Programación y Estructuras de Datos

CUADERNILLO 2

(curso 2010-2011, 2º. Cuatrimestre)

Parte 1 : clase base : "TABBCalendario"

Qué se pide :

En esta parte se pide implementar una clase que representa **árbol binario de búsqueda (ABB)** de objetos de tipo **TCalendario** ("**TABBCalendario"**).

Para ello, la clase del árbol ABB a desarrollar emplearán las clases **TCalendario**, **TVectorCalendario y TListaCalendario** desarrolladas en el **Cuadernillo 1**.

Para representar cada nodo del ABB, se tiene que definir la clase **TNodoABB** con su forma canónica (constructor, constructor de copia, destructor y sobrecarga del operador asignación) como mínimo.

Cuándo se exige:

Al alumno se le podrá exigir tenerlo terminado y funcionando a partir del día: **lunes 14/03/2011**

Prototipo de la Clase:

PARTE PRIVADA de TNodoABB

```
// Elemento (etiqueta) del nodo
TCalendario item;
// Subárbol izquierdo y derecho
TABBCalendario iz, de;
```

FORMA CANÓNICA de TNodoABB

```
//Constructor por defecto
TNodoABB ();
//Constructor copia
TNodoABB (TNodoABB &);
//Destructor
~TNodoABB ();
// Sobrecarga del operador asignación
TNodoABB & operator=( TNodoABB &);
```

// PARTE PRIVADA de TABBCalendario

```
// Puntero al nodo raíz
TNodoABB *raiz;
Para reducir el coste del cálculo de los recorridos en el árbol ABB, hace falta añadir los
siguientes métodos a la PARTE
PRIVADA de la clase (ver "Aclaraciones", detrás de la lista de métodos requeridos ) :
// Devuelve el recorrido en inorden
void InordenAux(TVectorCalendario &, int &);
// Devuelve el recorrido en preorden
void PreordenAux(TVectorCalendario &, int &);
// Devuelve el recorrido en postorden
void PostordenAux(TVectorCalendario &, int &);
```

// FORMA CANÓNICA de TABBCalendario

```
// Constructor por defecto
TABBCalendario ();
// Constructor de copia
TABBCalendario (TABBCalendario &);
// Destructor
~TABBCalendario ();
// Sobrecarga del operador asignación
TABBCalendario & operator=(TABBCalendario &);
```

// MÉTODOS de TABBCalendario

```
// Sobrecarga del operador igualdad
bool operator==( TABBCalendario &);
// Devuelve TRUE si el árbol está vacío, FALSE en caso contrario
bool EsVacio();
// Inserta el elemento en el árbol
bool Insertar(TCalendario &);
// Borra el elemento en el árbol
bool Borrar(TCalendario &);
// Devuelve TRUE si el elemento está en el árbol, FALSE en caso contrario
bool Buscar(TCalendario &);
// Devuelve el elemento en la raíz del árbol
TCalendario Raiz();
// Devuelve la altura del árbol (la altura de un árbol vacío es 0)
int Altura();
// Devuelve el número de nodos del árbol (un árbol vacío posee 0 nodos)
int Nodos();
// Devuelve el número de nodos hoja en el árbol (la raíz puede ser nodo hoja)
int NodosHoja();
// Devuelve el recorrido en inorden sobre un vector
TVectorCalendario Inorden();
// Devuelve el recorrido en preorden sobre un vector
TVectorCalendario Preorden();
// Devuelve el recorrido en postorden sobre un vector
TVectorCalendario Postorden();
// Suma de dos ABB
TABBCalendario operator+( TABBCalendario &);
// Resta de dos ABB
TABBCalendario operator-( TABBCalendario &);

// Búsqueda en un ABB a través de una lista auxiliar
int*  BuscarLista( TListaCalendario &);

// Sobrecarga del operador salida
friend ostream & operator<<(ostream &, TABBCalendario &);
```

Aclaraciones :

• La forma de emplear los métodos auxiliares de ordenación de la PARTE PRIVADA es (por ejemplo, para el caso del recorrido en inorden):

```
// Devuelve el recorrido en inorden
TVectorCalendario TABBCalendario::Inorden()
{
int posición = 1;
TVectorCalendario recorrido(Nodos());
InordenAux(recorrido, posicion);
return recorrido;
}
```

De este modo, se reduce el coste de crear múltiples objetos de tipo **TVectorCalendario**, ya que sólo se emplea uno durante todo el cálculo del recorrido.

