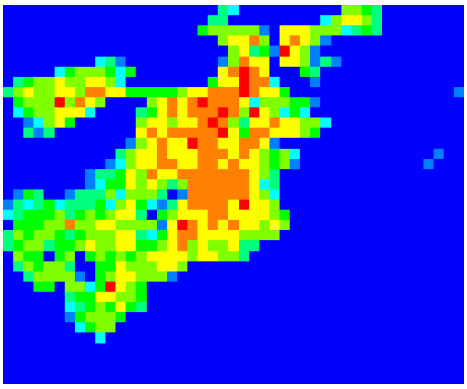
UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

# Metodología de la Programación

Curso 2024/2025



# Guion de prácticas

*Boston1*

*Módulo para arrays de objetos Crime*

*Marzo de 2025*

# Índice

# 1. Introducción

Como ya se indicó con anterioridad, las prácticas tienen por objeto realizar distintas aplicaciones que trabajan con datos sobre crímenes ocurridos en la ciudad de Boston. En esta práctica conservamos las clases que hemos desarrollado en la práctica anterior y añadiremos un nuevo módulo, el módulo `ArrayCrimesFunctions`, que no desarrolla una nueva clase, sino que proporciona un conjunto de funciones externas que operan sobre arrays de objetos de la clase `Crime`. Este conjunto de funciones externas nos servirán para practicar el paso de arrays como parámetro de una función. Este módulo desaparece en la siguiente práctica, pues será convertido en una clase, la clase `CrimeSet`.

En esta práctica vamos a desarrollar una aplicación que nos permita obtener algunas estadísticas acerca de un conjunto de crímenes cuyos datos serán leídos de la entrada estándar de forma similar a como hicimos en la práctica anterior.

# 2. Arquitectura de las prácticas

Como ya se indicó en la práctica anterior, las prácticas `Boston` se han diseñado por etapas; las primeras contienen estructuras más sencillas, sobre las cuales se asientan otras estructuras más complejas y se van completando con nuevas funcionalidades. En `Boston1` se hace un pequeño cambio en la clase `Crime` (se incorpora alguna funcionalidad adicional) y se introduce el nuevo módulo `ArrayCrimesFunctions`. Vea la figura 1.

Describamos brevemente los módulos que usaremos en esta práctica:

**A** DateTime
: La clase DateTime se proporcionó ya implementada con la práctica Boston0, y no necesita de ningún cambio en esta práctica.

**B** Coordinates
: La clase Coordinates no necesita de ningún cambio en esta práctica.

**C** Crime
: La clase Crime solo necesita añadir dos nuevos métodos (ver sección 4.4).

**D** ArrayCrimesFunctions
: Nuevo módulo que contendrá funciones externas que reciben como parámetro un array de objetos Crime, y cuya funcionalidad será trasladada posteriormente a la clase CrimeSet (ver sección 4.4).

**main.cpp** Contiene la función main() cuya finalidad será calcular algunas estadísticas a partir de un conjunto de crímenes.

