ASIGNATURA		CURSO	CALIFICACIÓN
ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN		**2007 / 2008**	
TITULACIÓN (EI – ETIS - ETIX)	GRUPO (A – B – C – D – E – F)	CONVOCATORIA **EXTRAORDINARIA – DICIEMBRE**	
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. En un corrupto país sudamericano, su presidente ha convocado un referéndum sobre cambios constitucionales que piensa utilizar para conocer los nombres de los opositores a su régimen. Para ello un sistema informático registrará en una estructura de datos la entrada de cada votante al colegio electoral y seguidamente el votante elegirá de forma anónima en la pantalla táctil de una máquina la papeleta que desea. La actividad de esta máquina queda registrada en otra estructura de datos. Para satisfacer sus perversas intenciones, el presidente necesitará poder correlacionar los datos del registro de entrada con el del registro de actividad de la máquina. ¿Cuáles son las estructuras más adecuadas para poder establecer esta correlación y de qué tipo (estáticas/dinámicas)? Se recuerda que siempre hay un porcentaje importante de abstención.

   SOLUCION: Dos colas dinámicas, puesto que en el mismo orden en que entran, en el mismo orden ejercen el voto. En una cola se almacenan los datos de la persona y en la otra el voto. Por tanto, al ir sacando elementos de ambas estructuras, establecemos la correlación.

2. Una biblioteca desea facilitar la elección de libros a sus usuarios. Para ello, además de proporcionar un listado alfabético de las obras, proporcionará otro ordenado de acuerdo con la popularidad de cada libro (medida como el número de préstamos). ¿Qué estructura de datos podemos utilizar para gestionar ambos listados?

   SOLUCION: Lista ordenada multienlazada (por título y por número de préstamos) y dinámica puesto que desconocemos el número de ejemplares que vamos a manejar

3. Las empresas de telefonía establecen distintas tarifas en función de la operadora del teléfono al que se llama. Para gestionar el gasto telefónico, el Ministerio de Sanidad y Consumo ha previsto la creación de una base de datos de números telefónicos de España. A través de una página web o mediante un SMS, un usuario podrá conocer la operadora de telefonía (móvil o fija) a la que pertenece el número al que desea llamar. ¿Qué estructura de datos necesitamos para gestionar eficientemente las consultas?

   SOLUCIÓN: AVL dinámico. La clave para organizar el árbol es el número de teléfono. Con un árbol AVL aseguramos el mínimo número de comparaciones para localizar un número de teléfono. Será dinámico puesto que ningún dato del enunciado limita los elementos.

4. Una empresa va a crear una cafetería de autoservicio en la que los usuarios interactuarán a través de una pantalla táctil. En esa pantalla se presentará un producto de la cafetería y el usuario podrá elegirlo pulsando sobre él, avanzar a otro producto pulsando sobre una flecha a la derecha o retroceder a un producto ya visto pulsando sobre una fecha a la izquierda. ¿Qué estructura de datos precisamos para almacenar los productos ofertados por la cafetería?

   SOLUCIÓN: Lista doblemente enlazada dinámica. Será dinámica puesto que ningún dato del problema limita el número de platos que el restaurante mantiene en su menú. Con la lista doblemente enlazada podremos efectuar desplazamientos al producto anterior o al siguiente sin coste computacional.

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

1. En la especificación de un operador de un TAD debe indicarse el algoritmo que se va a utilizar.

   SOLUCIÓN: FALSO. La especificación sólo indica nombre de la operación, parámetros de entrada y/o salida y comportamiento (objetivo, precondiciones y poscondiciones) La implementación es un paso posterior a la especificación.

2. En una cola de prioridad, en un mismo momento, no puede haber dos elementos con la misma prioridad.

   SOLUCIÓN: FALSO. Es posible que tengan la misma prioridad aún cuando uno sea atendido antes que al otro por la propia definición de cola.