• El criterio de ordenación para **TCalendario** sigue siendo el que se pide en el **Cuadernillo1.**

• Para simplificar los algoritmos, el árbol NO puede contener elementos repetidos.

• El constructor de copia tiene que realizar una copia exacta.

• El destructor tiene que liberar toda la memoria que ocupe el árbol.

• Si se asigna un árbol a un árbol no vacío, se destruye el árbol inicial. La asignación tiene que realizar una copia exacta.

• En el **operator==**, dos árboles son iguales si poseen los mismos elementos independientemente de la estructura interna del árbol.

• **Insertar(TCalendario &)**, devuelve TRUE si el elemento se puede insertar y FALSE en caso contrario (por ejemplo, porque ya existe en el árbol).

• En **Borrar(TCalendario &)**, devuelve TRUE si el elemento se puede borrar y FALSE en caso contrario (por ejemplo, porque no existe en el árbol). El criterio de borrado es sustituir por el mayor de la izquierda.

• En **Raiz()**, si el árbol está vacío, se tiene que devolver un **TCalendario** vacío.

• Los **3 recorridos** devuelven un vector (**TVectorCalendario**). Si el árbol está vacío, se devuelve un vector vacío (dimensión 0).

En la declaración de los 3 recorridos, se recomienda que el valor entero de la posición se pase por REFERENCIA.

• En **operator+**, primero se tiene que sacar una copia del operando (árbol) de la izquierda y a continuación insertar los elementos del operando (árbol) de la derecha según su recorrido por INORDEN.

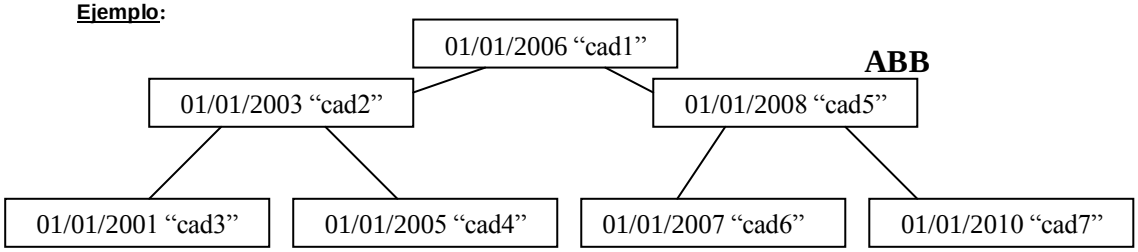
• En **operator-**, se recorre el operando (árbol) de la izquierda por INORDEN y si el elemento NO está en el operando (árbol) de la derecha, se inserta en el árbol resultante (inicialmente vacío) y el proceso se repite para todos los elementos del operando de la izquierda.

• Se permite amistad entre las clases **TABBCalendario** y **TNodoABB**.

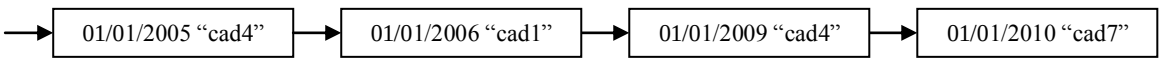
• El operador SALIDA muestra el RECORRIDO POR INORDEN del ABB, con el formato pedido en el Cuadernillo 1 para la clase **TVectorCalendario**.

• En **BuscarLista(TListaCalendario &) :**
Devuelve un vector de enteros del mismo tamaño que la lista pasada como parámetro. Para cada **TCalendario** de la lista, se busca en el **ABB**; si se encuentra se escribe exclusivamente en la posición correspondiente del vector, el año del nodo anterior según el recorrido Inorden. Si no se encuentra en el **ABB** o no tiene anterior según el recorrido Inorden se pone un 0 en la posición que corresponda del vector. La lista pasada como parámetro se recorre de izquierda a derecha.

Ejemplo:



TListaCalendario (parámetro)



vector de enteros devuelto

2003	2005	0	2008

Parte 2 **: clase base : "TAVLCalendario"**

Qué se pide :

En esta parte se pide implementar una clase que representa un **árbol equilibrado respecto a la altura (árbol AVL)** de objetos de tipo **TCalendario** ("**TAVLCalendario**").

