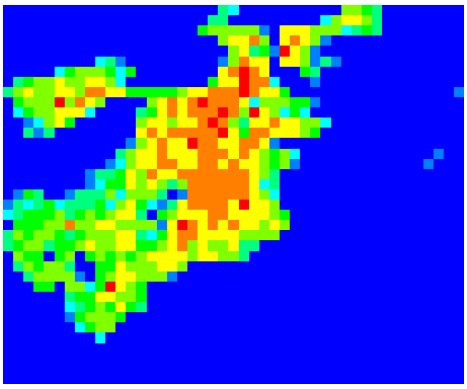
UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

Curso 2024/2025



Guion de prácticas

Boston3

Clase CrimeSet con vector dinámico

Abril de 2025

Índice

1. Introducción

En esta práctica descartamos el módulo `ArrayCrimeSetFunctions` de la práctica anterior, pues ya no lo vamos a necesitar. Conservamos el resto de las clases, aunque la clase `CrimeSet` es refactorizada para que utilice un array alojado en memoria dinámica, en lugar de un vector en memoria automática (la pila). Este cambio provocará que tengamos que revisar gran parte de los métodos de la clase `CrimeSet`. No será necesario revisar el resto de las clases.

Al igual que con la práctica anterior, en esta vamos a desarrollar una aplicación que nos permita obtener la fusión de una lista de ficheros `.crm`. En este caso, los nombres de los ficheros de entrada se pasarán al programa directamente por la línea de comandos. El nombre del fichero de salida también se pasará al programa por la línea de comandos.

2. Arquitectura de las prácticas

Como indicamos en la Introducción, en `Boston3` desaparece el módulo `ArrayCrimeSetFunctions`, y se refactoriza la clase `CrimeSet`. 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
: Necesita una importante refactorización para usar un vector dinámico de objetos `Crime` en lugar de uno automático (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 Boston3 persigue los siguientes objetivos:

- Practicar con una clase compuesta de un array dinámico de objetos y otros datos: la clase `CrimeSet`.
- Aprender a desarrollar los métodos básicos de una clase que contenga memoria dinámica: constructor de copia, destructor y operador de asignación.

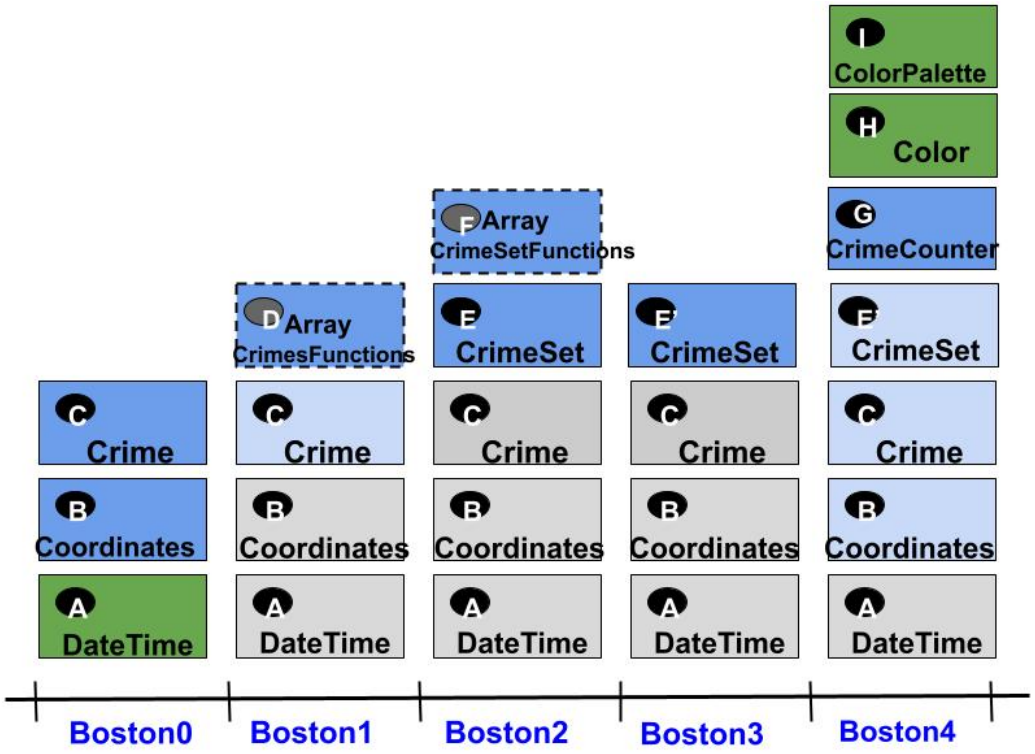


Figura 1: Arquitectura de las prácticas de MP 2025. Como podemos observar, los bloques correspondientes a cada módulo, aparecen en distintos colores. El color verde indentifica software ya desarrollado (completamente terminado), mientras que el azul indica software que requiere ser desarrollado por el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

