Práctica 3: Tipos de datos avanzados de C++ STL

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

Fecha última modificación: 27/11/2024

1 Introducción

Este enunciado describe la tercera práctica de laboratorio de Programación Avanzada y Estructuras de Datos. El objetivo general sigue siendo implementar en C++ un calendario de eventos. El calendario será ligeramente más sofisticado que en la práctica anterior. En dicho calendario, podremos guardar, para cada fecha, un evento con un título, una descripción y una categoría. Igual que en la práctica anterior, podremos consultar y guardar eventos, y el número de eventos que guardemos puede ser ilimitado.

A diferencia de las prácticas anteriores, no habrá que implementar ningún tipo abstracto de datos: todas las estructuras de datos que empleemos provendrán de la biblioteca STL de C++.

Otra diferencia importante es que en esta práctica el programa principal también tendrá que ser implementado por el alumnado.

Dividiremos la implementación, por tanto, en 3 clases (`Fecha`, `Evento` y `Calendario`) y un programa principal.

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 2.

```
//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` ahora deberá almacenar el siguiente contenido:

- La fecha del evento: instancia de la clase `Fecha`

- El título del evento: objeto de tipo `string`. El título nunca podrá estar vacío. Si en algún momento se intenta asignar una cadena de longitud cero (en el método `setTitulo` o en el constructor sobrecargado), esta cadena debe sustituirse por `sin título`.
- La descripción del evento: objeto de tipo `string`. En este caso, la descripción sí que podrá ser una cadena vacía.
- La categoría del evento: número entero, cuyo funcionamiento se explicará más adelante. En todo caso, ha de tenerse en cuenta que nunca podrá tener un valor inferior a -1. Si en algún momento se intenta asignar un valor inferior a -1 (en el método `setCategoria` o en el constructor sobrecargado), este valor debe sustituirse por -1.

La clase deberá tener los métodos que se muestran a continuación en su parte pública:

```
//Constructor por defecto: inicializa la fecha a 1/1/1900
//el título a "sin título"
// y la descripción a cadena vacía
//y la categoría a -1
Evento();
//Constructor sobrecargado: inicializa la fecha, título,
// descripción y categoría a los valores pasados por parámetro (en este orden)
Evento(const Fecha&, const string&,const string&, int);
//Constructor de copia
Evento(const Evento&);
//Operador de asignación
Evento& operator=(const Evento &);
//Destructor: pone la fecha a 1/1/1900, el título a "sin título"
//la descripción a cadena vacía y la categoría a 0
~Evento();
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) el título
string getTitulo() const;
//Devuelve (una copia de) la descripción
string getDescripcion() const;
//Devuelve la categoría
int getCategoria() const;
//Modifica la fecha
void setFecha(const Fecha& );
//Modifica el titulo
void setTitulo(const string &);
```

```
//Modifica la descripción
void setDescripcion(const string &);
//Modifica la categoría
void setCategoria(int);
//Devuelve una cadena con el contenido del evento
string aCadena(const vector<string>&) const;
```

Los operadores `operator==` y `operator!=` tendrán en cuenta los cuatro atributos del evento, de manera que dos eventos serán iguales únicamente si contienen la misma fecha, título, descripción y categoría.

Como se puede ver, los métodos `getFecha`, `getTitulo` 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`.

A la hora de convertir un evento en cadena, primero se mostrará 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 el título del evento, sin dejar ningún espacio en blanco entre los tres elementos. A continuación, y entre los caracteres `[` y `]`, se mostrará la descripción en texto de la categoría del evento. Finalmente, a continuación del carácter `]`, aparecerá un símbolo de dos puntos y la descripción del evento, de nuevo sin dejar ningún espacio en blanco entre ellos.

La descripción en texto de la categoría del evento deberá obtenerse a partir del vector que se pasa como parámetro. Deberá accederse a la posición correspondiente a la categoría del evento que se desea imprimir. Si la categoría del evento es `-1`, la descripción de la categoría será una cadena vacía.

Por ejemplo, para un evento planificado para el 9/9/2024 con título `Inicio clases Algoritmia`, descripción `Preparar diapositivas` y siendo `docencia` el contenido del vector en la posición correspondiente a la categoría, el método `aCadena` debería devolver (sin salto de línea al final):

```
lunes 9 de septiembre de 2024:Inicio clases Algoritmia[docencia]:Preparar diapositivas
```

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 evento 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();
//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 Evento&);
//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 el evento asociado a la fecha. Si no hay ningún evento asociado a la fecha,
//devuelve un objeto de tipo Evento creado con su constructor por defecto
Evento obtenerEvento(const Fecha&) const;
//Devuelve una cadena con el contenido completo del calendario
string aCadena(const vector<string>&) const;
//Deshace la última inserción
void deshacerInsercion();
//Deshace el último borrado
void deshacerBorrado();
//Devuelve una cadena con la información de los eventos que tienen
//como título el primer argumento
string aCadenaPorTitulo(const string&, const vector<string>&) const;
//Devuelve la categoría más frecuente en los eventos
int categoriaMasFrecuente() const;
//Devuelve el día del mes más frecuente en los eventos
int diaMasFrecuente() const;
//Devuelve el mes más frecuente en los eventos
int mesMasFrecuente() const;
//Devuelve el año más frecuente en los eventos
int anyoMasFrecuente() const;
```

