Apellidos:
Nombre:
Convocatoria:
DNI:

Examen TAD/PED septiembre 2004
Modalidad 0

Normas:
- La entrega del test no corre convocatoria.
- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
- **El test vale un 40% de la nota de teoría.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Las operaciones modificadoras de un TAD permiten generar, por aplicaciones sucesivas, todos los valores del TAD a especificar.	❑	❑	F	1.
Sea el método Primera perteneciente a la clase Tlista que devuelve la primera posición de la lista que lo invoca: *TPosicion Tlista::Primera( )* *class Tlista {* *{ TPosicion p;* *public: ...* *p.pos = lis;* *private:* *return p; }* *Tnodo *lis; }* En el método Primera, se invoca al constructor de Tlista.	❑	❑	F	2.
Sea el tipo cola definido en clase. La semántica de la operación cabeza es la siguiente: Var c:cola; x:item; cabeza(crear_cola())=error_item() si esvacia(c) entonces cabeza(encolar(c,x))=x sino cabeza(encolar(c,x)))=encolar(cabeza(c),x)	❑	❑	F	3.
El recorrido en postorden de un árbol binario es el inverso especular del recorrido en preorden del mismo árbol	❑	❑	V	4.
En la operación de borrado de un ítem en un árbol AVL, si se realiza una rotación II, al menos es necesario realizar otra rotación de cualquier tipo.	❑	❑	F	5.
Los nodos de grado 0 de un árbol 2-3 han de estar en el mismo nivel del árbol	❑	❑	V	6.
Al insertar un elemento en un árbol 2-3-4 se pueden realizar una operación de DIVIDERAIZ y otra de DIVIDEHIJODE2.	❑	❑	V	7.
En un árbol rojo-negro ha de haber al menos un enlace rojo.	❑	❑	F	8.
En un árbol B tiene que haber el mismo número de nodos en el hijo izquierdo de la raíz que en el hijo derecho.	❑	❑	F	9.
La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto multiplicado por 3 (C: Conjunto; x: Ítem): Operación(Crear) $\Leftrightarrow$ 0 Operación (Insertar(C, x)) $\Leftrightarrow$ 3 + Operación(C)	❑	❑	V	10.
En la dispersión abierta se pueden producir colisiones entre claves sinónimas y no sinónimas	❑	❑	F	11.
Un recorrido en inorden de un montículo nos devolverá todos los elementos de forma ordenada.	❑	❑	F	12.
Un árbol 2-3 cumple las propiedades de un árbol Leftist.	❑	❑	F	13.
Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de cruce es un grafo acíclico dirigido.	❑	❑	V	14.

Examen TAD/PED septiembre 2004

Normas:
- Tiempo para efectuar el ejercicio: **2 horas**
- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: ***Apellidos, Nombre***. Cada pregunta se escribirá en folios diferentes.
- Se dispone de 20 minutos para abandonar el examen sin que corra convocatoria.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible
- **Todas las preguntas tienen el mismo valor.** Este examen vale el 60% de la nota de teoría.
- **Publicación de notas de exámenes y prácticas**: 20 de septiembre. **ÚNICA fecha de revisión de exámenes:** 22 de septiembre. El lugar y la hora se publicará en el campus virtual. **EXAMEN DE PRÁCTICAS:** 20 de septiembre. El lugar y la hora se publicará en el campus virtual
- **Los alumnos que estén en 5ª o 6ª convocatoria deben indicarlo en la cabecera de todas las hojas**

1. Un palíndromo es una palabra o frase que se lee igual de izquierda a derecha, que de derecha a izquierda. Proporciona la sintaxis y semántica de la operación *palíndromo*, que actúa sobre una lista cuyos elementos son caracteres y devuelve CIERTO si el contenido de la lista es un palíndromo y FALSO en caso contrario. Se considera que una lista vacía o de un solo elemento son palíndromos. Sólo se pueden emplear las operaciones de la especificación básica de listas vista en clase.

