Programación y Estructuras de Datos

CUADERNILLO 1

(Curso 2014-2015)

Parte 1: **clase base: "TCalendario"**

Qué se pide:

Se pide construir una clase que representa una **fecha** con un **mensaje**.
La fecha viene dada en tres enteros: **dia**, **mes** y **anyo**; y el mensaje como una vector dinámico de caracteres.

Prototipo de la Clase:

PARTE PRIVADA

```
int dia, mes, anyo;
char* mensaje;
```

FORMA CANÓNICA

```
//Constructor por defecto: inicializa dia, mes y anyo a 1 de 1 del 1900 y mensaje a NULL
TCalendario ()
//Constructor que inicializa calendario con parámetros pasados
TCalendario(int dia, int mes, int anyo, char * mens);
//Constructor copia
TCalendario (TCalendario &);
//Destructor
~TCalendario();
// Sobrecarga del operador asignación
TCalendario & operator=(TCalendario &);
```

MÉTODOS

```
// Sobrecarga del operador SUMA de fecha + un número de dias;
TCalendario operator+(int);
// Sobrecarga del operador RESTA de fecha - un número de dias;
TCalendario operator-(int);
// Modifica la fecha incrementandola en un dia (con postincremento);
TCalendario operator++(int);
// Modifica la fecha incrementandola en un dia (con preincremento);
TCalendario & operator++();
// Modifica la fecha decrementándola en un dia (con postdecremento);
TCalendario operator--(int);
// Modifica la fecha decrementándola en un día (con predecremento);
TCalendario & operator--();
// Modifica la fecha
bool ModFecha (int, int, int);
// Modifica el mensaje
bool ModMensaje(char *);
// Sobrecarga del operador igualdad;
bool operator ==(TCalendario &);
// Sobrecarga del operador desigualdad;
bool operator !=(TCalendario &);
// Sobrecarga del operador >; (ver ACLARACIÓN sobre ORDENACIÓN)
bool operator>(TCalendario &);
// Sobrecarga del operador <; (ver ACLARACIÓN sobre ORDENACIÓN)
bool operator<(TCalendario &);
//TCalendario vacío
bool EsVacio();
// Devuelve el día del calendario;
int Dia();
// Devuelve el mes del calendario;
int Mes();
// Devuelve el año del calendario;
int Anyo();
// Devuelve el mensaje del calendario;
char *Mensaje();
```

FUNCIONES AMIGAS

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

Aclaraciones :

- El **Constructor por defecto** inicializa a esta fecha: **1 del 1 de 1900 y mensaje a NULL.**

- El método **EsVacio()** devuelve TRUE si el calendario tiene fecha 1 del 1 del 1900 y el mensaje es NULL (tanto si se acaba de crear como si ha recuperado este estado).

- En los **operadores : operator+ , operator- , &operator++, operator++, &operador--, operador-- :**

  El chequeo de los días, meses y años se realiza para ver si la fecha del **TCalendario** tratado por el método es CORRECTA, tanto si dicho **TCalendario** es una copia temporal como si actúa como "**this**" . Esto incluye:
  - número correcto de días y meses
  - control de años bisiestos (es decir: los años que sean divisibles por 4 serán bisiestos; aunque no serán bisiestos si son divisibles entre 100 (como los años 1700, 1800, 1900 y 2100) a no ser que sean divisibles por 400 (como los años 1600, 2000 ó 2400) .

  La fecha 1/1/1900 es la primera fecha correcta. Fechas anteriores no son correctas.
  Así, la fecha tratada, una vez modificada (incrementada o decrementada) por el método, ha de ser siempre CORRECTA de acuerdo al calendario estándar.

- En el **constructor "TCalendario(int dia, int mes, int anyo, char * mens)" :**

  El chequeo de los días, meses y años se realiza para ver si la fecha es CORRECTA. Esto incluye:
  - número correcto de días y meses
  - control de años bisiestos

  Para los parámetros de entrada, La fecha 1/1/1900 es la primera fecha correcta. Fechas anteriores no son correctas.
  Este constructor únicamente inicializa la fecha a los parámetros pasados. Si la fecha pasada es INCORRECTA se inicializa a 1/1/1900 y mensaje NULL.

