# Allgemein

Folgend ist die allgemeine Dokumentation angeführt, die beide Aufgabenstellungen betrifft.

Auch bei dieser Übung soll wieder das Testframework CUTE, BOOST verwendet werden um die Implementierung der Tests zu erleichtern und die zur Verfügung gestellten ASSERT Methoden Gebrauch machen zu können.

Auch diesmal soll das Projekt so aufbereitet werden, sodass alle Abhängigkeiten vorhanden sind und die Tests einfach gestartet werden können, ohne zusätzlichen Aufwand.

## Implementierung

Folgen sind die allgemeinen Implementierungen angeführt.

### TestsuiteRunner.cpp

Folgend ist die Implementierung des Testsuite Runner angeführt, der alle Tests sammelt und die Tests durchführt.

/\*

\* main.cpp

\* This is the main for all of the test suites.

\*

\* Created on: Jan 17, 2015

\* Author: Thomas Herzog

\*/

#include "SetTest.h"

#include "BagTest.h"

#include "ide\_listener.h"

#include "xml\_listener.h"

#include "cute\_runner.h"

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

**using** **namespace** ML**;**

**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 **=** makeSetTestSuite**();**

bagTestSuite **=** makeBagTestSuite**();**

setTestSuite**.**insert**(**setTestSuite**.**end**(),** bagTestSuite**.**begin**(),**

bagTestSuite**.**end**());**

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

WriteMetaInfo**();**

**}**

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

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

**}**

# Objektmengen (Sets)

Folgend ist die Dokumentation für die Aufgabenstellung Set angeführt.

## Lösungsidee

Da bereits in der Übung eine Liste implementiert wurde und diese sich nur durch die Eigenschaft dass sie Duplikate erlaubt von einem Set unterscheidet, soll diese Implementierung wiederverwendet werden, wobei eine Prüfung auf Duplikate erfolgen soll.

Wird versucht Duplikate einzufügen soll diese verhindert werden und eine dementsprechende Meldung auf die Konsole ausgegeben werden.

Da von der bereits implementierten Klasse List abgeleitet werden soll, sind auch keine Änderungen an den Klassen der MiniLib notwendig.

Da die Klasse List bereits eine Iterator Implementierung zur Verfügung stellt und diese alle notwendigen Operationen zur Verfügung stellt, mit den verwendeten Node Typen arbeiten kann, sowie mit dessen Variationen, gibt es keine Notwendigkeit einen neuen Iterator für den Set Container zu implementieren. Sollte dies vielleicht in weiterer Folge erforderlich sein, so kann leicht ein eigener Iterator hinzugefügt werden und die Methode NewIterator der Klasse List überschrieben werden.

Bei der Verwendung des Iterators soll darauf geachtet werden das gegen die Superklasse gearbeitet wird und nicht gegen die eigene Iterator Implementierung, es sei denn man benötigt zusätzliche Funktionalität, die nicht in der Klasse Iterator spezifiziert ist.

Da nur die Prüfungen auf Duplikate erfolgen müssen sollte die Implementierung des Set Containers trivial sein.

Die Mengenoperationen sollen so implementiert werden, dass diese sich wie in der Mathematik spezifiziert verhalten. Lediglich die Spezifikation welches Set die Source und welches Set den Target darstellt wird im Folgenden nach der Erklärung der möglichen Implementierungen der Mengenoperationen definiert.

Die Methoden Intersect(Schniottmenge), Union(Vereinigung) und Complement(A ohne B) könnten wie folgt implementiert werden.

1. void target->Intersect(Set\* source)  
   Die Instanz über die die Methode aufgerufen wird spiegelt das Resultat der Operation wieder
2. void source->Intersect(Set\* target)  
   Die übergebene Set Instanz spiegelt das Resultat der Operation wieder.
3. Set\* util->(Set\* source, Set\* target)  
   Es wird ein neues Set erstellt, welches das Resultat der Operation wiederspiegelt.

Ich würde die Version 3 bevorzugen, jedoch folgt diese Version nicht der Anforderung.

Daher soll Version 1 implementiert werden.

## Implementierung

Folgend sind die Implementierungen für die erste Aufgabenstellung angeführt.

### List.h

Folgend ist die Spezifikation des List Containers angeführt.

/\*

\* List.h

\* This is the implementation of the List container specification.

\*

\* Created on: Jan 17, 2015

\* Author: Thomas Herzog

\*/

#ifndef LIST\_H\_

#define LIST\_H\_

#include <ostream>

#include "MLCollection.h"

#include "MLObject.h"

#include "Node.h"

/\*\*

\* This is a list container which allows to manage nodes in a double connected list with anchor structure.

\* This structure was chosen because we want to avoid special cases.

\* No it is simple to append and prepend values since it is always the same an null values cannot occur.

\*/

class List**:** public ML**::**Collection **{**

protected**:**

/\*\*

\* The anchor holding the list elements

\*/

Node**\*** anchor**;**

/\*\*

\* The size of this list = node count

\*/

int size**;**

/\*\*

\* Finds a value in this list.

\*

\* @param:

\* object: the value wrapped by a managed node

\* @return:

\* the found node, nullptr otherwise

\*/

virtual Node**\*** Find**(**ML**::**Object**\*** object**)** const**;**

public**:**

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

// Constructor and destructor

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

/\*\*

\* This constructor creates an anchor element needed by the double connected list with anchor element.

\* This anchor is initialized properly.

\*/

List**();**

/\*\*

\* This destructor clears the list and deletes the anchor element.

\*/

**~**List**();**

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

// Collection manipulation

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

/\*\*

\* Adds a item to the list.

\* The item is wrapped into a node.

\* Default behavior is to prepend the item to the list.

\*

\* @param:

\* object: the value to be added to the list with a wrapping node instance

\*/

virtual void Add**(**ML**::**Object**\*** object**);**

/\*\*

\* Prepends a value to the start of the list by wrapping it into a node instance.

\*

\* **@param**

\* object: the object to be prepended to the list via a wrapping node instance

\*/

virtual void Prepend**(**ML**::**Object**\*** object**);**

/\*\*

\* Appends a value to the end of the list by wrapping it into a node instance.

\*

\* **@param**

\* object: the object to be appended to the list via wrapping node instance

\*/

virtual void Append**(**ML**::**Object**\*** object**);**

/\*\*

\* Removes the item from the list.

\* The wrapped value is not deleted but the wrapping node is.

\*

\* @param:

\* object: the item to be removed from the list

\* @return:

\* the removed object, nullptr if the object is not managed by this list

\*/

ML**::**Object**\*** Remove**(**ML**::**Object**\*** object**);**

/\*\*

\* Clears the list by removing all nodes from the list.

\* The node instances will be deleted but not the wrapped values.

\* The caller is responsible for the lifecycle of the managed values.

\* The anchor will remain as it is.

\*/

void Clear**();**

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

// Utils

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

/\*\*

\* Answers the question if the given value is managed by this list.

\*

\* @param:

\* object: the value to be checked if managed by this list.

\* @return:

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

\*/

bool Contains**(**ML**::**Object**\*** object**)** const**;**

/\*\*

\* Creates a new iterator which allows to iterate over the list

\* starting from the first node.

\*

\* @return:

\* the iterator for this list

\*/

ML**::**Iterator**\*** NewIterator**()** const**;**

/\*\*

\* Prints this list and its managed nodes to the ostream instance.

\*

\* @param:

\* ostream: the ostream to append the resulting print on

\*/

void Print**(**std**::**ostream **&** os**)** const**;**

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

// Getter and setter

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

/\*\*

\* Returns the current size of this list = node count

\*

\* @return:

\* the current node count

\*/

int Size**()** const**;**

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

// operator

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

/\*\*

\* operator for this implementation which calls the Print Method on the given list.

\*

\* @param:

\* ostream: the ostream to append result print on

\* @param:

\* list: the list to print out

\* @return:

\* the ostream with appended print result

\*/

friend std**::**ostream **&** **operator<<(**std**::**ostream **&** os**,** const List **&** list**);**

**};**

/\*\*

\* This is the list iterator which allows to iterate over a list instance.

\*/

class ListIterator**:** public ML**::**Iterator **{**

/\*\*

\* Crates a new iterator for a list.

\*

\* @return:

\* the iterator instance for this list

\*/

friend ML**::**Iterator**\*** List**::**NewIterator**()** const**;**

private**:**

/\*\*

\* The pointer to current visited node.

\*/

Node**\*** current**;**

/\*\*

\* The anchor node of the list

\*/

Node**\*** anchorNode**;**

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

// Constructor

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

/\*\*

\* Private constructor because it is not allowed to instantiate a list iterator

\* outside the list implementation.