2. Se tiene una clase *TMatriz*, definida de la siguiente forma en el archivo de cabeceras (*TMatriz.h*), que está construida por *layering* a partir de *TVector*, definida en *TVector.h*:

```
#ifndef _TMATRIZ_
#define _TMATRIZ_
#include "tvector.h"

class TMatriz
{
  friend ostream& operator<<(ostream &, TMatriz &);
  public:
         TMatriz(int,int); //CONSTRUCTOR
         ~TMatriz(); //DESTRUCTOR
         TMatriz(const TMatriz &); //CONSTRUCTOR DE COPIA
         TMatriz& operator=(TMatriz &);
         TVector& operator[](int);
  private:
         int nfilas,ncolumnas;
         TVector *filas;
         TVector vector_error; //TRATAMIENTO DE ERRORES
};
#endif
```

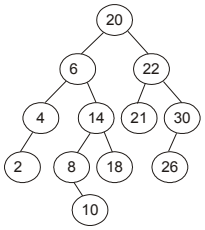
```
#ifndef _TVECTOR_
#define _TVECTOR_
class TVector
{
  friend ostream& operator<<(ostream &, TVector &);
  public:
         TVector(int x=10); //CONSTRUCTOR POR DEFECTO DIMENSION 10
         ~TVector(); //DESTRUCTOR
         TVector(const TVector &); //CONSTRUCTOR DE COPIA
         TVector& operator=(TVector &);
         int& operator[](int);
  private:
         int dimension;
         int* datos;
         int error; //TRATAMIENTO DE ERRORES
};
#endif
```

Se pide implementar los siguientes métodos para la clase *TMatriz***:**
- *TMatriz(int,int); //CONSTRUCTOR*
- *TMatriz(const TMatriz &); //CONSTRUCTOR DE COPIA*
- *TVector& operator[](int);*

Comentarios:
- Utiliza por *layering* **TVector vector_error** para devolver el vector por defecto
- Señala dónde se emplea el *layering* en la implementación de estos 3 métodos, indicando brevemente en qué consiste.
- El rango de las FILAS de *TMatriz* es [1..i] , la de COLUMNAS (posiciones del vector) es [1..j] ; es decir, empieza en "1" y no en "0"

3. a) En un árbol AVL inicialmente vacío, insertar los siguientes elementos: 10, 3, 7, 9, 11, 15, 2, 1, 4, 5.

b) En el siguiente árbol AVL borrar los siguientes elementos: 26, 18.



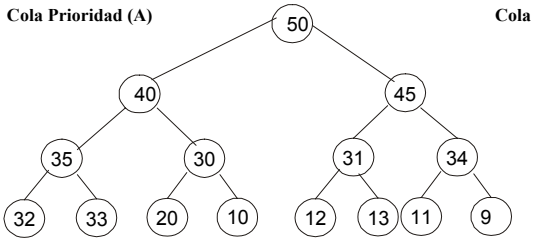
4.

a) Escribir el código del método *THeap THeap::Combinar(THeap& b)* que realice la combinación de dos colas de prioridad máximas representadas mediante montículos simples. El resultado se devolverá por valor sin modificar ninguno de los dos montículos. Para ello, utilizar las clases y funciones que se indican a continuación. En caso que sea necesario se pueden utilizar funciones auxiliares. NOTA: los errores de sintaxis se valorarán negativamente.

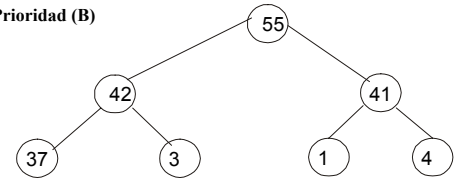
```
THeap {
public:
     THeap(THeap&);                        // Constructor de copia
     void Insertar(TElemento x);           // Inserta "x" en caso que no exista previamente en la cola
     TElemento Borrar();                   // Devuelve el máximo de la cola, realizando su borrado
     TElemento Máximo();                   // Devuelve el máximo de la cola
     Bool Vacio();                         // Devuelve "true" si no contiene ningún elemento
     … };
```

b) Realizar la combinación de los dos montículos que aparecen a continuación (*A.Combinar(B)*) utilizando el algoritmo escrito en el apartado anterior.

