

Práctica final - Cifras y Letras

Elena Arroyo Zamora, Carlos Castillo Sánchez



**UNIVERSIDAD
DE GRANADA**



Doble Grado en Ingeniería Informática y Matemáticas

Universidad de Granada

December 9, 2025

1 Prefacio	1
1.1 Introducción	1
1.1.1 Juego de las cifras	1
1.1.2 Juego de las letras	1
1.2 Ejecutables	2
1.2.1 Juego de las cifras	2
1.2.1.1 cifras	2
1.2.2 Juego de las letras	2
1.2.2.1 cantidad_letras	2
1.2.2.2 letras	2
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 BolsaLetras Class Reference	7
4.1.1 Detailed Description	7
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 BolsaLetras()	8
4.1.3 Member Function Documentation	8
4.1.3.1 add()	8
4.1.3.2 cargarBolsa()	8
4.1.3.3 get()	8
4.1.3.4 size()	9
4.1.3.5 toString()	9
4.1.4 Friends And Related Symbol Documentation	9
4.1.4.1 operator<<	9
4.1.4.2 operator>>	10
4.2 ConjuntoLetras Class Reference	10
4.2.1 Detailed Description	11
4.2.2 Member Function Documentation	11
4.2.2.1 begin() [1/2]	11
4.2.2.2 begin() [2/2]	11
4.2.2.3 end() [1/2]	11
4.2.2.4 end() [2/2]	12
4.2.2.5 insert()	12
4.2.3 Friends And Related Symbol Documentation	12
4.2.3.1 operator<<	12
4.2.3.2 operator>>	12
4.3 ConjuntoLetras::const_iterator Class Reference	13

4.3.1 Detailed Description	13
4.4 Diccionario Class Reference	13
4.4.1 Detailed Description	14
4.4.2 Member Function Documentation	14
4.4.2.1 begin()	14
4.4.2.2 end()	14
4.4.2.3 Esta()	14
4.4.2.4 PalabrasLongitud()	15
4.4.3 Friends And Related Symbol Documentation	15
4.4.3.1 operator<<	15
4.4.3.2 operator>>	15
4.5 ConjuntoLetras::iterator Class Reference	17
4.5.1 Detailed Description	17
4.6 Diccionario::iterator Class Reference	17
4.6.1 Detailed Description	18
4.6.2 Member Function Documentation	18
4.6.2.1 operator!=(())	18
4.6.2.2 operator*()	19
4.6.2.3 operator++()	19
4.6.2.4 operator==(())	19
4.6.3 Friends And Related Symbol Documentation	20
4.6.3.1 Diccionario	20
4.7 Letra Class Reference	20
4.7.1 Detailed Description	21
4.7.2 Constructor & Destructor Documentation	21
4.7.2.1 Letra() [1/2]	21
4.7.2.2 Letra() [2/2]	21
4.7.3 Member Function Documentation	21
4.7.3.1 getCantidad()	21
4.7.3.2 getCaracter()	22
4.7.3.3 getPuntuacion()	22
4.7.3.4 operator<()	22
4.7.3.5 setCantidad()	22
4.7.3.6 setCaracter()	23
4.7.3.7 setPuntuacion() [1/2]	23
4.7.3.8 setPuntuacion() [2/2]	23
4.7.4 Friends And Related Symbol Documentation	23
4.7.4.1 operator<<	23
4.7.4.2 operator>>	24
5 File Documentation	25
5.1 dictionary.h	25

5.2 letters_bag.h	26
5.3 letters_set.h	26
Index	29

Chapter 1

Prefacio

1.1 Introducción

Este proyecto consiste en el planteamiento, especificación e implementación del juego "Cifras y Letras". Está enmarcado dentro del plan de estudios de la asignatura "Estructuras de Datos", del doble grado en Ingeniería Informática y Matemáticas por la Universidad de Granada. El objetivo de estos programas es el uso de los contenedores que ofrece la Standard Template Library (STL), así como el manejo de los recursos de programación ofrecidos en las lecciones: uso de clases iteradoras, creación de clases a partir de contenedores de la STL, empleo de la recursividad como solución a ciertos problemas, etc.

En este caso, el proyecto está dividido en dos partes, una para cada juego. De este modo, se tienen los siguientes *sub-juegos*.

