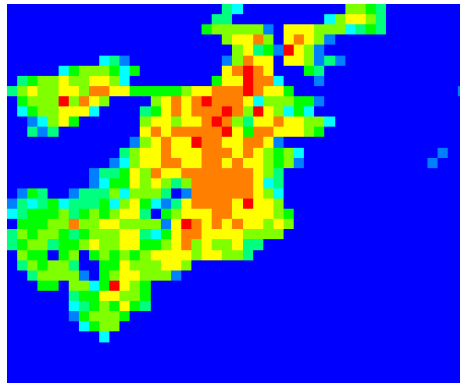




Metodología de la Programación

Curso 2024/2025



Guion de prácticas

Boston4

Práctica final

Abril de 2025

Índice

1. Introducción	5
2. Arquitectura de las prácticas	5
3. Objetivos	6
4. Color, paleta de colores y ficheros de imagen PPM	7
4.1. Color	7
4.2. Paleta de colores	7
4.3. Ficheros de imagen PPM	8
5. Práctica a entregar	10
5.1. Finalidad del programa	10
5.2. Sintaxis y ejemplos de ejecución	13
5.3. Módulos del proyecto	18
5.4. Para la entrega	20
6. Código para la práctica	21
6.1. Color.h	21
6.2. ColorPalette.h	23
6.3. Coordinates.h	25
6.4. Crime.h	26
6.5. CrimeSet.h	28
6.6. CrimeCounter.h	30
6.7. CrimeCounter.cpp	34
6.8. main.cpp	35



1. Introducción

El propósito de esta práctica final es obtener un mapa de calor para una zona de interés (área rectangular) que permita identificar las zonas más peligrosas de la ciudad de Boston.

En esta práctica conservamos todas las clases desarrolladas en la práctica anterior, aunque necesitamos añadir nueva funcionalidad a las clases `Coordinates`, `Crime` y `CrimeSet`. Además, esta práctica necesita de la creación de una nueva clase, la clase `CrimeCounter`, que contiene una matriz dinámica que servirá para el conteo de las frecuencias de las distintas áreas en las que se divida la zona de interés. Se incorporarán también las clases `Color` y `ColorPalette` que se proporcionan totalmente implementadas y no necesitan de ningún cambio.

El programa de esta práctica necesita controlar un mayor número de parámetros por línea de comandos que el de la práctica anterior, como veremos en la sección 5.2.

2. Arquitectura de las prácticas

Como indicamos en la Introducción, en `Boston4` aparecen tres nuevas clases: `CrimeCounter`, `Color` y `ColorPalette`. 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` requiere que se sobrecarguen los operadores de entrada y salida.

C Crime

La clase `Crime` requiere también que se sobrecarguen los operadores de entrada y salida. Además, deben sobrecargarse todos los operadores relacionales.

E CrimeSet

La clase `CrimeSet` requiere también que se sobrecarguen los operadores de entrada y salida. Además, deben sobrecargarse los operadores `[]` y `+=`.

G CrimeCounter

Nueva clase que contiene una matriz dinámica de enteros, usada en el conteo de frecuencias de crímenes en las áreas en que se divida la zona de interés.

H Color

Implementa la clase `Color`, una clase ya definida por completo, que va a ser de utilidad para la representación de cada uno de los colores que usaremos en nuestras imágenes mapa de calor. Un objeto

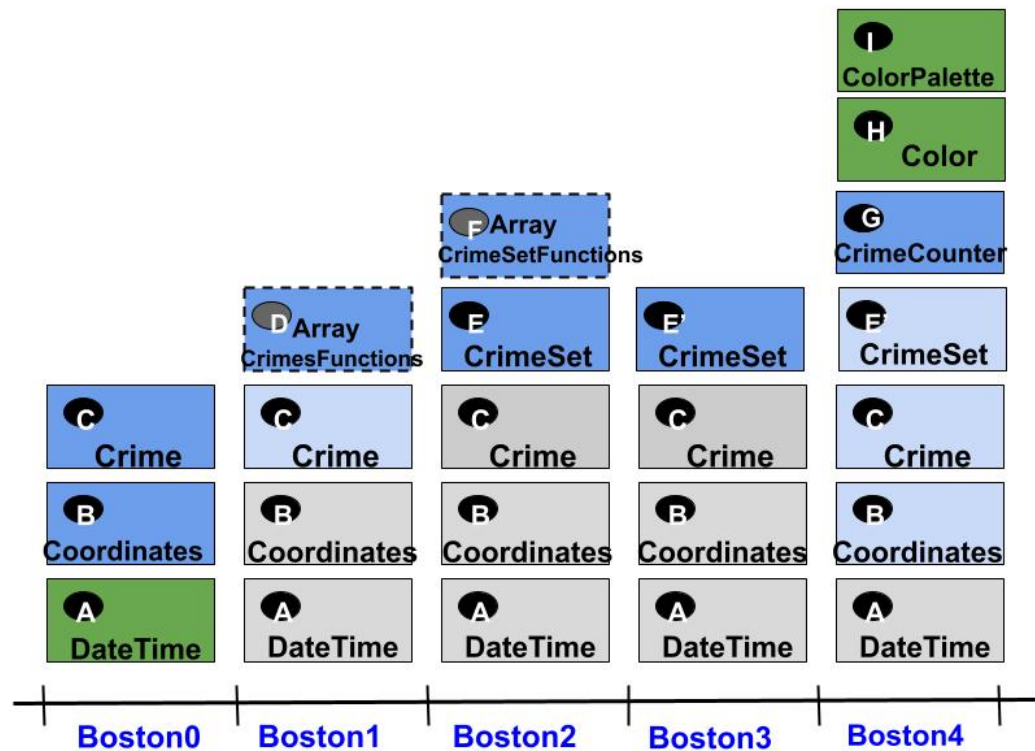


Figura 1: Arquitectura de las prácticas de MP 2025. Como podemos observar, los bloques correspondientes a cada módulo, aparecen en distintos colores. El color verde indentifica software ya desarrollado (completamente terminado), mientras que el azul indica software que requiere ser desarrollado por el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

de esta clase encapsula un color en el espacio RGB mediante tres componentes: rojo, verde y azul.

I ColorPalette

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

main.cpp Contiene la función main() cuya finalidad será la de obtener un mapa de calor para una determinada área de interés a partir de un fichero `crm` de crímenes .

3. Objetivos

El desarrollo de la práctica Boston4 persigue los siguientes objetivos:

- Practicar con punteros y memoria dinámica dentro de una clase para la implementación de una matriz bidimensional.

- Profundizar en el desarrollo de los métodos básicos de una clase con memoria dinámica: constructor de copia, destructor y operador de asignación.
- Practicar con la sobrecarga de distintos operadores tanto monarios como binarios.
- Profundizar en el uso del paso de parámetros a `main()` desde la línea de comandos (parámetros alternativos, optativos y por defecto).

4. Color, paleta de colores y ficheros de imagen PPM

Para la elaboración de un mapa coroplético (también llamado mapa de calor) de la ciudad de Boston, vamos a usar imágenes a **color**, asociando frecuencias bajas con colores fríos y frecuencias altas con colores cálidos. Una **paleta de colores** es una lista de n colores que asocia un color a cada índice (número entero entre 0 y $n - 1$, siendo n el número de colores de la paleta). La imagen final generada será almacenada en un formato de imagen interpretable por la mayoría de los visualizadores de imagen, el **formato** PPM. Veamos en qué consiste cada uno de los elementos mencionados.

4.1. Color

Usaremos la clase `Color` para la representación de cada uno de los colores de nuestras imágenes de mapa de calor. Esta clase se proporciona totalmente implementada, y no necesita modificarla. Un objeto de esta clase encapsula un color en el espacio RGB (del inglés *Red, Green, Blue*). Este espacio utiliza una combinación de los tres colores primarios (rojo, verde y azul) para obtener el resto de colores que podemos ver en una pantalla.

Cada uno de los tres componentes de un color es un valor entero que puede variar en el rango entre 0 y 255; 0 indica mínima intensidad y 255 máxima intensidad.

Podemos ver ejemplos de colores RGB en la figura 2. El color azul puro tiene por ejemplo los valores (0, 0, 255), en el que las componentes rojo y verde están a 0 (mínima intensidad) y la azul a 255 (máxima intensidad); el amarillo tiene los valores (255, 255, 0), mezcla de rojo y verde a máxima intensidad.

4.2. Paleta de colores

Una paleta de colores es una selección específica de colores, que nos va a permitir crear una estética visual coherente y atractiva en nuestras imágenes. En esta práctica vamos a utilizar paletas con colores equidistantes entre sí en el círculo cromático, que agrupamos en 4 grupos de colores:










	R	G	B	
0	0	0	255	
1	0	128	255	
2	0	255	255	
3	0	255	127	
4	0	255	0	
5	128	255	0	
6	255	255	0	
7	255	127	0	
8	255	0	0	

Figura 2: Paleta de 9 colores

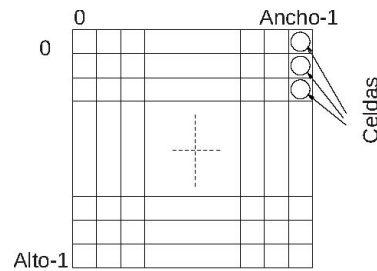
1. Colores desde azul puro (incluido) hasta azul claro (no incluido).
2. Colores desde azul claro (incluido) hasta verde (no incluido).
3. Colores desde verde (incluido) hasta amarillo (no incluido).
4. Colores desde amarillo (incluido) hasta rojo (no incluido).

