Práctica 3:
Palabras y Traducciones

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

DECSAI

Estructuras de Datos

Grado en Ingeniería Informática. Grupo D.

Índice de contenido

1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

1. Practicar con T.D.A implementados en la STL
2. Resolver un problema eligiendo las mejores estructuras de datos para las operaciones que se solicitan
3. Seguir asimilando los conceptos de documentación de un tipo de dato abstracto (T.D.A)

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

2. STL

EL objetivo en esta práctica es hacer uso de la librería STL en C++ http://www.cplusplus.com/reference/stl/. Con tal objetivo vamos a centrarnos en resolver un problema con dicha librería. Previo al uso de esta librería, como en la anterior práctica se requiere que el alumno diseñe nuevos T.D.A eficientes para resolver el problema.

3. Ejercicio

Esta práctica trabajaremos con conjuntos de palabras pertenecientes a un idioma y con traducciones de palabras entre diferentes idiomas. El primer objetivo en esta práctica será implementar los tipos de datos Palabra, Palabras y Traductor. El segundo objetivo será obtener un nuevo traductor a partir de dos traductores de entrada, de forma que si el primer traductor traduce por ejemplo del español al inglés y el segundo de español a francés el nuevo traductor traducirá de inglés a francés.

Pero antes de abordar este problema vamos a retomar los TDA definidos en la práctica 2 y los vamos a redefinir. Así las tareas que debemos abordar son las siguientes:

1. Modificar el TDA Palabra con dos representaciones distintas. Las representaciones se dan en el material (ver ficheros palabra1.h y palabra2.h). El estudiante tendrá que adaptar la implementación a estas representaciones.
2. Modificar el TDA Palabras
3. Probar que funciona el programa test (ver test.cpp situado en el directorio src del material). Este fichero no debe modificarse.
4. Crear el TDA Traductor. La representación se da en el material.
5. Crear un fichero test_traductor.cpp para probar el buen funcionamiento de las operaciones definidas para el T.D. Traductor.
6. Crear el programa ***conca_traductor.***

3.1. TDA Palabra

Un objeto del TDA Palabra contiene una palabra y el tipo de palabra que es. Como se vió en la práctica 2 un objeto Palabra se describe por el tipo de palabra, la palabra, un conjunto de sinónimos de la palabra.

En esta práctica vamos a modificar la representación del TDA Palabra de dos formas: 1) los sinónimos se almacenarán en un vector de la STL (fichero *palabra1.h*) o 2) sinónimos se almacenarán como un objeto de tipo set de la STL (fichero *palabra2.h*). Modificar la implementación de acuerdo a estas nuevas representaciones. Así la clase Palabra tendrá las siguientes representaciones

```
//palabra1.h
ifndef PALABRA__H
#define PALABRA__H
#include <iostream>
#include "vector"
using namespace std;
class Palabra{
  private:
    string tipo;
    string  word;
    vector<string> sinonimos;
...
```

```
//palabra2.h
#ifndef PALABRA__H
#define PALABRA__H
#include <iostream>
#include <set>
using namespace std;
class Palabra{
  private:
    string tipo;
    string  word;
    set<string> sinonimos;
...
```

Para escoger de forma dinámica la representación de sinónimos como ***vector*** (palabra1.h) o como ***set*** (*palabra2.h*) tendremos los ficheros *palabra.h* y *palabra.cpp* que serán los que indique que representación se escoge:

```
//palabra.h
#define CUAL_COMPILA 1
#if CUAL_COMPILA==1
     #include "palabra1.h"
#elif CUAL_COMPILA==2
     #include "palabra2.h"
#endif
```

```
//palabra.cpp

#include “palabra.h”
#if CUAL_COMPILA==1
     #include "palabra1.cpp"
#elif CUAL_COMPILA==2
     #include "palabra2.cpp"
#endif
```

Como se puede observar en estos ficheros la definición ***CUAL_COMPILA*** determina la representación escogida de Palabra.

