[1 HashTable 2](#_Toc415856365)

[1.1 Implementierung 3](#_Toc415856366)

[1.1.1 hashtable.hpp 3](#_Toc415856367)

[1.1.2 main.cpp 8](#_Toc415856368)

[1.1.3 hashtableTests.hpp 9](#_Toc415856369)

[1.1.4 hashTableTests.cpp 9](#_Toc415856370)

[1.2 Tests 13](#_Toc415856371)

# HashTable

Folgend ist die Dokumentation der Aufgabenstellung HashTable angeführt.

Als Vorlage soll das zur Verfügung gestellte Template verwendet werden.

Die Implementierung soll innerhalb des Templates (\*.hpp) erfolgen und nicht in eine \*.cpp Datei ausgelagert werden.

Als Container soll ein *vector* verwendet werden, der wiederum als Werte eine Liste enthält die letztendlich die Werte vom Typ V enthält.

Es soll ein *vector* verwendet werden da dieser es erlaubt mittels *operator[]* auf die Elemente des Containers zuzugreifen. Somit ist es möglich mittels Indexierung über den ermittelten Hash Index auf die *vector* Elemente zuzugreifen.

Falls der *load factor* den maximal erlaubten *load factor* überschreitet soll der *vector* um den Faktor 1.5 vergrößert werden.

Falls der *load factor* den minimal erlaubten *load factor* unterschreitet soll der *vector* um den Faktor 0.5 verkleinert werden, jedoch nur wenn die definierte minimale Kapazität nicht bei der Verkleinerung unterschritten wird. (Sollte 10 sein, steht aber frei)

Für die Behandlung der Kollision soll der Chaining Mechanismus verwendet werden, wobei die Elemente mit demselben Hash Index in der Liste miteinander verkettet werden. Da der *vector* bereits *list* Elemente enthält kann hierbei einfach über *table[idx].push\_front(..)* ein Element hinzugefügt werden. Bevor ein Element hinzugefügt wird muss ermittelt werden ob sich dieses Element nicht bereits in der Chain befindet, da ansonsten Duplikate auftreten würden.

Für die Implementierung von operator== sei folgendes angemerkt.

Zwei Hashtables sind logisch gleich wenn einer der beiden folgenden Bedingungen erfüllt ist:

1. Es handelt sich um dieselbe Instanz
2. Beide Hashtables haben dieselbe Größe und beinhalten dieselben Elemente.

Die beiden Methoden *insert* und *erease* sollen so implementiert werden, sodass keine Aktion ausgeführt wird wenn ein Element entweder bereits vorhanden ist (*insert*) oder nicht vorhanden ist (*erease*). Es sollen keine Fehler gemeldet werden.

Hilfsmethoden sowie die verwendeten Container sollen innerhalb des *private* Bereichs gehalten werden. Zugriffe sollen ausschließlich über, die zur Verfügung gestellten Schnittstellen erfolgen.

## Implementierung

### hashtable.hpp

Folgend ist die Implementierung der Aufgabenstellung angeführt.

#ifndef hashtable\_hpp

#define hashtable\_hpp

#include <cassert>

#include <vector>

#include <list>

#include <iostream>

#include <iterator>

#include <iomanip>

#include <climits>

#include <math.h>

#include <bitset>

#include <algorithm>

#include <sstream>

**using** **namespace** std**;**

template**<**typename V**,** typename H**,** typename C**>**

class hashtable**;**

/\*\*

\* Overloaded output operator

\*/

template**<**typename V**,** typename H**,** typename C**>**

std**::**ostream **&** **operator** **<<(**std**::**ostream **&** os**,** const hashtable**<**V**,** H**,** C**>** **&**ht**)** **{**

for\_each**(**ht**.**table**.**begin**(),** ht**.**table**.**end**(),**

**([&**ht**,** **&**os**](**const list**<**V**>&** l**)** **{**

for\_each**(**l**.**begin**(),** l**.**end**(),** **([&**ht**,** **&**os**](**const V**&** value**)** **{**

os **<<** "hash-key: " **<<** ht**.**calculateHashIdx**(**value**)** **<<** " - " **<<** value **<<** endl **<<** flush**;**

