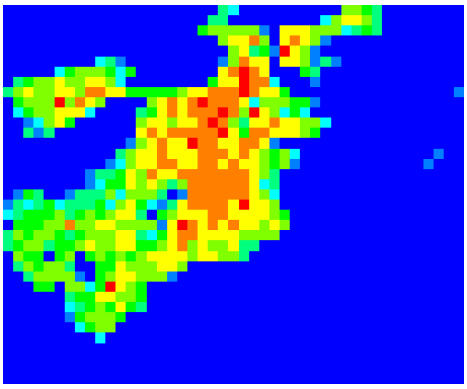
UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

Curso 2024/2025



Guion de prácticas

Boston0

class Coordinates, class Crime

Febrero de 2025

Índice

1. Las prácticas del curso, una visión general

El propósito final de las prácticas que vamos a realizar a lo largo de este curso es trabajar con un conjunto de datos reales sobre distintos tipos de delitos (crímenes), ocurridos en la ciudad de Boston entre los años 2017 y 2021 [1], y elaborar un mapa coroplético (también llamado mapa de calor) de la ciudad, esto es, una imagen en la que cada píxel va a representar un área de la ciudad y su color va a reflejar la frecuencia relativa de crímenes en tal área. Así, un punto muy caliente de la ciudad (donde tienen lugar muchos delitos) se destaca en tonalidades más rojas que otro donde la delincuencia es más baja.

Para abordar el complejo problema anterior, a lo largo de las sucesivas prácticas vamos a ir desarrollando un conjunto de clases que nos permitirán descomponer el problema inicial en problemas más sencillos de **resolver** y de **modelar**. Al mismo tiempo desarrollaremos un conjunto de **aplicaciones operativas** de gestión de datos (en nuestro caso, crímenes) tales como la lectura y filtrado de datos, cálculo de algunos datos estadísticos sencillos, fusión de distintos ficheros con subconjuntos de los crímenes del dataset completo, y terminando con la obtención de una imagen a color con el mapa de calor (similar a la que aparece en la figura de portada) a partir de las frecuencias de crímenes en cada subárea en la que dividamos la zona geográfica completa.

1.1. Los datos

En primer lugar, vamos a detenernos en conocer mejor los datos. El dataset, conocido como *Boston Crime*, está compuesto de un conjunto de registros con información sobre vulneraciones de la ley, que han sido denunciados en diversas comisarías de policía en un periodo de tiempo determinado. Estas infracciones toman diversas apelaciones en función de la gravedad de la ofensa cometida, así, hablamos de hurtos, robos, atropellos, violaciones, homicidios, etc. De forma abreviada, a cada uno de los registros del dataset se le denomina **crimen** [2].

Un crimen registrado en el dataset, contiene multitud de información, tal como (1) un número de orden de registro en el año correspondiente, (2) un identificador del crimen, (3) código de la infracción cometida, (4) descripción abreviada, (7) si se produjo con el agravante de uso de arma de fuego, (8) fecha y hora en la que tuvo lugar y, datos acerca del lugar donde se produjo: (5) distrito, (6) área, (9) dirección y (10) coordenadas GPS. Estas últimas van a ser indispensables para ubicar el crimen en la imagen (ver detalles en la tabla 1).

Los tipos de datos que usaremos para guardar la información de cada campo de un crimen son variados. Usaremos cadenas `string` en la mayoría de los campos, datos de tipo `int`, `bool`, `DateTime` y `Coordinates`.

[1] Los datos con los que vamos a trabajar son una simplificación de los datos originales disponibles en https://www.kaggle.com/datasets/shivamnegi1993/boston-crime-dataset-2022/data.

[2] No tiene por qué ser cruento! ...que has visto muchas películas americanas.

Los dominios y ejemplos de valores pueden encontrarse en la mencionada tabla.

1.2. Clase DateTime

La clase `DateTime` ya se proporciona implementada en su totalidad. Te invitamos a explorar su contenido, aunque lo importante es consultar su interfaz (ver sección 8.5) ya que vas a ser usuario de ella, esto es, vas a necesitar emplear los métodos públicos de la interfaz para definir o consultar el instante de tiempo en que ocurrió un crimen.

La parte privada de la clase tiene campos que guardan información sobre el año, mes, día, hora, minutos y segundos. La clase está dotada de la funcionalidad suficiente para la realización de las prácticas. Dispone de dos constructores, el de por defecto, y el que recibe un `string`. Este último permite construir un objeto `DateTime` con una cadena en el formato "AAAA-MM-dd hh:mm:ss". Además, dispone de métodos de consulta parcial (`get` por cada componente), un método modificador total (un único `set` que modifica todos los componentes del objeto) con una cadena en el mismo formato que en el constructor mencionado más arriba. También dispone de los operadores relacionales <, <=, ==, >, >= y !=. No dispone de los operadores de entrada/salida << y >>.

Ejemplo 1 *Ejemplo de uso de DateTime*

```
DateTime fecha1, fecha2;
DateTime futuro("2025-05-14 12:30:00");
fecha1.set("2024-01-01 00:30:00");
fecha2 = futuro;
if (fecha1 <= fecha2) // comparación
   cout << "fecha1 es anterior a fecha2" << endl;
else
   cout << "fecha2 es anterior a fecha1" << endl;
```

	Campo	Descripción	Tipo	Valor
0	COUNTER	Número entero que representa un contador que comienza desde 0 para cada año.	int	2784
1	IDENTIFIER_NUMBER	Identificador del delito. Es un valor entero almacenado como cadena, por ejemplo, "0020". Es único en un conjunto de datos de delitos.	string	182102975
2	OFFENSE_CODE	Cadena con un código interno cuyo grupo y descripción aparecen en los siguientes campos.	string	3820
3	OFFENSE _CODE_GROUP	Cadena con el grupo del delito.	string	Motor Vehicle Accident Response
4	OFFENSE_DESCRIPTION	Cadena con la descripción del delito.	string	M/V ACCIDENT INVOLVING PEDESTRIAN - INJURY
5	DISTRICT	Cadena con el distrito donde ocurrió el delito.	string	C6
6	REPORTING_AREA	Cadena con el área donde ocurrió el delito.	string	175
7	SHOOTING	Un booleano donde true representa que el delito fue causado por un disparo.	bool	false
8	OCCURRED_ON_DATE	Fecha y hora en que ocurrió el delito.	DateTime	2018-12-22 00:45:00
9	STREET	Nombre de la calle donde ocurrió el delito.	string	SOUTHAMPTON ST
10	LOCATION	Latitud y longitud donde ocurrió el delito.	Coordinates	42.331680, -71.067986

Cuadro 1: Tabla de descripción de datos de delitos

2. Arquitectura de las prácticas

Las prácticas `Boston` se han diseñado por etapas con productos operativos desde la primera etapa. Las primeras etapas contienen clases más sencillas, alguna completamente implementada y operativa, sobre las cuales se asientan otras más complejas. Además, a lo largo del curso se irán incorporando nuevas funcionalidades a clases anteriores. La figura 1 muestra el conjunto de módulos que vamos a emplear (*la arquitectura*) y que se va a ir desarrollando progresivamente en esta y las siguientes sesiones de prácticas. Las estructuras de datos que se van a emplear van a sufrir varias modificaciones que se incorporan conforme se van adquiriendo nuevos conocimientos (los impartidos en teoría que se van a aplicar) o bien conforme llegan nuevas especificaciones (nuevos problemas para resolver). Esta situación no es excepcional, sino que la modificación y evolución de clases es un proceso habitual en cualquier proceso de desarrollo de software.

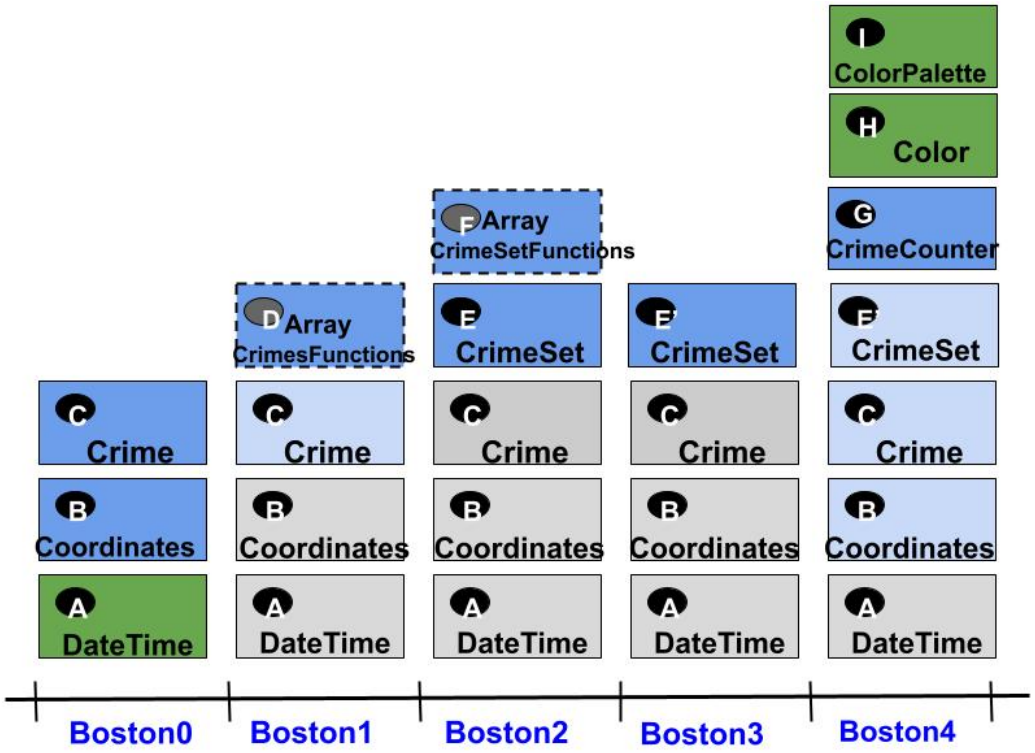


Figura 1: Arquitectura de las prácticas de MP 2025. Como podemos observar, los bloques correspondientes a cada módulo, aparecen en distintos colores. El color verde indentifica software ya desarrollado (completamente terminado), mientras que el azúl indica software que requiere ser desarrollado por el estudiante. Los cambios considerables (como los cambios en la estructura interna de una clase) se muestran en color intenso, mientras que pequeños cambios, como la incorporación de nuevas funcionalidades, se muestran en su correspondiente color suave. Finalmente, en gris se muestran módulos que no sufren cambios respecto a la versión anterior de las prácticas.