Para ello, la clase del árbol AVL a desarrollar empleará las clases **TCalendario** y **TVectorCalendario** desarrolladas en el **Cuadernillo 1.**

Para representar cada nodo del árbol, se tiene que definir la clase **TNodoAVL** con su forma canónica (constructor, constructor de copia, destructor y sobrecarga del operador asignación) como mínimo.

Cuándo se exige:

Al alumno se le podrá exigir tenerlo terminado y funcionando a partir del día: **lunes 09/05/2011**

Prototipo de las clases básicas : " **TNodoAVL**" y " **TAVLCalendario** ":

// PARTE PRIVADA de TNodoAVL

```
// El elemento (etiqueta) del nodo
TCalendario item ;
// Subárbol izquierdo y derecho
TAVLCalendario iz, de ;
// Factor de equilibrio
int fe ;
```

// FORMA CANÓNICA de TNodoAVL

```
//Constructor por defecto
TNodoAVL ();
//Constructor copia
TNodoAVL (TNodoAVL &);
//Destructor
~TNodoAVL ();
// Sobrecarga del operador asignación
TNodoAVL & operator=( TNodoAVL &);
```

***// PARTE PRIVADA de* TAVLCalendario**

// Puntero al nodo raíz

```
TNodoAVL *raiz;
Para reducir el coste del cálculo de los recorridos en el árbol AVL, hace falta añadir los
siguientes métodos a la PARTE
PRIVADA de la clase ( ver "Aclaraciones", detrás de la lista de métodos requeridos ) :
// Devuelve el recorrido en inorden
void InordenAux(TVectorCalendario &, int);
// Devuelve el recorrido en preorden
void PreordenAux(TVectorCalendario &, int);
// Devuelve el recorrido en postorden
void PostordenAux(TVectorCalendario &, int);
```

***// FORMA CANÓNICA de* TAVLCalendario**

// Constructor por defecto

```
TAVLCalendario ();
// Constructor de copia
TAVLCalendario (TAVLCalendario &);
// Destructor
~ TAVLCalendario ();
// Sobrecarga del operador asignación
TAVLCalendario & operator=( TAVLCalendario &);
```

***// MÉTODOS de* TAVLCalendario**

```
// Sobrecarga del operador igualdad
bool operator==( TAVLCalendario &);
// Sobrecarga del operador desigualdad
bool operator!=( TAVLCalendario &);
// Devuelve TRUE si el árbol está vacío, FALSE en caso contrario
bool EsVacio();
// Inserta el elemento en el árbol
bool Insertar(TCalendario &);
// Devuelve true si el elemento está en el árbol, false en caso contrario
bool Buscar(TCalendario &);
// Devuelve la altura del árbol (la altura de un árbol vacío es 0)
int Altura();
// Devuelve el número de nodos del árbol (un árbol vacío posee 0 nodos)
int Nodos();
// Devuelve el número de nodos hoja en el árbol (la raíz puede ser nodo hoja)
int NodosHoja();
// Devuelve el recorrido en inorden sobre un vector
TVectorCalendario Inorden();
// Devuelve el recorrido en preorden sobre un vector
TVectorCalendario Preorden();
// Devuelve el recorrido en postorden sobre un vector
TVectorCalendario Postorden();
// Borra un TCalendario del árbol AVL
bool Borrar(TCalendario &);
// Devuelve el elemento TCalendario raíz del árbol AVL
TCalendario Raiz();

// Sobrecarga del operador salida
friend ostream & operator<<(ostream &, TAVLCalendario &);
```

Aclaraciones :

• La forma de emplear los métodos auxiliares de ordenación de la PARTE PRIVADA es (por ejemplo, para el caso del recorrido en inorden):

// Devuelve el recorrido en inorden

```
TVectorCalendario TAVLCalendario::Inorden()
{
int posición = 1;
TVectorCalendario recorrido(Nodos());
InordenAux(recorrido, posicion);
return recorrido;
}
```

De este modo, se reduce el coste de crear múltiples objetos de tipo TVectorCalendario, ya que sólo se emplea uno durante todo el cálculo del recorrido.