- En el **Destructor :**

  Dejará la fecha **1 del 1 de 1900 y mensaje a NULL**.

- En el **operador "TCalendario operator+(int)" :**

  Si el número de días es POSITIVO, se opera. En caso contrario, NO se hace nada.
  NO modifican el objeto sobre el que se aplica. Observa que se devuelve por VALOR.

  Ejemplo **:**
  ```
  c = a + 40;  /* "a" queda como estaba. */
  ```

- En los **operadores "TCalendario operator-(int)" , "TCalendario operator--(int)" , "TCalendario &operator--()" :**

  Si al aplicar la RESTA o DECREMENTO, el **TCalendario** resultante quedase con una fecha INCORRECTA (o sea, ANTERIOR a 1/1/1900), entonces el objeto TCalendario resultante se inicializa a **1/1/1900** y **mensaje NULL**.

  Ejemplo 1 (RESTA) :
  ```
  TCalendario f1, f2 ;   /*se inicializan ambos con 1/1/1900 */
  f2 = f1 -  10 ;        /* el operator- dejaría a 'f2' con  una fecha
  previa a 1/1/1900 */
  ```
  Ejemplo 2 (PREDECREMENTO) :
  ```
  TCalendario f1, f2 ; /*se inicializan ambos con 1/1/1900 */
  f2 = --f1;     /* el operator-- dejaría a 'f2' con una fecha
  previa a 1/1/1900 */
  ```

- En el método **"bool ModFecha (int, int, int)" :**

  El chequeo de los días, meses y años se realiza para ver si la fecha es CORRECTA:
  - Número de días correcto ; número de mes correcto
  - Controlar el tema de los años bisiestos

  Para los parámetros de entrada, la fecha 1/1/1900 es la primera fecha correcta. Fechas anteriores no son correctas.

  Así pues, la fecha pasada debe ser correcta. Si es así, el método devuelve TRUE y cambia la fecha del objeto **TCalendario**. Si NO es así, el objeto **TCalendario** se queda como estaba antes de invocar a este método, y el método devuelve FALSE.

- En el método **" bool ModMensaje(char *)" :**

  La cadena entrante se copiará al mensaje del **TCalendario.**
  En caso de que la cadena entrante contenga NULL, la cadena "mensaje" del **TCalendario** también quedará a NULL.

- En el método **" Mensaje()" :**

  En caso de que la cadena "mensaje" contenga NULL, se devolverá NULL; debería devolver siempre el puntero al mensaje, independientemente de lo que valga.

- En el **operador "=="**, dos calendarios son iguales si poseen la misma fecha y el mismo mensaje. (Tener el "mismo mensaje" implica que el contenido del mensaje es exactamente el mismo). Lo equivalente se aplica al **operador "!=" .**
  (Tener en cuenta lo siguiente: una variable del tipo cadena que sea vacía (o sea, ""), NO es igual que una variable del tipo cadena que sea igual a NULL).

- Para el "**operator>**" y "**operator<**" , definimos el siguiente método de ordenación.

  Un TCalendario "T1" es MAYOR que TCalendario "T2" si ...
  - **(Criterio 1) :** la fecha de T1 es posterior a la de T2 (independientemente del mensaje) .
  - **(Criterio 2) :** la fecha de T1 es igual a la de T2 , y el mensaje de T1 es mayor (en comparación de cadenas) , al mensaje de T2 .
  - (Si la fecha de T1 es igual a la de T2 y el mensaje de T1 es igual al de T2, entonces tanto "**operator>**" como "**operator<**" devuelven FALSE) .
  - Cadena NULL es MENOR que cadena "", y ésta es MENOR que un espacio (" "), y MENOR que cualquier otra cadena con contenido.
  - (OJO!! la función **strcmp(NULL,<cad>)** y **strcmp(<cad>,NULL)** provocarán ERROR ; se recomienda prever esto ejecutando "if (**<cad>** = = NULL) …" , antes de pasar la cadena **<cad>** como argumento de **strcmp().**

