Práctica Final: Adivina Refrán

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 C

Índice de contenido

1.Introducción....3
2.Objetivo....3
3.Ejercicios....3
3.1. Tareas a realizar....3
3.1.1.Test refranes....4
3.1.2.Adivina....5
3.1.3.Estudio Prefijo....6
3.1.4.Refranes como una tabla hash....6
3.2.Ficheros....6
3.2.1.Fichero Refranes....6
3.2.2.Fichero csv....7
4.Módulos a desarrollar....7
4.1.Módulo Arbol General....7
4.2.Módulo Refranes....9
4.3.Módulo Refranes_hash....12
5.Práctica a entregar....13
Referencias....13

1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

- Resolver un problema eligiendo la mejor estructura de datos para las operaciones que se solicitan

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. Haber estudiado estructuras de datos jerárquicas: Árboles

2. Objetivo.

El objetivo de esta práctica es llevar a cabo el análisis, diseño e implementación de un proyecto. Con tal fin el alumno abordará un problema donde se requiere estructuras de datos que permiten almacenar grandes volúmenes de datos y poder acceder a ellos de la forma más eficiente.

3. Ejercicios

El alumno deberá implementar varios programas. Uno de ellos simulará el juego que dada una serie de letras el jugador es capaz de rellenar las letras que faltan para dar lugar a un refrán oculto.

Por ejemplo dado el siguiente refrán incompleto

N _ _ _ A _ _ T A _ _ _ _ A R _ A _ R _ N D _ _

sustituir correctamente los caracteres '_' da lugar a:

N U N C A E S T A R D E P A R A A P R E N D E R

3.1. Tareas a realizar

El alumno debe llevar a cabo las siguientes tareas:

1. Con objeto de que alumno practique con el T.D.A ArbolGeneral, se pide que implemente este tipo de dato. En base a este tipo de dato construya el T.D.A. Refranes que almacena el conjunto de refranes . En una primera aproximación el T.D.A Refranes usará un Arbol General para almacenar los refranes.
2. Definir el resto de T.DA. que vea necesario para la solución de los problemas propuestos.
3. Probar los módulos con programas test.
4. Construir los programas que a continuación se detallan.

Se puede usar la STL en todos los módulos excepto en la implementación del TDA ArbolGeneral.

A continuación se detallan los programas que se deberán desarrollar.

3.1.1. Test refranes

Este programa permitirá comprobar el buen funcionamiento del T.D.A Refranes que tiene en su representación un ArbolGeneral. Para ello el código fuente "test_refranes.cpp" deberá funcionar con los T.D.A ArbolGeneral y Refranes (ver sección 4) desarrollados.

```
1.  #include <iostream>
2.  #include <fstream>
3.  #include <cstdlib>
4.  #include <string>
5.  #include <ctype.h>
6.  #include "refranes.h"
7.  using namespace std;
8.
9.  string TodoMinuscula(const string &cad){
10.   string caux;
11.   for (unsigned char i=0;i<cad.size();++i)
12.      caux.push_back(tolower(cad[i]));
13.   return caux;
14. }
15.
16. int main(int argc, char * argv[]){
17.  if (argc!=3 && argc!=2){
18.   cout<<"Los parametros son:"<<endl;
19.   cout<<"1.Dime el nombre del fichero
20.         los refranes"<<endl;
21.   cout<<"2.-[Opcional] Prefijo de los
22.         refranes"<<endl;
23.    return 0;
24.  }
25.
26.  ifstream fin(argv[1]);
27.  if (!fin){
28.   cout<<"No puedo abrir el fichero
29.         "<<argv[1]<<endl;
30.   return 0;
31.  }
32.  int len=3;
33.  if (argc==3)
34.    len=atoi(argv[2]);
35.  Refranes refs(len);
36.
37. cout<<"Creado los refranes"<<endl;
38. fin>>refs;
39. cout<<"Refranes leidos :"<<endl;
40. cout<<refs<<endl;
41.
42. cout<<"Dime un Refran: ";
43. string refran;
44. getline(cin,refran);
45. string rr=TodoMinuscula(refran);
46. pair<bool, Refranes::iterator> a
    =refs.Esta(rr);
47. if (a.first)
48.  cout<<"El refrán "<<refran<<" si
49.         esta"<<endl;
50. else
51.  cout<<"No esta"<<endl;
52. //Implementar la busqueda de una
53. //subcadena en todos los refranes.
54. string patron;
55. cout<<"Dime un secuencia a buscar en los
56.        refranes:";
    getline(cin,patron);
57. patron=TodoMinuscula(patron);
58. //Recorremos los refranes buscando
59. //si contiene el patron
60. Refranes::iterator it;
61.   cout<<endl;
62.  for (it=refs.begin(); it!=refs.end();++it)
    {
63.    size_t found = (*it).find(patron);
64.    if (found!=string::npos)//si lo ha
65.                              //encontrado
66.         cout<<*it<<endl;
67.   }
68.
69. }
```

