Glossygloss

0.2

Generated by Doxygen 1.8.6

Fri Apr 25 2014 22:33:11

Contents

Chapter 1

Main Page

Glossygloss is set of classes to use several data structures, C++ containers. More might come soon.

Documentation

All documented things are `here`.

A PDF file *refman.pdf* is also available for offline doc.

You can generate the doc using `doxygen` and the config file *doxygen_config*

Usefull links :

- `C++ programming on wikibooks`
- what else ?

Compilation

Here is a sort intance showing how to compile and 'use' hashtable.hpp. It works as well for others files.

We use C++11, so to compile using our classes:

$ g++ -std=c++0x -Wall -pedantic -o sample_hashtable.bin sample_hashtable.cpp

Testing and usage

Once you compiled sample_hashtable.cpp, you can run the code using, assuming you are using an UNIX system.

$ chmod +x sample_hashtable.bin

First give execution permission to the compiled code.

$ ./sample_hashtable.bin lorem quod 50

And then, run the program. Words in the first file (lorem) will be maped to the words in quod. The last argument stands for the words number you want to put in the hashtable.

Copyright

Chapter 2

Todo List

File hashtable.hpp

: removing the last element of a alveoles chain makes trouble (seg fault)

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Class Documentation

6.1 Alveole< K, V > Class Template Reference

Class to define Hashtable alveoles.

`#include <hashtable.hpp>`

Public Member Functions

- Alveole (const Alveole< K, V > &other)
- Alveole (K key, V value)
- Alveole ()
- Alveole (K key, V value, Alveole< K, V > ∗next)
- ∼Alveole ()
- K getKey ()
- V getValue ()
- Alveole< K, V > ∗ getNext ()
- void setValue (V n_value)
- void setNext (Alveole< K, V > ∗n_next)
- string toString ()

Private Attributes

- K _key
- V _value
- Alveole< K, V > ∗ _next

6.1.1 Detailed Description

template<typename K, typename V>class Alveole< K, V >

Class to define Hashtable alveoles.

Alveole class embodies a Hashtable's alveole. An alveole store a pair <k,v>. Alveoles are simply-linked elements.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 template<typename K, typename V> Alveole< K, V >::Alveole (const Alveole< K, V > & *other*)

next aveole Copy constructor

Parameters

`in`	*other*	the alveole to copy

6.1.2.2 template<typename K, typename V> Alveole< K, V >::Alveole (K *key*, V *value*)

Pair constructor

Parameters

`in`	*key*	key of the pair
`in`	*value*	value of the pair

6.1.2.3 template<typename K, typename V> Alveole< K, V >::Alveole ( )

Empty constructor create an 'empty' alveole

6.1.2.4 template<typename K, typename V> Alveole< K, V >::Alveole (K *key*, V *value*, Alveole< K, V > * *next*)

Complex constructor

Parameters

`in`	*key*	key of the pair
`in`	*value*	value of the pair
`in`	*next*	adresse to the next alveole

6.1.2.5 template<typename K, typename V> Alveole< K, V >::~Alveole ( )

Destructor for Alveole

6.1.3 Member Function Documentation

6.1.3.1 template<typename K, typename V> K Alveole< K, V >::getKey ( )

Get the key of an alveole

Parameters

`out`	*key*	key stored into the alveole

6.1.3.2 template<typename K, typename V> Alveole<K,V>* Alveole< K, V >::getNext ( )

Which alveole coming next ?

Parameters

`out`	*ptr*	memory adress of the next alveole

6.1.3.3 template<typename K, typename V> V Alveole< K, V >::getValue ( )

Get the value stored into an alveole

Parameters

out	*value*	value of the alveole

6.1.3.4 template<typename K, typename V> void Alveole< K, V >::setNext (Alveole< K, V > * *n_next*)

Set the next adress of the next alveole

Parameters

in	*n_next*	adress of the new next alveole

6.1.3.5 template<typename K, typename V> void Alveole< K, V >::setValue (V *n_value*)

Set the value stored into an alveole

Parameters

in	*n_value*	The new value of the pair

6.1.3.6 template<typename K, typename V> string Alveole< K, V >::toString ( )