**}));**

**}));**

**return** os**;**

**}** /\* operator << \*/

template**<**typename V**,** typename H**,** typename C**>**

class hashtable **{**

friend std**::**ostream **&** **operator** **<<<**V**,** H**,** C**>(**std**::**ostream **&** os**,**

const hashtable**<**V**,** H**,** C**>** **&**ht**);**

public**:**

**typedef** V value\_type**;**

**typedef** H hash\_function\_type**;**

**typedef** C key\_equal\_function\_type**;**

**typedef** unsigned int size\_t**;**

**typedef** value\_type const **\*** const\_pointer**;**

**typedef** value\_type const **&** const\_reference**;**

**typedef** std**::**ptrdiff\_t difference\_type**;**

**typedef** const\_pointer pointer**;**

**typedef** const\_reference reference**;**

**typedef** std**::**size\_t size\_type**;**

private**:**

size\_t currentCapacity**;**

hash\_function\_type hasher**;**

key\_equal\_function\_type equals**;**

double maxLoadFactor**;**

double minLoadFactor**;**

const int MIN\_CAPACITY **=** 10**;**

int elementCount **=** 0**;**

vector**<**list**<**value\_type**>>** table**;**

/\*\*

\* This method calculates the hash index for the bakced table depending on the current set capacity.

\* @param

\* const V& value: the value to calculates hash index for

\* @return

\* the calculated hash index

\*/

int calculateHashIdx**(**const V**&** value**)** const **{**

**return** hasher**(**value**)** **%** currentCapacity**;**

**}** /\* calculateHashIdx \*/

/\*\*

\* Validates if an rehash of the backed values is necessary.

\*/

void validateForReHash**()** **{**

double currentLoad **=** load\_factor**();**

double factor **=** 0**;**

**if** **((**currentLoad **<** minLoadFactor**)**

**&&** **((**currentCapacity **\*** 0.5**)** **>** MIN\_CAPACITY**))** **{**

factor **=** 0.5**;**

**}** **else** **if** **(**currentLoad **>** maxLoadFactor**)** **{**

factor **=** 1.5**;**

**}** /\* if \*/

**if** **(**factor **!=** 0**)** **{**

rehash**(**currentCapacity **\*** factor**);**

**}** /\* if \*/

**}** /\* validateForReHash \*/

public**:**

/\*\*

\* Default constructor which initializes all members but the buckets.

\*/

hashtable**(**size\_t n\_buckets **=** 10**,** hash\_function\_type hasher **=** std**::**hash**<**V**>(),**

key\_equal\_function\_type equals **=** std**::**equal\_to**<**V**>(),**

double max\_load\_factor **=** 0.8**,** double min\_load\_factor **=** 0.2**)** **:**

currentCapacity**(**n\_buckets**),** hasher**(**hasher**),** equals**(**equals**),** maxLoadFactor**(**

max\_load\_factor**),** minLoadFactor**(**min\_load\_factor**)** **{**

clear**();**

**}** /\* hashtable \*/

/\*\*

\* No dynamic members therefore nothing to do.

\*/

virtual **~**hashtable**()** **{**

// nothing to do

**}** /\* ~hashtable \*/

void clear**()** **{**

elementCount **=** 0**;**

table**.**clear**();**

**if** **(**currentCapacity **<** MIN\_CAPACITY**)** **{**

currentCapacity **=** MIN\_CAPACITY**;**

**}** /\* if \*/

table**.**resize**(**currentCapacity**);**

**}** /\* clear \*/

/\*\*

\* Inserts the value in the given list if this value is not managed by this list. Does nothing otherwise.