Describamos brevemente los módulos que utilizaremos a lo largo de

todas las prácticas de esta asignatura:

A DateTime
Implementa la clase DateTime, una clase ya definida por completo, que va a ser de utilidad para la representación de instantes de tiempo (fecha y hora). El estudiante será usuario de esta clase, esto es, podrá utilizar los métodos públicos de la interfaz, en la construcción de funciones `main()`, en funciones externas y en otras clases que la pueden necesitar.

B Coordinates
Implementa la clase Coordinates, para la representación interna de coordenadas geográficas, útiles para medir posiciones en la Tierra. Consisten en un par de valores `float` (latitud y longitud) que expresan ángulos, expresados en grados. Los valores para la latitud deben estar comprendidos entre -90 (Sur) y 90 (Norte) observados en la vertical, mientras que los valores para la longitud deben estar comprendidos entre -180 y 180 (observados en horizontal) [3].

C Crime
Implementa la clase Crime, una estructura de datos para almacenar toda la información disponible para cada crimen registrado en el dataset Boston Crime. Como ya se ha indicado, consiste en una composición de datos de tipos elementales (int, booleano), cadenas de caracteres (string de la STL) y de otras clases (DateTime y Coordinates).

E CrimeSet
Implementa la clase CrimeSet, una estructura para almacenar un conjunto de objetos de la clase Crime. Será usado para tener una muestra de delitos sobre los que realizar consultas y realizar estadísticas. En una primera aproximación se usará un vector estático.

E' CrimeSet
Manteniendo la interfaz previa de esta clase, se cambiará su estructura interna para alojar el vector de crímenes, de forma más eficiente, en memoria dinámica.

G CrimeCounter
Implementa la clase CrimeCounter, la estructura que va a permitir alojar una matriz bidimensional en memoria dinámica; nos será de utilidad a la hora del recuento de crímenes en cada área en que dividamos nuestra zona de interés, y la obtención de una imagen con el mapa de calor.

H Color
Implementa la clase Color, una clase ya definida por completo, que

[3] Vea por ejemplo la página https://es.wikipedia.org/wiki/Coordenadas_geograficas si quiere ampliar información sobre coordenadas geográficas

va a ser de utilidad para la representación de cada uno de los colores que usaremos en nuestras imágenes mapa de calor. Un objeto de esta clase encapsula un color en el espacio RGB mediante tres componentes: rojo, verde y azul.

I ColorPalette
: Implementa la clase ColorPalette, una clase ya definida por completo, que va a ser de utilidad para definir la lista de colores que usaremos en nuestras imágenes mapa de calor.

Por razones didácticas se van a elaborar dos módulos adicionales, indicados en la figura con rectángulos con bordes en línea discontinua, que contienen funciones externas cuyas funcionalidades serán trasladadas posteriormente al interior de algunas clases. Indicar que estos módulos desaparecen en versiones posteriores aunque **todo el esfuerzo invertido es reaprovechado** en la refactorización del código.

D ArrayCrimesFunctions
: Módulo que contiene funciones externas cuyos parámetros son arrays de objetos Crime, y cuya funcionalidad será introducida posteriormente en la clase CrimeSet.

F ArrayCrimeSetFunctions
: Módulo que contiene funciones externas cuyo propósito es practicar con la memoria dinámica antes del primer parcial de prácticas.

Cada práctica requerirá además de la implementación de alguna(s) funcion(es) externa(s) y de una función `main()` propia para cubrir sus objetivos específicos.

3. Objetivos

El desarrollo de la práctica Boston0 persigue los siguientes objetivos:

- Repasar conceptos sobre clases y los **calificadores** public, private, const y static en métodos y datos miembro.
- Revisar conceptos básicos de funciones: paso de parámetros por valor y referencia y devolución por valor.
- Practicar el paso de objetos por referencia.
- Construir una clase compuesta de datos miembro de otros tipos: la clase `Crime`.
- Repasar el uso de arrays de objetos.
- Profundizar en la lectura de datos por redireccionamiendo desde la entrada estándar.
- Aprender a utilizar una clase a partir de la especificación de su interfaz pública (clase `DateTime`).
- Reforzar la comprensión de los conceptos de compilación separada.

4. Práctica a entregar

4.1. Finalidad del programa

El programa tiene por objeto leer de la entrada estándar un conjunto de datos sobre crímenes cometidos en la ciudad de Boston, mostrando aquellos que han ocurrido dentro de una determinada área de interés y dentro de un determinado periodo de tiempo de interés. El area de interés es un área rectangular definida por dos puntos, esquina inferior izquierda y esquina superior derecha, cuyas coordenadas se leerán de la entrada estándar. El periodo de interés se establece con un tiempo mínimo y un tiempo máximo, que serán leídos también de la entrada estándar. El programa llevará a cabo los siguientes pasos:

- En primer lugar, el programa debe leer de la entrada estándar dos objetos `Coordinates` para definir el área rectangular de interés. Lee en primer lugar las coordenadas de la esquina inferior izquierda y luego las de la esquina superior derecha.
- A continuación, el programa leerá de la entrada estándar dos objetos `DateTime` para establecer el periodo de tiempo de interés. Lee en primer lugar el mínimo periodo de tiempo y luego el máximo periodo de tiempo.
- Seguidamente, leerá de la entrada estándar un número entero n para definir el número de objetos `Crime` a leer a continuación.
- Posteriormente, leerá n objetos `Crime`, que deben ser guardados en un array de objetos `Crime`, teniendo en cuenta que solo deben guardarse aquellos cuyo `DateTime` se encuentre dentro del periodo de tiempo de interés y cuyas coordenadas geográficas sean válidas (vea la descripción del método `Coordinates::isValid()`). Cada objeto `Crime` debe quedar normalizado en el array; para ello puede emplear la función `Normalize()` que debe haber implementado en el fichero `Crime.cpp`.

  Debe tener cuidado de que el número de crímenes insertados en el array no exceda la capacidad de este.
- Finalmente, el programa debe mostrar en la salida estándar todos los crímenes del array y seguidamente todos los crímenes cuyas coordenadas estén dentro del área de interés. Tenga especial cuidado de mostrar la salida tal como se hace en los ejemplos de más abajo.

Indicar que, la forma en que se hace la lectura de los datos de crímenes se establece en esta primera práctica y permanece en el mismo formato en el resto de prácticas.

Ejemplo 2 *Un ejemplo de conjunto de datos que podría proporcionarse al programa por la entrada estándar es el siguiente (cuyo contenido también puede verse en el fichero* `data/input00.b0in`*):*

```
 42.33 , -71.16
 42.35 , -71.07
 2022-01-01 00:00:00
 2022-12-31 23:59:59
6
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,,,0,2022-02-02 00:00:00,WASHINGTON ST,42.343082,-71.141724
1,222648862,3831,UNKNOWN,M/V - LEAVING SCENE - PROPERTY
↪  DAMAGE,B2,288,0,2022-02-05 18:25:00,WASHINGTON ST,42.329750,-71.084541
2,222201764,724,UNKNOWN,AUTO THEFT,C6,200,0,2022-01-09 00:00:00,W
↪  BROADWAY,181.000000,181.000000
3,222201559,301,UNKNOWN,ROBBERY,D4,UNKNOWN,0,2023-03-05 13:00:00,ALBANY
↪  ST,42.333183,-71.073936
4,222111641,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14
↪  12:30:00,WASHINGTON ST,42.349056,-71.150497
5,222107076,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D4,UNKNOWN,0,2022-03-11 10:45:00,MASSACHUSETTS AVE & ALBANY ST
↪  BOSTON
MA 02118 UNI,42.333500,-71.073509
```

Como puede ver, las dos primeras líneas contienen datos sobre coordenadas geográficas que servirán para definir el área de interés. Las dos siguientes son dos periodos de tiempo que servirán para establecer el periodo de tiempo de interés. En la tercera línea, el número entero 6 indica que a continuación deben leerse 6 objetos `Crime`*. Fíjese que los campos de cada crimen están separados por el carácter* `,`*.*

Nota importante:
En el anterior ejemplo puede verse, que algunos campos de un objeto crimen pueden aparecer vacíos. Por ejemplo en los datos del primer crimen están vacíos dos campos. Los únicos campos que no se permite que estén vacíos son el campo COUNTER (debe contener un entero) y el campo IDENTIFIER_NUMBER que debe contener un string con al menos un carácter distinto de blanco o tabulador.

4.2. Sintaxis y ejemplos de ejecución

En esta práctica, la entrada al programa se realizará completamente desde la entrada estándar. Puesto que el número de datos que hay que proporcionar al programa es muy grande, haremos redireccionamiento de la entrada estándar desde un fichero, en lugar de usar el teclado.