1.1.1 Juego de las cifras

Este juego consiste en averiguar la mejor combinación de números de entre un conjunto dado junto con las cuatro operaciones básicas (a saber, suma, resta, producto, división entera) para obtener un número de 3 cifras dado. Los números básicos son $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100\}$, de los que se tomarán solamente 6 (con posibilidad de repetición). A la hora de hacer los cálculos, solo se podrá tomar cada elemento una sola vez (por lo tanto, se podrán hacer hasta 5 operaciones). El programa calcula una combinación de operandos y operaciones y da la solución paso a paso. En este caso, no solicita ninguna entrada por parte del usuario.

1.1.2 Juego de las letras

Este juego consiste en extraer de un conjunto de letras, la palabra con mayor puntuación posible y que forme parte de un diccionario especificado. Esta puntuación se calcula según **dos modalidades de juego**, lo cual se debe expresar con un argumento al ejecutarlo. Si la modalidad es **L (longitud)**, la palabra con más puntuación es la más larga. Si la modalidad es **P (puntuación)**, esta se calcula según el número de veces que aparezca cada caracter en el diccionario: a menor apariciones, mayor puntuación (se premia lo infrecuente). La **cantidad de letras** de las que se tiene que obtener la palabra es un parámetro que se expresa al ejecutar el programa. En este juego, se solicita al usuario que escriba su solución, y luego se dan otras soluciones iguales o mejores. Para llevar a cabo este juego, hay que ejecutar un programa previamente que cargue en un objeto de la clase [Diccionario](#) las posibles palabras que se pueden considerar válidas y que obtenga las puntuaciones de las letras según su frecuencia.

1.2 Ejecutables

1.2.1 Juego de las cifras

1.2.1.1 cifras

Este programa no necesita de ningún parámetro. Es un juego sencillo, y todo se realiza desde el propio programa y con la salida estándar, sin necesidad de ficheros externos.

```
./cifras
```

1.2.2 Juego de las letras

1.2.2.1 cantidad_letras

Este programa auxiliar carga en un objeto de la clase [Diccionario](#) las posibles palabras válidas y obtiene las puntuaciones de las letras según su frecuencia. La ejecución se realiza en terminal de la siguiente manera:

```
./cantidad_letras ./doc/diccionario.txt ./doc/letras.txt salida.txt
```

i) `./doc/diccionario.txt`: Corresponde con un fichero en el que aparecen todas las palabras válidas. El formato es una palabra por cada línea. De este fichero se obtendrá un objeto de la clase [Diccionario](#).

ii) `./doc/letras.txt`:

iii) `salida.txt`: Fichero de salida. Si no existe, se crea. Corresponde con un fichero del tipo "CantidadLetras", es decir, un fichero con tres columnas: la primera para los caracteres que aparecen, la segunda para la frecuencia absoluta de estos, y la tercera para la puntuación correspondiente. De este fichero se obtendrá en el programa principal un objeto de la clase [ConjuntoLetras](#).

1.2.2.2 letras

Este es el ejecutable principal del juego. Espera el diccionario, el conjunto de letras, la modalidad de juego (L o P) y la cantidad de letras a usar.

```
./letras ./doc/diccionario.txt ./doc/letras.txt [L|P] n
```

i) `./doc/diccionario.txt`: Fichero correspondiente a un diccionario.

ii) `./doc/letras.txt`: Fichero de tipo "CantidadLetras" para obtener un objeto de la clase `CantidadLetras` (`set<Letra>`).

iii) `[L|P]`: Modalidad de juego: L para longitud, P para puntuación.

iv) `n`: Cantidad de letras que tendrá la bolsa para el juego.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BolsaLetras	
TDA LettersBag	7
ConjuntoLetras	
TDA ConjuntoLetras	10
ConjuntoLetras::const_iterator	13
Diccionario	
TDA Dictionary	13
ConjuntoLetras::iterator	17
Diccionario::iterator	
Clase iteradora para el TDA Diccionario	17
Letra	
TDA Letra	20

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

include/dictionary.h	25
include/letters_bag.h	26
include/letters_set.h	26

Chapter 4

Class Documentation

4.1 BolsaLetras Class Reference

TDA LettersBag.

```
#include <letters_bag.h>
```

Public Member Functions

- [BolsaLetras](#) ()
Constructor por defecto.
- int [size](#) () const
Calcula la cantidad total de elementos que hay en la bolsa.
- void [add](#) ([Letra](#) letra)
Método para añadir un elemento a la bolsa.
- char [get](#) ()
Método para extraer un elemento aleatorio de la bolsa.
- string [toString](#) () const
Método para formatear como cadena la bolsa y obteniendo dicha cadena.
- void [cargarBolsa](#) (const [ConjuntoLetras](#) &conj)
Método para cargar una bolsa dado un conjunto de letras.