• El criterio de ordenación para TCalendario sigue siendo el que se pide en el **Cuadernillo1.**

• Para simplificar los algoritmos, el árbol NO puede contener elementos repetidos.

• El constructor de copia tiene que realizar una copia exacta.

• El destructor tiene que liberar toda la memoria que ocupe el árbol.

• Si se asigna un árbol a un árbol no vacío, se destruye el árbol inicial. La asignación tiene que realizar una copia exacta.

• En el **operator==** , dos árboles son iguales si poseen los mismos elementos independientemente de la estructura interna del árbol.

• **Insertar(TCalendario &)**, devuelve TRUE si el elemento se puede insertar y FALSE en caso contrario (por ejemplo, porque ya existe en el árbol).

• Si el árbol está vacío, los recorridos tienen que devolver un vector vacío.

• Se permite amistad entre las clases **TAVLCalendario** y **TNodoAVL**.

• **Borrar(TCalendario &)** devuelve TRUE si el elemento se puede borrar y FALSE en caso contrario (por ejemplo, porque no existe en el árbol). El criterio de borrado es sustituir por el mayor de la izquierda.

• El operador SALIDA muestra el RECORRIDO POR INORDEN del AVL, con el formato pedido en el Cuadernillo 1 para la clase **TVectorCalendario**.

Parte 3 **: clase base : "THASHCalendario"**

Qué se pide :

En esta parte se pide implementar una clase que representa una **tabla de dispersión abierta (tabla HASH)** de objetos de tipo TCalendario ("**THASHCalendario"**).

Para ello, la clase a desarrollar emplea las clases **TCalendario** y **TListaCalendario** desarrolladas en el **Cuadernillo 1**. Internamente, la tabla HASH contendrá el tamaño de la tabla y un vector de listas de **TCalendario** (vector de **TListaCalendario**), representado mediante un puntero a **TListaCalendario**, para el cual se reservará la correspondiente memoria en el momento de construir la tabla (***Figura 1***).

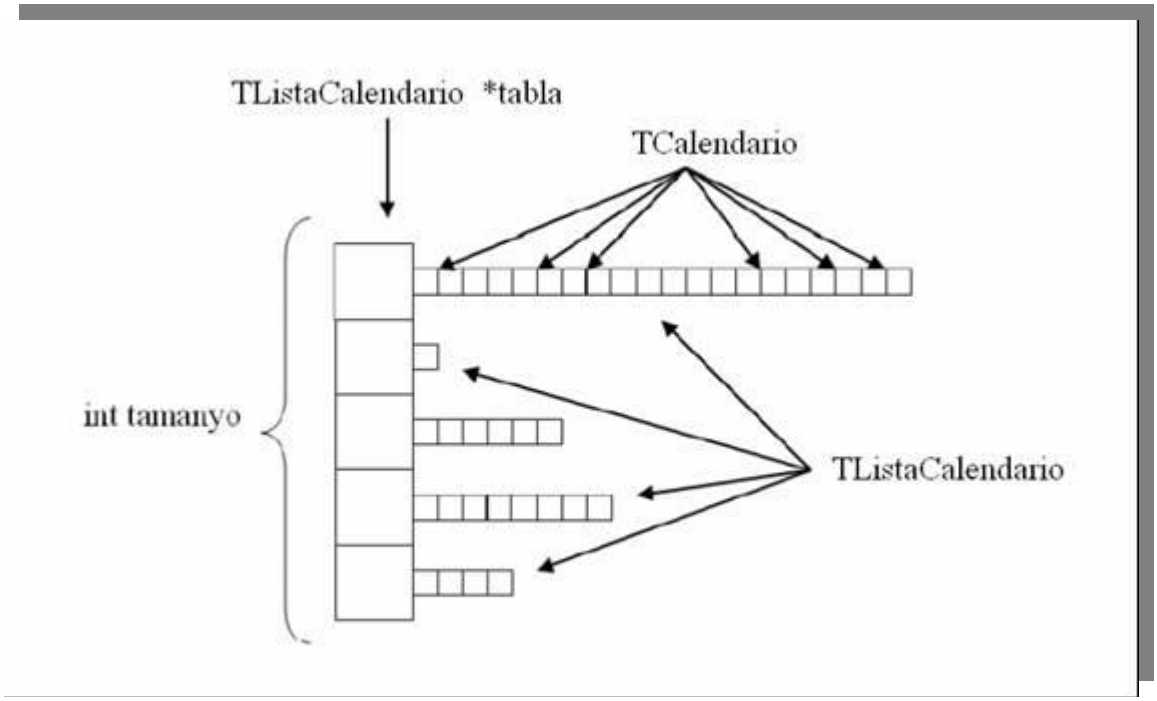


Figura 1*: Representación de la tabla de dispersión abierta*

Como **función de dispersión** se tiene que emplear **H(x) = x Mod N,** donde:

• **x** es el número resultante de unir todos los dígitos de la fecha del TCalendario en cuestión . Ejemplo : de la fecha **28/02/2006** resultaría **28022006.**
• **N** es el tamaño de la tabla de dispersión.
• **Mod** es una operación que devuelve el resto de la división entera.

Cuándo se exige:

Al alumno se le podrá exigir tenerlo terminado y funcionando a partir del día : **lunes 23/05/2011**

Prototipo de la Clase:

```
// PARTE PRIVADA de THASHCalendario

// tamaño de la tabla de listas ( TListaCalendario )
int tamanyo;
// Almacena las listas ( TListaCalendario ) que contiene la tabla
TListaCalendario *tabla;


// FORMA CANÓNICA de THASHCalendario

// Constructor por defecto
THASHCalendario ();
// Constructor indicando el tamaño de la tabla
THASHCalendario (int);
// Constructor de copia
THASHCalendario (THASHCalendario &);
// Destructor
~ THASHCalendario ();
// Sobrecarga del operador asignación
THASHCalendario & operator=(THASHCalendario &);


// MÉTODOS de THASHCalendario

// Sobrecarga del operador igualdad
bool operator==(THASHCalendario &);
// Devuelve true si la tabla está vacía, false en caso contrario
bool EsVacia();
// Inserta el elemento en la tabla
bool Insertar(TCalendario &);
// Busca y borra el elemento de la tabla
bool Borrar(TCalendario &);
// Devuelve TRUE si el elemento está en la tabla, FALSE en caso contrario
bool Buscar(TCalendario &);
// Devuelve el tamaño de la tabla
int Tamanyo();
// Devuelve el número de elementos en la tabla
int NElementos();
// Devuelve todos los elementos de la tabla en una lista ordenada
TListaCalendario Lista();
// Búsqueda en un HASH a través de una lista auxiliar
int*  BuscarLista( TListaCalendario &);


// Sobrecarga del operador salida
friend ostream & operator<<(ostream &, THASHCalendario &);
```

Aclaraciones :

• El criterio de ordenación para TCalendario sigue siendo el que se pide en el **Cuadernillo1.**

• La tabla NO puede contener elementos repetidos.

• El constructor por defecto crea una tabla de tamaño 0 (puntero interno a NULL, no se reserva memoria).

• En el constructor a partir de un tamaño, si el tamaño es menor que 0, se creará una tabla de tamaño 0 (como el constructor por defecto).

• El constructor de copia tiene que realizar una copia exacta.

• El destructor tiene que liberar toda la memoria que ocupe la tabla.

• Si se asigna una tabla a una tabla no vacía, se destruye la tabla inicial. La asignación tiene que realizar una copia exacta.

• En el **operator==**, dos tablas son iguales si poseen el mismo tamaño y los mismos elementos.

• El método **EsVacia()** de la tabla HASH devuelve TRUE cuando la tabla está vacía, es decir, sin ningún elemento insertado.

(Por ejemplo, si invoco al constructor con tamaño 10, realmente está vacía porque aún no he insertado ningún elemento; tras insertar el primer elemento, **EsVacia()** devolvería FALSE; si después lo borro, **EsVacia()** volvería a devolver TRUE.)

• En **Insertar(TCalendario &)**, el nuevo elemento se inserta en la posición de la tabla de listas, indicada por la función de dispersión. Devuelve TRUE si el elemento se puede insertar y FALSE en caso contrario (por ejemplo, porque ya existe en la tabla). Si el tamaño de la tabla es 0, NO se puede insertar ningún elemento (se devuelve FALSE).

• En **Borrar(TCalendario &)**, devuelve TRUE si el elemento se puede borrar y FALSE en caso contrario (por ejemplo, porque no existe en la tabla).

