

EJERCICIOS DE LA RELACION DE TADS

(Estos son algunos ejercicios difíciles de las relaciones de problemas. NO están hechos en clase!)

EJERCICIO 4.-

Al TAD Lista explicado en clase añade las siguientes operaciones. Impleméntalas utilizando variables dinámicas, y por otro lado también usando variables estáticas.

Algoritmo Longitud(lista:TipoLista):N

Algoritmo EnLista(lista:TipoLista;dato:TipoElem):B

(*Devuelve TRUE si dato está en lista, sin modificarla*)

Algoritmo InsertaDesp(VAR lista:TipoLista;dato_ant,nuevo_dato:TipoElem)

(*Inserta nuevo_dato en la lista después de dato_ant*)

Algoritmo BorraDesp(VAR lista:TipoLista;dato:TipoElem)

(*Borra el elemento que sigue a dato en lista*)

ALGORITMO Longitud(lista:TipoLista):N (*solución estática iterativa *)

VARIABLES

long:N

aux:TipoLista

INICIO

aux:=lista

long:=0

MIENTRAS aux<>NULO **HACER**

long:=long+1

aux:=Nodos[aux].enlace

FINMIENTRAS

DEVOLVER long

FIN

ALGORITMO Longitud(lista:TipoLista):N (*solución dinámica iterativa *)

VARIABLES

long:N

aux:TipoLista

INICIO

aux:=lista

long:=0

MIENTRAS aux<>NIL **HACER**

long:=long+1

aux:=aux^.sig

FINMIENTRAS

DEVOLVER long

FIN

ALGORITMO Longitud(lista:TipoLista):N (*solución estática recursiva *)

INICIO

SI lista=NULO **ENTONCES**

DEVOLVER 0

ENOTROCASO

DEVOLVER 1+Longitud(Nodos[lista].enlace)

FINSI

FIN

ALGORITMO Longitud(lista:TipoLista):N (*solución dinámica recursiva *)

INICIO

SI lista<>NIL **ENTONCES**

DEVOLVER 0

ENOTROCASO

DEVOLVER 1+Longitud(lista^.sig)

FINSI

FIN

ALGORITMO EnLista(lista:TipoLista;dato:TipoElemento):**B** (*solución estática iterativa *)

VARIABLES

encontrado:**B**
aux:TipoLista

INICIO

aux:=lista
encontrado:=**FALSE**
MIENTRAS aux<>**NULO AND (NOT encontrado) HACER**
 encontrado:=(Nodos[aux].elemento=dato)
 aux:=Nodos[aux].enlace
FINMIENTRAS
DEVOLVER encontrado

FIN

ALGORITMO EnLista(lista:TipoLista;dato:TipoElemento):**B** (*solución dinámica iterativa *)

VARIABLES

encontrado:**B**
aux:TipoLista

INICIO

aux:=lista
encontrado:=**FALSE**
MIENTRAS aux<>**NIL AND (NOT encontrado) HACER**
 encontrado:=(aux^.elemento=dato)
 aux:=aux^.sig
FINMIENTRAS
DEVOLVER encontrado

FIN

ALGORITMO EnLista(lista:TipoLista;dato:TipoElemento):**B** (*solución estática recursiva *)

INICIO

SI lista=**NULO** **ENTONCES**
 DEVOLVER FALSE
ENOTROCASO
 SI Nodos[lista].elemento=dato **ENTONCES**
 DEVOLVER TRUE
 ENOTROCASO
 DEVOLVER EnLista(Nodos[lista].enlace)
 FINSI

FINSI

FIN

ALGORITMO EnLista(lista:TipoLista;dato:TipoElemento):**B** (*solución dinámica recursiva *)

INICIO

SI lista=**NIL** **ENTONCES**
 DEVOLVER FALSE
ENOTROCASO
 SI lista^.elemento=dato **ENTONCES**
 DEVOLVER TRUE
 ENOTROCASO
 DEVOLVER EnLista(lista^.sig)
 FINSI

FINSI

FIN

ALGORITMO InsertaDesp(VAR lista:TipoLista;dato_ant,nuevo_dato:TipoElem) (*solución estática iterativa *)