3. En un AVL, una inserción puede obligar a realizar entre 0 y h rotaciones, donde h es la altura del árbol.

   SOLUCIÓN: FALSO. Después de realizar la primera rotación el árbol queda reequilibrado. Es tras el proceso de eliminación cuando la reestructuración podría alcanzar a la raíz del árbol, y por tanto, realizar entre 0 y h rotaciones.

4. En un árbol binario, cada nodo puede tener 0, 1 ó 2 hijos.

   SOLUCIÓN: VERDADERO. La condición de árbol binario fija en 2 el número máximo de descendientes directos de n nodo pero nada dice del número mínimo.

C) El procedimiento **insertarN** es una operación perteneciente al *TAD Lista* implementado mediante una lista dinámica **CIRCULAR DINÁMICA ENLAZADA SIMPLE**. Su objetivo es insertar un dato en la posición *enésima* de la lista. 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 y precondiciones) y muestra el código corregido.

```
procedure insertarN(Var L: tLista; d: tDato; n: integer);
{Objetivo:  Insertar el dato d en la posición que resulte de moverse de un
            un elemento a otro de la lista L tantas veces como indique n,
            comenzando desde el principio de la lista.
            Si n es menor que la longitud de la lista, el dato se
            insertará como elemento enesimo de la lista.
            Si la lista está vacía, lo inserta como único elemento.
Entrada:    una lista L
Salida:     la lista con el dato d insertado en la posición que indica n
PreCD:      la lista está inicializada, n > 0
}

procedure insertarN(VAR L : tLista; d: tDato; n: integer);
var p,q,r : tPos;
    i: integer;
begin
   if (L=nil) then
     begin
       new(q);
       q^.dato := d;
       L := q;
       L^.sig := L;
     end
   else begin
     new(p);
     p := L;
     for i := 1 to n do
       p := p^.sig
     q^.sig := p^.sig;
     p^.sig := q;
     if (q^.sig = L) then
        L:= q^.sig;
   end;

end; { insertar }
```

SOLUCIÓN:

1. *El bucle for debe terminar en n-1 para no contar un elemento de más (porque p es realmente la posición del nodo anterior al que se inserta). Así si n=1 el dato se inserta en cabeza.*

2. *Sobra new(p) y hay que sustituirlo por "new(q); q^.dato := d;"o bien ubicar esta sentencia antes del if para que pueda usarse tanto en la parte if como en la parte else.*

3. *Las dos últimas líneas sobran (no hacen nada)*

```
procedure insertarN(VAR L: tLista; d: tDato; n: integer);
var p,q,r : tPos;
    i: integer;
begin
   new(q);
   q^.dato := d;
   if (L=nil) then {insertar en lista vacia}
     begin
       L := q;
       L^.sig := L;
     end
   else begin
     p := L;
     for i := 1 to n-1 do
       p := p^.sig
     q^.sig := p^.sig;
     p^.sig := q;
   end;
end; { insertar }
```

EJERCICIO 2 (4 PUNTOS)

Un grupo de alumnos desean implementar un sencillo mecanismo de codificación de mensajes. El método es simple y consiste en recorrer el texto de inicio a fin. Si se encuentra una vocal, ésta se pasa directamente al texto codificado de salida. Cualquier otra secuencia consecutiva de caracteres no vocales (consonantes, números, signos de puntuación, etc.) se sustituye en el texto de salida por la misma secuencia pero invertida.

Por ejemplo, el texto de entrada "*Mensaje 1: Stop mision α.*" se transforma del modo siguiente:

M	E	N	S	A	J	E		1	:		S	t	o	p		m	i	s	i	o	n		α	.
↓	↓	↓	↓	↓	↓	↓	↓			↓			↓	↓	↓		↓	↓	↓	↓		↓			
M	E	S	N	A	J	E	t	S		:	1		o	m		p	i	s	i	o	.	α		n

Se pide:

A) La estructura para almacenar tanto el texto original como el resultado codificado será una cola, es decir, cada elemento de la cola será un carácter del mensaje. Realizar, utilizando el lenguaje Pascal:

1. La **definición de tipos** de un *TAD Cola* para el problema. Justifica la elección de la implementación (estática o dinámica).

