

Glossygloss

0.3

Generated by Doxygen 1.8.6

Thu Apr 17 2014 18:53:46

Contents

1	Main Page	1
2	Todo List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Class Documentation	11
6.1	Alveole< K, V > Class Template Reference	11
6.1.1	Detailed Description	11
6.1.2	Constructor & Destructor Documentation	11
6.1.2.1	Alveole	11
6.1.2.2	Alveole	12
6.1.2.3	Alveole	12
6.1.2.4	Alveole	12
6.1.2.5	~Alveole	12
6.1.3	Member Function Documentation	12
6.1.3.1	getKey	12
6.1.3.2	getNext	12
6.1.3.3	getValue	12
6.1.3.4	setNext	13
6.1.3.5	setValue	13
6.1.3.6	toString	13
6.1.4	Member Data Documentation	13
6.1.4.1	_key	13
6.1.4.2	_next	13
6.1.4.3	_value	13

6.2	Dictionnaire< V > Class Template Reference	13
6.2.1	Constructor & Destructor Documentation	14
6.2.1.1	Dictionnaire	14
6.2.1.2	~Dictionnaire	14
6.2.2	Member Function Documentation	14
6.2.2.1	ajouterMot	14
6.2.2.2	associerMot	14
6.2.2.3	contientMot	14
6.2.2.4	supprimerMot	14
6.2.2.5	valeurAssociee	15
6.2.3	Member Data Documentation	15
6.2.3.1	dico	15
6.3	HashException Class Reference	15
6.3.1	Detailed Description	15
6.3.2	Constructor & Destructor Documentation	16
6.3.2.1	HashException	16
6.3.2.2	~HashException	16
6.3.3	Member Function Documentation	16
6.3.3.1	what	16
6.3.4	Member Data Documentation	16
6.3.4.1	_cause	16
6.4	Hashtable< K, V > Class Template Reference	16
6.4.1	Detailed Description	17
6.4.2	Constructor & Destructor Documentation	17
6.4.2.1	Hashtable	17
6.4.2.2	~Hashtable	17
6.4.3	Member Function Documentation	17
6.4.3.1	contains	17
6.4.3.2	get	17
6.4.3.3	isEmpty	17
6.4.3.4	put	17
6.4.3.5	remove	18
6.4.3.6	toString	18
6.4.4	Member Data Documentation	18
6.4.4.1	_table	18
6.5	Node< T > Class Template Reference	18
6.5.1	Detailed Description	19
6.5.2	Constructor & Destructor Documentation	19
6.5.2.1	Node	19
6.5.2.2	Node	19

6.5.2.3	~Node	19
6.5.3	Member Function Documentation	19
6.5.3.1	append	19
6.5.3.2	contains	19
6.5.3.3	getTag	20
6.5.3.4	height	20
6.5.3.5	isLeaf	20
6.5.3.6	operator!=	20
6.5.3.7	operator=	20
6.5.3.8	operator==	20
6.5.3.9	remove	21
6.5.3.10	toString	21
6.5.4	Member Data Documentation	21
6.5.4.1	_childNbr	21
6.5.4.2	_children	21
6.5.4.3	_tag	21
6.6	pair< L, R > Class Template Reference	21
6.6.1	Detailed Description	22
6.6.2	Constructor & Destructor Documentation	22
6.6.2.1	pair	22
6.6.2.2	pair	22
6.6.2.3	~pair	22
6.6.3	Member Function Documentation	22
6.6.3.1	getLeft	22
6.6.3.2	getRight	22
6.6.4	Member Data Documentation	23
6.6.4.1	_left	23
6.6.4.2	_right	23
6.7	Tree< T > Class Template Reference	23
6.7.1	Detailed Description	23
6.7.2	Constructor & Destructor Documentation	23
6.7.2.1	Tree	23
6.7.2.2	Tree	24
6.7.2.3	Tree	24
6.7.2.4	~Tree	24
6.7.3	Member Function Documentation	24
6.7.3.1	contains	24
6.7.3.2	height	24
6.7.3.3	put	24
6.7.3.4	remove	24