- En el **operador "EsVacio()" :**

  Se considera un **TCalendario** vacío aquél que tiene el valor directamente generado por el **Constructor por defecto de TCalendario**, es decir: a **fecha 1 del 1 de 1900 y mensaje a NULL.**

- El **operador salida " operator <<"** , muestra por pantalla el contenido del calendario. Muestra el día, mes y año según el **formato DD/MM/AAAA** , y a continuación, tras un espacio en blanco el mensaje entre comillas . NO se tiene que generar un salto de línea al final. En caso de día y/o mes < 10, se muestran según el formato "0N"

  En caso de que la cadena "mensaje" contenga NULL , el operador salida la mostrará como "" , es decir, abrir y cerrar comillas sin espacio intermedio.

  Ejemplos de salida:

```
10/12/2007 "ESTE ES UN MENSAJE"
06/06/2006 ""
```

Parte 2: **VECTOR de TCalendario “TVectorCalendario”**

Qué se pide :

Se pide construir una clase que representa un **VECTOR de objetos de tipo "TCalendario"**.

Prototipo de la Clase:

PARTE PRIVADA

```
TCalendario *c;
int tamano;
TCalendario error;
```

FORMA CANÓNICA

```
// Constructor por defecto
TVectorCalendario();
// Constructor a partir de un tamaño
TVectorCalendario(int);
// Constructor de copia
TVectorCalendario(TVectorCalendario &);
// Destructor
~TVectorCalendario();
// Sobrecarga del operador asignación
TVectorCalendario & operator=(TVectorCalendario &);
```

MÉTODOS

```
// Sobrecarga del operador igualdad
bool operator==(TVectorCalendario &);
// Sobrecarga del operador desigualdad
bool operator!=(TVectorCalendario &);
// Sobrecarga del operador corchete (parte IZQUIERDA)
TCalendario & operator[](int);
// Sobrecarga del operador corchete (parte DERECHA)
TCalendario operator[](int) const;
// Tamaño del vector (posiciones TOTALES)
int Tamano();
// Cantidad de posiciones OCUPADAS (no vacías) en el vector
int Ocupadas();
// Devuelve true si existe el calendario en el vector
bool ExisteCal(TCalendario &);
// Mostrar por pantalla mensajes de TCalendario en el vector, de fecha IGUAL O POSTERIOR a la pasada
void MostrarMensajes(int,int,int);
// REDIMENSIONAR el  vectorde TCalendario
bool  Redimensionar(int);
```

FUNCIONES AMIGAS

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

Aclaraciones :

- El vector NO tiene por qué estar ordenado.
- El vector puede contener elementos repetidos. Incluso de los considerados "vacíos", es decir, con el valor directamente generado por el **constructor por defecto de TCalendario.**
- El **Constructor por defecto** crea un vector de dimensión 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 un vector de dimensión 0 (como el constructor por defecto).
- El **Destructor** tiene que liberar toda la memoria que ocupe el vector.
- Si se asigna un vector a un vector no vacío, se destruye el vector inicial. Puede ocurrir que se modifique el tamaño del vector al asignar un vector más pequeño o más grande.
- En el operador igualdad "**operator==**", dos vectores son iguales si poseen la misma dimensión y los mismos elementos TCalendario en las mismas posiciones.
- En la sobrecarga del corchete, **"operator[]"**, las posiciones van desde 1 (no desde 0) hasta el tamaño del vector. Si se accede a una posición que no existe, se tiene que devolver un **TCalendario** vacío (el mismo que crea el **constructor por defecto** de **TCalendario**).

  A la hora de INSERTAR elementos en el vector de **TCalendario** , se permite repetir elementos.

  Si se accede a una posición inexistente se debe devolver un **TCalendario** VACÍO. Se permite declarar en la parte PRIVATE un elemento de clase de nombre "**error**" con valor **TCalendario** vacío, para devolverlo por referencia.

