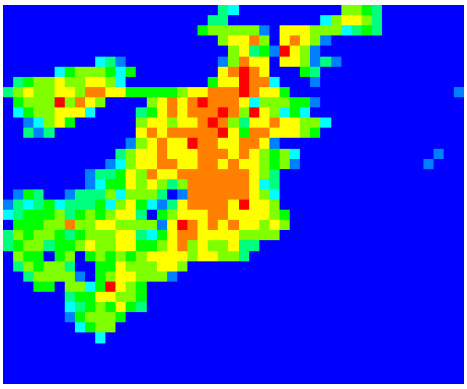
UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

Curso 2024/2025



Guion de prácticas

Boston2

Clase CrimeSet con vector estático. Módulo para arrays de objetos CrimeSet

Marzo de 2025

Índice

1. Introducción

En esta práctica conservamos las clases que hemos desarrollado en la práctica anterior, descartando el módulo `ArrayCrimesFunctions`, el cual es reconvertido en una nueva clase de funcionalidad parecida, la clase `CrimeSet`. Añadiremos además un nuevo módulo, el módulo `ArrayCrimeSetFunctions`, que no desarrolla una nueva clase, sino que proporciona un conjunto de funciones externas que operan sobre arrays en memoria dinámica de objetos de la clase `CrimeSet`. Este conjunto de funciones externas nos servirán para practicar el paso de punteros como parámetro de una función. Este módulo desaparece en la práctica Boston3, pues ya no se necesita.

En esta práctica vamos a desarrollar una aplicación que nos permita obtener la fusión de una lista de ficheros `.crm`. Cada uno de estos ficheros con un determinado formato contiene información acerca de un conjunto de crímenes.

Por otro lado, a partir de ahora, vamos a cambiar la forma de ejecución de nuestros programas. Se dejan de realizar lecturas desde teclado (o su equivalencia mediante redireccionamiento). Las lecturas de datos se van a realizar directamente desde ficheros, cuyos nombres, junto con argumentos adicionales, se van a especificar desde la línea de comandos, esto es, a través de los parámetros con los que ejecutamos nuestro programa. También, nuestro programa escribirá datos en ficheros de salida en lugar de en la salida estándar.

2. Arquitectura de las prácticas

En `Boston2` desaparece el módulo `ArrayCrimesFunctions`, y se introduce la clase `CrimeSet` y el nuevo módulo `ArrayCrimeSetFunctions`. Vea la figura 1. Las clase `Crime` y `Coordinates` no necesitan en principio ningún cambio.

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 no necesita de ningún cambio en esta práctica.

E CrimeSet
: Nueva clase que permite guardar un conjunto de objetos `Crime` (ver sección 5.3).

F ArrayCrimeSetFunctions
: Nuevo módulo que contendrá funciones externas que reciben como

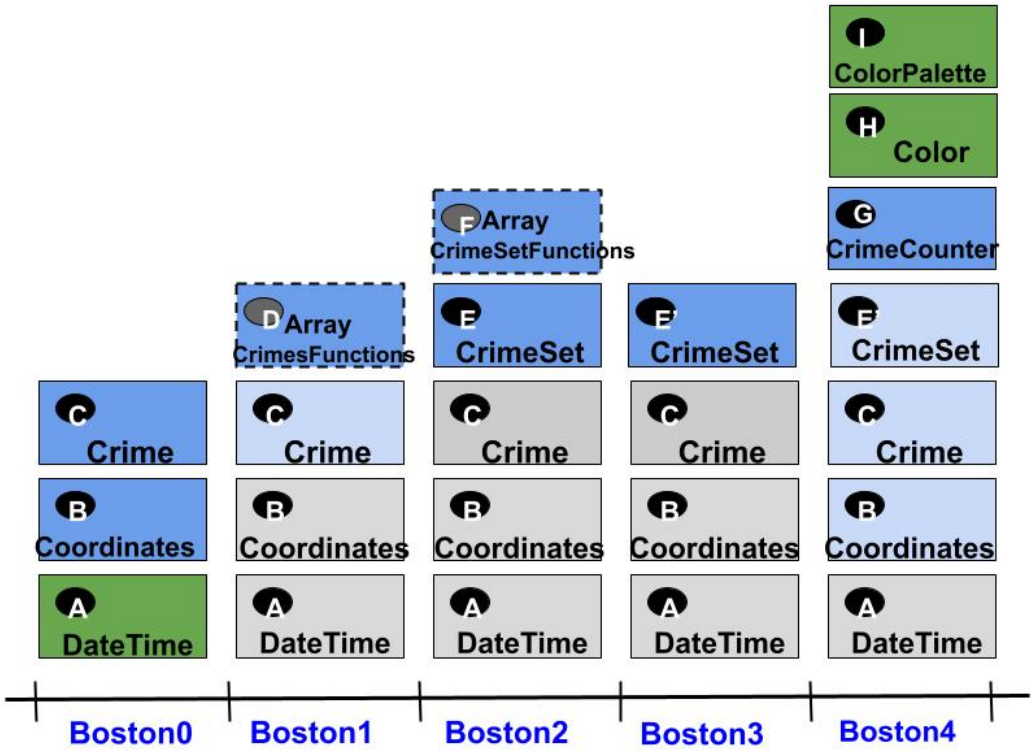


Figura 1: Arquitectura de las prácticas de MP 2025. Como podemos observar, los bloques correspondientes a cada módulo, aparecen en distintos colores. El color verde indentifica software ya desarrollado (completamente terminado), mientras que el azúl indica software que requiere ser desarrollado por el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

parámetro un array dinámico de objetos de la clase `CrimeSet` (ver sección 5.3).

main.cpp Contiene la función main() cuya finalidad será la de obtener la fusión de una lista de ficheros `crm`.

