



## **Práctica Final: Letras**

Dpto. Ciencias de la Computación e Inteligencia Artificial  
E.T.S. de Ingenierías Informática y de Telecomunicación  
Universidad de Granada



## **Estructuras de Datos**

Doble Grado en Ingeniería Informática y ADE  
Doble Grado en Ingeniería Informática y Matemáticas  
Grado en Ingeniería Informática

# Índice de contenido

1.Introducción.....	3
2.Objetivo.....	3
3.Ejercicios.....	3
3.1. Tareas a realizar.....	3
3.1.1.Test diccionario.....	3
3.1.2.Letras.....	4
3.1.3.Cantidad de Letras.....	6
3.2.Ficheros.....	7
3.2.1.Fichero diccionario.....	7
3.2.2.Fichero Letras.....	7
4.Módulos a desarrollar.....	8
5.Práctica a entregar.....	12
6.Valoración.....	12
7.Referencias.....	13



# 1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

- Resolver un problema eligiendo la mejor estructura de datos para las operaciones que se solicitan

Los requisitos para poder realizar esta práctica son:

1. Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos
2. Haber estudiado el Tema 2: Abstracción de datos. Templates.
3. Haber estudiado el Tema 3: T.D.A. Lineales.
4. Haber estudiado el Tema 4: STL e Iteradores.
5. Haber estudiado estructuras de datos no lineales

## 2. Objetivo.

El objetivo de esta práctica es llevar a cabo el análisis, diseño e implementación de un proyecto. Con tal fin el estudiante abordará un problema donde se requiere estructuras de datos que permiten almacenar grandes volúmenes de datos y poder acceder a ellos de la forma más eficiente.

## 3. Ejercicios

El estudiante deberá implementar varios programas. Uno de ellos simulará parte del juego “Cifras y Letras”. En este caso solamente deberá generar la sección de Letras. Este juego consiste en dada una serie de letras escogidas de forma aleatoria, obtener la palabra existente en un diccionario formada a partir de ellas de mayor longitud o que tenga la mayor puntuación.

Por ejemplo dadas las siguientes letras:

O   D   Y   R   M   E   T

una de las mejores soluciones sería METRO.

### 3.1. Tareas a realizar

Se deberán llevar a cabo las siguientes tareas:

1. Definir los T.D.A. que vea necesario para la solución del problema propuesto.
2. Probar los módulos con programas test.
3. Construir los programas que a continuación se detallan.

**Se puede usar la STL en todos los módulos**

A continuación se detallan los programas que se deberán desarrollar.

#### 3.1.1. Test diccionario

Este programa permitirá comprobar el buen funcionamiento del T.D.A Diccionario, representado como un set. Para ello el código fuente “testdiccionario.cpp” deberá funcionar

con el T.D.A Diccionario que se desarrolle (ver sección 4).

```
1.  #include <fstream>
2.  #include <iostream>
3.  #include <string>
4.  #include <vector>
5.  #include <set>
6.  int main(int argc, char * argv[]){
7.      if (argc!=2){
8.          cout<<"Los parametros son:"<<endl;
9.          cout<<"1.- El fichero con las palabras";
10.         return 0;
11.     }
12.     ifstream f(argv[1]);
13.     if (!f){
14.         cout<<"No puedo abrir el fichero
15.         "<<argv[1]<<endl;
16.         return 0;
17.     }
18.     Diccionario D;
19.     cout<<"Cargando diccionario...."<<endl;
20.     f>>D;
21.     cout<<"Leido el diccionario..."<<endl;
22.     cout<<D;
23.     int longitud;
24.     cout<<"Dime la longitud de las palabras que
25.     quieres ver";
26.     cin>>longitud;
27.     vector<string>
28.     v=D.PalabrasLongitud(longitud);
29.     cout<<"Palabras de Longitud
30.     "<<longitud<<endl;
31.     for (unsigned int i=0;i<v.size();i++)
32.         cout<<v[i]<<endl;
33.     string p;
34.     cout<<"Dime una palabra: ";
35.     cin>>p;
36.     if (D.Esta(p)) cout<<"Sí esa palabra existe";
37.     else cout<<"Esa palabra no existe";
38. }
```