\* @param:

\* const V& value: the value to inserted

\*/

void insert**(**const V **&**value**)** **{**

validateForReHash**();**

size\_t idx **=** calculateHashIdx**(**value**);**

**if** **(**find**(**table**[**idx**].**begin**(),** table**[**idx**].**end**(),** value**)**

**==** table**[**idx**].**end**())** **{**

table**[**idx**].**push\_front**(**value**);**

elementCount**++;**

**}** /\* if \*/

**}** /\* insert \*/

/\*\*

\* Deletes the value from the hashtable if this value is managed by this hashtable. Does nothing otherwise.

\* @param:

\* const V& value: the value to be removed from the hastable

\*/

void erase**(**const V **&**value**)** **{**

size\_t idx **=** calculateHashIdx**(**value**);**

list**<**string**>&** valueList **=** table**[**idx**];**

auto it **=** find**(**valueList**.**begin**(),** valueList**.**end**(),** value**);**

**if** **(**it **!=** valueList**.**end**())** **{**

valueList**.**remove**(**value**);**

elementCount**--;**

**}** /\* if \*/

validateForReHash**();**

**}** /\* erase \*/

/\*\*

\* Answers tthe question if this hashtable contains the given value

\* @param

\* const V& value: the values to be searched in the table

\* @return

\* true if this value is managed by this hastable, false otherwise

\*/

bool contains**(**const V **&**value**)** const **{**

size\_t idx **=** calculateHashIdx**(**value**);**

list**<**string**>** valueList **=** table**[**idx**];**

auto it **=** find**(**valueList**.**begin**(),** valueList**.**end**(),** value**);**

**return** **(**it **!=** valueList**.**end**());**

**}** /\* contains \*/

/\*\*

\* Rehashes the hashtable entries and rebuilds the map with the new has indexes

\* @param:

\* the new buckets

\*/

void rehash**(**size\_t newCapacity**)** **{**

currentCapacity **=** newCapacity**;**

vector**<**list**<**value\_type**>>** oldData**(**table**);**

table**.**clear**();**

table**.**resize**(**currentCapacity**);**

for\_each**(**oldData**.**begin**(),** oldData**.**end**(),**

**([this](**const list**<**value\_type**>** values**)** **{**

for\_each**(**values**.**begin**(),** values**.**end**(),** **([this](**const string value**)** **{**

size\_t idx **=** calculateHashIdx**(**value**);**

table**[**idx**].**push\_front**(**value**);**

**}));**

**}));**

**}** /\* rehash \*/

/\*\*

\* Gets the current load factor

\* @return

\* the current load factor

\*/

double load\_factor**()** const **{**

size\_t curSize **=** size**();**

**return** **(**curSize **>** 0**)** **?** **((**double**)** **(**size**())** **/** **(**currentCapacity**))** **:** 0.0**;**

**}** /\* load\_factor \*/

/\*\*

\* Gets the current size.

\* @return

\* the current size

\*/

size\_t size**()** const **{**

**return** elementCount**;**

**}** /\* size \*/

/\*\*

\* Returns the current capacity

\* @return

\* the current capacity

\*/

size\_t capacity**()** **{**

**return** currentCapacity**;**

**}** /\* capacity \*/

/\*\*

\* Answers the question if this hashtable is empty

\* @return

\* true if this instance has no entries, false otherwise

\*/

bool empty**()** const **{**

**return** size**()** **==** 0**;**

**}** /\* empty \*/

/\*\*

\* hashtables are logically equal if the fit one of the following requirements.