La figura 2 muestra un ejemplo de esta división usando 2 colores en cada grupo, más el color rojo que aparece en la última posición de la paleta.

La clase `ColorPalette` permite definir la lista de colores que usaremos en nuestras imágenes mapa de calor. Esta clase se proporciona totalmente implementada en *Boston4* y no necesita de ninguna modificación. Tiene un constructor con 4 parámetros enteros que permite seleccionar el número de colores en cada grupo. Tales parámetros toman por defecto el valor 2. Con estos valores por defecto, la paleta construida es la mostrada en la figura 2. Usando otros valores para los 4 parámetros, podemos construir otras paletas de colores diferentes.

4.3. Ficheros de imagen PPM

Un mapa de calor es una imagen digital en color que guardaremos en formato PPM en esta práctica. Una imagen en color puede verse como una matriz de puntos, llamados píxeles, determinada por sus dimensiones (ancho x alto). Desde un punto de vista de estructura de datos, una imagen se puede considerar como una matriz bidimensional de celdas. Cada celda de la matriz almacena la información de un píxel.



Por simplicidad, en esta práctica no vamos a construir una clase *Imagen*. Directamente obtendremos el fichero PPM del mapa de calor a partir de la matriz de frecuencias obtenida con un objeto *CrimeCounter*, usando el método `saveAsPPMTextImage()`.

El formato PPM¹ (acrónimo de **P**ortable **P**ix**M**ap) permite guardar imágenes en color de cualquier tamaño. En este formato, cada píxel se representa mediante tres números enteros (valores de luminosidad), las componentes RGB; cada una de ellas puede tomar valores en el intervalo [0,255]. El formato PPM guarda las imágenes como ficheros de texto (ASCII). Describamos a continuación lo que encontramos por orden en un fichero en formato PPM:

- Una *cadena mágica* que identifica el tipo de fichero. En nuestro caso será la cadena **P3** y un separador (salto de línea). P3 indica que el fichero contiene una imagen en color con valores de luminosidad codificados en *texto*.
- Un número indeterminado de comentarios. Los comentarios son de línea completa. El primer carácter de una línea de comentario siempre será el carácter '#'.
- El ancho o número de columnas (*c*), un separador y el alto o número de filas (*f*), y un separador (salto de línea).
- El máximo nivel de luminosidad de la imagen (*m*) y un separador (salto de línea). Por simplicidad valdrá 255.
- Los $c \times f$ píxeles de la imagen, que son tripletas (componentes RGB) de números enteros en formato texto. Entre cada dos números enteros hay un separador. Los píxeles aparecen por filas en el fichero, empezando desde la fila superior. Así, el primer píxel corresponde al píxel de la esquina superior izquierda, el segundo al píxel que está a su derecha, etc. (el último corresponde al píxel de la esquina inferior derecha de la imagen). En definitiva, los píxeles se organizan según el orden de lectura (izquierda a derecha y de arriba abajo).

¹Vea su especificación por ejemplo en <https://netpbm.sourceforge.net/doc/ppm.html> o <http://sintesis.ugto.mx/WintemplaWeb/02Wintempla/10Images/07PPM%20Image/index.htm>

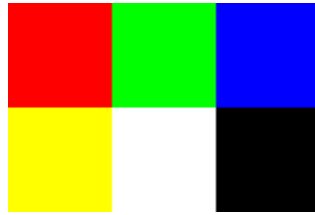


Figura 3: Imagen PPM de 3 columnas y 2 filas del ejemplo 1

Ejemplo 1 *Ejemplo de una imagen PPM de 3 columnas (ancho) y 2 filas (alto):*

```
P3
# "P3" means this is a PPM color image in ASCII
# "3 2" is the width and height of the image in pixels
# "255" is the maximum value for each color
# This, up through the "255" line below are the header.
# Everything after that is the image data: RGB triplets.
# In order the pixel of this image are: red, green, blue,
# yellow, white and black colors.
3 2
255
255 0 0 0 255 0 0 0 255
255 255 0 255 255 255 0 0 0
```

Aunque hemos mostrado el contenido del fichero PPM, poniendo en líneas separadas los píxeles de cada fila, sería también válido que apareciesen todas las filas en una sola línea. El fichero `sampleImage3x2.ppm` tiene el anterior contenido, que es una imagen con 6 píxeles, y puede verse de forma ampliada en la figura 3. Para su visualización use el siguiente comando:

```
linux> magick display -resize 500 -filter point data/sampleImage3x2.ppm
```

5. Práctica a entregar

5.1. Finalidad del programa

El programa de esta práctica tiene como propósito obtener un mapa de calor para el conjunto de crímenes ocurridos dentro de una determinada área de interés. El conjunto de crímenes se obtiene mediante la fusión de los crímenes contenidos en la lista de ficheros `crm` proporcionados al programa. El mapa de calor se grabará en un fichero de imagen a color en formato PPM (ver sección 4).

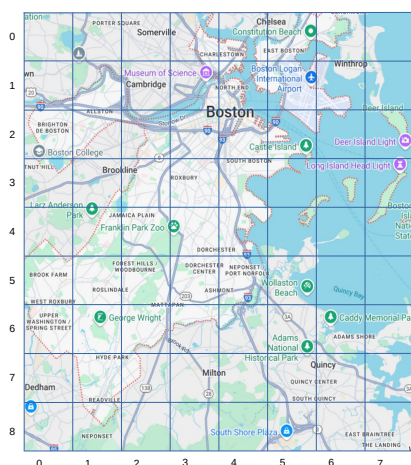
El área de interés se define como una zona rectangular a través de las coordenadas de la esquina inferior izquierda y la esquina superior derecha de tal rectángulo. El programa a construir usará para estas dos coordenadas los valores que toman las siguientes constantes estáticas definidas en la clase `CrimeCounter`: `DEFAULT_COORD_TOPRIGHT` y `DEFAULT_COORD_BOTTOMLEFT`. El valor que toman estas constantes se define en el fichero `CrimeCounter.cpp` (vea la sección 6.7), concretamente tales valores son (42,395042, -70,953728) y (42,207760, -71,178673) respectivamente.

Los nombres de ficheros `crim` de entrada son proporcionados al programa a través de la línea de comandos. El nombre del fichero PPM de salida se puede proporcionar también a través de la línea de comandos. Si no se proporciona, el fichero de salida será `tests/output/output.ppm`. Vea la sintaxis y los ejemplos de ejecución en la sección 5.2.

Un **mapa de calor** es una imagen en color en la que cada píxel representa un área de la ciudad y su color va a reflejar la frecuencia relativa de crímenes en tal área. Así, un punto muy caliente de la ciudad (donde tienen lugar muchos delitos) se destaca en tonalidades más rojas que otro donde la delincuencia es más baja. Para obtener el mapa de calor a partir de la lista de ficheros `crim` de entrada se sigue el proceso mostrado a continuación.

Descripción breve de los pasos a seguir

1. Obtener la fusión de los ficheros `crim` de entrada.
2. Dividir el área de interés en varias zonas rectangulares mediante una cuadrícula de un determinado número de filas y de columnas (figura 4a).
3. Calcular la frecuencia de los crímenes cuyas coordenadas geográficas caen dentro de cada rectángulo de la cuadrícula (figura 4b).
4. Se define una paleta de color con los colores deseados (figura 2).
5. Para cada valor en la matriz de frecuencias, obtener el índice de la paleta de color a que corresponde tal valor (figura 5a).
6. Obtener el fichero imagen PPM a partir de los índices de la paleta de color calculados en el paso anterior (figura 5b).



(a) División en rectángulos del área de interés

0	0	0	0	1742	6849	5639	1655	0
1	1395	5185	99	3557	31876	1972	0	0
2	4463	12732	9698	43806	21283	2048	0	6
3	7	39	16545	37512	16156	489	0	4
4	628	1019	8241	30481	21891	0	0	0
5	2817	8785	8730	26158	6175	0	0	0
6	1798	7581	7135	1352	0	0	0	0
7	0	2207	825	0	0	0	0	0
8	0	1	0	0	0	0	0	0
	0	1	2	3	4	5	6	7

(b) Frecuencias de crímenes

Figura 4: Obtención de frecuencias para cada zona del área de interés

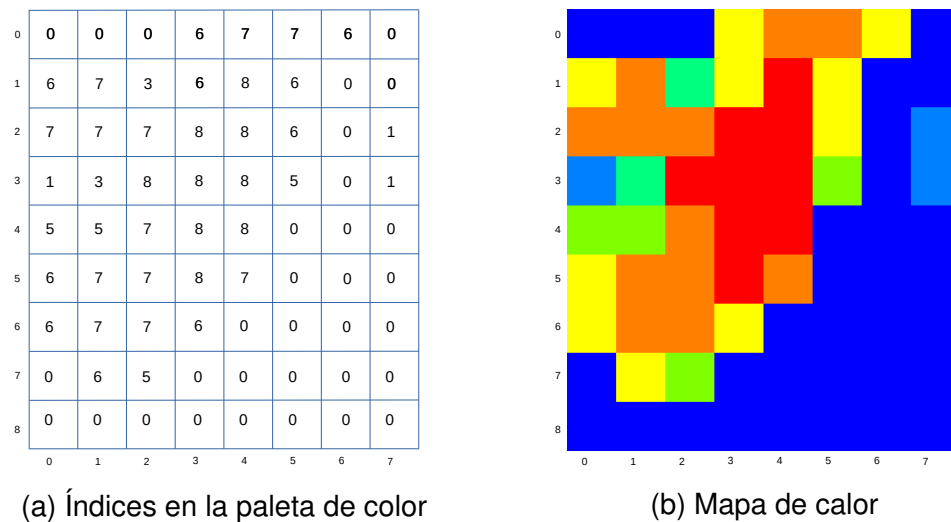


Figura 5: Obtención del mapa de calor

Como hemos indicado en la sección 4.3, un fichero PPM puede tener líneas de comentarios. Cada línea de comentario comienza con el carácter '#'. El programa debe guardar en el fichero PPM una línea de comentario con el contenido de los parámetros usados al ejecutar el programa (vea los ejemplos de ejecución en la sección 5.2). Líneas de comentario adicionales pueden ser añadidas si el usuario del programa usa la opción `-c` al ejecutarlo.