En este código, se carga el conjunto de refranes en memoria y luego se imprime por la salida estándar. A continuación se busca en el conjunto un refrán dado por el usuario y se indica si está en el conjunto o no. Finalmente dada una subcadena (variable *patrón*) por el usuario, el programa muestra todos los refranes que contiene esa subcadena. El alumno creará por lo tanto el programa test_refranes, que se deberá ejecutar en la línea de órdenes de la siguiente manera:

```
prompt%  test_refranes refranes_español.txt
```

El único parámetro que se da al programa es el nombre del fichero donde se almacena todos los refranes.

3.1.2. Adivina

Este programa dado un refrán con letras ocultas pide al usuario que vaya introduciendo letras para completar el refrán. El usuario podrá, si cree saber el refrán, introducir todo el refrán solución. En la línea de órdenes una posible llamada a este programa es:

```
(1)    prompt%  adivina  refranes_español.txt
o
(2)    prompt%  adivina  refranes_español.txt 10
```

Los parámetros de entrada son los siguientes:

1. El nombre del fichero con los refranes
2. Opcionalmente. Porcentaje de letras que se dejan sin ocultar. Si este parámetro no se da por defecto se toma 50 (50% de letras que se dejan sin ocultar de forma aleatoria)

Un ejemplo de ejecución es el siguiente:

```
****************Comienza el juego*****************

 _ a   e _ p _ _ i e n c i a   _ _   _ _   _ a _ r _   _ _   _ _   s a _ i _ _ _ _ _

Pulsa 0 para obtener ayuda
Dime un caracter (o el refran para resolver): l
Encontradas :3

l a   e _ p _ _ i e n c i a   _ _   l _   _ a _ r _   _ _   l _   s a _ i _ _ _ _ _

Pulsa 0 para obtener ayuda
Dime un caracter (o el refran para resolver): 0
Descubiertas :1

l a   e x p _ _ i e n c i a   _ _   l _   _ a _ r _   _ _   l _   s a _ i _ _ _ _ _

Pulsa 0 para obtener ayuda
Dime un caracter (o el refran para resolver): la experiencia es la madre de la sabiduria
La has adivinado:l a  e x p e r i e n c i a  e s  l a  m a d r e  d e  l a  s a b i d u r i a
¿Quieres seguir jugando[S/N]?N
```

En primer lugar el programa escoge uno de los refranes cargados en memoria y deja tantas letras sin ocultar según el porcentaje dado como segundo parámetro. Si este nos se introduce se tomará como el 50%. Estas letras se escogen, de forma aleatoria. A continuación el usuario podrá introducir una letra que cree que se encuentra en el refrán. Si es así, la máquina visualizará en el refrán estas letras, además indicando el numero de letras destapadas. En nuestro ejemplo en la primera iteración el usuario ha dado el carácter *l* y se han destapado 3 letras con valor *l*. En la segunda iteración el usuario inserta un valor de 0 para pedir ayuda a la máquina y esta le descubre algunas letras. En el ejemplo la letra *x* . Con esta ayuda el usuario se ve capaz de resolver e inserta todo el refrán: *la experiencia es la madre de la sabiduria*. A continuación la máquina le pregunta al usuario si desea volver a jugar. En caso afirmativo buscará un nuevo refrán y se aplicará los mismo pasos.

