

# Glossygloss

0.2

Generated by Doxygen 1.8.6

Tue Apr 1 2014 19:25:40



# Contents

<b>1</b>	<b><a href="#">glossygloss</a></b>	<b>1</b>
<b>2</b>	<b><a href="#">Hierarchical Index</a></b>	<b>3</b>
2.1	<a href="#">Class Hierarchy</a>	3
<b>3</b>	<b><a href="#">Class Index</a></b>	<b>5</b>
3.1	<a href="#">Class List</a>	5
<b>4</b>	<b><a href="#">File Index</a></b>	<b>7</b>
4.1	<a href="#">File List</a>	7
<b>5</b>	<b><a href="#">Class Documentation</a></b>	<b>9</b>
5.1	<a href="#">Alveole&lt; K, V &gt; Class Template Reference</a>	9
5.1.1	<a href="#">Detailed Description</a>	9
5.1.2	<a href="#">Constructor &amp; Destructor Documentation</a>	9
5.1.2.1	<a href="#">Alveole</a>	9
5.1.2.2	<a href="#">Alveole</a>	10
5.1.2.3	<a href="#">Alveole</a>	10
5.1.2.4	<a href="#">Alveole</a>	10
5.1.2.5	<a href="#">~Alveole</a>	10
5.1.3	<a href="#">Member Function Documentation</a>	10
5.1.3.1	<a href="#">getKey</a>	10
5.1.3.2	<a href="#">getNext</a>	10
5.1.3.3	<a href="#">getValue</a>	10
5.1.3.4	<a href="#">setNext</a>	11
5.1.3.5	<a href="#">setValue</a>	11
5.1.3.6	<a href="#">toString</a>	11
5.1.4	<a href="#">Member Data Documentation</a>	11
5.1.4.1	<a href="#">_key</a>	11
5.1.4.2	<a href="#">_next</a>	11
5.1.4.3	<a href="#">_value</a>	11
5.2	<a href="#">HashException Class Reference</a>	11
5.2.1	<a href="#">Detailed Description</a>	12

5.2.2	Constructor & Destructor Documentation . . . . .	12
5.2.2.1	HashException . . . . .	12
5.2.2.2	~HashException . . . . .	12
5.2.3	Member Function Documentation . . . . .	12
5.2.3.1	what . . . . .	12
5.2.4	Member Data Documentation . . . . .	12
5.2.4.1	_cause . . . . .	12
5.3	Hashtable< K, V > Class Template Reference . . . . .	13
5.3.1	Detailed Description . . . . .	13
5.3.2	Constructor & Destructor Documentation . . . . .	13
5.3.2.1	Hashtable . . . . .	13
5.3.2.2	~Hashtable . . . . .	13
5.3.3	Member Function Documentation . . . . .	13
5.3.3.1	contains . . . . .	13
5.3.3.2	get . . . . .	13
5.3.3.3	isEmpty . . . . .	14
5.3.3.4	put . . . . .	14
5.3.3.5	remove . . . . .	14
5.3.3.6	toString . . . . .	14
5.3.4	Member Data Documentation . . . . .	14
5.3.4.1	_table . . . . .	14
5.4	Node< T > Class Template Reference . . . . .	15
5.4.1	Detailed Description . . . . .	15
5.4.2	Constructor & Destructor Documentation . . . . .	15
5.4.2.1	Node . . . . .	15
5.4.2.2	~Node . . . . .	15
5.4.3	Member Function Documentation . . . . .	16
5.4.3.1	_children . . . . .	16
5.4.3.2	append . . . . .	17
5.4.3.3	height . . . . .	17
5.4.3.4	isLeaf . . . . .	17
5.4.3.5	operator!= . . . . .	17
5.4.3.6	operator= . . . . .	17
5.4.3.7	operator== . . . . .	17
5.4.3.8	remove . . . . .	18
5.4.3.9	toString . . . . .	18
5.4.4	Member Data Documentation . . . . .	18
5.4.4.1	_children . . . . .	18
5.4.4.2	_tag . . . . .	18
5.5	Tree< T > Class Template Reference . . . . .	18

