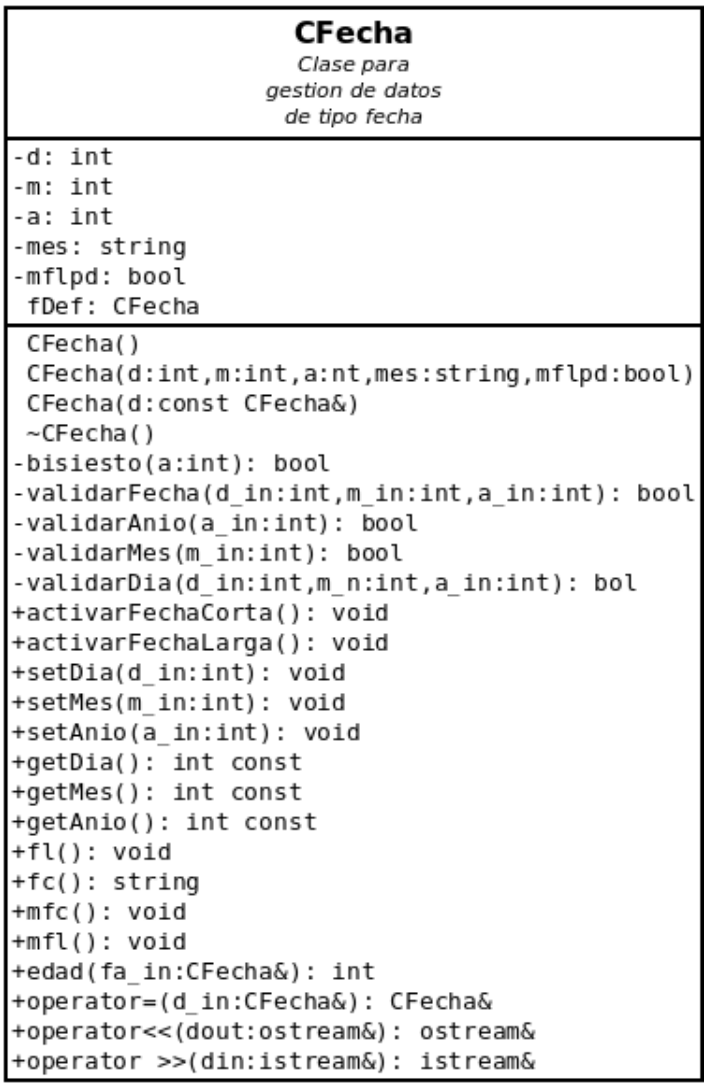
Resumen

Esta clase se diseñó y construyó para servir como medio de aprendizaje. Muchas de las técnicas uitilizadas son básicas pero son las primeras que se aprenden. Un ejemplo de ellos es la utilización del if-else en lugar del operador ?: que es más compacto.

Diseño



Utilización

Para poder utilizar la clase `CFecha`, debe incluir el archvo de cabecera `CFecha.h`. Hay varias formas en que puede hacerse esto, la misma se explica en la sección de instalación.

Vamos a crear una aplicación de prueba de la clase al que llamaremos `testClaseCFecha.cpp` y lo guardaremos en el mismo directorio CFECHA

```
// testClaseCFecha.cpp
#include <iostream>
#include "CFecha.h"

using namespace std;
```

```
int main(){
   CFecha miCumple; // miCumple se inicializa con los valores definido en fDef
   miCumple=(24,08,1968); // Asignamos una fecha a miCumple
   ...
}
```

Con la instrucción `CFecha miCumple` se define una variable de tipo CFecha, una instancia de esa clase. Se reserva memoria para alojar un objeto de esta clase y se inicializa con los valores por defecto (Definida en la variable estática `fDef`). `miCumple` es un objeto de la clase CFecha y posee toda la funcionalidad que le hayamos podido proporcionar a dicha clase.

