ASIGNATURA		CURSO	CALIFICACIÓN
ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN		**2004 / 2005**	
TITULACIÓN	GRUPO	CONVOCATORIA	
		EXTRAORDINARIA – DIC.	
APELLIDOS		NOMBRE	

EJERCICIO 1 (3 PUNTOS)

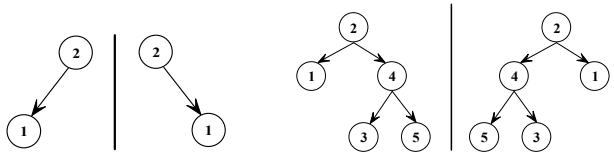
Dado el tipo abstracto de datos (TAD) `tArbolBin` que sirve para representar árboles binarios de enteros, del que sólo se conoce la parte de la interfaz siguiente:

```
type
   tInfo = integer;
   tArbolBin = …

   function EsVacio   (a: tArbolBin): boolean;
   function Raiz      (a: tArbolBin): tInfo;
   function RamaIzqda (a: tArbolBin): tArbolBin;
   function RamaDcha  (a: tArbolBin): tArbolBin;
```

A) Definir una función que devuelva el menor valor entero almacenado en un árbol binario no vacío:

```
function ValorMinimo (a:tArbolBin): tInfo;
```

B) Dados dos árboles A y B, se dice que B es el árbol especular de A si coincide con la imagen de A reflejada en un espejo. Así, los siguientes árboles son especulares el uno del otro:

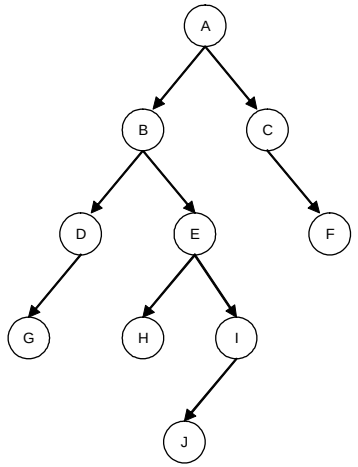


Escribir una función que dados dos árboles A y B nos indique si B es el árbol especular de A.

```
function EsEspecularDe (a,b:tArbolBin): boolean;
```

EJERCICIO 2 (3 PUNTOS)

A) Sea el árbol AVL siguiente:



resultado de insertar el nodo J, y para el cual no se conocen los valores de las claves.

Determinar si el árbol está o no balanceado, y en ese caso reestructurarlo, indicando los factores de equilibrio de cada nodo y la rotación efectuada, dibujando el estado del árbol después de cada operación.

B) Sobre el árbol resultado del apartado anterior efectuar la operación de eliminación del nodo B. Reestructurar el árbol en caso de ser necesario.

C) Determinar si las siguientes afirmaciones son ciertas o falsas, **justificando la respuesta**:

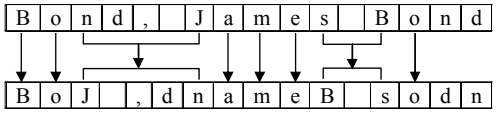
1. Un montículo es un árbol binario de búsqueda.
2. La diferencia de altura entre los subárboles izquierdo y derecho de cualquier nodo en un árbol binario de búsqueda no debe ser superior a 1.
3. El recorrido inorden de un montículo accede a los elementos en orden creciente.
4. Una cola de prioridad se comporta en ocasiones como una cola estándar.

EJERCICIO 3 (4 PUNTOS)

El agente 0069 ha inventado un método de codificación de mensajes secretos. Este método consiste en los pasos siguientes:

1. Se recorre el texto a codificar de principio a fin
2. Cuando se encuentra una vocal, ésta pasa directamente al texto codificado de salida
3. Cuando se encuentra una tira de caracteres no vocales se sustituyen en el texto de salida por la misma secuencia pero invertida.

Así, por ejemplo el testo de entrada "Bond, James Bond" se transforma del modo siguiente:



Se pide:

A) La estructura para almacenar tanto el texto original como el resultado codificado será una cola. Realizar, utilizando el lenguaje Pascal:

(1) La definición de tipos del TDA Cola. **Justifica** la elección de la implementación (estática o dinámica).