5.5.1	Detailed Description	19
5.5.2	Constructor & Destructor Documentation	19
5.5.2.1	Tree	19
5.5.2.2	Tree	19
5.5.2.3	~Tree	19
5.5.3	Member Function Documentation	19
5.5.3.1	add	19
5.5.3.2	contains	19
5.5.3.3	count	20
5.5.3.4	elements	20
5.5.3.5	height	20
5.5.3.6	operator!=	20
5.5.3.7	operator=	20
5.5.3.8	operator==	20
5.5.3.9	remove	20
5.5.4	Member Data Documentation	20
5.5.4.1	_root	20
5.6	TreeException Class Reference	20
5.6.1	Detailed Description	21
5.6.2	Constructor & Destructor Documentation	21
5.6.2.1	TreeException	21
5.6.2.2	~BagException	21
5.6.3	Member Function Documentation	21
5.6.3.1	what	21
5.6.4	Member Data Documentation	21
5.6.4.1	_cause	21
<b>6</b>	<b>File Documentation</b>	<b>23</b>
6.1	README.md File Reference	23
6.2	src/hashtable.hpp File Reference	23
6.2.1	Detailed Description	23
6.2.2	File description	23
6.2.3	Copyright	24
6.2.4	File informations	24
6.2.5	Macro Definition Documentation	24
6.2.5.1	ARRAYSIZE	24
6.2.5.2	END	24
6.2.6	Function Documentation	24
6.2.6.1	computehash	24
6.3	src/tree.hpp File Reference	24

---

6.3.1	Detailed Description . . . . .	25
6.3.2	File description . . . . .	25
6.3.3	Copyright . . . . .	25
6.3.4	File informations . . . . .	25
<b>Index</b>		<b>26</b>

# Chapter 1

## glossygloss

Glossygloss is set of classes to use several data structure like [Tree](#) and [Hashtable](#). More might come soon.

Documentation ?

All documented things are here : <http://blasterbug.github.io/glossygloss/>

Usefull : [http://en.wikibooks.org/wiki/C%2B%2B\\_Programming](http://en.wikibooks.org/wiki/C%2B%2B_Programming)





## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

Alveole< K, V > . . . . .	9
exception	
HashException . . . . .	11
TreeException . . . . .	20
Hashtable< K, V > . . . . .	13
Node< T > . . . . .	15
Tree< T > . . . . .	18



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Alveole&lt; K, V &gt;</a>	Class to define <a href="#">Hashtable</a> alveoles . . . . .	9
<a href="#">HashException</a>	Exception class to manage <a href="#">Hashtable</a> errors . . . . .	11
<a href="#">Hashtable&lt; K, V &gt;</a>	Maps a key to a value . . . . .	13
<a href="#">Node&lt; T &gt;</a>	Defines tree nodes . . . . .	15
<a href="#">Tree&lt; T &gt;</a>	<a href="#">Tree</a> is a recursive structure using nodes . . . . .	18
<a href="#">TreeException</a>	Exception class for trees . . . . .	20



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

src/ <a href="#">hashtable.hpp</a> . . . . .	23
src/ <a href="#">tree.hpp</a> . . . . .	24



## Chapter 5

# Class Documentation

### 5.1 `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`

#### 5.1.1 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.

#### 5.1.2 Constructor & Destructor Documentation

5.1.2.1 `template<typename K, typename V> Alveole< K, V >::Alveole ( const Alveole< K, V > &other )` `[inline]`

next aveole Copy constructor

## Parameters

<i>in</i>	<i>other</i>	the alveole to copy
-----------	--------------	---------------------

5.1.2.2 `template<typename K, typename V> Alveole< K, V >::Alveole ( K key, V value )` `[inline]`

## Pair constructor

## Parameters

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

5.1.2.3 `template<typename K, typename V> Alveole< K, V >::Alveole ( )` `[inline]`

Empty constructor create an 'empty' alveole

5.1.2.4 `template<typename K, typename V> Alveole< K, V >::Alveole ( K key, V value, Alveole< K, V > * next )`  
`[inline]`

