Práctica final - Cifras y Letras

Elena Arroyo Zamora, Carlos Castillo Sánchez

Doble Grado en Ingeniería Informática y Matemáticas

Universidad de Granada

December 14, 2025

Chapter 1

Mainpage

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:

```
./bin/cantidad_letras ./data/diccionario.txt ./data/letras.txt
```

i) `./data/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) `./data/letras.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.

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

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

ii) `./data/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:

Chapter 3

File Index

3.1 File List

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

Chapter 4

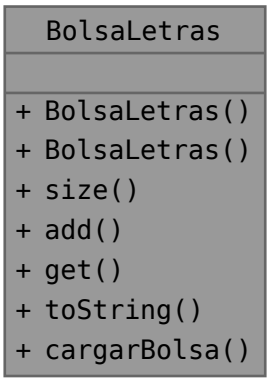
Class Documentation

4.1 BolsaLetras Class Reference

TDA LettersBag.

`#include <letters_bag.h>`

Collaboration diagram for BolsaLetras:



Public Member Functions

- BolsaLetras ()

  Constructor por defecto.
- BolsaLetras (const ConjuntoLetras &conjunto)

  Constructor con un parámetro.
- 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() [1/2]

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

Constructor por defecto.

Genera una bolsa vacía.

4.1.2.2 BolsaLetras() [2/2]

```
BolsaLetras::BolsaLetras (
            const ConjuntoLetras & conjunto )
```

Constructor con un parámetro.

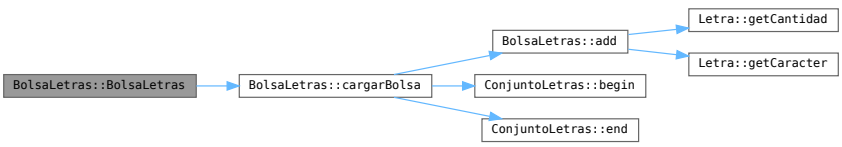
Genera una bolsa de letras de un conjunto de letras (TDA ConjuntoLetras) ya existente. Añade cada caracter tantas veces como indica el campo Cantidad en cada letra (TDA Letra) del conjunto.

```
ConjuntoLetras conjunto;
ifstream entrada("letras.txt") // <-- Fichero de salida del programa cantidad_letras

if (!entrada) {
   cout « "Error al abrir el fichero de entrada" « endl;
   return 1;
}

BoslaLetras(conjunto);
```

Here is the call graph for this function:



4.1.3 Member Function Documentation

4.1.3.1 add()

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

Método para añadir un elemento a la bolsa.

Este es el método que se encarga de introducir en la bolsa los caracteres tantas veces como indique el campo `cantidad` de la `letra` (TDA Letra).

Parameters

letra	Letra que se va a añadir a la bolsa

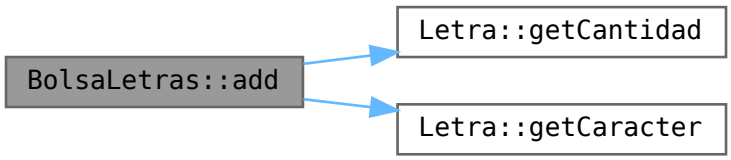
```
ifstream entrada("letras.txt") // <-- Fichero de salida del programa cantidad_letras

if (!entrada) {
   cout « "Error al abrir el fichero de entrada" « endl;
   return 1;
}

ConjuntoLetras conjunot ("letras.txt");
Letra letra('a', 3, 10);
BolsaLetras bolsa(conjunto);

bolsa.add(letra);
cout « bolsa « endl; // <-- "aaa"
```

Here is the call graph for this function:



4.1.3.2 cargarBolsa()

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

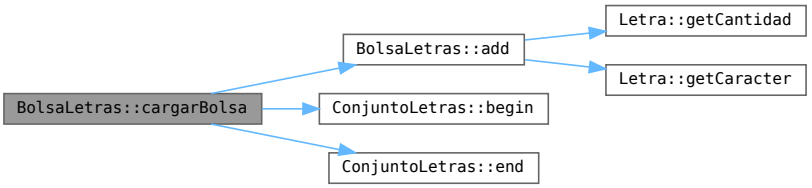
Método para cargar una bolsa dado un conjunto de letras.