Describamos con un poco más de detalle los pasos anteriores, que serán los que lleve a cabo la función `main()` de `Boston4`.

Descripción detallada de los pasos a seguir

1. El programa comienza procesando los argumentos opcionales (los que empiezan con el carácter '-') pasados al programa. Estos argumentos deben aparecer al principio de la línea de comandos, aunque pueden hacerlo en cualquier orden. Se debe analizar también si el programa se ejecutó al menos con un fichero de entrada `crm`.
2. A continuación se obtiene un objeto `CrimeSet` con la fusión de los ficheros `crm` pasados a través de la línea de comandos.
3. Seguidamente, se creará un objeto `CrimeCounter` que divida el área de interés en un determinado número de filas y columnas.

Como se indicó al principio de esta sección, el área de interés será el rectángulo delimitado por las coordenadas cuyos valores se toman de las siguientes constantes: `DEFAULT_COORD_TOPRIGHT` y `DEFAULT_COORD_BOTTOMLEFT`. Fíjese, que el constructor de la clase `CrimeCounter` tiene dos parámetros para fijar el área de interés, y que son parámetros por defecto que toman precisamente los valores anteriores.

El número de filas y columnas usados por el objeto `CrimeCounter` tomarán los valores 19x23, aunque es posible utilizar otros valores con los parámetros correspondientes de la línea de comandos.

4. Usar el objeto `CrimeCounter` anterior para calcular la frecuencia de crímenes en cada zona en la que se divide el área de interés del objeto que contiene la fusión.
5. Crear un objeto `string` con los parámetros usados para ejecutar el programa desde la línea de comandos. Añadir otras líneas de comentario en caso de que se haya utilizado el parámetro `-c` en la línea de comandos. Usar la función `FormatAsComment()` para añadir el carácter '#' al principio de cada línea del objeto `string` de comentarios.
6. Crear un objeto `ColorPalette` con los colores deseados.
7. Llamar al método `CrimeCounter::saveAsPPMTextImage()` con el objeto `CrimeCounter` calculado anteriormente, pasándole el objeto `ColorPalette` anterior y el objeto `string` con los comentarios, para obtener un fichero imagen PPM con el mapa de calor. Este método calcula para cada frecuencia el índice que le corresponde en la paleta de color, y guarda las intensidades RGB del color correspondiente en el fichero de salida; o sea, el método realiza las tareas de los pasos 5 y 6 de la descripción breve explicada más arriba (figuras 5a y 5b).

La forma que tiene este método para obtener un píxel en color (un objeto `Color`) a partir de un valor de frecuencia (`frequency`) es la siguiente:

```
int colorIndex = GetIndexPalette(palette, log(frequency+1),  
    ↪ log(maxFrequency+1));  
Color color = palette.getColor(colorIndex);
```

En el trozo de código anterior, `maxFrequency` es el mayor valor encontrado en la matriz de frecuencias.

5.2. Sintaxis y ejemplos de ejecución

Este programa tiene un gran número de parámetros opcionales que pueden pasarse a través de la línea de comandos. Tales parámetros son los primeros que aparecen, y podrían hacerlo en cualquier orden. Tras los parámetros opcionales aparece al menos un fichero `crm` de entrada, pudiendo aparecer otros ficheros `crm` adicionales.

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

```
linux> dist/Debug/GNU-Linux/boston4 [-o outputFilename] [-c text]  
    [-h int] [-w int] [-d int] [-b int] [-g int] [-y int]  
    <inputFile1.crm> {<inputFile.crm>}
```

La explicación de cada uno de los argumentos del programa es la siguiente:

- **-o outputFilename**: nombre del fichero PPM de salida, siendo `tests/output/output.ppm` el nombre por defecto si no se usa este parámetro.



- **-c text**: comentarios adicionales a incluir en el fichero PPM de salida. Por defecto, solo se incluyen la lista de argumentos de la línea de comandos. Si **text** contiene más de una palabra, es necesario encerrarlo entre comillas para que el SO lo considere un solo argumento de la línea de comandos.
- **-h high**: número de filas que usará la matriz de frecuencias (19 por defecto).
- **-w width**: número de columnas que usará la matriz de frecuencias (23 por defecto).
- **-d number**: número de tonos en azul oscuro a incluir en la paleta de color (2 por defecto).
- **-b number**: número de tonos en azul claro a incluir en la paleta de color (2 por defecto).
- **-g number**: número de tonos en verde a incluir en la paleta de color (2 por defecto).
- **-y number**: número de tonos en amarillo a incluir en la paleta de color (2 por defecto).
- **<inputFile1.crm> {<inputFile.crm>}**: ficheros **crm** de entrada. Debe haber al menos uno.

Nota: Los parámetros opcionales (los que comienzan con -) se pueden introducir en cualquier orden, pero deben estar al principio, antes de la lista de ficheros de entrada.

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

Ejemplo 2 *Faltan argumentos para la ejecución del programa. Se necesita ejecutar con al menos un fichero de entrada:*

```
linux> dist/Debug/GNU-Linux/boston4
```

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

```
ERROR in boston4 parameters
Run with the following arguments:
boston4 [-o outputFilename] [-c text] [-h int] [-w int] [-d int] [-b int] [-g
↵ int] [-y int] <inputFile1.crm> {<inputFile.crm>}

Parameters:
-o outputFilename: name of the output PPM file (tests/output/output.ppm by
↵ default)
-c text: extra comment to be included in the output PPM file (list of program
↵ parameters, by default)
-h high: number of rows (19 by default)
-w width: number of columns (23 by default)
-d number: number of dark blue tones in the color palette (2 by default)
-b number: number of blue tones in the color palette (2 by default)
-g number: number of green tones in the color palette (2 by default)
-y number: number of yellow tones in the color palette (2 by default)
<inputFile1.crm> {<inputFile.crm>}: input crm files
```



Ejemplo 3 Ejecución con parámetro opcional erróneo:

```
linux> dist/Debug/GNU-Linux/boston4 -f ../DataSets/crimes_easy01.crm  
↪ ../DataSets/crimes_easy01.crm
```

En este caso, el argumento `-f` no es válido para *Boston4*. 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. El mapa de calor que se quiere obtener tendrá 2 filas y 3 columnas:

```
linux> dist/Debug/GNU-Linux/boston4 -h 2 -w 3 ../DataSets/crimes_05.crm
```

En la anterior ejecución, solo se proporciona el fichero de entrada (`../DataSets/crimes_05.crm`), con lo que el mapa de calor resultado tendrá solo los crímenes de este fichero (5 crímenes). 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 con el mapa de calor obtenido con la anterior ejecución es `tests/output/output.ppm`. Si no tenía creada la carpeta `tests/output`, el programa no podrá grabar el fichero de salida en ella, y lanzará una excepción. El contenido del fichero `tests/output/output.crm` tendría que ser el siguiente:

Fichero `tests/output/output.ppm`:

```
P3  
#Running parameters: -h 2 -w 3 ../DataSets/crimes_05.crm  
3 2  
255  
0 0 255 255 0 0 0 0 255 0 0 255 0 0 255 0 0 255
```

El anterior fichero es una imagen PPM de 2 filas y 3 columnas, que por tanto tendrá 6 píxeles. Al ser una imagen tan pequeña, si la visualiza con un visor de imágenes no podrá ver prácticamente nada a no ser que haga un zoom. Para ello puede emplear por ejemplo el siguiente comando (*ImageMagick*):

```
linux> magick display -resize 500 -filter point tests/output/output.ppm
```

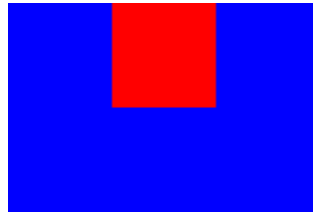



Figura 6: Mapa de calor con 2 filas y 3 columnas del ejemplo 4

La imagen del mapa de calor que se mostrará con el anterior comando puede verse en la figura 6.

