Data Structures Course Project

Generated by Doxygen 1.8.3.1

Mon Apr 22 2013 20:57:04

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 ArrayList< T > Class Template Reference

`#include <ArrayList.h>`

Classes

- class Iterator

Public Member Functions

- ArrayList ()
- ~ArrayList ()
- ArrayList & operator= (const ArrayList &x)
- ArrayList (const ArrayList &x)
- bool add (const T &e)
- void add (int index, const T &element)
- void clear ()
- bool contains (const T &e) const
- const T & get (int index) const
- bool isEmpty () const
- void remove (int index)
- bool remove (const T &e)
- void set (int index, const T &element)
- int size () const
- Iterator iterator ()

3.1.1 Detailed Description

template<class T>class ArrayList< T >

The ArrayList is just like vector in C++. You should know that "capacity" here doesn't mean how many elements are now in this list, where it means the length of the array of your internal implemention

The iterator iterates in the order of the elements being loaded into this list

3.1.2 Constructor & Destructor Documentation

3.1.2.1 template<class T > ArrayList< T >::ArrayList ( ) [inline]

TODO Constructs an empty array list.

3.1.2.2 template<class T > ArrayList< T >::~ArrayList ( ) [inline]

TODO Destructor

3.1.2.3 template<class T > ArrayList< T >::ArrayList (const ArrayList< T > & *x*) [inline]

TODO Copy-constructor

3.1.3 Member Function Documentation

3.1.3.1 template<class T > bool ArrayList< T >::add (const T & *e*) [inline]

TODO Appends the specified element to the end of this list.

3.1.3.2 template<class T > void ArrayList< T >::add (int *index,* const T & *element*) [inline]

TODO Inserts the specified element to the specified position in this list. The range of index parameter is [0, size], where index=0 means inserting to the head, and index=size means appending to the end.

Exceptions

IndexOutOfBound	

3.1.3.3 template<class T > void ArrayList< T >::clear ( ) [inline]

TODO Removes all of the elements from this list.

3.1.3.4 template<class T > bool ArrayList< T >::contains (const T & *e*) const [inline]

TODO Returns true if this list contains the specified element.

3.1.3.5 template<class T > const T& ArrayList< T >::get (int *index*) const [inline]

TODO Returns a const reference to the element at the specified position in this list. The index is zero-based, with range [0, size).

Exceptions

IndexOutOfBound	

3.1.3.6 template<class T > bool ArrayList< T >::isEmpty ( ) const [inline]

TODO Returns true if this list contains no elements.

3.1.3.7 template<class T > Iterator ArrayList< T >::iterator ( ) [inline]

TODO Returns an iterator over the elements in this list.

3.1.3.8 template<class T > ArrayList& ArrayList< T >::operator= (const ArrayList< T > & *x*) [inline]

TODO Assignment operator

3.1.3.9 template<class T > void ArrayList< T >::remove (int *index*) [inline]

TODO Removes the element at the specified position in this list. The index is zero-based, with range [0, size).

Exceptions

IndexOutOfBound	

3.1.3.10 template<class T > bool ArrayList< T >::remove (const T & *e*) [inline]

TODO Removes the first occurrence of the specified element from this list, if it is present. Returns true if it is present in the list, otherwise false.

3.1.3.11 template<class T > void ArrayList< T >::set (int *index,* const T & *element*) [inline]

TODO Replaces the element at the specified position in this list with the specified element. The index is zero-based, with range [0, size).

Exceptions

IndexOutOfBound	

3.1.3.12 template<class T > int ArrayList< T >::size ( ) const [inline]

TODO Returns the number of elements in this list.

