Práctica 1: Implementación de clases básicas en C++

Programación Avanzada y Estructuras de Datos, Grado en Ing. I.A., Universidad de Alicante
Curso 2024–2025

Fecha última modificación: 30/09/2024

1 Introducción

Este enunciado describe la primera práctica de laboratorio de Programación Avanzada y Estructuras de Datos. El objetivo general es implementar una serie de clases en C++ que nos permita trabajar con un calendario de eventos. En dicho calendario, podremos guardar, para cada fecha, un mensaje con la descripción del evento. Podremos guardar, consultar y borrar mensajes. Evidentemente, el número de eventos que guardemos puede ser ilimitado.

En esta primera práctica, únicamente será necesario implementar la clase `Fecha`. El resto de clases se implementará más adelante. ~~Dividiremos la implementación en 3 clases: `Fecha`, `Evento` y `Calendario`. Un calendario está compuesto (es decir, se trata de una relación de **composición**) de una serie de Eventos. Cada evento simplemente consta de una fecha y un mensaje. En esta primera práctica, sólo implementaremos la clase `Fecha`.~~

2 Clase `Fecha`

La clase `Fecha` codifica una fecha (día, mes y año) con la que se podrá operar de diferentes maneras. La clase deberá tener los métodos que se muestran a continuación en su parte pública, mientras que los atributos y métodos en la parte privada deben ser diseñados por el alumnado.

```
//Constructor por defecto: inicializa la fecha a 1/1/1900
Fecha();
//Constructor sobrecargado: inicializa la fecha según los parámetros
Fecha(int dia,int mes,int anyo);
//Constructor de copia
Fecha(const Fecha &);
//Destructor: pone la fecha a 1/1/1900
~Fecha();
//Operador de asignación
Fecha& operator=(const Fecha &);
//Operador de comparación
bool operator==(const Fecha &) const;
```

```
//Operador de comparación
bool operator!=(const Fecha &) const;
//Devuelve el día
int getDia() const;
//Devuelve el mes
int getMes() const;
//Devuelve el año
int getAnyo() const;
//Modifica el día: devuelve false si la fecha resultante es incorrecta
bool setDia(int);
//Modifica el mes: devuelve false si la fecha resultante es incorrecta
bool setMes(int);
//Modifica el anyo: devuelve false si la fecha resultante es incorrecta
bool setAnyo(int);
//Incrementa la fecha en el número de días pasado como parámetro.
//Si el parámetro es negativo, la decrementa
bool incrementaDias(int );
//Incrementa la fecha en el número de meses pasado como parámetro.
//Si el parámetro es negativo, la decrementa
bool incrementaMeses(int );
//Incrementa la fecha en el número de años pasado como parámetro.
//Si el parámetro es negativo, la decrementa
bool incrementaAnyos(int );
//Devuelve una representación como cadena de la fecha
string aCadena(bool larga, bool conDia) const;
```

El constructor por defecto debe inicializar la fecha a 1/1/1900. Bajo ningún concepto ser permitirán fechas anteriores a esa. Tampoco se permitirán, evidentemente, fechas inexistentes, es decir, con un valor de meses que no esté entre 1 y 12, y con un valor de días que no corresponda al rango de días permitido para el mes y año correspondiente (teniendo en cuenta años bisiestos).

Así, si en el constructor sobrecargado se introduce una combinación de día, mes y año incorrecta, la fecha deberá inicializarse a 1/1/1900. En los métodos que modifican la fecha, deberá devolverse `false` si, como consecuencia de la modificación, se obtendría una fecha incorrecta, y no realizar ninguna modificación en ese caso. Si la modificación da lugar a una fecha correcta, deberá devolverse `true`. En el método `incrementaDias`, ha de tenerse en cuenta que un incremento positivo nunca dará lugar a una fecha incorrecta. Si un incremento en días sobrepasa el final del mes, ha de incrementarse el mes (y el año, si es necesario). Por ejemplo, el resultado de incrementar 30 días la fecha 15/2/2024 es 16/3/2024 (2024 es bisiesto).