Ejemplo 5 Ejecución proporcionando la opción `-c` para incluir comentarios adicionales en el fichero PPM de salida.

```
linux> dist/Debug/GNU-Linux/boston4 -c "Primera línea de comentario adicional"
↪ -c "Segunda línea de comentario adicional" -h 2 -w 3
↪ ../DataSets/crimes_05.crm
```

De nuevo, el nombre del fichero con el mapa de calor obtenido con la anterior ejecución es `tests/output/output.ppm`. El mapa de calor obtenido es el mismo que en el ejemplo 4 (figura 6), pero ahora el fichero PPM obtenido contiene dos líneas de comentarios adicionales. El contenido de este fichero tendría que ser el siguiente:

Fichero `tests/output/output.ppm`:

```
P3
#Running parameters: -c Primera línea de comentario adicional -c Segunda línea
↪ de comentario adicional -h 2 -w 3 ../DataSets/crimes_05.crm
#Primera línea de comentario adicional
#Segunda línea de comentario adicional
3 2
255
0 0 255 255 0 0 0 0 255 0 0 255 0 0 255 0 0 255
```

Ejemplo 6 Ejecución proporcionando dos ficheros de entrada. El número de filas y columnas del mapa de calor toma los valores por defecto (19x23):

```
linux> dist/Debug/GNU-Linux/boston4 ../DataSets/crimes_01.crm
↪ ../DataSets/crimes_05.crm
```

El nombre del fichero de salida obtenido con la anterior ejecución es de nuevo `tests/output/output.ppm`. Usando el programa `magick` para visualizar este fichero obtendríamos la imagen de la figura 7.

Ejemplo 7 Ejecución proporcionando dos ficheros de entrada y el de salida:

```
linux> dist/Debug/GNU-Linux/boston4 -o /tmp/output.ppm
↪ ../DataSets/crimes_01.crm ../DataSets/crimes_05.crm
```

El fichero de salida es en este caso `/tmp/output.ppm`. El contenido de este fichero debe ser el mismo que el obtenido con el ejemplo anterior (ejemplo 6).

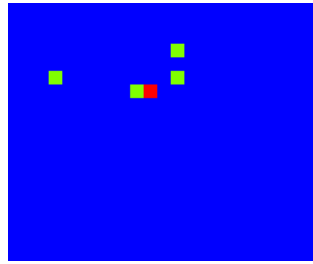


Figura 7: Mapa de calor con 19 filas y 23 columnas del ejemplo 6

Ejemplo 8 *Ejecución con el fichero crimes_all.crm que contiene información de 446093 crímenes. Se guarda el mapa de calor en el fichero /tmp/crimes_all.ppm.*

```
linux> dist/Debug/GNU-Linux/boston4 -h 37 -w 45 -o /tmp/crimes_all.ppm  
↪ ../DataSets/crimes_all.crm
```

Podemos mostrar el mapa de calor obtenido con:

```
linux> magick display -resize 500 -filter point /tmp/crimes_all.ppm
```

La imagen del mapa de calor que se mostrará con el anterior comando puede verse en la figura 8.

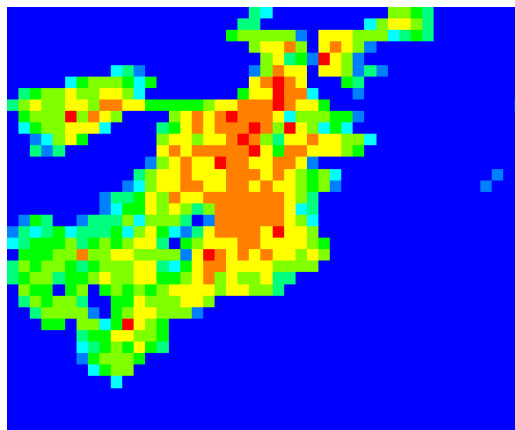


Figura 8: Mapa de calor con 37 filas y 45 columnas para el fichero crimes_all.crm

Ejemplo 9 *Ejecución con el fichero crimes_all.crm usando 4 tonos de amarillo en la paleta de color. Se guarda el mapa de calor en el fichero /tmp/crimes_all_y4.ppm.*

```
linux> dist/Debug/GNU-Linux/boston4 -y 4 -h 37 -w 45 -o /tmp/crimes_all_y4.ppm  
↪ ../DataSets/crimes_all.crm
```

Podemos mostrar el mapa de calor obtenido con:

```
linux> magick display -resize 500 -filter point /tmp/crimes_all_y4.ppm
```

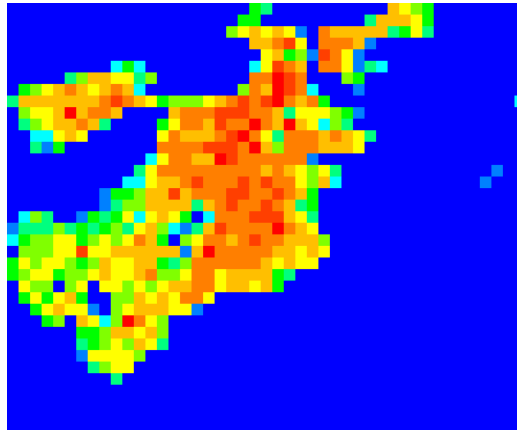


Figura 9: Mapa de calor con 37 filas y 45 columnas para el fichero `crimes_all.crm` usando 4 tonos de amarillo en la paleta de color

La imagen del mapa de calor que se mostrará con el anterior comando puede verse en la figura 9.

En la carpeta `tests` del proyecto `Boston4`, 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 `.ppm` y cuyo nombre es `outcome+nº`. Cada uno de estos ficheros `.ppm` corresponde con el que debería obtenerse ejecutando el programa con el correspondiente fichero de la carpeta `tests`. Así por ejemplo, el contenido de `data/outcome06.ppm` 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 detalladas sus especificaciones.

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

- Clase `Coordinates`. En esta clase solo debemos incorporar la sobrecarga de los operadores de entrada (`>>`) y salida (`<<`).
- Clase `Crime`. En esta clase debemos incorporar la sobrecarga de los operadores de entrada (`>>`) y salida (`<<`) y la sobrecarga de todos los operadores relacionales (`<`, `<=`, `==`, `>`, `>=`, `!=`). En el caso de los operadores relacionales, la comparación entre dos objetos `Crime` se hace en base al campo `DateTime` y a igualdad de este, en base al campo `ID`.
- Clase `CrimeSet`. En esta clase debemos incorporar:

- Sobrecarga de los operadores de entrada (>>) y salida (<<). Hay que tener cuidado de que el operador de entrada elimine los crímenes que contenía el objeto previamente.
 - Sobrecarga del operador [] tanto de consulta como de modificación. Fíjese en el fichero `CrimeSet.h` que la versión de modificación de este operador se pone en la sección privada de la clase. Esto se hace para evitar que un usuario de la clase pueda por ejemplo insertar un crimen con el mismo ID que otro crimen contenido en el objeto `CrimeSet`.
 - Sobrecarga del operador += para añadir al objeto implícito los crímenes del objeto pasado como parámetro (solo los que no estén ya en el objeto implícito).
- Clase `CrimeCounter`. Nueva clase que contiene una matriz dinámica de enteros usada en el conteo de frecuencias de crímenes en las áreas en que se divida la zona de interés. El constructor de esta clase crea una matriz de frecuencias con el número de filas y columnas deseado. Una vez creada, no hay ningún método que permita cambiar su tamaño. Describamos brevemente los datos miembro de esta clase:
- `_nRows`: número de filas de la matriz de frecuencias.
 - `_nCols`: número de columnas de la matriz de frecuencias.
 - `_frequency`: puntero a la matriz de frecuencias.
 - `_bottomLeftCoord`: coordenadas de la esquina inferior izquierda del área de interés.
 - `_topRightCoord`: coordenadas de la esquina superior derecha del área de interés.
 - `DEFAULT_COORD_BOTTOMLEFT`: objeto `Coordinates` público y constante usado como valor por defecto en el constructor de la clase para establecer las coordenadas de la esquina inferior izquierda del área de interés.
 - `DEFAULT_COORD_TOPRIGHT`: objeto `Coordinates` público y constante usado como valor por defecto en el constructor de la clase para establecer las coordenadas de la esquina superior derecha del área de interés. Este objeto y el anterior es inicializado en el fichero `CrimeCounter.cpp`.
 - `DEFAULT_ROWS`: constante privada usada como valor por defecto en el constructor de la clase para establecer el número de filas de la matriz de frecuencias.
 - `DEFAULT_COLUMNS`: constante privada usada como valor por defecto en el constructor de la clase para establecer el número de columnas de la matriz de frecuencias.



```
class CrimeCounter {
public:
    /**
     * Constant with the default Coordinates for the bottom left corner of the
     * input geographical area. It is set to the bottom left corner of the
     * Boston area
     */
    static const Coordinates DEFAULT.COORD.BOTTOMLEFT;

    /**
     * Constant with the default Coordinates for the top right corner of the
     * input geographical area. It is set to the top right corner of the
     * Boston area
     */
    static const Coordinates DEFAULT.COORD.TOPRIGHT;

    ...

private:
    /**
     * Integer constant with the default number of rows in the matrix of
     * frequencies
     */
    static const int DEFAULT.ROWS = 15;

    /**
     * Integer constant with the default number of columns in the matrix of
     * frequencies
     */
    static const int DEFAULT.COLUMNS = 10;

    int** _frequency; ///< 2D matrix with the crime frequency for each grid cell
    int _nRows; ///< number of rows of the 2D matrix of frequencies
    int _nCols; ///< number of columns of the 2D matrix of frequencies

    /**
     * Coordinates that defines the bottom left corner of the input geographical
     * area
     */
    Coordinates _bottomLeftCoord;

    /**
     * Coordinates that defines the top right corner of the input geographical
     * area
     */
    Coordinates _topRightCoord;

    ...
}; // end class CrimeCounter
```

- Las clases `Color` y `ColorPalette` han sido introducidas en la sección 4. Como se indicó en la sección 2, estas clases se proporcionan completamente definidas y no necesitan de ninguna modificación.
- 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 6.8), 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 `Boston4.zip`, sin más aditivos. Como alternativa, se sugiere utilizar el script `runZipProject.sh` que debe estar en la carpeta `scripts` de `Boston4`, junto con otras utilidades.