Friends

- ostream & [operator<<](#) (ostream &os, const [BolsaLetras](#) &b)
Sobrecarga del operador <<.
- istream & [operator>>](#) (istream &is, [BolsaLetras](#) &b)
Permite la lectura de un objeto tipo [BolsaLetras](#) desde un flujo de entrada. Se usará para leer bolsas que vienen formateadas según la salida del programa 'Conjunto_letras'. Es decir: "[carácter] [cantidad] [puntuación]" por cada línea/letra.

4.1.1 Detailed Description

TDA LettersBag.

Este TDA almacena un conjunto de char utilizado en el juego de letras. La estructura de datos subyacente es una string.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BolsaLetras()

```
BolsaLetras::BolsaLetras ( )
```

Constructor por defecto.

Genera una bolsa vacía.

4.1.3 Member Function Documentation

4.1.3.1 add()

```
void BolsaLetras::add (
    Letra letra )
```

Método para añadir un elemento a la bolsa.

Parameters

<i>letra</i>	Letra que se va a añadir a la bolsa
--------------	-------------------------------------

4.1.3.2 cargarBolsa()

```
void BolsaLetras::cargarBolsa (
    const ConjuntoLetras & conj )
```

Método para cargar una bolsa dado un conjunto de letras.

Genera la bolsa a partir de un conjunto de letras (TDA [ConjuntoLetras](#)).

Parameters

<i>conj</i>	: conjunto de letras a partir del cual se crea la bolsa
-------------	---

4.1.3.3 get()

```
char BolsaLetras::get ( )
```

Método para extraer un elemento aleatorio de la bolsa.

Al obtener el elemento, lo elimina de la bolsa.

Returns

Elemento aleatorio

4.1.3.4 size()

```
int BolsaLetras::size ( ) const
```

Calcula la cantidad total de elementos que hay en la bolsa.

Returns

Tamaño de la bolsa.

Note

Método de consulta.

4.1.3.5 toString()

```
string BolsaLetras::toString ( ) const
```

Método para formatear como cadena la bolsa y obteniendo dicha cadena.

Returns

Una cadena tipo std::string la bolsa formateada, de forma que por cada elemento, se imprime una línea.

Note

Método de consulta.

4.1.4 Friends And Related Symbol Documentation

4.1.4.1 operator<<

```
ostream & operator<< (
    ostream & os,
    const BolsaLetras & b ) [friend]
```

Sobrecarga del operador <<.

Posibilita la salida formateada de la bolsa por un flujo, que por defecto es la salida estándar.

Parameters

<i>os</i>	Flujo de salida. Por defecto, os = cout.
<i>b</i>	Bolsa.

Returns

Una referencia al flujo de salida.

4.1.4.2 operator>>

```
istream & operator>> (
    istream & is,
    BolsaLetras & b ) [friend]
```

Permite la lectura de un objeto tipo [BolsaLetras](#) desde un flujo de entrada. Se usará para leer bolsas que vienen formateadas según la salida del programa 'Conjunto_letras'. Es decir: "[carácter] [cantidad] [puntuación]" por cada línea/letra.

Sobrecarga del operador >>.

Parameters

<i>is</i>	Flujo de entrada del que se va a leer un objeto BolsaLetras
<i>b</i>	Bolsa en la que se van a insertar los elementos leídos.

Returns

Una referencia al flujo de entrada.

The documentation for this class was generated from the following files:

- include/letters_bag.h
- src/letters_bag.cpp

4.2 ConjuntoLetras Class Reference

TDA [ConjuntoLetras](#).

```
#include <letters_set.h>
```

Classes

- class [const_iterator](#)
- class [iterator](#)

Public Member Functions

- [Letra getLetra](#) (char character) const
- void [insert](#) (const [Letra](#) &letra)
Insertar una letra.
- [iterator begin](#) ()
Inicio del recorrido.
- [iterator end](#) ()
Final del recorrido.
- [const_iterator begin](#) () const
Inicio del recorrido.
- [const_iterator end](#) () const
Final del recorrido.

