UNIVERSIDAD
DE GRANADA

Estructuras de datos

2º Grado en Ingeniería Informática

Práctica 4. STL e iteradores

DECSAI

Departamento de Ciencias de la Computación e Inteligencia Artificial

Índice de contenidos

1 Introducción

Los objetivos que se pretenden alcanzar con la realización de este guión de prácticas son los siguientes:

- Practicar con tipos de datos abstractos (TDA) implementados en la Standard Template Library (STL).
- Aprender a definir y utilizar iteradores.
- Seguir asimilando los conceptos de documentación de un TDA.

Para realizar correctamente este guión de prácticas se deben haber estudiado los siguientes temas:

- Tema 1: Introducción a la eficiencia de los algoritmos.
- Tema 2: Abstracción de datos.
- Tema 3: Tipos de datos contenedores básicos.

Finalmente, aunque no es un requisito indispensable, haber realizado la práctica 2 ayudará a entender mejor el problema.

2 Standard Template Library (STL)

El objetivo de esta práctica es hacer uso de la STL de C++ (`http://www.cplusplus.com/reference/stl/`). Para ello, nos vamos a centrar en resolver un problema utilizando dicha librería. Previo al uso de la STL, se requiere el diseño de nuevos TDA eficientes para resolver un problema dado.

3 Ejercicio

Al igual que en la práctica 2, queremos desarrollar una aplicación que nos permita manejar un diccionario compuesto por una serie de palabras, cada una de las cuales puede contener varias definiciones. Con este fin, desarrollaremos varios TDA:

- `Término`. Se compone de una palabra y una o más definiciones asociadas a esa palabra. Cada una de las definiciones puede contener más de una palabra.
- `Diccionario`. Es una colección de términos ordenados alfabéticamente.

Se pide desarrollar el TDA `Término` y el TDA `Diccionario`. En particular, para cada uno de estos TDA, se pide:

- Dar la especificación. Establecer una definición y el conjunto de operaciones básicas.
- Para el TDA `Término`, dar una representación usando los contenedores `pair` y `vector` de la STL. Además, se debe definir un iterador que nos permita recorrer las definiciones asociadas a una palabra.

- Para el TDA `Diccionario`, dar una representación usando el contenedor `set` de la STL. Es imprescindible definir también un iterador para poder iterar sobre los `Términos` del diccionario.
- Para la estructura de datos del *tipo rep*, establecer cuál es el invariante de la representación y la función de abstracción.
- Fijado el *tipo rep*, realizar la implementación de las operaciones.
- Documentar los TDA desarrollados.
- Haciendo uso de `pruebadiccionario.cpp`, probar los TDA desarrollados.

3.1 Fichero de datos

Para probar el programa se usará un fichero compuesto por una serie de líneas. Cada una de ellas se corresponde con una palabra y su definición. El formato del fichero es el siguiente (ver Listado 3.1):

```
palabra;definición
```

```
cockney;the nonstandard dialect of natives of the east end of London
cockney;a native of the east end of London
cocteau;French writer and film maker who worked in many artistic media (1889−1963)
cody;United States showman famous for his Wild West Show (1846−1917)
cognac;high quality grape brandy distilled in the Cognac district of France
cohan;United States songwriter and playwright famous for his patriotic songs (1878−1942)
coke;carbon fuel produced by distillation of coal
coke;Coca Cola is a trademarked cola
coke;street names for cocaine
col;a pass between mountain peaks
colbert;butter creamed with parsley and tarragon and beef extract
cole;coarse curly−leafed cabbage
cole;a hardy cabbage with coarse curly leaves that do not form a head
coleridge;English romantic poet (1772−1834)
colette;French writer of novels about women (1873−1954)
colleen;an Irish girl
collier;someone who works in a coal mine
collins;tall  iced drink of liquor (usually gin) with fruit juice
collins;English writer noted for early detective novels (1824−1889)
cologne;a perfumed liquid made of essential oils and alcohol
```

Listado 3.1. Parte del fichero de datos `diccionario.txt`.

Como se puede observar, una palabra puede tener tantas entradas como definiciones posibles. En el directorio **datos** tenéis un diccionario de palabras en inglés (`diccionario.txt`) para probar los TDA desarrollados.

3.2 Módulos a desarollar

Se desarrollarán al menos dos módulos (aunque es posible añadir más módulos en caso de ser necesarios):

- El módulo asociado al TDA `Término` (`Termino.cpp` y `Termino.h`).

- El módulo asociado al TDA `Diccionario` (`Diccionario.cpp` y `Diccionario.h`).

Todos los módulos tienen que tener la documentación suficiente para que cualquier usuario pueda usarlos sin problemas. Para ello, se usará el programa `doxygen`.

3.2.1 Módulo `Término`

Este módulo está dedicado a la especificación e implementación del TDA `Término`. Este TDA está formado por una palabra y un conjunto de definiciones asociadas a esa palabra. Las operaciones más relevantes de este TDA son:

- Constructor por defecto.
- Constructor por parámetros.
- Constructor por copia.
- Operadores de consulta:
  - Obtener la palabra.
  - Obtener las definiciones asociadas a la palabra.
  - Obtener el número de definiciones asociadas a la palabra.
- Operadores de modificación:
  - Establecer la palabra.
  - Añadir nuevas definiciones asociadas a la palabra.