La **sintaxis de ejecución** del programa desde un terminal es:

```
linux> dist/Debug/GNU-Linux/boston0 < <inputFile.b0in>
```

Veamos algunos ejemplos de ejecución del programa desde la línea de comandos:

Ejemplo 3 *Redireccionamiento desde el fichero* `data/input00.b0in`

```
linux> dist/Debug/GNU-Linux/boston0 < data/input00.b0in
```

El contenido del fichero `data/input00.b0in` *es el que aparece en el ejemplo 2.*

La salida que debería obtenerse al ejecutar el programa sería la siguiente:

```
Total number of crimes inserted in the array: 4
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,,,0,2022-02-02 00:00:00,WASHINGTON ST,42.343082,-71.141724
1,222648862,3831,UNKNOWN,M/V - LEAVING SCENE - PROPERTY
↪  DAMAGE,B2,288,0,2022-02-05 18:25:00,WASHINGTON ST,42.329750,-71.084541
4,222111641,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14
↪  12:30:00,WASHINGTON ST,42.349056,-71.150497
5,222107076,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D4,UNKNOWN,0,2022-03-11 10:45:00,MASSACHUSETTS AVE & ALBANY ST
↪  BOSTON  MA 02118 UNI,42.333500,-71.073509

Crimes within the given geographic area:
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,,,0,2022-02-02 00:00:00,WASHINGTON ST,42.343082,-71.141724
4,222111641,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14
↪  12:30:00,WASHINGTON ST,42.349056,-71.150497
5,222107076,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D4,UNKNOWN,0,2022-03-11 10:45:00,MASSACHUSETTS AVE & ALBANY ST
↪  BOSTON  MA 02118 UNI,42.333500,-71.073509
```

La salida anterior puede verse también en el fichero `data/input00.b0out`.

En la carpeta `data` del proyecto Boston0, puede encontrar varios ficheros de texto con extensión `.b0in` con los que podrá hacer diferentes pruebas. Esta carpeta también contiene ficheros con extensión `.b0out`. Cada uno de estos ficheros `.b0out` corresponde con el que debería obtenerse ejecutando el programa con el correspondiente fichero de entrada. Así por ejemplo, usando `data/input00.b0in` como entrada debería obtenerse el contenido del fichero `data/input00.b0out`.

Además, se proporcionan una serie de ficheros de tests con extensión `.test` que contienen tests de integración, y que pueden usarse con la script `runTests.sh` para ejecutar automáticamente tales tests de integración según se explica en la sección 7.1.

4.3. Módulos del proyecto

Para la elaboración de la práctica dispone de una serie de ficheros (`Coordinates.h, Crime.h, main.cpp`) con código C++, donde se encuentran las especificaciones de lo que tiene que implementar. Dispone además de los ficheros `DateTime.h, DateTime.cpp` con la clase `DateTime` completamente terminada.

Analice con cuidado las **cabeceras y comentarios de cada función o método**, pues en ellos están detalladas todas sus especificaciones. En la declaración del prototipo de cada función o método, *el número de argumentos* y *los tipos* han sido establecidos y **no se han de cambiar**. Sin embargo, **DEBE REVISAR** la forma en que se hace el paso de argumento para cada parámetro (por valor, por referencia o por referencia constante) y el calificador **const** o (no const) para argumentos y métodos. Hay indicaciones de si los parámetros son de entrada, de salida y si los métodos son de consulta o modificación.

También debe tener en cuenta las **excepciones** que debe lanzar cada función cuando así se indica en la especificación (ver sección 4.4).

En esta práctica debe tener en cuenta las especificaciones indicadas en los siguientes módulos:

- Clase `Coordinates`. El fichero **Coordinates.h** (ver detalles en sección 8.1) contiene la declaración de la clase `Coordinates`. La clase `Coordinates` contiene los datos miembro `float _latitud` y `float _longitud` que se usan para almacenar coordenadas tanto válidas como no válidas.

```
class Coordinates {
...

private:
    const static float INVALID_COORDINATE;
    float _latitude;
    float _longitude;
}; // end class Cordinates
```

  En ocasiones, cuando el valor de un campo es desconocido, por ejemplo una coordenada, se establece un valor fuera del dominio del campo en cuestión. De esta forma, en el caso de tener una localización (Coordinates) desconocida se representa con valores de coordenadas no válidas como por ejemplo 181.0,181.0. Naturalmente, en lugar de usar literales se usa la constante `INVALID_COORDINATE`, declarada en Coordinates.h y definida en el fuente `Coordinates.cpp` [4].

- Clase `Crime`. El fichero **Crime.h** (ver detalles en sección 8.2) contiene la declaración de la clase `Crime` y la declaración de algunas funciones externas como `Trim()`, `Capitalize()` y `Normalize()`. La clase `Crime` contiene datos miembro para guardar cada uno de los campos de un objeto crimen.

```
class Crime {
...
private:
    int _counter; ///< COUNTER: A counter that starts from 0 every new registered year.
    /**
     * IDENTIFIER_NUMBER: Identifier of the crime. The string
     * for this field must contain at least one character
     * different from whitespace and character \t
     */
    std::string _id;
    std::string _code; ///< OFFENSE_CODE: A string with am internal code for the kind of crime
    std::string _group; ///< OFFENSE_CODE_GROUP: A string with a code for the group to which the
      crime belong
    std::string _description; ///< OFFENSE_DESCRIPTION: A string with a brief description of the
      crime
    std::string _district; ///< DISTRICT: the district where the crime took place
    std::string _areaReport; ///< REPORTING_AREA: the area where the crime took place
    bool _shooting; ///< SHOOTING: true if the crime was committed with a gunshot
    DateTime _dateTime; ///< OCCURRED_ON_DATE: date and time when the crime took place
    std::string _street; ///< STREET: name of the street where the crime took place
    Coordinates _location; ///< LOCATION: GPS coordinates where the crime took place

    /**
     * A const string that contains the value "UNKNOWN" that
     * will be assigned by default to string fields in a Crime
     * object.
     */
    static const std::string UNKNOWN_VALUE;

    /**
     * A const string with the default value for the DateTime of
     * a Crime object ("2017-01-20 02:00:00")
     */
    static const std::string DATETIME_DEFAULT;

    /**
     * A string that contains the default value for every field
     * in a Crime object. The content is:
     * "0," + UNKNOWN_VALUE + "," + UNKNOWN_VALUE + "," +
     * UNKNOWN_VALUE + "," + UNKNOWN_VALUE + "," + UNKNOWN_VALUE + "," + UNKNOWN_VALUE
     * + "," + ((std::string)"0") + "," + DATETIME_DEFAULT + "," +
     * UNKNOWN_VALUE + ",181,181"
     */
    static const std::string CRIME_DEFAULT;
}; // end class Crime
```

[4] Cosas del C++, excepto por los enteros, las constantes de otros tipos deben inicializarse en tiempo de ejecución. Comprueba el valor que tiene.

En esta primera práctica, los ficheros `Coordinates.h` y `Crime.h` tienen definidos correctamente todos los prototipos de cada método y función, por lo que no necesita modificarlos. No obstante, estúdielos, ya que en próximas prácticas sí que tendrá que revisar los nuevos métodos y funciones que vayan apareciendo en estos u otros ficheros.

Añada al proyecto los ficheros **Coordinates.cpp** y **Crime.cpp** e implemente los métodos propios y las funciones externas mencionadas.

- Módulo `main.cpp`. Este programa tiene por finalidad lo descrito en la sección 4.1. Teniendo en cuenta los detalles comentados en tal sección y los pasos esbozados en el código proporcionado para esta función (sección 8.4), complete el código de este módulo.

4.4. Las especificaciones sobre excepciones

Las excepciones son una forma de actuar (en tiempo de ejecución) ante circunstancias excepcionales, como cuando no se cumplen las precondiciones de un módulo, lo que podría producir un mal funcionamiento de nuestro programa que, bien pasase inadvertido, o que hiciese que este terminase de forma anómala. Por ejemplo, situaciones que podrían provocar estos problemas serían una división por cero, acceso a una posición fuera del rango dentro de un vector, intento de escribir en un fichero para el que no tenemos permiso de escritura, etc. En esos casos podrían usarse excepciones para que nuestro programa tenga controladas este tipo de situaciones de error.

Cuando se lanza una excepción, si esta no es capturada, esta es enviada a la función llamante. En la función llamante, si la excepción no es capturada, se envía de nuevo a su función llamante y así sucesivamente hasta que la excepción llega a `main()`. Si la excepción tampoco es capturada en `main()`, el programa termina y se muestra su descripción por la salida estándar.

A continuación, vamos a analizar las especificaciones sobre excepciones que nos podemos encontrar en esta asignatura en las cabeceras de métodos o funciones.

En primer lugar, veamos el método `setID()` de la clase `Crime`, que establece el campo ID del objeto `Crime` con el string (tras su normalización) recibido como parámetro. El propósito del ID es distinguir un crimen de otro por su identificador, por lo que no se permite la cadena vacía [5].

En el fichero `Crime.h` podemos ver la siguiente especificación para el método `setID()`

[5] Más concretamente, ni vacía, ni espacios en blanco o similares, de ahí que se le aplique la función Trim().

```
/**
 * @brief Sets the offense ID (primary key for identify the crime in
 * a dataset) of this object using the provided string @p id.
 * First of all, this method trims spaces and \t characters at the
 * beginning and at the end of the provided string. It the resulting
 * string is empty, this method does not modify the ID of this object
 * and an std::invalid_argument exception will be thrown; otherwise
 * it assigns the resulting string to the field for the offense ID.
 * Modifier method
 * @param id the offense ID value to be set. Input parameter
 * @throw std::invalid_argument Throws an std::invalid_argument exception
 * if the provided string @p id is an empty string after removing its
 * initial and final whitespaces and '\t' characters.
 */
void setId(const std::string & id);
```

Como puede verse, se indica que el método lanza la excepción `std::invalid_argument` si el string suministrado es vacío o similar. En posteriores prácticas van a encontrarse alguna excepciones diferentes como por ejemplo `std::out_of_range`. Un ejemplo que se detalla a continuación:

```
/**
 * @brief Gets a const reference to the crime at the given position
 * Query method
 * @param pos position in the CrimeSet. Input parameter
 * @throw std::out_of_range Throws an std::out_of_range exception if the
 * given position is not valid.
 * @return A const reference to the Crime at the given position
 */
const Crime & at(int pos) const;
```

Indicar que por simplicidad, en esta asignatura, nuestro propósito será únicamente detectar estas situaciones de error y especificar dónde o cuál es la situación de error que produjo la excepción, mediante el uso de una cadena de texto que describa tal error y que cada uno podrá definir como quiera. Esta descripción acompañará al nombre de la excepción como puede verse en los ejemplos siguientes. El texto definido se mostrará automáticamente en la salida estándar, en caso de que se lance la excepción durante la ejecución del programa, ya que según acabamos de decir, en caso de que se lance una excepción el programa terminará y se mostrará por la salida estándar la descripción de la excepción.

