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

Estructuras de Datos

Doble Grado en Ingeniería Informática y Matemáticas
Grado en Ingeniería Informática

Índice de contenido

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 e ciencia 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

Con la finalidad de mejorar nuestros conocimientos sobre Historia, queremos desarrollar una aplicación que nos permita obtener acontecimientos históricos sucedidos en una fecha y viceversa, averiguar el año en el que sucedió algún acontecimiento. Con este fin, desarrollaremos varios tipos de datos abstractos:

- EventoHistórico: Se compone de una fecha y uno o más eventos sucedidos en dicha fecha. La fecha representará únicamente el año en el que tuvieron lugar los acontecimientos en forma de número natural, y cada uno de los acontecimientos podrán contener más de una palabra.
- Cronología: Es un conjunto de eventos históricos ordenados por fecha.

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

Para el T.D.A Cronología usaremos para representarlo el contenedor ***map*** 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 Cronología tenga definido un ***iterador*** para poder iterar sobre los eventos históricos de la cronología. Y sobre el T.D.A EventoHistórico también definiremos un ***iterador*** que nos permita recorrer los eventos sucedidos en un año.

Además, se **implementará sobre el T.D.A Cronología la funcionalidad especificada en los siguientes epígrafes**. Para ello, debe elegir una de estas dos opciones: a) codificar esta funcionalidad en métodos de la propia clase Cronología; o b) codificar esta funcionalidad en funciones o procedimientos externos a la clase Cronología.

3.1. Unión de Cronologías.

El objetivo es construir un programa que dadas dos cronologías obtenga una cronología resultante con la unión de ambas.

Así, por ejemplo, si la entrada de este programa es la cronología con acontecimientos de Historia Universal del S.XX (fichero `datos/timeline_worldhistory.txt`) y la cronología con descubrimientos científicos del S.XX (fichero `datos/timeline_science.txt`), este programa obtendrá una cronología que incluye todos los eventos históricos de ambas (Hª Universal y descubrimientos científicos). De este modo, si en la cronología de Hª Universal del S.XX aparecen los acontecimientos:

1925#Benito Mussolini gains dictatorial powers in Italy#Mein Kampf is published#First televisual image created by John Logie Baird#Locarno Treaties are signed#Serum run to Nome

y en la cronología de descubrimientos científicos aparecen los acontecimientos:

1925#Erwin Schrödinger: Schrödinger equation (Quantum mechanics)

La cronología resultante debe incluir todos los acontecimientos anteriores para el año 1925 (el orden de los acontecimientos puede ser diferente al mostrado):

1925#Benito Mussolini gains dictatorial powers in Italy#Mein Kampf is published#First televisual image created by John Logie Baird#Locarno Treaties are signed#Serum run to Nome#Erwin Schrödinger: Schrödinger equation (Quantum mechanics)

El comando debe ejecutarse indicando los dos ficheros de cronologías que van a unirse (opcionalmente, un tercer nombre de fichero donde se almacena la cronlogía unión resultante):

```
prompt%  union_cronologias ../datos/timeline_worldhistory.txt ../datos/timeline_science.txt
```

Compruebe que el T.D.A Cronología propuesto y la implementación de la operación de Unión es compatible con el código especificado en `union_cronologia.cpp`

3.2. Filtrado por palabra clave

Dada una cronología, el objetivo de este programa es obtener un subconjunto de esta cronología que incluya únicamente los acontecimientos en los que aparezca la(s) palabra(s) clave(s) indicada como argumento del programa. Así, por ejemplo, si utilizamos de partida la cronología del fichero timeline_science.txt y la palabra clave “quantum”, esta función debe devolver una cronología que incluya únicamente los EventosHistóricos siguientes:

1900#Max Planck: Planck's law of black body radiation, basis for quantum theory

1924#Wolfgang Pauli: quantum Pauli exclusion principle

1988#The concept of a Quantum cellular automata was introduced thus advancing quantum computation and quantum computer

Si varios acontecimientos concurren el mismo año, sólo se devolverían como resultado del filtrado por palabra clave aquellos relacionados con la palabra clave.