   SOLUCIÓN: La elección es dinámica puesto que no está limitado el tamaño del mensaje.

```
const nulo = nil;
type
tinfo = char;
tPos   = ^tNodo;
      tNodo = record
            Info: tinfo;
            sig: tPos  ;
      end;
      tCola = record
            Ini, Fin: tPos;
      end;
```

2. La **implementación** de las operaciones básicas definidas en la especificación del tipo Cola, donde t*Info* representa el tipo de información que se almacena en la estructura:

ColaVacía () → t*Cola*
{Objetivo: Crear una cola vacía}

MeterCola (t*Info*, t*Cola*) → t*Cola*
{Objetivo: Insertar un nodo con cierta información en la cola
Entrada: t*Info*: elemento a insertar
t*Cola*: Estructura donde insertar
Salida: t*Cola:* Cola con el elemento insertado
PreCond: se supone memoria suficiente para realizar la inserción}

SacarCola (t*Cola*) → t*Info, tCola*
{Objetivo: Eliminar un elemento de la cola
Entrada: *tCola*: Estructura donde eliminar
Salida: t*Info*: elemento eliminado
t*Cola*: Cola sin el elemento eliminado.
PreCond: La *Cola* no está vacía}

EsColaVacia (t*Cola*) → *Boolean*
{Objetivo: Determinar si una cola está vacía}

```
Procedure ColaVacía (var Cola: tCola);
begin
   Cola.Ini:=nulo;
   Cola.Fin:=nulo;
end;


function EsColaVacia (Cola: tCola): boolean;
begin
   EsColaVacia:= Cola.Ini = nulo;
end;


procedure CrearNodo (x: tinfo; var nuevo: tPos );
{Precond: se supone memoria suficiente para crear la
variable}
begin
   new(nuevo);
   nuevo^.info:=x;
   nuevo^.sig:=nulo;
end;


procedure MeterCola (x:tinfo; var Cola: tCola );
var nuevo: tPos ;
begin
   CrearNodo (x, nuevo);
   if EsColaVacia(Cola) then
     Cola.Ini:= nuevo
   else Cola.Fin^.sig:= nuevo;
   Cola.Fin:= nuevo;
end;
```

```
procedure SacarCola (var x:tInfo; var Cola: tCola );
var
      aux: tPos
begin
      x:= Cola.Ini^.info;
   aux:= Cola.Ini;
   Cola.Ini:= Cola.Ini^.sig;
   If Cola.Ini:=nulo
then Cola.Fin:=nulo; {si la cola se queda vacia la
inicializo}
   dispose(aux);
end;
```

B) Consideremos ya implementado un *TAD Pila (tInfo* es del mismo tipo que el del TAD Cola) y del que sólo se conoce la parte de la interfaz siguiente:

Procedure PilaVacía *(var Pila: tPila): tPila;*
{Objetivo: Crear una pila vacía
Salida: La *Pila* inicializada }

Procedure MeterPila (*Info: tInfo; Pila: tPila*);
{Objetivo: Añadir un nodo con cierta información a la Pila
Entrada: *Info*: elemento a insertar
Pila: Estructura donde insertar
Salida : *Pila:* Pila con el elemento añadido
PreCond: Se supone memoria suficiente para realizar la inserción }

Procedure SacarPila (*var Pila: tPila; var Info: tInfo);*
{Objetivo: Extraer un elemento de la pila
Entrada: *Pila*: Estructura donde extraer
Salida: *Info*: elemento extraido de la *Pila*
La *Pila* sin el elemento extraído
PreCond: La *Pila* no debe estar vacía}

Function EsPilaVacia (*Pila: tPila): Boolean;*
{Objetivo: determinar si una pila está vacía
Entrada: *Pila:* Estructura a comprobar
Salida: Verdadero si está vacía, falso en caso contrario}

Utilizando el TAD Cola y el TAD Pila anteriores, escribir una rutina que codifique un mensaje de acuerdo a la especificación siguiente:

MensajeSecreto (Cola1) → Cola2
{Objetivo: obtener un mensaje codificado
Entrada: Cola1: cola que contiene el mensaje de entrada
Salida: Cola2: cola que contiene el mensaje de salida (codificado)}

```
Function EsVocal (c:char): boolean;
begin
    EsVocal:= c in ['a','e','I','o','u']
end;

Procedure Invertir (var Entrada: tPila; var Salida: tCola);
Var
    letra: char;
begin
    while not EsPilaVacia(Entrada) do begin
          SacarPila(letra, Entrada);
          MeterCola (Salida, letra);
    end
end;

Procedure MensajeSecreto (Entrada: tCola; var Salida: tCola);
var
    letra: char;
    Pila: tPila;
begin
    ColaVacia(Salida);
    PilaVacia(Pila);
    while not EsColaVacia(Entrada) do begin
          SacarCola(letra, Entrada);
          if EsVocal(letra) then begin
                Invertir(Pila, Salida);
                MeterCola (Salida, letra)
          end else MeterPila (Pila, letra);
    end;
    Invertir(Pila, Salida);
end;
```

EJERCICIO 3 (3 PUNTOS)

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

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

        function EsListaVacia (L: tListaOrdenada): boolean;

        function primerDato(L:tListaOrdenada): tinfo;
        // devuelve el dato almacenado en la primera posición de L
        // PreCD: L no vacía

        function restoDeLista (L:tListaOrdenada): tListaOrdenada
        // devuelve una lista que es como L sin su primer elemento
        // PreCD: L no vacía

        procedure Anadir (x:tinfo; var L: tLista);
        // Añade un elemento al final de una lista.
        PreCD: 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 intersección de L1 con L2. **Precondición**: Se supone memoria suficiente para realizar la operación y L3 está inicializada a lista vacía

Intersección (L1, L2) → L3

```
Procedure Interseccion (L1,L2: tListaOrdenada; var L3: tListaOrdenada);
var cont1, cont2: tinfo;
begin
 if not EsListaVacia(L1) and not EsListaVacia(L2)
 then begin
      cont1:= primerDato (L1);
      cont2:= primerDato (L2);
      if cont1=cont2 then begin
            Anadir (cont1, L3);
            Interseccion (restoDeLista (L1), restoDeLista(L2), L3)
      end else if cont1<cont2 then
            Interseccion(restoDeLista(L1), L2, L3)
      // cont1>cont2
      else Interseccion (L1, restoDeLista (L2), L2), L3)
 end
end;
```

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

```
type
        tinfo=integer;
        tArbolBin = ...;
        function EsArbolVacio (a: tArbolBin): boolean;
        function RamaIzqda (a: tArbolBin): tArbolBin;
        function RamaDcha (a: tArbolBin): tArbolBin;
        function Raiz (a: tArbolBin): tinfo;
```

Determinar: 1) ¿cuál es el objetivo de la siguiente operación? y 2) ¿Qué ocurriría si eliminamos las líneas 12 a 15? SOLUCIÓN: *El objetivo es devolver el menor entero almacenado en un árbol. Si eliminamos las líneas 12 a 15 daría un error de ejecución si una de las ramas fuese vacía.*

```
1. Function Fx (a:tArbolBin):integer;
2. //PreCD: A es un árbol no vacío
3.      Function min (x,y:integer):integer;
4.      begin
5.            if x<y
6.            then min := x
7.            else min := y
8.      end;
9. begin
10.     if EsArbolVacio(RamaIzqda(a)) and EsArbolVacio(RamaDcha(a))
11.     then    Fx:= Raiz(a)
12.     else    if EsArbolVacio(RamaIzqda(a))
13.           then Fx:= min(Raiz(a), Fx (RamaDcha(a))
14.           else     if EsArbolVacio(RamaDcha(a))
15.                    then Fx:= min(Raiz(a), Fx (RamaIzqda(a))
16.                    else Fx:= min(Raiz(a), min(Fx (RamaIzqda(a)), Fx (RamaDcha(a))))
17. end;
```