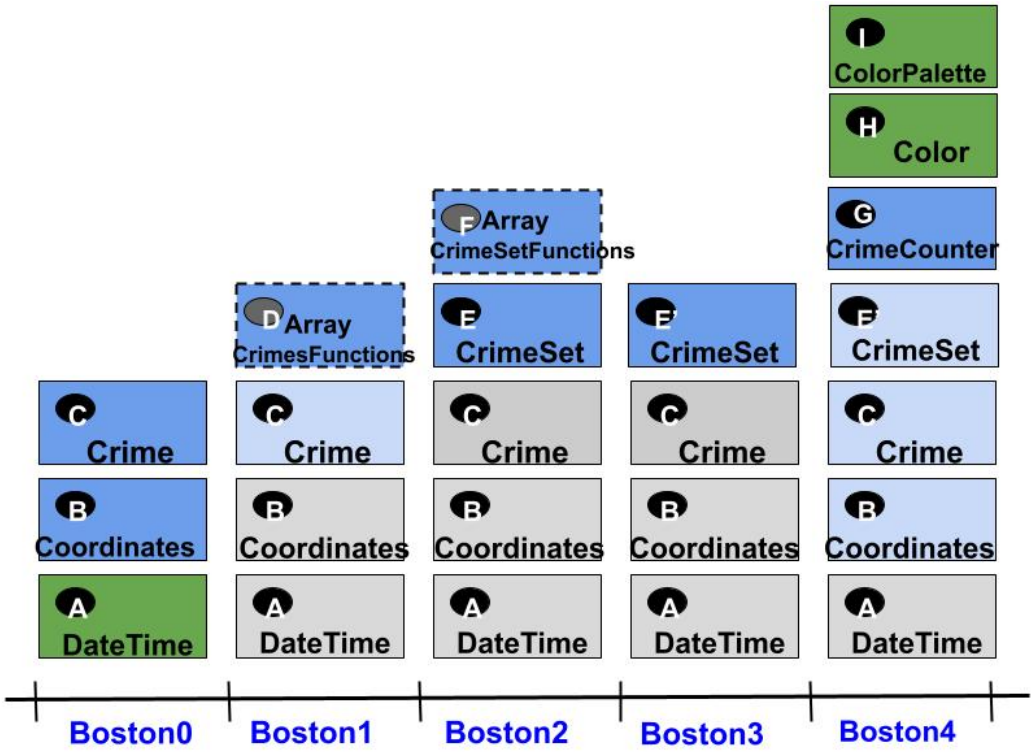



*Figura 1: Arquitectura de las prácticas de MP 2025. Como podemos observar, los bloques correspondientes a cada módulo, aparecen en distintos colores. El color verde indentifica software ya desarrollado (completamente terminado), mientras que el azúl indica software que requiere ser desarrollado por 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.*

# 3. Objetivos

El desarrollo de la práctica Boston1 persigue los siguientes objetivos:

- Comprender la importancia de la interfaz de una clase (ocultamiento de información y encapsulamiento).
- Seguir practicando con los **calificadores** public, private, const y static en métodos y datos miembro.
- Aprender a adecuar los pasos de parámetros a las especificaciones proporcionadas.
- Practicar el paso de arrays como parámetros de funciones.

# 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 en la salida estándar algunas estadísticas. Concretamente, mostrará el histograma por día de la semana y por hora del día. El histograma por día de la semana contiene el número total de crímenes (frecuencia) cometidos cada día de la semana. El histograma por hora del día contiene el número total de crímenes en cada hora del día. El programa debe mostrar también el día de la semana en que se comenten más crímenes (y cuantos se cometen en total en ese día) y también la hora del día en que se cometen más crímenes (y cuantos se cometen en total a esa hora). Finalmente el programa debe mostrar de forma ordenada la lista de aquellos crímenes en los que uno de los campos tenga el valor que se ha proporcionado previamente por la entrada estándar (filtrado del conjunto de crímenes por el valor de uno de los campos).

El programa llevará a cabo los siguientes pasos:

- En primer lugar, el programa debe leer de la entrada estándar un nombre y un valor para uno de los campos que tiene un crimen, que usaremos luego para hacer el filtrado del conjunto de crímenes.
- 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`, normalizando y almacenando cada uno en un array de objetos `Crime`, pero solo si tras la normalización el ID del crimen no es igual al valor desconocido (ver método `Crime::isIDUnknown()`) y el crimen no se encontraba ya (ver función externa `FindCrimeInArrayCrimes()`) en el array, ya que no permitimos que el array contenga dos crímenes con el mismo ID. Hay que tener cuidado de que el número de crímenes insertados en el array no exceda su capacidad. La capacidad del array proporcionada en main() con una constante, no está permitido modificarla.
- A continuación, el programa debe mostrar en la salida estándar todos los crímenes del array.
- Después, calculará y mostrará los histogramas por día de la semana y por hora del día.
- Luego, calculará y mostrará la frecuencia máxima acumulada en un día de la semana y en una hora del día.
- Finalmente, el programa calcula, ordena y muestra un nuevo array con aquellos crímenes en los que uno de los campos tenga el valor que se ha proporcionado previamente por la entrada estándar (filtrado del conjunto de crímenes por el valor de uno de los campos). Para

seleccionar los crímenes en los que un campo tenga un determinado valor puede ayudarse de la fucnión externa `SelectWhereEQArrayCrimes`, la cual se ayudará a su vez del nuevo método `Crime::getField(field)`.

