UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

Curso 2022/2023

Guion de prácticas

Language1

class BigramFreq

Marzo de 2023

Contents

1 Definición del problema

Como ya se indicó con anterioridad, las prácticas tienen como objeto principal trabajar con textos escritos en diferentes idiomas. Así pues, vamos a desarrollar un conjunto de aplicaciones sobre ficheros de texto que nos permitan averiguar automáticamente el idioma en el que está escrito un texto. Una vez definido el bigrama, clase `Bigram`, vamos a abordar el concepto de frecuencia de un bigrama. Para ello se define una nueva clase BigramFreq, composición formada por un objeto de la anterior y datos propios.

En esta práctica vamos a desarrollar una aplicación que nos permita conocer el conjunto de bigramas con mayor frecuencia que nos servirán de base para nuestras prediciones futuras. Con este propósito vamos a desarrollar un conjunto de funciones que operan sobre vectores y llevan a cabo la ordenación por frecuencia con vectores. Se trata de un paso intermedio, hasta introducir la siguiente clase.

2 Arquitectura de las prácticas

Como ya se indicó en la práctica anterior, la práctica `Language` 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 `Language1` se hace un cambio en la clase Bigram (se cambia la definición de los datos de la clase) y se introduce la clase `BigramFreq`, (bloques **A'** y **B** de la Figura 1).

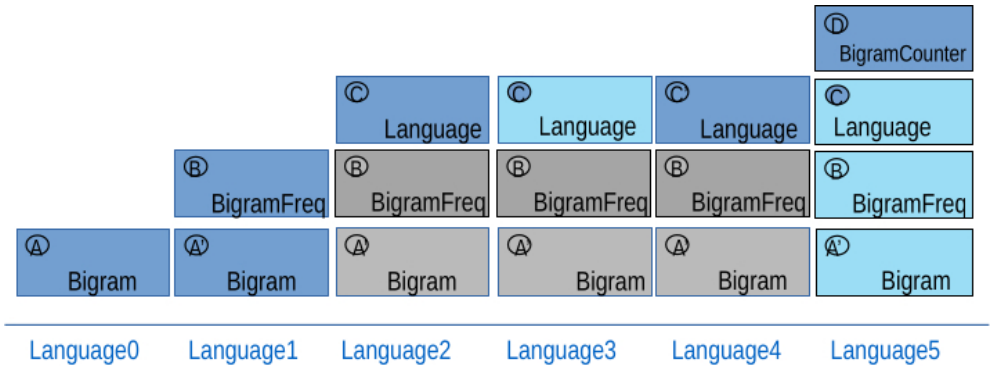

Figura 1: Arquitectura de las prácticas de MP 2023. Los cambios esenciales en las clases (cambio en estructura interna de la clase) se muestran en azul intenso; los que solo incorporan nuevas funcionalidades en azul tenue. En gris se muestran las clases que no sufren cambios en la evolución de las prácticas.

A' Bigram.cpp
: Manteniendo la interfaz previa, se refactoriza la clase Bigram, esto es, se redefine la clase con un c-string, incorporando además algunos métodos adicionales.

B BigramFreq.cpp
Implementa la clase BigramFreq, una composición formada por un objeto de la clase anterior y un entero para el registro de la frecuencia de un bigrama.

3 Objetivos

El desarrollo de esta práctica `Language1` persigue los siguientes objetivos:

- practicar con c-string,
- comprender la importancia de la interfaz de una clase (ocultamiento de información y encapsulamiento),
- practicar con referencias,
- practicar la devolución por referencia,
- practicar el paso de arrays a funciones,
- practicar con una clase compuesta, la clase `BigramFreq`.

4 Práctica a entregar

Para la elaboración de la práctica, dispone de una serie de ficheros que se encuentran en `Language1_nb.zip` o en el repositorio de `git`. Una vez descomprimido va a encontrar los ficheros: `Bigram.h`, `BigramFreq.h`,`BigramFreq.cpp`, `main.cpp`, `documentation.doxy`, entre otros.