3. Objetivos

El desarrollo de la práctica Boston2 persigue los siguientes objetivos:

- Practicar con una clase compuesta de un array de objetos y otros datos: la clase `CrimeSet`.
- Practicar con la devolución de objetos por valor y por referencia.
- Conocer cómo se pasan punteros como parámetros de funciones.
- Devolución de punteros por funciones.

- Uso de memoria dinámica en C++.
- Trabajar con arrays dinámicos.
- Aprender a escribir datos en un fichero de texto con el operador <<.
- Aprender a leer datos de un fichero de texto mediante el operador >> y la función `getline()`.
- Conocer cómo leer los parámetros pasados a la función `main()` desde la línea de comandos.
- Practicar con herramientas como el depurador de NetBeans y `valgrind` para rastrear errores en tiempo de ejecución.

4. Ficheros crm

En esta práctica los datos sobre los crímenes se van a leer/escribir en ficheros de texto en formato `crm`. Un fichero `crm` es un fichero de texto que contiene información acerca de un conjunto de crímenes. El fichero `crm` tiene el siguiente formato:

- La primera línea contiene la cadena mágica “MP-CRIME-T-1.0”.
- A continuación, de forma opcional, pueden aparecer una o varias líneas de comentarios. Cada una de ellas se identifica porque tiene el carácter '`#`' al principio de la línea.
- En una nueva línea aparece un número entero n que indica el número de crímenes que vienen a continuación.
- Las siguientes líneas contienen la información de cada uno de los n crímenes. Cada línea contiene la información de un crimen. Los campos de cada crimen están separados con el carácter '`,`'. Es posible que aparezcan espacios en blanco antes o después del valor de cada campo, de la misma forma que se encontraban en Boston0.

Nota: Un fichero `crm` suele estar ordenado según el valor del campo `COUNTER` pero no es obligatorio, y podría estar ordenado de otra forma distinta. Lo que sí es obligatorio, es que en un fichero `crm` no puede tener dos crímenes con idéntico ID.

Ejemplo 1 *Ejemplo de fichero crm con información de 5 crímenes: DataSets/crimes_05.crm*

```
MP-CRIME-T-1.0
#data imported from csv_2017_2022_repaired.csv
5
1,222648862,3831,unknown,M/V - LEAVING SCENE - PROPERTY
↪  DAMAGE,B2,288,1,2022-02-05 00:00:00,WASHINGTON ST,42.329750,-71.084541
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,1,2022-03-05 13:00:00,ALBANY
↪  ST,42.333183,-71.073936
```

```
5,222107076,3126,unknown,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D4,unknown,1,2022-03-11 10:45:00,MASSACHUSETTS AVE & ALBANY ST
↪  BOSTON  MA 02118 UNI,42.333500,-71.073509
7,222073971,611,unknown,LARCENY PICK-POCKET,A1,77,1,2022-02-01 10:07:00,NEW
↪  SUDBURY ST,42.361839,-71.059769
```

5. Práctica a entregar

5.1. Finalidad del programa

El programa tiene por objeto leer una lista de ficheros `crm` (vea en la sección 4 la descripción del formato de estos ficheros), cada uno de los cuales contiene información sobre un conjunto de crímenes, y hacer la fusión de todos ellos. Cada fichero `crm` leído da lugar a un objeto de la clase `CrimeSet`. La fusión de un objeto `CrimeSet` con otro segundo objeto, consiste en añadir al primero todos los objetos `Crime` del segundo que no están en el primero. Esta tarea se puede hacer con la ayuda del método `CrimeSet::join()`.

El resultado de la fusión de todos los ficheros `crm` se grabará en un nuevo fichero `crm` de salida. Los nombres de los ficheros `crm` de entrada se deben leer de un fichero de texto (ver ejemplo 2), cuyo nombre se proporciona al programa como primer argumento. Este fichero de texto contiene en cada línea el nombre de uno de los ficheros `crm` de entrada. El nombre del fichero `crm` de salida puede proporcionarse al programa de forma opcional como segundo argumento. Si no se proporciona, el nombre del fichero de salida ha de ser `tests/output/output.crm`.

El programa lleva a cabo los siguientes pasos:

- El programa comienza comprobando si el número de argumentos es el adecuado (se necesita especificar al menos el nombre del fichero de texto que contiene la lista de ficheros `crm` de entrada).
- A continuación lee y almacena en un array dinámico de objetos `CrimeSet` procedentes de cada uno de los ficheros `crm` de entrada.
- Seguidamente, para cada uno de los objetos `CrimeSet` del array dinámico, se llevan a cabo los siguientes pasos:
  - Normalizar cada uno de los objetos `Crime` del `CrimeSet`. Para hacer esto puede ayudarse del método `CrimeSet::normalize()`.
  - Seleccionar aquellos objetos `Crime` con una localización válida, descartando los que no. Para seleccionar los objetos `Crime` con localización válida, puede ayudarse del método `CrimeSet::selectValidLocation()`.
  - Seleccionar aquellos objetos `Crime` en los que el campo `Shooting` sea igual al valor 1 (`true`), descartando los que no. Para hacer esto, puede ayudarse del método `CrimeSet::selectWhereEQ()`.