**Ejemplo 1** *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/crimes6.b1in`*):*

```
Location 181.000000,181.000000
6
0,225520077,3126,unknown,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D14,786,0,2022-02-02 00:00:00,WASHINGTON ST,181.000000,181.000000
1,222111640,3831,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,0,2022-03-05 13:00:00,ALBANY
↪  ST,181.000000,181.000000
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, la primera línea contiene el nombre (*`Location`*) y valor* `181.000000,181.000000` *para uno de los campos de un crimen, que usaremos luego para hacer el filtrado. En la segunda línea, el número entero 6 indica que a continuación deben leerse 6 objetos* `Crime`*.*

## 4.2. Sintaxis y ejemplos de ejecución

De nuevo, 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/boston1 < <inputFile.b1in>
```

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

**Ejemplo 2** *Redireccionamiento desde el fichero* `data/crimes6.b1in`

```
linux> dist/Debug/GNU-Linux/boston1 < data/crimes6.b1in
```

*El contenido del fichero* `data/crimes6.b1in` *es el que aparece en el ejemplo 1.*

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

```
Records read: 6
6 crimes with valid and non-repeated ID:
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D14,786,0,2022-02-02 00:00:00,WASHINGTON ST,181.000000,181.000000
1,222111640,3831,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,0,2022-03-05 13:00:00,ALBANY
↪  ST,181.000000,181.000000
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

Histogram by day of the week:
SUNDAY 1
MONDAY 1
TUESDAY 0
WEDNESDAY 1
THURSDAY 0
FRIDAY 1
SATURDAY 2

Histogram by hour of the day:
0 2
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 1
11 0
12 1
13 1
14 0
15 0
16 0
17 0
18 1
19 0
20 0
21 0
22 0
23 0

Maximum cumulative number of crimes in a week day: 2
Day of maximum cumulative number of crimes in a week day: SATURDAY
Maximum cumulative number of crimes in a day hour: 2
Hour of maximum cumulative number of crimes in a day hour: 0

Sorted list of selected crimes where Location=181.000000,181.000000:

2,222201764,724,UNKNOWN,AUTO THEFT,C6,200,0,2022-01-09 00:00:00,W
↪  BROADWAY,181.000000,181.000000
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D14,786,0,2022-02-02 00:00:00,WASHINGTON ST,181.000000,181.000000
3,222201559,301,UNKNOWN,ROBBERY,D4,UNKNOWN,0,2022-03-05 13:00:00,ALBANY
↪  ST,181.000000,181.000000
```

*La salida anterior puede verse también en el fichero* `data/crimes6.b1out`.

En la carpeta `data` del proyecto Boston1, puede encontrar varios ficheros de texto con extensión `.b1in` con los que podrá hacer diferentes pruebas. Esta carpeta también contiene ficheros con extensión `.b1out`.

Cada uno de estos ficheros `.b1out` corresponde con el que debería obtenerse ejecutando el programa con el correspondiente fichero de entrada. Así por ejemplo, usando `data/crimes6.b1in` como entrada debería obtenerse el contenido del fichero `data/crimes6.b1out`.

Recuerde también que en cada práctica, 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.

## 4.3. Configurar el proyecto en Netbeans para esta práctica

Para la elaboración de la práctica `Boston1`, tiene disponible en el repositorio de `github`, una plantilla de proyecto `Netbeans`. Entre los ficheros proporcionados, puede encontrar:

- En la carpeta `include`: `ArrayCrimesFunctions.h`, `Coordinates.h`, `Crime.h`, `DateTime.h`.
- En la carpeta `src`: `ArrayCrimesFunctions.cpp`, `Coordinates.cpp`, `Crime.cpp`, `DateTime.cpp`, `main.cpp`.
- En la carpeta `doc`: `documentation.doxy`.
- En la carpeta `data`: una serie de ficheros con extensión `b1in` o bien `b1out` .
- En la carpeta `tests`: una serie de ficheros con extensión `.test`.
- En la carpeta `scripts`: una serie de scripts (ficheros con extensión `.sh`).

Para montar su propio proyecto `NetBeans` para la práctica Boston1, tiene tres opciones:

1. Hacer una copia de su proyecto `NetBeans` de la práctica anterior (`MiBoston0`), asignándole el nombre `MiBoston1`. Para ello realice los siguientes pasos:
   *a*) Desde el entorno de `NetBeans`, vista lógica, `Projects` → MiBoston0 → Copy. Asignar el nombre de proyecto `MiBoston1` a la copia.
   *b*) Añadir el nuevo módulo que aparece en esta práctica en su proyecto `MiBoston1`:
      - Módulo `ArrayCrimesFunctions` (`ArrayCrimesFunctions.h` y `ArrayCrimesFunctions.cpp`). Hay varias formas de hacerlo. Una de ellas es:

- Tomar los ficheros `ArrayCrimesFunctions.h` y `ArrayCrimesFunctions.cpp` proporcionados en el repositorio y copiarlos respectivamente a las carpetas `include` y `src` de su proyecto. Esto puede hacerlo con el explorador de archivos de Linux, desde un terminal de Linux, o incluso con `NetBeans` usando la vista `física`.
- Añadir `ArrayCrimesFunctions.h` y `ArrayCrimesFunctions.cpp` a la vista lógica del proyecto. Para ello desde el entorno de `NetBeans`, vista lógica, `Projects` → MiBoston1 → Add existing item → Header Files (o bien Source Files), y añadir cada uno de los ficheros anteriores a su lugar correspondiente.