6.7.3.5	toString	24
6.7.4	Member Data Documentation	25
6.7.4.1	_root	25
6.8	TreeException Class Reference	25
6.8.1	Detailed Description	25
6.8.2	Constructor & Destructor Documentation	25
6.8.2.1	TreeException	25
6.8.2.2	~TreeException	26
6.8.3	Member Function Documentation	26
6.8.3.1	what	26
6.8.4	Member Data Documentation	26
6.8.4.1	_cause	26
7	File Documentation	27
7.1	README.md File Reference	27
7.2	src/application.cpp File Reference	27
7.2.1	Function Documentation	27
7.2.1.1	computehash< string >	27
7.2.1.2	main	27
7.3	src/dictionnaire/dictionnaire.hpp File Reference	27
7.3.1	Detailed Description	27
7.3.2	File description	27
7.3.3	Copyright	28
7.3.4	File informations	28
7.4	src/hashtable.hpp File Reference	28
7.4.1	Detailed Description	28
7.4.2	File description	28
7.4.3	Copyright	29
7.4.4	File informations	29
7.4.5	Macro Definition Documentation	29
7.4.5.1	ARRAYSIZE	29
7.4.5.2	END	29
7.4.6	Function Documentation	29
7.4.6.1	computehash	29
7.5	src/pair.hpp File Reference	29
7.5.1	Detailed Description	30
7.5.2	File description	30
7.5.3	Copyright	30
7.5.4	File informations	30
7.6	src/sample_hashtable.cpp File Reference	30

7.6.1	Detailed Description	30
7.6.2	File description	30
7.6.3	Copyright	31
7.6.4	File informations	31
7.6.5	Macro Definition Documentation	31
7.6.5.1	K	31
7.6.5.2	V	31
7.6.6	Function Documentation	31
7.6.6.1	computehash< K >	31
7.6.6.2	main	31
7.7	src/sample_tree.cpp File Reference	31
7.7.1	Detailed Description	31
7.7.2	File description	31
7.7.3	Copyright	32
7.7.4	File informations	32
7.7.5	Macro Definition Documentation	32
7.7.5.1	K	32
7.7.6	Function Documentation	32
7.7.6.1	main	32
7.8	src/tree.hpp File Reference	32
7.8.1	Detailed Description	32
7.8.2	File description	32
7.8.3	Copyright	32
7.8.4	File informations	33

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

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.

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:

Alveole< K, V >	11
Alveole< string, V >	11
Dictionnaire< V >	13
exception	
HashException	15
TreeException	25
Hashtable< K, V >	16
Hashtable< string, V >	16
Node< T >	18
pair< L, R >	21
Tree< T >	23

Chapter 4

Class Index

4.1 Class List

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

Alveole< K, V >	
Class to define Hashtable alveoles	11
Dictionnaire< V >	13
HashException	
Exception class to manage Hashtable errors	15
Hashtable< K, V >	
Maps a key to a value	16
Node< T >	
Defines tree nodes	18
pair< L, R >	21
Tree< T >	
Tree is a recursive structure using nodes	23
TreeException	
Exception class for trees	25

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

src/ application.cpp	27
src/ hashtable.hpp	28
src/ pair.hpp	29
src/ sample_hashtable.cpp	30
src/ sample_tree.cpp	31
src/ tree.hpp	32
src/dictionnaire/ dictionnaire.hpp	27

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	<i>other</i>	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	<i>key</i>	key of the pair
in	<i>value</i>	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	<i>key</i>	key of the pair
in	<i>value</i>	value of the pair
in	<i>next</i>	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	<i>key</i>	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	<i>ptr</i>	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	<i>value</i>	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 address of the next alveole

Parameters

in	<i>n_next</i>	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	<i>n_value</i>	The new value of the pair
----	----------------	---------------------------

