Glossygloss

0.2

Generated by Doxygen 1.8.6

Thu Apr 3 2014 19:35:47

Contents

Chapter 1

Main Page

Glossygloss is set of classes to use several data structures as Tree and Hashtable. 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 ?

Copyright

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

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*)

next aveole Copy constructor

Parameters

in	*other*	the alveole to copy

5.1.2.2 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

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

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*)

Complex constructor

Parameters

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

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

Destructor for Alveole

5.1.3 Member Function Documentation

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

Get the key of an alveole

Parameters

out	*key*	key stored into the alveole

5.1.3.2 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

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

Get the value stored into an alveole

Parameters

out	*value*	value of the alveole

5.1.3.4 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

5.1.3.5 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

5.1.3.6 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

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 Dictionnaire< V > Class Template Reference

`#include <dictionnaire.hpp>`

Public Member Functions

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

- void ajouterMot (string mot, V v)
- void associerMot (string mot, V v)
- void supprimerMot (string mot)
- V valeurAssociee (string mot)

Private Attributes

- Hashtable< string, V > dico

5.2.1 Constructor & Destructor Documentation

5.2.1.1 template<typename V > Dictionnaire< V >::Dictionnaire ( )

Constructeur de la classe Dictionnaire

5.2.1.2 template<typename V > Dictionnaire< V >::~Dictionnaire ( )

Destructeur de la classe Dictionnaire

5.2.2 Member Function Documentation

5.2.2.1 template<typename V > void Dictionnaire< V >::ajouterMot (string *mot,* V *v*)

Fonction qui ajoute un mot non présent dans le dictionnaire

Parameters

in	*mot,le*	mot à ajouter
in	*v,la*	valeur associée

5.2.2.2 template<typename V > void Dictionnaire< V >::associerMot (string *mot,* V *v*)

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

Parameters

in	*mot,le*	mot à modifier
in	*v,la*	valeur à modifier

Exceptions

lève	une exception si le mot n'est pas présent

5.2.2.3 template<typename V > bool Dictionnaire< V >::contientMot (string *mot*)

Fonction qui renvoie vrai le mot est présent dans le Dictionnaire

Parameters

in	*mot,le*	mot à tester

out	*bool,vrai*	si présent, faux sinon.

5.2.2.4 template<typename V > void Dictionnaire< V >::supprimerMot (string *mot*)

Fonction qui supprime un mot présent dans le dictionnaire

Parameters

in	*mot,le*	mot à supprimer

Exceptions

lève	une exception si le mot n'est pas présent

5.2.2.5 template<typename V > V Dictionnaire< V >::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

5.2.3 Member Data Documentation

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

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

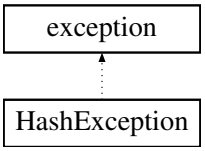
- src/dictionnaire/dictionnaire.hpp

5.3 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

5.3.1 Detailed Description

Exception class to manage Hashtable errors.

5.3.2 Constructor & Destructor Documentation

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

store exception description constructor called then HashExceptions are threw

Parameters

in	*cause*	description of exception origin

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

destructor currently, do anything special

5.3.3 Member Function Documentation

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

virtual fonction from superclass, usefull to get the exception description

5.3.4 Member Data Documentation

5.3.4.1 const char* HashException::_cause [private]

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

- src/hashtable.hpp

5.4 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

5.4.1 Detailed Description

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

Maps a key to a value.

5.4.2 Constructor & Destructor Documentation

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

array of alveoles Simple constructor

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

Destructor

5.4.3 Member Function Documentation

5.4.3.1 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

5.4.3.2 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

HashException	threw if key is not in the hashtable

5.4.3.3 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;

5.4.3.4 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 maped value to the key

Parameters

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

5.4.3.5 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	*key*	Key of the pair to delete

Exceptions

HashException	threw if table does not contain key

5.4.3.6 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	*desc*	a string representation of this hashtable.

5.4.4 Member Data Documentation

5.4.4.1 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

5.5 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

- T _tag

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

  children of the Node

5.5.1 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

5.5.2 Constructor & Destructor Documentation

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

Copy constructor

Parameters

in	*other*	Node to copy

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

Simple constructor

Parameters

in	*data*	to store into the Node

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

Destructor for Node

5.5.3 Member Function Documentation

5.5.3.1 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

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

Do the tag is element or one of his children ?

Parameters

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

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

What is the tag of the Node ?

Parameters

out	*tag*	The tag of the node

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

The height of the node

Parameters

out	*hgt*	height of the node

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

Is the node a leaf ?

Parameters

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

5.5.3.6 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

5.5.3.7 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

5.5.3.8 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

5.5.3.9 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

5.5.3.10 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)

5.5.4 Member Data Documentation

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

children of the Node

5.5.4.2 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

5.6 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)

Private Attributes

- Node< T > _root

5.6.1 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.

5.6.2 Constructor & Destructor Documentation

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

First node of the tree Default constructor

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

Copy constructor

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

Common constructor, create an tree

Parameters

in	*element*	Root of the tree

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

Destructor, destroy the whole tree

5.6.3 Member Function Documentation

5.6.3.1 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.

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

The height of the tree

Parameters

out	*hgt*	Height of the tree

5.6.3.3 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

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

Remove an element from the tree

Parameters

in	*data*	Element to remove

5.6.4 Member Data Documentation

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

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

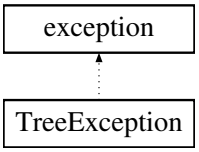
- src/tree.hpp

5.7 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

5.7.1 Detailed Description

exception class for trees

Usefull to manage errors and the unforeseen

5.7.2 Constructor & Destructor Documentation

5.7.2.1 TreeException::TreeException (char * *cause*)

store exception description constructor called then TreeExceptions are threw

Parameters

in	*cause*	description of exception origin

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

destructor currently, do anything special

5.7.3 Member Function Documentation

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

virtual fonction from superclass, usefull to get the exception description

5.7.4 Member Data Documentation

5.7.4.1 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/application.cpp File Reference

```
#include <functional>
#include <iostream>
#include <fstream>
#include "dictionnaire/dictionnaire.hpp"
```

Functions

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

6.2.1 Function Documentation

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

6.2.1.2 int main ( )

6.3 src/dictionnaire/dictionnaire.hpp File Reference

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

Classes

- class Dictionnaire< V >

6.3.1 Detailed Description

6.3.2 File description

Classe Test utilisant les deux implémentations du dictionnaire

6.3.3 Copyright

6.3.4 File informations

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

6.3.5 File description

Dictionnaire utilisant une hashtable

6.3.6 Copyright

6.3.7 File informations

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

6.4 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 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)

6.4.1 Detailed Description

6.4.2 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 !!

6.4.3 Copyright

6.4.4 File informations

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

6.4.5 Macro Definition Documentation

6.4.5.1 #define ARRAYSIZE 10

macro to define size of hash arrays

6.4.5.2 #define END 0

macro to define end of alveole chains

6.4.6 Function Documentation

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

Fonction you must define

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 } }

6.5 src/test_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)

6.5.1 Detailed Description

6.5.2 File description

File to test tree.cpp classes

6.5.3 Copyright

6.5.4 File informations

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

6.5.5 Macro Definition Documentation

6.5.5.1 #define K string

6.5.6 Function Documentation

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

6.6 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.

6.6.1 Detailed Description

6.6.2 File description

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

6.6.3 Copyright

6.6.4 File informations

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

Index