\*

\* @param:

\* anchor: the anchor node

\* @param:

\* start: the start node of the list

\*/

ListIterator**(**Node**\*** anchorNode**,** Node**\*** start**);**

public**:**

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

// Destructor

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

/\*\*

\* Destructor which does nothing because nodes values are managed by the list only.

\*/

**~**ListIterator**();**

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

// Utils

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

/\*\*

\* Returns the next element of the managed list.

\*

\* @return:

\* the next node nullptr otherwise

\*/

ML**::**Object **\***Next**();**

**};**

#endif /\* LIST\_H\_ \*/

### List.cpp

Folgend ist die Implementierung der Spezifikation List.h angeführt.

/\*

\* List.cpp

\* This is the implementation of the List container specification.

\*

\* Created on: Jan 17, 2015

\* Author: Thomas Herzog

\*/

#include "List.h"

#include <string>

#include <cassert>

**using** **namespace** ML**;**

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

List**::**List**()** **:**

size**(**0**)** **{**

Register**(**"List"**,** "Collection"**);**

/\* init anchor peropely \*/

Node**\*** node **=** **new** Node**();**

node**->**next **=** node**;**

node**->**prev **=** node**;**

anchor **=** node**;**

**}** /\* List::List \*/

List**::~**List**()** **{**

Clear**();**

**delete** anchor**;**

**}** /\* List::~List \*/

int List**::**Size**()** const **{**

**return** size**;**

**}** /\* List::Size \*/

void List**::**Add**(**Object**\*** object**)** **{**

Append**(**object**);**

**}** /\* List::Add \*/

bool List**::**Contains**(**Object**\*** object**)** const **{**

**return** Find**(**object**)** **!=** nullptr**;**

**}** /\* List::Contains \*/

Object**\*** List**::**Remove**(**Object**\*** object**)** **{**

Object**\*** value **=** nullptr**;**

Node**\*** foundNode **=** Find**(**object**);**

**if** **(**foundNode **!=** nullptr**)** **{**

foundNode**->**prev**->**next **=** foundNode**->**next**;**

foundNode**->**next**->**prev **=** foundNode**->**prev**;**

value **=** foundNode**->**val**;**

**delete** foundNode**;**

size**--;**

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

**return** value**;**

**}** /\* List::Remove \*/

void List**::**Clear**()** **{**

Node**\*** next **=** nullptr**;**

Node**\*** node **=** anchor**->**next**;**

**while** **(**node **!=** anchor**)** **{**

next **=** node**->**next**;**

**delete** node**;**

node **=** next**;**

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

anchor**->**next **=** anchor**;**

anchor**->**prev **=** anchor**;**

size **=** 0**;**

**}** /\* List::Clear \*/

Iterator**\*** List**::**NewIterator**()** const **{**

**return** **new** ListIterator**(**anchor**,** anchor**->**next**);**

**}** /\* List::NewIterator \*/

Node**\*** List**::**Find**(**Object**\*** object**)** const **{**

Node**\*** node **=** anchor**->**next**;**

**while** **((**node **!=** anchor**)** **&&** **(!**node**->**val**->**IsEqualTo**(**object**)))** **{**

node **=** node**->**next**;**

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

**return** **((**node **!=** anchor**)** **?** node **:** nullptr**);**

**}** /\* List::Find \*/

void List**::**Prepend**(**Object**\*** object**)** **{**

Node**\*** newNode **=** **new** Node**(**object**,** anchor**->**prev**,**

anchor**->**prev**->**next**);**

anchor**->**prev **=** newNode**;**

newNode**->**prev**->**next **=** newNode**;**

size**++;**

**}** /\* List::Prepend \*/

void List**::**Append**(**Object**\*** object**)** **{**

Node**\*** newNode **=** **new** Node**(**object**,** anchor**->**next**->**prev**,**

anchor**->**next**);**

anchor**->**next **=** newNode**;**

newNode**->**next**->**prev **=** newNode**;**

size**++;**

**}** /\* List::Append \*/

void List**::**Print**(**ostream **&** os**)** const **{**

os **<<** "-----------------" **<<** endl **<<** Class**()** **<<** "(" **<<** size **<<** "):" **<<** endl

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

Node**\*** node **=** anchor**->**next**;**

**while** **(**node **!=** anchor**)** **{**

os **<<** node**->**AsString**();**

node **=** node**->**next**;**

**if** **(**node **!=** anchor**)** **{**

os **<<** ", "**;**

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

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

**}** /\* List::Print \*/

ostream**&** **operator<<(**ostream **&** os**,** const List **&** list**)** **{**

list**.**Print**(**os**);**

**return** os**;**

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

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

// Iterator implementation

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

ListIterator**::**ListIterator**(**Node**\*** anchorNode**,** Node**\*** start**)** **:**

anchorNode**(**anchorNode**),** current**(**start**)** **{**

Register**(**"ListIterator"**,** "Iterator"**);**

**}** /\* ListIterator::ListIterator \*/

ListIterator**::~**ListIterator**()** **{**

// do nothing. current Node lifecylce managed by list not iterator

**}** /\* ListIterator::~ListIterator \*/

Object**\*** ListIterator**::**Next**()** **{**

Object**\*** object **=** nullptr**;**

**if** **(**current **!=** anchorNode**)** **{**

object **=** current**->**val**;**

current **=** current**->**next**;**

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

**return** object**;**

**}** /\* ListIterator::Next \*/

### Node.h

Folgend ist die Spezifikation der Container Node angeführt.

/\*

\* Node.h

\* This is the specification of the container Node.

\*

\* Created on: Jan 17, 2015

\* Author: Thomas Herzog

\*/

#ifndef NODE\_H\_

#define NODE\_H\_

#include <string>

#include "MLObject.h"

/\*\*

\* This class is used to wrap a value into a node instance to be able to manage values with a container implementation.

\*/

class Node**:** public ML**::**Object **{**

public**:**

/\*\*

\* The wrapped value

\*/

ML**::**Object**\*** val**;**

/\*\*

\* The pointer to the previous node

\*/

Node**\*** prev**;**

/\*\*

\* The pointer to the next node

\*/

Node**\*** next**;**

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

// Constructor and destructor

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

/\*\*

\* Constructor which realizes null values for not defined formal parameters.

\*

\* @param:

\* val: the to wrap value (default nullptr)

\* @param:

\* prev: the reference to the previous node (default nullptr)

\* @param:

\* next: the reference to the next node (default nullptr)

\*/

explicit Node**(**ML**::**Object**\*** val **=** nullptr**,** Node**\*** prev **=** nullptr**,**

Node**\*** next **=** nullptr**);**

/\*\*

\* This destructor does nothing because values are managed by this node for container usage only.

\* The lifecycle of the values is managed by the caller not the node or the container

\*/

virtual **~**Node**();**

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

// Utils

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

/\*\*

\* Returns a string representation of this node and its value.

\*

\* @return:

\* the string representation of this node

\*/

virtual std**::**string AsString**()** const**;**

**};**

#endif /\* NODE\_H\_ \*/

### Node.cpp

Folgend ist die Implementierung der Spezifikation Node.h angeführt.

/\*

\* Node.cpp

\* This is the implementation of the container Node specification.

\*

\* Created on: Jan 17, 2015

\* Author: Thomas Herzog

\*/

#include "MLCollection.h"

#include "Node.h"

#include <string>

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

**using** **namespace** ML**;**

Node**::**Node**(**Object**\*** val**,** Node**\*** prev**,** Node**\*** next**)** **:**

val**(**val**),** prev**(**prev**),** next**(**next**)** **{**

Register**(**"Node"**,** "Object"**);**

**}** /\* Node::Node \*/

Node**::~**Node**()** **{**

// Nothing to do. Member handled outside this instance

**}** /\* Node::~Node \*/

string Node**::**AsString**()** const **{**

string asString**(**"Node ("**);**

**(**val **!=** nullptr**)** **?**

asString**.**append**(**val**->**AsString**())** **:** asString**.**append**(**"nullptr"**);**

asString**.**append**(**")"**);**

**return** asString**;**

**}** /\* Node::AsString \*/

### Set.h

Folgend ist die Spezifikation des Set Containers angeführt.

/\*

\* Set.h

\* This is the specification of the Set container

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#ifndef SET\_H\_

#define SET\_H\_

#include "List.h"

/\*\*

\* This class implements a set container which disallows duplicate values in the container.

\* It inherits from List and adds check functionality on appending methods to check if the value is already managed.

\*

\* There is no need to define new iterator because would not differ from List implemented one.

\*/

class Set**:** public List **{**

protected**:**

/\*\*

\* Answers the question if this container has already a node which wraps the given value.

\* It prints a message to the console if value is already managed by this set.