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

Return a string description of the pair stored into the alveole

Parameters

out	<i>desc</i>	a string representation 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< V > Class Template Reference

```
#include <dictionnaire.hpp>
```

Public Member Functions

- [Dictionnaire](#) ()
- [~Dictionnaire](#) ()
- bool [contientMot](#) (string mot)

- void [ajouterMot](#) (string mot)
- bool [associerMot](#) (string mot)
- bool [supprimerMot](#) (string mot)
- [V valeurAssociee](#) (string mot)

Private Attributes

- [Hashtable](#)< string, [V](#) > [dico](#)

6.2.1 Constructor & Destructor Documentation

6.2.1.1 `template<typename V > Dictionnaire< V >::Dictionnaire ()`

Constructeur de la classe [Dictionnaire](#)

6.2.1.2 `template<typename V > Dictionnaire< V >::~~Dictionnaire ()`

Destructeur de la classe [Dictionnaire](#)

6.2.2 Member Function Documentation

6.2.2.1 `template<typename V > void Dictionnaire< V >::ajouterMot (string mot)`

Fonction qui ajoute un mot non présent dans le dictionnaire

Parameters

in	<i>mot,le</i>	mot à ajouter
in	<i>v,la</i>	valeur associée

6.2.2.2 `template<typename V > bool Dictionnaire< V >::associerMot (string mot)`

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

Parameters

in	<i>mot,le</i>	mot à modifier
in	<i>v,la</i>	valeur à modifier
out	<i>bool</i>	Renvoyer faux si le mot n'est pas présent, sinon vrai

6.2.2.3 `template<typename V > bool Dictionnaire< V >::contientMot (string mot)`

Fonction qui renvoie vrai le mot est présent dans le [Dictionnaire](#)

Parameters

in	<i>mot,le</i>	mot à tester
out	<i>bool,vrai</i>	si présent, faux sinon.

6.2.2.4 `template<typename V > bool Dictionnaire< V >::supprimerMot (string mot)`

Fonction qui supprime un mot présent dans le dictionnaire

Parameters

in	<i>mot,le</i>	mot à supprimer
out	<i>bool</i>	Renvoie vrai si le mot a été supprimé, sinon faux

6.2.2.5 `template<typename V > V Dictionnaire< V >::valeurAssociee (string mot)`

Fonction qui récupère la valeur associée au mot

Parameters

in	<i>mot,le</i>	mot dont on souhaite savoir la valeur associée
out	<i>valeur,la</i>	valeur associée

Exceptions

<i>lève</i>	une exception si le mot n'est pas présent dans le dictionnaire
-------------	--

6.2.3 Member Data Documentation

6.2.3.1 `template<typename V > Hashtable<string,V> Dictionnaire< V >::dico [private]`

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

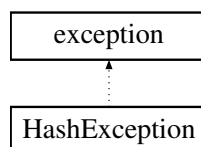
- [src/dictionnaire/dictionnaire.hpp](#)

6.3 HashException Class Reference

Exception class to manage [Hashtable](#) errors.

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

Inheritance diagram for HashException:



Public Member Functions

- [HashException](#) (const char *cause)
- virtual [~HashException](#) () throw ()
- virtual const char * [what](#) () const throw ()

Private Attributes

- const char * [_cause](#)

6.3.1 Detailed Description

Exception class to manage [Hashtable](#) errors.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 HashException::HashException (const char * *cause*)

store exception description constructor called then HashExceptions are threw

Parameters

in	<i>cause</i>	description of exception origin
----	--------------	---------------------------------

6.3.2.2 virtual HashException::~~HashException () throw) [virtual]

destructor currently, do anything special

6.3.3 Member Function Documentation

6.3.3.1 virtual const char* HashException::what () const throw) [virtual]

virtual fonction from superclass, usefull to get the exception description

6.3.4 Member Data Documentation

6.3.4.1 const char* HashException::_cause [private]

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

- [src/hashtable.hpp](#)

6.4 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](#) ()

Private Attributes

- [Alveole](#)< [K](#), [V](#) > ** [_table](#)

6.4.1 Detailed Description

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

Maps a key to a value.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 `template<typename K, typename V> Hashtable< K, V >::Hashtable ()`

array of alveoles Simple constructor

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

Destructor

6.4.3 Member Function Documentation

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

Do table contains key ?