En este código, se carga el diccionario en memoria y luego se imprime por la salida estándar. A continuación se muestran todas las palabras del diccionario de una longitud dada. Y finalmente dada una palabra por el usuario, el programa indica si existe tal palabra en el diccionario o no. El estudiante creará por lo tanto el programa testdiccionario, que se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
prompt% testdiccionario spanish
```

El único parámetro que se da al programa es el nombre del fichero donde se almacena el diccionario.

### 3.1.2. Letras

Este programa construye las palabras de longitud mayor (o de puntuación mayor) de un diccionario a partir de una serie de letras seleccionadas de forma aleatoria. El programa letras se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
prompt% letras spanish letras.txt 8 L
```

Los parámetros de entrada son los siguientes:

1. El nombre del fichero con el diccionario
2. El nombre del fichero con las letras
3. El número de letras que se deben generar de forma aleatoria

#### 4. Modalidad de juego:

- Longitud: Si el parámetro es  $L$  se buscará la palabra más larga.
- Puntuación: Si el parámetro es  $P$  se obtendrá la palabra de mayor puntuación.

Tras la ejecución en pantalla aparecerá lo siguiente:

```
Las letras son: T      I      E      O      I      T      U      S
Dime tu solucion:tieso
tieso  Puntuacion: 5

Mis soluciones son:
otitis Puntuacion: 6
tiesto Puntuacion: 6
Mejor Solucion:tiesto
¿Quieres seguir jugando[S/N]?N
```

En primer lugar el programa genera 8 letras. Estas letras se escogen, de forma aleatoria, entre las dadas en el fichero letras (ver sección 3.2.2 ). Una vez generadas las letras, el programa pide al usuario su solución, en el ejemplo la solución dada es “tieso”. A continuación se muestra la solución del usuario junto con su puntuación. Y finalmente se muestran las soluciones dadas por el programa. Para generar de forma aleatoria las letras con la que construir la palabra, el estudiante creará una *Bolsa de Letras* (contenedor de letras que se disponen de forma aleatoria) en la que el número de veces que aparece cada letra, en la *Bolsa de Letras*, viene dado por la columna *Cantidad* del fichero de letras.

En el caso de que el usuario haya escogido jugar en modo “Puntuacion”, como resultado se

```
*****Puntuaciones Letras*****      Z      10
A      1
B      3
C      3
D      2
E      1
F      4
G      2
H      4
I      1
J      8
L      1
M      3
N      1
O      1
P      3
Q      5
R      1
S      1
T      1
U      1
V      4
Y      4

Las letras son:
N      S      A      O      T      O      A      I
Dime tu solucion:sonata

sonata Puntuacion: 6
Mis Soluciones son:
asiano Puntuacion: 6
atonia Puntuacion: 6
ostion Puntuacion: 6
sonata Puntuacion: 6
sotana Puntuacion: 6
sotani Puntuacion: 6
sotano Puntuacion: 6
tisana Puntuacion: 6
toison Puntuacion: 6
Mejor Solucion:toison
¿Quieres seguir jugando[S/N]?N
```

obtendrán las palabras que acumulen una mayor puntuación. Para obtener la puntuación de la palabra simplemente tenemos que sumar las puntuaciones de las letras en la palabra (en el fichero de Letras la puntuación de cada letra viene dada en la columna Puntos).

En ambas versiones, se le preguntará al usuario si quiere seguir jugando. En caso afirmativo

se generará una nueva secuencia de letras aleatorias para jugar de nuevo. En otro caso el programa termina.

En la sección 3.2 se detallan los formatos de los ficheros de entrada al programa.