Veamos cómo vamos a proceder en la implementación al encontrarnos con una excepción en la especificación. En primer lugar, se identifica la excepción especificada. En los ejemplos anteriores tenemos especificada la excepción `std::invalid_argument`, o bien `std::out_of_range`. En segundo lugar, se lanza la excepción indicada con la palabra reservada `throw` **si se cumple la condición especificada** en la cabecera. En el caso del método `setId()`, la condición para lanzar la excepción es que la cadena proporcionada quede vacía tras eliminarle los blancos y tabuladores iniciales y finales. En tal caso se usa `throw <tipo-excepción>(` mensaje identificativo`)` para lanzar la excepción. A continuación se muestra la definición completa del método `setID()`.

```
void Crime::setId(const std::string &id) {
    string trimmedId=id;
    Trim(trimmedId); // eliminados espacios y tabuladores de principio y fin
    if (trimmedId.size() == 0)
        throw std::invalid_argument(
            std::string("void Crime::setId(const std::string &id): ") + // descripcion del lugar en que se
      produjo la excepción
            "id is empty"); // mensaje descriptivo del error
    this->_id = trimmedId;
}
```

Indicar de nuevo que, si se capturase una excepción se podría arreglar la situación de error, pero por simplicidad, en la asignatura Metología de

la Programación, tan solo vamos a usar la parte `throw` sin la componente `catch`; o sea, nunca vamos a capturar excepciones en esta asignatura. Si quiere saber más: (Abrir →).

5. Configuración de las prácticas

Para la elaboración de la práctica dispone de una serie de ficheros que se encuentran en el repositorio de `github`. Una vez descargado, va a encontrar entre otros, los ficheros `DateTime.h`, `Coordinates.h`, `Crime.h` y `main.cpp`. También encontrará ficheros con extensión `.b0in` (ficheros a usar como entrada estándar al programa) y ficheros con extensión `.b0out`, que contienen la salida esperada al ejecutar el programa con la entrada correspondiente. El fichero `documentation.doxy` es un fichero de texto con la configuración para ejecutar el programa `doxygen`.

Para montar la primera práctica, tendrá que crear un proyecto nuevo según se especifica en la sección 6. Pero antes, veamos cómo definir el espacio de trabajo.

El espacio de trabajo

ProyectosNetBeans : Carpeta raíz en la que se van a crear todos los proyectos de la asignatura. Tendremos un directorio por cada Boston, que se genera en el momento de crear un proyecto NetBeans. La estructura que deberíamos de tener es la que se muestra a continuación:

```
NetbeansProjects
├── Debugger*
├── HelloWorld
├── Boston0
├── Boston1
├── *
├── MPGeometry*
├── MyVector
├── Scripts
└── ValgrindShowcase
```

Figura 2: Árbol de directorios de la rama NetbeansProjects de la asignatura MP. Los directorios Boston se declinan de Boston0 a Boston4.

DataSets: Un conjunto de ficheros que contienen datasets de crímenes de diferentes tamaños, muchos procedentes del Boston Crime y otros sintetizados ad hoc para la realización de tests de prácticas. Así tendremos datasets que van desde 0 a los 446093 del data set completo. Son ficheros en formato texto y tiene la extensión `.crm`.

Scripts: Una serie de scripts Bash de apoyo a las funciones de NetBeans.

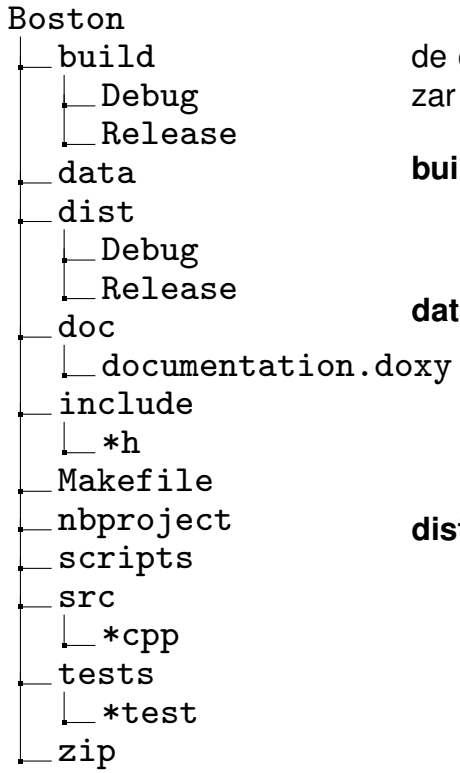


Figura 3: Estructura interna de cada proyecto Boston

Para tener bien ordenado el contenido de cada proyecto Boston, se va a organizar en las siguientes carpetas:

build Es una carpeta temporal que contiene código precompilado y pendiente de enlazar.

data Lo usaremos en las primeras prácticas para contener ficheros con ejemplos de entrada al programa y la correspondiente salida esperada al ejecutarlo.

dist Contiene los binarios ejecutables. Vamos a generar dos versiones por cada programa:

- **Release** Es la versión más eficiente y que menos espacio ocupa. Es la que debería entregarse al cliente.
- **Debug** Es la versión sobre la que se depuran errores y se corrigen defectos. Ocupa más espacio que la anterior ya que contiene información extra para poder usar el depurador. Puede llegar a ser el doble de grande.

doc Aquí se almacena la documentación del proyecto, tanto del código, como la generada por los tests.

include Contiene los ficheros de cabeceras **.h**.

nbproject Es la carpeta que utiliza NetBeans para almacenar la configuración del proyecto.

scripts En esta carpeta colocaremos los scripts de apoyo a Netbeans que se han desarrollado en esta asignatura y se puede ampliar con otros más. Se utilizarán en prácticas posteriores. Por ahora se hace todo manual.

src Contiene los ficheros fuente del proyecto **.cpp**.

tests Contiene los tests de integridad de cada proyecto. Son ficheros con extensión **.test**.

zip Carpeta para dejar las copias de seguridad del proyecto.

6. Configurar el proyecto en NetBeans

Para crear un proyecto Boston desde cero.

1. Crear el proyecto en NetBeans como `C++ Application` con nombre Boston0 para esta práctica.

   Crear las carpetas: `include, src, data, doc, scripts, zip`, dentro del directorio Boston0. Esto se puede hacer desde un terminal usando el comando `mkdir <nombreDir>` o bien desde el entorno de NetBeans, a través de la pestaña **File**. Es necesario añadir las carpetas una a una.

   Carpetas como `build, dist, nbproyect` son creadas y gestionadas automáticamente por NetBeans, por lo que no deben tocarse.

2. Colocar cada uno de los ficheros en su sitio, los ficheros `*.h` en `include`, los `*.cpp` en `src`, etc. Estas acciones y las realizadas en el paso anterior se han llevado a cabo a través del S.O.. Compruébalo con la instrucción unix: `ls`. La estructura debe ser como la indicada en la figura 3.

3. Propiedades de proyecto. Compilador

   a) Poner el estándar a C++14.

   b) Añadir en `Include Directories` la carpeta con los ficheros de cabecera del propio proyecto (**./include**).

   c) En `Additional Options` es interesante añadir las opciones `-Wall` (activar todos los mensajes de advertencia del compilador) y `-pedantic` (da mensajes de advertencia si nuestro programa no sigue las normas ISO C e ISO C++). Por ejemplo, el uso de `-pedantic` hará que nos aparezca una advertencia si declaramos un array de un tamaño que no sea una constante, algo prohibido en esta asignatura:

   ```
   int main(){
       int variable=10;
       double array[variable];
   }
   ```

4. Desde la vista lógica del proyecto.

   a) En `Header Files`, `Add existing item`: ficheros `DateTime.h`, `Coordinates.h` y `Crime.h`.

   b) En `Source Files`, `Add existing item`: ficheros `DateTime.cpp`, `Coordinates.cpp`, `Crime.cpp` y `main.cpp`.

5. Sobre el nombre del proyecto, botón derecho `Set As Main Project` o desde el menú principal `Run` – `Set Main Project` y seleccionar este proyecto.

6. Con esto ya se podría ejecutar el proyecto normalmente. Obviamente, no hace nada porque todavía está vacío. A partir de aquí empezaría el desarrollo del proyecto para el que cada programador seguirá un orden determinado y, probablemente, diferente a todos los demás.

   Nota: No deje para el final la comprobación del correcto funcionamiento de todos los métodos y no compruebe solo los métodos utilizados en `main()`. **Todos los métodos han de ser testados** para comprobar que cada uno funciona como se espera, aunque no sea utilizado en el `main()` solicitado para la práctica [6]. Para ello deberá realizar una batería de pruebas.

La práctica deberá ser entregada en Prado, durante el periodo habilitado en la propia actividad de entrega, y consistirá en un fichero ZIP del proyecto. Se puede montar el zip desde NetBeans, a través de **File** → **Export project** → **To zip**. El nombre Boston0.zip, sin más aditivos, es el mismo nombre para todos los estudiantes. Para ello se sugiere utilizar el script `runZipProject.sh` (ver sección 7). Compruebe que lo entregado es compilable y operativo y, no olvide poner el nombre de los componentes del equipo en la cabecera del fichero `main.cpp`.

7. Uso de scripts

En primer lugar, hemos de completar nuestro espacio de trabajo con la carpeta **Scripts**, también disponible en el repositorio de la asignatura (ver figura 2).

Como ya se indicara, la carpeta **Scripts** contiene una serie de scripts bash de apoyo a las funciones de NetBeans. Es única para todos los proyectos NetBeans, y se encuentra al mismo nivel que las carpetas de los proyectos Boston.

Sin embargo, existe una carpeta `scripts` (con `s` minúscula) por cada uno de los proyectos Boston (ver figura 3). No confundir con la carpeta `Scripts` anterior. La carpeta `scripts` es la que contiene los scripts que tú vas a poder ejecutar manualmente [7].