En los métodos `incrementaMeses` e `incrementaAnyos` un incremento positivo sí que podría dar lugar a una fecha incorrecta. Por ejemplo, si incrementamos en un mes la fecha 30/1/2024, o si incrementamos en un año la fecha 29/2/2024.

Por último, en el método `aCadena`, la representación como cadena debe contener el día de la semana si el segundo parámetro tiene el valor `true`. El primer parámetro determina si el formato es largo (`true`) o corto (`false`). El formato largo para la fecha 16/3/2024 sería `sábado 16 de marzo de 2024`. En caso de desactivar el día, el método debería devolver simplemente `16 de marzo de 2024`. El formato corto sería con día de la semana para misma fecha sería `sábado 16/3/2024` y el formato corto sin día de la semana, simplemente `16/3/2024`.

Recuerda que todos los meses del año tienen 31 días excepto febrero (28 o 29 días, dependiendo de si el año es bisiesto) y abril, junio, septiembre y noviembre (30 días). Los años bisiestos son aquellos que son divisibles entre 4. Además, si son divisibles entre 100, han de serlo también entre 400. Por ejemplo, 2024 es bisiesto porque es divisible entre 4 y no entre 100. 1900 no es bisiesto porque es divisible entre 4, pero también lo es entre 100 y no entre 400. 2000 sí es bisiesto porque es divisible entre 4, 100 y 400.

Respecto al cálculo del día de la semana, ten en cuenta no será necesario realizar el cálculo para ningún día anterior al 1/1/1900. Dicho día fue lunes. Si asumimos que el 0 representa el lunes, el 1 el martes, y así sucesivamente hasta el 6, que es el domingo, es fácil calcular el día de la semana para cualquier fecha. Sólo tienes que calcular el número de días que han transcurrido desde el 1/1/1900 y calcular su resto con el número 7. Por ejemplo, entre el 1/1/1900 y el 11/9/2024 han transcurrido 45544 días. $45544\%7 = 2$, por tanto, es miércoles.

3 Clase `Evento`

Esta clase se implementará más adelante. No es necesario que esté implementada para el examen del próximo día 16/10/2024

La clase `Evento` simplemente debe almacenar una fecha (instancia de la clase `Fecha`) y una descripción del evento a realizar en dicha fecha, codificada como un objeto de tipo `string`. La clase deberá tener los métodos que se muestran a continuación en su parte pública, mientras que los atributos y métodos en la parte privada deben ser diseñados por el alumnado.

```
//Constructor por defecto: inicializa la fecha a 1/1/1900 ...
//y la descripción a "sin descripción"
Evento();
//Constructor sobrecargado: inicializa la fecha según los parámetros
Evento(const Fecha&, string)
//Constructor de copia
Evento(const Evento&);
//Operador de asignación
Evento& operator=(const Evento &);
//Destructor: pone la fecha a 1/1/1900 y la descripción a "sin descripción"
~Evento();
//Devuelve (una copia de) la fecha
Fecha getFecha() const;
//Devuelve (una copia de) la descripción
```

```
string getDescripcion() const;
//Modifica la fecha
void setFecha(const Fecha& );
//Modifica la descripción
bool setDescripcion(string);
```

Ha de tenerse en cuenta que un evento nunca puede tener una descripción vacía (cadena de longitud 0). Si en algún momento se intenta asignar una cadena de longitud cero (en el método `setDescripcion` o en el constructor sobrecargado), esta cadena debe sustituirse por `sin descripción`.

Como se puede ver, los métodos `getFecha` y `getDescripcion` devuelven esos datos por valor. Esto quiere decir que el objeto devuelto es una copia del original y, si se modifica, esas modificaciones no se reflejarán en el objeto de tipo `Evento`.