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

- ArrayList.h

3.2 ElementNotExist Class Reference

Public Member Functions

- **ElementNotExist** (std::string msg)
- std::string **getMessage** () const

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

- ElementNotExist.h

3.3 HashMap< K, V, H >::Entry Class Reference

Public Member Functions

- **Entry** (K k, V v)
- K **getKey** ()
- V **getValue** ()

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

- HashMap.h

3.4 TreeMap< K, V >::Entry Class Reference

Public Member Functions

- **Entry** (K k, V v)
- K **getKey** ()
- V **getValue** ()

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

- TreeMap.h

3.5 HashMap< K, V, H > Class Template Reference

```
#include <HashMap.h>
```

Classes

- class Entry
- class Iterator

Public Member Functions

- HashMap ()
- ~HashMap ()
- HashMap & operator= (const HashMap &x)
- HashMap (const HashMap &x)
- Iterator iterator () const
- void clear ()
- bool containsKey (const K &key) const
- bool containsValue (const V &value) const
- const V & get (const K &key) const
- bool isEmpty () const
- void put (const K &key, const V &value)
- void remove (const K &key)
- int size () const

3.5.1 Detailed Description

template<class K, class V, class H>class HashMap< K, V, H >

HashMap is a map implemented by hashing. Also, the 'capacity' here means the number of buckets in your internal implemention, not the current number of the elements.

Template argument H are used to specify the hash function. H should be a class with a static function named "hashCode", which takes a parameter of type K and returns a value of type int. For example, the following class

```
class Hashint {
public:
    static int hashCode(int obj) {
        return obj;
    }
};
```

specifies an hash function for integers. Then we can define:

```
HashMap<int, int, Hashint> hash;
```

Hash function passed to this class should observe the following rule: if two keys are equal (which means key1 == key2), then the hash code of them should be the same. However, it is not generally required that the hash function should work in the other direction: if the hash code of two keys are equal, the two keys could be different.

Note that the correctness of HashMap should not rely on the choice of hash function. This is to say that, even the given hash function always returns the same hash code for all keys (thus causing a serious collision), methods of HashMap should still function correctly, though the performance will be poor in this case.

The order of iteration could be arbitary in HashMap. But it should be guaranteed that each (key, value) pair be iterated exactly once.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 template<class K, class V, class H> HashMap< K, V, H >::HashMap ( ) [inline]

TODO Constructs an empty hash map.

3.5.2.2 template<class K, class V, class H> HashMap< K, V, H >::~HashMap ( ) [inline]

TODO Destructor

3.5.2.3 template<class K, class V, class H> HashMap< K, V, H >::HashMap (const HashMap< K, V, H > & *x*) [inline]

TODO Copy-constructor

3.5.3 Member Function Documentation

3.5.3.1 template<class K, class V, class H> void HashMap< K, V, H >::clear ( ) [inline]

TODO Removes all of the mappings from this map.

3.5.3.2 template<class K, class V, class H> bool HashMap< K, V, H >::containsKey (const K & *key*) const [inline]

TODO Returns true if this map contains a mapping for the specified key.

3.5.3.3 template<class K, class V, class H> bool HashMap< K, V, H >::containsValue (const V & *value*) const [inline]

TODO Returns true if this map maps one or more keys to the specified value.

3.5.3.4 template<class K, class V, class H> const V& HashMap< K, V, H >::get (const K & *key*) const [inline]

TODO Returns a const reference to the value to which the specified key is mapped. If the key is not present in this map, this function should throw ElementNotExist exception.

Exceptions

ElementNotExist	

3.5.3.5 template<class K, class V, class H> bool HashMap< K, V, H >::isEmpty ( ) const [inline]

TODO Returns true if this map contains no key-value mappings.

3.5.3.6 template<class K, class V, class H> Iterator HashMap< K, V, H >::iterator ( ) const [inline]

TODO Returns an iterator over the elements in this map.

3.5.3.7 template<class K, class V, class H> HashMap& HashMap< K, V, H >::operator= (const HashMap< K, V, H > & *x*) [inline]

TODO Assignment operator

3.5.3.8 template<class K, class V, class H> void HashMap< K, V, H >::put (const K & *key*, const V & *value*) [inline]

TODO Associates the specified value with the specified key in this map.

3.5.3.9 template<class K, class V, class H> void HashMap< K, V, H >::remove (const K & *key*) [inline]

TODO Removes the mapping for the specified key from this map if present. If there is no mapping for the specified key, throws ElementNotExist exception.

