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	**Ordinario de Junio 2003**	PLAN DE ESTUDIOS	**2000/2002**
ESPECIALIDAD	**Común**	CURSO	**1º**
TURNO	**Mañana**	CURSO ACADÉMICO	**2002-2003**
CARÁCTER	**Cuatrimestral**	PROGRAMA	**Ingeniería Superior/ Ingeniería Técnica**
DURACIÓN APROXIMADA	**2 horas y media**		

Soluciones propuestas

Preguntas teórico-prácticas

1. Recursividad. Concepto de recursividad. Tipos de recursividad. Ventajas e inconvenientes de la recursividad. ¿En qué ocasiones es preferible utilizar métodos recursivos en lugar de métodos iterativos?

Apartado 5.8 del libro de texto y apuntes de clase

Práctica

Escriba el algoritmo de ordenación QuickSort

```
procedimiento QuickSort(E vector : v; E entero : iz,de)
var
   entero : i,j
   TipoElemento : pivote,aux
inicio
   i ← iz
   j ← de
   pivote ← v[(iz+de) div 2]
   repetir
      mientras v[i] < pivote hacer
         i ← i `+ 1
      fin_mientras
      mientras v[j] > pivote hacer
         j ← j - 1
      fin_mientras
      si i >= j entonces
          aux ← v[i]
          v[i] ← v[j]
          v[j] ← aux
          i ← i + 1
          j ← j – 1
      fin_si
   hasta_que i > j
   si iz < j entonces
      QuickSort(v,iz,j)
   fin_si
   si i < de entonces
      QuickSort(v,i,de)
   fin_si
fin_procedimiento
```

Dada la siguiente lista de números: 6 3 5 8 4, realice una traza del QuickSort sobre la misma, indicando los valores que tomarían las variables en cada llamada recursiva.

					iz	de	pivote	i	j
4	3	5	8	6	1	5	5	4	2
3	4	5	8	6	1	2	4	2	1
3	4	5	6	8	4	5	8	5	4

Puntuación: 1,5 puntos

2. Programación Orientada a Objetos. Explique los conceptos de objeto, clase, instancia y herencia.

Apuntes proporcionados en clase por el profesor

Práctica

Realice la declaración de una clase Asignatura. La clase contendrá atributos para almacenar el código y el nombre de la asignatura. Así mismo deberá tener dos atributos de tipo real para almacenar la nota de prácticas. Su constructor deberá inicializar los atributos de código y nombre.

Declare además las clases derivadas AsignaturaAnual y AsignaturaCuatrimestral. Ambas clases heredan de la clase Asignatura. La clase AsignaturaAnual tendrá un atributo para la nota teórica del primer cuatrimestre y otra para la del segundo cuatrimestre.

La clase AsignaturaCuatrimestral tendrá un atributo para indicar el cuatrimestre (primer o segundo cuatrimestre) y otro para la nota teórica.

Las dos clases derivadas deberán tener un método calcularNotaFinal. En el caso de una asignatura cuatrimestral la nota final se calculará teniendo en cuenta que la nota de prácticas vale un 30 por ciento de la nota final y la nota teórica un 60 por ciento de la nota final. En el caso de una asignatura anual la nota teórica final será la media de las notas del primer y segundo cuatrimestre.

```
clase Asignatura
var
   protegido cadena : código
   protegido cadena: nombre
   protegido real : notaPracticas
constructor Asignatura(E cadena: cod, nom)
inicio
   código ← cod
   nombre ← nom
fin_constructor
fin_clase

clase AsignaturaAnual hereda_de Asignatura
var
   protegido real : notaPrimerCuatrimestre
   protegido real : notaSegundoCuatrimestre
   real método calcularNotaFinal()
var
   real : notaTeorica
inicio
   notaTeorica ← (notaPrimerCuatrimestre + notaSegundoCuatrimestre) / 2
   devolver(notaTeorica * 0.60 + notaPracticas * 0.40)
fin_método
fin_clase

clase AsignaturaCuatrimestral hereda_de Asignatura
var
   protegido real : notaTeorica
inicio
   devolver(notaTeorica * 0.60 + notaPracticas * 0.40)
fin_método
fin_clase
```

