UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

GI

Curso 2022/2023

Guion de prácticas

Language0

class Bigram

Marzo de 2023

Contents

1 Las prácticas del curso, una visión general

Todas las prácticas que se van a desarrollar en este curso 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, lo que se conoce como aprendizaje supervisado (Abrir →). Una aproximación al problema es como sigue:

En primer lugar procederemos a aprender un modelo de idioma a partir de unos textos que han sido especificados como de un determinado idioma. Se aprenden modelos para cada uno de los idiomas que se van a considerar.

En segundo lugar, procederemos con la clasificación. A la llegada de un nuevo texto, en un idioma desconocido, trataremos de hallar el modelo al que más se ajusta el nuevo texto para realizar nuestra predicción.

El idioma de un texto puede averiguarse por ejemplo, considerando las frecuencias de pares de caracteres consecutivos (*bigramas*). Así, una alta frecuencia del bigrama “th” será indicativo del idioma inglés mientras que una alta frecuencia para el bigrama “ña” indicativo del idioma español.

Los ficheros de texto que vamos a considerar ***.txt** [1], contienen datos que han sido codificados como texto usando un esquema como ISO8859. A diferencia de los ficheros binarios, aparecen exactamente como en la pantalla del ordenador, es decir, como una secuencia de caracteres. Para conocer más acerca de esta codificación ver sección 8, para más detalles mirar (Abrir →).

Como ya sabemos, un fichero de texto puede abrirse y editarse con programas editores como `notepad` en Windows o `gedit` en Ubuntu Linux. Se puede conocer la codificación de un fichero de texto, utilizando el comando Linux:

```
linux> file quijote.txt
```

siendo válidas las siguientes salidas:

```
ascii
ISO-8859
iso-8859-1
iso-8859-15
```

Se puede cambiar la codificación de un texto a ISO-8859-15 con el comando:

```
linux> iconv  -f UTF-8 -t ISO-8859-15 < input.txt > output.txt
```

Nota: siempre deberá de verificar que la codificación de los ficheros que utilice sea la elegida para un correcto funcionamiento de nuestros programas. Para más detalles ver sección 8.

[1] Algunos de los que vamos a utilizar son *quijote.txt, fortunata.txt, aliceWonder.txt, lesMiserables.txt* entre otros...

2 Arquitectura de las prácticas

La práctica `Language` se ha diseñado por etapas, las primeras contienen clases más sencillas, sobre las cuales se asientan otras más complejas y en el progreso además se van completando nuevas funcionalidades a clases anteriores. La Figura 1 muestra el diseño de la arquitectura que se va a emplear y que se va a ir desarrollando progresivamente en esta y las siguientes sesiones de prácticas. Indicar aquí, que las estructuras de datos que se van emplear van a sufrir varias modificaciones y van a evolucionar conforme se van adquiriendo nuevos conocimientos (los impartidos en teoría que se van a aplicar) o bien conforme llegan nuevas especificaciones (nuevos problemas para resolver). Esta situación no es excepcional sino que, la modificación y evolución de clases es un proceso habitual en cualquier desarrollo de software.

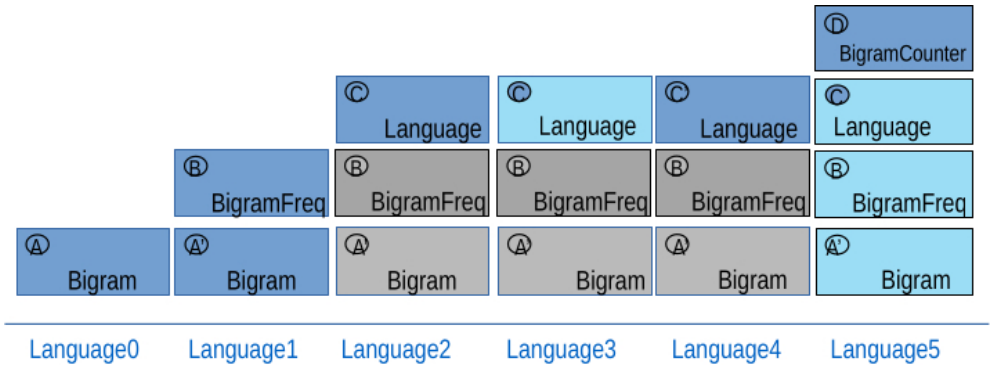


Figura 1: Arquitectura de las prácticas de MP 2023. Los cambios esenciales en las clases (cambio en estructura interna de la clase) se muestran en azul intenso; los que solo incorporan nuevas funcionalidades en azul tenue. En gris se muestran las clases que no sufren cambios en la evolución de las prácticas.

