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	Parcial de Abril de 2003	PLAN DE ESTUDIOS	2002
ESPECIALIDAD	Común	CURSO	1º
TURNO	Mañana	CURSO ACADÉMICO	2002/2003
CARÁCTER	Cuatrimestral	PROGRAMA	Ingeniería Superior en Informática
DURACIÓN APROXIMADA			

Pregunta teórica

Recursividad
Concepto de recursividad. Tipos de recursividad. Condición de salida en los procesos recursivos.
Capítulo 5 del libro de texto

Codifique un procedimiento recursivo al que se le pase como argumento una cadena y la escriba primero con los caracteres en orden y, después con los caracteres en orden inverso.

```
procedimiento Examen(E cadena : c)
var
   carácter : car
inicio
   si longitud(c) > 0 entonces
      car ← subcadena(c,1,1)
      escribir(car)
      Examen(subcadena(c,2))
      escribir(car)
   fin_si
fin_procedimiento
```

Puntuación: 1,5 puntos

Pregunta práctica

Se desean implementar conjuntos utilizando una lista enlazada. Los elementos del conjunto estarán ordenados de forma ascendente y no tendrán repeticiones.

Codifique los métodos adecuados para:

1. Incluir un elemento en un conjunto.

Declaraciones de tipos de datos

```
tipos
   … = TipoElemento //El tipo de elementos del conjunto
   puntero_a nodo = lista //El conjunto sería una lista enlazada
   registro = nodo
      TipoElemento : info
      lista : sig
    fin_registro
```

```
procedimiento Incluir(E/S lista : c; E TipoElemento : e)
var
   lista : act, ant
   lógico : encontrado
inicio
   act ← c
   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

   //Sólo hay que incluir el elemento si el elemento no está
   si no encontrado entonces
      //Si hay que insertarlo al comienzo de la lista
      si c = act entonces
          LInsertar(c,e)
      si_no
          LInsertar(ant↑.sig,e)
      fin_si
   fin_si
fin_procedimiento

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

2. Comprobar si un elemento pertenece a un conjunto.

```
lógico función Pertenece(E lista : c; E TipoElemento : e)
var
   lista : act
   lógico : encontrado
inicio
   act ← c
   encontrado ← falso
   mientras no  encontrado y (act <> nulo) hace
      si act↑.info >= e entonces
           encontrado ← verdad
      si_no
           act ← act↑.sig
      fin_si
   fin_mientras
   devolver(act↑.info = e)
fin_función
```

3. Realizar un procedimiento que obtenga el conjunto unión de dos conjuntos.

```
procedimiento Unión (E lista : c1, c2 ; E/S lista : union)
var
   TipoElemento : e1, e2
inicio
   crearLista(union)
   mientras no EsListaVacia(c1) y no EsListaVacia(c2) hacer
      LPrimero(c1,e1)
      LPrimero(c2, e2)
      si e1 < e2 entonces
         LInsertar(union,e1)
```

```
            LSiguiente(c1,c1)
         si_no
            si e2 < e1 entonces
               LInsertar(union,e2)
               LSiguiente(c2,c2)
            si_no
               LInsertar(union,c1)
               LSiguiente(c1,c1)
               LSiguiente(c2,c2)
            fin_si
         fin_si
      fin_mientras
      mientras no EsListaVacia(c1) hacer
         LPrimero(c1,e1)
         LInsertar(union,e1)
         LSiguiente(c1,c1)
      fin_mientras
      mientras no EsListaVacia(c2) hacer
         LPrimero(c2,e2)
         LInsertar(union,e2)
         LSiguiente(c2,c2)
      fin_mientras
      InvertirLista(union)
fin_procedimiento

procedimiento CrearLista(E/S lista : l)
inicio
   l ← nulo
fin_procedimiento

lógico: función EsListaVacía(E lista : l)
inicio
   devolver(l = nulo)
fin_función

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

procedimiento LPrimero(E lista : l; E/S TipoElemento : e)
inicio
   si l = nulo entonces
      // Error, la listaestá vacía
   si_no
      e ← l↑.info
   fin_si
fin_procedimiento

procedimiento LSiguiente(E lista : l; E/S lista  : sig)
inicio
   si l = nulo entonces
      // Error, la listaestá vacía
   si_no
      sig ← l↑.sig
   fin_si
```

```
fin_procedimiento

procedimiento InvertirLista(E/S lista : l)
var
   lista : act,sig,ant
inicio
   si l <> nulo entonces
      act ← l↑.sig
      ant ← l
      l↑.sig ← nulo
      mientras act <> nulo hacer
         sig ← act↑.sig
         act↑.sig ←ant
         ant ← act
         act ← sig
      fin_mientras
      l ← ant
   fin_si
fin_procdimiento
```

Puntuación: 3,5 puntos