6. Código para la práctica

6.1. Color.h

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

/**
 * @file: Color.h
 * @author Silvia Acid Carrillo <acid@decsai.ugr.es>
 * @author Andrés Cano Utrera <acu@decsai.ugr.es>
 * @author Luis Castillo Vidal <L.Castillo@decsai.ugr.es>
 *
 * Created on October 2, 2024, 10:48AM
 */

#ifndef COLOR_H
#define COLOR_H

#include <iostream>

/**
 * @class Color
 * @brief A color object Color(red, green, blue) is used to encapsulate colors
 * in the default RGB color space. An RGB color value is specified with
 * 3 components (alpha components): red, green and blue. Each component is
 * represented using and unsigned char value (alpha value) in the range 0 – 255.
 * Value of 255 means that the basic color is completely
 * opaque and an alpha value of 0 means that the color is completely transparent.
 * For example:
 * - Color(255, 0, 0) represents red color, that is, red is set to
 * its highest value (255), and the other two (green and blue) are set to 0
 * - Color(0, 255, 0) represents green color
 * - Color(0, 0, 255) represents blue color
 * - Color(0, 0, 0) represents black color
 * - Color(255, 255, 255) represents white color
 */
class Color {
public:
    /**
     * @brief Creates a RGB color with the specified red, green, and
     * blue values, each one in the range (0 – 255).
     * Each one of the three components (red, green and blue) is 255 by default.
     * @param red the red component. Input parameter
     * @param green the green component. Input parameter
     * @param blue the blue component. Input parameter
     * @throw std::invalid_argument This method throws an std::invalid_argument
     * exception if red, green or blue are outside of the range [0..255],
     * both inclusive.
     */
    Color(int red = DEFAULT_RGB_VALUE, int green = DEFAULT_RGB_VALUE,
          int blue = DEFAULT_RGB_VALUE);

    /**
     * @brief Returns the red component in the range 0–255 in the default RGB space.
     * Query method
     * @return The red component of this Color
     */
    unsigned char getRed() const;

    /**
     * @brief Returns the green component in the range 0–255 in the default RGB space.
     * Query method
     * @return The green component of this Color
     */
    unsigned char getGreen() const;

    /**
     * @brief Returns the blue component in the range 0–255 in the default RGB space.
     * Query method
     * @return The blue component of this Color
     */
    unsigned char getBlue() const;

    /**
     * @brief Returns a string representation of this Color with the three
     * component values separated by a whitespace.
     * Query method
     * @return the string composed
     */
    std::string toString() const;

private:
    /**
     * Constant with the default value for each RGB value
     */
    static const unsigned char DEFAULT_RGB_VALUE=255;

    unsigned char _r; ///< Red component
    unsigned char _g; ///< Green component
    unsigned char _b; ///< Blue component

    /**
     * @brief function that verifies that an alpha value is in the correct
     * domain.
     * Query method
     * @param v any of the 3 components of the Color object.
     */
};
```



```
    * @return true when v in the correct range; false otherwise
    */
    bool isValid(int v);
}; // end class Color

#endif /* COLOR.H */
```



6.2. ColorPalette.h

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

/**
 * @file: ColorPalette.h
 * @author Silvia Acid Carrillo <acid@decsai.ugr.es>
 * @author Andrés Cano Utrera <acu@decsai.ugr.es>
 * @author Luis Castillo Vidal <L.Castillo@decsai.ugr.es>
 * @
 *
 * Created on October 2, 2024, 1:20PM
 */

#ifndef COLORPALETTE.H
#define COLORPALETTE.H

#include "Color.h"

/**
 * @class ColorPalette
 * @brief A palette contains a list of Colors.
 * The maximum number of colors in the list is MAX.NUMBER.COLORS.
 */
class ColorPalette {
public:
    /**
     * @brief Builds a ColorPalette with a list of 4 intervals of "primary"
     * colors and the red color (Color(255, 0, 0)).
     * This constructor has 4 integer parameters to control the
     * number of colors in each interval:
     * -# \p da: number of colors from dark blue Color(0, 0, 255) (included)
     * to light blue Color(0, 255, 255) (not included).
     * -# \p bl: number of colors from light blue Color(0, 255, 255) (included)
     * to green Color(0, 255, 0) (not included).
     * -# \p gr: number of colors from green Color(0, 255, 0) (included)
     * to yellow Color(255, 255, 0) (not included).
     * -# \p ye: number of colors from yellow Color(255, 255, 0) (included)
     * to red Color(255, 0, 0) (not included).
     *
     * The total number of colors in the palette is the addition of all the
     * tones in each interval plus 1 (red color).
     *
     * Examples:
     * - ColorPalette(0,0,0,0) has only one color, the red one Color(255, 0, 0)
     * - ColorPalette(1,1,1,1) has got 5 colors:
     * Color(0, 0, 255), Color(0, 255, 255), Color(0, 255, 0),
     * Color(255, 255, 0) and Color(255, 0, 0).
     * - ColorPalette(3,0,0,0) has 4 colors: Color(0, 0, 255),
     * Color(0, 85, 255), Color(0, 170, 255) and Color(255, 0, 0).
     *
     * @note If the total number of colors in the new palette is greater than
     * ColorPalette::MAX.NUMBER.COLORS or any of the provided parameters is
     * negative this constructor builds a palette using the values
     * da=2, bl=2, gr=2, ye=2
     *
     * @param da number of tones from dark blue (included) to light blue
     * (not included)
     * @param bl number of tones from light blue (included) to green
     * (not included)
     * @param gr number of tones from green (included) to yellow (not included)
     * @param ye number of tones from yellow (included) to red (not included)
     */
    ColorPalette(int da=2, int bl=2, int gr=2, int ye=2);

    /**
     * @brief Returns the number of colors into the palette
     * (a value among 1 and MAX.NUMBER.COLORS).
     * Query method
     * @return number of colors into the palette.
     */
    int getNumColors() const;

    /**
     * @brief Gets a const reference to the Color at the given position
     * A value between [0.. numColors[ .
     * Query method
     * @param index the position to consider. Input parameter
     * @throw std::out_of_range Throws an std::out_of_range exception if the
     * given index is not valid
     * @return A const reference to the Color at the given position.
     */
    const Color & getColor(int index) const;

    /**
     * @brief Returns a string composed one by one by the Colors stored into
     * the palette separated by "\n".
     * Query method
     * @return A string with the list of Colors into the palette separated by
     * "\n".
     */
    std::string toString() const;
private:
    static const int MAX.NUMBER.COLORS =100; ///< The capacity of the array _palette
    Color _palette[MAX.NUMBER.COLORS]; ///< array with the list of colors
};
```



```
int _numColors; ///< Number of used elements in _palette

/**
 * Private method to fill colors in the palette. See
 * @ref ColorPalette(int, int, int, int) for more details.
 * @throw std::invalid_argument Throws an std::invalid_argument exception
 * if the total number of colors in the palette would be greater than
 * ColorPalette::MAX_NUMBER_COLORS. In that case, the total number of
 * colors in the palette will be set to 0.
 * @throw std::invalid_argument Throws an std::invalid_argument exception
 * if the any of the provided parameters is negative. In that case, the
 * total number of colors in the palette will be set to 0.
 */
void init(int nColorsFromBlue, int nColorsFromLightBlue,
          int nColorsFromGreen, int nColorsFromYellow);
}; // end class ColorPalette

#endif /* COLORPALETTE.H */
```




6.3. Coordinates.h

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

/**
 * @file Coordinate.h
 * @author Silvia Acid Carrillo <acid@decsai.ugr.es>
 * @author Andrés Cano Utrera <acu@decsai.ugr.es>
 * @author Luis Castillo Vidal <L.Castillo@decsai.ugr.es>
 *
 * Created on September 25, 2024, 1:03PM
 */

#ifndef COORDINATES.H
#define COORDINATES.H

/**
 * @class Coordinates
 * @brief GPS Coordinates (geographic coordinates) for measuring positions in
 * the Earth. It consists of a pair of float values (latitude and longitude)
 * that express angles (degrees). See for example
 * <a href="https://en.wikipedia.org/wiki/Geographic_coordinate_system">
 * Wikipedia Geographic coordinate system</a> <br>
 * Values for latitude are between -90 and 90, while values for longitude
 * must be between -180 and 180. It is possible to build a Coordinates object
 * with a latitude or longitude out of that ranges, but that object will be
 * considered as invalid (method \ref isValid() will return true).<br>
 * The value latitude == 0 is the EQUATOR, and the value longitude == 0 is the
 * Greenwich meridian. <br>
 *
 * - A point with a positive value for latitude is located in the northern
 * hemisphere; a point with latitude==90 is in the North pole. <br>
 * - A point with a negative latitude is located in the southern hemisphere. <br>
 * - A point with a positive longitude is located to the right of the Greenwich
 * meridian. <br>
 * - A point with a negative longitude is located to the left of the Greenwich
 * meridian. <br>
 * <br>
 * Examples: <br>
 * - Granada: 37.178056,-3.600833
 * - Valencia: 39.47,-0.376389
 * - Boston: 42.360278,-71.057778
 * - Quito: -0.22, -78.5125
 * - Johannesburgo: -26.204444, 28.045556
 * <br>
 */