- Refactorizar la clase `CrimeSet` para reducir al mínimo el número de cambios a realizar en los métodos anteriores de la clase.
- Seguir practicando con la lectura de los parámetros pasados a un programa por la línea de comandos.
- Ahondar en el uso del depurador de NetBeans y `valgrind` para rastrear errores en tiempo de ejecución.

4. Vector dinámico con capacidad

Como se ha visto en las clases de teoría, existen varias formas para gestionar vectores dinámicos.

1. Una de ellas sería hacer que en todo momento, la reserva de memoria dinámica del vector se ajuste al número de objetos que estemos usando en ese momento. Esta forma hace que se necesiten

dos datos para gestionar un vector dinámico: un puntero (llamado por ejemplo `_array`) al primer elemento del vector y un entero que indique el número de elementos de que disponemos (llamado por ejemplo `_util`). En esta situación, cada operación de añadir un objeto supondría realojar el vector [1].

2. Una forma alternativa consiste en llevar por un lado, la cuenta del espacio reservado (o sea, la *capacidad*) del vector y por otro, el número de objetos que estemos usando en cada momento. Llamemos por ejemplo, `_capacidad` y `_util` a los datos que guardan estos valores. Dentro de esta alternativa, a su vez, existen varias opciones en la forma en que el vector dinámico se realoja cuando se agota su capacidad:

   a) Una opción sería hacer que el nuevo vector multiplique su tamaño por algún factor dado respecto al que tenía anteriormente.

   b) Otra opción sería hacer que el nuevo vector tenga como tamaño el que tenía anteriormente más un bloque en el que quepan un número fijo de elementos adicionales.

En esta práctica, usaremos la alternativa 2, ya que permite que el vector dinámico no deba ser continuamente realojado conforme le vamos añadiendo nuevos elementos. Y concretamente nos decantamos por la opción 2*a*.

Ejemplo 1 *Sea un vector reservado en un bloque de capacidad para 5 objetos, con tan solo 3 objetos operativos. Un esquema puede verse en la figura 2.*

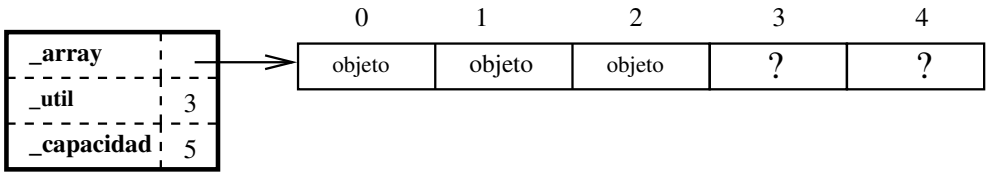


Figura 2: Reserva de espacio para 5 objetos, aunque el vector tan solo disponga de 3 objetos operativos, el resto están a valores por defecto (valor ?).

En el estado del ejemplo 1, a la llegada sucesiva de dos nuevos objetos, estos se podrán alojar en el mismo vector hasta ocupar la capacidad total, sin tener que realojar el vector. En este caso, basta con incrementar `_util`, como podemos ver en la figura 3.

Siguiendo con nuestro ejemplo, si llegan nuevos objetos, será necesario realojar el vector en otro espacio redimensionado a una capacidad mayor. Supongamos que cada vez que se agota la capacidad del vector

[1] Como ejercicio didáctico es muy útil pero, está muy lejos de ser realista, dado el esfuerzo que supone el realojo, que al final se traduce en tiempos.

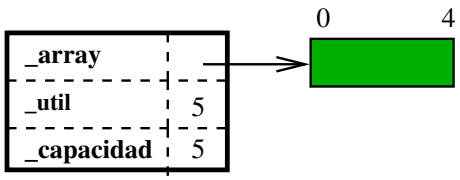


Figura 3: Reserva de espacio para 5 objetos, el vector está completo (`_util` *coincide con* `_capacidad`*).*

usamos un factor de aumento de la capacidad de 2. Al hacerlo, tendríamos un nuevo vector con capacidad para 10 objetos tal como podemos ver en la figura 4 a). Esto es, se ha doblado su capacidad.