(2) La implementación de las operaciones básicas definidas en la especificación del tipo Cola:

Cola_Vacía () → *Cola*
{Objetivo: Crear una cola vacía}

Meter_Cola (*Info, Cola*) → *Cola*
{Objetivo: Insertar un nodo con cierta información en la cola
PreCond: se supone memoria suficiente para realizar la inserción }

Sacar_Cola (*Cola*) → *Info, Cola*
{Objetivo: Eliminar un elemento de la cola
Salida: *Info*: contenido del elemento eliminado
Cola sin el elemento eliminado.
PreCond: La *Cola* no está vacía}

Es_Cola_Vacia (*Cola*) → *Boolean*
{Objetivo: Determinar si una cola está vacía}

B) Consideremos ya implementado un TDA Pila en el cual el campo de información de los nodos es del mismo tipo que el empleado en la definición del TDA Cola y del que sólo se conoce la parte de la interfaz siguiente:

Pila_Vacía () → *Pila*
{Objetivo: Crear una pila vacía }

Meter_Pila (*Pila, Info*) → *Pila*
{Objetivo: Añadir un nodo con cierta información a la Pila
Entrada: *Pila*: acceso a la pila
Info: contenido del elemento a insertar
Salida : *Pila* con el elemento añadido
PreCond: La *Pila* debe estar inicializada
Se supone memoria suficiente para realizar la inserción }

Sacar_Pila (*Pila*) → *Info, Pila*
{Objetivo: Eliminar un elemento de la pila
Entrada: *Pila*: acceso a la pila
Salida: *Info*: contenido del elemento eliminado de la *Pila*
La pila *Pila* sin el elemento eliminado
PreCond: La *Pila* debe estar inicializada y no vacía}

Es_Pila_Vacia (*Pila*) → *Boolean*
{Objetivo: determinar si una pila está vacía}

Utilizando el TDA Cola y el TDA Pila escribir la rutina que codifique un mensaje de acuerdo a la especificación siguiente:

Mensaje_Secreto (Cola1) → Cola2
{Objetivo: obtener un mensaje codificado de acuerdo al algoritmo del agente 0069
Entrada: Cola1: cola que contiene el Mensaje de entrada
Salida: Cola2: cola que contiene el Mensaje codificado de salida}

Soluciones

EJERCICIO 1

A)

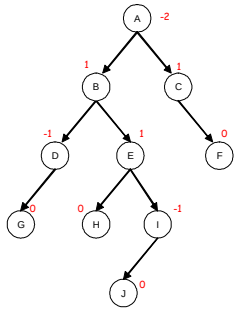
```
function EsEspecularDe (A, B: arbolbin): boolean;
begin
   if EsVacio(A) and EsVacio(B)
   then EsEspecularDe := true
   else if EsVacio(A) or EsVacio(B)
   then EsEspecularDe := false
      else EsEspecularDe := (Raiz(A) = Raiz(B)) and
                  EsEspecularDe (RamzIzqda(A), RamaDrcha(B)) and
                  EsEspecularDe (RamzDrcha(A), RamaIzqda(B));

end;
```

B)

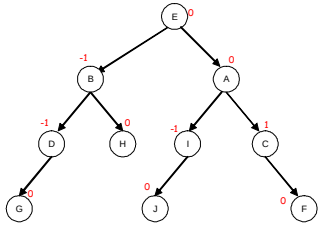
```
Function ValorMinimo (a:tArbolBin):integer;

  Function min (x,y:integer):integer;
  begin
    if x<y
       then min := x
    else min := y
  end;

begin
  if EsArbolVacio(RamaIzqda(a)) and EsArbolVacio(RamaDcha(a))
  then ValorMinimo := Raiz(a)
  else if EsArbolVacio(RamaIzqda(a))
       then ValorMinimo := min(Raiz(a), ValorMinimo(RamaDcha(a))
       else if EsArbolVacio(RamaDcha(a))
            then ValorMinimo := min(Raiz(a), ValorMinimo(RamaIzqda(a))
            else ValorMinimo := min(Raiz(a),
               min(ValorMinimo(RamaIzqda(a)),ValorMinimo(RamaDcha(a))))
end;
```

EJERCICIO 2

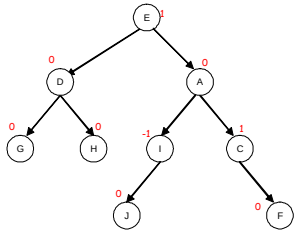
A) El árbol AVL siguiente está desequilibrado:

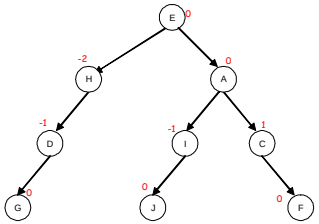


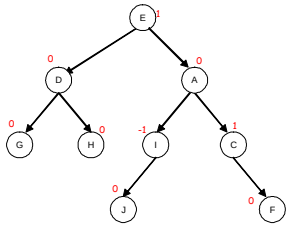
Se reequilibra realizando una rotación ID, con el resultado



B) Se borra la clave B. Se puede sustituir por la mayor de las claves menores (solución más lógica porque no hay que realizar rotaciones), o por la menor de las claves mayores (habría que realizar una rotación II)







C) Verdadero o falso y por qué.

