UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

Curso 2023/2024

Guion de prácticas

Kmer1

class KmerFreq

Marzo de 2024

Índice

1. Definición del problema

Como ya se indicó con anterioridad, las prácticas tienen por objeto trabajar con secuencias de nucleótidos procedentes de genomas de individuos de especies diferentes. Así pues, vamos a desarrollar un conjunto de aplicaciones sobre ficheros de texto que contienen trozos de genoma que nos permitan averiguar automáticamente la especie a la que pertenece un determinado genoma. Una vez definido el k-mero con la clase `Kmer`, vamos a abordar el concepto de frecuencia de un k-mero. Para ello se introduce una nueva clase `KmerFreq`, composición formada por un objeto de la anterior y un entero para la frecuencia.

En esta práctica vamos a desarrollar una aplicación que nos permita conocer el conjunto de k-meros con mayor frecuencia que nos servirán de base para nuestras predicciones futuras. Con este propósito vamos a desarrollar un conjunto de funciones externas que operan sobre vectores y llevan a cabo la ordenación por frecuencia con vectores de objetos de la clase `KmerFreq`. Se trata de una etapa intermedia, hasta introducir la clase `Profile` en la siguiente práctica.

2. Arquitectura de las prácticas

Como ya se indicó en la práctica anterior, la práctica `Kmer` se ha diseñado por etapas; las primeras contienen estructuras más sencillas, sobre las cuales se asientan otras estructuras más complejas y se van completando con nuevas funcionalidades. En `Kmer1` se hace un cambio en la clase `Kmer`: se incorpora alguna funcionalidad adicional y se introduce la clase `KmerFreq`, (bloques **A** y **B** de la Figura 1).

A Kmer.cpp
: Manteniendo la interfaz previa (o sea, la parte pública de la clase), se refactoriza la clase Kmer, esto es, se redefine incorporando algunos métodos adicionales que podría necesitar un biólogo.

B KmerFreq.cpp
: Implementa la clase KmerFreq, una composición formada por un objeto de la clase anterior y un entero para el registro de la frecuencia de un k-mero dentro de una secuencia genómica.

3. Objetivos

El desarrollo de esta práctica `Kmer1` persigue los siguientes objetivos:

- comprender la importancia de la interfaz de una clase (ocultamiento de información y encapsulamiento),
- practicar con el calificador const de métodos,
- practicar con referencias,

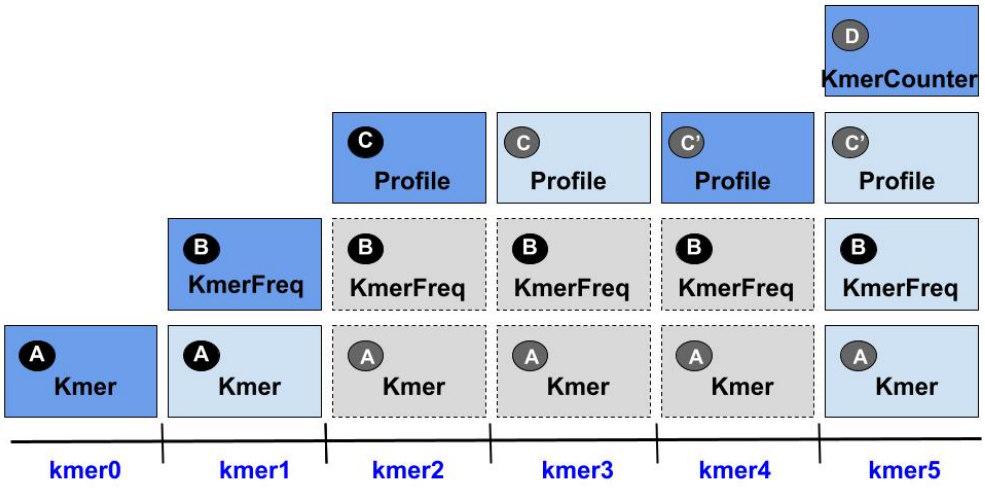

Figura 1: Arquitectura de las prácticas de MP 2024. Los cambios esenciales en las clases (cambio en estructura interna de la clase) se muestran en azul intenso; los que solo incorporan nuevas funcionalidades en azul tenue. En gris se muestran las clases que no sufren cambios en la evolución de las prácticas.