En la sección 3.2 se detallan los formatos de los ficheros de entrada al programa.

3.1.3. Estudio Prefijo

El programa estudio_prefijo muestra información acerca de la longitud del prefijo usado del refrán para construir el ArbolGeneral (ver el módulo ArbolGeneral). El programa se deberá ejecutar de la siguiente manera

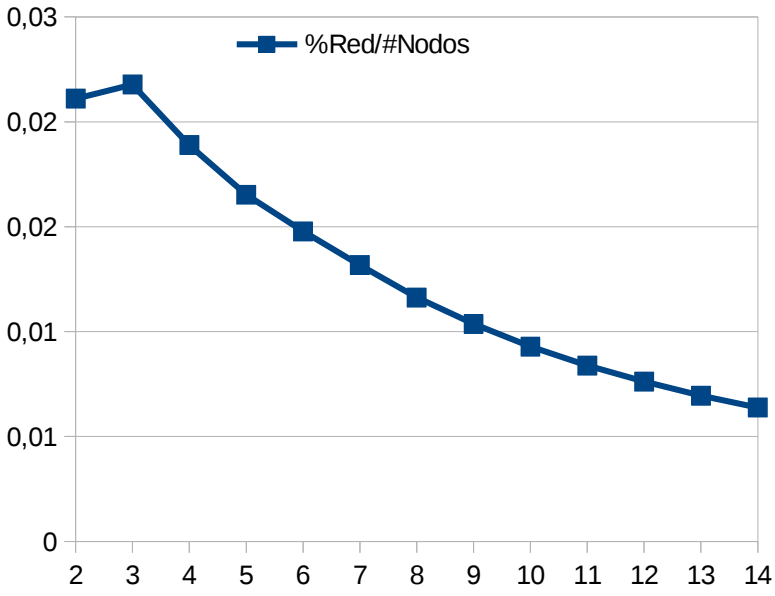
```
prompt%  estudio_prefijo refranes_español.txt estudio_esp.csv
```

Los parámetros de entrada son los siguientes:

1. El nombre del fichero con los refranes
2. El nombre del fichero de salida (se escribirá en formato csv[1])

Este programa para distintas longitudes de prefijo analiza: el número de caracteres totales que tienen todos los refranes, los caracteres almacenados en el árbol, porcentajes de reducción de caracteres en el árbol frente al total, y la razón entre este porcentaje y el numero de nodos en el árbol. Así la anterior ejecución genera un fichero "estudio_esp.csv" con la siguiente información

Prefijo	Car. Total	Car. Arb	%Red	# Nodos	%Red/#Nodos
2	5145	4904	4,68	222	0,021
3	5145	4819	6,34	291	0,022
4	5145	4765	7,39	391	0,019
5	5145	4720	8,26	500	0,017
6	5145	4679	9,06	613	0,015
7	5145	4647	9,68	735	0,013
8	5145	4626	10,09	868	0,012
9	5145	4609	10,42	1005	0,010
10	5145	4597	10,65	1147	0,009
11	5145	4588	10,82	1292	0,008
12	5145	4581	10,96	1439	0,008
13	5145	4577	11,04	1589	0,007
14	5145	4574	11,10	1740	0,006



En este ejemplo se puede observar

que la mejor longitud de prefijo para construir nuestro árbol es 3 ya que tiene la mayor razón reducción de caracteres por numero de nodos, con un valor de 0.022.