Return a string descriptionof the pair stored into the alveole

Parameters

out	*desc*	a string represention of the alveole

6.1.4 Member Data Documentation

6.1.4.1 template<typename K, typename V> K Alveole< K, V >::_key `[private]`

6.1.4.2 template<typename K, typename V> Alveole<K,V>* Alveole< K, V >::_next `[private]`

value of the pair

6.1.4.3 template<typename K, typename V> V Alveole< K, V >::_value `[private]`

key of the pair

The documentation for this class was generated from the following file:

- src/hashtable.hpp

6.2 Dictionnaire Class Reference

`#include <dictionnaire_arbre.hpp>`

Public Member Functions

- Dictionnaire ()
- ∼Dictionnaire ()
- bool contientMot (string mot)

- void ajouterMot (string mot)
- void associerMot (string mot)
- int valeurAssociee (string mot)
- void plusFrequentes (pair$<$ string, int $>$ $*$frequences)
- Dictionnaire ()
- $\sim$Dictionnaire ()
- bool contientMot (string mot)
- void ajouterMot (string mot)
- void associerMot (string mot)
- int valeurAssociee (string mot)
- void plusFrequentes (pair$<$ string, int $>$ $*$frequences)

Private Attributes

- TreeString dico
- Hashtable$<$ string, int $>$ dico

6.2.1 Constructor & Destructor Documentation

6.2.1.1 Dictionnaire::Dictionnaire ( )

Constructeur de la classe Dictionnaire

6.2.1.2 Dictionnaire::$\sim$Dictionnaire ( )

Destructeur de la classe Dictionnaire

6.2.1.3 Dictionnaire::Dictionnaire ( )

Constructeur de la classe Dictionnaire

6.2.1.4 Dictionnaire::$\sim$Dictionnaire ( )

Destructeur de la classe Dictionnaire

6.2.2 Member Function Documentation

6.2.2.1 void Dictionnaire::ajouterMot (string *mot*)

Fonction qui ajoute un mot non présent dans le dictionnaire

Parameters

in	*mot*	le mot à ajouter

6.2.2.2 void Dictionnaire::ajouterMot (string *mot*)

Fonction qui ajoute un mot non présent dans le dictionnaire

Parameters

`in`	*mot*	le mot à ajouter

6.2.2.3 void Dictionnaire::associerMot (string *mot*)

Fonction qui modifie la valeur d'un mot présent dans le dictionnaire

Parameters

`in`	*mot*	le mot à modifier
`out`	*bool*	Renvoyer faux si le mot n'est pas présent, sinon vrai

6.2.2.4 void Dictionnaire::associerMot (string *mot*)

Fonction qui modifie la valeur d'un mot présent dans le dictionnaire

Parameters

`in`	*mot*	le mot à modifier
`out`	*bool*	Renvoyer faux si le mot n'est pas présent, sinon vrai

6.2.2.5 bool Dictionnaire::contientMot (string *mot*)

Fonction qui renvoie vrai le mot est présent dans le Dictionnaire

Parameters

`in`	*mot*	le mot à tester
`out`	*bool*	vrai si présent, faux sinon.

6.2.2.6 bool Dictionnaire::contientMot (string *mot*)

Fonction qui renvoie vrai le mot est présent dans le Dictionnaire

Parameters

`in`	*mot*	le mot à tester
`out`	*bool*	vrai si présent, faux sinon.

6.2.2.7 void Dictionnaire::plusFrequentes (pair$<$ string, int $>$ $*$ *frequences*)

Fonction qui retourne les dix mots les plus fréquents dans un tableau

Parameters

int]	frequences tableau des paires$<$mots,occurences$>$ les plus fréquents

6.2.2.8 void Dictionnaire::plusFrequentes (pair$<$ string, int $>$ $*$ *frequences*)

Fonction qui retourne les dix mots les plus fréquents dans un tableau

Parameters

int]	frequences tableau des paires<mots,occurences> les plus fréquents

6.2.2.9 int Dictionnaire::valeurAssociee (string *mot*)

Fonction qui récupère la valeur associée au mot

Parameters

`in`	*mot*	le mot dont on souhaite savoir la valeur associée
`out`	*valeur*	la valeur associée, 0 peut indiquer l'absence du mot

6.2.2.10 int Dictionnaire::valeurAssociee (string *mot*)

Fonction qui récupère la valeur associée au mot

Parameters

`in`	*mot*	le mot dont on souhaite savoir la valeur associée
`out`	*valeur*	la valeur associée

Exceptions

lève	une exception si le mot n'est pas présent dans le dictionnaire

6.2.3 Member Data Documentation

6.2.3.1 TreeString Dictionnaire::dico `[private]`

6.2.3.2 Hashtable<string,int> Dictionnaire::dico `[private]`

stockage des mots dans une table de hashage

The documentation for this class was generated from the following files:

- src/dictionnaire_arbre.hpp
- src/dictionnaire_hash.hpp

6.3 Hashtable< K, V > Class Template Reference

Maps a key to a value.