- saber adecuar los pasos de parámetros a las especificaciones,
- practicar el paso de arrays a funciones,
- practicar con una clase compuesta, la clase `KmerFreq`.

4. Práctica a entregar

El programa a desarrollar en esta práctica tiene como objetivo leer una lista de parejas $\{k$-mero, frecuencia$\}$, normalizar cada k-mero, eliminar aquellas parejas con algún nucleótido no válido o frecuencia igual a cero, y finalmente ordenar la lista de parejas resultante de mayor a menor frecuencia.

Para la elaboración de la práctica `Kmer1`, tiene disponible en el repositorio de `Prado` o bien de `github`, una plantilla de proyecto `Netbeans`. Entre los ficheros proporcionados, puede encontrar:

- En la carpeta `include`: `ArrayKmerFreqFunctions.h`, `Kmer.h`, `KmerFreq.h`.
- En la carpeta `src`: `ArrayKmerFreqFunctions.cpp`, `Kmer.cpp`, `KmerFreq.cpp`, `main.cpp`.
- En la carpeta `doc`: `documentation.doxy`.
- En la carpeta `data`: una serie de ficheros con extensión `k1in` o bien `k1out` .

- En la carpeta `tests`: una serie de ficheros con extensión `.test`.
- En la carpeta `scripts`: una serie de scripts (ficheros con extensión `.sh`).

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.

Nota: Se han retirado a propósito todos los `const` y `&` para los métodos de KmerFreq y las funciones externas de ArrayKmerFreqFunctions. Por tanto, revise todas las cabeceras. *1) El número de argumentos* y *2) los tipos han sido establecidos* y **no se han de cambiar**. Se le invita a definir todas las función(es) externa(s) adicional(es) que estime oportuno para una adecuada modularización del código.

4.1. Configurar el proyecto en Netbeans

Para montar su propio proyecto `NetBeans`, tiene varias opciones:

1. Hacer una copia de su proyecto `NetBeans` de la práctica anterior (`Kmer0`), asignándole el nombre `Kmer1`. Para ello realice los siguientes pasos:

   a) Desde el entorno de `NetBeans`, vista lógica, `Projects` → Kmer0 → Copy. Asignar el nombre de proyecto `Kmer1` a la copia.

   b) Añadir los nuevos módulos de esta práctica en su proyecto `Kmer1`:

   - Clase `KmerFreq` (ficheros `KmerFreq.h` y `KmerFreq.cpp`). Hay varias formas de hacerlo. Una de ellas es:
     - Tomar los ficheros `KmerFreq.h` y `KmerFreq.cpp` proporcionados en el repositorio y copiarlos respectivamente a las carpetas `include` y `src` de su proyecto. Esto puede hacerlo con el explorador de archivos de Linux, desde un terminal de Linux, o incluso con `NetBeans` usando la vista `física`.
     - Añadir `KmerFreq.h` y `KmerFreq.cpp` a la vista lógica del proyecto. Para ello desde el entorno de `NetBeans`, vista lógica, `Projects` → Kmer1 → Add existing item → Header Files (o bien Source Files), y añadir cada uno de los ficheros anteriores a su lugar correspondiente.
   - Módulo `ArrayKmerFreqFunctions` (`ArrayKmerFreqFunctions.h` y `ArrayKmerFreqFunctions.cpp`). Copiarlos a su lugar correspondiente de forma similar a cómo se ha hecho en el paso anterior.

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

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

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

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

2. Hacer una copia del proyecto `Kmer1` del repositorio en la carpeta donde tenga sus proyectos de la asignatura.

   a) Desde el entorno de NetBeans, vista lógica, `Projects` → Kmer1 → Copy. Elegir su carpeta de proyectos `NetBeans` y asignar el nombre de proyecto `Kmer1` a la copia.

   b) En su nuevo proyecto `Kmer1` copiar el código desarrollado en la práctica anterior (clase `Kmer`) en el nuevo proyecto. O sea, la implementación de los métodos y funciones externas que hizo en `Kmer.cpp`.

3. Montar el proyecto `Kmer1` desde cero (ver sección: Configurar el proyecto en NetBeans, del guion Kmer0).

   a) Construir el proyecto según se explicó en la práctica anterior (Kmer0).

   b) En su nuevo proyecto `Kmer1` copiar el código desarrollado en la práctica anterior (clase `Kmer`) en el nuevo proyecto. O sea, la implementación de los métodos y funciones externas que hizo en `Kmer.cpp`.

