Guion de prácticas

Práctica Final

Mayo de 2018

UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

Curso 2017/2018

Índice

1. Introducción

Los objetivos de este guion de prácticas son los siguientes:

1. Practicar con un problema en el que es necesaria la modularización. Para desarrollar los programas de esta práctica, el alumno debe crear distintos archivos, compilarlos, y enlazarlos para obtener los ejecutables.
2. Practicar con el uso de la memoria dinámica. El alumno deberá usar estructuras de datos que se alojan en memoria dinámica.
3. Profundizar en los conceptos relacionados con la abstracción de tipos de datos.
4. Practicar con el uso de clases como herramienta para implementar los tipos de datos donde se requiera encapsular la representación.
5. Usar los tipos que ofrece C++ para el uso de ficheros.

Los requisitos para poder realizar esta práctica son:

1. Saber manejar punteros, memoria dinámica, clases y ficheros.
2. Conocer el diseño de programas en módulos independientes, así como la compilación separada, incluyendo la creación de bibliotecas y de archivos makefile.
3. El alumno debe realizar esta práctica una vez que haya estudiado los contenidos sobre clases. La práctica está diseñada para que se lleve a cabo mientras se asimila la última parte de la asignatura (conceptos avanzados de clases y ficheros). Así, el alumno puede empezar a realizarla después de haber asimilado los conceptos básicos sobre clases, incluyendo constructores, destructores y sobrecarga del operador de asignación.
   Podrá observar que parte del contenido de esta práctica coincide con prácticas anteriores. Por lo que podrá reutilizar material ya elaborado y depurada, así el tiempo para la elaboración de la práctica final se verá reducido de forma significativa.

2. Programa learn. Aprendizaje de un idioma a partir de ficheros de texto de ejemplo

A partir de un conjunto de ficheros de texto escritos en un mismo idioma, el programa **learn** genera un fichero de salida que contiene la lista de los bigramas diferentes hallados en todos los ficheros de entrada de entrenamiento. Cada bigrama se obtiene a partir de cualesquiera dos caracteres que aparezcan juntos dentro de una palabra de un texto de entrada. La sintaxis del programa **learn** es como sigue:

```
learn {-c|-a} [-l nombreldioma] [-f ficheroSalida.bgr] texto1.txt {texto2.txt texto3.txt}
```

Los argumentos son:

-c | **-a** paramétro obligatorio, debe de figurar uno de los dos necesariamente:
-c crea un nuevo fichero de bigramas
-a añade los bigramas a fichero de bigramas
parámetros optativos:
-l indica el nombre del idioma del que se aprende (**unknown** por defecto)
-f indica el nombre del fichero de salida <**.bgr**> (**out.bgr** por defecto)

El programa **learn** acepta un número variable de ficheros de entrada <.txt>, siempre que al menos uno aparezca en la llamada. El fichero de salida <.bgr> sigue el formato introducido en la Práctica 5, contiene una lista formada por cada bigrama y el número de veces (frecuencia) que aparece en los ficheros de entrada. La principal novedad en esta práctica es que la lista de bigramas se guarda de forma ordenada por frecuencia de mayor a menor. Un ejemplo de ejecución podría ser el siguiente:

```
learn -c -l spanish -f spanish.bgr textoSpanish1.txt textoSpanish2.txt
```

El resultado de esta ejecución es un fichero en disco ("spanish.bgr" en el ejemplo), que contendrá la unión y el recuento de los bigramas que aparecen en los ficheros de entrada proporcionados. El formato del fichero **spanish.bgr** es exactamente el comentado en la Práctica 5 de la asignatura.

Hasta ahora se han utilizado las clases *Bigrama*, (clase base) e *Idioma* (vector de Bigramas) para realizar las operaciones de unión de Bigramas e Idiomas en las Prácticas 4 y 5.El objetivo de esta práctica es extraer los bigramas de un documento de texto (fichero.txt) siempre que el contenido de estos ficheros de texto se ajuste a las siguientes reglas.