4 Clase `Calendario`

Esta clase se implementará más adelante. No es necesario que esté implementada para el examen del próximo día 16/10/2024

Esta es la clase principal, que representa todos los eventos del calendario. Un objeto de tipo `Calendario` es una **composición** de objetos de tipo `Evento`, por lo que los eventos asociados al calendario no podrán existir al margen de éste. La clase `Calendario` deberá tener los métodos que se muestran a continuación en su parte pública:

```
//Constructor por defecto: calendario sin ningún evento
Calendario();
//Constructor de copia
Calendario(const Calendario&);
//Operador de asignación
Calendario& operator=(const Calendario &);
//Destructor
~Calendario();
//Añade un evento al calendario. Si ya existía un evento en esa fecha,
//devuelve false y no hace nada. En caso contrario, devuelve true.
bool anyadirEvento(Fecha, string);
//Elimina un evento del calendario. Si no había ningún evento asociado a esa fecha,
//devuelve false y no hace nada. En caso contrario, devuelve true.
bool eliminarEvento(Fecha);
//Comprueba si hay algún evento asociado a la fecha dada
bool comprobarEvento(Fecha) const;
//Obtiene la descripción asociada al evento. Si no hay ningún evento asociado a la fecha,
//devuelve la cadena vacía
string obtenerDescripcion(Fecha) const;
```

```
//Añade todos los eventos del calendario que se pasa como parámetro al calendario actual,
//excepto los que están en una fecha que ya existe en el calendario
void importarEventos(const Calendario& );
//Devuelve una cadena con el contenido completo del calendario
string aCadena() const;
```

Como el número de eventos que puede contener un calendario puede crecer indefinidamente, se recomienda almacenarlos en un array dinámico, que podrá ir creciendo a medida que se van añadiendo más eventos al calendario. Por tanto, habrá que incluir en la parte privada de la clase un puntero a `Evento`, por ejemplo con `Evento* eventos;`. También es recomendable almacenar el tamaño de ese array en otro atributo privado, como `int tam;`. Inicialmente, aunque el array sea, por ejemplo, de 10 posiciones, ninguna de las 10 almacenará un evento válido. Por lo tanto, deberemos guardar también cuántas posiciones están ocupadas (inicialmente, 0) con `int ocupadas;`.

A la hora de añadir un evento, habrá que almacenarlo en la primera posición libre del array. Y si están todas ocupadas, deberemos reservar memoria para un array más grande, copiar todo el contenido al nuevo array y liberar la memoria del array "pequeño". Cuando borramos un evento, tendremos que desplazar todos los eventos que hay a su izquierda. Cuando queramos iterar sobre todos los eventos, lo haremos sobre todas las posiciones “ocupadas”.

A la hora de convertir un calendario en cadena, se mostrará un evento por línea. En cada línea, primero se mostará la fecha en el formato largo con día de la semana tal y como se describe en la clase`Fecha`, a continuación el símbolo de dos puntos (*:*), y después la descripción del evento, sin dejar ningún espacio en blanco entre los tres elementos. Por ejemplo, para un calendario con un evento planificado 9/9/2024 con descripción `Inicio clases Algoritmia` y otro evento con fecha 11/9/2024 con descripción `Inicio clases PAED`, el método `aCadena` debería devolver:

```
lunes 9 de septiembre de 2024:Inicio clases Algoritmia
miércoles 11 de septiembre de 2024:Inicio clases PAED
```

No es necesario seguir ningún orden particular a la hora de imprimir los eventos.

5 Compilación

Incluye la clase `Fecha` en los ficheros `Fecha.h` y `Fecha.cc`; ~~la clase `Evento` en los ficheros `Evento.h` y `Evento.cc`; y la clase `Calendario` en los ficheros `Calendario.h` y `Calendario.cpp`~~.
Para probar la clase, necesitaremos un programa principal, que se incluirá en el fichero `Ejemplo.cc`.