Para montar el proyecto NetBeans, hay dos opciones: proceder como en la práctica anterior o bien hacer una copia del proyecto Language0 en un nuevo proyecto.

Opción 1) se monta un proyecto desde cero (ver sección: Configurar el proyecto en NetBeans, del guion Language0.)
Opción 2) desde el entorno de NetBeans, vista lógica, `Projects` → Language0 → Copy. Guardar la copia con el nombre Language1. Se obtiene como resultado una copia idéntica pero bajo un nombre de proyecto y un directorio físico distinto.

Sea como fuere como haya obtenido el proyecto Language1, es necesario primero, ubicar los ficheros procedentes de `Language1_nb.zip` o del repositorio de `git`, en la vista física `Files`, en sus directorios correspondientes.
Segundo, debe configurar convenientemente la vista lógica del proyecto para incluir los ficheros nuevos.
Entre los nuevos ficheros va a encontrar ArrayBigramFreqFunctions.h y su homólogo ArrayBigramFreqFunctions.cpp, que contiene un conjunto de funciones externas para desarrollar, que van a utilizar arrays de objetos de

la clase BigramFreq; son utilidades construidas a partir de la clase.

A la hora de editar los ficheros `*.h`, fíjese en las cabeceras de los métodos y en los comentarios de cada método y función pues, en ellos están detallados sus especificaciones. Indicar que, *se han retirado a propósito todos los const y* & para los métodos de BigramFreq. Por tanto, revise todas las cabeceras. *El número de argumentos* y *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 oportuna(s) para una adecuada modularización del código.

4.1 La clase Bigram

Se refactoriza la clase Bigram, ahora contiene un c-string de longitud 3 (un array de char con 2 posiciones para los caracteres del bigrama, más el carácter `'\0'` en la última posición), en vez de un string. Se incluyen además dos nuevos métodos, un nuevo constructor con c-string como argumento y un método modificador `toUpper()`. La función externa anterior se queda obsoleta por lo que se elimina o se pone entre comentarios. Salvo por los cambios que acabamos de comentar, la interfaz es idéntica a la de Language0. Debe comprender que una interfaz de una clase bien diseñada, hace que los programas que usan la 'nueva' clase no tengan que modificarse al cambiar la parte privada de la clase. A continuación se muestran detalles del código:

```
class Bigram {
...
    /**
     * @brief It builds a Bigram object with @text (a c-string) as the
     * text of the bigram. If the c-string @p text contains a number of character
     * other than two, then the text of the bigram will be initialized with
     * "--"
     *
     * @param text the input text for the bigram.
     */
    Bigram(char text[]);
    /**
     * Converts lowercase letters in this bigram to uppercase. Modifier.
     */
    void toUpper();
private:
    /**
     * The text in this Bigram. Should be a c-string with two characters that
     * includes the '\0' at the end.
     */
    char _text[3];
};
```

4.2 La clase BigramFreq

Ya sabemos que un bigrama es cualquier secuencia de dos caracteres consecutivos, que aparecen en un texto determinado, al que le vamos a añadir una frecuencia con la que esta secuencia aparece en el texto. Por ejemplo, el siguiente texto:

"Hola a todas las personas"

contendrá los siguientes bigramas (primer componente del par), con sus frecuencias asociadas (segunda componente de BigramFreq):

{{”ho”},1}, {{”ol”},1}, {{”la”},2}, {{”to”},1}, {{”od”},1}, {{”da”},1},
{{”as”},3}, {{”pe”},1}, {{”er”},1}, {{”rs”},1}, {{”so”},1}, {{”on”},1}

Como sabemos, no se diferencian las mayúsculas, ni se contemplan los separadores como los espacios en blanco, ni caracteres especiales que no sean alfabéticos.

4.3 El módulo main

Se pide construir un programa que lea: n y un conjunto de pares bigramas-frecuencia y los almacene en memoria. Donde:

n es el número de pares bigramas-frecuencia que se tienen que leer y almacenar en un array de objetos BigramFreq,

el conjunto de n pares: cadena de dos caracteres y un entero (bigrama-frecuencia) en la forma que se detalla en los ejemplos.

