ASIGNATURA		CURSO	CALIFICACIÓN
ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN		**2005 / 2006**	
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. Queremos desarrollar un programa capaz de encontrar un camino en un laberinto que se almacena como un conjunto de posiciones que pueden estar libres (representan caminos) o no (representan paredes). El programa deberá elegir un camino y, en caso de que no le lleve a la salida, poder retornar a la última bifurcación para continuar por una dirección diferente. ¿Cuál es la estructura de datos más adecuada para ir almacenando el camino recorrido?

2. Queremos desarrollar un programa de juegos con barajas de 40 cartas. Una de las operaciones que tendrá que poder realizar es barajar dichas cartas antes de cualquier partida. ¿Qué estructura de datos es la que permite almacenar las cartas de forma que la operación de barajar resulte eficiente?

3. Se quiere automatizar la gestión de los pacientes que están en lista de espera para un servicio médico, de manera que se distribuyan por 3 niveles de gravedad y se atienda siempre al paciente más grave que lleva más tiempo esperando. ¿Qué estructura de datos deberemos utilizar para almacenar los datos de los pacientes?

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

1. La operación *Ultimo(tlista)->tPos* es más eficiente en la implementación estática de listas que en la dinámica circular.

2. Una de las aplicaciones de los montículos es la ordenación de secuencias numéricas.

3. Si insertamos una secuencia *ordenada* de claves en un Árbol Binario de Búsqueda se obtiene una estructura equivalente a una lista ordenada.

4. Una lista en la que las inserciones y eliminaciones se realizan siempre por el mismo extremo equivale a una pila.

C) La siguiente función realiza un recorrido en anchura de un árbol binario. Para ello hace uso del TADArbolBin (cuyo campo tInfo es un char) y del TADCola (cuyo campo tInfo es un tArbolBin). Explica los errores que presenta el código siguiente respecto a la utilización del TAD, su funcionamiento y/o el cumplimiento de la especificación de la operación (objetivo, entradas, salidas y precondiciones). Muestra el código **corregido**.

```
function RecAnchura(A:tArbolBin): string;
{ Objetivo:   Devolver un string con el recorrido en anchura
              de un árbol binario
  Entrada:    El árbol a recorrer
  Salida:     Un string con el recorrido en anchura del árbol o
              un string vacío si el árbol está vacío
    PreCD:    El árbol está correctamente inicializado }
var
    C:tCola;
    Aux:tArbolBin;
    s: string;
begin
    s := '';
    if not EsArbolVacio(A) then
      begin
          C.Frente:=NULO_C;
          C.Fin:=NULO_C;
          InsertarCola(C, A);
          while not EsColaVacia(C) do
            begin
                Aux:= Frente(C); Eliminar(C);
                s := s + Raiz(Aux) + '-';
                InsertarCola(C, HijoIzquierdo(Aux));
                InsertarCola(C, HijoDerecho(Aux));
            end;
      end;
end;
```

EJERCICIO 2 (4 PUNTOS)

El mecanismo de cifrado de César es una técnica simple de codificación de mensajes que se basa en desplazar cada letra de un mensaje una cantidad constante (*k*) de posiciones (si k=3 entonces la A se sustituiría por una D y una B se sustituiría por una E). Podemos realizar una mejora a esta técnica de codificación si utilizamos una *clave repetitiva*. En lugar de desplazar cada carácter una cantidad constante, podemos desplazar los distintos caracteres en diferentes cantidades, utilizando una lista de valores clave. Si el mensaje es más largo que la lista de valores clave, simplemente volveremos a comenzar con la lista de claves desde el principio.

Por ejemplo, si el mensaje que queremos enviar es “Caen listas circulares” y la clave es “KlaatuBaradaNikto” la codificación del mensaje sería como sigue:

Mensaje	C	A	E	N		L	I	S	T	A	S		C	I	R	C	U	L	A	R	E	S
Clave	K	L	A	A	T	U	B	A	R	A	D	A	N	I	K	T	O	K	L	A	A	T
Codificación	N	M	F	O	4	a	K	T	f	B	W	!	Q	R	]	W	d	W	M	S	F	g

donde cada letra de la clave indica el desplazamiento que hay que hacer, así la A implica desplazar UNA posición la letra del mensaje, la B DOS posiciones, etc. El mensaje codificado sería “NMFO4aKTfBW!QR]WjWMSFg” (por ejemplo, una E en el mensaje y una A en la clave resulta en una F en la codificación)

Hemos tenido en cuenta las siguientes simplificaciones:

- El mensaje sólo incluye caracteres imprimibles del ASCII estándar.

- La clave está en mayúsculas y sólo incluye caracteres entre la A y la Z del ASCII estándar.
- El mensaje codificado puede salirse fuera del rango A-Z. En el ejemplo anterior podemos ver como una S (ascii=83) en el mensaje desplazada T veces (20) da como resultado una "g" (ascii=83+20=103).