\* 1. Same instance

\* 2. Other instance

\* a. Same size

\* b. Contain same elements

\*

\* @param

\* const hashtable& other: the other instance to compare with

\* @return

\* true if the hastables are logically equal, false otherwise

\*/

bool **operator** **==(**const hashtable **&**other**)** const **{**

bool result **=** **false;**

// same instance

**if** **(this** **==** **(&**other**))** **{**

result **=** **true;**

**}**

// Other instance has same size

**else** **if** **(**size**()** **==** other**.**size**())** **{**

int i **=** 0**;**

result **=** **true;**

// check if other contains same elements

**while** **((**result**)** **&&** **(**i **<** other**.**table**.**size**()))** **{**

auto it **=** other**.**table**[**i**].**begin**();**

**while** **((**result**)** **&&** **(**it **!=** other**.**table**[**i**].**end**()))** **{**

result **=** contains**((\***it**));**

it**++;**

**}** /\* while \*/

i**++;**

**}** /\* while \*/

**}** /\* if \*/

**return** result**;**

**}** /\* operator== \*/

**typedef** std**::**iterator**<**std**::**bidirectional\_iterator\_tag**,** value\_type**,**

difference\_type**,** const\_pointer**,** const\_reference**>** iterator\_base**;**

/\*\*

\* The hashtable iterator class

\* NOT WORKING AT ALL !!!!!!!!!!!!!!!!!!!!!!!!!

\*/

class const\_iterator**:** public iterator\_base **{**

public**:**

**typedef** typename iterator\_base**::**difference\_type difference\_type**;**

**typedef** typename iterator\_base**::**iterator\_category iterator\_category**;**

**typedef** typename iterator\_base**::**pointer pointer**;**

**typedef** typename iterator\_base**::**reference reference**;**

**typedef** typename iterator\_base**::**value\_type value\_type**;**

private**:**

int idx **=** 0**;**

const vector**<**list**<**value\_type**>>&** table**;**

list**<**string**>** values**;**

const bool isEnd**;**

const bool reverse**;**

typename list**<**value\_type**>::**const\_iterator listIt**;**

void prepareIterator**()** **{**

**if** **(**isEnd**)** **{**

idx **=** **(**table**.**size**()** **-** 1**);**

values **=** table**.**at**(**idx**);**

listIt **=** values**.**end**();**

**if** **(**reverse**)** **{**

setNextIterator**();**

**}**

**}** **else** **{**

idx **=** 0**;**

values **=** table**.**at**(**idx**);**

listIt **=** values**.**begin**();**

**if** **(!**reverse**)** **{**

setNextIterator**();**

**}**

**}**

**}**

void setNextIterator**()** **{**

**if** **(**isEnd**)** **{**

**while** **((**idx **!=** 0**)** **&&** **(**listIt **==** values**.**begin**()))** **{**

idx**--;**

values **=** table**.**at**(**idx**);**

listIt **=** values**.**end**();**

**}**

**}** **else** **{**

**while** **((**idx **!=** **(**table**.**size**()** **-** 1**))** **&&** **(**listIt **==** values**.**end**()))** **{**