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

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

Sea cual sea la forma elegida para crear su proyecto `Kmer1`, debe revisar la clase `Kmer` según lo indicado en la sección 4.2.

- Debe revisar en `Kmer.h` los prototipos de los nuevos métodos que aparecen en esta clase.
- En `Kmer.cpp` debe comprobar si la implementación de algún método de la clase `Kmer` o alguna función externa heredados desde la práctica `Kmer0`, necesita alguna modificación.

Recuerde además revisar los prototipos de los métodos de la clase `KmerFreq` y las funciones externas de `ArrayKmerFreqFunctions`.

4.2. La clase Kmer

Esta clase necesita alguna revisión para incluir en ella varios cambios:

- `toUpper()`, se incluye un método modificador que pasa a mayúsculas.
- `toLower()`, se incluye un método modificador que pasa a minúsculas.
- `normalize()`, método que cambia su implementación para hacer uso de los métodos recién indicados.

Indicar que la funciones externas `ToUpper()` y `ToLower()` de la práctica anterior, han sido marcadas como obsoletas (deprecated) al introducir los nuevos métodos `toUpper()` y `toLower()`. Esto quiere decir que se podrían eliminar en versiones posteriores del módulo que las contiene, y por ello es recomendable no usarlas a partir de ahora. Salvo por los cambios que acabamos de comentar, la interfaz es idéntica a la de Kmer0.

4.3. La clase KmerFreq

Ya sabemos que un k-mero es una secuencia corta de nucleótidos de longitud k, que puede encontrarse dentro de un genoma determinado. A este le vamos a añadir la frecuencia con la que esta secuencia aparece en el genoma.

```
class KmerFreq {
...

private:
    Kmer _kmer; ///< the Kmer object
    int _frequency; ///< the frequency
}; // end class KmerFreq
```

Ejemplo 1 *Dado el siguiente trozo de genoma: "GCGCCCcGTG" podemos obtener la siguiente lista de KmerFreq, 3-meros (primer componente del par), con sus frecuencias asociadas (segunda componente de KmerFreq):*

$$\{GCG\text{–}1\}, \{CGC\text{–}1\}, \{GCC\text{–}1\}, \{CCC\text{–}2\}, \{CCG\text{–}1\}, \{CGT\text{–}1\}, \{GTG\text{–}1\}$$

Los 3-meros del ejemplo anterior han sido normalizados, esto es, la cadena convertida a mayúsculas y comprobado que cada carácter es un nucleótido válido. En caso contrario se sustituye por el carácter '_' (carácter que ha sido definido con la constante `Kmer::MISSING_NUCLEOTIDE` en el fichero `Kmer.h`).

La frecuencia de un k-mero es el número de veces en que este se encuentra en la secuencia genómica.

4.4. El módulo ArrayKmerFreqFunctions

Para lograr la operatividad que se va a necesitar en la función `main()` (leer el conjunto de parejas $\{k$-mero, frecuencia$\}$ y guardarlos en un array, convertir los k-meros a mayúscula, eliminar los k-meros con caracteres no válidos o frecuencia cero, ordenar el array de parejas de mayor a menor frecuencia, y mostrar el array por la salida estándar) se van a definir una serie de funciones externas que realicen dichas tareas.

La mayoría las funciones mencionadas más arriba van a necesitar pasar el array de objetos de la clase KmerFreq como parámetro, en sus diferentes modalidades de paso de parámetros, según que modifique o no los objetos de dicho array.

De esta forma, entre los nuevos ficheros que se proporcionan en el repositorio, va a encontrar los ficheros `ArrayKmerFreqFunctions.h` (sección 5) y `ArrayKmerFreqFunctions.cpp` que proporcionarán un conjunto de funciones externas que van a utilizar arrays de objetos de la clase `KmerFreq`. En el fichero `ArrayKmerFreqFunctions.h`, se encuentran las especificaciones de cada función, algunas como `NormalizeArrayKmerFreq()` y `ZipArrayKmerFreq` más específicas se detallan con ejemplos (en `ArrayKmerFreqFunctions.h`) y pseudocódigo (en `ArrayKmerFreqFunctions.cpp`).

