UNIVERSIDAD PONTIFICIA DE SALAMANCA EN MADRID
Facultad de Informática
Escuela Universitaria de Informática
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software

Cuadernillo de examen

ASIGNATURA:	**Fundamentos de Programación II**	CÓDIGO:	**113**
CONVOCATORIA:	**Junio de 2007**	PLAN DE ESTUDIOS:	**2000 / 2002**
ESPECIALIDAD:		CURSO:	**1º**
TURNO:	**Mañana**	CURSO ACADÉMICO:	**2006/2007**
CARÁCTER:	**Cuatrimestral (2º cuatrimestre)**	PROGRAMA:	**Ingeniería en Informática / Ingeniería Técnica en Informática**

DURACIÓN APROXIMADA: **2 horas y media**

Solución propuesta

Preguntas teórico-prácticas

1. Recursividad. Partes de un programa recursivo. Tipos de recursividad. Ventajas e inconvenientes de un algoritmo recursivo frente a uno iterativo

Capítulo 14 del libro de texto y apuntes de clase

Aplicación
Diseñe una función de **búsqueda secuencial recursiva** que devuelva la posición de un elemento de tipo entero que se pasará como parámetro dentro de un array de números enteros.
Realice un seguimiento de la función para los números 13 y 8 dentro de un array formado por los números 2, 13, 25 4 y 6.

```
entero función Buscar(valor entero: a[]; valor entero : e; valor entero:n)
inicio
   si n = 0 entonces //Si el array tiene 0 elementos, el elemento no está
       devolver(0)
   si_no
       si a[n] = e entonces //El elemento está en la posición n
         devolver(n)
       si_no
         //Llamada recursiva
         devolver(Buscar(a,e,n-1)
       fin_si
   fin_si
fin_función
```

Puntuación: 1,5 puntos

2. Archivos indexados. Características de los archivos indexados. ¿Cómo se realiza un acceso directo dentro de un archivo indexado? ¿Y un acceso secuencial?

Apartado 9.13 del libro de texto y apuntes de clase

Aplicación
En un archivo indexado se tienen almacenadas una serie de personas. El archivo tiene información sobre el DNI de la persona, su nombre y su dirección. El campo clave es el DNI y el índice tiene capacidad para almacenar hasta 250 personas.
Diseñe las estructuras de datos necesarias para gestionar el archivo indexado. Diseñe un procedimiento que elimine una persona del archivo. El DNI de la persona a eliminar se pasará como argumento al procedimiento.

```
const
   MaxReg = 300
   NumPersonas = 200
tipos
   registro = rPersona
      cadena: DNI,nombre,dirección
      entero : estado
   fin_registro
   archivo_d de rPersona = aPersona
   registro = RIndice
```

```
      entero : clave
      entero : NRR
   fin_registro
   array[0..NumPersonas] de RIndice = vIndice

procedimiento Borrar(ref aPersona:A; valor cadena : DNI; ref vIndice:Ind;
                     ref entero:n)
var
   entero : p,i
inicio
   p ← buscar(Ind,DNI,n)
   si p = 0 entonces
      //No está
   si_no
      //Dar una baja lógica en el área de datos
     leer(A,Ind[p].NRR,R)
      R.estado ← 2
      escribir(A,Ind[p].NRR,R)
      //Eliminar del índice
      desde i ← p hasta n-1 hacer
          Ind[i] ← Ind[i+1]
      fin_desde
      n ← n - 1
   fin_si
fin_procedimiento

entero función Buscar(valor vIndice:v; valor cadena:el; valor entero:n)
var
   entero: izq, der, cen
inicio
   izq ← 1
   der ← n
   repetir
      cen ← (izq + der) div 2
      si v[cen] > el entonces
         der ← cen - 1
      si_no
         izq ← cen + 1
      fin_si
   hasta_que (v[cen].DNI = el) o (izq > der)
   si v[cen] = el entonces
       devolver(cen)
   si_no
       devolver(0)
   fin_si
fin_función
```

Puntuación: 2 puntos

3. Árboles. Concepto de árbol general. Concepto de árbol binario. Defina los siguientes términos: Nodo terminal, altura de un árbol, peso de un árbol, nivel de un nodo, árbol binario de búsqueda, árbol equilibrado. Dibuje un esquema de un árbol binario equilibrado.

