Práctica 4: STL e Iteradores

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

DECSAI

Estructuras de Datos

Doble Grado en Ingeniería Informática y Matemáticas
Grado en Ingeniería Informática

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 la mejor estructura 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

En una academia de lenguajes queremos obtener una aplicación que nos permita traducir una palabra de un determinado idioma a otra u otras. Es decir una palabra de un idioma puede tener varias traducciones. Con tal fin vamos a desarrollar varios tipos de datos abstractos:

- Palabra: se compone de una palabra en el idioma origen y se transforma en una o más palabras en el idioma destino. Las palabras pueden contener más de un vocablo. P.e abandonar se traduce en inglés como “give up”. Las vocablos no contendrán acentos, tildes ni el carácter ñ. Si existe más de un vocablo se separa por un espacio.
- Traductor: Es un conjunto de palabras ordenadas por la palabra en el idioma origen.

Se pide desarrollar el T.D.A. Palabra definido en la práctica 2. Para el T.D.A Palabra se debe dar una representación usando los contenedores de la STL ***pair*** y ***set*** o ***multiset***.

Para el T.D.A Traductor usaremos para representarlo el contenedor ***map*** *o* ***multimap*** de la STL. Puede que sea necesario usar incluso más de un contenedor para acelerar las operaciones que se quieren implementar.

Será indispensable que el T.D.A Traductor tenga definido un ***iterador*** para poder iterar sobre las palabras del traductor. Y sobre el T.D.A Palabra también definiremos un ***iterador*** que nos permita recorrer las palabras traducción.

3.1. Traductor Inverso.

El objetivo es construir un programa que dado un traductor obtenga el traductor inverso.

Así por ejemplo si la entrada es el traductor de español-inglés, este programa obtendrá el traductor de inglés-español.

El comando podrá ejecutarse de dos formas. Una dando un único parámetro

```
prompt%  traductor_inverso spanish_english
```

En esta ejecución el programa recibe el traductor origen y su objetivo es obtener el traductor inverso (en este ejemplo english_spanish). La salida se hará en la salida estándar.

Si el programa se ejecuta dándole un parámetro más, este será el nombre del fichero donde dejaremos el traductor inverso.

```
prompt%  traductor_inverso spanish_english english_spanish
```

El T.D.A Traductor propuesto tiene que servir para poderlo usar con el siguiente código :

```
1.  #include "traductor.h"
2.  #include <fstream>
3.  #include <iostream>
4.  using namespace std;
5.  void GetTraductorInverso(const Traductor & t_origen, Traductor & t_destino){
6.
7.  }
8.  void ImprimeTraductor(const Traductor &T,ostream &os){
9.    Traductor::const_iterator it;
10.   for (it=T.begin(); it!=T.end();++it){
11.     os<<(*it).first<<";";
12.     vector<string>::const_iterator it_d;
13.     for (it_d=(*it).second.begin();
14.          it_d!=(*it).second.end();  ++it_d)
15.               os<<(*it_d)<<";";
16.  }
17. }
18. int main(int argc, char * argv[]){
19.  if (argc!=2 || argc!=3){
20.     cout<<".-Dime el nombre de fichero
21.              del traductor origen"<<endl;
22.    cout<<".-[opcionalmente] El nombre
23.              de fichero del    traductor destino"<<endl;
24.     return 0;
25.  }
26.  ifstream f (argv[1]);
27.  if (!f){
28.      cout<<"No puedo abrir el fichero "
29.     <<argv[1]<<endl;    return 0;
30.  }
31.  Traductor t_ori;
32.  f>>t_ori; //Cargamos en memoria, en el traductor.
33.
34.  Traductor t_des;
35.   GetTraductorInverso(t_ori,t_des);
36.   if (argc==2)
37.    ImprimeTraductor(t_des,cout);
38.  else{
39.     ofstream fout(argv[2]);
40.     if (!fout){
41.    cout<<"No puedo crear el fichero "
42.              <<argv[2]<<endl;
43.     return 0;
44.    }
45.    ImprimeTraductor(t_des,fout);
46.  }
    }
```