3.1.4. Refranes como una tabla hash

En esta sección se vuelve a pedir al alumno que implemente el programa adivina pero ahora refranes cambiará de representación como una tabla hash (implementada como un unordered_set) como se detalla en la sección 4.3. Por lo tanto el alumno creará un programa ***adivina_hash*** que hace lo mismo que adivina pero que refranes estará representado de forma diferente.

3.2. Ficheros

3.2.1. Fichero Refranes

El fichero refranes se componen de un conjunto de frases o refranes de un determinado idioma, cada uno en un línea. Este conjunto de frases serán los refranes que se consideren como válidos.

1 *El formato csv separa los campos con el carácter ;*

Un ejemplo de fichero refranes es el siguiente:

```
a buen entendedor pocas palabras bastan
a caballo regalado no le mires los dientes
a lo hecho pecho
a quien cuida la peseta nunca le falta un duro
a quien madruga dios lo ayuda
a rio revuelto ganancia de pescadores
borron y cuenta nueva
cuando el gato duerme bailan los ratones
cuando el rio suena agua lleva
cuando hay hambre no hay pan duro
...
```

En el material que se le da al alumno/a tiene a su disposición refranes o frases hechas en español, francés e inglés (ficheros refranes_español.txt, refranes_frances.txt, refranes_ingles.txt, respectivamente). Para facilitar todo el proceso los refranes no contienen mayúsculas, caracteres que ocupan más de un byte y símbolos de puntuación.

3.2.2. Fichero csv

Los archivos CSV (en inglés comma-separated values) es un tipo de fichero en formato abierto que nos sirve para representar datos en forma de tabla, en las que las columnas se separa o bien por comas o punto y coma y las filas se separan por salto de línea.

4. Módulos a desarrollar

4.1. Módulo Árbol General

Para crear los programas propuestos hará falta desarrollar el T.D.A ArbolGeneral. Este módulo debe ser lo suficiente flexible para poder desarrollar los programas propuestos. La interfaz y representación propuesta para ArbolGeneral la podéis encontrar en el fichero ArbolGeneral.h. El alumno implementará los métodos propuestos de acuerdo a la especificación dada. El T.D.A ArbolGeneral implementa un árbol en el que cada nodo puede tener un numero indeterminado de hijos. La información que almacena un nodo es:

- El elemento de tipo Tbase (plantilla)
- Un puntero al hijo más a la izquierda
- Un puntero al padre
- Un puntero al hermano a la derecha.

Con esta definición de nodo el T.D.A ArbolGeneral se puede representar como un puntero al nodo raíz. Por ejemplo un objeto de tipo ArbolGeneral, instanciando cada elemento a string, es el que se muestra en la Figura 1.

En el ejemplo de la figura 1 se ha usado un prefijo de longitud 3, esto significa que el árbol tiene 5 niveles (contando la raíz): 3 para los caracteres comunes (prefijo) ,1 para la raíz (que contiene la cadena vacía) y 1 para las hojas.

