Cuadernillo de examen

Asignatura:	**Fundamentos de Programación II**			Código:	**113**
Titulación:	**Ingeniería Informática / Ingeniería Técnica en Informática**			Plan de estudios:	**2000/2002**
Curso:	**1º**	Carácter:	**Obligatoria**	Especialidad:	**Común**
Convocatoria:	**Junio 2006**	Curso académico:	**2005/2006**	Turno:	**Mañana**
Duración aproximada:	**2 horas y media**				

Solución propuesta

Preguntas teóricas

1. Archivos directos con transformación de clave. Utilidad de las funciones hash. Enumere las funciones hash que conozca y explique su funcionamiento. ¿Qué es un sinónimo? ¿Qué es una colisión? ¿Cómo se pueden solucionar las colisiones?

Apartado 9.11 del libro de texto y apuntes de clase

Aplicación
Se tiene creado un archivo directo por transformación de claves. En él se desea guardar información sobre una serie de alumnos, con los campos Expediente, Nombre, Fecha de matrícula, siendo su clave primaria el campo numérico Expediente. El archivo va a contener un total de 2000 registros. Codifique un procedimiento que permita dar, de alta a un nuevo alumno.

```
const
   NumRegs = 2000
tipos
   registro = alumno
      entero : expediente
      cadena : nombre, FechaMatricula
      entero : estado // estado = 1 registro ocupado
   fin_registro
   archivo_d de alumno = alumnos

procedimiento AñadirAlumno(ref alumnos : A; valor alumno : R)
var
   entero : NRR //Número de registro relativo
   alumno : RAux //Un registro auxiliar para ver si una posición está libre
inicio
   //El archivo ya está abierto en modo lectura/escritura
   //El argumento R contiene todos los datos del nuevo alumno
   NRR ← hash(R)
   leer(A, RAux, NRR)
   si RAux.estado = 1 entonces //Si la posición está ocupada
      //Se mueve a la primera posición libre disponible
      repetir
         NRR ← NRR mod NumRegs + 1 //Se lee la siguiente posición o a la primera
         leer(A, RAux, NRR)
      hasta_que RAux.estado <> 1 //hasta encontrar una posición no ocupada
   fin_si
   //En este punto NRR siempre indica una posición no ocupada
   R.estado ← 1
   escribir(A,R,NRR)
fin_procedimiento
```

Puntuación: 1,5 puntos

2. Árboles. Defina los siguientes conceptos: nodo raíz, hoja, altura de un árbol, peso de un árbol, árboles equivalentes, árbol equilibrado

Apartado 13.2.1 y 13.3.1 del libro de texto y apuntes de clase

Aplicación
Se tiene un array con 100 elementos de tipo cadena que se desean almacenar en un árbol binario de búsqueda. Codifique un procedimiento que permita realizar la inserción de los elementos del array en un árbol binario de búsqueda, teniendo en cuenta que no se deben almacenar los elementos repetidos.

```
tipos
   array[1..100] de cadena = vector
   cadena = tipoElemento //Tipo base del árbol
   puntero_a nodo = árbol
   registro
      info : tipoElemento
      árbol : hiz,hde
   fin_registro

procedimiento VolcarEnÁrbol(valor vector : v; valor entero : n;  ref árbol : a)
var
   entero : i
incio
   a ← nulo
   desde i ← 1 hasta n hacer
      InsertarElementoEnÁrbol(a,v[i])
   fin_desde
fin_procedimiento

procedimiento InsertarElementoEnÁrbol(ref arbol : a; valor tipoElemento : e)
inicio
   si a = nulo entonces
      reservar(a)
      a↑.info ← e
      a↑.hiz ← nulo
      a↑.hde ← nulo
   si_no
      si a↑.info > e entonces
        InsertarElementoEnÁrbol(a↑.hiz,e)
      si_no
        si a↑.info < e entonces
           InsertarElementoEnÁrbol(a↑.hde,e)
        si_no
           //El elemento ya existe y no se inserta
        fin_si
     fin_si
   fin_si
fin_procedimiento
```

Puntuación: 1,5 puntos

3. Programación orientada a objetos. Describa la diferencia entre clase y objeto. Explique la estructura de un mensaje dentro de la programación orientada a objetos. Explique brevemente las relaciones entre clases. Explique brevemente en que consiste el polimorfismo.

