

HashTable

0.1

Generated by Doxygen 1.8.6

Thu Mar 27 2014 23:22:20

Contents

1	glossygloss	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	Alveole Class Reference	9
5.1.1	Detailed Description	9
5.1.2	Constructor & Destructor Documentation	9
5.1.2.1	Alveole	9
5.1.2.2	Alveole	9
5.1.2.3	Alveole	10
5.1.3	Member Function Documentation	10
5.1.3.1	_next	10
5.1.3.2	_next	10
5.1.3.3	_value	10
5.1.3.4	c_value	10
5.1.3.5	getKey	10
5.1.3.6	getValue	10
5.1.3.7	isQueue	10
5.1.3.8	setValue	10
5.2	HashException Class Reference	11
5.2.1	Detailed Description	11
5.2.2	Constructor & Destructor Documentation	11
5.2.2.1	HashException	11
5.2.2.2	~BagException	11
5.2.3	Member Function Documentation	11

5.2.3.1	what	11
5.3	Knot Class Reference	12
5.3.1	Detailed Description	12
5.3.2	Constructor & Destructor Documentation	12
5.3.2.1	Knot	12
5.3.2.2	~Knot	12
5.3.3	Member Function Documentation	12
5.3.3.1	_children	12
5.3.3.2	append	12
5.3.3.3	height	13
5.3.3.4	isLeaf	13
5.3.3.5	operator!=	13
5.3.3.6	operator=	13
5.3.3.7	operator==	13
5.3.3.8	remove	13
5.3.3.9	toString	14
5.4	Tree Class Reference	14
5.4.1	Detailed Description	14
5.4.2	Constructor & Destructor Documentation	15
5.4.2.1	Tree	15
5.4.2.2	Tree	15
5.4.2.3	~Tree	15
5.4.3	Member Function Documentation	15
5.4.3.1	add	15
5.4.3.2	contains	15
5.4.3.3	count	15
5.4.3.4	elements	15
5.4.3.5	height	15
5.4.3.6	operator!=	15
5.4.3.7	operator=	15
5.4.3.8	operator==	15
5.4.3.9	remove	16
6	File Documentation	17
6.1	HashException.hpp File Reference	17
6.1.1	Detailed Description	17
6.1.2	File description	17
6.1.3	Copyright	17
6.1.4	File informations	17
6.2	HashTable.hpp File Reference	17

6.2.1	Detailed Description	18
6.2.2	File description	18
6.2.3	Copyright	18
6.2.4	File informations	18
6.2.5	Macro Definition Documentation	18
6.2.5.1	END	18
6.3	README.md File Reference	18
6.4	tree.hpp File Reference	18
Index		20

Chapter 1

glossygloss

Glossygloss est un petit programme écrit en C++ permettant de stocker dans un dictionnaire un mot associé à une valeur.

Usefull : http://fr.wikibooks.org/wiki/Programmation_C%2B%2B

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Alveole	9
exception	
HashException	11
Knot	12
Knot< T >	12
Tree	14

Chapter 3

Class Index

3.1 Class List

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

Alveole	9
HashException	11
Knot		
	Class for knots of a tree	12
Tree		
	Class for the tree, use Knot	14

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

HashException.hpp	17
HashTable.hpp	17
tree.hpp	18

Chapter 5

Class Documentation

5.1 Alveole Class Reference

```
#include <HashTable.hpp>
```

Public Member Functions

- [Alveole](#) (const [Alveole](#)< K, V > &other)
- [Alveole](#) (K key, V value) [_key](#)(key)
- [c_value](#) (value)
- [_next](#) (END)
- [Alveole](#) (K key, V value, [Alveole](#)< K, V > *next) [_key](#)(key)
- [_value](#) (value)
- [_next](#) (next)
- bool [isQueue](#) ()
- K [getKey](#) ()
- V [getValue](#) ()
- void [setValue](#) (V n_value)