*c*) `main()`: El contenido de `main()` debe ser sustituido por el contenido del `main()` proporcionado en el repositorio.

*d*) El contenido de la carpeta `tests` debe borrarse (en caso de que existiese) y copiar en ella el contenido de la carpeta `tests` del proyecto del repositorio.

*e*) Si no tenía la carpeta `scripts` en su proyecto copiar tal carpeta desde el proyecto del repositorio.

*f*) El fichero `documentation.doxy` debe ser sustituido por el proporcionado en el repositorio.

2. Hacer una copia de la carpeta del proyecto `Boston1` del repositorio en la carpeta donde tenga sus proyectos de la asignatura, poniendo el nombre `MiBoston1` a la copia.

   *a*) Desde el entorno de NetBeans, vista lógica, `Projects` → Boston1 → Copy. Elegir su carpeta de proyectos `NetBeans` y asignar el nombre de proyecto `MiBoston1` a la copia.

   *b*) En su nuevo proyecto `MiBoston1` copiar el código desarrollado en la práctica anterior (clase `Crime`) en el nuevo proyecto. O sea, la implementación de los métodos y funciones externas que hizo en `Crime.cpp`.

3. Montar el proyecto `MiBoston1` desde cero (ver sección: Configurar el proyecto en NetBeans, del guion Boston0).

   *a*) Construir el proyecto según se explicó en la práctica anterior (Boston0).

   *b*) En su nuevo proyecto `MiBoston1` copiar el código desarrollado en la práctica anterior (clase `Crime`) en el nuevo proyecto. O sea, la implementación de los métodos y funciones externas que hizo en `Crime.cpp`.

   *c*) Añadir los nuevos módulos de esta práctica de forma similar a cómo se explica con la opción 1.

   *d*) Incluir el contenido de las carpetas `tests` y `scripts` de forma similar a cómo se explica con la opción 1.

## 4.4. Módulos del proyecto

A la hora de implementar los nuevos métodos y funciones de esta práctica, fíjese en las cabeceras de tales métodos y funciones en los ficheros `*.h` correspondientes y en los comentarios que les acompañan, pues en ellos están detalladas sus especificaciones.

**Nota importante:** Se han retirado a propósito todos los `const` y `&` para los parámetros de los nuevos métodos y funciones externas que aparecen en los módulos `ArrayCrimesFunctions` y `Crime`. Debe revisar la forma en que se hace el paso de argumento para cada parámetro (por valor, por referencia o por referencia constante). Del mismo modo, a partir de esta práctica, los nuevos métodos que aparezcan en alguna clase [1] deben calificarse con `const` en caso necesario. Por tanto, revise todas las cabeceras de los nuevos métodos y funciones externas. El número de argumentos y los tipos han sido establecidos y **no se han de cambiar**. Se le invita a definir todos los métodos y funciones externas adicionales que estime oportuno para una adecuada modularización del código.

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

- Clase `Crime`. En esta clase solo tiene que añadir dos nuevos métodos (vea la sección 5.1). El resto de funciones y métodos de este módulo no necesita de ningún cambio por ahora. Los nuevos métodos son:
  - `normalize()`: como puede leer en su especificación, realizará la misma tarea que la función externa `Normalize()` pero sobre el objeto implícito, en lugar de sobre el objeto recibido como parámetro.
  - `getField(field)`: este método devuelve un string con el valor que tiene el crimen, para el campo proporcionado (Counter, ID, Code, etc).
- Módulo `ArrayCrimesFunctions`. Para lograr la operatividad que se va a necesitar en la función `main()` se van a definir una serie de funciones externas que realicen dichas tareas. La funciones a desarrollar en este módulo van a necesitar pasar el array de objetos de la clase Crime como parámetro, en sus diferentes modalidades de paso de parámetros, según que modifiquen o no los objetos de dicho array. En el fichero `ArrayCrimesFunctions.h`, se encuentran las especificaciones de cada función.

