ASIGNATURA		CURSO	CALIFICACIÓN
ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN		**2005 / 2006**	
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. El Ministerio de Hacienda quiere gestionar el sistema de devolución rápida de manera que se ingresen primero las cantidades menores y, en caso de cantidades iguales, se respete el orden de presentación de las declaraciones de la Renta. ¿Cuál sería, en este caso, la estructura más apropiada para almacenar el NIF, cantidad a devolver y datos bancarios de los contribuyentes?

2. Se desea utilizar una estructura para almacenar expresiones aritméticas de cualquier longitud. ¿Cuál sería la estructura más eficiente para poder obtener rápidamente la representación infija, prefija y posfija de la expresión almacenada?

3. Queremos desarrollar un servicio que permita realizar consultas a una guía telefónica vía internet, de manera que, a partir de los apellidos y el nombre de una persona se suministre al usuario de la guía el número de teléfono correspondiente. ¿Cuál es la estructura más adecuada para guardar los datos de este guía?

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 de búsqueda equilibrado.

2. La estructura más eficiente para resolver un problema de búsqueda es una lista ordenada.

3. Una lista de colas se comporta SIEMPRE como una cola de prioridades.

4. Si insertamos una secuencia ORDENADA de claves en un AVL se obtiene una estructura equivalente a una lista ordenada.

C) La función LocalizarElemento se integra dentro del TAD listaOrdenada implementado mediante una lista dinámica. Su objetivo es realizar una búsqueda de un elemento por su valor (tInfo) y devolver la posición del mismo (tPos). Explica los errores que presenta el código siguiente respecto a su funcionamiento, su eficiencia y/o el cumplimiento de la especificación de la operación (objetivo, entradas, salidas y precondiciones). Muestra el código corregido.

```
function LocalizarElemento (L: tLista; Valor: tInfo): tPos;
{ Objetivo:   Localiza en la lista la posición de la primera
        ocurrencia del valor indicado
   Entrada:   Una lista ordenada ascendentemente y
              el valor a localizar
    Salida:   La posición del valor en la lista o nulo si no
        existe
     PreCD:   La lista está inicializada }
begin
      while (L^.info<>Valor) and (L<>NULO) do
          L:=L^.sig;
      if L^.info=Valor
          then LocalizarElemento:=L
          else LocalizarElemento:=NULO
end;
```

EJERCICIO 2 (4 PUNTOS)

Un servidor de DNS (Servidor de Nombres del Dominio) inverso almacena convenientemente pares de la forma (dirección IP, lista de alias). Cada dirección IP es de la forma "aaa.bbb.ccc.ddd" y tiene asociada un conjunto de alias que representan los distintos nombres de dominio asociados a dicha dirección IP. El servidor mantiene estos pares ordenados de forma ascendente por las direcciones IP, y el número máximo de alias y de direcciones no es conocido a priori.

Para gestionar las direcciones IP y los alias correspondientes se utiliza una multilista (*TAD DNSInverso*). Esta estructura está compuesta por una única lista **ordenada** de direcciones IP, y para cada dirección, una lista de los posibles alias. Utilizaremos un Tipo Abstracto de Datos Lista, llamado *TAD ListaAlias*, para gestionar la lista de alias. Para simplificar el problema puede suponerse que dos direcciones IP pueden compararse con los operadores "<", "=" y ">".

Se pide:

A) Realizar en lenguaje Pascal la definición de tipos del *TAD ListaAlias* y el *TAD DNSInverso.* Determinar la MEJOR estructura de datos y la MEJOR implementación para resolver el problema planteado (para ser considerada correcta, la respuesta deberá estar **justificada**):

B) **Suponiendo implementadas** las siguientes operaciones para el *TAD ListaAlias*, donde **Alias** es de tipo *tAlias* y **Lista** de tipo ***tListaAlias*** es la lista de alias,

Function ListaVacia: tListaAlias;
{Objetivo: Crear una lista vacía
Salida: Una lista vacía}