Además para recorrer los sinónimos se ha definido en Palabra dos iteradores

De forma que nuestra clase será algo parecido a :

```
class Palabra{
  private:
    string tipo;
    string  word;
    vector<string> sinonimos;
  public:
       ….
       class iterator_sinonimos{
          private:
            vector<string>::iterator it;
          public:
               ...
           friend class Palabra;
       };
```

```
class const_iterator_sinonimos{
          private:
            vector<string>::const_iterator it;
          public:
               ….
           friend class Palabra;
      };
    iterator_sinonimos begin();
    iterator_sinonimos end();
    const_iterator_sinonimos begin()const;
    const_iterator_sinonimos end()const;

};//end Palabra
```

Para poder comprobar la compilación y buen funcionamiento de este módulo se aconseja al estudiante comentar en test.cpp todo excepto la sección 0 y 1, y entonces ejecutar **make bin/test1**. Ejecutar en la lineas de comandos **bin/test1 datos/dic_english.txt** y comprobar que funciona correctamente. Para testear el TDA Palabra con la segunda representación modificar en palabra.h **CUAL_COMPILA** a 2 y volver a ejecutar **make clean** y **make bin/test1**. Ejecutar y comprobar que es correcto.

3.2. T.D.A Palabras

Un objeto del T.D.A Palabra contiene una colección de palabras en un idioma. En esta práctica la representación del T.D.A Palabras vendrá dada por un string que mantendrá el idioma y el conjunto de palabras se almacenará en un objeto de tipo *list* de la STL. Así podemos ver en el fichero palabras.h la siguiente representación:

```
//palabras.h
class Palabras{
 private:
    string idioma;
    list<Palabra> datos;
….
```

Por lo tanto el estudiante implementará las operaciones descritas en *palabras.h* adaptándolas a la representación. Además deberá completar en *palabras.h* la representación de un iterador que permitirá recorrer las palabras almacenadas en el objeto datos. Como ejemplo al estudiante se le ha dado implementado el iterador de palabras (ver *palabras1*.h o *palabras2.h*) que puede analizar para ver como podría ahora implementar el iterador en *palabras.h*. Además deberá implementar los operadores *begin* y *end* para estsablecer el comienzo y fin de un iterador.

3.3. Programa test

El TDA Palabra y Palabras tiene que funcionar con el fichero test.cpp. Para ello se debe generar el programa test (p.e ejecutar **make bin/test**). Un ejemplo de llamada podría ser:

```
prompt>bin/test datos/dic_english.txt
```

Este programa recibe un fichero con un conjunto de palabras en un idioma. Con estos datos testea los TDA Palabra y Palabras, además de los iteradores previamente comentados. Si visualizamos el fichero test.cpp el alumno/a verá seis partes diferenciadas:

- ***Sección 1***: En esta sección se comprueba la declaración, lectura, consulta y escritura del T.D.A ***Palabra.***
- ***Sección 2:*** Se prueba el iterator de palabra para recorrer los sinónimos de una palabra.
- ***Sección 3:*** En esta sección se comprueba la declaración, lectura y escritura del T.D.A ***Palabras***. La lectura hará uso de los métodos de inserción. Hay que tener en cuenta que los ficheros de entrada no están ordenados. Y por lo tanto cada vez que se inserta una palabra se debe insertar de forma ordenada por valor de palabra y a igualdad de valor de palabra se ordena por tipo. Además se comprobarán otras operaciones como número de elementos y la operación de borrado. Se prueba la clase iterator de Palabras.
- ***Sección 4:*** Se prueba la creación de un nuevo objeto Palabras pero ordenado por tipo y a igualdad de tipo por palabra. Se propone al estudiante que implemente la operación **getOrdenadasTipo()** usando una ordenación clásica (p.e burbuja) y compruebe lo lento que va. Alternativamente el estudiante podría invocar al método sort de list para realizar la ordenación. Esto es posible siempre y cuando se haya definido e implementado el operador menor entre dos palabras. Este método esta definido e

implementado en palabra.h y palabra.cpp, respectivamente.

- ***Sección 5:***En esta sección se muestran varias informaciones sobre el conjunto de palabras. En primer lugar se lista los tipos diferentes en el conjunto de palabras. Este conjunto de tipos lo obtenemos en un vector dinámico de forma ordenada. En segundo lugar se obtiene las palabras de un determinado tipo, para ello creamos un nuevo objeto Palabras. Y finalmente obtenemos los tipos de una determinada palabra. Una palabra puede tener diferentes tipos por ejemplo la palabra ***"sol"*** tiene como tipos "relations" (p.e "Mi hermano es un sol") o puede tener el tipo "geography" en el sentido del astro Sol.
- ***Sección 6***: En esta sección se comprueba si dos palabras dadas por el usuario y con un tipo concreto son sinónimas.