  Como podrá observar en el fichero `ArrayCrimesFunctions.h`, las distintas funciones a implementar reciben como parámetro el array de objetos Crime, y un entero que nos permite conocer el número de elementos utilizados en el array, o bien dos enteros que marcan el intervalo de posiciones utilizadas en el array.

[1] En esta práctica aparecen los nuevos métodos `normalize()` y `getField()` en la clase `Crime`

- 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 5.3), complete el código de este módulo.

## 4.5. Para la entrega

La práctica deberá ser entregada en Prado, en la fecha que se indica en cada 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, en esta ocasión es Boston1.zip, sin más aditivos.
Como alternativa, se sugiere utilizar el script `runZipProject.sh` que debe estar en la carpeta `scripts` de Boston1, junto con otras utilidades.

# 5. Código para la práctica

## 5.1. 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 (latitude, 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 latitude and longitude fields must contain
     * an integer 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);

    /**
     * @brief Gets a string with the value of the provided field.
     * Query method
     * @param field a string for selecting any of the field of a crime object.
     * Namely the possible labels used for the @p field parameter are:
     * Counter
     * ID
     * Code
     * Group
     * Description
     * District
     * Area
     * Street
     * Shooting
     * DateTime
     * Location
     *
     * The above labels correspond respectively to the record fields: <br>
     * _counter, _id, _code, _group, _description, _district, _areaReport, _street,
     * _shooting, _dateTime, _location.
     * 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 returned string will be obtained using the
     * std::to_string() function.
     * - In the case of Shooting the returned string will be "0" or "1".
     * - In the case of DateTime and Location the returned string will be
     * obtained with the toString() method of the corresponding class.
     * Input parameter
     * @throw invalid_argument exception if the label of the field is not valid
     * @return a string with the value of the selected field of the object.
     */
    std::string getField(std::string field);

    /**
     * @brief Normalizes every string field in this 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
     * For the ID field only the conversion to uppercase is necessary because an
     * ID cannot contains leading or trailing blanks
     * Modifier method
     */
    void normalize();