- A continuación, se obtendrá un objeto `CrimeSet` resultado que contenga la fusión (*join*) de todos los `CrimeSet` del array dinámico.
- Se tendrá que incluir en el `CrimeSet` resultado (usar el método `CrimeSet::setComment(comment)`) un comentario que indique la procedencia de tal fichero `crm`. Tenga en cuenta que la cadena `comment` no debe incluir el carácter '`#`', pues tal carácter lo incluye automáticamente al principio de cada línea de comentarios del fichero de salida el método `CrimeSet::saveComments()`, método que debería usarse desde `CrimeSet::save()`. Por ejemplo, en el fichero de salida deberíamos obtener lo siguiente si este se obtuvo a partir del fichero `data/input06.b2in`:

```
#Fusion of the crm files whose names are in the file: data/input06.b2in
```

  Vea los ejemplos de ejecución de la sección 5.2.
- Finalmente, el `CrimeSet` resultado debe ser ordenado y grabado en el fichero `crm` de salida. Para la ordenación se ayudará del método `CrimeSet::sort()` que ordena los objetos `Crime` del `CrimeSet` por instante de tiempo y a igualdad de tiempo entre dos objetos `Crime`, se tiene en cuenta el orden alfabético de sus IDs.

5.2. Sintaxis y ejemplos de ejecución

Como se indicó en la Introducción, en esta práctica cambiamos la forma de ejecución de nuestros programas. En esta práctica usaremos los parámetros de la línea de comandos, para pasarle al programa unos valores que pueden hacer cambiar la ejecución de un programa. En esta ocasión se gestiona un parámetro obligatorio y otro opcional (2 parámetros como mucho), que se corresponden respectivamente con el nombre de un fichero de entrada y el nombre de un fichero de salida. El fichero de entrada es un fichero de texto que contiene en cada línea el nombre de un fichero `crm`. El fichero de salida es en el que se guardará el resultado de la fusión.

Ejemplo 2 *Un ejemplo de fichero de entrada al programa es el siguiente (el contenido también puede verse en el fichero* `data/input06.b2in`*):*

```
../DataSets/crimes_01.crm
../DataSets/crimes_05.crm
```

Como puede ver, el fichero contiene dos líneas, cada una con el nombre de un fichero crm.

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

```
linux> dist/Debug/GNU-Linux/boston2 <inputFile.txt> [<outputFile.crm>]
```

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

Ejemplo 3 *Faltan argumentos para la ejecución del programa:*

```
linux> dist/Debug/GNU-Linux/boston2
```

La salida del programa será la que genera la función `showHelp()` *incluida en* `main.cpp`*:*

```
ERROR in boston2 parameters
Run with the following arguments:
boston2 <inputFile.txt> [<outputFile.crm>]

Parameters:
<inputFile.txt>: name of the input text file that contains the names of the
↪  crm files
<outputFile.crm>: name of the output crm file (tests/output/output.crm by
↪  default)
```

Ejemplo 4 *Ejecución proporcionando solo el fichero de entrada:*

```
linux> dist/Debug/GNU-Linux/boston2 data/input06.b2in
```

El contenido del fichero `data/input06.b2in` *es el que aparece en el ejemplo 2. Como puede ver, el objetivo de la anterior ejecución es obtener la fusión de los ficheros crm* `crimes_01.crm` *y* `crimes_05.crm` *cuyos contenidos son los siguientes:*

Fichero crimes_01.crm:

```
MP-CRIME-T-1.0
#data imported from csv_2017_2022_repaired.csv
# line 2
# line 3
1
0,225520077,3126,unknown,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D14,786,1,2022-02-05 00:00:00,WASHINGTON ST,42.343082,-71.141724
```

Fichero crimes_05.crm:

```
MP-CRIME-T-1.0
#data imported from csv_2017_2022_repaired.csv
5
1,222648862,3831,unknown,M/V - LEAVING SCENE - PROPERTY
↪  DAMAGE,B2,288,1,2022-02-05 00:00:00,WASHINGTON ST,42.329750,-71.084541
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,1,2022-03-05 13:00:00,ALBANY
↪  ST,42.333183,-71.073936
5,222107076,3126,unknown,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D4,unknown,1,2022-03-11 10:45:00,MASSACHUSETTS AVE & ALBANY ST
↪  BOSTON  MA 02118 UNI,42.333500,-71.073509
7,222073971,611,unknown,LARCENY PICK-POCKET,A1,77,1,2022-02-01 10:07:00,NEW
↪  SUDBURY ST,42.361839,-71.059769
```