A Bigram.cpp
: Implementa la clase Bigram, una clase que se define inicialmente con un string, para la que se van a desarrollar los primeros métodos.

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 de la clase anterior y un entero para el registro de la frecuencia de un bigrama.

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.

C' Language.cpp
Manteniendo la interfaz previa, se cambia la estructura de la clase para alojar el vector de BigramFreq de forma más eficiente en memoria dinámica.

D BigramCounter.cpp
Implementa la clase BigramCounter, la estructura que va a permitir alojar una matriz bidimensional en memoria dinámica; nos será de utilidad para el aprendizaje de un modelo de idioma.

Este trabajo progresivo, con la incorporación de nuevos métodos, se ha planificado en hitos sucesivos con entregas en Prado. Cada práctica requerirá además de la implementación de alguna(s) funcion(es) externa(s) y de una función main() propia para cubrir sus objetivos específicos.

3 Objetivos

El desarrollo de la práctica Language0 persigue los siguientes objetivos:

- repasar conceptos sobre clases y calificadores de métodos,
- repasar conceptos básicos de funciones,
- practicar el paso de parámetros por referencia,
- practicar la devolución por referencia,
- conocer acerca de codificación de caracteres,
- reforzar la comprensión de los conceptos de compilación separada.

4 Language0

En esta práctica vamos a comenzar con el primer procesamiento de texto necesario para la obtención de un modelo de idioma. Indicar que, el mínimo elemento de información con el que vamos a trabajar a lo largo de toda la práctica son los bigramas [2]. Estos provendrán de unos textos que van a ser analizados y a partir de los cuales vamos a obtener unas estadísticas simples.

Para la extracción de los bigramas vamos a considerar *únicamente* una lista de caracteres alfabéticos válidos como los que se especifican a continuación:
`abcdefghijklmnopqrstuvwxyz...áàâ...` el conjunto de caracteres válidos completos se encuentra definido en el fichero `main.cpp`.

[2] O bien, podrían considerarse la frecuencia de simples caracteres, o bien de ternas de caracteres, trigramas... ngramas, lo que daría una mayor precisión a la predicción.

El primer paso consiste pues, en convertir todos los caracteres a minúscula [3], se ignoran por tanto caracteres no válidos en una palabra como: ? !=. , ; : etc. a la hora de formar bigramas.

Así, dado el texto siguiente:

```
Abc.de2a!!
```

El conjunto de bigramas que se obtiene es:

```
ab
bc
de
```

5 Práctica a entregar

Para la elaboración de la práctica dispone de una serie de ficheros con código C++, donde se encuentran las especificaciones de lo que tiene que implementar. Los primeros, relacionados con la clase Bigram son: `Bigram.h, Bigram.cpp, main.cpp....`

Fíjese en las cabeceras de los métodos y en los comentarios de cada método pues en ellos están detalladas sus especificaciones. Indicar que, en las declaraciones de prototipos, *el número de argumentos* y *los tipos han sido establecidos* y **no se han de cambiar**. Sin embargo, se deja a *libre elección* el paso de argumento para cada parámetro y el tipo de calificador **const** o (no const) de argumentos y métodos para que, sean los más adecuados para su propósito.

- Implementar los métodos indicados en el fichero **Bigram.h** (ver detalles en sección 5.1).

- El módulo **main.cpp** tiene por objetivo leer dos cadenas de caracteres *cadena1* y *cadena2*. La primera hará las veces del texto de entrada, mientras que la segunda de un bigrama a buscar.

  cadena1 servirá para montar a partir de ella el conjunto de bigramas hallados (válidos). El conjunto de bigramas es un vector de bigramas y podrá contener bigramas repetidos. Los bigramas se insertan en el vector desde la posición 0 en adelante en el mismo orden en que se encuentran en *cadena1*.

  cadena2 es un bigrama que de encontrarse en el vector resultado deberá ser modificado en el vector. El cambio consiste en convertir a mayúsculas todas las ocurrencias del bigrama del vector.

  Nota: El código del main que elabore en esta ocasión, no es necesario que estén correctamente modularizado por falta de los conocimientos necesarios sobre paso de vectores a funciones.