`#include <hashtable.hpp>`

Public Member Functions

- Hashtable ()
- ∼Hashtable ()
- bool contains (const K &key)
- V get (const K &key)
- bool isEmpty ()
- void put (K key, V value)
- void remove (const K &key)

FIXME : remove last element of a list lead to a seg. fault.

- string toString ()
- void getPairs (forward_list< pair< string, int >> &pairs)

Private Attributes

- Alveole< K, V > ** _table

6.3.1 Detailed Description

template<typename K, typename V>class Hashtable< K, V >

Maps a key to a value.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 template<typename K, typename V> Hashtable< K, V >::Hashtable ( )

array of alveoles Simple constructor

6.3.2.2 template<typename K, typename V> Hashtable< K, V >::~Hashtable ( )

Destructor

6.3.3 Member Function Documentation

6.3.3.1 template<typename K, typename V> bool Hashtable< K, V >::contains (const K & *key*)

Do table contains key ?

Parameters

in	*key*	key to find
out	*bool*	True if the key is here, else false

6.3.3.2 template<typename K, typename V> V Hashtable< K, V >::get (const K & *key*)

Return the value maped to the specified key

Parameters

in	*key*	a key in the hashtable
out	*value*	value associated with the key

Exceptions

HashtableException	threw if key is not in the hashtable

6.3.3.3 template<typename K, typename V> void Hashtable< K, V >::getPairs (forward_list< pair< string, int >> & *pairs*)

Get a list of all kay and their value in pairs

Parameters

in	*pairs*	Vector which contains keys to find

6.3.3.4 template<typename K, typename V> bool Hashtable< K, V >::isEmpty ( )

Tests if this hashtable maps no keys to values.

Parameters

out	*bool*	true if no elements in the hashtable, else false;

6.3.3.5 template<typename K, typename V> void Hashtable< K, V >::put (K *key*, V *value*)

Map the specified key to the specified value in this hashtable. or update the maped value to the key

Parameters

in	*key*	key of the pair
in	*value*	value of the pair

6.3.3.6 template<typename K, typename V> void Hashtable< K, V >::remove (const K & *key*)

FIXME : remove last element of a list lead to a seg. fault.

Remove the key (and its corresponding value) from this hashtable.

Parameters

in	*key*	Key of the pair to delete

Exceptions

HashtableException	threw if table does not contain key

6.3.3.7 template<typename K, typename V> string Hashtable< K, V >::toString ( )

Return a description of the hashtable, enclosed in braces as well as {key, value}.

Parameters

out	*desc*	a string representation of this hashtable.

6.3.4 Member Data Documentation

6.3.4.1 template<typename K, typename V> Alveole<K,V>** Hashtable< K, V >::_table [private]

The documentation for this class was generated from the following file:

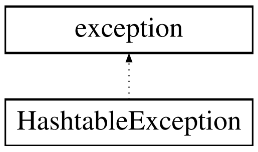
- src/hashtable.hpp

6.4 HashtableException Class Reference

Exception class to manage Hashtable errors.

```
#include <hashtable.hpp>
```

Inheritance diagram for HashtableException:



Public Member Functions

- HashtableException (const char ∗cause)
- virtual ∼HashtableException () throw ()
- virtual const char ∗ what () const throw ()

Private Attributes

- const char ∗ _cause

6.4.1 Detailed Description

Exception class to manage Hashtable errors.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 HashtableException::HashtableException (const char ∗ *cause*)

store exception description constructor called then HashtableExceptions are threw

Parameters

`in`	*cause*	description of exception origin

6.4.2.2 virtual HashtableException::∼HashtableException ( ) throw) `[virtual]`

destructor currently, do anything special

6.4.3 Member Function Documentation

6.4.3.1 virtual const char∗ HashtableException::what ( ) const throw) `[virtual]`

virtual fonction from superclass, usefull to get the exception description

6.4.4 Member Data Documentation

6.4.4.1 const char∗ HashtableException::_cause `[private]`

The documentation for this class was generated from the following file:

- src/hashtable.hpp

6.5 Node< T > Class Template Reference

Defines tree nodes.

```
#include <tree.hpp>
```

Public Member Functions

- Node (const Node< T > &other)
- Node (T data)
- ~Node ()
- Node< T > & operator= (Node< T > &other)
- bool operator== (const Node< T > &rhs)
- bool operator!= (const Node< T > &rhs)
- bool isLeaf ()
- int height ()
- void append (T n_data)
- void remove (T data)
- T getTag ()
- bool contains (T element)
- string toString ()
- Node (const Node &other)
- Node (char data, int frequency)
- Node ()
- ~Node ()
- Node & operator= (const Node &other)
- bool operator== (const Node &rhs)
- bool operator!= (const Node &rhs)
- bool isLeaf ()
- int height ()
- Node ∗ append (const char n_data, int frequency)
- char getTag ()
- string toString ()
- void toList (forward_list< string > &words, string word)
- void toFrequencedList (forward_list< pair< string, int >> &words, string word)

Private Attributes

- int _childNbr

  Number of children.
- T _tag

  letter stored into Node, the tag
- forward_list< Node< T > > _children

  children of the Node
- int _wordFrequency
- char _tag

  letter stored into Node, the tag
- forward_list< Node ∗ > _children

  children of the Node

6.5.1 Detailed Description

template<typename T = char>class Node< T >

Defines tree nodes.

Class for nodes of a tree. A Node store a tag and can have several children

Class for nodes of a TreeString. A Node store a letter and can have several children.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 template<typename T = char> Node< T >::Node (const Node< T > & *other*)

Copy constructor

Parameters

`in`	*other*	Node to copy

6.5.2.2 template<typename T = char> Node< T >::Node (T *data*)

Simple constructor

Parameters

`in`	*data*	to store into the Node

6.5.2.3 template<typename T = char> Node< T >::∼Node ( )

Destructor for Node

6.5.2.4 template<typename T = char> Node< T >::Node (const Node< T > & *other*)

Copy constructor

Parameters

`in`	*other*	Node to copy

6.5.2.5 template<typename T = char> Node< T >::Node (char *data*, int *frequency*)

Simple constructor

Parameters

`in`	*data*	to store into the Node
`in`	*end*	is it the last letter of a word ?

6.5.2.6 template<typename T = char> Node< T >::Node ( )

Empty constructor

6.5.2.7 template<typename T = char> Node< T >::~Node ( )

Destructor for Node

6.5.3 Member Function Documentation

6.5.3.1 template<typename T = char> void Node< T >::append (T *n_data*)

Hook up a new child to the node

Parameters

in	*n_data*	new data to store as a child of the node

6.5.3.2 template<typename T = char> Node* Node< T >::append (const char *n_data*, int *frequency*)

Hook up a new child to the node

Parameters

in	*n_data*	new data to store as a child of the node
in	*frequency*	if greater than 0, end of a word.
out	*newchild*	return the adress of the new child created

6.5.3.3 template<typename T = char> bool Node< T >::contains (T *element*)

Do the tag is element or one of his children ?

Parameters

in	*element*	Element to look for
out	*bool*	True if node or one of his child has the right tag, else false.

6.5.3.4 template<typename T = char> T Node< T >::getTag ( )

What is the tag of the Node ?

Parameters

out	*tag*	The tag of the node

6.5.3.5 template<typename T = char> char Node< T >::getTag ( )

What is the tag of the Node ?

Parameters

out	*tag*	The tag of the node

6.5.3.6 template<typename T = char> int Node< T >::height ( )

The height of the node

Parameters

out	*hgt*	height of the node

6.5.3.7 template<typename T = char> int Node< T >::height ( )

The height of the node

Parameters

out	*hgt*	height of the node

6.5.3.8 template<typename T = char> bool Node< T >::isLeaf ( )

Is the node a leaf ?

Parameters

out	*bool*	true, if no child, else false

6.5.3.9 template<typename T = char> bool Node< T >::isLeaf ( )

Is the node a leaf ?