VARIABLES

anterior,nuevoNodo: TipoLista

INICIO (* Calculamos la posición de dato_ant *)

anterior:=lista

MIENTRAS Nodos[anterior].elemento<>dato_ant **HACER** (* suponemos que dato_ant existe *)

anterior:=Nodos[anterior].enlace

FINMIENTRAS

ObtenerNodo(nuevoNodo)

SI (nuevoNodo<> **Nulo**) **ENTONCES**

Nodos[nuevoNodo].elemento := elem

Nodos[nuevoNodo].enlace :=Nodos[anterior].enlace

Nodos[anterior].enlace := nuevoNodo

FINSI

FIN

ALGORITMO InsertaDesp(VAR lista:TipoLista;dato_ant,nuevo_dato:TipoElem) (*solución dinámica iterativa *)

VARIABLES

anterior,nuevoNodo: TipoLista

INICIO (* Calculamos la posición de dato_ant *)

anterior:=lista

MIENTRAS anterior^.elemento<>dato_ant **HACER** (* suponemos que dato_ant existe *)

anterior:=anterior^.sig

FINMIENTRAS

ASIGNAR(nuevoNodo,TAMAÑO(Nodo))

nuevoNodo^.elemento := elem

nuevoNodo^.sig :=anterior^.sig

anterior^.sig := nuevoNodo

FIN

ALGORITMO InsertaDesp(VAR lista:TipoLista;dato_ant,nuevo_dato:TipoElem) (*solución estática recursiva *)

VARIABLES

nuevoNodo: TipoLista

INICIO (* suponemos que dato_ant existe *)

SI Nodos[lista].elemento=dato_ant **ENTONCES**

ObtenerNodo(nuevoNodo)

SI (nuevoNodo<> **Nulo**) **ENTONCES**

Nodos[nuevoNodo].elemento := elem

Nodos[nuevoNodo].enlace :=Nodos[lista].enlace

Nodos[lista].enlace := nuevoNodo

FINSI

ENOTROCASO

InsertaDesp(Nodos[lista].enlace;dato_ant,nuevo_dato)

FINSI

FIN

ALGORITMO InsertaDesp(VAR lista:TipoLista;dato_ant,nuevo_dato:TipoElem) (*solución dinámica recursiva *)

VARIABLES

nuevoNodo: TipoLista

INICIO (* suponemos que dato_ant existe *)

SI lista^.elemento=dato_ant **HACER**

ASIGNAR(nuevoNodo,TAMAÑO(Nodo))

nuevoNodo^.elemento := elem

nuevoNodo^.sig :=lista^.sig

lista^.sig:=nuevoNodo

ENOTROCASO

InsertaDesp(lista^.sig;dato_ant,nuevo_dato)

FINSI

FIN

ALGORITMO BorraDesp(VAR lista:TipoLista;dato:TipoElem) (*solución estática iterativa *)