- Los ficheros de texto que vamos a considerar, están todos en la misma codificación. En concreto recomendamos usar la codificación ISO 8859-15 (también conocida como Alfabeto Latino n.º 1 o ISO Latín 1.) que usa un 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.

- ¡Atención! Cualquier editor puede alterar la codificación de los ficheros, es necesario comprobar la codificación de los ficheros antes de aplicar `learn`, con el comando: $file$. Ejemplo:

  ```
  file --mime-encoding diderot.txt
  ```

  siendo válidas las siguientes clasificaciones

  ```
  us-ascii
  iso-8859-1
  ```

- Es posible que la visualización de un fichero con contenido ISO 8859-15 en distintos programas no sea la adecuada y se observen caracteres ilegibles. Las figuras en la sección 5 muestran como cambiar la codificación de algunos de ellos.
- Todos los caracteres se pasan a minúscula y se ignoran caracteres no válidos como " !=. , ; : etc. En el fichero **main.cpp** disponible para descarga en **DECSAI** aparece declarada una variable que contiene todos los caracteres válidos para cualquier idioma con codificación ISO 8859-15.
- Para procesar cada uno de los ficheros de entrenamiento, se van leyendo las palabras una a una ignorando los separadores (blanco, tabulador,
  n) y cualquier caracter no válido.

En la plataforma **DECSAI** podrán encontrar una serie de ficheros de texto de ejemplo en el formato indicado escritos en 3 idiomas diferentes. Todos ellos deberán incluirse en la carpeta **data**/ del proyecto.

- **aliceWonder.txt**
- **changedMan.txt**.
- **fortunata..txt**
- **lesMiserables.txt**
- **diderot.txt**
- **fortunata.txt**
- **quijote.txt**

3. Diseño propuesto

Se propone crear las siguientes clases para resolver el problema, cada una de ellas declarada en su respectivo fichero de cabeceras **.h** e implementada en su fichero **.cpp**:

Bigrama . Ya desarrollada en la Práctica 4 y empleada en las Prácticas 4 y 5.

Idioma . Ya desarrollada y empleada en la Práctica 5. En esta práctica se modificará la definición de la clase, cuyos datos miembro y métodos públicos y privados deberán ser según lo expuesto en la Sección 4.1 de este documento. Contiene el identificador del idioma y la lista de bigramas ordenados por frecuencia y permitirá leer y grabar su contenido en ficheros en disco. Esta clase deberá usarse para representar bigramas con frecuencias estrictamente mayores a 0.

ContadorBigramas . Esta es una nueva clase que hay que implementar según la declaración expuesta en la Sección 4.2. Encapsula una matriz 2D dinámica de enteros, con tantas filas y tantas columnas como caracteres válidos tengamos en un idioma, de forma que cada entero guarda la frecuencia del bigrama determinado por los dos caracteres que representa la fila y columna correspondiente. Así, la frecuencia del bigrama ”de“ estará almacenada en la fila que corresponda a la ”d“ y en la columna que corresponda a la ”e“. El número de filas y columnas de esta matriz bidimensional viene determinado por el número de caracteres válidos que se van a considerar. Como se ha comentado antes, en el fichero **main.cpp** aparece declarada e inicializada una variable **string** que contiene los caracteres válidos para cualquier idioma ISO 8859-15. Los métodos de esta clase permiten:

1. inicializar un objeto con los bigramas contenidos en un objeto de tipo **Idioma**
2. traducir la matriz bidimensional a una forma más compacta, es decir, a un objeto de tipo **Idioma**, ordenando previamente las frecuencias de mayor a menor y eliminando aquellos bigramas con frecuencia nula.