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

```
prompt% filtradoPalabraClave timeline_science.txt quantum timeline_quantum.txt
```

Los parámetros son los siguientes: 1.El nombre del fichero con la cronología 2. Palabra clave (opcional). Si no se especifica la palabra clave en línea de comandos, el programa la pedirá por teclado. 3. El nombre del fichero para guardar la cronología obtenida. (opcional). Si no se especifica nombre de fichero, se imprimirá por la salida estándar.

3.3. Filtrado por intervalo

Dada una cronología, el objetivo de este programa es obtener un subconjunto de esta cronología que incluya únicamente los EventosHistóricos acontecidos en un intervalo de tiempo especificado ([anio_inicio, anio_fin]). Así, por ejemplo, si utilizamos de partida la cronología del fichero timeline_science.txt y el intervalo [1915, 1925], esta función debe devolver una cronología que incluya únicamente los EventosHistóricos siguientes:

1915#Albert Einstein: theory of general relativity

1924#Wolfgang Pauli: quantum Pauli exclusion principle

1925#– Erwin Schrödinger: Schrödinger equation (Quantum mechanics)

Nótese que el intervalo de tiempo se considera cerrado, es decir, deben incluirse los extremos.

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

```
prompt% filtradoIntervalo timeline_science.txt 1915 1925 timeline_quantum.txt
```

Los parámetros son los siguientes: 1.El nombre del fichero con la cronología 2. Límite inferior del intervalo. 3. Límite superior del intervalo. 4. El nombre del fichero para guardar la cronología obtenida. (opcional). Si no se especifica nombre de fichero, se imprimirá por la salida estándar.

3.4. Recuento de eventos

Dada una cronología, el objetivo de este programa es obtener descriptores estadísticos básicos relacionados con el volumen y distribución de EventoHistórico en el periodo de tiempo abarcado por la cronología. Para ello, se propone calcular el número total de años para los que se especifican EventoHistóricos en la cronología, el número total de acontecimientos, y el máximo y promedio de estos acontecimientos por año.

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

```
prompt% estadisticaEventos timeline_science.txt
```

Los parámetros son los siguientes: 1.El nombre del fichero con la cronología. La salida se imprimirá por la salida estándar.

3.5. Iteradores en EventoHistorico y Cronologia.

Los T.D.A. EventoHistorico y Cronologia son T.D.A.s contenedores de otros tipos de datos. Para implementar las operaciones 3.1 a 3.4 anteriores es necesario acceder a los elementos que componen estos T.D.A.s. Los **iteradores** son un nuevo T.D.A. 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, nuestras clases EventoHistórico y Cronología 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. El tipo const_iterator no permite la modificación de los elementos del contenedor. Además de la definición de estos tipos, las clases EventoHistórico y Cronología deben dotar a estos tipos iterator y const_iterator de las operaciones básicas para manejo de iteradores: operadores ++, – –, *, =, ==, !=, funciones begin y end.

Todos los tipos contenedores de la STL incluyen la implementación de los tipos iterator/const_iterator con las operaciones básicas enumeradas anteriormente. Dado que nuestro T.D.A. EventoHistórico y T.D.A Cronología utilizan contenedores de la STL como tipos rep, podemos hacer uso de esta funcionalidad ya implementada en la STL para dotar a nuestros tipos de iteradores. Para ello, en primer lugar debemos realizar una **definición de tipos iterator y const_iterator para las clases EventoHistórico y Cronología** que los relacione con los tipos iterator y const_iterator de los tipos rep subyacentes de la STL:

```
class EventoHistorico{
public:
...
        typedef typename set<string>::iterator iterator;
        typedef typename set<string>::const_iterator const_iterator;
...
}
```

```
class Cronologia{
public:
...
        typedef typename map<string, EventoHistorico>::iterator iterator;
        typedef typename map<string, EventoHistorico>::const_iterator const_iterator;
...
}
```

En segundo lugar, debemos programar los métodos begin() y end() para las clases EventoHistórico y Cronología 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 Cronología habría que implementar los siguientes cuatro métodos (igual para EventoHistórico):

```
class Cronología{
public:
...
        iterator begin ();
        const_iterator begin () const;
        iterator end ();
        const_iterator end () const;
...
}
```

Finalmente, debemos dotar a nuestros iteradores para Cronología y EventoHistórico de los operadores básicos de iteradores enumerados anteriormente (++, – – , *, etc.). 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 invocaremos estos operadores sobre el iterador de Cronología y EventoHistórico para beneficiarnos de esta funcionalidad ya programada en la STL.