Apartado 16.1, 16.2, 17.3, 17.5 y 17.9 del libro de texto y apuntes de clase

Aplicación
Defina una clase Persona con los atributos DNI, nombre y edad. Codifique un constructor que permita crear una instancia de la clase persona a partir de los tres datos que se pasarán como parámetro. Cree una clase Empleado que herede de Persona y la añada el atributo sueldo. Modifique el constructor de la superclase para que admita también como argumento el sueldo del empleado. Codifique un método que permita incrementar el sueldo del empleado en un porcentaje que se pasará como argumento.
Codifique además las líneas de código correspondientes para: crear una instancia de la clase Persona, una instancia de la clase Empleado y que envíe un mensaje a esta última instancia para que aumente su sueldo en un 5%.

```
clase Persona
   var
      cadena : DNI,nombre
```

```
      entero : edad
   constructor Persona(valor cadena : d,n ; valor entero : e)
   inicio
      DNI ← d
      nombre ← n
      edad ← e
   fin_constructor
fin_clase

clase Empleado hereda_de Persona
   var
      real : sueldo
   constructor Empleado(valor cadena : d,n ; valor entero : e; valor real : s)
   inicio
      super(d,n,e)
      sueldo ← s
    fin_constructor

    procedimiento AumentarSueldo(valor real : porcentaje)
    inicio
       sueldo ← sueldo + sueldo * porcentaje / 100
    fin_procedimiento
fin_clase


var
   //Crear una instancia de una persona
   Persona : p ← nuevo Persona('1234334D',"Pepe Pérez",34)
   //Crear una instancia de un empleado
   Empleado : e ← nuevo Empleado('4453454F',"Ana Gutiérrez",23,1200.5)
...

//Aumentar el sueldo un 5%
e.AumentarSueldo(5)
```

Puntuación: 1,5 puntos

Preguntas prácticas

En un archivo directo se tienen almacenados los resultados de una oposición. Cada registro contiene los siguientes campos:

- DNI
- Nombre
- Nota obtenida en el primer examen.
- Nota obtenida en el segundo examen

Se pide:

1. Realizar la declaración de todas las estructuras de datos necesarias para realizar los supuestos que aparecen a continuación.

```
const
   NumRegs = ...
tipos
   //Declaraciones para el archivo y su registro
   registro = opositor
      cadena : DNI, nombre
      real : nota1,nota2
   fin_registro
   archivo_d de opositor = ExamenOposición
   //Declaraciones de la lista
   opositor = tipoElemento
   puntero_a nodo = lista
```

```
registro = nodo
   tipoElemento : info
   lista : l
fin_registro
//Declaraciones de la cola
registro = cola
   puntero_a nodo : p,f
fin_registro
```

Puntuación: 0,5 puntos

2. Codifique un procedimiento que permita cargar todos los registros en la lista enlazada Opositores. La lista debe estar ordenada por el campo Nombre.

```
procedimiento CargarArchivoEnLista(ref ExamenOposición : A; ref lista : Opositores)
var
   entero : i
   opositor : R
inicio
   abrir(A, "OPOSICIONES.DAT", lectura)
   Opositores ← nulo //Crear la lista de opositores
   desde i ← 1 hasta NumRegs hacer
      leer(A,R,i)
      InsertarOrdenado(Opositores,R)
   fin_desde
   cerrar(A)
fin_procedimiento

procedimiento InsertarOrdenado(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 = act↑.info 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↑.sig ← l
   aux↑.info ← e
   l ← aux
fin_procedimiento
```

Puntuación: 1,25 puntos

3. Codifique un procedimiento que elimine de la lista Opositores a todos los opositores que hayan suspendido algunos de los dos exámenes (nota menor que 5).

```
procedimiento BorrarSuspensos(ref lista : Opositores)
```

```
var
   lista : act, ant
inicio
   act ← Opositores
   mientras act <> nulo hacer
      si (act↑.info.nota1 < 5) o (act↑.info.nota2 < 5) entonces
         //Ha suspendido algún examen
         si act = Opositores entonces
            //Se trata del primer elemento de la lista
            LBorrar(Opositores)
            act ← Opositores //El puntero actual vuelve a ser el comienzo de la lista
         si_no
            LBorrar(ant↑.sig)
            act ← ant↑.sig //El puntero actual apunta al nuevo siguiente elemento
         fin_si
      si_no
         //Se actualizan los punteros siguiente y anterior
         ant ← act
         act ← act↑.sig
      fin_si
   fin_mientras
fin_procedimiento

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: 1,25 puntos

4. Codifique un procedimiento que copie en otra lista a todos aquellos opositores cuya nota media sea mayor o igual que la nota de corte. La nota de corte es la nota que deben superar los opositores para aprobar la oposición, y se pasará como argumento al procedimiento. En la lista resultante los opositores también deberán estar ordenados por nombre.

```
procedimiento CopiarAprobados(valor lista : Opositores; ref lista : aprobados ;
                              valor real : notaCorte)
//Se trata de un procedimiento recursivo para que los opositores
//queden ordenados otra vez por nombre
inicio
   si Opositores = nulo entonces
      //Se crea la nueva lista
      aprobados ← nulo
   si_no
      CopiarAprobados(Opositores↑.sig, aprobados, notaCorte)
      si (Opositores↑.info.nota1 + Opositores↑.info.nota1) / 2 >= notaCorte entonces
         //Se copia la información en la lista de aprobados
         LInsertar(aprobados, Opositores↑.info)
      fin_si
   fin_si
fin_procedimiento
```

Puntuación: 1,25 puntos

5. Codifique otro procedimiento que permita almacenar en una cola todos los opositores cuya nota sea mayor de 8.

```
procedimiento AlmacenarEnCola(valor lista : Opositores ; ref Cola : c)
inicio
   CrearCola(c)
   mientras Opositores <> nulo hacer
      si (Opositores↑.info.nota1 + Opositores↑.info.nota1) / 2 > 8 entonces
         CInsertar(c, Opositores↑.info)
      fin_si
      Opositores ← Opositores↑.sig
   fin_mientras
fin_procedimiento

procedimiento CrearCola(ref cola : c)
inicio
   c.p ← nulo
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