Veamos con detalle, cómo se procede cuando se agota la capacidad del vector dinámico (_util coincide con `_capacidad`) y llega un nuevo objeto para ser insertado en el vector. Será necesario realojar el vector anterior en uno más grande, copiar toda la información previa en el nuevo espacio y finalmente liberar el espacio anterior. Gráficamente el proceso puede seguirse en la figura 4.

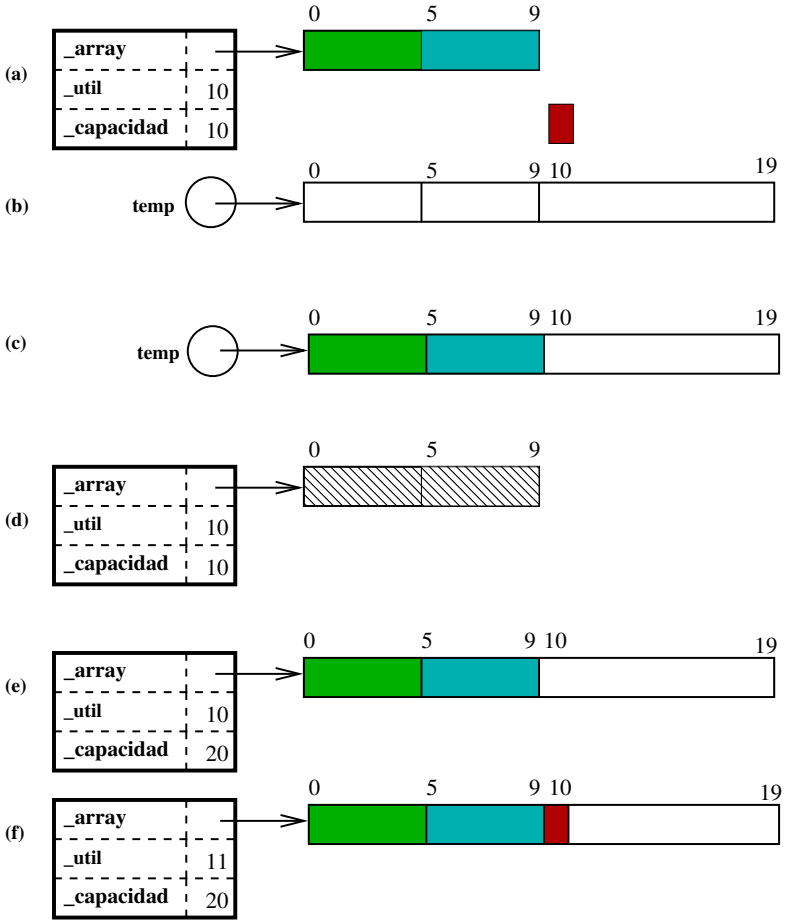


Figura 4: Proceso completo de realojo y adición de un elemento.

Los pasos que se han seguido son los siguientes:

1. En (a), el vector está al completo a la llegada del nuevo objeto marcado en rojo.

2. En b) se reserva nueva memoria dinámica, mediante el puntero auxiliar `temp`, con capacidad para $10 * 2$ objetos.
3. En c) se copia el contenido del vector dinámico en el nuevo vector apuntado por `temp`.
4. En d) se libera la memoria dinámica del vector dinámico original.
5. En e) se corrige la dirección del puntero `_array` para que apunte a la dirección de `temp`, y se ajusta el valor de la capacidad del nuevo vector. Llegado a este punto, ya ha llevado a cabo el realojo del vector en uno de mayor capacidad.
6. Por último, en f) se añade al vector dinámico el nuevo objeto, ya que hay espacio disponible. Se ajusta también el valor del número de elementos usados en el vector.

Si la gestión del vector dinámico se hubiera hecho con la alternativa 1, habría que haber realojado el vector con cada operación de adición de un objeto. Con la alternativa elegida, se logra reducir el esfuerzo de realojo.