La ejecución anterior del programa debe generar el fichero de salida `tests/output/output.crm` *cuyo contenido debería ser el siguiente:*

```
MP-CRIME-T-1.0
#Fusion of the crm files whose names are in the file: data/input06.b2in
5
7,222073971,611,UNKNOWN,LARCENY PICK-POCKET,A1,77,1,2022-02-01 10:07:00,NEW
↪  SUDBURY ST,42.361839,-71.059769
```

```
1,222648862,3831,UNKNOWN,M/V - LEAVING SCENE - PROPERTY
↪  DAMAGE,B2,288,1,2022-02-05 00:00:00,WASHINGTON ST,42.329750,-71.084541
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D14,786,1,2022-02-05 00:00:00,WASHINGTON ST,42.343082,-71.141724
3,222201559,301,UNKNOWN,ROBBERY,D4,UNKNOWN,1,2022-03-05 13:00:00,ALBANY
↪  ST,42.333183,-71.073936
5,222107076,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON
↪  WARRANT,D4,UNKNOWN,1,2022-03-11 10:45:00,MASSACHUSETTS AVE & ALBANY ST
↪  BOSTON  MA 02118 UNI,42.333500,-71.073509
```

El contenido del anterior fichero crm de salida coincide con el contenido del fichero `data/outcome06.crm` *que puede encontrar en el repositorio de la asignatura.*

Ejemplo 5 *Ejecución proporcionando el fichero de entrada y el de salida:*

```
linux> dist/Debug/GNU-Linux/boston2 data/input06.b2in tests/output/output.crm
```

El contenido del fichero `data/input06.b2in` *es el mismo que el usado en el ejemplo anterior (puede ver su contenido en ejemplo 2).*

La ejecución anterior del programa debe generar el fichero de salida `tests/output/output.crm` *cuyo contenido es el mismo que el que se obtendría con la ejecución del ejemplo 4.*

En la carpeta `data` del proyecto Boston2, puede encontrar varios ficheros de texto con extensión `.b2in` con los que podrá hacer diferentes validaciones. Esta carpeta también contiene ficheros con extensión `.crm` y cuyo nombre es `outcome`+nº. Cada uno de estos ficheros `.crm` corresponde con el que debería obtenerse ejecutando el programa con el correspondiente fichero de entrada. Así por ejemplo, usando `data/input06.b2in` como entrada debería obtenerse `outcome06.crm` como salida.

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.

5.3. 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: Se han retirado a propósito todos los `const` y `&` para los parámetros de los métodos y funciones de los nuevos módulos `CrimeSet` y `ArrayCrimeSetFunctions`. 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, revise los métodos de la clase `CrimeSet` para ver si es necesario calificarlos con `const`. Por tanto, revise todas las cabeceras de 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 todas la(s) función(es) externa(s) adicional(es) que estime oportuno para una adecuada modularización del código.

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

- Clase `CrimeSet`. El fichero **CrimeSet.h** (ver detalles en sección 6.1) contiene la declaración de la clase `CrimeSet`. La clase `CrimeSet` permite guardar un conjunto de objetos de la clase `Crime`. En esta práctica la clase usa un array alojado en memoria automática (en la pila) de capacidad fija. En la práctica Boston3, modificaremos esta clase para que use un array alojado en memoria dinámica. Como puede ver, el dato miembro `_crimes` es el array de objetos `Crime`, que tiene una capacidad máxima de 2000 objetos. El dato miembro `_nCrimes` permite conocer en todo momento el número de elementos usados en el array.

```
class CrimeSet {
...
private:
    static const int DIM_VECTOR_CRIMES = 2000; ///< The capacity of the array _crimes
    static const std::string MAGIC_STRING_T; ///< A const string with the magic string for text
      files

    /**
     * string that contains several lines of comments (text in natural
     * language). Each line, except possibly the last one, ends with the
     * character '\n'
     */
    std::string _comment;

    Crime _crimes[DIM_VECTOR_CRIMES]; ///< Array of crimes
    int _nCrimes; ///< Number of used elements in _crimes
}; // end class CrimeSet
```

  Como puede ver más arriba, un objeto `CrimeSet` contiene también el dato miembro `_comment` que sirve para guardar cadenas con algún texto o comentario que permita indicar por ejemplo cual es la procedencia de los crímenes, o cualquier otra información.

  La constante string `MAGIC_STRING_T`, inicializada al valor `"MP-CRIME-T-1.0"` en el fichero `CrimeSet.cpp`, contiene la cadena de caracteres que aparece al principio de los ficheros `crm`.

  Un objeto `CrimeSet` no puede contener dos objetos `Crime` con idéntico ID. Ningún método público de la clase `CrimeSet` permite agregar un objeto `Crime` al `CrimeSet` si su ID es igual al de otro objeto ya incluido en el `CrimeSet`.

- Módulo `ArrayCrimeSetFunctions`. 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. Estas funciones trabajan con arrays dinámicos de objetos de la clase `CrimeSet`. En este módulo, la capacidad de los arrays dinámicos coincide con el número de elementos utilizados. Las funciones de este módulo van a necesitar pasar el array de objetos de la clase `CrimeSet` como parámetro, en sus diferentes modalidades de paso de parámetros, según que modifiquen o no los objetos de dicho array. En el fichero `ArrayCrimeSetFunctions.h`, se encuentran las especificaciones detalladas de cada función, pero un resumen sería el siguiente:

- `PrintArrayCrimeSet()`: permite mostrar el contenido de los objetos `CrimeSet` del array dinámico en la salida estándar.
- `ReadArrayCrimeSet()`: permite leer de un flujo de entrada un objeto `CrimeSet` y añadirlo al final del array dinámico. Esta función debe realojar el array dinámico cada vez que es llamada, para aumentar su capacidad en uno, y que así puede añadirse el nuevo objeto `CrimeSet`.
- `AllocateArrayCrimeSet()`: permite reservar memoria dinámica para el array dinámico con una determinada capacidad.
- `DeallocateArrayCrimeSet()`: permite liberar la memoria dinámica ocupada por el array dinámico.
- `AppendCrimeArrayCrimeSet()`: permite realojar el array dinámico, aumentando su capacidad en uno, y añadir un nuevo objeto `CrimeSet` al final.