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

```
linux> dist/Debug/GNU-Linux/language0 < data/SimpleText.txt
```

[3] Mediante la función *tolower()* : (Abrir →), también existe su homóloga *toupper()*.

5.1 Los detalles se encuentran en el .h

```
/*
 * Metodología de la Programación: Language0
 * Curso 2022/2023
 */

/**
 * @file Bigram.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 2 February 2023, 11:00
 */

#ifndef BIGRAM_H
#define BIGRAM_H

#include <iostream>
#include <string>

/**
 * @class Bigram
 * @brief It represents a pair of characters. It is used to store pairs of
 * consecutive characters from a text.
 * It uses a string to store the pair of characters
 */
class Bigram {
public:
    /**
     * @brief It builds a Bigram object with @p text as the
     * text of the bigram. If the string @p text contains a number of characters
     * other than two, then the text of the bigram will be initialized with
     * "--"
     *
     * @param text the text for the bigram. It should be a string with just two
     * characters.
     */
    Bigram(const std::string& text="--");

    /**
     * @brief It builds a Bigram object using the two characters passed as
     * parameters of this constructor as the text of the bigram
     *
     * @param first the first character for the bigram
     * @param second the second character for the bigram
     */
    Bigram(char first, char second);

    /**
     * @brief Obtains a copy of the text of this bigram as a string object
     * @return The text of this bigram as a string object
     */
    std::string getText() const;

    /**
     * @brief Obtains a copy of the text of this bigram as a string object
     * @return The text of this bigram as a string object
     */
    std::string toString() const;

    /**
     * @brief Gets a const reference to the character at the given position
     * @param index the position to consider
     * @throw std::out_of_range Throws a std::out_of_range exception if the
     * index is not equals to 0 or 1
     * @return A const reference to the character at the given position
     */
    const char& at(int index) const;

    /**
     * @brief Gets a reference to the character at the given position
     * @param index the position to consider
     * @throw std::out_of_range Throws a std::out_of_range exception if the
     * index is not equals to 0 or 1
     * @return A reference to the character at the given position
     */
    char& at(int index);


private:
    /**
     * The text in this Bigram. Should be a string with two characters
     */
    std::string _text;
};


/**
 * Checks if the given character is contained in @p validCharacters. That is, if
 * the given character can be consireded as part of a word.
 * @param character The character to check
 * @param validCharacters The set of characters that we consider as possible
 * characters in a word. Any other character is considered as a separator.
 * @return true if the given character is contained in @p validCharacters; false
 * otherwise
 */
bool isValidCharacter(char character, const std::string& validCharacters);
```

```
/**
 * Converts lowercase letters in the given bigram to uppercase
 * @param bigram
 */
void toUpper(Bigram &bigram);

#endif /* BIGRAM_H */
```

5.2 Ejemplos de ejecución

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

```
Textos;imple_Tex te
```

se muestra a continuación la salida del programa `language0`

```
11
te
ex
xt
to
os
im
mp
pl
le
te
ex

11
TE
ex
xt
to
os
im
mp
pl
le
TE
ex
```

Ejemplo 2. Dado el contenido del fichero `data/SimpleText_no.txt`

```
Textos;imple_Tex ab
```

la salida es:

```
11
te
ex
xt
to
os
im
mp
pl
le
te
ex

11
te
ex
xt
to
os
im
mp
pl
le
te
ex
```

6 Configuración de las prácticas

Para la elaboración de la práctica dispone de una serie de ficheros que se encuentran en Language0_nb.zip. Una vez descomprimido va a encontrar entre otros: `Bigram.h, Bigram.cpp, main.cpp, documentation.doxy,`

`SimpleText.txt etc..` Para montar la primera práctica tendrá que crear un proyecto nuevo como se especifica en la sección 7. Pero antes, veamos cómo definir el espacio de trabajo.

El espacio de trabajo

ProyectosNetBeans : Carpeta raíz en la que se van a crear todos los proyectos de la asignatura. Tendremos un directorio por cada Language, que se genera en el momento de crear un proyecto NetBeans. La estructura que deberíamos de tener es la que se muestra a continuación.