5.1.1 Detailed Description

[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 [Alveole::Alveole](#) ([const Alveole](#)< K, V > & *other*) [\[inline\]](#)

Copy constructor

Parameters

in	other	the alveole to copy
--------------------	-----------------------	---------------------

5.1.2.2 [Alveole::Alveole](#) ([K key](#), [V value](#))

Pair constructor

Parameters

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

5.1.2.3 Alveole::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

5.1.3 Member Function Documentation

5.1.3.1 Alveole::_next (END) [inline]

5.1.3.2 Alveole::_next (next) [inline]

5.1.3.3 Alveole::_value (value)

5.1.3.4 Alveole::c_value (value)

5.1.3.5 K Alveole::getKey () [inline]

Get the key of an alveole

Parameters

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

5.1.3.6 V Alveole::getValue () [inline]

Get the value stored into an alveole

Parameters

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

5.1.3.7 bool Alveole::isQueue () [inline]

Does alveole have next ?

Parameters

out	<i>true</i>	if elements coming next, else false
-----	-------------	-------------------------------------

5.1.3.8 void Alveole::setValue (V *n_value*) [inline]

Set the value stored into an alveole

Parameters

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

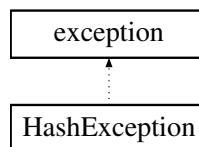
The documentation for this class was generated from the following file:

- [HashTable.hpp](#)

5.2 HashException Class Reference

```
#include <HashException.hpp>
```

Inheritance diagram for HashException:



Public Member Functions

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

5.2.1 Detailed Description

Exception class to manage hashtable errors

5.2.2 Constructor & Destructor Documentation

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

constructor called then HashExceptions are threw

Parameters

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

5.2.2.2 virtual HashException::~~BagException () 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

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

- [HashException.hpp](#)

5.3 Knot Class Reference

class for knots of a tree

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

Public Member Functions

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

get a representation of the knot

5.3.1 Detailed Description

class for knots of a tree

5.3.2 Constructor & Destructor Documentation

5.3.2.1 [Knot::Knot](#) (const [Knot](#)< T > & *other*)

Copy constructor

Parameters

in	<i>other</i>	Knot to copy
--------------------	--------------	------------------------------

5.3.2.2 [Knot::~Knot](#) () [\[inline\]](#)

Destructor for knot

5.3.3 Member Function Documentation

5.3.3.1 [Knot::_children](#) (other._*children*) [\[inline\]](#)

Simple constructor

Parameters

in	<i>data</i>	to store into the knot
--------------------	-------------	------------------------

5.3.3.2 void [Knot::append](#) (< T > *n_data*) [\[inline\]](#)

Hook up a new child to the knot

Parameters

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

5.3.3.3 int Knot::height () [inline]

The height of the knot

Parameters

out	<i>height</i>	of the knot
-----	---------------	-------------

5.3.3.4 bool Knot::isLeaf () [inline]

Is the knot a leaf ?

Parameters

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

5.3.3.5 bool Knot::operator!= (const Knot< T > & lhs, const Knot< T > & rhs) [inline]

inequality operator

Parameters

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

5.3.3.6 Knot<T>& Knot::operator= (Knot< T > & other) [inline]

assignment operator overload

Parameters

in	<i>other</i>	knot to assign
out	<i>assigned</i>	knot

5.3.3.7 Knot::operator== (const Knot< T > & lhs, const Knot< T > & rhs) [inline]

equality operator

Parameters

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

5.3.3.8 void Knot::remove (< T > data) [inline]

Remove a leaf from the knot

Parameters

<code>in</code>	<code>data</code>	data of the knot's tag to remove
-----------------	-------------------	----------------------------------

5.3.3.9 string Knot::toString ()

get a representation of the knot

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

- [tree.hpp](#)