Parameters

in	<i>key</i>	key to find
out	<i>bool</i>	True if the key is here, else false

6.4.3.2 `template<typename K, typename V> V Hashtable< K, V >::get (const K & key)`

Return the value mapped to the specified key

Parameters

in	<i>key</i>	a key in the hashtable
out	<i>value</i>	value associated with the key

Exceptions

<i>HashException</i>	threw if key is not in the hashtable
--------------------------------------	--------------------------------------

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

Tests if this hashtable maps no keys to values.

Parameters

out	<i>bool</i>	true if no elements in the hashtable, else false;
-----	-------------	---

6.4.3.4 `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 mapped value to the key

Parameters

in	<i>key</i>	key of the pair
in	<i>value</i>	value of the pair

6.4.3.5 `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	<i>key</i>	Key of the pair to delete
----	------------	---------------------------

Exceptions

<i>HashException</i>	threw if table does not contain key
--------------------------------------	-------------------------------------

6.4.3.6 `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	<i>desc</i>	a string representation of this hashtable.
-----	-------------	--

6.4.4 Member Data Documentation

6.4.4.1 `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](#)

6.5 `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](#) ()

Private Attributes

- int `_childNbr`
Number of children.
- T `_tag`
letter stored into `Node`, the tag
- forward_list< `Node`< T > > `_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

6.5.2 Constructor & Destructor Documentation

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

Copy constructor

Parameters

<code>in</code>	<code>other</code>	<code>Node</code> to copy
-----------------	--------------------	---------------------------

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

Simple constructor

Parameters

<code>in</code>	<code>data</code>	to store into the <code>Node</code>
-----------------	-------------------	-------------------------------------

6.5.2.3 `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

<code>in</code>	<code>n_data</code>	new data to store as a child of the node
-----------------	---------------------	--

6.5.3.2 `template<typename T = char> bool Node< T >::contains (T element)`

Do the tag is element or one of his children ?

Parameters

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

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

What is the tag of the [Node](#) ?

Parameters

out	<i>tag</i>	The tag of the node
-----	------------	---------------------

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

The height of the node

Parameters

out	<i>hgt</i>	height of the node
-----	------------	--------------------

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

Is the node a leaf ?

Parameters

out	<i>bool</i>	true, if no child, else false
-----	-------------	-------------------------------

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

inequality operator

Parameters

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

6.5.3.7 `template<typename T = char> Node<T> & Node< T >::operator= (Node< T > & other)`

assignment operator overload

Parameters

in	<i>other</i>	node to assign
out	<i>note</i>	assigned node

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

equality operator

Parameters

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

6.5.3.9 `template<typename T = char> void Node< T >::remove (T data)`

Remove a leaf from the node

Parameters

in	<i>data</i>	data of the node's tag to remove
----	-------------	----------------------------------

Exceptions

TreeException	Threw if data is not removed
-------------------------------	------------------------------

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

Get a string representation of the node and his child

Parameters

out	<i>desc</i>	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> T Node< T >::_tag` [private]

letter stored into [Node](#), the tag

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

- [src/tree.hpp](#)

6.6 pair< L, R > Class Template Reference

```
#include <pair.hpp>
```

Public Member Functions

- [pair](#) (L value1, R value2)
- [pair](#) (const [pair](#)< L, R > &other)
- [~pair](#) ()

- L [getLeft](#) ()
- R [getRight](#) ()

Private Attributes

- L [_left](#)
- R [_right](#)

6.6.1 Detailed Description

```
template<typename L, typename R>class pair< L, R >
```

A pair is set of two elements.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 `template<typename L, typename R> pair< L, R >::pair (L value1, R value2)`

the second value stored in the pair Common constructor

Parameters

in	<i>value1</i>	The first value to store
in	<i>value2</i>	The second value to store

