Cuadernillo de examen

ASIGNATURA:	**Fundamentos de Programación II**		
CONVOCATORIA:	**Junio 2012**	PLAN DE ESTUDIOS:	**2010**
CURSO:	**1º**	CURSO ACADÉMICO:	**2011/2012**
TURNO:		TITULACIÓN:	**Grado Ingeniería Informática**
CARÁCTER:	**Obligatoria**	DURACIÓN APROXIMADA:	**2 horas y media**

Soluciones propuestas al examen

Preguntas teórico-prácticas

1. Enumerar y Explicar los distintos de soportes para el almacenamiento de archivos. Enumerar y explicar los distintos tipos de acceso en los archivos. Enumerar y explicar los distintos tipos de organización de archivos. Representar gráficamente la organización de un archivo indexado. Representar gráficamente cómo se realiza el alta de un registro en un archivo indexado a partir de una clave.

Documentación del tema 1. Diapositivas 13-21

Aplicación

El archivo secuencial PAGOS2011.DAT guarda información sobre los pagos que han realizado distintas empresas (el archivo contiene un máximo de 5000 pagos). Contiene la siguiente información: Mes (un número correspondiente a un mes del año 2011), Empresa (una cadena con la empresa que ha realizado el pago) y PagoRealizado (la cantidad pagada). Realizar un procedimiento que permita rellenar un array de registros con la misma información. El procedimiento deberá devolver el número de registros cargados en el array.

```
const
   numPagos = 5000
tipos
   registro = pago
      entero : mes
      cadena : empresa
      real : pagoRealizado
   fin_registro
   array[1..numPagos] de pago = pagos

procedimiento CargarPagos(ref pagos : p; ref entero : n)
var
   archivo_s de pago : A
   pago : R //El registro que se leerá del archivo
inicio
   abrir(A,lectura,'PAGOS2011.DAT')
   leer(A,R)
   n ← 0
   mientras no fda(A) hacer
      n ← n + 1
      p[n] ← R
      leer(A,R)
   fin_mientras
   cerrar(A)
fin_procedimiento
```

Puntuación: 1,5 puntos

2. ¿Qué es una estructura de datos dinámica? ¿Qué diferencias existen entre las estructuras de datos dinámicas lineales y no lineales? Tipos de estructuras dinámicas. ¿Para qué se utilizan las pilas? ¿Para qué se utilizan las colas? ¿En que se diferencia la implementación de pilas y colas?

Documentación del tema 3. Diapositivas 14, 15, 19, 22, 32, 44 y 50

Aplicación

Crear un procedimiento que, dada una lista con elementos de tipo entero ordenada de forma ascendente, saque por pantalla esa misma lista ordenada descendentemente (no pueden generarse nodos nuevos).

```
tipos
   entero = tipoElemento
   puntero_a nodo = lista
   registro = nodo
      tipoElemento : info
      lista : sig
   fin_registro

//La recursividad permite listar la lista al revés sin mover sus elemento
procedimiento ListarAlRevés(valor lista: l)
inicio
   si l <> nulo entonces
      ListarAlRevés(l↑.sig)
      escribir(l↑.info)
   fin_si
fin_procedimiento
```

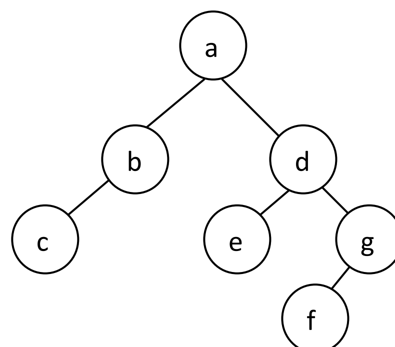
Puntuación: 2 puntos

3. Árboles. Definir el concepto de árbol. Definir los conceptos: nodo hoja, nodos hermanos, árbol completo, árbol equilibrado y recorrido en profundidad.

Documentación del tema 3. Diapositivas 4-9

Aplicación

Se tienen almacenados una serie de caracteres en un árbol binario de búsqueda y se requiere un recorrido preorden. Implementar dicho procedimiento y haga un seguimiento del árbol:




```
procedimiento PreOrden(valor árbol: a)
inicio
   si a <> nulo entonces
      //Procesar elemento raíz
      escribir(a↑.raíz)
      PreOrden(a↑.hizq)
      PreOrden(a↑.hder)
   fin_si
fin_procedimiento
```

Recorrido: a, b, c, d, e, g, f

Puntuación: 1,5 puntos

Preguntas prácticas

En una lista simplemente enlazada se almacenan los préstamos de una biblioteca. Por cada nodo se guarda el ISBN del libro, la fecha de préstamo (una cadena en formato aaaammdd) y el número de socio al que se le ha prestado. La lista está ordenada de forma ascendente por ISBN y cada socio puede tener prestados varios libros.

a) Definir las estructuras de datos necesarias para realizar los puntos que aparecen a continuación

```
tipos
   registro = tipoElemento
      cadena : isbn, fecha
      entero : nSocio
   fin_registro
   puntero_a nodo = lista
   registro = nodo
```

```
      tipoElemento : info
      lista : sig
   fin_registro
```

Puntuación: 0,5 puntos

b) Codificar un subprograma que permita añadir un préstamo a la lista, manteniendo el estado de ordenación.

```
procedimiento AñadirPréstamo(ref lista : l; valor TipoElemento : e)
var
   lista : act,ant
   lógico : encontrado
inicio
   encontrado← falso
   act ← l
   mientras no encontrado y (act <> nulo) hacer
      si e.fecha <= act↑.info.fecha 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_prodedimiento

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

Puntuación: 1,5 puntos

c) Codificar un subprograma que almacene en otra lista los libros que se han prestado a un socio cuyo número de socio se pasará como argumento. La lista resultante también estará ordenada ascendentemente por ISBN.

```
procedimiento CopiarSocios(valor lista:l;ref lista:nueva;valor entero:núm)
inicio
   si l = nulo entonces
      nueva ← nulo
   si_no
      CopiarSocios(l↑.sig,nueva,núm)
      si l↑.info.nSocio = núm entonces
         LInsertar(nueva, l↑.info)
      fin_si
   fin_si
fin_procedimiento
```

Puntuación: 1,5 puntos

d) Codificar un subprograma que permita eliminar de la lista los libros cuya fecha de devolución sea posterior al 3/6/2012 (20120603).

```
procedimiento BorrarPréstamos(ref lista : l)
var
   lista : act, ant
inicio
   act ← l
   mientras act <> nulo hacer
      si act↑.info.fecha > '20120603' entonces
         si act = l entonces //Borrar el primero de la lista
            LBorrar(l)
            act ← l
         si_no
            LBorrar(ant↑.sig)
            act ← ant↑.sig
         fin_si
      si_no
         ant ← act
         act ← act↑.sig
      fin_si
   fin_mientras
fin_procedimiento

procedimiento LBorrar(ref lista : l)
var
   lista : aux
inicio
   si l = nulo entonces
      // error, la lista está vacía
   si_no
      aux ← l
      l ← l↑.sig
      liberar(aux)
   fin_si
fin_procedimiento
```

Puntuación: 1,5 puntos