Como se ha indicado, en la carpeta `scripts` se encuentran una serie de utilidades bash, ficheros con extensión `*.sh` tales como:

- `runUpdate.sh` que actualiza los scripts de la carpeta `scripts` con los scripts actualizados que hayamos descargado del repositorio en la carpeta `Scripts`.
- `runDocumentation.sh` que va a facilitar el proceso de generación de la documentación con doxygen y su visualización en un navegador.

[6] Considere por ejemplo todos los constructores que se han declarado en la clase *Crime* y, seguramente no se emplearán todos en la función `main()` de boston0 pero sí en sucesivas prácticas... Cuando se define una clase, es cuando esta se depura, incluso antes.

[7] Estos hacen uso de las utilidades en Scripts; por ello, para un correcto funcionamiento, es importante que `Scripts` esté correctamente ubicada en su lugar.

- `runZipProject.sh` que facilita el proceso de elaborar un zip, eliminando previamente todos los archivos binarios que no tiene sentido exportar como `*.bin, *.o,` etc. Guarda en la carpeta zip, el fichero que tiene que entregar en Prado.
- `runTests.sh` que ejecuta los tests de integración disponibles en la carpeta `tests` del proyecto proporcionado en el repositorio.

Para ejecutar un script desde NetBeans, basta con situarse con el cursor del ratón sobre el script, pulsar botón derecho del ratón y elegir `Run`. Por ejemplo, en el caso del script `runZipProject.sh`, tras ejecutar el script, se obtendrá un fichero con extensión `.zip` en el directorio `zip`.

7.1. El script runTests.sh

Con cada proyecto Boston se proporcionan varios tests de integración, cada uno de ellos en un fichero con extensión `.test`, y que deben estar colocados en la carpeta `tests`. Un test de integración es una prueba de nuestro programa, en la que se definen los datos que nuestro programa lee y la salida que debería producir con esos datos.

Por ejemplo, el fichero `crimes6_easy.test` del proyecto Boston0, tiene el contenido que se muestra a continuación:

```
%%%CALL < data/input00.b0in
%%%DESCRIPTION Easy test with 6 Crimes almost complete to read, already
↪  capitalized without any whiteSpace to be trimed
%%%OUTPUT
Total number of crimes inserted in the array: 4
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,,,0,2022-02-02 00:00:00,WASHINGTON ST,42.343082,-71.141724
1,222648862,3831,UNKNOWN,M/V - LEAVING SCENE - PROPERTY
↪  DAMAGE,B2,288,0,2022-02-05 18:25:00,WASHINGTON ST,42.329750,-71.084541
4,222111641,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14
↪  12:30:00,WASHINGTON ST,42.349056,-71.150497
5,222107076,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D4,UNKNOWN,0,2022-03-11 10:45:00,MASSACHUSETTS AVE & ALBANY ST
↪  BOSTON  MA 02118 UNI,42.333500,
-71.073509

Crimes within the given geographic area:
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,,,0,2022-02-02 00:00:00,WASHINGTON ST,42.343082,-71.141724
4,222111641,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14
↪  12:30:00,WASHINGTON ST,42.349056,-71.150497
5,222107076,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D4,UNKNOWN,0,2022-03-11 10:45:00,MASSACHUSETTS AVE & ALBANY ST
↪  BOSTON  MA 02118 UNI,42.333500,
-71.073509
```

En la asignatura no es imprescindible entender el formato de estos ficheros de tests, pero lo comentamos aquí brevemente. En el test anterior, se indica que si el programa `Boston0` se llama redirigiendo la entrada estándar desde el fichero `data/input00.b0in`, entonces la salida que debería producir el programa es la que se muestra tras la línea `%%%OUTPUT`. El contenido del fichero `data/input00.b0in` aparece en la sección 4.2.

Para ejecutar los tests de integración, usaremos el script `runTests.sh`. Hay varias formas de hacerlo:

1. Desde NetBeans podemos ejecutar todos los tests de integridad de la siguiente forma: seleccionamos la vista física del proyecto (*Files*), situamos el cursor del ratón sobre el script, pulsamos botón derecho del ratón y elegimos `Run`.

2. Desde un terminal de Netbeans o de nuestro sistema operativo Linux podemos ejecutar todos los tests de integridad situándonos dentro de la carpeta de nuestro proyecto y ejecutando:

```
Linux> script/runTests.sh
```

3. Desde un terminal de Netbeans o de nuestro sistema operativo Linux podemos ejecutar uno solo de los tests situándonos dentro de la carpeta de nuestro proyecto y ejecutando:

```
Linux> script/runTests.sh tests/nombreTestAEjecutar.sh
```

Al hacerlo con cualquiera de las opciones anteriores, nos aparecerá en pantalla la lista de tests ejecutados y podremos ver si nuestro programa los ha pasado o no. Por ejemplo, en Boston0, si ejecutamos desde un terminal el script de la siguiente forma:

```
Linux> script/runTests.sh tests/crimes6_easy.test
```

obtendremos por la salida (en caso de que nuestro programa pase el test) lo siguiente:

```
Loading ansiterminal...

VALIDATION TOOL v2.0
Checking tests folder           OK
Setting everything up           OK
Test #1  Easy test with 6 Crimes almost complete to read,
 ↪  already capitalized without any whiteSpace to be trimed

Testing tests/crimes6_easy.test                 Generating
 ↪  fresh binaries
[  OK  ] Test #1 [tests/crimes6_easy.test] (
 ↪  dist/Debug/GNU-Linux/boston0teacher  < data/input00.b0in)
End of tests
```

8. Código para la práctica

8.1. Coordinates.h

```
/*
 * Metodología de la Programación
 * Curso 2024/2025
 */

/**
 * @file Coordinate.h
 *
 * Created on September 25, 2024, 1:03PM
 */

#ifndef COORDINATES_H
#define COORDINATES_H

/**
 * @class Coordinates
 * @brief GPS Coordinates (geographic coordinates) for measuring positions in
 * the Earth. It consists of a pair of float values (latitude and longitude)
 * that express angles (degrees). See for example
 * <a href="https://en.wikipedia.org/wiki/Geographic_coordinate_system">
 * Wikipedia Geographic coordinate system</a> <br>
 * Values for latitude are between -90 and 90, while values for longitude
 * must be between -180 and 180. It is possible to build a Coordinates object
 * with a latitude or longitude out of that ranges, but that object will be
 * considered as invalid (method \ref isValid() will return true).<br>
 * The value latitude == 0 is the EQUATOR, and the value longitude == 0 is the
 * Greenwich meridian. <br>

 * - A point with a positive value for latitude is located in the northern
 * hemisphere; a point with latitude==90 is in the North pole. <br>
 * - A point with a negative latitudeis located in the southern hemisphere. <br>
 * - A point with a positive longitude is located to the right of the Greenwich
 * meridian. <br>
 * - A point with a negative longitude is located to the left of the Greenwich
 * meridian. <br>
 * <br>
 *
 * Examples: <br>
 * - Granada: 37.178056,-3.600833
 * - Valencia: 39.47,-0.376389
 * - Boston: 42.360278,-71.057778
 * - Quito: -0.22, -78.5125
 * - Johannesburgo: -26.204444, 28.045556
 * <br>
 */


#include <iostream>
#include <string>

class Coordinates {
public:

    /**
     * @brief Constructor that builds an object with the given latitude
     * and longitude parameters. Both parameters have the value
     * INVALID_COORDINATE as default value.
     * @param latitude The latitude value to set. Default value
     * INVALID_COORDINATE.
     * Input parameter
     * @param longitude The longitude value to set. Default value
     * INVALID_COORDINATE.
     * Input parameter
     */
    Coordinates(float latitude = INVALID_COORDINATE,
            float longitude = INVALID_COORDINATE);

    /**
     * @brief Gets the latitude value
     * Query method
     * @return The latitude as a float.
     */
    float getLatitude() const;

    /**
     * @brief Gets the longitude value.
     * Query method
     * @return The longitude as a float.
     */
    float getLongitude() const;

    /**
     * @brief Checks if the current coordinates are valid; that is it
     * determines if the latitude and longitude values are within
     * the aceptable ranges for geographical coordinates. <br>
     * Latitude must be within the range [-90, 90], and longitude within the
     * range [-180,180]. <br>
     * Query method
     * @return true if the coordinates are valid, false otherwise.
     */
    bool isValid() const;
```

```
    /**
     * @brief Sets the latitude value in this object.
     * Modifier method
     * @param latitude The latitude value to set. Input parameter
     */
    void setLatitude(float latitude);

    /**
     * @brief Sets the longitude value in this object.
     * Modifier method
     * @param longitude The longitude value to be set. Input parameter
     */
    void setLongitude(float longitude);

    /**
     * @brief Sets both components, the latitude and longitude values.
     * Modifier method
     * @param latitude The latitude value be to set. Input parameter
     * @param longitude The longitude value to be set. Input parameter
     */
    void set(float latitude, float longitude);

    /**
     * @brief Obtains a string with the latitude and longitude (comma separated).
     * To convert latitude and longitude to string you must use the
     * std::to_string() function.
     * Query method
     * @return A string representing the coordinates in the specified format.
     */
    std::string toString() const;

    /**
     * @brief Computes the dimension of the rectangle bounded by two
     * Coordinates objects (this object and the one provided as parameter
     * to this method).
     * The dimension of a rectangle that goes from Coordinates $A(lat_1,long_1)$
     * to $B(lat_2,long_2)$ is calculated as:
     * \f$ (|lat_2-lat_1|,|long_2-long_1|) \f$
     * The length is symmetric, that is, A.length(B) is equal to B.length(A).
     * This is valid whenever the area does not cross the meridian 180
     * (Greenwich antimeridian) which is the international date change line.
     * Query method.
     * @param second A Coordinates object B. Input parameter.
     * @return A Coordinates object that will contain the dimension of the
     * rectangle bounded by two Coordinates objects (this object and the one
     * provided as parameter to this method).
     */
    Coordinates length(const Coordinates &second) const;

    /**
     * @brief Verifies if this object Coordinates is inside the area
     * determined by the two given Coordinates A (@p bottomLeft) and
     * B (@p topRight).
     *  ____
     *      +-----------+(B)
     *      |           |
     *      |           |  +(O3)
     *      |           |
     *      |    +(O2)  |
     *      |           |
     *      |           |
     *   (A)+-----------+
     *         +(O1)
     *  ____
     *
     * Examples of false return:
     * - Object O3 surpasses longitude of B
     * - Object O1 has a latitude below that of A.
     * Example of true return:
     * - Object O3 is inside the given area
     *
     * Query method.
     * @pre both @p bottomLeft and @p topRight coordinates are valid and
     * they define an area that does not cross the Greenwich antimeridian,
     * that is, A <= B both in latitude and longitude; otherwise,
     * unexpected return value could be obtained. This is not checked in this
     * method.
     * @param bottomLeft The Coordinates of the bottom left point. Input parameter
     * @param topRight The Coordinates of the top right point. Input parameter
     * @return true if the Coordinates of this object is inside the area
     * determined by the two given Coordinates
     */
    bool isInsideArea(const Coordinates &bottomLeft, const Coordinates &topRight) const;

private:
    const static float INVALID_COORDINATE;  ///< Constant that represents an invalid latitude or longitude
    float _latitude; ///< Latitude
    float _longitude; ///< Longitude
}; // end class Coordinates

#endif /* COORDINATES_H */
```