- En "**MostrarMensajes (int,int,int)",** se muestra por pantalla cada uno de los elementos **TCalendario** de fecha IGUAL O POSTERIOR a la fecha dada en parámetros, recorriendo el vector desde la posición **1** hasta tamaño **N.**

  El formato de salida será: la salida propia del elemento **TCalendario**, y cada **TCalendario** se separa del siguiente con una coma y espacio (", ") y toda la salida encerrada entre corchetes.

  Si se pasa una fecha inválida como parámetro, no se muestra ningún **TCalendario**, así que la salida es "**[]**"

  Ejemplo (teniendo como parámetro de entrada la fecha 01/01/2006):

  ```
  [06/06/2006 "ESTE ES OTRO MENSAJE", 10/12/2007 "ESTE ES UN MENSAJE"]
  ```

- En el método **"Ocupadas()" :**

  Se consideran "posiciones vacías" del vector, aquellas que contengan un **TCalendario** de los que devuelven TRUE al aplicarle el método **EsVacio()** de **TCalendario** (fecha **1 del 1 de 1900 y mensaje a NULL).**

- En el método **"bool Redimensionar(int)" :**

  A esta función se le pasa un entero y hay que redimensionar el vector al tamaño del entero.

  La salida será TRUE cuando se haya producido realmente un redimensionamiento, y FALSE cuando el vector permanezca con la dimensión antigua.

  Los valores del entero que se pasa por parámetro pueden ajustarse a estos casos:
  - Si el entero es menor o igual que 0 , el método devolverá FALSE, sin hacer nada más.
  - Si el entero es de igual tamaño que el actual del vector, el método devolverá FALSE, sin hacer nada más.
  - Si el entero es mayor que 0 y mayor que el tamaño actual del vector, hay que copiar los componentes del vector en el vector nuevo, que pasará a tener el tamaño que indica el entero. Las nuevas posiciones serán vacías, es decir, objetos **TCalendario** inicializados a 1/1/1900 y mensaje NULL
  - Si el entero es mayor que 0 y menor que el tamaño actual del vector, se deben eliminar los **TCalendario** que sobren por la derecha, dejando el nuevo tamaño igual al valor del entero.

- El **operador salida " operator <<"** muestra el contenido del vector desde la primera hasta la última posición en una sola línea. Para cada elemento, primero se mostrará la POSICIÓN del elemento entre paréntesis y a continuación, separado por un espacio en blanco, el elemento **TCalendario** en sí. Cada elemento **TCalendario** se tiene que separar del siguiente por una coma y espacio (", "). Toda la salida deber estar encerrada **entre corchetes "[]"**.
  NO se tiene que generar un salto de línea al final. Si la dimensión del vector es 0, se muestra la cadena **"[]".**

  Ejemplo:

  ```
  [(1) 06/06/2006 "ESTE ES OTRO MENSAJE", (2) 10/12/2007 "ESTE
  ES UN MENSAJE"]
  ```

Parte 3: **LISTA de TCalendario "TListaCalendario"**

Qué se pide:

Se pide construir una clase que representa una **LISTA de objetos de tipo "TCalendario"**.
Organización de la lista: será ENLAZADA SIMPLE y ORDENADA, manteniendo el criterio de ordenación de **TCalendario** ya explicado.

Para representar cada NODO de la lista, se tiene previamente que definir la **clase TNodoCalendario**, con su FORMA CANÓNICA (constructor, constructor de copia, destructor y sobrecarga del operador asignación) como mínimo.

La lista será accesible por una POSICIÓN. Por tanto, también se pide construir una clase que representa una **POSICIÓN en LISTA de objetos de tipo "TCalendario"**.

Prototipo de la Clase "TNodoCalendario":

PARTE PRIVADA

```
TCalendario c; // Uso de LAYERING sobre la clase

TNodoCalendario *siguiente;
```

FORMA CANÓNICA

```
// Constructor por defecto
TNodoCalendario ();

// Constructor de copia
TNodoCalendario (TNodoCalendario &);

// Destructor
~ TNodoCalendario ();

// Sobrecarga del operador asignación
TNodoCalendario & operator=(TNodoCalendario &);
```

Prototipo de la Clase "TListaCalendario":