6.6.2.2 `template<typename L, typename R> pair< L, R >::pair (const pair< L, R > & other)`

Copy constructor

Parameters

in	<i>other</i>	The pair to copy
----	--------------	------------------

6.6.2.3 `template<typename L, typename R> pair< L, R >::~~pair ()`

Destructor for pair

6.6.3 Member Function Documentation

6.6.3.1 `template<typename L, typename R> L pair< L, R >::getLeft ()`

Get the first value

Parameters

out	<i>value1</i>	The first value stored into the pair
-----	---------------	--------------------------------------

6.6.3.2 `template<typename L, typename R> R pair< L, R >::getRight ()`

Get the second value

Parameters

out	value2	The second value stored by the pair
-----	--------	-------------------------------------

6.6.4 Member Data Documentation

6.6.4.1 `template<typename L, typename R> L pair< L, R >::_left` [private]

6.6.4.2 `template<typename L, typename R> R pair< L, R >::_right` [private]

the first value of the pair

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

- [src/pair.hpp](#)

6.7 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](#)

6.7.1 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.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 `template<typename T = string> Tree< T >::Tree ()`

First node of the tree Default constructor

6.7.2.2 `template<typename T = string> Tree< T >::Tree (const Tree< T > & other)`

Copy constructor

6.7.2.3 `template<typename T = string> Tree< T >::Tree (T element)`

Common constructor, create an tree

Parameters

in	<i>element</i>	Root of the tree
----	----------------	------------------

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

Destructor, destroy the whole tree

6.7.3 Member Function Documentation

6.7.3.1 `template<typename T = string> bool Tree< T >::contains (T element)`

Is the element in the tree ?

Parameters

in	<i>element</i>	Search the element in the Tree
out	<i>bool</i>	True if element is here, else false.

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

The height of the tree

Parameters

out	<i>hgt</i>	Height of the tree
-----	------------	--------------------

6.7.3.3 `template<typename T = string> void Tree< T >::put (T element)`

Put an element in the tree

Parameters

in	<i>element</i>	New element to put into the tree
----	----------------	----------------------------------

6.7.3.4 `template<typename T = string> void Tree< T >::remove (T element)`

Remove an element from the tree

Parameters

in	<i>data</i>	Element to remove
----	-------------	-------------------

6.7.3.5 `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
-----	------	-----------------------------------

6.7.4 Member Data Documentation

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

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

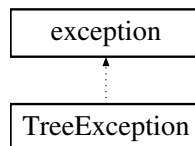
- [src/tree.hpp](#)

6.8 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](#)

6.8.1 Detailed Description

exception class for trees

Usefull to manage errors and the unforeseen

6.8.2 Constructor & Destructor Documentation

6.8.2.1 `TreeException::TreeException (char * cause)`

store exception description constructor called then TreeExceptions are threw

Parameters

in	<i>cause</i>	description of exception origin
----	--------------	---------------------------------

6.8.2.2 virtual `TreeException::~~TreeException () throw` [virtual]

destructor currently, do anything special

6.8.3 Member Function Documentation

6.8.3.1 virtual const char* `TreeException::what () const throw` [virtual]

virtual fonction from superclass, usefull to get the exception description

6.8.4 Member Data Documentation

6.8.4.1 char* `TreeException::_cause` [private]

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

- [src/tree.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/dictionnaire.hpp"
```

Functions

- `template<> unsigned computehash< string > (string element)`
- `int main ()`

7.2.1 Function Documentation

7.2.1.1 `template<> unsigned computehash< string > (string element)`

7.2.1.2 `int main ()`

7.3 src/dictionnaire/dictionnaire.hpp File Reference

```
#include "../hashtable.hpp"
```

Classes

- `class Dictionnaire< V >`

7.3.1 Detailed Description

7.3.2 File description

`Dictionnaire` utilisant une hashtable

7.3.3 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.

7.3.4 File informations

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

7.4 src/hashtable.hpp File Reference