3.1. Un problema de ejemplo

Se proporciona el fichero **test.txt**, cuyo contenido es el siguiente:

```
Dónde digo: "digo", digo: ¿ Diego ?
```

Sea la ejecución siguiente

```
dist/Debug/GNU-Linux/learn -c -f unkown.bgr data/test.txt
```

Esta ejecución carga el texto del fichero **test.txt**, analiza los bigramas encontrados, ignorando caracteres no válidos, y genera un fichero **unknown.bgr** cuyo contenido es el siguiente (los bigramas empatados con la misma frecuencia podrían aparecer en cualquier orden relativo):

```
MP-BIGRAMAS_IDIOMA-T-1.0
unknown
9
di 4
go 4
ig 3
eg 1
de 1
ie 1
dó 1
nd 1
ón 1
```

3.2. Corrección de la práctica

Durante el desarrollo de la práctica, cada alumno deberá crear su propio proyecto en NetBeans desde cero, colocando cada fichero en su carpeta correspondiente y entregar un fichero ZIP exactamente con el mismoo formato que en las prácticas anteriores (copiar la regla zip: de los makefiles de prácticas anteriores). No se corregirá ninguna práctica que no cumpla esta condición.

Descompresión y compilación

```
$$ unzip practica6.zip
Archive:  practica6.zip
  inflating: Makefile
   creating: build/
   creating: data/
  inflating: data/aliceWonder.txt
  inflating: data/fortunata.txt
  inflating: data/diderot.txt
  inflating: data/test.txt
  inflating: data/changedMan.txt
  inflating: data/quijote.txt
  inflating: data/lesMiserables.txt
   creating: dist/
   creating: doc/
  inflating: doc/learn.doxy
   creating: include/
  inflating: include/Idioma.h
  inflating: include/Bigrama.h
  inflating: include/ContadorBigramas.h
   creating: nbproject/
  inflating: nbproject/configurations.xml
  inflating: nbproject/Makefile-Debug.mk
  inflating: nbproject/Makefile-impl.mk
  inflating: nbproject/Package-Release.bash
  inflating: nbproject/Makefile-Release.mk
   creating: nbproject/private/
  inflating: nbproject/Makefile-variables.mk
  inflating: nbproject/project.xml
  inflating: nbproject/Package-Debug.bash
   creating: src/
  inflating: src/Idioma.cpp
  inflating: src/learn.cpp
  inflating: src/ContadorBigramas.cpp
  inflating: src/Bigrama.cpp
   creating: zip/

$$ make
"make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-conf
make[1]: se entra en el directorio '/home/lcv/Tmp/learn'
"make"  -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux/learn
make[2]: se entra en el directorio '/home/lcv/Tmp/learn'
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/Bigrama.o.d"
g++    -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/src/Bigrama.o.d"
-o build/Debug/GNU-Linux/src/Bigrama.o src/Bigrama.cpp
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/ContadorBigramas.o.d"
g++    -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/src/ContadorBigramas.o.d"
-o build/Debug/GNU-Linux/src/ContadorBigramas.o src/ContadorBigramas.cpp
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/Idioma.o.d"
g++    -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/src/Idioma.o.d"
-o build/Debug/GNU-Linux/src/Idioma.o src/Idioma.cpp
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/learn.o.d"
g++    -c -g -Iinclude -MMD -MP -MF "build/Debug/GNU-Linux/src/learn.o.d"
-o build/Debug/GNU-Linux/src/learn.o src/learn.cpp
mkdir -p dist/Debug/GNU-Linux
g++     -o dist/Debug/GNU-Linux/learn build/Debug/GNU-Linux/src/Bigrama.o
build/Debug/GNU-Linux/src/ContadorBigramas.o build/Debug/GNU-Linux/src/Idioma.o
build/Debug/GNU-Linux/src/learn.o
make[2]: se sale del directorio '/home/lcv/Tmp/learn'
make[1]: se sale del directorio '/home/lcv/Tmp/learn'
```

Primer caso de prueba

```
$$ dist/Debug/GNU-Linux/learn -c -f unkown.bgr data/test.txt
...

$$ more unkown.bgr
MP-BIGRAMAS_IDIOMA-T-1.0
unknown
9
di 4
go 4
ig 3
eg 1
de 1
ie 1
dó 1
nd 1
ón 1
```