Puntuación: 1,5 puntos

Preguntas prácticas

1. Una tienda de productos informáticos mantiene los productos de su almacén en un archivo indexado. La estructura de campos es la siguiente:

Campo	Tipo	Observaciones
CodProd	Entero	Código del producto. Clave principal del archivo
Desc	Cadena	Descripción del producto
Precio	Real	
Stock	Entero	Número de unidades en el almacén
Ocupado	Lógico	Verdad o falso según el registro esté o no ocupado

El índice está gestionado mediante un árbol binario de búsqueda en el que, por cada registro ocupado, existe un nodo con el campo y el número de registro donde está almacenado.

Se pide:

- Declarar las estructuras de datos necesarias para la gestión del archivo y su índice.

```
const
   MaxReg = … //Número máximo de registros en el área de datos
tipo
   //Declaraciones para el área de datos
   registro = RProducto
      entero :CodProd
      cadena : Descr
      real : precio
      entero : stock
      lógio : ocupado
   fin_registro
   archivo_d de RProducto = AProducto
   //Declaraciones para el árbol
   registro = TipoElemento
      entero : CodProd
      entero : NRR //Número de registro relativo donde se ha almacenado el producto
   fin_registro
   puntero_a nodo = árbol
   registro = nodo
      TipoElemento : info
      árbol : hizq, hder
   fin_registro
```

- Codificar un procedimiento que permita dar un alta en el archivo indexado. El procedimiento deberá controlar que no se almacenan códigos de producto repetidos.

```
procedimiento Alta(E/S AProductos:AProd;E RProducto:R;E/S árbol : a)
var
   RProducto : RAux
   entero : NRR
   TipoElemento : e
inicio
   //La función buscar busca un CodProd en el índice
   //Devuelve nulo si el código no existe
   si (buscar(a,R)) <> nulo entonces
      //Ya está
   si_no
      //Alta en área de datos
      //Buscar hueco utilizando una función hash
      NRR ← hash(R.CodProd)
      leer(AProd,RAux,NRR)
      mientras RAux.ocupado hacer
          NRR ← NRR mod MaxReg + 1
          leer(AProd,RAux,NRR)
      fin_mientras
      R.ocupado ← verdad
      //Escribe en el área de datos
      escribir(AProd,R,NRR)
```

```
      //Inserta el CodProd y el número de registro relativo en el árbol
      e.CodProd ← CodProd
      e.NRR ← NRR
      Insertar(a,e)
fin_procedimiento

árbol : función Buscar(E árbol : a; E TipoElemento : e)
inicio
   si a = nulo entonces
      devolver(nulo)
   si_no
      si a↑.info.CodProd = e.CodProd entonces
         devolver(a)
      si_no
         si a↑.info.CodProd > e.CodProd entonces
            Buscar(a↑.hizq,e)
         si_no
            Buscar(a↑.hder,e)
         fin_si
     fin_si
   fin_si
fin_función

procedimiento Insertar(E/S árbol :a ; TipoElemento : e)
inicio
   si a = nulo
      //Hemos llegado a una hoja. Insertamos
      reservar(a)
      a↑.info ← e
      a↑.hizq ← nulo
      a↑.hder ← nulo
   si_no
      si act↑.info.CodProd > e.CodProd entonces
         Insertar(act↑.hizq,e)
      si_no
         Insertar(act↑.hder,e)
      fin_si
   fin_si
fin_mientras
```

- Codificar un procedimiento que obtenga un listado ordenado de aquellos productos cuyo campo Stock sea menor o igual a 0.

```
procedimiento Listado(E arbol : a; E/S AProductos : AProd)
var
   RProducto : R
inicio
   si a <> nulo entonces
      Listado(a↑.hizq,AProd)
      //Procesar el registro a↑.info.NRR
      leer(AProd,R, a↑.info.NRR)
      si R.stock <= 0 entonces
         escribir(R.CodProd, R.Descr, R.Precio, R.stock
      fin_si
      Listado(a↑.hder,AProd)
   fin_si
fin_procedimiento
```

Puntuación: 3,5 puntos

2. En una pila están almacenados una serie de números enteros.

Codifique los procedimientos necesarios para:

- Almacenar en una lista enlazada L1 los números menores de 100 ordenados de menor a mayor, eliminándolos a su vez de la pila.

```
procedimiento MenoresQue100(E pila : p; E/S lista : L1)
var
   pila : pAux
   TipoElemento : e
inicio
   crearPila(pAux)
   crearLista(L1)
   mientras no EsPilaVacía(p) hacer
      Cima(p,e)
      PBorrar(p)
      si e < 100 entonces
         InsertarOrdenado(L1,e)
      si_no
         PInsertar(pAux,e)
      fin_si
   fin_mientras
   //Se reinsertan en P los elemento de pAux
   mientras no EsPilaVacía(pAux) hacer
      Cima(pAux,e)
      PBorrar(pAux)
      PInsertar(p,e)
   fin_mientras
fin_procedimiento

procedimiento CrearPila(E/S pila : p)
inicio
   p ← nulo
fin_procedimiento

lógico: función EsPilaVacía(E pila : p)
inicio
   devolver(p = nulo)
fin_función

procedimiento PInsertar(E/S pila : p  ; E TipoElemento : e)
var
   pila : aux
inicio
   reservar(aux)
   aux↑.sig ← p
   aux↑.info ← e
   p ← aux
fin_procedimiento

procedimiento Cima(E pila : p  ; E/S TipoElemento : e)
inicio
   si p = nulo entonces
      // Error, la pila está vacía
   si_no
      e ← p↑.info
   fin_si
fin_procedimiento

procedimiento PBorrar( E/S pila : p)
var
   pila : aux
inicio
   si EsPilaVacía(p) entonces
      // error, la pila está vacía
   si_no
      aux ← p
      p ← p↑.sig
      liberar(aux)
   fin_si
fin_procedimiento


procedimiento CrearLista(E/S lista : l)
```

```
inicio
   l ← nulo
fin_procedimiento

procedimiento InsertarOrdenado(E/S lista : l; E TipoElemento : e)
var
   lista : act, ant
   lógico : encontrado
inicio
   act ← l
   encontrado ← falso
   mientras no  encontrado y (act <> nulo) hacer
      si act↑.info >= e entonces
           encontrado ← verdad
      si_no
           ant ← act
           act ← act↑.sig
      fin_si
   fin_mientras
   //Si hay que insertarlo al comienzo de la lista
   si l = act entonces
       LInsertar(l,e)
   si_no
       LInsertar(ant↑.sig,e)
   fin_si
fin_procedimiento

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

- Borrar de la lista L1 los números impares almacenándolos en la lista enlazada L2.

```
procedimiento BorrarImpares(E/S lista L1; E/S lista : L2)
var
   lista : act,ant
inicio
   crearLista(L2)
   act ← L1
   ant ← nulo
   mientras no EsListaVacía(act) hacer
      si act↑.info mod 2 <> 0 entonces
         LInsertar(L2, act↑.info)
         //Si es el primer elemento de la lista hay que borrar el primer
         //elemento de la lista
         si act = L1 entonces
            LBorrar(L1)
         si_no
            LBorrar(act↑.sig)
         fin_si
      fin_si
      ant ← act
      act ← act↑.sig
   fin_mientras
fin_procedimiento

procedimiento LBorrar(E/S 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
```

- Hacer un listado ordenado de menor a mayor de la lista L2

```
//En el procedimiento BorrarImpares,
//los elementos se han insertado de mayor a menor.
//Para hacer un listado de menor a mayor hay que utilizar pilas
//o hacer un algoritmo recursivo
procedimiento Listar(E lista : L2)
inicio
   si no EsListaVacia(L2) entonces
      Listar(L2↑.sig)
      escribir(L2↑.info)
   fin_si
fin_procedimiento
```

Nota: el alumno debe realizar las declaraciones de todas las estructuras de datos utilizadas en el ejercicio.

```
tipos
   entero = TipoElemento //El tipo de elementos de la pila y las listas
   puntero_a nodo = lista
   registro = nodo
      TipoElemento : info
      lista : sig
   fin_registro
   puntero_a nodo = pila
var
   pila : p
   lista : L1,L2
```

Puntuación: 3,5 puntos