PRÁCTICA 4ª: Ampliación de la clase implementada en la práctica 3 (**en sus dos versiones**).

OBJETIVOS: Manejo de los conceptos de constructor y destructor. Parámetros por defecto. Constructor copia. Sobrecarga de funciones.

TEMPORIZACIÓN:
Publicación del enunciado: Semana del 7 de octubre.
Entrega: Semana del 22 de octubre junto con las prácticas 3 y 5.
Límite de entrega (con penalización): Semana del 29 de octubre.

BIBLIOGRAFÍA
Programación orientada a objetos con C++
Autor: Fco. Javier Ceballos
Editorial: RA-MA.

Se añadirán a la clase `CHora` implementada en la práctica 3ª los siguientes elementos:

- Un constructor que permita iniciar un objeto con ninguno, todos o algunos de los datos (horas, minutos, segundos y formato: AM, PM o 24 HORAS). Esto implica que se han de definir sus parámetros con valores por omisión: 0, 0, 0 y "24 HORAS".

```
CHora(int hh = 0, int mm = 0, int ss = 0, char *ff = "24 HORAS");
```

- Un constructor copia (no utilizar el operador = para la implementación de este constructor).

- Un destructor que libere la memoria asignada dinámicamente al objeto.

- La sobrecarga del operador de asignación.

En la definición de ambos constructores se utilizará una lista de iniciadores; en esta lista, se iniciará cada atributo con el valor correspondiente, excepto `m_pszFormato` que se iniciará con el valor 0 (`NULL`) ya que está pendiente de una asignación de memoria mediante `AsignarFormato`.

El operador de asignación no permitirá hacer una copia de un objeto sobre sí mismo y ajustará el tamaño de la cadena `m_pszFormato` destino al tamaño del origen (reutilice, siempre que sea posible, el código ya escrito).

El formato debe almacenarse en todos los casos en mayúsculas.

Hacer que el método `EsHoraCorrecta` de `CHora` sea público.

Hacer que los constructores, el destructor y el operador de asignación visualicen un mensaje indicando que han sido invocados.

El programa principal se escribirá en un fichero `práctica4.cpp` y mostrará el siguiente menú desde su función `main`:

1. Crear un objeto local. Defina un objeto de ámbito local en los bloques donde sea necesario. *(Esta opción mostrará el siguiente submenú)*
    1. Con una hora predeterminada.
    2. Introduciendo la hora.
    3. Introduciendo la hora y los minutos.
    4. Introduciendo la hora, los minutos y los segundos.
    5. Introduciendo la hora, los minutos, los segundos y el formato.
    6. Volver al menú principal.
2. Crear un objeto dinámicamente. Para ello, definir un puntero a nivel de **main** iniciado a 0, para después crear el objeto invocando a **new** donde sea necesario. *El objeto será destruido en la opción 5*.
3. Constructor copia.
   (Creará un objeto de ámbito local a partir del objeto dinámico del punto 2)
4. Operador de asignación.
   (Copiará en un objeto de ámbito local el dinámico del punto 2)
5. Terminar *(liberará los objetos dinámicos)*

Cada opción invocará a la función externa *VisualizarHora* de la práctica 3.

En las prácticas 2 y 3 se utilizaron los ficheros `utils.h` y `utils.cpp` que incluían las funciones `LeerInt`, `LeerFloat`, `LeerCadena`, `CrearMenu`, `ConverMayus`, etc. Utilizando el concepto de función sobrecargada, escriba las sobrecargas de `bool LeerDato(`*`tipo`*`& dato)` que sustituyan a las funciones `Leer...` anteriores. Por ejemplo:

`LeerInt()` será sustituida por `bool LeerDato(int& dato)`
`LeerFloat()` será sustituida por `bool LeerDato(float& dato)`
`LeerDouble()` será sustituida por `bool LeerDato(double& dato)`
`LeerCadena()` será sustituida por `bool LeerDato(string& dato)` y por
`bool LeerDato(char *dato)`
...

Estas funciones devolverán `true` cuando devuelvan un valor validado y `false` cuando el usuario pulse `Ctrl+z` (`EOF`). Puede utilizar `cin.exceptions` pero no plantillas (`template`). Para más detalles, eche una ojeada al apartado "Ejercicios resueltos" del capítulo "Excepciones" en la bibliografía especificada.

Todas estas funciones, junto con las funciones `CrearMenu`, `ConverMayus` y cualquier otra que se estime necesaria, se implementarán como métodos **static** de una clase `CUtils` perteneciente al espacio de nombres `utils`. Por ejemplo:

```
// Archivo utils.h
...

namespace utils
{
```

```
  class CUtils
  {
    public:
      ...
      static bool leerDato(int& dato);
      ...
  };
};


// Archivo utils.cpp
...
bool CUtils::leerDato(int& dato)
{
  // Habilitar excepciones
  cin.exceptions(ios::failbit | ios::badbit);

  try
  {
    cin >> dato;
    // Eliminar caracteres sobrantes. Por ejemplo '\n'.
    cin.ignore(numeric_limits<int>::max(), '\n');
    // Deshabilitar excepciones
    cin.exceptions(ios::goodbit);

  ...
```

Todos los métodos/funciones que reciban punteros como parámetros deberán contemplar el caso de recibir el valor `NULL`.

En la versión de la práctica que utiliza el tipo `string` en lugar de `char *`, razone qué métodos de la clase `CHora` pueden eliminarse por resultar innecesarios. En esta versión utilizará el método `bool LeerDato(string& dato)` para leer un `string` y `string& ConverMayus(string& str)` para convertir un `string` a mayúsculas.

En la definición de ambos constructores se utilizará una lista de iniciadores para iniciar todos sus atributos con los valores correspondientes.