5. Práctica a entregar

5.1. Finalidad del programa

Al igual que en Boston2, el programa tiene por objeto leer una lista de ficheros `crm`, cada uno de los cuales contiene información sobre un conjunto de crímenes, y hacer la fusión de todos ellos. En este caso, los nombres de los ficheros `crm` de entrada se pasarán directamente al programa por la línea de comandos (vea los ejemplos de ejecución de la sección 5.2). Otra diferencia respecto a Boston2, es que los ficheros `crm` de entrada no deben cargarse en un array dinámico en la función `main()`, simplemente la fusión se calculará conforme se van leyendo los ficheros de entrada. Para ello el programa usará un objeto `CrimeSet` (llamado por ejemplo `fusion`) usado para ir obteniendo la fusión de los ficheros leídos hasta el momento.

El resultado de la fusión de todos los ficheros `crm` se grabará en un nuevo fichero `crm` de salida cuyo nombre puede proporcionarse al programa con el parámetro opcional `-o <outputFile.crm>`. Si no se proporciona, su nombre ha de ser `tests/output/output.crm`.

Hay alguna diferencia más respecto a Boston2. Vea a continuación los detalles de los pasos que lleva a cabo Boston3:

- El programa comienza procesando los argumentos pasados al programa. Se debe analizar si el programa se ejecutó al menos con un fichero de entrada `crm` y ver si se usó el parámetro opcional `-o <outputFile.crm>` para establecer el fichero `crm` de salida.
- Seguidamente, para cada uno de los ficheros `crm` de entrada se llevan a cabo los siguientes pasos:

  - Cargar el fichero `crm` en un objeto `CrimeSet`.
  - Añadir al objeto `fusion` los crímenes del objeto `CrimeSet` normalizado previo (hacer la fusión).

- Debe incluirse en el `CrimeSet` resultado un comentario (usar el método `CrimeSet::setComment(comment)`) que indique los nombres de los ficheros de entrada `crm` usados para hacer la fusión. Tenga en cuenta que la cadena `comment` no debe incluir el carácter '`#`' de cada línea, pues tal carácter lo incluirá 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 de los ficheros indicados:

```
#Fusion of the following crm files:
#../DataSets/crimes_01.crm
#../DataSets/crimes_05.crm
```

  Vea los ejemplos de ejecución de la sección 5.2.

- Normalizar el objeto resultado de la fusión de todos los ficheros `crm` de entrada (objeto `fusion`).
- Ordenar el objeto `fusion` obtenido tras la normalización del paso anterior.
- Finalmente, el `CrimeSet` resultado debe ser grabado en el fichero `crm` de salida.

5.2. Sintaxis y ejemplos de ejecución

Como se indicó en la Introducción, en esta práctica usaremos directamente los parámetros de la línea de comandos, para pasarle al programa los nombres de los ficheros `crm` de entrada. El nombre del fichero `crm` de salida se puede pasar al programa con un parámetro opcional desde la línea de comandos: `[-o <outputFile.crm>]`.

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

```
linux> dist/Debug/GNU-Linux/boston3 [-o <outputFile.crm>] <inputFile1.crm>
↪  {<inputFile.crm>}
```

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

Ejemplo 2 *Faltan argumentos para la ejecución del programa:*

```
linux> dist/Debug/GNU-Linux/boston3
```

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

```
ERROR in boston3 parameters
Run with the following arguments:
boston3 [-o <outputFile.crm>] <inputFile1.crm> {<inputFile.crm>}

Parameters:
-o <outputFile.crm>: name of the output file (tests/output/output.crm by
↪  default)
<inputFile1.crm> {<inputFile.crm>}: input crm files
```

Ejemplo 3 *Ejecución con parámetro opcional erróneo:*

```
linux> dist/Debug/GNU-Linux/boston3 -f
↪  ../DataSets/crimes_easy01.crm ../DataSets/crimes_easy01.crm
```

En este caso, el argumento `-f` *no es válido para Boston3. La salida del programa será de nuevo la que genera la función* `showHelp()` *(ver ejemplo 2).*

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

```
linux> dist/Debug/GNU-Linux/boston3 ../DataSets/crimes_05.crm
```

En la anterior ejecución, solo se proporciona un fichero de entrada (`../DataSets/crimes_05.crm`*), con lo que el resultado tendrá solo los crímenes de este fichero, pero aparecerán normalizados y ordenados según el criterio comentado anteriormente. El contenido de este fichero de entrada es el siguiente:*