• El **operador salida** muestra el contenido de la tabla desde la primera posición hasta la última. Todo el contenido de la tabla se muestra entre símbolos "#". Entre el primer "#" y el primer elemento y entre el último elemento y el último "#" NO tienen que aparecer espacios en blanco. Para cada posición de la tabla, primero se mostrará la posición entre paréntesis (empezando desde 0) y a continuación, separado por un espacio en blanco, la lista **TListaCalendario** (tal y como se definió en el **Cuadernillo 1**) contenida en dicha posición de la tabla . Cada posición se tiene que separar de la siguiente por un espacio en blanco (a continuación de la última posición no se tiene que mostrar nada). NO se tiene que generar un salto de línea al final. Si el tamaño de la tabla es 0, se tiene que mostrar la cadena "##".

Ejemplo : (*no fijarse en el HASH que generarían esas fechas, sólo en el formato de salida*)

```
#(0) <06/06/2006 "mens1-lista1" 10/12/2007 "mens2-lista1"> (1) <06/06/2007 "mens1-lista2">#
```

• En **BuscarLista(TListaCalendario &) :**

Devuelve un vector de enteros del mismo tamaño que la lista pasada como parámetro. Cada posición del vector almacenará un número entero que se corresponde con el lugar que ocupa cada elemento de la lista en la tabla hash que invoca a la función. Los elementos de la tabla HASH se empiezan a numerar desde 1. Si algún elemento de la lista pasada como parámetro no existe en la tabla, se pondrá un 0 en su posición correspondiente del vector. La lista pasada como parámetro se recorre de izquierda a derecha.

Ejemplo (*el ejemplo está hecho con números naturales por simplificación):*

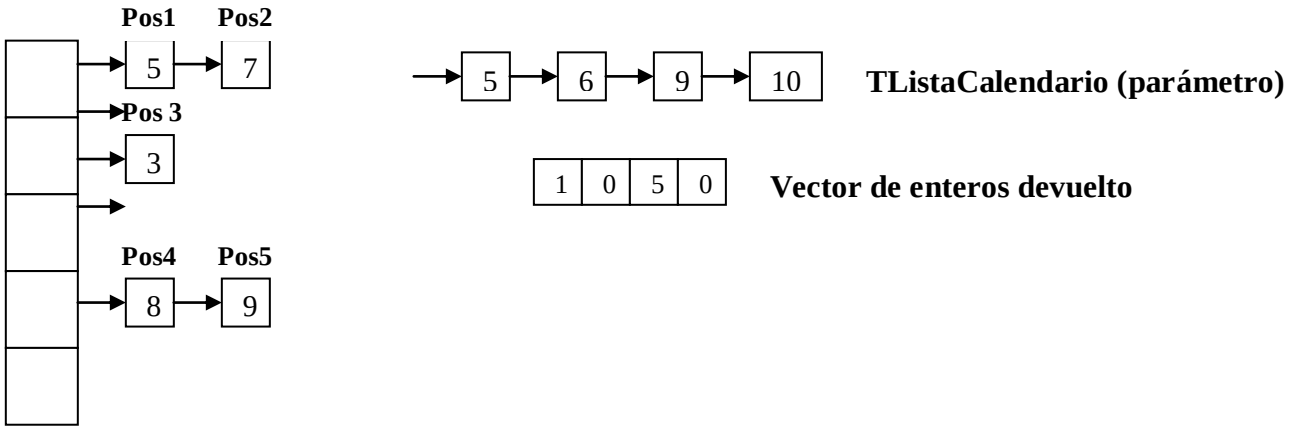


Tabla HASH

ANEXO 2. Condiciones de ENTREGA.

2.1. Dónde, cómo, cuándo.

La entrega de la práctica se realizará :

- Servidor : en el SERVIDOR DE PRÁCTICAS, cuya URL es : http://pracdlsi.dlsi.ua.es/

- Fecha: las fechas de entrega y Examen Práctico se publicarán en Campus Virtual oportunamente.

- A título INDIVIDUAL (aunque el alumno la haya realizado en pareja), y por tanto requerirá del alumno que conozca su USUARIO y CONTRASEÑA en el Servidor de Prácticas.