Además, se podrían sobrecargar los operadores de escritura y lectura en flujos, así como añadir cualquier otra operación que se considere necesaria.

3.2.2 Módulo `Diccionario`

Este módulo está dedicado a la especificación e implementación del TDA `Diccionario`. Este TDA está formado por un conjunto de términos ordenados alfabéticamente (de acuerdo a la palabra asociada a cada término). Las operaciones más relevantes de este TDA son:

- Constructor por defecto.
- Constructor por parámetros.
- Constructor por copia.
- Operadores de consulta:
  - Obtener las definiciones asociadas a una palabra.
  - Obtener todos los términos del diccionario.
  - Obtener el número de términos del diccionario.
- Operadores de modificación:
  - Añadir un nuevo término.
  - Eliminar un término.

Además, se deben implementar las siguientes operaciones:

- Sobrecarga de los operadores de escritura y lectura en flujos.
- Filtrado por intervalo. Dado un diccionario, el objetivo es obtener un subconjunto de este diccionario que incluya únicamente los términos cuya palabra asociada esté en el intervalo especificado por `[carácter_inicio, carácter_fin]`.
- Filtrado por palabra clave. Dado un diccionario, el objetivo es obtener un subconjunto de este diccionario que incluya únicamente las palabras en cuya definición aparezca la palabra clave. Si una palabra tiene varias definiciones, solo se devolverían como resultado del filtrado por palabra clave aquellas definiciones relacionadas con la palabra clave.
- Recuento de definiciones. Dado un diccionario, el objetivo es obtener el número total de definiciones, el máximo de definiciones asociadas a una única palabra y el promedio de definiciones por palabra.

3.3 Iteradores en `Término` y `Diccionario`

Los TDA `Término` y `Diccionario` son TDA contenedores de otros tipos de datos. Para implementar algunas de las operaciones anteriores es necesario acceder a los elementos que componen estos TDA. Un iterador es un nuevo TDA que abstrae la idea de indexación de elementos de un contenedor, permitiendo acceder a estos elementos de forma individual e iterar sobre el contenedor.

Para utilizar este recurso, tanto `Término` como `Diccionario` deben definir un tipo `iterator` y otro `const_iterator` propio. El tipo `iterator` se empleará para acceder y recorrer elementos del contenedor permitiendo su modificación, mientras que el tipo `const_iterator` no permitirá la modificación de los elementos del contenedor. Además de la definición de estos tipos, los TDA `Término` y `Diccionario` deben dotar a estos tipos `iterator` y `const_iterator` de las operaciones básicas para manejo de iteradores: operadores `++`, `--`, `*`, `=`, `==`, `!=`, funciones `begin` y `end`.

Los tipos contenedores de la STL incluyen la implementación de los tipos `iterator` y `const_iterator` con las operaciones básicas enumeradas anteriormente. Como nuestros TDA `Término` y `Diccionario` utilizan contenedores de la STL como *tipo rep*, podemos usar esta funcionalidad ya implementada para dotar a nuestros TDA de iteradores. Para ello, en primer lugar, se debe realizar una definición de tipos `iterator` y `const_iterator` para los TDA `Término` y `Diccionario` que los relacione con los tipos `iterator` y `const_iterator` de los *tipos rep* subyacentes de la STL (ver Listado 3.2 y Listado 3.3).

```
class Termino {

 public:

  ...

  typedef vector<string>::iterator iterator;
  typedef vector<string>::const_iterator const_iterator;

  ...

}
```

Listado 3.2. Iteradores del TDA `Término`.

```
class Diccionario {

 public:

  ...

  typedef set<Termino>::iterator iterator;
  typedef set<Termino>::const_iterator const_iterator;

  ...

}
```

Listado 3.3. Iteradores del TDA `Diccionario`.

En segundo lugar, debemos programar los métodos `begin` y `end` para los TDA `Término` y `Diccionario` que devuelvan el `iterator/const_iterator` devuelto a su vez por los métodos `begin` y `end`, respectivamente, de los *tipos rep* subyacentes de la STL. Por ejemplo, para el TDA `Diccionario` habría que implementar los cuatro métodos mostrados en el Listado 3.4 (lo mismo habría que hacer para el TDA `Término`).

```
class Diccionario {

 public:

  ...

  Diccionario::iterator begin();
  Diccionario::const_iterator begin() const;
  Diccionario::iterator end();
  Diccionario::const_iterator end() const;

  ...

}
```

Listado 3.4. Métodos `begin` y `end` del TDA `Diccionario`.

Finalmente, se debe dotar a los iteradores creados para los TDA `Término` y `Diccionario` de los operadores básicos enumerados anteriormente (`++`, `--`, `*`, `=`, `==`, `!=`). Sin embargo, si nuestro método `begin` devuelve el iterador del *tipo rep* subyacente de la STL, como todos los contenedores de la STL ya tienen implementados estos operadores para sus iteradores, simplemente se invocará a estos operadores sobre el iterador de `Término` y `Diccionario` para beneficiarse de esta funcionalidad ya programada en la STL.