Friends

- ostream & [operator<<](#) (ostream &salida, const [ConjuntoLetras](#) &conj)
Escribe en un flujo de salida un conjunto de letras.
- istream & [operator>>](#) (istream &entrada, [ConjuntoLetras](#) &conj)
Lee de un flujo de entrada un conjunto de letras.

4.2.1 Detailed Description

TDA [ConjuntoLetras](#).

Este TDA representa un conjunto de letras, con la información necesaria para jugar una partida al juego de las letras, es decir, el número de repeticiones que tenemos de la letra y la puntuación que dicha letra otorga cuando se utiliza en una palabra

4.2.2 Member Function Documentation

4.2.2.1 begin() [1/2]

```
ConjuntoLetras::iterator ConjuntoLetras::begin ( )
```

Inicio del recorrido.

Returns

Posicion de inicio del recorrido

4.2.2.2 begin() [2/2]

```
ConjuntoLetras::const\_iterator ConjuntoLetras::begin ( ) const
```

Inicio del recorrido.

Returns

Posicion de inicio del recorrido

4.2.2.3 end() [1/2]

```
ConjuntoLetras::iterator ConjuntoLetras::end ( )
```

Final del recorrido.

Returns

Posicion de final del recorrido

4.2.2.4 end() [2/2]

```
ConjuntoLetras::const_iterator ConjuntoLetras::end ( ) const
```

Final del recorrido.

Returns

Posicion de final del recorrido

4.2.2.5 insert()

```
void ConjuntoLetras::insert (
    const Letra & letra )
```

Insertar una letra.

Parameters

<i>letra</i>	letra a insertar @doc letra se inserta de forma ordenada
--------------	--

4.2.3 Friends And Related Symbol Documentation

4.2.3.1 operator<<

```
ostream & operator<< (
    ostream & salida,
    const ConjuntoLetras & conj ) [friend]
```

Escribe en un flujo de salida un conjunto de letras.

Parameters

<i>salida</i>	flujo de salida
<i>letra</i>	el objeto que se escribe.

Returns

el flujo de salida

4.2.3.2 operator>>

```
istream & operator>> (
    istream & entrada,
    ConjuntoLetras & conj ) [friend]
```

Lee de un flujo de entrada un conjunto de letras.

Parameters

<i>entrada</i>	flujo de entrada
<i>letra</i>	el objeto donde se realiza la lectura.

Returns

el flujo de entrada

The documentation for this class was generated from the following files:

- include/letters_set.h
- src/letters_set.cpp

4.3 ConjuntoLetras::const_iterator Class Reference

```
#include <letters_set.h>
```

Public Member Functions

- **const_iterator** (set< [Letra](#) >::const_iterator iter)
- bool **operator!=** (const [ConjuntoLetras::const_iterator](#) &iter)
- bool **operator==** (const [ConjuntoLetras::const_iterator](#) &iter)
- const [Letra](#) & **operator*** () const
- [const_iterator](#) & **operator++** ()
- [const_iterator](#) & **operator=** (const [ConjuntoLetras::const_iterator](#) &iter)

4.3.1 Detailed Description

[ConjuntoLetras::const_iterator](#) permite recorrer los elementos del conjunto de letras en orden ascendente

The documentation for this class was generated from the following files:

- include/letters_set.h
- src/letters_set.cpp

4.4 Diccionario Class Reference

TDA Dictionary.

```
#include <dictionary.h>
```

Classes

- class [iterator](#)
Clase iteradora para el TDA [Diccionario](#).

Public Member Functions

- **Diccionario ()**
Construye un diccionario vacío.
- **int size () const**
Devuelve el número de palabras en el diccionario.
- **vector< string > PalabrasLongitud (int longitud)**
Obtiene todas las palabras en el diccionario de un longitud dada.
- **bool Esta (const string &palabra) const**
Indica si una palabra está en el diccionario o no.
- **const iterator begin () const**
Calcula la primera posición (dirección de la palabra correspondiente) del diccionario.
- **const iterator end () const**
Calcula la última posición (dirección de la palabra correspondiente) del diccionario.

Friends

- **istream & operator>> (istream &is, Diccionario &D)**
Lee de un flujo de entrada un diccionario.
- **ostream & operator<< (ostream &os, const Diccionario &D)**
Escribe en un flujo de salida un diccionario.

4.4.1 Detailed Description

TDA Dictionary.