\*

\* @param:

\* obj: the value to be checked if already managed by the set

\* @return:

\* true if already managed, false otherwise.

\*/

virtual bool IsAlreadyManaged**(**ML**::**Object**\*** obj**)** const**;**

public**:**

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

// Constructor and destructor

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

/\*\*

\* This constructor does nothing since List provides all necessary functionality.

\*/

Set**();**

/\*\*

\* This destructor does nothing since this set has no dynamic members.

\*/

virtual **~**Set**();**

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

// overwritten for Set behavior

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

/\*\*

\* Overwritten to avoid duplicates.

\* **@see** List::Add(Object\* object)

\*/

virtual void Add**(**ML**::**Object**\*** object**);**

/\*\*

\* Overwritten to avoid duplicates.

\* **@see** List::Append(Object\* object)

\*/

virtual void Append**(**ML**::**Object**\*** object**);**

/\*\*

\* Overwritten to avoid duplicates.

\* **@see** List::Prepend(Object\* object)

\*/

virtual void Prepend**(**ML**::**Object**\*** object**);**

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

// utils

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

/\*\*

\* Builds a intersection between this set and the given one.

\* This set will be modified and will become the intersected set

\*

\* @param:

\* set: the set to intersect with

\*/

virtual void Intersect**(**Set**\*** set**);**

/\*\*

\* Builds a union between this set and the given one.

\* This set will represent the united set.

\*

\* @param:

\* set: the set to unite with this set

\*/

virtual void Union**(**Set**\*** set**);**

/\*\*

\* Builds a complement between this set and the given one.

\* This set will represent the complemented set.

\*

\* @param:

\* set: the set to build the complemented with this set

\*/

virtual void Complement**(**Set**\*** set**);**

**};**

#endif /\* SET\_H\_ \*/

### Set.cpp

Folgend ist die Implementierung der Spezifikation Set.h angeführt.

/\*

\* Set.cpp

\* This is the implementation of the Set container specification.

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#include "Set.h"

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

**using** **namespace** ML**;**

Set**::**Set**()** **:**

List**()** **{**

Register**(**"Set"**,** "List"**);**

// Nothing to do. List initializes backed container

**}** /\* Set::Set \*/

Set**::~**Set**()** **{**

// Nothing to do

**}** /\* Set::~Set \*/

bool Set**::**IsAlreadyManaged**(**Object**\*** object**)** const **{**

Node**\*** node **=** Find**(**object**);**

**if** **(**node **!=** nullptr**)** **{**

cout **<<** "Object already managed by this set !!!" **<<** endl **<<** flush**;**

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

**return** node **!=** nullptr**;**

**}** /\* Set::IsAlreadyManaged \*/

void Set**::**Add**(**ML**::**Object**\*** object**)** **{**

/\* avoid duplicates \*/

**if** **(!**IsAlreadyManaged**(**object**))** **{**

List**::**Add**(**object**);**

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

**}** /\* Set::Add \*/

void Set**::**Append**(**ML**::**Object**\*** object**)** **{**

/\* avoid duplicates \*/

**if** **(!**IsAlreadyManaged**(**object**))** **{**

List**::**Append**(**object**);**

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

**}** /\* Set::Append \*/

void Set**::**Prepend**(**ML**::**Object**\*** object**)** **{**

/\* avoid duplicates \*/

**if** **(!**IsAlreadyManaged**(**object**))** **{**

List**::**Prepend**(**object**);**

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

**}** /\* Set::Prepend \*/

void Set**::**Intersect**(**Set**\*** set**)** **{**

/\* null set not allowed will not change anything \*/

**if** **(**set **==** nullptr**)** **{**

cout **<<** "Given set is not allowed to be null !!!" **<<** endl **<<** flush**;**

**}**

/\* if source is empty this set cannot contain any element \*/

**else** **if** **(**set**->**Size**()** **==** 0**)** **{**

Clear**();**

**}**

/\* find out what to nodes are supposed to be removed \*/

**else** **{**

Iterator**\*** it **=** NewIterator**();**

Object**\*** obj **=** it**->**Next**();**

**while** **(**obj **!=** nullptr**)** **{**

**if** **(**set**->**Find**(**obj**)** **==** nullptr**)** **{**

Remove**(**obj**);**

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

obj **=** it**->**Next**();**

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

**delete** it**;**

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

**}** /\* Set::Intersect \*/

void Set**::**Union**(**Set**\*** set**)** **{**

/\* null set not allowed will not change anything \*/

**if** **(**set **==** nullptr**)** **{**

cout **<<** "Given set is not allowed to be null !!!" **<<** endl **<<** flush**;**

**}**

/\* if source wmpty nothign to do \*/

**else** **if** **(**set**->**Size**()** **>** 0**)** **{**

Iterator**\*** it **=** set**->**NewIterator**();**

Object**\*** obj **=** it**->**Next**();**

**while** **(**obj **!=** nullptr**)** **{**

**if** **(**Find**(**obj**)** **==** nullptr**)** **{**

Append**(**obj**);**

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

obj **=** it**->**Next**();**

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

**delete** it**;**

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

**}** /\* Set::Union \*/

void Set**::**Complement**(**Set**\*** set**)** **{**

/\* null set not allowed will not change anything \*/

**if** **(**set **==** nullptr**)** **{**

cout **<<** "Given set is not allowed to be null !!!" **<<** endl **<<** flush**;**

**}**

/\* if source is empt nothing to do \*/

**else** **if** **(**set**->**Size**()** **>** 0**)** **{**

Iterator**\*** it **=** set**->**NewIterator**();**

Object**\*** obj **=** it**->**Next**();**

**while** **(**obj **!=** nullptr**)** **{**

**if** **(**Find**(**obj**)** **!=** nullptr**)** **{**

Remove**(**obj**);**

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

obj **=** it**->**Next**();**

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

**delete** it**;**

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

**}** /\* Set::Complement \*/

### SetTest.h

Folgend ist die Spezifikation der Test für den Set Container angeführt.

/\*

\* SetTest.h

\* This is the Set Test specification.

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#ifndef SETTEST\_H\_

#define SETTEST\_H\_

#include "cute.h"

#include "../Set.h"

cute**::**suite makeSetTestSuite**();**

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

// Add, Append, Prepend tests

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

void test\_set\_add\_multpile\_values**();**

void test\_set\_append\_multpile\_values**();**

void test\_set\_prepend\_multpile\_values**();**

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

// Remove tests

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

void test\_set\_remove\_not\_found**();**

void test\_set\_remove\_empty**();**

void test\_set\_remove**();**

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

// Intersect tests

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

void test\_set\_intersect\_null\_source**();**

void test\_set\_intersect\_empty\_source**();**

void test\_set\_intersect\_empty\_result**();**

void test\_set\_intersect**();**

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

// union tests

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

void test\_set\_union\_null\_source**();**

void test\_set\_union\_empty\_source**();**

void test\_set\_union\_duplicates**();**

void test\_set\_union**();**

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

// complement tests

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

void test\_set\_complement\_null\_source**();**

void test\_set\_complement\_empty\_source**();**

void test\_set\_complement\_duplicates**();**

void test\_set\_complement**();**

#endif /\* SETTEST\_H\_ \*/

### SetTest.cpp

Folgend ist die Implementierung der Spezifikation SetTest.h angeführt.

/\*

\* SetTest.cpp

\* This is the implementation of the Set Test specification.

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#include "SetTest.h"

#include "MLString.h"

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

**using** **namespace** ML**;**

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

suite makeSetTestSuite**()** **{**

suite setTestSuite**;**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_add\_multpile\_values**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_append\_multpile\_values**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_prepend\_multpile\_values**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_remove\_not\_found**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_remove\_empty**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_remove**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_intersect\_null\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_intersect\_empty\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_intersect\_empty\_result**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_intersect**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_union\_null\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_union\_empty\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_union\_duplicates**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_union**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_complement\_null\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_complement\_empty\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_complement\_duplicates**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_set\_complement**));**

**return** setTestSuite**;**

**}**

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

// Add, Append, Prepend tests

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

void test\_set\_add\_multpile\_values**()** **{**

Set**\*** set **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

set**->**Add**(**val1**);**

set**->**Add**(**val2**);**

set**->**Add**(**val3**);**

set**->**Add**(**val4**);**

/\* Add duplicates \*/

set**->**Add**(**val1**);**

set**->**Add**(**val2**);**

cout **<<** **\***set **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**4**,** set**->**Size**());**

**delete** set**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_set\_append\_multpile\_values**()** **{**

Set**\*** set **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

set**->**Append**(**val1**);**

set**->**Append**(**val2**);**

set**->**Append**(**val3**);**

set**->**Append**(**val4**);**

/\* Append duplicates \*/

set**->**Append**(**val1**);**

set**->**Append**(**val2**);**

cout **<<** **\***set **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**4**,** set**->**Size**());**