De este modo, la implementación de una función que muestra una cronología completa en un ostream podría implementarse del siguiente modo:

```
//Método que recorre todos los EventosHistóricos de la cronología c y para cada
//EventoHistorico recorre todos sus acontecimientos -strings- para imprimirlos en os.

void ImprimeCronologia (const Cronologia &c, ostream &os){
   Cronologia::const_iterator it;
   for (it=c.begin(); it!=c.end();++it){
      os<<(*it).first<<"#";          //año.
      EventoHistorico::const_iterator it_ev;
      for (it_ev=(*it).second.begin(); it_ev!=(*it).second.end();++it_ev)
       os<<(*it_ev)<<"#";
   }
}
```

Además de los pequeños extractos de código para Cronología y EventoHistórico que se muestran en este guión, en el material adjunto a la práctica se proporciona la implementación de un T.D.A para guías telefónicas que ilustra la definición y utilización de contenedores de la STL y de iteradores sobre este TDA.

3.6. Módulos a desarrollar.

Habrá al menos dos módulos que deberéis desarrollar: 1) el módulo asociado a EventoHistórico (EventoHistorico.cpp y EventoHistorico.h) y 2) el módulo asociado a Cronología (Cronologia.cpp y Cronologia.h). Además, es necesario inlcuir los archivos fuente para las funcionalidades que se piden en los apartados 3.1 a 3.4: estadisticaEventos.cpp, filtradoIntervalo.cpp, unionCronologias.cpp y filtradoPalabraClave.cpp. Es posible añadir más módulos si lo estimáis necesario.

3.7. Ficheros de pruebas.

Para poder probar nuestro programa usaremos un fichero compuesto de una serie de líneas. Cada línea se corresponde con un año y a continuación, separados por el carácter '#', los acontecimientos que suciedieron en ese año. Por ejemplo un trozo del fichero de películas timeline_movies.txt:

```
…..
1990#Home Alone#Goodfellas#Ghost#An Angel at My Table#Edward Scissorhands#Miller's Crossing#Dances with Wolves#Total Recall
1991#The Silence of the Lambs#Terminator 2: Judgment Day#JFK#A Brighter Summer Day#Thelma & Louise#Beauty and the Beast
….
```

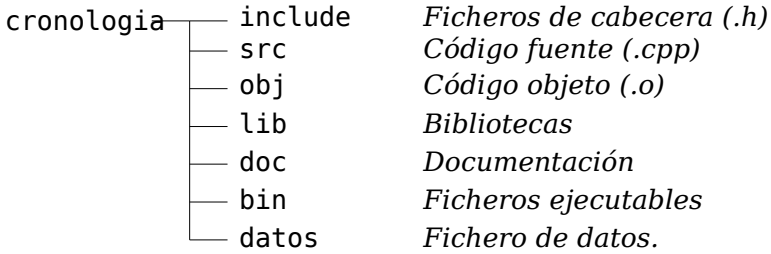
En el directorio **datos** tenéis ejemplos de ficheros de cronologías:

- timeline_world_history.txt – Eventos históricos del s. XX.
- timeline_movies.txt – Películas relevantes por año.
- timeline_science.txt – Descubrimientos científicos por año.
- timeline_algorithms.txt – Algoritmos relevantes por año.

4. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre “cronologia.*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.

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 “*cronologia*”) para ejecutar:

```
prompt% tar zcv cronologia.tgz cronologia
```

5. Referencias

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