### 3.1.3. Cantidad de Letras

El programa `cantidad_letras` obtiene la cantidad de instancias de cada letra (ver fichero `letras` en la sección 3.2.2 ). El programa se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
prompt% cantidad_letras spanish letras.txt salida.txt
```

Los parámetros de entrada son los siguientes:

1. El nombre del fichero con el diccionario
2. El nombre del fichero con las letras
3. El fichero donde escribir el conjunto de letras con la cantidad de apariciones calculada.

Este programa una vez haya cargado el fichero diccionario en memoria y el conjunto de letras, obtiene para cada letra en el conjunto el número de veces que aparece en el diccionario, es decir encuentra la frecuencia de aparición (Fabs.). Finalmente obtiene el tanto por ciento, sobre el total de las frecuencias, del número de veces que aparece cada letra. Este valor será la cantidad (Frel.).

A modo de ejemplo para la anterior ejecución, el fichero `salida.txt` obtenido es:

#Letra	Fabs.	Frel.
A	16	0,14
B	2	0,018
C	6	0,053
D	5	0,044
E	10	0,088
F	2	0,018
G	2	0,018
H	1	0,009
I	9	0,079
J	1	0,009
L	5	0,044
M	4	0,035
N	7	0,061
O	10	0,088
P	3	0,026
Q	1	0,009
R	10	0,088
S	5	0,044
T	6	0,053
U	4	0,035
V	2	0,018
X	1	0,009
Y	1	0,009
Z	1	0,009

## 3.2. Ficheros

### 3.2.1. Fichero diccionario

El fichero diccionario se componen de un conjunto de palabras, cada una en un línea. Este conjunto de palabras serán las palabras que se consideren como válidas.

Un ejemplo de fichero diccionario es el siguiente:

```
a
aaronica
aaronico
ab
abab
ababillarse
ababol
abaca
abacera
abaceria
abacero
abacial
abaco
abad
abada
abadejo
abadenga
abadengo
```

### 3.2.2. Fichero Letras

Un ejemplo de fichero de letras es el que se muestra a la derecha.

El formato del fichero es el siguiente:

1. En primer lugar aparece una línea encabezada con el carácter # donde se describe las columnas del fichero (Letra Cantidad Puntos)
2. A continuación cada línea se corresponde con la información de una letra:
  - Valor de la letra
  - Número de veces que aparece la letra en la Bolsa de Letras
  - Puntos asignados a la letra.

#Letra	Cantidad	Puntos
A	12	1
E	12	1
O	9	1
I	6	1
S	6	1
N	5	1
L	1	1
R	6	1
U	5	1
T	4	1
D	5	2
G	2	2
C	5	3
B	2	3
M	2	3
P	2	3
H	2	4
F	1	4
V	1	4
Y	1	4
Q	1	5
J	1	8
X	1	8
Z	1	10

## 4. Módulos a desarrollar

### Módulo Diccionario

Será necesario construir el T.D.A Diccionario. Un objeto del T.D.A. Diccionario almacena palabras de un lenguaje. El T.D.A Diccionario será representado como un **set** *instanciado a string* . Así en líneas generales el módulo Diccionario se detalla a continuación:

```
#ifndef __Diccionario_h__
#define __Diccionario_h__
#include <set>

class Diccionario{
private:
    set<string> datos;
public:
    /**
     * @brief Construye un diccionario vacío.
     */

    Diccionario()
    /**
     * @brief Devuelve el numero de palabras en el diccionario
     */
    int size() const ;

    /**
     * @brief Obtiene todas las palabras en el diccionario de un longitud dada
     * @param longitud: la longitud de las palabras de salida
     * @return un vector con las palabras de longitud especifica en el parametro de entrada
     */
    vector<string> PalabrasLongitud(int longitud);

    /**
     * @brief Indica si una palabra está en el diccionario o no
     * @param palabra: la palabra que se quiere buscar
     * @return true si la palabra esta en el diccionario. False en caso contrario
     */
    bool Esta(string palabra);