Como podrá observar en el fichero `ArrayCrimeSetFunctions.h` (ver sección 6.3), las distintas funciones a implementar reciben como parámetro entre otros, un puntero a un objeto `CrimeSet`, que es el puntero al array dinámico, y un entero que nos permite conocer el número de elementos utilizados (que coincide con su capacidad) en el array.

- Módulo `main.cpp`. Este programa tiene por finalidad lo descrito en la sección 5.1. Teniendo en cuenta los detalles comentados en tal sección y los pasos esbozados en el código proporcionado para esta función (sección 6.4), complete el código de este módulo.

5.4. 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 Boston2.zip, sin más aditivos.
Como alternativa, se sugiere utilizar el script `runZipProject.sh` que debe estar en la carpeta `scripts` de Boston2, junto con otras utilidades.

6. Código para la práctica

6.1. CrimeSet.h

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

/**
 * @file  CrimeSet.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>
 *
 * @note To be implemented by students
 * Created on September 17, 2024, 5:43PM
 */

#ifndef CRIMESET_H
#define CRIMESET_H

#include <string>
#include <iostream>

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

/**
 * @class CrimeSet
 * @brief A CrimeSet defines a set of instances of the Crime class that includes
 * data on crimes (offenses) committed anywhere in the world and with the types
 * of crimes specifically considered within its own jurisdiction.
 * This class uses an array of Crimes objects with a fixed capacity
 * to store the set of crimes.
 * This class also contains a field to store comments,
 * free text that allows, for example, to describe in natural language the
 * time period considered, the origin, the query applied to obtain the set,
 * etc.
 * In general, the records in the set are not sorted by date and time
 * when each crime was committed.
 * Usually the records appear in the order in which they were recorded when
 * the crimes were collected by the police department.
 * The public methods of this class do not allow a CrimeSet to contain two
 * Crime objects with identical IDs.
 *
 * @see See the content of the files *.crm in the DataSets folder as examples
 * of files that contain information about CrimeSets
 */
class CrimeSet {
public:
    /**
     * @brief Basic constructor and initializer. It builds a CrimeSet object
     * with a set of 0 Crimes and a capacity of DIM_VECTOR_CRIMES. The field for
     * the comments is set as an empty string
     */
    CrimeSet();

    /**
     * @brief Returns the number of crimes stored in the set.
     * Query method
     * @return The number of crimes. Positive integer.
     */
    int getSize();

    /**
     * @brief Gets the capacity of the array of CrimeSet objects
     * Query method
     * @return The capacity of the array of CrimeSet objects
     */
    int getCapacity();

    /**
     * @brief Returns the field comment of this CrimeSet. This is a string that
     * may be composed of zero or several lines of free text.
     * Query method
     * @return A string with the lines of comments
     */
    std::string getComment();

    /**
     * @brief Sets the string comment for this CrimeSet object using
     * the provided string @p text. The string @p text can contain zero or
     * several lines; each one ends with '\n'.
     * If the last line in the string  @p text does not end with '\n' then
     * the character '\n' will be appended at the end of the comment string
     * of this object.
     * If @p text is an empty string, the comment string of this object will
     * be an empty string
     * Modifier method
     * @param text string with several lines of comments. Input parameter
     */
    void setComment(std::string text);

    /**
     * @brief Obtains a string with information about this CrimeSet object,
```

```
     * in the following format:
     * - First line, the number of crimes
     * - A line for each Crime with information about that Crime. Each line
     * is obtained using the method Crime::toString()
     * Query method
     * @return string with information about this CrimeSet object
     */
    std::string toString();

    /**
     * @brief Removes all the Crimes from this object, leaving the container
     * with a size of 0 and an empty comment
     * Modifier method
     */
    void clear();

    /**
     * @brief Appends the given Crime at the end of the array of Crimes of
     * this CrimeSet object, but only if that Crime was not found in this
     * CrimeSet.
     * @throw std::out_of_range Throws a std::out_of_range exception if the
     * number of elements in the array of Crimes is equals to the capacity
     * of that array. In that case, the array is full, and no more elements
     * can be appended to the array.
     * @param crime The Crime to append to this object. Input parameter
     * @return true if the given Crime was inserted in this CrimeSet object;
     * false otherwise
     */
    bool append(Crime crime);

    /**
     * @brief Gets the crime at the provided 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
     * provided position is not valid.
     * @return A reference to the Crime at the provided position
     */
    Crime at(int pos);

    /**
     * @brief Searches the provided Crime in the array of crimes in this
     * CrimeSet. If found, it returns the position where it was found. If not,
     * it returns -1. We consider that position 0 is the first crime in the
     * list of crimes and this->getSize()-1 the last crime.
     * In order to find a crime, consider only equality in the ID field.
     * Query method
     * @param crime A crime. Input parameter
     * @param initialPos initial position where to do the search.
     * Input parameter
     * @param finalPos final position where to do the search. Input parameter
     * @return If found, it returns the position where the crime
     * was found. Otherwise it returns -1
     */
    int findCrime(Crime crime, int initialPos, int finalPos);