5.4 Tree Class Reference

class for the tree, use [Knot](#)

```
#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

5.4.1 Detailed Description

class for the tree, use [Knot](#)

5.4.2 Constructor & Destructor Documentation

5.4.2.1 `Tree::Tree (const Tree< T > &)`

copy constructor

5.4.2.2 `Tree::Tree ()`

common constructor

5.4.2.3 `Tree::~~Tree ()`

destructor

5.4.3 Member Function Documentation

5.4.3.1 `void Tree::add (T)`

add an element in the tree

5.4.3.2 `bool Tree::contains (T)`

Is the element in the tree ?

5.4.3.3 `int Tree::count (T)`

count among of appearances of a particular element

5.4.3.4 `T [] Tree::elements ()`

get the whole list of elements in the tree

5.4.3.5 `int Tree::height ()`

the height of the tree

5.4.3.6 `bool Tree::operator!= (const Tree< T > &, const Tree< T > &)`

ne operator

5.4.3.7 `Tree<T>& Tree::operator= (Tree< T >)`

assignment operator

5.4.3.8 `bool Tree::operator== (const Tree< T > &, const Tree< T > &)`

equal operator

5.4.3.9 void Tree::remove (T)

remove an element from the tree

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

- [tree.hpp](#)

Chapter 6

File Documentation

6.1 HashException.hpp File Reference

```
#include <string>
```

Classes

- class [HashException](#)

6.1.1 Detailed Description

6.1.2 File description

Exception class for hash classes.

6.1.3 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.

6.1.4 File informations

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

6.2 HashTable.hpp File Reference

```
#include <string>
#include "HashException.hpp"
```

Classes

- class [Alveole](#)

Macros

- `#define` [END](#) 0

6.2.1 Detailed Description

6.2.2 File description

data structure to store pair 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\$ 0.1 \$Author\$ Benjamin Sientzoff \$URL\$ <http://www.github.com/blasterbug>

6.2.5 Macro Definition Documentation

6.2.5.1 `#define` END 0

6.3 README.md File Reference

6.4 tree.hpp File Reference

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

Classes

- class [Knot](#)
class for knots of a tree

- class [Tree](#)

class for the tree, use [Knot](#)

Index

- ~BagException
 - HashException, 11
- ~Knot
 - Knot, 12
- ~Tree
 - Tree, 15
- _children
 - Knot, 12
- _next
 - Alveole, 10
- _value
 - Alveole, 10
- add
 - Tree, 15
- Alveole, 9
 - _next, 10
 - _value, 10
 - Alveole, 9, 10
 - c_value, 10
 - getKey, 10
 - getValue, 10
 - isQueue, 10
 - setValue, 10
- append
 - Knot, 12
- c_value
 - Alveole, 10
- contains
 - Tree, 15
- count
 - Tree, 15
- END
 - HashTable.hpp, 18
- elements
 - Tree, 15
- getKey
 - Alveole, 10
- getValue
 - Alveole, 10
- HashException, 11
 - ~BagException, 11
 - HashException, 11
 - HashException, 11
 - what, 11
- HashException.hpp, 17
- HashTable.hpp, 17
 - END, 18
- height
 - Knot, 13
 - Tree, 15
- isLeaf
 - Knot, 13
- isQueue
 - Alveole, 10
- Knot, 12
 - ~Knot, 12
 - _children, 12
 - append, 12
 - height, 13
 - isLeaf, 13
 - Knot, 12
 - operator=, 13
 - operator==, 13
 - remove, 13
 - toString, 14
- operator=
 - Knot, 13
 - Tree, 15
- operator==
 - Knot, 13
 - Tree, 15
- README.md, 18
- remove
 - Knot, 13
 - Tree, 15
- setValue
 - Alveole, 10
- toString
 - Knot, 14
- Tree, 14
 - ~Tree, 15
 - add, 15
 - contains, 15
 - count, 15
 - elements, 15
 - height, 15
 - operator=, 15
 - operator==, 15
 - remove, 15
 - Tree, 15
- tree.hpp, 18

what

 HashException, [11](#)