Genera la bolsa a partir de un conjunto de letras (TDA ConjuntoLetras), de forma que en la bolsa se incluyen los caracteres correspondientes a las letras tantas veces como indica su campo `cantidad`.

Parameters

conj	Conjunto de letras a partir del cual se crea la bolsa

Here is the call graph for this function:



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.

Here is the call graph for this function:

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

os	Flujo de salida. Por defecto, os = cout.
b	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

is	Flujo de entrada del que se va a leer un objeto BolsaLetras
b	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:

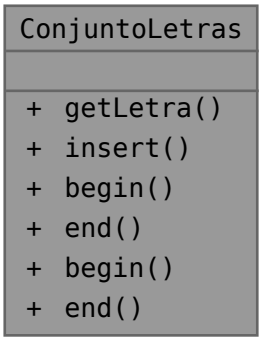
- include/letters_bag.h
- src/letters_bag.cpp

4.2 ConjuntoLetras Class Reference

TDA ConjuntoLetras.

`#include <letters_set.h>`

Collaboration diagram for ConjuntoLetras:



Classes

- class const_iterator

  ConjuntoLetras::const_iterator permite recorrer los elementos del conjunto de letras en orden ascendente.
- class iterator

  ConjuntoLetras::iterator permite recorrer los elementos del conjunto de letras en orden ascendente.

Public Member Functions

- Letra getLetra (char caracter) const
  Método de búsqueda de una Letra a partir del caracter que la identifica.
- 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 getLetra()

```
Letra ConjuntoLetras::getLetra (
            char caracter ) const
```

Método de búsqueda de una Letra a partir del caracter que la identifica.

Parameters

caracter	Carácter que se quiere buscar en el ConjuntoLetras.

Returns

La Letra (TDA Letra) correspondiente.

Note

En caso de no existir una extrada con dicho carácter en el Conjunto de Letras, se devuelve una letra creada por el constructor por defecto.

4.2.2.6 insert()

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

Insertar una letra.

Parameters

letra	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

salida	flujo de salida
letra	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

entrada	flujo de entrada
letra	el objeto donde se realiza la lectura.

Returns

el flujo de entrada

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

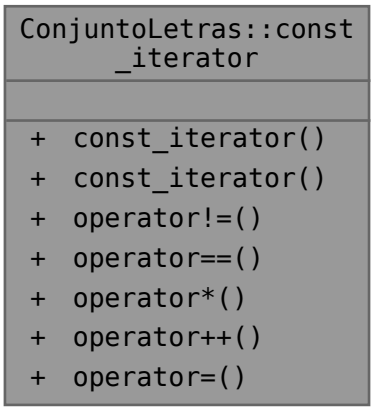
- include/letters_set.h
- src/letters_set.cpp

4.3 ConjuntoLetras::const_iterator Class Reference

ConjuntoLetras::const_iterator permite recorrer los elementos del conjunto de letras en orden ascendente.

`#include <letters_set.h>`

Collaboration diagram for ConjuntoLetras::const_iterator:



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:

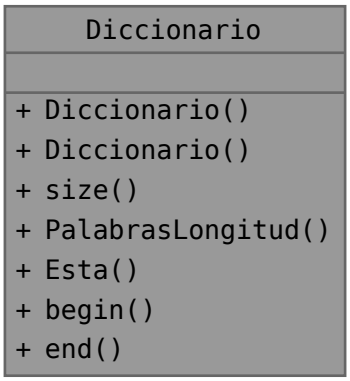
- include/letters_set.h
- src/letters_set.cpp

4.4 Diccionario Class Reference

TDA Diccionario.