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

Scripts: Una serie de scripts Bash de apoyo a las funciones de NetBeans. Se rellenará en prácticas sucesivas.

```
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

Para tener bien ordenado el contenido de cada proyecto Language, se va a organizar en las siguientes carpetas:

build Es una carpeta temporal que contiene código precompilado y pendiente de enlazar

dist Contiene los binarios ejecutables. Vamos a generar dos versiones por cada programa:

- **Release** Es la versión más eficiente y que menos espacio ocupa. Es la que debería entregarse al cliente.
- **Debug** Es la versión sobre la que se depuran errores y se corrigen defectos. Es más grande porque el binario contiene información extra para poder usar el depurador. Puede llegar a ser el doble de grande que la anterior.

data Ficheros de datos, bases de datos, ficheros de configuración que pueda necesitar el programa durante su ejecución, que no sean para testearla.

doc Aquí se almacena la documentación del proyecto, tanto del código, como la generada por los tests.

include Contiene los ficheros de cabeceras **.h**.

nbproject Es la carpeta que utiliza NetBeans para almacenar la configuración del proyecto

scripts En esta carpeta colocaremos los scripts de apoyo a Netbeans que se han desarrollado en esta asignatura y se puede ampliar con otros más. Se utilizarán en prácticas posteriores. Por ahora se hace todo manual.

src Contiene los principales ficheros fuente del proyecto **.cpp**, sin incluir los ficheros de test, que también son **.cpp**, que van en la siguiente carpeta.

zip Carpeta para dejar las copias de seguridad del proyecto.

7 Configurar el proyecto en NetBeans

Para crear un proyecto Language desde cero.

1. Crear el proyecto en NetBeans como `C++ Application` con nombre Language0 para esta práctica. Crear las carpetas: `include, src, data, doc, script, zip`, dentro del directorio Language0. Esto se puede hacer desde un terminal usando el comando `mkdir <nombreDir>` o bien desde el entorno de NetBeans, a través de la pestaña **File**. Es necesario añadir las carpetas una a una.

   Carpetas como `build, dist, nbproyect` son creadas y gestionadas automáticamente por NetBeans, no tocar.

2. Colocar cada uno de los ficheros en su sitio, el/los ficheros `*.h` en `include`, los `*.cpp` en `src`, etc. Estas acciones y las realizadas en el paso anterior se han llevado a cabo através del SO. Compruébalo con la instrucción unix: `ls`. La estructura debe ser como la indicada en la figura 2.

3. Propiedades de proyecto. Compilador

   (a) Poner el estándar a C++14

   (b) Añadir en `Include Directories ...` : el propio proyecto **./include**

4. Desde la vista lógica del proyecto.

   (a) En `Header Files`, `Add existing item`: fichero `Bigram.h`.

   (b) En `Source Files`, `Add existing item`: ficheros `main.cpp` y `Bigram.cpp`.