8.2. Crime.h

```
/*
 * Metodología de la Programación
 * Curso 2024/2025
 */

/**
 * @file Crime.h
 *
 * @note To be implemented by students
 * Created on September 10, 2024, 6:26PM
 */

#ifndef CRIME_H
#define CRIME_H

#include <iostream>
#include <string>

#include "Coordinates.h"
#include "DateTime.h"

/**
 * @class Crime
 * @brief It stores data about a crime occurred in Boston city between
 * 2017-01-20 02:00:00 and 2022-02-02 00:00:00
 * The data of each crime comes from a csv file located in
 * https://www.kaggle.com/datasets/shivamnegi1993/boston-crime-dataset-2022/data
 * where some fields will be ignored.
 *
 * The selected fields for a Crime are:
 * - 0 COUNTER: integer number that represents a counter that begins
 *      from 0 for each year.
 * - 1 IDENTIFIER_NUMBER: Identifier of the crime. It is an integer value
 *      stored as a string, that is, like "0020". It is unique in a dataset of
 *      crimes. The string for this field must contain at least one character
 *      different from whitespace and \t character. An ID will never contain
 *      leading or trailing blanks.
 * - 2 OFFENSE_CODE: string with an internal code whose group and description
 *      come in the next two fields.
 * - 3 OFFENSE_CODE_GROUP: string with the group of the crime.
 * - 4 OFFENSE_DESCRIPTION: string with the description of the crime.
 * - 5 DISTRICT: string with the district where the crime took place.
 * - 6 REPORTING_AREA: string with the area where the crime took place
 * - 7 SHOOTING: a boolean where true represents that the crime was caused by
 *      a gunshot.
 * - 8 OCCURRED_ON_DATE: date and time when the crime took place.
 * - 9 STREET: name of the street where it took place.
 * - 10 LOCATION (latitud, longitude) where it took place.

 * Example of a crime:
 * 2784,182102975,3820,Motor Vehicle Accident Response,M/V ACCIDENT INVOLVING PEDESTRIAN - INJURY,C6
     ,175,0,2018-12-22 00:45:00,SOUTHAMPTON ST,42.331680,-71.067986 <br>
 * corresponding values to the fields: <br>
 * COUNTER 2784 <br>
 * IDENTIFIER_NUMBER 182102975 <br>
 * OFFENSE_CODE 3820 <br>
 * OFFENSE_CODE_GROUP Motor Vehicle Accident Response <br>
 * OFFENSE_DESCRIPTION M/V ACCIDENT INVOLVING PEDESTRIAN - INJURY <br>
 * DISTRICT C6 <br>
 * REPORTING_AREA 175 <br>
 * SHOOTING false <br>
 * OCCURRED_ON_DATE 2018-12-22 00:45:00 <br>
 * STREET SOUTHAMPTON ST <br>
 * LOCATION 42.331680,-71.067986 <br>
 */
class Crime {
public:
    /**
     * @brief default constructor that builds a Crime object with a default
     * value for every field. The default values for each field are obtained
     * from the string @ref Crime::CRIME_DEFAULT
     */
    Crime();

    /**
     * @brief Builds a Crime object with the data contained in the provided
     * string. This string is a line of a csv file with fields separated by
     * commas. The details are described in the documentation of this class.
     * @pre The @p line parameter has the same preconditions as the method
     * Crime::set(const std::string & line)
     * @param line a string with the data to build the object. Input parameter
     */
    Crime(const std::string & line);

    /**
     * @brief Returns the counter of this Crime
     * Query method
     * @return The counter of this Crime
     */
    int getCounter() const;

    /**
     * @brief Returns the offense ID, a primary key for identify any crime in
     * a dataset. This ID is unique in a dataset of crimes.
     * Query method
     * @return The offense ID.
     */
```

```
    std::string getId() const;

    /**
     * @brief Returns the offense code of this crime.
     * Query method
     * @return The offense code number
     */
    std::string getCode() const;

    /**
     * @brief Returns the group offense of this crime. It is a string that may
     * contain identifying words to group the offense
     * Query method
     * @return The group offense of this crime
     */
    std::string getGroup() const;

    /**
     * @brief Returns the description for this crime. It contains some keys
     * words for the description of this crime.
     * Query method
     * @return The description of this crime
     */
    std::string getDescription() const;

    /**
     * @brief Returns the name of the district for this crime.
     * Query method
     * @return The district of this crime
     */
    std::string getDistrict() const;

    /**
     * @brief Returns the name of street where this crime took place.
     * Query method
     * @return The street where this crime took place
     */
    std::string getStreet() const;

    /**
     * @brief Returns the coded area where this crime took place. It is a
     * a string that may contain a letter and a number
     * Query method
     * @return The area of this crime
     */
    std::string getAreaReport() const;

    /**
     * @brief Returns if the crime was caused by a gunshot.
     * Query method
     * @return true if this crime was caused by a gunshot; false otherwise
     */
    bool isShooting() const;

    /**
     * @brief Returns the date and time of this crime.
     * Query method
     * @return The date and time
     */
    DateTime getDateTime() const;

    /**
     * @brief Returns the value of the Coordinates where this crime took place.
     * Query method
     * @return Coordinates where this crime took place
     */
    Coordinates getLocation() const;

    /**
     * @brief Returns if the object has UNKNOWN ID.
     * Query method
     * @return bool true if the crime ID is equals to constant UNKNOWN_VALUE;
     * false otherwise.
     */
    bool isIDUnknown() const;

    /**
     * @brief Obtains a string with the fields of this Crime object, by using
     * comma as separator. Fields appears in the following order:
     * - 0  COUNTER <br>
     * - 1  IDENTIFIER_NUMBER <br>
     * - 2  OFFENSE_CODE <br>
     * - 3  OFFENSE_CODE_GROUP <br>
     * - 4  OFFENSE_DESCRIPTION <br>
     * - 5  DISTRICT <br>
     * - 6  REPORTING_AREA <br>
     * - 7  SHOOTING <br>
     * - 8  OCCURRED_ON_DATE  <br>
     * - 9  STREET <br>
     * - 10 LATITUDE <br>
     * - 11 LONGITUDE <br>
     *
     * Depending on the type of the field, it may be necessary to convert the
     * value to a string as for example with: _counter, _shotting, _dateTime,
     * _location:
     * - In the case of _counter the string will be obtained using the
     * std::to_string() function.
     * - In the case of _shotting the  string will be "0" or "1".
     * - In the case of _dateTime and _location the string will be
     * obtained with the toString() method of the corresponding class.
```

```
     *
     * Query method
     * @return string that contains the fields of this Crime in csv format
     * (by using commas as separator).
     */
    std::string toString() const;

    /**
     * @brief Sets the counter for this crime with the provide value
     * Modifier method
     * @param c the counter value to be set. Input parameter
     */
    void setCounter(int c);

    /**
     * @brief Sets the offense ID (primary key for identify the crime in
     * a dataset) of this object using the provided string @p id.
     * First of all, this method trims spaces and \t characters at the
     * beginning and at the end of the provided string. It the resulting
     * string is empty, this method does not modify the ID of this object
     * and an std::invalid_argument exception will be thrown; otherwise
     * it assigns the resulting string to the field for the offense ID.
     * Modifier method
     * @param id the offense ID value to be set. Input parameter
     * @throw std::invalid_argument Throws an std::invalid_argument exception
     * if the provided string @p id is an empty string after removing its
     * initial and final whitespaces and '\t' characters.
     */
    void setId(const std::string & id);

    /**
     * @brief Sets the offense code of this crime.
     * Modifier method
     * @param code the offense code value to be set. Input parameter
     */
    void setCode(const std::string &code);

    /**
     * @brief Sets the group offense that belongs the crime.
     * Modifier method
     * @param group the offense group value to be set. Input parameter
     */
    void setGroup(const std::string &group);

    /**
     * @brief Sets the description for the offense.
     * Modifier method
     * @param description the offense description value to be set.
     * Input parameter
     * @return A string that contains some keys words for the description
     */
    void setDescription(const std::string & description);

    /**
     * @brief Sets the name of the district for the crime.
     * Modifier method
     * @param district the identifier for the district to be set. Input parameter
     */
    void setDistrict(const std::string &district);

    /**
     * @brief Sets the coded area where the offense took place.
     * Modifier method
     * @param areaReport the area value to be set. Input parameter
     * @return A string that may contain a letter and a number
     */
    void setAreaReport(const std::string &areaReport);

    /**
     * @brief Sets the name of street where the offense  took place.
     * Modifier method
     * @param street the street name that may contain the name with the type of
     * route to be set. Input parameter
     */
    void setStreet(const std::string &street);

    /**
     * @brief It gives the aggravating evidence for the crime.
     * Modifier method
     * @param shotting  the boolean value for shotting to be set. Input parameter
     */
    void setShooting(bool shotting);

    /**
     * @brief Sets the DataTime field (date and time) of this crime object
     * with the data in the provided string \p time. If \p time is empty or it
     * is in invalid format, the date and time of this object will be set with
     * @ref DateTime::DATETIME_DEFAULT.
     *
     * @note The implementation of this method needs only call to
     * method DateTime::set(time).
     *
     * Modifier method
     * @param time Datetime in the format specified in @ref DateTime.
     * Input parameter
     */
    void setDateTime(const std::string & time);

    /**
     * @brief Sets the coordinates where the crime took place.
```

```
     * Modifier method
     * @param coordinates the coordinates value to set. Input parameter
     */
    void setLocation(const Coordinates & coordinates);

    /**
     * @brief It sets all the data members of the object by extracting their
     * values (in the order indicated in the below list) from the provided string
     * @p line that describes the crime in csv format, that is, concatenation
     * of every field by using commas as separator between fields.
     * To set the value of each field in this class, this method will use each
     * one of the set methods in this class
     *
     * The values to set the object are:
     * - 0  COUNTER <br>
     * - 1  IDENTIFIER_NUMBER <br>
     * - 2  OFFENSE_CODE <br>
     * - 3  OFFENSE_CODE_GROUP <br>
     * - 4  OFFENSE_DESCRIPTION <br>
     * - 5  DISTRICT <br>
     * - 6  REPORTING_AREA <br>
     * - 7  SHOOTING <br>
     * - 8  OCCURRED_ON_DATE  <br>
     * - 9  STREET <br>
     * - 10 LATITUDE <br>
     * - 11 LONGITUDE <br>
     *
     * Modifier method

     * @pre The string for the COUNTER field must contain an integer number;
     * otherwise an unpredictable behavior or runtime error will be obtained
     * @pre The string for the ID field must contain at least one character
     * different from whitespace and \t character; otherwise an unpredictable
     * behavior or runtime error will be obtained
     * @pre The string for the SHOOTING field must contain "0" or "1";
     * otherwise an unpredictable behavior or runtime error will be obtained
     * @pre The string for both latitud and longitude fields must contain
     * a real number; otherwise an unpredictable behavior or runtime error
     * will be obtained
     * @pre The string for the OCCURRED_ON_DATE field must be in valid format
     * "year-month-day hour:minutes:seconds"; otherwise an unpredictable
     * behavior or runtime error will be obtained
     * @param line a string that contains a crime in csv format. Input parameter
     */
    void set(const std::string & line);

private:
    int _counter; ///< COUNTER: A counter (integer)

    /**
     * IDENTIFIER_NUMBER: Identifier of the crime. The string for this field
     * must contain at least one character different from whitespace and \t
     * character
     */
    std::string _id;

    std::string _code; ///< OFFENSE_CODE: A string with am internal code for the kind of crime
    std::string _group; ///< OFFENSE_CODE_GROUP: A string with a code for the group to which the crime
      belong
    std::string _description; ///< OFFENSE_DESCRIPTION: A string with a brief description of the crime
    std::string _district; ///< DISTRICT: the district where the crime took place
    std::string _areaReport; ///< REPORTING_AREA: the area where the crime took place
    bool _shooting; ///< SHOOTING: true if the crime was committed with a gunshot
    DateTime _dateTime; ///< OCCURRED_ON_DATE: date and time when the crime took place
    std::string _street; ///< STREET: name of the street where the crime took place
    Coordinates _location; ///< LOCATION: GPS coordinates where the crime took place

    /**
     * A const string that contains the value "UNKNOWN" that will be assigned
     * by default to string fields in a Crime object
     */
    static const std::string UNKNOWN_VALUE;

    /**
     * A const string with the default value for the DateTime of a Crime object
     * ("2017-01-20 02:00:00")
     */
    static const std::string DATETIME_DEFAULT;

    /**
     * A string that contains the default value for every field in a Crime object
     * The content is: "0," + UNKNOWN_VALUE + "," + UNKNOWN_VALUE + "," +
     * UNKNOWN_VALUE + "," + UNKNOWN_VALUE + "," + UNKNOWN_VALUE + "," + UNKNOWN_VALUE +
     *  "," + ((std::string)"0") + "," + DATETIME_DEFAULT + "," +
     * UNKNOWN_VALUE + ",181,181"
     */
    static const std::string CRIME_DEFAULT;
}; // end class Crime

/**
 * Removes spaces and \t characters at the beginning and at the end of the
 * provided string @p myString. If the provided string @p myString is empty
 * or contains only spaces or \t characters then @p myString will contain an
 * empty string after calling to this function.
 * @note This function can be easily implemented using the methods
 * find_first_not_of(string) and find_last_not_of(string) of class string.
 * @param myString a string. Input/Output parameter
 */
void Trim(std::string & myString);
```

```
/**
 * Converts to uppercase all the characters in the provided string @p myString
 * @param myString a string. Input/Output parameter
 */
void Capitalize(std::string & myString);

/**
 * Normalizes every string field in the provided Crime object. That is:
 * -# It removes spaces and \t characters at the beginning and at the end
 * of every string field
 * -# It converts to uppercase every string field except the fields for code and
 * area report.
 * For the ID field only the conversion to uppercase is necesary because an ID
 * cannot contains leading or trailing blanks
 */
void Normalize(Crime & crime);

#endif /* CRIME_H */
```