Una vez en memoria, en un array de BigramFreq, el programa los ordena por frecuencia en orden decreciente(*) y se modifican para convertirlos en mayúsculas. Finalmente se muestra por pantalla el contenido completo del array como: n seguido de la lista ordenada de pares bigrama-frecuencia, ver los ejemplos de ejecución.

(*): La ordenación final es única. A la hora de realizar la ordenación por frecuencia, sea cual sea el algoritmo que vaya a implementar, tiene que adaptar éste para tratar de forma especial los empates en valor de frecuencia. Si dos objetos tienen idéntica frecuencia, el par se ordena por orden alfabético de bigrama.
Ejemplo, dados los siguientes pares:
{{”pa”},20}, {{”pe”},16}, {{”er”},16}, {{”on”},16}, {{”rc”},7}.
En el vector debe aparecer como:

```
{{``pa''},20}
{{``er''},16}
{{``on''},16}
{{``pe''},16}
{{``rc''},7}
```

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

```
Linux> dist/Debug/GNU-Linux/language1 < data/SimpleTextbigrams.txt
```

4.4 Ejemplos de ejecución

En el fichero Languague1_nb.zip o entre los ficheros procedentes del repositorio de `git`, encontrará la carpeta `data` con ficheros de prueba, algunos de los cuales se detallan aquí.

Ejemplo 1. Dado el contenido del fichero `data/SimpleTextbigrams.txt`

```
11
 et 1
ex 2
im 1
 le 1
mp 1
 os 1
 pl 1
 si 1
 te 3
 to 1
 xt 2
```

se muestra a continuación la salida del programa `language1`

```
11
TE 3
EX 2
XT 2
ET 1
IM 1
LE 1
MP 1
OS 1
PL 1
SI 1
TO 1
```

Ejemplo 2. Dado el contenido del fichero `data/30bigrams.txt` la salida del programa es:

```
30
DE 822
DO 401
CO 369
DA 288
CA 276
 CI 268
 DI 203
HA 190
BA 181
CE 166
CU 157
GU 91
HO 91
GO 87
GA 76
 BI 70
FU 69
FA 62
 HI 59
HE 47
BE 42
BU 41
 FI 39
 GI 38
DU 37
FE 34
BO 28
GE 26
HU 23
FO 19
```

Ejemplo 3. Si se ejecutan las siguientes instrucciones:

```
language1 <data/30bigrams.txt >data/30bigrams.txt.out
language1 <data/30bigrams.txt.out >data/30bigrams.txt.out.out
```

Esto es, se utiliza `30bigrams.txt.out` como entrada para una nueva ejecución de language1, el contenido de `30bigrams.txt.out.out` debe ser idéntico al contenido de `30bigrams.txt.out`. Ya que, los valores están ordenados según el criterio establecido y los bigramas en mayúsculas. Puede comprobarse fácilmente mediante el comando de linux `diff` o `wdiff`.

4.5 Para la entrega

La práctica deberá ser entregada en Prado, en la fecha que se indica en cada entrega, y consistirá en un fichero ZIP del proyecto. Se puede

```
Language
├── build
├── dist
│   ├── Debug
│   └── Release
├── data
├── doc
│   └── doc.doxy
├── include
│   └── *h
├── Makefile
├── nbproject
├── scripts
├── src
│   └── *cpp
└── zip
```

Figura 2: Estructura interna de cada proyecto Language

montar el zip desde NetBeans, a través de **File** → **Export project** → **To zip**. El nombre, en esta ocasión es Language1.zip, sin más aditivos.
Como alternativa, se sugiere utilizar el script `runZipProject.sh` que debe estar en la carpeta `script` de Language1, junto con otras utilidades. Para ello, leer la siguiente sección.