VARIABLES

ptrNodo,aux: TipoLista

```

INICIO (* Calculamos la posición de dato *)
    ptrNodo:=lista
MIENTRAS Nodos[ptrNodo].elemento<>dato HACER (* suponemos que dato existe *)
    ptrNodo:=Nodos[ptrNodo].enlace
FINMIENTRAS
SI Nodos[ptrNodo].enlace <> Nulo ENTONCES
    aux:=Nodos[ptrNodo].enlace
    Nodos[ptrNodo].enlace:= Nodos[aux].enlace
    LiberarNodo(aux)
FINSI
FIN
ALGORITMO BorraDesp(VAR lista:TipoLista;dato:TipoElem)    (*solución dinámica iterativa *)
VARIABLES
    ptrNodo,aux: TipoLista
INICIO (* Calculamos la posición de dato *)
    ptrNodo:=lista
MIENTRAS ptrNodo^.elemento<>dato HACER (* suponemos que dato existe *)
    ptrNodo:= ptrNodo ^.sig
FINMIENTRAS
SI ptrNodo ^.sig<>NIL ENTONCES
    aux:=ptrNodo^.sig
    ptrNodo^.sig:= aux ^.sig
    LIBERAR(aux,TAMAÑO(Nodo))
FINSI
FIN
ALGORITMO BorraDesp(VAR lista:TipoLista;dato:TipoElem)    (*solución estática recursiva *)
VARIABLES
    aux: TipoLista
INICIO (* suponemos que dato_ant existe *)
SI Nodos[lista].elemento=dato ENTONCES
    SI Nodos[lista].enlace <> Nulo ENTONCES
        aux:=Nodos[lista].enlace
        Nodos[lista].enlace := Nodos[aux].enlace
        LiberarNodo(aux)
    FINSI
ENOTROCASO
    BorraDesp(Nodos[lista].enlace;dato)
FINSI
FIN
ALGORITMO BorraDesp(VAR lista:TipoLista;dato:TipoElem)    (*solución dinámica recursiva *)
VARIABLES
    aux: TipoLista
INICIO (* suponemos que dato_ant existe *)
SI lista^.elemento=dato_ant HACER
    SI lista ^.sig<>NIL ENTONCES
        aux:=lista^.sig
        lista^.sig:= aux ^.sig
        LIBERAR(aux,TAMAÑO(Nodo))
    FINSI
ENOTROCASO
    BorraDesp(lista^.sig;dato)
FINSI
FIN

```

EJERCICIO 5.- Supongamos que tenemos una máquina con un solo registro y seis instrucciones:

LD	A	Sitúa el operando A en el registro
ST	A	Sitúa el contenido del registro en A
AD	A	Suma A al contenido del registro
SB	A	Resta A del contenido del registro
ML	A	Multiplica el contenido del registro por A
DV	A	Divide el contenido del registro por A

Escribe un programa que acepte una expresión en postfija que contenga operandos simples (formados por letras mayúsculas) y operadores (+, -, *, /), y que escriba en la pantalla la secuencia de instrucciones del tipo anterior para evaluar la expresión en dicha máquina, dejando el resultado en el registro. Por ejemplo, dada la expresión: ABC*+DE-, el programa debería dar como resultado:

```
LD B
ML C
ST Temp1
LD A
AD Temp1
ST Temp2
LD D
SB E
ST Temp3
LD Temp2
DV Temp3
ST Temp4
LD Temp4
```

ALGORITMO Maquina

DESDE Pila **IMPORTA** TipoPila, CrearPila, SacarPila, MeterPila, DestruirPila, PilaLlena, PilaVacía

CONSTANTES

MAX=80

FINCADENA=CHR(0)

TIPOS

Tcadena=ARRAY [1..MAX] DE \$

VARIABLES

operador:CONJUNTO DE \$

expresion:Tcadena

pila:TipoPila

x,temp:\$

i:N

error:B

INICIO

ESCRIBIR("Introduzca la expresión: ")

LEER(expresion)

operador:={'+', '-', '*', '/'}

i:=1

temp:='T' (* para que la pila sea de caracteres uso como temporal de la T en adelante *)

error:=FALSE

Pila:=CrearPila()

MIENTRAS (i<=MAX) **AND** (expresion[i]<>FINCADENA) **AND** (NOT error) **HACER**

SI NOT expresion[i] **EN** operador **ENTONCES**

SI PilaLlena(pila) **ENTONCES**

error:=TRUE

ESCRIBIR("¡¡ERROR!!, Se ha llenado la Pila...")

ENOTROCASO

MeterPila(pila,expresion[i])

