| ASIGNATURA | | CURSO | CALIFICACIÓN |
|---|---|---|---|
| **ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN** | | **2004 / 2005** | |
| 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** (estática o dinámica) para resolver el problema, **justificando** la respuesta:

1. Se trata de una empresa de alquileres de coches que necesita un programa para gestionar sus clientes de manera que permita hacer búsquedas eficientes dado un DNI. El número de clientes es desconocido a priori y muy variable

2. Queremos implementar en un editor de texto básico la posibilidad de aplicar la operación "Deshacer cambio" almacenando los últimos 10 cambios realizados.

3. El INSERSO desea informatizar la gestión de las 2000 plazas disponibles para los viajes que oferta a la tercera edad. Estas plazas se asignan primero a las personas de menor renta y las demás peticiones permanecen en espera. Se sabe que el 90% de las plazas (1800) se cubren ya en la asignación inicial, y que el 10% (200) restante se asignan siempre entre las primeras 400 personas en espera. ¿Qué estructura es adecuada para almacenar la lista de espera?

**B)** Contestar **Verdadero o Falso** a las siguientes preguntas, **justificando** la respuesta:

1. Al eliminar las claves de un montículo min éstas se obtienen en orden descendente.

2. En un árbol AVL se cumple que, para cada nodo, la mitad de sus descendientes están en el subárbol izquierdo y la otra mitad en el derecho.

3. Una cola se comporta como una lista en la que las inserciones se realizan por el final y las eliminaciones por el principio

4. En una lista ordenada, en la que hay inserciones y eliminaciones frecuentes, la mejor implementación es la estática.

**C)** Sea una implementación dinámica del TAD ListaCircular explica los errores que presentaría el código siguiente respecto a su funcionamiento y/o el cumplimiento de la especificación de la operación (objetivo, entradas, salidas y precondiciones):

```
Procecure InsertarFin (var L: tLista; x: tInfo);
{Objetivo:   Insertar un elemento al final de la lista con los datos indicados
 Entrada:    La lista y los datos a insertar
 Salida:     La lista con el elemento x insertado al final
 PreCD:      La lista está inicializada y siempre se puede hacer la inserción}
var
  Nuevo:tPos;
begin
  new(Nuevo);
  Nuevo^.info:=x;
  if not EsVacia(L) then  begin
        Nuevo^.sig:=L^.sig;
        L^.sig:=Nuevo;
     end
  else  begin
        Nuevo^.sig:=Nuevo;
        L:=Nuevo;
     end;
end;
```

**EJERCICIO 2** (4 PUNTOS)

Se trata de modelar un taller de reparaciones de coches. Los clientes pueden elegir a la entrada tres tipos de servicios (1: urgente, 2: normal, 3: ahorro). La diferencia fundamental entre los servicios es la rapidez en la realización de la reparación. Así los coches se atienden por tipo de servicio y, dentro del mismo tipo de servicio, por orden de entrada. Para ello a cada coche se le asocia un tiempo de entrada (supondremos un entero).

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

**Se pide:**

**A)** Dado que el número de coches es ilimitado, pero el de colas es limitado (3 tipos de servicios) por lo que para representar el sistema completo de colas puede optarse por una representación totalmente dinámica o sólo parcialmente. Indicar la opción elegida y justificar su uso.

**B)** Realizar la **definición de tipos** de un TAD para representar la cola correspondiente a un tipo de servicio. Los coches se identificarán por su número de matrícula. Además, escribir la **especificación*** de **TODAS** las operaciones de este TAD e **implementar SÓLO** las operaciones siguientes:

- añadir un coche a la cola
- extraer un coche de la cola.

**C)** Utilizando el TAD del paso anterior, **definición** de tipos de un TAD para representar el sistema. Se deberán **especificar*** **e implementar** las operaciones:

- añadir un coche al sistema
- extraer un coche del sistema
- reasignar coches en colas

* Las especificaciones serán breves: nombre operación, parámetros, objetivo y precondiciones.

**EJERCICIO 3** (3 PUNTOS)

Dado el tipo abstracto de datos (TAD) tArbolBin que sirve para representar árboles binarios de enteros, del que sólo se conoce la parte de la interfaz

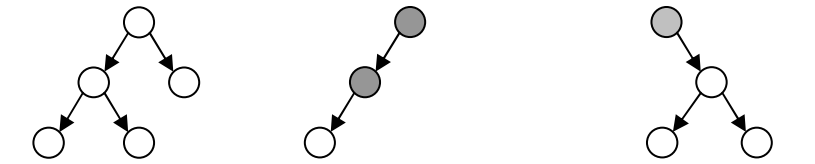
```
type
     tArbolBin = ...

function EsArbolVacio (a:tArbolBin):boolean;
function Raiz (a:tArbolBin):integer;
function RamaIzda (a:tArbolBin):tArbolBin;
function RamaDcha (a:tArbolBin):tArbolBin;
```

**A)** Pretendemos determinar el número de nodos de un **árbol binario de búsqueda** (A) que almacenan valores mayores o iguales a un valor *n*. Para ello hemos creado la siguiente función contarMayoresOIguales, pero no está bien definida. ¿Qué error hemos cometido en la definición de esta función recursiva? ¿Cómo sería la versión correcta de contarMayoresOIguales?

```
function contarMayoresOIguales(A:tABB; n:integer):integer;
begin
  if esArbolVacio(A) then
     contarMayoresOIguales := 0
  else
     contarMayoresOIguales := 1 +
                        contarMayoresOIguales (RamaIzda(A))+
                        contarMayoresOIguales (RamaDcha(A))
end;
```