    /**
     * @brief Searches the provided Crime in the array of crimes in this
     * CrimeSet. If found, it returns the position where it was found. If not,
     * it returns -1. We consider that position 0 is the first crime in the
     * list of crimes and this->getSize()-1 the last crime.
     * In order to find a crime consider only equality in the ID field.
     * Query method
     * @param crime A crime. Input parameter
     * @return If found, it returns the position where the crime
     * was found. Otherwise it returns -1
     */
    int findCrime(Crime crime);

    /**
     * @brief Loads into this object the CrimeSet information stored in the
     * given file. See files *.crm in the folder DataSets as example of files
     * with CrimeSet information.
     * @note Note that this method should remove the Crime objects that
     * this object previously contained.
     * @note This method throws an exception in some error cases (see below).
     * Before throwing the corresponding exception, this method clears
     * the object (it calls to clear() method) to leave the object in a
     * consistent state.
     * Modifier method
     * @param fileName The name of the file where the CrimeSet is stored.
     * Input parameter
     * @throw std::ios_base::failure Throws a std::ios_base::failure exception
     * if the given file cannot be opened or if an error occurs while reading
     * from the file.
     * @throw throw std::invalid_argument Throws a std::invalid_argument
     * exception if an invalid magic string is found in the given file
     * @throw std::out_of_range Throws a std::out_of_range exception if the
     * number of crimes in the given file is negative.
     */
    void load(std::string fileName);

    /**
     * @brief Saves this CrimeSet object in the given file.
     * @note This method sends each Crime to the output stream in the same
     * format as the one described in the Crime::toString() method
     * Query method
     * @param fileName A string with the name of the file where this CrimeSet
     * object will be saved. Input parameter
     * @throw std::ios_base::failure Throws a std::ios_base::failure exception
```

```
     * if the given file cannot be opened or if an error occurs while writing
     * to the file
     */
    void save(std::string fileName);

    /**
     * @brief Appends to this CrimeSet object, the list of
     * Crime objects contained in the provided CrimeSet @p crimeSet that are
     * not found (using CrimeSet::findCrime(Crime)) in this CrimeSet.
     * This method could be implemented with the help of the method
     * CrimeSet::append(Crime crime), to append to this
     * CrimeSet, the Crimes of the provided CrimeSet @p crimeSet.
     * Modifier method
     * @param crimeSet A CrimeSet object. Input parameter
     */
    void join(CrimeSet crimeSet);

    /**
     * @brief Normalizes each one of the Crime objects in this CrimeSet
     * Modifier method
     */
    void normalize();

    /**
     * @brief It takes the array of Crimes of this object 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.
     * Query method
     * @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 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 computeHistogram(int dataField, int histogram[]);

    /**
     * @brief Returns a CrimeSet object with those crimes whose label of
     * attribute is provided with the provided @p field parameter and whose
     * value is equals to the provided @p value parameter.
     * This function enables simple queries as: Code = "unknown" or
     * Code = "3114" as well as Street = "WASHINGTON ST" or
     * DateTime = dd.toString() (being dd, a DateTime object),
     * but no other relational operators as greater than, etc.
     * The new CrimeSet will have the same comments as this object plus a line
     * with the content: "Restricted to " + field + " = " + value + "\n"
     * Query method
     * @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
     * @return A CrimeSet with the selected Crimes.
     */
    CrimeSet selectWhereEQ(std::string field, std::string value);

    /**
     * @brief Returns a CrimeSet object with those crimes whose latitude and
     * longitude are both valid.
     * Example of invalid location for a crime are:
     * - (91,100) latitude is out of range
     * - (90,181) longitude is out of range
     * - (181,181) both are not valid
     * The new CrimeSet will have the same comments as this object plus a line
     * with the content: "Restricted to valid Coordinates\n""
     * Query method
     * @return A CrimeSet with those crimes whose latitude and longitude are
     * both valid
     */
    CrimeSet selectValidLocation();

    /**
     * Sorts the array of Crimes by increasing alphabetical order of the field
     * with the DateTime (sorting by date and time when the Crime took place).
     * If two Crimes objects have the same DateTime, then  increasing
     * alphabetical order of the field ID of those two objects will
     * be considered.
     * Modifier method
     */
    void sort();

private:
    static const int DIM_VECTOR_CRIMES = 2000; ///< The capacity of the array _crimes
    static const std::string MAGIC_STRING_T; ///< A string with the magic string for text files

    /**
     * string that contains several lines of comments (text in natural
     * language). Each line, except possibly the last one, ends with the
     * character '\n'
     */
    std::string _comment;