Almacena las palabras de un fichero de texto y permite iterar sobre ellas

4.4.2 Member Function Documentation

4.4.2.1 begin()

```
const Diccionario::iterator Diccionario::begin ( ) const
```

Calcula la primera posición (dirección de la palabra correspondiente) del diccionario.

Returns

Un puntero a la primera palabra del diccionario.

4.4.2.2 end()

```
const Diccionario::iterator Diccionario::end ( ) const
```

Calcula la última posición (dirección de la palabra correspondiente) del diccionario.

Returns

Un puntero a la última palabra del diccionario.

4.4.2.3 Esta()

```
bool Diccionario::Esta (  
    const string & palabra ) const
```

Indica si una palabra está en el diccionario o no.

Parameters

<i>palabra</i>	la palabra que se quiere buscar
----------------	---------------------------------

Returns

true si la palabra está en el diccionario. False en caso contrario

4.4.2.4 PalabrasLongitud()

```
vector< string > Diccionario::PalabrasLongitud (
    int longitud )
```

Obtiene todas las palabras en el diccionario de un longitud dada.

Parameters

<i>longitud</i>	la longitud de las palabras de salida
-----------------	---------------------------------------

Returns

un vector con las palabras de longitud especifica en el parametro de entrada

4.4.3 Friends And Related Symbol Documentation**4.4.3.1 operator<<**

```
ostream & operator<< (
    ostream & os,
    const Diccionario & D ) [friend]
```

Escribe en un flujo de salida un diccionario.

Parameters

<i>os</i>	flujo de salida.
<i>D</i>	el objeto diccionario que se escribe.

Returns

el flujo de salida.

4.4.3.2 operator>>

```
istream & operator>> (
    istream & is,
    Diccionario & D ) [friend]
```

Lee de un flujo de entrada un diccionario.

Parameters

<i>is</i>	flujo de entrada.
<i>D</i>	el objeto donde se realiza la lectura.

Returns

el flujo de entrada.

The documentation for this class was generated from the following files:

- include/dictionary.h
- src/dictionary.cpp

4.5 ConjuntoLetras::iterator Class Reference

```
#include <letters_set.h>
```

Public Member Functions

- **iterator** (set< [Letra](#) >::iterator iter)
- bool **operator!=** (const [ConjuntoLetras::iterator](#) &iter)
- bool **operator==** (const [ConjuntoLetras::iterator](#) &iter)
- const [Letra](#) & **operator*** () const
- [iterator](#) & **operator++** ()
- [iterator](#) & **operator=** (const [ConjuntoLetras::iterator](#) &iter)

4.5.1 Detailed Description

[ConjuntoLetras::iterator](#) permite recorrer los elementos del conjunto de letras en orden ascendente

The documentation for this class was generated from the following files:

- include/letters_set.h
- src/letters_set.cpp

4.6 Diccionario::iterator Class Reference

Clase iteradora para el TDA [Diccionario](#).

```
#include <dictionary.h>
```

Public Member Functions

- **iterator** ()
Constructor por defecto.
- const string & **operator*** () const
*Sobrecarga del operador *.*
- **iterator** & **operator++** ()
Sobrecarga del operador ++.
- bool **operator==** (const **iterator** &i)
Sobrecarga del operador ==.
- bool **operator!=** (const **iterator** &i)
Sobrecarga del operador !=.

Friends

- class **Diccionario**
*friend class **Diccionario***

4.6.1 Detailed Description

Clase iteradora para el TDA **Diccionario**.

Se basa en un `set<string>::iterator`.

4.6.2 Member Function Documentation

4.6.2.1 **operator"!=()**

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

Sobrecarga del operador !=.

Calcula si las palabras a las que apuntan el iterador implícito y el iterador del parámetro

i son diferentes.

Parameters

<i>i</i>	
----------	--

Returns

true si las palabras son diferentes; false si son iguales.

4.6.2.2 operator*()

```
const string & Diccionario::iterator::operator* ( ) const
```

Sobrecarga del operador *.

Returns

Devuelve el elemento del diccionario al que apunta el iterador.

Note

Método de consulta.

4.6.2.3 operator++()

```
Diccionario::iterator & Diccionario::iterator::operator++ ( )
```

Sobrecarga del operador ++.

Calcula la posición siguiente del diccionario. Como es de imaginar, es la dirección de la siguiente palabra del diccionario.

Returns

Puntero a la siguiente posición según la indicación anterior.