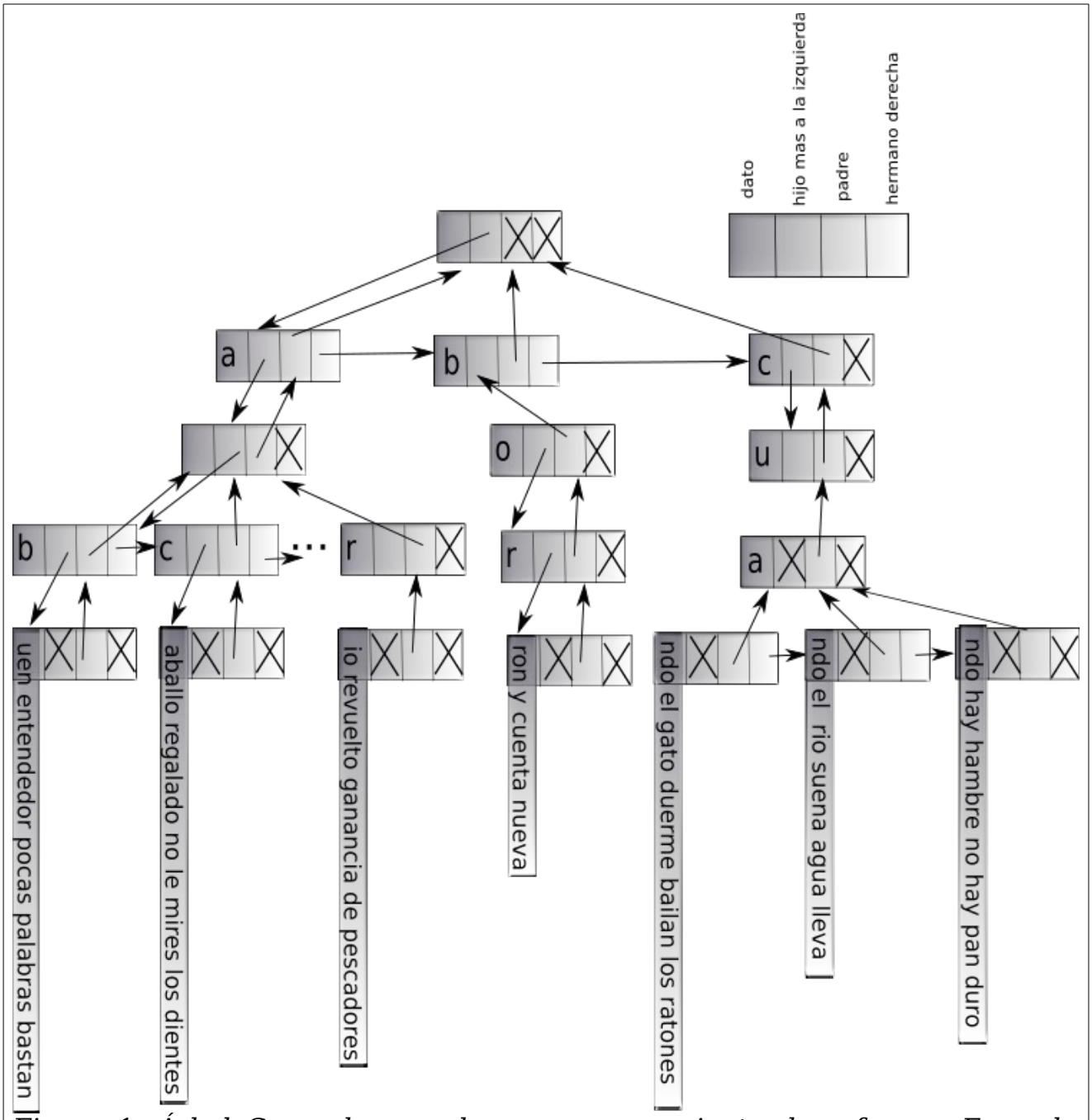


Figura 1: Árbol General para almacenar un conjunto de refranes. En cada nivel tenemos una letra del refrán. Los refranes se terminan en los nodos hojas. El numero de niveles del árbol general es una constante, en este caso es 5, ya que el prefijo de los refranes se ha tomado como 3.

En la figura 1 se puede observar que el refrán situado más a la derecha es “cuando hay hambre no hay pan duro”. Otros refranes que comparte el prefijo “cua” son “cuando el gato duerme bailan los ratones” o “cuando el rio suena agua lleva”. Hay que tener en cuenta que si el nodo no es hoja el nodo almacena un string que contiene solamente un carácter. Si el nodo es hoja el string tendrá un número de caracteres de 1 o más, estos caracteres son el sufijo del refrán.