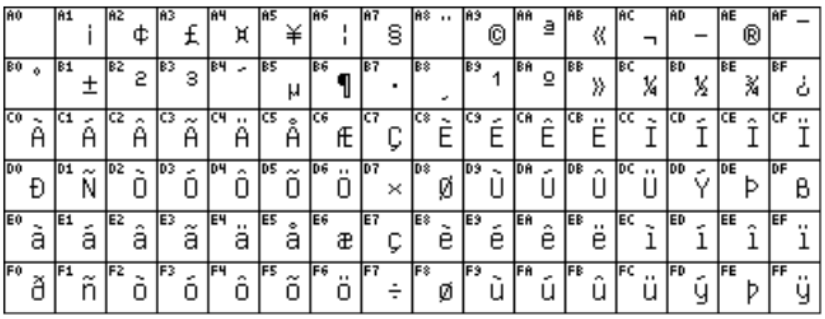
5. Sobre el nombre del proyecto, botón derecho `Set As Main Project` o desde el menú principal `Run` – `Set Main Project` y seleccionar este proyecto.

6. Con esto ya se podría ejecutar el proyecto normalmente. Obviamente, no hace nada porque todavía está vacío. A partir de aquí empezaría el desarrollo del proyecto para el que cada programador seguirá un orden determinado y, probablemente, diferente a todos los demás, hasta que la aplicación esté terminada. En ese momento habrá que probarla, para ver que todo funciona como se espera, en un proceso que, de nuevo, dependerá de cada programador en concreto.

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, através de **File** $\rightarrow$ **Export project** $\rightarrow$ **To zip**. El nombre, en esta ocasión es Language0.zip, sin más aditivos, es el mimo para todos los estudiantes. Puede guardar en la carpeta zip, el fichero que va a entregar en Prado.

8 La Codificación de caracteres ISO8859-1 y otros

Como ya se ha indicado, los ficheros de texto que vamos a considerar, están todos en la misma codificación. En concreto vamos a usar la codificación ISO 8859-1 (también conocida como Alfabeto Latino n.o 1 o ISO Latín 1.) que usa un *único* byte por cada carácter y que define la codificación del alfabeto latino, incluyendo los diacríticos (como letras acentuadas, ñ, ç), y letras especiales (como ß, Ø), necesarios para la escritura de las siguientes lenguas originarias de Europa occidental: afrikáans, alemán, español, catalán, euskera, aragonés, asturiano, danés, escocés, feroés, finés, francés, gaélico, gallego, inglés, islandés, italiano, holandés, noruego, portugués y sueco.



También lo podrá encontrar como norma ISO/IEC 8859-15 que consiste en una una revisión de la ISO 8859-1, que incorpora el símbolo del Euro y algunos caracteres necesarios para dar soporte completo al francés, finés y estonio. Esta norma es la que se encuentra en la mayoría de los editores actuales como los que se muestran en las imágenes siguientes.

Se ha elegido esta codificación para hacer la lectura de caracteres más simple, un carácter de un texto se corresponde con un byte/char.

Lo habitual cuando se trabaja con caracteres internacionales, la tabla

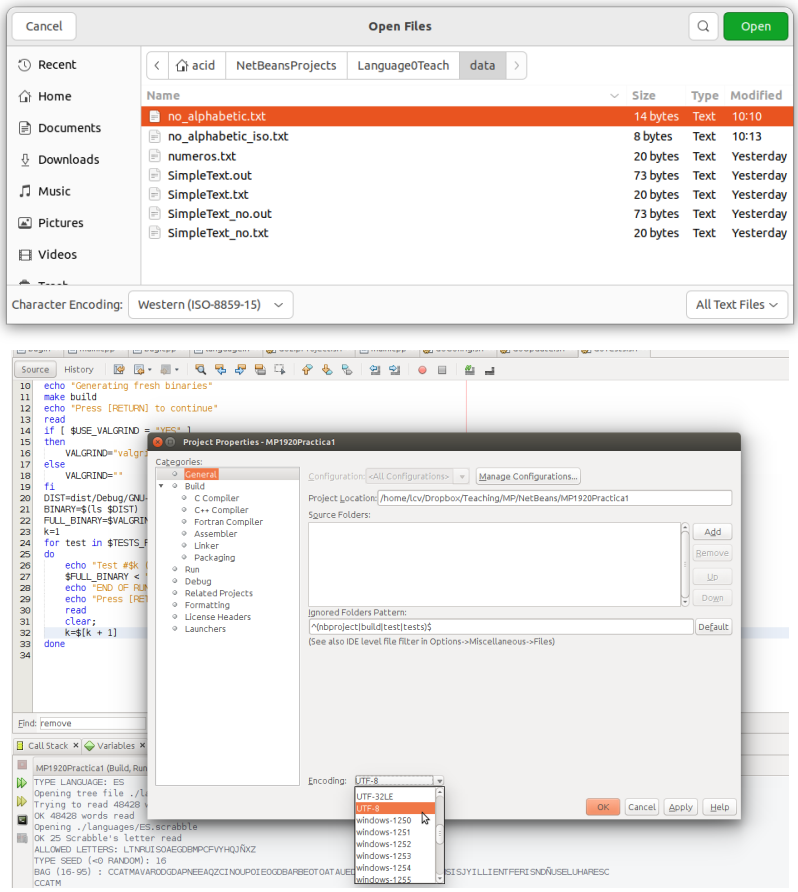
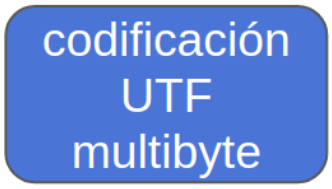


Figura 3: Selección de la codificación de caracteres en varios programas. Arriba el editor gedit, en el momento de abrir, abajo con NetBeans

de caracteres de 256 posiciones se queda muy corta y se usa una codificación alternativa. La más común en los Sistemas Operativos modernos es Unicode UTF[4], la cual puede usar 1, 2 o más bytes para representar un único caracter multinacional. Lo cual nos complicaría mucho la tarea a la hora de leer.



Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

[4]Codificación UTF8 (Abrir →)