## Complex constructor

## Parameters

<i>in</i>	<i>key</i>	key of the pair
<i>in</i>	<i>value</i>	value of the pair
<i>in</i>	<i>next</i>	adresse to the next alveole

5.1.2.5 `template<typename K, typename V> Alveole< K, V >::~~Alveole ( )` `[inline]`

Destructor for [Alveole](#)

## 5.1.3 Member Function Documentation

5.1.3.1 `template<typename K, typename V> K Alveole< K, V >::getKey ( )` `[inline]`

Get the key of an alveole

## Parameters

<i>out</i>	<i>key</i>	key stored into the alveole
------------	------------	-----------------------------

5.1.3.2 `template<typename K, typename V> Alveole<K,V>* Alveole< K, V >::getNext ( )` `[inline]`

Which alveole coming next ?

## Parameters

<i>out</i>	<i>ptr</i>	memory adress of the next alveole
------------	------------	-----------------------------------

5.1.3.3 `template<typename K, typename V> V Alveole< K, V >::getValue ( )` `[inline]`

Get the value stored into an alveole



## Parameters

out	<i>value</i>	value of the alveole
-----	--------------	----------------------

5.1.3.4 `template<typename K, typename V> void Alveole< K, V >::setNext ( Alveole< K, V > * n_next )` `[inline]`

Set the next address of the next alveole

## Parameters

in	<i>n_next</i>	adress of the new next alveole
----	---------------	--------------------------------

5.1.3.5 `template<typename K, typename V> void Alveole< K, V >::setValue ( V n_value )` `[inline]`

Set the value stored into an alveole

## Parameters

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

5.1.3.6 `template<typename K, typename V> string Alveole< K, V >::toString ( )` `[inline]`

Return a string description of the pair stored into the alveole

## Parameters

out	<i>desc</i>	a string representation of the alveole
-----	-------------	--

## 5.1.4 Member Data Documentation

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

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

value of the pair

5.1.4.3 `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:

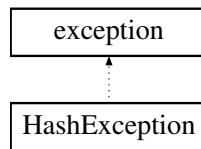
- [src/hashtable.hpp](#)

## 5.2 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](#)

### 5.2.1 Detailed Description

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

### 5.2.2 Constructor & Destructor Documentation

5.2.2.1 `HashException::HashException ( const char * cause )` `[inline]`

store exception description constructor called then HashExceptions are threw

Parameters

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

5.2.2.2 `virtual HashException::~~HashException ( ) throw` `[inline]`, `[virtual]`

destructor currently, do anything special

### 5.2.3 Member Function Documentation

5.2.3.1 `virtual const char* HashException::what ( ) const throw` `[inline]`, `[virtual]`

virtual fonction from superclass, usefull to get the exception description

### 5.2.4 Member Data Documentation

5.2.4.1 `const char* HashException::_cause` `[private]`

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

- [src/hashtable.hpp](#)

## 5.3 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](#) ()

### Private Attributes

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

#### 5.3.1 Detailed Description

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

Maps a key to a value.

#### 5.3.2 Constructor & Destructor Documentation

5.3.2.1 `template<typename K , typename V > Hashtable< K, V >::Hashtable ( ) [inline]`

array of alveoles Simple constructor

5.3.2.2 `template<typename K , typename V > Hashtable< K, V >::~~Hashtable ( ) [inline]`

Destructor

#### 5.3.3 Member Function Documentation

5.3.3.1 `template<typename K , typename V > bool Hashtable< K, V >::contains ( const K & key ) [inline]`

Do table contains key ?

Parameters

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

5.3.3.2 `template<typename K , typename V > V Hashtable< K, V >::get ( const K & key ) [inline]`

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**

<a href="#"><i>HashException</i></a>	threw if key is not in the hashtable
--------------------------------------	--------------------------------------

**5.3.3.3** `template<typename K , typename V > bool Hashtable< K, V >::isEmpty ( ) [inline]`

Tests if this hashtable maps no keys to values.