2.2. Ficheros a entregar y comprobaciones.

La práctica debe ir organizada en 3 subdirectorios:

DIRECTORIO 'include' : contiene los ficheros (nombres en MINÚSCULAS) :
- **"tcalendario.h"**
- **"tvectorcalendario.h"**
- **"tlistacalendario.h" (**incluye las clases **: TListaCalendario , TNodoCalendario y TListaPos)**
- **"tabbcalendario.h"**
- **"tavlcalendario.h" (**incluye las clases**: TAVLCalendario y TNodoAVL)**
- **"thashcalendario.h"**

DIRECTORIO 'lib' : contiene los ficheros (NO deben entregarse los ficheros objeto "**.o"**) **:**
- **"tcalendario.cpp"**
- **"tvectorcalendario.cpp"**
- **"tlistacalendario.cpp" (**incluye las clases**: TListaCalendario , TNodoCalendario y TListaPos)**
- **"tabbcalendario.cpp"**
- **"tavlcalendario.cpp" (**incluye las clases**: TAVLCalendario y TNodoAVL)**
- **"thashcalendario.cpp"**

DIRECTORIO 'src' : contiene los ficheros :
- **"tad.cpp" (**fichero aportado por el alumo para comprobación de tipos de datos. No se tiene en cuenta para la corrección**)**

Además, en el directorio raíz , deberá aparecer el fichero **"nombres.txt"** : fichero de texto con los datos de los autores.

El formato de este fichero es:
1_DNI: *DNI1*
1_NOMBRE: *APELLIDO1.1 APELLIDO1.2, NOMBRE1*
2_DNI: *DNI2*
2_NOMBRE: *APELLIDO2.1 APELLIDO2.2, NOMBRE2*

2.3. Entrega en formato DOXYGEN.

Comentarios para DOXYGEN

Para una correcta entrega en **formato Doxygen**, es necesario seguir las pautas que se indican el fichero "**Tutorial_DOXYGEN.pdf**", que se proporcionará como material de Campus Virtual.
Hay que incluir en los ficheros fuente todos los comentarios necesarios en **formato Doxygen**. Estos comentarios deben estar en forma corta y detallada, y deben definirse para :
- **Ficheros** : debe incluir **nombre** y **dni** de los autores
- **Clases** : escribir propósito de la clase: aprox. 3 líneas
- **Métodos (públicos o privados)** : 1 línea para funciones triviales. 2 líneas para operaciones específicas más complicadas (para éstas , explicar también : parámetros de entrada, parámetros de salida y funciones dependientes)

Generar documentación DOXYGEN

Para usar correctamente la **herramienta Doxygen**, es necesario colocar en el directorio raíz de la estructura de ficheros a entregar , el fichero de configuración **Doxygen** llamado "**DOXYFILE".**
Además, debe usarse un **MAKEFILE** adecuado para trabajar con **Doxygen** , de forma que cada vez que se ejecuta MAKE, se puede generar la documentación **Doxygen.**
Tanto el "**DOXYFILE"** como el **MAKEFILE** adecuado para **Doxygen** y para todos los TADs pedidos en la práctica, serán oportunamente publicados como materiales de Campus Virtual.
Cada vez que se ejecuta el **MAKEFILE** , **Doxygen** generará automáticamente un subdirectorio , **'doc'** , que contendrá la documentación de la práctica (extraída de los comentarios de los fuentes) , en formato HTML. Para ello, se ejecuta :

```
"make doc"
```

2.4 Entrega final

Sólo se deben entregar los ficheros detallados anteriormente (ninguno más).

Cuando llegue el momento de la entrega, toda la estructura de directorios ya explicada (¡ATENCIÓN! excepto el **MAKEFILE** , el **DOXYFILE** y la **carpeta 'doc'**, que hay que retirar antes), debe estar comprimida en un fichero llamado "**NNNNNNNN.tgz**" (sustituyendo **NNNNNNNN** por el **propio DNI**) , de forma que éste NO supere los 300 K .

Ejemplo : **`tar –czvf 12345678.tgz *`**

2.5. Otros avisos referentes a la entrega .

No se devuelven las prácticas entregadas. Cada alumno es responsable de conservar sus prácticas.

La detección de prácticas similares (“copiados”) supone el automático suspenso de TODOS los autores de las prácticas similares. Cada alumno es responsable de proteger sus prácticas. Se recuerda que a los alumnos con prácticas copiadas no se les guardará ninguna nota (ni teoría ni prácticas), para convocatorias posteriores.