Compilaremos cada uno de los ficheros `.cc` como:

```
g++ -std=c++11 -Wall -c Fecha.cc
g++ -std=c++11 -Wall -c Ejemplo.cc
```

Finalmente, enlazaremos todos los objetos para obtener el ejecutable:

```
g++ -o Ejemplo Fecha.o Ejemplo.o
```

Se han dejado a disposición del alumnado en Moodle varios ficheros para facilitar el proceso de compilación y desarrollo de la clase Fecha:

- `makefile`: automatiza el proceso de compilación
- `Fecha.h` y `Fecha.cc`: definición de los métodos públicos
- `Ejemplo.cc`: programa principal que ejecuta una serie de pruebas para validar si la implementación de la clase Fecha es correcta

Atención: las pruebas que contiene `Ejemplo.cc` no son exhaustivas. El hecho de que una implementación pase todas las pruebas no implica que carezca de errores. Es obligación del alumnado comprobar que su implementación es acorde a la especificación que se presenta en el enunciado.

6 Entrega y evaluación

No es necesario entregar la práctica. El día 16/10/2024, en los turnos de laboratorio, se llevará a cabo un examen de prácticas en el que deberá realizarse una pequeña modificación sobre el código solución de la práctica. Deberéis emplear el código que habéis desarrollado.

El examen se corregirá automáticamente mediante el sistema de juez en línea `jutge.org`. Sigue las instrucciones de tu profesor de prácticas para registrarte en el sistema y hacer tus primeros envíos.

A Implementación del método `incrementaDias` sin clases

Antes de que se explique cómo funcionan las clases en C++ en clase de teoría, puedes probar tu implementación del método `incrementaDias` como una función. De momento, vamos a ignorar el valor `bool` que devuelve el método y a implementar una función que devuelve `void`, es decir, nada. Pasaremos por referencia el día, mes y año y la función deberá modificarlos de acuerdo al parámetro `incremento`. El siguiente código muestra un programa que emplea la función `incrementaDias` modificar una fecha e imprime el resultado. Puedes emplearlo como base para hacer pruebas.

```
#include <iostream>
using namespace std;

void incrementaDias(int &dia, int &mes, int &anyo, int incremento){
        //Completa esta función
}

int main(){
   int d,m,a;
   d=15;m=2;a=2024;

   cout << "El resultado de incrementar "<<d<<"/"<<m<<"/"<<a<<" en 30 días es ";
   incrementaDias(d,m,a,30);
   cout << d<<"/"<<m<<"/"<<a<< endl; //Debería imprimir 16/3/2024

   return 0;
}
```

Si guardas el programa en un fichero llamado `incrementa.cc`, puedes compilarlo y ejecutarlo introduciendo las siguientes órdenes en el terminal:

```
g++ -Wall -std=c++11 -o incrementa incrementa.cc
./incrementa
```

B Ejemplos de resultados de incrementar/decrementar fechas

Puedes emplear el sitio web `https://www.timeanddate.com/date/dateadd.html` para calcular incrementos y decrementos de fechas y comprobar que tu programa es correcto. Si no sabes por dónde empezar, la siguiente tabla muestra algunos de incrementos y resultados esperados sobre la fecha 17/9/2024.

incremento	Fecha resultante
3	20/9/2024
13	30/9/2024
14	1/10/2024
20	7/10/2024
30	17/10/2024
31	18/10/2024
60	16/11/2024
61	17/11/2024
105	31/12/2024
106	1/1/2025
164	28/2/2025
165	1/3/2025
365	17/9/2025
1536	1/12/2028
-3	14/9/2024
-16	1/9/2024
-17	31/8/2024
-30	18/8/2024
-31	17/8/2024
-60	19/7/2024
-61	18/7/2024
-91	18/6/2024
-92	17/6/2024
-200	1/3/2024
-201	29/2/2024
-202	28/2/2024
-365	18/9/2023
-9287	15/4/1999