#include <iostream>
#include <string>

class Coordinates {
public:
    // Retrieve all previous declarations and take into account
    // the new declarations included here...
    //

    friend std::ostream operator<<(std::ostream os, Coordinates obj);
    friend std::ostream operator>>(std::ostream is, Coordinates obj);

private:
    const static float INVALID_COORDINATE; // ...
    // Retrieve all previous declarations
};

/**
 * @brief Overloading of the stream insertion operator for Coordinates class.
 * Sends to the output stream the latitude and longitude converted to strings,
 * separated by a comma. It must use the std::to_string() function to
 * convert latitude and longitude to strings.
 * @param os The output stream to be used. Input/output parameter
 * @param obj The Coordinates object. Input parameter
 * @return @p os A reference to the output stream
 */
std::ostream operator<<(std::ostream os, Coordinates obj);

/**
 * @brief Overloading of the stream extraction operator for Coordinates class.
 * First it reads from the input stream the latitude, then a character
 * (the comma character) and finally the longitude. Those two values (float
 * numbers) will be used to set the latitude and longitude of the given
 * Coordinates object.
 * @param is The input stream to be used. Input/output parameter
 * @param obj The Coordinates object. Output parameter
 * @return @p is A reference to the input stream
 */
std::istream operator>>(std::istream is, Coordinates obj);
```



6.4. Crime.h

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

/**
 * @file Crime.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 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.DESCRPTION: 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.DESCRPTION 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:
    // Retrieve all previous declarations and take into account
    // the new declarations included here...
    // ...

    friend std::ostream operator<<(std::ostream os, Crime crime);
    friend std::istream operator>>(std::istream is, Crime crime);

private:
    // Retrieve all previous declarations
};

// Retrieve all previous declarations of external functions: Trim, Capitalize
// and Normalize

/**
 * @brief Overloading of the stream insertion operator for Crime class.
 * It inserts every field of the object separated by commas (,) in
 * the output string.
 * @param os The output stream to be used. Input/Output parameter
 * @param crime the Crime object. Input parameter
 * @return @p os A reference to the output stream
 */
std::ostream operator<<(std::ostream os, Crime crime);

/**
 * @brief Overloading of the stream extraction operator for Crime class. It
 */
```



```
* reads a record from the input string that will set the
* list of fields of the provided crime object.
* @param is The input stream to be used. Input/Output parameter
* @param crime the Crime object. Input/Output parameter
* @return @p is the input stream
*/
std::istream operator>>(std::istream is, Crime crime);

/**
 * @brief Overloading of the relational operator < for Crime class
 * @param crime1 The first object to be compared. Input parameter
 * @param crime2 The second object to be compared. Input parameter
 * @return true if the DateTime of @p crime1 is smaller (before) than that of
 * @p crime2 or if both DateTimes are equals and the ID of
 * @p crime1 is minor than the ID of @p crime2; false otherwise
 */
bool operator<(Crime crime1, Crime crime2);

/**
 * @brief Overloading of the operator > for Crime class. It uses the DateTime
 * and ID of the given crimes to compare them as in operator<
 * @param crime1 a Crime object. Input parameter
 * @param crime2 a Crime object. Input parameter
 * @return true if crime1 > crime2; false otherwise
 */
bool operator>(Crime crime1, Crime crime2);

/**
 * @brief Overloading of the operator == for Crime class. It uses the DateTime
 * and ID of the given crimes to compare them as in operator<
 * @param crime1 a Crime object. Input parameter
 * @param crime2 a Crime object. Input parameter
 * @return true if the two Crimes have the same DateTime and ID;
 * false otherwise
 */
bool operator==(Crime crime1, Crime crime2);

/**
 * @brief Overloading of the operator != for Crime class. It uses the DateTime
 * and ID of the given crimes to compare them as in operator<
 * @param crime1 a Crime object. Input parameter
 * @param crime2 a Crime object. Input parameter
 * @return true if the two Crimes are not equals (see operator==);
 * false otherwise
 */
bool operator!=(Crime crime1, Crime crime2);

/**
 * @brief Overloading of the operator <= for Crime class. It uses the DateTime
 * and ID of the given crimes to compare them as in operator<
 * @param crime1 a Crime object. Input parameter
 * @param crime2 a Crime object. Input parameter
 * @return true if crime1 <= crime2; false otherwise
 */
bool operator<=(Crime crime1, Crime crime2);

/**
 * @brief Overloading of the operator >= for Crime class. It uses the DateTime
 * and ID of the given crimes to compare them as in operator<
 * @param crime1 a Crime object. Input parameter
 * @param crime2 a Crime object. Input parameter
 * @return true if crime1 >= crime2; false otherwise
 */
bool operator>=(Crime crime1, Crime crime2);

#endif /* CRIME_H */
```



6.5. 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
 */

#ifndef CRIMESSET_H
#define CRIMESSET_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 Overloading of the [] operator for CrimeSet class
     * Query method
     * @param index index of the element
     * @return Aant reference to the Crime object at position @p index
     */
    Crime operator[](int index);

    /**
     * @brief Overloads the operator += for the CrimeSet class. It appends to
     * this CrimeSet object, the list of Crime objects contained in the
     * provided CrimeSet @p crimeSet that are not found in this CrimeSet.
     * Modifier method
     * @param other The right CrimeSet operand. Input parameter
     * @return A reference to this object
     */
    CrimeSet operator+=(CrimeSet other);

    friend std::ostream operator<<(std::ostream os, CrimeSet crimeSet);
    friend std::istream operator>>(std::istream is, CrimeSet crimeSet);

private:
    /**
     * @brief Overloading of the [] operator for CrimeSet class
     * Modifier method
     * @param index index of the element
     * @return A reference to the Crime object at position @p index
     */
    Crime operator[](int index);

    // Retrieve all previous declarations
};

/**
 * @brief Overloading of the stream insertion operator for CrimeSet class.
 * @note This operator sends each Crime to the output stream in the same
 * format as the one described in the Crime::toString() method
 * @param os The output stream to be used. Input/output parameter
 * @param crimeSet the CrimeSet object. Input parameter
 * @return @p os a reference to the output stream
 */
std::ostream operator<<(std::ostream os, const CrimeSet crimeSet);
```