idx**++;**

values **=** table**.**at**(**idx**);**

listIt **=** values**.**begin**();**

**}**

**}**

**}**

public**:**

const\_iterator**(**const vector**<**list**<**value\_type**>>&** table**,**

bool isEnd **=** **false,** bool reverse **=** **false)** **:**

table**(**table**),** isEnd**(**isEnd**),** reverse**(**reverse**)** **{**

prepareIterator**();**

**}**

bool **operator** **==(**const\_iterator const **&** rhs**)** const **{**

**return** **(**idx **==** rhs**.**idx**)** **&&** **(**listIt **==** rhs**.**listIt**);**

**}**

bool **operator** **!=(**const\_iterator const **&** rhs**)** const **{**

**return** **(**idx **!=** rhs**.**idx**)** **&&** **(**listIt **!=** rhs**.**listIt**);**

**}**

reference **operator** **\*()** const **{**

**return** **\***listIt**;**

**}**

pointer **operator** **->()** const **{**

**return** listIt**;**

**}**

const\_iterator **&** **operator** **++()** **{**

// List still has elements

**if** **(**listIt **!=** values**.**end**())** **{**

listIt**++;**

**}**

// End reached therefore start form new

**if** **((**listIt **!=** values**.**end**())** **&&** **(**idx **<** table**.**size**()))** **{**

idx**++;**

values **=** table**[**idx**];**

listIt **=** values**.**begin**();**

setNextIterator**();**

**}**

**return** **(\*this);**

**}**

const\_iterator **&** **operator** **--()** **{**

// List still has elements

**if** **(**listIt **!=** values**.**begin**())** **{**

listIt**--;**

**}**

// list end reached

**if** **((**listIt **!=** values**.**begin**())** **&&** **(**idx **!=** 0**))** **{**

idx**--;**

values **=** table**[**idx**];**

listIt **=** values**.**end**();**

setNextIterator**();**

**}**

**return** **(\*this);**

**}**

const\_iterator **operator** **++(**int**)** **{**

const\_iterator res**(\*this);**

**++(\*this);**

**return** res**;**

**}**

const\_iterator **operator** **--(**int**)** **{**

const\_iterator res**(\*this);**

**--(\*this);**

**return** res**;**

**}**

**};**

**typedef** const\_iterator iterator**;**

const\_iterator begin**()** const **{**

**return** const\_iterator**(**table**);**

**}** /\*begin \*/

const\_iterator end**()** const **{**

**return** const\_iterator**(**table**,** **true);**

**}** /\* end \*/

const\_iterator rbegin**()** const **{**

**return** const\_iterator**(**table**,** **false,** **true);**

**}** /\* rbegin \*/

const\_iterator rend**()** const **{**

**return** const\_iterator**(**table**,** **true,** **true);**

**}** /\* rend \*/

**};**

#endif // hashtable\_hpp

### main.cpp

Folgend ist die Main Source angeführt, die die Tests Funktionen ausführt.

/\*

\* main.cpp

\* This is the main which invokes the tests.

\*

\* Created on: Mar 27, 2015

\* Author: Thomas Herzog

\*/

#include "test/hashtableTests.hpp"

#include "ide\_listener.h"

#include "xml\_listener.h"

#include "cute\_runner.h"

**using** **namespace** std**;**

**using** **namespace** cute**;**

static void runSuite**(**int argc**,** char const **\***argv**[])** **{**

suite setTestSuite**,** bagTestSuite**;**

xml\_file\_opener xmlfile**(**argc**,** argv**);**

xml\_listener**<**ide\_listener**<>** **>** lis**(**xmlfile**.**out**);**

setTestSuite **=** createTestSuite**();**

makeRunner**(**lis**,** argc**,** argv**)(**setTestSuite**,** "AllTests"**);**

**}** /\* runSuite \*/

int main**(**int argc**,** char const **\***argv**[])** **{**

runSuite**(**argc**,** argv**);**

**}** /\* main \*/

### hashtableTests.hpp

Folgend ist die Spezifikation der Tests angeführt.

/\*

\* hashtableTests.hpp

\* This header file specifies the test for the hashtable implementation

\*

\* Created on: Apr 3, 2015

\* Author: Thomas Herzog

\*/

#ifndef HASHTABLETESTS\_HPP\_

#define HASHTABLETESTS\_HPP\_

#include <cute.h>

#include "../hashtable.hpp"

cute**::**suite createTestSuite**();**

//#####################################################

// hashtable tests

//#####################################################

void test\_insert\_new**();**

void test\_insert\_duplicate**();**

void test\_erease\_existing**();**

void test\_erease\_not\_existing**();**

void test\_contains\_true**();**

void test\_contains\_false**();**

void test\_clear\_empty**();**

void test\_clear\_full**();**

void test\_increase\_load\_factor**();**

void test\_decrease\_load\_factor**();**

void test\_equal\_operator\_same\_instance**();**

void test\_equal\_operator\_different\_instance\_true**();**

void test\_equal\_operator\_different\_instance\_false**();**

#endif /\* HASHTABLETESTS\_HPP\_ \*/

### hashTableTests.cpp

Folgend ist die Implementierung der hashtableTests.hpp angeführt.

