UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

DGIM y GI-C

Curso 2022/2023

Guion de prácticas

Language2

class Language

Abril de 2023

Contents

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 apartir 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 la composición de los textos de Quijote + Fortunata + RomanceroGitano.

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, como funciones externas y alguna funcionalidad específica como: la fusión. Se da así un paso más en el encapsulamiento de la clase.

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 indeterminados de ficheros language, `*.bgr`, especificados desde la línea de comandos, que se van a fusionar en un solo y 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` en 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.

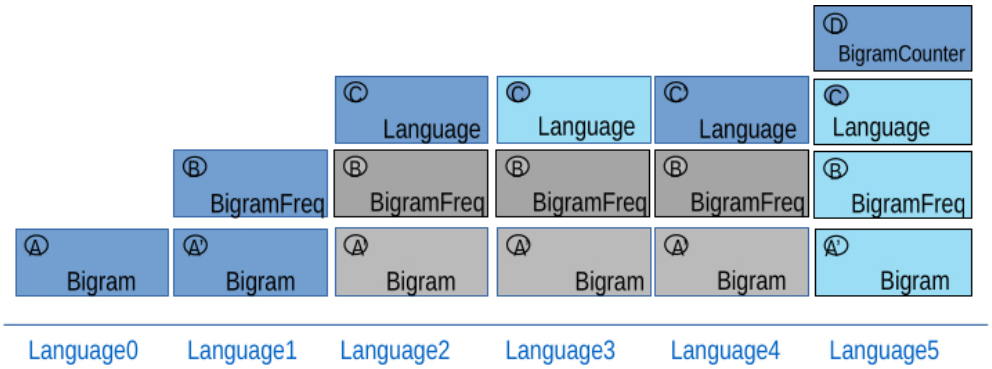

Figura 1: Arquitectura de las prácticas de MP 2023. Los cambios esenciales en las clases (cambio en estructura interna de la clase) se muestran en azul intenso; los que solo incorporan nuevas funcionalidades en azul tenue. En gris se muestran las clases que no sufren cambios en la evolución de las prácticas.

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 `Language1` 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 identificados en este fichero 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 en orden decreciente por‘ frecuencia. A continuación se muestra un fichero language en francés. Se puede detectar 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
548
ue 36957
de 34013
en 33672
es 32637
qu 32470
er 27172
os 25527
la 23061
do 21590
... hasta 548 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`. 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 NetBeans cambie el Makefile.

- 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, 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 es opcional, no obligatorio.

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 en orden decreciente por frecuencia.
4. Cada fichero de entrada indicado 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 apartir de distintas fuentes. Si dos languages no coinciden en el mismo idioma, segunda línea de la cabecera, no procede a hacer la fusión.
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 que hay en memoria, se suman las dos frecuencias.
   (b) Si el bigrama nuevo no existe en el language que hay en memoria, se añade al final del language en memoria con la frecuencia leída.
6. Antes de salvar el objeto language de memoria en un fichero `.bgr`, recuerde que debe de estar ordenado en orden decreciente por frecuencia.
7. El programa debe recibir también desde la línea de órdenes, y siempre en primer lugar, un bigrama determinado. Debe buscar el bigrama en el idioma resultante de fusionar todos los ficheros de idiomas y dar la frecuencia de éste en la fusión. Como se ha indicado anteriormente, ésta frecuencia fusionada debe coincidir con la suma de las frecuencias parciales en cada fichero de idioma indicado en la llamada.

5.1 Ejemplos de ejecución

El fichero languague2_nb.zip contiene un conjunto de 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
LE 1
MP 1
OS 1
PL 1
SI 1
ET 1
TO 1
IM 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. Dado el contenido del fichero `data/30Bigrams2.txt`, los bigramas se encuentran en el orden requerido (de mayor a menor frecuencia) luego, la salida del programa es idéntica a los datos originales salvo por que, los bigramas están en mayúsculas.

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. Este se puede obtener de cualquiera de las formas descritas en el guion de prácticas anterior.

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 a 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.
     * @param numberBigrams The number of bigrams to use in this Language.
     * Input parameter
     */
    Language(int numberBigrams);

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

    /**
     * @brief Sets a new identifier for this language object. Modifier method.
     * @param id The new identifier. Input parameter
     */
    void setLanguageId(std::string id);


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

    /**
     * @brief Gets a reference to the BigramFreq at the given position of the
     * vector in this object. Query and modifier 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 reference to the BigramFreq at the given position
     */
    BigramFreq at(int index);

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

    /**
     * @brief Sort the vector of BigramFreq in decreasing order of frequency.
     * Modifier method
     */
    void sort();
```

```
    /**
     * @brief Saves this Language object in the given file. Query method
     * @param fileName A c-string with the name of the file where this Language
     * object will be saved. Input parameter
     * @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(char fileName[]);

    /**
     * @brief Loads the Language object stored in the given file into this
     * object. Modifier method
     * @param fileName A c-string with the name of the file where the Language
     * will be stored. Input parameter
     * @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(char fileName[]);

   /**
    * @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).
    * Query method
    * @return A string with the number of bigrams and the list of pairs of
    * bigram-frequency in the object
    */
    std::string toString();


    /**
     * @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.
     * Modifier method
     * @param bigramFreq The BigramFreq to append to this object. Input parameter
     */
    void append(BigramFreq bigramFreq);

    /**
     * @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
     * @param bigram A bigram
     * @return If found, it returns the position where the bigram
     * was found. If not, it returns -1
     */
    int findBigram(const Bigram& bigram) const;

    /**
     * @brief Appends to this Language object, the list of pairs
     * bigram-frequency contained in @p language.
     * Modifier method
     * @param language A Language object. Input parameter
     */
    void join(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 */
```