```
/**
 * @brief Overloading of the stream extraction operator for CrimeSet class.
 * @note This operator should remove any crime previously contained in the
 * provided CrimeSet @p crimeSet.
 * @note This operator throws an exception in some error cases (see below).
 * Before throwing the corresponding exception, this operator clears
 * the provided object (it calls to clear() method) to leave it in a
 * consistent state.
 * @note This operator does not read the lines of comments, only the information
 * about crimes (an integer n with the number of crimes and n lines with Crime
 * information).
 * @throw std::out_of_range Throws a std::out_of_range if the number of Crimes
 * read from the file is negative.
 * @param is The input stream to be used. Input/output parameter
 * @param crimeSet The CrimeSet object to be filled. Input/output parameter
 * @return @p is A reference to the input stream
 */
std::istream operator>>(std::istream is, CrimeSet crimeSet);
```



6.6. CrimeCounter.h

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

/**
 * @file: CrimeCounter.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
 */

#ifndef CRIMECOUNTER_H
#define CRIMECOUNTER_H

#include <cstring>
#include <string>
#include "CrimeSet.h"
#include "ColorPalette.h"

/**
 * @class CrimeCounter
 * @brief It is a helper class used to calculate the number of crimes
 * (frequency) occurring within each one of the smaller areas (cells) in which
 * is divided a given target geographical area of the world. The target
 * geographical area is determined by the next two fields in this
 * class (Coordinates objects):
 * - _bottomLeftCoord: Bottom left corner of the target geographical area
 * - _topRightCoord: Top right corner of the target geographical area
 *
 * The target geographical area is divided into smaller rectangular areas (cells)
 * using a grid of nRows x nColumns cells.
 * This class uses a dynamic two-dimensional matrix of integers to store
 * the frequencies for each cell of the grid. Each integer in this matrix
 * allows to save the frequency of one of the cells.
 * The field _frequency of this class is a pointer to that matrix.
 */
class CrimeCounter {
public:
    /**
     * Constant with the default Coordinates for the bottom left corner of the
     * input geographical area. It is set to the bottom left corner of the
     * Boston area
     */
    static const Coordinates DEFAULT_COORD_BOTTOMLEFT;

    /**
     * Constant with the default Coordinates for the top right corner of the
     * input geographical area. It is set to the top right corner of the
     * Boston area
     */
    static const Coordinates DEFAULT_COORD_TOPRIGHT;

    /**
     * @brief Constructor of the class. It allocates memory for the
     * two-dimensional matrix of frequencies and sets the Coordinates
     * (bottom left corner and top right corner) that defines the target
     * geographical area.
     * Each element of the matrix will be initialized with 0 in this constructor.
     * If the provided number of rows or columns is less or equals to zero,
     * the matrix will be set with 0 rows and columns.
     *
     * This constructor uses default values for the parameters
     * @p bottomLeftCoord and @p topRightCoord that have been taken from the
     * border coordinates of the dataset for Boston city (1). That full
     * dataset is the source of the different datasets used in this course.
     * (1) (csv file located in
     * https://www.kaggle.com/datasets/shivamnegi1993/boston-crime-dataset-2022/data).
     *
     * @pre bottomLeftCoord.latitude < topRightCoord.latitude
     * and bottomLeftCoord.longitude < topRightCoord.longitude in order to
     * calculate correctly the area set by the two Coordinates; otherwise
     * incorrect calculations could be obtained with this class. This constructor
     * does not check this precondition.
     * @throw std::invalid_argument Throws an std::invalid_argument exception
     * if any of the provided coordinates is invalid. In this case, the
     * fields of the object are initialized as follows:
     * - _nRows=0
     * - _nCols=0
     * - _frequency=nullptr
     * - _bottomLeftCoord and _topRightCoord fields are left with the
     * default value assigned by the default constructor of the
     * class Coordinates;
     * After the fields are initialized as described above, the exception
     * is thrown.
     *
     * @param nRows Number of rows for the frequency matrix. Input parameter
     * @param nColumns Number of columns for the frequency matrix. Input parameter.
     * @param bottomLeftCoord Coordinates of the bottom left corner of the
     * target geographical area. Input parameter
     * @param topRightCoord Coordinates of the top right corner of the
     * target geographical area. Input parameter
     */
    CrimeCounter(int nRows = DEFAULT_ROWS, int nColumns = DEFAULT_COLUMNS,
                 Coordinates bottomLeftCoord = DEFAULT_COORD_BOTTOMLEFT,
```



```
Coordinates topRightCoord = DEFAULT.COORD.TOPRIGHT);

/**
 * @brief Copy constructor. Builds an exact copy of the provided
 * object @p orig.
 * @param orig object used as source for the copy. Input parameter
 */
CrimeCounter(CrimeCounter orig);

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

/**
 * @brief Overloads the assignment operator for CrimeCounter class.
 * Modifier method
 * @param orig the CrimeCounter object used as source for the assignment.
 * Input parameter
 * @return A reference to this object
 */
CrimeCounter operator=(CrimeCounter orig);

/**
 * @brief Returns the numbers of rows of the frequency matrix in
 * this object.
 * Query method
 * @return The numbers of rows of the frequency matrix in this object
 */
int getNumRows();

/**
 * @brief Returns the numbers of columns of the frequency matrix in
 * this object.
 * Query method
 * @return The numbers of columns of the frequency matrix in this object
 */
int getNumCols();

/**
 * @brief Returns the bottom left corner (Coordinates) of the target
 * geographical area of this object.
 * Query method
 * @return Returns the bottom left corner (Coordinates) of the target
 * geographical area of this object.
 */
Coordinates getLeftLocation();

/**
 * @brief Returns the top right corner (Coordinates) of the target
 * geographical area of this object.
 * Query method
 * @return The top right corner (Coordinates) of the target
 * geographical area of this object.
 */
Coordinates getRightLocation();

/**
 * @brief Returns a string with the content of this object.
 * This string is as follows:
 * - First line, the content of the private data _nRows and _nCols separated
 * by a white space
 * - Second line, the _bottomLeftCoord coordinates (using comma between
 * latitude and longitude)
 * - Third line, the _topRightCoord coordinates (using comma between
 * latitude and longitude)
 * - Next, a line for each row of the frequency matrix with the content
 * of that row (white space between values).
 * Query method
 * @return A string with the content of this object
 */
std::string toString();

/**
 * @brief Returns the maximum number of crimes (frequency) located in the
 * frequency matrix of this object.
 * Query method
 * @return the maximum frequency found in the 2D matrix. Returns 0 if the
 * matrix has 0 rows and columns.
 */
int getMaxFrequency();

/**
 * @brief Returns the number of counted crimes, that is the sum of
 * the frequencies in the frequency matrix.
 * @note That number includes only those crimes with valid coordinates
 * located inside the target geographical area.
 * Query method
 * @return the number of total crimes
 */
int getTotalLocated();

/**
 * @brief Sets all values in the frequency matrix to 0.
 * Modifier method
 */
void clear();

/**
 * @brief Increases with the value @p frequency the current frequency of
```



```
* the cell of the frequency matrix that corresponds to the location of
* the given Crime.
* If the coordinates of the given Crime are not valid or the Crime
* is not inside the area of the target geographical area of this object,
* this method performs no action (does not increase any frequency).
* If @p frequency is 0 or frequency is not provided, then 1 is added
* to the current frequency of the selected grid cell.
* Modifier method
* @throw std::invalid_argument The method throws an
* std::invalid_argument exception when the given crime Coordinates are
* valid but outside the target area.
* @param crime The crime to consider. Input parameter
* @param frequency The quantity to add at the current frequency
* of the corresponding grid cell. Input parameter
*/
void increaseFrequency(Crime crime, int frequency = 0);

/**
 * @brief Calculates the frequencies of the crimes that can be
 * obtained from the provided CrimeSet.
 * Only the crimes of the provided CrimeSet, with a location inside the
 * target area of this object will be considered; others will be ignored.
 * This method sets to zero the frequency of each cell before starting to
 * calculate frequencies. In this way, if this method is called twice
 * consecutively, then this CrimeCounter will contain only the frequencies
 * calculated in the last call.
 * Modifier method
 * @param crimes A CrimeSet object. Input parameter
 */
void calculateFrequencies(CrimeSet crimes);

/**
 * @brief Saves the matrix of frequencies of this object as a
 * PPM color image. In a PPM (portable pixmap format) color image,
 * each pixel (Color) of the image is saved as three integer
 * numbers: the red, green and blue components (each one in the range 0-255).
 * This method follows the following steps to build the PPM file:
 * - It saves the string "P3" in a line.
 * - Then it saves the provided comments (if they are not empty). It is necessary
 * to insert the character '#' at the beginning of the line.
 * - Next it saves the number of columns and rows (separated by white space).
 * - Next it saves the integer 255
 * - Finally it saves the pixels of the image (three integers for each pixel)
 * separated by white space. Pixels should be traversed by rows, and then
 * by column.
 *
 * In this method, each frequency value in the matrix, is translated to
 * a color of the output image with the help of the functions
 * int GetIndexPalette() and ColorPalette::getColor()
 * in the following way:
 *
 * int pixelIndex = GetIndexPalette(palette, log((+this)(row,column)+1),
 * log(maxFrequency+1));
 * Color color = palette.getColor(pixelIndex);
 *
 * where maxFrequency is the maximum frequency value found in the frequency
 * matrix.
 *
 * Example of a PPM file:
 * P3
 * # "P3" means this is a PPM color image in ASCII
 * # "3 2" is the width and height of the image in pixels
 * # "255" is the maximum value for each color
 * # This, up through the "255" line below are the header.
 * # Everything after that is the image data: RGB triplets.
 * # In order the pixel of this image are: red, green, blue,
 * # yellow, white and black colors.
 * 3 2
 * 255
 * 255 0 0
 * 0 255 0
 * 0 0 255
 * 255 255 0
 * 255 255 255
 * 0 0 0
 *
 * Query method
 * @param fileName String with the name of the file for the output image.
 * Input parameter
 * @param palette The palette used to translate frequencies to colors
 * of the output image. Input parameter
 * @param comment The comment may include 0 or several lines separated by \n. Input parameter
 * any of them is prefixed with the '#' character, ending by "\n"
 * @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 saveAsPPMTextImage(std::string fileName,
    ColorPalette palette, std::string comment = "");

private:

/**
 * Integer constant with the default number of rows in the matrix of
 * frequencies
 */
static const int DEFAULT_ROWS = 15;

/**
 * Integer constant with the default number of columns in the matrix of
```




```
* frequencies
*/
static const int DEFAULT_COLUMNS = 10;

int** _frequency; ///< 2D matrix with the crime frequency for each grid cell
int _nRows; ///< number of rows of the 2D matrix of frequencies
int _nCols; ///< number of columns of the 2D matrix of frequencies

/**
 * Coordinates that defines the bottom left corner of the input geographical
 * area
 */
Coordinates _bottomLeftCoord;

/**
 * Coordinates that defines the top right corner of the input geographical
 * area
 */
Coordinates _topRightCoord;

/**
 * @brief Obtains the row and column in the frequency matrix that
 * corresponds to the provided Coordinates @p coordinates.
 *
 * This method performs the following steps to obtain row and column:
 * - Compute the height (height) and width (width) of the cells in which
 * is divided the target geographical area. For the height the latitude
 * is used and for the width the longitude is used.
 * - Compute the vertical (vd) and horizontal (hd) distances
 * of the provided Coordinates @p coordinates to the bottom left corner
 * of the target geographical area.
 * - Compute row = _nRows - vd/height and column=hd/width
 * - The previous row or column values must be decremented by one when
 * they are equals respectively to the number of rows or columns of
 * the matrix, to avoid that the obtained value is an index outside the
 * valid range of indices of rows or columns of the frequency matrix.
 * Query method
 * @pre the location @p coordinates is valid and is inside the area set by
 * the Coordinates _bottomLeftCoord, _topRightCoord; otherwise an exception
 * will be thrown
 * @throw std::invalid_argument This method throws an
 * std::invalid_argument exception when the given Coordinates are
 * not valid or they are outside the target geographical area.
 * @param coordinates The Coordinate to look for its location into the
 * matrix. Input parameter
 * @param row The corresponding row into the frequency matrix.
 * Output parameter
 * @param column The corresponding column into the frequency matrix.
 * Output parameter
 */
void CrimeCounter::getRowColumn(const Coordinates &coordinates, int& row,
int& column) const;

/**
 * @brief Overloading of the () operator to access to the element at a
 * given position in the matrix of frequencies
 * @param row Row of the element
 * @param column Column of the element
 * @return A reference to the element at the given position
 */
int operator()(int row, int column);

/**
 * @brief Overloading of the () operator to access to the element at a
 * given position in the matrix of frequencies
 * @param row Row of the element
 * @param column Column of the element
 * @return A reference to the element at the given position
 */
int operator()(int row, int column);
}; // end class CrimeCounter

/**
 * @brief Computes the index into the given palette that corresponds to the
 * provided frequency. This index is calculated as follow:
 * - If frequency == maxFrequency then index=palette.getNumColors()-1;
 * - Else index=frequency * palette.getNumColors() / maxFrequency
 * @param palette The given palette. Input parameter
 * @param frequency The input value for which we want to obtain its
 * corresponding index in the palette.
 * Input parameter
 * @param maxFrequency Maximum value of frequency to consider. It should be
 * previously obtained as the maximum value in the frequency matrix.
 * Input parameter
 * @return index into the Color palette that corresponds to the provided
 * frequency
 */
int GetIndexPalette(const ColorPalette &palette, float frequency, float maxFrequency);

#endif /* CRIMECOUNTER.H */
```