De este modo, la sobrecarga del operador de escritura en flujo del TDA `Diccionario` podría implementarse de la forma mostrada en el Listado 3.5.

```
 ostream& operator<< (ostream & os, const Diccionario & d){
   Diccionario::const_iterator it;
   for(it = d.begin(); it != d.end(); ++it){
     os << *it;
   }
   return os;
 }
```

Listado 3.5. Sobrecarga del operador de escritura en flujo del TDA `Diccionario`.

Para que la implementación mostrada en el Listado 3.5 funcione correctamente, es necesario tener también sobrecargado el operador de escritura en flujo del TDA `Termino`. Una posible sobrecarga de este operador puede verse en el Listado 3.6

```
ostream& operator<< (ostream & os, const Termino & t){
  Termino::const_iterator it;
  for(it = t.begin(); it != t.end(); ++it){
    os << t.termino.first << "-->" << (*it) << endl;
  }
  return os;
}
```

Listado 3.6. Sobrecarga del operador de escritura en flujo del TDA `Termino`.

3.4 Fichero de prueba

En el directorio **src** se encuentra el fichero `pruebadiccionario.cpp` (ver Listado 3.7). Este código debe funcionar con los TDA desarrollados.

```
#include "Diccionario.h"
#include <fstream>
#include <iostream>

using namespace std;

int main(int argc, char * argv[]){

 if (argc!=2){
    cout<<"Dime el nombre del fichero con el diccionario"<<endl;
    return 0;
 }

 ifstream f (argv[1]);
 if (!f){
  cout<<"No puedo abrir el fichero "<<argv[1]<<endl;
  return 0;
 }

 Diccionario mi_diccionario;
 f>>mi_diccionario; //Cargamos en memoria el diccionario
 //Usamos un iterador para recorrer los terminos del diccionario e imprimirlos
 Diccionario::const_iterator it;
 for(it = mi_diccionario.begin(); it != mi_diccionario.end(); ++it){
   cout << (*it) << endl;
 }

 /* Exhibir aqui la funcionalidad programada para el TDA Diccionario / TDA
    Termino
  - Obtener las definiciones asociadas a una palabra
  - Obtener el (sub)diccionario de términos comprendidos en
      [caracter_inicial, caracter_final]
  - Obtener el (sub)diccionario de términos asociados a una palabra clave.
      Ejemplo: el diccionario de terminos asociados a "color"
  - Obtener el numero total de definiciones, el maximo de definiciones
      asociadas a una unica palabra y el promedio de definiciones por palabra
```

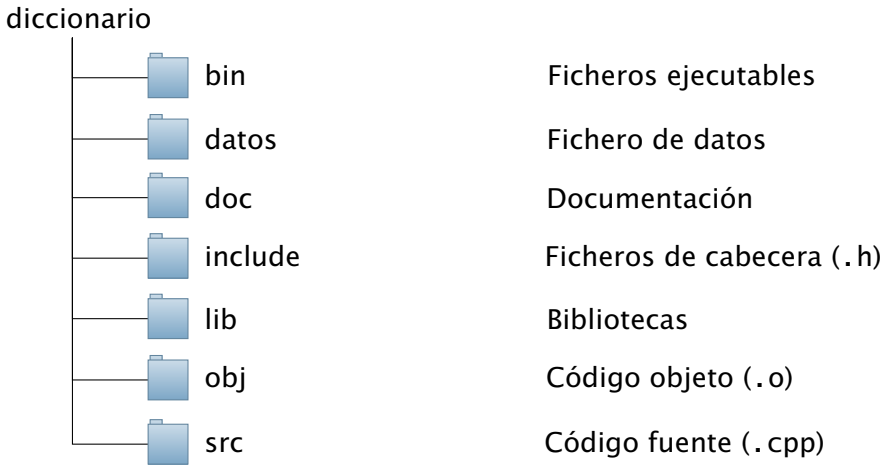
```
   - Cualquier otra funcionalidad que considereis de interes
  */

}
```

Listado 3.7. `pruebadiccionario.cpp`.

4 Entrega

Se deberán empaquetar todos los ficheros relacionados con la práctica en un fichero con nombre `diccionario.tar`. No se incluirán ficheros objeto ni ficheros ejecutables. Por tanto, se deben eliminar los ficheros temporales y aquellos que se generen a partir de los fuentes.

Se deberá incluir el fichero `Makefile` para realizar la compilación. Los ficheros deben estar distribuidos en directorios:



Para realizar la entrega, se deben eliminar los ficheros que no se incluirán en ella. Estando en la carpeta superior (en el mismo nivel de la carpeta **diccionario**) se ejecutará la siguiente instrucción:

```
tar -cvf diccionario.tar diccionario
```

tras lo cual se dispondrá de un nuevo fichero llamado `diccionario.tar` que contendrá la carpeta **diccionario** y todas las carpetas y ficheros que cuelgan de ella.