**B)** Definir una función que dado un árbol <u>no vacío</u> (del cual sólo se conoce la interfaz arriba indicada) obtenga el número de nodos que tienen solamente un hijo. En la imagen se muestran varios ejemplos de árboles y lo que debería devolver la función en cada caso.



nodosUnHijo = 0 | nodosUnHijo = 2 (los sombreados) | nodosUnHijo = 1 (el sombreado)

# Soluciones

## Ejercicio 1:

### A)

1. SOLUCION: AVL dinámico
2. SOLUCIÓN: Pila estática.
3. SOLUCIÓN: Cola estática de tamaño 400

### B)

1. SOLUCIÓN: Falso
2. SOLUCIÓN: Falso
3. SOLUCIÓN: Verdadero
4. SOLUCIÓN: Falso

### C)

SOLUCIÓN: Está insertando al principio, no al final de la lista, es necesario que el último paso del algoritmo sea siempre L:=Nuevo porque L apunta al último elemento en una cola circular.

```
procedure InsertarFin (var L: tLista; x: tInfo);
var
  Nuevo: tPos;
begin
  new(Nuevo);
  Nuevo^.info:=x;
  if not EsVacia(L) then
    begin
      Nuevo^.sig:=L^.sig;
      L^.sig:=Nuevo;
    end
  else
    Nuevo^.sig:=Nuevo;
  L:=Nuevo;
end;
```

## Ejercicio 2:

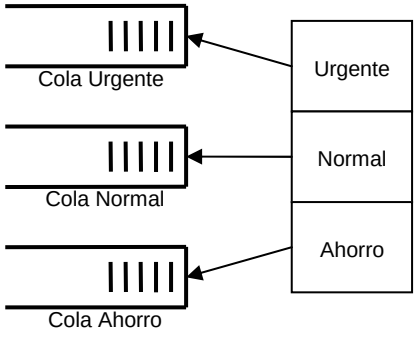
Cola

---

**Definición de tipos:**

```
tTiempo = integer;
tMatricula = string;
tPos = ^tNodo;
tNodo = record
  coche: tMatricula;
  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 coche:tMatricula; Var entrada:tTiempo);
{Objetivo : obtener la informacion del elemento al frente de la cola
Entrada   : C: variable de acceso a  la cola
Salida    : coche: matricula 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; coche: tMatricula; entrada: tTiempo);
{Objetivo : Inserta un nodo (al final de la cola)
Entrada   : coche: matricula del coche
               entrada: tiempo de entrada en la cola
               C: variable de acceso a la cola
Salida    : La Cola con el coche insertado al final
PreCond   : La Cola debe estar inicializada, y existe memoria suficiente}

var
       nuevo: tPos;
begin
       new(nuevo);
       nuevo^.coche:= coche;
       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 coche (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:

```
tTipoServicio = (urgente, normal, ahorro); (* También tipo subrango 1..3 *)
tColaPri = array [tTipoServicio] of tCola;
```

## Especificación e Implementación de operaciones:

```
Procedure AnadirCoche (Var CP: tColaPri; coche: tMatricula;
                       tipoServicio: tTipoServicio; tiempo: tTiempo);
{Objetivo: añadir un coche a la CP clasificándolo según tipoServicio
 Entradas:  Cola de Prioridad,
            coche: matricula del coche
            tipoServicio: Tipo de servicio del coche
            tiempo: Tiempo del sistema
 Salidas: La cola de prioridad con el coche en una de las colas
 Precondiciones: No existe ningún coche con esa misma matricula y la CP
                 está inicializada, así como cada una de las colas}

Begin
  AnadirC (CP[tipoServicio], coche, Tiempo)
end;
```

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

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

## Ejercicio 3:

### A)

SOLUCIÓN:

```
  No estamos utilizando adecuadamente las propiedades de
  un árbol binario de búsqueda.

  function contarMayores(A:tABB; n:integer):integer;
  begin
    if esArbolVacio(A) then
       contarMayores := 0
      else if n=Raiz(A) then
             contarMayores := 1 + contarMayores(ramaDcha(A))
           else if n > Raiz(A) then
                  contarMayores := contarMayores(ramaDcha(A))
                else contarMayores := 1 + contarMayores(RamaIzda(A))
                                        + contarMayores(RamaDcha(A))
  end;
```

### B)

SOLUCIÓN:

```
  function hijosUnicos(A:tArbolBin):integer;
  begin
    if esArbolVacio(ramaIzda(A)) AND  esArbolVacio(ramaDcha(A)) then
       hijosUnicos := 0
    else if esArbolVacio(ramaIzda(A)) then
           hijosUnicos := 1+ hijosUnicos(ramaDcha(A))
         else if esArbolVacio(ramaDcha(A)) then
                hijosUnicos := 1+ hijosUnicos(ramaIzda(A))
              else hijosUnicos:= hijosUnicos(ramaIzda(A)) +
                                 hijosUnicos(ramaDcha(A))
  end;
```