4.6.2.4 operator==()

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

Sobrecarga del operador ==.

*

Calcula si la palabra del diccionario a la que apunta el iterador implícito es igual a la que apunta el parámetro

i.

Parameters

<i>i</i>	
----------	--

Returns

true si ambas son iguales; false en caso contrario.

4.6.3 Friends And Related Symbol Documentation

4.6.3.1 Diccionario

```
friend class Diccionario [friend]
```

friend class [Diccionario](#)

La clase [Diccionario](#) se declara como amiga de la iteradora para poder acceder a los miembros privados del diccionario, que en este caso es todo el set de palabras.

The documentation for this class was generated from the following files:

- include/dictionary.h
- src/dictionary.cpp

4.7 Letra Class Reference

TDA [Letra](#).

```
#include <letters_set.h>
```

Public Member Functions

- [Letra](#) ()
Constructor sin parámetros @doc Establece la cantidad a 0 y la puntuación a 0.
- [Letra](#) (char car)
Constructor.
- [Letra](#) (char car, int cant, int punt)
Constructor con parámetros.
- void [setCaracter](#) (char c)
Establece el caracter de la letra.
- void [setCantidad](#) (int c)
Establece la cantidad de la letra.
- void [setPuntuacion](#) (int p)
Establece la puntuación de la letra.
- void [setPuntuacion](#) (int apariciones, int total, int num_letras)
Establece la puntuación de la letra.
- char [getCaracter](#) () const
Devuelve el caracter asociado a una letra.
- int [getCantidad](#) () const
Devuelve la cantidad asociada a una letra.
- int [getPuntuacion](#) () const
Devuelve la puntuación asociada a una letra.
- bool [operator<](#) (const [Letra](#) &l) const
Sobrecarga del operador < para [Letra](#).

Friends

- ostream & [operator<<](#) (ostream &salida, const [Letra](#) &letra)
Escribe en un flujo de salida una letra.
- istream & [operator>>](#) (istream &entrada, [Letra](#) &letra)
Lee de un flujo de entrada una letra.

4.7.1 Detailed Description

TDA [Letra](#).

Contiene información sobre un determinado carácter del juego de las letras. En concreto, almacena información sobre el número de repeticiones de la letra en la partida y de la puntuación que otorga al utilizarse en una palabra

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Letra() [1/2]

```
Letra::Letra (
    char car ) [inline]
```

Constructor.

Parameters

<i>car</i>	caracter que representa la letra @doc Establece la cantidad y la puntuación a 0
------------	---

4.7.2.2 Letra() [2/2]

```
Letra::Letra (
    char car,
    int cant,
    int punt ) [inline]
```

Constructor con parámetros.

Parameters

<i>car</i>	caracter que representa la letra
<i>cant</i>	cantidad de veces que aparece la letra
<i>punt</i>	puntuación asociada a la letra

4.7.3 Member Function Documentation

4.7.3.1 getCantidad()

```
int Letra::getCantidad ( ) const [inline]
```

Devuelve la cantidad asociada a una letra.

Returns

Cantidad asociada a una letra

4.7.3.2 getCaracter()

```
char Letra::getCaracter ( ) const [inline]
```

Devuelve el caracter asociado a una letra.

Returns

Caracter asociado a una letra

4.7.3.3 getPuntuacion()

```
int Letra::getPuntuacion ( ) const [inline]
```

Devuelve la puntuación asociada a una letra.

Returns

Puntuación asociada a una letra

4.7.3.4 operator<()

```
bool Letra::operator< (
    const Letra & l ) const
```

Sobrecarga del operador < para [Letra](#).

Parameters

/	letra con la que se compara el objeto implícito
---	---

Returns

true si el objeto implícito es menor que l false en otro caso @doc Una letra L1 es menor que L2 si : L1.caracter < L2.caracter

4.7.3.5 setCantidad()

```
void Letra::setCantidad (
    int c ) [inline]
```

Establece la cantidad de la letra.

Parameters

<i>c</i>	cantidad a establecer
----------	-----------------------

4.7.3.6 setCaracter()

```
void Letra::setCaracter (
    char c ) [inline]
```

Establece el caracter de la letra.

Parameters

<i>c</i>	caracter a establecer
----------	-----------------------

4.7.3.7 setPuntuacion() [1/2]

```
void Letra::setPuntuacion (
    int apariciones,
    int total,
    int num_letras )
```

Establece la puntuación de la letra.