**delete** set**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_set\_prepend\_multpile\_values**()** **{**

Set**\*** set **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

set**->**Prepend**(**val1**);**

set**->**Prepend**(**val2**);**

set**->**Prepend**(**val3**);**

set**->**Prepend**(**val4**);**

/\* Prepend duplicates \*/

set**->**Prepend**(**val1**);**

set**->**Prepend**(**val2**);**

cout **<<** **\***set **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**4**,** set**->**Size**());**

**delete** set**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

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

// Remove tests

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

void test\_set\_remove\_not\_found**()** **{**

Set**\*** set **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

set**->**Add**(**val1**);**

set**->**Add**(**val2**);**

cout **<<** "Before remove:" **<<** endl **<<** **\***set **<<** endl **<<** endl **<<** flush**;**

set**->**Remove**(**val3**);**

cout **<<** "After remove:" **<<** endl **<<** **\***set **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** set**->**Size**());**

**delete** set**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**}**

void test\_set\_remove\_empty**()** **{**

Set**\*** set **=** **new** Set**();**

String**\*** val3 **=** **new** String**(**"value 3"**);**

cout **<<** "Before remove:" **<<** endl **<<** **\***set **<<** endl **<<** endl **<<** flush**;**

set**->**Remove**(**val3**);**

cout **<<** "After remove:" **<<** endl **<<** **\***set **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**0**,** set**->**Size**());**

**delete** set**;**

**delete** val3**;**

**}**

void test\_set\_remove**()** **{**

Set**\*** set **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

set**->**Add**(**val1**);**

set**->**Add**(**val2**);**

cout **<<** "Before remove:" **<<** endl **<<** **\***set **<<** endl **<<** endl **<<** flush**;**

set**->**Remove**(**val2**);**

cout **<<** "After remove:" **<<** endl **<<** **\***set **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**1**,** set**->**Size**());**

**delete** set**;**

**delete** val1**;**

**delete** val2**;**

**}**

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

// Intersect

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

void test\_set\_intersect\_null\_source**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

target**->**Intersect**(**nullptr**);**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

**delete** target**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_set\_intersect\_empty\_source**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

Set**\*** source **=** **new** Set**();**

target**->**Intersect**(**source**);**

ASSERT\_EQUAL**(**0**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_set\_intersect\_empty\_result**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

Set**\*** source **=** **new** Set**();**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

source**->**Add**(**val3**);**

source**->**Add**(**val4**);**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Source:" **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

target**->**Intersect**(**source**);**

cout **<<** "After Intersect Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl

**<<** flush**;**

ASSERT\_EQUAL**(**0**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_set\_intersect**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

Set**\*** source **=** **new** Set**();**

String**\*** val3 **=** **new** String**(**"value 1"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

source**->**Add**(**val3**);**

source**->**Add**(**val4**);**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Source:" **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

target**->**Intersect**(**source**);**

cout **<<** "AFter intersect Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl

**<<** flush**;**

ASSERT\_EQUAL**(**1**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

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

// union tests

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

void test\_set\_union\_null\_source**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Union**(**nullptr**);**

cout **<<** "After union Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

**delete** target**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_set\_union\_empty\_source**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

Set**\*** source **=** **new** Set**();**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Source:" **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

target**->**Union**(**source**);**

cout **<<** "After union Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_set\_union\_duplicates**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

Set**\*** source **=** **new** Set**();**

String**\*** val3 **=** **new** String**(**"value 1"**);**

String**\*** val4 **=** **new** String**(**"value 2"**);**

source**->**Add**(**val3**);**

source**->**Add**(**val4**);**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Source:" **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

target**->**Union**(**source**);**

cout **<<** "After union Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_set\_union**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

Set**\*** source **=** **new** Set**();**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

source**->**Add**(**val3**);**

source**->**Add**(**val4**);**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Source:" **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

target**->**Union**(**source**);**

cout **<<** "After intersect Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl

**<<** flush**;**

ASSERT\_EQUAL**(**4**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

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

// complement tests

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

void test\_set\_complement\_null\_source**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Complement**(**nullptr**);**

cout **<<** "After union Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

**delete** target**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_set\_complement\_empty\_source**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

Set**\*** source **=** **new** Set**();**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Complement**(**source**);**

cout **<<** "After union Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_set\_complement\_duplicates**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

Set**\*** source **=** **new** Set**();**

String**\*** val3 **=** **new** String**(**"value 1"**);**

String**\*** val4 **=** **new** String**(**"value 2"**);**

source**->**Add**(**val3**);**

source**->**Add**(**val4**);**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Source:" **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

target**->**Complement**(**source**);**

cout **<<** "After union Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**0**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_set\_complement**()** **{**

Set**\*** target **=** **new** Set**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Add**(**val1**);**

target**->**Add**(**val2**);**

Set**\*** source **=** **new** Set**();**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

source**->**Add**(**val3**);**

source**->**Add**(**val4**);**

cout **<<** "Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Source:" **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

target**->**Complement**(**source**);**

cout **<<** "After union Target:" **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

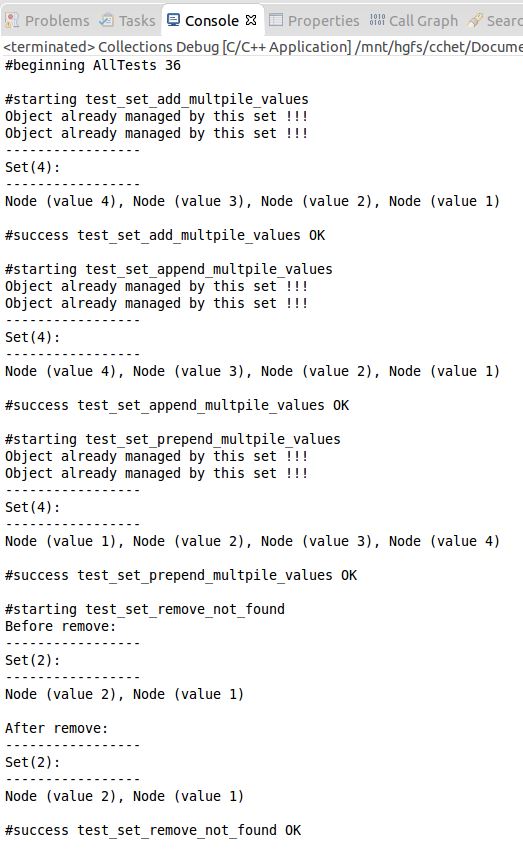
## Tests

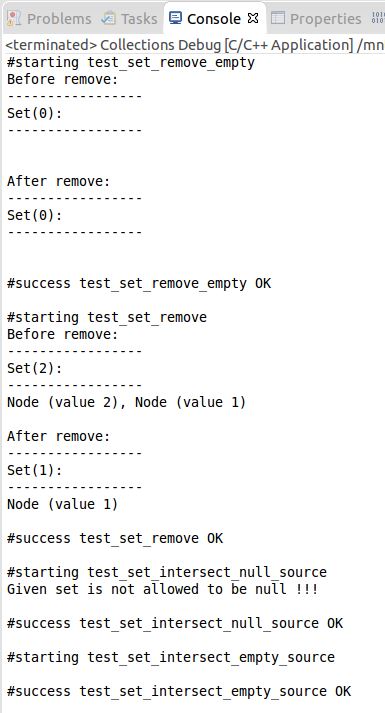
Folgend sind die Tests für die Set Implementierung angeführt.

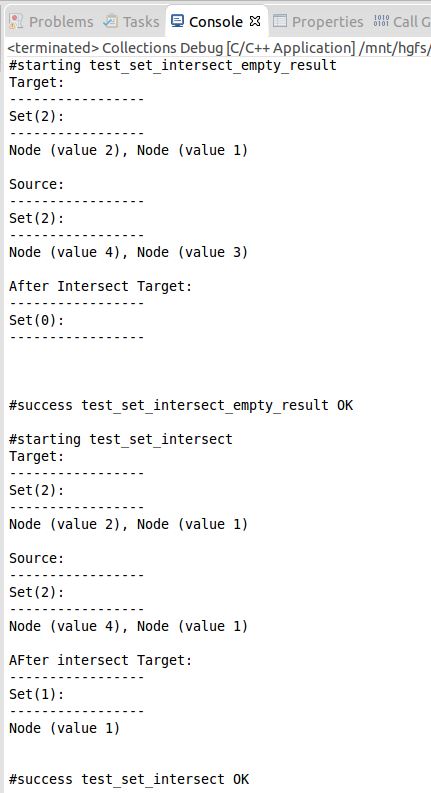
Es wird hierbei davon ausgegangen das die List Implementierung korrekt funktioniert.