Segundo caso de prueba

```
$$ dist/Debug/GNU-Linux/learn -c -l spanish -f spanish.bgr data/fortunata.txt data/quijote.txt
...

$$ more spanish.bgr
MP-BIGRAMAS_IDIOMA-T-1.0
spanish
630
de 66481
en 66090
ue 65446
es 61431
qu 58701
er 54288
la 52087
os 47247
ra 45034
as 42656
an 41196
...
```

4. Apéndice 1. Código de ejemplo

4.1. Idioma.h

Nuevo diseño de la clase **Idioma**.

```
#ifndef IDIOMA_H
#define IDIOMA_H

#include "Bigrama.h"

class Idioma {
public:

    Idioma();
    Idioma(int nbg);
    Idioma(const Idioma& orig);
    ~Idioma();
    Idioma& operator=(const Idioma& orig);
    std::string getIdioma() const;
    void setIdioma(const std::string& id);
    Bigrama getPosicion(int p) const;
    void setPosicion(int p, const Bigrama & bg);
    inline int getSize() const { return _nBigramas; };
    int findBigrama(const std::string& bg) const;
    double distancia(const Idioma& otro) const;
    void ordenar();
    bool salvarAFichero(const char *fichero) const;
    bool cargarDeFichero(const char *fichero);

    friend std::ostream & operator<<(std::ostream & os, const Idioma & i);
    friend std::istream & operator>>(std::istream & os, Idioma & i);

private:
    std::string _idioma;
    Bigrama* _conjunto;
    int _nBigramas;

    void reservarMemoria(int n);
    void liberarMemoria();
    void copiar(const Idioma& otro);
};

std::ostream & operator<<(std::ostream & os, const Idioma & i);
std::istream & operator>>(std::istream & is, Idioma & i);
#endif
```

4.2. ContadorBigramas.h

```
#ifndef CONTADOR_BIGRAMAS_H
#define CONTADOR_BIGRAMAS_H

#include "Idioma.h"

class ContadorBigramas {
public:
    ContadorBigramas(const std::string& caracteresValidos);
    ContadorBigramas(const ContadorBigramas & orig);
    ~ContadorBigramas();
    int getSize() const;
    int getBigramasActivos() const;
    bool addBigrama(const char cadena[], int frecuencia=0);

    ContadorBigramas& operator=(const ContadorBigramas& orig);
    ContadorBigramas& operator+=(const ContadorBigramas& rhs);

    bool calcularFrecuenciasBigramas(const char* nfichero);
    Idioma toIdioma() const;
    void fromIdioma(const Idioma &i);
private:
    int** _bigramas;
    std::string _caracteresValidos;

    void reservarMemoria(int n);
    void liberarMemoria();
    void copiar(const ContadorBigramas & otro);
};

#endif
```