4.5. El módulo main.cpp

El programa a construir debe leer un entero n, y un conjunto de n pares $\{k$-mero–frecuencia$\}$, mediante la función externa `ReadArrayKmerFreq()`:

- n es el número de pares $\{k$-mero–frecuencia$\}$ que se tienen que leer y almacenar en un array de objetos `KmerFreq`,
- un par $\{k$-mero–frecuencia$\}$ es una pareja formada por una cadena de nucleótidos y un entero en la forma que se detalla en los ejemplos.

Una vez leído el array de objetos `KmerFreq`, su contenido se debe normalizar mediante la función externa `NormalizeArrayKmerFreq()`. Este paso consiste en pasar a mayúscula todos los caracteres de cada k-mero y en reemplazar cada carácter no válido por el carácter `Kmer::MISSING_NUCLEOTIDE`.

A continuación, el array resultante se debe comprimir mediante la función externa `ZipArrayKmerFreq(<valor>)`. Este paso consiste en eliminar del array aquellos pares $\{k$-mero–frecuencia$\}$ que tengan algún carácter `Kmer::MISSING_NUCLEOTIDE` en la composición del k-mero, o bien tenga una frecuencia igual o inferior al valor definido (cero en nuestro caso). Los dos criterios para la eliminación se especifican a través de los parámetros de la función anterior, pudiéndose aplicar cada criterio de forma independiente.

Posteriormente, el array debe ordenarse en orden decreciente de frecuencia usando la función externa `SortArrayKmerFreq()`. A la hora

de realizar la ordenación por frecuencia decreciente, sea cual sea el algoritmo que vaya a implementar, tiene que adaptar éste para tratar de forma especial los empates en frecuencia. Si dos objetos tienen idéntica frecuencia, el par se ordena por orden alfabético de la cadena de los k-meros empatados.

Finalmente, se debe mostrar por la salida estándar el contenido completo del array usando la función externa `PrintArrayKmerFreq()`. Se escribirá en una línea, el valor: n, seguido de la lista ordenada de pares k-mero–frecuencia. Vea los ejemplos de ejecución de la sección 4.6.

4.6. Ejemplos de ejecución

En la carpeta `data` del proyecto proporcionado en el repositorio de la asignatura, dispone de varios ficheros de prueba, algunos de los cuales se detallan aquí.

Puede también ejecutar una serie de pruebas de forma automática, usando la script `runTests.sh`. Esta script usa los ficheros de texto con extensión `.test` proporcionados en el repositorio. Cada uno de estos ficheros contiene un test de integración. Si miramos el contenido de estos ficheros, podemos ver cuál sería la salida esperada del programa en cada caso.

Ejemplo 2 *Un ejemplo de llamada al programa desde un terminal podría ser:*

```
Linux> dist/Debug/GNU-Linux/kmer1 < data/4pairsDNA.k1in
```

El contenido del fichero `data/4pairsDNA.k1in` *es el siguiente:*

```
4
Ct 5
hG 4
nG 1
cT 4
```

Como puede verse, los datos que necesita leer el programa de la entrada estándar son:

- *Un número entero para definir el valor de n.*
- *Una lista de n pares $\{k$-mero-frecuencia$\}$.*

Una vez leídos los datos, vamos a tratar en detalle el problema de la normalización. Esta se va a realizar mediante la función `NormalizeArrayKmerFreq()`*. Normalizar el array consisten, en primer lugar, aplicar el método* `normalize()` *a cada uno de los kmers del array, cuyo resultado sería:*

```
4
CT 5
_G 4
_G 1
CT 4
```

Como resultado del proceso anterior, podemos observar que aparecen varias entradas con k-meros idénticos lo que no es deseable para su explotación futura. Luego, en segundo lugar se revisan cada uno de los kmers del array para dejar una única entrada por k-mero con las frecuencias acumuladas de los k-meros idénticos anteriormente presentes. El resultado sería:

```
2
CT 9
_G 5
```

Pero este no es el resultado final pues, aún quedan por aplicar las funciones `ZipArrayKmerFreq()` *y* `SortArrayKmerFreq()`*. La primera elimina el* `KmerFreq: _G 5` *por contener un* `MISSING_NUCLEOTIDE` *en el k-mer, la segunda no tiene ningún tiene efecto en el array.*
La salida obtenida al ejecutar el programa del ejemplo sería:

```
1
CT 9
```

En el fichero `data/pairsDNA_normalized_zipped.k1out` *puede encontrarse también la salida anterior.*

En la carpeta `data` del proyecto proporcionado en el repositorio puede encontrar otros ficheros con extensión `.k1in` que puede utilizar para ejecutar el programa de forma similar a los ejemplos anteriores. También dispone de ficheros con extensión `.k1out` que contienen las correspondientes salidas.

