UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

Curso 2022/2023

Guion de prácticas

Language2

class Language

Marzo de 2023

Índice

1. Definición del problema

Como ya sabemos, 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.

En esta práctica vamos a implementar el modelo para un idioma, *language* [1] para evitar confusión, mediante la clase Language. Un language, se construye a partir de un texto, contabilizando las frecuencias de todos los bigramas que se han hallado en el texto fuente. Inicialmente, habrá tantos languages como textos se analicen. No obstante, se pueden crear languages enriquecidos, una suerte language aglutinado, que integre varios languages de un mismo idioma, obtenido por fusión. Pensemos en un language procedente de la composición de los textos de Quijote + Fortunata + BodasdeSangre.
De forma más concreta, la clase Language va a contener un vector de BigramFreq junto con las funcionalidades, que se encontraban dispersas en la versión anterior de la práctica, definidas como funciones externas y a la que le vamos a añadir alguna funcionalidad específica como: la fusión. Se da así, un paso más en el encapsulamiento de la clase.

Por otro lado, a partir de ahora, vamos a cambiar la forma de ejecución de nuestros programas. Se dejan de realizar lecturas desde teclado (o su equivalencia mediante redireccionamientos). La lecturas de datos se van a realizar directamente desde ficheros y la variabilidad de los datos se va a especificar desde la línea de comandos, esto es, a través de los parámetros de nuestro programa.
Con esta nueva forma de proceder, la aplicación consiste en leer un número indeterminado de ficheros language, `(*.bgr)`, especificados desde la línea de comandos, que se van a fusionar en un solo language. Finalmente, el language resultante se va a salvar en un nuevo fichero `.bgr`.

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 `Language2` se implementa la clase `Language`, bloque **C** de la Figura 1 y se refactorizan las funciones externas presentes en el módulo `ArrayBigramFreqFunctions` dentro de la clase Language. Desaparece pues, este módulo del proyecto Language2.

[1] Utilizaremos language en lugar de idioma, para evitar confusión. Todo el mundo piensa en idioma como un concepto único, sin embargo vamos a trabajar con varios ficheros languages de español, por ejemplo.

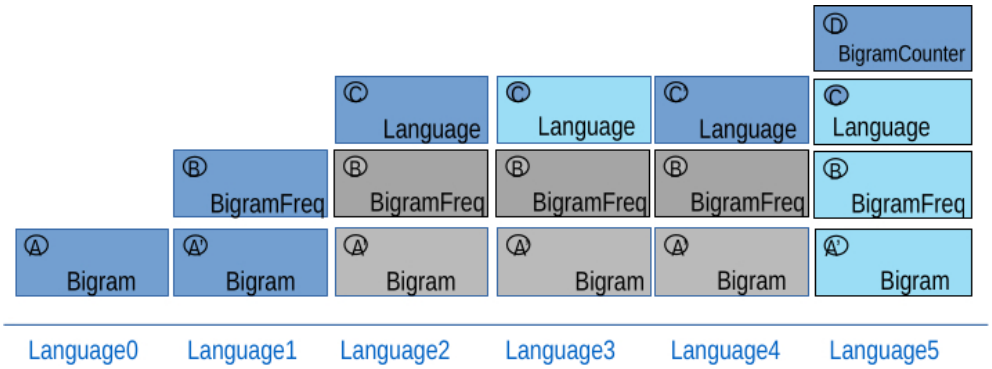

Figura 1: Arquitectura de las prácticas de MP 2023. Los cambios esenciales en las clases (cambio en estructura interna de la clase) se muestran en azul intenso; los que solo incorporan nuevas funcionalidades en azul tenue. En gris se muestran las clases que no sufren cambios en la evolución de las prácticas.

C Language.cpp
: Implementa la clase Language, una estructura para almacenar las frecuencias de un conjunto de bigramas. Será nuestro modelo para un idioma, esto es, se usará para el conteo de frecuencias calculadas a partir de un documento de texto en un idioma. En una primera aproximación se usará un vector estático.

3. Objetivos

El desarrollo de esta práctica `Language2` persigue los siguientes objetivos:

- Conocer la estructura de los ficheros de language `*.bgr`.
- Leer datos de un fichero de texto.
- Escribir datos en un fichero de texto.
- Practicar con una clase compuesta, la clase `Language`.
- Practicar el paso de parámetros a `main()` desde la línea de comandos.

4. Un fichero language

Un fichero language es un fichero que contiene una lista de bigramas identificados como frecuentes en un determinado idioma. El fichero language tiene el siguiente formato (todos los que se proporcionan para esta práctica están codificados en ISO8859-1).