Parameters

out	*bool*	true, if no child, else false

6.5.3.10 template<typename T = char> bool Node< T >::operator!= (const Node< T > & *rhs*)

inequality operator

Parameters

in	*lhs*	first node to compare
in	*rhs*	second node to compare
out	*bool*	true if nodes have not the same memory adress, else false

6.5.3.11 template<typename T = char> bool Node< T >::operator!= (const Node< T > & *rhs*)

inequality operator

Parameters

in	*lhs*	first node to compare
in	*rhs*	second node to compare
out	*bool*	true if nodes have not the same memory adress, else false

6.5.3.12 template<typename T = char> Node<T>& Node< T >::operator= (Node< T > & *other*)

assignment operator overload

Parameters

in	*other*	node to assign
out	*note*	assigned node

6.5.3.13 template<typename T = char> Node& Node< T >::operator= (const Node< T > & *other*)

assignment operator overload

Parameters

in	*other*	node to assign
out	*note*	assigned node

6.5.3.14 template<typename T = char> bool Node< T >::operator== (const Node< T > & *rhs*)

equality operator

Parameters

in	*lhs*	left hand side, first node to compare
in	*rhs*	right hand side, second node to compare
out	*bool*	true if nodes have the same memory adress, else false

6.5.3.15 template<typename T = char> bool Node< T >::operator== (const Node< T > & *rhs*)

equality operator

Parameters

in	*lhs*	left hand side, first node to compare
in	*rhs*	right hand side, second node to compare
out	*bool*	true if nodes have the same memory adress, else false

6.5.3.16 template<typename T = char> void Node< T >::remove (T *data*)

Remove a leaf from the node

Parameters

in	*data*	data of the node's tag to remove

Exceptions

TreeException	Threw if data is not removed

6.5.3.17 template<typename T = char> void Node< T >::toFrequencedList (forward_list< pair< string, int >> & *words*, string *word*)

Put each word in a list

Parameters

in	*words*	List containing all words and his frequency in pairs
in	*string*	wordCom Word which is currently reconvene

6.5.3.18 template<typename T = char> void Node< T >::toList (forward_list< string > & *words*, string *word*)

Put each words in a list

Parameters

in	*words*	List containing all words
in	*string*	wordCom Word which is currently reconvene

6.5.3.19 template<typename T = char> string Node< T >::toString ( )

Get a string representation of the node and his child

Parameters

out	*desc*	Description of the node (and his child)

6.5.3.20 template<typename T = char> string Node< T >::toString ( )

Get a string representation of the node and his child

Parameters

out	*desc*	Description of the node (and his child)

6.5.4 Member Data Documentation

6.5.4.1 template<typename T = char> int Node< T >::_childNbr [private]

Number of children.

6.5.4.2 template<typename T = char> forward_list<Node<T> > Node< T >::_children [private]

children of the Node

6.5.4.3 template<typename T = char> forward_list<Node*> Node< T >::_children [private]

children of the Node

6.5.4.4 template<typename T = char> T Node< T >::_tag [private]

letter stored into Node, the tag

6.5.4.5 template<typename T = char> char Node< T >::_tag [private]

letter stored into Node, the tag

6.5.4.6 template< typename T = char > int Node< T >::_wordFrequency `[private]`

the end of a word and his frequency if _wordFrequency > 0, it's a word end

The documentation for this class was generated from the following files:

- src/tree.hpp
- src/treestring.hpp

6.6 Tree< T > Class Template Reference

Tree is a recursive structure using nodes.

`#include <tree.hpp>`

Public Member Functions

- Tree ()
- Tree (const Tree< T > &other)
- Tree (T element)
- ~Tree ()
- bool contains (T element)
- int height ()
- void put (T element)
- void remove (T element)
- string toString ()

Private Attributes

- Node< T > _root

6.6.1 Detailed Description

template< typename T = string > class Tree< T >

Tree is a recursive structure using nodes.

A root value and subtrees of children, represented as a set of linked nodes.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 template< typename T = string > Tree< T >::Tree ( )

First node of the tree Default constructor

6.6.2.2 template< typename T = string > Tree< T >::Tree (const Tree< T > & *other*)

Copy constructor

6.6.2.3 template< typename T = string > Tree< T >::Tree (T *element*)

Common constructor, create an tree

Parameters

in	*element*	Root of the tree

6.6.2.4 template<typename T = string> Tree< T >::~Tree ( )

Destructor, destroy the whole tree

6.6.3 Member Function Documentation

6.6.3.1 template<typename T = string> bool Tree< T >::contains (T *element*)

Is the element in the tree ?