PARTE PRIVADA

```
// Primer item de la lista
TNodoCalendario *primero;
```

FORMA CANÓNICA

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

MÉTODOS

```
// Sobrecarga del operador igualdad
bool operator==(TListaCalendario &)
//Sobrecarga del operador suma
TListaCalendario operator+ (TListaCalendario &);
//Sobrecarga del operador resta
TListaCalendario operator- (TListaCalendario &);
// Inserta el elemento en la posición que le corresponda dentro de la lista
Bool Insertar(TCalendario &);
// Busca y borra el elemento
bool Borrar(TCalendario &);
// Borra el elemento que ocupa la posición indicada
bool Borrar (TListaPos &);
//Borra todos los Calendarios con fecha ANTERIOR a la pasada.
bool Borrar(int,int,int);
// Devuelve true si la lista está vacía, false en caso contrario
bool EsVacia();
// Obtiene el elemento que ocupa la posición indicada
TCalendario Obtener(TListaPos &)
// Devuelve true si el Calendario está en la lista.
bool Buscar(TCalendario &);
// Devuelve la longitud de la lista
int Longitud();
// Devuelve la primera posición en la lista
TListaPos Primera();
// Devuelve la última posición en la lista
TListaPos Ultima();
// Suma de dos sublistas en una nueva lista
TListaCalendario SumarSubl (int I_L1, int F_L1, TListaCalendario & L2, int I_L2, int
F_L2);
// Extraer un rango de nodos de la lista
TListaCalendario ExtraerRango (int n1, int n2)
```

FUNCIONES AMIGAS

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

Prototipo de la Clase “TListaPos”:

PARTE PRIVADA

```
// Para implementar la POSICIÓN a NODO de la LISTA de TCalendario
TNodoCalendario *pos;
```

FORMA CANÓNICA

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

MÉTODOS

```
// Sobrecarga del operador igualdad
bool operator==(TListaPos &);
// Sobrecarga del operador desigualdad
bool operator!=(TListaPos &);
// Devuelve la posición siguiente
TListaPos Siguiente();
// Posición vacía
bool EsVacia();
```

Aclaraciones de la Clase “TListaCalendario”:

- La lista NO puede contener elementos repetidos.

- La lista estará ordenada en ORDEN ASCENDENTE manteniendo el criterio de ordenación de **TCalendario** explicado en la PARTE 1.

  RECORDAR : un TCalendario "T1" es MAYOR que TCalendario "T2" si ...
  - **(Criterio 1) :** la fecha de T1 es posterior a la de T2 (independientemente del mensaje) .
  - **(Criterio 2) :** la fecha de T1 es igual a la de T2 , y el mensaje de T1 es mayor (en comparación de cadenas) , al mensaje de T2 .
  - (Si la fecha de T1 es igual a la de T2 y el mensaje de T1 es igual al de T2, entonces tanto "**operator>**" como "**operator<**" devuelven FALSE) .
  - Cadena NULL es MENOR que cadena “”, y ésta es MENOR que un espacio (“ ”), y MENOR que cualquier otra cadena con contenido.
  - *(OJO!! la función **strcmp(NULL,<cad>)** y **strcmp(<cad>,NULL)** provocarán ERROR ; se recomienda prever esto ejecutando “if (**<cad>** = = NULL) …” , antes de pasar la cadena **<cad>** como argumento de **strcmp().***

- Se permite AMISTAD entre las clases **TListaCalendario, TNodoCalendario y TListaPos.**

- El **Constructor por defecto** crea una lista vacía.

- El **Constructor de copia** tiene que realizar una copia exacta.

- El **Destructor** tiene que liberar toda la memoria que ocupe la lista.

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

- En el **operador igualdad "operator=="**, dos listas son iguales si poseen los mismos elementos en el mismo orden.

- El **operador suma "operator+"**une los elementos de dos listas en una nueva lista ordenada.

- El **operador resta "operator-"**devuelve una lista nueva que contiene los elementos de la primera lista (operando de la izquierda) que NO existen en la segunda lista (operando de la derecha).

- "**Insertar**", inserta el calendario en la lista de forma que se mantenga el orden de la misma. Se devuelve TRUE si el elemento se ha podido insertar (no existe previamente en la lista).