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
 * For the ID field only the conversion to uppercase is necessary because an
 * ID cannot contains leading or trailing blanks
 */
void Normalize(Crime & crime);
```

## 5.2. ArrayCrimesFunctions.h

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

/**
 * @file ArrayCrimesFunctions.h
 *
 * Created on September 19, 2024, 1:14PM
 */

#ifndef ARRAYCRIMESFUNCTIONS_H
#define ARRAYCRIMESFUNCTIONS_H

#include "Crime.h"

/**
 * @brief Initializes to 0 the elements of the provided array @p array
 * @param array Array of integers. Output parameter
 * @param size Size of the array @p array. Input parameter
 */
void InitializeArrayInts(int array[], int size);

/**
 * @brief Obtains the maximum value and its position (index) in the
 * provided array of integers @p array
 * @param array Array of integers. Input parameter
 * @param size The size of @p array. Input parameter
 * @param max The maximum value found in the vector. Output parameter
 * @param posMax The position in the array of the value with the maximun. Output parameter
 */
void ComputeMaxPosArrayInts(int array[], int size, int max, int posMax);

/**
 * @brief Displays the provided histogram (cumulative frequencies of crimes
 * by day of the week or hour of the day) on the standard output
 * @param dataField  Integer to select the type of histogram to display (0 or 1):
 * - value 0 when the histogram contains frequencies by days of the week. In this
 * case, 7 lines will be displayed, one for each day of the week; first line
 * for Sunday, second for Monday, and so on. Use method
 * DateTime::dayName(int day) to recover a string with the name of each
 * day of the week given an integer (0 for Sunday, 1 for Monday, ...). An
 * example of output when using dataField==0 is the following:
SUNDAY 25
MONDAY 11
TUESDAY 12
WEDNESDAY 5
THURSDAY 13
FRIDAY 17
SATURDAY 32
 * - value 1 when the histogram contains frequencies by hours of the day. In
 * this case, 24 lines will be displayed, one for each hour of the day;
 * first line for hour 0, 1 for hour 1, and so on. An example of output when
 * using dataField==1 is the following:
0 23
1 21
2 15
3 12
4 7
5 3
6 2
7 0
8 5
...
 * - Other values for dataField produce empty output
 * Input parameter
 * @param histogram histogram with cumulative frequencies by day of the week
 * or hour of the day. Input parameter
 */
void PrintHistogramArrayCrimes(int dataField, int histogram[]);

/**
 * @brief Displays the content of the provided array of Crimes on the standard
 * output. Each Crime is displayed in a new line with the help of the
 * Crime::toString() method
 * @param crimes array of Crimes. Input parameter
 * @param nCrimes number of crimes in the array. Input parameter
 */
void PrintArrayCrimes(Crime crimes[], int nCrimes);

/**
 * @brief Finds the position of the element with the minimum crime in the
 * subarray of @p array that begins at position @p initialPos and ends at position
 * @p finalPos (both included). The order
 * criterion among Crimes uses the dateTime and ID fields: crime1<crime2 if
 * crime1.dateTime < crime2.dateTime. If crime1.dateTime == crime2.dateTime
 * then the alphabetical order of the ID field is used.
 * @param array An array of Crime. Input parameter
 * @param initialPos The initial position of the subarray. Input parameter
 * @param finalPos The final position of the subarray. Input parameter
 * @return the position of the element with minimum crime in the subarray
 * that begins at position @p initialPos and ends at position @p finalPos
 * (both included). If no element can be retrieved, it returns -1
*/
int PosMinArrayCrimes(Crime array[], int initialPos, int finalPos);

/**
```

```
 * @brief Swaps the elements at positions @p first and @p second in the given
 * array of Crime
 * @param array An array of Crime. Input/output parameter
 * @param nElements The number of elements used by the array. Input parameter
 * @param first the position of the first element to be swaped. Input parameter
 * @param second the position of the second element to be swaped. Input parameter
 * @throw Throws a std::out_of_range exception if first or second are positions
 * out of the range of the given array
*/
void SwapElementsArrayCrimes(Crime array[], int nElements, int first, int second);

/**
 * @brief Sorts the given array of Crime in increasing order using a sorting
 * algorithm. The criterion order among Crimes is the same as the one
 * explained in the function PosMinArrayCrimes, that uses the dateTime and ID
 * fields of each Crime.
 *
 * @note It is not allowed to use any sorting algorithm available in any
 * C++ library.
 * @param array An array of Crime. Input/output parameter
 * @param nElements The number of elements used by the array. Input parameter
*/
void SortArrayCrimes(Crime array[], int nElements);

/**
 * @brief Searchs if the Id of the provided Crime is in the subarray of @p array
 * that begins at position @p initialPos and ends at position @p finalPos
 * (both included)
 * @param array An array of Crime objects. Input parameter
 * @param Crime The Crime to be found. Input parameter
 * @param initialPos The initial position of the subarray. Input parameter
 * @param finalPos The final position of the subarray. Input parameter
 * @return the position where the Id of the given Crime is in the subarray of
 * @p array that begins at position @p initialPos and ends at position @p finalPos
 * (both included). If the given Crime is not found, then -1 is returned.
 */
int FindCrimeInArrayCrimes(Crime array[], Crime crime,
        int initialPos, int finalPos);

/**
 * It takes the array of Crimes @p crimes as input and calculates the
 * cumulative frequency distribution of crimes (histogram) by days of the week
 * or by hours of the day. The histogram is saved in the array @p histogram.
 * The DateTime field of each Crime contains the information about when it
 * took place.
 * @pre the array @p histogram should have enough capacity to save the
 * calculated frequencies. Otherwise, possible runtime errors will be obtained
 * @throw std::out_of_range if @p dataField is neither 0 nor 1
 * @param crimes Input array of Crime objects. Input parameter
 * @param nCrimes The size of @p crimes. Input parameter
 * @param dataField Integer to select the type of calculation to be performed:
 * - value 0 to calculate by day of the week
 * - value 1 to calculate by hours of the day
 * Input parameter
 * @param histogram output array where the cumulative frequencies will be saved.
 * Input/output parameter
 */
void ComputeHistogramArrayCrimes(Crime crimes[], int nCrimes,
        int dataField, int histogram[]);

/**
 * @brief Saves in the array @p outputCrimes those crimes in the array
 * @p inputCrimes whose name of attribute is provided with @p field parameter
 * and whose value is equals to @p value parameter.
 * This function enables simple queries as: "Code" == "unknown" or "Code"== "3114"
 * as well as "Street"== "WASHINGTON ST" or "DateTime"==dd.to_string()
 * (being dd, a DateTime object),* but no other relational operators as
 * greater than etc.
 * @pre The array @p outputCrimes must have enough capacity to save all the
 * Crimes that verify the given condition.
 * @param inputCrimes Input array of crimes. Input parameter
 * @param inputCrimesSize size of the array @p inputCrimes. Input parameter
 * @param field The label of the selected attribute of the Crime class. See
 * the description of method Crime::getField(string) for possible values.
 * Input parameter
 * @param value The value of the selected attribute. Input parameter
 * @param outputCrimes Output array where the selected Crimes will be stored.
 * Output parameter
 * @param outputCrimesSize Size of the array @p outputCrimes. Output parameter
 */
void SelectWhereEQArrayCrimes(Crime inputCrimes[], int  inputCrimesSize,
        std::string  field, std::string value,
        Crime outputCrimes[], int outputCrimesSize);

#endif /* ARRAYCRIMESFUNCTIONS_H */
```