`#include <dictionary.h>`

Collaboration diagram for Diccionario:



Classes

- class iterator

  Clase iteradora para el TDA Diccionario.

Public Member Functions

- Diccionario ()

  Constructor sin parámetros.
- Diccionario (string fichero_diccionario)

  Constructor con un parámetro.
- 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 el diccionario correspondiente al objeto implícito.

4.4.1 Detailed Description

TDA Diccionario.

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

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Diccionario() [1/2]

```
Diccionario::Diccionario ( )
```

Constructor sin parámetros.

Construye un diccionario vacío.

4.4.2.2 Diccionario() [2/2]

```
Diccionario::Diccionario (
            string fichero_diccionario )
```

Constructor con un parámetro.

Parameters

fichero_diccionario	Nombre del fichero que se va a tomar como diccionario.

Precondition

El fichero debe tener el formato adecuado, es decir, una palabra por línea.

Note

Este constructor se basa en el operador >>.

4.4.3 Member Function Documentation

4.4.3.1 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.

4.4.3.2 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.

4.4.3.3 Esta()

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

Indica si una palabra está en el diccionario o no.

Parameters

palabra	la palabra que se quiere buscar

Returns

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

```
Diccionario d("fichero_diccionario.txt");
string palabra ("Hola");

if (d.Esta(palabra)
  cout « "La palabra " « palabra « " pertenece al diccionario." « endl;

else
  cout « "La palabra " « palabra « " no pertenece al diccionario." « endl;
```

4.4.3.4 PalabrasLongitud()

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

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

Parameters

Longitud	la longitud de las palabras de salida.

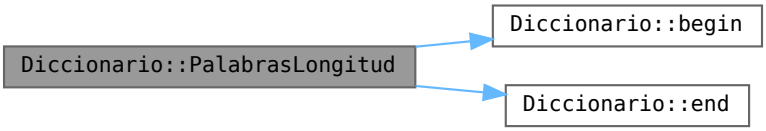
Returns

Un vector con las palabras de longitud especifica en el parametro de entrada.

```
Diccionario d("fichero_diccionario.txt");
vector<string> v;

v = d.PalabrasLongitud(5);
cout « v « endl; // Muestra las palabras de longitud = 5 pertenecientes al diccionario
                 // cargado desde el fichero "fichero_diccionario.txt"
```

Here is the call graph for this function:



4.4.4 Friends And Related Symbol Documentation

4.4.4.1 operator<<

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

Escribe en un flujo de salida el diccionario correspondiente al objeto implícito.

Parameters

os	flujo de salida.
D	el objeto diccionario que se escribe.

Returns

el flujo de salida.

4.4.4.2 operator>>

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

Lee de un flujo de entrada un diccionario.

Parameters

is	flujo de entrada.
D	el objeto donde se realiza la lectura.

Returns

el flujo de entrada.

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

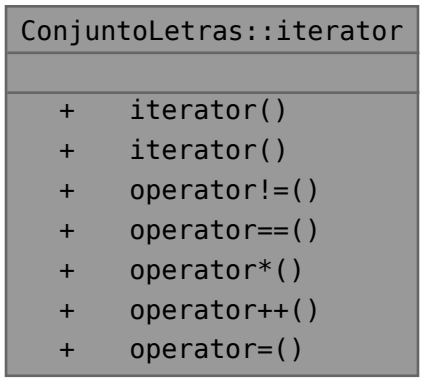
- include/dictionary.h
- src/dictionary.cpp

4.5 ConjuntoLetras::iterator Class Reference

ConjuntoLetras::iterator permite recorrer los elementos del conjunto de letras en orden ascendente.