Function EsListaVacía (Lista: tListaAlias): Boolean;
{Objetivo: Determinar si una lista está vacía
Entrada: Lista
Salida: true si la lista está vacía, false en caso contrario.
PreCond: La Lista debe estar inicializada}

Procedure Insertar (var Lista: tListaAlias; alias: tAlias);
{Objetivo: Añadir a la lista un alias de una dirección (se permiten repetidos)
Entrada: Lista y alias
Salida: Lista con un nuevo alias
PreCond: La lista debe estar inicializada, y se supone memoria suficiente}

Procedure BorrarLista (var Lista: tListaAlias);
{Objetivo: Elimina todos los elementos de la lista
Entrada: Lista a eliminar
Salida: Lista vacia
PreCond: La lista debe estar inicializada}

Realizar la implementación de las operaciones especificadas a continuación para el *TAD DNSInverso*, usando la definición de tipos del apartado **A**.

CreaDNSInverso () → tDNSInverso
{Objetivo: Crear un servidor DNSInverso vacío
Salida: Un servidor DNSInverso vacío}

EsDNSInversoVacio (tDNSInverso) → Boolean
{Objetivo: Determinar si un servidor DNSInverso está vacío
Entrada: DNSInverso
Salida: true si el servidor está vacío, false en caso contrario.
PreCond: La variable tDNSInverso está inicializada }

Existe (tDNSInverso, tDirecciónIP) → boolean
{Objetivo: Indica si la DirecciónIP está en en el servidor DNSInverso
Entrada: DNSInverso y dirección IP a buscar
Salida: true si la dirección está en el servidor, false en caso contrario
PreCond: La variable tDNSInverso está inicializada }

Encontrar (tDNSInverso, tDirecciónIP) → tListaAlias
{Objetivo: Devuelve la lista de alias asociados a la dirección IP
Entrada: DNSInverso y dirección IP a encontrar
Salida: Lista de alias asociados
PreCond: La variable tDNSInverso está inicializada, y la dirección existe en el servidor}

InsertarEnDNSInverso (tDNSInverso, tDirecciónIP, tAlias) → DNSInverso
{Objetivo: Añadir al servidor DNSInverso de forma ordenada por direcciónIP un par dirección IP–alias
Entrada: DNSInverso, dirección IP y alias
Salida: DNSInverso con un nuevo alias para la dirección IP si ésta ya existía, o un nuevo par si la dirección IP no existía en el servidor
PreCond: La variable tDNSInverso está inicializada, y hay memoria suficiente}

BorrarEnDNSInverso (tDNSInverso, tDireccionIP) → tDNSInverso
{Objetivo: Elimina la dirección IP del servidor y la lista de alias asociados
Entrada: DNSInverso y dirección IP a buscar
Salida: El servidor sin la dirección IP
PreCond: La variable tDNSInverso está inicializada y la dirección existe en el servidor}

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) Dada la siguiente operación de la cual no tenemos ninguna documentación, determinar: 1) ¿cuál es su objetivo?; y 2) Si eliminamos las líneas de código 5, 6 y 7, ¿cuál es ahora su cometido?

```
1  function operacion (A: tArbolBin): integer;
2  begin
3    if esArbolVacio(A)
4    then operacion:= 0
5    else
6        if EsArbolVacio(ramaIzda(A)) and EsArbolVacio(ramaDrcha(A))
7        then operacion:= 0
8        else operacion:= 1 + operacion(ramaIzqda(A) +
                            operacion(ramaDcha(A));
9  end;
```

B) Escribir una función que devuelva el número de nodos de un árbol binario que son hijos izquierdo.

Soluciones

EJERCICIO 1 (3 PUNTOS)

1A)

1. SOLUCIÓN: Cola de prioridad implementada como lista ordenada dinámica.
2. SOLUCIÓN: Árbol binario de expresión dinámico.
3. SOLUCIÓN: AVL dinámico.

1B)

1. SOLUCIÓN: Falso. Está equilibrado pero no es ABB.
2. SOLUCIÓN: Falso. AVL
3. SOLUCIÓN: Falso. Depende de cómo se comporten las operaciones de inserción y eliminación de la estructura global.
4. SOLUCIÓN: Falso. El reequilibrado nunca permitiría que todos los nodos estuviesen almacenados en una única rama.