8.3. Crime.cpp

```
/*
 * Metodología de la Programación
 * Curso 2024/2025
 */

/**
 * @file Crime.cpp
 *
 * Last modified on February 12, 20245, 15:13PM
 */

#include "Crime.h"

using namespace std;

/*
 * Initialization of the static string that contains the value assigned to any
 * string field (string) which is found empty in the data of a Crime
 */
const std::string Crime::UNKNOWN_VALUE = "UNKNOWN";

/*
 * Initialization of the static string that contain the default value for the
 * DateTime field in a Crime
 */
const string Crime::DATETIME_DEFAULT = "2017-01-20 02:00:00";

/*
 * Initialization of the static string with the default values for every field in
 * a Crime
 */
const string Crime::CRIME_DEFAULT("0," + UNKNOWN_VALUE + "," + UNKNOWN_VALUE + "," +
        UNKNOWN_VALUE + "," + UNKNOWN_VALUE + "," + UNKNOWN_VALUE + "," + UNKNOWN_VALUE +
        "," + ((std::string)"1") + "," + Crime::DATETIME_DEFAULT + "," +
        UNKNOWN_VALUE + ",181,181");

Crime::Crime() {
}

Crime::Crime(const string &line) {
}

int Crime::getCounter() const {
}

std::string Crime::getId() const {
}

std::string Crime::getCode() const {
}

std::string Crime::getGroup() const {
}

std::string Crime::getDescription() const {
}

std::string Crime::getDistrict() const {
}

std::string Crime::getStreet() const {
}

std::string Crime::getAreaReport() const {
}

bool Crime::isShooting() const {
}

DateTime Crime::getDateTime() const {
}

Coordinates Crime::getLocation() const {
}

bool Crime::isIDUnknown() const {
}

std::string Crime::toString() const {
}

void Crime::setCounter(int c) {
}

void Crime::setId(const std::string &id) {
}

void Crime::setCode(const std::string &code) {
}

void Crime::setGroup(const std::string &group) {
}

void Crime::setDescription(const std::string &description) {
}

void Crime::setDistrict(const std::string &district) {
```

```
}

void Crime::setAreaReport(const std::string &areaReport) {
}

void Crime::setStreet(const std::string &street) {
}

void Crime::setShooting(bool shooting) {
}

void Crime::setDateTime(const string & time) {
}

void Crime::setLocation(const Coordinates & coordinates) {
}

void Crime::set(const std::string & line) {
    const int NFIELDS = 12;
    string data;        //piece of data detached from line;
    size_t pos, posn;   // aux for the beginning of the fields

    pos = 0;
    posn = line.find(',', pos); // beginning of the next field
    for (int nfield = 0; nfield < NFIELDS - 1 && posn != string::npos; nfield++) {
        data = line.substr(pos, posn - pos);
        pos = posn + 1;

        switch (nfield) {
            case 0: // counter
                setCounter(stoi(data));
                break;
            case 1: // ID
                setId(data);
                break;
            case 2: // code
                setCode(data);
                break;
            case 3: // group
                setGroup(data);
                break;
            case 4: // desc
                setDescription(data);
                break;
            case 5: // District alphaNumeric
                ...
        }
  ...
} //end of set()

void Trim(string & myString) {
}

void Capitalize(string & myString) {
}

void Normalize(Crime & crime) {
}
```