    /**
     * @brief Lee de un flujo de entrada un diccionario
     * @param is:flujo de entrada
     * @param D: el objeto donde se realiza la lectura.
     * @return el flujo de entrada
     */
    friend istream & operator>>(istream & is,Diccionario &D);

    /**
     * @brief Escribe en un flujo de salida un diccionario
     * @param os:flujo de salida
     * @param D: el objeto diccionario que se escribe
     * @return el flujo de salida
     */
    friend ostream & operator<<(ostream & os, const Diccionario &D);
};
```



```
#endif
```

Podríamos generar un iterador sobre el T.D.A Diccionario que nos permita recorrer de manera ordenada todas las palabras del diccionario. Una posible especificación y representación de este iterador es:

```
#ifndef __Diccionario_h__
```

```
#define __Diccionario_h__
```

```
#include <set>
```

```
class Diccionario{
```

```
private:
```

```
    set<string> datos;
```

```
public:
```

```
    ...
```

```
class iterator{
```

```
private:
```

```
    set<string>::iterator it;
```

```
public:
```

```
    iterator ();
```

```
    string operator *();
```

```
    iterator & operator ++();
```

```
    bool operator ==(const iterator &i)
```

```
    bool operator !=(const iterator &i)
```

```
    friend class Diccionario;
```

```
};
```

```
iterator begin();
```

```
iterator end();
```

```
};
```

```
#endif
```

## Módulo Letra y Conjunto de Letras

El T.D.A Letra almacena una letra. Una letra se especifica con tres valores:

1. El carácter de la propia letra
2. La cantidad de veces que puede aparecer.
3. La puntuación de una letra.

El T.D.A Conjunto\_Letras permitirá tener en memoria un fichero Letras. Este T.D.A se define como una colección de letras, en las que no hay letras repetidas.

## Módulo Bolsa de Letras

Este módulo será útil para el programa *letras*. El T.D.A Bolsa\_Letras almacena caracteres correspondientes a una letra de un Conjunto de Letras. Este carácter aparece en la Bolsa\_Letras repetido tantas veces como diga el campo cantidad de la letra. Por lo tanto en la Bolsa de Letras aparecen las letras de forma aleatoria. En el programa "*letras*" la secuencia de letras con las que se juega se cogen de la Bolsa de Letras.

## 5. Práctica a entregar

El estudiante deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre *“letras.tgz”* y entregarlo antes de la fecha que se publicará en la página web de la asignatura. Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta datos. Es recomendable que haga una “limpieza” para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

El estudiante debe incluir el archivo *Makefile* para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios:

letras	— include	<i>Ficheros de cabecera (.h)</i>
	— src	<i>Código fuente (.cpp)</i>
	— obj	<i>Código objeto (.o)</i>
	— lib	<i>Bibliotecas</i>
	— doc	<i>Documentación</i>
	— bin	<i>Ficheros ejecutables</i>
	— datos	<i>Fichero de datos.</i>

Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta *“letras”*) para ejecutar:

```
prompt% tar zcv letras.tgz letras
```

*tras lo cual, dispondrá de un nuevo archivo letras.tgz que contiene la carpeta letras así como todas las carpetas y archivos que cuelgan de ella.*

***La práctica se puede realizar por parejas. Ambos miembros de la pareja deberán subir los ficheros a prado a través del enlace correspondiente a la entrega de la práctica final. Se deberá incluir un fichero readme.txt en el que se indique exactamente qué se ha hecho y cómo se ha hecho, y si hay algo que no funciona.***

## 6. Valoración

*Se podrá alcanzar una máxima puntuación en la práctica de 1.5 puntos*

## 7. Referencias

- Garrido, A., Fdez-Valdivia, J. *“Abstracción y estructuras de datos en C++”*. Delta publicaciones, 2006.
- Garrido, A. *Estructuras de datos avanzadas: con soluciones en C++*. Editorial Universidad de Granada. 2018