/\*

\* hashtableTests.cpp

\* This is the implementation of the specification hashtableTests.hpp

\*

\* Created on: Apr 3, 2015

\* Author: Thomas Herzog

\*/

#include "hashtableTests.hpp"

**using** **namespace** std**;**

**using** **namespace** cute**;**

suite createTestSuite**()** **{**

suite testSuite**;**

testSuite**.**push\_back**(**CUTE**(**test\_insert\_new**));**

testSuite**.**push\_back**(**CUTE**(**test\_insert\_duplicate**));**

testSuite**.**push\_back**(**CUTE**(**test\_erease\_existing**));**

testSuite**.**push\_back**(**CUTE**(**test\_erease\_not\_existing**));**

testSuite**.**push\_back**(**CUTE**(**test\_contains\_true**));**

testSuite**.**push\_back**(**CUTE**(**test\_contains\_false**));**

testSuite**.**push\_back**(**CUTE**(**test\_clear\_empty**));**

testSuite**.**push\_back**(**CUTE**(**test\_clear\_full**));**

testSuite**.**push\_back**(**CUTE**(**test\_increase\_load\_factor**));**

testSuite**.**push\_back**(**CUTE**(**test\_decrease\_load\_factor**));**

testSuite**.**push\_back**(**CUTE**(**test\_equal\_operator\_same\_instance**));**

testSuite**.**push\_back**(**CUTE**(**test\_equal\_operator\_different\_instance\_true**));**

testSuite**.**push\_back**(**CUTE**(**test\_equal\_operator\_different\_instance\_false**));**

**return** testSuite**;**

**}** /\* createTestSuite \*/

static void printTable**(**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**,**

string desc**)** **{**

cout **<<** "Hashtable content: " **<<** desc **<<** endl **<<** "------------------------"

**<<** endl **<<** ht **<<** endl **<<** "------------------------" **<<** endl**;**

**}** /\* printTable \*/

//#####################################################

// hashtable tests

//#####################################################

void test\_insert\_new**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

printTable**(**ht**,** "before insert"**);**

ht**.**insert**(**string**(**"Hello"**));**

ht**.**insert**(**string**(**"you"**));**

ht**.**insert**(**string**(**"C++"**));**

printTable**(**ht**,** "after insert"**);**

ASSERT\_EQUAL**(**3**,** ht**.**size**());**

**}** /\* test\_insert\_new \*/

void test\_insert\_duplicate**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

printTable**(**ht**,** "before insert"**);**

ht**.**insert**(**string**(**"Hello"**));**

ht**.**insert**(**string**(**"Hello"**));**

ht**.**insert**(**string**(**"you"**));**

ht**.**insert**(**string**(**"you"**));**

ht**.**insert**(**string**(**"C++"**));**

ht**.**insert**(**string**(**"C++"**));**

printTable**(**ht**,** "after insert"**);**

ASSERT\_EQUAL**(**3**,** ht**.**size**());**

**}** /\* test\_insert\_duplicate \*/

void test\_erease\_existing**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

string s**(**"C++"**);**

ht**.**insert**(**string**(**"Hello"**));**

ht**.**insert**(**string**(**"you"**));**

ht**.**insert**(**s**);**

printTable**(**ht**,** "before erease"**);**

ht**.**erase**(**s**);**

printTable**(**ht**,** "after erease"**);**

ASSERT\_EQUAL**(**2**,** ht**.**size**());**

**}** /\* test\_erease\_existing \*/

void test\_erease\_not\_existing**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