A la hora de convertir un calendario en cadena, se mostrará un evento por línea, **en orden cronológico** y **sin salto de línea después del último elemento**. 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 el título del evento, sin dejar ningún espacio en blanco entre los tres elementos. A continuación, y entre los caracteres `[` y `]`, se mostrará la descripción en texto de la categoría del evento. Finalmente, a continuación del carácter `]`, aparecerá un símbolo de dos puntos y la descripción del evento, de nuevo sin dejar ningún espacio en blanco entre ellos.

La descripción en texto de la categoría del evento deberá obtenerse a partir del vector que

se pasa como parámetro. Deberá accederse a la posición correspondiente a la categoría del evento que se desea imprimir. Si la categoría del evento es `-1`, la descripción de la categoría será una cadena vacía.

Por ejemplo, para un calendario con un evento con fecha 9/9/2024, con título `Inicio clases Algoritmia`, descripción `Preparar diapositivas` y `docencia` como contenido del vector en la posición correspondiente a la categoría; y otro evento con fecha 12/9/2024, con título `Análisis de sangre`, descripción `Ir en ayunas` y categoría -1, el método `aCadena` debería devolver:

```
lunes 9 de septiembre de 2024:Inicio clases Algoritmia[docencia]:Preparar diapositivas
jueves 12 de septiembre de 2024:Análisis de sangre[]:Ir en ayunas
```

4.2 Deshaciendo inserciones y borrados

Cada vez que se realice una inserción de un evento con éxito (es decir en una fecha no que existía anteriormente), la clase `Calendario` deberá encargarse de almacenar esa información en la estructura de datos más adecuada para mantener un historial de inserciones. El método `deshacerInsercion` eliminará el evento añadido en la última inserción exitosa. Si se vuelve a llamar, eliminará el evento añadido en la anterior inserción exitosa, y así sucesivamente. Si el método `deshacerInsercion` se invoca cuando no se ha realizado ninguna inserción, o cuando ya se han deshecho todas las inserciones, no deberá hacer nada.

De manera parecida, cada vez que se realice un borrado con éxito, la clase `Calendario` deberá encargarse de almacenar esa información en la estructura de datos más adecuada para mantener un historial de borrados. El método `deshacerBorrado` volverá a insertar el evento borrado en el último borrado exitoso. Si se vuelve a llamar, insertará el evento borrado en el anterior borrado exitoso, y así sucesivamente. Si el método `deshacerBorrado` se invoca cuando no se ha realizado ningún borrado, o cuando ya se han deshecho todos los borrados, no deberá hacer nada.

4.3 Búsqueda por título

No hay ninguna restricción que impida la inserción de más de un evento con el mismo título. El método `aCadenaPorTitulo` deberá devolver una cadena de texto con los eventos que tengan el título que se ha pasado como parámetro. La coincidencia debe ser exacta: por ejemplo, si la cadena de búsqueda pasada como parámetro es `Inicio clases Algoritmia` pero hay un evento con título `inicio clases Algoritmia`, no se considerará ese evento. El formato de la cadena de texto devuelta debe ser el mismo que `aCadena`. Si no hay ningún evento con el título buscado, el método debe devolver la cadena vacía.

4.4 Elementos más frecuentes

Los métodos `categoriaMasFrecuente`, `diaMasFrecuente`, `mesMasFrecuente` y `anyoMasFrecuente` devuelven respectivamente la categoría, día, mes y año más frecuentes de los eventos almacenados en el calendario. En caso de haber varios elementos con la frecuencia más alta, el

método deberá devolver aquel con el valor más alto. Por ejemplo, si tenemos eventos en las fechas 11/9/2024, 25/9/2024, 20/10/2024, 12/12/2024 y 31/12/2024, el método `mesMasFrecuente` deberá devolver `12`. Si el calendario está vacío, los métodos devolverán `-2`.

4.5 Implementación y tiempos de ejecución

Debes elegir el tipo de datos más adecuado de aquellos presentes en la biblioteca STL de C++ (se han explicado en clase de teoría) para la representación de la clase `Calendario`. El objetivo es que todas las operaciones tengan un coste asintótico lo más pequeño posible respecto al número de eventos almacenados en el calendario (que puede crecer hasta el infinito). Es posible que debas incluir en la parte privada de la clase `Calendario` varias estruturas de datos con información redundante para que todas las búsquedas sean eficientes. A la hora de probar la práctica en `jutge.org`, se pondrán límites de tiempo muy ajustados, de manera que se considerará que un programa es erróneo si tarda demasiado tiempo, aunque la salida sea correcta.