I. Un montículo es un árbol binario de búsqueda. Falso. En un montículo MAX para cualquier nodo se cumple que el valor de su clave es mayor que cualquier otra clave de sus descendientes (o menor si es un montículo MIN). En un árbol binario de búsqueda, para todo nodo se cumple que el valor de su clave es mayor (o menor) que cualquier clave de cualquier nodo del subárbol izquierdo y menor (o mayor) que cualquier clave de cualquier nodo del subárbol derecho.

II. La diferencia de altura entre los subárboles izquierdo y derecho de cualquier nodo en un árbol binario de búsqueda no debe ser superior a 1. Falso. Un árbol binario de búsqueda no tiene por que estar equilibrado.

III. Un recorrido inorden en un montículo MAX accede a los elementos del árbol en orden creciente. Falso. Un montículo MAX sólo nos asegura que la clave de un nodo padre es mayor que la de sus hijos, pero no establece el lugar donde almacenar cada clave: si a la izquierda o a la derecha. Es al borrar elementos de un montículo cuando SI obtenemos una lista decreciente.

IV. Una cola de prioridad se comporta en ocasiones como una cola estándar. Cierto, sólo en el caso de que los elementos tengan todos la misma prioridad.

EJERCICIO 3

A)

```
UNIT TDA_cola;
INTERFACE
    const   nulo = nil;
    type
        tinfo = char;
        tPos  = ^tNodo;
        tNodo = record
                Info: tinfo;
                sig: tPos  ;
        end;
        tCola = record
                Ini, Fin: tPos;
        end;

    procedure Cola_Vacia (var Cola: tCola);
    function Es_cola_vacia (Cola: tCola): boolean;
    procedure Meter_Cola (x:tinfo; var Cola: tCola );
    procedure Sacar_Cola (var x: tinfo; var Cola: tCola);


IMPLEMENTATION

    Procedure Cola_Vacía (var Cola: tCola);
    begin
        with Cola do begin
                Ini:=nulo;
                Fin:=nulo;
        end
    end;

    function Es_cola_vacia (Cola: tCola): boolean;
    begin
        Es_cola_vacia:= Cola.Ini = nulo;
    end;

    procedure Crear_nodo (x: tinfo; var nuevo: tPos );
    {Precond: se supone memoria suficiente para crear la variable}
    begin
        new(nuevo);
        nuevo ^.info:=x;
        nuevo ^.sig:=nulo;
    end;

    procedure Meter_Cola (x:tinfo; var Cola: tCola );
    var     nuevo: tPos ;
    begin
        Crear_nodo (x, nuevo);
        with Cola do begin
                if Es_vacia(Cola)
                then Ini:= nuevo
                else Fin^.sig:= nuevo;
                Fin:= nuevo;
        end
    end;


    procedure Sacar_Cola (var x:tInfo; var Cola: tCola );
    var     auxt: tPos
    begin
        with Cola do begin
                x:= Ini^.info;
                aux:= Ini;
                Ini:=Ini^.sig;
                If Ini:=nulo then Fin:=nulo; {si la cola se queda vacia la inicializo}
                dispose(aux);
        end
    end;
end.
```

B)

```
Function Es_vocal (c:char): boolean;
Begin
        Es_vocal:= c in ['a','e','I','o','u']
End;

Procedure Invertir(var Entrada: tPila; var Salida: Tcola);
Var
    letra: char;
Begin
    while not Es_pila_vacia(Entrada) do
    begin
        Sacar_Pila(letra, Entrada);
        Meter_Cola (Salida, letra);
    end
End;

Procedure Mensaje_Secreto (Entrada: tCola; var Salida: tCola);
var
    letra: char;
    pila:TPila;
begin
    Cola_Vacia(Salida);
    Pila_Vacia(Pila);
    while not Es_cola_vacia(Entrada) do
    begin
        letra := Sacar_Cola(letra, Entrada);
        if Es_vocal(letra)
        then begin
            Invertir(Pila, Salida);
            Meter_Cola (cola, letra)
        End
        Else Meter_Pila (pila, letra);
    end;
    Invertir(Pila, Salida);
end;
```