## 5.3. main.cpp

```
/*
 * Metodología de la Programación: Boston1
 * Curso 2024/2025
 */

/**
 * @file    main.cpp
 * @author estudiante1: apellidos*, nombre*
 * @author estudiante2: apellidos*, nombre* (solo si procede)
 *
 * Last modified on February 12, 20245, 20:13PM
 */

#include <string>
#include <iostream>

#include "DateTime.h"
#include "Crime.h"
#include "ArrayCrimesFunctions.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 some statistics about them.
 * This program first reads from the standard input a name and a value for one of
 * the fields of a Crime (this will be used later to select the crimes with
 * that field equals to the provided value).
 * Then it reads an integer number n to define the number of Crime objects to
 * read.
 * Thereafter, it reads n Crime objects, normalizing and storing each one
 * in the array inputCrimes only if the ID of the normalized crime is not
 * unknown and there is not already a crime in the array inputCrimes with the
 * same ID (the array cannot contains two crimes with the same ID).
 * Be careful that the number of crimes inserted in the array does not exceed
 * its capacity.
 * Next, the program shows in the standard output all the Crimes in the array
 * Hereafter, it computes and shows the histograms by day of the week and
 * by hour of the day.
 * Subsequently, it computes and shows the maximum cumulative number of crimes
 * in a week day and the maximum cumulative number of crimes in a  day hour.
 * Finally the program computes, sorts and shows an array with those Crimes
 * that verify the condition that the previously provided field is equals to
 * the given value.
 * Be careful to show the output as in the below example.
 *
 * Running example:
 * > dist/Debug/GNU-Linux/boston1 < data/crimes22.b1in
Records read: 22
20 crimes with valid and non-repeated ID:
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON WARRANT,D14,786,0,2022-02-02 00:00:00,WASHINGTON
      ST,42.343082,-71.141724
2,222201764,724,UNKNOWN,AUTO THEFT,C6,200,0,2022-01-09 00:00:00,W BROADWAY,42.341286,-71.054680
3,222201559,301,UNKNOWN,ROBBERY,D4,UNKNOWN,0,2022-03-05 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
6,22209348,801,UNKNOWN,ASSAULT - SIMPLE,C6,235,0,2022-02-08 22:37:00,DORCHESTER AVE,42.321373,-71.056793
7,222073971,611,UNKNOWN,LARCENY PICK-POCKET,A1,77,0,2022-02-27 10:07:00,NEW SUDBURY ST,42.361839,-71.059769
8,222072974,619,UNKNOWN,LARCENY ALL OTHERS,C6,186,0,2022-01-14 15:20:00,ALLSTATE ROAD,181.000000,181.000000
9,222055138,3201,UNKNOWN,PROPERTY - LOST/ MISSING,E13,574,0,2022-01-23 13:15:00,WASHINGTON ST
      ,42.309719,-71.104294
10,222045598,2610,UNKNOWN,TRESPASSING,B2,280,0,2022-01-21 12:30:00,DUDLEY ST,42.328728,-71.083824
11,222040102,3410,UNKNOWN,TOWED MOTOR VEHICLE,E13,569,0,2022-02-11 00:00:00,STEDMAN ST,42.304298,-71.107262
12,22203312,540,UNKNOWN,BURGLARY - COMMERICAL,A1,UNKNOWN,0,2022-01-16 01:40:00,LINCOLN STREET
      ,181.000000,181.000000
13,222027001,2670,UNKNOWN,HARASSMENT/ CRIMINAL HARASSMENT,E5,559,0,2022-02-20 04:00:00,WASHINGTON ST
      ,42.284718,-71.129990
14,222021808,3114,UNKNOWN,INVESTIGATE PROPERTY,D4,624,0,2022-03-29 00:00:00,BOYLSTON ST
      ,181.000000,181.000000
15,222021807,3115,UNKNOWN,INVESTIGATE PERSON,D4,UNKNOWN,0,2022-03-29 03:27:00,W SPRINGFIELD ST
      ,42.339302,-71.079124
16,222021806,3115,UNKNOWN,INVESTIGATE PERSON,B3,444,0,2022-03-29 01:35:00,BLUE HILL AVE,42.295765,-71.087563
17,222021803,1402,UNKNOWN,VANDALISM,B2,UNKNOWN,0,2022-03-29 02:00:00,NORFOLK AVE,42.327137,-71.072182