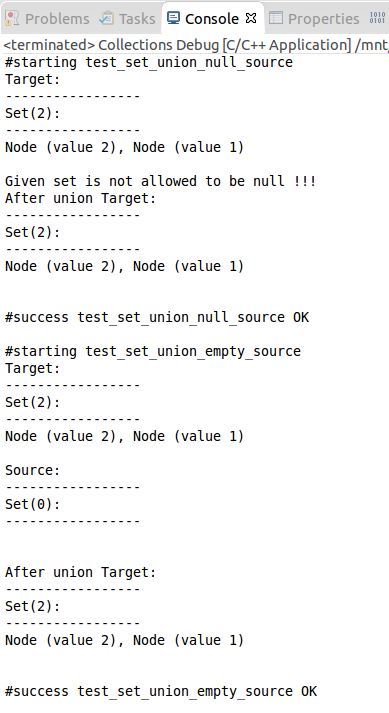
Sie wird in diesen Tests sowieso implizit getestet, bis auf die Möglichkeit Duplikate einzufügen.

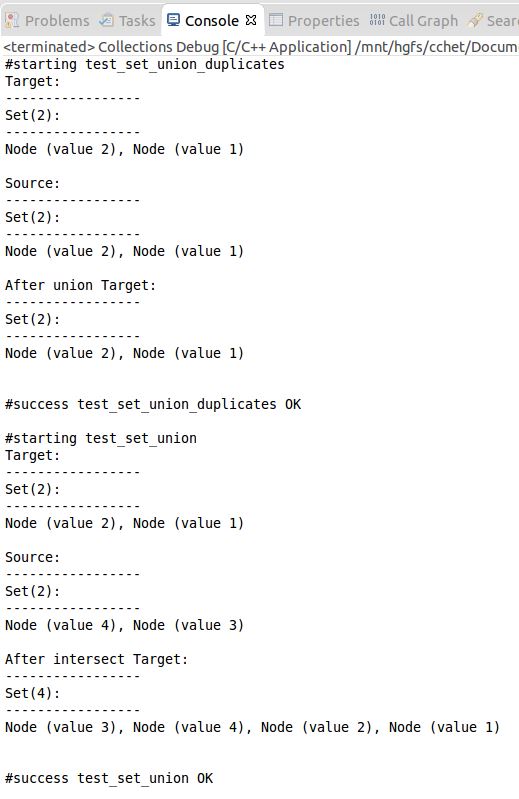
Da alle Tests auf einmal durchlaufen erfolgt die Ausgabe der Meta-Informationen am Ende der Bag Tests.

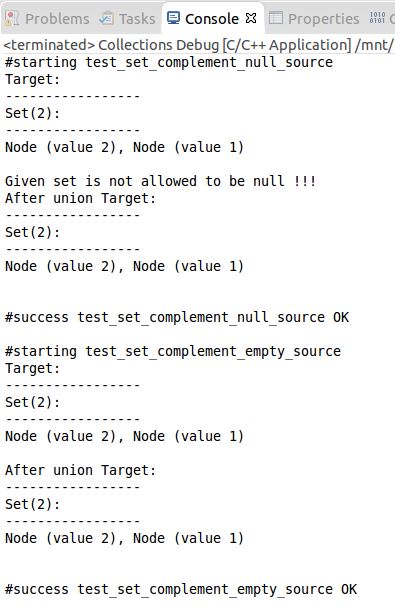


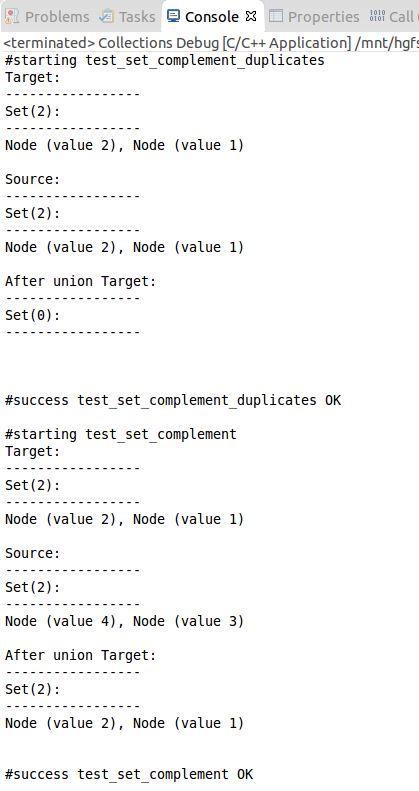












# Objektbehälter (Bag)

Folgend ist die Dokumentation der zweiten Aufgabenstellung angeführt.

## Lösungsidee

Laut Aufgabenstellung soll dieser Container von Set ableiten und dieselben Funktionalitäten zur Verfügung stellen, was an sich nicht in Ordnung ist, da Mengenoperationen auf Mengen angewendet werden sollten und der Bag Container nun mahl keine Menge darstellt, da er Duplikate erlaubt. Daher müssen die Mengenoperationen erneut implementiert und deren Verhalten spezifiziert werden. Dies wird später genauer erläutert.

Um Speicherschonend arbeiten zu können sollen die Duplikate nicht gespeichert sondern lediglich die Anzahl der Werte, die im Container verwaltet werden, gespeichert werden. Dies stellt kein Problem dar, da die Duplikate exakt dem gespeicherten Wert entsprechen und sich von diesen nicht unterschieden. Hierbei verlässt man sich auf die Methode IsEqualTo(…), die true zurückliefert wenn zwei Objekt gleich sind und false wenn nicht.

Daher soll die Klasse BagNode eingeführt werden, die von Node ableitet und sich die Anzahl der Werte im Container merken soll.

Der Bag Container soll aber trotzdem die Anzahl aller gespeicherten Werte halten obwohl diese nicht die tatsächliche Anzahl der verwalteten Knoten wiederspiegelt. Da aber mittels Iterator über die Elemente iteriert wird und dieser unabhängig von der aktuellen Anzahl und der gehaltenen Größe ist, stellt dies kein Problem dar. Der Iterator braucht die gehaltene Size nicht.

Bei einer Iteration über die Elemente soll nur über die aktuell verwalteten Knoten iteriert werden und der Aufrufer ist verantwortlich sich um die Duplikate, welche über die gehaltene Anzahl in der Klasse BagNode definiert ist, zu kümmern. Damit ist gemeint der Aufrufer muss sich im Klaren sein, dass die Duplikate nur über die Anzahl definiert sind und nicht im Container als Knoten vorhanden sind.

### Spezifikation der Mengenoperationen

Im Folgenden wird die aktuelle Instanz über die die Operationen durchgeführt werden als Target und die übergebene Bag Container Instanz als Source bezeichnet.

Um die Performanz zu steigern sollen die Duplikate mittels Arithmetik hinzugefügt oder entfernt werden. Damit ist gemeint sollten Duplikate erkannt werden sollen diese über Arithmetik der Anzahl hinzugefügt oder entfernt werden. Nur der Knoten, der tatsächlich im Container vorhanden ist oder hinzugefügt werden muss, soll wie gehabt in den Container hinzugefügt oder gelöscht werden.

Die Remove und Append Methode würden zwar funktionieren, jedoch wären zu viele Iterationen über die backing List erforderlich, was hierbei nicht notwendig ist.

Das folgend spezifizierte Verhalten der Mengenoperationen betrachtet die Duplikate als wären Sie gültige Elemente einer Menge und ignoriert das sie eigentlich Duplikate sind.

**Schnittmenge:**

1. Knoten im Target, die mehr Duplikate definieren als die Knoten im Source, sollen die Anzahl der Duplikate der Knoten des Source wiederspiegeln.
2. Knoten im Target, die weniger Duplikate definieren als die Knoten im Source, sollen so verbleiben und nicht geändert werden.
3. Knoten die im Target vorhanden sind aber nicht im Source sollen vollständig gelöscht werden.

**Vereinigung:**

1. Knoten die im Source vorhanden sind aber nicht im Target sollen mit Ihren Duplikaten in den Target Container übernommen werden.
2. Knoten die Im Target vorhanden sind aber weniger Duplikate definieren, sollen die Anzahl der Duplikate des Knoten im Source wiederspiegeln.
3. Knoten die im Target vorhanden sind und mehr Duplikate definieren als die Knoten im Source sollen so verbleiben wie sie sind.

**Komplement:**

1. Knoten die im Target und Source vorhanden sind, sollen vollständig gelöscht werden wenn
   1. der Knoten im Target weniger Duplikate definiert als die Knoten im Source
   2. ansonsten sollen die Knoten des Target die   
      Summe: targetNode.count – sourceNode.count als Duplikate definieren.
2. Knoten die im Target vorhanden sind und weniger Duplikate definieren als die Knoten im Source, sollen vollständig entfernt werden.
3. Knoten die nur im Target vorhanden sind sollen unverändert bleiben.