**Parameters**

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

**5.3.3.4** `template<typename K , typename V > void Hashtable< K, V >::put ( K key, V value ) [inline]`

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

**5.3.3.5** `template<typename K , typename V > void Hashtable< K, V >::remove ( const K & key ) [inline]`

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**

<a href="#"><i>HashException</i></a>	threw if table does not contain key
--------------------------------------	-------------------------------------

**5.3.3.6** `template<typename K , typename V > string Hashtable< K, V >::toString ( ) [inline]`

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.
-----	-------------	--

**5.3.4 Member Data Documentation**

**5.3.4.1** `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](#)

## 5.4 Node< T > Class Template Reference

Defines tree nodes.

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

### Public Member Functions

- [Node](#) (const [Node](#)< T > &other) [\\_tag](#)(other.\_tag)
- [\\_children](#) (other.\_children) [Node](#)< T >(T data)
- [~Node](#) ()
- [Node](#)< T > & [operator=](#) ([Node](#)< T > &other)
- [operator==](#) (const [Node](#)< T > &lhs, const [Node](#)< T > &rhs)
- bool [operator!=](#) (const [Node](#)< T > &lhs, const [Node](#)< T > &rhs)
- bool [isLeaf](#) ()
- int [height](#) ()
- void [append](#) (< T > n\_data)
- void [remove](#) (< T > data)
- string [toString](#) ()

*get a representation of the node*

### Private Attributes

- T [\\_tag](#)  
*letter stored into Knot, the tag*
- list< Knot< T > > [\\_children](#)  
*children of the Knot*

### 5.4.1 Detailed Description

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

Defines tree nodes.

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

### 5.4.2 Constructor & Destructor Documentation

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

Copy constructor

Parameters

<a href="#">in</a>	<a href="#">other</a>	<a href="#">Node</a> to copy
--------------------	-----------------------	------------------------------

5.4.2.2 `template<typename T = char> Node< T >::~~Node ( ) [inline]`

Destructor for [Node](#)

### 5.4.3 Member Function Documentation

5.4.3.1 `template<typename T = char> Node< T >::_children ( other._children )` `[inline]`

Simple constructor

## Parameters

in	<i>data</i>	to store into the <a href="#">Node</a>
----	-------------	--

**5.4.3.2** `template<typename T = char> void Node< T >::append ( < T > n_data ) [inline]`

Hook up a new child to the node

## Parameters

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

**5.4.3.3** `template<typename T = char> int Node< T >::height ( ) [inline]`

The height of the node

## Parameters

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

**5.4.3.4** `template<typename T = char> bool Node< T >::isLeaf ( ) [inline]`

Is the node a leaf ?

## Parameters

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

**5.4.3.5** `template<typename T = char> bool Node< T >::operator!= ( const Node< T > & lhs, const Node< T > & rhs ) [inline]`

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

**5.4.3.6** `template<typename T = char> Node<T>& Node< T >::operator= ( Node< T > & other ) [inline]`

assignment operator overload

## Parameters

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

**5.4.3.7** `template<typename T = char> Node< T >::operator== ( const Node< T > & lhs, const Node< T > & rhs ) [inline]`

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

**5.4.3.8** `template<typename T = char> void Node< T >::remove ( < T > data )` `[inline]`

Remove a leaf from the node

**Parameters**

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

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

get a representation of the node

**5.4.4 Member Data Documentation**

**5.4.4.1** `template<typename T = char> list<Knot<T> > Node< T >::_children` `[private]`

children of the Knot

**5.4.4.2** `template<typename T = char> T Node< T >::_tag` `[private]`

letter stored into Knot, the tag

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

- [src/tree.hpp](#)

**5.5 Tree< T > Class Template Reference**

[Tree](#) is a recursive structure using nodes.