Ejemplo 3 *Un ejemplo de llamada al programa desde un terminal podría ser:*

```
Linux> dist/Debug/GNU-Linux/kmer1 < data/6pairsDNA_missing.k1in
```

El contenido del fichero `data/6pairsDNA_missing.k1in` *es el siguiente:*

```
6
GC 2
cG 0
Cc 3
cu 1
ug 2
gu 1
```

La salida obtenida al ejecutar el programa sería la siguiente:

```
2
CC 3
GC 2
```

En el fichero `data/6pairsDNA_missing.k1out` *pueden encontrase también la salida anterior.*

El ejemplo anterior muestra que algunos pares $\{k$-mero–frecuencia$\}$ han sido eliminados tras la aplicación de `ZipArrayKmerFreq(<valor>)`. Concretamente, han sido eliminados los pares que contenían un k-mero con nucleótidos desconocidos y los que tenían una frecuencia igual a 0.

Ejemplo 4 *Si se ejecuta el programa del ejemplo anterior de la siguiente forma:*

```
Linux> kmer1 < data/6pairsDNA_missing.k1in > data/6pairsDNA_missing.k1in.out
```

Esto es, se utiliza `6pairsDNA_missing.k1in` *como entrada para la ejecución de Kmer1, y se guarda la salida estándar del programa en el fichero* `data/6pairsDNA_missing.k1in.out`*.*

El contenido del fichero `data/6pairsDNA_missing.k1in.out` obtenido con la anterior ejecución, debería de ser idéntico al contenido del fichero proporcionado en el repositorio `data/6pairsDNA_missing.k1out`, que contiene la salida esperada. Puede usar los comandos de linux `diff` o `wdiff` para comprobar si la salida obtenida coincide con la esperada.

4.7. 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 Kmer1.zip, sin más aditivos.
Como alternativa, se sugiere utilizar el script `runZipProject.sh` que debe estar en la carpeta `script` de Kmer1, junto con otras utilidades. Para ello, leer la siguiente sección.

