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

ASIGNATURA:	**Fundamentos de Programación II**	CÓDIGO:	**113**
CONVOCATORIA:	**Parcial de abril de 2005**	PLAN DE ESTUDIOS:	**2000/2002**
ESPECIALIDAD:	**Común**	CURSO:	**1º**
TURNO:	**Mañana**	CURSO ACADÉMICO:	**2004/2005**
CARÁCTER:	**Cuatrimestral**	PROGRAMA:	**Ingeniería Informática** **Ingeniería Técnica en Informática**

Solución propuesta al examen

Pregunta teórico-práctica

Concepto de recursividad. Tipos de recursividad. Salida de procedimientos recursivos.

Aplicación

Codifique un procedimiento recursivo que reciba un número y permita escribirlo al revés.

```
procedimiento NumeroReves(E entero: n)
inicio
    si n > 0 entonces
        escribir(n mod 10)
        NumeroReves(n div 10)
    fin_si
fin_procedimiento
```

Puntuación: 5 puntos

Pregunta práctica

Se tienen almacenados en una lista enlazada los nombres de una serie de alumnos y alumnas y las notas de obtenidas en el examen de la asignatura de Fundamentos de Programación I.. La lista enlazada está ordenada por el nombre del alumno.
Se pide:

1. Declarar las estructuras de datos necesarias para poder realizar los módulos que aparecen a continuación.
   Puntuación: 0,5 puntos

```
tipos
   tipoElemento = registro
      cadena : nombre
      real : nota
   fin_registro

   registro = cola
      puntero_a nodo : p,f
   fin_registro
   puntero_a nodo = lista
   registro = nodo
     tipoElemento : info
     lista : sig
   fin_registro
```

2. Codificar un módulo que permita almacenar en una cola los alumnos con una nota superior a 7.
   Puntuación: 1,5 puntos

```
procedimiento almacenarEnCola(E lista : l; E/S cola : c)
var
   tipoElemento : e
inicio
   //Crear la cola
   c.p ← nulo
   c.f ← nulo

   mientras l <> nulo hacer
```

```
      si l↑.info.nota > 7 entonces
         CInsertar(c,l↑.info)
      fin_si
      l ← l↑.sig
      fin_si
   fin_mientras
fin_procedimiento

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

3. Codificar un módulo que cree otra lista con todos los elementos de la lista, pero ordenados por la nota.
Puntuación: 1,5 puntos

```
procedimiento NuevaLista(E lista : l; E/S lista: nueva)
inicio
   //Crear la lista nueva
   nueva ← nulo
   mientras l <> nulo hacer
      InsertarOrdenado(nueva, l↑.info)
      l ← l↑.sig
   fin_mientras
fin_procedimiento

procedimiento InsertarOrdenado(E/S lista : l; E tipoElemento: e)
var
   lista : act, ant, aux
   lógico : encontrado
inicio
   encontrado ← falso
   act ← l
   mientras no encontrado y (act <> nulo) hacer
      si act↑.info.nota > e.nota entonces
         encontrado ← verdad
      si_no
         ant ← act
         act ← sig↑.sig
      fin_si
   fin_mientras

   //Reservar espacio para un nuevo nodo
   reservar(aux)
   aux↑.info ← e
   aux↑.sig ← act
   //Si es el primer nodo lo enlazo con l
   si act = l entonces
     l ← aux
```

```
   si_no
      //Si no lo enlazo con el campo siguiente del nodo anterior
      ant↑.sig ← aux
   fin_si
fin_procedimiento
```

4. Codificar un módulo que elimine de la lista aquellos alumnos que tengan la asignatura suspensa.
Puntuación: 1,5 puntos

```
procedimiento EliminarSuspensos(E/S lista : l)
var
   lista : act, ant, aux
inicio
   act ← l
   mientras act <> nulo hacer
      si act↑.info.nota < 5 entonces
         //Guardo la dirección del noto actual
         aux ← act
         //Si es el primer nodo de la estrucura...
         si act = l entonces
            //Borro el primer nodo
            l ← l↑.sig
            //El nodo actual será el primer nodo
            act ← l
         si_no
            //Borro el nodo siguiente al nodo anterior
            ant.sig ← aux↑.sig
            //El nodo actual será el siguiente nodo
            act ← aux↑.sig
        fin_si
        liberar(aux)
      si_no
         ant ← act
         act ← act↑.sig
      fin_si
   fin_mientras
fin_procedimiento
```