Como ya se comentó en la práctica 2 el modo de trabajo que se recomienda al estudiantes es el siguiente:

1. En primer lugar desarrollar el T.D.A Palabra. En el directorio ***include*** del material se ha dado el fichero *palabra1.h (y palabra2.h)* donde se puede ver la representación del TDA junto con las cabeceras de las operaciones propuestas para el T.D.A Palabra. El alumno/a deberá aquí documentar el T.D.A. Para ello dará una especificación del T.D.A junto con la función de abstracción e invariante de representación. Ahora cada una de las operaciones deberá documentarse. Para generar la documentación se usará ***doxygen***. En el directorio ***src*** crearemos el el fichero *palabra1.cpp* (o palabra2.cpp) que contendrá la implementación de las operaciones indicadas en *palabra1.h*. Compilar el fichero palabra.cpp para obtener palabra.o. Para ello podemos ejecutar ***make obj/palabra.o.*** Si la compilación no genera ningún fallo pasamos al paso 2. Dependiendo de si estamos testeando la representación con vector (palabra1.h) o con set (palabra2.h) modificaremos en palabra.h la definición de **CUAL_COMPILA**.
2. En el fichero test.cpp comentar todo para dejar solamente la sección 0 y 1 que testea el T.D.A Palabra. Para ello comentar el resto de secciones y comentar ***#include "palabras.h".*** Ahora ejecutar ***make bin/test1.*** Si compila bien ahora probamos el funcionamiento ejecutando ***bin/test1 datos/dic_english.txt.*** Si todo funciona bien pasamos al paso 3.
3. En este paso desarrollaremos el T.D.A Palabras. En el directorio ***include*** del material se ha dado el fichero *palabras.h* donde se puede ver la representación del TDA junto con las cabeceras de las operaciones propuestas para el T.D.A Palabras. El alumno deberá aquí documentar el T.D.A. Para ello dará una especificación del T.D.A junto con la función de abstracción e invariante de representación. Ahora cada una de las operaciones deberá documentarse. Para generar la documentación se usará ***doxygen***. En el directorio ***src*** crearemos el fichero *palabras.cpp* que contendrá la implementación de las operaciones indicadas en *palabra.h*. Compilar el fichero palabras.cpp para obtener palabras.o. Para ello podemos ejecutar ***make obj/palabras.o.*** Si la compilación no genera ningún fallo pasamos al paso 4.
4. En el fichero test.cpp descomentamos ***#include "palabras.h".*** A continuación vamos descomentando las siguientes secciones. Se propone descomentar una sección, compilar y ejecutar. Si todo va bien pasamos a la siguiente sección. Así hasta completar todas las secciones.

Para generar la documentación es necesario que los fichero ***.h*** contengan la documentación doxygen. Una vez completada se podrá ejecutar ***make documentacion.*** Si todo va bien en el directorio doc debe haber creado un directorio html. Visualizar en un navegador el fichero ***index.html*** y comprobar el resultado obtenido.

3.4. TDA Traductor

El alumno/a creará el TDA Traductor que representa como se traduce un conjunto de palabras en un idioma origen en un idioma destino.

La representación que vamos a dar para el TDA **Traductor** es:

```
class Traductor{
 private:
     string idioma_origen,
            idioma_destino;
     //first tipo y palabra en el idioma origen
     //second  palabra en el idioma destino
     multimap<pair<string,string>,string> tra;
 public:
…
 class iterator{
 private:
 multimap<pair<string,string>,string>::iterator
it;
     …
     };
    class const_iterator{
        private:

multimap<pair<string,string>,string>::const_ite
rator it;
     …
     }:
     iterator begin();
     iterator end();
     const_iterator begin()const;
     const_iterator end()const;

};//end Traductor
```

Como se puede observar hemos definido para Traductor dos iteradores uno no constante y otro constante, que itera por las traducciones.