FINSI

ENOTROCASO

SI PilaVacía(pila) **ENTONCES**

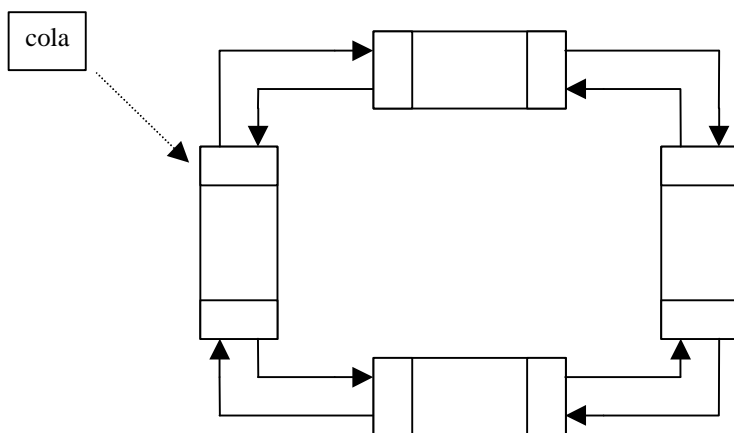
```

error:=TRUE
ESCRIBIR("¡¡ERROR!!, La expresión no es sintácticamente correcta")
ENOTROCASO
  SacarPila(pila,x) (* Saco el 1º operando *)
  Escribir("LD ",x)
  SI PilaVacía(pila) ENTONCES
    error:=TRUE
    ESCRIBIR("¡¡ERROR!!, La expresión no es sintácticamente correcta")
  ENOTROCASO
    SacarPila(pila,x) (* Saco el 2º operando *)
    CASO expresion[i] SEA
      '+': Escribir("AD ",x)
      '-': Escribir("SB ",x)
      '*': Escribir("ML ",x)
      '/': Escribir("DV ",x)
    FINCASO
    Escribir("ST ",temp)
    SI PilaLlena(pila) ENTONCES
      error:=TRUE
      ESCRIBIR("¡¡ERROR!!, Se ha llenado la Pila...")
    ENOTROCASO
      MeterPila(pila,temp)
      temp:=CHR(ORD(temp)+1) (* La siguiente letra *)
    FINSI
  FINSI
FINSI
FINSI
FINSI
i:=i+1
FINMIENTRAS
SI NOT error ENTONCES
  SI PilaVacía(pila) ENTONCES
    ESCRIBIR("¡¡ERROR!!, La expresión no es sintácticamente correcta")
  ENOTROCASO
    SacarPila(pila,x)
    ESCRIBIR("LD ",x)
    SI NOT PilaVacía(pila) ENTONCES
      ESCRIBIR("¡¡ERROR!!, La expresión no es sintácticamente correcta")
    FINSI
  FINSI
FINSI
FINSI
Destruir(pila)
FIN

```

EJERCICIO 8.-

Una cola de doble entrada es una colección de elementos homogéneos en la que se permiten inserciones y extracciones en ambos extremos. Diseña el TAD ColaDoble con las operaciones usuales.



MODULO ColaDoble

Definición

TIPOS

TColaD

TipoElemento = (* Cualquier tipo *)

Tlugar =(Frente,Final)

Algoritmo CrearCola(): TColaD

Algoritmo ColaVacia cola:TColaD): B

Algoritmo ColaLlena cola:TColaD):B

Algoritmo MeterCola(VAR cola: TColaD;elem: TipoElemento; pos:Tlugar)

Algoritmo SacarCola(VAR cola: TColaD; VAR elem: TipoElemento; pos:Tlugar)

Algoritmo DestruirCola(VAR cola:TColaD)

Implementación

TIPOS

TColaD = PUNTERO A TnodoD

TNodoD= REGISTRO

valor:TipoElem

ant,sig:TNodoD

FINREGISTRO

Algoritmo CrearCola(): TColaD

INICIO

DEVOLVER NIL

FIN

Algoritmo ColaVacia cola:TColaD): B

INICIO

DEVOLVER cola=NIL

FIN

Algoritmo ColaLlena cola:TColaD):B

INICIO

DEVOLVER FALSE

FIN

Algoritmo MeterCola(VAR cola: TColaD;elem: TipoElemento; pos:Tlugar)

VARIABLES

nuevoNodo:TipoCola

INICIO (* Notese que al ser una estructura circular, insertar por le frente es equivalente a insertar por el final y colocar la cabeza en el anterior *)