## Implementierung

Folgend sind die Implementierung der zweiten Aufgabenstellung angeführt.

### BagNode.h

Folgend ist die Spezifikation Gab Knoten angeführt.

/\*

\* BagNode.h

\* This is the bag container BagNode specification.

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#ifndef BAGNODE\_H\_

#define BAGNODE\_H\_

#include "List.h"

#include <string>

/\*\*

\* This class can be used in bag implementation which holds information about the count of the duplicates.

\* Therefore that we talk about duplicates we do not need to hold the references or copy the duplicates, the duplicate count provides enough information,

\* since a bagNode instance will provide all necessary information about the node and its duplicates.

\*/

class BagNode**:** public Node **{**

public**:**

/\*\*

\* The count of the duplicate nodes.

\*/

int count**;**

/\*\*

\* Constructor which only sets the count to 1.

\* The rest is done by the parent class

\*/

explicit BagNode**(**ML**::**Object**\*** val **=** nullptr**,** Node**\*** prev **=** nullptr**,** Node**\*** next **=**

nullptr**);**

/\*\*

\* Destructor which does nothing because no dynamic members.

\*/

**~**BagNode**();**

/\*\*

\* Returns the string representation of this instance.

\*

\* @return:

\* the string representation of this node

\*/

virtual std**::**string AsString**()** const**;**

**};**

#endif /\* BAGNODE\_H\_ \*/

### BagNode.cpp

Folgend ist die Implementierung der Spezifikation BagNode.h angeführt.

/\*

\* BagNode.cpp

\* This is the implementation of the bag container BagNode specification.

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#include "BagNode.h"

#include <sstream>

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

**using** **namespace** ML**;**

BagNode**::**BagNode**(**ML**::**Object**\*** val**,** Node**\*** prev**,** Node**\*** next**)** **:**

Node**(**val**,** prev**,** next**),** count**(**1**)** **{**

Register**(**"BagNode"**,** "Node"**);**

**}** /\* BagNode::BagNode \*/

BagNode**::~**BagNode**()** **{**

// nothing to do

**}** /\* BagNode::~BagNode \*/

std**::**string BagNode**::**AsString**()** const **{**

string result**(**"BagNode("**);**

/\* seems std::to\_String not supported by this compiler \*/

stringstream ss**;**

ss **<<** count**;**

**(**val **!=** nullptr**)** **?**

result**.**append**(**val**->**AsString**())** **:** result**.**append**(**"nullptr"**);**

result**.**append**(**", "**).**append**(**ss**.**str**()).**append**(**")"**);**

**return** result**;**

**}** /\* BagNode::AsString \*/

### Bag.h

Folgend ist die Spezifikation des Bag Container angeführt.

/\*

\* Bag.h

\* This is the Bag container specification.

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#ifndef BAG\_H\_

#define BAG\_H\_

#include "Set.h"

#include "BagNode.h"

/\*\*

\* This class realizes a bag container and inherits from the Set class which provides the proper implementation.

\* Since this container will not contain duplicates nodes (duplicates count on BagNode only), we can inherit from Set

\* and not from List.

\* There is no need to create own nodes for the duplicates since the duplicates will not provide more information

\* as the unique instance would do.

\*/

class Bag**:** public Set **{**

protected**:**

/\*\*

\* Overwritten for type safety.

\* @see: Node\* Set::Find(Object\* obj)

\*/

BagNode**\*** Find**(**ML**::**Object**\*** object**)** const**;**

public**:**

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

// Constructor and destructor

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

/\*\*

\* Constructor which does nothing because handled by parent class

\*/

Bag**();**

/\*\*

\* Destructor which does nothing because no dynamic members.

\*/

virtual **~**Bag**();**

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

// overwritten for Bag behavior

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

/\*\*

\* Overwritten to allow duplicates and handle them properly.

\* **@see** Set::Add(Object\* object)

\*/

virtual void Add**(**ML**::**Object**\*** object**);**

/\*\*

\* Overwritten to allow duplicates and handle them properly.

\* **@see** Set::Append(Object\* object)

\*/

virtual void Append**(**ML**::**Object**\*** object**);**

/\*\*

\* Overwritten to allow duplicates and handle them properly.

\* **@see** Set::Prepend(Object\* object)

\*/

virtual void Prepend**(**ML**::**Object**\*** object**);**

/\*\*

\* Deletes one of the duplicates if more than one item is present otherwise parent class handles deletion.

\*

\* @param:

\* object: the object to be deleted

\*/

virtual ML**::**Object**\*** Remove**(**ML**::**Object**\*** object**);**

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

// overwritten for type safety and bag behavior .utils

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

/\*\*

\* Special handling for Bag container, since a set is not allowed to contain duplicates but a bag does

\* we need to handle this intersection differently.

\*

\* 1. If a target node does not exist in the source will be removed completely.

\* 2. If a target node has less duplicates defined than the target node will remain as it is.

\* 3. If a target node has more duplicates defined as the source node than the target node will define the source node defined duplicate count.

\*

\*/

virtual void Intersect**(**Bag**\*** bag**);**

/\*\*

\* Special handling for Bag container, since since a set is not allowed to contain duplicates but a bag does

\* we need to handle this intersection differently.

\*

\* 1. If a node exists in the source but not the target then this node will be appended to this container with the

\* current set duplicate count of the source node

\* 2. If a target node has lees duplicates as a source node increases the duplicate counter to face the source node duplicate count.

\* 3. If a target node has more duplicates defined as the source node will not be modified.

\*/

virtual void Union**(**Bag**\*** bag**);**

/\*\*

\* Special handling for Bag container, since since a set is not allowed to contain duplicates but a bag does

\* we need to handle this intersection differently.

\*

\* 1. If a node exists in the source and target and the target node has more duplicates defined,

\* then the target node will define the sum: targetNode.count - sourceNode.count;

\* 2. If a node exists in the source and target node and the target node has less duplicates defined,

\* then the node will be removed completely

\* 3. If a target node does not exist in the source then this node remains as it is

\*/

virtual void Complement**(**Bag**\*** bag**);**

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

// operator

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

/\*\*

\* The operator which prints this bag instance.

\* The already in List implemented Method Print will be reused since there is no difference in the printed result.

\* The nodes will provide their own AsString method and therefore the nodes will be printed properly.

\*/

friend std**::**ostream **&** **operator<<(**std**::**ostream **&** os**,** const Bag **&** bag**);**

**};**

#endif /\* BAG\_H\_ \*/

### Bag.cpp

Folgend ist die Implementierung der Spezifikation Bag.h angeführt.

/\*

\* Bag.cpp

\* This is the implementation of the bag container specification.

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#include "Bag.h"

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

**using** **namespace** ML**;**

Bag**::**Bag**()** **:**

Set**()** **{**

Register**(**"Bag"**,** "Set"**);**

**}** /\* Bag::Bag \*/

Bag**::~**Bag**()** **{**

// Nothing to do

**}** /\* Bag::~Bag \*/

BagNode**\*** Bag**::**Find**(**Object**\*** object**)** const **{**

/\* Save for brutal cast because only BagNode instances are allowed \*/

**return** **(**BagNode**\*)** Set**::**Find**(**object**);**

**}**/\* Bag::Find \*/

void Bag**::**Add**(**Object**\*** object**)** **{**

Append**(**object**);**

**}** /\* Bag::Add \*/

void Bag**::**Append**(**Object**\*** object**)** **{**

BagNode**\*** node **=** Find**(**object**);**

/\* no duplicates, normal append \*/

**if** **(**node **==** nullptr**)** **{**

BagNode**\*** newNode **=** **new** BagNode**(**object**,** anchor**->**prev**,**

anchor**->**prev**->**next**);**

anchor**->**prev **=** newNode**;**

newNode**->**prev**->**next **=** newNode**;**

**}**

/\* else increase count of duplicates \*/

**else** **{**

node**->**count**++;**

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

size**++;**

**}** /\* Bag::Append \*/

void Bag**::**Prepend**(**Object**\*** object**)** **{**

BagNode**\*** node **=** Find**(**object**);**

/\* no duplicates, normal prepend \*/

**if** **(**node **==** nullptr**)** **{**

BagNode**\*** newNode **=** **new** BagNode**(**object**,** anchor**->**next**->**prev**,**

anchor**->**next**);**

anchor**->**next **=** newNode**;**

newNode**->**next**->**prev **=** newNode**;**

**}**

/\* else increase count of duplicates \*/

**else** **{**

node**->**count**++;**

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

size**++;**

**}** /\* Bag::Prepend \*/

Object**\*** Bag**::**Remove**(**Object**\*** object**)** **{**

BagNode**\*** node **=** Find**(**object**);**

/\* if node found and has duplicates \*/

**if** **((**node **!=** nullptr**)** **&&** **(**node**->**count **>** 1**))** **{**

node**->**count**--;**

size**--;**

**return** object**;**

**}**

/\* else delegate to parent class \*/

**else** **{**

**return** Set**::**Remove**(**object**);**

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

**}** /\* Bag::Remove \*/

void Bag**::**Intersect**(**Bag**\*** bag**)** **{**

**if** **(**bag **==** nullptr**)** **{**

cout **<<** "Given set is not allowed to be null !!!" **<<** endl **<<** flush**;**