    Crime _crimes[DIM_VECTOR_CRIMES]; ///< Array of crimes
    int _nCrimes; ///< Number of used elements in _crimes
```

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

    /**
     * @brief Reads comment lines from the provided input stream and appends
     * them to the end of the string _comment.
     * In the provided input stream, comment lines are text lines that
     * begin with the character '#'.
     * This method reads lines until a line is read that does not begin with
     * the '#' character.
     * The '#' characters will not be saved in the string _comment, that is,
     * this character will be discarded.
     * In the string _comment, each comment line will have the character '\n' at
     * the end.
     * @param inputStream input stream from which comments will be read
     */
    void readComments(std::istream inputStream);

    /**
     * @brief Saves the line comments contained in the field _comment in the
     * provided output stream. Each line will be saved as '#' plus the content
     * of that line. This method can be easily implemented with the help of the
     * function FormatAsComment()
     * Query method
     * @param outputStream output stream where the comments will be saved
     */
    void saveComments(std::ostream outputStream);
}; // end class CrimeSet

/**
 * @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 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, second line 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[]);

/**
 * Returns a string with the same content as the provided string @p comment,
 * but with the character @p commentCharacter inserted at the begining
 * of each line, thus obtaining a formatted comment. The character
 * @p commentCharacter is '#' by default.
 * In case that the last line in @p comment does not have '\n' at the end,
 * this function add it to the end of the returned string.
 * @param comment The input string to be used to format the comment.
 * @param commentCharacter The character to use at the begining of the formatted
 * string
 * @return A string with the same content as the provided string @p comment,
 * but with the character @p commentCharacter inserted at the begining
 * of each line
 */
std::string FormatAsComment(std::string comment, char commentCharacter='#');

#endif /* CRIMESET_H */
```

6.2. CrimeSet.cpp

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

/**
 * @file  CrimeSet.cpp
 * @note To be implemented by students
 *
 * Created on November 26, 2024, 11:04AM
 */

#include <iostream>
#include <string>

#include "CrimeSet.h"

using namespace std;

const string CrimeSet::MAGIC_STRING_T="MP-CRIME-T-1.0";

string FormatAsComment(string comment, char commentCharacter){
    size_t start = 0, end;
    string formatedComment;
    if (comment.size() > 0) { // If comment has at least one character
        while ((end = comment.find('\n', start)) != std::string::npos) {
            formatedComment += commentCharacter + comment.substr(start, end - start) + "\n";
            start = end + 1;
        }
        if(start<comment.size()){ // This happens when last line of comment does not have \n
            formatedComment += commentCharacter + comment.substr(start, comment.size() - start) + "\n";
        }
    }
    return formatedComment;
}
```

6.3. ArrayCrimeSetFunctions.h

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

/**
 * @file  ArrayCrimeSetFunctions.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>
 * @note To be implemented by students
 *
 * Created on November 26, 2024, 11:04AM
 */

#ifndef ARRAYCRIMESETFUNCTIONS_H
#define ARRAYCRIMESETFUNCTIONS_H

#include "CrimeSet.h"

/**
 * @brief Reads the CrimeSets contained in the files whose names are provided
 * in the given input stream (one name of file in each line) and appends then
 * at the end of the dynamic array @p arrayCrimeSet.
 * In case that the input stream contains empty lines, they should be ignored,
 * because an empty line corresponds with the empty string for the name of
 * the file to be read.
 * This function can be implemented with the help of the function
 * AppendCrimeArrayCrimeSet() that automatically reallocate the dynamic array,
 * each time a new CrimeSet is appended to the array.
 * nCrimeSets will contain the number of CrimeSet saved in the the dynamic array
 * @param inputStream Input stream. Input/output parameter
 * @param arrayCrimeSet A pointer to a dynamic array of CrimeSet.
 * The pointer is an input/output parameter and the content of the array can
 * be modified
 * @param nCrimeSets Number of CrimeSets in the dynamic array arrayCrimeSet.
 * Input/output parameter
 */
void ReadArrayCrimeSet(std::istream inputStream, CrimeSet *arrayCrimeSet,
        int nCrimeSets);

/**
 * @brief Shows in the standard output the CrimeSet contained in the
 * provided array @p arrayCrimeSet. Each CrimeSet is shown with the
 * help of the method CrimeSet::toString()
 * @param arrayCrimeSet A pointer to a dynamic array of CrimeSet.
 * The pointer is an input parameter and the content of the array is not modified
 * @param nCrimeSets Number of CrimeSets in the dynamic array arrayCrimeSet.
 * Input parameter
 */
void PrintArrayCrimeSet(CrimeSet *arrayCrimeSet, int nCrimeSets);

/**
 * @brief Allocates memory for a dynamic array of CrimeSets with a capacity
 * equals to @p nCrimeSets
 * @param nCrimeSets number of CrimeSets to allocate in the dynamic array.
 * Input parameter
 * @return A pointer to the allocated memory for the dynamic array. If
 * @p nCrimeSets <= 0 it returns nullptr
 */
CrimeSet * AllocateArrayCrimeSet(int nCrimeSets);

/**
 * @brief Deallocates the memory pointed by the provided dynamic array of
 * CrimeSets @arrayCrimeSet. That pointer will also be set to nullptr
 * @param arrayCrimeSet A pointer to a dynamic array of CrimeSet.
 * The pointer is an input/output parameter and the content of the array is
 * modified
 */
void DeallocateArrayCrimeSet(CrimeSet * arrayCrimeSet);

/**
 * @brief Appends the provided CrimeSet @p newCrimeSet at the end of the
 * dynamic array of CrimeSet @p arrayCrimeSet. This function starts by
 * reallocating the array, increasing its capacity by 1 to allow saving the
 * new CrimeSet.
 * @param arrayCrimeSet A pointer to a dynamic array of CrimeSet.
 * The pointer is an input/output parameter and the content of the array can
 * be modified
 * @param nCrimeSets Number of CrimeSets in the dynamic array arrayCrimeSet.
 * Input/output parameter
 * @param newCrimeSet The new CrimeSet to be inserted at the end of the
 * dynamic array. Input parameter
 */
void AppendCrimeArrayCrimeSet(CrimeSet * arrayCrimeSet, int nCrimeSets,
        CrimeSet newCrimeSet);

#endif /* ARRAYCRIMESETFUNCTIONS_H */
```