5 Programa principal

Para facilitar que podamos probar la clase `Calendario` con muchos eventos, deberá crearse un programa principal que leerá desde entrada estándar (`cin`) tanto los datos de los eventos como las operaciones a realizar con ellos.

Al comienzo del fichero encontraremos los nombres de las categorías, con un nombre de categoría por línea. Tras ellas, aparecerá una línea con el texto `[FIN_CATEGORIAS]`, como se muestra en el siguiente ejemplo:

```
docencia
deporte
reuniones
citas médicas
[FIN_CATEGORIAS]
```

Una vez leídas las categorías, estas deben almacenarse en un `vector` para ser empleadas posteriormente cuando sea necesario imprimir un evento.

A continuación, el fichero contendrá, en una línea cada uno, los métodos a ejecutar. La línea comenzará con el nombre del método (por ejemplo, `categoriaMasFrecuente` o `insertarEvento`; nunca será un método de la forma canónica) seguido de un espacio en blanco. El contenido después del espacio en blanco dependerá de los parámetros del método:

- Si el método tiene un objeto de tipo `Fecha` como parámetro, el día, mes y año aparecerán después del nombre del método, separados por un espacio en blanco. Por ejemplo:`comprobarEvento 12 9 2024`

- Si el método tiene un objeto de tipo `string` como parámetro, el contenido del `string` aparecerá a continuación del espacio en blanco. Para facilitar la lectura, todos los espacios en blanco del parámetro de tipo `string` aparecerán en el fichero transformados en el carácter _. Por ejemplo:`aCadenaPorTitulo Inicio_clases_Algoritmia`
- Si el método tiene un objeto de tipo `Evento` como parámetro, después del nombre del método aparecerán, separados por espacios en blanco, la fecha, el título, la descripción y la categoría (como número entero), en los formatos descritos anteriormente. Por ejemplo:`insertarEvento 9 9 2024 Inicio_clases_Algoritmia Preparar_diapositivas 0`
- Si el método tiene un objeto de tipo `vector<string>` como parámetro, deberá emplearse el vector de categorías leído al principio del fichero.

Tras leer cada línea, el programa deberá ejecutar el método correspondiente e imprimir su resultado por salida estándar (`cout`) seguido de un salto de línea (`endl`):

- Si el tipo de retorno es `void`, no deberá imprimirse nada
- Si el tipo de retorno del método es `int`, deberá imprimirse el entero tal cual
- Si el tipo de retorno es `bool`, deberá imprimirse `0` para `false` y `1` para `true`
- Si el tipo de retorno es `Evento`, deberá imprimirse el resultado de invocar al método `aCadena` del evento correspondiente, pasándole como parámetro el vector construido con los datos que se encuentran al principio del fichero

Tras el último método a ejecutar, el fichero contendrá la línea `[FIN]`. Este podría ser un ejemplo del contenido completo de un fichero de entrada:

```
docencia
deporte
reuniones
citas médicas
[FIN_CATEGORIAS]
insertarEvento 9 9 2024 Inicio_clases_Algoritmia Preparar_diapositivas 0
insertarEvento 10 9 2024 Inicio_clases_PAED Preparar_práctica 0
insertarEvento 14 9 2024 Salida_ciclista Con_el_club_ciclista_del_pueblo 1
insertarEvento 22 11 2024 Vacuna_gripe No_olvidar_tarjeta_SIP 3
insertarEvento 12 9 2024 Reunión_proyecto Preparar_resumen_tareas_último_año 2
deshacerInsercion
eliminarEvento 12 9 2024
eliminarEvento 9 9 2024
aCadena
categoriaMasFrecuente
comprobarEvento 12 9 2024
[FIN]
```

La salida debería ser:

```
1
1
1
1
1
0
1
martes 10 de septiembre de 2024:Inicio clases PAED[docencia]:Preparar práctica
sábado 14 de septiembre de 2024:Salida ciclista[deporte]:Con el club ciclista del pueblo
viernes 22 de noviembre de 2024:Vacuna gripe[citas médicas]:No olvidar tarjeta SIP
3
0
```

6 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. El programa principal, se incluirá en el fichero `main.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 main.cc
```

Finalmente, enlazaremos todos los objetos para obtener el ejecutable:

```
g++ -g -o main Fecha.o Evento.o Calendario.o main.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.
- `main.cc`: programa principal. A completar por el alumnado.
- Carpeta `pruebas`. Contiene una serie de ficheros de entrada (nombre de fichero acabado en `.entrada.txt`) y su salida esperada (nombre de fichero acabado en `.salida.txt`). 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**.

7 Entrega y evaluación

No es necesario entregar la práctica. El día 18/12/2024, en los turnos de laboratorio, se llevará a cabo un examen de prácticas en el que deberá realizarse una 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`.