```
#include <string>
#include <cassert>
```

Classes

- class [HashException](#)
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 10`
macro to define size of hash arrays

Functions

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

7.4.1 Detailed Description

7.4.2 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 !!

7.4.3 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.

7.4.4 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)

7.4.5 Macro Definition Documentation

7.4.5.1 #define ARRAYSIZE 10

macro to define size of hash arrays

7.4.5.2 #define END nullptr

macro to define end of alveole chains

7.4.6 Function Documentation

7.4.6.1 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	<i>element</i>	element to compute hashcode from
out	<i>hashcode</i>	the hashcode of element, an unsigned integer

template<> unsigned [computehash<string>\(string element\)](#)

your implementation of hashcode function

7.5 src/pair.hpp File Reference

```
#include <string>
#include <cassert>
```

Classes

- class [pair< L, R >](#)

7.5.1 Detailed Description

7.5.2 File description

data structure to store pairs. A pair is a set of two values, not necessary the same type

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\$ 2014/04/15 \$Rev\$ 0.2 \$Author\$ Benjamin Sientzoff \$URL\$ <http://www.github.com/blasterbug>

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.

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.

7.6.4 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"
```

Macros

- `#define K string`

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

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.

7.7.4 File informations

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

7.7.5 Macro Definition Documentation

7.7.5.1 `#define K string`

7.7.6 Function Documentation

7.7.6.1 `int main (int argc, const char ** argv)`

7.8 src/tree.hpp File Reference

```
#include <cassert>
#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.

7.8.1 Detailed Description

7.8.2 File description

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

7.8.3 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.

7.8.4 File informations

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

Index

- ~Alveole
 - Alveole, [12](#)
- ~Dictionnaire
 - Dictionnaire, [14](#)
- ~HashException
 - HashException, [16](#)
- ~Hashtable
 - Hashtable, [17](#)
- ~Node
 - Node, [19](#)
- ~Tree
 - Tree, [24](#)
- ~TreeException
 - TreeException, [26](#)
- ~pair
 - pair, [22](#)
- _cause
 - HashException, [16](#)
 - TreeException, [26](#)
- _childNbr
 - Node, [21](#)
- _children
 - Node, [21](#)
- _key
 - Alveole, [13](#)
- _left
 - pair, [23](#)
- _next
 - Alveole, [13](#)
- _right
 - pair, [23](#)
- _root
 - Tree, [25](#)
- _table
 - Hashtable, [18](#)
- _tag
 - Node, [21](#)
- _value
 - Alveole, [13](#)
- ARRAYSIZE
 - hashtable.hpp, [29](#)
- ajouterMot
 - Dictionnaire, [14](#)
- Alveole
 - ~Alveole, [12](#)
 - _key, [13](#)
 - _next, [13](#)
 - _value, [13](#)
 - Alveole, [11](#), [12](#)
 - getKey, [12](#)
 - getNext, [12](#)
 - getValue, [12](#)
 - setNext, [13](#)
 - setValue, [13](#)
 - toString, [13](#)
- Alveole< K, V >, [11](#)
- append