- La primera línea contiene siempre la cadena “MP-LANGUAGE-T-1.0”.

- La segunda línea contiene una cadena que describe qué idioma es.
- La tercera línea contiene el número de bigramas diferentes que están asociados al idioma descrito.
- Las siguientes líneas contienen la lista de bigramas, con sus frecuencias, según el número de bigramas especificados.

Nota: Un fichero language, además de estar en el formato indicado, debe estar ordenado según el criterio establecido en la práctica anterior [2].

A continuación se muestra un fichero language en francés, donde se pueden identificar los 6 bigramas más frecuentes.

```
MP-LANGUAGE-T-1.0
french
6
es 1774
en 1266
le 1220
re 1117
de 1103
on 1100
```

A continuación se muestra un extracto de un fichero language en el idioma español.

```
MP-LANGUAGE-T-1.0
spanish
554
ue 36957
de 34013
en 33672
es 32637
qu 32470
er 27172
os 25527
la 23061
do 21590
... hasta 554 bigramas en total ...
```

5. Práctica a entregar

Para la elaboración de la práctica, dispone de una serie de ficheros que se encuentran en `Language2_nb.zip` o en el repositorio de `git`. El proyecto nuevo es `Language2`, recupere y ubique los ficheros proporcionados en los directorios adecuados. No se olvide de revisar la vista lógica del proyecto, para editar la nueva configuración del proyecto y que NetBeans efectúe los cambios en el Makefile.

[2] La ordenación es por valor decreciente de frecuencia, y en caso de empate en frecuencia se resuelve por orden alfabético creciente de bigrama.

- Implementar los métodos indicados en el fichero **Language.h** (ver detalles en sección 6).

- El módulo **main.cpp** tiene por objetivo gestionar diferentes ficheros language `*.bgr` (con el formato indicado en sección 4) pertenecientes a un mismo idioma, con el fin de obtener un único language fusión de los primeros. El resultado se guarda en un nuevo fichero language con el mismo formato que los anteriores.

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

```
Linux> dist/Debug/GNU-Linux/language2 <file1.bgr> [<file2.bgr> ... <filen.bgr>]
<outputFile.bgr>
```

Notación: Los `[]` indican que no es obligatorio, es opcional.

1. Todos los parámetros se pasan al programa desde la línea de órdenes.

2. El programa debe de almacenar el contenido de un fichero language en un objeto de la clase Language, tal y como se describe en la sección 6.

3. El programa recibe al menos dos ficheros language `<file1.bgr>` y `<fileoutput.bgr>`. El primero (y los siguientes excepto el último) es(son) de lectura, mientras que el último `<fileoutput.bgr>`, de escritura, es donde se salva el objeto language resultante de la fusión. Todos los ficheros language tienen extensión **bgr**, con el formato descrito anteriormente y están en el orden descrito en el guion precedente. Recuerde, orden decreciente por frecuencia y en caso de empate, se ordena por orden alfabético de bigrama.

4. Cada fichero de entrada debe estar asociado a un mismo idioma, aunque los bigramas que contengan puedan ser diferentes de un fichero a otro, por ejemplo, porque se han obtenido a partir de distintas fuentes. Si un language tiene un identificador de idioma diferente al primro, entonces no se incluirá en la fusión. El language de salida tendrá como identificador de idioma, el que tenga el primero de los ficheros.

5. El programa debe leer todos y cada uno de los ficheros language de entrada, indicados en la línea de órdenes y fusionarlo en un único language. La fusión a realizar se descompone en fusión de pares de languages, según el siguiente procedimiento:

   a) Si el bigrama leído de un fichero nuevo ya existe en el language de salida, se suman las dos frecuencias.

   b) Si el bigrama nuevo no existe en el language de salida, se añade al final de este language con la frecuencia leída.

6. Antes de salvar el objeto language de salida en un fichero `.bgr`, recuerde que debe ordenarse en orden decreciente por frecuencia y en caso de empate, se ordena por orden alfabético de bigrama.

5.1. Ejemplos de ejecución

En el fichero languague2_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: Faltan argumentos para la ejecución del programa

```
Linux>dist/Debug/GNU-Linux/language2
Linux>dist/Debug/GNU-Linux/language2 30bigrams.bgr
```

La salida del programa da un mensaje similar a este:

```
Error, run with the following parameters:
language2 <file1.bgr> [<file2.bgr> ... <filen.bgr>] <outputFile.bgr>
```

De ahora en adelante prescindiremos del path completo: `dist/Debug/GNU-Linux/` para la ejecución del programa language2 y lo encontraremos de forma abreviada como `language2`.