Apartados 13.2 y 13.3 del libro de texto y apuntes de clase

Aplicación
Se tienen almacenadas dentro de un árbol binario de búsqueda las ventas realizadas por una empresa a lo largo del mes de mayo de 2007. Cada nodo contiene información del código del artículo vendido y el importe de la venta.

Diseñe las estructuras de datos necesarias para almacenar dicha información. Diseñe un procedimiento que devuelva el importe total de las ventas realizadas por la empresa.

```
tipos
   registro = TipoElemento
      cadena: código
      real : importe
   fin_registro

   puntero_a nodo = árbol
   registro = nodo
      TipoElemento = info
      árbol = hizq, hder
   fin_registro

real función SumarImporte(valor arbol: a)
inicio
   si a = nulo entonces
      devolver(0)
   si_no
      devolver(a↑.info.importe + SumarImporte(a↑.hizq) +
                                 SumarImporte(a↑.der))
   fin_si
fin_función
```

Puntuación: 1,5 puntos

Preguntas prácticas

Un club deportivo tiene almacenadas las actividades a las que están apuntados sus socios en una lista enlazada. Cada nodo de la lista guarda el número de socio y una cadena con el nombre de la actividad a la que está apuntado. La lista está ordenada ascendentemente por número de socio y cada socio puede estar apuntado a más de una actividad.

Se pide:

a) Defina **todas** las estructuras de datos necesarias para realizar los algoritmos que aparecen a continuación.

```
tipos
   registro = TipoElemento
     cadena : numSocio, actividad
   fin_registro

   puntero_a nodo : Lista

   registro = nodo
      TipoElemento : info
      Lista = sig
   fin_registro

   registro = cola
      puntero_a nodoCola : p,f
   fin_registro

   registro = nodoCola
      TipoElementoCola : info
      puntero_a nodoCola = sig
   fin_registro

   registro = TipoElementoCola
     cadena : numSocio, totalAPagar
   fin_registro
```

Puntuación: 0,5 puntos

b) Codifique un procedimiento que almacene en otra lista a todos los socios apuntados a la actividad de "Tiro con arco". La lista resultante también deberá estar ordenada por número de socio.

```
procedimiento CopiarSocios(valor lista : l; ref lista : tiro)
inicio
   si l = nulo entonces
      tiro ← nulo
   si_no
      CopiarSocios(l,tiro)
      //En el retorno, para copiarlos en el mismo orden
      //si la actividad es tiro con arco, lo inserto en la lista tiro
      si l↑.info = 'Tiro con arco' entonces
         LInsertar(tiro, l↑.info)
      fin_si
   fin_si
fin_procedimiento

procedimiento LInsertar(ref lista : l; valor TipoElemento : e)
var
   lista : aux
inicio
   nuevo(aux)
   aux↑.sig ← l
   aux↑.info ← e
   l ← aux
fin_procedimiento
```

Puntuación: 1 punto

c) Codifique un procedimiento que elimine de la lista a un socio. El número de socio se pasará como argumento al procedimiento y se **eliminarán todas las apariciones** del mismo en la lista.

```
procedimiento BorrarSocio(ref lista : l; valor cadena : numSocio)
var
   lista : act,ant
inicio
   ant ← nulo
   act ← l
   mientras act <> nulo hacer
      si act↑.info.numSocio = numSocio entonces
         //Si es el primero de la lista, borro el elemento al que apunta l
         si act = l entonces
            LBorrar(l)
            act ← l
         si_no
            //Se borra el elemento al que apunta anterior
            LBorrar(ant↑.sig)
            act ← ant↑.sig
         fin_si
      si_no
         //Si no es el socio, se avanza al siguiente nodo
         ant ← act
         act ← act↑.sig
      fin_si
   fin_mientras
fin_prodedimiento

procedimiento LBorrar(ref lista : e)
var
   lista : aux
inicio
   si EsListaVacía(l) entonces
      // error, la lista está vacía
   si_no
```

```
      aux ← l
      l ← l↑.sig
      liberar(aux)
   fin_si
fin_procedimiento
```