Exceptions

ElementNotExist	

3.5.3.10 template<class K, class V, class H> int HashMap< K, V, H >::size ( ) const [inline]

TODO Returns the number of key-value mappings in this map.

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

- HashMap.h

3.6 IndexOutOfBound Class Reference

Public Member Functions

- **IndexOutOfBound** (std::string msg)
- std::string **getMessage** () const

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

- IndexOutOfBound.h

3.7 TreeMap< K, V >::Iterator Class Reference

Public Member Functions

- bool hasNext ()
- const Entry & next ()

3.7.1 Member Function Documentation

3.7.1.1 template<class K , class V > bool TreeMap< K, V >::Iterator::hasNext ( ) [inline]

TODO Returns true if the iteration has more elements.

3.7.1.2 template<class K , class V > const Entry& TreeMap< K, V >::Iterator::next ( ) [inline]

TODO Returns the next element in the iteration.

Exceptions

ElementNotExist	exception when hasNext() == false

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

- TreeMap.h

3.8 LinkedList< T >::Iterator Class Reference

Public Member Functions

- bool hasNext ()
- const T & next ()
- void remove ()

3.8.1 Member Function Documentation

3.8.1.1 template<class T> bool LinkedList< T >::Iterator::hasNext ( ) [inline]

TODO Returns true if the iteration has more elements.

3.8.1.2 template<class T> const T& LinkedList< T >::Iterator::next ( ) [inline]

TODO Returns the next element in the iteration.

Exceptions

ElementNotExist	exception when hasNext() == false

3.8.1.3 template<class T> void LinkedList< T >::Iterator::remove ( ) [inline]

TODO Removes from the underlying collection the last element returned by the iterator

Exceptions

ElementNotExist	

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

- LinkedList.h

3.9 ArrayList< T >::Iterator Class Reference

Public Member Functions

- bool hasNext ()
- const T & next ()
- void remove ()

3.9.1 Member Function Documentation

3.9.1.1 template<class T > bool ArrayList< T >::Iterator::hasNext ( ) [inline]

TODO Returns true if the iteration has more elements.

3.9.1.2 template<class T > const T& ArrayList< T >::Iterator::next ( ) [inline]

TODO Returns the next element in the iteration.

Exceptions

ElementNotExist	exception when hasNext() == false

3.9.1.3 template<class T > void ArrayList< T >::Iterator::remove ( ) [inline]

TODO Removes from the underlying collection the last element returned by the iterator

Exceptions

ElementNotExist	

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

- ArrayList.h

3.10 HashMap< K, V, H >::Iterator Class Reference

Public Member Functions

- bool hasNext ()
- const Entry & next ()

3.10.1 Member Function Documentation

3.10.1.1 template<class K, class V, class H> bool HashMap< K, V, H >::Iterator::hasNext ( ) [inline]

TODO Returns true if the iteration has more elements.

3.10.1.2 template<class K, class V, class H> const Entry& HashMap< K, V, H >::Iterator::next ( ) [inline]

TODO Returns the next element in the iteration.

Exceptions

ElementNotExist	exception when hasNext() == false

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

- HashMap.h

3.11 LinkedList< T > Class Template Reference

```
#include <LinkedList.h>
```

Classes

- class Iterator

Public Member Functions

- LinkedList ()
- LinkedList (const LinkedList< T > &c)
- LinkedList< T > & operator= (const LinkedList< T > &c)
- ~LinkedList ()
- bool add (const T &e)
- void addFirst (const T &elem)
- void addLast (const T &elem)
- void add (int index, const T &element)
- void clear ()
- bool contains (const T &e) const
- const T & get (int index) const
- const T & getFirst () const
- const T & getLast () const
- bool isEmpty () const
- void remove (int index)
- bool remove (const T &e)

- void removeFirst ()
- void removeLast ()
- void set (int index, const T &element)
- int size () const
- Iterator iterator ()

3.11.1 Detailed Description

template<class T>class LinkedList< T >

A linked list.