6.4. main.cpp

```
/*
 * Metodología de la Programación: Boston2
 * Curso 2024/2025
 */

/**
 * @file   main.cpp
 * @author estudiante1: apellidos*, nombre*
 * @author estudiante2: apellidos*, nombre* (solo si procede)
 * @note To be implemented by students
 * Created on September 25, 2024, 2:01PM
 */

#include <string>
#include <iostream>
#include <fstream>

#include "CrimeSet.h"
#include "ArrayCrimeSetFunctions.h"

using namespace std;

void showHelp(std::ostream& outputStream);

/**
 * Shows help about the use of this program in the given output stream
 * @param outputStream The output stream where the help will be shown (for example,
 * cout, cerr, etc)
 */
void showHelp(std::ostream& outputStream) {
    outputStream << "ERROR in boston2 parameters" << endl;
    outputStream << "Run with the following arguments:" << endl;
    outputStream << "boston2 <inputFile.txt> [<outputFile.crm>]" << endl;
    outputStream << endl;
    outputStream << "Parameters:" << endl;
    outputStream << "<inputFile.txt>: name of the input text file that contains the names of the crm files"
     << endl;
    outputStream << "<outputFile.crm>: name of the output crm file (tests/output/output.crm by default)" <<
      endl;
    outputStream << endl;
}

/**
 * The purpose of this program is to read a list of crm files,
 * each one containing information about a CrimeSet, and make the fusion of
 * all the CrimeSets, saving the result in a new file.
 * The names of the input crm files are contained in a text file whose name
 * is provided to the program as the first argument.
 * The name of the output crm file can be provided to the program as the second
 * argument (if not provided the output file is tests/output/output.crm). See below
 * the Running sintax.
 *
 * The program begins by checking if the number of arguments is valid.
 * Next, it reads and saves in a dynamic array of CrimeSet objects, each one
 * of the crm files whose names are provided with the first argument of the
 * program.
 * Then, for each one of the CrimeSet in the dynamic array, the following steps
 * are done:
 * - Normalize the CrimeSet. To do that, you can use the CrimeSet::normalize()
 * method.
 * - Select only those Crimes with a valid location (discarding all with invalid
 * location). To do that, you can use the CrimeSet::selectValidLocation() method.
 * - Select only those Crimes where Shooting=="1". To do that, you can use the
 * CrimeSet::selectWhereEQ() method.
 * Hereafter, you must obtain a CrimeSet with the fusion (join) of all the
 * CrimeSet in the array. See in the running examples, which comment should
 * have the resulting CrimeSet.
 * Finally, the resulting CrimeSet should be sorted and saved in an output
 * file.
 *
 * Running sintax:
 * > dist/Debug/GNU-Linux/boston2 <inputFile.txt> [<outputFile.crm>]
 *
 * Running example:
 * > dist/Debug/GNU-Linux/boston2 data/input06.b2in /tmp/output.crm
 *
 * > cat tests/output/output.crm
MP-CRIME-T-1.0
#Fusion of the crm files whose names are in the file: data/input06.b2in
5
7,222073971,611,UNKNOWN,LARCENY PICK-POCKET,A1,77,1,2022-02-01 10:07:00,NEW SUDBURY ST,42.361839,-71.059769
1,222648862,3831,UNKNOWN,M/V - LEAVING SCENE - PROPERTY DAMAGE,B2,288,1,2022-02-05 00:00:00,WASHINGTON ST
      ,42.329750,-71.084541
0,225520077,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON WARRANT,D14,786,1,2022-02-05 00:00:00,WASHINGTON
      ST,42.343082,-71.141724
3,222201559,301,UNKNOWN,ROBBERY,D4,UNKNOWN,1,2022-03-05 13:00:00,ALBANY ST,42.333183,-71.073936
5,222107076,3126,UNKNOWN,WARRANT ARREST - OUTSIDE OF BOSTON WARRANT,D4,UNKNOWN,1,2022-03-11 10:45:00,
      MASSACHUSETTS AVE & ALBANY ST BOSTON  MA 02118 UNI
,42.333500,-71.073509
 * @return Returns 1 in case of error in program arguments; 0 otherwise
 */
int main(int argc, char* argv[]) {
    string outputFileName = "tests/output/output.crm"; // Name of the output crm file

    // Check if the number of arguments is valid.

    // Read and save in a dynamic array of CrimeSet objects,
```

```
    //    each one of the crm files from the input file

    // Loop for every CrimeSet in the dynamic array
            // Normalize the CrimeSet (normalize each Crime in the CrimeSet)
            // Select only those Crimes with a valid location (discard all with invalid location)
            // Select only those Crimes where Shotting =="1"

    // Make the fusion of all the CrimeSets in the array

    // Sort the array of the resulting CrimeSet
    // Save the CrimeSet in the output crm file

}
```