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

**Public Member Functions**

- [Tree](#) (const [Tree](#)< T > &)  
*copy constructor*
- [Tree](#) ()  
*common constructor*
- [~Tree](#) ()  
*destructor*
- [Tree](#)< T > & [operator=](#) ([Tree](#)< T >)  
*assignment operator*
- bool [operator==](#) (const [Tree](#)< T > &, const [Tree](#)< T > &)  
*equal operator*
- bool [operator!=](#) (const [Tree](#)< T > &, const [Tree](#)< T > &)  
*ne operator*



- bool `contains` (T)  
*Is the element in the tree ?*
- int `count` (T)  
*count among of appearances of a particular element*
- int `height` ()  
*the height of the tree*
- void `add` (T)  
*add an element in the tree*
- void `remove` (T)  
*remove an element from the tree*
- T[] `elements` ()  
*get the whole list of elements in the tree*

### Private Attributes

- `Node< T > _root`

#### 5.5.1 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.

#### 5.5.2 Constructor & Destructor Documentation

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

copy constructor

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

common constructor

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

destructor

#### 5.5.3 Member Function Documentation

5.5.3.1 `template<typename T = string> void Tree< T >::add ( T )`

add an element in the tree

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

Is the element in the tree ?

**5.5.3.3** `template<typename T = string> int Tree< T >::count ( T )`

count among of appearances of a particular element

**5.5.3.4** `template<typename T = string> T [] Tree< T >::elements ( )`

get the whole list of elements in the tree

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

the height of the tree

**5.5.3.6** `template<typename T = string> bool Tree< T >::operator!= ( const Tree< T > & , const Tree< T > & )`

ne operator

**5.5.3.7** `template<typename T = string> Tree<T>& Tree< T >::operator= ( Tree< T > )`

assignment operator

**5.5.3.8** `template<typename T = string> bool Tree< T >::operator== ( const Tree< T > & , const Tree< T > & )`

equal operator

**5.5.3.9** `template<typename T = string> void Tree< T >::remove ( T )`

remove an element from the tree

## 5.5.4 Member Data Documentation

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

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

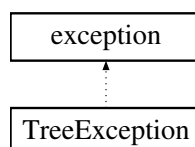
- [src/tree.hpp](#)

## 5.6 TreeException Class Reference

exception class for trees

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

Inheritance diagram for TreeException:



## Public Member Functions

- [TreeException](#) (char \*cause)
- virtual [~BagException](#) () throw ()
- virtual const char \* [what](#) () const throw ()

## Private Attributes

- char \* [\\_cause](#)

### 5.6.1 Detailed Description

exception class for trees

Usefull to manage errors and the unforeseen

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 [TreeException::TreeException](#) ( char \* *cause* ) `[inline]`

store exception description constructor called then TreeExceptions are threw

Parameters

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

#### 5.6.2.2 [virtual TreeException::~~BagException](#) ( ) throw ) `[inline], [virtual]`

destructor currently, do anything special

### 5.6.3 Member Function Documentation

#### 5.6.3.1 [virtual const char\\* TreeException::what](#) ( ) const throw ) `[inline], [virtual]`

virtual fonction from superclass, usefull to get the exception description

### 5.6.4 Member Data Documentation

#### 5.6.4.1 [char\\* TreeException::\\_cause](#) `[private]`

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

- [src/tree.hpp](#)



## Chapter 6

# File Documentation

### 6.1 README.md File Reference

### 6.2 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](#) 0  
*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)

#### 6.2.1 Detailed Description

#### 6.2.2 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 !!

### 6.2.3 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.

### 6.2.4 File informations

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

### 6.2.5 Macro Definition Documentation

#### 6.2.5.1 #define ARRAYSIZE 10

macro to define size of hash arrays

#### 6.2.5.2 #define END 0

macro to define end of alveole chains

### 6.2.6 Function Documentation

#### 6.2.6.1 template<typename K > unsigned computehash ( K element )

Fonction you must define

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 } }

## 6.3 src/tree.hpp File Reference

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

### Classes

- class [TreeException](#)  
*exception class for trees*
- class [Node< T >](#)  
*Defines tree nodes.*

- class `Tree< T >`  
*Tree is a recursive structure using nodes.*

### 6.3.1 Detailed Description

### 6.3.2 File description

Implémentation d'un arbre pour stocker des mots. Chaque noeud stocke une lettre.