string s**(**"not-existing"**);**

ht**.**insert**(**string**(**"Hello"**));**

ht**.**insert**(**string**(**"you"**));**

ht**.**insert**(**string**(**"C++"**));**

printTable**(**ht**,** "before erease"**);**

ht**.**erase**(**s**);**

printTable**(**ht**,** "after erease"**);**

ASSERT\_EQUAL**(**3**,** ht**.**size**());**

**}** /\* test\_erease\_not\_existing \*/

void test\_contains\_true**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

string s**(**"C++"**);**

ht**.**insert**(**string**(**"Hello"**));**

ht**.**insert**(**string**(**"you"**));**

ht**.**insert**(**s**);**

printTable**(**ht**,** ""**);**

ASSERT\_EQUAL**(true,** ht**.**contains**(**s**));**

**}** /\* test\_contains\_true \*/

void test\_contains\_false**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

string s**(**"not-existing"**);**

ht**.**insert**(**string**(**"Hello"**));**

ht**.**insert**(**string**(**"you"**));**

ht**.**insert**(**string**(**"C++"**));**

printTable**(**ht**,** ""**);**

ASSERT\_EQUAL**(false,** ht**.**contains**(**s**));**

**}** /\* test\_contains\_false \*/

void test\_clear\_empty**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

printTable**(**ht**,** "before clear"**);**

ht**.**clear**();**

printTable**(**ht**,** "after clear"**);**

ASSERT\_EQUAL**(**0**,** ht**.**size**());**

**}** /\* test\_clear\_empty \*/

void test\_clear\_full**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

ht**.**insert**(**string**(**"Hello"**));**

ht**.**insert**(**string**(**"you"**));**

ht**.**insert**(**string**(**"C++"**));**

printTable**(**ht**,** "before clear"**);**

ht**.**clear**();**

printTable**(**ht**,** "after clear"**);**

ASSERT\_EQUAL**(**0**,** ht**.**size**());**

**}** /\* test\_clear\_full \*/

void test\_increase\_load\_factor**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

**for** **(**int i **=** 0**;** i **<** 20**;** i**++)** **{**

stringstream ss**;**

ss **<<** "Element\_" **<<** i**;**

ht**.**insert**(**ss**.**str**());**

**}**

ASSERT\_EQUAL**(**33**,** ht**.**capacity**());**

**}** /\* test\_increase\_load\_factor \*/

void test\_decrease\_load\_factor**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**30**);**

**for** **(**int i **=** 0**;** i **<** 20**;** i**++)** **{**

stringstream ss**;**

ss **<<** "Element\_" **<<** i**;**

ht**.**insert**(**ss**.**str**());**

**}**

**for** **(**int i **=** 0**;** i **<** 20**;** **++**i**)** **{**

stringstream ss**;**

ss **<<** "Element\_" **<<** i**;**

ht**.**erase**(**ss**.**str**());**

**}**

ASSERT\_EQUAL**(**16**,** ht**.**capacity**());**

**}** /\* test\_decrease\_load\_factor \*/

void test\_equal\_operator\_same\_instance**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht**(**10**);**

ht**.**insert**(**string**(**"Hello"**));**

ASSERT\_EQUAL**(true,** ht **==** ht**);**

**}** /\* test\_equal\_operator\_same\_instance \*/

void test\_equal\_operator\_different\_instance\_true**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht1**(**10**);**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht2**(**10**);**

ht1**.**insert**(**string**(**"Hello"**));**

ht2**.**insert**(**string**(**"Hello"**));**

ASSERT\_EQUAL**(true,** ht1 **==** ht2**);**

**}** /\* test\_equal\_operator\_different\_instance\_true \*/

void test\_equal\_operator\_different\_instance\_false**()** **{**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht1**(**10**);**

hashtable**<**string**,** std**::**hash**<**string**>,** std**::**equal\_to**<**string**>>** ht2**(**10**);**

ht1**.**insert**(**string**(**"Hello"**));**

ht2**.**insert**(**string**(**"You too"**));**

ASSERT\_EQUAL**(false,** ht1 **==** ht2**);**

**}** /\* test\_equal\_operator\_different\_instance\_false\*/

## Tests

Folgend sind die Tests angeführt.