Parameters

<i>apariciones</i>	num veces que aparece una letra en un diccionario
<i>total</i>	total de letras de un diccionario
<i>num_letras</i>	num letras diferentes de un diccionario @doc La puntuación estará entre 1 y 10

4.7.3.8 setPuntuacion() [2/2]

```
void Letra::setPuntuacion (
    int p ) [inline]
```

Establece la puntuación de la letra.

Parameters

<i>p</i>	puntuación a establecer
----------	-------------------------

4.7.4 Friends And Related Symbol Documentation**4.7.4.1 operator<<**

```
ostream & operator<< (
```

```
ostream & salida,  
const Letra & letra ) [friend]
```

Escribe en un flujo de salida una letra.

Parameters

<i>salida</i>	flujo de salida
<i>letra</i>	el objeto que se escribe.

Returns

el flujo de salida

4.7.4.2 operator>>

```
istream & operator>> (  
    istream & entrada,  
    Letra & letra ) [friend]
```

Lee de un flujo de entrada una letra.

Parameters

<i>entrada</i>	flujo de entrada
<i>letra</i>	el objeto donde se realiza la lectura.

Returns

el flujo de entrada

The documentation for this class was generated from the following files:

- include/letters_set.h
- src/letters_set.cpp

Chapter 5

File Documentation

5.1 dictionary.h

```
00001 #ifndef __DICTIONARY_H__
00002 #define __DICTIONARY_H__
00003
00004 #include <string>
00005 #include <vector>
00006 #include <set>
00007
00008 using namespace std;
00009
00015 class Diccionario{
00016 private:
00021     set<string> datos;
00022
00023 public:
00027     Diccionario();
00028
00032     int size() const ;
00033
00039     vector<string> PalabrasLongitud(int longitud);
00040
00046     bool Esta(const string & palabra) const;
00047
00054     friend istream & operator>(istream & is, Diccionario &D);
00055
00062     friend ostream & operator<<(ostream & os, const Diccionario &D);
00063
00064
00069     class iterator{
00070
00071     private:
00076         set<string>::iterator it;
00077
00078     public:
00082         iterator ();
00083
00089         const string & operator*() const;
00090
00097         iterator & operator ++();
00098
00106         bool operator ==(const iterator &i);
00107
00115         bool operator !=(const iterator &i);
00116
00123         friend class Diccionario;
00124     };
00125
00131     const iterator begin() const;
00132
00138     const iterator end() const;
00139 };
00140
00149 string mayusculas(string &palabra);
00150 #endif
```

5.2 letters_bag.h

```

00001 #ifndef __LETTERS_BAG_H__
00002 #define __LETTERS_BAG_H__
00003
00004 #include <string>
00005 #include "letters_set.h"
00006
00007 using namespace std;
00008
00015 class BolsaLetras {
00016 private:
00021     string bolsa_letras;
00022
00023 public:
00028     BolsaLetras();
00029
00035     int size() const;
00036
00041     void add(Letra letra);
00042
00048     char get();
00049
00056     string toString() const;
00057
00063     void cargarBolsa (const ConjuntoLetras &conj) ;
00064
00073     friend ostream& operator<<(ostream& os, const BolsaLetras& b);
00074
00085     friend istream& operator>>(istream& is, BolsaLetras& b);
00086 };
00087
00088 #endif

```

5.3 letters_set.h

```

00001 #ifndef __LETTER_SET_H__
00002 #define __LETTER_SET_H__
00003
00004 #include <set>
00005 #include <sstream>
00006 using namespace std;
00007
00016 class Letra {
00017 private:
00018     char caracter;
00019     int cantidad;
00020     int puntuacion;
00022 public:
00027     Letra () : cantidad(0), puntuacion(0) {};
00028
00034     Letra (char car) : caracter(toupper(car)), cantidad(0), puntuacion(0) {};
00035
00042     Letra (char car, int cant, int punt) : caracter(toupper(car)) , cantidad(cant) , puntuacion(punt)
00043     {};
00048
00049     void setCaracter(char c) { caracter = c; };
00054
00055     void setCantidad(int c) { cantidad = c; };
00060
00061     void setPuntuacion (int p) { puntuacion = p; };
00070
00071     void setPuntuacion (int apariciones , int total, int num_letras);
00076
00077     char getCaracter() const { return caracter; };
00082
00083     int getCantidad() const { return cantidad; };
00088
00089     int getPuntuacion() const { return puntuacion; };
00098
00099     bool operator<(const Letra& l) const;
00100
00107     friend ostream& operator<<(ostream& salida, const Letra& letra);
00108
00115     friend istream& operator>>(istream& entrada, Letra& letra);
00116 };
00117
00118
00119
00129 class ConjuntoLetras {
00130 private:
00131     set<Letra> letras;