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

- "**Borrar(TListaPos &)**", devuelve TRUE si el elemento se puede borrar y FALSE en caso contrario (por ejemplo, porque la posición no es válida). No es necesario comprobar que **TListaPos** apunte a un nodo de la lista.

- "**Borrar(int,int,int)**", devuelve TRUE si hay elementos que se pueden borrar y FALSE en caso contrario (por ejemplo, porque la lista NO tenga calendarios anteriores a la fecha pasada como parámetro).

- **TListaCalendario "SumarSubl (int I_L1, int F_L1, TListaCalendario & L2, int I_L2, int F_L2)" :**

  Los elementos de la primera sublista son los contenidos entre las posiciones **I_L1** y **F_L1** (ambas posiciones incluidas) de la lista que invoca a la función y los elementos de la segunda sublista son los contenidos entre las posiciones **I_L2** y **F_L2** (ambas posiciones incluidas) de la lista L2 pasada como parámetro.

  El valor devuelto será una nueva lista que contendrá la “suma” de las dos sublistas obtenidas (“suma” entendida como el conjunto ordenado de los elementos de ambas).

  Cosas a tener en cuenta :
  - Se comienza a numerar las posiciones de la lista a partir de 1.
  - Si **F_L1** o **F_L2** sobrepasan la longitud de las listas “por exceso”: se seleccionan sólo los elementos contenidos entre **I_L1** y longitud de sublista 1 o los elementos contenidos entre **I_L2** y longitud de sublista 2.
  - Si **I_L1** o **I_L2** son menor o igual a 0: se seleccionan sólo los elementos contenidos entre la posición 1 de sublista 1 y **F_L1** (o los elementos contenidos entre la posición 1 de sublista 2 y **F_L2**).
  - Si **I_L1 > F_L1 ó I_L2 > F_L2**: estos rangos son ilógicos, pues no engloban elemento alguno: por ello, las sublistas que generarían se consideran vacías.

- En la lista de resultado, se ha de mantener las 2 propiedades: es ORDENADA y NO CONTIENE REPETIDOS.

La lista sobre la que trabaja el método **SumarSubl** no se verá modificada tras la operación.

Ejemplo: (el ejemplo está hecho con letras por simplificación) :

Supongamos que L1 y L2 contienen los siguientes elementos:
L1=<a,c,d,e,f,m>
L2=<b,c,d,g,h,i>

En el programa principal pondríamos, por ejemplo:
```
L3=L1.SumarSubl(2,4,L2,0,7);
```

Las sublistas obtenidas serían las siguientes:
sublista_L1=<c,d,e> → Los elementos contenidos entre las posiciones 2 a 4 de L1
sublista_L2=< b,c,d,g,h,i > → Los elementos contenidos entre las posiciones 1 a 6 de L2 (ya que los límites "reales" son 1 y 6, no 0 y 7).

El resultado devuelto por la función sería el siguiente:
L3=<b,c,d,e,g,h,i> → sublista_L1 + sublista_L2

Ejemplo 2: (el ejemplo está hecho con letras por simplificación) :

Supongamos que L1 y L2 contienen los siguientes elementos:
L1=<a,c,d,e,f,m>
L2=<b,c,d,g,h,i>

En el programa principal pondríamos, por ejemplo :
```
L3=L1.SumarSubl(4,2,L2,0,7);
```

Las sublistas obtenidas serían las siguientes:
sublista_L1=<> → No hay elementos contenidos entre las posiciones 4 a 2 de L1
sublista_L2=<b,c,d,g,h,i> → Los elementos contenidos entre las posiciones 1 a 6 de L2 (ya que los límites "reales" son 1 y 6, no 0 y 7).