Las prácticas no se pueden modificar una vez corregidas Y evaluadas (no hay revisión del código). Por lo tanto, es esencial ajustarse a las condiciones de entrega establecidas en este enunciado. En especial, llevar cuidado con los nombres de los ficheros.

ANEXO 3. Condiciones de corrección.

ANTES de la evaluación:

La práctica se programará en el Sistema Operativo Linux, y en el lenguaje C++. Deberá compilar con la versión instalada en los laboratorios de la Escuela Politécnica Superior.

A lo largo del período cuatrimestral, el profesor irá realizando una REVISIÓN MANUAL , PERSONALIZADA e INDIVIDUAL del código de cada alumno. El resultado de esta revisión (como ya se expresa en las NORMAS DE LA ASIGNATURA), puede contabilizar HASTA 0'5 puntos (sobre 10) de la nota final de prácticas de PED; esto se evaluará en función de 2 factores :

a) Nivel de cumplimentación de los objetivos propuestos en fechas (ver apartado "**Cuándo se exige**" decada clase propuesta)
b) Pruebas de calidad del código.

La evaluación:

La práctica se corregirá casi en su totalidad de un modo automático, por lo que los nombres de las clases, métodos, ficheros a entregar, ejecutables y formatos de salida descritos en el enunciado de la práctica SE HAN DE RESPETAR EN SU TOTALIDAD.

A la hora de la corrección del Examen de Prácticas (y por tanto, de la práctica del Cuadernillo) , se evaluará :
- El correcto funcionamiento de los TADs propuestos en en Cuadernillo
- El correcto funcionamiento de el/los nuevo/s método/s propuestos para programar durante el tiempo del Examen.

Uno de los objetivos de la práctica es que el alumno sea capaz de comprender un conjunto de instrucciones y sea capaz de llevarlas a cabo. Por tanto, es esencial ajustarse completamente a las especificaciones de la práctica.

Cuando se corrige la práctica , el corrector automático proporcionará ficheros de corrección llamados "**tad.cpp**". Este fichero utilizará la sintaxis definida para cada clase y los nombres de los ficheros asignados a cada una de ellas: únicamente contendrá una serie de instrucciones **#include** con los nombres de los ficheros "**.h**".

ANEXO 4. Utilidades

Almacenamiento de todos los ficheros en un único fichero

Usar el comando **tar** y **mcopy** para almacenar todos los ficheros en un único fichero y copiarlo en un disco:

- $ **tar** cvzf practica.tgz *
- $ **mcopy** practica.tgz a:/

Para recuperarlo del disco en la siguiente sesión:

- $ **mcopy** a:/practica.tgz
- $ **tar** xvzf practica.tgz

Cuando se copien ficheros binarios **(.tgz, .gif, .jpg**, etc.) no se debe emplear el parámetro -t en **mcopy**, ya que sirve para convertir ficheros de texto de Linux a DOS y viceversa.

Utilización del depurador gdb

El propósito de un depurador como **gdb** es permitir que el programador pueda "ver"qué está ocurriendo dentro de un programa mientras se está ejecutando.

Los comandos básicos de gdb son:

1. **r (run)**: inicia la ejecución de un programa. Permite pasarle parámetros al programa. Ejemplo: r fichero.txt.
2. **l (list)**: lista el contenido del fichero con los números de línea.
3. **b (breakpoint)**: fija un punto de parada. Ejemplo: b 10 (breakpoint en la línea 10), b main (breakpoint en la función main).
4. **c (continue)**: continúa la ejecución de un programa.
5. **n (next)**: ejecuta la siguiente orden; si es una función la salta (no muestra las líneas de la función) y continúa con la siguiente orden.
6. **s (step)**: ejecuta la siguiente orden; si es una función entra en ella y la podemos ejecutar línea a línea.
7. **p (print)**: muestra el contenido de una variable. Ejemplo: p auxiliar, p this (muestra la dirección del objeto), p *this (muestra el objeto completo).
8. **h (help)**: ayuda.

Utilización de la herramienta VALGRIND

(Se proporcionará como Material complementario en Campus Virtual, un tutorial acerca de las características de la herramienta VALGRIND, que básicamente se usará para depuración de errores de código) .