The iterator iterates in the order of the elements being loaded into this list.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 template<class T> LinkedList< T >::LinkedList ( ) [inline]

TODO Constructs an empty linked list

3.11.2.2 template<class T> LinkedList< T >::LinkedList (const LinkedList< T > & *c*) [inline]

TODO Copy constructor

3.11.2.3 template<class T> LinkedList< T >::~LinkedList ( ) [inline]

TODO Desturctor

3.11.3 Member Function Documentation

3.11.3.1 template<class T> bool LinkedList< T >::add (const T & *e*) [inline]

TODO Appends the specified element to the end of this list.

3.11.3.2 template<class T> void LinkedList< T >::add (int *index,* const T & *element*) [inline]

TODO Inserts the specified element to the specified position in this list. The range of index parameter is [0, size], where index=0 means inserting to the head, and index=size means appending to the end.

Exceptions

IndexOutOfBound	

3.11.3.3 template<class T> void LinkedList< T >::addFirst (const T & *elem*) [inline]

TODO Inserts the specified element to the beginning of this list.

3.11.3.4 template<class T> void LinkedList< T >::addLast (const T & *elem*) [inline]

TODO Insert the specified element to the end of this list. Equivalent to add.

3.11.3.5 template<class T> void LinkedList< T >::clear ( ) [inline]

TODO Removes all of the elements from this list.

3.11.3.6 template<class T> bool LinkedList< T >::contains (const T & *e*) const [inline]

TODO Returns true if this list contains the specified element.

3.11.3.7 template<class T> const T& LinkedList< T >::get (int *index*) const [inline]

TODO Returns a const reference to the element at the specified position in this list. The index is zero-based, with range [0, size).

Exceptions

IndexOutOfBound	

3.11.3.8 template<class T> const T& LinkedList< T >::getFirst ( ) const [inline]

TODO Returns a const reference to the first element.

Exceptions

ElementNotExist	

3.11.3.9 template<class T> const T& LinkedList< T >::getLast ( ) const [inline]

TODO Returns a const reference to the last element.

Exceptions

ElementNotExist	

3.11.3.10 template<class T> bool LinkedList< T >::isEmpty ( ) const [inline]

TODO Returns true if this list contains no elements.

3.11.3.11 template<class T> Iterator LinkedList< T >::iterator ( ) [inline]

TODO Returns an iterator over the elements in this list.

3.11.3.12 template<class T> LinkedList<T>& LinkedList< T >::operator= (const LinkedList< T > & *c*) [inline]

TODO Assignment operator

3.11.3.13 template<class T> void LinkedList< T >::remove (int *index*) [inline]

TODO Removes the element at the specified position in this list. The index is zero-based, with range [0, size).

Exceptions

IndexOutOfBound	

3.11.3.14 template<class T> bool LinkedList< T >::remove (const T & *e*) [inline]

TODO Removes the first occurrence of the specified element from this list, if it is present. Returns true if it is present in the list, otherwise false.

3.11.3.15 template<class T> void LinkedList< T >::removeFirst ( ) [inline]

TODO Removes the first element from this list.

Exceptions

ElementNotExist	

3.11.3.16 template<class T> void LinkedList< T >::removeLast ( ) [inline]

TODO Removes the last element from this list.

Exceptions

ElementNotExist	

3.11.3.17 template<class T> void LinkedList< T >::set (int *index,* const T & *element*) [inline]

TODO Replaces the element at the specified position in this list with the specified element. The index is zero-based, with range [0, size).

Exceptions

IndexOutOfBound	

3.11.3.18 template<class T> int LinkedList< T >::size ( ) const [inline]

TODO Returns the number of elements in this list.