`#include <letters_set.h>`

Collaboration diagram for ConjuntoLetras::iterator:



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:

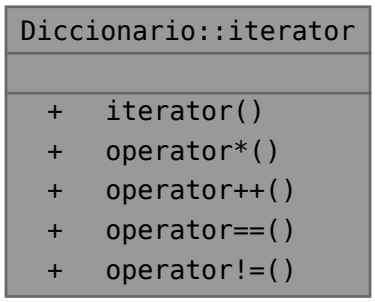
- include/letters_set.h
- src/letters_set.cpp

4.6 Diccionario::iterator Class Reference

Clase iteradora para el TDA Diccionario.

`#include <dictionary.h>`

Collaboration diagram for Diccionario::iterator:



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. Permitirá recorrer diccionarios de la forma que si indica a continuación:

```
Diccionario d;
ifstream entrada("fichero_diccionario.txt");

if (!entrada) {
   cout « "Error al abrir el fichero";
   return 1;
}

for (Diccionario::iterator it = d.begin(); it != d.end(); it++) {
   // código de iteración
}
```

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	

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	Dirección del elemento con que el iterador implícito se compara.

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:

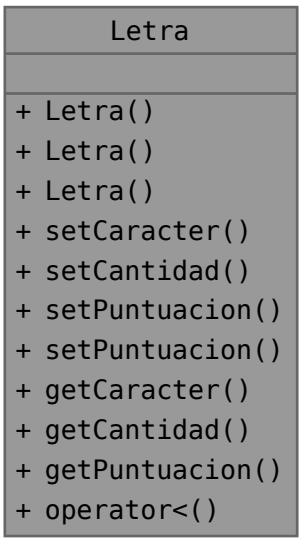
- include/dictionary.h
- src/dictionary.cpp

4.7 Letra Class Reference

TDA Letra.

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

Collaboration diagram for Letra:



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, pair< int, int > max_min)

  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

car	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

car	caracter que representa la letra
cant	cantidad de veces que aparece la letra
punt	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

l	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

c	cantidad a establecer

4.7.3.6 setCaracter()

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

Establece el caracter de la letra.

Parameters

c	caracter a establecer

4.7.3.7 setPuntuacion() [1/2]

```
void Letra::setPuntuacion (
            int apariciones,
            pair< int, int > max_min )
```

Establece la puntuación de la letra.

Parameters

apariciones	num veces que aparece una letra en un diccionario
total	total de letras de un diccionario
num_letras	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

p	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

salida	flujo de salida
letra	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

entrada	flujo de entrada
letra	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
00010
00011 // ------------------------------ Clase Diccionario -------------------------
00017 class Diccionario{
00018 private:
00023     set<string> datos;
00024
00025 public:
00030     Diccionario();
00031
00038     Diccionario(string fichero_diccionario);
00039
00043     int size() const ;
00044
00059     vector<string> PalabrasLongitud(int longitud);
00060
00077     bool Esta(const string & palabra) const;
00078
00085     friend istream & operator»(istream & is,Diccionario &D);
00086
00094     friend ostream & operator«(ostream & os, const Diccionario &D);
00095
00096     //------------------------- Clase Iterator --------------------------------
00116     class iterator{
00117     private:
00122         set<string>::iterator it;
00123
00124     public:
00128         iterator ();
00129
00135         const string & operator*() const;
00136
00143         iterator & operator ++();
00144
00152         bool operator ==(const iterator &i);
00153
00161         bool operator !=(const iterator &i);
00162
00169         friend class Diccionario;
00170     };
00171
00177     const iterator begin() const;
00178
00184     const iterator end() const;
00185 };
00186
00187
00188 // ---------------------------- Funciones externas --------------------------
00208 string mayusculas(const string &palabra);
00209 #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
00009
00010 // ------------------------- Clase BolsaLetras ------------------------------
00016 class BolsaLetras {
00017 private:
00022     string bolsa_letras;
00023
00024 public:
00029     BolsaLetras();
00030
00031
00053     BolsaLetras(const ConjuntoLetras & conjunto);
00054
00060     int size() const;
00061
00087     void add(Letra letra);
00088
00094     char get();
00095
00102     string toString() const;
00103
00111     void cargarBolsa (const ConjuntoLetras &conj) ;
00112
00121     friend ostream& operator<<(ostream& os, const BolsaLetras& b);
00122
00133     friend istream& operator>>(istream& is, BolsaLetras& b);
00134 };
00135
00136 #endif
```

5.3 letters_set.h