18,222021802,3006,UNKNOWN,SICK/INJURED/MEDICAL - PERSON,C11,UNKNOWN,0,2022-03-29 02:03:00,DORCHESTER AVE
      ,42.313225,-71.057152
19,222021800,3108,UNKNOWN,FIRE REPORT,A7,UNKNOWN,0,2022-03-29 00:36:00,MERIDIAN ST,42.379017,-71.039207
21,222111640,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14 12:30:00,WASHINGTON ST,42.349056,-71.150497

Histogram by day of the week:
SUNDAY 5
MONDAY 2
TUESDAY 7
WEDNESDAY 1
THURSDAY 0
FRIDAY 4
SATURDAY 1

Histogram by hour of the day:
0 5
1 2
2 2
3 1
4 1
5 0
6 0
7 0
```

```
8 0
9 0
10 2
11 0
12 3
13 2
14 0
15 1
16 0
17 0
18 0
19 0
20 0
21 0
22 1
23 0

Maximum cumulative number of crimes in a week day: 7
Day of maximum cumulative number of crimes in a week day: TUESDAY
Maximum cumulative number of crimes in a day hour: 5
Hour of maximum cumulative number of crimes in a day hour: 0

Sorted list of selected crimes where Code=619:
8,222072974,619,UNKNOWN,LARCENY ALL OTHERS,C6,186,0,2022-01-14 15:20:00,ALLSTATE ROAD,181.000000,181.000000
21,222111640,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14 12:30:00,WASHINGTON ST,42.349056,-71.150497
4,222111641,619,UNKNOWN,LARCENY ALL OTHERS,D14,778,0,2022-02-14 12:30:00,WASHINGTON ST,42.349056,-71.150497
 */
int main(int argc, char* argv[]) {
    const int DIM_ARRAY = 200; // capacity of the two arrays of crimes
    Crime inputCrimes[DIM_ARRAY], // array of crimes read from the standard input
        selectedCrimes[DIM_ARRAY]; // array of crimes that verify a given condition

    int frequencyByDay[7]; // array to save the cumulative number of crimes (histogram) for every week day
    int frequencyByHour[24]; // array to save the cumulative number of crimes (histogram) for every hour

    // Read a name and a value for one of the fields of a Crime
    // Remember to take off the character \n at the end of previous value
    // Read number n to define the number of Crime objects
    // Remember to take off the character \n after previous number

    // Loop to read n Crime objects
        // Read a Crime object, normalize and insert it in the array
        //      if its ID is not unknown and there is not any crime
        //      in the array with identical ID


    cout << "Records read: " << ... << endl;
    cout << ... << " crimes with valid and non-repeated ID: " << endl;
    // Show all the Crimes in the array

    // Compute the histograms by day of the week and
    //    by hour of the day.

    // Show the histograms by day of the week and
    //    by hour of the day.
    cout << "\nHistogram by day of the week:" << endl;

    cout << "\nHistogram by hour of the day:" << endl;

    // Compute the maximum cumulative number of crimes in a week day
    //    and the maximum cumulative number of crimes in a day hour.

    // Show the maximum cumulative number of crimes in a week day
    //    and the maximum cumulative number of crimes in a  day hour.
    cout << "\nMaximum cumulative number of crimes in a week day: " << ...
         << endl;
    cout << "Day of maximum cumulative number of crimes in a week day: " <<
            ... << endl;
    cout << "Maximum cumulative number of crimes in a day hour: " << ...
         << endl;
    cout << "Hour of maximum cumulative number of crimes in a day hour: " <<
            ... << endl;

    // Compute and sort an array with those Crimes
    //    that verify the condition that the read field is equals to
    //    the given value.
    cout << "\nSorted list of selected crimes where " << .. << "=" <<
            .. << ":" << endl<<endl;

    // Show the array content
}
```