Se pide:

A) **Definición de tipos** de una implementación **dinámica** de una lista circular e **implementación de las siguientes funciones**:

ListaVacia → tLista
Objetivo: Inicializa una lista a vacío

EsVacia (tLista) → Boolean
Objetivo: Determina si una lista es vacía o no

InsertarFin (tLista, tInfo) → Lista, Boolean
Objetivo: Inserta un elemento al final de la lista
Salidas: La lista con el elemento insertado al final y el valor True si se ha podido realizar la inserción o False en caso contrario

DevolverContenido (tLista, tPos) → tInfo
Objetivo: Devuelve el valor apuntado por una posición
PreCD: La posición es válida

Primero (tLista) → tPos
Objetivo: Devuelve la posición del primer elemento de la lista
PreCD: La lista no está vacía

Ultimo (tLista) → tPos
Objetivo: Devuelve la posición del último elemento de la lista
PreCD: La lista no está vacía

Siguiente (tLista, tPos) → tPos
Objetivo: Dada una posición nos devuelve la siguiente de la lista
Salidas: La posición siguiente a la indicada
PreCD: La posición es válida

EliminarPosicion (tLista, tPos) → tLista
Objetivo: Borra el elemento de la lista apuntado por Posición
Salidas: La lista sin el elemento de la posición indicada
PreCD: La posición es válida

B) Codificar la siguiente función, en la cual se deberá usar el TAD creado en el apartado anterior con el fin de almacenar la clave en una **lista circular**:

Codificador (Mensaje: String; Clave: String; Operación: Boolean) → String
Objetivo: Dado un mensaje y una clave los codifica siguiendo el método de clave repetitiva descrito
Entradas: Un string conteniendo el mensaje, otro conteniendo la clave y un valor booleano que si es true indica que hay que hacer una codificación y si es false una decodificación.
Salidas: Un string con el mensaje codificado/decodificado
PreCD: El mensaje y la clave están en mayúsculas y no están vacíos, la clave no contiene caracteres fuera del rango A-Z

Nota: Recordar que son de utilidad la función Ord(Char) que dado un carácter devuelve su código ASCII y la función Chr(integer) que dado un código ASCII devuelve el carácter a que pertenece.

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 del 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 nivel de la hoja más profunda. Para ello hemos creado la siguiente función `hojaMasProfunda`, pero podría no estar bien construida. ¿Qué error, si existe, hemos cometido en la definición de esta función recursiva? ¿Cómo sería la versión correcta de `hojaMasProfunda`?

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

B) Escribir la declaración dinámica de tipos del TAD tArbolBin para manejar enteros, y codificar una nueva operación `EliminarArbol` dentro del TAD, que toma un árbol binario como entrada y elimina todos los nodos del árbol, obteniendo un árbol vacío. Incluir los comentarios pertinentes en cuanto a objetivo, entradas, salidas, precondiciones y poscondiciones de la operación.

Soluciones

EJERCICIO 1 (3 PUNTOS)

1A)

1. SOLUCIÓN: pila, dinámica si queremos permitir cualquier tamaño de tablero.
2. SOLUCIÓN: Lista estática por memoria y eficiencia en las operaciones de intercambio.
3. SOLUCIÓN: Cola de prioridad implementada como una lista estática de 3 elementos y colas dinámicas.

1B)

1. SOLUCIÓN: Falso. Es igual.
2. SOLUCIÓN: Verdadero. HeapSort.
3. SOLUCIÓN: Verdadero. Tendría sólo ramas izquierdas (claves en orden decreciente) o derechas (creciente).
4. SOLUCIÓN: Verdadero. Tiene un comportamiento FIFO.

1C)

SOLUCIÓN:

Existen 3 errores en el código:

(1) Al inicializar la cola se accede a la implementación de la misma.
(2) No deberían meterse árboles vacíos en la cola de nodos (podría dar errores al intentar acceder a la raíz o a sus hijos).
(3) La función no devuelve el string calculado.

```
function RecAnchura(A:tArbolBin): string;
var
   C:tCola;
   Aux:tArbolBin;
   s: string;
begin
   s:='';
   if not EsArbolVacio(A) then
      begin
         C:=ColaVacia; // Solución error (1)
         InsertarCola(C, A);
         while not EsColaVacia(C) do
            begin
               // Sacamos un elemento de la cola y lo imprimimos
               Aux:=EliminarCola(C);
               s := s + IntToStr(Raiz(Aux)) + '-';

               // Introducimos a sus hijos en la cola. Solución error (2)
               if not EsArbolVacio (Aux^.Izq) then
                  InsertarCola(C, HijoIzquierdo(Aux));
               if not EsArbolVacio (Aux^.Dch) then
                  InsertarCola(C, HijoDerecho(Aux));
            end;
      end;
   RecAnchura:=s; // Solución error (3)
end;
```

EJERCICIO 2 (4 PUNTOS)