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
```

El nombre del fichero de salida obtenido con la anterior ejecución es `tests/ouput/output.crm`*, que tendrá el siguiente contenido:*

Fichero tests/ouput/output.crm:

```
MP-CRIME-T-1.0
#Fusion of the following crm files:
#../DataSets/crimes_05.crm
5
2,222201764,724,UNKNOWN,AUTO THEFT,C6,200,0,2022-01-09 00:00:00,W
↪  BROADWAY,42.341286,-71.054680
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
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
```

Ejemplo 5 *Ejecución proporcionando dos ficheros de entrada:*

```
linux> dist/Debug/GNU-Linux/boston3 ../DataSets/crimes_01.crm
↪  ../DataSets/crimes_05.crm
```

El nombre del fichero de salida obtenido con la anterior ejecución es de nuevo `tests/ouput/output.crm`*, que en este caso tendrá el siguiente contenido:*

```
MP-CRIME-T-1.0
#Fusion of the following crm files:
#../DataSets/crimes_01.crm
#../DataSets/crimes_05.crm
6
2,222201764,724,UNKNOWN,AUTO THEFT,C6,200,0,2022-01-09 00:00:00,W
↪  BROADWAY,42.341286,-71.054680
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
```

Ejemplo 6 *Ejecución proporcionando dos ficheros de entrada y el de salida:*

```
linux> dist/Debug/GNU-Linux/boston3 -o /tmp/output.crm
↪  ../DataSets/crimes_01.crm ../DataSets/crimes_05.crm
```

El fichero de salida es en este caso `/tmp/output.crm`*. El contenido de este fichero debe ser el mismo que el obtenido con el ejemplo anterior (ejemplo 5).*

En la carpeta `tests` del proyecto Boston3, puede encontrar varios ficheros de texto con extensión `.test` con los que podrá hacer diferentes validaciones con el script `runTests.sh`. La carpeta `data` 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 la carpeta `tests`. Así por ejemplo, el contendido de `data/outcome06.crm` es el que debería obtenerse como salida si hubieramos ejecutado el test del fichero `tests/Integrity06.test`.

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 detallados sus especificaciones.

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`. En esta práctica hay un cambio importante en esta clase respecto a la práctica anterior. Ahora el array de objetos `Crime` está alojado en memoria dinámica en lugar de en memoria automática. Este array puede aumentar su capacidad cuando se agote al insertar nuevos crímenes. Describamos brevemente los datos miembro de la nueva versión de la clase:

  - `_crimes`: es ahora un puntero a un array de objetos `Crime`.
  - `_nCrimes`: igual que en la práctica anterior, permite conocer en todo momento el número de elementos usados en el array.
  - `_capacity`: guarda la capacidad actual del array dinámico.
  - `_comment`: igual que en la práctica anterior, guarda una cadena con los comentarios asociados al `CrimeSet`.
  - `INITIAL_CAPACITY`: constante que indica la capacidad con la que el constructor de la clase creará el array dinámico inicialmente.
  - `GROWING_RATIO`: constante que proporciona el factor de crecimiento que se aplicará a la capacidad actual del array para calcular la nueva capacidad cuando esta se agote. Su valor es inicializado en el fichero `CrimeSet.cpp`.

```
class CrimeSet {
...
private:
    static const int INITIAL_CAPACITY = 8; ///< Default initial capacity for the dynamic array
      _crimes. Should be a number >= 0

    /**
     * Ratio to be used to compute the next capacity to allocate in the
     * dynamic array of Crimes. When the dynamic array is full and a new
     * Crime must be appended, it is reallocated with a
     * capacity calculated as ceil(_capacity * GROWING_RATIO)
     */
    static const float GROWING_RATIO;

