Glossygloss

0.2

Generated by Doxygen 1.8.6

Tue Apr 1 2014 19:25:40

Contents

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

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*) [inline]

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*) [inline]

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 ( ) [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

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 ( ) [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

out	*key*	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

out	*ptr*	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	*value*	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 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*) [inline]

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 ( ) [inline]

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:

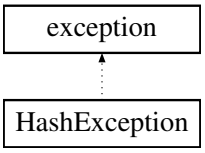
- src/hashtable.hpp

5.2 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

in	*cause*	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	*key*	key to find
out	*bool*	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 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

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

Tests if this hashtable maps no keys to values.

Parameters

out	*bool*	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 maped value to the key

Parameters

in	*key*	key of the pair
in	*value*	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	*key*	Key of the pair to delete

Exceptions

HashException	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	*desc*	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

in	*other*	Node 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	*data*	to store into the Node

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	*n_data*	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	*hgt*	height of the node

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

Is the node a leaf ?

Parameters

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

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

assignment operator overload

Parameters

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

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

Remove a leaf from the node

Parameters

in	*data*	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:

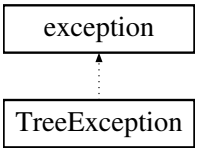
- src/tree.hpp

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

in	*cause*	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

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	*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 } }

6.3 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

6.3.4 File informations

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

Index