**}** **else** **if** **(**bag**->**Size**()** **==** 0**)** **{**

Clear**();**

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

Iterator**\*** it **=** NewIterator**();**

Object**\*** obj **=** it**->**Next**();**

BagNode**\*** bagNode **=** nullptr**;**

BagNode**\*** thisNode **=** nullptr**;**

**while** **(**obj **!=** nullptr**)** **{**

thisNode **=** Find**(**obj**);**

/\* node not present in source bag \*/

**if** **((**bagNode **=** bag**->**Find**(**obj**))** **==** nullptr**)** **{**

/\* remove all nodes with duplicates \*/

size **-=** **(**thisNode**->**count **-** 1**);**

Remove**(**obj**);**

**}**

/\* append as much as common \*/

**else** **if** **(**thisNode**->**count **>** bagNode**->**count**)** **{**

/\* remove too much duplicates \*/

int diff **=** **(**thisNode**->**count **-** bagNode**->**count**);**

thisNode**->**count **=** **(**thisNode**->**count **-** diff**);**

size **-=** diff**;**

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

obj **=** it**->**Next**();**

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

**delete** it**;**

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

**}** /\* Bag::Intersect \*/

void Bag**::**Union**(**Bag**\*** bag**)** **{**

**if** **(**bag **==** nullptr**)** **{**

cout **<<** "Given Bag is not allowed to be null !!!" **<<** endl **<<** flush**;**

**}** **else** **if** **(**bag**->**Size**()** **>** 0**)** **{**

Iterator**\*** it **=** bag**->**NewIterator**();**

Object**\*** obj **=** it**->**Next**();**

BagNode**\*** bagNode **=** nullptr**;**

BagNode**\*** thisNode **=** nullptr**;**

**while** **(**obj **!=** nullptr**)** **{**

bagNode **=** bag**->**Find**(**obj**);**

/\* node found in this bag \*/

**if** **((**thisNode **=** Find**(**obj**))** **==** nullptr**)** **{**

Append**(**obj**);**

Find**(**obj**)->**count **+=** **(**bagNode**->**count **-** 1**);**

size **+=** **(**bagNode**->**count **-** 1**);**

/\* for \*/

**}**

**else** **if** **(**thisNode**->**count **<** bagNode**->**count**)** **{**

int diff **=** **(**bagNode**->**count **-** thisNode**->**count**);**

size**+=**diff**;**

thisNode**->**count**+=**diff**;**

**}**

/\* if \*/

obj **=** it**->**Next**();**

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

**delete** it**;**

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

**}** /\* Bag::Union \*/

void Bag**::**Complement**(**Bag**\*** bag**)** **{**

**if** **(**bag **==** nullptr**)** **{**

cout **<<** "Given set is not allowed to be null !!!" **<<** endl **<<** flush**;**

**}** **else** **if** **(**bag**->**Size**()** **>** 0**)** **{**

Iterator**\*** it **=** bag**->**NewIterator**();**

Object**\*** obj **=** it**->**Next**();**

BagNode**\*** thisNode **=** nullptr**;**

BagNode**\*** bagNode **=** nullptr**;**

**while** **(**obj **!=** nullptr**)** **{**

**if** **((**thisNode **=** Find**(**obj**))** **!=** nullptr**)** **{**

bagNode **=** bag**->**Find**(**obj**);**

/\* Only remove to much duplicates \*/

**if** **(**thisNode**->**count **>** bagNode**->**count**)** **{**

thisNode**-=**bagNode**->**count**;**

size**-=**bagNode**->**count**;**

**}**

/\* if source has more then this bag remove all \*/

**else** **if** **(**bagNode**->**count **>=** thisNode**->**count**)** **{**

size**-=(**thisNode**->**count **-** 1**);**

thisNode**->**count **=** 1**;**

Remove**(**obj**);**

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

**}**

obj **=** it**->**Next**();**

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

**delete** it**;**

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

**}** /\* Bag::Complement \*/

ostream**&** **operator<<(**ostream **&** os**,** const Bag **&** bag**)** **{**

bag**.**Print**(**os**);**

**return** os**;**

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

### BagTest.h

Folgend ist die Spezifikation der Test für den Bag Container angeführt.

/\*

\* BagTest.h

\* This is the Bag Test specification.

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#ifndef BAGTEST\_H\_

#define BAGTEST\_H\_

#include "cute.h"

#include "../Bag.h"

cute**::**suite makeBagTestSuite**();**

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

// Add, Append, Prepend tests

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

void test\_bag\_add\_multpile\_values**();**

void test\_bag\_append\_multpile\_values**();**

void test\_bag\_prepend\_multpile\_values**();**

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

// Remove tests

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

void test\_bag\_remove\_not\_found**();**

void test\_bag\_remove\_empty**();**

void test\_bag\_remove**();**

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

// Intersect tests

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

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

// Intersect tests

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

void test\_bag\_intersect\_null\_source**();**

void test\_bag\_intersect\_empty\_source**();**

void test\_bag\_intersect\_empty\_result**();**

void test\_bag\_intersect**();**

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

// union tests

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

void test\_bag\_union\_null\_source**();**

void test\_bag\_union\_empty\_source**();**

void test\_bag\_union\_nothing\_todo**();**

void test\_bag\_union**();**

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

// complement tests

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

void test\_bag\_complement\_null\_source**();**

void test\_bag\_complement\_empty\_source**();**

void test\_bag\_complement\_empty\_result**();**

void test\_bag\_complement**();**

#endif /\* BAGTEST\_H\_ \*/

### BagTest.cpp

Folgend ist die Implementierung der Spezifikation BagTest.cpp angeführt.

/\*

\* BagTest.cpp

\* This is the implementation of the Bag Test specification.

\*

\* Created on: Jan 18, 2015

\* Author: Thomas Herzog

\*/

#include "BagTest.h"

#include "MLString.h"

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

**using** **namespace** ML**;**

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

suite makeBagTestSuite**()** **{**

suite setTestSuite**;**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_add\_multpile\_values**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_append\_multpile\_values**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_prepend\_multpile\_values**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_remove\_not\_found**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_remove\_empty**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_remove**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_intersect\_null\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_intersect\_empty\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_intersect\_empty\_result**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_intersect**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_union\_null\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_union\_empty\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_union\_nothing\_todo**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_union**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_complement\_null\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_complement\_empty\_source**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_complement\_empty\_result**));**

setTestSuite**.**push\_back**(**CUTE**(**test\_bag\_complement**));**

**return** setTestSuite**;**

**}**

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

// Add, Append, Prepend tests

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

void test\_bag\_add\_multpile\_values**()** **{**

Bag**\*** bag **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

bag**->**Add**(**val1**);**

bag**->**Add**(**val2**);**

bag**->**Add**(**val3**);**

bag**->**Add**(**val4**);**

/\* Add duplicates \*/

bag**->**Add**(**val1**);**

bag**->**Add**(**val2**);**

cout **<<** **\***bag **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**6**,** bag**->**Size**());**

**delete** bag**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_bag\_append\_multpile\_values**()** **{**

Bag**\*** bag **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

bag**->**Append**(**val1**);**

bag**->**Append**(**val2**);**

bag**->**Append**(**val3**);**

bag**->**Append**(**val4**);**

/\* Add duplicates \*/

bag**->**Append**(**val1**);**

bag**->**Append**(**val2**);**

cout **<<** **\***bag **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**6**,** bag**->**Size**());**

**delete** bag**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_bag\_prepend\_multpile\_values**()** **{**

Bag**\*** bag **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

bag**->**Prepend**(**val1**);**

bag**->**Prepend**(**val2**);**

bag**->**Prepend**(**val3**);**

bag**->**Prepend**(**val4**);**

/\* Add duplicates \*/

bag**->**Prepend**(**val1**);**

bag**->**Prepend**(**val2**);**

cout **<<** **\***bag **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**6**,** bag**->**Size**());**