ASIGNAR(nuevoNodo,TAMAÑO(TNodoD))

nuevoNodo^.valor:=elem

```

SI ColaVacia cola ENTONCES
    ASIGNAR(cola, TAMAÑO(Tcabecera))
    cola:=nuevoNodo
    cola^.sig:=nuevoNodo
    cola^.ant:=nuevoNodo
ENOTROCASO
    nuevoNodo^.sig:=cola
    nuevoNodo^.ant:=cola^.ant
    cola^.ant^.sig:=nuevoNodo
    cola^.ant:=nuevoNodo
    SI pos =Frente ENTONCES
        cola:=cola^.ant
    FINSI
FINSI
FIN

```

Algoritmo SacarCola(VAR cola: TColaD; VAR elem: TipoElemento; pos: Tlugar)

VARIABLES

aux: TipoCola

INICIO (* Precondición: NOT ColaVacia(col) *)

(* Notese que al ser una estructura circular, eliminar por el final es equivalente a colocar la cabeza en el anterior y luego eliminar por el frente*)

```

SI cola =cola^.sig ENTONCES (*Solo hay un elemento *)
    elem:=cola^.valor (* Almaceno el valor a devolver *)
    LIBERAR(cola, TAMAÑO(TNodoD))

```

ENOTROCASO

```

SI pos =Final ENTONCES
    cola:=cola^.ant

```

FINSI

aux:=cola

elem:=aux^.valor (* Almaceno el valor a devolver *)

aux^.ant^.sig:=aux^.sig

aux^.sig^.ant:=aux^.ant

aux^.ant:=NIL

aux^.sig:=NIL

LIBERAR(aux, TAMAÑO(TNodoD))

FINSI

FIN

Algoritmo DestruirCola(VAR cola:TColaD)

VARIABLES

aux:TColaD

INICIO (* Precondición: NOT ColaVacia(col) . Hay por lo menos un nodo*)

cola^.ant^.sig:=NIL (* rompo el círculo *)

MIENTRAS cola<>NIL **HACER**

aux:=cola

cola:=cola^.sig

LIBERAR(aux, TAMAÑO(TNodoD))

FINMIENTRAS

FIN

EJERCICIO 14.- Una forma de codificar y decodificar mensajes con objeto de mantener su secreto consiste en usar el *método de la rejilla*. Una rejilla es una tarjeta perforada que, si se sitúa sobre una hoja de papel permite ver solo algunas porciones de la misma. En la codificación las letras del mensaje se escriben a través de dichas perforaciones, y en la decodificación del mensaje se irán leyendo sucesivamente aquellas letras que sean visibles a través de la rejilla.

Ejemplo:

```

123456
1 XX XX
2 X XXXX
3 X X XX
4 XXXXXX
5 XX X
6 XX XXX

```

Hay perforaciones en las posiciones (1,3) (1,6) (2,2) (3,2) (3,4) (5,1) (5,4) (5,6) y (6,3).

Se podrán usar rejillas cuadradas con un máximo de seis filas y seis columnas.

Para codificar un mensaje cada letra del mismo se irá situando en el papel en la posición que quede libre en la rejilla, comenzando por la posición (1,1)..(1,6),(2,1)..(2,6) etc. Puede ocurrir que aun queden letras por codificar cuando se alcance la última posición libre de la rejilla. En éste caso se girará la rejilla 90° en algún sentido (de las agujas del reloj ó el opuesto), y se repetirá el proceso (la rejilla está diseñada para que no haya posibilidad de superponer dos letras). Mientras queden caracteres del mensaje por codificar, seguiremos girando la rejilla en el sentido seleccionado inicialmente y repitiendo el proceso.

Para decodificar un mensaje se usará la rejilla, superponiéndola sobre la hoja con el mensaje codificado, y se leerán los caracteres a través de los huecos de la rejilla, siguiendo el mismo mecanismo empleado al codificar. La rejilla se girará siempre cuatro veces, ya que después se encontrará en la posición inicial.