6.7. CrimeCounter.cpp

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

/**
 * @file: CrimeCounter.cpp
 * @author Silvia Acid Carrillo <acid@decsai.ugr.es>
 * @author Andrés Cano Utrera <acu@decsai.ugr.es>
 * @author Luis Castillo Vidal <L.Castillo@decsai.ugr.es>
 * Created on September 24, 2024, 3:19PM
 */

#include <iostream>
#include <fstream>
#include <cmath>

#include "CrimeCounter.h"
using namespace std;

/**
 * DEFAULT_COORD_BOTTOMLEFT is set to the bottom left corner of the Boston area
 */
const Coordinates CrimeCounter::DEFAULT_COORD_BOTTOMLEFT(42.207760, -71.178673);

/**
 * DEFAULT_COORD_TOPRIGHT is set to the top right corner of the Boston area
 */
const Coordinates CrimeCounter::DEFAULT_COORD_TOPRIGHT(42.395042, -70.953728);

/* CrimeCounter::CrimeCounter(
... */
*/

void saveAsPPMTextImage(std::string fileName,
    ColorPalette palette, std::string comment = "") { // declaration must be revised
    // Compute maxFrequency as the maximum frequency in the frequency matrix
    // Save heading and comments into the ppm file
    // For every cell in the frequency matrix
        // colorIndex = GetIndexPalette(palette,
            // log(this->frequency[row][column]+1), log(maxFrequency+1));
    // Save into the ppm file the RGB components of the color at
        // the position colorIndex of the palette
}

void CrimeCounter::getRowColumn(const Coordinates &coordinates, int& row,
    int& column) const {
    if (!coordinates.isValid() ||
        !coordinates.isInsideArea(_bottomLeftCoord, _topRightCoord)) {
        throw std::invalid_argument(
            string("void CrimeCounter::getRowColumn((const Coordinates &loc, int& row, int& column) const: ")
            + coordinates.toString() + " is not a valid location or is outside the target area");
    }
    Coordinates targetArea = this->_bottomLeftCoord.length(this->_topRightCoord);
    float cellHeight = targetArea.getLatitude() / _nRows;
    float cellWidth = targetArea.getLongitude() / _nCols;
    Coordinates delta = _bottomLeftCoord.length(coordinates);

    row = _nRows - (delta.getLatitude() / cellHeight);
    column = (delta.getLongitude() / cellWidth);
    if (row == _nRows) row = _nRows - 1;
    if (column == _nCols) column = _nCols - 1;
}

// External functions
int GetIndexPalette(const ColorPalette &palette, float frequency, float maxFrequency) {
    return frequency == maxFrequency ? palette.getNumColors() - 1 :
        frequency * palette.getNumColors() / maxFrequency;
}
```



6.8. main.cpp

```
/*
 * Metodología de la Programación: Boston4
 * Curso 2024/2025
 */

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

#include <string>
#include <iostream>

#include "Coordinates.h"
#include "CrimeSet.h"
#include "CrimeCounter.h"
#include "ColorPalette.h"

using namespace std;

void showHelp(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(ostream& outputStream) {
    outputStream << "ERROR in boston4 parameters" << endl;
    outputStream << "Run with the following arguments:" << endl;
    outputStream << "boston4 [-o outputFilename] [-c text] ";
    outputStream << "[-h int] [-w int] [-d int] [-b int] [-g int] [-y int] ";
    outputStream << "<inputFile1.crm> [<inputFile2.crm>]" << endl;
    outputStream << endl;
    outputStream << "Parameters:" << endl;
    outputStream << "-o outputFilename: name of the output PPM file (tests/output/output.ppm by default)" <<
        endl;
    outputStream << "-c text: extra comment to be included in the output PPM file (list of program
        parameters, by default)" << endl;
    outputStream << "-h high: number of rows (19 by default)" << endl;
    outputStream << "-w width: number of columns (23 by default)" << endl;
    outputStream << "-d number: number of dark blue tones in the color palette (2 by default)" << endl;
    outputStream << "-b number: number of blue tones in the color palette (2 by default)" << endl;
    outputStream << "-g number: number of green tones in the color palette (2 by default)" << endl;
    outputStream << "-y number: number of yellow tones in the color palette (2 by default)" << endl;
    outputStream << "<inputFile1.crm> [<inputFile2.crm>]: input crm files" << endl;
    outputStream << endl;
}

/**
 * The purpose of this program is to obtain a heat map for the set of crimes
 * contained in the fusion of a list of input crm files, saving the result
 * in a PPM color image file.
 * The names of the input crm files are passed as arguments to the program.
 * The name of the output PPM file can be provided to the program as an optional
 * argument (if not provided, the output file is tests/output/output.ppm). See below
 * the Running syntax.
 * A PPM file can have comment lines. Each comment line begins with the
 * '#' character. This program will save in the output PPM file a comment line
 * with the content of the running parameters, but additional lines could be
 * appended with the -c parameter (see the running example).
 *
 * The program begins by processing the running arguments.
 * Next, the program obtains the fusion of the crm files whose names are passed
 * as arguments to the program.
 * Hereafter, the program calculates the frequency for each cell in which
 * is divided the target geographical area. To do that, an object of the class
 * CrimeCounter is used. This object is created using a given number of rows
 * and columns, and with a given coordinates to fix the target geographical
 * area.
 * Finally, a PPM image with the heat map is saved in the output file. This
 * PPM image is build and saved with the method
 * CrimeCounter::saveAsPPMTextImage() that obtain an image from the frequency
 * matrix of the previous CrimeCounter object. Remember to fill in the content
 * of the comment string (third parameter of the previous method) as
 * indicated above. You can insert the character '#' at the beginning of each
 * line in the comment string, using the function FormatAsComment().
 *
 * Running syntax:
 * > dist/Debug/GNU-Linux/boston4 boston4 [-o outputFilename] [-c text]
 * [-h int] [-w int] [-d int] [-b int] [-g int] [-y int]
 * <inputFile1.crm> [<inputFile2.crm>]
 *
 * Running example:
 * > dist/Debug/GNU-Linux/boston4 -c "Heat map for crimes.20.crm" -h 3 -w 3 -o /tmp/output.ppm ../DataSets/
    crimes.20.crm
 *
 * > cat /tmp/output.ppm
P3
#Running parameters: -c Heat map for crimes.20.crm -h 3 -w 3 -o /tmp/output.ppm ../DataSets/crimes.20.crm
#Heat map for crimes.20.crm
3 3
255
128 255 0 255 0 0 0 0 255 255 255 0 255 0 0 0 0 255 0 0 255 0 0 255 0 0 255
 *
 * > display -resize 500 -filter point /tmp/output.ppm

```



```
*
* @return Returns 1 in case of error in program arguments; 0 otherwise
*/
int main(int argc, char* argv[]) {
    string ofilename="tests/output/output.ppm"; // output PPM file name

    Coordinates topRightCoor=CrimeCounter::DEFAULT.COORD.TOPRIGHT, // (42.395042,-70.953728)
    bottonLeftCoor=CrimeCounter::DEFAULT.COORD.BOTTOMLEFT; // (42.207760,-71.178673)

    int firstInputFile=-1; // index in argv[] with the name of the first input file
    bool hasBeenReadInitialParameters = false; // true if all the parameters starting with - has been read
    CrimeSet crimeSet, // CrimeSet to obtain the fusion of the input crm files
    currentCrimeSet; // CrimeSet to load each one of the input crm files

    // Loop to process program arguments
    // (finish when the optional arguments have been read)

    // Obtain the fusion of the input crm files with operator+= of CrimeSet class
    // Load the first input crm file into the crimeSet object
    // Loop to load the rest of the crm files
    // Load the next crm file into the currentCrimeSet object
    // join crimeSet with currentCrimeSet using operator+= of CrimeSet

    // Calculate the frequencies of crimeSet using a CrimeCounter object

    // Compose a string with the comments for the output PPM file
    // and append the '#' character at the beginning of each line

    // Build a ColorPalette object
    // Use saveAsPPMTextImage to save the heat map in the output file , using
    // the selected palette and the comment

    return 0;
}
```