### 6.3.3 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.

### 6.3.4 File informations

\$Date\$ 2014/03/27 \$Rev\$ 0.1 \$Author\$ Benjamin Sientzoff \$URL\$ <http://www.github.com/blasterbug>

# Index

- ~Alveole
  - Alveole, [10](#)
- ~BagException
  - TreeException, [21](#)
- ~HashException
  - HashException, [12](#)
- ~Hashtable
  - Hashtable, [13](#)
- ~Node
  - Node, [15](#)
- ~Tree
  - Tree, [19](#)
- \_cause
  - HashException, [12](#)
  - TreeException, [21](#)
- \_children
  - Node, [16](#), [18](#)
- \_key
  - Alveole, [11](#)
- \_next
  - Alveole, [11](#)
- \_root
  - Tree, [20](#)
- \_table
  - Hashtable, [14](#)
- \_tag
  - Node, [18](#)
- \_value
  - Alveole, [11](#)
- ARRAYSIZE
  - hashtable.hpp, [24](#)
- add
  - Tree, [19](#)
- Alveole
  - ~Alveole, [10](#)
  - \_key, [11](#)
  - \_next, [11](#)
  - \_value, [11](#)
  - Alveole, [9](#), [10](#)
  - getKey, [10](#)
  - getNext, [10](#)
  - getValue, [10](#)
  - setNext, [11](#)
  - setValue, [11](#)
  - toString, [11](#)
- Alveole< K, V >, [9](#)
- append
  - Node, [17](#)
- computehash
  - hashtable.hpp, [24](#)
- contains
  - Hashtable, [13](#)
  - Tree, [19](#)
- count
  - Tree, [19](#)
- END
  - hashtable.hpp, [24](#)
- elements
  - Tree, [20](#)
- get
  - Hashtable, [13](#)
- getKey
  - Alveole, [10](#)
- getNext
  - Alveole, [10](#)
- getValue
  - Alveole, [10](#)
- HashException, [11](#)
  - ~HashException, [12](#)
  - \_cause, [12](#)
  - HashException, [12](#)
  - HashException, [12](#)
  - what, [12](#)
- Hashtable
  - ~Hashtable, [13](#)
  - \_table, [14](#)
  - contains, [13](#)
  - get, [13](#)
  - Hashtable, [13](#)
  - isEmpty, [14](#)
  - put, [14](#)
  - remove, [14](#)
  - toString, [14](#)
- Hashtable< K, V >, [13](#)
- hashtable.hpp
  - ARRAYSIZE, [24](#)
  - computehash, [24](#)
  - END, [24](#)
- height
  - Node, [17](#)
  - Tree, [20](#)
- isEmpty
  - Hashtable, [14](#)
- isLeaf



- Node, [17](#)
- Node
  - ~Node, [15](#)
  - \_children, [16](#), [18](#)
  - \_tag, [18](#)
  - append, [17](#)
  - height, [17](#)
  - isLeaf, [17](#)
  - Node, [15](#)
  - operator=, [17](#)
  - operator==, [17](#)
  - remove, [18](#)
  - toString, [18](#)
- Node< T >, [15](#)
- operator=
  - Node, [17](#)
  - Tree, [20](#)
- operator==
  - Node, [17](#)
  - Tree, [20](#)
- put
  - Hashtable, [14](#)
- README.md, [23](#)
- remove
  - Hashtable, [14](#)
  - Node, [18](#)
  - Tree, [20](#)
- setNext
  - Alveole, [11](#)
- setValue
  - Alveole, [11](#)
- src/hashtable.hpp, [23](#)
- src/tree.hpp, [24](#)
- toString
  - Alveole, [11](#)
  - Hashtable, [14](#)
  - Node, [18](#)
- Tree
  - ~Tree, [19](#)
  - \_root, [20](#)
  - add, [19](#)
  - contains, [19](#)
  - count, [19](#)
  - elements, [20](#)
  - height, [20](#)
  - operator=, [20](#)
  - operator==, [20](#)
  - remove, [20](#)
  - Tree, [19](#)
- Tree< T >, [18](#)
- TreeException, [20](#)
  - ~BagException, [21](#)
  - \_cause, [21](#)
  - TreeException, [21](#)
- TreeException, [21](#)
- what, [21](#)
- what
  - HashException, [12](#)
  - TreeException, [21](#)