Se pide diseñar e implementar un TAD rejilla con las operaciones adecuadas para codificar y decodificar mensajes, que serán:

Algoritmo CrearRejilla(VAR rejilla: TipoRejilla; Numhuecos:N)

(* Crea una rejilla con Numhuecos, cuyas coordenadas serán leídas de teclado *)

Algoritmo Sentido(VAR rejilla:TipoRejilla;sentido:B)

(* Establece el sentido en el que se girará la rejilla

Si sentido = TRUE => El de las agujas del reloj

sentido = FALSE => El opuesto a las agujas del reloj. *)

Algoritmo Codificar(rejilla:TipoRejilla)

(* Lee un texto normal de teclado y muestra por pantalla el resultado de su codificación según la rejilla)

Algoritmo Decodificar(rejilla: TipoRejilla)

(* Lee un texto codificado de teclado y muestra por pantalla el resultado de descifrarlo según la rejilla indicada *)

MODULO Rejilla

Definición (* mediante tipos transparentes *)

CONSTANTES

MAX=6

MAXCADENA=80*20 (* una página de texto *)

FINCADENA = CHR(0)

TIPOS

TipoRejilla = ARRAY [1..MAX][1..MAX] DE B

Tcadena=ARRAY [1..MAXCADENA] DE \$

ALGORITMO CrearRejilla(VAR rejilla: TipoRejilla; Numhuecos:N)

ALGORITMO Sentido(VAR rejilla:TipoRejilla;sentido:B)

ALGORITMO Codificar(rejilla:TipoRejilla)

ALGORITMO Decodificar(rejilla: TipoRejilla)

Implementación

ALGORITMO CrearRejilla(VAR rejilla: TipoRejilla; Numhuecos:N)

VARIABLES

n,i,j:N

INICIO

PARA i:=1 **HASTA** MAX **HACER**

PARA j:=1 **HASTA** MAX **HACER**

rejilla[i][j]:=FALSE

```

                FINPARA
    FINPARA
    PARA n:=1 HASTA Numhuecos HACER
        ESCRIBIR ("Hueco en : ("
        LEER(i)
        ESCRIBIR(",")
        LEER(j)
        ESCRIBIR(")")
        rejilla[i][j]:=TRUE
    FINPARA
FIN

```

ALGORITMO Sentido(VAR rejilla:TipoRejilla;sentido:B)

VARIABLES

i,j:N
aux:TipoRejilla

INICIO

```

    PARA i:=0 HASTA MAX HACER
        PARA j:=0 HASTA MAX HACER
            SI sentido ENTONCES
                aux[i][j]:=rejilla[j][MAX-i+1]
            ENOTROCASO
                aux[i][j]:=rejilla[i][MAX-j+1]
            FINSI
        FINPARA
    FINPARA
    rejilla:=aux

```

FIN

ALGORITMO Codificar(rejilla:TipoRejilla)

VARIABLES

entrada,salida:Tcadena
i,j:N

INICIO

```

    ESCRIBIR("Introduzca el Texto a codificar:")
    LEER(entrada)
    i:=1
    j:=1
    MIENTRAS (i<=MAXCADENA) AND (entrada[i]<>FINCADENA) HACER
        SI rejilla[1+ j DIV MAX][1+ j MOD MAX] ENTONCES (*hay un hueco *)
            salida[j]:=entrada[i]
            i:=i+1
        ENOTROCASO
            salida[j]=CHR(ORD('a') +RAND(26)) (* un carácter aleatorio entre a y z *)
        FINSI
        j:=j+1
        SI (j MOD MAX*MAX=0) ENTONCES
            Sentido(rejilla,TRUE)
        FINSI
    FINMIENTRAS
    SI j<MAXCADENA ENTONCES
        salida[j]:=FINCADENA
    FINSI
    ESCRIBIR("Texto a codificado:",salida)

```

FIN

ALGORITMO Decodificar(rejilla: TipoRejilla)

VARIABLES

entrada,salida:Tcadena
i,j:N

INICIO

ESCRIBIR("Introduzca el Texto a Decodificar:")

LEER(entrada)

i:=1

j:=1

MIENTRAS (j<=MAXCADENA) **AND** (entrada[j]<>FINCADENA) **HACER**

SI rejilla[1+j DIV MAX][1+j MOD MAX] **ENTONCES** (*hay un hueco *)

 salida[i]:=entrada[j]

 i:=i+1

ENOTROCASO

 (* Ignorar ese carácter *)