5 Uso de scripts

En primer lugar, hemos de completar, nuestro espacio de trabajo, con la carpeta Scripts, también disponible en Languague1_nb.zip o el repositorio de `git`.

```
ProyectosNetbeans
├── MPGeometry*
├── Language0
├── Language1
├── *
└── Scripts
```

Scripts: Una serie de scripts Bash de apoyo a las funciones de NetBeans. Es única, y como vemos se encuentra al mismo nivel que los Language.

Sin embargo, existe una carpeta `script`, una por cada Language, ver figura 2. No confundir con la `Scripts` anterior, aquí es dónde se encuentran los scripts que vamos a ejecutar manualmente [1].

Como se ha indicado, en la carpeta `script` se encuentran una serie de utilidades bash, ficheros con extensión `*.sh` como:
`runDocumentation.sh` que va a facilitar el proceso de generación de

[1] Estos hacen uso de las utilidades en Scripts, por eso, para un correcto funcionamiento es importante que Scripts esté correctamente ubicado en su lugar.

la documentación con doxygen y su visualización en un navegador.
O también, `runZipProject.sh` que facilita el proceso de elaborar un zip, eliminando previamente todos los archivos binarios que no tiene sentido exportar como `*.bin, *.o,` etc.

Para ejecutar un script, basta con situarse sobre el script, pulsar botón derecho del ratón y `Run`. Una vez se ejecuta el script, el fichero zip se encuentra en el directorio `zip`.

6 Código para la práctica

A BigramFreq.h

```
/*
 * Metodología de la Programación: Language1
 * Curso 2022/2023
 */

/*
 * @file   BigramFreq.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 29 January 2023, 11:00
 */

#ifndef BIGRAM_FREQ_H
#define BIGRAM_FREQ_H

#include <string>

#include "Bigram.h"

/**
 * @class BigramFreq
 * @brief A pair of a Bigram object and a frequency (an int), that gives the
 * frequency of a Bigram (times it appears) in a text.
 */
class BigramFreq {
public:
    /**
     * @brief Base constructor. It builds a BigramFreq object with "__" as
     * the text of the bigram and 0 as the frequency
     */
    BigramFreq();

    /**
     * @brief Gets a const reference to the Bigram of this BigramFreq object.
     * Query method
     * @return A const reference to the Bigram of this BigramFreq object
     */
    Bigram getBigram();

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

    /**
     * @brief Sets the Bigram of this BigramFreq object
     * Modifier method
     * @param bigram The new Bigram value for this object. Input parameter
     */
    void setBigram(Bigram bigram);

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

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

```
private:
    Bigram _bigram; ///< the Bigram object
    int _frequency; ///< the frequency
};


#endif /* BIGRAM_FREQ_H */
```

B ArrayBigramFreqFunctions.h

```
/*
 * Metodología de la Programación: Language1
 * Curso 2022/2023
 */

/**
 * @file ArrayBigramFreqFunctions.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 7 February 2023, 19:45
 */
#ifndef ARRAYBIGRAMFREQFUNCTIONS_H
#define ARRAYBIGRAMFREQFUNCTIONS_H

#include "BigramFreq.h"

/**
 * @brief Reads the number of used elements and the elements of an array of
 * BigramFreq
 * @param array The 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. Output parameter
*/
void readArrayBigramFreq(BigramFreq array[], int dim, int nElements);

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

/**
 * @brief Swaps the elements at positions @p first and @p second in the given
 * array of BigramFreq
 * @param array The array of BigramFreq. 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
*/
void swapElementsArrayBigramFreq(BigramFreq array[], int nElements, int first,
                int second);


/**
 * @brief Sorts the given array of BigramFreq in decreasing order of
 * frequency
 * @param array The array of BigramFreq. Input/Output parameter
 * @param nElements The number of elements used by the array. Input parameter
*/
void sortArrayBigramFreq(BigramFreq array[], int nElements);

/**
 * @brief Converts to uppercase all the bigrams within the given array
 * @param array An array of BigramFreq objects. Input/Output parameter
 * @param nElements The number of elements in the array. Input parameter
 */
void toUpperArrayBigramFreq(BigramFreq array[], int nElements);

#endif /* ARRAYBIGRAMFREQFUNCTIONS_H */
```