```



```
00132 public:
00133
00134     Letra getLetra (char caracter) const;
00135
00136     // Letra& operator[](char caracter);
00137
00144     friend ostream& operator<<(ostream& salida, const ConjuntoLetras& conj);
00145
00152     friend istream& operator>>(istream& entrada, ConjuntoLetras& conj);
00153
00159     void insert (const Letra &letra);
00160
00161
00166     class iterator {
00167     private:
00168         set<Letra>::iterator el_iterador;
00169     public:
00170         iterator() {}
00171         iterator(set<Letra>::iterator iter) : el_iterador(iter) {};
00172         bool operator!=(const ConjuntoLetras::iterator &iter);
00173         bool operator==(const ConjuntoLetras::iterator &iter);
00174         const Letra& operator*() const;
00175         iterator& operator++();
00176         iterator& operator=(const ConjuntoLetras::iterator &iter);
00177     };
00178
00183     class const_iterator {
00184     private:
00185         set<Letra>::const_iterator el_iterador;
00186     public:
00187         const_iterator() {}
00188         const_iterator(set<Letra>::const_iterator iter) : el_iterador(iter) {};
00189         bool operator!=(const ConjuntoLetras::const_iterator &iter);
00190         bool operator==(const ConjuntoLetras::const_iterator &iter);
00191         const Letra& operator*() const;
00192         const_iterator& operator++();
00193         const_iterator& operator=(const ConjuntoLetras::const_iterator &iter);
00194     };
00195
00196
00201     iterator begin();
00202
00207     iterator end();
00208
00213     const_iterator begin() const;
00214
00219     const_iterator end() const;
00220
00221 };
00222
00223
00224 #endif
```


Index

add
 BolsaLetras, 8

begin
 ConjuntoLetras, 11
 Diccionario, 14

BolsaLetras, 7
 add, 8
 BolsaLetras, 8
 cargarBolsa, 8
 get, 8
 operator<<, 9
 operator>>, 9
 size, 8
 toString, 9

cargarBolsa
 BolsaLetras, 8

ConjuntoLetras, 10
 begin, 11
 end, 11
 insert, 12
 operator<<, 12
 operator>>, 12

ConjuntoLetras::const_iterator, 13

ConjuntoLetras::iterator, 17

Diccionario, 13
 begin, 14
 Diccionario::iterator, 20
 end, 14
 Esta, 14
 operator<<, 15
 operator>>, 15
 PalabrasLongitud, 15

Diccionario::iterator, 17
 Diccionario, 20
 operator!=, 18
 operator++, 19
 operator==, 19
 operator*, 18

end
 ConjuntoLetras, 11
 Diccionario, 14

Esta
 Diccionario, 14

get
 BolsaLetras, 8

getCantidad
 Letra, 21

getCaracter
 Letra, 22

getPuntuacion
 Letra, 22

include/dictionary.h, 25

include/letters_bag.h, 26

include/letters_set.h, 26

insert
 ConjuntoLetras, 12

Letra, 20
 getCantidad, 21
 getCaracter, 22
 getPuntuacion, 22
 Letra, 21
 operator<, 22
 operator<<, 23
 operator>>, 24
 setCantidad, 22
 setCaracter, 23
 setPuntuacion, 23

operator!=
 Diccionario::iterator, 18

operator<
 Letra, 22

operator<<
 BolsaLetras, 9
 ConjuntoLetras, 12
 Diccionario, 15
 Letra, 23

operator>>
 BolsaLetras, 9
 ConjuntoLetras, 12
 Diccionario, 15
 Letra, 24

operator++
 Diccionario::iterator, 19

operator==
 Diccionario::iterator, 19

operator*
 Diccionario::iterator, 18

PalabrasLongitud
 Diccionario, 15

Prefacio, 1

setCantidad
 Letra, 22

- setCaracter
 - Letra, [23](#)
- setPuntuacion
 - Letra, [23](#)
- size
 - BolsaLetras, [8](#)
- toString
 - BolsaLetras, [9](#)