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

- LinkedList.h

3.12 TreeMap< K, V > Class Template Reference

`#include <TreeMap.h>`

Classes

- class Entry
- class Iterator

Public Member Functions

- TreeMap ()
- ~TreeMap ()
- TreeMap & operator= (const TreeMap &x)
- TreeMap (const TreeMap &x)
- Iterator iterator () const
- void clear ()
- bool containsKey (const K &key) const
- bool containsValue (const V &value) const
- const V & get (const K &key) const
- bool isEmpty () const
- void put (const K &key, const V &value)
- void remove (const K &key)
- int size () const

3.12.1 Detailed Description

template<class K, class V>class TreeMap< K, V >

TreeMap is the balanced-tree implementation of map. The iterators must iterate through the map in the natural order (operator<) of the key.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 template<class K , class V > TreeMap< K, V >::TreeMap ( ) [inline]

TODO Constructs an empty tree map.

3.12.2.2 template<class K , class V > TreeMap< K, V >::~TreeMap ( ) [inline]

TODO Destructor

3.12.2.3 template<class K , class V > TreeMap< K, V >::TreeMap (const TreeMap< K, V > & *x*) [inline]

TODO Copy-constructor

3.12.3 Member Function Documentation

3.12.3.1 template<class K , class V > void TreeMap< K, V >::clear ( ) [inline]

TODO Removes all of the mappings from this map.

3.12.3.2 template<class K , class V > bool TreeMap< K, V >::containsKey (const K & *key*) const [inline]

TODO Returns true if this map contains a mapping for the specified key.

3.12.3.3 template<class K , class V > bool TreeMap< K, V >::containsValue (const V & *value*) const [inline]

TODO Returns true if this map maps one or more keys to the specified value.

3.12.3.4 template<class K , class V > const V& TreeMap< K, V >::get (const K & *key*) const [inline]

TODO Returns a const reference to the value to which the specified key is mapped. If the key is not present in this map, this function should throw ElementNotExist exception.

Exceptions

ElementNotExist	

3.12.3.5 template<class K , class V > bool TreeMap< K, V >::isEmpty ( ) const [inline]

TODO Returns true if this map contains no key-value mappings.

3.12.3.6 template<class K , class V > Iterator TreeMap< K, V >::iterator ( ) const [inline]

TODO Returns an iterator over the elements in this map.

3.12.3.7 template<class K , class V > TreeMap& TreeMap< K, V >::operator= (const TreeMap< K, V > & *x*) [inline]

TODO Assignment operator

3.12.3.8 template<class K , class V > void TreeMap< K, V >::put (const K & *key,* const V & *value*) [inline]

TODO Associates the specified value with the specified key in this map.

3.12.3.9 template<class K , class V > void TreeMap< K, V >::remove (const K & *key*) [inline]

TODO Removes the mapping for the specified key from this map if present. If there is no mapping for the specified key, throws ElementNotExist exception.

Exceptions

ElementNotExist	

3.12.3.10 template<class K , class V > int TreeMap< K, V >::size ( ) const [inline]

TODO Returns the number of key-value mappings in this map.

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

- TreeMap.h

Chapter 4

File Documentation

4.1 ArrayList.h File Reference

```
#include "IndexOutOfBound.h"
#include "ElementNotExist.h"
```

Classes

- class ArrayList< T >
- class ArrayList< T >::Iterator

4.2 ElementNotExist.h File Reference

```
#include <string>
```

Classes

- class ElementNotExist

4.2.1 Detailed Description

Thrown when an required element does not exist For example, iter.next(); while iter.hasNext() == false

4.3 HashMap.h File Reference

```
#include "ElementNotExist.h"
```

Classes

- class HashMap< K, V, H >
- class HashMap< K, V, H >::Entry
- class HashMap< K, V, H >::Iterator

4.4 IndexOutOfBound.h File Reference

```
#include <string>
```

Classes

- class IndexOutOfBound

4.4.1 Detailed Description

Thrown when an index is out of range For example, list.get(10); while list.size() == 5 raises this exception.

4.5 LinkedList.h File Reference

```
#include "IndexOutOfBound.h"
#include "ElementNotExist.h"
```

Classes

- class LinkedList< T >
- class LinkedList< T >::Iterator

4.6 TreeMap.h File Reference

```
#include "ElementNotExist.h"
```

Classes

- class TreeMap< K, V >
- class TreeMap< K, V >::Entry
- class TreeMap< K, V >::Iterator

Index