El resultado devuelto por la función sería el siguiente:
L3=<b,c,d,g,h,i> → sublista_L1 + sublista_L2

- El **operador TListaCalendario ExtraerRango (int n1, int n2) :**

Devuelve una lista con los elementos **TCalendario** comprendidos entre las posiciones **n1 y n2** (ambas incluidas) de la lista que invoca a la función. Los nodos comprendidos entre **n1 y n2** (ambos incluidos) deben borrarse de la lista que invoca a la función.
Cosas a tener en cuenta:

- Se comienza a numerar las posiciones de la lista a partir de 1.
- Si **n2** sobrepasa la longitud de la lista invocante "por exceso": se seleccionan sólo los elementos contenidos entre **n1** y la longitud de la lista.
- Si **n1** es menor o igual a 0: se seleccionan sólo los elementos contenidos entre la posición 1 de la lista y **n2** .
- Si **n1 = n2**: devolverá una lista con 1 sólo elemento, extrayéndolo de la lista llamante.
- Si **n1 > n2**: devolverá una lista VACÍA sin modificar a la llamante, pues los límites no engloban elemento alguno.

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

Listas iniciales:
L1=<1 4 6 7 8>
L2=<>

Llamada a la función:

```
L2=L1.ExtraerRango(2,4)
```

Listas finales después de ejecutar la función:

```
L1=<1 8>
L2=<4 6 7>
```

Otra llamada a la función :

```
L2=L1.ExtraerRango(4,2)
```

Listas finales después de ejecutar la función:

```
L1=<1 4 6 7 8>
L2=<>
```

Otra llamada a la función :

```
L2=L1.ExtraerRango(0,18)
```

Listas finales después de ejecutar la función:

```
L1=<>
L2=<1 4 6 7 8>
```

- El **operador salida " operator <<"** muestra el contenido de la lista desde la cabeza hasta el final de la lista.

  Todo el contenido se muestra **entre los caracteres "<" y ">"**. Entre **"<"** y el primer elemento y el último elemento y **">"** NO tienen que aparecer espacios en blanco. Cada elemento se tiene que separar del siguiente por un espacio en blanco (a continuación del último elemento NO se tiene que generar un espacio en blanco). NO se tiene que generar un salto de línea al final. Si la lista está vacía, se tiene que mostrar la cadena "<>".

  Ejemplo:

  ```
  <06/06/2006 "ESTE ES OTRO MENSAJE" 10/12/2007 "ESTE ES UN MENSAJE">
  ```

Aclaraciones de la Clase "TListaPos":

- Evidentemente, una posición puede dejar de ser válida en cualquier momento (por ejemplo, la lista a la que apunta la posición puede variar o incluso ser destruida).
- En cualquier caso, NO es necesario comprobar que un **TListaPos** apunta realmente a un nodo de la lista (no se planteará el caso en los TAD de prueba).

- Sí que hay que comprobar (en concreto, para las operaciones de inserción, borrado y Obtener, propias de **TListaCalendario**), que el objeto **TListaPos** no es vacío.

- En **"Siguiente()"**, si la posición actual es la última de la lista, se tiene que devolver una posición vacía.

- En " **EsVacia()**" : devuelve TRUE si el puntero interno ("pos") es NULL . En caso contrario devuelve FALSE.

- En el **operador igualdad**, dos posiciones son iguales si apuntan a la misma posición de la lista.

ANEXO 1. Notas de aplicación general sobre el contenido del Cuadernillo.

Cualquier modificación o comentario del enunciado se publicará oportunamente en el Campus Virtual.

En su momento, se publicarán como materiales del Campus Virtual los distintos FICHEROS de MAIN (**tad.cpp**) que servirán al alumno para realizar unas pruebas básicas con los TAD propuestos.
No obstante, se recomienda al alumno que aporte sus propios ficheros **tad.cpp** y realice sus propias pruebas con ellos.
(El alumno tiene que crearse su propio conjunto de ficheros de prueba para verificar de forma exhaustiva el correcto funcionamiento de la práctica. Los ficheros que se publicarán en el CV son sólo una muestra y en ningún caso contemplan todos los posibles casos que se deben verificar)

Todas las operaciones especificadas en el Cuadernillo son obligatorias.

Si una clase hace uso de otra clase, en el código nunca se debe incluir el fichero **.cpp**, sólo el **.h**.

El paso de parámetros como constantes o por referencia se puede cambiar dependiendo de la representación de cada tipo y de los algoritmos desarrollados. Del mismo modo, el alumno debe decidir si usa el modificador CONST , o no.