Con respecto a las operaciones que se detallan, cabe destacar que se ha implementado un iterador, *iter_preorden,* para poder iterar por los nodos del ArbolGeneral siguiendo el recorrido preorden.

```
#ifndef __ArbolGeneral_h__
#define __ArbolGeneral_h__
template <class Tbase>
class ArbolGeneral{
private:
….
public:
    ...
   class iter_preorden{
       private:
        Nodo it;
        Nodo raiz;
        int level; //altura del nodo it dentro del arbol  con raiz “raiz”
       public:
        iter_preorden();

        Tbase & operator*();

        int getlevel();

       Nodo GetNodo ();

        bool Hoja()

        iter_preorden & operator ++();

        bool operator == (const iter_preorden &i);

        bool operator != (const iter_preorden &i);

        friend class ArbolGeneral;
    };


     iter_preorden begin();

     iter_preorden end();

};
#endif
```

El método begin inicializa un iter_preorden para que apunte a la raiz (el primer nodo en recorrido preorden). La función end inicializa un iter_preorden para que apunte al nodo nulo.

El campo level del iter_preorden indica el nivel por el que va el iterador. Por convenio level se inicializa a -1 en begin (que apunta al nodo raiz).

4.2. Módulo Refranes

También será necesario construir el T.D.A Refranes. Un objeto del T.D.A. Refranes representa a dichos o frases típicas de un lenguaje. El T.D.A Refranes será representado como un ArbolGeneral instanciado a *string*. Otros dos campos necesarios en la representación del

T.D.A. Refranes es la longitud del prefijo, que determinará el número de niveles del árbol general y también almacenaremos el número de refranes. Otros campo de interés será el número total de caracteres almacenadas (la suma de los caracteres en todos los refranes). Así en líneas generales el módulo Refranes se detalla a continuación:

```
#ifndef __REFRANES__H
#define __REFRANES__H
#include "ArbolGeneral.h"
#include <fstream>
#include <iostream>
using namespace std;
class Refranes{
  private:
    ArbolGeneral<string> ab;//para almacenar los refranes
    int len_prefijo;//la longitud del prefijo para construir el arbol
    int n_ref;//numero de refranes
    int caracteres_totales; //total de caracteres
  public:
    /**@brief Constructor por defecto. Inicia len_prefijo a 3
     * */
    Refranes();

    /**@brief Constructor por defecto. Inicia len_prefijo a  un valor
     **/

    Refranes(int lpre);

    /** @brief devuelve el numero de refranes almacenados
     */
    int size()const;

    /** @brief Inserta un refrán en el conjunto
     * @param r: refrán a insertar
     * @note el numero de refranes se incrementa en uno mas
     */
    void Insertar(const string & refran);

    /** @brief Borra un refrán en el conjunto
     * @param r: refran a borrar
     * @note el numero de refranes se decrementa en uno
     */
    void BorrarRefran(const string &refran);

    class iterator;
    class const_iterator; //declaración adelantada de los iteradores de refranes

    /** @brief Devuelve si un refran esta en el conjunto. Si esta devuelve un iterador a el
     * @param r: refran a buscar
     * @return una pareja que contiene si el refran esta y en caso afirmativo un iterador a el
    */
    pair<bool, iterator> Esta(const string &refran);
```

```
    /** @brief Elimina todos los refranes
     */
    void clear();

    /** @brief Lectura/Escritura de un conjunto de refranes
     */
    friend istream & operator>>(istream &is, Refranes &R);
    friend ostream & operator<<(ostream &is,const Refranes &R);

    /**@brief Devuelve el numero total de los caracteres en todos los refranes
     */
    int Caracteres_Refranes()const{ return caracteres_totales;}

    /**@brief Devuelve el numero de caracteres usados
     * @note que no tiene que coincidir con el numero de caracteres de todos los refranes
           almacenados
     */
    int Caracteres_Almacenados();

    /**@brief Numero de nodos necesarios para la configuración
     */
    int Numero_Nodos()const{ return ab.size();}

    /***************************************/
    class iterator{
    private:
      ArbolGeneral<string>::iter_preorden it;
      string cad; //caracteres hasta el iterador
    public:
      iterator():
      string & operator *();
      bool operator==(const iterator &i)const;
      bool operator!=(const iterator &i)const;
      iterator &operator ++()
      friend class Refranes;
      friend class const_iterator;
    };


    class const_iterator{
     ….
    };

    /***************BEGIN y END*************/
    iterator begin();
    iterator end();

    const_iterator begin()const;
    const_iterator end()const;
};
#endif
```