Parameters

in	*element*	Search the element in the Tree
out	*bool*	True if element is here, else false.

6.6.3.2 template<typename T = string> int Tree< T >::height ( )

The height of the tree

Parameters

out	*hgt*	Height of the tree

6.6.3.3 template<typename T = string> void Tree< T >::put (T *element*)

Put an element in the tree

Parameters

in	*element*	New element to put into the tree

6.6.3.4 template<typename T = string> void Tree< T >::remove (T *element*)

Remove an element from the tree

Parameters

in	*data*	Element to remove

6.6.3.5 template<typename T = string> string Tree< T >::toString ( )

Get a string representation of the Tree Each node tags is separated with a comma

Parameters

out	*desc*	String representation of the tree

6.6.4 Member Data Documentation

6.6.4.1 template<typename T = string> Node<T> Tree< T >::_root [private]

The documentation for this class was generated from the following file:

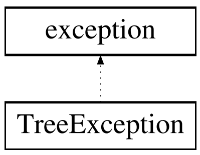
- src/tree.hpp

6.7 TreeException Class Reference

exception class for trees

```
#include <tree.hpp>
```

Inheritance diagram for TreeException:



Public Member Functions

- TreeException (char *cause)
- virtual ~TreeException () throw ()
- virtual const char * what () const throw ()

Private Attributes

- char * _cause

6.7.1 Detailed Description

exception class for trees

Usefull to manage errors and the unforeseen

6.7.2 Constructor & Destructor Documentation

6.7.2.1 TreeException::TreeException (char * *cause*)

store exception description constructor called then TreeExceptions are threw

Parameters

in	*cause*	description of exception origin

6.7.2.2 virtual TreeException::~TreeException ( ) throw) [virtual]

destructor currently, do anything special

6.7.3 Member Function Documentation

6.7.3.1 virtual const char∗ TreeException::what ( ) const throw) [virtual]

virtual fonction from superclass, usefull to get the exception description

6.7.4 Member Data Documentation

6.7.4.1 char∗ TreeException::_cause [private]

The documentation for this class was generated from the following file:

- src/tree.hpp

6.8 TreeString Class Reference

Tree is a recursive structure using nodes.

```
#include <treestring.hpp>
```

Public Member Functions

- TreeString ()
- TreeString (const TreeString &other)
- ∼TreeString ()
- int height ()
- void put (const string &word)
- string toString ()
- void getWords (forward_list< string > &list)
- void getWordsFrequencies (forward_list< pair< string, int >> &words)

Private Attributes

- Node _root

6.8.1 Detailed Description

Tree is a recursive structure using nodes.

A root value and subtrees of children, represented as a set of linked nodes.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 TreeString::TreeString ( )

First node of the tree Default constructor

6.8.2.2 TreeString::TreeString (const TreeString & *other*)

Copy constructor

6.8.2.3 TreeString::∼TreeString ( )

Destructor, destroy the whole tree

6.8.3 Member Function Documentation

6.8.3.1 void TreeString::getWords (forward_list< string > & *list*)

Put each word in a list The list must be initialized !

Parameters

`in`	*list*	List containing string for each word stored in Tree

6.8.3.2 void TreeString::getWordsFrequencies (forward_list< pair< string, int >> & *words*)

Get a list of all words stored in Tree and their frequencies, i.e. how times a word was added

Parameters

int]	list List of pair containing for each word in Tree his frequency

6.8.3.3 int TreeString::height ( )

The height of the tree

Parameters

`out`	*hgt*	Height of the tree

6.8.3.4 void TreeString::put (const string & *word*)

Put a word in the tree

Parameters

`in`	*word*	New element to put into the tree

6.8.3.5 string TreeString::toString ( )

Get a string representation of the Tree

Parameters

`out`	*desc*	A string reprensation of the Tree where each Node tag is separated by a comma

6.8.4 Member Data Documentation

6.8.4.1 Node TreeString::_root `[private]`

The documentation for this class was generated from the following file:

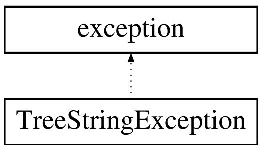
- src/treestring.hpp

6.9 TreeStringException Class Reference

exception class for trees

`#include <treestring.hpp>`

Inheritance diagram for TreeStringException:



Public Member Functions

- TreeStringException (char ∗cause)
- virtual ∼TreeStringException () throw ()
- virtual const char ∗ what () const throw ()