Cola Prioridad (A)



Cola Prioridad (B)



Examen TAD/PED septiembre 2004. Soluciones

1)

Sintaxis:

palindromo(lista) → bool

Semántica:

VAR x: caracter; l: lista;

palindromo(crear()) = CIERTO
palindromo(inscabeza(crear(), x)) = CIERTO

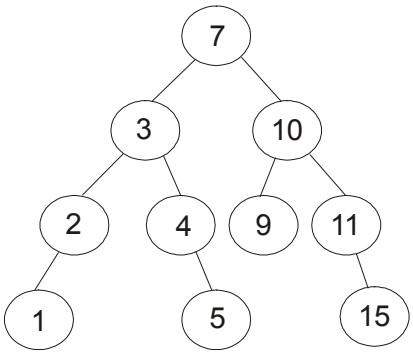
si x == obtener(l, ultima(l)) entonces
palindromo(inscabeza(l, x)) = palindromo(borrar(l, ultima(l)))
sino
palindromo(inscabeza(l, x)) = FALSO

2)

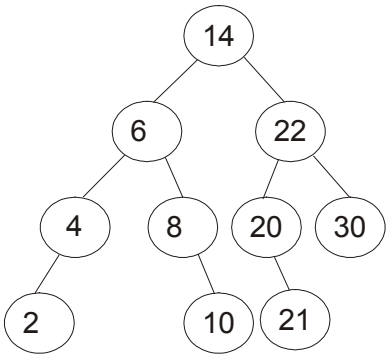
```
/*//////// Constructor de la matriz por layering a partir del TVector.
  IMPORTANTE: llamada al constructor del TVector para vector_error
*/
TMatriz::TMatriz(int f, int c):vector_error() //  LAYERING : constructor defecto
{
  nfilas = f;
  ncolumnas = c;
  filas = new TVector[nfilas] (ncolumnas); //OJO A ESTA LLAMADA
  for ( int i=0; i<nfilas; i++ )
    for ( int j=1; j<=ncolumnas; j++ )
      filas[i][j] = 0;    ];    // LAYERING: corchete vector
}
/*************************************************************************/

/*//////// Constructor de copia de la matriz.
  IMPORTANTE: llamada al constructor de copia del TVector para vector_error
*/
TMatriz::TMatriz(const TMatriz &origen):vector_error(origen.vector_error) // LAYERING: constructor copia
vector
{
  nfilas = origen.nfilas; // LAYERING: constructor copia
  ncolumnas = origen.ncolumnas; // LAYERING: constructor copia vector

  filas = new TVector[nfilas] (ncolumnas); //OJO A ESTA LLAMADA
  for ( int i=0; i<nfilas; i++ )
    for ( int j=1; j<=ncolumnas; j++ )
      filas[i][j] = origen.filas[i][j];    // LAYERING: corchete vector

}
/*************************************************************************/

TVector&
TMatriz::operator[](int f)
{
  if(f >=1 && f <= nfilas) return (filas[f-1]); // RANGO : 1..i

  else {
          cout<<"Fila "<<f<<" invalida para esta matriz"<<endl;
          return(vector_error);
  }
}
/*************************************************************************/
```

3)

a)



b)



4) **a)**

```
THeap
THeap::Combinar(THeap& b) {
        THeap temp(*this);
        THeap bTemp(b);
        while(!bTemp.Vacio())
                temp.Insertar(bTemp.Borrar());
        return temp;
}
```

b)