Práctica 4: Estructuras de datos con la STL.

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 desde la perspectiva de la STL.
2. Construir un TDA Diccionario con el tipo list o un TDA Guía de teléfonos con el tipo map

Los requisitos para poder realizar esta práctica son:

1. Haber estudiado el Tema de T.D.A. Lineales con la STL

2. Ejercicio

Se han de implementar estos 2 tipos:

2.1. Crear el TDA Diccionario usando el tipo list de la STL

```
/*Tipo elemento que define el diccionario. T es el tipo de dato asociado a una clave que
no se repite (DNI p.ej.) y list<U> es una lista de datos (string p.ej) asociados a la clave
de tipo T. El diccionario está ordenado de menor a mayor clave.
*/
template <class T,class U>
struct data{
   T clave;
   list<U> info_asoci;
 };


 /*Comparador de datos. Ordena 2 registros de acuerdo a la clave de tipo T. Puede usarse como
 un funtor.
 */
template <class T, class U>
bool operator< (const data<T,U> &d1,const data <T,U>&d2){
        if (d1.clave<d2.clave)
                return true;
        return false;
}

/*Un diccionario es una lista de datos de los definidos anteriormente. Cuidado porque se
manejan listas de listas. Se añaden 2 funciones privadas que hacen más facil la
```

```
implementación
de algunos operadores o funciones de la parte pública. Copiar copia un diccioario en
otro y borrar elimina todos los elementos de un diccionario. La implementación de copiar
puede hacerse usando iteradores o directamente usando la función assign.
*/
template <class T,class U>
class Diccionario{
        private:

/* Busca la clave p en el diccionario. Si está devuelve un iterador a
            dónde está clave. Si no está, devuelve end() y deja el iterador de salida
            apuntando al sitio dónde debería estar la clave
            */
            bool Esta_Clave(const T &p, typename  list<data<T,U> >::iterator &it_out){

      /* Inserta un nuevo registro en el diccionario. Lo hace a través de la clave
            e inserta la lista con toda la información asociada a esa clave. Si el
            diccionario no estuviera ordenado habría que usar la función sort()
            */
            void Insertar(const T& clave,const list<U> &info){

/*Añade una nueva informacion asocida a una clave que está en el diccionario.
            la nueva información se inserta al final de la lista de información.
          Si no esta la clave la inserta y añade la informacion asociada.
            */
            void AddSignificado_Palabra(const U & s ,const T &p){

/* Devuelve la información (una lista) asociada a una clave p. Podrían
            haberse definido operator[] como
            data<T,U> & operator[](int pos){ return datos.at(pos);}
            const data<T,U> & operator[](int pos)const { return datos.at(pos);}
             */
             list<U>  getInfo_Asoc(const T & p) {

/*Devuelve el tamaño del diccionario*/
            int size()const{

/*Funciones begin y end asociadas al diccionario*/
```

/* añadir al menos 3 métodos nuevos que den más funcionalidad al diccionario

2.2. Crear el TDA Guía de teléfonos usando el tipo map de la STL

```
istream & operator>>(istream &is,pair<string,string> &d){

        getline(is,d.first,'\t');
        getline(is,d.second);
        return is;
}


class Guia_Tlf{
        private:
                map<string,string> datos; //si admites que haya nombres repetidos tendr□as que usar un


/**
                  @brief Acceso a un elemento
                  @param nombre: nombre del elemento  elemento acceder
                  @return devuelve el valor asociado a un nombre, es decir el teléfono
                */
                string & operator[](const string &nombre) {

**
                 @brief Insert un nuevo telefono
                 @param nombre: nombre clave del nuevo telefono
                 @param tlf: numero de telefono
                  @return : un pair donde first apunta al nuevo elemento insertado y bool es true si se ha insertado el nuevo tlf o false en caso contrario
                */
                pair<map<string,string>::iterator,bool>  insert(string nombre, string tlf){

**
                 @brief Insert un nuevo telefono
                 @param p: pair con el nombre y el telefono asociado
```

```
                    @return : un pair donde first apunta al nuevo elemento insertado y bool es true si se ha insertado el nuevo tlf o false en caso contrario
                */
                pair<map<string,string>::iterator,bool>  insert(pair<string,string> p){

/**
                  @brief Borrar un telefono
                  @param nombre: nombre que se quiere borrar
                  @note: en caso de que fuese un multimap borraria todos con ese nombre
                */

/**
                  @brief Union de guias de telefonos
                  @param g: guia que se une
                  @return: una nueva guia resultado de unir el objeto al que apunta this y g
                */
                Guia_Tlf operator+(const Guia_Tlf & g){


/**
                  @brief Diferencia de guias de telefonos
                  @param g: guia que se une
                   @return: una nueva guia resultado de la diferencia del objeto al que apunta this y g
                */
                Guia_Tlf operator-(const Guia_Tlf & g){

/**
                  @brief Escritura de la guia de telefonos
                  @param os: flujo de salida. Es MODIFICADO
                  @param g: guia de telefonos que se escribe
                  @return el flujo de salida
                 */


                friend ostream & operator<<(ostream & os, Guia_Tlf & g){

/**
```

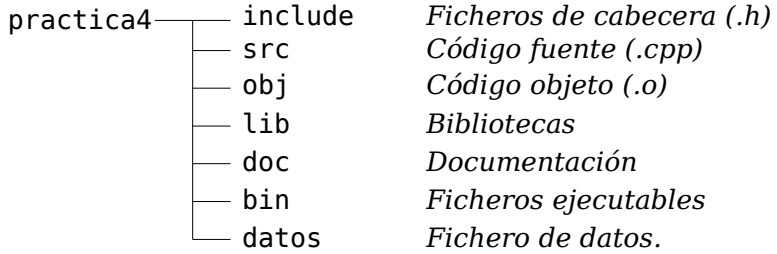
```
  @brief Lectura de  la guia de telefonos
  @param is: flujo de entrada. ES MODIFICADO
  @param g: guia de telefonos. ES MODIFICADO
  @return el flujo de entrada
*/

friend istream & operator>>(istream & is, Guia_Tlf & g)
```

/* añadir al menos 3 métodos nuevos que den más funcionalidad al la guia de teléfonos

3. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "dic-tel.*tgz*" y entregarlo antes de la fecha que se publicará en la página web de la asignatura. 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 "*practica4*") para ejecutar:

```
prompt% tar zcv dic-tel.tgz practica4
```

tras lo cual, dispondrá de un nuevo archivo dic-tel.tgz que contiene la carpeta practic4 así como todas las carpetas y archivos que cuelgan de ella.

4. Referencias

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