Puntuación: 1 punto

d) Codifique un procedimiento que permita almacenar un nuevo elemento a la lista. El procedimiento recibirá el número de socio y la actividad a la que se apunta. Si el socio ya está apuntado a esa actividad aparecerá un mensaje de error indicándolo.

```
procedimiento InsertarOrdenado(E/S lista : l; E TipoElemento : e)
var
   lista : act,ant
   lógico : encontrado
inicio
   //La función buscar devuelve nulo si el elemento e no existe
   si Buscar(l,e) <> nulo entonces
      //Error, El socio ya está apuntado a esa actividad
   si_no
      encontrado← falso
      act ← l
      mientras no encontrado y (act <> nulo) hacer
         si (e.numSocio <= act↑.info.numSocio) entonces
            encontrado ← verdad
         si_no
            ant ← act
            act ← act↑.sig
         fin_si
      fin_mientras
      si act = l entonces
         LInsertar(l,e)
      si_no
         LInsertar(ant↑.sig,e)
      fin_si
   fin_si
fin_prodedimiento

lista función Buscar(valor lista : l; valor TipoElemento : elem)
var
   TipoElemento : e
inicio
   LPrimero(l,e)
   mientras (e.numSocio <> elem.numSocio) y (e.actividad <> elem.actividad)
             y (l <> nulo) hacer
      LPrimero(l,e)
      LSiguiente(l,l)
   fin_mientras
   si (e.numSocio = elem.numSocio) y (e.actividad = elem.actividad) entonces
      devolver(l)
   si_no
      devolver(nulo)
   fin_si
fin_función

procedimiento LPrimero(valor lista : l; ref TipoElemento : e)
inicio
   si l = nulo entonces
      // Error, la lista está vacía
```

```
   si_no
      e ← l↑.info
   fin_si
fin_procedimiento
```

Puntuación: 1,25 puntos

e) Suponiendo que todas las actividades cuestan lo mismo (20 euros al mes), codifique un procedimiento que calcule lo que tiene que pagar cada socio y acumule los resultados en una cola. La cola resultante tendrá un único nodo por cada socio y cada nodo guardará el número de socio y el total a abonar mensualmente.

```
procedimiento TotalAPagar(valor lista : l; ref cola : c)
var
   tipoElemento : e
   tipoElementoCola : eCola
inicio
   //Se inicializa la cola
   c.p ← nulo
   mientras l <> nulo hacer
      LPrimero(l,e)
      //Se inicializa el elemento de la cola
      eCola.numSocio ← e.numSocio
      eCola.totalAPagar ← 0
      //Acumula todas las actividades de un socio
      //El bucle repetir se ejecuta hasta que no cambia el socio o
      //se acaba la lista
      repetir
         eCola.totalAPagar ← eCola.totalAPagar + 20
         l ← l↑.sig
         LPrimero(l,e)
      hasta_que (e.numSocio <> eCola.numSocio) o (l = nulo)
      //Se inserta en la cola
      CInsertar(c,eCola)
   fin_mientras
fin_procedimiento

procedimiento CInsertar(ref cola : c  ; valor TipoElemento : e)
var
  puntero_a nodo : aux
inicio
  reservar(aux)
  aux↑.sig ← nulo
  aux↑.info ← e
  si c.p = nulo entonces
    c.p ← aux
  si_no
    c.f↑.sig ← aux
  fin_si
  c.f ← aux
fin_procedimiento
```

Puntuación: 1,25 puntos