1C) SOLUCIÓN:

Existen 3 errores en el código:

(1) No se comprueba si la lista es vacía (y por la precondición es posible que eso ocurra). ALTERNATIVA: Existen tres soluciones: 1) Añadir la precondición "Lista no vacía"; 2) Comprobar al principio que la lista no esté vacía; o 3) alterar el orden de condiciones en `while (L<>NULO) and (L^.info<>Valor)`.

(2) Por una cuestión de eficiencia, en la primera condición del while el "< >" tiene que cambiarse por un "<". Como la lista está ordenada, podemos estar seguros de que el elemento no está en la misma si encontramos un elemento mayor, evitando así recorridos innecesarios.

(3) Tal como está planteada la condición del bucle, al final del mismo L podría tomar el valor nulo si el elemento que buscamos es mayor que el último almacenado en la lista. En este caso, la sentencia if posterior daría un error al intentar acceder a L^.info. Existen dos soluciones: 1) La búsqueda llega como mucho hasta el último elemento y no a la posición NULO, o 2) Se modifica la condición del if a `if (L<>nulo) and (L^.info=Valor)`. En este último caso es fundamental mantener el orden indicado entre las dos condiciones.

Nota: La modificación de L es perfectamente válida al estar pasada como valor.

Teniendo en cuenta que existen varias posibles soluciones, una de ellas sería:

```
function LocalizarElemento (L: tLista; Valor: tInfo): tPos;
begin
    if EsVacia(L) then // Solución Error (1)
       LocalizarElemento:=NULO
    else
       begin
          // Buscamos hasta encontrar
          while (L^.info<Valor) and (L^.sig<>NULO) do    // un valor mayor o igual = Solución Error(2)
             L:=L^.sig;                                  // o hasta llegar al último elemento = Solución Error(3)
          if L^.info=Valor // Comprobamos por qué ha salido del bucle
             then LocalizarElemento:=L
             else LocalizarElemento:=NULO
       end
end;
```

EJERCICIO 2 (4 PUNTOS)

A) La implementación más adecuada, tanto para la lista que mantiene los nombres como para la que mantiene los valores, es la dinámica ya que, en ningún caso el número máximo de valores o nombre es conocido ni estable.

```
UNIT TADListaAlias;
INTERFACE
const
        nuloListaAlias = nil;
type
        tAlias = string;
        tPosAlias = ^tNodoAlias;
        tNodoAlias = record
                        valor: tAlias;
                        sig: tPosAlias;
                end;
        tListaAlias = tPosAlias;
```

y

```
UNIT TADDNSInverso;
INTERFACE
        USES TADListaAlias;
        const
                nuloDNS=nil;
        type
                tDireccionIP = string;
                tPosDNS  = ^tNodoDNS;
                tNodoDNS = record
                                IP: tDireccionIP;
                                aliases: tListaAlias;
                                sig: tPosDNS;
                        end;
                tDNSInverso = tPosDNS;
```

B)

```
Procedure CreaDNSInverso (var DNS: tDNSInverso);
begin
        DNS:=nulo;
end;

function EsDNSInversoVacio (DNS: tDNSInverso): Boolean;
Begin
          EsDNSInversoVacio:=(DNS=nulo);
End;

Function Existe (DNS: tDNSInverso; IP: tDireccionIP): boolean;
begin
        if EsDNSInversoVacio (DNS)
        then Existe:=false
        else begin
                while (DNS ^. IP < IP) and (DNS ^.sig <> nulo) do
                        DNS:= DNS ^.sig;
                Existe:= (DNS ^. IP = IP);
             end
end;
```

```
Function Encontrar (DNS: tDNSInverso; IP: tDireccionIP): TADListaAlias;
begin
        while (DNS ^. IP <> IP) do
                DNS:= DNS ^.sig;
        Encontrar:= DNS^.aliases;
end;
```

