Práctica 2: Listas enlazadas

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

Fecha última modificación: 22/10/2024

1 Introducción

Este enunciado describe la primera práctica de laboratorio de Programación Avanzada y Estructuras de Datos. El objetivo general sigue siendo 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.

Dividiremos la implementación en 3 clases principales: `Fecha`, `Evento` y `Calendario`. Cada evento simplemente consta de una fecha y un mensaje. La clase `Calendario` contendrá una colección de calendarios y deberá implementarse como una lista enlazada de objetos de tipo `Evento`, ordenados por fecha. Además, para su funcionamiento requerirá dos clases auxiliares: `NodoCalendario` e `IteradorCalendario`.

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. El funcionamiento deberá ser exactamente el mismo que en la práctica 1, excepto por el hecho de que se añaden los operadores de comparación `operator<` y `operator>`.

```
//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;
//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 operador `operator<` debe devolver `true` si la fecha de la izquierda (la que invoca al operador) es estrictamente anterior a la fecha de la derecha. El operador `operator>` debe devolver `true` si la fecha de la izquierda (la que invoca al operador) es estrictamente posterior a la fecha de la derecha. Por ejemplo, 21/10/2024 es menor que 1/11/2024, pero 21/10/2024 no es menor que 21/10/2024.

3 Clase `Evento`

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&, const 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();
bool operator==(const Evento &) const;
//Operador de comparación
bool operator!=(const Evento &) const;
//Operador de comparación
bool operator<(const Evento &) const;
//Operador de comparación
bool operator>(const Evento &) const;
//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(const 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`.

Para facilitar la ordenación de los eventos en el calendario, los operadores de comparación `operator<` y `operator>` compararán únicamente las fechas de los eventos. Los operadores `operator==` y `operator!=` sí que tendrán en cuenta la descripción, de manera que dos eventos serán iguales únicamente si contienen la misma fecha y 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`

4.1 Funcionamiento general

La clase `Calendario` es la clase principal que representa todos los eventos del calendario. Un objeto de tipo `Calendario` contendrá una serie de objetos de tipo `Evento`, cuyo número podrá crecer indefinidamente. No se permitirá más de un mensaje por fecha, de manera que sólo podrá existir un evento para una fecha determinada. 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();
//Devuelve un iterador al comienzo del calendario
IteradorCalendario begin() const;
//Devuelve un iterador al final del calendario: puntero a nullptr
IteradorCalendario end() const;
//Devuelve el evento apuntado por el iterador
Evento getEvento(const IteradorCalendario & it) const;
//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 insertarEvento(const Fecha&, const 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(const Fecha&);
//Comprueba si hay algún evento asociado a la fecha dada
bool comprobarEvento(const Fecha&) const;
//Obtiene la descripción asociada al evento. Si no hay ningún evento asociado a la fecha,
//devuelve la cadena vacía
string obtenerDescripcion(const 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;
```

A la hora de convertir un calendario en cadena, se mostrará un evento por línea, **en orden cronológico**. 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
```

Los métodos `begin` y `end` permiten emplear un iterador para recorrer los elementos de la lista, y el método `getEvento` permite obtener un evento empleando un iterador. Dichos métodos se describen más adelante.

4.2 Implementación mediante lista enlazada

La clase `Calendario` deberá implementarse **obligatoriamente** como una **lista enlazada**. Por tanto, y según se ha explicado en las clases de teoría de la asignatura, la parte privada de la clase `Calendario` deberá contener en su parte privada un puntero `head` al primer nodo de la lista: `NodoCalendario* head;`. Cada nodo de la lista contendrá un puntero al siguiente nodo de la lista y un objeto de tipo `Evento`.

La clase `NodoCalendario`, que representa un nodo de la lista, deberá tener los siguientes métodos en su parte pública:

```
//Constructor por defecto
NodoCalendario();
//Constructor sobrecargado
NodoCalendario( NodoCalendario*, const Evento& );
//Constructor de copia
NodoCalendario(const NodoCalendario&);
//Operador de asignación
NodoCalendario& operator=(const NodoCalendario &);
//Destructor
~NodoCalendario();
//Devuelve el puntero al siguiente nodo de la lista
NodoCalendario* getSiguiente() const;
//Devuelve el evento almacenado en el nodo
Evento getEvento() const;
//Modifica el puntero al siguiente nodo de la lista
void setSiguiente(NodoCalendario* );
//Modifica el evento
void setEvento(const Evento& );
```

En la parte privada, podríamos declarar el puntero al siguiente nodo como `NodoCalendario* siguiente;` y el objeto de tipo evento como `Evento evento;`

El constructor por defecto deberá inicializar el puntero a siguiente con valor `nullptr` y el evento a los valores correspondientes a su constructor por defecto. Es importante tener en cuenta que al hacer una copia de un nodo mediante el constructor de copia o el operador de asignación **no deberá reservarse memoria**. De igual modo, en el destructor **no deberá liberarse memoria**. La reserva y liberación de memoria de para los nodos se realizará desde la clase `Calendario`.

Los nodos de la lista deberán estar obligatoriamente ordenados de menor a mayor fecha, es decir, en orden cronológico. Los operadores `operator<` y `operator>` de las clases `Evento` y `Fecha` definen dicho orden. Por tanto, al insertar un evento, este debe quedar en la posición correspondiente de la lista de manera que la misma continúe ordenada. La inserción de un evento implica la reserva de memoria para un nuevo nodo que albergará el evento (siempre que no existiera ya un evento para la fecha correspondiente). La eliminación de un evento implica la liberación de la memoria del nodo que lo alberga.

4.3 Iterando sobre eventos

Debido a la implementación enlazada empleada, no es posible acceder a cada evento empleando un índice numérico y, por tanto, no podemos iterar sobre todos los eventos con el típico bucle `for` que emplearíamos en un vector. Para facilitar la iteración sobre los elementos, se emplean objetos de tipo "iterador". En nuestro claso, la clase `IteradorCalendario` no es más que un puntero a un objeto de tipo `NodoCalendario` junto con una serie de métodos que nos permiten ir avanzando por la lista y conocer cuándo hemos llegado al final.

El siguiente ejemplo demuestra cómo emplearemos la clase `IteradorCalendario` para recorrer todos los eventos e imprimir únicamente el año de los mismos:

```
Calendario c;
//Inserciones en el calendario
//..

for(InteradorCalendario it=c.begin(); it != c.end(); it.step()){
    cout << c.getEvento(it).getFecha().getAnyo();
}
```

Para que el siguiente código funcione, deberemos crear una clase `IteradorCalendario` con los siguientes métodos públicos:

```
//Constructor por defecto: puntero a nullptr
IteradorCalendario();
//Constructor de copia
IteradorCalendario(const IteradorCalendario&);
//Destructor: puntero a nullptr
~IteradorCalendario();
//Operador de asignación
IteradorCalendario& operator=(const IteradorCalendario&);
```

```
//Incrementa el iterador en un paso
void step();
//Operador de comparación
bool operator==(const IteradorCalendario&) const;
//Operador de comparación
bool operator!=(const IteradorCalendario&) const;
```

Además, deberemos dar permiso a la clase `Calendario` para acceda al puntero que hay en su interior añadiendo la siguiente instrucción al final de la definición de la clase: `friend class Calendario;`. La parte privada de la clase `IteradorCalendario` simplemente deberá contener un puntero a `NodoCalendario`, es decir: `NodoCalendario* pt;`. El operador asignación y constructor de copia simplemente deberán copiar el puntero, mientras que los operadores de comparación simplemente compararán su valor.

La clave de la clase `IteradorCalendario` está en el método `step`, que modifica el puntero interno `pt` para que apunte al siguiente nodo de la lista. Si el puntero interno está apuntando a `nullptr`, el método `step` no hace nada. Si el puntero interno está apuntando al último nodo de la lista, al aplicar `step` pasará a apuntar a `nullptr`.

Por último, necesitaremos implementar los métodos `begin`, `end` y `getEvento` de la clase `Calendario`. `begin` debe devolver un iterador que apunte al primer nodo de la lista (o a `nullptr` si la lista está vacía), y `end` debe devolver un iterador que apunte a `nullptr`. `getEvento` debe devolver el evento que se encuentra almacenado en el nodo apuntado por el iterador.

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 las clases `Calendario`, `NodoCalendario` e `IteradorCalendario` 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++ -g -std=c++11 -Wall -c Evento.cc
g++ -g -std=c++11 -Wall -c Calendario.cc
g++ -g -std=c++11 -Wall -c Fecha.cc
g++ -g -std=c++11 -Wall -c Ejemplo.cc
```

Finalmente, enlazaremos todos los objetos para obtener el ejecutable:

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

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

- `makefile`: automatiza el proceso de compilación
- `*.h`, `*.cc` : definición de los métodos públicos de todas las clases que hay que implementar

- `Ejemplo.cc`: programa principal que ejecuta una serie de pruebas **muy básicas**. Más adelante se proporcionará una versión que ejecuta pruebas más avanzadas. No obstante, recuerda que **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 20/11/2024, 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`.