5. Apéndice 2. Cambiando la codificación de los programas

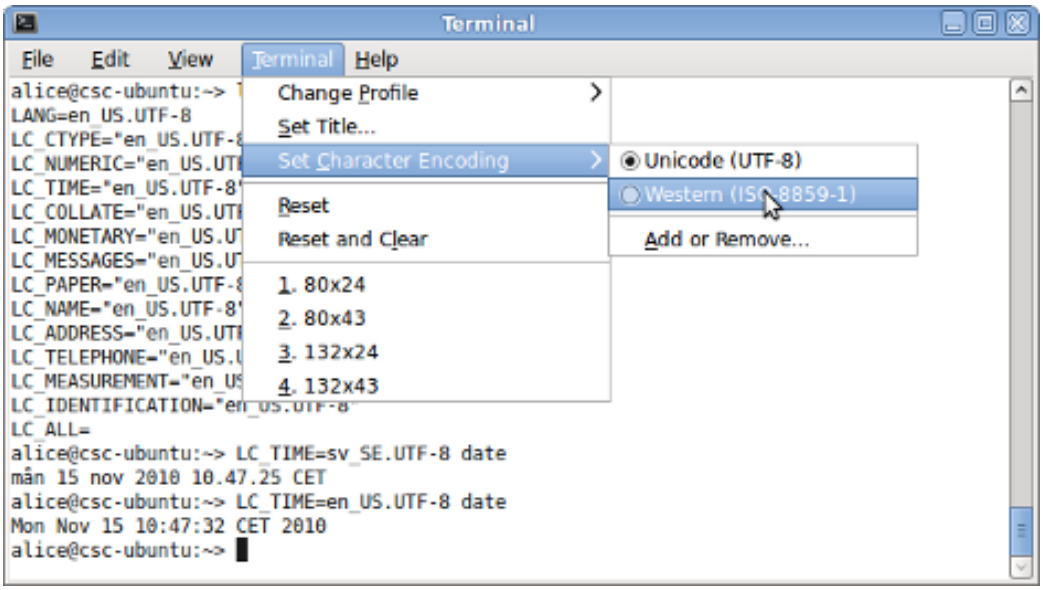

Figura 1: Cambiando la codificación de una terminal

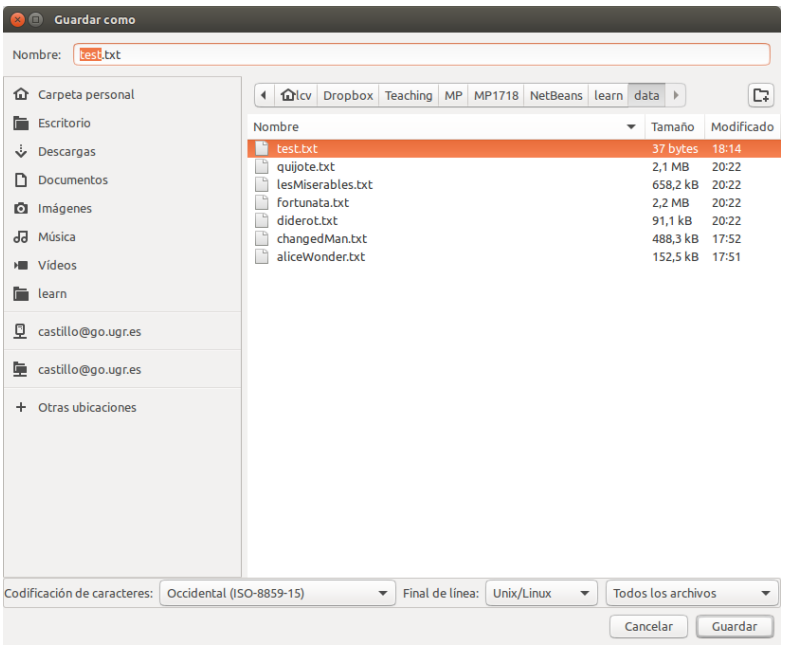

Figura 2: Cambiando la codificación de un fichero desde **gedit** mediante la opción ”Guardar como“. Abajo a la izquierda.

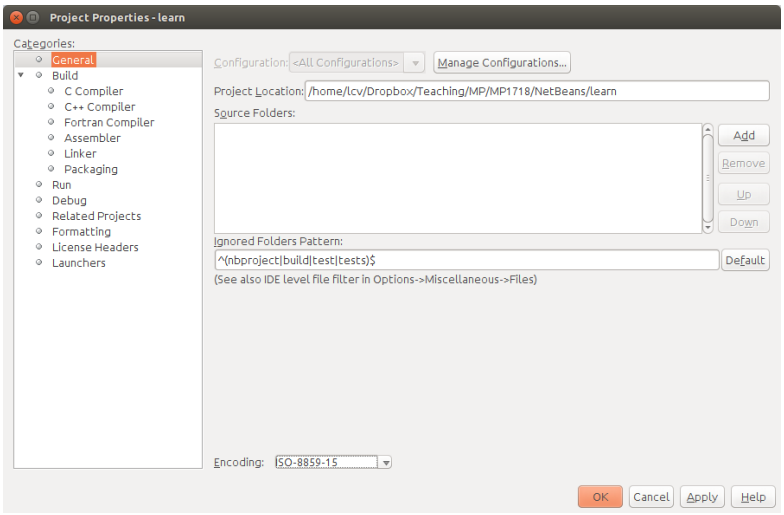

Figura 3: Cambiando la codificación de un proyecto en NetBeans desde la ventana de propiedades del proyecto. Abajo a la izquierda.