En la parte PUBLIC no debe aparecer ninguna operación que haga referencia a la representación del tipo, sólo se pueden añadir operaciones de enriquecimiento de la clase.

En la parte PRIVATE de las clases se pueden añadir todos los atributos y métodos que sean necesarios para la implementación de los tipos.

Tratamiento de **excepciones**: todos los métodos darán un mensaje de error (en **cerr**) cuando el alumno determine que se produzcan **excepciones**; para ello, se pueden añadir en la parte privada de la clase aquellas operaciones y variables auxiliares que se necesiten para controlar las excepciones.

Se considera **excepción** aquello que no permite la normal ejecución de un programa (por ejemplo, problemas al reservar memoria, problemas al abrir un fichero, etc.). **NO** se considera **excepción** aquellos errores de tipo **lógico** debidos a las especificidades de cada clase.

De cualquier modo, todos los métodos deben devolver siempre una variable del tipo que se espera.
Los mensajes de error se mostrarán siempre por la salida de error estándar (cerr). El formato será:

ERROR: mensaje_de_error *(al final un salto de línea).*

ANEXO 2. Condiciones de ENTREGA.

2.1. Dónde, cómo, cuándo, valor.

La entrega de la práctica se realizará:

- Servidor: en el SERVIDOR DE PRÁCTICAS, cuya URL es : http://pracdlsi.dlsi.ua.es/
- Fecha: se habilitará la posibilidad de entrega en el Servidor de prácticas entre estas fechas:
  - **Lunes, 16/03/2015.**
  - **Viernes, 20/03/2015 (23:59)**
- A título INDIVIDUAL ; por tanto requerirá del alumno que conozca su USUARIO y CONTRASEÑA en el Servidor de Prácticas.
- Se podrá realizar cuantas entregas quiera el alumno: solo se corregirá la última práctica entregada. Tras cada entrega, el servidor enviará al alumno un INFORME DE COMPILACIÓN, para que el alumno compruebe que lo que ha entregado cumple las especificaciones pedidas y que se ha podido generar el ejecutable correctamente. Este informe también se podrá consultar desde la página web de entrega de prácticas del DLSI (http://pracdlsi.dlsi.ua.es e introducir el nombre de usuario y password).
- Valor :
  - **El Cuadernillo 1 vale un 7% de la nota final de PED.**
  - El Cuadernillo 2 vale un 7% de la nota final de PED
  - El Cuadernillo 3 vale un 6% de la nota final de PED.
  - El EXAMEN PRÁCTICO vale un 30% de la nota final de PED.

2.2. Ficheros a entregar y comprobaciones.

La práctica debe ir organizada en 3 subdirectorios:

DIRECTORIO 'include' : contiene los ficheros (en MINÚSCULAS) :
- **"tcalendario.h"**
- **"tvectorcalendario.h"**
- **"tlistacalendario.h" (**incluye**: TListaCalendario , TNodoCalendario y TListaPos)**

DIRECTORIO 'lib' : contiene los ficheros (NO deben entregarse los ficheros objeto "**.o**") **:**

- **"tcalendario.cpp"**
- **"tvectorcalendario.cpp"**
- **"tlistacalendario.cpp" (**incluye**: TListaCalendario , TNodoCalendario y TListaPos)**

DIRECTORIO 'src' : contiene los ficheros :

- **"tad.cpp" (**fichero aportado por el alumno 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*

2.3 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**), debe estar comprimida en un fichero de forma que éste NO supere los 300 K .

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

2.4. 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.

NOTA IMPORTANTE: 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 y el formato especificado para la salida.

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.

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 para el 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**". **No se permitirá ninguna modificación de código de la práctica una vez haya sido entregada y corregida.**

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 aconseja el uso de la herramienta VALGRIND para comprobar el manejo correcto de la memoria dinámica. Se puede encontrar una descripción más detallada de dicha herramienta en el libro "**C++ paso a paso**" de Sergio Luján.

Modo de uso:

valgrind - -tool=memcheck - -leak-check=full nombre_del_ejecutable