FINSI

 j:=j+1

SI (j MOD MAX*MAX=0) **ENTONCES**

 Sentido(rejilla,TRUE)

FINSI

FINMIENTRAS

SI i<MAXCADENA **ENTONCES**

 salida[i]:=FINCADENA

FINSI

ESCRIBIR("Texto decodificado:",salida)

FIN

EJERCICIO 15.-

Un *mapa* es una función de un conjunto llamado dominio en otro conjunto llamado rango. Así que un mapa define una colección dinámica de vinculaciones del dominio en el rango. A cada vinculación le llamamos lazo. Un número arbitrario de lazos puede crearse, modificarse y destruirse durante la vida de un mapa. Si no hay lazos, consideramos que el mapa está vacío.

a) Diseña un TAD Mapa donde el dominio es un tipo enumerado (e1, e2, ..., en) cualquiera y el rango es cualquier tipo de datos. Las operaciones a implementar son:

Algoritmo CrearMapa(VAR mapa:Mapa)

(* crea un mapa vacío *)

Algoritmo AñadirLazo(d:Dominio;r:Rango;VAR mapa:Mapa)

(* Añade un lazo al mapa *)

Algoritmo EliminarLazo(d:Dominio;VAR mapa:Mapa)

(* Elimina un lazo del mapa *)

Algoritmo EstaAsociado(d:Dominio;mapa:Mapa;VAR lazo:B;VAR r:Rango)

(* pone lazo a TRUE y r al valor del Rango asociado a d si existe tal lazo; en otro caso pone lazo a

FALSE *)

Algoritmo MapasIguales(mapa1,mapa2:Mapa):B

(* Determina si mapa1 y mapa2 son iguales *)

MODULO ModMapa

Definición

TIPOS

Dominio = (* Cualquier Enumerado *)

Rango = (* Cualquier Tipo *)

Mapa;

ALGORITMO CrearMapa(VAR mapa:Mapa)

ALGORITMO AñadirLazo(d:Dominio;r:Rango;VAR mapa:Mapa)

ALGORITMO EliminarLazo(d:Dominio;VAR mapa:Mapa)

ALGORITMO EstaAsociado(d:Dominio;mapa:Mapa;VAR lazo:B;VAR r:Rango)

ALGORITMO MapasIguales(mapa1,mapa2:Mapa):**B**

Implementación

TIPOS

Mapa = **PUNTERO A** NodoLazo

NodoLazo = **REGISTRO**

dom:Dominio

ran:Rango

sig:Mapa

FINREGISTRO

ALGORITMO CrearMapa(VAR mapa:Mapa)

INICIO

DEVOLVER NIL

FIN

ALGORITMO AñadirLazo(d:Dominio;r:Rango;VAR mapa:Mapa)

VARIABLES

nuevoLazo:Mapa

INICIO

ASIGNAR(nuevoLazo,**TAMAÑO**(NodoLazo))

nuevoLazo^.dom:=d

nuevoLazo^.ran:=r

nuevoLazo^.sig:=mapa

mapa:=nuevoLazo

FIN

ALGORITMO EliminarLazo(d:Dominio;VAR mapa:Mapa)

VARIABLES

ptrLazo:Mapa

INICIO

SI mapa<>NIL **ENTONCES**

SI mapa^.dom=d **ENTONCES**

ptrLazo:=mapa

mapa:=mapa^.sig

LIBERAR(ptrLazo,TAMAÑO(NodoLazo))

ENOTROCASO

EliminarLazo(d,mapa^.sig)

FINSI

FINSI

FIN

ALGORITMO EstaAsociado(d:Dominio;mapa:Mapa;VAR lazo:B;VAR r:Rango)

INICIO

SI mapa=NIL **ENTONCES**

lazo:=FALSE

ENOTROCASO

SI mapa^.dom=d **ENTONCES**

lazo:=TRUE

r:=mapa^.ran

ENOTROCASO

EstaAsociado(d,mapa^.sig,lazo,r)