**delete** bag**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

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

// Remove tests

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

void test\_bag\_remove\_not\_found**()** **{**

Bag**\*** bag **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

bag**->**Prepend**(**val1**);**

bag**->**Prepend**(**val2**);**

cout **<<** "Before Remove Target: " **<<** endl **<<** **\***bag **<<** endl **<<** endl **<<** flush**;**

bag**->**Remove**(**val3**);**

cout **<<** "After Remove Target: " **<<** endl **<<** **\***bag **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** bag**->**Size**());**

**delete** bag**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**}**

void test\_bag\_remove\_empty**()** **{**

Bag**\*** bag **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

cout **<<** "Before Remove Target: " **<<** endl **<<** **\***bag **<<** endl **<<** endl **<<** flush**;**

bag**->**Remove**(**val1**);**

cout **<<** "After Remove Target: " **<<** endl **<<** **\***bag **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**0**,** bag**->**Size**());**

**delete** bag**;**

**delete** val1**;**

**}**

void test\_bag\_remove**()** **{**

Bag**\*** bag **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

bag**->**Prepend**(**val1**);**

bag**->**Prepend**(**val2**);**

bag**->**Prepend**(**val3**);**

bag**->**Prepend**(**val4**);**

/\* Add duplicates \*/

bag**->**Prepend**(**val1**);**

bag**->**Prepend**(**val2**);**

cout **<<** "Before Remove Target: " **<<** endl **<<** **\***bag **<<** endl **<<** endl **<<** flush**;**

bag**->**Remove**(**val1**);**

bag**->**Remove**(**val1**);**

bag**->**Remove**(**val2**);**

bag**->**Remove**(**val2**);**

bag**->**Remove**(**val3**);**

cout **<<** "After Remove Target: " **<<** endl **<<** **\***bag **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**1**,** bag**->**Size**());**

**delete** bag**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

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

// Intersect tests

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

void test\_bag\_intersect\_null\_source**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

target**->**Prepend**(**val3**);**

target**->**Prepend**(**val4**);**

cout **<<** "Before Intersect Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Intersect**(**nullptr**);**

cout **<<** "After Intersect Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**4**,** target**->**Size**());**

**delete** target**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_bag\_intersect\_empty\_source**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

target**->**Prepend**(**val3**);**

target**->**Prepend**(**val4**);**

Bag**\*** source **=** **new** Bag**();**

cout **<<** "Before Intersect Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Before Intersect Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Intersect**(**source**);**

cout **<<** "After Intersect Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "After Intersect Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**0**,** target**->**Size**());**

ASSERT\_EQUAL**(**0**,** source**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_bag\_intersect\_empty\_result**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

Bag**\*** source **=** **new** Bag**();**

String**\*** val3 **=** **new** String**(**"value 3"**);**

String**\*** val4 **=** **new** String**(**"value 4"**);**

source**->**Prepend**(**val3**);**

source**->**Prepend**(**val4**);**

cout **<<** "Before Intersect Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Before Intersect Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Intersect**(**source**);**

cout **<<** "After Intersect Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "After Intersect Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**0**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_bag\_intersect**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

String**\*** val3 **=** **new** String**(**"value 3"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

target**->**Prepend**(**val2**);**

target**->**Prepend**(**val2**);**

target**->**Prepend**(**val3**);**

Bag**\*** source **=** **new** Bag**();**

String**\*** val4 **=** **new** String**(**"value 1"**);**

String**\*** val5 **=** **new** String**(**"value 2"**);**

source**->**Prepend**(**val4**);**

source**->**Prepend**(**val5**);**

source**->**Prepend**(**val5**);**

source**->**Prepend**(**val5**);**

source**->**Prepend**(**val5**);**

source**->**Prepend**(**val5**);**

source**->**Prepend**(**val5**);**

cout **<<** "Before Intersect Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Before Intersect Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Intersect**(**source**);**

cout **<<** "After Intersect Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "After Intersect Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**4**,** target**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**delete** val5**;**

**}**

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

// union tests

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

void test\_bag\_union\_null\_source**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

cout **<<** "Before Union Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Union**(**nullptr**);**

cout **<<** "After Union Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

**delete** target**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_bag\_union\_empty\_source**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

Bag**\*** source **=** **new** Bag**();**

cout **<<** "Before Union Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Before Union Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Union**(**source**);**

cout **<<** "After Union Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "After Union Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

ASSERT\_EQUAL**(**0**,** source**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_bag\_union\_nothing\_todo**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

target**->**Prepend**(**val2**);**

Bag**\*** source **=** **new** Bag**();**

String**\*** val3 **=** **new** String**(**"value 1"**);**

String**\*** val4 **=** **new** String**(**"value 2"**);**

source**->**Prepend**(**val3**);**

source**->**Prepend**(**val3**);**

source**->**Prepend**(**val4**);**

source**->**Prepend**(**val4**);**

cout **<<** "Before Union Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Before Union Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Union**(**source**);**

cout **<<** "After Union Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "After Union Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**4**,** target**->**Size**());**

ASSERT\_EQUAL**(**4**,** source**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_bag\_union**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

target**->**Prepend**(**val2**);**

Bag**\*** source **=** **new** Bag**();**

String**\*** val3 **=** **new** String**(**"value 1"**);**

String**\*** val4 **=** **new** String**(**"value 2"**);**

String**\*** val5 **=** **new** String**(**"value 3"**);**

source**->**Prepend**(**val3**);**

source**->**Prepend**(**val4**);**

source**->**Prepend**(**val4**);**

source**->**Prepend**(**val4**);**

source**->**Prepend**(**val4**);**

source**->**Prepend**(**val5**);**

source**->**Prepend**(**val5**);**

cout **<<** "Before Union Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Before Union Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Union**(**source**);**

cout **<<** "After Union Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "After Union Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**10**,** target**->**Size**());**

ASSERT\_EQUAL**(**7**,** source**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**delete** val5**;**

**}**

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

// complement tests

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

void test\_bag\_complement\_null\_source**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

cout **<<** "Before Complement Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Complement**(**nullptr**);**

cout **<<** "After Complement Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

**delete** target**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_bag\_complement\_empty\_source**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

Bag**\*** source **=** **new** Bag**();**

cout **<<** "Before Complement Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Before Complement Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Complement**(**source**);**

cout **<<** "After Complement Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "After Complement Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**2**,** target**->**Size**());**

ASSERT\_EQUAL**(**0**,** source**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**}**

void test\_bag\_complement\_empty\_result**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val2**);**

Bag**\*** source **=** **new** Bag**();**

String**\*** val3 **=** **new** String**(**"value 1"**);**

String**\*** val4 **=** **new** String**(**"value 2"**);**

source**->**Prepend**(**val3**);**

source**->**Prepend**(**val4**);**

cout **<<** "Before Complement Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Before Complement Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Complement**(**source**);**

cout **<<** "After Complement Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "After Complement Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**0**,** target**->**Size**());**

ASSERT\_EQUAL**(**2**,** source**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

void test\_bag\_complement**()** **{**

Bag**\*** target **=** **new** Bag**();**

String**\*** val1 **=** **new** String**(**"value 1"**);**

String**\*** val2 **=** **new** String**(**"value 2"**);**

/\* target has more than source \*/

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val1**);**

target**->**Prepend**(**val1**);**

/\* target has less than source \*/

target**->**Prepend**(**val2**);**

Bag**\*** source **=** **new** Bag**();**

String**\*** val3 **=** **new** String**(**"value 1"**);**

String**\*** val4 **=** **new** String**(**"value 2"**);**

source**->**Prepend**(**val3**);**

source**->**Prepend**(**val3**);**

source**->**Prepend**(**val3**);**

source**->**Prepend**(**val4**);**

source**->**Prepend**(**val4**);**

cout **<<** "Before Complement Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "Before Complement Target: " **<<** endl **<<** **\***target **<<** endl **<<** endl **<<** flush**;**

target**->**Complement**(**source**);**

cout **<<** "After Complement Source: " **<<** endl **<<** **\***source **<<** endl **<<** endl **<<** flush**;**

cout **<<** "After Complement Target: " **<<** endl **<<** **\***target **<<** endl **<<** flush**;**

ASSERT\_EQUAL**(**1**,** target**->**Size**());**

ASSERT\_EQUAL**(**5**,** source**->**Size**());**

**delete** target**;**

**delete** source**;**

**delete** val1**;**

**delete** val2**;**

**delete** val3**;**

**delete** val4**;**

**}**

## Tests

Folgend sind die Tests für die Bag Implementierung angeführt.

Es wird hierbei davon ausgegangen das die List Implementierung korrekt funktioniert.

Sie wird in diesen Tests sowieso implizit getestet, bis auf die Möglichkeit Duplikate einzufügen.

Am Ende der Tests werden die Meta-Informationen ausgegeben um zu zeigen dass alle Elemente vollständig gelöscht werden.