De igual forma que lo hicimos en la clase ArbolGeneral en el T.D.A Refranes hemos definido un iterador (y su versión constante) que nos permite recorrer de manera ordenada todos los refranes. Así si un iterador de tipo Refranes, en la Figura 1 se encuentra apuntado al refrán "a buen entendedor pocas palabras bastan" cuando se aplica sobre este el operador ++ saltará al refrán "a caballo regalado no le mires los dientes". Para ello el operador ++ tendrá que hacer avanzar a su el iterador de ArbolGeneral, que tiene en su representación, y en el campo *cad* id poniendo los caracteres por los que pasa hasta llegar a un nodo hoja.

4.3. Módulo Refranes_hash

En este modulo de nuevo vamos a definir el T.D.A Refranes pero ahora usando una tabla hash. En particular usaremos para la representación en vez de un ArbolGeneral un objeto de tipo unordered_set de la STL, instanciado a string. Los métodos necesarios serán los mismos definidos en el módulo Refranes simplemente cambiaremos la representación.

```
#ifndef __REFRANES__H
#define __REFRANES__H
#include <unordered_set>
#include <fstream>
#include <iostream>
using namespace std;
class Refranes{
  private:
    class my_hash{
     private:
      unsigned int len; //caracteres sobre los que calcular la función hash
     public:
         my_hash(int l=3);
         void set_len(int l);
         //el operador () calcula la función hash sobre los len caracteres de clave.
         size_t operator()(const string & clave) const;
    } ;

    unordered_set<string,my_hash> ab;//almacena los refranes
    int len_efijo;//la longitud del prefijo para definir la función hash
    int n_ref;
    int caracteres_totales;

   …
};
#endif
```

Como se puede observar la definición

```
    unordered_set<string,my_hash> ab;
```

define una tabla hash cuya función hash se obtendrá usando el operador () de la clase my_hash. Los iteradores de Refranes se deberán representar en base a un iterador de unordered_set.

Sin cambiar nada en adivina.cpp (programa principal del programa adivina), Refranes_hash sustituirá al modulo Refranes. Por lo tanto la interfaz de Refranes y Refranes_hash es la misma.

Por otro lado la función hash que se implementará en el operador () de la clase my_hash tiene que ser una función rápida de calcular y que no genere muchas colisiones. Una idea que aquí se le da al alumno es la siguiente. Suponed :

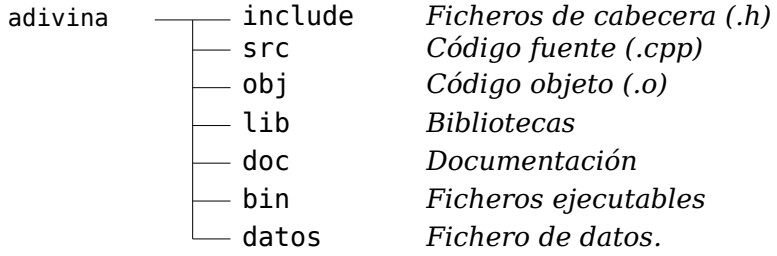
- Nuestros refranes cuentan con 28 caracteres diferentes
- Vamos a calcular sobre la cadena "$a_1a_2a_3a_4a_5$" nuestra función hash suponiendo que para calcularla solamente tenemos en cuenta los tres primero caracteres. Una forma de hacer esto podría ser:

$$hash\_valor = a_1 * 28^2 + a_2 * 28^1 + a_3 \ 28^0$$

5. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "adivina*.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 "*adivina*") para ejecutar:

```
prompt% tar zcv adivina.tgz adivina
```

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

Referencias

- http://es.wikipedia.org/wiki/CSV