datos[q+1].dato;
   end;
end;
```

```
procedure  InsertarOrdenado  (var  listaR:  tListaR;  Dato:  tInfoR;
nButaca: tNumB; Lb: tListaB);
var
  pos, pos2: tPosR;
  numero: tNumB;
  estado:tEstadoB;
begin
    if (eslistaReservasVacia(listaR))
    then begin
            listaR.ultimo:= listaR.ultimo+1;
            listaR.datos[listaR.ultimo]:= Dato;
    end else begin
        pos:= 1;
        ObtenerDatoButaca (Lb,listaR.datos[pos], numero, estado);
         while ((pos < listaR.ultimo) and (numero < nButaca)) do begin
             pos:= pos+1;
             ObtenerDatoButaca(Lb, listaR.datos[pos], numero, estado);
         end;
         if (numero > nButaca)
         then begin
           for pos2:= listaR.ultimo downto pos
                       listaR.datos[pos2+1]:=listaR.datos[pos2];
           listaR.datos[pos]:=Dato;
         end else listaR.datos[listaR.ultimo]:= Dato;
         listaR.ultimo:= listaR.ultimo+1;
      end;
end;



function InsertarReserva ( var ListaR: tListaR;
                               nButaca: tNumButaca;
                                    Lb: tListaB): boolean;
var
    estado:tEstadoB;
    numero:NumButaca;
    p:tPosB;
    nuevo:tPosR;
begin
    {si no ha alcanzado el maximo de reservas}
    if ListaR.ultimo<MAX then begin
        {busca la posicion p de la butaca correspondiente al nButaca}

        p:= PrimeroButaca (Lb);
        ObtenerDatoButaca (Lb, p, numero, estado);
        While (numero<>nButaca) do begin
            p:= SiguienteButaca (Lb, p);
            ObtenerDatoButaca (Lb, p, numero, estado);
        end;

        if estado=libre then begin
            InsertarOrdenado (ListaR, p, nButaca, Lb);
            InsertarReserva:= true;
        end else InsertarReserva:= false;
    end else InsertarReserva:= false;
end;
```

```
function BuscarReserva (  ListaR: tListaR; nButaca: tNumButaca;
                              Lb: tListaB): tPosR;
var
   estado:tEstadoB;    numero:NumButaca;       p:tPosB;
begin
   if not EsListaReservasVacia (ListaR) then begin
      p:= 0;
      repeat
         p:= p+1;
         ObtenerDatoButaca (Lb, ListaR.info[p], numero, estado);
      until (numero>=nButaca) or (p = ListaR.ultimo);
      if (numero = nButaca) then
         BuscarReserva:= p
      else
         BuscarReserva:= nulo;
end;
```

B)

```
procedure buscarButaca (    L: tListaB; butaca: tNumB; var p: tPosB;
                              var estado: tEstadoB);
//función auxiliar
var
   NumButaca: tNumB;
begin
   p:= PrimeroButaca (L);
   ObtenerDatoButaca (L,p,NumButaca,estado);
   while (NumButaca <> butaca) do begin // número de butaca es válido
      P:=SiguienteButaca (L, p);
      ObtenerDatoButaca (L, p, NumButaca, estado);
   end;
end;

Function HacerReserva ( var M: tMultilista; var B: tListaB;
                           nombre: tNomE; butaca: tNumB): boolean;
var
       posButaca: tPosB;       estado: tEstadoB;
   posEspectador: tPosM;     reservas: tListaR;
begin
   HacerReserva:= true;
   posEspectador:= BuscarM (M, nombre);
   if posEspectador <> nulo then begin   {existe el espectador}
      ObtenerDatoM (M, posEspectador, nombre, reservas);
      if  InsertarReserva (reservas, butaca, B) then begin
         ActualizaDatoM (M, posEspectador, nombre, reservas);
         ActualizaDatoB (B, posButaca, reservada);
         HacerReserva:= true;
      end else
         HacerReserva:= false;
   end else begin
      ListaReservasVacia (reservas);
      if  InsertarReserva (reservas, butaca, B) then begin
         InsertarOrdenadoM (M, nombre, Reservas);
         ActualizaDatoB (B, posButaca, reservada);
         HacerReserva:= true;
      end
   end;
end;
```

EJERCICIO 3 (3 PUNTOS)

3A)

```
function multiplicación(T:tTren): REAL
begin
  if esTrenVacio(T) then
    multiplicacion := 1
  else multiplicación := locomotora(T)* multiplicacion(restoDeTren(T))
end;
```

otra opción

```
function multiplicación(T:tTren): REAL
begin
  if esTrenVacio(T)
  then multiplicacion := 0
  else if esTrenVacio(restoDeTren(T))
    then multiplicación := locomotora(T)
    else
       multiplicación := locomotora(T)* multiplicacion(restoDeTren(T))
end;
```

3B)

```
function suma(A : tArbolBin): integer;
begin
  if EsArbolVacio(A) then
     suma := 0
  else if EsArbolVacio(HijoIzdo (A)) and EsArbolVacio(HijoDcho(A))
then
     suma := Raiz(A)
  else
     suma := suma(HijoIzdo(A)) + suma(HijoDcho(A))
end;
```