8.4. main.cpp

```
/*
 * Metodología de la Programación: Boston0
 * Curso 2024/2025
 */

/**
 * @file    main.cpp
 * @author estudiante1: apellidos*, nombre*
 * @author estudiante2: apellidos*, nombre* (solo si procede)
 */

#include <string>
#include <iostream>

#include "Coordinates.h"
#include "DateTime.h"
#include "Crime.h"

using namespace std;

/**
 * The purpose of this program is to read a set of data on crimes committed
 * in the city of Boston, showing those that have occurred within certain
 * geographic coordinates and within the date range provided to the program.
 * This program first reads from the standard input two Coordinates objects, one
 * Coordinates to define the minimun coordinates and another Coordinates to
 * define the maximum coordinates.
 * Then it reads two DateTime objects, one DateTime to define the minimun
 * DateTime and another DateTime to define the maximum DateTime.
 * Next it reads an integer number n to define the number of Crime objects to
 * read.
 * Thereafter, it reads n Crime objects, saving in an array of Crimes only the
 * Crimes whose DateTime is between the minimum and maximun DateTimes and whose
 * coordinates are valid (@see Coordinates::isValid() method description).
 * Each Crime object is normalized when inserted into the array.
 * Be careful that the number of crimes inserted in the array does not exceed
 * its capacity.
 * Finally, the program shows in the standard output all the Crimes in the array
 * and then all the Crimes which Coordinates are inside the area defined by
 * the minimum and maximum Coordinates provided to the program.
 * Be careful to show the output as in the below example.
 *
 * Running example:
 * > dist/Debug/GNU-Linux/boston0 < data/input00.b0in
Total number of crimes inserted in the array: 4
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON WARRANT,,,0,2022-02-02 00:00:00,WASHINGTON ST
      ,42.343082,-71.141724
1,222648862,3831,UNKNOWN,M/V - LEAVING SCENE - PROPERTY DAMAGE,B2,288,0,2022-02-05 18:25:00,WASHINGTON ST
      ,42.329750,-71.084541
4,222111641,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14 12:30:00,WASHINGTON ST,42.349056,-71.150497
5,222107076,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON WARRANT,D4,UNKNOWN,0,2022-03-11 10:45:00,
      MASSACHUSETTS AVE & ALBANY ST BOSTON  MA 02118 UNI,42.333500,-71.073509

Crimes within the given geographic area:
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON WARRANT,,,0,2022-02-02 00:00:00,WASHINGTON ST
      ,42.343082,-71.141724
4,222111641,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14 12:30:00,WASHINGTON ST,42.349056,-71.150497
5,222107076,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON WARRANT,D4,UNKNOWN,0,2022-03-11 10:45:00,
      MASSACHUSETTS AVE & ALBANY ST BOSTON  MA 02118 UNI,42.333500,-71.073509
 */
int main(int argc, char* argv[]) {
    const int DIM_ARRAY = 10; // dimension of the array arrayCrimes
    Crime arrayCrimes[DIM_ARRAY]; // array of Crimes

    // Read latitude and longitude for 2 Coordinates objects

    // Remember to take off the \n character after longitude of max Coordinates
    // Read 2 DateTime objects

    // Read the number of crimes
    // Remember to take off the \n character after number of crimes

    // Read the Crimes and put them in the array arrayCrimes
    //    whenever they follow the required restrictions and
    //    once normalized

    cout << "Total number of crimes inserted in the array: ";
    // display the number and the current content for the arrayCrimes

    cout << "\nCrimes within the given geographic area:" << endl;
    // display all the Crimes in the array which Coordinates are inside the given area
}
```

8.5. DateTime.h

```
/*
 * Metodología de la Programación
 * Curso 2024/2025
 */

/**
 * @file DateTime.h
 * @author Silvia Acid Carrillo <acid@decsai.ugr.es>
 * @author Andrés Cano Utrera <acu@decsai.ugr.es>
 * @author Luis Castillo Vidal <L.Castillo@decsai.ugr.es>
 *
 * Created on September 17, 2024, 5:43PM
 */

#ifndef DATETIME_H
#define DATETIME_H

#include <string>

/**
 * @class DateTime
 * @brief Used to represent dates and times in
 * the format YYYY-MM-dd hh:mm::ss, where
 *
 *  ----
 *       YYYY-MM-dd hh:mm:ss
 *       +----+----+----+----+
 *       |    |    |    |    |
 *       0    5    10   15   20
 *  ----
 *
 *    - YYYY Year     Any value allowed
 *    - MM Month      values in [1,12]
 *    - dd day        values depending on the calendar
 *    - hh hour       values in [0,23]
 *    - mm minute     values in [0,59]
 *    - ss seconds    values in [0,59]
 *
 *    The hour is in the UTC (Coordinated Universal Time) format.
 */
class DateTime {
private:
    int _year; ///< year
    int _month; ///< month (from 1 to 12)
    int _day; ///< day (from 1 to 31)
    int _hour; ///< hour (from 0 to 23)
    int _min; ///< (from 0 to 59)
    int _sec; ///< (from 0 to 59)

    /**
     *  Array with the names of days of the week in English
     */
    static const std::string DAY_NAMES [];

    void initDefault();

public:
    /**
     * String with the default value for a DateTime object
     * ("2000-01-01 00:00:00")
     */
    static const std::string DATETIME_STRING_DEFAULT;

    /**
     * @brief Base constructor. It sets the default date and time
     */
    DateTime();

    /**
     * @brief Constructor that builds a DataTime from the data in the
     * provided string. If the string contains a valid date
     * and time, it builds an object with the provided content; otherwise it
     * builds an object with the default DateTime. See
     * @ref DateTime::DATETIME_STRING_DEFAULT
     * @pre The provided string date must be in the correct format
     * (YYYY-MM-dd hh:mm:ss), that is no errors of conversion is obtained when
     * using stoi() for each field.
     * @param date The string containing date and time
     * Input parameter
     */
    DateTime(const std::string & date);

    /**
     * @brief It returns the year.
     * Query method
     * @return The year of the date as an integer value
     */
    int year() const;

    /**
     * @brief It returns the month.
     * Query method
     * @return The month of the date within its valid bounds
     */
    int month() const;

    /**
```

```
     * @brief It returns the day.
     * Query method
     * @return The day of the date within its valid bounds
     */
    int day() const;

    /**
     * @brief It returns the hour.
     * Query method
     * @return  The hour of the date within its valid bounds
     */
    int hour() const;

    /**
     * @brief It returns the minutes.
     * Query method
     * @return The number of minutes of the date within its valid bounds
     */
    int minutes() const;

    /**
     * @brief It returns the number of seconds.
     * Query method
     * @return The seconds of the date within its valid bounds
     */
    int seconds() const;

    /**
     * @brief Set this DataTime object with the data in the provided string \p line.
     * A correct format is as follows: (YYYY-MM-dd hh:mm:ss). If \p line is
     * not in the correct format this DataTime will be set with the data
     * in @ref DataTime::DATETIME_DEFAULT
     * Modifier method
     * @param line the string given from the csv format YYYY-MM-dd hh:mm::ss.
     * Input parameter
     */
    void set(const std::string &line);

    /**
     * @brief Transforms this DataTime object to a string using the
     * format: YYYY-MM-dd hh:mm::ss
     * Query method
     * @return string that contains the dateTime in the indicated format.
     */
    std::string toString() const;

    /**
     * @brief Computes the week day from this DateTime object.
     * The implementation is based on the Zeller's congruence; look at
     * wikipedia https://en.wikipedia.org/wiki/Zeller%27s_congruence
     * Query method.
     * @return values in the range 0..6, where 0 is Sunday, 1 is Monday ...
     */
    int weekDay() const;

    /**
     * Returns a string with the name of the provided day of the week (0 to 6).
     * - 0: SUNDAY
     * - 1: MONDAY
     * - ...
     * - 6: SATURDAY
     *
     * @param day selected day to return (0 to 6)
     * @throw std::invalid_argument Throws an std::invalid_argument exception
     * if day is not in the interval [0, 6]
     * @return a string with the name of the provided day of the week (0 to 6).
     */
    static const std::string& dayName(int day);
}; // end class DateTime

/**
 * @brief Auxiliar function to check if the provided date and time are in
 * correct format, taking into account, leap year, day in a month, etc.
 * @param year input
 * @param month input
 * @param day input
 * @param hour input
 * @param min input
 * @param sec input
 * @return true if is correct, else return false
 */
bool isCorrect(int year, int month, int day, int hour, int min, int sec);

/**
 * @brief Auxiliar function to obtain from the provided string @p line
 * the 6 components of a DateTime object.
 * Please consider using string::substr(int, int)[https://en.cppreference.com/w/cpp/string/basic_string/
     substr] to cut the line
 * into the appropriate substrings and then convert it into integer values with
 * the function stoi(string) [https://en.cppreference.com/w/cpp/string/basic_string/stol]
 * ~~~~
 *       YYYY-MM-dd hh:mm:ss
 *       +----+----+----+----+
 *       |    |    |    |    |
 *       0    5    10   15   20
 * ~~~~
 *
 * @param line input string
 * @param y output int
 * @param m output int
```

```
 * @param d output int
 * @param h output int
 * @param mn output int
 * @param s output int
 */
void split(const std::string& line, int &y, int & m, int & d, int & h, int &mn, int &s);

/**
 * @brief Overloading of the relational operator < for DateTime class
 * @param dateTime1 The first object to be compared. Input parameter
 * @param dateTime2 The second object to be compared. Input parameter
 * @return true if the DateTime of @p dateTime1 is smaller (before) than that of
 * @p dateTime2; false otherwise
 */
bool operator<(const DateTime& dateTime1, const DateTime& dateTime2);

/**
 * @brief Overloading of the operator > for DateTime class
 * @param dateTime1 a DateTime object. Input parameter
 * @param dateTime2 a DateTime object. Input parameter
 * @return true if dateTime1 > dateTime2; false otherwise
 */
bool operator>(const DateTime& dateTime1, const DateTime& dateTime2);

/**
 * @brief Overloading of the operator == for DateTime class
 * @param dateTime1 a DateTime object. Input parameter
 * @param dateTime2 a DateTime object. Input parameter
 * @return true if the two DateTimes are the same, that is same date and hour;
 * false otherwise
 */
bool operator==(const DateTime& dateTime1, const DateTime& dateTime2);

/**
 * @brief Overloading of the operator != for DateTime class
 * @param dateTime1 a DateTime object. Input parameter
 * @param dateTime2 a DateTime object. Input parameter
 * @return true if the two DateTimes are not equals (see operator==);
 * false otherwise
 */
bool operator!=(const DateTime& dateTime1, const DateTime& dateTime2);

/**
 * @brief Overloading of the operator <= for DateTime class
 * @param dateTime1 a DateTime object. Input parameter
 * @param dateTime2 a DateTime object. Input parameter
 * @return true if dateTime1 <= dateTime2; false otherwise
 */
bool operator<=(const DateTime& dateTime1, const DateTime& dateTime2);

/**
 * @brief Overloading of the operator >= for DateTime class
 * @param dateTime1 a DateTime object. Input parameter
 * @param dateTime2 a DateTime object. Input parameter
 * @return true if dateTime1 >= dateTime2; false otherwise
 */
bool operator>=(const DateTime& dateTime1, const DateTime& dateTime2);

#endif /* DATETIME_H */
```