Si deseamos asignar una fecha diferente podemos utilizar cualquiera de las formas siguientes: `miCumple=(24,08,1968)` o bien `miCumple=CFecha(24,08,1968)`, en este último caso se crea una nueva variable de tipo `CFecha` y se asignan cada atributo uno a uno utilizando el operador de asignación (=) sobrecargado. Una vez copiada, la variable recién creada se destruye.

Es importante definir una fecha por defecto en las aplicaciones para, en caso de inicializarse un objeto de la clase con una fecha inválida, pueda esta incializarse con los valores por defecto. Esta variable de tipo fecha y perteneciente a la misma clase es estática y salta la comprobación por lo que se debe tener sumo cuidado al inicializarlas. El siguiente código muestra cómo hacerlo.

```
// pCFecha.cpp
# include "CFecha.h" // Depende como lo hayas instalado
...
CFecha CFecha::fDef(1,1,1974);
...
int main() {
   ...
}
```

Es posible utilizar la fecha del sistema para asignar de forma automática la fecha actual a una variable de tipo CFecha. Utilizando en código siguiente podemos obtener el efecto buscado.

```
\\ pCFecha.cpp
...
#include <ctime>
...
int main() {
   ...
   struct tm *fh;
   time_t segundos;
   time(&segundos);
   fh=localtime(&segundos);
   int dh=(int)fh->tm_mday
   int mh=(int)fh->tm_mon +1; // tm en time.h
   int ah=(int)fh->tm_year+1900;

   CFecha hoy(dh,mh,ah);
   ...
}
```

La aplicación cuyo código archivo fuente es `pCFecha.cpp` posee ahora una variable global llamada `hoy` igual a la fecha del sistema y puede ser muy útil para calcular la edad de una persona a partir de la fecha de nacimiento.

Métodos de actualización

La clase posee los siguientes métodos (funciones) que permiten actualizar los datos de un objeto de la clase en forma particular. Los prototipos de estas funciones son:

```
bool setDia(int dia_in),
bool setMes(int mes_in) y
bool setAnio(int a_in)
```

Una Introducción al diseño de CFecha

La clase utiliza tres variables de tipo `int`, en su sección privada, para representar día, mes y año. Además utiliza un objeto de tipo `string` para la representación del mes en forma de cadena y una variable de tipo *bool* para activar y desativar las formas en que la fecha es mostrada por medio de un objeto del flujo de salida estandar ostream.

Po último utiliza una variable de tipo CFecha estática para utilizarla como fecha por defecto. Al ser estática se debe de tener cuidado de no introducir una fecha incorrecta ya que no se efectúa la comprobación del mismo. En la siguiente sección se muestra una parte del archivo de cabecera *CFecha.h*

```
/* ********** CFecha.h *************** */
/* ************************************
PROGRAMACION ORIENTADA A OBJETOS CON C++
prof. Armando B. VERA
************************************ */
#ifndef _CFECHA_H_
#define _CFECHA_H_

#include <iostream>
#include <ostream>
#include <istream>
#include <string>
#include <ctime>
#include <limits>

class CFecha{
   private:
      int d;
      int m;
      int a;
      std::string mes;
      bool mflpd; // true: muestra fecha larga por defecto

      bool bisiesto(int a_in);
      bool validaFecha(int d_in, int m_in, int a_in);
      bool validaAnio(int a_in);
      bool validaMes(int m_in);
      bool validaDia(int d_in, int m_in, int a_in);

      static CFecha fDef; // Fecha por defecto
   ...
}
```

Además de los cinco atributos descritos anteriormente observamos declarado cuatro funciones de tipo *bool* cuya funcón es la de validar o comprobar si las fechas son correctas antes de ser asignados a una variable (objeto) del tipo `CFecha`, queda excluida de esta comprobación la variable estática `fDef`. Las definiciones de estas funciones figuran en el archvio *CFecha.cpp* que se estudiará más adelante.