De entre las operaciones definidas para Traductor cabe resaltar dos de ellas:

- **Palabras getTraduccion(const string &tipo, const string &p)const; .** Esta operación obtiene un objeto de tipo Palabras con todas las palabras que son las traducciones en el idioma destino de la palabra de entrada *p* con tipo *tipo.* Por ejemplo si tipo=adjective y p =completo el objeto que devuelve debe contener como conjunto de palabras {full,total} siendo en el ejemplo el idioma origen español e idioma destino inglés.
- **Traductor TraductorInverso()const;**. Esta operación obtiene un nuevo objeto de tipo Traductor que es el resultado de invertir el Traductor implicito. Es decir si el objeto Traductor implicito es de español a inglés obtenemos como traductor de salida de inglés a español.

3.5. Programa conca_traductor

El objetivo es obtener un nuevo traductor a partir de dos objetos de tipo Traductor relacionados. Relacionados en el sentido que el idioma origen es el mismo. Entonces el programa obtendrá el traductor que traduce entre los idiomas no comunes. Una posible ejecución desde la línea de órdenes podría ser la siguiente:

```
prompt>bin/conca_traductor datos/tra_spanish_english.txt datos/tra_spanish_french.txt
datos/tra_english_french.txt
```

Este programa recibe dos fichero con dos traductores (ver sección 3.7). Obtiene un nuevo traductor que traduce los dos idiomas destino (inglés y francés).

3.6. Fichero con los palabras.

El fichero con las palabras en un idioma está compuesto de una serie de líneas.

```
Ingles
#TIPO;PALABRA;SINONIMOS(separadas por ;)
country;China;People's Republic of China;mainland China;Communist China;Red
China;PRC;Cathay;
country;People's Republic of China;China;mainland China;Communist China;Red
China;PRC;Cathay;
….
```

El fichero contiene:

1. La primera línea contiene el idioma de las palabras.

2. La segunda línea es un comentario. Indicando que atributos tendrá cada una de las palabras.
3. A continuación en cada línea viene la información de cada palabra. Cada atributo de un palabra se encuentra separada por ";". Los atributos son
    - Tipo de la palabra
    - Palabra (puede contener mas de un vocablo p.e " People's Republic of China").
    - Secuencia de palabras sinónimas, separadas por ";".

3.7. Fichero con un Traductor

El fichero con la información de un Traductor se compone de:

1. La primera linea con el idioma origen
2. La segunda línea con el idioma destino
3. Un comentario, que se inicia con '#', describiendo las siguientes lineas que describe las lineas que vienen a continuación.
4. Cada linea a continuación conforma la traducción de una palabra. Una traducción se compone del tipo de palabra, palabra en el idioma origen, palabra en el idioma destino. Los tres campos aparecen separados por ';'. Observe que puede habar más de una linea para un mismo tipo y palabra en el idioma origen.

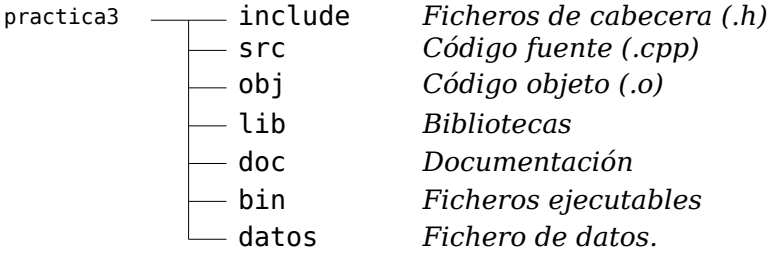
Un ejemplo de fichero con un traductor podría ser:

```
Spanish
English
#TIPO;PALABRA IDIOMA ORIGEN;PALABRA IDIOMA DESTINO
country;China;China
country;China;People's Republic of China
country;China;mainland China
….
profession;actor;actor
profession;actor;histrion
profession;actor;player
profession;actor;thespian
...
```

4. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "practica3.*tgz*" y entregarlo antes de la fecha que se publicará en la página web de la asignatura (en PRADO). Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables. Es recomendable que haga una "limpieza" para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

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



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 "*practica3*") para ejecutar:

```
prompt% tar zcv practica3.tgz practica3
```

tras lo cual, dispondrá de un nuevo archivo practica3.tgz que contiene la carpeta practica3 así como todas las carpetas y archivos que cuelgan de ella.