Private Attributes

- char ∗ _cause

6.9.1 Detailed Description

exception class for trees

Usefull to manage errors and the unforeseen

6.9.2 Constructor & Destructor Documentation

6.9.2.1 TreeStringException::TreeStringException (char ∗ *cause*)

store exception description constructor called then TreeExceptions are threw

Parameters

in	*cause*	description of exception origin

6.9.2.2 virtual TreeStringException::∼TreeStringException ( ) throw) [virtual]

destructor currently, do anything special

6.9.3 Member Function Documentation

6.9.3.1 virtual const char∗ TreeStringException::what ( ) const throw) [virtual]

virtual fonction from superclass, usefull to get the exception description

6.9.4 Member Data Documentation

6.9.4.1 char∗ TreeStringException::_cause [private]

The documentation for this class was generated from the following file:

- src/treestring.hpp

Chapter 7

File Documentation

7.1 README.md File Reference

7.2 src/application.cpp File Reference

```
#include <functional>
#include <iostream>
#include <fstream>
#include "dictionnaire_arbre.hpp"
```

Functions

- int main (int argc, const char **argv)

7.2.1 Detailed Description

7.2.2 File description

Programme permettant de lire des mots dans un fichier texte passé en paramètre et qui calcule leurs fréquence.

7.2.3 Copyright

7.2.4 File informations

$Date$ 2014/04/02 $Rev$ 0.2 $Author$ François Hallereau & Benjamin Sientzoff $URL$ `http://www.github.com/blasterbug/glossygloss`

7.2.5 Function Documentation

7.2.5.1 int main (int *argc*, const char ∗∗ *argv*)

7.3 src/dictionnaire_arbre.hpp File Reference

```
#include "treestring.hpp"
#include <utility>
```

Classes

- class Dictionnaire

Functions

- bool triPair (const pair< string, int > &first, const pair< string, int > &second)

7.3.1 Function Documentation

7.3.1.1 bool triPair (const pair< string, int > & *first*, const pair< string, int > & *second*)

Fonction qui permet de trier un container de pairs construit avec des strings et des entiers. Le critère de tri est l'ordre naturel sur les entiers appliqué à l'entier de la pair.

Parameters

in	*first*	la première pair à comparer
in	*second*	la seconde pair à comparer
out	*bool*	vrai si first>seconde sinon faux

7.4 src/dictionnaire_hash.hpp File Reference

```
#include "hashtable.hpp"
#include <utility>
```

Classes

- class Dictionnaire

Functions

- bool triPair (const pair< string, int > &first, const pair< string, int > &second)
- template<>
  unsigned computehash< string > (string element)

7.4.1 Detailed Description

7.4.2 File description

Dictionnaire utilisant une Hashtable

7.4.3 Copyright

7.4.4 File informations

$Date$ 2014/03/27 Rev 0.2 $Author$ Benjamin Sientzoff & François Hallereau URL `http://www.-github.com/blasterbug`

7.4.5 Function Documentation

7.4.5.1 template<> unsigned **computehash**< string > (string *element*)

Fonction pour calculer les clés de hashage de string

7.4.5.2 bool triPair (const pair< string, int > & *first,* const pair< string, int > & *second*)

Fonction qui permet de trier un container de pairs construit avec des strings et des entiers. Le critère de tri est l'ordre naturel sur les entiers appliqué à l'entier de la pair.

Parameters

`in`	*first*	la première pair à comparer
`in`	*second*	la seconde pair à comparer
`out`	*bool*	vrai si first>seconde sinon faux

7.5 src/hashtable.hpp File Reference

```
#include <string>
#include <cassert>
#include <utility>
#include <forward_list>
```

Classes

- class HashtableException

Exception class to manage Hashtable errors.

- class Alveole< K, V >

  Class to define Hashtable alveoles.
- class Hashtable< K, V >

  Maps a key to a value.

Macros

- #define END nullptr

  macro to define end of alveole chains
- #define ARRAYSIZE 25

  macro to define size of hash arrays
- #define NDEBUG

Functions

- template<typename K >
  unsigned computehash (K element)

7.5.1 Detailed Description

7.5.2 File description

data structure to store pairs in a table a hashcode is compute with k to evaluate the suitable place to store the pair

!! WARNING: int hashCode(K key) must be implemented !!