6. Apéndice 3. Un poco de ayuda

6.1. ¿Cómo realizar el aprendizaje del Idioma?

Para realizar el aprendizaje debemos abrir, uno a uno, los ficheros de texto que contienen el documento de entrada. Una vez abierto, pasamos a recorrerlo identificando los pares de caracteres consecutivos en el mismo. Estos caracteres pueden ser letras, dígitos, separadores (blanco, tabulador, salto de línea,) o símbolos de puntuación (comas, puntos, puntos y comas, etc.)

Una vez que tenemos un par de caracteres, deberemos chequear si forman un bigrama válido. En este sentido, se dice que un bigrama es válido si se encuentra compuesto de dos caracteres que pertenecen a un conjunto de caracteres predefinidos, por ejemplo $validos = \{\texttt{abcdefghij}\}$. Así, si el contenido del fichero es el siguiente:

```
gafa: –fachada–, hija.
```

identificaremos los bigramas: `ga af fa fa ac ch .... ja`

Para cada bigrama en el fichero (par de caracteres válidos) debemos de calcular la frecuencia de aparición. Para facilitar este proceso debemos almacenar las frecuencias en una matriz bidimensional, `F`, de tamaño $n \times n$, siendo n el número de caracteres válidos. En la tabla siguiente podemos ver un esquema de la matriz asociada a nuestro ejemplo, con $n = 10$. Inicialmente, todas las posiciones de la matriz tendrán el valor cero.

En la posición $F[i][j]$ se almacenará la frecuencia con la que aparece el bigrama que tiene como primer carácter el elemento (i-1)-ésimo y como segundo carácter el elemento (j-1)-ésimo en el conjunto $validos$, respectivamente. Así, el bigrama `ab` se almacena en la posición $F[0][1]$ y el bigrama `ch` se almacena en la posición $F[2][7]$. Por tanto, dado un nuevo bigrama, podemos actualizar su frecuencia fácilmente si conocemos las posiciones de los caracteres en el conjunto $valido$ e incrementamos en uno su valor.

F	a 0	b 1	c 2	d 3	e 4	f 5	g 6	h 7	i 8	j 9
a 0	1	2	3	0	3	2	6	8	9	0
b 1	3	2	6	8	9	0	3	0	3	2
c 2	1	2	0	0	3	0	4	1	0	0
..	.....									
j 9	4	0	0	2	6	8	5	2	0	0

Una vez que se han procesado todos los ficheros de entrada, podemos salvar el fichero idioma salida transformando esta tabla en un vector de bigramas, que será ordenado por frecuencia de aparición de los bigramas y los volcamos en el fichero de salida correspondiente.

6.2. ¿Cómo actualizar un fichero de idioma?

Cuando llamamos al programa `learn` con la opción de actualizar, por ejemplo

```
learn –a –l ingles –f ingles.bgr documentoIngles1.txt documentoIngles2.txt
```

el objetivo es actualizar la frecuencias considerando las frecuencias de los bigramas que aparecen en `documentoIngles1.txt documentoIngles2.txt`. En este caso, el proceso es ligeramente distinto, por lo que lo detallaremos a continuación:

En un primer paso debemos abrir el fichero de idioma `ingles.bgr` y cargar dicho idioma en memoria, en un objeto `idioma`. A partir de dicho idioma debemos de obtener la matriz bidimensional de frecuencias F, donde $F[i][j]$ representa el valor actual del bigrama asociado a la secuencia de caracteres (i-1)-ésimo, (j-1)-ésimo del conjunto de caracteres válidos.

A partir de aquí se sigue el mismo procesamiento que en el caso anterior. Notar que de forma abstracta, lo único que hacemos es inicializar el contador de bigramas, F, con los datos obtenidos a partir de un fichero Idioma.