5. Código para la práctica

5.1. Kmer.h

```
/*
 * Metodología de la Programación: Kmer1
 * Curso 2023/2024
 */

/**
 * @file Kmer.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>
 * @author Javier Martínez Baena <jbaena@ugr.es>
 *
 * Created on 24 October 2023, 14:00
 */

#ifndef KMER_H
#define KMER_H

#include <iostream>
#include <string>

/**
 * @class Kmer
 * @brief It represents a list of k consecutive nucleotides of a DNA or RNA
 * sequence. Each nucleotide is represented with a character like
 * 'A', 'C', 'G', 'T', 'U'.
 */
class Kmer {
public:
    /**
     * A static const character representing an unknown nucleotide. It is used
     * when we do not known which nucleotide we have in a given position of
     * a Kmer
     */
    static const char MISSING_NUCLEOTIDE = '-';

    /**
     * @brief It builds a Kmer object using a string with @p k characters
     * (nucleotides). Each character will be set to the value
```

```
     * @p MISSING_NUCLEOTIDE.
     * @throw std::invalid_argument Throws an std::invalid_argument exception
     * if @p k is less or equal than zero
     * @param k the number of nucleotides in this Kmer. It should be an integer
     * greater than zero. Input parameter
     */
    Kmer(int k=1);

    /**
     * @brief It builds a Kmer object with the characters in the string @p text
     * representing the list of nucleotides of the new Kmer.
     * @throw std::invalid_argument Throws an std::invalid_argument exception
     * if the given text is empty
     * @param text a string with the characters representing the nucleotides for
     * the kmer. It should be a string with at least one character. Input parameter
     */
    Kmer(const std::string& text);

    /**
     * @brief Returns the number of nucleotides in this Kmer.
     * Query method
     * @return the number of nucleotides in this Kmer
     */
    int getK() const;

    /**
     * @brief Returns the number of nucleotides in this Kmer.
     * Query method
     * @return the number of nucleotides in this Kmer
     */
    int size() const;

    /**
     * @brief Returns a string with a list of characters, each one representing
     * a nucleotide of this Kmer.
     * Query method
     * @return The text of this Kmer as a string object
     */
    std::string toString() const;

    /**
     * @brief Gets a const reference to the character (nucleotide) at the given
     * position.
     * Query method
     * @param index the position to consider. Input parameter
     * @throw std::out_of_range Throws an std::out_of_range exception if the
     * index is not in the range from 0 to k-1 (both included).
     * @return A const reference to the character at the given position
     */
    const char& at(int index) const;

    /**
     * @brief Gets a reference to the character (nucleotide) at the given
     * position.
     * Modifier method
     * @param index the position to consider. Input parameter
     * @throw std::out_of_range Throws an std::out_of_range exception if the
     * index is not in the range from 0 to k-1 (both included).
     * @return A reference to the character at the given position
     */
    char& at(int index);

    /**
     * @brief Converts uppercase letters in this Kmer to lowercase
     * Modifier method
     */
    void toLower();

    /**
     * @brief Converts lowercase letters in this Kmer to uppercase
     * Modifier method
     */
    void toUpper();

    /**
     * @brief Normalizes this Kmer. That is, it converts all the characters to
     * uppercase. Then, invalid characters are replaced by the
     * MISSING_NUCLEOTIDE value.
     * Modifier method
     * @param validNucleotides a string with the list of characters
     * (nucleotides) that should be considered as valid. Input parameter
     */
    void normalize(const std::string& validNucleotides);

    /**
     * @brief Returns the complementary of this Kmer. For example, given the Kmer
     * "TAGAC", the complementary is "ATCTG" (assuming that we use.
     * @p nucleotides="ATGC" and @p complementaryNucleotides="TACG").
     * If a nucleotide in this object is not in @p nucleotides, then that
     * nucleotide remains the same in the returned kmer.
     * Query method
     * @param nucleotides A string with the list of possible nucleotides. Input parameter
     * @param complementaryNucleotides A string with the list of complementary
     * nucleotides. Input parameter
     * @throw std::invalid_argument Throws an std::invalid_argument exception if
     * the sizes of @p nucleotides and @p complementaryNucleotides are not
     * the same
     * @return The complementary of this Kmer
     */
    Kmer complementary(const std::string& nucleotides,
```

```
        const std::string& complementaryNucleotides) const;

private:
    /**
     * A string with a list of characters representing the nucleotides in
     * this Kmer.
     */
    std::string _text;
}; // end class Kmer

/**
 * @brief Checks if the given nucleotide is contained in @p validNucleotides.
 * That is, if the given character can be considered as part of a genetic
 * sequence.
 * @param nucleotide The nucleotide (a character) to check. Input parameter
 * @param validNucleotides The set of characters that we consider as possible
 * characters in a genetic sequence. Input parameter
 * @return true if the given character is contained in @p validNucleotides;
 * false otherwise
 */
bool IsValidNucleotide(char nucleotide, const std::string& validNucleotides);

/**
 * @brief Converts to lowercase the characters (nucleotides) of the given Kmer
 * @deprecated This function could go away in future versions
 * @param kmer A Kmer object. Output parameter
 */
void ToLower(Kmer& kmer);

/**
 * @brief Converts to uppercase the characters (nucleotides) of the given Kmer
 * @deprecated This function could go away in future versions
 * @param kmer A Kmer object. Output parameter
 */
void ToUpper(Kmer& kmer);

#endif /* KMER_H */
```