FINSI

FINSI

FIN

ALGORITMO MapasIguales(mapa1,mapa2:Mapa):B

INICIO

SI (mapa1=NIL) **OR** (mapa2=NIL) **ENTONCES**

DEVOLVER mapa1=mapa2

ENOTROCASO

SI (mapa1^.dom<>mapa2^.dom) **OR** (mapa2^.ran<>mapa2^.ran) **ENTONCES**

DEVOLVER FALSE

ENOTROCASO

DEVOLVER MapasIguales(mapa1^.sig,mapa2^.sig)

FINSI

FINSI

FIN

FIN (* Modulo ModMapa *)

b) Si en el TAD Mapa Dominio = Color = (blanca, roja, negra) y Rango = CARDINAL, podemos implementar el TAD Urna. Una variable del tipo Urna representa una urna con un número cualquiera de bolas blancas, rojas y negras. Este número es el elemento del rango asociado al color correspondiente en el dominio. Por ejemplo, una urna con dos bolas negras y una blanca estaría representada por un mapa con los lazos (negra,2) y (blanca,1).

Utilizando los procedimientos ya implementados en el TAD mapa, diseña:

Algoritmo AñadirBola(c:Color;VAR urna:Urna)

(* Añade una bola de color c a la urna*)

Algoritmo SacarBola(VAR urna:Urna;VAR c:Color)

(* Extrae una bola de la urna. Supón que existe una función Aleatorio():Color que devuelve un color de forma aleatoria al invocarla*)

DESDE ModMapa **IMPORTA** Mapa, CrearMapa, AñadirLazo, EliminarLazo, EstaAsociado
TIPOS

Dominio= Color
Rango = N
Urna = Mapa

ALGORITMO AñadirBola(c:Color;VAR urna:Urna)

VARIABLES

numBolas:N
hayBolas:B

INICIO

EstaAsociado(c,urna,hayBolas,numBolas) **ENTONCES**
SI NOT hayBolas **ENTONCES**
 numBolas:=1

ENOTROCASO

 numBolas:=numBolas+1
 EliminarLazo(c,urna)

FINSI

AñadirLazo(c,numBolas,urna)

FIN

ALGORITMO SacarBola(VAR urna:Urna;VAR c:Color)

VARIABLES

numBolas:N
hayBolas:B

INICIO (* Suponemos que la urna no está vacía *)

REPETIR

 c:=Aleatorio()
 EstaAsociado(c,urna,hayBolas,numBolas)

HASTA QUE hayBolas

numBolas:=numBolas-1

EliminarLazo(c,urna)

SI numBolas > 0 **ENTONCES**

 AñadirLazo(c,numBolas,urna)

FINSI

FIN

RELACION DE FICHEROS

Ejercicio 3.-

Diseñar un algoritmo que recoja del teclado un valor numérico entero y el nombre de un fichero binario conteniendo datos enteros; y que busque en dicho fichero el valor dado. En caso de encontrarlo el algoritmo deberá indicar el lugar (posición del registro) en el que se ha encontrado por primera vez.

ALGORITMO BuscaZ

VARIABLES

fich:**FICHERO**

name:Tcadena

busco,x:**Z**

pos:**N**

encontrado:**B**

INICIO

ESCRIBIR(“Nombre del Fichero: “)

LEER(name)

ESCRIBIR(“Entero a Buscar:”)

LEER(busco)

encontrado:=**FALSE**

fich:=**ABRIR**(name)

LEERBIN(fich,x)

MIENTRAS (**NOT** encontrado) **AND** (**NOT EOF**(fich)) **HACER**

SI x=busco **ENTONCES**

 encontrado:=**TRUE**

 pos=**POSICION**(fich) – **TAMAÑO**(x)

FINSI

 LEERBIN(fich,x)

FINMIENTRAS

CERRAR(fich)

SI encontrado **ENTONCES**

 ESCRIBIR(“Encontrado en la posición: “,pos)

ENOTROCASO

 ESCRIBIR(busco,“ NO ha sido Encontrado”)

FINSI

FIN