A)

```
unit ListaCircular;

interface

const
  NULO = nil;

type
  tInfo = integer;
  tLista = ^tNodo;
  tPos = tLista;
  tNodo = record
            info: tInfo;
            sig: tPos;
          end;

function ListaVacia: tLista;
function EsVacia (L: tLista): boolean;
function InsertarFin (var L: tLista; Valor: tInfo):boolean;
function DevolverContenido (L: tLista; P: tPos): tInfo;
function Primero (L:tLista): tPos;
function Ultimo (L:tLista): tPos;
function Siguiente (L:tLista; P: tPos): tPos;
procedure EliminarPosicion (var L: tLista; P: tPos);


implementation

{************************************************************************}

function ListaVacia: tLista;
begin
  ListaVacia:=NULO;
end;

{************************************************************************}

function EsVacia (L: tLista): boolean; // NO CAMBIA
begin
  EsVacia:=(L=NULO)
end;

{************************************************************************}

function InsertarFin (var L: tLista; Valor: tInfo):boolean;
var
  Nuevo: tPos;
begin
  new(Nuevo);
  if Nuevo<>NULO then
    begin
      Nuevo^.info:=Valor;
      if not EsVacia(L) then
        begin // L apunta al ultimo elemento de la lista cuando hay mas de 1 elemento
          Nuevo^.sig:=L^.sig;
          L^.sig:=Nuevo;
        end
      else
        // L apunta al 1er elemento solo cuando hay un elemento
        Nuevo^.sig:=Nuevo;
      L:=Nuevo; // L apunta al nuevo elemento (que es el ultimo de la lista)
      InsertarFin:=true;
    end
  else
    InsertarFin:=false;
end;

{************************************************************************}

function DevolverContenido (L: tLista; P: tPos): tInfo;
begin
  DevolverContenido:=P^.info;
```

```
end;

{**************************************************************************}

function Primero (L:tLista): tPos;
begin
  Primero:=L^.sig;
end;

{**************************************************************************}

function Ultimo (L:tLista): tPos;
begin
  Ultimo:=L;
end;

{**************************************************************************}

function Siguiente (L:tLista; P: tPos): tPos;
begin
  Siguiente:=P^.sig
end;

{**************************************************************************}

procedure EliminarPosicion (var L: tLista; P: tPos);
var
  ant, aux:tPos;
begin
  if not (EsVacia(L)) then  // Si la lista es vacia no hay nada que hacer
    begin
      // Buscamos el anterior sabiendo que P existe
      ant:=L;
      aux:=L^.sig;
      while (aux<>P) do
        begin
          ant:=aux;
          aux:=aux^.sig;
        end;
      // Comprobamos por quÚ nos hemos salido
      if (aux=L) then // CASO ESPECIAL: Si es el ultimo hay que actualizar L
        begin
          if (L=L^.sig) then
            L:=NULO // CASO ESPECIAL: si solo hay un elemento L vale NULO
          else
            begin
              L:=ant; // Si borramos el ultimo el nuevo ultimo es el anterior
              ant^.sig:=aux^.sig;
            end;
          end
        else // CASO COMUN: borramos un elemento que no es L
          ant^.sig:=aux^.sig;

      // Para todos los casos liberamos la memoria de aux
      dispose(aux);
    end;
end;

{**************************************************************************}

end.
```

B)

```
program Cifrado;

uses
  ListaCircular;

var
  s:string;


function Codificador(Mensaje: String; Clave: String; Operacion: Boolean): String;
var
  L: tLista;
  msg: string;
  i,k,ascii: integer;
  aux:tPos;
  c:char;
begin
  msg:='';


  // Metemos la clave en la lista circular;
  L:=ListaVacia;
  for i:=1 to length(Clave) do
    InsertarFin(L, Ord(Clave[i]));

  // (De)codificamos el mensaje
  aux:=Primero(L);
  for i:=1 to length (Mensaje) do
    begin
      ascii:=DevolverContenido(L,aux);
      k:= ascii - Ord('A') + 1;
      if (Operacion)
        then c:=chr(Ord(Mensaje[i])+k)
        else c:=char(Ord(Mensaje[i])-k);
      msg:=msg + c;
      aux:=Siguiente(L,aux);
    end;

  // Liberamos la memoria reservada por la cola
  while (not EsVacia(L)) do
    EliminarPosicion(L, Primero(L));

  // Devolvemos el mensaje (de)codificado
  Codificador:=msg;
end;

begin
  s:= Codificador('CAEN LISTAS CIRCULARES','KLAATUBARADANIKTO',True);
  writeln('El mensaje cifrado es: ', s);
  s:= Codificador(s,'KLAATUBARADANIKTO',False);
  writeln('El mensaje descifrado es: ', s);
end.
```

EJERCICIO 3 (3 PUNTOS)

3A)

SOLUCION:

añadir casos base

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

3B)

SOLUCIÓN:

```
Const
     Nulo = nil;
type
     tPos = ^tNodo;
     tNodo = record
             info: integer;
             izq, der: tPos;
     end;
     tArbolbin = tPos;


Procedure EliminarArbol (var Arbol: tArbolBin);
{Objetivo: eliminar todos los nodos de un árbol binario
Entradas: Árbol binario
Salidas: Un árbol vacío
Precd: El árbol se supone inicializado}

begin
     if not EsArbolVacio(Arbol)
     then begin
             EliminarArbol(RamaIzqda(Arbol));
             EliminarArbol(RamaDrcha(Arbol));
             dispose (Arbol);
             Arbol:= nulo;
     end;
end;
```