3.2. Concatenación de Traductores

Dados dos traductores, cada uno con su idioma origen e idioma destino, el objetivo de este programa es obtener el traductor que surge del idioma origen del primer traductor al idioma destino del segundo traductor. Así por ejemplo si los dos traductores de partida son ***spanish_english*** ***english_french*** al concatenar obtendremos el traductor de ***spanish_french***

El comando podrá ejecutarse desde la línea de órdenes de la siguiente manera:

```
prompt%  concatenar spanish_english english_french spanish_french
```

Los parámetros son los siguientes:

1. El nombre del fichero con el primer traductores a concatenar
2. El nombre del fichero con el segundo traductor a concatenar
3. El nombre del fichero del traductor salida.

3.3. Intersección de dos traductores.

Dados dos traductores, cada uno con su idioma origen y el mismo idioma destino, se pide establecer el traductor intersección. Obtendremos el traductor intersección como el traductor de las palabras intersección de los dos idiomas origen al idioma destino común. Este nuevo traductor contiene aquellas palabras que coinciden en ambos idiomas origen (contiene los mismos vocablos) y se traducen de forma semejante en el idioma destino (al menos tienen una palabra destino idéntica). Por ejemplo: teniendo como idiomas origen español y francés y destino común inglés tendremos como entradas de la intersección:

autobus;bus;omnibus;
menu;menu;
miel;honey
moto;moto;
normal;normal;
tomate;tomato;

Así por ejemplo **miel** es una palabra válida tanto en francés como en español y se traduce al inglés de igual forma (**honey**).

Este programa podrá ejecutarse desde la línea de órdenes de la siguiente manera:

```
prompt%  interseccion spanish_english french_english inter_spanishfrench_english
```

Los parámetros son los siguientes:

1. El nombre del fichero con el primer traductores
2. El nombre del fichero con el segundo traductor
3. El nombre del fichero del traductor intersección.

3.4. Fichero prueba

Para poder probar nuestro programa usaremos un fichero compuesto de una serie de líneas. Cada línea se corresponde con una palabra en el idioma origen y a continuación, separados por el carácter ';', las palabras en el idioma destino. Por ejemplo un trozo del fichero inglés-español es el que se muestra a continuación:

```
…..
warm;caliente;
warn;avisar;
warning;aviso;
warrant;garantia;orden;orden por escrito;
wash;lavar;
wash up;fregar;lavar;
wasp;avispa;
waste;desechos;detrito;acabar;
watch;reloj de pulsera;reloj;contemplar;mirar;observar;ver;
….
```

En el directorio **doc** tenéis ejemplos de ficheros traductores: english_spanish (inglés a español), spanish_english, french_english, english_french.

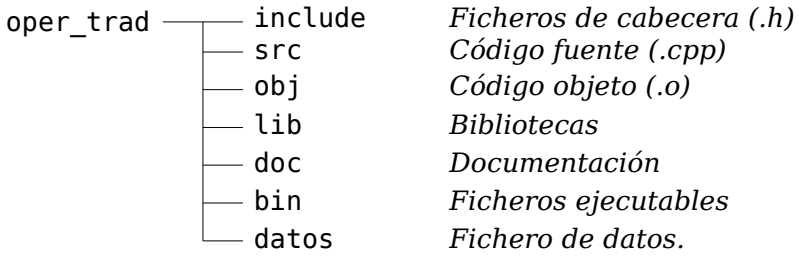
3.5. Módulos a desarrollar.

Habrá al menos dos módulos que deberéis desarrollar: 1) el módulo asociado a Palabra (palabra.cpp y palabra.h) y 2) el módulo asociado a Traductor (traductor.cpp e traductor.h). Es posible añadir más módulos si lo estimáis necesario.

4. Práctica a entregar

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

El 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 "*oper_trad*") para ejecutar:

```
prompt% tar zcv oper_trad.tgz oper_trad
```

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

5. Referencias

- [GAR06b] Garrido, A. Fdez-Valdivia, J. *"Abstracción y estructuras de datos en C++"*. Delta publicaciones, 2006.