Ejemplo 2: Se suministra un solo fichero language de entrada, no hay fusión.

```
Linux>language2 30bigrams.bgr 30bigrams.bgr.out
```

El fichero language `30bigrams.bgr.out` es idéntico al de entrada.

Ejemplo 3: Se hace fusión de dos ficheros language del mismo idioma (spanish), donde todos los bigramas son comunes.

```
Linux>language2 30bigrams.bgr 30bigrams2.bgr 30bigrams+30bigrams2.bgr
```

Proceso seguido:

```
Abre el fichero 30bigrams.bgr
Idioma detectado: spanish
Lee 30 bigramas
Abre el fichero 30bigrams2.bgr
Idioma detectado: spanish
Lee 30 bigramas
Realiza fusión. Total bigramas: 30
Ordena y salva en fichero 30bigrams+30bigrams2.bgr
```

El fichero obtenido `30bigrams+30bigrams2.bgr` tendría el siguiente contenido:

```
MP-LANGUAGE-T-1.0
spanish
30
he 4105
ha 1407
de 1374
ce 971
hi 929
do 892
co 830
ho 706
be 697
ca 641
di 525
fo 429
da 403
ge 365
ci 323
ga 303
go 290
bu 284
cu 273
ba 267
bo 244
fi 229
gu 228
fe 211
fu 191
bi 186
fa 179
gi 134
du 112
hu 85
```

El resultado, es un language en español que contiene el mismo número de bigramas y se han acumulado las frecuencias.

Ejemplo 4: Se hace fusión de dos ficheros language del mismo idioma (spanish), donde hay 6 bigramas no comunes

```
Linux>language2 30bigrams.bgr 36bigrams.bgr 30bigrams+36bigrams.bgr
```

Proceso seguido:

```
Abre el fichero 30bigrams.bgr
Idioma detectado: spanish
Lee 30 bigramas
Abre el fichero 36bigrams.bgr
Idioma detectado: spanish
Lee 36 bigramas
Realiza fusión. Total bigramas: 36
Ordena y salva en fichero 30bigrams+36bigrams.bgr
```

El fichero obtenido `30bigrams+36bigrams.bgr` tendría el siguiente contenido:

```
MP-LANGUAGE-T-1.0
spanish
36
de 1644
do 802
co 738
da 576
ca 552
ci 536
di 406
ha 380
ba 362
ce 332
cu 314
gu 182
ho 182
go 174
br 156
ga 152
bi 140
fu 138
fa 124
hi 118
he 94
be 84
bu 82
fi 78
gi 76
du 74
fe 68
bo 56
gr 53
ge 52
hu 46
fo 38
dr 37
cr 36
fr 15
ct 11
```

Los bigramas no comunes son `br,gr,dr,cr,fr,ct`, cuyos respectivos valores de frecuencia coinciden con los valores que tenían en el archivo de language origen de cada bigrama.

Ejemplo 5: Se pretende hacer fusión de varios ficheros language de idiomas diferentes.

```
Linux>language2 30bigrams.bgr lesMiserables.bgr changedMan.bgr 30bigrams2.bgr
what.bgr
```

Cuando durante el proceso se abre el fichero `lesMiserables.bgr`, y se detecta que el idioma no coincide con el de referencia `30bigrams.bgr`, el programa abandona su lectura e ignora su procesamiento continuando con los siguientes argumentos, caso de haberlos. En este ejemplo, el resultado de la ejecución es similar al Ejemplo 4.

5.2. 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, `Language2.zip`. Se procederá como en prácticas anteriores.