Para implementar la operación de Inserción desarrollamos dos operaciones auxiliares:

```
Procedure Crear_nodo (var nuevo: tPosDNS; IP: tDireccionIP; alias: tAlias);
begin
        new(nuevo);
        nuevo ^.IP:=IP;
        nuevo ^.sig:=nulo;
        nuevo^.aliases:= ListaVacia;
        Insertar (nuevo^.aliases, alias);
end;

function Pos_inser (n: tDireccionIP; DNS: tDNSInverso): tPosDNS  ;
{ Recorre la lista buscando la posición adecuada para insertar un alias.
        Si el DNS ya existe, devuelve la posición que ocupa para añadir el alias
        Si no existe y hay que insertarlo al principio devuelve nulo
        En otro caso, devuelve la posición del nodo después del que habrá que insertar el
        nuevo DNS
PreCD: tDNSInversono vacío }
var anterior: tPosDNS;
begin
        anterior:= nulo;
        while (n > DNS^.IP) and (DNS ^.sig <>nulo) do
        begin
                anterior:= DNS;
                DNS:= DNS ^.sig;
        end;
        if      n >= DNS ^. IP      {Hay que insertar al final o el elemento ya existe}
        then    anterior:= DNS;
        Pos_inser:=anterior;
end;

procedure InsertarEnDNSInverso (var DNS: tDNSInverso; IP: tDireccionIP; alias: tAlias);
var pos, nuevo: tPosDNS;
begin
        if EsDNSVacio(DNS)
        then Crear_nodo (DNS,IP,alias);
        else begin
                pos:= Pos_inser (IP, DNS);
                if pos =nulo then    {no existe y se añade como primer nodo}
                begin
                        Crear_nodo (nuevo,IP,alias);
                        nuevo^.sig:= DNS;
                        DNS:=nuevo;
                end
                else    if pos ^.IP <> IP
                        then begin   {no existe ese IP y se añade en el medio o al final}
                                Crear_nodo (nuevo,IP,alias);
                                nuevo^.sig:= pos ^.sig;
                                pos ^.sig:=nuevo;
                        end
                        else {el IP ya existe en la lista, solo hay que insertar el valor}
                                Insertar (pos^.aliases, alias);
        end
end;
```

```
procedure BorrarEnDNSInverso (var DNS: tDNSInverso; IP: tDireccionIP);
Var     pos: tPosDNS;
begin
        pos:=DNS;
        if IP=DNS^.IP       {si hay que eliminar el primero}
        then begin
                DNS:= DNS^.sig
                BorrarLista(pos^.aliases);
                dispose (pos);
        end else begin                          {si hay que eliminar al medio o al final}
                while (pos^.sig^. IP <> IP) do
                        pos:= pos^.sig;
                {al salir del bucle pos^.sig  es el nodo a eliminar}
                pos^.sig:= pos^.sig^.sig;
                BorrarLista(pos^.sig^.aliases);
                dispose(pos^.sig)
        end; {else}
end;
```

EJERCICIO 3 (3 PUNTOS)

3A) Calcula el número de nodos internos. Eliminando las líneas 6 y 7, calcula el número total de nodos (internos + hojas)

```
1  function operacion (A: tArbolBin): integer;
2  begin
3    if esArbolVacio(A)
4    then operacion:= 0
5    else
6        if EsArbolVacio(ramaIzda(A)) and EsArbolVacio(ramaDrcha(A))
7        then operacion:= 0
8        else operacion:= 1 + operacion(ramaIzqda(A) +
                             operacion(ramaDcha(A));
9  end;
```

3B)

```
function hijosIzquierdo (Arbol: tArbolBin): integer;
begin
   if EsArbolVacio(Arbol)
   then hijosIzquierdo:= 0
   else
     if EsArbolVacio(RamaIzqda(Arbol))
     then hijosIzquierdo:= hijosIzquierdo (RamaDrcha(Arbol))
     else hijosIzquierdo:= 1 +  hijosIzquierdo (RamaIzqda(Arbol)) +
                                hijosIzquierdo (RamaDrcha(Arbol))
end;
```