7.5.3 Copyright

This source code is protected by the French intellectual property law.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; version 2 of the License.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

7.5.4 File informations

$Date$ 2014/03/27 Rev 0.2 $Author$ Benjamin Sientzoff URL `http://www.github.com/blasterbug`

Todo : removing the last element of a alveoles chain makes trouble (seg fault)

7.5.5 Macro Definition Documentation

7.5.5.1 #define ARRAYSIZE 25

macro to define size of hash arrays

7.5.5.2 #define END nullptr

macro to define end of alveole chains

7.5.5.3 #define NDEBUG

7.5.6 Function Documentation

7.5.6.1 template<typename K > unsigned computehash (K *element*)

Fonction you must define when you're using Hashable An exemple is given in the sample file

Parameters

in	*element*	element to compute hashcode from
out	*hashcode*	the hashcode of element, an unsigned integer

template<> unisgned computehash<string>(string element)

your implementation of hashcode function

7.6 src/sample_hashtable.cpp File Reference

```
#include <functional>
#include <iostream>
#include <cstdlib>
#include <fstream>
#include "hashtable.hpp"
```

Macros

- #define K string
- #define V string

Functions

- template<>
  unsigned computehash< K > (K element)
- int main (int argc, const char **argv)

7.6.1 Detailed Description

7.6.2 File description

a sample of hashtable usages.

7.6.3 Copyright

This source code is protected by the French intellectual property law.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; version 2 of the License.

7.6.4 File informations

$Date$ 2014/03/28 Rev 0.1 $Author$ Benjamin Sientzoff URL `http://www.github.com/blasterbug`

7.6.5 Macro Definition Documentation

7.6.5.1 #define K string

7.6.5.2 #define V string

7.6.6 Function Documentation

7.6.6.1 template<> unsigned **computehash**< K >(K *element*)

7.6.6.2 int main (int *argc,* const char ** *argv*)

7.7 src/sample_tree.cpp File Reference

```
#include <functional>
#include <iostream>
#include <cstdlib>
#include <fstream>
#include "tree.hpp"
```

Functions

- int main (int argc, const char **argv)

7.7.1 Detailed Description

7.7.2 File description

a sample showing how to use a Tree

7.7.3 Copyright

7.7.4 File informations

$Date$ 2014/04/03 Rev 0.1 $Author$ Benjamin Sientzoff URL `http://www.github.com/blasterbug`

7.7.5 Function Documentation

7.7.5.1 int main (int *argc,* const char ** *argv*)

7.8 src/sample_treestring.cpp File Reference

```
#include <functional>
#include <iostream>
#include <cstdlib>
#include <fstream>
#include "treestring.hpp"
```

Functions

- int main (int argc, const char **argv)

7.8.1 Detailed Description

7.8.2 File description

a sample to show how to use TreeString class

7.8.3 Copyright

7.8.4 File informations

$Date$ 2014/04/21 Rev 0.1 $Author$ Benjamin Sientzoff URL `http://www.github.com/blasterbug`

7.8.5 Function Documentation

7.8.5.1 int main (int *argc,* const char ** *argv*)

7.9 src/tree.hpp File Reference

```
#include <string>
#include <forward_list>
```

Classes

- class TreeException

  exception class for trees
- class Node< T >

  Defines tree nodes.
- class Tree< T >

  Tree is a recursive structure using nodes.

7.9.1 Detailed Description

7.9.2 File description

Tree is a recursive structure using nodes. Node stores a value (tag) and has several children

7.9.3 Copyright

This source code is protected by the French intellectual property law.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; version 2 of the License.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

7.9.4 File informations

$Date$ 2014/03/27 Rev 0.3 $Author$ Benjamin Sientzoff URL `http://www.github.com/blasterbug`

7.10 src/treestring.hpp File Reference

```
#include <cassert>
#include <string>
#include <forward_list>
#include <utility>
#include <sstream>
```

Classes

- class TreeStringException

exception class for trees

- class Node< T >

  Defines tree nodes.

- class TreeString

  Tree is a recursive structure using nodes.

7.10.1 Detailed Description

7.10.2 File description

TreeString is a recursive structure using nodes. a Node stores a letter and has several children

7.10.3 Copyright

7.10.4 File informations

$Date$ 2014/04/5 Rev 0.1 $Author$ Benjamin Sientzoff URL `http://www.github.com/blasterbug`

Index