```
00001 #ifndef __LETTER_SET_H__
00002 #define __LETTER_SET_H__
00003
00004 #include <set>
00005 #include <sstream>
00006 #include <vector>
00007
00008 using namespace std;
00009
00010
00011 // ------------------------- Clase Letra ------------------------------------
00020 class Letra {
00021 private:
00022     char caracter;
00023     int cantidad;
00024     int puntuacion;
00026 public:
00031     Letra () : cantidad(0), puntuacion(0) {};
00032
00038     Letra (char car) : caracter(toupper(car)), cantidad(0), puntuacion(0) {};
00039
00046     Letra (char car, int cant, int punt) : caracter(toupper(car)) , cantidad(cant) , puntuacion(punt)
      {};
00047
00052     void setCaracter(char c) { caracter = c; };
00053
00058     void setCantidad(int c) { cantidad = c; };
00059
00064     void setPuntuacion (int p) { puntuacion = p; };
00065
00074     void setPuntuacion (int apariciones , pair<int,int> max_min);
00075
00080     char getCaracter() const { return caracter; };
00081
00086     int getCantidad() const { return cantidad; };
00087
00092     int getPuntuacion() const { return puntuacion; };
00093
00102     bool operator<(const Letra& l) const;
00103
00110     friend ostream& operator<<(ostream& salida, const Letra& letra);
00111
```

```
00118     friend istream& operator»(istream& entrada, Letra& letra);
00119 };
00120
00121
00122 // ------------------------ Clase ConjuntoLetras -----------------------------
00131 class ConjuntoLetras {
00132 private:
00133     set<Letra> letras;
00134 public:
00144     Letra getLetra (char caracter) const;
00145
00152     friend ostream& operator«(ostream& salida, const ConjuntoLetras& conj);
00153
00160     friend istream& operator»(istream& entrada, ConjuntoLetras& conj);
00161
00167     void insert (const Letra &letra);
00168
00169
00170     // -------------------------- Clase Iterator -------------------------
00175     class iterator {
00176     private:
00177         set<Letra>::iterator el_iterador;
00178     public:
00179         iterator() {}
00180
00181         iterator(set<Letra>::iterator iter) : el_iterador(iter) {};
00182
00183         bool operator!=(const ConjuntoLetras::iterator &iter);
00184
00185         bool operator==(const ConjuntoLetras::iterator &iter);
00186
00187         const Letra& operator*() const;
00188
00189         iterator& operator++();
00190
00191         iterator& operator=(const ConjuntoLetras::iterator &iter);
00192     };
00193
00194
00195     // ------------------------ Clase Const_iterator -------------------------
00200     class const_iterator {
00201     private:
00202         set<Letra>::const_iterator el_iterador;
00203     public:
00204
00205         const_iterator() {}
00206
00207         const_iterator(set<Letra>::const_iterator iter) : el_iterador(iter) {};
00208
00209         bool operator!=(const ConjuntoLetras::const_iterator &iter);
00210
00211         bool operator==(const ConjuntoLetras::const_iterator &iter);
00212
00213         const Letra& operator*() const;
00214
00215         const_iterator& operator++();
00216
00217         const_iterator& operator=(const ConjuntoLetras::const_iterator &iter);
00218     };
00219
00220
00225     iterator begin();
00226
00231     iterator end();
00232
00237     const_iterator begin() const;
00238
00243     const_iterator end() const;
00244 };
00245 #endif
```

Index