5.2. KmerFreq.h

```
/*
 * Metodología de la Programación: Kmer1
 * Curso 2023/2024
 */

/**
 * @file KmerFreq.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>
 * @author Javier Martínez Baena <jbaena@ugr.es>
 *
 * Created on 27 October 2023, 11:03
 */


#ifndef KMER_FREQ_H
#define KMER_FREQ_H

#include "Kmer.h"

/**
 * @class KmerFreq
 * @brief A pair formed by a Kmer object and a frequency (an int),
 * that gives the frequency of a Kmer (times it appears) in a genoma.
 */
class KmerFreq {
public:
    /**
     * @brief Base constructor.
     * It builds a KmerFreq object containing a Kmer with one nucleotide, the
     * unknown nucleotide (Kmer::MISSING_NUCLEOTIDE) and 0 as its frequency
     */
    KmerFreq();

    /**
     * @brief Gets a const reference to the Kmer of this KmerFreq object
     * Query method
     * @return A const reference to the Kmer of this KmerFreq object
     */
    Kmer getKmer();

    /**
     * @brief Gets the frequency of this KmerFreq object
     * Query method
     * @return The frequency of this KmerFreq object
     */
    int getFrequency();

    /**
     * @brief Sets the Kmer of this KmerFreq object.
     * Modifier method
     * @param kmer The new Kmer value for this object. Input parameter
     */
    void setKmer(Kmer kmer);
```

```
    /**
     * @brief Sets the frequency of this KmerFreq object
     * Modifier method
     * @throw std::out_of_range if @p frequency is negative
     * @param frequency the new frequency value for this KmerFreq object.
     * Input parameter
     */
    void setFrequency(int frequency);

    /**
     * @brief Obtains a string with the string and frequency of the kmer
     * in this object (separated by a whitespace).
     * Query method.
     * @return A string with the nucleotide and frequency of the kmer
     * in this object
     */
    std::string toString();

private:
    Kmer _kmer; ///< the Kmer object
    int _frequency; ///< the frequency
}; // end class KmerFreq

#endif /* KMER_FREQ_H */
```

5.3. ArrayKmerFreqFunctions.h

```
/*
 * Metodología de la Programación: Kmer1
 * Curso 2023/2024
 */

/**
 * @file ArrayKmerFreqFunctions.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>
 * @author Javier Martínez Baena <jbaena@ugr.es>
 *
 * Created on 27 October 2023, 11:00
 */
#ifndef ARRAYKMERMFREQFUNCTIONS_H
#define ARRAYKMERFREQFUNCTIONS_H

#include "KmerFreq.h"

/**
 * @brief Reads the number of used elements and the elements of an array of
 * KmerFreq
 * If @p nElements is greater than @p dim, then only @p dim elements are read
 * If @p nElements is negative, then zero elements are read
 * @param array An array where the elements will be stored. Output parameter
 * @param dim The capacity of the array. Input parameter
 * @param nElements The number of elements used by the array. Note that this
 * integer could be modified in this function. Output parameter
*/
void ReadArrayKmerFreq(KmerFreq array[], int dim, int nElements);

/**
 * @brief Prints in the standard output the number of used elements and the
 * elements of an array of KmerFreq
 * @param array The array of KmerFreq to be printed. Input parameter
 * @param nElements The number of elements used by the array. Input parameter
*/
void PrintArrayKmerFreq(KmerFreq array[], int nElements);

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

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

```
/**
 * @brief Sorts the given array of KmerFreq in decreasing order of
 * frequency using a sort algorithm.
 * Note: It is not allowed to use any sorting algorithm available in any
 * C++ library.
 * @param array An array of KmerFreq. Input/output parameter
 * @param nElements The number of elements used by the array. Input parameter
*/
void SortArrayKmerFreq(KmerFreq array[], int nElements);


/**
 * @brief Normalizes the given array of KmerFreq. That is, for each Kmer in
 * the array, all its characters are converted to uppercase. Then, invalid
 * characters are replaced by the UNKNOWN_NUCLEOTIDE value.
 *
 * If after the previous normalization process of every kmer, identical kmers
 * are obtained, these will be merged into the first identical kmer by
 * adding their frequencies.
 * For example, supposse the following list of kmers:
4
Ct 5
hG 4
nG 1
cT 4
 *
 * After the process of normalization of every kmer, we obtain the following
 * list of kmers:
 *
4
CT 5
_G 4
_G 1
CT 4
 *
 * The final step will transform the list into:
2
CT 9
_G 5
 *
 * @param array An array of KmerFreq. Input/output parameter
 * @param nElements The number of elements used by the array. Output parameter
 * @param validNucleotides a string with the list of characters (nucleotides)
 * that should be considered as valid. Input parameter
*/
void NormalizeArrayKmerFreq(KmerFreq array[], int nElements,
        std::string validNucleotides);

/**
 * @brief Deletes the KmerFreq object from the argument array that is at
 * position @p pos.
 * @param array An array of KmerFreq.  nput/output parameter
 * @param nElements The number of elements used by the array. Note that the
 * number of elements in the argument array will be modified. Input/output parameter
 * @param pos The index of the position to be deleted. Input parameter
 * @throw std::out_of_range Throws an std::out_of_range exception if @p pos
 * is not in the range from 0 to nElements-1 (both included).
 */
void DeletePosArrayKmerFreq(KmerFreq array[], int nElements, int pos);