 - Node, [19](#)
- application.cpp
 - computehash< string >, [27](#)
 - main, [27](#)
- associerMot
 - Dictionnaire, [14](#)
- computehash
 - hashtable.hpp, [29](#)
- computehash< K >
 - sample_hashtable.cpp, [31](#)
- computehash< string >
 - application.cpp, [27](#)
- contains
 - Hashtable, [17](#)
 - Node, [19](#)
 - Tree, [24](#)
- contientMot
 - Dictionnaire, [14](#)
- dico
 - Dictionnaire, [15](#)
- Dictionnaire
 - ~Dictionnaire, [14](#)
 - ajouterMot, [14](#)
 - associerMot, [14](#)
 - contientMot, [14](#)
 - dico, [15](#)
 - Dictionnaire, [14](#)
 - supprimerMot, [14](#)
 - valeurAssociee, [15](#)
- Dictionnaire< V >, [13](#)
- END
 - hashtable.hpp, [29](#)
- get
 - Hashtable, [17](#)
- getKey
 - Alveole, [12](#)
- getLeft
 - pair, [22](#)

- getNext
 - Alveole, [12](#)
- getRight
 - pair, [22](#)
- getTag
 - Node, [20](#)
- getValue
 - Alveole, [12](#)
- HashException, [15](#)
 - ~HashException, [16](#)
 - _cause, [16](#)
 - HashException, [16](#)
 - HashException, [16](#)
 - what, [16](#)
- Hashtable
 - ~Hashtable, [17](#)
 - _table, [18](#)
 - contains, [17](#)
 - get, [17](#)
 - Hashtable, [17](#)
 - isEmpty, [17](#)
 - put, [17](#)
 - remove, [18](#)
 - toString, [18](#)
- Hashtable< K, V >, [16](#)
- hashtable.hpp
 - ARRAYSIZE, [29](#)
 - computehash, [29](#)
 - END, [29](#)
- height
 - Node, [20](#)
 - Tree, [24](#)
- isEmpty
 - Hashtable, [17](#)
- isLeaf
 - Node, [20](#)
- K
 - sample_hashtable.cpp, [31](#)
 - sample_tree.cpp, [32](#)
- main
 - application.cpp, [27](#)
 - sample_hashtable.cpp, [31](#)
 - sample_tree.cpp, [32](#)
- Node
 - ~Node, [19](#)
 - _childNbr, [21](#)
 - _children, [21](#)
 - _tag, [21](#)
 - append, [19](#)
 - contains, [19](#)
 - getTag, [20](#)
 - height, [20](#)
 - isLeaf, [20](#)
 - Node, [19](#)
 - operator=, [20](#)
 - operator==, [20](#)
 - remove, [21](#)
 - toString, [21](#)
 - Node< T >, [18](#)
- operator=
 - Node, [20](#)
- operator==
 - Node, [20](#)
- pair
 - ~pair, [22](#)
 - _left, [23](#)
 - _right, [23](#)
 - getLeft, [22](#)
 - getRight, [22](#)
 - pair, [22](#)
- pair< L, R >, [21](#)
- put
 - Hashtable, [17](#)
 - Tree, [24](#)
- README.md, [27](#)
- remove
 - Hashtable, [18](#)
 - Node, [21](#)
 - Tree, [24](#)
- sample_hashtable.cpp
 - computehash< K >, [31](#)
 - K, [31](#)
 - main, [31](#)
 - V, [31](#)
- sample_tree.cpp
 - K, [32](#)
 - main, [32](#)
- setNext
 - Alveole, [13](#)
- setValue
 - Alveole, [13](#)
- src/application.cpp, [27](#)
- src/dictionnaire/dictionnaire.hpp, [27](#)
- src/hashtable.hpp, [28](#)
- src/pair.hpp, [29](#)
- src/sample_hashtable.cpp, [30](#)
- src/sample_tree.cpp, [31](#)
- src/tree.hpp, [32](#)
- supprimerMot
 - Dictionnaire, [14](#)
- toString
 - Alveole, [13](#)
 - Hashtable, [18](#)
 - Node, [21](#)
 - Tree, [24](#)
- Tree
 - ~Tree, [24](#)
 - _root, [25](#)

- contains, [24](#)
- height, [24](#)
- put, [24](#)
- remove, [24](#)
- toString, [24](#)
- Tree, [23](#), [24](#)
- Tree< T >, [23](#)
- TreeException, [25](#)
 - ~TreeException, [26](#)
 - _cause, [26](#)
 - TreeException, [25](#)
 - TreeException, [25](#)
 - what, [26](#)
- V
 - sample_hashtable.cpp, [31](#)
- valeurAssociee
 - Dictionnaire, [15](#)
- what
 - HashException, [16](#)
 - TreeException, [26](#)