6. Código para la práctica

A. Language.h

```
/*
 * Metodología de la Programación: Language2
 * Curso 2022/2023
 */

/**
 * @file Language.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 LANGUAGE_H
#define LANGUAGE_H


#include <iostream>
#include "BigramFreq.h"

/**
 * @class Language
 * @brief It defines a model for a given language. It contains a vector of
 * pairs Bigram-frequency (objects of the class BigramFreq) and an identifier
 * (string) of the language.
 */
class Language {
public:

    /**
     * @brief Base constructor. It builds a Language object with "unknown" as
     * identifier, and an empty vector of pairs Bigram-frequency.
     */
    Language();

    /**
     * @brief It builds a Language object with "unknow" as
     * identifier, and a vector of @p numberBigrams pairs Bigram-frequency.
     * Each pair will be initialized as "--" for the Bigram and 0 for the
     * frequency.
     * @throw std::out_of_range Throws a std::out_of_range exception if
     * @p numberBigrams > @p DIM_VECTOR_BIGRAM_FREQ or numberBigrams<0
     * @param numberBigrams The number of bigrams to use in this Language
     */
    Language(int numberBigrams);

    /**
     * @brief Returns the identifier of this language object
     * @return A const reference to the identifier of this language object
     */
    const std::string& getLanguageId() const;

    /**
     * @brief Sets a new identifier for this language object
     * @param id The new identifier
     */
    void setLanguageId(const std::string& id);


    /**
     * @brief Gets a const reference to the BigramFreq at the given position
     * of the vector in this object
     * @param index the position to consider
     * @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 BigramFreq at the given position
     */
    const BigramFreq& at(int index) const;

    /**
     * @brief Gets a reference to the BigramFreq at the given position of the
     * vector in this object
     * @param index the position to consider
     * @throw std::out_of_range Throws an std::out_of_range exception if the
     * given index is not valid
     * @return A reference to the BigramFreq at the given position
     */
    BigramFreq& at(int index);

    /**
     * @brief Gets the number of BigramFreq objects
     * @return The number of BigramFreq objects
     */
    int getSize() const;

    /**
     * @brief Searchs the given bigram in the list of bigrams in this
     * Language. If found, it returns the position where it was found. If not,
     * it returns -1. We consider that position 0 is the one of the first
     * bigram in the list of bigrams and this->getSize()-1 the one of the last
```

```
     * bigram
     * @param bigram A bigram
     * @pre The list of bigrams should be ordered in decreasing order of
     * frequency. This is not checked in this method.
     * @return If found, it returns the position where the bigram
     * was found. If not, it returns -1
     */
    int findBigram(const Bigram& bigram) const;

    /**
     * @brief Obtains a string with the following content:
     * - In the first line, the number of bigrams in this Language
     * - In the following lines, each one of the pairs bigram-frequency
     * (separated by a whitespace).
     * @return A string with the number of bigrams and the list of pairs of
     * bigram-frequency in the object
     */
    std::string toString() const;

    /**
     * @brief Sort the vector of BigramFreq in decreasing order of frequency.
     * If two BigramFreq objects have the same frequency, then the alphabetical
     * order of the bigrams of those objects will be considered (the object
     * with a bigram that comes first alphabetically will appear first)
     * Modifier method
     */
    void sort();

    /**
     * @brief Saves this Language object in the given file
     * @param fileName A c-string with the name of the file where this Language
     * object will be saved
     * @throw std::ios_base::failure Throws a std::ios_base::failure exception
     * if the given file cannot be opened or if an error occurs while writing
     * to the file
     */
    void save(const char fileName[]) const;

    /**
     * @brief Loads into this object the Language object stored in the given
     * file
     * @param fileName A c-string with the name of the file where the Language
     * will be stored.
     * @throw std::out_of_range Throws a std::out_of_range exception if the
     * number of bigrams in the given file, cannot be allocated in this Language
     * because it exceeds the maximum capacity
     * @throw std::ios_base::failure Throws a std::ios_base::failure exception
     * if the given file cannot be opened or if an error occurs while reading
     * from the file
     * @throw throw std::invalid_argument Throws a std::invalid_argument if
     * an invalid magic string is found in the given file
     */
    void load(const char fileName[]);

    /**
     * @brief Appends a copy of the given BigramFreq to this Language object.
     * If the bigram is found in this object, then its frequency is increased
     * with the one of the given BigramFreq object. If not, a copy of the
     * given BigramFreq object is appended to the end of the list of
     * BigramFreq objects in this Language.
     * @param bigramFreq The BigramFreq to append to this object
     */
    void append(const BigramFreq& bigramFreq);

    /**
     * @brief Appends to this Language object, the list of pairs
     * bigram-frequency contained in @p language.
     * @param language A Language object
     */
    void join(const Language& language);

private:
    static const int DIM_VECTOR_BIGRAM_FREQ = 2000; ///< The capacity of the array _vectorBigramFreq
    std::string _languageId; ///< language identifier
    BigramFreq _vectorBigramFreq[DIM_VECTOR_BIGRAM_FREQ]; ///< array of BigramFreq
    int _size; ///< Number of elements in _vectorBigramFreq
    static const std::string MAGIC_STRING_T; ///< A const string with the magic string for text files
};

#endif /* LANGUAGE_H */
```