/**
 * @brief Deletes the KmerFreq objects from the argument array which verifies
 * one the following two criteria:
 *  -# The argument deleteMissing is true and the Kmer contains an unknown
 * nucleotide
 *  -# The frequency is less or equals to @p lowerBound.
 *
 * Note that the number of elements in the argument array could be modified.
 *
 * @param array An array of KmerFreq. Input/output parameter
 * @param nElements The number of elements used by the array. Note that the
 * number of elements in the argument array could be modified. Input/output parameter
 * @param deleteMissing A bool value that indicates whether kmers with any
 * unknown nucleotide should be removed. This parameter is false by default.
 * Input parameter
 * @param lowerBound An integer value that defines which KmerFreq objects
 * should be deleted from the given argument array. KmerFreq objects with a
 * frequency less or equals to this value, are deleted. This parameter has zero
 * as default value. Input parameter
 */
void ZipArrayKmerFreq(KmerFreq array[], int nElements,
        bool deleteMissing=false, int lowerBound=0);

#endif /* ARRAYKMERFREQFUNCTIONS_H */
```

5.4. ArrayKmerFreqFunctions.cpp

```
/*
 * Metodología de la Programación: Kmer1
 * Curso 2023/2024
 */

/**
 * @file ArrayKmerFreqFunctions.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>
 * @author Javier Martínez Baena <jbaena@ugr.es>
 *
 * Created on 27 October 2023, 12:00
 */


#include "ArrayKmerFreqFunctions.h"


void NormalizeArrayKmerFreq(KmerFreq array[], int nElements,
        string validNucleotides){

    // Loop to traverse and normalize each one of the kmers in array
          // Normalize kmer i


    // Loop to traverse the kmers in array from position 1 to position nElements-1
          // index = Position of array[i].getKmer() in the subarray that begins
          //         at position 0 and ends at position i-1
          // If array[i].getKmer() was found in the the subarray from 0 to i-1
              // Accumulate the frequencies of the kmers at positions
              //    index and i in the kmer at position index
              // Delete from the array, the kmer at position i
}
```

5.5. main.cpp

```
/*
 * Metodología de la Programación: Kmer1
 * Curso 2023/2024
 */

/*
 * File:   main.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>
 * @author Javier Martínez Baena <jbaena@ugr.es>
 *
 * Created on 27 October 2023, 12:00
 */

#include <iostream>
#include <string>

#include "KmerFreq.h"
#include "ArrayKmerFreqFunctions.h"

using namespace std;

/**
 * This program reads from the stardard input, an integer n (number of
 * elements in the list of pairs kmer-frequency) and a list of n pairs
 * kmer-frequency. The pairs should be stored in an array (local variable
 * in main() ), and then each one of the Kmers in the array must be
 * normalized (all its characters are converted to uppercase
 * and invalid characters are replaced by the MISSING_NUCLEOTIDE character)
 * and zipped (those pairs with a missing nucleotide or with a
 * frequency less or equals to zero must be deleted from the array). Next,
 * the array should be sorted in decreasing
 * order of frequency (if two pairs have the same frequency, the pair with a
 * minor Kmer should appear first in the array). Finally, the program shows the
 * array in the standard output.
 *
 * Running example:
 * > kmer1 < data/6pairsDNA_missing.k1in
2
CC 3
GC 2
 */
int main(int argc, char* argv[]) {
    // This string contains the list of nucleotides that are considered as
    // valid within a genetic sequence. The rest of characters are considered as
    // unknown nucleotides
    const string VALID_NUCLEOTIDES = "ACGT";

    // This is a constant with the dimension of the array kmers
    const int DIM_ARRAY_KMERS=100;

    // This is the array where the kmers read from the standard input
    // will be stored
    KmerFreq kmers[DIM_ARRAY_KMERS];

    int nKmers; // Number of elements in the array kmers

    // Read an integer n (number of pairs to read)

    // Read the n pairs kmers-frequency from the standard input and put them
    //      in the array kmers

    // Normalizes each kmer in the array kmers

    // Zip the kmers in the array kmers
```

```
    // Sort the array kmers
    // Print the array kmers in the standard output
    return 0;
}
```