Interfaz

Se utiliza un constructor con valores por defecto. Como es lógico, esta debe estar en su sección pública. Iremos mostrando fragmentos del archvo de cabecera *CFecha.h* conforme vamos explicando.

```
class CFecha{
...
public:
CFecha(int d_in=0, int m_in=0, int a_in=0, std::string mes="", bool mflpd=false);
static void set_fDef(int d_in, int m_in, int a_in, std::string mes, bool mflpd);
CFecha(const CFecha& f);
~CFecha();
...
}
```

En la primera línea de su parte pública figura el constructor. Se utilica un constructor con valores por defecto que, como puede notarse, no son fechas válidas. Esto se hace de esta forma para asignarse la fecha por defecto en caso de que el usuario pretenda introducir una fecha inválida o no introduzca ninguna fecha.

El constructor nos permite introducir fechas como `CFecha miCumple(24,08,68)` o bien, con la instrucción `CFecha miCumple=(24,08,68)`.

Tambieén podemos definir variables (objetos) de tipo fecha así `CFecha miFecha`. En este caso, se asignan los valores por defecto, pero como nos son fechas válidas, no pasa la comprobación y se asigna la fecha por defecto, es decir la fecha definida en `fDef`

el atributo de tipo *bool* `mflpd` (Mostrar fecha larga por defecto) se utiliza para determinar cómo se mostrará la fecha cuando se utilice el operador "<<" de inserción en el flujo de salida ostream. Si este atributo es *true* se mostrará "25 de Febrero de 2015" cuando se utilice el `cout`, y si es falso, se mostrará "25 de Feb de 2015".

Los métodos de actualización deben pasar por un filtro para determinar si son valores válidos. Si no los son, se debe proporcionar algún mecanismo que permita cambiar los valores incorrectos o bien considerar la posibilidad de no efectuar la actualización, pero eviando un mensaje informando de tal situación.

El método *bool setDia(int d_in)*

La definición de este método es el siguiente:

```
...

bool CFecha::setDia(int d_in){
   if(validaDia(d_in, m, a) && validaMes(m) && validaAnio(a)){
      d=d_in;
      return true;
   } else {
     throw errorFecha(); // Tratamiento de error
     return false;
   }
}
...
```

Este método es invocado desde un objeto del tipo `CFecha` con valores ya asignados a sus parámetros, por lo tanto debe comprobar si el día que se pasa por parámetro puede ser asignado a ese objeto fecha. Supongamos que la fecha es "25 de febrero de 2015" y queremos cambiar a la fecha "29 de febrero de 2015". En este caso, nuestra clase debe impedir tal asignación ya que el día 29 para el mes de febrero de ese año no existe. La opción más sensata es no hacer el cambio y enviar o mostrar un aviso sobre la causa de la no asignación. La función (o el método si si lo prefiere) *bool validaDia(int d_in)* producirá un valor falso haciendo que el if no se cumpla. Es importante hacer notar que *bool validaDia(intd_in)* llama a la función *bool bisiesto(int a_in)*

utilización

Vamos a suponer la situación planteada en lo párrafos anteriores. Definimos un objeto fecha al que llamaremos **diaD**, es decir, `CFecha diaD=CFecha(25,02,2015)`. Queremos que **diaD** en lugar de 25 sea 29. Llamamos al método para hacer dicho cambio

```
diaD.setDia(29);
```

Tras haber sido llamado el método de actualización, nuestra fecha seguirá con la fecha antigua. El encargdo de darle el tratamiento adecuado es `throw errorFecha()`. Aquí hay que hacer notar que es fundamental que el diseño de la clase debe contemplar todas esas posibilidades y darle la posiblidad de un tratamiento adecuado.

Tratamiento del error en la clase

Cuando se tiene la responsabilidad de diseñar una clase, si la idea es hacerlo robusta, funcional, simple y usable hay que verse como un usuario.