    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; ///< Pointer to a dynamic array of crimes
    int _nCrimes; ///< Number of used elements in the dynamic array _crimes
    int _capacity; ///< Number of reserved elements in the dynamic array _crimes
}; // end class CrimeSet
```

- 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 6.2), 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 Boston3.zip, sin más aditivos.
Como alternativa, se sugiere utilizar el script `runZipProject.sh` que debe estar en la carpeta `scripts` de Boston3, 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 <fstream>
#include <cmath>        /* ceil */

#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 a dynamic array of objects of the Crime class 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:
    // Retrieve all previous declarations and take into account
    // the new declarations included here...
    //

    /**
     * @brief Basicructor and initializer. It builds a CrimeSet object
     * with a set of of 0 Crimes and a capacity calculated as follows:
     * - If size>=0 and size<INITIAL_CAPACITY, then the capacity of the
     * dynamic array will be set to INITIAL_CAPACITY.
     * - If size>=INITIAL_CAPACITY, then the capacity of the dynamic array
     * will be equals to size.
     * The field for the comments is set as an empty string
     * @throw std::out_of_range Throws a std::out_of_range exception if
     * @p size<0. In this case, the fields of the object are initialized
     * as follows:
     * - _comment = ""
     * - _capacity = 0
     * - _nCrimes = 0
     * - _crimes = nullptr
     * After the fields are initialized as described above, the exception
     * is thrown.
     * @param size The number of Crimes that will have the set
     */
    CrimeSet(int size=0);

    /**
     * @brief Copyructor. Builds a deep copy of the provided CrimeSet
     * @p orig.
     * @param orig CrimeSet original. Input parameter
     */
    CrimeSet(CrimeSet orig);

    /**
     * @brief Destructor
     */
    ~CrimeSet();

    /**
     * @brief Overloads the assignment operator for CrimeSet class.
     * Modifier method
     * @param orig the CrimeSet object used as source for the assignment.
     * Input parameter
     * @return A reference to this object
```

```
     */
    CrimeSet operator=(CrimeSet orig);

private:
    static const int INITIAL_CAPACITY = 8; ///< Default initial capacity for the dynamic array _crimes.
     Should be a number >= 0

    /**
     * Ratio to be used to compute the next capacity to allocate in the
     * dynamic array of Crimes. When the dynamic array is full and a new
     * Crime must be appended, it is reallocated with a
     * capacity calculated as ceil(_capacity * GROWING_RATIO)
     */
    static const float GROWING_RATIO;

    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; ///< Pointer to a dynamic array of crimes
    int _nCrimes; ///< Number of used elements in the dynamic array _crimes
    int _capacity; ///< Number of reserved elements in the dynamic array _crimes


    // fill free to define as much method to manage the dynamic memory
    // ...

}; // end class CrimeSet
// retrieve previous functions ...

#endif /* CRIMESET_H */
```

6.2. main.cpp

```
/*
 * Metodología de la Programación: Boston3
 * Curso 2024/2025
 */


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

#include <string>
#include <iostream>

#include "CrimeSet.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 boston3 parameters" << endl;
    outputStream << "Run with the following arguments:" << endl;
    outputStream << "boston3 [-o <outputFile.crm>] <inputFile1.crm> {<inputFile.crm>}" << endl;
    outputStream << endl;
    outputStream << "Parameters:" << endl;
    outputStream << "-o <outputFile.crm>: name of the output file (tests/output/output.crm by default)" <<
      endl;
    outputStream << "<inputFile1.crm> {<inputFile.crm>}: input crm files" << 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 crm file.
 * The names of the input crm files are passed as arguments to the program.
 * The name of the output crm file can be provided to the program as an optional
 * argument (if not provided, the output file is /tmp/output.crm). See below
 * the Running sintax.
 *
 * The program begins by processing the running arguments.
 * Next, it obtains the fusion of the crm files whose names are passed as
 * arguments to the program. The resulting CrimeSet will have a comment with
 * the list of the input crm files used to do the fusion. See below, the Running
 * example.
 * Finally, the resulting CrimeSet is normalized, sorted and saved in the
 * output crm file.
 *
 * Running sintax:
 * > dist/Debug/GNU-Linux/boston3 [-o <outputFile.crm>] <inputFile1.crm> {<inputFile.crm>}
 *
 * Running example:
 * > dist/Debug/GNU-Linux/boston3 -o /tmp/output.crm ../DataSets/crimes_01.crm ../DataSets/crimes_05.crm
 *
 * > cat /tmp/output.crm
MP-CRIME-T-1.0
#Fusion of the following crm files:
#../DataSets/crimes_01.crm
#../DataSets/crimes_05.crm
6
2,222201764,724,UNKNOWN,AUTO THEFT,C6,200,0,2022-01-09 00:00:00,W BROADWAY,42.341286,-71.054680
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
    string comments = "Fusion of the following crm files: \n"; // Comments for the result of fusion


    // Process program arguments

    // for every file from the main arguments
        // load the file
        // do the fusion
        // compose the comment

    // set the comments, normalize, sort and save

    return 0;
}
```